<table>
<thead>
<tr>
<th></th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5 Minute Quick Start</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>What’s next?</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Table of contents</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>Other links</td>
<td>253</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>255</td>
</tr>
</tbody>
</table>
Featuretools is a framework to perform automated feature engineering. It excels at transforming temporal and relational datasets into feature matrices for machine learning.
Below is an example of using Deep Feature Synthesis (DFS) to perform automated feature engineering. In this example, we apply DFS to a multi-table dataset consisting of timestamped customer transactions.

```python
In [1]: import featuretools as ft
```

### 1.1 Load Mock Data

```python
In [2]: data = ft.demo.load_mock_customer()
```

### 1.2 Prepare data

In this toy dataset, there are 3 tables. Each table is called an entity in Featuretools.

- **customers**: unique customers who had sessions
- **sessions**: unique sessions and associated attributes
- **transactions**: list of events in this session

```python
In [3]: customers_df = data["customers"]
In [4]: customers_df
Out[4]:
   customer_id  zip_code  join_date  date_of_birth
0          1    60091  2011-04-17 10:48:33  1994-07-18
1          2   13244  2012-04-15 23:31:04  1986-08-18
2          3   13244  2011-08-13 15:42:34  2003-11-21
3          4    60091  2011-04-08 20:08:14  2006-08-15
4          5    60091  2010-07-17 05:27:50  1984-07-28
```

```python
In [5]: sessions_df = data['sessions']
In [6]: sessions_df.sample(5)
Out[6]:
   session_id  customer_id  device   session_start
13          14          1  tablet  2014-01-01 03:28:00
6           7           3  tablet  2014-01-01 01:39:40
1           2           5  mobile 2014-01-01 00:17:20
28          29          1  mobile 2014-01-01 07:10:05
```

(continues on next page)
First, we specify a dictionary with all the entities in our dataset.

Second, we specify how the entities are related. When two entities have a one-to-many relationship, we call the “one” entity the “parent entity”. A relationship between a parent and child is defined like this:

In [10]: relationships = [(
        "sessions", "session_id", "transactions", "session_id"),
        ("customers", "customer_id", "sessions", "customer_id")]

Note: To manage setting up entities and relationships, we recommend using the EntitySet class which offers convenient APIs for managing data like this. See Representing Data with EntitySets for more information.

### 1.3 Run Deep Feature Synthesis

A minimal input to DFS is a set of entities, a list of relationships, and the “target_entity” to calculate features for. The output of DFS is a feature matrix and the corresponding list of feature definitions.

Let’s first create a feature matrix for each customer in the data

```python
In [11]: feature_matrix_customers, features_defs = ft.dfs(entities=entities,
           relationships=relationships,
           target_entity="customers")
```

```mermaid
graph TB
A[zip_code] -- COUNT(sessions) --> B[NUM_UNIQUE(sessions.device)] -- MODE(sessions.
A -- SUM(transactions.amount) --> C[STD(transactions.amount)] -- MAX(transactions.
A -- MEAN(transactions.amount)) -- COUNT(transactions) --> D[NUM_UNIQUE(transactions.product_id)]
A -- MODE(transactions.product_id) --> E[DAY(date_of_birth)] -- DAY(join_date) --> F[YEAR(date_of_birth)]
```

```python
In [12]: feature_matrix_customers
```

```mermaid
graph TB
A[zip_code] -- COUNT(sessions) --> B[NUM_UNIQUE(sessions.device)] -- MODE(sessions.
A -- SUM(transactions.amount) --> C[STD(transactions.amount)] -- MAX(transactions.
A -- MEAN(transactions.amount)) -- COUNT(transactions) --> D[NUM_UNIQUE(transactions.product_id)]
A -- MODE(transactions.product_id) --> E[DAY(date_of_birth)] -- DAY(join_date) --> F[YEAR(date_of_birth)]
```
### 1.3. Run Deep Feature Synthesis

<table>
<thead>
<tr>
<th>customer_id</th>
<th>mobile</th>
<th>9025.62</th>
<th>40.442059</th>
<th>139.</th>
</tr>
</thead>
<tbody>
<tr>
<td>60091</td>
<td>0.019698</td>
<td>5.81</td>
<td>71.631905</td>
<td></td>
</tr>
<tr>
<td>126</td>
<td>181</td>
<td>171994</td>
<td>2011</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>78.59</td>
<td>582.19317</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1057.97</td>
<td>312.745952</td>
<td></td>
<td></td>
</tr>
<tr>
<td>279.510713</td>
<td>6.954507</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.759314</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.589386</td>
<td>7.322191</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.062019</td>
<td>1613.93</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26.36</td>
<td>88.755625</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.640252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46.905665</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.778170</td>
<td>2.440005</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.424949</td>
<td>0.000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.780493</td>
<td>-0.312355</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.946018</td>
<td>809.97</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50.623125</td>
<td>118.90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.038434</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30.450261</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1128.202500</td>
<td>9.823750</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72.774140</td>
<td>5.000000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.39816</td>
<td>132.248250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33.068244</td>
<td>15.750000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(continued from previous page)

<table>
<thead>
<tr>
<th>2</th>
<th>13244</th>
<th>7</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>desktop</td>
<td>7200.28</td>
<td>37.705178</td>
<td>146.</td>
</tr>
<tr>
<td>81</td>
<td>0.098259</td>
<td>8.73</td>
<td>77.422366</td>
</tr>
<tr>
<td>93</td>
<td>18</td>
<td>15</td>
<td>1986</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>154.60</td>
<td>35</td>
<td>931.63</td>
<td></td>
</tr>
<tr>
<td>251.609234</td>
<td>11.477071</td>
<td>0.509798</td>
<td></td>
</tr>
<tr>
<td>3.450328</td>
<td>56.46</td>
<td>96.581000</td>
<td></td>
</tr>
<tr>
<td>47.935920</td>
<td>18</td>
<td>2.154929</td>
<td></td>
</tr>
<tr>
<td>-0.440929</td>
<td>0.235296</td>
<td>-1.539467</td>
<td></td>
</tr>
<tr>
<td>-0.303276</td>
<td>61.910000</td>
<td>100.04</td>
<td></td>
</tr>
<tr>
<td>-0.763603</td>
<td>27.839228</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1028.611429</td>
<td>22.085714</td>
<td>5.000000</td>
<td></td>
</tr>
<tr>
<td>78.415122</td>
<td>-0.039663</td>
<td>13.285714</td>
<td></td>
</tr>
<tr>
<td>36.957218</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>desktop</td>
<td>2014</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 desktop</td>
<td>13244</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>6236.62</td>
<td>43.683296</td>
<td>149.</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.418230</td>
<td>5.89</td>
<td></td>
</tr>
<tr>
<td>36.957218</td>
<td>11.477071</td>
<td>0.000000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2003</td>
<td>2011</td>
<td></td>
</tr>
<tr>
<td>20.66</td>
<td>29</td>
<td>2.246479</td>
<td></td>
</tr>
<tr>
<td>66.21</td>
<td>847.63</td>
<td>2.246479</td>
<td></td>
</tr>
<tr>
<td>219.021420</td>
<td>20.66</td>
<td>2.246479</td>
<td></td>
</tr>
<tr>
<td>11.174282</td>
<td>2.246479</td>
<td>0.678544</td>
<td></td>
</tr>
<tr>
<td>0.429374</td>
<td>0.429374</td>
<td>-2.449490</td>
<td></td>
</tr>
<tr>
<td>2.428992</td>
<td>1477.97</td>
<td>-2.449490</td>
<td></td>
</tr>
<tr>
<td>20.66</td>
<td>82.109444</td>
<td>-2.449490</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>50.110120</td>
<td>0.854976</td>
<td></td>
</tr>
<tr>
<td>50.110120</td>
<td>1039.436667</td>
<td>0.381014</td>
<td></td>
</tr>
<tr>
<td>2.246479</td>
<td>0.381014</td>
<td>11.035000</td>
<td></td>
</tr>
<tr>
<td>0.678544</td>
<td>42.883316</td>
<td>4.833333</td>
<td></td>
</tr>
<tr>
<td>-0.941078</td>
<td>15.500000</td>
<td>3.450328</td>
<td></td>
</tr>
<tr>
<td>-1.507217</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>55.579412</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1039.436667</td>
<td>11.035000</td>
<td>4.833333</td>
<td></td>
</tr>
<tr>
<td>1039.436667</td>
<td>1039.436667</td>
<td>1039.436667</td>
<td></td>
</tr>
<tr>
<td>42.883316</td>
<td>15.500000</td>
<td>3.450328</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
### 1.3. Run Deep Feature Synthesis

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(continues on next page)
We now have dozens of new features to describe a customer’s behavior.

1.4 Change target entity

One of the reasons DFS is so powerful is that it can create a feature matrix for any entity in our data. For example, if we wanted to build features for sessions.

```
In [13]: feature_matrix_sessions, features_defs = ft.dfs(entities=entities,
       ...: relationships=relationships,
       ...: target_entity="sessions")
       ...

In [14]: feature_matrix_sessions.head(5)
Out[14]:
     customer_id device  SUM(transactions.amount)  STD(transactions.amount)  
    MAX(transactions.amount)  SKEW(transactions.amount)  MIN(transactions.amount)  
    MEAN(transactions.product_id)  COUNT(transactions)  NUM_UNIQUE(transactions.product_id)  
    MODE(transactions.product_id)  DAY(session_start)  YEAR(session_start)  
    MONTH(session_start)  WEEKDAY(session_start)  customers.zip_code  NUM_  
    UNIQUE(transactions.DAY(transaction_time))  NUM_UNIQUE(transactions.  
    YEAR(transaction_time))  NUM_UNIQUE(transactions.MONTH(transaction_time))  NUM_  
    UNIQUE(transactions.WEEKDAY(transaction_time))  MODE(transactions.DAY(transaction_  
    time))  MODE(transactions.YEAR(transaction_time))  MODE(transactions.  
    MONTH(transaction_time))  MODE(transactions.WEEKDAY(transaction_time))  customers.  
    COUNT(sessions)  customers.NUM_UNIQUE(sessions.device)  customers.MODE(sessions.  
    device)  customers.SUM(transactions.amount)  customers.STD(transactions.amount)  
    customers.MAX(transactions.amount)  customers.SKEW(transactions.amount)  customers.  
    MIN(transactions.amount)  customers.MEAN(transactions.amount)  customers.  
    COUNT(transactions)  customers.NUM_UNIQUE(transactions.product_id)  customers.  
    MODE(transactions.product_id)  customers.DAY(date_of_birth)  customers.DAY(join_  
    date)  customers.YEAR(date_of_birth)  customers.YEAR(join_date)  customers.  
    MONTH(date_of_birth)  customers.MONTH(join_date)  customers.WEEKDAY(date_of_birth)  
    customers.WEEKDAY(join_date)  session_id
    ...
     1  2  desktop  1229.01  41.600976  141.66  0.295458  20.91
     76.813125  16  1  2014
     1  2  13244
```

(continues on next page)
### 1.4. Change target entity

<table>
<thead>
<tr>
<th>1</th>
<th>1 mobile</th>
<th>746.96</th>
<th>45.893591</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>135.25</td>
<td>-0.160550</td>
<td>9.32</td>
</tr>
<tr>
<td>2</td>
<td>74.696000</td>
<td>10</td>
<td>2014</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>60091</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2014</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>2014</td>
</tr>
<tr>
<td>6</td>
<td>mobile</td>
<td>6349.66</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>7.55</td>
<td>80.375443</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>1984</td>
<td>2010</td>
</tr>
<tr>
<td>9</td>
<td>2014</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>mobile</td>
<td>8727.68</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>5.73</td>
<td>80.070459</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>109</td>
<td>15</td>
<td>2006</td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1 mobile</td>
<td>1613.93</td>
<td>40.187205</td>
</tr>
<tr>
<td>15</td>
<td>5.81</td>
<td>71.631905</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>126</td>
<td>18</td>
<td>1994</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>mobile</td>
<td>777.02</td>
<td>11</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>60091</td>
<td>1</td>
</tr>
</tbody>
</table>

(continues on next page)
• Learn about *Representing Data with EntitySets*
• Apply automated feature engineering with *Deep Feature Synthesis*
• Explore runnable demos based on real world use cases
• Can’t find what you’re looking for? Ask for *Help*
3.1 Install

Featuretools is available for Python 3.5, 3.6 and 3.7 The recommended way to install Featuretools is using `pip` or `conda`:

```
python -m pip install featuretools
```

or from the Conda-forge channel on anaconda.org:

```
conda install -c conda-forge featuretools
```

3.1.1 Add-ons

You can install add-ons individually or all at once by running:

```
python -m pip install featuretools[complete]
```

**Update checker:** Receive automatic notifications of new Featuretools releases:

```
python -m pip install featuretools[update_checker]
```

**TSFresh Primitives:** Use 60+ primitives from `tsfresh` in Featuretools:

```
python -m pip install featuretools[tsfresh]
```

**Categorical Encoding:** Encode categorical data for integration into Featuretools/machine learning workflows:

```
python -m pip install featuretools[categorical_encoding]
```

**NLP Primitives:** Use Natural Language Processing Primitives for data with text in Featuretools:

```
python -m pip install featuretools[nlp_primitives]
```

**AutoNormalize:** Automated creation of normalized `EntitySet` from denormalized data:

```
python -m pip install featuretools[autonormalize]
```

**Featuretools Sklearn Transformer:** Deep Feature Synthesis as a scikit-learn pipelines transformer

```
python -m pip install featuretools[sklearn_transformer]
```
### 3.1.2 Installing Graphviz

In order to use `EntitySet.plot` you will need to install the graphviz library.

Conda users:

```bash
conda install python-graphviz
```

Ubuntu:

```bash
sudo apt-get install graphviz
pip install graphviz
```

Mac OS:

```bash
brew install graphviz
pip install graphviz
```

Windows:

```bash
conda install python-graphviz
```

### 3.1.3 Install from Source

To install featuretools from source, clone the repository from GitHub:

```bash
git clone https://github.com/featuretools/featuretools.git
cd featuretools
python setup.py install
```

or use `pip` locally if you want to install all dependencies as well:

```bash
pip install .
```

You can view the list of all dependencies within the `extras_require` field of `setup.py`.

### 3.1.4 Development

Before making contributing to the codebase, please follow the guidelines [here](#).

**Virtualenv**

We recommend developing in a `virtualenv`:

```bash
mkvirtualenv featuretools
```

**Install development requirements**

Run:

```bash
make installdeps
```
Test

Note: In order to run the featuretools tests you will need to have graphviz installed as described above.

Run featuretools tests:

```
make test
```

Before committing make sure to run linting in order to pass CI:

```
make lint
```

Some linting errors can be automatically fixed by running the command below:

```
make lint-fix
```

Build Documentation

Build the docs with the commands below:

```
cd docs/
# small changes
make html
# rebuild from scratch
make clean html
```

Note: The Featuretools library must be importable to build the docs.

3.2 Representing Data with EntitySets

An EntitySet is a collection of entities and the relationships between them. They are useful for preparing raw, structured datasets for feature engineering. While many functions in Featuretools take entities and relationships as separate arguments, it is recommended to create an EntitySet, so you can more easily manipulate your data as needed.

3.2.1 The Raw Data

Below we have a two tables of data (represented as Pandas DataFrames) related to customer transactions. The first is a merge of transactions, sessions, and customers so that the result looks like something you might see in a log file:

```
In [1]: import featuretools as ft
In [2]: data = ft.demo.load_mock_customer()
In [3]: transactions_df = data["transactions"].merge(data["sessions"]).merge(data["customers"])
```
3.2.2 Creating an EntitySet

First, we initialize an EntitySet. If you’d like to give it name, you can optionally provide an id to the constructor.

```
In [7]: es = ft.EntitySet(id="customer_data")
```

3.2.3 Adding entities

To get started, we load the transactions dataframe as an entity.

```
In [8]: es = es.entity_from_dataframe(entity_id="transactions",
   ...:     dataframe=transactions_df,
   ...:     index="transaction_id",
   ...:     time_index="transaction_time",
   ...
```

And the second dataframe is a list of products involved in those transactions.

```
In [5]: products_df = data["products"]

In [6]: products_df
Out[6]
```
variable_types={"product_id": ft.variable_types.Categorical,
"zip_code": ft.variable_types.ZIPCode})

In [9]: es
Out[9]:
Entityset: customer_data
    Entities:
        transactions [Rows: 500, Columns: 11]
    Relationships:
        No relationships

Note: You can also display your entity set structure graphically by calling `EntitySet.plot()`.

This method loads each column in the dataframe as a variable. We can see the variables in an entity using the code below.

In [10]: es["transactions"].variables
Out[10]:
[<Variable: transaction_id (dtype = index)>,
 <Variable: session_id (dtype = numeric)>,
 <Variable: transaction_time (dtype: datetime_time_index, format: None)>,
 <Variable: amount (dtype = numeric)>,
 <Variable: customer_id (dtype = numeric)>,
 <Variable: device (dtype = categorical)>,
 <Variable: session_start (dtype: datetime, format: None)>,
 <Variable: join_date (dtype: datetime, format: None)>,
 <Variable: date_of_birth (dtype: datetime, format: None)>,
 <Variable: product_id (dtype = categorical)>,
 <Variable: zip_code (dtype = zipcode)>]

In the call to `entity_from_dataframe`, we specified three important parameters

- The `index` parameter specifies the column that uniquely identifies rows in the dataframe
- The `time_index` parameter tells Featuretools when the data was created.
- The `variable_types` parameter indicates that “product_id” should be interpreted as a Categorical variable, even though it just an integer in the underlying data.

Now, we can do that same thing with our products dataframe

In [11]: es = es.entity_from_dataframe(entity_id="products",
   dataframes=products_df,
   index="product_id")

In [12]: es
Out[12]:
Entityset: customer_data
    Entities:
        transactions [Rows: 500, Columns: 11]
        products [Rows: 5, Columns: 2]
With two entities in our entity set, we can add a relationship between them.

### 3.2.4 Adding a Relationship

We want to relate these two entities by the columns called “product_id” in each entity. Each product has multiple transactions associated with it, so it is called it the **parent entity**, while the transactions entity is known as the **child entity**. When specifying relationships we list the variable in the parent entity first. Note that each `ft.Relationship` must denote a one-to-many relationship rather than a relationship which is one-to-one or many-to-many.

```python
In [13]: new_relationship = ft.Relationship(es["products"]['product_id'],
                                   ....: es["transactions"]['product_id'])
In [14]: es = es.add_relationship(new_relationship)
In [15]: es
Out[15]:
Entityset: customer_data
  Entities:
  transactions [Rows: 500, Columns: 11]
  products [Rows: 5, Columns: 2]
  Relationships:
  transactions.product_id -> products.product_id
```

Now, we see the relationship has been added to our entity set.

### 3.2.5 Creating entity from existing table

When working with raw data, it is common to have sufficient information to justify the creation of new entities. In order to create a new entity and relationship for sessions, we “normalize” the transaction entity.

```python
In [16]: es = es.normalize_entity(base_entity_id="transactions",
                               ....: new_entity_id="sessions",
                               ....: index="session_id",
                               ....: make_time_index="session_start",
                               ....: additional_variables=["device", "customer_id", "zip_code", "session_start", "join_date"])
In [17]: es
Out[17]:
Entityset: customer_data
  Entities:
  transactions [Rows: 500, Columns: 6]
  products [Rows: 5, Columns: 2]
  sessions [Rows: 35, Columns: 6]
  Relationships:
  transactions.product_id -> products.product_id
  transactions.session_id -> sessions.session_id
```

Looking at the output above, we see this method did two operations
1. It created a new entity called “sessions” based on the “session_id” and “session_start” variables in “transactions”

2. It added a relationship connecting “transactions” and “sessions”.

If we look at the variables in transactions and the new sessions entity, we see two more operations that were performed automatically.

In [18]: es["transactions"].variables
Out[18]:
[<Variable: transaction_id (dtype = index)>,
 <Variable: session_id (dtype = id)>,
 <Variable: transaction_time (dtype: datetime_time_index, format: None)>,
 <Variable: amount (dtype = numeric)>,
 <Variable: date_of_birth (dtype: datetime, format: None)>,
 <Variable: product_id (dtype = id)>]

In [19]: es["sessions"].variables

1. It removed “device”, “customer_id”, “zip_code” and “join_date” from “transactions” and created a new variables in the sessions entity. This reduces redundant information as the those properties of a session don’t change between transactions.

2. It copied and marked “session_start” as a time index variable into the new sessions entity to indicate the beginning of a session. If the base entity has a time index and make_time_index is not set, normalize entity will create a time index for the new entity. In this case it would create a new time index called “first_transactions_time” using the time of the first transaction of each session. If we don’t want this time index to be created, we can set make_time_index=False.

If we look at the dataframes, can see what the normalize_entity did to the actual data.

In [20]: es["sessions"].df.head(5)
Out[20]:
<table>
<thead>
<tr>
<th>session_id</th>
<th>device</th>
<th>customer_id</th>
<th>zip_code</th>
<th>session_start</th>
<th>join_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>desktop</td>
<td>2</td>
<td>13244</td>
<td>2014-01-01 00:00:00</td>
<td>2012-04-15 23:31:04</td>
</tr>
<tr>
<td>2</td>
<td>mobile</td>
<td>5</td>
<td>60091</td>
<td>2014-01-01 00:17:20</td>
<td>2010-07-17 05:27:50</td>
</tr>
<tr>
<td>3</td>
<td>mobile</td>
<td>4</td>
<td>60091</td>
<td>2014-01-01 00:28:10</td>
<td>2011-04-08 20:08:14</td>
</tr>
<tr>
<td>4</td>
<td>mobile</td>
<td>1</td>
<td>60091</td>
<td>2014-01-01 01:11:30</td>
<td>2011-04-08 20:08:14</td>
</tr>
<tr>
<td>5</td>
<td>mobile</td>
<td>4</td>
<td>60091</td>
<td>2014-01-01 01:11:30</td>
<td>2011-04-08 20:08:14</td>
</tr>
</tbody>
</table>

In [21]: es["transactions"].df.head(5)

To finish preparing this dataset, create a “customers” entity using the same method call.

3.2. Representing Data with EntitySets 19
In [22]: es = es.normalize_entity(base_entity_id="sessions",
.....:   new_entity_id="customers",
.....:   index="customer_id",
.....:   make_time_index="join_date",
.....:   additional_variables=["zip_code", "join_date"]

In [23]: es
Out[23]:
Entityset: customer_data
   Entities:
   transactions [Rows: 500, Columns: 6]
   products [Rows: 5, Columns: 2]
   sessions [Rows: 35, Columns: 4]
   customers [Rows: 5, Columns: 3]
Relationships:
  transactions.product_id -> products.product_id
  transactions.session_id -> sessions.session_id
  sessions.customer_id -> customers.customer_id

3.2.6 Using the EntitySet

Finally, we are ready to use this EntitySet with any functionality within Featuretools. For example, let’s build a feature matrix for each product in our dataset.

In [24]: feature_matrix, feature_defs = ft.dfs(entityset=es,
.....:   target_entity="products")

In [25]: feature_matrix
Out[25]:
   brand  SUM(transactions.amount)  STD(transactions.amount)
   MAX(transactions.amount)  SKEW(transactions.amount)  MIN(transactions.amount)
   MEAN(transactions.amount)  COUNT(transactions)  NUM_UNIQUE(transactions.session_id)
   MODE(transactions.session_id)  NUM_UNIQUE(transactions.DAY(transaction_time))
   NUM_UNIQUE(transactions.WEEKDAY(date_of_birth))  NUM_UNIQUE(transactions.
   YEAR(transaction_time))  NUM_UNIQUE(transactions.MONTH(transaction_time))
   NUM_UNIQUE(transactions.WEEKDAY(transaction_time))  NUM_UNIQUE(transactions.
   customer_id)  NUM_UNIQUE(transactions.MONTH(date_of_birth))
   NUM_UNIQUE(transactions.YEAR(date_of_birth))  MODE(transactions.DAY(transaction_time))
   MODE(transactions.MONTH(transaction_time))  MODE(transactions.YEAR(date_of_birth))
   MODE(transactions.WEEKDAY(date_of_birth))  MODE(transactions.WEEKDAY(transaction_}
   time))  MODE(transactions.MONTH(transaction_time))  MODE(transactions.
   WEEKDAY(transaction_time))  MODE(transactions.sessions.customer_id)
   MODE(transactions.MONTH(date_of_birth))  MODE(transactions.YEAR(date_of_birth))
   product_id
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>7489.79</td>
<td>42.479989</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>149.56</td>
<td>0.125525</td>
<td>6.84</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>429314</td>
<td>102</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>2014</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1994</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>7021.43</td>
<td>46.336308</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>149.95</td>
<td>0.151934</td>
<td>5.73</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>319891</td>
<td>92</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>desktop</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>2014</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1994</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>7008.12</td>
<td>38.871405</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>148.31</td>
<td>0.223938</td>
<td>5.89</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>001250</td>
<td>96</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>desktop</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>2014</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1994</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>8088.97</td>
<td>42.492501</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>146.46</td>
<td>-0.132077</td>
<td>5.81</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>311038</td>
<td>106</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>desktop</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>2014</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1994</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>7931.55</td>
<td>42.131902</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>149.02</td>
<td>0.098248</td>
<td>5.91</td>
<td>34</td>
</tr>
</tbody>
</table>

### 3.2. Representing Data with EntitySets
As we can see, the features from DFS use the relational structure of our entity set. Therefore it is important to think carefully about the entities that we create.

### 3.3 Deep Feature Synthesis

Deep Feature Synthesis (DFS) is an automated method for performing feature engineering on relational and temporal data.

#### 3.3.1 Input Data

Deep Feature Synthesis requires structured datasets in order to perform feature engineering. To demonstrate the capabilities of DFS, we will use a mock customer transactions dataset.

**Note:** Before using DFS, it is recommended that you prepare your data as an `EntitySet`. See *Representing Data with EntitySets* to learn how.

```python
In [1]: import featuretools as ft
In [2]: es = ft.demo.load_mock_customer(return_entityset=True)
In [3]: es
Out[3]:
Entityset: transactions
   Entities:
      transactions [Rows: 500, Columns: 5]
      products [Rows: 5, Columns: 2]
      sessions [Rows: 35, Columns: 4]
      customers [Rows: 5, Columns: 4]
   Relationships:
      transactions.product_id -> products.product_id
      transactions.session_id -> sessions.session_id
      sessions.customer_id -> customers.customer_id
```

Once data is prepared as an `EntitySet`, we are ready to automatically generate features for a target entity - e.g. `customers`.

#### 3.3.2 Running DFS

Typically, without automated feature engineering, a data scientist would write code to aggregate data for a customer, and apply different statistical functions resulting in features quantifying the customer’s behavior. In this example, an expert might be interested in features such as: *total number of sessions* or *month the customer signed up*.

These features can be generated by DFS when we specify the target_entity as `customers` and "count" and "month" as primitives.

```python
In [4]: feature_matrix, feature_defs = ft.dfs(entityset=es, target_entity="customers", agg_primitives=["count"],
```

(continues on next page)
In the example above, "count" is an aggregation primitive because it computes a single value based on many sessions related to one customer. "month" is called a transform primitive because it takes one value for a customer transforms it to another.

**Note:** Feature primitives are a fundamental component to Featuretools. To learn more read [Feature primitives](#).

### 3.3.3 Creating “Deep Features”

The name Deep Feature Synthesis comes from the algorithm’s ability to stack primitives to generate more complex features. Each time we stack a primitive we increase the “depth” of a feature. The max_depth parameter controls the maximum depth of the features returned by DFS. Let us try running DFS with max_depth=2.

```
In [6]: feature_matrix, feature_defs = ft.dfs(entityset=es,
   ...:                 target_entity="customers",
   ...:                 agg_primitives=['mean', 'sum', 'mode'],
   ...:                 trans_primitives=['month', 'hour'],
   ...:                 max_depth=2)
```

```
In [7]: feature_matrix
```

```
+-----------+-----------------+-----------------+------------+
<table>
<thead>
<tr>
<th>zip_code</th>
<th>count (sessions)</th>
<th>month (date)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 60091</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>4 60091</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1 60091</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>3 13244</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>2 13244</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>[5 rows x 17 columns]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

With a depth of 2, a number of features are generated using the supplied primitives. The algorithm to synthesize these definitions is described in this paper. In the returned feature matrix, let us understand one of the depth 2 features.

```
In [8]: feature_matrix[['MEAN(sessions.SUM(transactions.amount))']]
```

```
+-----------+---------------------+
<table>
<thead>
<tr>
<th>customer_id</th>
<th>MEAN (sessions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1058.276667</td>
</tr>
<tr>
<td></td>
<td>(continues on next page)</td>
</tr>
</tbody>
</table>
```
For each customer this feature

1. calculates the sum of all transaction amounts per session to get total amount per session,
2. then applies the mean to the total amounts across multiple sessions to identify the average amount spent per session

We call this feature a “deep feature” with a depth of 2.

Let’s look at another depth 2 feature that calculates for every customer the most common hour of the day when they start a session

```
In [9]: feature_matrix[['MODE(sessions.HOUR(session_start))']]
Out[9]:
customer_id
5 0
4 1
1 6
3 5
2 3
```

For each customer this feature calculates

1. The hour of the day each of his or her sessions started, then
2. uses the statistical function mode to identify the most common hour he or she started a session

Stacking results in features that are more expressive than individual primitives themselves. This enables the automatic creation of complex patterns for machine learning.

### 3.3.4 Changing Target Entity

DFS is powerful because we can create a feature matrix for any entity in our dataset. If we switch our target entity to “sessions”, we can synthesize features for each session instead of each customer. Now, we can use these features to predict the outcome of a session.

```
In [10]: feature_matrix, feature_defs = ft.dfs(entityset=es, 
....: target_entity="sessions", 
....: agg_primitives=["mean", "sum", "mode"], 
....: trans_primitives=["month", "hour"], 
....: max_depth=2)
```

```
In [11]: feature_matrix.head(5)
Out[11]:
   customer_id ... customers.HOUR(join_date)
  session_id ...
1  2 ... 23
2  5 ...  5
3  4 ... 20
4  1 ... 10
```

(continues on next page)
As we can see, DFS will also build deep features based on a parent entity, in this case the customer of a particular session. For example, the feature below calculates the mean transaction amount of the customer of the session.

```
In [12]: feature_matrix[['customers.MEAN(transactions.amount)']].head(5)
```

```
Out[12]:

<table>
<thead>
<tr>
<th>session_id</th>
<th>customers.MEAN(transactions.amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>77.422366</td>
</tr>
<tr>
<td>2</td>
<td>80.375443</td>
</tr>
<tr>
<td>3</td>
<td>80.070459</td>
</tr>
<tr>
<td>4</td>
<td>71.631905</td>
</tr>
<tr>
<td>5</td>
<td>80.070459</td>
</tr>
</tbody>
</table>
```

**Improve feature output**

To learn about the parameters to change in DFS read *Tuning Deep Feature Synthesis*.

### 3.4 Feature primitives

Feature primitives are the building blocks of Featuretools. They define individual computations that can be applied to raw datasets to create new features. Because a primitive only constrains the input and output data types, they can be applied across datasets and can stack to create new calculations.

#### 3.4.1 Why primitives?

The space of potential functions that humans use to create a feature is expansive. By breaking common feature engineering calculations down into primitive components, we are able to capture the underlying structure of the features humans create today.

A primitive only constrains the input and output data types. This means they can be used to transfer calculations known in one domain to another. Consider a feature which is often calculated by data scientists for transactional or event logs data: *average time between events*. This feature is incredibly valuable in predicting fraudulent behavior or future customer engagement.

DFS achieves the same feature by stacking two primitives "time_since_previous" and "mean"

```
In [1]: feature_defs = ft.dfs(entityset=es, ...
   ...: target_entity="customers",
   ...: agg_primitives=["mean"],
   ...: trans_primitives=["time_since_previous"],
   ...: features_only=True)
```

```
In [2]: feature_defs
Out[2]:

[<Feature: zip_code>,
 <Feature: MEAN(transactions.amount)>,
 <Feature: TIME_SINCE_PREVIOUS(join_date)>],
```

(continues on next page)
Note: When dfs is called with features_only=True, only feature definitions are returned as output. By default this parameter is set to False. This parameter is used quickly inspect the feature definitions before the spending time calculating the feature matrix.

A second advantage of primitives is that they can be used to quickly enumerate many interesting features in a parameterized way. This is used by Deep Feature Synthesis to get several different ways of summarizing the time since the previous event.

```
In [3]: feature_matrix, feature_defs = ft.dfs(entityset=es,
    ...: target_entity="customers",
    ...: agg_primitives=["mean", "max", "min",
    ...: "std", "skew"],
    ...: trans_primitives=["time_since_previous
    ...: ->
    ...: \])

In [4]: feature_matrix[["MEAN(sessions.TIME_SINCE_PREVIOUS(session_start))",
    ...: "MAX(sessions.TIME_SINCE_PREVIOUS(session_start))",
    ...: "MIN(sessions.TIME_SINCE_PREVIOUS(session_start))",
    ...: "STD(sessions.TIME_SINCE_PREVIOUS(session_start))",
    ...: "SKEW(sessions.TIME_SINCE_PREVIOUS(session_start))"]]
```

```
Out[4]:
    MEAN(sessions.TIME_SINCE_PREVIOUS(session_start))  MAX(sessions.TIME_SINCE_PREVIOUS(session_start))  
    MIN(sessions.TIME_SINCE_PREVIOUS(session_start))  STD(sessions.TIME_SINCE_PREVIOUS(session_start))  
    SKEW(sessions.TIME_SINCE_PREVIOUS(session_start))
    customer_id
    ...
    5  1170.0 1007.500000  715.0
    4  1625.0 157.884451  1170.0
    4  1.507217 999.375000  650.0
    1  1625.0 308.688904  1170.0
    1  1.065177 966.875000  715.0
    3  1170.0 177.613813  888.333333
    3  0.254557 177.613813  650.0
    2  1170.0 194.638554  725.833333
    2  0.434581 725.833333  520.0
    1  975.0 194.638554  0.162631
    1  0.162631
```
### 3.4.2 Aggregation vs Transform Primitive

In the example above, we use two types of primitives.

**Aggregation primitives:** These primitives take related instances as an input and output a single value. They are applied across a parent-child relationship in an entity set. E.g: "count", "sum", "avg_time_between".

**Transform primitives:** These primitives take one or more variables from an entity as an input and output a new variable for that entity. They are applied to a single entity. E.g: "hour", "time_since_previous", "absolute".

For a DataFrame that lists and describes each built-in primitive in Featuretools, call `ft.list_primitives()`. In addition, a list of all available primitives can be obtained by visiting primitives.featurelabs.com.

```python
In [5]: ft.list_primitives().head(5)
Out[5]:
   name          type       description                                                                                       
0  median  aggregation    Determines the middlemost number in a list of ...                                                                 
1  n_most_common  aggregation    Determines the `n` most common elements.                                                           
2     num_true  aggregation    Counts the number of 'True' values.                                                                 
3  time_since_last  aggregation    Calculates the time elapsed since the last dat...                                               
4        max  aggregation    Calculates the highest value, ignoring 'NaN' v...
```

### 3.4.3 Defining Custom Primitives

The library of primitives in Featuretools is constantly expanding. Users can define their own primitive using the APIs below. To define a primitive, a user will

- Specify the type of primitive **Aggregation** or **Transform**
- Define the input and output data types
- Write a function in python to do the calculation
- Annotate with attributes to constrain how it is applied

Once a primitive is defined, it can stack with existing primitives to generate complex patterns. This enables primitives known to be important for one domain to automatically be transferred to another.

**Simple Custom Primitives**

```python
In [6]: from featuretools.primitives import make_agg_primitive, make_trans_primitive
In [7]: from featuretools.variable_types import Text, Numeric
In [8]: def absolute(column):
...:     return abs(column)
...:
In [9]: Absolute = make_trans_primitive(function=absolute,
...:   input_types=[Numeric],
...:   return_type=Numeric)
```

Above we created a new transform primitive that can be used with Deep Feature Synthesis using `make_trans_primitive` and a python function we defined. Additionally, we annotated the input data types that the primitive can be applied to and the data type it returns.

Similarly, we can make a new aggregation primitive using `make_agg_primitive`. 

## 3.4. Feature primitives
In [10]: def maximum(column):
...:    return max(column)
...:
In [11]: Maximum = make_agg_primitive(function=maximum,
...:                               input_types=[Numeric],
...:                               return_type=Numeric)
...

Because we defined an aggregation primitive, the function takes in a list of values but only returns one.

Now that we've defined two primitives, we can use them with the dfs function as if they were built-in primitives.

In [12]: feature_matrix, feature_defs = ft.dfs(entityset=es,
...:                                            target_entity="sessions",
...:                                            agg_primitives=[Maximum],
...:                                            trans_primitives=[Absolute],
...:                                            max_depth=2)
...

In [13]: feature_matrix["customers.MAXIMUM(transactions.amount)",
...:                   "MAXIMUM(transactions.ABSOLUTE(amount))"].head(5)
Out[13]:
        customers.MAXIMUM(transactions.amount)  MAXIMUM(transactions.
        session_id  transactions.ABSOLUTE(amount))
  0           146.81               141.66
  1           149.02               135.25
  2           149.95               147.73
  3           139.43               129.00
  4           149.95               139.20

Word Count Example

Here we define a function, word_count, which counts the number of words in each row of an input and returns a list of the counts.

In [14]: def word_count(column):
...:    '''
...:    Counts the number of words in each row of the column. Returns a list
...:    of the counts for each row.
...:    '''
...:    word_counts = []
...:    for value in column:
...:        words = value.split(None)
...:        word_counts.append(len(words))
...:    return word_counts
...

Next, we need to create a custom primitive from the word_count function.
Since `WordCount` is a transform primitive, we need to add it to the list of transform primitives DFS can use when generating features.

By adding some aggregation primitives as well, Deep Feature Synthesis was able to make four new features from one new primitive.

### Multiple Input Types

If a primitive requires multiple features as input, `input_types` has multiple elements, eg `[Numeric, Numeric]` would mean the primitive requires two Numeric features as input. Below is an example of a primitive that has multiple input features.
```python
......:     days = pd.DatetimeIndex(datetime).weekday.values
......:     df = pd.DataFrame({'numeric': numeric, 'time': days})
......:     return df[df['time'] == 6]['numeric'].mean()
......:
In [23]: MeanSunday = make_agg_primitive(function=mean_sunday,
......:                               input_types=[Numeric, Datetime],
......:                               return_type=Numeric)

In [24]: feature_matrix, features = ft.dfs(entityset=es,
......:                               target_entity="sessions",
......:                               agg_primitives=[MeanSunday],
......:                               trans_primitives=[],
......:                               max_depth=1)

In [25]: feature_matrix[['"MEAN_SUNDAY(log.value, datetime)"', "MEAN_SUNDAY(log.value_2, datetime)"]]
Out[25]:
<table>
<thead>
<tr>
<th></th>
<th>MEAN_SUNDAY(log.value, datetime)</th>
<th>MEAN_SUNDAY(log.value_2, datetime)</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2</td>
<td>2.5</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>7.0</td>
<td>3.0</td>
</tr>
<tr>
<td>4</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>5</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>
```

### 3.5 Handling Time

When performing feature engineering with temporal data, carefully selecting the data that is used for any calculation is paramount. By annotating entities with a **time index** column and providing a **cutoff time** during feature calculation, Featuretools will automatically filter out any data after the cutoff time before running any calculations.

#### 3.5.1 What is the Time Index?

The time index is the column in the data that specifies when the data in each row became known. For example, let’s examine a table of customer transactions:

```python
In [1]: import featuretools as ft

In [2]: es = ft.demo.load_mock_customer(return_entityset=True, random_seed=0)

In [3]: es['transactions'].df.head()
Out[3]:
<table>
<thead>
<tr>
<th>transaction_id</th>
<th>session_id</th>
<th>transaction_time</th>
<th>amount</th>
<th>product_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>298</td>
<td>298</td>
<td>2014-01-01 00:00:00</td>
<td>127.64</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2014-01-01 00:01:05</td>
<td>109.48</td>
<td>2</td>
</tr>
<tr>
<td>308</td>
<td>308</td>
<td>2014-01-01 00:02:10</td>
<td>95.06</td>
<td>3</td>
</tr>
<tr>
<td>116</td>
<td>116</td>
<td>2014-01-01 00:03:15</td>
<td>78.92</td>
<td>4</td>
</tr>
<tr>
<td>371</td>
<td>371</td>
<td>2014-01-01 00:04:20</td>
<td>31.54</td>
<td>3</td>
</tr>
</tbody>
</table>
```
In this table, there is one row for every transaction and a `transaction_time` column that specifies when the transaction took place. This means that `transaction_time` is the time index because it indicates when the information in each row became known and available for feature calculations.

However, not every datetime column is a time index. Consider the `customers` entity:

<table>
<thead>
<tr>
<th>customer_id</th>
<th>join_date</th>
<th>date_of_birth</th>
<th>zip_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5 2010-07-17 05:27:50</td>
<td>1984-07-28</td>
<td>60091</td>
</tr>
<tr>
<td>4</td>
<td>4 2011-04-08 20:08:14</td>
<td>2006-08-15</td>
<td>60091</td>
</tr>
<tr>
<td>1</td>
<td>1 2011-04-17 10:48:33</td>
<td>1994-07-18</td>
<td>60091</td>
</tr>
<tr>
<td>3</td>
<td>3 2011-08-13 15:42:34</td>
<td>2003-11-21</td>
<td>13244</td>
</tr>
<tr>
<td>2</td>
<td>2 2012-04-15 23:31:04</td>
<td>1986-08-18</td>
<td>13244</td>
</tr>
</tbody>
</table>

Here, we have two time columns, `join_date` and `date_of_birth`. While either column might be useful for making features, the `join_date` should be used as the time index because it indicates when that customer first became available in the dataset.

**Important:** The time index is defined as the first time that any information from a row can be used. If a cutoff time is specified when calculating features, rows that have a later value for the time index are automatically ignored.

### 3.5.2 What is the Cutoff Time?

The `cutoff_time` specifies the last point in time that a row’s data can be used for a feature calculation. Any data after this point in time will be filtered out before calculating features.

For example, let’s consider a dataset of timestamped customer transactions, where we want to predict whether customers 1, 2 and 3 will spend $500 between 04:00 on January 1 and the end of the day. When building features for this prediction problem, we need to ensure that no data after 04:00 is used in our calculations.

We pass the cutoff time to `featuretools.dfs()` or `featuretools.calculate_feature_matrix()` using the `cutoff_time` argument like this:

```python
In [5]: fm, features = ft.dfs(entityset=es,
...:                          target_entity='customers',
...:                          cutoff_time=pd.Timestamp("2014-1-1 04:00"),
...:                          instance_ids=[1,2,3],
```

(continues on next page)
In [6]: fm
cutoff_time_in_index=True)

Out[6]:

| zip_code | COUNT(sessions) | NUM_UNIQUE(sessions.device) | SUM(transactions.amount) | STD(transactions.amount) | MAX(transactions.amount) | COUNT(transactions) | NUM_UNIQUE(transactions.product_id) | MIN(transactions.amount) | MEAN(transactions.amount) | SUM(sessions.MIN(transactions.amount)) | SUM(sessions.MODE(transactions.amount)) | MAX(sessions.SUM(transactions.amount)) | MAX(sessions.MIN(transactions.amount)) | MAX(sessions.MEAN(transactions.amount)) | MAX(sessions.NUM_UNIQUE(transactions.product_id)) | MAX(sessions.MAX(transactions.amount)) | MAX(sessions.STD(transactions.amount)) | MAX(sessions.COUNT(transactions)) | SKEW(sessions.SUM(transactions.amount)) | SKEW(sessions.MIN(transactions.amount)) | SKEW(sessions.MEAN(transactions.amount)) | SKEW(sessions.NUM_UNIQUE(transactions.product_id)) | SKEW(sessions.MAX(transactions.amount)) | SKEW(sessions.STD(transactions.amount)) | SKEW(sessions.COUNT(transactions)) | NUM_UNIQUE(sessions.WEEKDAY(session_start)) | NUM_UNIQUE(sessions.MONTH(session_start)) | NUM_UNIQUE(sessions.DAY(session_start)) | NUM_UNIQUE(sessions.YEAR(session_start)) | NUM_UNIQUE(sessions.MODE(transactions.product_id)) | MODE(transactions.WEEKDAY(session_start)) | MODE(transactions.MONTH(session_start)) | MODE(transactions.DAY(session_start)) | MODE(transactions.YEAR(session_start)) | MODE(transactions.MODE(transactions.product_id)) | NUM_UNIQUE(transactions.sessions.device) | MODE(transactions.sessions.customer_id) | Custome...
### 3.5. Handling Time

<table>
<thead>
<tr>
<th></th>
<th>2014-01-01 04:00:00</th>
<th>60091</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>tablet</td>
<td>4958.19</td>
<td>42.309717</td>
</tr>
<tr>
<td>4</td>
<td>139.23</td>
<td>-0.006928</td>
<td>5.81</td>
</tr>
<tr>
<td>5</td>
<td>74.002836</td>
<td>67</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>2011</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>18</td>
<td>1994</td>
</tr>
<tr>
<td>8</td>
<td>27.62</td>
<td>204.607793</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>601700</td>
<td>20</td>
<td>304.12</td>
</tr>
<tr>
<td>10</td>
<td>-0.505043</td>
<td>540.04</td>
<td>1.12</td>
</tr>
<tr>
<td>11</td>
<td>169.572874</td>
<td>271.917637</td>
<td>1.23</td>
</tr>
<tr>
<td>12</td>
<td>285833</td>
<td>10.426572</td>
<td>12.43</td>
</tr>
<tr>
<td>13</td>
<td>5.027226</td>
<td>5.678908</td>
<td>1613.67</td>
</tr>
<tr>
<td>14</td>
<td>93</td>
<td>8.74</td>
<td>85.469167</td>
</tr>
<tr>
<td>15</td>
<td>-0.234349</td>
<td>46.905665</td>
<td>25.12</td>
</tr>
<tr>
<td>16</td>
<td>1.197406</td>
<td>1.452325</td>
<td>1.34</td>
</tr>
<tr>
<td>17</td>
<td>-0.233453</td>
<td>1.</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>235445</td>
<td>1.614843</td>
<td>1025.63</td>
</tr>
<tr>
<td>19</td>
<td>129.00</td>
<td>64.557200</td>
<td>12.07</td>
</tr>
<tr>
<td>20</td>
<td>1239.5475</td>
<td>39.825249</td>
<td>6.905</td>
</tr>
<tr>
<td>21</td>
<td>76.150425</td>
<td>1239.5475</td>
<td>135.00</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>-0.126261</td>
<td>16.75</td>
</tr>
<tr>
<td>23</td>
<td>0100</td>
<td>42.393218</td>
<td>10</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>2014</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>tablet</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>13244</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>31</td>
<td>desktop</td>
<td>4150.30</td>
<td>1025.63</td>
</tr>
<tr>
<td>32</td>
<td>146.81</td>
<td>-0.134786</td>
<td>12.07</td>
</tr>
<tr>
<td>33</td>
<td>84.7000000</td>
<td>49</td>
<td>5</td>
</tr>
<tr>
<td>34</td>
<td>2012</td>
<td>5</td>
<td>1986</td>
</tr>
<tr>
<td>35</td>
<td>6</td>
<td>105.24</td>
<td>340.00</td>
</tr>
<tr>
<td>36</td>
<td>791792</td>
<td>569.29</td>
<td>1</td>
</tr>
<tr>
<td>37</td>
<td>0.045171</td>
<td>569.29</td>
<td>20.00</td>
</tr>
<tr>
<td>38</td>
<td>157.262738</td>
<td>307.743859</td>
<td>20.00</td>
</tr>
<tr>
<td>39</td>
<td>424007</td>
<td>8.983533</td>
<td>0.01</td>
</tr>
<tr>
<td>40</td>
<td>3.470527</td>
<td>3.862210</td>
<td>1320.00</td>
</tr>
<tr>
<td>41</td>
<td>64</td>
<td>56.46</td>
<td>96.581000</td>
</tr>
<tr>
<td>42</td>
<td>0.295458</td>
<td>47.935920</td>
<td>16.00</td>
</tr>
<tr>
<td>43</td>
<td>-0.823347</td>
<td>1.815491</td>
<td>0.00</td>
</tr>
<tr>
<td>44</td>
<td>0.651941</td>
<td>0.459305</td>
<td>0.00</td>
</tr>
<tr>
<td>45</td>
<td>966834</td>
<td>-0.169238</td>
<td>634.84</td>
</tr>
<tr>
<td>46</td>
<td>138.38</td>
<td>76.813125</td>
<td>3</td>
</tr>
<tr>
<td>47</td>
<td>-0.455197</td>
<td>27.839228</td>
<td>3</td>
</tr>
<tr>
<td>48</td>
<td>1037.5750</td>
<td>85.197948</td>
<td>3</td>
</tr>
</tbody>
</table>
Even though the entityset contains the complete transaction history for each customer, only data with a time index up to and including the cutoff time was used to calculate the features above.

**Using a Cutoff Time DataFrame**

Oftentimes, the training examples for machine learning will come from different points in time. To specify a unique cutoff time for each row of the resulting feature matrix, we can pass a dataframe where the first column is the instance id and the second column is the corresponding cutoff time.

**Note:** Only the first two columns are used to calculate features. Any additional columns passed through are appended to the resulting feature matrix. This is typically used to pass through machine learning labels to ensure that they stay aligned with the feature matrix.

```python
In [7]: cutoff_times = pd.DataFrame()
In [8]: cutoff_times['customer_id'] = [1, 2, 3, 1]
In [9]: cutoff_times['time'] = pd.to_datetime(['2014-1-1 04:00', '2014-1-1 04:00', '2014-1-1 04:00', '2014-1-1 04:00'])
```
In [10]: cutoff_times['label'] = [True, True, False, True]

In [11]: cutoff_times
Out[11]:

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2014-01-01 04:00:00</td>
<td>True</td>
</tr>
<tr>
<td>1</td>
<td>2 2014-01-01 05:00:00</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>3 2014-01-01 06:00:00</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>1 2014-01-01 08:00:00</td>
<td>True</td>
</tr>
</tbody>
</table>

In [12]: fm, features = ft.dfs(entityset=es,
   ....: target_entity='customers',
   ....: cutoff_time=cutoff_times,
   ....: cutoff_time_in_index=True)

In [13]: fm
Out[13]:
zip_code  COUNT(transactions)  NUM_UNIQUE(transactions.device)
        MAX(transactions.amount)  SKEW(transactions.amount)  MIN(transactions.amount)
        MEAN(transactions.amount)  COUNT(transactions)  NUM_UNIQUE(transactions.product_id)
        MODE(transactions.product_id)  DAY(date_of_birth)  DAY(join_date)  YEAR(date_of_birth)
        YEAR(join_date)  MONTH(date_of_birth)  MONTH(join_date)  WEEKDAY(date_of_birth)
        WEEKDAY(join_date)  SUM(transactions.MIN(transactions.amount))  SUM(transactions.SUM(transactions.amount))
        SUM(transactions.NUM_UNIQUE(transactions.product_id))  SUM(transactions.SKEW(transactions.amount))
        SUM(transactions.MAX(transactions.amount))  SUM(transactions.STD(transactions.amount))
        STD(sessions.SUM(transactions.amount))  STD(sessions.MAX(transactions.amount))  STD(sessions.NUM_UNIQUE(transactions.product_id))
        STD(sessions.SKEW(transactions.amount))  STD(sessions.MAX(transactions.amount))  STD(sessions.STD(transactions.amount))
        STD(sessions.COUNT(transactions))  STD(sessions.MIN(transactions.amount))
        MEAN(sessions.SUM(transactions.amount))  MEAN(sessions.MAX(transactions.amount))  MEAN(sessions.NUM_UNIQUE(transactions.product_id))
        MEAN(sessions.SKEW(transactions.amount))  MEAN(sessions.MAX(transactions.amount))  MEAN(sessions.STD(transactions.amount))
        MEAN(sessions.COUNT(transactions))  MEAN(sessions.MIN(transactions.amount))
        NUM_UNIQUE(sessions.WEEKDAY(session_start))  NUM_UNIQUE(sessions.MONTH(session_start))  NUM_UNIQUE(sessions.DAY(session_start))
        NUM_UNIQUE(sessions.YEAR(session_start))  NUM_UNIQUE(sessions.MODE(transactions.product_id))
        MODE(sessions.WEEKDAY(session_start))  MODE(sessions.MONTH(session_start))  MODE(sessions.DAY(session_start))
        MODE(sessions.YEAR(session_start))  MODE(sessions.MODE(transactions.product_id))  NUM_UNIQUE(transactions.sessions.customer_id)  MODE(transactions.sessions.device)  label
| 1 | 2014-01-01 04:00:00 | 60091 | 4 |
| 3 | tablet | 4958.19 | 42.309717 |
| 139.23 | -0.006928 | 5.81 |
| 74.002836 | 67 | 5 |
| 4 | 18 | 17 | 1994 |
| 2011 | 7 | 4 | 0 |
| 6 | 27.62 | 304. |
| 601700 | 20 |
| -0.505043 | 540.04 |
| 169.572874 | 271.917637 |
| 1285833 | 10.426572 |
| 0.0 | 0.500353 |
| 5.027226 | 5.678908 | 1613. |
| 93 | 8.74 | 85.469167 |
| -0.234349 | 46.905665 | 25 |
| 1.197406 | -0.233453 |
| 0.0 | -0.451371 |
| 235445 | 1.614843 | 1025.63 |
| 64.557200 | -0.830975 |
| 129.00 | 39.825249 | 12 |
| 1239.5475 | 76.150425 | 13565.6 |
| 1001000 | 42.393218 |
| 1 | 1 |
| 1 | 3 |
### 3.5. Handling Time

The `featuretools` library supports different types of time handling in its feature generation process. When working with time series data, it is important to consider how your features behave over time. Here are some examples of how features can be generated with different time handling strategies:

```python
2 2014-01-01 05:00:00 13244 5
-2 2014-01-01 05:00:00 13244 5
-1 2014-01-01 05:00:00 13244 5
 1 2014-01-01 05:00:00 13244 5
 2 2014-01-01 05:00:00 13244 5
 3 2014-01-01 05:00:00 13244 5
 4 2014-01-01 05:00:00 13244 5
 5 2014-01-01 05:00:00 13244 5
```

For more information, please refer to the featuretools documentation, release 0.13.4.
We can now see that every row of the feature matrix is calculated at the corresponding time in the cutoff time dataframe. Because we calculate each row at a different time, it is possible to have a repeat customer. In this case, we calculated the feature vector for customer 1 at both 04:00 and 08:00.

### 3.5.3 Training Window

By default, all data up to and including the cutoff time is used. We can restrict the amount of historical data that is selected for calculations using a “training window.”

Here’s an example of using a two hour training window:

```python
In [14]: window_fm, window_features = ft.dfs(entityset=es,
                      ..., target_entity="customers",
                      ..., cutoff_time=cutoff_times,
                      ..., cutoff_time_in_index=True,
                      ..., training_window="2 hour")

In [15]: window_fm
Out[15]:
```

zip_code  COUNT(sessions)  NUM_UNIQUE(sessions.
    device)  MODE(sessions.device)  SUM(transactions.amount)  STD(transactions.amount)
    MAX(transactions.amount)  SKEW(transactions.amount)  MIN(transactions.amount)
    MEAN(transactions.amount)  COUNT(transactions)  NUM_UNIQUE(transactions.
    product_id)  MODE(transactions.product_id)  NUM_UNIQUE(transactions.
    product_id)
    DAY(date_of_birth)  YEAR(date_of_birth)  MONTH(date_of_birth)  WEEKDAY(date_of_
    birth)  DAY(join_date)  YEAR(join_date)  MONTH(join_date)  WEEKDAY(join_date)  SUM(sessions.MIN(transactions.amount))  SUM(sessions.
    MIN(transactions.amount))  SUM(sessions.MAX(transactions.amount))  SUM(sessions.
    MAX(transactions.amount))  SUM(sessions.SKEW(transactions.amount))  SUM(sessions.
    SKEW(transactions.amount))  SUM(sessions.STD(transactions.amount))  SUM(sessions.
    STD(transactions.amount))  SUM(sessions.NUM_UNIQUE(transactions.
    product_id))  STD(sessions.MIN(transactions.amount))  STD(sessions.MAX(transactions.amount))  STD(sessions.
    MAX(transactions.amount))  STD(sessions.MEAN(transactions.amount))  STD(sessions.
    MEAN(transactions.amount))  STD(sessions.NUM_UNIQUE(transactions.
    product_id))  STD(sessions.SKEW(transactions.
    amount))  STD(sessions.MAX(transactions.amount))  STD(sessions.COUNT(transactions))
    MAX(sessions.SUM(transactions.amount))  MAX(sessions.MIN(transactions.amount))
    MAX(sessions.MEAN(transactions.amount))  MAX(sessions.NUM_UNIQUE(transactions.
    product_id))  MAX(sessions.SKEW(transactions.amount))  MAX(sessions.
    STD(transactions.amount))  MAX(sessions.COUNT(transactions))  MAX(sessions.
    SUM(transactions.amount))  MAX(sessions.MIN(transactions.amount))  MAX(sessions.
    MAX(transactions.amount))  MAX(sessions.MEAN(transactions.amount))  MAX(sessions.
    NUM_UNIQUE(transactions.product_id))  MAX(sessions.SKEW(transactions.
    product_id))  MAX(sessions.SKEW(transactions.amount))  MAX(sessions.
    STD(transactions.amount))  MAX(sessions.COUNT(transactions))  MAX(sessions.
    SUM(transactions.amount))
    MIN(sessions.MIN(transactions.amount))  MIN(sessions.MAX(transactions.amount))  MIN(sessions.
    MAX(transactions.amount))  MIN(sessions.MEAN(transactions.amount))  MIN(sessions.
    NUM_UNIQUE(transactions.
    product_id))  MIN(sessions.SKEW(transactions.
    amount))  MIN(sessions.MAX(transactions.amount))  MIN(sessions.COUNT(transactions))
    MIN(sessions.SKEW(transactions.amount))  MIN(sessions.MIN(transactions.amount))  MIN(sessions.
    MAX(transactions.amount))  MIN(sessions.MEAN(transactions.amount))  MIN(sessions.
    NUM_UNIQUE(transactions.
    product_id))  MIN(sessions.SKEW(transactions.
    amount))  MIN(sessions.MAX(transactions.amount))  MIN(sessions.
    MIN(transactions.amount))  MIN(sessions.SUM(transactions.amount))  MIN(sessions.
    SUM(transactions.amount))  MIN(sessions.MIN(transactions.amount))  MIN(sessions.
    MAX(transactions.amount))  MIN(sessions.MEAN(transactions.amount))  MIN(sessions.
    NUM_UNIQUE(transactions.
    product_id))  MIN(sessions.SKEW(transactions.
    amount))  MIN(sessions.MAX(transactions.amount))  MIN(sessions.
    STD(transactions.amount))  MIN(sessions.COUNT(transactions))  MIN(sessions.
    SUM(transactions.amount))
    NUM_UNIQUE(sessions.WEEKDAY(session_start))  NUM_UNIQUE(sessions.DAY(session_start))
    NUM_UNIQUE(sessions.YEAR(session_start))  NUM_UNIQUE(sessions.MODE(transactions.
    product_id))  MODE(sessions.WEEKDAY(session_start))  MODE(sessions.DAY(session_start))
    MODE(sessions.YEAR(session_start))  MODE(sessions.MODE(transactions.
    product_id))  NUM_UNIQUE(transactions.sessions.
    device)  MODE(transactions.sessions.
    device)  label
    customer_id  NUM_UNIQUE(transactions.sessions.
    device)  MODE(transactions.sessions.
    device)  label
customer_id  time

(continues on next page)
<table>
<thead>
<tr>
<th></th>
<th>2014-01-01 04:00:00</th>
<th>60091</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>desktop</td>
<td>2077.66</td>
<td>43.772157</td>
</tr>
<tr>
<td>2</td>
<td>139.09</td>
<td>-0.187686</td>
<td>5.81</td>
</tr>
<tr>
<td>3</td>
<td>76.950370</td>
<td>27</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>2011</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>desktop</td>
<td>12.59</td>
<td>155.</td>
</tr>
<tr>
<td>7</td>
<td>604500</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>-0.604638</td>
<td>271.81</td>
<td>86.730914</td>
</tr>
<tr>
<td>9</td>
<td>86.730914</td>
<td>10.842658</td>
<td>0.</td>
</tr>
<tr>
<td>10</td>
<td>0.000000</td>
<td>0.747633</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>4.504270</td>
<td>2.121320</td>
<td>1052.</td>
</tr>
<tr>
<td>12</td>
<td>0.226337</td>
<td>46.905665</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>NaN</td>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>NaN</td>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>70.135333</td>
<td>NaN</td>
<td>1025.63</td>
</tr>
<tr>
<td>16</td>
<td>-0.830975</td>
<td>31</td>
<td>132.</td>
</tr>
<tr>
<td>17</td>
<td>39.825249</td>
<td>12</td>
<td>77.802250</td>
</tr>
<tr>
<td>18</td>
<td>1038.830000</td>
<td>6.295000</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>77.802250</td>
<td>5.</td>
<td>8.</td>
</tr>
<tr>
<td>20</td>
<td>0.302319</td>
<td>109.500185</td>
<td>23.</td>
</tr>
<tr>
<td>21</td>
<td>43.365457</td>
<td>1.0500000</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>2014</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>NaN</td>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>2014</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>2014-01-01 05:00:00</td>
<td>13244</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td>desktop</td>
<td>2605.61</td>
<td>36.077146</td>
</tr>
<tr>
<td>29</td>
<td>146.81</td>
<td>-0.198611</td>
<td>12.07</td>
</tr>
<tr>
<td>30</td>
<td>84.051935</td>
<td>31</td>
<td>5</td>
</tr>
<tr>
<td>31</td>
<td>4</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>32</td>
<td>2012</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>33</td>
<td>6</td>
<td>90.35</td>
<td>253.</td>
</tr>
<tr>
<td>34</td>
<td>240615</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>35</td>
<td>-0.110009</td>
<td>404.04</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>109.500185</td>
<td>203.331699</td>
<td>23.</td>
</tr>
<tr>
<td>37</td>
<td>329038</td>
<td>10.587085</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>0.000000</td>
<td>0.242542</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>14.342521</td>
<td>2.516611</td>
<td>1004.</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>55.46</td>
<td>96.581000</td>
</tr>
<tr>
<td>41</td>
<td>0.130019</td>
<td>47.935920</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>-1.660092</td>
<td>1.397956</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>1.659252</td>
<td>0.</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>-0.1083626</td>
<td>634.84</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>121470</td>
<td>0.585583</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>77.304615</td>
<td>3.</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>5</td>
<td>-0.314918</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>118.85</td>
<td>27.839228</td>
<td>1025.63</td>
</tr>
<tr>
<td>49</td>
<td>608000</td>
<td>36.500062</td>
<td>84.413538</td>
</tr>
<tr>
<td>50</td>
<td>868.536667</td>
<td>5.</td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
3.5. Handling Time

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>ID</th>
<th>Feature</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
<th>Value 4</th>
<th>Value 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-01-01</td>
<td>6:00:00</td>
<td>13244</td>
<td>desktop</td>
<td>1925.82</td>
<td>37.130891</td>
<td>6.65</td>
<td>2003</td>
<td>4</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>08:00:00</td>
<td>60091</td>
<td>mobile</td>
<td>3124.15</td>
<td>477.281339</td>
<td>45.</td>
<td>944.</td>
<td>1</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>10:00:00</td>
<td>60091</td>
<td>mobile</td>
<td>3124.15</td>
<td>477.281339</td>
<td>45.</td>
<td>944.</td>
<td>1</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>12:00:00</td>
<td>60091</td>
<td>mobile</td>
<td>3124.15</td>
<td>477.281339</td>
<td>45.</td>
<td>944.</td>
<td>1</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>14:00:00</td>
<td>60091</td>
<td>mobile</td>
<td>3124.15</td>
<td>477.281339</td>
<td>45.</td>
<td>944.</td>
<td>1</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>16:00:00</td>
<td>60091</td>
<td>mobile</td>
<td>3124.15</td>
<td>477.281339</td>
<td>45.</td>
<td>944.</td>
<td>1</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>18:00:00</td>
<td>60091</td>
<td>mobile</td>
<td>3124.15</td>
<td>477.281339</td>
<td>45.</td>
<td>944.</td>
<td>1</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>20:00:00</td>
<td>60091</td>
<td>mobile</td>
<td>3124.15</td>
<td>477.281339</td>
<td>45.</td>
<td>944.</td>
<td>1</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>22:00:00</td>
<td>60091</td>
<td>mobile</td>
<td>3124.15</td>
<td>477.281339</td>
<td>45.</td>
<td>944.</td>
<td>1</td>
</tr>
</tbody>
</table>

(continued on next page)
We can see that the counts for the same feature are lower after we shorten the training window:

\[
\begin{array}{c|c|c}
\text{In [16]: } & \text{fm["COUNT(transactions)" ]} & \\
\text{Out[16]:} & \text{COUNT(transactions)} & \\
\hline
\text{customer_id} & \text{time} & \\
1 & 2014-01-01 04:00:00 & 67 \\
2 & 2014-01-01 05:00:00 & 62 \\
3 & 2014-01-01 06:00:00 & 44 \\
1 & 2014-01-01 08:00:00 & 126 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{In [17]: } & \text{window_fm["COUNT(transactions)" ]} & \\
\hline
\text{customer_id} & \text{time} & \\
1 & 2014-01-01 04:00:00 & 27 \\
2 & 2014-01-01 05:00:00 & 31 \\
3 & 2014-01-01 06:00:00 & 29 \\
1 & 2014-01-01 08:00:00 & 47 \\
\end{array}
\]

### 3.5.4 Setting a Last Time Index

The training window in Featuretools limits the amount of past data that can be used while calculating a particular feature vector. A row in the entity is filtered out if the value of its time index is either before or after the training window. This works for entities where a row occurs at a single point in time. However, a row can sometimes exist for a duration.

For example, a customer's session has multiple transactions which can happen at different points in time. If we are trying to count the number of sessions a user has in a given time period, we often want to count all the sessions that had any transaction during the training window. To accomplish this, we need to not only know when a session starts, but also when it ends. The last time that an instance appears in the data is stored as the last_time_index of an Entity. We can compare the time index and the last time index of the sessions entity above:

\[
\begin{array}{c}
\text{In [18]: es["sessions"].df["session_start"].head()} & \\
\text{Out[18]:} & \\
1 & 2014-01-01 00:00:00 \\
2 & 2014-01-01 00:17:20 \\
3 & 2014-01-01 00:28:10 \\
4 & 2014-01-01 00:44:25 \\
5 & 2014-01-01 01:11:30 \\
\text{Name: session_start, dtype: datetime64[ns]} & \\
\end{array}
\]

\[
\begin{array}{c}
\text{In [19]: es["sessions"].last_time_index.head()} & \\
\hline
1 & 2014-01-01 00:16:15 \\
2 & 2014-01-01 00:27:05 \\
3 & 2014-01-01 00:43:20 \\
4 & 2014-01-01 01:10:25 \\
5 & 2014-01-01 01:22:20 \\
\text{Name: last_time, dtype: datetime64[ns]} & \\
\end{array}
\]
Featuretools can automatically add last time indexes to every Entity in an Entityset by running EntitySet.add_last_time_indexes(). If a last_time_index has been set, Featuretools will check to see if the last_time_index is after the start of the training window. That, combined with the cutoff time, allows DFS to discover which data is relevant for a given training window.

### 3.5.5 Approximating Features by Rounding Cutoff Times

For each unique cutoff time, Featuretools must perform operations to select the data that’s valid for computations. If there are a large number of unique cutoff times relative to the number of instances for which we are calculating features, the time spent filtering data can add up. By reducing the number of unique cutoff times, we minimize the overhead from searching for and extracting data for feature calculations.

One way to decrease the number of unique cutoff times is to round cutoff times to an earlier point in time. An earlier cutoff time is always valid for predictive modeling — it just means we’re not using some of the data we could potentially use while calculating that feature. So, we gain computational speed by losing a small amount of information.

To understand when an approximation is useful, consider calculating features for a model to predict fraudulent credit card transactions. In this case, an important feature might be, “the average transaction amount for this card in the past”. While this value can change every time there is a new transaction, updating it less frequently might not impact accuracy.

**Note:** The bank BBVA used approximation when building a predictive model for credit card fraud using Featuretools. For more details, see the “Real-time deployment considerations” section of the white paper describing the work involved.

The frequency of approximation is controlled using the approximate parameter to `featuretools.dfs()` or `featuretools.calculate_feature_matrix()`. For example, the following code would approximate aggregation features at 1 day intervals:

```python
fm = ft.calculate_feature_matrix(features=features, entityset=es_transactions, cutoff_time=ct_transactions, approximate="1 day")
```

In this computation, features that can be approximated will be calculated at 1 day intervals, while features that cannot be approximated (e.g. “what is the destination of this flight?”) will be calculated at the exact cutoff time.

### 3.5.6 Secondary Time Index

It is sometimes the case that information in a dataset is updated or added after a row has been created. This means that certain columns may actually become known after the time index for a row. Rather than drop those columns to avoid leaking information, we can create a secondary time index to indicate when those columns become known.

The Flights entityset is a good example of a dataset where column values in a row become known at different times. Each trip is recorded in the trip_logs entity, and has many times associated with it.

```python
In [20]: es_flight = ft.demo.load_flight(nrows=100) Downloading data ...

In [21]: es_flight
```

(continues on next page)
Entities:

- trip_logs [Rows: 100, Columns: 21]
- flights [Rows: 13, Columns: 9]
- airlines [Rows: 1, Columns: 1]
- airports [Rows: 6, Columns: 3]

Relationships:

- trip_logs.flight_id -> flights.flight_id
- flights.carrier -> airlines.carrier
- flights.dest -> airports.dest

In [22]: es_flight['trip_logs'].df.head(3)

<table>
<thead>
<tr>
<th>trip_log_id</th>
<th>flight_id</th>
<th>date_scheduled</th>
<th>scheduled_dep_time</th>
<th>scheduled_arr_time</th>
<th>dep_time</th>
<th>dep_delay</th>
<th>taxi_out</th>
<th>taxi_in</th>
<th>arr_delay</th>
<th>arr_time</th>
<th>dep_delay</th>
<th>carrier_delay</th>
<th>weather_delay</th>
<th>national_airspace_delay</th>
<th>security_delay</th>
<th>late_aircraft_delay</th>
<th>canceled</th>
<th>diverted</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>30</td>
<td>2016-09-03</td>
<td>2017-01-01 13:14:00</td>
<td>2017-01-01 15:05:00</td>
<td>2017-01-01 13:03:00</td>
<td>-11.0</td>
<td>12.0</td>
<td>10.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>2016-09-03</td>
<td>2017-01-01 11:30:00</td>
<td>2017-01-01 15:40:00</td>
<td>2017-01-01 11:24:00</td>
<td>-6.0</td>
<td>28.0</td>
<td>5.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>46</td>
<td>2016-09-03</td>
<td>2017-01-01 09:23:00</td>
<td>2017-01-01 10:42:00</td>
<td>2017-01-01 09:23:00</td>
<td>-2.0</td>
<td>18.0</td>
<td>8.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For every trip log, the time index is `date_scheduled`, which is when the airline decided on the scheduled departure and arrival times, as well as what route will be flown. We don’t know the rest of the information about the actual departure/arrival times and the details of any delay at this time. However, it is possible to know everything about how a trip went after it has arrived, so we can use that information at any time after the flight lands.

Using a secondary time index, we can indicate to Featuretools which columns in our flight logs are known at the time the flight is scheduled, plus which are known at the time the flight lands.

In Featuretools, when creating the entity, we set the secondary time index to be the arrival time like this:

```python
es = ft.EntitySet('Flight Data')
arr_time_columns = ['arr_delay', 'dep_delay', 'carrier_delay', 'weather_delay',
                    'national_airspace_delay', 'security_delay',
                    'late_aircraft_delay', 'canceled', 'diverted',
```
By setting a secondary time index, we can still use the delay information from a row, but only when it becomes known.

**Hint:** It’s often a good idea to use a secondary time index if your entityset has inline labels. If you know when the label would be valid for use, it’s possible to automatically create very predictive features using historical labels.

### Flight Predictions

Let’s make some features at varying times using the flight example described above. Trip 14 is a flight from CLT to PHX on January 31, 2017 and trip 92 is a flight from PIT to DFW on January 1. We can set any cutoff time before the flight is scheduled to depart, emulating how we would make the prediction at that point in time.

We set two cutoff times for trip 14 at two different times: one which is more than a month before the flight and another which is only 5 days before. For trip 92, we’ll only set one cutoff time, three days before it is scheduled to leave.

![Flight predictions diagram]

Our cutoff time dataframe looks like this:

```python
In [23]: ct_flight = pd.DataFrame()
In [24]: ct_flight['trip_log_id'] = [14, 14, 92]
In [25]: ct_flight['time'] = pd.to_datetime(['2016-12-28',
                                          '2017-1-25',
                                          '2016-1-28'])
In [26]: ct_flight['label'] = [True, True, False]
In [27]: ct_flight
Out[27]:
   trip_log_id  time  label
0        14  2016-12-28  True
1        14  2017-1-25  True
2        92  2016-1-28  False
```
Now, let's calculate the feature matrix:

```
In [28]: fm, features = ft.dfs(entityset=es_flight,
                         ....: target_entity='trip_logs',
                         ....: cutoff_time=ct_flight,
                         ....: cutoff_time_in_index=True,
                         ....: agg_primitives=['max'],
                         ....: trans_primitives=['month'],)
```

```
In [29]: fm[['flight_id', 'label', 'flights.MAX(trip_logs.arr_delay)',
          'MONTH(scheduled_dep_time)']]
```

Out [29]:

<table>
<thead>
<tr>
<th>flight_id</th>
<th>label</th>
<th>flights.MAX(trip_logs.arr_delay)</th>
<th>MONTH(scheduled_dep_time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>2016-12-28</td>
<td>AA-494:CLT-&gt;PHX</td>
<td>True</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2017-01-25</td>
<td>AA-494:CLT-&gt;PHX</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>2016-12-28</td>
<td>AA-496:PIT-&gt;DFW</td>
<td>False</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Let's understand the output:

1. A row was made for every id-time pair in `ct_flight`, which is returned as the index of the feature matrix.
2. The output was sorted by cutoff time. Because of the sorting, it's often helpful to pass in a label with the cutoff time dataframe so that it will remain sorted in the same fashion as the feature matrix. Any additional columns beyond `id` and `cutoff_time` will not be used for making features.
3. The column `flights.MAX(trip_logs.arr_delay)` is not always defined. It can only have any real values when there are historical flights to aggregate. Notice that, for trip 14, there wasn’t any historical data when we made the feature a month in advance, but there were flights to aggregate when we shortened it to 5 days. These are powerful features that are often excluded in manual processes because of how hard they are to make.

### 3.5.7 Creating and Flattening a Feature Tensor

The `make_temporal_cutoffs()` function generates a series of equally spaced cutoff times from a given set of cutoff times and instance ids.

This function can be paired with DFS to create and flatten a feature tensor rather than making multiple feature matrices at different delays.

The function takes in the following parameters:

- `instance_ids` (list, pd.Series, or np.ndarray): A list of instances.
- `cutoffs` (list, pd.Series, or np.ndarray): An associated list of cutoff times.
- `window_size` (str or pandas.DateOffset): The amount of time between each cutoff time in the created time series.
• `start (datetime.datetime or pd.Timestamp)`: The first cutoff time in the created time series.

• `num_windows (int)`: The number of cutoff times to create in the created time series.

Only two of the three options `window_size`, `start`, and `num_windows` need to be specified to uniquely determine an equally-spaced set of cutoff times at which to compute each instance.

If your cutoff times are the ones used above:

```
In [30]: cutoff_times
Out[30]:
    customer_id  time     label
   0          1  2014-01-01 04:00:00  True
   1          2  2014-01-01 05:00:00  True
   2          3  2014-01-01 06:00:00  False
   3          1  2014-01-01 08:00:00  True
```

Then passing in `window_size='1h'` and `num_windows=2` makes one row an hour over the last two hours to produce the following new dataframe. The result can be directly passed into DFS to make features at the different time points.

```
In [31]: temporal_cutoffs = ft.make_temporal_cutoffs(cutoff_times['customer_id'],
   ....:   cutoff_times['time'],
   ....:   window_size='1h',
   ....:   num_windows=2)

In [32]: temporal_cutoffs
Out[32]:
   time     instance_id
   0 2014-01-01 03:00:00          1
   1 2014-01-01 04:00:00          1
   2 2014-01-01 04:00:00          2
   3 2014-01-01 05:00:00          2
   4 2014-01-01 05:00:00          3
   5 2014-01-01 06:00:00          3
   6 2014-01-01 07:00:00          1
   7 2014-01-01 08:00:00          1
```

```
In [33]: fm, features = ft.dfs(entityset=es,
   ....:   target_entity='customers',
   ....:   cutoff_time=temporal_cutoffs,
   ....:   cutoff_time_in_index=True)

In [34]: fm
Out[34]:
```
<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>value1</th>
<th>value2</th>
<th>value3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 03:00:00</td>
<td>60091</td>
<td>3932.56</td>
<td>42.769602</td>
</tr>
<tr>
<td>3</td>
<td>139.23</td>
<td>0.140387</td>
<td>5.81</td>
<td></td>
</tr>
<tr>
<td>71.501091</td>
<td></td>
<td>55</td>
<td>171994</td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td>1</td>
<td>18</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>20.84</td>
<td></td>
</tr>
<tr>
<td>132533</td>
<td></td>
<td></td>
<td></td>
<td>219.</td>
</tr>
<tr>
<td>129.747625</td>
<td></td>
<td></td>
<td></td>
<td>0.325932</td>
</tr>
<tr>
<td>571507</td>
<td></td>
<td></td>
<td></td>
<td>400.95</td>
</tr>
<tr>
<td>5.178021</td>
<td></td>
<td></td>
<td></td>
<td>1.</td>
</tr>
<tr>
<td>93</td>
<td></td>
<td></td>
<td></td>
<td>84.440000</td>
</tr>
<tr>
<td>-0.234349</td>
<td></td>
<td></td>
<td></td>
<td>25.</td>
</tr>
<tr>
<td>0.685199</td>
<td></td>
<td></td>
<td></td>
<td>1.552040</td>
</tr>
<tr>
<td>1.173675</td>
<td></td>
<td></td>
<td></td>
<td>0.</td>
</tr>
<tr>
<td>763052</td>
<td></td>
<td></td>
<td></td>
<td>1052.03</td>
</tr>
<tr>
<td>64.557200</td>
<td></td>
<td></td>
<td></td>
<td>1.</td>
</tr>
<tr>
<td>-0.134754</td>
<td></td>
<td></td>
<td></td>
<td>15.</td>
</tr>
<tr>
<td>129.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1310.853333</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73.044178</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td>Value</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>----------------</td>
<td>----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>2014-01-01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>tablet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>60091</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>2014-01-01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>desktop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>13244</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>2014-01-01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>tablet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>2014-01-01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>desktop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>13244</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>2014-01-01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>tablet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.5. Handling Time

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>2014-01-01</td>
<td></td>
</tr>
<tr>
<td>Device</td>
<td>tablet</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>1037.575000</td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>85.197948</td>
<td></td>
</tr>
</tbody>
</table>

49
<table>
<thead>
<tr>
<th>Time</th>
<th>Duration</th>
<th>User</th>
<th>Cpu</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-01-01 05:00:00</td>
<td>13244</td>
<td>5</td>
<td>5155.26</td>
<td>38.047944</td>
</tr>
<tr>
<td>146.81</td>
<td>-0.121811</td>
<td>12.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>83.149355</td>
<td>62</td>
<td>5</td>
<td>0.316873</td>
<td>12.07</td>
</tr>
<tr>
<td>2012</td>
<td>8</td>
<td>18</td>
<td>15</td>
<td>1986</td>
</tr>
<tr>
<td>6</td>
<td>127.06</td>
<td>418.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.295458</td>
<td>5</td>
<td>47.395920</td>
<td>1.959531</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>-0.667265</td>
<td>1.082192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>213518</td>
<td>-0.379092</td>
<td>0.63484</td>
<td></td>
<td>6.65</td>
</tr>
<tr>
<td>58.960000</td>
<td>32</td>
<td>27.839228</td>
<td></td>
<td></td>
</tr>
<tr>
<td>118.85</td>
<td>0.0</td>
<td>58.3619281</td>
<td>25.412000</td>
<td></td>
</tr>
<tr>
<td>628000</td>
<td>38.197555</td>
<td>12.400000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2014</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2014-01-01 05:00:00</td>
<td>13244</td>
<td>2</td>
<td>1886.72</td>
<td>41.199361</td>
</tr>
<tr>
<td>146.81</td>
<td>0.637074</td>
<td>6.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58.960000</td>
<td>32</td>
<td>27.839228</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>11</td>
<td>8</td>
<td>4</td>
<td>118.</td>
</tr>
<tr>
<td>370745</td>
<td>10</td>
<td>1.150043</td>
<td>273.05</td>
<td></td>
</tr>
<tr>
<td>83.432017</td>
<td>2.107178</td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>088944</td>
<td>5.099599</td>
<td>6.65</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.838080</td>
<td>0.0</td>
<td>0.061424</td>
<td></td>
<td>944.</td>
</tr>
<tr>
<td>85</td>
<td>8.19</td>
<td>62.791333</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.618455</td>
<td>5</td>
<td>47.264797</td>
<td>17</td>
<td>NaN</td>
</tr>
<tr>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>941.87</td>
<td></td>
</tr>
<tr>
<td>NaN</td>
<td>NaN</td>
<td>55.579412</td>
<td>126.</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>0.531588</td>
<td>36.167220</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>943.360000</td>
<td>7.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>59.185373</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Chapter 3. Table of contents
### 3.5. Handling Time

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>User</th>
<th>CPU</th>
<th>GPU</th>
<th>Time Stamp</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-01-01</td>
<td>06:00:00</td>
<td>1324</td>
<td>2867</td>
<td>0.318</td>
<td>146.31</td>
<td>6.65</td>
</tr>
<tr>
<td>2011</td>
<td>00:00:00</td>
<td>44</td>
<td>5</td>
<td>1</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td>2013</td>
<td>11:00:00</td>
<td>126.6</td>
<td>290.</td>
<td>16</td>
<td>0.618455</td>
<td>17</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>07:00:00</td>
<td>3</td>
<td>7605</td>
<td>139.4</td>
<td>722323</td>
<td>91.76</td>
</tr>
<tr>
<td>2011</td>
<td>00:00:00</td>
<td>110</td>
<td>5</td>
<td>1</td>
<td>0.640252</td>
<td>25</td>
</tr>
<tr>
<td>2014-01-01</td>
<td>07:00:00</td>
<td>6009</td>
<td>7</td>
<td>7605</td>
<td>139.4</td>
<td>91.76</td>
</tr>
<tr>
<td>2011</td>
<td>00:00:00</td>
<td>110</td>
<td>5</td>
<td>1</td>
<td>0.640252</td>
<td>25</td>
</tr>
</tbody>
</table>

(continues on next page)
3.6 Tuning Deep Feature Synthesis

There are several parameters that can be tuned to change the output of DFS.

```python
In [1]: import featuretools as ft

In [2]: es = ft.demo.load_mock_customer(return_entityset=True)

In [3]: es
Out[3]:
```
Entityset: transactions
Entities:
  transactions [Rows: 500, Columns: 5]
  products [Rows: 5, Columns: 2]
  sessions [Rows: 35, Columns: 4]
  customers [Rows: 5, Columns: 4]
Relationships:
  transactions.product_id -> products.product_id
  transactions.session_id -> sessions.session_id
```
3.6.1 Using “Seed Features”

Seed features are manually defined, problem specific, features a user provides to DFS. Deep Feature Synthesis will then automatically stack new features on top of these features when it can.

By using seed features, we can include domain specific knowledge in feature engineering automation.

In [4]: expensive_purchase = ft.Feature(es["transactions"]["amount"]) > 125

In [5]: feature_matrix, feature_defs = ft.dfs(entityset=es,
                   ...:              target_entity="customers",
                   ...:              agg_primitives=["percent_true"],
                   ...:              seed_features=[expensivePurchase])

In [6]: feature_matrix[['PERCENT_TRUE(transactions.amount > 125)']]

Out[6]:

<table>
<thead>
<tr>
<th>customer_id</th>
<th>PERCENT_TRUE(transactions.amount &gt; 125)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.227848</td>
</tr>
<tr>
<td>4</td>
<td>0.220183</td>
</tr>
<tr>
<td>1</td>
<td>0.119048</td>
</tr>
<tr>
<td>3</td>
<td>0.182796</td>
</tr>
<tr>
<td>2</td>
<td>0.129032</td>
</tr>
</tbody>
</table>

We can now see that PERCENT_TRUE was automatically applied to this boolean variable.

3.6.2 Add “interesting” values to variables

Sometimes we want to create features that are conditioned on a second value before we calculate. We call this extra filter a “where clause”.

By default, where clauses are built using the interesting_values of a variable.

In [7]: es["sessions"]["device"].interesting_values = ["desktop", "mobile", "tablet"]

We then specify the aggregation primitive to make where clauses for using where_primitives

In [8]: feature_matrix, feature_defs = ft.dfs(entityset=es,
                   ...:              target_entity="customers",
                   ...:              agg_primitives=["count", "avg_time_between"],
                   ...:              where_primitives=["count", "avg_time_between"],
                   ...:              trans_primitives=[{}])

In [9]: feature_matrix

Out[9]:

<table>
<thead>
<tr>
<th>zip_code</th>
<th>COUNT(sessions)</th>
<th>AVG_TIME_BETWEEN(sessions.session_start)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>COUNT(transactions)</td>
<td>AVG_TIME_BETWEEN(transactions.transaction_time)</td>
</tr>
<tr>
<td></td>
<td>COUNT(sessions WHERE device = tablet)</td>
<td>COUNT(sessions WHERE device = mobile)</td>
</tr>
<tr>
<td></td>
<td>COUNT(sessions WHERE device = desktop)</td>
<td>AVG_TIME_BETWEEN(sessions.session_start WHERE device = tablet)</td>
</tr>
<tr>
<td></td>
<td>WHERE device = tablet</td>
<td>AVG_TIME_BETWEEN(sessions.session_start WHERE device = mobile)</td>
</tr>
<tr>
<td></td>
<td>mobile</td>
<td>AVG_TIME_BETWEEN(sessions.session_start WHERE device = desktop)</td>
</tr>
<tr>
<td></td>
<td>COUNT(transactions WHERE sessions.device = tablet)</td>
<td>COUNT(transactions WHERE sessions.device = mobile)</td>
</tr>
<tr>
<td></td>
<td>COUNT(transactions WHERE sessions.device = desktop)</td>
<td>AVG_TIME_BETWEEN(transactions.sessions.session_start WHERE sessions.device = tablet)</td>
</tr>
<tr>
<td></td>
<td>AVG_TIME_BETWEEN(sessions.session_start WHERE sessions.device = desktop)</td>
<td>AVG_TIME_BETWEEN(transactions.sessions.session_start WHERE sessions.device = mobile)</td>
</tr>
</tbody>
</table>

3.6. Tuning Deep Feature Synthesis
| customer_id | 5 | 60091 | 6 | 5577.000000 |
|            |   |        |   |            | 363.333333 |
|            |   |        |   |            |            |
|            | 2 |        |   |            | 13942.500000 |
|            |   |        |   |            | 357.000000 |
|            | 500000 | 14 |   | 345.892857 |
|            |   |        |   |            | 0.000000 |
|            |   |        |   |            | 796.714286 |
|            |   |        |   |            | 376.071429 |
|            |   |        |   |            | 65.000000 |
|            |   |        |   |            | 809.714286 |
|            | 4 | 60091 | 8 | 2516.428571 |
|            |   | 109    |   | 168.518519 |
|            |   |        |   | 4.000000 |
|            |   | 3       |   | 3336.666667 |
|            |   |        |   | 163.000000 |
|            | 101852 | 18    |   | 223.108108 |
|            |   |        |   | 0.000000 |
|            |   |        |   | 192.500000 |
|            |   |        |   | 238.918919 |
|            |   |        |   | 65.000000 |
|            |   |        |   | 206.250000 |
|            | 1 | 60091 | 8 | 3305.714286 |
|            |   | 126    |   | 192.920000 |
|            |   |        |   | 3.000000 |
|            |   | 2       |   | 11570.000000 |
|            |   |        |   | 185.000000 |
|            | 120000 | 43    |   | 275.000000 |
|            |   |        |   | 419.404762 |
|            |   |        |   | 420.727273 |
|            |   |        |   | 302.500000 |
|            |   |        |   | 442.619049 |
|            |   |        |   | 438.454545 |
|            | 3 | 13244 | 6 | 5096.000000 |
|            |   |        |   | 287.554348 |
|            |   |        |   | 1.000000 |
|            |   |        |   | 276.000000 |
|            | 956522 | 15    |   | 4745.000000 |
|            |   |        |   | 62.000000 |
|            |   |        |   | 16.000000 |
Now, we have several new potentially useful features. For example, the two features below tell us how many sessions a customer completed on a tablet, and the time between those sessions.

\[
\begin{array}{l}
\text{In [10]:} \\
\text{feature_matrix["COUNT(sessions WHERE device = tablet)", "AVG_TIME_BETWEEN(sessions.session_start WHERE device = tablet)"]}
\end{array}
\]

\[
\begin{array}{l}
\text{Out[10]:} \\
\text{COUNT(sessions WHERE device = tablet) AVG_TIME_BETWEEN(sessions.session_start WHERE device = tablet) customer_id}
\end{array}
\]

\[
\begin{array}{l}
\text{5 1} \\
\text{4 NaN} \\
\text{1 3} \\
\text{3 NaN} \\
\text{2 2}
\end{array}
\]

We can see that customer who only had 0 or 1 sessions on a tablet, had NaN values for average time between such sessions.

### 3.6.3 Encoding categorical features

Machine learning algorithms typically expect all numeric data. When Deep Feature Synthesis generates categorical features, we need to encode them.

\[
\begin{array}{l}
\text{In [11]: feature_matrix, feature_defs = ft.dfs(entityset=es,} \\
\text{....: target_entity="customers"}, \\
\text{....: agg_primitives=["mode"],} \\
\text{....: max_depth=1)}
\end{array}
\]

\[
\begin{array}{l}
\text{In [12]: feature_matrix} \\
\text{Out[12]:} \\
\text{zip_code MODE(sessions.device) DAY(date_of_birth) DAY(join_date) YEAR(date_of_birth) YEAR(join_date) MONTH(date_of_birth) MONTH(join_date) WEEKDAY(date_of_birth) WEEKDAY(join_date)}
\end{array}
\]
This feature matrix contains 2 categorical variables, `zip_code` and `MODE(sessions.device)`. We can use the feature matrix and feature definitions to encode these categorical values. Featuretools offers functionality to apply one hot encoding to the output of DFS.

```
In [13]: feature_matrix_enc, features_enc = ft.encode_features(feature_matrix,  
                      feature_defs)

In [14]: feature_matrix_enc
Out[14]:
```

```
zip_code = 60091  zip_code = 13244  zip_code is unknown  MODE(sessions.
  device) = mobile  MODE(sessions.device) = desktop  MODE(sessions.device) is unknown,
  DAY(date_of_birth) = 18  DAY(date_of_birth) = 28  DAY(date_of_birth) = 21,
  DAY(join_date) = 15  DAY(join_date) = 13  DAY(join_date) = 8  DAY(join_date) is
  unknown  YEAR(date_of_birth) = 2006  YEAR(date_of_birth) = 2003  YEAR(date_of_  
  birth) is unknown  YEAR(join_date) = 2011  YEAR(join_date) = 2012  YEAR(join_date)  
  = 2010  YEAR(join_date) is unknown  MONTH(date_of_birth) = 8  MONTH(date_of_birth)  
  = 7  MONTH(date_of_birth) = 11  MONTH(date_of_birth) is unknown  MONTH(join_date)  
  = 4  MONTH(join_date) = 8  MONTH(join_date) = 7  MONTH(join_date) is unknown,
  WEEKDAY(date_of_birth) = 0  WEEKDAY(date_of_birth) = 5  WEEKDAY(date_of_birth) = 4,
  WEEKDAY(date_of_birth) = 1  WEEKDAY(date_of_birth) is unknown  WEEKDAY(join_date)  
  = 6  WEEKDAY(join_date) = 5  WEEKDAY(join_date) = 4  WEEKDAY(join_date) is unknown
```
5 1 0 0
→ 1 0 0 0
→ 0 0 1 0
→ 0 0 0 1
→ 0 0 0 0
→ 0 0 0 0
→ 0 0 0 0
→ 0 0 0 0

3.6. Tuning Deep Feature Synthesis
The returned feature matrix is now all numeric. Additionally, we get a new set of feature definitions that contain the encoded values.

```
In [15]: print(features_enc)
```

These features can be used to calculate the same encoded values on new data. For more information on feature engineering in production, read *Deployment*.

### 3.7 Specifying Primitive Options

By default, DFS will apply primitives across all entities and columns. This behavior can be altered through a few different parameters. Entities and variables can be optionally ignored or included for an entire DFS run or on a per-primitive basis, enabling greater control over features and less run time overhead.

```
In [1]: from featuretools.tests.testing_utils import make_ecommerce_entityset
In [2]: es = make_ecommerce_entityset()
In [3]: feature_matrix, features_list = ft.dfs(entityset=es,
...:     target_entity='customers',
...:     agg_primitives=['mode'],
...:     trans_primitives=['weekday'])
In [4]: features_list
Out[4]:
[<Feature: cohort>,
 <Feature: age>,
 <Feature: région_id>,
 <Feature: loves_ice_cream>,
 <Feature: cancel_reason>,
 <Feature: engagement_level>]
```
3.7.1 Specifying Options for an Entire Run

The `ignore_entities` and `ignore_variables` parameters of DFS control entities and variables (columns) that should be ignored for all primitives. This is useful for ignoring columns or entities that don’t relate to the problem or otherwise shouldn’t be included in the DFS run.

```python
# ignore the 'log' and 'cohorts' entities entirely
# ignore the 'date_of_birth' variable in 'customers' and the 'device_name' variable in 'sessions'
In [5]: feature_matrix, features_list = ft.dfs(entityset=es,
   ...: target_entity='customers',
   ...: agg_primitives=['mode'],
```

(continues on next page)
DFS completely ignores the 'log' and 'cohorts' entities when creating features. It also ignores the variables 'device_name' and 'date_of_birth' in 'sessions' and 'customers' respectively. However, both of these options can be overridden by individual primitive options in the `primitive_options` parameter.

### 3.7.2 Specifying for Individual Primitives

Options for individual primitives or groups of primitives are set by the `primitive_options` parameter of DFS. This parameter maps any desired options to specific primitives. In the case of conflicting options, options set at this level will override options set at the entire DFS run level, and the include options will always take priority over their ignore counterparts.

#### Specifying Entities for Individual Primitives

Which entities to include/ignore can also be specified for a single primitive or a group of primitives. Entities can be ignored using the `ignore_entities` option in `primitive_options`, while entities to explicitly include are set by the `include_entities` option. When `include_entities` is given, all entities not listed are ignored by the primitive. No variables from any excluded entity will be used to generate features with the given primitive.

```python
# ignore the 'cohorts' and 'log' entities, but only for the primitive 'mode'
# include only the 'customers' entity for the primitives 'weekday' and 'day'
In [7]: feature_matrix, features_list = ft.dfs(entityset=es,
...:                             target_entity='customers',
...:                             agg_primitives=['mode'],
...:                             trans_primitives=['weekday', 'day'],
...:                             primitive_options={
...:                             'mode': {'ignore_entities': ['cohorts', 'log'],
...:                                    'include_entities': ['customers']}
...:                         })
```
In this example, DFS would only use the 'customers' entity for both weekday and day, and would use all entities except 'cohorts' and 'log' for mode.

### Specifying Columns for Individual Primitives

Specific variables (columns) can also be explicitly included/ignored for a primitive or group of primitives. Variables to ignore is set by the `ignore_variables` option, while variables to include is set by `include_variables`. When the `include_variables` option is set, no other variables from that entity will be used to make features with the given primitive.

```python
# Include the variables 'product_id' and 'zipcode', 'device_type', and 'cancel_reason' for 'mean'
# Ignore the variables 'signup_date' and 'cancel_date' for 'weekday'
In [9]: feature_matrix, features_list = ft.dfs(entityset=es,
                     ...: target_entity='customers',
                     ...: agg_primitives=['mode'],
                     ...: trans_primitives=['weekday'],
                     ...: primitive_options={
                     ...:     'mode': {'include_variables': {'log': ['product_id', 'zipcode'],
                     ...:                 'sessions': ['device_type']},
```

(continues on next page)
In [10]: features_list
Out[10]:
[<Feature: cohort>,
 <Feature: age>,
 <Feature: région_id>,
 <Feature: loves_ice_cream>,
 <Feature: cancel_reason>,
 <Feature: engagement_level>,
 <Feature: MODE(sessions.device_type)>,
 <Feature: MODE(log.zipcode)>,
 <Feature: MODE(log.product_id)>,
 <Feature: WEEKDAY(upgrade_date)>,
 <Feature: WEEKDAY(date_of_birth)>,
 <Feature: cohorts.cohort_name>,
 <Feature: régions.language>,
 <Feature: MODE(sessions.MODE(log.product_id))>,
 <Feature: MODE(sessions.MODE(log.zipcode))>,
 <Feature: MODE(log.sessions.device_type)>,
 <Feature: cohorts.MODE(customers.cancel_reason)>,
 <Feature: cohorts.MODE(sessions.device_type)>,
 <Feature: cohorts.MODE(log.zipcode)>,
 <Feature: cohorts.MODE(log.product_id)>,
 <Feature: cohorts.WEEKDAY(cohort_end)>,
 <Feature: régions.MODE(customers.cancel_reason)>,
 <Feature: régions.MODE(sessions.device_type)>,
 <Feature: régions.MODE(log.zipcode)>,
 <Feature: régions.MODE(log.product_id)>]

Here, mode will only use the variables 'product_id' and 'zipcode' from the entity 'log', 'device_type' from the entity 'sessions', and 'cancel_reason' from 'customers'. For any other entity, mode will use all variables. The weekday primitive will use all variables in all entities except for 'signup_date' and 'cancel_date' from the 'customers' entity.

Specifying GroupBy Options

GroupBy Transform Primitives also have the additional options include_groupby_entities, ignore_groupby_entities, include_groupby_variables, and ignore_groupby_variables. These options are used to specify entities and columns to include/ignore as groupings for inputs. By default, DFS only groups by ID columns. Specifying include_groupby_variables overrides this default, and will only group by variables given. On the other hand, ignore_groupby_variables will continue to use only the ID columns, ignoring any variables specified that are also ID columns. Note that if including non-ID columns to group by, the included columns must also be a discrete type.

In [11]: feature_matrix, features_list = ft.dfs(entityset=es,
.....:                               target_entity='log',
(continues on next page)
agg_primitives=[],
trans_primitives=[],
groupby_trans_primitives=['cum_sum',
'cum_count
→'],
primitive_options={
'cum_sum': {'ignore_groupby_→variables': {'log': ['product_id']}},
'cum_count': {'include_groupby_→variables': {'log': ['product_id',
→ 'priority_level']},
'ignore_groupby_→entities': ['sessions']})

In [12]: features_list
Out[12]:
[Feature: session_id>,
Feature: product_id>,
Feature: value>,
Feature: value_2>,
Feature: zipcode>,
Feature: countrycode>,
Feature: subregioncode>,
Feature: value_many_nans>,
Feature: priority_level>,
Feature: purchased>,
Feature: CUM_SUM(value_2) by session_id>,
Feature: CUM_SUM(value) by session_id>,
Feature: CUM_SUM(value_many_nans) by session_id>,
Feature: CUM_COUNT(product_id) by priority_level>,
Feature: CUM_COUNT(product_id) by product_id>,
Feature: CUM_COUNT(session_id) by priority_level>,
Feature: CUM_COUNT(session_id) by product_id>,
Feature: sessions.device_name>,
Feature: sessions.customer_id>,
Feature: sessions.device_type>,
Feature: products.rating>,
Feature: products.department>,
Feature: sessions.customers.cohort>,
Feature: sessions.customers.age>,
Feature: sessions.customers.region_id>,
Feature: sessions.customers.loves_ice_cream>,
Feature: sessions.customers.cancel_reason>,
Feature: sessions.customers.engagement_level>,
Feature: CUM_SUM(products.rating) by sessions.customer_id>,
Feature: CUM_SUM(products.rating) by session_id>,
Feature: CUM_COUNT(sessions.customer_id) by priority_level>,
Feature: CUM_COUNT(sessions.customer_id) by product_id>,
Feature: CUM_COUNT(sessions.customer_id) by products.department>]

We ignore 'product_id' as a groupby for cum_sum but still use any other ID columns in that or any other entity. For 'cum_count', we use only 'product_id' and 'priority_level' as groupbys. Note that cum_sum doesn’t use 'priority_level' because it’s not an ID column, but we explicitly include it for cum_count. Finally, note that specifying groupby options doesn’t affect what features the primitive is applied to. For example, cum_count ignores the entity sessions for groupbys, but the feature <Feature: CUM_COUNT(sessions.

3.7. Specifying Primitive Options

63
customer_id) by product_id> is still made. The groupby is from the target entity log, so the feature is valid given the associated options. To ignore the sessions entity for cum_count, the ignore_entities option for cum_count would need to include sessions.

3.7.3 Specifying for each Input for Multiple Input Primitives

For primitives that take multiple columns as input, such as Trend, the above options can be specified for each input by passing them in as a list. If only one option dictionary is given, it is used for all inputs. The length of the list provided must match the number of inputs the primitive takes.

```python
In [13]: feature_matrix, features_list = ft.dfs(entityset=es,
   ....: target_entity='customers',
   ....: agg_primitives=['trend'],
   ....: trans_primitives=[],
   ....: primitive_options=
   ....:   {'trend': [{'ignore_variables':
   ....:       'log': ['value_many_nans']},
   ....:       {'include_variables':
   ....:       'customers': ['signup_date'],
   ....:       'log': ['datetime']})
```

```python
In [14]: features_list
```

```
[<Feature: cohort>,
 <Feature: age>,
 <Feature: région_id>,
 <Feature: loves_ice_cream>,
 <Feature: cancel_reason>,
 <Feature: engagement_level>,
 <Feature: TREND(log.value, datetime)>,
 <Feature: TREND(log.value_2, datetime)>,
 <Feature: cohorts.cohort_name>,
 <Feature: régions.language>,
 <Feature: cohorts.TREND(customers.age, signup_date)>,
 <Feature: cohorts.TREND(log.value, datetime)>,
 <Feature: cohorts.TREND(log.value_2, datetime)>,
 <Feature: régions.TREND(customers.age, signup_date)>,
 <Feature: régions.TREND(log.value, datetime)>,
 <Feature: régions.TREND(log.value_2, datetime)>]
```

Here, we pass in a list of primitive options for trend. We ignore the variable 'value_many_nans' for the first input to trend, and include the variables 'signup_date' from 'customers' for the second input.

3.8 Improving Computational Performance

Feature engineering is a computationally expensive task. While Featuretools comes with reasonable default settings for feature calculation, there are a number of built-in approaches to improve computational performance based on dataset and problem specific considerations.
3.8.1 Reduce number of unique cutoff times

Each row in a feature matrix created by Featuretools is calculated at a specific cutoff time that represents the last point in time that data from any entity in an entity set can be used to calculate the feature. As a result, calculations incur an overhead in finding the subset of allowed data for each distinct time in the calculation.

Note: Featuretools is very precise in how it deals with time. For more information, see Handling Time.

If there are many unique cutoff times, it is often worthwhile to figure out how to have fewer. This can be done manually by figuring out which unique times are necessary for the prediction problem or automatically using approximate.

3.8.2 Adjust chunk size

By default, Featuretools calculates rows with the same cutoff time simultaneously. The chunk_size parameter limits the maximum number of rows that will be grouped and then calculated together. If calculation is done using parallel processing, the default chunk size is set to be $1 / n_{jobs}$ to ensure the computation can be spread across available workers. Normally, this behavior works well, but if there are only a few unique cutoff times it can lead to higher peak memory usage (due to more intermediate calculations stored in memory) or limited parallelism (if the number of chunks is less than $n_{jobs}$).

By setting chunk_size, we can limit the maximum number of rows in each group to specific number or a percentage of the overall data when calling ft.dfs or ft.calculate_feature_matrix:

```python
# use maximum 100 rows per chunk
feature_matrix, features_list = ft.dfs(entityset=es, target_entity="customers", chunk_size=100)
```

We can also set chunk size to be a percentage of total rows:

```python
# use maximum 5% of all rows per chunk
feature_matrix, features_list = ft.dfs(entityset=es, target_entity="customers", chunk_size=.05)
```

3.8.3 Partition and Distribute Data

When an entire dataset is not required to calculate the features for a given set of instances, we can split the data into independent partitions and calculate on each partition. For example, imagine we are calculating features for customers and the features are “number of other customers in this zip code” or “average age of other customers in this zip code”. In this case, we can load in data partitioned by zip code. As long as we have all of the data for a zip code when calculating, we can calculate all features for a subset of customers.

An example of this approach can be seen in the Predict Next Purchase demo notebook. In this example, we partition data by customer and only load a fixed number of customers into memory at any given time. We implement this easily using Dask, which could also be used to scale the computation to a cluster of computers. A framework like Spark could be used similarly.

An additional example of partitioning data to distribute on multiple cores or a cluster using Dask can be seen in the Featuretools on Dask notebook. This approach is detailed in the Parallelizing Feature Engineering with Dask article on the Feature Labs engineering blog. Dask allows for simple scaling to multiple cores on a single computer or multiple machines on a cluster.
For a similar partition and distribute implementation using Apache Spark with PySpark, refer to the Feature Engineering on Spark notebook. This implementation shows how to carry out feature engineering on a cluster of EC2 instances using Spark as the distributed framework. A write-up of this approach is described in the Featuretools on Spark article on the Feature Labs engineering blog.

### 3.8.4 Running Featuretools with Spark and Dask

The Featuretools development team is continually working to improve integration with Dask and Spark for performing feature engineering at scale. If you have a big data problem and are interested in testing our latest Dask or Spark integrations for free, please let us know by completing this simple request form.

### 3.8.5 Featuretools Enterprise

If you don’t want to build it yourself, Featuretools Enterprise has native integrations with Apache Spark and Dask. More information is available here.

### 3.9 Parallel Feature Computation

Featuretools can optionally compute features on multiple cores. The simplest way to control the amount of parallelism is to specify the `n_jobs` parameter:

```python
fm = ft.calculate_feature_matrix(features=features, 
                                entityset=entityset, 
                                cutoff_time=cutoff_time, 
                                n_jobs=2, 
                                verbose=True)
```

The above command will start 2 processes to compute chunks of the feature matrix in parallel. Each process receives its own copy of the entity set, so memory use will be proportional to the number of parallel processes. Because the entity set has to be copied to each process, there is overhead to perform this operation before calculation can begin. To avoid this overhead on successive calls to `calculate_feature_matrix`, read the section below on using a persistent cluster.

### 3.9.1 Running Featuretools with Spark and Dask

The Featuretools development team is continually working to improve integration with Dask and Spark for performing feature engineering at scale. If you have a big data problem and are interested in testing our latest Dask or Spark integrations for free, please let us know by completing this simple request form.

Continue reading below to learn how to perform parallel feature computation with the current integrations.

### 3.9.2 Using persistent cluster

Behind the scenes, Featuretools uses dask’s distributed scheduler to implement multiprocessing. When you only specify the `n_jobs` parameter, a cluster will be created for that specific feature matrix calculation and destroyed once calculations have finished. A drawback of this is that each time a feature matrix is calculated, the entity set has to be transmitted to the workers again. To avoid this, we would like to reuse the same cluster between calls. The way to do this is by creating a cluster first and telling featuretools to use it with the `dask_kwargs` parameter:
import featuretools as ft
from dask.distributed import LocalCluster

cluster = LocalCluster()
fm_1 = ft.calculate_feature_matrix(features=features_1,
                                 entityset=entityset,
                                 cutoff_time=cutoff_time,
                                 dask_kwargs={'cluster': cluster},
                                 verbose=True)

The ‘cluster’ value can either be the actual cluster object or a string of the address the cluster’s scheduler can be reached at. The call below would also work. This second feature matrix calculation will not need to resend the entityset data to the workers because it has already been saved on the cluster:

fm_2 = ft.calculate_feature_matrix(features=features_2,
                                 entityset=entityset,
                                 cutoff_time=cutoff_time,
                                 dask_kwargs={'cluster': cluster.scheduler.address},
                                 verbose=True)

Note: When using a persistent cluster, Featuretools publishes a copy of the EntitySet to the cluster the first time it calculates a feature matrix. Based on the EntitySet’s metadata the cluster will reuse it for successive computations. This means if two EntitySets have the same metadata but different row values (e.g. new data is added to the EntitySet), Featuretools won’t recopy the second EntitySet in later calls. A simple way to avoid this scenario is to use a unique EntitySet id.

3.9.3 Using the distributed dashboard

Dask.distributed has a web-based diagnostics dashboard that can be used to analyze the state of the workers and tasks. It can also be useful for tracking memory use or visualizing task run-times. An in-depth description of the web interface can be found here.
The dashboard requires an additional python package, bokeh, to work. Once bokeh is installed, the web interface will be launched by default when a LocalCluster is created. The cluster created by featuretools when using `n_jobs` does not enable the web interface automatically. To do so, the port to launch the main web interface on must be specified in `dask_kwargs`:

```python
fm = ft.calculate_feature_matrix(features=features,
                                 entityset=entityset,
                                 cutoff_time=cutoff_time,
                                 n_jobs=2,
                                 dask_kwargs={'diagnostics_port': 8787}
                                 verbose=True)
```

### 3.9.4 Parallel Computation by Partitioning Data

As an alternative to Featuretools’ parallelization, the data can be partitioned and the feature calculations run on multiple cores or a cluster using Dask or Apache Spark with PySpark. This approach may be necessary with a large `EntitySet` because the current parallel implementation sends the entire `EntitySet` to each worker which may exhaust the worker memory. For more information on partitioning the data and using Dask or Spark, see *Improving*
Computational Performance. Dask and Spark allow Featuretools to scale to multiple cores on a single machine or multiple machines on a cluster.

## 3.10 Deployment

Deployment of machine learning models requires repeating feature engineering steps on new data. In some cases, these steps need to be performed in near real-time. Featuretools has capabilities to ease the deployment of feature engineering.

### 3.10.1 Saving Features

First, let’s build some generate some training and test data in the same format. We use a random seed to generate different data for the test.

**Note:** Features saved in one version of Featuretools are not guaranteed to load in another. This means the features might need to be re-created after upgrading Featuretools.

```python
In [1]: import featuretools as ft

In [2]: es_train = ft.demo.load_mock_customer(return_entityset=True)

In [3]: es_test = ft.demo.load_mock_customer(return_entityset=True, random_seed=33)

In [4]: feature_matrix, feature_defs = ft.dfs(entityset=es_train, target_entity="customers")

In [5]: feature_matrix_enc, features_enc = ft.encode_features(feature_matrix, feature_defs)

In [6]: feature_matrix_enc

Out[6]:
    zip_code = 60091  zip_code = 13244  zip_code is unknown  COUNT(sessions)
    NUM_UNIQUE(sessions.device)  MODE(sessions.device) = mobile  MODE(sessions.device) = desktop
    STD(transactions.amount)  MAX(transactions.amount)  SKEW(transactions.amount)
    MIN(transactions.amount)  MEAN(transactions.amount)  COUNT(transactions)
    NUM_UNIQUE(transactions.product_id)  MODE(transactions.product_id) = 4
    MODE(transactions.product_id) = 5  MODE(transactions.product_id) = 2
    MODE(transactions.product_id) = 1  MODE(transactions.product_id) is unknown
    DAY(date_of_birth) = 18  DAY(date_of_birth) = 28  DAY(date_of_birth) = 21
    DAY(date_of_birth) = 15  DAY(date_of_birth) is unknown  DAY(join_date) = 17
    DAY(join_date) = 13  DAY(join_date) = 8  DAY(join_date) is unknown
    DAY(join_date) = 2011  YEAR(join_date) = 2012  YEAR(join_date) = 2010
    MONTH(join_date) is unknown  MONTH(join_date) = 8  MONTH(join_date) = 7
    WEEKDAY(join_date) is unknown  WEEKDAY(join_date) = 4  WEEKDAY(join_date) = 5
    WEEKDAY(date_of_birth) = 1  WEEKDAY(date_of_birth) is unknown  WEEKDAY(join_date) = 6
    WEEKDAY(join_date) = 5  WEEKDAY(join_date) is unknown  SUM(sessions.
    NUM_UNIQUE(transactions.product_id))  SUM(sessions.SKEW(transactions.amount))
    SUM(sessions.MAX(transactions.amount))  SUM(sessions.STD(transactions.amount))
    STD(sessions.SUM(transactions.amount))  STD(sessions.MIN(transactions.amount))
    STD(sessions.MEAN(transactions.amount))  STD(sessions.NUM_UNIQUE(transactions.
```

(continues on next page)
<table>
<thead>
<tr>
<th>customer_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>44.095630</td>
</tr>
<tr>
<td>7.55</td>
</tr>
</tbody>
</table>

(continues on next page)
<table>
<thead>
<tr>
<th>3</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>43.683296</td>
<td>149.15</td>
<td>0.418230</td>
<td>0.438</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.89</td>
<td>67.060430</td>
<td>93</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>66.21</td>
<td>405.237462</td>
<td>2.286086</td>
<td>2.286</td>
<td></td>
<td></td>
</tr>
<tr>
<td>847.63</td>
<td>257.299895</td>
<td>5.424407</td>
<td>0.408230</td>
<td></td>
<td></td>
</tr>
<tr>
<td>219.021420</td>
<td>5.424407</td>
<td>0.408230</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.174282</td>
<td>0.429374</td>
<td>10.724241</td>
<td>2.428992</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.678544</td>
<td>-2.449490</td>
<td>0.429374</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.941078</td>
<td>-0.245703</td>
<td>0.429374</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.507217</td>
<td>889.21</td>
<td>55.579412</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55.579412</td>
<td>889.21</td>
<td>55.579412</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.06</td>
<td>82.109444</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50.110120</td>
<td>82.109444</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.246479</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.678544</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1.507217</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55.579412</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.289466</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35.704680</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1039.436667</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67.539577</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.381014</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42.883316</td>
<td>0.854976</td>
<td>0.854976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>37.705178</td>
<td>146.81</td>
<td>0.035353</td>
<td>0.035353</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.73</td>
<td>77.423666</td>
<td>93</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Now, we can use `featuretools.save_features()` to save a list of features to a json file

```python
In [7]: ft.save_features(features_enc, "feature_definitions.json")
```

### 3.10.2 Calculating Feature Matrix for New Data

We can use `featuretools.load_features()` to read in a list of saved features to calculate for our new entity set.

```python
In [8]: saved_features = ft.load_features('feature_definitions.json')
```

After we load the features back in, we can calculate the feature matrix.

```python
In [10]: feature_matrix = ft.calculate_feature_matrix(saved_features, es_test)
```

```python
Out[10]:
zip_code = 60091  zip_code = 13244  zip_code is unknown COUNT(sessions)  
  zip_code = desktop  zip_code = mobile  zip_code is unknown COUNT(sessions)  
  NUM_UNIQUE(sessions.device)  MODE(sessions.device) = mobile  MODE(sessions.device)  
  = desktop  MODE(sessions.device) is unknown SUM(transactions.amount)  
  STD(transactions.amount)  MAX(transactions.amount)  SKEW(transactions.amount)  
  MIN(transactions.amount)  MEAN(transactions.amount)  COUNT(transactions)  
  NUM_UNIQUE(transactions.product_id)  MODE(transactions.product_id) = 4  
  MODE(transactions.product_id) = 5  MODE(transactions.product_id) = 2  
  MODE(transactions.product_id) = 1  MODE(transactions.product_id) is unknown  
  DAY(date_of_birth) = 18  DAY(date_of_birth) = 28  DAY(date_of_birth) = 21  
  DAY(date_of_birth) = 15  DAY(date_of_birth) is unknown  DAY(join_date) = 17  
  DAY(join_date) = 13  DAY(join_date) = 8  DAY(join_date) is unknown  
  YEAR(date_of_birth) = 1986  YEAR(date_of_birth) = 1984  YEAR(date_of_birth) is unknown  
  YEAR(join_date) = 2011  YEAR(join_date) = 2012  YEAR(join_date) = 2010  
  YEAR(join_date) is unknown  MONTH(date_of_birth) = 8  MONTH(date_of_birth) = 7  
  MONTH(date_of_birth) = 11  MONTH(date_of_birth) is unknown  MONTH(join_date) = 4  
  MONTH(join_date) = 8  MONTH(join_date) = 7  MONTH(join_date) is unknown  
  WEEKDAY(date_of_birth) = 0  WEEKDAY(date_of_birth) = 5  WEEKDAY(date_of_birth) = 4  
  WEEKDAY(date_of_birth) = 1  WEEKDAY(date_of_birth) is unknown  WEEKDAY(join_date) = 6  
  WEEKDAY(join_date) = 5  WEEKDAY(join_date) is unknown  SUM(sessions)  
  NUM_UNIQUE(transactions.product_id)  SUM(sessions.MEAN(transactions.amount))  
  SUM(sessions)  SUM(sessions.MAX(transactions.amount))  SUM(sessions.STD(transactions.amount))  
  STD(sessions.SUM(transactions.amount))  STD(sessions.MIN(transactions.amount))  
  STD(sessions.MEAN(transactions.amount))  STD(sessions.NUM_UNIQUE(transactions.product_id))  
  STD(sessions.product_id)  STD(sessions.SKEW(transactions.amount))  STD(sessions)  
  MAX(transactions.amount)  STD(sessions.COUNT(transactions))  MAX(sessions)  
  SUM(transactions.amount)  MAX(sessions.MIN(transactions.amount))  MAX(sessions)  
  NUM_UNIQUE(transactions.productId)  MAX(sessions.NUM_UNIQUE(transactions.productId))  
  MAX(sessions.SKEW(transactions.amount))  MAX(sessions.STD(transactions.amount))  
  MAX(sessions.COUNT(transactions))  SKEW(sessions.SUM(transactions.amount))  
  MIN(sessions.SKEW(transactions.amount))  SKEW(sessions.MEAN(transactions.amount))  
  SKEW(sessions.NUM_UNIQUE(transactions.productId))  SKEW(sessions.MAX(transactions.amount))  
  SKEW(sessions.SKEW(transactions.amount))  SKEW(sessions)  
  COUNT(transactions)  MIN(sessions.SUM(transactions.amount))  MIN(sessions)  
  MIN(sessions)  MIN(sessions.MIN(transactions.amount))  MIN(sessions.MAX(transactions.amount))  
  MIN(sessions.SKEW(transactions.amount))  MIN(sessions.MAX(transactions.amount))  
  MIN(sessions.SKEW(transactions.amount))  MIN(sessions.MIN(transactions.amount))  
  MIN(sessions.STD(transactions.amount))  MIN(sessions.COUNT(transactions))  
  MIN(sessions.SUM(transactions.amount))  MIN(sessions.MIN(transactions.amount))  
  COUNT(sessions)  NUM_UNIQUE(sessions.WEEKDAY(session_start))  
  UNIQUE(sessions.MONTH(join_date))  NUM_UNIQUE(sessions.DAY(session_start))  
  NUM_UNIQUE(sessions.DAY(session_start))  NUM_UNIQUE(sessions.WEEKDAY(session_start))  
  NUM_UNIQUE(sessions.WEEKDAY(session_start))  NUM_UNIQUE(sessions.DAY(session_start))  
  NUM_UNIQUE(sessions.weekday(session_start))  NUM_UNIQUE(sessions.MEAN(transactions.amount))  
  MAX(sessions.NUM_UNIQUE(transactions.productId))  MAX(sessions.MEAN(transactions.amount))  
  MAX(sessions.NUM_UNIQUE(transactions.productId))  MAX(sessions.MEAN(transactions.amount))  
  MAX(sessions.NUM_UNIQUE(transactions.productId))  MAX(sessions.MEAN(transactions.amount))  
  MAX(sessions.NUM_UNIQUE(transactions.productId))  MAX(sessions.MEAN(transactions.amount))  
  MAX(sessions.NUM_UNIQUE(transactions.productId))  MAX(sessions.MEAN(transactions.amount))  
  MAX(sessions.NUM_UNIQUE(transactions.productId))  MAX(sessions.MEAN(transactions.amount))  
(continues on next page)
<table>
<thead>
<tr>
<th>customer_id</th>
<th>True</th>
<th>False</th>
<th>False</th>
<th>False</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>9</td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td>10178.43</td>
</tr>
<tr>
<td>36.523849</td>
<td>147.55</td>
<td>-0.179621</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.19</td>
<td>80.781190</td>
<td>126</td>
<td></td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>5</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>193.08</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>733.862898</td>
<td>-1.797214</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1180.90</td>
<td>319.497611</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>333.923377</td>
<td>17.851716</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.875823</td>
<td>0.440959</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.324894</td>
<td>11.457114</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.272002</td>
<td>1650.65</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60.29</td>
<td>104.565000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41.627134</td>
<td>0.417250</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>-0.385392</td>
<td>1.490781</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.078619</td>
<td>-1.619848</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.230847</td>
<td>-1.743267</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.086578</td>
<td>557.32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69.665000</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.624344</td>
<td>118.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22.026552</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1130.936667</td>
<td>21.453333</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>81.540322</td>
<td>4.777778</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-0.199690</td>
<td>131.21111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35.499735</td>
<td>14.000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>3</td>
<td>True</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
</tbody>
</table>
As you can see above, we have the exact same features as before, but calculated using the test data.

### 3.10.3 Exporting Feature Matrix

**Save as csv**

The feature matrix is a pandas dataframe that we can save to disk

```python
In [11]: feature_matrix.to_csv("feature_matrix.csv")
```

We can also read it back in as follows:

```python
In [12]: saved_fm = pd.read_csv("feature_matrix.csv", index_col="customer_id")
In [13]: saved_fm
```

(continued on next page)
<table>
<thead>
<tr>
<th>True</th>
<th>False</th>
<th>False</th>
<th>False</th>
<th>6.47</th>
<th>79.128364</th>
<th>73</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td></td>
<td></td>
<td>41.998795</td>
<td>147.64</td>
<td>-0.173042</td>
</tr>
<tr>
<td>5</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>64.238824</td>
<td>2</td>
<td>119.86</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>3.741661</td>
<td>1418.99</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>124.58</td>
<td>136.6725</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>5</td>
<td>0.164928</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>51.192518</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>0.130163</td>
<td>2.290693</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.796000</td>
<td>-1.206607</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1.165955</td>
<td>-1.336611</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-0.243198</td>
<td>546.69</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>64.238824</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>9877.67</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>39.913352</td>
<td>148.34</td>
<td>0.040395</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8.00</td>
<td>76.571085</td>
<td>129</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
3.11 Advanced Custom Primitives Guide

3.11.1 Functions With Additional Arguments

One caveat with the makePrimitive functions is that the required arguments of function must be input features. Here we create a function for StringCount, a primitive which counts the number of occurrences of a string in a Text input. Since string is not a feature, it needs to be a keyword argument to string_count.

```python
In [1]: def string_count(column, string=None):
    ...:     '''Count the number of times the value string occurs'''
    ...:     assert string is not None, "string to count needs to be defined"
    ...:     counts = [element.lower().count(string) for element in column]
    ...:     return counts

In order to have features defined using the primitive reflect what string is being counted, we define a custom generate_name function.

```python
In [2]: def string_count_generate_name(self, base_feature_names):
    ...:     return u'STRING_COUNT(%s, "%s")' % (base_feature_names[0], self.kwargs['string'])

Now that we have the function, we create the primitive using the make_trans_primitive function.

```python
In [3]: StringCount = make_trans_primitive(function=string_count,
    ...:     input_types=[Text],
    ...:     return_type=Numeric,
    ...:     cls_attributes={"generate_name": string_count_generate_name})

Passing in string="test" as a keyword argument when initializing the StringCount primitive will make “test” the value used for string when string_count is called to calculate the feature values. Now we use this primitive to define features and calculate the feature values.

```python
In [4]: from featuretools.tests.testing_utils import make_ecommerce_entityset
In [5]: es = make_ecommerce_entityset()
In [6]: feature_matrix, features = ft.dfs(entityset=es,
    ...:     target_entity="sessions",
    ...:     agg_primitives=['sum', 'mean', 'std'],
    ...:     trans_primitives=[StringCount(string="the")])
In [7]: feature_matrix.columns
Out[7]: Index(["device_name", 'customer_id', 'device_type', 'SUM(log.value_2)',
    ...:     'SUM(log.value)', 'SUM(log.value_many_nans)', 'MEAN(log.value_2)', 'MEAN(log.value)
    ...:     ', 'MEAN(log.value_many_nans)', 'STD(log.value_2)', 'STD(log.value)', 'STD(log.
    ...:     value_many_nans)', 'customers.cohort', 'customers.age', 'customers.region_id',
    ...:     'customers.loves_ice_cream', 'customers.cancel_reason', 'customers.engagement_level',
    ...:     'SUM(log.STRING_COUNT(comments, "the"))', 'SUM(log.products.rating)', 'MEAN(log.
    ...:     STRING_COUNT(comments, "the"))', 'STD(log.products.rating)', 'customers.SUM(log.value_2)',
    ...:     'customers.SUM(log.value)', 'customers.SUM(log.value_many_nans)', 'customers.
    ...:     MEAN(log.value_2)', 'customers.MEAN(log.value)', 'customers.MEAN(log.value_many_
    ...:     nans)', 'customers.STD(log.value_2)', 'customers.STD(log.value)', 'customers.
    ...:     STD(log.value_many_nans)', 'customers.STRING_COUNT(favorite_quote, "the")',
    ...:     'customers.cohorts.cohort_name', 'customers.region_id.language'], dtype='object')
```

(continues on next page)
3.11.2 Features with Multiple Outputs

With the `make_primitive` functions, it is possible to have multiple columns output from a single feature. In order to do that, the output must be formatted as a list of arrays/series where each item in the list corresponds to an output from the primitive. In each of these list items (either arrays or series), there must be one element for each input element.

Take, for example, a primitive called `case_count`. For each given string, this primitive outputs the number of uppercase and the number of lowercase letters. So, this primitive must return a list with 2 elements, one corresponding to the number of lowercase letters and one corresponding to the number of uppercase letters. Each element in the list is a series/array having the same number of elements as the number of input strings. Below you can see this example in action, as well as the proper way to specify multiple outputs in the `make_trans_primitive` function.

```python
In [9]: def case_count(array):
    ...:     '''Return the count of upper case and lower case letters in text'''
    ...:     upper = np.array([len(re.findall('[A-Z]', i)) for i in array])
    ...:     lower = np.array([len(re.findall('[a-z]', i)) for i in array])
    ...:     ret = [upper, lower]
    ...:     return ret
    ...
```

We must use the `num_output_features` attribute to specify the number of outputs when creating the primitive using the `make_trans_primitive` function.

```python
In [10]: CaseCount = make_trans_primitive(function=case_count,
    ...:     input_types=[Text],
    ...:     return_type=Numeric,
    ...:     number_output_features=2)
    ...
```

When we call `dfs` on this entityset, there are 6 instances (one for each of the strings in the dataset) of our two created
features in this feature matrix.

```python
In [12]: feature_matrix, features = ft.dfs(entityset=es,
   ....:    target_entity="sessions",
   ....:    agg_primitives=[],
   ....:    trans_primitives=[CaseCount])

In [13]: feature_matrix.columns
Out[13]:
Index(['device_name', 'customer_id', 'device_type', 'customers.cohort',
      'customers.age', 'customers.région_id', 'customers.loves_ice_cream',
      'customers.cancel_reason', 'customers.engagement_level',
      'customers.cohorts.cohort_name',
      'customers.régions.language',
      'customers.CASE_COUNT(favorite_quote)[0]',
      'customers.CASE_COUNT(favorite_quote)[1]'],
   dtype='object')

In [14]: feature_matrix[['customers.CASE_COUNT(favorite_quote)[0]',
   ....:   'customers.CASE_COUNT(favorite_quote)[1]']]
```

### 3.12 Frequently Asked Questions

Here we are attempting to answer some commonly asked questions that appear on Github, and Stack Overflow.

[1]:
```python
import featuretools as ft
import pandas as pd
import numpy as np
```

2020-03-27 18:15:52,815 featuretools - WARNING Featuretools failed to load plugin nlp_primitives from library nlp_primitives.__init__. For a full stack trace, set logging to debug.

[2]:
```python
es = ft.demo.load_mock_customer(return_entityset=True)
es
```

**3.12.1 EntitySet**

**How do I get a list of variable (column) names, and types in an EntitySet?**

After you create your EntitySet, you may wish to view the column names. An EntitySet contains multiple Dataframes, one for each entity.
sessions [Rows: 35, Columns: 4]  
customers [Rows: 5, Columns: 4]  
Relationships:  
transactions.product_id -> products.product_id  
transactions.session_id -> sessions.session_id  
sessions.customer_id -> customers.customer_id

If you want view the variables (columns), and types for the “transactions” entity, you can do the following:

```
[3]: es['transactions'].variables

[3]: [<Variable: transaction_id (dtype = index)>,
    <Variable: session_id (dtype = id)>,
    <Variable: transaction_time (dtype: datetime_time_index, format: None)>,
    <Variable: amount (dtype = numeric)>,
    <Variable: product_id (dtype = id)>]
```

If you want to view the underlying Dataframe, you can do the following:

```
[4]: es['transactions'].df.head()

[4]:     transaction_id  session_id  transaction_time          amount  product_id
       298             298  2014-01-01 00:00:00  127.64           5
       2               2  2014-01-01 00:01:05  109.48           2
      308             308  2014-01-01 00:02:10   95.06           3
      116             116  2014-01-01 00:03:15   78.92           4
      371             371  2014-01-01 00:04:20   31.54           3
```

**What is the difference between copy_variables and additional_variables?**

The function `normalize_entity` creates a new entity and a relationship from unique values of an existing entity. It takes 2 similar arguments:

- `additional_variables` removes variables from the base entity and moves them to the new entity.
  - `copy_variables` keeps the given variables in the base entity, but also copies them to the new entity.

```
[5]: data = ft.demo.load_mock_customer()  
transactions_df = data['transactions'].merge(data['sessions']).merge(data['customers'])  
products_df = data['products']

es = ft.EntitySet(id="customer_data")  
es = es.entity_from_dataframe(entity_id="transactions",
    dataframe=transactions_df,
    index="transaction_id",
    time_index="transaction_time")

es = es.entity_from_dataframe(entity_id="products",
    dataframe=products_df,
    index="product_id")

new_relationship = ft.Relationship(es["products"]['product_id'], es["transactions"]['product_id'])  
es = es.add_relationship(new_relationship)
```

Before we normalize to create a new entity, let’s look at base entity
Notice the columns `session_id`, `session_start`, `join_date`, `device`, `customer_id`, and `zip_code`.

Above, we normalized the columns to create a new entity. - For `additional_variables`, the following column `['join_date']` will be removed from the `products` entity, and moved to the new `device` entity.

- For `copy_variables`, the following columns `['device', 'customer_id', 'zip_code', 'session_start']` will be copied from the `products` entity to the new `device` entity.

Let’s see this in the actual `EntitySet`.

Notice above how `['device', 'customer_id', 'zip_code', 'session_start']` are still in the `transactions` entity, while `['join_date']` is not. But, they have all been moved to the `sessions` entity, as
Why did variable type change to Id, Index, or datetime_time_index?

During the creation of your EntitySet, you might be wondering why your variable type changed.

Notice how the variable type of session_id is Numeric, and the variable type of session_start is Datetime.

Now, let’s normalize the transactions entity to create a new entity.

The type for session_id is now Id in the transactions entity, and Index in the new entity, sessions. This is the case because when we normalize the entity, we create a new relationship between the transactions and sessions. There is a one to many relationship between the parent entity, sessions, and child entity, transactions.

Therefore, session_id has type Id in transactions because it represents an Index in another entity. There would be a similar effect if we added another entity using entity_from_dataframe and add_relationship.

In addition, when we created the new entity, we specified a time_index which was the variable (column) session_start. This changed the type of session_start to datetime_time_index in the new sessions entity because it now represents a time_index.
How do I combine two or more interesting values?

You might want to create features that are conditioned on multiple values before they are calculated. This would require the use of `interesting_values`. However, since we are trying to create the feature with multiple conditions, we will need to modify the Dataframe before we create the EntitySet.

Let’s look at how you might accomplish this.

First, let’s create our Dataframes.

```python
[12]: data = ft.demo.load_mock_customer()
    transactions_df = data["transactions"].merge(data["sessions"]).merge(data["customers")
    products_df = data["products"]

[13]: transactions_df.head()

    transaction_id  session_id  transaction_time  product_id  amount  \\
    0            298         1 2014-01-01 00:00:00      5   127.64  
    1             2         1 2014-01-01 00:01:05      2   109.48  
    2           308         1 2014-01-01 00:02:10      3    95.06  
    3           116         1 2014-01-01 00:03:15      4    78.92  
    4           371         1 2014-01-01 00:04:20      3    31.54  

    customer_id  device  session_start  zip_code  join_date  \\

    date_of_birth  \\
    0        1986-08-18  
    1        1986-08-18  
    2        1986-08-18  
    3        1986-08-18  
    4        1986-08-18  

[14]: products_df.head()

    product_id  brand  \\
    0           1  B  
    1           2  B  
    2           3  B  
    3           4  B  
    4           5  A  
```

Now, let’s modify our `transactions` Dataframe to create the additional column that represents multiple conditions for our feature.

```python
[15]: transactions_df["product_id_device"] = transactions_df["product_id"].astype(str) + "\n    and " + transactions_df["device"]
```

Here, we created a new column called `product_id_device`, which just combines the `product_id` column, and the `device` column.

Now let’s create our EntitySet.
Now, we are ready to add our interesting values.

First, let’s view our options for what the interesting values could be.

```python
interesting_values = transactions_df['product_id_device'].unique().tolist()
```

If you wanted to, you could pick a subset of these, and the where features created would only use those conditions. In our example, we will use all the possible interesting values.
Here, we set all of these values as our interesting values for this specific entity and variable. If we wanted to, we could make interesting values in the same way for more than one variable, but we will just stick with this one for this example.

```python
[18]: es['sessions']['product_id_device'].interesting_values = interesting_values
```

Now we can run DFS.

```python
[19]: feature_matrix, feature_defs = ft.dfs(entityset=es,
             target_entity="customers",
             agg_primitives=["count"],
             where_primitives=["count"],
             trans_primitives=[])
```

```python
feature_matrix.head()
```

```
COUNT(sessions)  COUNT(transactions)  \
<table>
<thead>
<tr>
<th>customer_id</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>93</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>79</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>109</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>126</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>93</td>
</tr>
</tbody>
</table>

COUNT(sessions WHERE product_id_device = 2 and desktop)  \
<table>
<thead>
<tr>
<th>customer_id</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
</tr>
</tbody>
</table>

COUNT(sessions WHERE product_id_device = 3 and desktop)  \
<table>
<thead>
<tr>
<th>customer_id</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
</tr>
</tbody>
</table>

COUNT(sessions WHERE product_id_device = 2 and mobile)  \
<table>
<thead>
<tr>
<th>customer_id</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.0</td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>2.0</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
</tr>
</tbody>
</table>

COUNT(sessions WHERE product_id_device = 5 and mobile)  \
<table>
<thead>
<tr>
<th>customer_id</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
</tr>
</tbody>
</table>

COUNT(sessions WHERE product_id_device = 1 and mobile)  \
<table>
<thead>
<tr>
<th>customer_id</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0</td>
</tr>
</tbody>
</table>
```
COUNT(sessions WHERE product_id_device = 4 and tablet) \
customer_id
2 1.0
5 1.0
4 0.0
1 0.0
3 0.0

COUNT(sessions WHERE product_id_device = 5 and desktop) \
customer_id
2 1
5 1
4 1
1 1
3 2

COUNT(sessions WHERE product_id_device = 4 and mobile) ... \ncustomer_id ... ... 
2 1.0 ... 
5 1.0 ... 
4 0.0 ... 
1 3.0 ... 
3 0.0 ... 

COUNT(transactions WHERE sessions.product_id_device = 1 and tablet) \ncustomer_id
2 15.0
5 0.0
4 0.0
1 0.0
3 0.0

COUNT(transactions WHERE sessions.product_id_device = 2 and tablet) \ncustomer_id
2 0.0
5 0.0
4 0.0
1 0.0
3 15.0

COUNT(transactions WHERE sessions.product_id_device = 5 and mobile) \ncustomer_id
2 0.0
5 0.0
4 0.0
1 0.0
3 0.0

COUNT(transactions WHERE sessions.product_id_device = 2 and mobile) \ncustomer_id
2 13.0
5 0.0

(continues on next page)
To better understand the where clause features, let’s examine one of those features. The feature \( \text{COUNT(sessions WHERE product_id_device = 5 and tablet)} \), tells us how many sessions the customer purchased product_id 5 while on a tablet. Notice how the feature depends on multiple conditions (product_id = 5 & device = tablet).
### 3.12.2 DFS

#### Why is DFS not creating aggregation features?

You may have created your `EntitySet`, and then applied DFS to create features. However, you may be puzzled as to why no aggregation features were created.

- This is most likely because you have a single table in your entity, and DFS is not capable of creating aggregation features with fewer than 2 entities. Featuretools looks for a relationship, and aggregates based on that relationship.

Let’s look at a simple example.

```python
[21]: data = ft.demo.load_mock_customer()
transactions_df = data["transactions"].merge(data["sessions"]).merge(data["customers "])
es = ft.EntitySet(id="customer_data")
es = es.entity_from_dataframe(entity_id="transactions",
dataframe=transactions_df,
index="transaction_id")
es
```

Notice how we only have 1 entity in our `EntitySet`. If we try to create aggregation features on this `EntitySet`, it will not be possible because DFS needs 2 entities to generate aggregation features.

```python
[22]: feature_matrix, feature_def = ft.dfs(entityset=es, target_entity="transactions")
feature_defs
```

(continues on next page)
None of the above features are aggregation features. To fix this issue, you can add another entity to your EntitySet.

**Solution #1 - You can add new entity if you have additional data.**

```python
[23]:
    products_df = data["products"]
    es = es.entity_from_dataframe(entity_id="products",
                                  dataframe=products_df,
                                  index="product_id")
```

Notice how we now have an additional entity in our EntitySet, called `products`.

**Solution #2 - You can normalize an existing entity.**

```python
[24]:
    es = es.normalize_entity(base_entity_id="transactions",
                              new_entity_id="sessions",
                              index="session_id",
                              make_time_index="session_start",
                              additional_variables=["device", "customer_id", "zip_code",
                                                    "join_date"],
                              copy_variables=["session_start"])
```

Notice how we now have an additional entity in our EntitySet, called `sessions`. Here, the normalization created a relationship between `transactions` and `sessions`. However, we could have specified a relationship between `transactions` and `products` if we had only used Solution #1.

Now, we can generate aggregation features.

```python
[25]:
    feature_matrix, feature_defs = ft.dfs(entityset=es, target_entity="transactions")
    feature_defs[[-10]
```
A few of the aggregation features are:

- `<Feature: sessions.SUM(transactions.amount)>`
- `<Feature: sessions.STD(transactions.amount)>`
- `<Feature: sessions.MAX(transactions.amount)>`
- `<Feature: sessions.SKEW(transactions.amount)>`
- `<Feature: sessions.MIN(transactions.amount)>`
- `<Feature: sessions.MEAN(transactions.amount)>`
- `<Feature: sessions.COUNT(transactions)>`

**How do I speed up the runtime of DFS?**

One issue you may encounter while running `ft.dfs` is slow performance. While Featuretools has generally optimal default settings for calculating features, you may want to speed up performance when you are calculating on a large number of features.

One quick way to speed up performance is by adjusting the `n_jobs` settings of `ft.dfs` or `ft.calculate_feature_matrix`.

```python
# setting n_jobs to -1 will use all cores
feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_entity="customers",
                                      n_jobs=-1)
```

```python
feature_matrix, feature_defs = ft.calculate_feature_matrix(entityset=es,
```

(continues on next page)
For more ways to speed up performance, please visit:

- *Improving Computational Performance*

### How do I include only certain features when running DFS?

When using DFS to generate features, you may wish to include only certain features. There are multiple ways that you do this:

- Use the `ignore_variables` to specify variables in an entity that should not be used to create features. It is a dictionary mapping an entity id to a list of variable names to ignore.
- Use `drop_contains` to drop features that contain any of the strings listed in this parameter.
- Use `drop_exact` to drop features that exactly match any of the strings listed in this parameter.

Here is an example of using all three parameters:

```python
[26]: es = ft.demo.load_mock_customer(return_entityset=True)
feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_entity="customers",
                                      ignore_variables={
                                          "transactions": ["amount"],
                                          "customers": ["age", "gender", "date_of_birth"]
                                      },  # ignore these variables
                                      drop_contains=["customers.SUM("],  # drop features that contain these strings
                                      drop_exact=["STD(transactions.quanity)"])  # drop features that exactly match
```

### How do I specify primitives on a per column or per entity basis?

When using DFS to generate features, you may wish to use only certain features or entities for specific primitives. This can be done through the `primitive_options` parameter. The `primitive_options` parameter is a dictionary or list of dictionaries that maps a primitive or a tuple of primitives to a dictionary containing options for the primitive(s). This parameter can also be a list of option dictionaries if the primitive takes multiple inputs. Each dictionary supplies options for their respective input column. There are multiple ways to control how primitives get applied through these options:

- Use `ignore_entities` to specify entities that should not be used to create features for that primitive. It is a list of entity ids to ignore.
- Use `include_entities` to specify the only entities to be included to create features for that primitive. It is a list of entity ids to include.
- Use `ignore_variables` to specify variables in an entity that should not be used to create features for that primitive. It is a dictionary mapping an entity id to a list of variable names to ignore.
- Use `include_variables` to specify the only variables in an entity that should be used to create features for that primitive. It is a dictionary mapping an entity id to a list of variable names to include.
You can also use `primitive_options` to specify which entities or variables you wish to use as groupbys for groupby transformation primitives:

- Use `ignore_groupby_entities` to specify entities that should not be used to get groupbys for that primitive. It is a list of entity ids to ignore.
- Use `include_groupby_entities` to specify the only entities that should be used to get groupbys for that primitive. It is a list of entity ids to include.
- Use `ignore_groupby_variables` to specify variables in an entity that should not be used as groupbys for that primitive. It is a dictionary mapping an entity id to a list of variable names to ignore.
- Use `include_groupby_variables` to specify the only variables in an entity that should be used as groupbys for that primitive. It is a dictionary mapping an entity id to a list of variable names to include.

Here is an example of using some of these options:

```python
[27]: es = ft.demo.load_mock_customer(return_entityset=True)
feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_entity="customers",
                                      primitive_options={"mode": {"ignore_entities": [
                                          "sessions"],
                                          "include_variables": {"products": ["brand"],
                                          "transactions": ["product_id"]},
                                          "sessions" entity and only include "brands" in the
                                          "product_id" in the "transactions" entity
                                          "entities": ["sessions", "transactions"]
                                          "include the entities "sessions" and "transactions"
                                          "count", "mean": {"include_entities": ["sessions", "transactions"]
                                          "count and mean, only"}}

For a more examples of specifying options for DFS, please visit:

- Specifying Primitive Options

If I didn’t specify the cutoff_time, what date will be used for the feature calculations?

The cutoff time will be set to the current time using `cutoff_time = datetime.now()`.

How do I select a certain amount of past data when calculating features?

You may encounter a situation when you wish to make prediction using only a certain amount of historical data. You can accomplish this using the `training_window` parameter in `ft.dfs`. When you use the `training_window`, Featuretools will use the historical data between the `cutoff_time` and `cutoff_time - training_window`.

In order to make the calculation, Featuretools will check the time in the `time_index` column of the `target_entity`.

```python
[28]: es = ft.demo.load_mock_customer(return_entityset=True)
es['customers'].time_index
```
Our target_entity has a time_index, which is needed for the training_window calculation. Here, we are creating a cutoff time dataframe so that we can have a unique training window for each customer.

```python
[cutoff_times = pd.DataFrame()
cutoff_times['customer_id'] = [1, 2, 3, 1]
cutoff_times['time'] = pd.to_datetime(['2014-1-1 04:00', '2014-1-1 05:00', '2014-1-1 06:00', '2014-1-1 08:00'])
cutoff_times['label'] = [True, True, False, True]
feature_matrix, feature_defs = ft.dfs(entityset=es,
    target_entity="customers",
cutoff_time=cutoff_times,
cutoff_time_in_index=True,
training_window="1 hour")
feature_matrix.head()```

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>zip_code</th>
<th>COUNT(sessions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>60091</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>13244</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>13244</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>60091</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>NUM_UNIQUE(sessions.device)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>MODE(sessions.device)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>tablet</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>tablet</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>desktop</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>mobile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>SUM(transactions.amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>1025.63</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>1004.96</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>980.97</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>1420.09</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>STD(transactions.amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>39.825249</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>33.725036</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>34.188788</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>32.324534</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>MAX(transactions.amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>139.09</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>118.85</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>128.26</td>
</tr>
<tr>
<td>customer_id</td>
<td>time</td>
<td>SKEW(transactions.amount)</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>-0.830975</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>-0.314918</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>-0.386592</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>-1.038434</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>MIN(transactions.amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>6.78</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>21.82</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>20.06</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>11.62</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>MEAN(transactions.amount)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>85.469167</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>77.304615</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>81.747500</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>88.755625</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>MODE(sessions.YEAR(session_start))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>2014</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>2014</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>2014</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>2014</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>MODE(sessions.DAY(session_start))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>MODE(sessions.MODE(transactions.product_id))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>MODE(sessions.WEEKDAY(session_start))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
<th>MODE(sessions.MONTH(session_start))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 04:00:00</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 05:00:00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 06:00:00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2014-01-01 08:00:00</td>
<td>1</td>
</tr>
</tbody>
</table>
Above, we ran DFS with the `training_window` argument of 1 hour to create features that only used customer data collected in the last hour (from the cutoff time we provided).

### How do I apply DFS to a single table?

You can run DFS on a single table. Featuretools will be able to generate features for your data, but only transform features.

For example:

```python
transactions_df = ft.demo.load_mock_customer(return_single_table=True)
```

```python
es = ft.EntitySet(id="customer_data")
es = es.entity_from_dataframe(entity_id="transactions", dataframe=transactions_df, index="transaction_id", time_index="transaction_time")
```

```python
feature_matrix, feature_defs = ft.dfs(entityset=es, target_entity="transactions", trans_primitives=['time_since', 'day', 'is_weekend', 'cum_min', 'minute', 'num_words', 'weekday', 'cum_count'],
```
Before we examine the output, let’s look at our original single table.

```
[31]: transactions_df.head()
```

<table>
<thead>
<tr>
<th>transaction_id</th>
<th>session_id</th>
<th>transaction_time</th>
<th>product_id</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>298</td>
<td>1 2014-01-01 00:00:00</td>
<td>5</td>
<td>127.64</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>1 2014-01-01 00:09:45</td>
<td>5</td>
<td>57.39</td>
</tr>
<tr>
<td>2</td>
<td>495</td>
<td>1 2014-01-01 00:14:05</td>
<td>5</td>
<td>69.45</td>
</tr>
<tr>
<td>3</td>
<td>460</td>
<td>10 2014-01-01 02:33:50</td>
<td>5</td>
<td>123.19</td>
</tr>
<tr>
<td>4</td>
<td>302</td>
<td>10 2014-01-01 02:37:05</td>
<td>5</td>
<td>64.47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>device</th>
<th>session_start</th>
<th>zip_code</th>
<th>join_date</th>
</tr>
</thead>
</table>

Now we can look at the transformations that Featuretools was able to apply to this single entity (table) to create feature matrix.

```
[32]: feature_matrix.head()
```

<table>
<thead>
<tr>
<th>session_id</th>
<th>product_id</th>
<th>amount</th>
<th>customer_id</th>
<th>device</th>
<th>zip_code</th>
<th>transaction_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>298</td>
<td>1</td>
<td>5</td>
<td>127.64</td>
<td>2</td>
<td>desktop</td>
<td>12344</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>109.48</td>
<td>2</td>
<td>desktop</td>
<td>12344</td>
</tr>
<tr>
<td>308</td>
<td>1</td>
<td>3</td>
<td>95.06</td>
<td>2</td>
<td>desktop</td>
<td>12344</td>
</tr>
<tr>
<td>116</td>
<td>1</td>
<td>4</td>
<td>78.92</td>
<td>2</td>
<td>desktop</td>
<td>12344</td>
</tr>
<tr>
<td>371</td>
<td>1</td>
<td>3</td>
<td>31.54</td>
<td>2</td>
<td>desktop</td>
<td>12344</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>brand</th>
<th>TIME_SINCE(transaction_time)</th>
<th>DAY(join_date)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1.967986e+08</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>1.967985e+08</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>1.967984e+08</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>1.967983e+08</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DAY(date_of_birth)</th>
<th>CUM_MEAN(CUM_MIN(customer_id))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
CUM_MEAN(PERCENTILE(session_id)) \ transaction_id  
298 0.017 
2 0.017 
308 0.017 
116 0.017 
371 0.017 

CUM_MEAN(CUM_MIN(session_id)) \ CUM_MEAN(PERCENTILE(amount)) \ transaction_id  
298 1.0 0.846000 
2 1.0 0.793000 
308 1.0 0.743333 
116 1.0 0.693500 
371 1.0 0.597200 

CUM_MEAN(CUM_MIN(amount)) \ CUM_MEAN(MINUTE(join_date)) \ transaction_id  
298 127.640000 31.0 
2 118.560000 31.0 
308 110.726667 31.0 
116 102.775000 31.0 
371 88.528000 31.0 

CUM_MEAN(MINUTE(transaction_time)) \ transaction_id  
298 0.0 
2 0.5 
308 1.0 
116 1.5 
371 2.0 

CUM_MEAN(PERCENTILE(customer_id)) \ transaction_id  
298 0.346 
2 0.346 
308 0.346 
116 0.346 
371 0.346 

CUM_MEAN(MINUTE(session_start)) \ transaction_id  
298 0.0 
2 0.0 
308 0.0 
116 0.0 
371 0.0 

CUM_MEAN(MINUTE(date_of_birth)) \ transaction_id  
298 0.0 
2 0.0 
308 0.0 
116 0.0 
371 0.0 
[5 rows x 61 columns]
Can I automatically normalize a single table?

Yes, another open source library AutoNormalize, also produced by Feature Labs, automates table normalization and integrates with Featuretools. To install run:

```bash
python -m pip install featuretools[autonormalize]
```

A normalized `EntitySet` will help Featuretools to generate more features. For example:

```python
from featuretools.autonormalize import autonormalize as an
es = an.normalize_entity(es)
es.plot()
```

```bash
100% || 10/10 [00:03<00:00,  3.05it/s]
```

As you can see, AutoNormalize creates a relational `EntitySet`. Below, we run `dfs` on the `EntitySet`, and you can see all the features created; take note of the aggregated features.

```python
feature_matrix, feature_defs = ft.dfs(entityset=es,
target_entity="transaction_id",
trans_primitives=[])  
```

```plaintext
session_id product_id amount session_id.customer_id  
transaction_id
298 1 5 127.64 2
2 1 2 109.48 2
308 1 3 95.06 2
116 1 4 78.92 2
371 1 3 31.54 2

session_id.device product_id.brand  
transaction_id
298 desktop A
2 desktop B
308 desktop B
116 desktop B
371 desktop B

session_id.SUM(transaction_id.amount)  
transaction_id
298 1229.01
2 1229.01
308 1229.01
116 1229.01
371 1229.01

session_id.STD(transaction_id.amount)  
transaction_id
298 41.600976
2 41.600976
308 41.600976
116 41.600976
371 41.600976

session_id.MAX(transaction_id.amount)  
transaction_id
298 141.66
```

(continues on next page)
(continued from previous page)

<table>
<thead>
<tr>
<th>session_id.SKEW(transaction_id.amount)</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.295458</td>
</tr>
<tr>
<td>308</td>
<td>0.295458</td>
</tr>
<tr>
<td>116</td>
<td>0.295458</td>
</tr>
<tr>
<td>371</td>
<td>0.295458</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>session_id.customer_id.zip_code</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td></td>
</tr>
<tr>
<td>298</td>
<td>13244</td>
</tr>
<tr>
<td>2</td>
<td>13244</td>
</tr>
<tr>
<td>308</td>
<td>13244</td>
</tr>
<tr>
<td>116</td>
<td>13244</td>
</tr>
<tr>
<td>371</td>
<td>13244</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product_id.SUM(transaction_id.amount)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td></td>
</tr>
<tr>
<td>298</td>
<td>7931.55</td>
</tr>
<tr>
<td>2</td>
<td>7021.43</td>
</tr>
<tr>
<td>308</td>
<td>7008.12</td>
</tr>
<tr>
<td>116</td>
<td>8088.97</td>
</tr>
<tr>
<td>371</td>
<td>7008.12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product_id.STD(transaction_id.amount)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td></td>
</tr>
<tr>
<td>298</td>
<td>42.131902</td>
</tr>
<tr>
<td>2</td>
<td>46.336308</td>
</tr>
<tr>
<td>308</td>
<td>38.871405</td>
</tr>
<tr>
<td>116</td>
<td>42.492501</td>
</tr>
<tr>
<td>371</td>
<td>38.871405</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product_id.MAX(transaction_id.amount)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td></td>
</tr>
<tr>
<td>298</td>
<td>149.02</td>
</tr>
<tr>
<td>2</td>
<td>149.95</td>
</tr>
<tr>
<td>308</td>
<td>148.31</td>
</tr>
<tr>
<td>116</td>
<td>146.46</td>
</tr>
<tr>
<td>371</td>
<td>148.31</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product_id.SKEW(transaction_id.amount)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td></td>
</tr>
<tr>
<td>298</td>
<td>0.098248</td>
</tr>
<tr>
<td>2</td>
<td>0.151934</td>
</tr>
<tr>
<td>308</td>
<td>0.223938</td>
</tr>
<tr>
<td>116</td>
<td>-0.132077</td>
</tr>
<tr>
<td>371</td>
<td>0.223938</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>product_id.MIN(transaction_id.amount)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>transaction_id</td>
<td></td>
</tr>
<tr>
<td>298</td>
<td>5.91</td>
</tr>
<tr>
<td>2</td>
<td>5.73</td>
</tr>
</tbody>
</table>

(continues on next page)
How do I prevent label leakage with DFS?

One concern you might have with using DFS is about label leakage. You want to make sure that labels in your data aren’t used incorrectly to create features and the feature matrix.

**Featuretools is particularly focused on helping users avoid label leakage.**

There are two ways to prevent label leakage depending on if your data has timestamps or not.

1. **Data without timestamps**

In the case where you do not have timestamps, you can create one `EntitySet` using only the training data and then run `ft.dfs`. This will create a feature matrix using only the training data, but also return a list of feature definitions. Next, you can create an `EntitySet` using the test data and recalculate the same features by calling `ft.calculate_feature_matrix` with the list of feature definitions from before.

Here is what that flow would look like:
First, let’s create our training data.

```
[35]: train_data = pd.DataFrame({"customer_id": [1, 2, 3, 4, 5],
                               "age": [40, 50, 10, 20, 30],
                               "gender": ["m", "f", "m", "f", "f"],
                               "signup_date": pd.date_range('2014-01-01 01:41:50',
                                            periods=5, freq='25min'),
                               "labels": [True, False, True, False, True]})
```

```
train_data.head()
```

```
+---------+-------+-----+-----------------+-------+
<table>
<thead>
<tr>
<th>customer_id</th>
<th>age</th>
<th>gender</th>
<th>signup_date</th>
<th>labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>m</td>
<td>2014-01-01 01:41:50</td>
<td>True</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>f</td>
<td>2014-01-01 02:06:50</td>
<td>False</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>m</td>
<td>2014-01-01 02:31:50</td>
<td>True</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>f</td>
<td>2014-01-01 02:56:50</td>
<td>False</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>f</td>
<td>2014-01-01 03:21:50</td>
<td>True</td>
</tr>
</tbody>
</table>
```

Now, we can create an entityset for our training data.

```
[36]: es_train_data = ft.EntitySet(id="customer_train_data")
es_train_data = es_train_data.entity_from_dataframe(entity_id="customers",
dataframe=train_data,
index="customer_id")
es_train_data
```

```
Entityset: customer_train_data
Entities:
  customers [Rows: 5, Columns: 5]
Relationships:
  No relationships
```

Next, we are ready to create our features, and feature matrix for the training data.

```
[37]: feature_matrix_train, feature_defs = ft.dfs(entityset=es_train_data,
target_entity="customers")
```

```
feature_matrix_train
```

```
+---------+-------+-----+-----------------+-------+-------+--------+--------+
<table>
<thead>
<tr>
<th>customer_id</th>
<th>age</th>
<th>gender</th>
<th>signup_date</th>
<th>labels</th>
<th>DAY(signup_date)</th>
<th>YEAR(signup_date)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40</td>
<td>m</td>
<td>2014-01-01 01:41:50</td>
<td>True</td>
<td>1</td>
<td>2014</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>f</td>
<td>2014-01-01 02:06:50</td>
<td>False</td>
<td>1</td>
<td>2014</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>m</td>
<td>2014-01-01 02:31:50</td>
<td>True</td>
<td>1</td>
<td>2014</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>f</td>
<td>2014-01-01 02:56:50</td>
<td>False</td>
<td>1</td>
<td>2014</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>f</td>
<td>2014-01-01 03:21:50</td>
<td>True</td>
<td>1</td>
<td>2014</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MONTH(signup_date)</td>
<td>WEEKDAY(signup_date)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
```

We will also encode our feature matrix to make machine learning compatible features.

```
[38]: feature_matrix_train_enc, features_enc = ft.encode_features(feature_matrix_train,
                                                                   feature_defs)
feature_matrix_train_enc.head()
```

```
```

```
```

3.12. Frequently Asked Questions
Notice how the whole feature matrix only includes numeric values now.

Now we can use the feature definitions to calculate our feature matrix for the test data, and avoid label leakage.

```python
[39]:
test_train = pd.DataFrame({"customer_id": [6, 7, 8, 9, 10],
  "age": [20, 25, 55, 22, 35],
  "gender": ["f", "m", "m", "m", "m"],
  "signup_date": pd.date_range('2014-01-01 01:41:50', periods=5, freq='25min')})

# lets add NaN label column to the test Dataframe
test_train['labels'] = np.nan

es_test_data = ft.EntitySet(id="customer_test_data")
es_test_data = es_test_data.entity_from_dataframe(entity_id="customers",
dataframe=test_train,
index="customer_id",
time_index="signup_date")
```

(continues on next page)
# Use the feature definitions from earlier

```python
feature_matrix_enc_test = ft.calculate_feature_matrix(features=features_enc,
entityset=es_test_data)
```

```python
feature_matrix_enc_test.head()
```

<table>
<thead>
<tr>
<th>customer_id</th>
<th>age</th>
<th>gender = f</th>
<th>gender = m</th>
<th>gender is unknown</th>
<th>labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>20</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>NaN</td>
</tr>
<tr>
<td>7</td>
<td>25</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>NaN</td>
</tr>
<tr>
<td>8</td>
<td>55</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>NaN</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>NaN</td>
</tr>
<tr>
<td>10</td>
<td>35</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>NaN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>DAY(signup_date) = 1</th>
<th>DAY(signup_date) is unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>7</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>8</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>9</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>10</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>YEAR(signup_date) = 2014</th>
<th>YEAR(signup_date) is unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>7</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>8</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>9</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>10</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>MONTH(signup_date) = 1</th>
<th>MONTH(signup_date) is unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>7</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>8</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>9</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>10</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>customer_id</th>
<th>WEEKDAY(signup_date) = 2</th>
<th>WEEKDAY(signup_date) is unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>7</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>8</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>9</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>10</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

Note: Disregard the difference between the False/True above, and 0/1 in the earlier feature matrix. A simple casting would address this difference.

## 2. Data with timestamps

If your data has timestamps, the best way to prevent label leakage is to use a list of **cutoff times**, which specify the last point in time data is allowed to be used for each row in the resulting feature matrix. To use **cutoff times**, you need to set a time index for each time sensitive entity in your entity set.

### 3.12. Frequently Asked Questions
Tip: Even if your data doesn’t have time stamps, you could add a column with dummy timestamps that can be used by Featuretools as time index.

When you call `ft.dfs`, you can provide a Dataframe of cutoff times like this:

```
[40]: cutoff_times = pd.DataFrame({"customer_id": [1, 2, 3, 4, 5],
                                 "time": pd.date_range('2014-01-01 01:41:50', periods=5),
                                 "freq":"25min"})
cutoff_times.head()
```

```
<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2014-01-01 01:41:50</td>
</tr>
<tr>
<td>1</td>
<td>2 2014-01-01 02:06:50</td>
</tr>
<tr>
<td>2</td>
<td>3 2014-01-01 02:31:50</td>
</tr>
<tr>
<td>3</td>
<td>4 2014-01-01 02:56:50</td>
</tr>
<tr>
<td>4</td>
<td>5 2014-01-01 03:21:50</td>
</tr>
</tbody>
</table>
```

```
[41]: train_test_data = pd.DataFrame({"customer_id": [1, 2, 3, 4, 5],
                                "age": [20, 25, 55, 22, 35],
                                "gender": ["f", "m", "m", "m", "m"],
                                "signup_date": pd.date_range('2010-01-01 01:41:50', periods=5),
                                "freq":"25min"})
```

```
<table>
<thead>
<tr>
<th>customer_id</th>
<th>age</th>
<th>gender</th>
<th>DAY(signup_date)</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>f</td>
<td>1</td>
<td>2014-01-01 01:41:50</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>m</td>
<td>1</td>
<td>2014-01-01 02:06:50</td>
</tr>
<tr>
<td>3</td>
<td>55</td>
<td>m</td>
<td>1</td>
<td>2014-01-01 02:31:50</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>m</td>
<td>1</td>
<td>2014-01-01 02:56:50</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>m</td>
<td>1</td>
<td>2014-01-01 03:21:50</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>customer_id</th>
<th>YEAR(signup_date)</th>
<th>MONTH(signup_date)</th>
<th>WEEKDAY(signup_date)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2010</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2010</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2010</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>2010</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2010</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
```

(continues on next page)
Above, we have created a feature matrix that uses cutoff times to avoid label leakage. We could also encode this feature matrix using `ft.encode_features`.

**What is the difference between passing a primitive object versus a string to DFS?**

There are 2 ways to pass primitives to DFS: the primitive object, or a string of the primitive name.

We will use the Transform primitive called `TimeSincePrevious` to illustrate the differences.

First, let’s use the string of primitive name.

```python
[42]: es = ft.demo.load_mock_customer(return_entityset=True)

[43]: feature_matrix, feature_defs = ft.dfs(entityset=es,
      target_entity="customers",
      agg_primitives=[],
      trans_primitives=["time_since_previous"]

feature_matrix
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>TIME_SINCE_PREVIOUS(join_date)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>60091</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>60091</td>
<td>22948824.0</td>
</tr>
<tr>
<td>1</td>
<td>60091</td>
<td>744019.0</td>
</tr>
<tr>
<td>3</td>
<td>13244</td>
<td>10212841.0</td>
</tr>
<tr>
<td>2</td>
<td>13244</td>
<td>21282510.0</td>
</tr>
</tbody>
</table>

Now, let’s use the primitive object.

```python
[44]: from featuretools.primitives import TimeSincePrevious

feature_matrix, feature_defs = ft.dfs(entityset=es,
      target_entity="customers",
      agg_primitives=[],
      trans_primitives=[TimeSincePrevious])

feature_matrix
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>TIME_SINCE_PREVIOUS(join_date)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>60091</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>60091</td>
<td>22948824.0</td>
</tr>
<tr>
<td>1</td>
<td>60091</td>
<td>744019.0</td>
</tr>
<tr>
<td>3</td>
<td>13244</td>
<td>10212841.0</td>
</tr>
<tr>
<td>2</td>
<td>13244</td>
<td>21282510.0</td>
</tr>
</tbody>
</table>

As we can see above, the feature matrix is the same.

However, if we need to modify controllable parameters in the primitive, we should use the primitive object. For instance, let’s make `TimeSincePrevious` return units of hours (the default is in seconds).

```python
[45]: from featuretools.primitives import TimeSincePrevious

time_since_previous_in_hours = TimeSincePrevious(unit='hours')

feature_matrix, feature_defs = ft.dfs(entityset=es,
```

(continues on next page)
3.12.3 Features

How can I select features based on some attributes (a specific string, an explicit primitive type, a return type, a given depth)?

You may wish to select a subset of your features based on some attributes.

Let’s say you wanted to select features that had the string amount in its name. You can check for this by using the get_name function on the feature definitions.

```python
[46]: es = ft.demo.load_mock_customer(return_entityset=True)
feature_defs = ft.dfs(entityset=es,
                     target_entity="customers",
                     features_only=True)
features_with_amount = []
for x in feature_defs:
    if 'amount' in x.get_name():
        features_with_amount.append(x)
features_with_amount[0:5]
```

You might also want to only select features that are aggregation features.

```python
[47]: from featuretools import AggregationFeature
features_only_aggregations = []
for x in feature_defs:
    if type(x) == AggregationFeature:
        features_only_aggregations.append(x)
features_only_aggregations[0:5]
```

---

Featuretools Documentation, Release 0.13.4

(continued from previous page)

```python

feature_matrix

<table>
<thead>
<tr>
<th>zip_code</th>
<th>TIME_SINCE_PREVIOUS(join_date, unit=hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_id</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>60091</td>
</tr>
<tr>
<td>4</td>
<td>60091</td>
</tr>
<tr>
<td>1</td>
<td>60091</td>
</tr>
<tr>
<td>3</td>
<td>13244</td>
</tr>
<tr>
<td>2</td>
<td>13244</td>
</tr>
</tbody>
</table>
```
Also, you might only want to select features that are calculated at a certain depth. You can do this by using the `get_depth` function.

```python
features_only_depth_2 = []
for x in feature_defs:
    if x.get_depth() == 2:
        features_only_depth_2.append(x)
features_only_depth_2[0:5]
```

Finally, you might only want features that return a certain type. You can do this by using the `variable_type` function.

```python
from featuretools.variable_types import Numeric

features_only_numeric = []
for x in feature_defs:
    if x.variable_type == Numeric:
        features_only_numeric.append(x)
features_only_numeric[0:5]
```

Once you have your specific feature list, you can use `ft.calculate_feature_matrix` to generate a feature matrix for only those features.

For our example, let’s use the features with only the string `amount` in its name.

```python
# change → to your specific feature list
feature_matrix = ft.calculate_feature_matrix(entityset=es, features=features_with_amount)

feature_matrix.head()
```

(continues on next page)
<table>
<thead>
<tr>
<th>customer_id</th>
<th>SUM(sessions.MEAN(transactions.amount))</th>
<th>customer_id</th>
<th>SUM(sessions.MIN(transactions.amount))</th>
<th>customer_id</th>
<th>SUM(sessions.MAX(transactions.amount))</th>
<th>customer_id</th>
<th>SUM(sessions.STD(transactions.amount))</th>
<th>customer_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>7.55</td>
<td>80.375443</td>
<td>4</td>
<td>5.73</td>
<td>80.070459</td>
<td>1</td>
<td>5.81</td>
<td>71.631905</td>
</tr>
<tr>
<td>4</td>
<td>5.73</td>
<td></td>
<td>4</td>
<td>5.89</td>
<td>67.060430</td>
<td>3</td>
<td>8.73</td>
<td>77.422366</td>
</tr>
<tr>
<td>1</td>
<td>5.81</td>
<td></td>
<td>1</td>
<td>8.73</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5.89</td>
<td></td>
<td>3</td>
<td>8.73</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8.73</td>
<td></td>
<td>2</td>
<td>8.73</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MIN(sessions.SUM(transactions.amount))  

<table>
<thead>
<tr>
<th>customer_id</th>
<th>MIN(sessions.SUM(transactions.amount))</th>
<th>customer_id</th>
<th>MIN(sessions.MAX(transactions.amount))</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>543.18</td>
<td>128.51</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>771.68</td>
<td>139.20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>809.97</td>
<td>118.90</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>889.21</td>
<td>126.74</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>634.84</td>
<td>100.04</td>
<td></td>
</tr>
</tbody>
</table>

MIN(sessions.STD(transactions.amount))  

<table>
<thead>
<tr>
<th>customer_id</th>
<th>MIN(sessions.STD(transactions.amount))</th>
<th>customer_id</th>
<th>MIN(sessions.MAX(transactions.amount))</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>259.873954</td>
<td>128.51</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>356.125829</td>
<td>139.20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>312.745952</td>
<td>118.90</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>257.299895</td>
<td>126.74</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>258.700528</td>
<td>100.04</td>
<td></td>
</tr>
</tbody>
</table>

(continues on next page)
<table>
<thead>
<tr>
<th></th>
<th>MIN(sessions.SKEW(transactions.amount))</th>
<th>MEAN(sessions.MEAN(transactions.amount))</th>
<th>MEAN(sessions.SUM(transactions.amount))</th>
<th>MEAN(sessions.MIN(transactions.amount))</th>
<th>MEAN(sessions.MAX(transactions.amount))</th>
<th>MEAN(sessions.STD(transactions.amount))</th>
<th>MEAN(sessions.SKEW(transactions.amount))</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>36.734681</td>
<td>78.705187</td>
<td>1058.276667</td>
<td>14.415000</td>
<td>139.960000</td>
<td>43.312326</td>
<td>0.002397</td>
</tr>
<tr>
<td>4</td>
<td>29.026424</td>
<td>81.207189</td>
<td>1090.960000</td>
<td>16.438750</td>
<td>144.748750</td>
<td>44.515729</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>30.450261</td>
<td>72.774140</td>
<td>1128.202500</td>
<td>9.823750</td>
<td>132.246250</td>
<td>39.093244</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>35.704680</td>
<td>67.539577</td>
<td>1039.436667</td>
<td>11.035000</td>
<td>141.271667</td>
<td>42.883316</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>27.839228</td>
<td>78.415122</td>
<td>1028.611429</td>
<td>22.085714</td>
<td>133.090000</td>
<td>36.957218</td>
<td></td>
</tr>
</tbody>
</table>
Above, notice how all the column names for our feature matrix contain the string `amount`.

**How do I create where features?**

Sometimes, you might want to create features that are conditioned on a second value before it is calculated. This extra filter is called a “where clause”. You can create these features using the using the `interesting_values` of a variable.

If you have categorical columns in your `EntitySet`, you can use then `add_interesting_values`. This function will find interesting values for your categorical variables, which can then be used to generate “where” clauses.

First, let’s create our `EntitySet`.

```python
[51]: es = ft.demo.load_mock_customer(return_entityset=True)
es
```

```plaintext
Entityset: transactions
Entities:
  transactions [Rows: 500, Columns: 5]
  products [Rows: 5, Columns: 2]
  sessions [Rows: 35, Columns: 4]
  customers [Rows: 5, Columns: 4]
Relationships:
  transactions.product_id -> products.product_id
  transactions.session_id -> sessions.session_id
  sessions.customer_id -> customers.customer_id
```

Now we can add the interesting variables for the categorical variables.

```python
[52]: es.add_interesting_values()
```

Now we can run DFS with the `where_primitives` argument to define which primitives to apply with where clauses. In this case, let’s use the primitive `count`.

```python
[53]: feature_matrix, feature_defs = ft.dfs(entityset=es,
                                           target_entity="customers",
                                           agg_primitives=["count"],
                                           where_primitives=["count"],
                                           trans_primitives=[])
```

```plaintext
feature_matrix.head()
```

```plaintext
<table>
<thead>
<tr>
<th>customer_id</th>
<th>zip_code</th>
<th>COUNT(sessions)</th>
<th>COUNT(transactions)</th>
<th>COUNT(sessions WHERE device = tablet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>60091</td>
<td>6</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>60091</td>
<td>8</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>60091</td>
<td>8</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>13244</td>
<td>6</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13244</td>
<td>7</td>
<td>93</td>
<td></td>
</tr>
</tbody>
</table>
```

(continues on next page)
<table>
<thead>
<tr>
<th>customer_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

COUNT(sessions WHERE device = mobile) \\n| customer_id |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

COUNT(sessions WHERE device = desktop) \\n| customer_id |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

COUNT(sessions WHERE customers.zip_code = 60091) \\n| customer_id |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

COUNT(sessions WHERE customers.zip_code = 13244) \\n| customer_id |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

COUNT(transactions WHERE sessions.device = tablet) \\n| customer_id |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

COUNT(transactions WHERE sessions.device = mobile) \\n| customer_id |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

COUNT(transactions WHERE sessions.device = desktop)

(continues on next page)
We have now created some useful features. One example of a useful feature is the \( \text{COUNT}(\text{sessions WHERE device = tablet}) \). This feature tells us how many sessions a customer completed on a tablet.

3.12.4 Primitives

What is the difference between the primitive types (Transform, GroupBy Transform, & Aggregation)?

You might curious to know the difference between the primitive groups. Let’s review the differences between transform, groupby transform, and aggregation primitives.

First, let’s create a simple EntitySet.

```python
import pandas as pd
import featuretools as ft

df = pd.DataFrame(
    {
        "id": [1, 2, 3, 4, 5, 6],
        "time_index": pd.date_range("1/1/2019", periods=6, freq="D"),
        "group": ["a", "b", "a", "c", "a", "b"],
        "val": [5, 1, 10, 20, 6, 23],
    }
)
es = ft.EntitySet()
es = es.entity_from_dataframe(entity_id="observations",
    dataframe=df,
    index="id",
    time_index="time_index")
es = es.normalize_entity(base_entity_id="observations",
    new_entity_id="groups",
    index="group")
es.plot()
```

After calling `normalize_entity`, the variable “group” has the type “id” because it identifies another entity. Alternatively, it could be set using the `variable_types` parameter when we first call `es.entity_from_dataframe()`.
Transform Primitive

The cum_sum primitive calculates the running sum in list of numbers.

```
[56]: from featuretools.primitives import CumSum

    cum_sum = CumSum()
    cum_sum([1, 2, 3, 4, 5]).tolist()

[56]: [1, 3, 6, 10, 15]
```

If we apply it using the trans_primitives argument it will calculate it over the entire observations entity like this:

```
[57]: feature_matrix, feature_defs = ft.dfs(target_entity="observations",
               entityset=es,
               agg_primitives=[],
               trans_primitives=["cum_sum"],
               groupby_trans_primitives=[])

feature_matrix

[57]:

    | group | val | CUM_SUM(val) |
    |------|-----|-------------|
    | id   |     |            |
    | 1    | a   | 5          |
    | 2    | b   | 6          |
    | 3    | a   | 16         |
    | 4    | c   | 36         |
    | 5    | a   | 42         |
    | 6    | b   | 65         |
```

Groupby Transform Primitive

If we apply it using groupby_trans_primitives, then DFS will first group by any id variables before applying the transform primitive. As a result, we get the cumulative sum by group.

```
[58]: feature_matrix, feature_defs = ft.dfs(target_entity="observations",
               entityset=es,
               agg_primitives=[],
               trans_primitives=["cum_sum"],
               groupby_trans_primitives=["cum_sum"])

feature_matrix

[58]:

    | group | val | CUM_SUM(val) by group |
    |------|-----|-----------------------|
    | id   |     |                       |
    | 1    | a   | 5.0                   |
    | 2    | b   | 1.0                   |
    | 3    | a   | 15.0                  |
    | 4    | c   | 20.0                  |
    | 5    | a   | 21.0                  |
    | 6    | b   | 24.0                  |
```

Aggregation Primitive

Finally, there is also the aggregation primitive “sum”. If we use sum, it will calculate the sum for the group at the cutoff time for each row. Because we didn’t specify a cutoff time it will use all the data for each group for each row.
If we set the cutoff time of each row to be the time index, then use sum as an aggregation primitive, the result is the same as cum_sum. (Though the order is different in the displayed dataframe).

How do I get a list of all Aggregation and Transform primitives?

You can do `featuretools.list_primitives()` to get all the primitive in Featuretools. It will return a Dataframe with the names, type, and description of the primitives. You can also visit primitives.featurelabs.com to obtain a list of all available primitives.
How do I change the units for a TimeSince primitive?

There are a few primitives in Featuretools that make some time-based calculation. These include TimeSince, TimeSincePrevious, TimeSinceLast, TimeSinceFirst.

You can change the units from the default seconds to any valid time unit, by doing the following:

```python
from featuretools.primitives import TimeSince, TimeSincePrevious, TimeSinceLast,
→TimeSinceFirst

time_since = TimeSince(unit="minutes")
time_since_previous = TimeSincePrevious(unit="hours")
time_since_last = TimeSinceLast(unit="days")
time_since_first = TimeSinceFirst(unit="years")

es = ft.demo.load_mock_customer(return_entityset=True)

feature_matrix, feature_defs = ft.dfs(entityset=es,
→target_entity="customers",
→agg_primitives=[time_since_last, time_since_
→previous],
→trans_primitives=[time_since, time_since_
→previous])
```
Above, we changed the units to the following: - minutes for \texttt{TimeSince} - hours for \texttt{TimeSincePrevious} - days for \texttt{TimeSinceLast} - years for \texttt{TimeSinceFirst}.

Now we can see that our feature matrix contains multiple features where the units for the \texttt{TimeSince} primitives are changed.

```python
[65]: feature_matrix.head()

| customer_id | zip_code | TIME_SINCE_LAST(sessions.session_start, unit=days) | \  
| 5          | 60091    | 2277.426374  |
| 4          | 60091    | 2277.537717  |
| 1          | 60091    | 2277.462485  |
| 3          | 13244    | 2277.397034  |
| 2          | 13244    | 2277.420355  |

| customer_id | TIME_SINCE_FIRST(sessions.session_start, unit=years) | \  
| 5          | 6.239617  |  |
| 4          | 6.239597  |  |
| 1          | 6.239566  |  |
| 3          | 6.239460  |  |
| 2          | 6.239650  |  |

| customer_id | TIME_SINCE_LAST(transactions.transaction_time, unit=days) | \  
| 5          | 2277.421108 |  |
| 4          | 2277.530946 |  |
| 1          | 2277.451200 |  |
| 3          | 2277.385749 |  |
| 2          | 2277.411328 |  |

| customer_id | TIME_SINCE_FIRST(transactions.transaction_time, unit=years) | \  
| 5          | 6.239617  |  |
| 4          | 6.239597  |  |
| 1          | 6.239566  |  |
| 3          | 6.239460  |  |
| 2          | 6.239650  |  |

| customer_id | TIME_SINCE(join_date, unit=minutes) | \  
| 5          | 5.099808e+06 |  |
| 4          | 4.717328e+06 |  |
| 1          | 4.704928e+06 |  |
| 3          | 4.534713e+06 |  |
| 2          | 4.180005e+06 |  |

| customer_id | TIME_SINCE_PREVIOUS(join_date, unit=hours) | \  
| 5          | NaN |  |
| 4          | 6374.673333 |  |
| 1          | 206.671944  |  |
| 3          | 2836.900278 |  |
| 2          | 5911.808333 |  |

| customer_id | TIME_SINCE_LAST(transactions.sessions.session_start, unit=days) | \  
| 5          | 2277.426374 |  |
```

(continues on next page)
There are now features where time unit is different from the default of seconds, such as \( \text{TIME\_SINCE\_LAST}(\text{sessions.session_start}, \text{unit=days}) \), and \( \text{TIME\_SINCE\_FIRST}(\text{sessions.session_start}, \text{unit=years}) \).

### 3.12.5 Modeling

#### How does my train & test data work with Featuretools and sklearn’s train\_test\_split?

You might be wondering how to properly use your train & test data with Featuretools, and sklearn’s \texttt{train\_test\_split}. There are a few things you must do to ensure accuracy with this workflow.

Let’s imagine we have a Dataframes for our train data, with the labels.

```python
[66]:
train_data = pd.DataFrame({
    "customer_id": [1, 2, 3, 4, 5],
    "age": [20, 25, 55, 22, 35],
    "gender": ["f", "m", "m", "m", "m"],
    "signup_date": pd.date_range('2010-01-01 01:41:50',
                                 periods=5, freq='25min'),
    "labels": [False, True, True, False, False]
})
```

```bash
[66]:
customer_id  age  gender  signup_date  labels
0   1   20    f  2010-01-01 01:41:50  False
1   2   25    m  2010-01-01 02:06:50  True
2   3   55    m  2010-01-01 02:31:50  True
3   4   22    m  2010-01-01 02:56:50  False
4   5   35    m  2010-01-01 03:21:50  False
```

Now we can create our \texttt{EntitySet} for the train data, and create our features. To prevent label leakage, we will use cutoff times (see \textit{earlier question}).

```python
[67]:
es_train_data = ft.EntitySet(id="customer_data")
es_train_data = es_train_data.entity_from_dataframe(entity_id="customers",
                                                  dataframe=train_data, index="customer_id")
cutoff_times = pd.DataFrame({
    "customer_id": [1, 2, 3, 4, 5],
    "time": pd.date_range('2014-01-01 01:41:50', periods=5, freq='25min'))
feature_matrix_train, features = ft.dfs(entityset=es_train_data, target_entity="customers",
                                 (continues on next page)
We will also encode our feature matrix to compatible for machine learning algorithms.

```python
[68]: feature_matrix_train_enc, feature_enc = ft.encode_features(feature_matrix_train,
                 --features)
```

We will also encode our feature matrix to compatible for machine learning algorithms.
<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 01:41:50</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 02:06:50</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 02:31:50</td>
</tr>
<tr>
<td>4</td>
<td>2014-01-01 02:56:50</td>
</tr>
<tr>
<td>5</td>
<td>2014-01-01 03:21:50</td>
</tr>
</tbody>
</table>

DAY(signup_date) is unknown

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 01:41:50</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 02:06:50</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 02:31:50</td>
</tr>
<tr>
<td>4</td>
<td>2014-01-01 02:56:50</td>
</tr>
<tr>
<td>5</td>
<td>2014-01-01 03:21:50</td>
</tr>
</tbody>
</table>

YEAR(signup_date) = 2010

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 01:41:50</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 02:06:50</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 02:31:50</td>
</tr>
<tr>
<td>4</td>
<td>2014-01-01 02:56:50</td>
</tr>
<tr>
<td>5</td>
<td>2014-01-01 03:21:50</td>
</tr>
</tbody>
</table>

YEAR(signup_date) is unknown

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 01:41:50</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 02:06:50</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 02:31:50</td>
</tr>
<tr>
<td>4</td>
<td>2014-01-01 02:56:50</td>
</tr>
<tr>
<td>5</td>
<td>2014-01-01 03:21:50</td>
</tr>
</tbody>
</table>

MONTH(signup_date) = 1

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 01:41:50</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 02:06:50</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 02:31:50</td>
</tr>
<tr>
<td>4</td>
<td>2014-01-01 02:56:50</td>
</tr>
<tr>
<td>5</td>
<td>2014-01-01 03:21:50</td>
</tr>
</tbody>
</table>

MONTH(signup_date) is unknown

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 01:41:50</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 02:06:50</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 02:31:50</td>
</tr>
<tr>
<td>4</td>
<td>2014-01-01 02:56:50</td>
</tr>
<tr>
<td>5</td>
<td>2014-01-01 03:21:50</td>
</tr>
</tbody>
</table>

WEEKDAY(signup_date) = 4

<table>
<thead>
<tr>
<th>customer_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-01-01 01:41:50</td>
</tr>
<tr>
<td>2</td>
<td>2014-01-01 02:06:50</td>
</tr>
<tr>
<td>3</td>
<td>2014-01-01 02:31:50</td>
</tr>
<tr>
<td>4</td>
<td>2014-01-01 02:56:50</td>
</tr>
<tr>
<td>5</td>
<td>2014-01-01 03:21:50</td>
</tr>
</tbody>
</table>

WEEKDAY(signup_date) is unknown
from sklearn.model_selection import train_test_split

X = feature_matrix_train_enc.drop(['labels'], axis=1)
y = feature_matrix_train_enc['labels']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

Now you can use the encoded feature matrix with sklearn’s `train_test_split`. This will allow you to train your model, and tune your parameters.

### How are categorical variables encoded when splitting training and testing data?

You might be wondering what happens when categorical variables are encoded with your training and testing data. You might be curious to know what happens if the train data has a categorical variable that is not present in the testing data.

Let’s explore a simple example to see what happens during the encoding process.

```python
train_data = pd.DataFrame({'customer_id': [1, 2, 3, 4, 5],
                          'product_purchased': ['coke zero', 'car', 'toothpaste', 'coke zero', 'car']})
```

```python
es_train = ft.EntitySet(id="customer_data")
es_train = es_train.entity_from_dataframe(entity_id="customers",
dataframe=train_data,
index="customer_id")

feature_matrix_train, features = ft.dfs(entityset=es_train,
target_entity='customers')
feature_matrix_train
```

```
<table>
<thead>
<tr>
<th>customer_id</th>
<th>product_purchased</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>coke zero</td>
</tr>
<tr>
<td>2</td>
<td>car</td>
</tr>
<tr>
<td>3</td>
<td>toothpaste</td>
</tr>
<tr>
<td>4</td>
<td>coke zero</td>
</tr>
<tr>
<td>5</td>
<td>car</td>
</tr>
</tbody>
</table>
```

We will use `ft.encode_features` to properly encode the `product_purchased` column.

```python
feature_matrix_train_encoded, features_encoded = ft.encode_features(feature_matrix_train,
features)
feature_matrix_train_encoded.head()
```

```python
<table>
<thead>
<tr>
<th>customer_id</th>
<th>product_purchased = coke zero</th>
<th>product_purchased = car</th>
<th>product_purchased = toothpaste</th>
<th>product_purchased is unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

(continues on next page)
Now let's imagine we have some test data that doesn't have one of the categorical values (toothpaste). Also, the test data has a value that wasn't present in the train data (water).

```python
[72]: test_data = pd.DataFrame({"customer_id": [6, 7, 8, 9, 10],
                           "product_purchased": ["coke zero", "car", "coke zero",
                                             "coke zero", "water"]})
```

```python
es_test = ft.EntitySet(id="customer_data")
es_test = es_test.entity_from_dataframe(entity_id="customers",
dataframe=test_data,
index="customer_id")

feature_matrix_test = ft.calculate_feature_matrix(entityset=es_test,
features=features_encoded)
feature_matrix_test.head()
```

As seen above, we were able to successfully handle the encoding, and deal with the following complications: - toothpaste was present in the training data but not present in the testing data - water was present in the test data but not present in the training data.

### 3.12.6 Errors & Warnings

#### Why am I getting this error ‘Index is not unique on dataframe’?

You may be trying to create your `EntitySet`, and run into this error.

```
AssertionError: Index is not unique on dataframe
```

This is because each entity in your `EntitySet` needs a unique index.

Let's look at a simple example.

```python
[73]: product_df = pd.DataFrame({"id": [1, 2, 3, 4, 4],
                            'rating': [3.5, 4.0, 4.5, 1.5, 5.0]})
product_df
```

3.12. Frequently Asked Questions
Notice how the `id` column has a duplicate index of 4. If you try to create an entity with this Dataframe, you will run into the following error.

```python
es = ft.EntitySet(id="product_data")
es = es.entity_from_dataframe(entity_id="products",
dataframe=product_df,
index="id")
```

```
AssertionError Traceback (most recent call last)
<ipython-input-63-a6e02ba6fa47> in
   2 es = es.entity_from_dataframe(entity_id="products",
   3       dataframe=product_df,
----> 4       index="id")
~/featuretools/featuretools/entityset/entityset.py in entity_from_dataframe(self,
   486       entity_id, dataframe, index, variable_types, make_index, time_index, secondary_time_index,
   487       index, already_sorted)
     ...)
```

To fix the above error, you can do one of the following solutions:

**Solution #1 - You can create a unique index on your Dataframe.**

```python
product_df = pd.DataFrame({'id': [1, 2, 3, 4, 5],
```
Notice how we now have a unique index column called `id`.

```
es = es.entity_from_dataframe(entity_id="products",
dataframe=product_df,
index="id")
es
```

As seen above, we can now create our entity for our `EntitySet` without an error by creating a unique index in our Dataframe.

**Solution #2 - Set `make_index` to True in your call to `entity_from_dataframe` to create a new index on that data**

- `make_index` creates a unique index for each row by just looking at what number the row is, in relation to all the other rows.

```
product_df = pd.DataFrame({'id': [1, 2, 3, 4, 4],
                      'rating': [3.5, 4.0, 4.5, 1.5, 5.0]})
es = ft.EntitySet(id="product_data")
es = es.entity_from_dataframe(entity_id="products",
dataframe=product_df,
index="product_id",
make_index=True)
es['products'].df
```

As seen above, we created our entity for our `EntitySet` without an error using the `make_index` argument.

**Why am I getting the following warning ‘Using training_window but last_time_index is not set’?**

If you are using a training window, and you haven’t set a `last_time_index` for your entity, you will get this warning. The training window attribute in Featuretools limits the amount of past data that can be used while calculating...
a particular feature vector.

You can add the `last_time_index` to all entities automatically by calling `add_last_time_indexes()` after you create your EntitySet. This will remove the warning.

```python
[77]:
es = ft.demo.load_mock_customer(return_entityset=True)
es.add_last_time_indexes()
```

Now we can run DFS without getting the warning.

```python
[78]:
cutoff_times = pd.DataFrame()
cutoff_times['customer_id'] = [1, 2, 3, 1]
cutoff_times['time'] = pd.to_datetime(['2014-1-1 04:00', '2014-1-1 05:00', '2014-1-1 06:00', '2014-1-1 08:00'])
cutoff_times['label'] = [True, True, False, True]

feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_entity="customers",
                                      cutoff_time=cutoff_times,
                                      cutoff_time_in_index=True,
                                      training_window="1 hour")
```

**last_time_index vs. time_index**

- The `time_index` is when the instance was first known.
- The `last_time_index` is when the instance appears for the last time.
- For example, a customer’s session has multiple transactions which can happen at different points in time. If we are trying to count the number of sessions a user has in a given time period, we often want to count all the sessions that had any transaction during the training window. To accomplish this, we need to not only know when a session starts (`time_index`), but also when it ends (`last_time_index`). The last time that an instance appears in the data is stored as the last_time_index of an Entity.
- Once the last_time_index has been set, Featuretools will check to see if the last_time_index is after the start of the training window. That, combined with the cutoff time, allows DFS to discover which data is relevant for a given training window.

**Why am I getting errors with Featuretools on Google Colab?**

Google Colab, by default, has Featuretools 0.4.1 installed. You may run into issues following our newest guides, or latest documentation while using an older version of Featuretools. Therefore, we suggest you upgrade to the latest featuretools version by doing the following in your notebook in Google Colab:

```
!pip install -U featuretools
```

You may need to Restart the runtime by doing **Runtime -> Restart Runtime**. You can check latest Featuretools version by doing following:

```python
import featuretools as ft
print(ft.__version__)
```

You should see a version greater than 0.4.1
3.13 Help

Couldn’t find what you were looking for? The Featuretools community is happy to provide support to users of Featuretools.

3.13.1 Discussion

Conversation happens in the following places:

1. **General usage questions** are directed to StackOverflow with the #featuretools tag.
2. **Feature requests** can be made on the Feature Request Board.
3. **Bug reports** are managed on the GitHub issue tracker.
4. **Chat** and collaboration within the community occurs on Slack. For general usage questions, please post on Stack Overflow where answers are more searchable by other users.
5. **Everything else**, **core developers** can be reached at help@featuretools.com.

3.13.2 Asking for help

All users levels, including beginners, should feel free to ask questions and report bugs when using featuretools. You can get better answers if follow a few simple guidelines:

1. **Use the right resource**: We suggest using Github or StackOverflow. Questions asked at these locations will be more searchable for other users.
   - Slack should be used for community discussion and collaboration.
   - For general questions on how something should work or tips, use StackOverflow.
   - Bugs should be reported on Github.
2. **Ask in one place only**: Please post your question in one place (StackOverflow or Github).
3. **Use examples**: Make minimal, complete, verifiable examples. You will get much better answers if your provide code that people can use to reproduce your problem.
4. **Sharing data privately**: If you have data that you can't share publicly, feel free to send an email to help@featurelabs.com.

3.13.3 Commercial support

Featuretools Enterprise offers expert support from the creators and core developers of Featuretools. Whether you need help getting a project off the ground or scaling Featuretools usage across your organization, we’ll provide our expertise to help you build the best machine learning models possible. More information can be found here.

3.14 Featuretools Enterprise

Featuretools Enterprise offers additional primitives, functionality, performance and expert support.
3.14.1 Premium Primitives

Feature primitives are the building blocks of Featuretools. They define individual computations that can be applied to raw datasets to create new features. Featuretools Enterprise contains over 100 domain-specific premium primitives to help you build better features for more accurate models. A list of all premium primitives can be obtained by visiting primitives.featurelabs.com.

3.14.2 Running Featuretools with Spark and Dask

Looking to easily scale Featuretools to bigger datasets or integrate it into your existing big data infrastructure? Whether it’s on-premise or in the cloud, you can run Featuretools Enterprise with Apache Spark and Dask. We have yet to encounter a dataset that is too large to handle.

The Featuretools development team is continually working to improve integration with Dask and Spark for performing feature engineering at scale. If you have a big data problem and are interested in testing our latest Dask or Spark integrations for free, please let us know by completing this simple request form.

3.14.3 Expert Support

All subscriptions come with guidance from the creators and core developers of Featuretools. Whether you need help getting a project off the ground or scaling Featuretools usage across your organization, we’ll provide our expertise to help you build the best machine learning models possible.

3.14.4 Early access

The team at Feature Labs is always working on the next big thing. With a subscription to Featuretools Enterprise, you can test and provide feedback on new tools and automation technologies before they are released.

Visit featurelabs.com to learn more about Featuretools Enterprise.

3.15 Limitations

3.15.1 In-memory

Featuretools is intended to be run on datasets that can fit in memory on one machine. For advice on handing large dataset refer to Improving Computational Performance.

If you would like to test Feature Labs APIs for running Featuretools natively on Apache Spark or Dask, please let us know here.

3.15.2 Bring your own labels

If you are doing supervised machine learning, you must supply your own labels and cutoff times. To structure this process, you can use Compose, which is an open source project for automatically generating labels with cutoff times.
3.16 Glossary

**child entity** An entity that references another entity via relationship. The “many” in a one-to-many relationship.

**cutoff time** The last point in time data is allowed to be used when calculating a feature

**entity** Equivalent to a table in relational database. Represented by the Entity class.

**EntitySet** A collection of entities and the relationships between them. Represented by the EntitySet class.

**feature** A transformation of data used for machine learning. featuretools has a custom language for defining features as described [here](#). All features are represented by subclasses of FeatureBase.

**feature engineering** The process of transforming data into representations that are better for machine learning.

**instance** Equivalent to a row in a relational database. Each entity has many instances, and each instance has a value for each variable and feature defined on the entity.

**parent entity** An entity that is referenced by another entity via relationship. The “one” in a one-to-many relationship.

**relationship** A mapping between a parent entity and a child entity. The child entity must contain a variable referencing the ID variable on the parent entity. Represented by the Relationship class.

**target entity** The entity on which we will be making a features for.

**variable** Equivalent to a column in a relational database. Represented by the Variable class.

3.17 Featuretools External Ecosystem

New projects are regularly being built on top of Featuretools, highlighting the importance of automated feature engineering. On this page, we have a list of libraries, use cases / demos, and tutorials that leverage Featuretools. If you would like to add a project, please contact us or submit a pull request on GitHub.

**Note:** We are proud and excited to share the work of people using Featuretools, but we cannot endorse or provide support for the tools on this page.

3.17.1 Libraries

**MLBlocks**

- MLBlocks is a simple framework for composing end-to-end tunable Machine Learning Pipelines by seamlessly combining tools from any python library with a simple, common and uniform interface. MLBlocks contains a primitive that uses Featuretools.

**Cardea**

- Cardea is a machine learning library built on top of the FHIR data schema. It uses a number of automl tools, including Featuretools.
3.17.2 Demos & Use Cases

Predict customer lifetime value

- A common use case for machine learning is to predict customer lifetime value. This article walks through the importance of this prediction problem using Featuretools in the process.

Predict NHL playoff matches

- Many users of Kaggle are eager to use Featuretools to improve their model performance. In this blog post, a Kaggle user takes a dataset of plays from National Hockey League games and creates a model to predict if a game is a playoff match.

Predict poverty of households in Costa Rica

- Social programs have a difficult time determining the right people to give aid. Using a dataset of Costa Rican household characteristics, this Kaggle kernel predicts the poverty of households.

Predicting Functional Threshold Power (FTP)

- This notebook and accompanying report evaluates the use of machine learning for predicting a cyclist’s FTP using data collected from previous training sessions. Featuretools is used to generate a set of independent variables that capture changes in performance over time.

Note: For more demos written by Feature Labs, see featuretools.com/demos

3.17.3 Tutorials

Automated Feature Engineering in Python

- This article provides a walk-through of how to use a retail dataset with DFS.

A Hands-On Guide to Automated Feature Engineering

- A in-depth tutorial that works through using Featuretools to predict future product sales at “BigMart”.

Simple Automatic Feature Engineering

- A walk-through that applies Featuretools to a sample dataset and creates a classifier to predict clients who make large orders.

Introduction to Automated Feature Engineering Using DFS

- This article demonstrates using Featuretools helps automate the manual process of feature engineering on a dataset of home loans.
Automated Feature Engineering Workshop

- An automated feature engineering workshop using Featuretools hosted at the 2017 Data Summer Conference.

Tutorial in Japanese

- A tutorial of Featuretools that demonstrates integrating with the feature selection library Boruta and the hyper parameter tuning library Optuna.

Building a Churn Prediction Model using Featuretools

- A video tutorial that shows how to build a churn prediction model using Featuretools along with Spark, XG-Boost, and Google Cloud Platform.

Automated Feature Engineering Workshop in Russian

- A video tutorial that shows how to predict if an applicant is capable of repaying a loan using Featuretools.

3.18 API Reference

3.18.1 Demo Datasets

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>load_retail([id, nrows, return_single_table])</code></td>
<td>Returns the retail entityset example.</td>
</tr>
<tr>
<td><code>load_mock_customer([n_customers, ...])</code></td>
<td>Return dataframes of mock customer data</td>
</tr>
<tr>
<td><code>load_flight([month_filter, ...])</code></td>
<td>Download, clean, and filter flight data from 2017.</td>
</tr>
</tbody>
</table>

```python
featuretools.demo.load_retail(id='demo_retail_data', nrows=None, return_single_table=False)
```

Returns the retail entityset example. The original dataset can be found here.

We have also made some modifications to the data. We changed the column names, converted the customer_id to a unique fake customer_name, dropped duplicates, added columns for total and cancelled and converted amounts from GBP to USD. You can download the modified CSV in gz compressed (7 MB) or uncompressed (43 MB) formats.

**Parameters**

- `id (str)` – Id to assign to EntitySet.
- `nrows (int)` – Number of rows to load of the underlying CSV. If None, load all.
- `return_single_table (bool)` – If True, return a CSV rather than an EntitySet. Default is False.

**Examples**

```python
In [1]: import featuretools as ft
```

(continues on next page)
In [2]: es = ft.demo.load_retail()

In [3]: es

Out[3]:

Entityset: demo_retail_data
Entities:
   orders (shape = [22190, 3])
   products (shape = [3684, 3])
   customers (shape = [4372, 2])
   order_products (shape = [401704, 7])

Load in subset of data

In [4]: es = ft.demo.load_retail(nrows=1000)

In [5]: es

Out[5]:

Entityset: demo_retail_data
Entities:
   orders (shape = [67, 5])
   products (shape = [606, 3])
   customers (shape = [50, 2])
   order_products (shape = [1000, 7])

featuretools.demo.load_mock_customer

featuretools.demo.load_mock_customer(n_customers=5, n_products=5, n_sessions=35, n_transactions=500, random_seed=0, return_single_table=False, return_entityset=False)

Return dataframes of mock customer data

featuretools.demo.load_flight

featuretools.demo.load_flight(month_filter=None, categorical_filter=None, nrows=None, demo=True, return_single_table=False, verbose=False)

Download, clean, and filter flight data from 2017. The original dataset can be found here.

Parameters

- **month_filter (list[int])** – Only use data from these months (example is [1, 2]). To skip, set to None.
- **categorical_filter (dict[str->str])** – Use only specified categorical values. Example is {'dest_city': ['Boston, MA'], 'origin_city': ['Boston, MA']} which returns all flights in OR out of Boston. To skip, set to None.
- **nrows (int)** – Passed to nrows in pd.read_csv. Used before filtering.
- **demo (bool)** – Use only two months of data. If False, use the whole year.
- **return_single_table (bool)** – Exit the function early and return a dataframe.
- **verbose (bool)** – Show a progress bar while loading the data.
Examples

```
In [1]: import featuretools as ft

In [2]: es = ft.demo.load_flight(.verbose=True,
   ...:  month_filter=[1],
   ...:  categorical_filter={'origin_city':['Boston, MA']}
->)
   ...
100%|xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx| 100/100 [01:16<00:00, 1.31it/s]

In [3]: es
Out[3]:
Entityset: Flight Data
  Entities:
   airports [Rows: 55, Columns: 3]
   flights [Rows: 613, Columns: 9]
   trip_logs [Rows: 9456, Columns: 22]
   airlines [Rows: 10, Columns: 1]
  Relationships:
   trip_logs.flight_id -> flights.flight_id
   flights.carrier -> airlines.carrier
   flights.dest -> airports.dest
```

3.18.2 Deep Feature Synthesis

`dfs(entities, relationships, entityset, ...)` Calculates a feature matrix and features given a dictionary of entities and a list of relationships.

**Parameters**

- **entities** (`dict[str -> tuple(pd.DataFrame, str, str)]`) – Dictionary of entities. Entries take the format (entity id -> (dataframe, id column, (time_column))).
- **relationships** (`list[(str, str, str, str)]`) – List of relationships between entities. List items are a tuple with the format (parent entity id, parent variable, child entity id, child variable).
- **entityset** (`EntitySet`) – An already initialized entityset. Required if entities and relationships are not defined.
- **target_entity** (`str`) – Entity id of entity on which to make predictions.
- **cutoff_time** *(pd.DataFrame or Datetime)* – Specifies times at which to calculate each instance. The resulting feature matrix will use data up to and including the cutoff_time. Can either be a DataFrame with ‘instance_id’ and ‘time’ columns, a DataFrame with the name of the index variable in the target entity and a time column, a list of values, or a single value to calculate for all instances. If the dataframe has more than two columns, any additional columns will be added to the resulting feature matrix.

- **instance_ids** *(list)* – List of instances on which to calculate features. Only used if cutoff_time is a single datetime.

- **agg_primitives** *(list[str or AggregationPrimitive], optional)* – List of Aggregation Feature types to apply.


- **trans_primitives** *(list[str or TransformPrimitive], optional)* – List of Transform Feature functions to apply.

  Default: [“day”, “year”, “month”, “weekday”, “haversine”, “num_words”, “num_characters”]

- **groupby_trans_primitives** *(list[str or TransformPrimitive], optional)* – list of Transform primitives to make GroupByTransformFeatures with

- **allowed_paths** *(list[list[str]*) – Allowed entity paths on which to make features.

- **max_depth** *(int)* – Maximum allowed depth of features.

- **ignore_entities** *(list[str], optional)* – List of entities to blacklist when creating features.

- **ignore_variables** *(dict[str -> list[str]], optional)* – List of specific variables within each entity to blacklist when creating features.

- **primitive_options** *(list[dict[str or tuple[str] -> dict] or dict[str or tuple[str] -> dict, optional])* – Specify options for a single primitive or a group of primitives. Lists of option dicts are used to specify options per input for primitives with multiple inputs. Each option dict can have the following keys:

  - **"include_entities"** List of entities to be included when creating features for the primitive(s). All other entities will be ignored (list[str]).

  - **"ignore_entities"** List of entities to be blacklisted when creating features for the primitive(s) (list[str]).

  - **"include_variables"** List of specific variables within each entity to include when creating features for the primitive(s). All other variables in a given entity will be ignored (dict[str -> list[str]]).

  - **"ignore_variables"** List of specific variables within each entity to blacklist when creating features for the primitive(s) (dict[str -> list[str]]).

  - **"include_groupby_entities"** List of Entities to be included when finding groupbys. All other entities will be ignored (list[str]).

  - **"ignore_groupby_entities"** List of Entities to be blacklisted when finding groupbys (list[str]).
"include_groupby_variables"  List of specific variables within each entity to include as groupbys, if applicable. All other variables in each entity will be ignored (dict[str -> list[str]]).

"ignore_groupby_variables"  List of specific variables within each entity to blacklist as groupbys (dict[str -> list[str]]).

- **seed_features** (list[FeatureBase]) – List of manually defined features to use.

- **drop_contains** (list[str], optional) – Drop features that contains these strings in name.

- **drop_exact** (list[str], optional) – Drop features that exactly match these strings in name.

- **where_primitives** (list[str or PrimitiveBase], optional) – List of Primitives names (or types) to apply with where clauses.

  Default:
  
  ```python
  ['count']
  ```

- **max_features** (int, optional) – Cap the number of generated features to this number. If -1, no limit.

- **features_only** (bool, optional) – If True, returns the list of features without calculating the feature matrix.

- **cutoff_time_in_index** (bool) – If True, return a DataFrame with a MultiIndex where the second index is the cutoff time (first is instance id). DataFrame will be sorted by (time, instance_id).

- **training_window** (Timedelta or str, optional) – Window defining how much time before the cutoff time data can be used when calculating features. If None, all data before cutoff time is used. Defaults to None. Month and year units are not relative when Pandas Timedeltas are used. Relative units should be passed as a Featuretools Timedelta or a string.

- **approximate** (Timedelta) – Bucket size to group instances with similar cutoff times by for features with costly calculations. For example, if bucket is 24 hours, all instances with cutoff times on the same day will use the same calculation for expensive features.

- **save_progress** (str, optional) – Path to save intermediate computational results.

- **n_jobs** (int, optional) – number of parallel processes to use when calculating feature matrix

- **chunk_size** (int or float or None or "cutoff time", optional) – Number of rows of output feature matrix to calculate at time. If passed an integer greater than 0, will try to use that many rows per chunk. If passed a float value between 0 and 1 sets the chunk size to that percentage of all instances. If passed the string “cutoff time”, rows are split per cutoff time.

- **dask_kwargs** (dict, optional) – Dictionary of keyword arguments to be passed when creating the dask client and scheduler. Even if n_jobs is not set, using dask_kwargs will enable multiprocessing. Main parameters:

  - **cluster** (str or dask.distributed.LocalCluster): cluster or address of cluster to send tasks to. If unspecified, a cluster will be created.

  - **diagnostics port** (int): port number to use for web dashboard. If left unspecified, web interface will not be enabled.
Valid keyword arguments for LocalCluster will also be accepted.

- **return_variable_types** ([list[Variable] or str, optional] – Types of variables to return. If None, default to Numeric, Discrete, and Boolean. If given as the string ‘all’, use all available variable types.

- **progress_callback** ([callable]) – function to be called with incremental progress updates. Has the following parameters:
  
  - **update**: percentage change (float between 0 and 100) in progress since last call
  - **progress_percent**: percentage (float between 0 and 100) of total computation completed
  - **time_elapsed**: total time in seconds that has elapsed since start of call

**Examples**

```python
from featuretools.primitives import Mean

# cutoff times per instance
entities = {
    "sessions" : (session_df, "id"),
    "transactions" : (transactions_df, "id", "transaction_time")
}
relationships = [("sessions", "id", "transactions", "session_id")]
feature_matrix, features = dfs(entities=entities,
                               relationships=relationships,
                               target_entity="transactions",
                               cutoff_time=cutoff_times)

feature_matrix

features = dfs(entities=entities,
               relationships=relationships,
               target_entity="transactions",
               features_only=True)
```

### 3.18.3 Wrappers

**Scikit-learn (BETA)**

`wrappers.DFSTransformer([entities, ...])` Transformer using Scikit-Learn interface for Pipeline uses.

```python
class featuretools.wrappers.DFSTransformer

wrappers.DFSTransformer([entities, ...])

Transformer using Scikit-Learn interface for Pipeline uses.
```

Transformer using Scikit-Learn interface for Pipeline uses.
__init__ (entities=None, relationships=None, entityset=None, target_entity=None, agg_primitives=None, trans_primitives=None, allowed_paths=None, max_depth=2, ignore_entities=None, ignore_variables=None, seed_features=None, drop_contains=None, drop_exact=None, where_primitives=None, max_features=-1, verbose=False)

Creates Transformer

Parameters

- **entities** (dict[str -> tuple(pd.DataFrame, str, str)]) – Dictionary of entities. Entries take the format {entity id -> (dataframe, id column, time_column)}.

- **relationships** (list[(str, str, str, str)]) – List of relationships between entities. List items are a tuple with the format (parent entity id, parent variable, child entity id, child variable).

- **entityset** (EntitySet) – An already initialized entityset. Required if entities and relationships are not defined.

- **target_entity** (str) – Entity id of entity on which to make predictions.

- **agg_primitives** (list[str or AggregationPrimitive], optional) – List of Aggregation Feature types to apply.
  
  Default: ['sum', 'std', 'max', 'skew', 'min', 'mean', 'count', 'percent_true', 'num_unique', 'mode']

- **trans_primitives** (list[str or TransformPrimitive], optional) – List of Transform Feature functions to apply.
  
  Default: ['day', 'year', 'month', 'weekday', 'haversine', 'num_words', 'num_characters']

- **allowed_paths** (list[list[str]]) – Allowed entity paths on which to make features.

- **max_depth** (int) – Maximum allowed depth of features.

- **ignore_entities** (list[str], optional) – List of entities to blacklist when creating features.

- **ignore_variables** (dict[str -> list[str]], optional) – List of specific variables within each entity to blacklist when creating features.

- **seed_features** (list[FeatureBase]) – List of manually defined features to use.

- **drop_contains** (list[str], optional) – Drop features that contains these strings in name.

- **drop_exact** (list[str], optional) – Drop features that exactly match these strings in name.

- **where_primitives** (list[str or PrimitiveBase], optional) – List of Primitives names (or types) to apply with where clauses.
  
  Default:
  
  ['count']

- **max_features** (int, optional) – Cap the number of generated features to this number. If -1, no limit.
Example

```python
In [1]: import featuretools as ft
In [2]: import pandas as pd
In [3]: from featuretools.wrappers import DFSTransformer
In [4]: from sklearn.pipeline import Pipeline
In [5]: from sklearn.ensemble import ExtraTreesClassifier

# Get example data
In [6]: n_customers = 3
In [7]: es = ft.demo.load_mock_customer(return_entityset=True, n_customers=5)
In [8]: y = [True, False, True]

# Build dataset
In [9]: pipeline = Pipeline(steps=[
    ...:     ('ft', DFSTransformer(entityset=es,
    ...:         target_entity=" customers",
    ...:         max_features=3)),
    ...:     ('et', ExtraTreesClassifier(n_estimators=100))
    ...: ])

# Fit and predict
In [10]: pipeline.fit([1, 2, 3], y=y) # fit on first 3 customers
Out[10]:
Pipeline(memory=None,
        steps=[('ft',
                <featuretools_sklearn_transformer.transformer.DFSTransformer
                object at 0x7f1eb21bf198>),
                ('et',
                 ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0,
                 class_weight=None, criterion='gini',
                 max_depth=None, max_features='auto',
                 max_leaf_nodes=None, max_samples=None,
                 min_impurity_decrease=0.0,
                 min_impurity_split=None,
                 min_samples_leaf=1, min_samples_split=2,
                 min_weight_fraction_leaf=0.0,
                 n_estimators=100, n_jobs=None,
                 oob_score=False, random_state=None,
                 verbose=0, warm_start=False))],
        verbose=False)
In [11]: pipeline.predict_proba([4,5]) # predict probability of each class on last 2
array([[0., 1.],
       [0., 1.]])
In [12]: pipeline.predict([4,5]) # predict on last 2
```
(continues on next page)
array([ True,  True])

# Same as above, but using cutoff times
In [13]: ct = pd.DataFrame()

In [14]: ct['customer_id'] = [1, 2, 3, 4, 5]

In [15]: ct['time'] = pd.to_datetime(['2014-1-1 04:00',
                           '2014-1-2 17:20',
                           '2014-1-4 09:53',
                           '2014-1-4 13:48',
                           '2014-1-5 15:32'])

In [16]: pipeline.fit(ct.head(3), y=y)
Out[16]: Pipeline(memory=None,
  steps=[('ft',
    <featuretools_sklearn_transformer.transformer.DFSTransformer object at 0x7f1eb21bf198>),
    ('et',
      ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0,
        class_weight=None, criterion='gini',
        max_depth=None, max_features='auto',
        max_leaf_nodes=None, max_samples=None,
        min_impurity_decrease=0.0,
        min_impurity_split=None,
        min_samples_leaf=1, min_samples_split=2,
        min_weight_fraction_leaf=0.0,
        n_estimators=100, n_jobs=None,
        oob_score=False, random_state=None,
        verbose=0, warm_start=False))],
  verbose=False)

In [17]: pipeline.predict_proba(ct.tail(2))
array([[0.46, 0.54],
       [0. , 1. ]])

In [18]: pipeline.predict(ct.tail(2))
array([ True,  True])

Methods

__init__(entities, relationships, ...) Creates Transformer
fit(cutoff_time_ids[, y]) Wrapper for DFS
fit_transform(X[, y]) Fit to data, then transform it.
get_params([deep])
transform(cutoff_time_ids) Wrapper for calculate_feature_matrix
3.18.4 Timedelta

Timedelta(value[, unit, delta_obj]) Represents differences in time.

featuretools.Timedelta

class featuretools.Timedelta(value, unit=None, delta_obj=None)
Represents differences in time.

Timedeltas can be defined in multiple units. Supported units:

- “ms” : milliseconds
- “s” : seconds
- “h” : hours
- “m” : minutes
- “d” : days
- “o”/”observations” : number of individual events
- “mo” : months
- “Y” : years

Timedeltas can also be defined in terms of observations. In this case, the Timedelta represents the period spanned by value.

For observation timedeltas:

```python
>>> three_observations_log = Timedelta(3, "observations")
>>> three_observations_log.get_name()
'3 Observations'
```

__init__ (value[, unit, delta_obj])

Parameters

- **value** (float, str, dict) – Value of timedelta, string providing both unit and value, or a dictionary of units and times.
- **unit** (str) – Unit of time delta.
- **delta_obj** (pd.Timedelta or pd.DateOffset) – A time object used internally to do time operations. If None is provided, one will be created using the provided value and unit.

Methods

__init__ (value[, unit, delta_obj])

<table>
<thead>
<tr>
<th>param value</th>
<th>Value of timedelta, string providing</th>
</tr>
</thead>
</table>

check_value(value, unit)

fix_units()

from_dictionary(dictionary)

get_arguments()

get_name()

get_unit_type()
### 3.18.5 Time utils

*make_temporal_cutoffs*(instance_ids, cutoffs)   Makes a set of equally spaced cutoff times prior to a set of input cutoffs and instance ids.

```python
featuretools.make_temporal_cutoffs
```

- **Parameters**
  - `instance_ids` *(list, np.ndarray, or pd.Series)* – list of instance ids. This function will make a new datetime series of multiple cutoff times for each value in this array.
  - `cutoffs` *(list, np.ndarray, or pd.Series)* – list of datetime objects associated with each instance id. Each one of these will be the last time in the new datetime series for each instance id
  - `window_size` *(pd.Timedelta, optional)* – amount of time between each datetime in each new cutoff series
  - `num_windows` *(int, optional)* – number of windows in each new cutoff series
  - `start` *(list, optional)* – list of start times for each instance id

### 3.18.6 Feature Primitives

A list of all Featuretools primitives can be obtained by visiting primitives.featurelabs.com.

**Primitive Types**
**TransformPrimitive()**  
Feature for entity that is based off one or more other features in that entity.

**AggregationPrimitive()**

---

**featuretools.primitives.TransformPrimitive**

class featuretools.primitives.TransformPrimitive

Feature for entity that is based off one or more other features in that entity.

    __init__()
    Initialize self. See help(type(self)) for accurate signature.

**Methods**

    __init__()
    Initialize self.

    generate_name(base_feature_names)
    generate_names(base_feature_names)
    get_args_string()
    get_arguments()
    get_filepath(filename)
    get_function()

**Attributes**

    base_of
    base_of_exclude
    commutative
    default_value
    input_types
    max_stack_depth
    name
    number_output_features
    return_type
    uses_calc_time
    uses_full_entity

---

**featuretools.primitives.AggregationPrimitive**

class featuretools.primitives.AggregationPrimitive

    __init__()
    Initialize self. See help(type(self)) for accurate signature.

**Methods**

    __init__()
    Initialize self.

Continued on next page
### Table 11 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>generate_name(base_feature_names,...)</code></td>
<td></td>
</tr>
<tr>
<td><code>generate_names(base_feature_names,...)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_args_string()</code></td>
<td></td>
</tr>
<tr>
<td><code>get_arguments()</code></td>
<td></td>
</tr>
<tr>
<td><code>get_filepath(filename)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_function()</code></td>
<td></td>
</tr>
</tbody>
</table>

#### Attributes

- `base_of`
- `base_of_exclude`
- `commutative`
- `default_value`
- `input_types`
- `max_stack_depth`
- `name`
- `number_output_features`
- `return_type`
- `stack_on`
- `stack_on_exclude`
- `stack_on_self`
- `uses_calc_time`

#### Primitive Creation Functions

- `make_agg_primitive(function, input_types,...)` Returns a new aggregation primitive class.
- `make_trans_primitive(function, input_types,...)` Returns a new transform primitive class

```python
featuretools.primitives.make_agg_primitive

featuretools.primitives.make_agg_primitive(function, input_types, return_type, name=None, stack_on_self=True, stack_on=None, stack_on_exclude=None, base_of=None, base_of_exclude=None, description=None, cls_attributes=None, uses_calc_time=False, defualt_value=None, commutative=False, number_output_features=1)
```

Returns a new aggregation primitive class. The primitive infers default values by passing in empty data.

#### Parameters

- **function** (`function`) – Function that takes in a series and applies some transformation to it.
- **input_types** (`list[Variable]`) – Variable types of the inputs.
- **return_type** (`Variable`) – Variable type of return.
- **name** (`str`) – Name of the function. If no name is provided, the name of `function` will be used.
- **stack_on_self**(bool) – Whether this primitive can be in input_types of self.
- **stack_on**(list[PrimitiveBase]) – Whitelist of primitives that can be input_types.
- **stack_on_exclude**(list[PrimitiveBase]) – Blacklist of primitives that cannot be input_types.
- **base_of**(list[PrimitiveBase]) – Whitelist of primitives that can have this primitive in input_types.
- **base_of_exclude**(list[PrimitiveBase]) – Blacklist of primitives that cannot have this primitive in input_types.
- **description**(str) – Description of primitive.
- **cls_attributes**(dict[str -> anytype]) – Custom attributes to be added to class. Key is attribute name, value is the attribute value.
- **uses_calc_time**(bool) – If True, the cutoff time the feature is being calculated at will be passed to the function as the keyword argument ‘time’.
- **default_value**(Variable) – Default value when creating the primitive to avoid the inference step. If no default value if provided, the inference happen.
- **commutative**(bool) – If True, will only make one feature per unique set of base features.
- **number_output_features**(int) – The number of output features (columns in the matrix) associated with this feature.

**Example**

```python
In [1]: from featuretools.primitives import make_agg_primitive
In [2]: from featuretools.variable_types import DatetimeTimeIndex, Numeric
In [3]: def time_since_last(values, time=None):
   ...:     time_since = time - values.iloc[-1]
   ...:     return time_since.total_seconds()
   ...:
In [4]: TimeSinceLast = make_agg_primitive(
   ...:     function=time_since_last,
   ...:     input_types=[DatetimeTimeIndex],
   ...:     return_type=Numeric,
   ...:     description="Time since last related instance",
   ...:     uses_calc_time=True)
```

**featuretools.primitives.make_trans_primitive**

```
featuretools.primitives.make_trans_primitive(function, input_types, return_type, name=None, description=None, cls_attributes=None, uses_calc_time=False, commutative=False, number_output_features=1)
```

Returns a new transform primitive class
Parameters

- **function**: Function that takes in a series and applies some transformation to it.
- **input_types**: Variable types of the inputs.
- **return_type**: Variable type of return.
- **name**: Name of the primitive. If no name is provided, the name of function will be used.
- **description**: Description of primitive.
- **cls_attributes**: Custom attributes to be added to class. Key is attribute name, value is the attribute value.
- **uses_calc_time**: If True, the cutoff time the feature is being calculated at will be passed to the function as the keyword argument ‘time’.
- **commutative**: If True, will only make one feature per unique set of base features.
- **number_output_features**: The number of output features (columns in the matrix) associated with this feature.

Example

```python
In [1]: from featuretools.primitives import make_trans_primitive

In [2]: from featuretools.variable_types import Variable, Boolean

In [3]: def pd_is_in(array, list_of_outputs=None):
    ...:     if list_of_outputs is None:
    ...:         list_of_outputs = []
    ...:     return pd.Series(array).isin(list_of_outputs)

In [4]: def isin_generate_name(self):
    ...:     return u"%s.isin(%s)" % (self.base_features[0].get_name(),
    ...:                        str(self.kwargs['list_of_outputs']))

In [5]: IsIn = make_trans_primitive(
    ...:     function=pd_is_in,
    ...:     input_types=[Variable],
    ...:     return_type=Boolean,
    ...:     name="is_in",
    ...:     description="For each value of the base feature, checks "
    ...:     "whether it is in a list that provided.",
    ...:     cls_attributes={"generate_name": isin_generate_name})
```

Aggregation Primitives

| Count() | Determines the total number of values, excluding NaN. | Continued on next page |
Table 14 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Mean([skipna])</code></td>
<td>Computes the average for a list of values.</td>
</tr>
<tr>
<td><code>Sum()</code></td>
<td>Calculates the total addition, ignoring NaN.</td>
</tr>
<tr>
<td><code>Min()</code></td>
<td>Calculates the smallest value, ignoring NaN values.</td>
</tr>
<tr>
<td><code>Max()</code></td>
<td>Calculates the highest value, ignoring NaN values.</td>
</tr>
<tr>
<td><code>Std()</code></td>
<td>Computes the dispersion relative to the mean value, ignoring NaN.</td>
</tr>
<tr>
<td><code>Median()</code></td>
<td>Determines the middlemost number in a list of values.</td>
</tr>
<tr>
<td><code>Mode()</code></td>
<td>Determines the most commonly repeated value.</td>
</tr>
<tr>
<td><code>AvgTimeBetween([unit])</code></td>
<td>Computes the average number of seconds between consecutive events.</td>
</tr>
<tr>
<td><code>TimeSinceLast([unit])</code></td>
<td>Calculates the time elapsed since the last datetime (default in seconds).</td>
</tr>
<tr>
<td><code>TimeSinceFirst([unit])</code></td>
<td>Calculates the time elapsed since the first datetime (in seconds).</td>
</tr>
<tr>
<td><code>NumUnique()</code></td>
<td>Determines the number of distinct values, ignoring NaN values.</td>
</tr>
<tr>
<td><code>PercentTrue()</code></td>
<td>Determines the percent of True values.</td>
</tr>
<tr>
<td><code>All()</code></td>
<td>Calculates if all values are ‘True’ in a list.</td>
</tr>
<tr>
<td><code>Any()</code></td>
<td>Determines if any value is ‘True’ in a list.</td>
</tr>
<tr>
<td><code>First()</code></td>
<td>Determines the first value in a list.</td>
</tr>
<tr>
<td><code>Last()</code></td>
<td>Determines the last value in a list.</td>
</tr>
<tr>
<td><code>Skew()</code></td>
<td>Computes the extent to which a distribution differs from a normal distribution.</td>
</tr>
<tr>
<td><code>Trend()</code></td>
<td>Calculates the trend of a variable over time.</td>
</tr>
<tr>
<td><code>Entropy([dropna, base])</code></td>
<td>Calculates the entropy for a categorical variable.</td>
</tr>
</tbody>
</table>

featuretools.primitives.Count

class featuretools.primitives.Count

Determines the total number of values, excluding NaN.

Examples

```python
>>> count = Count()
>>> count([1, 2, 3, 4, 5, None])
5
```

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()

Initialize self.

generate_name(base_feature_names,...)
generate_names(base_feature_names,...)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()
Attributes

- base_of
- base_of_exclude
- commutative
- default_value
- input_types
- max_stack_depth
- name
- number_output_features
- stack_on
- stack_on_exclude
- stack_on_self
- uses_calc_time

**featuretools.primitives.Mean**

*class featuretools.primitives.Mean*(skipna=True)*

Computes the average for a list of values.

**Parameters**

- *skipna* (bool) – Determines if to use NA/null values. Defaults to True to skip NA/null.

**Examples**

```python
>>> mean = Mean()
>>> mean([1, 2, 3, 4, 5, None])
3.0
```

We can also control the way NaN values are handled.

```python
>>> mean = Mean(skipna=False)
>>> mean([1, 2, 3, 4, 5, None])
nan
```

**__init__**(skipna=True)

Initialize self. See help(type(self)) for accurate signature.

**Methods**

**__init__**([skipna])

Initialize self.

- generate_name(base_feature_names,...)
- generate_names(base_feature_names,...)
- get_args_string()
- get_arguments()
- get_filepath(filename)
- get_function()
Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time

featuretools.primitives.Sum

class featuretools.primitives.Sum
Calculates the total addition, ignoring NaN.

Examples

```python
>>> sum = Sum()
>>> sum([1, 2, 3, 4, 5, None])
15.0
```

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()
Initialize self.
generate_name(base_feature_names,...)
generate_names(base_feature_names,...)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
<table>
<thead>
<tr>
<th>name</th>
<th>number_output_features</th>
<th>stack_on</th>
<th>stack_on_exclude</th>
<th>stack_on_self</th>
<th>uses_calc_time</th>
</tr>
</thead>
</table>

### featuretools.primitives.Min

class featuretools.primitives.Min

Calculates the smallest value, ignoring NaN values.

#### Examples

```python
>>> min = Min()
>>> min([1, 2, 3, 4, 5, None])
1.0
```

**__init__()**

Initialize self. See help(type(self)) for accurate signature.

#### Methods

- __init__(): Initialize self.
- generate_name(base_feature_names,...)
- generate_names(base_feature_names,...)
- get_args_string()
- get_arguments()
- get_filepath(filename)
- get_function()

#### Attributes

- base_of
- base_of_exclude
- commutative
- default_value
- input_types
- max_stack_depth
- name
- number_output_features
- stack_on
- stack_on_exclude
- stack_on_self
- uses_calc_time
**featuretools.primitives.Max**

```python
class featuretools.primitives.Max
    Calculates the highest value, ignoring NaN values.

Examples

```python
>>> max = Max()
>>> max([1, 2, 3, 4, 5, None])
5.0
```

__init__()
Initializes self. See help(type(self)) for accurate signature.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong></td>
</tr>
<tr>
<td>generate_name</td>
</tr>
<tr>
<td>generate_names</td>
</tr>
<tr>
<td>get_args_string</td>
</tr>
<tr>
<td>get_arguments</td>
</tr>
<tr>
<td>get_filepath</td>
</tr>
<tr>
<td>get_function</td>
</tr>
</tbody>
</table>

**Attributes**

- base_of
- base_of_exclude
- commutative
- default_value
- input_types
- max_stack_depth
- name
- number_output_features
- stack_on
- stack_on_exclude
- stack_on_self
- uses_calc_time

**featuretools.primitives.Std**

```python
class featuretools.primitives.Std
    Computes the dispersion relative to the mean value, ignoring NaN.

Examples

```
```python
>>> std = Std()
>>> round(std([1, 2, 3, 4, 5, None]), 3)
1.414
```

```python
def __init__(self):
    Initialize self. See help(type(self)) for accurate signature.
```

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__()</code></td>
<td>Initialize self.</td>
</tr>
<tr>
<td><code>generate_name(base_feature_names, ...)</code></td>
<td></td>
</tr>
<tr>
<td><code>generate_names(base_feature_names, ...)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_args_string()</code></td>
<td></td>
</tr>
<tr>
<td><code>get_arguments()</code></td>
<td></td>
</tr>
<tr>
<td><code>get_filepath(filename)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_function()</code></td>
<td></td>
</tr>
</tbody>
</table>

### Attributes

- `base_of`
- `base_of_exclude`
- `commutative`
- `default_value`
- `input_types`
- `max_stack_depth`
- `name`
- `number_output_features`
- `stack_on`
- `stack_on_exclude`
- `stack_on_self`
- `uses_calc_time`

**featuretools.primitives.Median**

```python
class featuretools.primitives.Median:
```

Determines the middlemost number in a list of values.

### Examples

```python
>>> median = Median()
>>> median([5, 3, 2, 1, 4])
3.0
```

*NaN* values are ignored.

```python
>>> median([5, 3, 2, 1, 4, None])
3.0
```

```python
def __init__(self):
    Initialize self. See help(type(self)) for accurate signature.
```
Methods

```python
__init__()   Initialize self.
generate_name(base_feature_names,...)
generate_names(base_feature_names,...)
get_args_string()  
generate_arguments()   
get_filepath(filename)
get_function()
```

Attributes

```python
base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time
```

**featuretools.primitives.Mode**

```python
class featuretools.primitives.Mode
```

Determines the most commonly repeated value.

**Description:** Given a list of values, return the value with the highest number of occurrences. If list is empty, return NaN.

**Examples**

```python
>>> mode = Mode()
>>> mode(['red', 'blue', 'green', 'blue'])
'blue'
```

```python
__init__()  
Initialize self. See help(type(self)) for accurate signature.
```

**Methods**

```python
__init__()   Initialize self.
generate_name(base_feature_names,...)
generate_names(base_feature_names,...)
get_args_string()  
```

Continued on next page
### Table 29 – continued from previous page

- `get_arguments()`
- `get_filepath(filename)`
- `get_function()`

#### Attributes

- `base_of`
- `base_of_exclude`
- `commutative`
- `default_value`
- `input_types`
- `max_stack_depth`
- `name`
- `number_output_features`
- `return_type`
- `stack_on`
- `stack_on_exclude`
- `stack_on_self`
- `uses_calc_time`

#### featuretools.primitives.AvgTimeBetween

**class** `featuretools.primitives.AvgTimeBetween(unit='seconds')`  
Computes the average number of seconds between consecutive events.

**Description:** Given a list of datetimes, return the average time (default in seconds) elapsed between consecutive events. If there are fewer than 2 non-null values, return `NaN`.

**Parameters**  
- **`unit (str)`** – Defines the unit of time. Defaults to seconds. Acceptable values: years, months, days, hours, minutes, seconds, milliseconds, nanoseconds

**Examples**

```python  
>>> from datetime import datetime  
>>> avg_time_between = AvgTimeBetween()  
... times = [datetime(2010, 1, 1, 11, 45, 0),  
...          datetime(2010, 1, 1, 11, 55, 15),  
...          datetime(2010, 1, 1, 11, 57, 30)]  
>>> avg_time_between(times)  
375.0  
>>> avg_time_between = AvgTimeBetween(unit="minutes")  
>>> avg_time_between(times)  
6.25  
```

**__init__ (unit='seconds')**  
Initialize self. See help(type(self)) for accurate signature.

#### Methods
__init__([unit]) Initialize self.

generate_name(base_feature_names, ...)
generate_names(base_feature_names, ...)

get_args_string()
get_arguments()

get_filepath(filename)

get_function()

Attributes

base_of
base_of_exclude

commutative
default_value

input_types

max_stack_depth

name

number_output_features

stack_on

stack_on_exclude

stack_on_self

uses_calc_time

featuretools.primitives.TimeSinceLast

class featuretools.primitives.TimeSinceLast (unit='seconds')

Calculates the time elapsed since the last datetime (default in seconds).

Description: Given a list of datetimes, calculate the time elapsed since the last datetime (default in seconds).

Uses the instance’s cutoff time.

Parameters:

- unit (str) – Defines the unit of time to count from. Defaults to seconds. Acceptable values: years, months, days, hours, minutes, seconds, milliseconds, nanoseconds

Examples

```python
>>> from datetime import datetime
>>> time_since_last = TimeSinceLast()
>>> cutoff_time = datetime(2010, 1, 1, 12, 0, 0)
>>> times = [datetime(2010, 1, 1, 11, 45, 0),
...          datetime(2010, 1, 1, 11, 55, 15),
...          datetime(2010, 1, 1, 12, 0, 0)]
>>> time_since_last(times, time=cutoff_time)
150.0
```

```python
>>> from datetime import datetime
>>> time_since_last = TimeSinceLast(unit = "minutes")
>>> cutoff_time = datetime(2010, 1, 1, 12, 0, 0)
>>> times = [datetime(2010, 1, 1, 11, 45, 0),
...          datetime(2010, 1, 1, 11, 55, 15),
...          datetime(2010, 1, 1, 12, 0, 0)]
```

(continues on next page)
... datetime(2010, 1, 1, 11, 57, 30)]
>>> time_since_last(times, time=cutoff_time)
2.5

__init__(unit='seconds')
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(unit)
Generate names.
generate_name(base_feature_names,...)
generate_names(base_feature_names,...)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time

featuretools.primitives.TimeSinceLast

class featuretools.primitives.TimeSinceFirst(unit='seconds')
Calculates the time elapsed since the first datetime (in seconds).

Description: Given a list of datetimes, calculate the time elapsed since the first datetime (in seconds). Uses the
instance’s cutoff time.

Parameters unit (str) – Defines the unit of time to count from. Defaults to seconds. Acceptable
values: years, months, days, hours, minutes, seconds, milliseconds, nanoseconds

Examples

>>> from datetime import datetime
>>> time_since_first = TimeSinceFirst()
>>> cutoff_time = datetime(2010, 1, 1, 12, 0, 0)
>>> times = [datetime(2010, 1, 1, 11, 45, 0),
...          datetime(2010, 1, 1, 11, 55, 15),
...          datetime(2010, 1, 1, 11, 57, 30)]

>>> time_since_first(times, time=cutoff_time)
900.0

>>> from datetime import datetime

>>> time_since_first = TimeSinceFirst(unit = "minutes")

>>> cutoff_time = datetime(2010, 1, 1, 12, 0, 0)

>>> times = [datetime(2010, 1, 1, 11, 45, 0),
...          datetime(2010, 1, 1, 11, 55, 15),
...          datetime(2010, 1, 1, 11, 57, 30)]

>>> time_since_first(times, time=cutoff_time)
15.0

__init__(unit='seconds')
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(unit)
Initialize self.
generate_name(base_feature_names,...)
generate_names(base_feature_names,...)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
comutative
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time

featuretools.primitives.NumUnique

class featuretools.primitives.NumUnique
Determines the number of distinct values, ignoring NaN values.
Examples

```python
>>> num_unique = NumUnique()
>>> num_unique(['red', 'blue', 'green', 'yellow'])
4

NaN values will be ignored.

>>> num_unique(['red', 'blue', 'green', 'yellow', None])
4
```

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

- __init__(): Initialize self.
- generate_name(base_feature_names, ...)
- generate_names(base_feature_names, ...)
- get_args_string()
- get_arguments()
- get_filepath(filename)
- get_function()

Attributes

- base_of
- base_of_exclude
- commutative
- default_value
- input_types
- max_stack_depth
- name
- number_output_features
- stack_on
- stack_on_exclude
- stack_on_self
- uses_calc_time

featuretools.primitives.PercentTrue

class featuretools.primitives.PercentTrue

Determines the percent of True values.

Description: Given a list of booleans, return the percent of values which are True as a decimal. NaN values are treated as False, adding to the denominator.
Examples

```python
>>> percent_true = PercentTrue()
>>> percent_true([True, False, True, True, None])
0.6
```

Methods

```python
__init__()
    Initialize self. See help(type(self)) for accurate signature.
```

Attributes

```python
base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time
```

`featuretools.primitives.All`

class featuretools.primitives.All
    Calculates if all values are ‘True’ in a list.

Description: Given a list of booleans, return `True` if all of the values are `True`.

Examples

```python
>>> all = All()
>>> all([False, False, False, True])
False
```

__init__()
    Initialize self. See help(type(self)) for accurate signature.
Methods

```python
__init__()  # Initialize self.
generate_name(base_feature_names,...)
generate_names(base_feature_names,...)
get_args_string()
get_arguments()
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
get_args_string()
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
get_arguments()
generate_names(base_feature_names,...)
```

Attributes

```python
base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time
```

```python
class featuretools.primitives.Any
class featuretools.primitives.Any
    Determines if any value is ‘True’ in a list.
    
    Description: Given a list of booleans, return True if one or more of the values are True.

Examples

```python
>>> any = Any()
>>> any([False, False, False, True])
True
```

```python
__init__()
    Initialize self. See help(type(self)) for accurate signature.
```

Methods

```python
__init__()
    Initialize self.
generate_name(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
generate_names(base_feature_names,...)
```
### Table 43 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_filepath(filename)</td>
<td></td>
</tr>
<tr>
<td>get_function()</td>
<td></td>
</tr>
</tbody>
</table>

### Attributes

- base_of
- base_of_exclude
- commutative
- default_value
- input_types
- max_stack_depth
- name
- number_output_features
- stack_on
- stack_on_exclude
- stack_on_self
- uses_calc_time

### featuretools.primitives.First

**class featuretools.primitives.First**

Determines the first value in a list.

#### Examples

```python
>>> first = First()
>>> first([1, 2, 3, 4, 5, None])
1.0
```

__init__()

Initialize self. See help(type(self)) for accurate signature.

#### Methods

- __init__() Initialize self.
- generate_name(base_feature_names,...)
- generate_names(base_feature_names,...)
- get_args_string()
- get_arguments()
- get_filepath(filename)
- get_function()
Table 46 – continued from previous page

<table>
<thead>
<tr>
<th>commutative</th>
<th>default_value</th>
<th>input_types</th>
<th>max_stack_depth</th>
<th>name</th>
<th>number_output_features</th>
<th>return_type</th>
<th>stack_on</th>
<th>stack_on_exclude</th>
<th>stack_on_self</th>
<th>uses_calc_time</th>
</tr>
</thead>
</table>

**featuretools.primitives.Last**

**class featuretools.primitives.Last**

Determines the last value in a list.

**Examples**

```python
>>> last = Last()
>>> last([1, 2, 3, 4, 5, None])
nan
```

**__init__()**

Initialize self. See help(type(self)) for accurate signature.

**Methods**

**__init__()**

Initialize self.

**generate_name(base_feature_names,...)**

**generate_names(base_feature_names,...)**

**get_args_string()**

**get_arguments()**

**get_filepath(filename)**

**get_function()**

**Attributes**

**base_of**

**base_of_exclude**

**commutative**

**default_value**

**input_types**

**max_stack_depth**

**name**

**number_output_features**

**return_type**
Table 48 – continued from previous page

<table>
<thead>
<tr>
<th>stack_on</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack_on_exclude</td>
</tr>
<tr>
<td>stack_on_self</td>
</tr>
<tr>
<td>uses_calc_time</td>
</tr>
</tbody>
</table>

**featuretools.primitives.Skew**

class featuretools.primitives.Skew

Computes the extent to which a distribution differs from a normal distribution.

**Description:** For normally distributed data, the skewness should be about 0. A skewness value > 0 means that there is more weight in the left tail of the distribution.

**Examples**

```python
>>> skew = Skew()
>>> skew([1, 10, 30, None])
1.0437603722639681
```

**__init__()**

Initialize self. See help(type(self)) for accurate signature.

**Methods**

<table>
<thead>
<tr>
<th><strong>init</strong>()</th>
<th>Initialize self.</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate_name(base_feature_names,...)</td>
<td></td>
</tr>
<tr>
<td>generate_names(base_feature_names,...)</td>
<td></td>
</tr>
<tr>
<td>get_args_string()</td>
<td></td>
</tr>
<tr>
<td>get_arguments()</td>
<td></td>
</tr>
<tr>
<td>get_filepath(filename)</td>
<td></td>
</tr>
<tr>
<td>get_function()</td>
<td></td>
</tr>
</tbody>
</table>

**Attributes**

| base_of | |
| base_of_exclude | |
| commutative | |
| default_value | |
| input_types | |
| max_stack_depth | |
| name | |
| number_output_features | |
| stack_on | |
| stack_on_exclude | |
| stack_on_self | |
| uses_calc_time | |
featuretools.primitives.Trend

class featuretools.primitives.Trend
Calculates the trend of a variable over time.

Description: Given a list of values and a corresponding list of datetimes, calculate the slope of the linear trend of values.

Examples

```python
>>> from datetime import datetime
>>> trend = Trend()
>>> times = [datetime(2010, 1, 1, 11, 45, 0),
...          datetime(2010, 1, 1, 11, 55, 15),
...          datetime(2010, 1, 1, 11, 57, 30),
...          datetime(2010, 1, 1, 12),
...          datetime(2010, 1, 1, 12, 15)]
>>> round(trend([1, 2, 3, 4, 5], times), 3)
-0.053
```

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__() Initialize self.
generate_name(base_feature_names, ...)
generate_names(base_feature_names, ...)
get_args_string()\nget_arguments()\nget_filepath(filename)\nget_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time
class featuretools.primitives.Entropy (dropna=False, base=None)

Calculates the entropy for a categorical variable

Description: Given a list of observations from a categorical variable return the entropy of the distribution. NaN values can be treated as a category or dropped.

Parameters

- **dropna** (bool) – Whether to consider NaN values as a separate category Defaults to False.
- **base** (float) – The logarithmic base to use Defaults to e (natural logarithm)

Examples

```python
>>> pd_entropy = Entropy()
>>> pd_entropy([1,2,3,4])
1.3862943611198906
```

__init__ (dropna=False, base=None)

Initialize self. See help(type(self)) for accurate signature.

Methods

- __init__([dropna, base]) Initialize self.
- generate_name(base_feature_names,...)
- generate_names(base_feature_names,...)
- get_args_string()
- get_arguments()
- get_filepath(filename)
- get_function()

Attributes

- base_of
- base_of_exclude
- commutative
- default_value
- input_types
- max_stack_depth
- name
- number_output_features
- stack_on
- stack_on_exclude
- stack_on_self
- uses_calc_time

Transform Primitives
## Combine features

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IsIn</code></td>
<td>Determines whether a value is present in a provided list.</td>
</tr>
<tr>
<td><code>And()</code></td>
<td>Element-wise logical AND of two lists.</td>
</tr>
<tr>
<td><code>Or()</code></td>
<td>Element-wise logical OR of two lists.</td>
</tr>
<tr>
<td><code>Not()</code></td>
<td>Negates a boolean value.</td>
</tr>
</tbody>
</table>

### featuretools.primitives.IsIn

```python
class featuretools.primitives.IsIn(list_of_outputs=None):
    Determines whether a value is present in a provided list.
```

#### Examples

```python
>>> items = ['string', 10.3, False]
>>> is_in = IsIn(list_of_outputs=items)
>>> is_in(['string', 10.5, False]).tolist()
[True, False, True]
```

#### Methods

- `__init__(list_of_outputs=None)`: Initialize self. See help(type(self)) for accurate signature.

#### Attributes

- `base_of`
- `base_of_exclude`
- `commutative`
- `default_value`
- `input_types`
- `max_stack_depth`
- `name`
- `number_output_features`
- `uses_calc_time`
- `uses_full_entity`

3.18. API Reference
featuretools.primitives.And

```python
class featuretools.primitives.And
    Element-wise logical AND of two lists.

    Description: Given a list of booleans X and a list of booleans Y, determine whether each value in X is True, and whether its corresponding value in Y is also True.
```

**Examples**

```python
>>> _and = And()
>>> _and([False, True, False], [True, True, False]).tolist()
[False, True, False]
```

```python
__init__()

    Initialize self. See help(type(self)) for accurate signature.
```

**Methods**

```python
__init__()  

generate_name(base_feature_names)

generate_names(base_feature_names)

get_args_string()

get_arguments()

get_filepath(filename)

get_function()
```

**Attributes**

```python
base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity
```

---

featuretools.primitives.Or

```python
class featuretools.primitives.Or
    Element-wise logical OR of two lists.

    Description: Given a list of booleans X and a list of booleans Y, determine whether each value in X is True, or whether its corresponding value in Y is True.
```
Examples

```python
>>> _or = Or()
>>> _or([False, True, False], [True, True, False]).tolist()
[True, True, False]
```

___init___()

Initialize self. See help(type(self)) for accurate signature.

Methods

___init___()

Initialize self.

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.Not

class featuretools.primitives.Not

Negates a boolean value.

Examples

```python
>>> not_func = Not()
>>> not_func([True, True, False]).tolist()
[False, False, True]
```

___init___()

Initialize self. See help(type(self)) for accurate signature.

Methods
Attributes

- `base_of`
- `base_of_exclude`
- `commutative`
- `default_value`
- `input_types`
- `max_stack_depth`
- `name`
- `number_output_features`
- `uses_calc_time`
- `uses_full_entity`

General Transform Primitives

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Absolute()</code></td>
<td>Computes the absolute value of a number.</td>
</tr>
<tr>
<td><code>Percentile()</code></td>
<td>Determines the percentile rank for each value in a list.</td>
</tr>
<tr>
<td><code>TimeSince(unit)</code></td>
<td>Calculates time from a value to a specified cutoff date-time.</td>
</tr>
</tbody>
</table>

`featuretools.primitives.Absolute`

```python
>>> absolute = Absolute()
>>> absolute([3.0, -5.0, -2.4]).tolist()
[3.0, 5.0, 2.4]
```

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()

Initialize self.

generate_name(base_feature_names)
### Featuretools.primitives.Percentile

**class featuretools.primitives.Percentile**

Determines the percentile rank for each value in a list.

#### Examples

```python
>>> percentile = Percentile()
>>> percentile([10, 15, 1, 20]).tolist()
[0.5, 0.75, 0.25, 1.0]
```

Nan values are ignored when determining rank

```python
>>> percentile([10, 15, 1, None, 20]).tolist()
[0.5, 0.75, 0.25, nan, 1.0]
```

#### __init__()

Initialize self. See help(type(self)) for accurate signature.

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>()</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>generate_name(base_feature_names)</td>
<td>generate_names(base_feature_names)</td>
</tr>
</tbody>
</table>
Attributes

- `base_of`
- `base_of_exclude`
- `commutative`
- `default_value`
- `input_types`
- `max_stack_depth`
- `name`
- `number_output_features`
- `uses_calc_time`
- `uses_full_entity`

**featuretools.primitives.TimeSince**

```python
class featuretools.primitives.TimeSince(unit='seconds')

Calculates time from a value to a specified cutoff datetime.

**Parameters**
- `unit` *(str)* – Defines the unit of time to count from. Defaults to Seconds. Acceptable values: years, months, days, hours, minutes, seconds, milliseconds, nanoseconds

**Examples**

```python
>>> from datetime import datetime
>>> time_since = TimeSince()
>>> times = [datetime(2019, 3, 1, 0, 0, 0, 1),
...          datetime(2019, 3, 1, 0, 0, 1, 0),
...          datetime(2019, 3, 1, 0, 2, 0, 0)]
>>> cutoff_time = datetime(2019, 3, 1, 0, 0, 0, 0)
>>> values = time_since(array=times, time=cutoff_time)
>>> list(map(int, values))
[0, -1, -120]
```

Change output to nanoseconds

```python
>>> from datetime import datetime
>>> time_since_nano = TimeSince(unit='nanoseconds')
>>> times = [datetime(2019, 3, 1, 0, 0, 0, 1),
...          datetime(2019, 3, 1, 0, 0, 1, 0),
...          datetime(2019, 3, 1, 0, 2, 0, 0)]
>>> cutoff_time = datetime(2019, 3, 1, 0, 0, 0, 0)
>>> values = time_since_nano(array=times, time=cutoff_time)
>>> list(map(lambda x: int(round(x)), values))
[-1000, -1000000000, -120000000000]
```

`__init__`(unit='seconds')

Initialize self. See help(type(self)) for accurate signature.

**Methods**
__init__([unit]) Initialize self.

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

Datetime Transform Primitives

Second() Determines the seconds value of a datetime.
Minute() Determines the minutes value of a datetime.
Weekday() Determines the day of the week from a datetime.
IsWeekend() Determines if a date falls on a weekend.
Hour() Determines the hour value of a datetime.
Day() Determines the day of the month from a datetime.
Week() Determines the week of the year from a datetime.
Month() Determines the month value of a datetime.
Year() Determines the year value of a datetime.

featuretools.primitives.Second
class featuretools.primitives.Second
Determines the seconds value of a datetime.

Examples

```python
>>> from datetime import datetime
>>> dates = [datetime(2019, 3, 1), ...
    ...    datetime(2019, 3, 3, 11, 10, 50), ...
    ...    datetime(2019, 3, 31, 19, 45, 15)]
>>> second = Second()
>>> second(dates).tolist()
[0, 50, 15]
```
Featuretools Documentation, Release 0.13.4

__init__()  
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()  
Initialize self.

generate_name(base_feature_names)  
generate_names(base_feature_names)  
get_args_string()  
get_arguments()  
get_filepath(filename)  
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.Minute

class featuretools.primitives.Minute  
Determines the minutes value of a datetime.

Examples

```python
>>> from datetime import datetime
>>> dates = [datetime(2019, 3, 1),
...          datetime(2019, 3, 3, 11, 10, 50),
...          datetime(2019, 3, 31, 19, 45, 15)]
>>> minute = Minute()
>>> minute(dates).tolist()
[0, 10, 45]
```

__init__()  
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()  
Initialize self.  
Continued on next page
**Attributes**

- `base_of`
- `base_of_exclude`
- `commutative`
- `default_value`
- `input_types`
- `max_stack_depth`
- `name`
- `number_output_features`
- `uses_calc_time`
- `uses_full_entity`

**featuretools.primitives.Weekday**

class featuretools.primitives.Weekday

Determines the day of the week from a datetime.

**Description:** Returns the day of the week from a datetime value. Weeks start on Monday (day 0) and run through Sunday (day 6).

**Examples**

```python
>>> from datetime import datetime
>>> dates = [datetime(2019, 3, 1), ...
  ...   datetime(2019, 6, 17, 11, 10, 50), ...
  ...   datetime(2019, 11, 30, 19, 45, 15)]
>>> weekday = Weekday()
>>> weekday(dates).tolist()
[4, 0, 5]
```

**__init__**

Initialize self. See help(type(self)) for accurate signature.

**Methods**

```python
__init__()  Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
```
### featuretools.primitives.IsWeekend

#### class featuretools.primitives.IsWeekend

Determines if a date falls on a weekend.

#### Examples

```python
>>> from datetime import datetime
>>> dates = [datetime(2019, 3, 1), ...
          datetime(2019, 6, 17, 11, 10, 50), ...
          datetime(2019, 11, 30, 19, 45, 15)]
>>> is_weekend = IsWeekend()
>>> is_weekend(dates).tolist()
[False, False, True]
```

#### __init__()

Initialize self. See help(type(self)) for accurate signature.

#### Methods

- `__init__()`
- `generate_name(base_feature_names)`
- `generate_names(base_feature_names)`
- `get_args_string()`
- `get_arguments()`
- `get_filepath(filename)`
- `get_function()`

#### Attributes
**featuretools.primitives.Hour**

class featuretools.primitives.Hour
Determines the hour value of a datetime.

**Examples**

```python
>>> from datetime import datetime
>>> dates = [datetime(2019, 3, 1),
          ...          datetime(2019, 3, 3, 11, 10, 50),
          ...          datetime(2019, 3, 31, 19, 45, 15)]
>>> hour = Hour()
>>> hour(dates).tolist()
[0, 11, 19]
```

```
__init__()
Initialize self. See help(type(self)) for accurate signature.
```

**Methods**

```
__init__()
Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()
```

**Attributes**

```
base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity
```

---

3.18. API Reference 177

Continued on next page
Table 81 – continued from previous page

<table>
<thead>
<tr>
<th>number_output_features</th>
<th>uses_calc_time</th>
<th>uses_full_entity</th>
</tr>
</thead>
</table>

**featuretools.primitives.Day**

**class featuretools.primitives.Day**

Determines the day of the month from a datetime.

**Examples**

```python
>>> from datetime import datetime
>>> dates = [datetime(2019, 3, 1),
          ...           datetime(2019, 3, 3),
          ...           datetime(2019, 3, 31)]
>>> day = Day()
>>> day(dates).tolist()
[1, 3, 31]
```

**__init__**

Initialize self. See help(type(self)) for accurate signature.

**Methods**

<table>
<thead>
<tr>
<th><strong>init</strong>(self)</th>
<th>Initialize self.</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate_name(base_feature_names)</td>
<td></td>
</tr>
<tr>
<td>generate_names(base_feature_names)</td>
<td></td>
</tr>
<tr>
<td>get_args_string()</td>
<td></td>
</tr>
<tr>
<td>get_arguments()</td>
<td></td>
</tr>
<tr>
<td>get_filepath(filename)</td>
<td></td>
</tr>
<tr>
<td>get_function()</td>
<td></td>
</tr>
</tbody>
</table>

**Attributes**

| base_of | |
| base_of_exclude | |
| commutative | |
| default_value | |
| input_types | |
| max_stack_depth | |
| name | |
| number_output_features | |
| uses_calc_time | |
| uses_full_entity | |
featuretools.primitives.Week

class featuretools.primitives.Week
Determinedes the week of the year from a datetime.

Description: Returns the week of the year from a datetime value. The first week of the year starts on January 1, and week numbers increment each Monday.

Examples

```python
>>> from datetime import datetime
>>> dates = [datetime(2019, 1, 3),...
    ... datetime(2019, 6, 17, 11, 10, 50),
    ... datetime(2019, 11, 30, 19, 45, 15)]
>>> week = Week()
>>> week(dates).tolist()
[1, 25, 48]
```

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__() Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.Month

class featuretools.primitives.Month
Determines the month value of a datetime.
Examples

```python
>>> from datetime import datetime
>>> dates = [datetime(2019, 3, 1),
...          datetime(2019, 6, 17, 11, 10, 50),
...          datetime(2019, 11, 30, 19, 45, 15)]
>>> month = Month()
>>> month(dates).tolist()
[3, 6, 11]
```

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()
Initialize self.

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.Year

class featuretools.primitives.Year
Determines the year value of a datetime.

Examples

```python
>>> from datetime import datetime
>>> dates = [datetime(2019, 3, 1),
...          datetime(2048, 6, 17, 11, 10, 50),
...          datetime(1950, 11, 30, 19, 45, 15)]
>>> year = Year()
>>> year(dates).tolist()
[2019, 2048, 1950]
```
__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__() Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

Cumulative Transform Primitives

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diff()</td>
<td>Compute the difference between the value in a list and the previous value in that list.</td>
</tr>
<tr>
<td>TimeSincePrevious([unit])</td>
<td>Compute the time since the previous entry in a list.</td>
</tr>
<tr>
<td>CumCount()</td>
<td>Calculates the cumulative count.</td>
</tr>
<tr>
<td>CumSum()</td>
<td>Calculates the cumulative sum.</td>
</tr>
<tr>
<td>CumMean()</td>
<td>Calculates the cumulative mean.</td>
</tr>
<tr>
<td>CumMin()</td>
<td>Calculates the cumulative minimum.</td>
</tr>
<tr>
<td>CumMax()</td>
<td>Calculates the cumulative maximum.</td>
</tr>
</tbody>
</table>

featuretools.primitives.Diff

class featuretools.primitives.Diff
Compute the difference between the value in a list and the previous value in that list.

Description: Given a list of values, compute the difference from the previous item in the list. The result for the first element of the list will always be NaN. If the values are datetimes, the output will be a timedelta.
Examples

```python
>>> diff = Diff()
>>> values = [1, 10, 3, 4, 15]
>>> diff(values).tolist()
[nan, 9.0, -7.0, 1.0, 11.0]
```

```python
__init__()

Initialize self. See help(type(self)) for accurate signature.
```

Methods

```python
__init__()

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()
```

Attributes

```python
base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity
```

featuretools.primitives.TimeSincePrevious

class featuretools.primitives.TimeSincePrevious(unit='seconds')

Compute the time since the previous entry in a list.

Parameters

`unit` (`str`) – Defines the unit of time to count from. Defaults to Seconds. Acceptable
values: years, months, days, hours, minutes, seconds, milliseconds, nanoseconds

Description:
Given a list of datetimes, compute the time in seconds elapsed since the previous item in the list.
The result for the first item in the list will always be `NaN`.

Examples

```python
>>> from datetime import datetime

>>> time_since_previous = TimeSincePrevious()

>>> dates = [datetime(2019, 3, 1, 0, 0, 0),
```

(continues on next page)
...    datetime(2019, 3, 1, 0, 2, 0),
...    datetime(2019, 3, 1, 0, 3, 0),
...    datetime(2019, 3, 1, 0, 2, 30),
...    datetime(2019, 3, 1, 0, 10, 0)]

>>> time_since_previous(dates).tolist()
[nan, 120.0, 60.0, -30.0, 450.0]

__init__(unit='seconds')

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(unit)

Initialize self.

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.CumCount

class featuretools.primitives.CumCount

Calculates the cumulative count.

Description: Given a list of values, return the cumulative count (or running count). There is no set window, so the count at each point is calculated over all prior values. NaN values are counted.

Examples

>>> cum_count = CumCount()

>>> cum_count([1, 2, 3, 4, None, 5]).tolist()
[1, 2, 3, 4, 5, 6]

__init__()

Initialize self. See help(type(self)) for accurate signature.
Methods

__init__()  Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.CumSum

class featuretools.primitives.CumSum
Calculates the cumulative sum.

Description: Given a list of values, return the cumulative sum (or running total). There is no set window, so the sum at each point is calculated over all prior values. NaN values will return NaN, but in the window of a cumulative calculation, they’re ignored.

Examples

```python
>>> cum_sum = CumSum()
>>> cum_sum([1, 2, 3, 4, None, 5]).tolist()
[1.0, 3.0, 6.0, 10.0, nan, 15.0]
```

__init__()  Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()  Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
### featuretools.primitives.CumMean

**class featuretools.primitives.CumMean**

Calculates the cumulative mean.

**Description:** Given a list of values, return the cumulative mean (or running mean). There is no set window, so the mean at each point is calculated over all prior values. NaN values will return NaN, but in the window of a cumulative calculation, they’re treated as 0.

**Examples**

```python
>>> cum_mean = CumMean()
>>> cum_mean([1, 2, 3, 4, None, 5]).tolist()
[1.0, 1.5, 2.0, 2.5, nan, 2.5]
```

**__init__()**

Initialize self. See help(type(self)) for accurate signature.

**Methods**

**__init__()**

Initialize self.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate_name(base_feature_names)</td>
<td></td>
</tr>
<tr>
<td>generate_names(base_feature_names)</td>
<td></td>
</tr>
<tr>
<td>get_args_string()</td>
<td></td>
</tr>
<tr>
<td>get_arguments()</td>
<td></td>
</tr>
<tr>
<td>get_filepath(filename)</td>
<td></td>
</tr>
<tr>
<td>get_function()</td>
<td></td>
</tr>
</tbody>
</table>

**Attributes**
class featuretools.primitives.CumMin
Calculates the cumulative minimum.

Description: Given a list of values, return the cumulative min (or running min). There is no set window, so the
min at each point is calculated over all prior values. NaN values will return NaN, but in the window of a
cumulative calculation, they’re ignored.

Examples

```python
>>> cum_min = CumMin()
>>> cum_min([1, 2, -3, 4, None, 5]).tolist()
[1.0, 1.0, -3.0, -3.0, nan, -3.0]
```

__init__()  
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__() Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

Continued on next page
featuretools.primitives.CumMax

class featuretools.primitives.CumMax
Calculates the cumulative maximum.

Description: Given a list of values, return the cumulative max (or running max). There is no set window, so the max at each point is calculated over all prior values. NaN values will return NaN, but in the window of a cumulative calculation, they’re ignored.

Examples

```python
>>> cum_max = CumMax()
>>> cum_max([1, 2, 3, 4, None, 5]).tolist()
[1.0, 2.0, 3.0, 4.0, nan, 5.0]
```

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__() Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

Text Transform Primitives
**NumCharacters()**
Calculates the number of characters in a string.

**NumWords()**
Determines the number of words in a string by counting the spaces.

### featuretools.primitives.NumCharacters

**class** `featuretools.primitives.NumCharacters`
Calculates the number of characters in a string.

**Examples**

```python
>>> num_characters = NumCharacters()
>>> num_characters(["This is a string",
...                 "second item",
...                 "final1"]).tolist()
[16, 11, 6]
```

**__init__()**
Initialize self. See `help(type(self))` for accurate signature.

**Methods**

- `__init__()` Initialize self.
- `generate_name(base_feature_names)`
- `generate_names(base_feature_names)`
- `get_args_string()`
- `get_arguments()`
- `get_filepath(filename)`
- `get_function()`

**Attributes**

- `base_of`
- `base_of_exclude`
- `commutative`
- `default_value`
- `input_types`
- `max_stack_depth`
- `name`
- `number_output_features`
- `uses_calc_time`
- `uses_full_entity`

### featuretools.primitives.NumWords

**class** `featuretools.primitives.NumWords`
Determines the number of words in a string by counting the spaces.
Examples

```python
>>> num_words = NumWords()
>>> num_words(['This is a string',
...            'Two words',
...            'no-spaces',
...            'Also works with sentences. Second sentence!']).tolist()
[4, 2, 1, 6]
```

`__init__()`
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__()`
Initialize self.

`generate_name(base_feature_names)`

`generate_names(base_feature_names)`

`get_args_string()`

`get_arguments()`

`get_filepath(filename)`

`get_function()`

Attributes

`base_of`

`base_of_exclude`

`commutative`

`default_value`

`input_types`

`max_stack_depth`

`name`

`number_output_features`

`uses_calc_time`

`uses_full_entity`

Location Transform Primitives

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latitude()</td>
<td>Returns the first tuple value in a list of LatLong tuples.</td>
</tr>
<tr>
<td>Longitude()</td>
<td>Returns the second tuple value in a list of LatLong tuples.</td>
</tr>
<tr>
<td>Haversine(unit)</td>
<td>Calculates the approximate haversine distance between two LatLong variable types.</td>
</tr>
</tbody>
</table>

`featuretools.primitives.Latitude`

class `featuretools.primitives.Latitude`

`Returns the first tuple value in a list of LatLong tuples.` For use with the LatLong variable type.
Examples

```python
>>> latitude = Latitude()
>>> latitude([(42.4, -71.1),
...           (40.0, -122.4),
...           (41.2, -96.75)]).tolist()
[42.4, 40.0, 41.2]
```

`__init__()`

Initialize self. See help(type(self)) for accurate signature.

Methods

```python
__init__()

Initialize self.

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()
```

Attributes

<table>
<thead>
<tr>
<th>Base of</th>
<th>Base of exclude</th>
<th>Commutative</th>
<th>Default value</th>
<th>Input types</th>
<th>Max stack depth</th>
<th>Name</th>
<th>Number output features</th>
<th>Uses calc time</th>
<th>Uses full entity</th>
</tr>
</thead>
</table>

`featuretools.primitives.Longitude`

class `featuretools.primitives.Longitude`

Returns the second tuple value in a list of LatLong tuples. For use with the LatLong variable type.

Examples

```python
>>> longitude = Longitude()
>>> longitude([(42.4, -71.1),
...            (40.0, -122.4),
...            (41.2, -96.75)]).tolist()
[-71.1, -122.4, -96.75]
```

`__init__()`

Initialize self. See help(type(self)) for accurate signature.
Methods

__init__() Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.Haversine

class featuretools.primitives.Haversine(unit='miles')
Calculates the approximate haversine distance between two LatLong variable types.

Parameters unit (str) – Determines the unit value to output. Could be miles or kilometers. Default is miles.

Examples

>>> haversine = Haversine()
>>> distances = haversine([(42.4, -71.1), (40.0, -122.4)], ...
... [40.0, -122.4], (41.2, -96.75)])
>>> np.round(distances, 3).tolist()
[2631.231, 1343.289]

Output units can be specified

>>> haversine_km = Haversine(unit='kilometers')
>>> distances_km = haversine_km([(42.4, -71.1), (40.0, -122.4)], ...
... [(40.0, -122.4), (41.2, -96.75)])
>>> np.round(distances_km, 3).tolist()
[4234.555, 2161.814]

__init__ (unit='miles') Initialize self. See help(type(self)) for accurate signature.
Methods

```python
__init__([unit]) Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()
```

Attributes

```python
base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity
```

Natural Language Processing Primitives

Natural Language Processing primitives create features for textual data. For more information on how to use and install these primitives, see [here](#).

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiversityScore()</td>
<td>Calculates the overall complexity of the text based on the total</td>
</tr>
<tr>
<td>LSA()</td>
<td>Calculates the Latent Semantic Analysis Values of Text Input</td>
</tr>
<tr>
<td>MeanCharactersPerWord()</td>
<td>Determines the mean number of characters per word.</td>
</tr>
<tr>
<td>PartOfSpeechCount()</td>
<td>Calculates the occurrences of each different part of speech.</td>
</tr>
<tr>
<td>PolarityScore()</td>
<td>Calculates the polarity of a text on a scale from -1 (negative) to 1 (positive)</td>
</tr>
<tr>
<td>PunctuationCount()</td>
<td>Determines number of punctuation characters in a string.</td>
</tr>
<tr>
<td>StopwordCount()</td>
<td>Determines number of stopwords in a string.</td>
</tr>
<tr>
<td>TitleWordCount()</td>
<td>Determines the number of title words in a string.</td>
</tr>
<tr>
<td>UniversalSentenceEncoder()</td>
<td>Transforms a sentence or short paragraph to a vector using <a href="https://tfhub.dev/google/universal-sentence-encoder/2">tfhub model</a></td>
</tr>
<tr>
<td>UpperCaseCount()</td>
<td>Calculates the number of upper case letters in text.</td>
</tr>
</tbody>
</table>
nlp_primitives.DiversityScore

class nlp_primitives.DiversityScore

Calculates the overall complexity of the text based on the total number of words used in the text

Description: Given a list of strings, calculates the total number of unique words divided by the total number of words in order to give the text a score from 0-1 that indicates how unique the words used in it are. This primitive only evaluates the ‘clean’ versions of strings, so ignoring cases, punctuation, and stopwords in its evaluation.

If a string is missing, return NaN

Examples

```python
>>> diversity_score = DiversityScore()
>>> diversity_score(["hi hi hi", "hello its me", "hey what hey what", "a dog ate a basket"]).tolist()
[0.3333333333333333, 1.0, 0.5, 1.0]
```

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>()</td>
<td>Initialize self.</td>
</tr>
<tr>
<td>generate_name(base_feature_names)</td>
<td></td>
</tr>
<tr>
<td>generate_names(base_feature_names)</td>
<td></td>
</tr>
<tr>
<td>get_args_string()</td>
<td></td>
</tr>
<tr>
<td>get_arguments()</td>
<td></td>
</tr>
<tr>
<td>get_filepath(filename)</td>
<td></td>
</tr>
<tr>
<td>get_function()</td>
<td></td>
</tr>
</tbody>
</table>

Attributes

- base_of
- base_of_exclude
- commutative
- default_value
- input_types
- max_stack_depth
- name
- number_output_features
- uses_calc_time
- uses_full_entity

nlp_primitives.LSA

class nlp_primitives.LSA

Calculates the Latent Semantic Analysis Values of Text Input
Description: Given a list of strings, transforms those strings using tf-idf and single value decomposition to go from a sparse matrix to a compact matrix with two values for each string. These values represent that Latent Semantic Analysis of each string. These values will represent their context with respect to (nltk’s brown sentence corpus.) [https://www.nltk.org/book/ch02.html#brown-corpus]

If a string is missing, return NaN.

Examples

```python
>>> lsa = LSA()
>>> x = ["he helped her walk," , "me me me eat food", "the sentence doth long"]
>>> res = lsa(x).tolist()
>>> for i in range(len(res)): res[i] = [abs(round(x, 2)) for x in res[i]]
>>> res
[[0.0, 0.0, 0.01], [0.0, 0.0, 0.0]]
```

Now, if we change the values of the input corpus, to something that better resembles the given text, the same given input text will result in a different, more discerning, output. Also, NaN values are handled, as well as strings without words.

```python
>>> lsa = LSA()
>>> x = ["the earth is round", ",", np.NaN, ",./"]
>>> res = lsa(x).tolist()
>>> for i in range(len(res)): res[i] = [abs(round(x, 2)) for x in res[i]]
>>> res
[[0.01, 0.0, nan, 0.0], [0.0, 0.0, nan, 0.0]]
```

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__() Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time

Continued on next page
nlp_primitives.MeanCharactersPerWord

class nlp_primitives.MeanCharactersPerWord
Determines the mean number of characters per word.

Description: Given list of strings, determine the mean number of characters per word in each string. A word is defined as a series of any characters not separated by white space. Punctuation is removed before counting. If a string is empty or NaN, return NaN.

Examples

```python
>>> x = ['This is a test file', 'This is second line', 'third line $1,000']
>>> mean_characters_per_word = MeanCharactersPerWord()
>>> mean_characters_per_word(x).tolist()
[3.0, 4.0, 5.0]
```

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__() Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

nlp_primitives.PartOfSpeechCount

class nlp_primitives.PartOfSpeechCount
Calculates the occurrences of each different part of speech.
Description: Given a list of strings, categorize each word in the string as a different part of speech, and return the total count for each of 15 different categories of speech.

If a string is missing, return NaN.

Examples

```python
>>> x = ['He was eating cheese', '']
>>> part_of_speech_count = PartOfSpeechCount()
>>> part_of_speech_count(x).tolist()
[[0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 1.0], [0.0, 0.0], [0.0, 0.
˓→0], [0.0, 0.0], [0.0, 0.0], [1.0, 0.0], [0.0, 0.0], [0.0, 0.0], [0.0, 0.0], [1.
˓→0], [0.0, 0.0], [0.0, 0.0]]
```

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()

Initialize self.
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

nlp_primitives.PolarityScore

class nlp_primitives.PolarityScore

Calculates the polarity of a text on a scale from -1 (negative) to 1 (positive)

Description: Given a list of strings assign a polarity score from -1 (negative text), to 0 (neutral text), to 1 (positive text). The function returns a score for every given piece of text. If a string is missing, return ‘NaN’
Examples

```python
>>> x = ['He loves dogs', 'She hates cats', 'There is a dog', '']
>>> polarity_score = PolarityScore()
>>> polarity_score(x).tolist()
[0.677, -0.649, 0.0, 0.0]
```

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()
Initialize self.

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

nlp_primitives.PunctuationCount

class nlp_primitives.PunctuationCount
Determines number of punctuation characters in a string.

Description: Given list of strings, determine the number of punctuation characters in each string. Looks for any of the following:

!"#$%&'()*+,-./:;<=>?@[^_`]{|}~

If a string is missing, return NaN.

Examples

```python
>>> x = ['This is a test file.', 'This is second line', 'third line: $1,000']
>>> punctuation_count = PunctuationCount()
>>> punctuation_count(x).tolist()
[1.0, 0.0, 3.0]
```
__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

nlp_primitives.StopwordCount

class nlp_primitives.StopwordCount

Determines number of stopwords in a string.

Description: Given list of strings, determine the number of stopwords characters in each string. Looks for any of the English stopwords defined in nltk.corpus.stopwords. Case insensitive.

If a string is missing, return NaN.

Examples

```python
>>> x = ['This is a test string.', 'This is second string', 'third string']
>>> stopword_count = StopwordCount()
>>> stopword_count(x).tolist()
[3.0, 2.0, 0.0]
```

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods
__init__()  Initialize self.

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
gevaluate_arguments()
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
gevaluate_arguments()
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
gevaluate_arguments()
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
gevaluate_arguments()
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
gevaluate_arguments()
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
gevaluate_arguments()
generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
gevaluate_arguments()

Attributes

base_of
base_of_exclude
commutative
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

nlp_primitives.TitleWordCount

class nlp_primitives.TitleWordCount
   Determines the number of title words in a string.
       Description: Given list of strings, determine the number of title words in each string. A title word is defined as any word starting with a capital letter. Words at the start of a sentence will be counted.
       If a string is missing, return NaN.

Examples

   >>> x = ['My favorite movie is Jaws.', 'this is a string', 'AAA']
   >>> title_word_count = TitleWordCount()
   >>> title_word_count(x).tolist()
   [2.0, 0.0, 1.0]

   __init__()
   Initialize self. See help(type(self)) for accurate signature.

Methods

__init__()
   Initialize self.
    generate_name(base_feature_names)
    generate_names(base_feature_names)
    get_args_string()
    get_arguments()
    get_filepath(filename)
get_function()

Attributes

- base_of
- base_of_exclude
- commutative
- default_value
- input_types
- max_stack_depth
- name
- number_output_features
- uses_calc_time
- uses_full_entity

nlp_primitives.UniversalSentenceEncoder

class nlp_primitives.UniversalSentenceEncoder
    Transforms a sentence or short paragraph to a vector using [tfhub model](https://tfhub.dev/google/universal-sentence-encoder/2)

    Parameters
    None

    Examples

    >>> universal_sentence_encoder = UniversalSentenceEncoder()
    >>> sentences = ["I like to eat pizza",
    ...               "The roller coaster was built in 1885.",
    ...               ""
    ... ]
    >>> output = universal_sentence_encoder(sentences)
    >>> len(output)
    512
    >>> len(output[0])
    3
    >>> values = output[:3, 0]
    >>> [round(x, 4) for x in values]
    [0.0178, 0.0616, -0.0089]

    __init__()
    Initialize self. See help(type(self)) for accurate signature.

    Methods

    __init__()  
    Initialize self.
    generate_name(base_feature_names)
    generate_names(base_feature_names)
    get_args_string()
    get_arguments()
    get_filepath(filename)
Table 135 – continued from previous page

Attributes

- `base_of`
- `base_of_exclude`
- `commutative`
- `default_value`
- `input_types`
- `max_stack_depth`
- `name`
- `number_output_features`
- `uses_calc_time`
- `uses_full_entity`

**nlp_primitives.UpperCaseCount**

class nlp_primitives.UpperCaseCount

Calculates the number of upper case letters in text.

**Description:** Given a list of strings, determine the number of characters in each string that are capitalized. Counts every letter individually, not just every word that contains capitalized letters.

If a string is missing, return `NaN`

**Examples**

```python
>>> x = ['This IS a string.', 'This is a string', 'aaa']
>>> upper_case_count = UpperCaseCount()
>>> upper_case_count(x).tolist()
[3.0, 1.0, 0.0]
```

__init__()  
Initialize self. See help(type(self)) for accurate signature.

**Methods**

__init__()  
Initialize self.

generate_name(base_feature_names)
generate_names(base_feature_names)
get_args_string()
get_arguments()
get_filepath(filename)
get_function()
### Feature methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>FeatureBase.rename(name)</code></td>
<td>Rename Feature, returns copy</td>
</tr>
<tr>
<td><code>FeatureBase.get_depth(stop_at)</code></td>
<td>Returns depth of feature</td>
</tr>
</tbody>
</table>

#### featuretools.feature_base.FeatureBase.rename

`FeatureBase.rename(name)`

Rename Feature, returns copy

#### featuretools.feature_base.FeatureBase.get_depth

`FeatureBase.get_depth(stop_at=None)`

Returns depth of feature

### 3.18.7 Feature calculation

#### featuretools.calculate_feature_matrix

`calculate_feature_matrix(features[, ...])` Calculates a matrix for a given set of instance ids and calculation times.

#### featuretools.calculate_feature_matrix

`featuretools.calculate_feature_matrix(features, entityset=None, cutoff_time=None, instance_ids=None, entities=None, relationships=None, cutoff_time_in_index=False, training_window=None, approximate=None, save_progress=None, verbose=False, chunk_size=None, n_jobs=1, dask_kwargs=None, progress_callback=None)` Calculates a matrix for a given set of instance ids and calculation times.

**Parameters**

- **features** (list[FeatureBase]) – Feature definitions to be calculated.
- **entityset** (EntitySet) – An already initialized entityset. Required if `entities` and `relationships` not provided.
- **cutoff_time** (pd.DataFrame or Datetime) – Specifies at which time to cal-
calculate the features for each instance. The resulting feature matrix will use data up to and including the cutoff time. Can either be a DataFrame with ‘instance_id’ and ‘time’ columns, DataFrame with the name of the index variable in the target entity and a time column, or a single value to calculate for all instances. If the dataframe has more than two columns, any additional columns will be added to the resulting feature matrix.

- **instance_ids** *(list)* – List of instances to calculate features on. Only used if cutoff_time is a single datetime.

- **entities** *(dict[str -> tuple(pd.DataFrame, str, str)])* – dictionary of entities. Entries take the format {entity id: (dataframe, id column, time_column)}.

- **relationships** *(list[(str, str, str, str)])* – list of relationships between entities. List items are a tuple with the format (parent entity id, parent variable, child entity id, child variable).

- **cutoff_time_in_index** *(bool)* – If True, return a DataFrame with a MultiIndex where the second index is the cutoff time (first is instance id). DataFrame will be sorted by (time, instance_id).

- **training_window** *(Timedelta or str, optional)* – Window defining how much time before the cutoff time data can be used when calculating features. If None, all data before cutoff time is used. Defaults to None.

- **approximate** *(Timedelta or str)* – Frequency to group instances with similar cutoff times by for features with costly calculations. For example, if bucket is 24 hours, all instances with cutoff times on the same day will use the same calculation for expensive features.

- **verbose** *(bool, optional)* – Print progress info. The time granularity is per chunk.

- **chunk_size** *(int or float or None)* – maximum number of rows of output feature matrix to calculate at time. If passed an integer greater than 0, will try to use that many rows per chunk. If passed a float value between 0 and 1 sets the chunk size to that percentage of all rows. if None, and n_jobs > 1 it will be set to 1/n_jobs

- **n_jobs** *(int, optional)* – number of parallel processes to use when calculating feature matrix.

- **dask_kwargs** *(dict, optional)* – Dictionary of keyword arguments to be passed when creating the dask client and scheduler. Even if n_jobs is not set, using dask_kwargs will enable multiprocessing. Main parameters:

  - **cluster** *(str or dask.distributed.LocalCluster)*: cluster or address of cluster to send tasks to. If unspecified, a cluster will be created.

  - **diagnostics port** *(int)*: port number to use for web dashboard. If left unspecified, web interface will not be enabled.

Valid keyword arguments for LocalCluster will also be accepted.

- **save_progress** *(str, optional)* – path to save intermediate computational results.

- **progress_callback** *(callable)* – function to be called with incremental progress updates. Has the following parameters:

  - **update**: percentage change (float between 0 and 100) in progress since last call
  - **progress_percent**: percentage (float between 0 and 100) of total computation completed
  - **time_elapsed**: total time in seconds that has elapsed since start of call
### 3.18.8 Feature encoding

**encode_features**(feature_matrix, features[, ...])  
Encode categorical features

**featuretools.encode_features**

**featuretools.encode_features**(feature_matrix, features, top_n=10, include_unknown=True, to_encode=None, inplace=False, drop_first=False, verbose=False)

Encode categorical features

**Parameters**

- **features**(list[PrimitiveBase]) – Feature definitions in feature_matrix.
- **top_n**(int or dict[string -> int]) – Number of top values to include. If dict[string -> int] is used, key is feature name and value is the number of top values to include for that feature. If a feature’s name is not in dictionary, a default value of 10 is used.
- **include_unknown**(pd.DataFrame) – Add feature encoding an unknown class. defaults to True
- **to_encode**(list[str]) – List of feature names to encode. features not in this list are unencoded in the output matrix defaults to encode all necessary features.
- **inplace**(bool) – Encode feature_matrix in place. Defaults to False.
- **drop_first**(bool) – Whether to get k-1 dummies out of k categorical levels by removing the first level. defaults to False
- **verbose**(str) – Print progress info.

**Returns**

encoded feature_matrix, encoded features

**Return type** (pd.DataFrame, list)

**Example**

```python
In [1]: f1 = ft.Feature(es['log']['product_id'])
In [2]: f2 = ft.Feature(es['log']['purchased'])
In [3]: f3 = ft.Feature(es['log']['value'])
In [4]: features = [f1, f2, f3]
In [5]: ids = [0, 1, 2, 3, 4, 5]
In [6]: feature_matrix = ft.calculate_feature_matrix(features, es, ...:
...:

In [7]: fm_encoded, f_encoded = ft.encode_features(feature_matrix, ...
...:
```

(continues on next page)
```python
In [8]: f_encoded
Out[8]:
[<Feature: product_id = coke zero>,
 <Feature: product_id = car>,
 <Feature: product_id = toothpaste>,
 <Feature: product_id is unknown>,
 <Feature: purchased>,
 <Feature: value>]

In [9]: fm_encoded, f_encoded = ft.encode_features(feature_matrix,
......: features, top_n=2)
......:

In [10]: f_encoded
Out[10]:
[<Feature: product_id = coke zero>,
 <Feature: product_id = car>,
 <Feature: product_id is unknown>,
 <Feature: purchased>,
 <Feature: value>]

In [11]: fm_encoded, f_encoded = ft.encode_features(feature_matrix, features,
......: include_unknown=False)
......:

In [12]: f_encoded
Out[12]:
[<Feature: product_id = coke zero>,
 <Feature: product_id = car>,
 <Feature: product_id = toothpaste>,
 <Feature: purchased>,
 <Feature: value>]

In [13]: fm_encoded, f_encoded = ft.encode_features(feature_matrix, features,
......: to_encode=['purchased'])
......:

In [14]: f_encoded
Out[14]: [<Feature: product_id>, <Feature: purchased>, <Feature: value>]

In [15]: fm_encoded, f_encoded = ft.encode_features(feature_matrix, features,
......: drop_first=True)
......:

In [16]: f_encoded
Out[16]:
[<Feature: product_id = coke zero>,
 <Feature: product_id = car>,
 <Feature: product_id is unknown>,
 <Feature: purchased>,
 <Feature: value>]
```

### 3.18.9 Saving and Loading Features
**save_features**

Saves the features list as JSON to a specified filepath/S3 path, writes to an open file, or returns the serialized features as a JSON string. If no file provided, returns a string.

**Parameters**

- **features** (list[FeatureBase]) – List of Feature definitions.
- **location** (str or FileObject, optional) – The location of where to save the features list which must include the name of the file, or a writeable file handle to write to. If location is None, will return a JSON string of the serialized features. Default: None
- **profile_name** (str, bool) – The AWS profile specified to write to S3. Will default to None and search for AWS credentials. Set to False to use an anonymous profile.

**Note:** Features saved in one version of Featuretools are not guaranteed to work in another. After upgrading Featuretools, features may need to be generated again.

**Example**

```python
f1 = ft.Feature(es["log"]['product_id'])
f2 = ft.Feature(es["log"]['purchased'])
f3 = ft.Feature(es["log"]['value'])
features = [f1, f2, f3]
filepath = os.path.join('/Home/features/', 'list.json')
ft.save_features(features, filepath)
f = open(filepath, 'w')
ft.save_features(features, f)
features_str = ft.save_features(features)
```

**See also:**

`load_features()`

**load_features**

Loads the features from a filepath, S3 path, URL, an open file, or a JSON formatted string.

**Parameters**

- **features** (str or FileObject) – The location of where features has
- saved which this must include the name of the file, or a JSON formatted (been)
- or a readable file handle where the features have been saved (string)
  • profile_name (str, bool) – The AWS profile specified to write to S3. Will default to None and search for AWS credentials. Set to False to use an anonymous profile.

Returns Feature definitions list.
Return type features (list[FeatureBase])

Note: Features saved in one version of Featuretools or python are not guaranteed to work in another. After upgrading Featuretools or python, features may need to be generated again.

Example

```python
filepath = os.path.join('/Home/features/', 'list.json')
ft.load_features(filepath)

f = open(filepath, 'r')
ft.load_features(f)

feature_str = f.read()
ft.load_features(feature_str)
```

See also:

`save_features()`

### 3.18.10 EntitySet, Entity, Relationship, Variable Types

**Constructors**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EntitySet((id, entities, relationships))</code></td>
<td>Stores all actual data for a entityset</td>
</tr>
<tr>
<td><code>Entity(id, df, entityset[, variable_types, ...])</code></td>
<td>Represents an entity in a Entityset, and stores relevant metadata and data</td>
</tr>
<tr>
<td><code>Relationship(parent_variable, child_variable)</code></td>
<td>Class to represent an relationship between entities</td>
</tr>
</tbody>
</table>

**featuretools.EntitySet**

```python
class featuretools.EntitySet(id=None, entities=None, relationships=None)
Stores all actual data for a entityset
```

id

`entity_dict`

relationships

`time_type`

Properties: metadata
__init__(id=None, entities=None, relationships=None)

Creates EntitySet

Parameters

• **id**(str) – Unique identifier to associate with this instance

• **entities**(dict[str -> tuple(pd.DataFrame, str, str)]) – Dictionary of entities. Entries take the format {entity id -> (dataframe, id column, (time_column), (variable_types))}. Note that time_column and variable_types are optional.

• **relationships**(list[((str, str, str, str))]) – List of relationships between entities. List items are a tuple with the format (parent entity id, parent variable, child entity id, child variable).

Example

```python
dict = {
    "cards" : (card_df, "id"),
    "transactions" : (transactions_df, "id", "transaction_time")
}
relationships = [("cards", "id", "transactions", "card_id")]

ft.EntitySet("my-entity-set", dict, relationships)
```

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(id, entities, relationships)</td>
<td>Creates EntitySet</td>
</tr>
<tr>
<td>add_interesting_values(max_values, verbose)</td>
<td>Find interesting values for categorical variables, to be used to generate “where” clauses</td>
</tr>
<tr>
<td>add_last_time_indexes(updated_entities)</td>
<td>Calculates the last time index values for each entity (the last time an instance or children of that instance were observed).</td>
</tr>
<tr>
<td>add_relationship(relationship)</td>
<td>Add a new relationship between entities in the entityset</td>
</tr>
<tr>
<td>add_relationships(relationships)</td>
<td>Add multiple new relationships to a entityset</td>
</tr>
<tr>
<td>concat(other[, inplace])</td>
<td>Combine entityset with another to create a new entityset with the combined data of both entitysets.</td>
</tr>
<tr>
<td>entity_from_dataframe(entity_id, dataframe)</td>
<td>Load the data for a specified entity from a Pandas DataFrame.</td>
</tr>
<tr>
<td>find_backward_paths(start_entity_id,...)</td>
<td>Generator which yields all backward paths between a start and goal entity.</td>
</tr>
<tr>
<td>find_forward_paths(start_entity_id,...)</td>
<td>Generator which yields all forward paths between a start and goal entity.</td>
</tr>
<tr>
<td>get_backward_entities(entity_id[, deep])</td>
<td>Get entities that are in a backward relationship with entity</td>
</tr>
<tr>
<td>get_backward_relationships(entity_id)</td>
<td>get relationships where entity “entity_id” is the parent.</td>
</tr>
<tr>
<td>get_forward_entities(entity_id[, deep])</td>
<td>Get entities that are in a forward relationship with entity</td>
</tr>
</tbody>
</table>
Table 144 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_forward_relationships</td>
<td>Get relationships where entity “entity_id” is the child</td>
</tr>
<tr>
<td>has_unique_forward_path</td>
<td>Is the forward path from start to end unique?</td>
</tr>
<tr>
<td>normalize_entity</td>
<td>Create a new entity and relationship from unique values of an existing variable.</td>
</tr>
<tr>
<td>plot</td>
<td>Create a UML diagram-ish graph of the EntitySet.</td>
</tr>
<tr>
<td>to_csv</td>
<td>Write entityset to disk in the csv format, location specified by path.</td>
</tr>
<tr>
<td>to_dictionary</td>
<td></td>
</tr>
<tr>
<td>to_parquet</td>
<td>Write entityset to disk in the parquet format, location specified by path.</td>
</tr>
<tr>
<td>to_pickle</td>
<td>Write entityset in the pickle format, location specified by path.</td>
</tr>
</tbody>
</table>

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>entities</td>
<td>Returns the metadata for this EntitySet.</td>
</tr>
<tr>
<td>metadata</td>
<td></td>
</tr>
</tbody>
</table>

**featuretools.Entity**

```python
class featuretools.Entity(id, df, entityset, variable_types=None, index=None, time_index=None, secondary_time_index=None, last_time_index=None, already_sorted=False, make_index=False, verbose=False):
```

Represents an entity in an EntitySet, and stores relevant metadata and data

An Entity is analogous to a table in a relational database

See also:

Relationship, Variable, EntitySet

__init__ (id, df, entityset, variable_types=None, index=None, time_index=None, secondary_time_index=None, last_time_index=None, already_sorted=False, make_index=False, verbose=False)

Create Entity

**Parameters**

- **id (str)** – Id of Entity.
- **df (pd.DataFrame)** – Dataframe providing the data for the entity.
- **entityset (EntitySet)** – Entityset for this Entity.
- **variable_types (dict[str -> dict[str -> type]])** – An entity’s variable_types dict maps string variable ids to types (Variable) or (type, kwargs) to pass keyword arguments to the Variable.
- **index (str)** – Name of id column in the dataframe.
- **time_index (str)** – Name of time column in the dataframe.
- **secondary_time_index (dict[str -> str])** – Dictionary mapping columns in the dataframe to the time index column they are associated with.
• **last_time_index**(*pd.Series*) – Time index of the last event for each instance across all child entities.

• **make_index** (*bool, optional*) – If True, assume index does not exist as a column in dataframe, and create a new column of that name using integers the (0, len(dataframe)). Otherwise, assume index exists in dataframe.

**Methods**

```python
__init__(id, df, entityset[, ...]) Create Entity
add_interesting_values([max_values, verbose]) Find interesting values for categorical variables, to be used to
convert_variable_type(variable_id, new_type) Convert variable in dataframe to different type
delete_variables(variable_ids) Remove variables from entity’s dataframe and from self.variables
query_by_values(instance_vals[, ...]) Query instances that have variable with given value
set_index(variable_id[, unique])

param variable_id Name of an existing variable to set as index.
```

```python
set_secondary_time_index(secondary_time_index)
set_time_index(variable_id[, already_sorted])
update_data(df[, already_sorted, ...]) Update entity’s internal dataframe, optionally making sure data is sorted, reference indexes to other entities are consistent, and last_time_index are consistent.
```

**Attributes**

```python
df Dataframe providing the data for the entity.
last_time_index Time index of the last event for each instance across all child entities.
shape Shape of the entity’s dataframe
variable_types Dictionary mapping variable id’s to variable types
```

`featuretools.Relationship`

class `featuretools.Relationship`(*parent_variable, child_variable*)

Class to represent an relationship between entities

See also:

`EntitySet, Entity, Variable`

```python
__init__(parent_variable, child_variable) Create a relationship
```

**Parameters**

• **parent_variable** (*Discrete*) – Instance of variable in parent entity. Must be a Discrete Variable

• **child_variable** (*Discrete*) – Instance of variable in child entity. Must be a Discrete Variable
Methods

__init__(parent_variable, child_variable)  Create a relationship
from_dictionary(arguments, es)
to_dictionary()

Attributes

child_entity  Child entity object
child_name  The name of the child, relative to the parent.
child_variable  Instance of variable in child entity
parent_entity  Parent entity object
parent_name  The name of the parent, relative to the child.
parent_variable  Instance of variable in parent entity

EntitySet load and prepare data

EntitySet.entity_from_dataframe(entity_id, ...)  Load the data for a specified entity from a Pandas DataFrame.
EntitySet.add_relationship(relationship)  Add a new relationship between entities in the entityset
EntitySet.normalize_entity(base_entity_id, ...)  Create a new entity and relationship from unique values of an existing variable.
EntitySet.add_interesting_values(...)  Find interesting values for categorical variables, to be used to generate “where” clauses

featuretools.EntitySet.entity_from_dataframe

EntitySet.entity_from_dataframe(entity_id, dataframe, index=None, variable_types=None, make_index=False, time_index=None, secondary_time_index=None, already_sorted=False)  Load the data for a specified entity from a Pandas DataFrame.

Parameters

- entity_id (str) – Unique id to associate with this entity.
- dataframe (pandas.DataFrame) – Dataframe containing the data.
- index (str, optional) – Name of the variable used to index the entity. If None, take the first column.
- variable_types (dict[str -> Variable], optional) – Keys are of variable ids and values are variable types. Used to to initialize an entity’s store.
- make_index (bool, optional) – If True, assume index does not exist as a column in dataframe, and create a new column of that name using integers. Otherwise, assume index exists.
- time_index (str, optional) – Name of the variable containing time data. Type must be in variables.DateTime or be able to be cast to datetime (e.g. str, float, or numeric.)
- secondary_time_index (dict[str -> Variable]) – Name of variable containing time data to use a second time index for the entity.
• **already_sorted** (bool, optional) – If True, assumes that input dataframe is already sorted by time. Defaults to False.

**Notes**

Will infer variable types from Pandas dtype

**Example**

```
In [1]: import featuretools as ft
In [2]: import pandas as pd
In [3]: transactions_df = pd.DataFrame({"id": [1, 2, 3, 4, 5, 6], ...
... "session_id": [1, 2, 1, 3, 4, 5], ...
... "amount": [100.40, 20.63, 33.32, 13.12, 67.22, 1.00], ...
... "transaction_time": pd.date_range(start="10:00", periods=6, freq="10s"), ...
... "fraud": [True, False, True, False, True, True]})
...:
In [4]: es = ft.EntitySet("example")
In [5]: es.entity_from_dataframe(entity_id="transactions", ...
... index="id", ...
... time_index="transaction_time", ...
... dataframe=transactions_df)
...:
Out[5]:
Entityset: example
Entities:
transactions [Rows: 6, Columns: 5]
Relationships:
No relationships

In [6]: es["transactions"]

Entity: transactions
Variables:
  id (dtype: index)
  session_id (dtype: numeric)
  amount (dtype: numeric)
  transaction_time (dtype: datetime_time_index)
  fraud (dtype: boolean)
Shape:
  (Rows: 6, Columns: 5)

In [7]: es["transactions"]["df"]
```

(continues on next page)
2 2 2 20.63 2020-03-27 10:00:10 False
3 3 1 33.32 2020-03-27 10:00:20 True
4 4 3 13.12 2020-03-27 10:00:30 False
5 5 4 67.22 2020-03-27 10:00:40 True
6 6 5 1.00 2020-03-27 10:00:50 True

featuretools.EntitySet.add_relationship

EntitySet.add_relationship(relationship)

Add a new relationship between entities in the entityset

Parameters relationship (Relationship) – Instance of new relationship to be added.

featuretools.EntitySet.normalize_entity

EntitySet.normalize_entity(base_entity_id, new_entity_id, index, additional_variables=None, copy_variables=None, make_time_index=None, make_secondary_time_index=None, new_entity_time_index=None, new_entity_secondary_time_index=None)

Create a new entity and relationship from unique values of an existing variable.

Parameters

- base_entity_id (str) – Entity id from which to split.
- new_entity_id (str) – Id of the new entity.
- index (str) – Variable in old entity that will become index of new entity. Relationship will be created across this variable.
- additional_variables (list[str]) – List of variable ids to remove from base_entity and move to new entity.
- copy_variables (list[str]) – List of variable ids to copy from old entity and move to new entity.
- make_time_index (bool or str, optional) – Create time index for new entity based on time index in base_entity, optionally specifying which variable in base_entity to use for time_index. If specified as True without a specific variable, uses the primary time index. Defaults to True if base entity has a time index.
- make_secondary_time_index (dict[str -> list[str]], optional) – Create a secondary time index from key. Values of dictionary are the variables to associate with the secondary time index. Only one secondary time index is allowed. If None, only associate the time index.
- new_entity_time_index (str, optional) – Rename new entity time index.
- new_entity_secondary_time_index (str, optional) – Rename new entity secondary time index.

featuretools.EntitySet.add_interesting_values

EntitySet.add_interesting_values(max_values=5, verbose=False)

Find interesting values for categorical variables, to be used to generate “where” clauses.
Parameters

- `max_values (int)` – Maximum number of values per variable to add.
- `verbose (bool)` – If True, print summary of interesting values found.

Returns  None

**EntitySet serialization**

`read_entityset(path[, profile_name])`  
Read entityset from disk, S3 path, or URL.

**featuretools.read_entityset**

`featuretools.read_entityset (path, profile_name=None, **kwargs)`  
Read entityset from disk, S3 path, or URL.

Parameters

- `path (str)` – Directory on disk, S3 path, or URL to read `data_description.json`.
- `profile_name (str, bool)` – The AWS profile specified to write to S3. Will default to None and search for AWS credentials. Set to False to use an anonymous profile.
- `kwargs (keywords)` – Additional keyword arguments to pass as keyword arguments to the underlying deserialization method.

**EntitySet.to_csv**

`EntitySet.to_csv(path, sep=',', encoding='utf-8', engine='python', compression=None, profile_name=None)`  
Write entityset to disk in the csv format, location specified by `path`. Path could be a local path or a S3 path. If writing to S3 a tar archive of files will be written.

Parameters

- `path (str)` – Location on disk to write to (will be created as a directory)
- `sep (str)` – String of length 1. Field delimiter for the output file.
- `encoding (str)` – A string representing the encoding to use in the output file, defaults to ‘utf-8’.
- `engine (str)` – Name of the engine to use. Possible values are: {'c', 'python'}.
- `compression (str)` – Name of the compression to use. Possible values are: {'gzip', 'bz2', 'zip', 'xz', None}.
- `profile_name (str)` – Name of AWS profile to use, False to use an anonymous profile, or None.
featuretools.entityset.EntitySet.to_pickle

EntitySet.to_pickle(path, compression=None, profile_name=None)

Write entityset in the pickle format, location specified by path. Path could be a local path or a S3 path. If writing to S3 a tar archive of files will be written.

Parameters

- **path** (str) – location on disk to write to (will be created as a directory)
- **compression** (str) – Name of the compression to use. Possible values are: {‘gzip’, ‘bz2’, ‘zip’, ‘xz’, None}.
- **profile_name** (str) – Name of AWS profile to use, False to use an anonymous profile, or None.

featuretools.entityset.EntitySet.to_parquet

EntitySet.to_parquet(path, engine='auto', compression=None, profile_name=None)

Write entityset to disk in the parquet format, location specified by path. Path could be a local path or a S3 path. If writing to S3 a tar archive of files will be written.

Parameters

- **path** (str) – location on disk to write to (will be created as a directory)
- **engine** (str) – Name of the engine to use. Possible values are: {‘auto’, ‘pyarrow’, ‘fastparquet’}.
- **compression** (str) – Name of the compression to use. Possible values are: {‘snappy’, ‘gzip’, ‘brotli’, None}.
- **profile_name** (str) – Name of AWS profile to use, False to use an anonymous profile, or None.

EntitySet query methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntitySet.<strong>getitem</strong>(entity_id)</td>
<td>Get entity instance from entityset</td>
</tr>
<tr>
<td>EntitySet.find_backward_paths(...)</td>
<td>Generator which yields all backward paths between a start and goal entity.</td>
</tr>
<tr>
<td>EntitySet.find_forward_paths(...)</td>
<td>Generator which yields all forward paths between a start and goal entity.</td>
</tr>
<tr>
<td>EntitySet.get_forward_entities(entity_id[, deep])</td>
<td>Get entities that are in a forward relationship with entity</td>
</tr>
<tr>
<td>EntitySet.get_backward_entities(entity_id[, ...])</td>
<td>Get entities that are in a backward relationship with entity</td>
</tr>
</tbody>
</table>

featuretools.entityset.EntitySet.__getitem__

EntitySet.__getitem__(entity_id)

Get entity instance from entityset

Parameters

- **entity_id** (str) – Id of entity.

Returns
Instance of entity. None if entity doesn’t exist.

Return type \textit{Entity}

\texttt{featuretools.entityset.EntitySet.find_backward_paths}

\texttt{EntitySet.find_backward_paths} \texttt{(start_entity_id, goal_entity_id)}

Generator which yields all backward paths between a start and goal entity. Does not include paths which contain cycles.

Parameters

- \texttt{start_entity_id} (\texttt{str}) – Id of entity to start the search from.
- \texttt{goal_entity_id} (\texttt{str}) – Id of entity to find backward path to.

See also:

\texttt{BaseEntitySet.find_forward_paths()}

\texttt{featuretools.entityset.EntitySet.find_forward_paths}

\texttt{EntitySet.find_forward_paths} \texttt{(start_entity_id, goal_entity_id)}

Generator which yields all forward paths between a start and goal entity. Does not include paths which contain cycles.

Parameters

- \texttt{start_entity_id} (\texttt{str}) – id of entity to start the search from
- \texttt{goal_entity_id} (\texttt{str}) – id of entity to find forward path to

See also:

\texttt{BaseEntitySet.find_backward_paths()}

\texttt{featuretools.entityset.EntitySet.get_forward_entities}

\texttt{EntitySet.get_forward_entities} \texttt{(entity_id, deep=False)}

Get entities that are in a forward relationship with entity

Parameters

- \texttt{entity_id} (\texttt{str}) – Id entity of entity to search from.
- \texttt{deep} (\texttt{bool}) – if True, recursively find forward entities.

Yields a tuple of (descendent_id, path from entity_id to descendent).

\texttt{featuretools.entityset.EntitySet.get_backward_entities}

\texttt{EntitySet.get_backward_entities} \texttt{(entity_id, deep=False)}

Get entities that are in a backward relationship with entity

Parameters

- \texttt{entity_id} (\texttt{str}) – Id entity of entity to search from.
- \texttt{deep} (\texttt{bool}) – if True, recursively find backward entities.
Yields a tuple of (descendent_id, path from entity_id to descendant).

**EntitySet visualization**

```
EntitySet.plot() Create a UML diagram-ish graph of the EntitySet.
```

**featuretools.entityset.EntitySet.plot**

```
EntitySet.plot(to_file=None)
Create a UML diagram-ish graph of the EntitySet.
```

**Parameters**

- **to_file** (str, optional) – Path to where the plot should be saved. If set to None (as by default), the plot will not be saved.

**Returns**

- **Graph object that can directly be displayed in** Jupyter notebooks.

**Return type** graphviz.Digraph

**Entity methods**

```
Entity.convert_variable_type(variable_id, ...) Convert variable in dataframe to different type

Entity.add_interesting_values(max_values, ...) Find interesting values for categorical variables, to be used to
```

**featuretools.entityset.Entity.convert_variable_type**

```
Entity.convert_variable_type(variable_id, new_type, convert_data=True, **kwargs)
Convert variable in dataframe to different type
```

**Parameters**

- **variable_id** (str) – Id of variable to convert.
- **new_type** (subclass of Variable) – Type of variable to convert to.
- **entityset** (BaseEntitySet) – EntitySet associated with this entity.
- **convert_data** (bool) – If True, convert underlying data in the EntitySet.

**Raises** RuntimeError – Raises if it cannot convert the underlying data

**Examples**

```python
>>> from featuretools.tests.testing_utils import make_ecommerce_entityset
>>> es = make_ecommerce_entityset()
>>> es["customers"].convert_variable_type("engagement_level", vtypes.Categorical)
```

**featuretools.entityset.Entity.add_interesting_values**

```
Entity.add_interesting_values(max_values=5, verbose=False)
```

### 3.18. API Reference
Find interesting values for categorical variables, to be used to generate “where” clauses

Parameters

- `max_values (int)` – Maximum number of values per variable to add.
- `verbose (bool)` – If True, print summary of interesting values found.

Returns None

Relationship attributes

<table>
<thead>
<tr>
<th>Relationship attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Relationship.parent_variable</code></td>
<td>Instance of variable in parent entity</td>
</tr>
<tr>
<td><code>Relationship.child_variable</code></td>
<td>Instance of variable in child entity</td>
</tr>
<tr>
<td><code>Relationship.parent_entity</code></td>
<td>Parent entity object</td>
</tr>
<tr>
<td><code>Relationship.child_entity</code></td>
<td>Child entity object</td>
</tr>
</tbody>
</table>

```python
featuretools.entityset.Relationship.parent_variable

Relationship.parent_variable
Instance of variable in parent entity
```

```python
featuretools.entityset.Relationship.child_variable

Relationship.child_variable
Instance of variable in child entity
```

```python
featuretools.entityset.Relationship.parent_entity

Relationship.parent_entity
Parent entity object
```

```python
featuretools.entityset.Relationship.child_entity

Relationship.child_entity
Child entity object
```

Variable types

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index(id, entity[, name])</td>
<td>Represents variables that uniquely identify an instance of an entity</td>
</tr>
<tr>
<td>Id(id, entity[, name, categories])</td>
<td>Represents variables that identify another entity</td>
</tr>
<tr>
<td>TimeIndex(id, entity[, name])</td>
<td>Represents time index of entity</td>
</tr>
<tr>
<td>DatetimeTimeIndex(id, entity[, name, format])</td>
<td>Represents time index of entity that is a datetime</td>
</tr>
<tr>
<td>NumericTimeIndex(id, entity[, name, range, ...])</td>
<td>Represents time index of entity that is numeric</td>
</tr>
<tr>
<td>Datetime(id, entity[, name, format])</td>
<td>Represents variables that are points in time</td>
</tr>
<tr>
<td>Numeric(id, entity[, name, range, ...])</td>
<td>Represents variables that contain numeric values</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Categorical</strong> (id, entity[, name, categories])</td>
<td>Represents variables that can take an unordered discrete values</td>
</tr>
<tr>
<td><strong>Ordinal</strong> (id, entity[, name])</td>
<td>Represents variables that take on an ordered discrete value</td>
</tr>
<tr>
<td><strong>Boolean</strong> (id, entity[, name])</td>
<td>Represents variables that take on one of two values</td>
</tr>
<tr>
<td><strong>Text</strong> (id, entity[, name])</td>
<td>Represents variables that are arbitrary strings</td>
</tr>
<tr>
<td><strong>LatLong</strong> (id, entity[, name])</td>
<td>Represents an ordered pair (Latitude, Longitude) To make a latlong in a dataframe do <code>data['latlong'] = data[['latitude', 'longitude']].apply(tuple, axis=1)</code></td>
</tr>
<tr>
<td><strong>ZIPCode</strong> (id, entity[, name, categories])</td>
<td>Represents a postal address in the United States.</td>
</tr>
<tr>
<td><strong>IPAddress</strong> (id, entity[, name])</td>
<td>Represents a computer network address.</td>
</tr>
<tr>
<td><strong>FullName</strong> (id, entity[, name])</td>
<td>Represents a person’s full name.</td>
</tr>
<tr>
<td><strong>EmailAddress</strong> (id, entity[, name])</td>
<td>Represents an email box to which email message are sent.</td>
</tr>
<tr>
<td><strong>URL</strong> (id, entity[, name])</td>
<td>Represents a valid web url (with or without http/www)</td>
</tr>
<tr>
<td><strong>PhoneNumber</strong> (id, entity[, name])</td>
<td>Represents any valid phone number.</td>
</tr>
<tr>
<td><strong>DateOfBirth</strong> (id, entity[, name, format])</td>
<td>Represents a date of birth as a datetime</td>
</tr>
<tr>
<td><strong>CountryCode</strong> (id, entity[, name, categories])</td>
<td>Represents an ISO-3166 standard country code.</td>
</tr>
<tr>
<td><strong>SubRegionCode</strong> (id, entity[, name, categories])</td>
<td>Represents an ISO-3166 standard sub-region code.</td>
</tr>
<tr>
<td><strong>FilePath</strong> (id, entity[, name])</td>
<td>Represents a valid filepath, absolute or relative</td>
</tr>
</tbody>
</table>

**featuretools.variable_types.Index**

```python
class featuretools.variable_types.Index (id, entity, name=None)

    Represents variables that uniquely identify an instance of an entity

    count

    Type  int

    __init__ (id, entity, name=None)
    Initialize self. See help(type(self)) for accurate signature.

    Methods

    __init__ (id, entity[, name])
    Initialize self.

    create_from (variable)
    Create new variable this type from existing

    to_data_description ()

    Attributes

    dtype
    entityset
    interesting_values
    name
    series
    type_string
```

### 3.18. API Reference
featuretools.variable_types.Id

```python
class featuretools.variable_types.Id(id, entity, name=None, categories=None):
    Represents variables that identify another entity
    __init__(id, entity, name=None, categories=None)
    Initialize self. See help(type(self)) for accurate signature.
```

Methods

```
__init__(id, entity[, name, categories]) Initialize self.
create_from(variable) Create new variable this type from existing
to_data_description()
```

Attributes

```
dtype
entityset
interesting_values
name
series
type_string
```

featuretools.variable_types.TimeIndex

```python
class featuretools.variable_types.TimeIndex(id, entity, name=None):
    Represents time index of entity
    __init__(id, entity, name=None)
    Initialize self. See help(type(self)) for accurate signature.
```

Methods

```
__init__(id, entity[, name]) Initialize self.
create_from(variable) Create new variable this type from existing
to_data_description()
```

Attributes

```
dtype
entityset
interesting_values
name
series
type_string
```
featuretools.variable_types.DatetimeTimeIndex

class featuretools.variable_types.DatetimeTimeIndex (id, entity, name=None, format=None)

Represents time index of entity that is a datetime

__init__ (id, entity, name=None, format=None)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__ (id, entity[, name, format]) Initialize self.
create_from (variable) Create new variable this type from existing
to_data_description ()

Attributes

dtype
datetime
interesting_values
name
series
type_string

featuretools.variable_types.NumericTimeIndex

class featuretools.variable_types.NumericTimeIndex (id, entity, name=None, range=None,
start_inclusive=True, end_inclusive=False)

Represents time index of entity that is numeric

__init__ (id, entity, name=None, range=None, start_inclusive=True, end_inclusive=False)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__ (id, entity[, name, range, ...]) Initialize self.
create_from (variable) Create new variable this type from existing
to_data_description ()

Attributes

dtype
datetime
interesting_values
name
series
featuretools.variable_types.Datetime

class featuretools.variable_types.Datetime(id, entity, name=None, format=None)

Represents variables that are points in time

Parameters

- format (str) – Python datetime format string documented here.

__init__ (id, entity, name=None, format=None)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__ (id, entity[, name, format]) Initialize self.
create_from (variable) Create new variable this type from existing
to_data_description ()

Attributes

- dtype
- entityset
- interesting_values
- name
- series
- type_string

featuretools.variable_types.Numeric

class featuretools.variable_types.Numeric(id, entity, name=None, range=None, start_inclusive=True, end_inclusive=False)

Represents variables that contain numeric values

Parameters

- range (list, optional) – List of start and end. Can use inf and -inf to represent
  infinity. Unconstrained if not specified.
- start_inclusive (bool, optional) – Whether or not range includes the start value.
- end_inclusive (bool, optional) – Whether or not range includes the end value

max

Type: float

min

Type: float

std

Type: float

mean
Type  float

__init__ (id, entity, name=None, range=None, start_inclusive=True, end_inclusive=False)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__ (id, entity[, name, range, ...]) Initialize self.
create_from (variable) Create new variable this type from existing
to_data_description ()

Attributes
dtype
entityset
interesting_values
name
series
type_string

featuretools.variable_types.Categorical

class featuretools.variable_types.Categorical (id, entity, name=None, categories=None)
Represents variables that can take an unordered discrete values

Parameters  categories (list) – List of categories. If left blank, inferred from data.

__init__ (id, entity, name=None, categories=None)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__ (id, entity[, name, categories]) Initialize self.
create_from (variable) Create new variable this type from existing
to_data_description ()

Attributes
dtype
entityset
interesting_values
name
series
type_string
featuretools.variable_types.Ordinal

```py
class featuretools.variable_types.Ordinal(id, entity, name=None)
    Represents variables that take on an ordered discrete value
    __init__(id, entity, name=None)
        Initialize self. See help(type(self)) for accurate signature.
```

Methods

- `__init__(id, entity[, name])` Initialize self.
- `create_from(variable)` Create new variable this type from existing
- `to_data_description()`

Attributes

- `dtype`
- `entityset`
- `interesting_values`
- `name`
- `series`
- `type_string`

featuretools.variable_types.Boolean

```py
class featuretools.variable_types.Boolean(id, entity, name=None, true_values=None, false_values=None)
    Represents variables that take on one of two values
    Parameters
    • `true_values` (list) – List of valued true values. Defaults to [1, True, “true”, “True”, “yes”, “t”, “T”]
    • `false_values` (list) – List of valued false values. Defaults to [0, False, “false”, “False”, “no”, “f”, “F”]
    __init__(id, entity, name=None, true_values=None, false_values=None)
        Initialize self. See help(type(self)) for accurate signature.
```

Methods

- `__init__(id, entity[, name, true_values, . . . ])` Initialize self.
- `create_from(variable)` Create new variable this type from existing
- `to_data_description()`

Attributes

- `dtype`
Table 177 – continued from previous page

<table>
<thead>
<tr>
<th>entityset</th>
<th>interesting_values</th>
<th>name</th>
<th>series</th>
<th>type_string</th>
</tr>
</thead>
</table>

featuretools.variable_types.Text

class featuretools.variable_types.Text(id, entity, name=None)

  Represents variables that are arbitrary strings

  __init__ (id, entity, name=None)

    Initialize self. See help(type(self)) for accurate signature.

  Attributes

    dtype
    entityset
    interesting_values
    name
    series
    type_string

featuretools.variable_types.LatLong

class featuretools.variable_types.LatLong(id, entity, name=None)

  Represents an ordered pair (Latitude, Longitude) To make a latlong in a dataframe do data[‘latlong’] = data[['latitude', 'longitude']].apply(tuple, axis=1)

  __init__ (id, entity, name=None)

    Initialize self. See help(type(self)) for accurate signature.

  Attributes

    dtype
    entityset
    interesting_values
    name
    series
    type_string
featuretools.variable_types.ZIPCode

class featuretools.variable_types.ZIPCode(id, entity, name=None, categories=None)
    Represents a postal address in the United States. Consists of a series of digits which are casts as string. Five digit and 9 digit zipcodes are supported.

    __init__(id, entity, name=None, categories=None)
    Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(id, entity[, name, categories]) Initialize self.
create_from(variable) Create new variable this type from existing
to_data_description()
Featuretools Documentation, Release 0.13.4

featuretools.variable_types.FullName

class featuretools.variable_types.FullName(id, entity, name=None)
    Represents a person’s full name. May consist of a first name, last name, and a title.
    __init__(id, entity, name=None)
        Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(id, entity[, name])  Initialize self.
create_from(variable)       Create new variable this type from existing

to_data_description()

Attributes

dtype
data
data

featuretools.variable_types.EmailAddress

class featuretools.variable_types.EmailAddress(id, entity, name=None)
    Represents an email box to which email message are sent. Consists of a local-part, an @ symbol, and a domain.
    __init__(id, entity, name=None)
        Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(id, entity[, name])  Initialize self.
create_from(variable)       Create new variable this type from existing

to_data_description()

Attributes
featuretools.variable_types.URL

```python
class featuretools.variable_types.URL(id, entity, name=None)
    Represents a valid web url (with or without http/www)
    __init__(id, entity, name=None)
        Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(id, entity[, name])
create_from(variable)
to_data_description()
```

Attributes

dtype
entityset
interesting_values
name
series
type_string

featuretools.variable_types.PhoneNumber

```python
class featuretools.variable_types.PhoneNumber(id, entity, name=None)
    Represents any valid phone number. Can be with/without parenthesis. Can be with/without area/country codes.
    __init__(id, entity, name=None)
        Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(id, entity[, name])
create_from(variable)
to_data_description()
```

Attributes
featuretools.variable_types.DateOfBirth

class featuretools.variable_types.DateOfBirth(id, entity, name=None, format=None)

Represents a date of birth as a datetime

__init__(id, entity, name=None, format=None)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(id, entity[, name, format]) Initialize self.
create_from(variable) Create new variable this type from existing
to_data_description()

Attributes

dtype
dataset
interesting_values
name
series
type_string

featuretools.variable_types.CountryCode

class featuretools.variable_types.CountryCode(id, entity, name=None, categories=None)

Represents an ISO-3166 standard country code. ISO 3166-1 (countries) are supported. These codes should be
in the Alpha-2 format. e.g. United States of America = US

__init__(id, entity, name=None, categories=None)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(id, entity[, name, categories]) Initialize self.
create_from(variable) Create new variable this type from existing
to_data_description()

Attributes
featuretools.variable_types.SubRegionCode

class featuretools.variable_types.SubRegionCode(id, entity, name=None, categories=None)

Represents an ISO-3166 standard sub-region code. ISO 3166-2 codes (sub-regions are supported. These codes should be in the Alpha-2 format. e.g. United States of America, Arizona = US-AZ

__init__(id, entity, name=None, categories=None)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(id, entity[, name, categories]) Initialize self.
create_from(variable) Create new variable this type from existing
to_data_description()

Attributes

dtype
datatype
interesting_values
name
series
type_string

featuretools.variable_types.FilePath

class featuretools.variable_types.FilePath(id, entity, name=None)

Represents a valid filepath, absolute or relative

__init__(id, entity, name=None)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(id, entity[, name]) Initialize self.
create_from(variable) Create new variable this type from existing
to_data_description()
**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtype</td>
</tr>
<tr>
<td>entityset</td>
</tr>
<tr>
<td>interesting_values</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>series</td>
</tr>
<tr>
<td>type_string</td>
</tr>
</tbody>
</table>

**Feature Selection**

```python
featuretools.selection.remove_low_information_features
```

Select features that have at least 2 unique values and that are not all null

**Parameters**

- `feature_matrix` (pd.DataFrame) – DataFrame whose columns are feature names and rows are instances
- `features` (list[featuretools.FeatureBase] or list[str], optional) – List of features to select

**Returns** (feature_matrix, features)

### 3.19 Changelog

**v0.13.4 Mar 27, 2020**

**Warning:** The next non-bugfix release of Featuretools will not support Python 3.5

- **Fixes**
  - Fix `ft.show_info()` not displaying in Jupyter notebooks (GH#863)
- **Changes**
  - Added Plugin Warnings at Entry Point (GH#850, GH#869)
- **Documentation Changes**
  - Add links to primitives.featurelabs.com (GH#860)
  - Add source code links to API reference (GH#862)
  - Update links for testing Dask/Spark integrations (GH#867)
  - Update release documentation for featuretools (GH#868)
- **Testing Changes**
– Miscellaneous changes (GH#861)

Thanks to the following people for contributing to this release: @frances-h, @FreshLeaf8865, @jeff-hernandez, @rwedge, @thelhomebrewnerd

v0.13.3 Feb 28, 2020

• Fixes
  – Fix a connection closed error when using n_jobs (GH#853)

• Changes
  – Pin msgpack dependency for Python 3.5; remove dataframe from Dask dependency (GH#851)

• Documentation Changes
  – Update link to help documentation page in Github issue template (GH#855)

Thanks to the following people for contributing to this release: @frances-h, @rwedge

v0.13.2 Jan 31, 2020

• Enhancements
  – Support for Pandas 1.0.0 (GH#844)

• Changes
  – Remove dependency on s3fs library for anonymous downloads from S3 (GH#825)

• Testing Changes
  – Added GitHub Action to automatically run performance tests (GH#840)

Thanks to the following people for contributing to this release: @frances-h, @rwedge

v0.13.1 Dec 28, 2019

• Fixes
  – Raise error when given wrong input for ignore_variables (GH#826)
  – Fix multi-output features not created when there is no child data (GH#834)
  – Removing type casting in Equals and NotEquals primitives (GH#504)

• Changes
  – Replace pd.timedelta time units that were deprecated (GH#822)
  – Move sklearn wrapper to separate library (GH#835, GH#837)

• Testing Changes
  – Run unit tests in windows environment (GH#790)
  – Update boto3 version requirement for tests (GH#838)

Thanks to the following people for contributing to this release: @jeffzi, @kmax12, @rwedge, @systemshift

v0.13.0 Nov 30, 2019

• Enhancements
  – Added GitHub Action to auto upload releases to PyPI (GH#816)

• Fixes
  – Fix issue where some primitive options would not be applied (GH#807)
– Fix issue with converting to pickle or parquet after adding interesting features (GH#798, GH#823)
– Diff primitive now calculates using all available data (GH#824)
– Prevent DFS from creating Identity Features of globally ignored variables (GH#819)

• Changes
  – Remove python 2.7 support from serialize.py (GH#812)
  – Make smart_open, boto3, and s3fs optional dependencies (GH#827)

• Documentation Changes
  – remove python 2.7 support and add 3.7 in install.rst (GH#805)
  – Fix import error in docs (GH#803)
  – Fix release title formatting in changelog (GH#806)

• Testing Changes
  – Use multiple CPUS to run tests on CI (GH#811)
  – Refactor test entityset creation to avoid saving to disk (GH#813, GH#821)
  – Remove get_values() from test_es.py to remove warnings (GH#820)

Thanks to the following people for contributing to this release: @frances-h, @jeff-hernandez, @rwedge, @systemshift

Breaking Changes
• The libraries used for downloading or uploading from S3 or URLs are now optional and will no longer be installed by default. To use this functionality they will need to be installed separately.
• The fix to how the Diff primitive is calculated may slow down the overall calculation time of feature lists that use this primitive.

v0.12.0 Oct 31, 2019

• Enhancements
  – Added First primitive (GH#770)
  – Added Entropy aggregation primitive (GH#779)
  – Allow custom naming for multi-output primitives (GH#780)

• Fixes
  – Prevents user from removing base entity time index using additional_variables (GH#768)
  – Fixes error when a multioutput primitive was supplied to dfs as a groupby trans primitive (GH#786)

• Changes
  – Drop Python 2 support (GH#759)
  – Add unit parameter to AvgTimeBetween (GH#771)
  – Require Pandas 0.24.1 or higher (GH#787)

• Documentation Changes
  – Update featuretools slack link (GH#765)
  – Set up repo to use Read the Docs (GH#776)
– Add First primitive to API reference docs (GH#782)

**Testing Changes**

– CircleCI fixes (GH#774)
– Disable PIP progress bars (GH#775)

Thanks to the following people for contributing to this release: @ablacke-ayx, @BoopBoopBeepBoop, @jeffzi, @kmax12, @r wedge, @thehomebrewnerd, @twdobson

v0.11.0 Sep 30, 2019

**Warning:** The next non-bugfix release of Featuretools will not support Python 2

**Enhancements**

– Improve how files are copied and written (GH#721)
– Add number of rows to graph in entityset.plot (GH#727)
– Added support for pandas DateOffsets in DFS and Timedelta (GH#732)
– Enable feature-specific top_n value using a dictionary in encode_features (GH#735)
– Added progress_callback parameter to dfs() and calculate_feature_matrix() (GH#739, GH#745)
– Enable specifying primitives on a per column or per entity basis (GH#748)

**Fixes**

– Fixed entity set deserialization (GH#720)
– Added error message when DateTimeIndex is a variable but not set as the time_index (GH#723)
– Fixed CumCount and other group-by transform primitives that take ID as input (GH#733, GH#754)
– Fix progress bar undercounting (GH#743)
– Updated training_window error assertion to only check against observations (GH#728)
– Don’t delete the whole destination folder while saving entityset (GH#717)

**Changes**

– Raise warning and not error on schema version mismatch (GH#718)
– Change feature calculation to return in order of instance ids provided (GH#676)
– Removed time remaining from displayed progress bar in dfs() and calculate_feature_matrix() (GH#739)
– Raise warning in normalize_entity() when time_index of base_entity has an invalid type (GH#749)
– Remove toolz as a direct dependency (GH#755)
– Allow boolean variable types to be used in the Multiply primitive (GH#756)

**Documentation Changes**

– Updated URL for Compose (GH#716)

**Testing Changes**
– Update dependencies (GH#738, GH#741, GH#747)

Thanks to the following people for contributing to this release: @angela97lin, @chidauri, @christopherbunn, @frances-h, @jeff-hernandez, @kmax12, @MarcoGorelli, @rwedge, @thehomebrewnerd

**Breaking Changes**

- Feature calculations will return in the order of instance ids provided instead of the order of time points instances are calculated at.

v0.10.1 Aug 25, 2019

- **Fixes**
  - Fix serialized LatLong data being loaded as strings (GH#712)

- **Documentation Changes**
  - Fixed FAQ cell output (GH#710)

Thanks to the following people for contributing to this release: @gsheni, @rwedge

v0.10.0 Aug 19, 2019

| Warning: The next non-bugfix release of Featuretools will not support Python 2 |

- **Enhancements**
  - Give more frequent progress bar updates and update chunk size behavior (GH#631, GH#696)
  - Added drop_first as param in encode_features (GH#647)
  - Added support for stacking multi-output primitives (GH#679)
  - Generate transform features of direct features (GH#623)
  - Added serializing and deserializing from S3 and deserializing from URLs (GH#685)
  - Added nlp_primitives as an add-on library (GH#704)
  - Added AutoNormalize to Featuretools plugins (GH#699)
  - Added functionality for relative units (month/year) in Timedelta (GH#692)
  - Added categorical-encoding as an add-on library (GH#700)

- **Fixes**
  - Fix performance regression in DFS (GH#637)
  - Fix deserialization of feature relationship path (GH#665)
  - Set index after adding ancestor relationship variables (GH#668)
  - Fix user-supplied variable_types modification in Entity init (GH#675)
  - Don’t calculate dependencies of unnecessary features (GH#667)
  - Prevent normalize entity’s new entity having same index as base entity (GH#681)
  - Update variable type inference to better check for string values (GH#683)

- **Changes**
  - Moved dask, distributed imports (GH#634)

- **Documentation Changes**
– Miscellaneous changes (GH#641, GH#658)
– Modified doc_string of top_n in encoding (GH#648)
– Hyperlinked ComposeML (GH#653)
– Added FAQ (GH#620, GH#677)
– Fixed FAQ question with multiple question marks (GH#673)

• Testing Changes
  – Add master, and release tests for premium primitives (GH#660, GH#669)
  – Miscellaneous changes (GH#672, GH#674)

Thanks to the following people for contributing to this release: @alexjwang, @allisonportis, @ayushpatidar, @CJStadler, @ctduffy, @gsheni, @jeff-herandez, @jeremyliweishih, @kmax12, @rwedge, @zhxt95,

v0.9.1 July 3, 2019

• Enhancements
  – Speedup groupby transform calculations (GH#609)
  – Generate features along all paths when there are multiple paths between entities (GH#600, GH#608)

• Fixes
  – Select columns of dataframe using a list (GH#615)
  – Change type of features calculated on Index features to Categorical (GH#602)
  – Filter dataframes through forward relationships (GH#625)
  – Specify Dask version in requirements for python 2 (GH#627)
  – Keep dataframe sorted by time during feature calculation (GH#626)
  – Fix bug in encode_features that created duplicate columns of features with multiple outputs (GH#622)

• Changes
  – Remove unused variance_selection.py file (GH#613)
  – Remove Timedelta data param (GH#619)
  – Remove DaysSince primitive (GH#628)

• Documentation Changes
  – Add installation instructions for add-on libraries (GH#617)
  – Clarification of Multi Output Feature Creation (GH#638)
  – Miscellaneous changes (GH#632, GH#639)

• Testing Changes
  – Miscellaneous changes (GH#595, GH#612)

Thanks to the following people for contributing to this release: @CJStadler, @kmax12, @rwedge, @gsheni, @kleidal, @ctduffy

v0.9.0 June 19, 2019

• Enhancements
– Add unit parameter to timesince primitives (GH#558)
– Add ability to install optional add on libraries (GH#551)
– Load and save features from open files and strings (GH#566)
– Support custom variable types (GH#571)
– Support entitysets which have multiple paths between two entities (GH#572, GH#544)
– Added show_info function, more output information added to CLI featuretools info (GH#525)

• Fixes
– Normalize_entity specifies error when ‘make_time_index’ is an invalid string (GH#550)
– Schema version added for entityset serialization (GH#586)
– Renamed features have names correctly serialized (GH#585)
– Improved error message for index/time_index being the same column in normalize_entity and entity_from_dataframe (GH#583)
– Removed all mentions of allow_where (GH#587, GH#588)
– Removed unused variable in normalize entity (GH#589)
– Change time since return type to numeric (GH#606)

• Changes
– Refactor get_pandas_data_slice to take single entity (GH#547)
– Updates TimeSincePrevious and Diff Primitives (GH#561)
– Remove unnecessary time_last variable (GH#546)

• Documentation Changes
– Add Featuretools Enterprise to documentation (GH#563)
– Miscellaneous changes (GH#552, GH#573, GH#577, GH#599)

• Testing Changes
– Miscellaneous changes (GH#559, GH#569, GH#570, GH#574, GH#584, GH#590)

Thanks to the following people for contributing to this release: @alexjwang, @allisonportis, @CJStadler, @ct-duffy, @gsheni, @kmax12, @rwedge

v0.8.0 May 17, 2019
– Rename NUnique to NumUnique (GH#510)
– Serialize features as JSON (GH#532)
– Drop all variables at once in normalize_entity (GH#533)
– Remove unnecessary sorting from normalize_entity (GH#535)
– Features cache their names (GH#536)
– Only calculate features for instances before cutoff (GH#523)
– Remove all relative imports (GH#530)
– Added FullName Variable Type (GH#506)
– Add error message when target entity does not exist (GH#520)
– New demo links (GH#542)
• Remove duplicate features check in DFS (GH#538)
• featuretools.primitives entry point expects list of primitive classes (GH#529)
• Update ALL_VARIABLE_TYPES list (GH#526)
• More Informative N Jobs Prints and Warnings (GH#511)
• Update sklearn version requirements (GH#541)
• Update Makefile (GH#519)
• Remove unused parameter in Entity._handle_time (GH#524)
• Remove build_ext code from setup.py (GH#513)
• Documentation updates (GH#512, GH#514, GH#515, GH#521, GH#522, GH#527, GH#545)
• Testing updates (GH#509, GH#516, GH#517, GH#539)

Thanks to the following people for contributing to this release: @bphi, @CharlesBradshaw, @CJStadler, @glentennis, @gsheni, @kmax12, @rwedge

Breaking Changes
• NUnique has been renamed to NumUnique.
  Previous behavior

```python
from featuretools.primitives import NUnique
```

New behavior

```python
from featuretools.primitives import NumUnique
```

v0.7.1 Apr 24, 2019
• Automatically generate feature name for controllable primitives (GH#481)
• Primitive docstring updates (GH#489, GH#492, GH#494, GH#495)
• Change primitive functions that returned strings to return functions (GH#499)
• CLI customizable via entrypoints (GH#493)
• Improve calculation of aggregation features on grandchildren (GH#479)
• Refactor entrypoints to use decorator (GH#483)
• Include doctests in testing suite (GH#491)
• Documentation updates (GH#490)
• Update how standard primitives are imported internally (GH#482)

Thanks to the following people for contributing to this release: @bukosabino, @CharlesBradshaw, @glentennis, @gsheni, @jeff-hernandez, @kmax12, @minksky, @rwedge, @thehomebrewnerd

v0.7.0 Mar 29, 2019
• Improve Entity Set Serialization (GH#361)
• Support calling a primitive instance’s function directly (GH#461, GH#468)
• Support other libraries extending featuretools functionality via entrypoints (GH#452)
• Remove featuretools install command (GH#475)
• Add GroupByTransformFeature (GH#455, GH#472, GH#476)
• Update Haversine Primitive (GH#435, GH#462)
• Add commutative argument to SubtractNumeric and DivideNumeric primitives (GH#457)
• Add FilePath variable_type (GH#470)
• Add PhoneNumber, DateOfBirth, URL variable types (GH#447)
• Generalize infer_variable_type, convert_variable_data and convert_all_variable_data methods (GH#423)
• Documentation updates (GH#438, GH#446, GH#458, GH#469)
• Testing updates (GH#440, GH#444, GH#445, GH#459)

Thanks to the following people for contributing to this release: @bukosabino, @CharlesBradshaw, @ColCarroll, @glentennis, @grayskripko, @gsheni, @jeff-hernandez, @jkinley, @kmax12, @RogerTangos, @rwedge

Breaking Changes

• ft.dfs now has a groupby_trans_primitives parameter that DFS uses to automatically construct features that group by an ID column and then apply a transform primitive to search group. This change applies to the following primitives: CumSum, CumCount, CumMean, CumMin, and CumMax.

  Previous behavior

  ```python
  ft.dfs(entityset=es,
          target_entity='customers',
          trans_primitives=["cum_mean"]
  )
  ```

  New behavior

  ```python
  ft.dfs(entityset=es,
          target_entity='customers',
          groupby_trans_primitives=["cum_mean"]
  )
  ```

  • Related to the above change, cumulative transform features are now defined using a new feature class, GroupByTransformFeature.

    Previous behavior

    ```python
    ft.Feature([base_feature, groupby_feature],
                primitive=CumulativePrimitive)
    ```

    New behavior

    ```python
    ft.Feature(base_feature, groupby=groupby_feature,
                primitive=CumulativePrimitive)
    ```

v0.6.1 Feb 15, 2019

• Cumulative primitives (GH#410)
• Entity.query_by_values now preserves row order of underlying data (GH#428)
• Implementing Country Code and Sub Region Codes as variable types (GH#430)
• Added IPAddress and EmailAddress variable types (GH#426)
• Install data and dependencies (GH#403)
• Add TimeSinceFirst, fix TimeSinceLast (GH#388)
• Allow user to pass in desired feature return types (GH#372)
• Add new configuration object (GH#401)
• Replace NUnique get_function (GH#434)
• _calculate_idenity_features now only returns the features asked for, instead of the entire entity (GH#429)
• Primitive function name uniqueness (GH#424)
• Update NumCharacters and NumWords primitives (GH#419)
• Removed Variable.dtype (GH#416, GH#433)
• Change to zipcode rep, str for pandas (GH#418)
• Remove pandas version upper bound (GH#408)
• Make S3 dependencies optional (GH#404)
• Check that agg_primitives and trans_primitives are right primitive type (GH#397)
• Mean primitive changes (GH#395)
• Fix transform stacking on multi-output aggregation (GH#394)
• Fix list_primitives (GH#391)
• Handle graphviz dependency (GH#389, GH#396, GH#398)
• Testing updates (GH#402, GH#417, GH#433)
• Documentation updates (GH#400, GH#409, GH#415, GH#417, GH#420, GH#421, GH#422, GH#431)

Thanks to the following people for contributing to this release: @CharlesBradshaw, @csala, @floscha, @gsheni, @jxwolstenholme, @kmax12, @RogerTangos, @rwedge

v0.6.0 Jan 30, 2018
• Primitive refactor (GH#364)
• Mean ignore NaNs (GH#379)
• Plotting entitysets (GH#382)
• Add seed features later in DFS process (GH#357)
• Multiple output column features (GH#376)
• Add ZipCode Variable Type (GH#367)
• Add primitive.get_filepath and example of primitive loading data from external files (GH#380)
• Transform primitives take series as input (GH#385)
• Update dependency requirements (GH#378, GH#383, GH#386)
• Add modulo to override tests (GH#384)
• Update documentation (GH#368, GH#377)
• Update README.md (GH#366, GH#373)
• Update CI tests (GH#359, GH#360, GH#375)

Thanks to the following people for contributing to this release: @floscha, @gsheni, @kmax12, @RogerTangos, @rwedge

v0.5.1 Dec 17, 2018
• Add missing dependencies (GH#353)
• Move comment to note in documentation (GH#352)

v0.5.0 Dec 17, 2018
• Add specific error for duplicate additional/copy_variables in normalize_entity (GH#348)
• Removed EntitySet._import_from_dataframe (GH#346)
• Removed time_index_reduce parameter (GH#344)
• Allow installation of additional primitives (GH#326)
• Fix DatetimeIndex variable conversion (GH#342)
• Update Sklearn DFS Transformer (GH#343)
• Clean up entity creation logic (GH#336)
• remove casting to list in transform feature calculation (GH#330)
• Fix sklearn wrapper (GH#335)
• Add readme to pypi
• Update conda docs after move to conda-forge (GH#334)
• Add wrapper for scikit-learn Pipelines (GH#323)
• Remove parse_date_cols parameter from EntitySet._import_from_dataframe (GH#333)

Thanks to the following people for contributing to this release: @bukosabino, @georgewambold, @gsheni, @jeff-hernandez, @kmax12, and @rwedge.

v0.4.1 Nov 29, 2018
• Resolve bug preventing using first column as index by default (GH#308)
• Handle return type when creating features from Id variables (GH#318)
• Make id an optional parameter of EntitySet constructor (GH#324)
• Handle primitives with same function being applied to same column (GH#321)
• Update requirements (GH#328)
• Clean up DFS arguments (GH#319)
• Clean up Pandas Backend (GH#302)
• Update properties of cumulative transform primitives (GH#320)
• Feature stability between versions documentation (GH#316)
• Add download count to GitHub readme (GH#310)
• Fixed #297 update tests to check error strings (GH#303)
• Remove usage of fixtures in agg primitive tests (GH#325)

v0.4.0 Oct 31, 2018
• Remove ft.utils.gen_utils.getsize and make pympler a test requirement (GH#299)
• Update requirements.txt (GH#298)
• Refactor EntitySet.find_path(...) (GH#295)
• Clean up unused methods (GH#293)
• Remove unused parents property of Entity (GH#283)
• Removed relationships parameter (GH#284)
• Improve time index validation (GH#285)
- Encode features with “unknown” class in categorical (GH#287)
- Allow where clauses on direct features in Deep Feature Synthesis (GH#279)
- Change to fullargspec (GH#288)
- Parallel verbose fixes (GH#282)
- Update tests for python 3.7 (GH#277)
- Check duplicate rows cutoff times (GH#276)
- Load retail demo data using compressed file (GH#271)

v0.3.1 Sept 28, 2018
- Handling time rewrite (GH#245)
- Update deep_feature_synthesis.py (GH#249)
- Handling return type when creating features from DatetimeTimeIndex (GH#266)
- Update retail.py (GH#259)
- Improve Consistency of Transform Primitives (GH#236)
- Update demo docstrings (GH#268)
- Handle non-string column names (GH#255)
- Clean up merging of aggregation primitives (GH#250)
- Add tests for Entity methods (GH#262)
- Handle no child data when calculating aggregation features with multiple arguments (GH#264)
- Add is_string utils function (GH#260)
- Update python versions to match docker container (GH#261)
- Handle where clause when no child data (GH#258)
- No longer cache demo csvs, remove config file (GH#257)
- Avoid stacking “expanding” primitives (GH#238)
- Use randomly generated names in retail csv (GH#233)
- Update README.md (GH#243)

v0.3.0 Aug 27, 2018
- Improve performance of all feature calculations (GH#224)
- Update agg primitives to use more efficient functions (GH#215)
- Optimize metadata calculation (GH#229)
- More robust handling when no data at a cutoff time (GH#234)
- Workaround categorical merge (GH#231)
- Switch which CSV is associated with which variable (GH#228)
- Remove unused kwargs from query_by_values, filter_and_sort (GH#225)
- Remove convert_links_to_integers (GH#219)
- Add conda install instructions (GH#223, GH#227)
- Add example of using Dask to parallelize to docs (GH#221)
v0.2.2 Aug 20, 2018

- Remove unnecessary check no related instances call and refactor (GH#209)
- Improve memory usage through support for pandas categorical types (GH#196)
- Bump minimum pandas version from 0.20.3 to 0.23.0 (GH#216)
- Better parallel memory warnings (GH#208, GH#214)
- Update demo datasets (GH#187, GH#201, GH#207)
- Make primitive lookup case insensitive (GH#213)
- Use capital name (GH#211)
- Set class name for Min (GH#206)
- Remove variable_types from normalize entity (GH#205)
- Handle parquet serialization with last time index (GH#204)
- Reset index of cutoff times in calculate feature matrix (GH#198)
- Check argument types for .normalize_entity (GH#195)
- Type checking ignore entities. (GH#193)

v0.2.1 July 2, 2018

- Cpu count fix (GH#176)
- Update flight (GH#175)
- Move feature matrix calculation helper functions to separate file (GH#177)

v0.2.0 June 22, 2018

- Multiprocessing (GH#170)
- Handle unicode encoding in repr throughout Featuretools (GH#161)
- Clean up EntitySet class (GH#145)
- Add support for building and uploading conda package (GH#167)
- Parquet serialization (GH#152)
- Remove variable stats (GH#171)
- Make sure index variable comes first (GH#168)
- No last time index update on normalize (GH#169)
- Remove list of times as on option for cutoff_time in calculate_feature_matrix (GH#165)
- Config does error checking to see if it can write to disk (GH#162)

v0.1.21 May 30, 2018

- Support Pandas 0.23.0 (GH#153, GH#154, GH#155, GH#159)
- No EntitySet required in loading/saving features (GH#141)
- Use s3 demo csv with better column names (GH#139)
- more reasonable start parameter (GH#149)
- add issue template (GH#133)
- Improve tests (GH#136, GH#137, GH#144, GH#147)
• Remove unused functions (GH#140, GH#143, GH#146)
• Update documentation after recent changes / removals (GH#157)
• Rename demo retail csv file (GH#148)
• Add names for binary (GH#142)
• EntitySet repr to use get_name rather than id (GH#134)
• Ensure config dir is writable (GH#135)

v0.1.20 Apr 13, 2018
• Primitives as strings in DFS parameters (GH#129)
• Integer time index bugfixes (GH#128)
• Add make_temporal_cutoffs utility function (GH#126)
• Show all entities, switch shape display to row/col (GH#124)
• Improved chunking when calculating feature matrices (GH#121)
• fixed num characters nan fix (GH#118)
• modify ignore_variables docstring (GH#117)

v0.1.19 Mar 21, 2018
• More descriptive DFS progress bar (GH#69)
• Convert text variable to string before NumWords (GH#106)
• EntitySet.concat() reindexes relationships (GH#96)
• Keep non-feature columns when encoding feature matrix (GH#111)
• Uses full entity update for dependencies of uses_full_entity features (GH#110)
• Update column names in retail demo (GH#104)
• Handle Transform features that need access to all values of entity (GH#91)

v0.1.18 Feb 27, 2018
• fixes related instances bug (GH#97)
• Adding non-feature columns to calculated feature matrix (GH#78)
• Relax numpy version req (GH#82)
• Remove entity_from_csv, tests, and lint (GH#71)

v0.1.17 Jan 18, 2018
• LatLong type (GH#57)
• Last time index fixes (GH#70)
• Make median agg primitives ignore nans by default (GH#61)
• Remove Python 3.4 support (GH#64)
• Change normalize_entity to update secondary_time_index (GH#59)
• Unpin requirements (GH#53)
• associative -> commutative (GH#56)
• Add Words and Chars primitives (GH#51)
v0.1.16 Dec 19, 2017
- fix EntitySet.combine_variables and standardize encode_features (GH#47)
- Python 3 compatibility (GH#16)

v0.1.15 Dec 18, 2017
- Fix variable type in demo data (GH#37)
- Custom primitive kwarg fix (GH#38)
- Changed order and text of arguments in make_trans_primitive docstring (GH#42)

v0.1.14 November 20, 2017
- Last time index (GH#33)
- Update Scipy version to 1.0.0 (GH#31)

v0.1.13 November 1, 2017
- Add MANIFEST.in (GH#26)

v0.1.11 October 31, 2017
- Package linting (GH#7)
- Custom primitive creation functions (GH#13)
- Split requirements to separate files and pin to latest versions (GH#15)
- Select low information features (GH#18)
- Fix docs typos (GH#19)
- Fixed Diff primitive for rare nan case (GH#21)
- added some missing doc strings (GH#23)
- Trend fix (GH#22)
- Remove as_dir=False option from EntitySet.to_pickle() (GH#20)
- Entity Normalization Preserves Types of Copy & Additional Variables (GH#25)

v0.1.10 October 12, 2017
- NumTrue primitive added and docstring of other primitives updated (GH#11)
- fixed hash issue with same base features (GH#8)
- Head fix (GH#9)
- Fix training window (GH#10)
- Add associative attribute to primitives (GH#3)
- Add status badges, fix license in setup.py (GH#1)
- fixed head printout and flight demo index (GH#2)

v0.1.9 September 8, 2017
- Documentation improvements
  - New `featuretools.demo.load_mock_customer` function

v0.1.8 September 1, 2017
- Bug fixes
• Added Percentile transform primitive

v0.1.7 August 17, 2017
• Performance improvements for approximate in calculate_feature_matrix and dfs
• Added Week transform primitive

v0.1.6 July 26, 2017
• Added load_features and save_features to persist and reload features
• Added save_progress argument to calculate_feature_matrix
• Added approximate parameter to calculate_feature_matrix and dfs
• Added load_flight to ft.demo

v0.1.5 July 11, 2017
• Windows support

v0.1.3 July 10, 2017
• Renamed feature submodule to primitives
• Renamed prediction_entity arguments to target_entity
• Added training_window parameter to calculate_feature_matrix

v0.1.2 July 3rd, 2017
• Initial release

3.20 Feature types

Featuretools groups features into four general types:

• Identity features
  • Transform and Cumulative Transform features
  • Aggregation features
  • Direct features

3.20.1 Identity Features

In Featuretools, each feature is defined as a combination of other features. At the lowest level are IdentityFeature features which are equal to the value of a single variable.

Most of the time, identity features will be defined transparently for you, such as in the transform feature example below. They may also be defined explicitly:

```python
In [1]: time_feature = ft.Feature(es['transactions']['transaction_time'])

In [2]: time_feature
Out[2]: <Feature: transaction_time>
```
3.20.2 Direct Features

Direct features are used to “inherit” feature values from a parent to a child entity. Suppose each event is associated with a single instance of the entity `products`. This entity has metadata about different products, such as brand, price, etc. We can pull the brand of the product into a feature of the event entity by including the event entity as an argument to `Feature`. In this case, `Feature` is an alias for `primitives.DirectFeature`:

```python
In [3]: brand = ft.Feature(es["products"]['brand'], entity=es["transactions"])
In [4]: brand
Out[4]: <Feature: products.brand>
```

3.20.3 Transform Features

Transform features take one or more features on an `Entity` and create a single new feature for that same entity. For example, we may want to take a fine-grained “timestamp” feature and convert it into the hour of the day in which it occurred.

```python
In [5]: from featuretools.primitives import Hour
In [6]: ft.Feature(time_feature, primitive=Hour)
Out[6]: <Feature: HOUR(transaction_time)>
```

Using algebraic and boolean operations, transform features can combine other features into arbitrary expressions. For example, to determine if a given event event happened in the afternoon, we can write:

```python
In [7]: hour_feature = ft.Feature(time_feature, primitive=Hour)
In [8]: after_twelve = hour_feature > 12
In [9]: after_twelve
Out[9]: <Feature: HOUR(transaction_time) > 12>
In [10]: at_twelve = hour_feature == 12
In [11]: before_five = hour_feature <= 17
In [12]: is_afternoon = after_twelve & before_five
In [13]: is_afternoon
Out[13]: <Feature: AND(HOUR(transaction_time) > 12, HOUR(transaction_time) <= 17)>
```

3.20.4 Aggregation Features

Aggregation features are used to create features for a `parent entity` by summarizing data from a `child entity`. For example, we can create a `Count` feature which counts the total number of events for each customer:

```python
In [14]: from featuretools.primitives import Count
In [15]: total_events = ft.Feature(es["transactions"]["transaction_id"], parent_entity=es["customers"], primitive=Count)
In [16]: fm = ft.calculate_feature_matrix([total_events], es)
```

(continues on next page)
In [17]: fm.head()
Out[17]:
    COUNT(transactions)
customer_id
5    79
4   109
1   126
3    93
2    93

Note: For users who have written aggregations in SQL, this concept will be familiar. One key difference in featuretools is that `GROUP BY` and `JOIN` are implicit. Since the parent and child entities are specified, featuretools can infer how to group the child entity and then join the resulting aggregation back to the parent entity.

Often times, we only want to aggregate using a certain amount of previous data. For example, we might only want to count events from the past 30 days. In this case, we can provide the `use_previous` parameter:

```
In [18]: total_events_last_30_days = ft.Feature(es["transactions"]['transaction_id'],
  ...:    parent_entity=es["customers"],
  ...:    use_previous="30 days",
  ...:    primitive=Count)

In [19]: fm = ft.calculate_feature_matrix([total_events_last_30_days], es)

In [20]: fm.head()
Out[20]:
    COUNT(transactions, Last 30 Days)
customer_id
5      0.0
4      0.0
1      0.0
3      0.0
2      0.0
```

Unlike with cumulative transform features, the `use_previous` parameter here is evaluated relative to instances of the parent entity, not the child entity. The above feature translates roughly to the following: “For each customer, count the events which occurred in the 30 days preceding the customer’s timestamp.”

Find the list of the supported aggregation features [here](#).

### 3.20.5 Where clauses

When defining aggregation or cumulative transform features, we can provide a `where` parameter to filter the instances we are aggregating over. Using the `is_afternoon` feature from earlier, we can count the total number of events which occurred in the afternoon:

```
In [21]: afternoon_events = ft.Feature(es["transactions"]['transaction_id'],
  ...:    parent_entity=es["customers"],
  ...:    where=is_afternoon,
  ...:    primitive=Count).rename("afternoon_events")
```
The where argument can be any previously-defined boolean feature. Only instances for which the where feature is True are included in the final calculation.

### 3.20.6 Aggregations of Direct Feature

Composing multiple feature types is an extremely powerful abstraction that Featuretools makes simple. For instance, we can aggregate direct features on a child entity from a different parent entity. For example, to calculate the most common brand a customer interacted with:

```python
In [24]: from featuretools.primitives import Mode
In [25]: brand = ft.Feature(es["products"]['brand'], entity=es['transactions'])
In [26]: favorite_brand = ft.Feature(brand, parent_entity=es['customers'], primitive=Mode)
In [27]: fm = ft.calculate_feature_matrix([favorite_brand], es)
In [28]: fm.head()
```

```
Out[28]:  
    MODE(transactions.products.brand)  
  customer_id
   5    B
   4    B
   1    B
   3    B
   2    B
```

**Side note: Feature equality overrides default equality**

Because we can check if two features are equal (or a feature is equal to a value), we override Python’s equals (==) operator. This means to check if two feature objects are equal (instead of their computed values in the feature matrix), we need to compare their hashes:

```python
In [29]: hour_feature.hash() == hour_feature.hash()
Out[29]: True
In [30]: hour_feature.hash() != hour_feature.hash()
```

```
\\\\\\\\Out[30]: False
```

dictionaries and sets use equality underneath, so those keys need to be hashes as well.
3.21 Save Intermediate Feature Matrix Results

In this tutorial, we will go over the how to save intermediate results when computing the feature matrix.

```python
In [31]: myset = set()
In [32]: myset.add(hour_feature.hash())
In [33]: hour_feature.hash() in myset
Out[33]: True
In [34]: mydict = dict()
In [35]: mydict[hour_feature.hash()] = hour_feature
In [36]: hour_feature.hash() in mydict
Out[36]: True
```

If you want to save intermediate computations as CSVs, simply pass the location of a directory of where the computation should be saved. For example, if you pass a directory called “ft_temp”, CSV files will be output to the directory, named according to the timestamp that it represents.

```python
import featuretools as ft

In this example, we will use a dataset of retail data of customers from a UK website from December 2010 to December 2011.

```python
es = ft.demo.load_retail(nrows=10000)
```  
let's use a simple feature for this example.

```python
region = ft.Feature(es["customers"]['Country'])
```  
We can supply “cutoff times” to specify that we want to calculate features one year after a customer’s first invoice.

```python
import pandas as pd
cutoff_times = es["customers"].df["CustomerID", "first_invoices_time"].rename(    columns=("CustomerID": "instance_id", "first_invoices_time": "time"))
cutoff_times["time"] = cutoff_times["time"] + pd.Timedelta("365 days")
```  
Here is what some of the cutoff times look like.

```
<table>
<thead>
<tr>
<th>instance_id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>17850.0</td>
<td>2011-12-01 08:26:00</td>
</tr>
<tr>
<td>13047.0</td>
<td>2011-12-01 08:34:00</td>
</tr>
<tr>
<td>12583.0</td>
<td>2011-12-01 08:45:00</td>
</tr>
<tr>
<td>13748.0</td>
<td>2011-12-01 09:00:00</td>
</tr>
<tr>
<td>15100.0</td>
<td>2011-12-01 09:09:00</td>
</tr>
<tr>
<td>15291.0</td>
<td>2011-12-01 09:32:00</td>
</tr>
<tr>
<td>14688.0</td>
<td>2011-12-01 09:37:00</td>
</tr>
<tr>
<td>14527.0</td>
<td>2011-12-01 09:41:00</td>
</tr>
<tr>
<td>15311.0</td>
<td>2011-12-01 09:41:00</td>
</tr>
<tr>
<td>17809.0</td>
<td>2011-12-01 09:41:00</td>
</tr>
</tbody>
</table>
```

If you want to save intermediate computations as CSVs, simply pass the location of a directory of where the computation should be saved. For example, if you pass a directory called “ft_temp”, CSV files will be output to the directory, named according to the timestamp that it represents.
```python
[6]:
    import os
    save_progress = os.path.join(os.getcwd(), 'ft_temp')
    if not os.path.exists(save_progress):
        os.makedirs(save_progress)

[7]:
    fm_save = ft.calculate_feature_matrix([region],
                                           entityset=es,
                                           cutoff_time=cutoff_times.sample(10),
                                           save_progress=save_progress)

As seen below, there are now files in the directory, named by timestamp.

[8]:
    % ls ft_temp/
    ft_2011_12_01_03-08-00-000000.csv  ft_2011_12_02_05-03-00-000000.csv
    ft_2011_12_01_09-00-00-000000.csv  ft_2011_12_02_05-19-00-000000.csv
    ft_2011_12_01_12-43-00-000000.csv  ft_2011_12_02_12-07-00-000000.csv
    ft_2011_12_01_12-51-00-000000.csv  ft_2011_12_02_12-18-00-000000.csv
    ft_2011_12_02_03-19-00-000000.csv  ft_2011_12_03_12-57-00-000000.csv

[9]:
    import shutil
    shutil.rmtree(save_progress)
```

3.21. Save Intermediate Feature Matrix Results
CHAPTER
FOUR

OTHER LINKS

- genindex
- search
Symbols

__getitem__() (featuretools.entityset.EntitySet method), 215
__init__() (featuretools.Entity method), 209
__init__() (featuretools.EntitySet method), 208
__init__() (featuretools.Relationship method), 210
__init__() (featuretools.Timedelta method), 142
__init__() (featuretools.primitives.Absolute method), 170
__init__() (featuretools.primitives.AggregationPrimitive method), 144
__init__() (featuretools.primitives.All method), 160
__init__() (featuretools.primitives.And method), 168
__init__() (featuretools.primitives.Any method), 161
__init__() (featuretools.primitives.AvgTimeBetween method), 155
__init__() (featuretools.primitives.Count method), 148
__init__() (featuretools.primitives.CumCount method), 183
__init__() (featuretools.primitives.CumMax method), 187
__init__() (featuretools.primitives.CumMean method), 185
__init__() (featuretools.primitives.CumMin method), 186
__init__() (featuretools.primitives.CumSum method), 184
__init__() (featuretools.primitives.Day method), 178
__init__() (featuretools.primitives.Diff method), 182
__init__() (featuretools.primitives.Entropy method), 166
__init__() (featuretools.primitives.First method), 162
__init__() (featuretools.primitives.Haversine method), 191
__init__() (featuretools.primitives.Hour method), 177
__init__() (featuretools.primitives.IsIn method), 167
__init__() (featuretools.primitives.IsWeekend method), 176
__init__() (featuretools.primitives.Last method), 163
__init__() (featuretools.primitives.Latitude method), 190
__init__() (featuretools.primitives.Longitude method), 190
__init__() (featuretools.primitives.Max method), 152
__init__() (featuretools.primitives.Mean method), 149
__init__() (featuretools.primitives.Median method), 153
__init__() (featuretools.primitives.Min method), 151
__init__() (featuretools.primitives.Minute method), 174
__init__() (featuretools.primitives.Mode method), 154
__init__() (featuretools.primitives.Month method), 180
__init__() (featuretools.primitives.Not method), 169
__init__() (featuretools.primitives.NumCharacters method), 188
__init__() (featuretools.primitives.NumUnique method), 159
__init__() (featuretools.primitives.NumWords method), 189
__init__() (featuretools.primitives.Or method), 169
__init__() (featuretools.primitives.PercentTrue method), 160
__init__() (featuretools.primitives.Percentile method), 171
__init__() (featuretools.primitives.Second method), 173
__init__() (featuretools.primitives.Skew method), 164
__init__() (featuretools.primitives.Std method), 153
__init__() (featuretools.primitives.Sum method), 150
__init__() (featuretools.primitives.TimeSince method), 172
__init__() (featuretools.primitives.TimeSinceFirst method), 158
__init__() (featuretools.primitives.TimeSinceLast method), 157
__init__() (featuretools.primitives.TimeSinceLast method), 172
Categorical (class in featuretools.variable_types), 223
child entity, 131
child_variable (featuretools.entityset.Relationship attribute), 218
convert_variable_type() (featuretools.entityset.Relationship attribute), 218
CountryCode (class in featuretools.primitives), 183
CumMax (class in featuretools.primitives), 187
CumMean (class in featuretools.primitives), 185
CumMin (class in featuretools.primitives), 186
CumSum (class in featuretools.primitives), 184
cutoff time, 131
DateOfBirth (class in featuretools.variable_types), 229
DateTime (class in featuretools.variable_types), 222
DateTimeTimeIndex (class in featuretools.variable_types), 221
dfs() (in module featuretools), 135
DFSTransformer (class in featuretools.wrappers), 138
Diff (class in featuretools.primitives), 181
diversityScore (class in nlp_primitives), 193
EmailAddress (class in featuretools.variable_types), 227
encode_features() (in module featuretools), 204
diff (class in featuretools.primitives), 178
entropy, 131
Entity (class in featuretools), 209
dict (featuretools.EntitySet attribute), 207
dict_from_dataframe() (featuretools.EntitySet method), 211
EntitySet, 131
Entoty (class in featuretools), 207
Entropy (class in featuretools.primitives), 166
feature, 131
feature engineering, 131
FilePath (class in featuretools.variable_types), 230
find_backward_paths() (featuretools.entityset.EntitySet method), 216
find_forward_paths() (featuretools.entityset.EntitySet method), 216
First (class in featuretools.primitives), 162
FullName (class in featuretools.variable_types), 227
Hour (class in featuretools.primitives), 177
Id (class in featuretools.variable_types), 220
id (featuretools.EntitySet attribute), 207
Index (class in featuretools.variable_types), 219
instance, 131
IPAddress (class in featuretools.primitives), 226
IsIn (class in featuretools.primitives), 167
IsWeekend (class in featuretools.primitives), 176
Last (class in featuretools.primitives), 163
Latitude (class in featuretools.primitives), 189
LatLong (class in featuretools.variable_types), 225
load_features() (in module featuretools), 206
load_flight() (in module featuretools.demo), 134
load_mock_customer() (in module featuretools.demo), 134
make_agg_primitive() (in module featuretools.primitives), 145
make_temporal_cutoffs() (in module featuretools), 143
make_trans_primitive() (in module featuretools.primitives), 146
Max (class in featuretools.primitives), 152
Max (class in featuretools.primitives), 152
max (featuretools.variable_types.Numeric attribute), 222
Mean (class in featuretools.primitives), 149
mean (featuretools.variable_types.Numeric attribute), 222
MeanCharactersPerWord (class in nlp_primitives), 195
Median (class in featuretools.primitives), 153
Min (class in featuretools.primitives), 151
min (featuretools.variable_types.Numeric attribute), 222
Featuretools Documentation, Release 0.13.4

Minute (class in featuretools.primitives), 174
Mode (class in featuretools.primitives), 154
Month (class in featuretools.primitives), 179

normalize_entity() (featuretools.EntitySet method), 213
Not (class in featuretools.primitives), 169
NumCharacters (class in featuretools.primitives), 188
Numeric (class in featuretools.primitives), 188
NumericTimeIndex (class in featuretools.variable_types), 222
NumUnique (class in featuretools.primitives), 158
NumWords (class in featuretools.primitives), 188

Or (class in featuretools.primitives), 168
Ordinal (class in featuretools.variable_types), 224

parent_entity, 131
parent_entity (featuretools.entityset.Relationship attribute), 218
parent_variable (featuretools.entityset.Relationship attribute), 218
PartOfSpeechCount (class in nlp_primitives), 195
Percentile (class in featuretools.primitives), 171
PercentTrue (class in featuretools.primitives), 159
PhoneNumber (class in featuretools.variable_types), 228
plot() (featuretools.entityset.EntitySet method), 217
PolarityScore (class in nlp_primitives), 196
PunctuationCount (class in nlp_primitives), 197

read_entityset() (in module featuretools), 214
relationship, 131
Relationship (class in featuretools), 210
relationships (featuretools.EntitySet attribute), 207
remove_low_information_features() (in module featuretools.selection), 231
rename() (featuretools.feature_base.FeatureBase method), 202

save_features() (in module featuretools), 206
Second (class in featuretools.primitives), 173
Skew (class in featuretools.primitives), 164
Std (class in featuretools.primitives), 152
std (featuretools.variable_types.Numeric attribute), 222
StopwordCount (class in nlp_primitives), 198
SubRegionCode (class in featuretools.variable_types), 230

Sum (class in featuretools.primitives), 150

target_entity, 131
Text (class in featuretools.variable_types), 225
time_type (featuretools.EntitySet attribute), 207
timedelta (class in featuretools), 142
TimeIndex (class in featuretools.variable_types), 220
TimeSince (class in featuretools.primitives), 172
TimeSinceFirst (class in featuretools.primitives), 157
TimeSinceLast (class in featuretools.primitives), 156
TimeSincePrevious (class in featuretools.primitives), 182
TitleWordCount (class in nlp_primitives), 199
to_csv() (featuretools.entityset.EntitySet method), 214
to_parquet() (featuretools.entityset.EntitySet method), 215
to_pickle() (featuretools.entityset.EntitySet method), 215
TransformPrimitive (class in featuretools.primitives), 144
Trend (class in featuretools.primitives), 165

UniversalSentenceEncoder (class in nlp_primitives), 200
UpperCaseCount (class in nlp_primitives), 201
URL (class in featuretools.variable_types), 228

variable, 131

Week (class in featuretools.primitives), 179
Weekday (class in featuretools.primitives), 175

Year (class in featuretools.primitives), 180

ZIPCode (class in featuretools.variable_types), 226