Featuretools Documentation

Release 0.20.0

Feature Labs, Inc.

Oct 19, 2020

CONTENTS

Featuretools is a framework to perform automated feature engineering. It excels at transforming temporal and relational datasets into feature matrices for machine learning.

CHAPTER

ONE

5 MINUTE QUICK START

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.

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

1.1 Load Mock Data

```
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

```
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

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)

(continued from previous page)

```
24          25            3  desktop 2014-01-01 05:59:40

In [7]: transactions_df = data["transactions"]

In [8]: transactions_df.sample(5)
Out[8]:
     transaction_id  session_id    transaction_time product_id  amount
74              232           5 2014-01-01 01:20:10          1  139.20
231              27          17 2014-01-01 04:10:15          2   90.79
434              36          31 2014-01-01 07:50:10          3   62.35
420              56          30 2014-01-01 07:35:00          3   72.70
54              444           4 2014-01-01 00:58:30          4   43.59
```

First, we specify a dictionary with all the entities in our dataset.

```
In [9]: entities = {
   ...:    "customers" : (customers_df, "customer_id"),
   ...:    "sessions" : (sessions_df, "session_id", "session_start"),
   ...:    "transactions" : (transactions_df, "transaction_id", "transaction_time")
   ...: }
   ...:
```

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

```
(parent_entity, parent_variable, child_entity, child_variable)
```

In this dataset we have two relationships

```
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 ouput 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

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

In [12]: feature_matrix_customers
Out[12]:
           zip_code  COUNT(sessions) MODE(sessions.device)  NUM_UNIQUE(sessions.
↪device)  COUNT(transactions)  MAX(transactions.amount)  MEAN(transactions.amount) ␣
↪MIN(transactions.amount)  MODE(transactions.product_id)  NUM_UNIQUE(transactions.
↪product_id)  SKEW(transactions.amount)  STD(transactions.amount)  SUM(transactions.
↪amount)  DAY(date_of_birth)  DAY(join_date)  MONTH(date_of_birth)  MONTH(join_date)␣
↪ WEEKDAY(date_of_birth)  WEEKDAY(join_date)  YEAR(date_of_birth)  YEAR(join_date)
↪MAX(sessions.COUNT(transactions))  MAX(sessions.MEAN(transactions.amount))  ␣
↪MAX(sessions.MIN(transactions.amount))  MAX(sessions.NUM_UNIQUE(transactions.
↪product_id))  MAX(sessions.SKEW(transactions.amount))  MAX(sessions.
↪STD(transactions.amount))  MAX(sessions.SUM(transactions.amount))  MEAN(sessions.
```

(continues on next page)

(continued from previous page)

```
customer_id
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
1              60091                    8                 mobile
↪ 3                  126                      139.43                    71.631905
↪                5.81                              4
↪ 5                  0.019698                    40.442059                       9025.62
↪                 18                17                       7                 4
↪             0                     6                  1994             2011
↪                      25                                 88.755625
↪                26.36                                                   5
↪                     0.640252                                46.905665
↪                  1613.93                        15.750000
↪       132.246250                               72.774140
↪        9.823750                                          5.000000
↪                 -0.059515                                39.093244
↪             1128.202500                                12
↪         118.90                                50.623125
↪                      5                                -1.038434
↪           30.450261                                    809.97
↪          1                                               4
↪         1                                   2
↪2014                                      1
↪            4                                         1
↪                 1                                       1
↪    1.946018                               -0.780493
↪  -0.424949                               2.440005
↪           0.000000                               -0.312355
↪          0.778170                        4.062019
↪7.322191                               13.759314                               6.
↪954507                                         0.000000
↪      0.589386                             279.510713
↪    1057.97                              582.193117
```

(continues on next page)

(continued from previous page)

```
2              13244                    7                desktop
↪ 3                    93                     146.81                  77.422366
↪              8.73                                4
↪ 5                    0.098259                  37.705178                   7200.28
↪               18                15                   8                 4
↪            0                     6                1986             2012
↪                   18                                96.581000
↪              56.46                                                   5
↪                   0.755711                              47.935920
↪                 1320.64                           13.285714
↪       133.090000                               78.415122
↪       22.085714                                         5.000000
↪                 -0.039663                              36.957218
↪            1028.611429                             8
↪         100.04                                61.910000
↪                        5                               -0.763603
↪           27.839228                               634.84
↪          1                                           3
↪         1                                     2
↪2014                                        1
↪           4                                          1
↪                1                                        1
↪  -0.303276                               -1.539467
↪  0.235296                                2.154929
↪           0.000000                                 0.013087
↪         -0.440929                          3.450328
↪17.221593                               11.477071                              15.
↪874374                                          0.000000
↪       0.509798                             251.609234
↪     931.63                              548.905851
↪ 154.60                                                    35
↪       -0.277640                              258.700528
↪           2                             desktop
↪         1                                          3
3              13244                    6                desktop
↪ 3                    93                     149.15                  67.060430
↪              5.89                                1
↪ 5                    0.418230                  43.683296                   6236.62
↪               21                13                   11                8
↪            4                     5                2003             2011
↪                   18                                82.109444
↪              20.06                                                   5
↪                   0.854976                              50.110120
↪                 1477.97                           15.500000
↪       141.271667                               67.539577
↪       11.035000                                         4.833333
↪                 0.381014                               42.883316
↪            1039.436667                            11
↪         126.74                                55.579412
↪                        4                               -0.289466
↪           35.704680                               889.21
↪          1                                           1
↪         1                                     2
↪2014                                        1
↪           4                                          1
↪                1                                        1
↪  -1.507217                               -0.941078
↪  0.678544                                1.000771
↪         -2.449490                                -0.245703
↪         2.246479                           2.428992
↪10.724241                               11.174282                              5.
↪424407                                          0.408248
↪       0.429374                             219.021420
↪     847.63                              405.237462
```

(continues on next page)

(continued from previous page)

```
4             60091                8                mobile
↪ 3                 109                   149.95                 80.070459
↪             5.73                             2
↪ 5                 -0.036348                45.068765                   8727.68
↪              15               8                    8                 4
↪           1                   4                2006              2011
↪                   18                          110.450000
↪             54.83                                               5
↪                  0.382868                          54.293903
↪               1351.46                         13.625000
↪      144.748750                              81.207189
↪      16.438750                                     4.625000
↪                0.000346                           44.515729
↪          1090.960000                             10
↪        139.20                                70.638182
↪                   4                                  -0.711744
↪         29.026424                             771.68
↪          1                                      1
↪        1                                2
↪2014                                    1
↪           5                                     1
↪                1                                   1
↪   0.282488                             0.027256
↪  1.980948                             2.103510
↪         -0.644061                              -1.065663
↪        -0.391805                       3.335416
↪3.514421                        13.027258                               16.
↪960575                                   0.517549
↪      0.387884                          235.992478
↪   1157.99                           649.657515
↪ 131.51                                            37
↪       0.002764                          356.125829
↪           4                         mobile
↪          1                                    3
5             60091                6                mobile
↪ 3                  79                   149.02                 80.375443
↪             7.55                             5
↪ 5                 -0.025941                44.095630                   6349.66
↪              28              17                    7                 7
↪           5                   5                1984              2010
↪                   18                           94.481667
↪             20.65                                               5
↪                  0.602209                          51.149250
↪               1700.67                         13.166667
↪      139.960000                              78.705187
↪      14.415000                                     5.000000
↪                0.002397                           43.312326
↪          1058.276667                              8
↪        128.51                                66.666667
↪                   5                                  -0.539060
↪         36.734681                             543.18
↪          1                                      3
↪        1                                2
↪2014                                    1
↪           5                                     1
↪                1                                   1
↪  -0.317685                            -0.333796
↪  0.335175                            -0.470410
↪          0.000000                                  0.204548
↪         0.472342                       3.600926
↪7.928001                        11.007471                               4.
↪961414                                   0.000000
↪      0.415426                          402.775486
↪    839.76                           472.231119
```

(continues on next page)

(continued from previous 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  COUNT(transactions)  MAX(transactions.amount) ␣
↪MEAN(transactions.amount)  MIN(transactions.amount)  MODE(transactions.product_id) ␣
↪NUM_UNIQUE(transactions.product_id)  SKEW(transactions.amount)  STD(transactions.
↪amount)  SUM(transactions.amount)  DAY(session_start)  MONTH(session_start) ␣
↪WEEKDAY(session_start)  YEAR(session_start) customers.zip_code  MODE(transactions.
↪DAY(transaction_time))  MODE(transactions.MONTH(transaction_time)) ␣
↪MODE(transactions.WEEKDAY(transaction_time))  MODE(transactions.YEAR(transaction_
↪time))  NUM_UNIQUE(transactions.DAY(transaction_time))  NUM_UNIQUE(transactions.
↪MONTH(transaction_time))  NUM_UNIQUE(transactions.WEEKDAY(transaction_time))  NUM_
↪UNIQUE(transactions.YEAR(transaction_time))  customers.COUNT(sessions) customers.
↪MODE(sessions.device)  customers.NUM_UNIQUE(sessions.device)  customers.
↪COUNT(transactions)  customers.MAX(transactions.amount)  customers.
↪MEAN(transactions.amount)  customers.MIN(transactions.amount)  customers.
↪MODE(transactions.product_id)  customers.NUM_UNIQUE(transactions.product_id) ␣
↪customers.SKEW(transactions.amount)  customers.STD(transactions.amount)  customers.
↪SUM(transactions.amount)  customers.DAY(date_of_birth)  customers.DAY(join_date) ␣
↪customers.MONTH(date_of_birth)  customers.MONTH(join_date)  customers.WEEKDAY(date_
↪of_birth)  customers.WEEKDAY(join_date)  customers.YEAR(date_of_birth)  customers.
↪YEAR(join_date)
session_id                                                                          ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪
1                     2  desktop                   16                     141.66    ␣
↪          76.813125                  20.91                                  3       ␣
↪                                5                   0.295458              41.600976 ␣
↪                 1229.01                 1                      1                   ␣
↪     2                   2014                 13244                                 ␣
↪    1                                            1                                  ␣
↪          2                                        2014                             ␣
↪                1                                                  1                ␣
↪                                   1                                                ␣
↪1                           7                          desktop
```

(continues on next page)

(continued from previous page)

```
2                      5   mobile                       10                    135.25
↪           74.696000                    9.32                                  5
↪                             5                  -0.160550                  45.893591
↪                   746.96                 1                          1
↪      2                 2014              60091
↪    1                                         1
↪          2                                     2014
↪                1                                                      1
↪                                 1
↪1                          6                              mobile
↪              3                                79                               149.02
↪                           80.375443                                  7.55
↪                           5                                                  5
↪                   -0.025941                            44.095630
↪          6349.66                              28                          17
↪                    7                              7                                  5
↪                           5                            1984
↪2010
3                      4   mobile                       15                    147.73
↪           88.600000                    8.70                                  1
↪                             5                  -0.324012                  46.240016
↪                  1329.00                 1                          1
↪      2                 2014              60091
↪    1                                         1
↪          2                                     2014
↪                1                                                      1
↪                                 1
↪1                          8                              mobile
↪              3                               109                               149.95
↪                           80.070459                                  5.73
↪                           2                                                  5
↪                   -0.036348                            45.068765
↪          8727.68                              15                           8
↪                    8                              4                                  1
↪                           4                            2006
↪2011
4                      1   mobile                       25                    129.00
↪           64.557200                    6.29                                  5
↪                             5                   0.234349                  40.187205
↪                  1613.93                 1                          1
↪      2                 2014              60091
↪    1                                         1
↪          2                                     2014
↪                1                                                      1
↪                                 1
↪1                          8                              mobile
↪              3                               126                               139.43
↪                           71.631905                                  5.81
↪                           4                                                  5
↪                    0.019698                            40.442059
↪          9025.62                              18                          17
↪                    7                              4                                  0
↪                           6                            1994
↪2011
5                      4   mobile                       11                    139.20
↪           70.638182                    7.43                                  5
↪                             5                   0.336381                  48.918663
↪                   777.02                 1                          1
↪      2                 2014              60091
↪    1                                         1
↪          2                                     2014
↪                1                                                      1
↪                                 1
↪1                          8                              mobile
```

(continues on next page)

(continued from previous page)

```
```

CHAPTER

TWO

WHAT'S NEXT?

- 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*

CHAPTER

THREE

TABLE OF CONTENTS

3.1 Install

Featuretools is available for Python 3.6, 3.7, and 3.8. 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* or *featuretools.graph_feature()* you will need to install the graphviz library.

Conda users:

```
conda install python-graphviz
```

Ubuntu:

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

Mac OS:

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

Windows:

```
conda install python-graphviz
```

3.1.3 Install from Source

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

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

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

```
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:

```
mkvirtualenv featuretools
```

Install development requirements

Run:

```
make installdeps
```

Test

Note: In order to the 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 scatch
make clean html
```

Note: The Featuretools library must be import-able to build the docs.

3.2 Getting Started

For a quick introduction to Featuretools, check out our *5 minute quick start guide*.

How to start working with Featuretools; the main concepts:

3.2.1 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.

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"])

In [4]: transactions_df.sample(10)
Out[4]:
     transaction_id  session_id    transaction_time product_id  amount  customer_id  ␣
↪device        session_start zip_code           join_date date_of_birth
264             380          21 2014-01-01 05:14:10          5   57.09            4 ␣
↪desktop 2014-01-01 05:02:15    60091 2011-04-08 20:08:14    2006-08-15
19              244          10 2014-01-01 02:34:55          2  116.95            2  ␣
↪tablet 2014-01-01 02:31:40    13244 2012-04-15 23:31:04    1986-08-18
314             299           6 2014-01-01 01:32:05          4   64.99            1  ␣
↪tablet 2014-01-01 01:23:25    60091 2011-04-17 10:48:33    1994-07-18
290              78           4 2014-01-01 00:54:10          1   37.50            1  ␣
↪mobile 2014-01-01 00:44:25    60091 2011-04-17 10:48:33    1994-07-18
379             457          27 2014-01-01 06:37:35          1   19.16            1  ␣
↪mobile 2014-01-01 06:34:20    60091 2011-04-17 10:48:33    1994-07-18
335             477           9 2014-01-01 02:30:35          3   41.70            1 ␣
↪desktop 2014-01-01 02:15:25    60091 2011-04-17 10:48:33    1994-07-18
293             103           4 2014-01-01 00:57:25          5   20.79            1  ␣
↪mobile 2014-01-01 00:44:25    60091 2011-04-17 10:48:33    1994-07-18
271             390          22 2014-01-01 05:21:45          2   54.83            4 ␣
↪desktop 2014-01-01 05:21:45    60091 2011-04-08 20:08:14    2006-08-15
404             476          29 2014-01-01 07:24:10          4  121.59            1  ␣
↪mobile 2014-01-01 07:10:05    60091 2011-04-17 10:48:33    1994-07-18
179              90           3 2014-01-01 00:35:45          1   75.73            4  ␣
↪mobile 2014-01-01 00:28:10    60091 2011-04-08 20:08:14    2006-08-15
```

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]:
  product_id brand
0          1     B
1          2     B
2          3     B
3          4     B
4          5     A
```

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")
```

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",
   ...:                               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 in 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 = zip_code)>]
```

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",
   ....:                               dataframe=products_df,
   ....:                               index="product_id")
   ....:

In [12]: es
Out[12]:
Entityset: customer_data
  Entities:
    transactions [Rows: 500, Columns: 11]
    products [Rows: 5, Columns: 2]
  Relationships:
    No relationships
```

With two entities in our entity set, we can add a relationship between them.

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.

```
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.

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.

```
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"])
   ....:
```

(continues on next page)

(continued from previous page)

```
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
Out[19]:
[<Variable: session_id (dtype = index)>,
 <Variable: device (dtype = categorical)>,
 <Variable: customer_id (dtype = numeric)>,
 <Variable: zip_code (dtype = zip_code)>,
 <Variable: session_start (dtype: datetime_time_index, format: None)>,
 <Variable: join_date (dtype: datetime, format: None)>]
```

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]:
   session_id   device  customer_id zip_code       session_start           join_date
1           1  desktop            2    13244 2014-01-01 00:00:00 2012-04-15 23:31:04
2           2   mobile            5    60091 2014-01-01 00:17:20 2010-07-17 05:27:50
3           3   mobile            4    60091 2014-01-01 00:28:10 2011-04-08 20:08:14
4           4   mobile            1    60091 2014-01-01 00:44:25 2011-04-17 10:48:33
5           5   mobile            4    60091 2014-01-01 01:11:30 2011-04-08 20:08:14
```

(continues on next page)

(continued from previous page)

```
In [21]: es["transactions"].df.head(5)
Out[21]:
     transaction_id  session_id    transaction_time  amount  date_of_birth  product_id
298             298           1 2014-01-01 00:00:00  127.64     1986-08-18           5
2                 2           1 2014-01-01 00:01:05  109.48     1986-08-18           2
308             308           1 2014-01-01 00:02:10   95.06     1986-08-18           3
116             116           1 2014-01-01 00:03:15   78.92     1986-08-18           4
371             371           1 2014-01-01 00:04:20   31.54     1986-08-18           3
```

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

```
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
```

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  COUNT(transactions)  MAX(transactions.amount)  MEAN(transactions.
↪amount)  MIN(transactions.amount)  MODE(transactions.session_id)  NUM_
↪UNIQUE(transactions.session_id)  SKEW(transactions.amount)  STD(transactions.
↪amount)  SUM(transactions.amount)  MODE(transactions.DAY(date_of_birth)) ␣
↪MODE(transactions.DAY(transaction_time))  MODE(transactions.MONTH(date_of_birth)) ␣
↪MODE(transactions.MONTH(transaction_time))  MODE(transactions.WEEKDAY(date_of_
↪birth))  MODE(transactions.WEEKDAY(transaction_time))  MODE(transactions.YEAR(date_
↪of_birth))  MODE(transactions.YEAR(transaction_time))  MODE(transactions.sessions.
↪customer_id) MODE(transactions.sessions.device)  NUM_UNIQUE(transactions.DAY(date_
↪of_birth))  NUM_UNIQUE(transactions.DAY(transaction_time))  NUM_UNIQUE(transactions.
↪MONTH(date_of_birth))  NUM_UNIQUE(transactions.MONTH(transaction_time))  NUM_
↪UNIQUE(transactions.WEEKDAY(date_of_birth))  NUM_UNIQUE(transactions.
↪WEEKDAY(transaction_time))  NUM_UNIQUE(transactions.YEAR(date_of_birth))  NUM_
↪UNIQUE(transactions.YEAR(transaction_time))  NUM_UNIQUE(transactions.sessions.
↪customer_id)  NUM_UNIQUE(transactions.sessions.device)
```

(continues on next page)

(continued from previous page)

```
product_id                                                                                      ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪                                                                                               ␣
↪
1              B                   102                    149.56                     73.
↪429314                      6.84                               3                              ␣
↪              34                  0.125525                    42.479989                       ␣
↪     7489.79                                   18                                              ␣
↪        1                                      7                                               ␣
↪         1                                         0                                           ␣
↪              2                                  1994                                          ␣
↪          2014                                        1                                       ␣
↪desktop                                            4                                           ␣
↪               1                                           3                                   ␣
↪                          1                                              4                     ␣
↪                                       1                                                       ␣
↪        5                                         1                                            ␣
↪                    5                                     3
2              B                    92                    149.95                     76.
↪319891                      5.73                              28                              ␣
↪              34                  0.151934                    46.336308                       ␣
↪     7021.43                                   18                                              ␣
↪        1                                      8                                               ␣
↪         1                                         0                                           ␣
↪              2                                  2006                                          ␣
↪          2014                                        4                                       ␣
↪desktop                                            4                                           ␣
↪               1                                           3                                   ␣
↪                          1                                              4                     ␣
↪                                       1                                                       ␣
↪        5                                         1                                            ␣
↪                    5                                     3
3              B                    96                    148.31                     73.
↪001250                      5.89                               1                              ␣
↪              35                  0.223938                    38.871405                       ␣
↪     7008.12                                   18                                              ␣
↪        1                                      8                                               ␣
↪         1                                         0                                           ␣
↪              2                                  2006                                          ␣
↪          2014                                        4                                       ␣
↪desktop                                            4                                           ␣
↪               1                                           3                                   ␣
↪                          1                                              4                     ␣
↪                                       1                                                       ␣
↪        5                                         1                                            ␣
↪                    5                                     3
4              B                   106                    146.46                     76.
↪311038                      5.81                              29                              ␣
↪              34                 -0.132077                    42.492501                       ␣
↪     8088.97                                   18                                              ␣
↪        1                                      7                                               ␣
↪         1                                         0                                           ␣
↪              2                                  1994                                          ␣
↪          2014                                        1                                       ␣
```

(continues on next page)

(continued from previous page)

```
5             A                  104                     149.02                   76.
↪264904                     5.91                                4                 ␣
↪             34                 0.098248                   42.131902             ␣
↪    7931.55                                 18                                   ␣
↪       1                                   7                                     ␣
↪        1                                     0                                  ␣
↪             2                               1994                                ␣
↪         2014                                    1                              ␣
↪mobile                                      4                                    ␣
↪              1                                          3                       ␣
↪                          1                                              4       ␣
↪                                         1                                       ␣
↪      5                                            1                             ␣
↪                  5                                         3
```

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.

Dask and Koalas EntitySets

EntitySets can also be created using Dask dataframes or Koalas dataframes. For more information refer to *Using Dask EntitySets (BETA)* and *Using Koalas EntitySets (BETA)*.

3.2.2 Deep Feature Synthesis

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

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.

```
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
```

(continues on next page)

(continued from previous page)

```
    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`.

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.

```
In [4]: feature_matrix, feature_defs = ft.dfs(entityset=es,
   ...:                                       target_entity="customers",
   ...:                                       agg_primitives=["count"],
   ...:                                       trans_primitives=["month"],
   ...:                                       max_depth=1)
   ...:

In [5]: feature_matrix
Out[5]:
             zip_code  COUNT(sessions)  MONTH(date_of_birth)  MONTH(join_date)
customer_id
5               60091                6                     7                 7
4               60091                8                     8                 4
1               60091                8                     7                 4
3               13244                6                    11                 8
2               13244                7                     8                 4
```

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*.

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
```

(continues on next page)

(continued from previous page)

```
Out[7]:
            zip_code  ... MODE(transactions.sessions.device)
customer_id           ...
5              60091  ...                             mobile
4              60091  ...                             mobile
1              60091  ...                             mobile
3              13244  ...                            desktop
2              13244  ...                            desktop

[5 rows x 17 columns]
```

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))']]
Out[8]:
             MEAN(sessions.SUM(transactions.amount))
customer_id
5                                        1058.276667
4                                        1090.960000
1                                        1128.202500
3                                        1039.436667
2                                        1028.611429
```

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]:
             MODE(sessions.HOUR(session_start))
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.

Note: You can graphically visualize the lineage of a feature by calling *`featuretools.graph_feature()`* on it.

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.MONTH(join_date)
session_id               ...
1                     2  ...                          4
2                     5  ...                          7
3                     4  ...                          4
4                     1  ...                          4
5                     4  ...                          4

[5 rows x 19 columns]
```

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]:
            customers.MEAN(transactions.amount)
session_id
1                                     77.422366
2                                     80.375443
3                                     80.070459
4                                     71.631905
5                                     80.070459
```

Improve feature output

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

3.2.3 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.

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)>,
 <Feature: MEAN(sessions.MEAN(transactions.amount))>,
 <Feature: MEAN(sessions.TIME_SINCE_PREVIOUS(session_start))>]
```

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                                                                        ␣
↪
↪                                                                                  ␣
↪
```

(continues on next page)

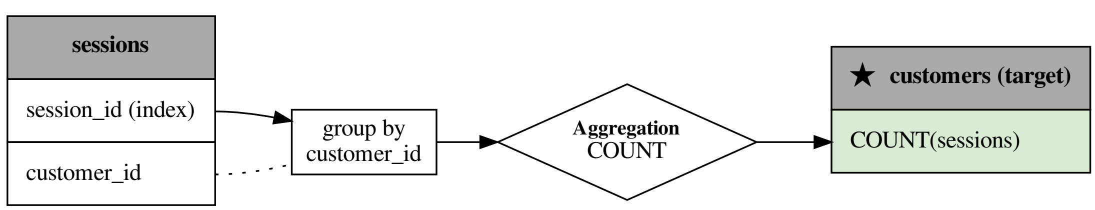
(continued from previous page)

```
5                                                              1007.500000                                                               ␣
↪                    1170.0                                                      715.0       ␣
↪                               157.884451                                                   ␣
↪-1.507217
4                                                               999.375000                   ␣
↪                    1625.0                                                      650.0       ␣
↪                               308.688904                                                   ␣
↪ 1.065177
1                                                               966.875000                   ␣
↪                    1170.0                                                      715.0       ␣
↪                               171.754341                                                   ␣
↪-0.254557
3                                                               888.333333                   ␣
↪                    1170.0                                                      650.0       ␣
↪                               177.613813                                                   ␣
↪ 0.434581
2                                                               725.833333                   ␣
↪                     975.0                                                      520.0       ␣
↪                               194.638554                                                   ␣
↪ 0.162631
```

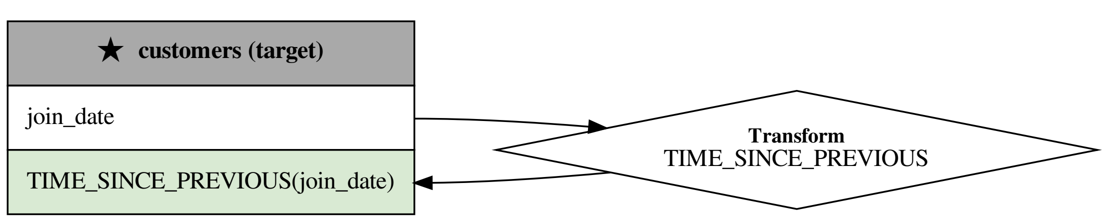
Aggregation vs Transform Primitive

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"`.



The above graphs were generated using the *graph_feature* function. These feature lineage graphs help to visually show how primitives were stacked to generate a feature.

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.

```
In [5]: ft.list_primitives().head(5)
Out[5]:
               name         type  dask_compatible  koalas_compatible                                                
↪                 description
0     n_most_common  aggregation            False              False              
↪Determines the `n` most common elements.
1        num_unique  aggregation             True               True  Determines the 
↪number of distinct values, igno...
2  time_since_first  aggregation            False              False  Calculates the 
↪time elapsed since the first da...
3               min  aggregation             True               True  Calculates the 
↪smallest value, ignoring `NaN` ...
4   time_since_last  aggregation            False              False  Calculates the 
↪time elapsed since the last dat...
```

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 transfered to another.

Simple Custom Primitives

```
In [6]: from featuretools.primitives import make_agg_primitive, make_trans_primitive

In [7]: from featuretools.variable_types import NaturalLanguage, 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*.

```
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.
↪ABSOLUTE(amount))
session_id                                                                    ␣
↪
1                                           146.81                          ␣
↪141.66
2                                           149.02                          ␣
↪135.25
3                                           149.95                          ␣
↪147.73
4                                           139.43                          ␣
↪129.00
5                                           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.

```
In [15]: WordCount = make_trans_primitive(function=word_count,
   ....:                                  input_types=[NaturalLanguage],
   ....:                                  return_type=Numeric)
   ....:
```

```
In [16]: from featuretools.tests.testing_utils import make_ecommerce_entityset

In [17]: es = make_ecommerce_entityset()
```

Since WordCount is a transform primitive, we need to add it to the list of transform primitives DFS can use when generating features.

```
In [18]: feature_matrix, features = ft.dfs(entityset=es,
   ....:                                   target_entity="sessions",
   ....:                                   agg_primitives=["sum", "mean", "std"],
   ....:                                   trans_primitives=[WordCount])
   ....:

In [19]: feature_matrix[["customers.WORD_COUNT(favorite_quote)", "STD(log.WORD_
↪COUNT(comments))", "SUM(log.WORD_COUNT(comments))", "MEAN(log.WORD_COUNT(comments))
↪"]]
Out[19]:
    customers.WORD_COUNT(favorite_quote)  STD(log.WORD_COUNT(comments))  SUM(log.WORD_
↪COUNT(comments))  MEAN(log.WORD_COUNT(comments))
id                                                                                   ␣
↪
0                                      9                     540.436860              ␣
↪           2500                             500
1                                      9                     583.702550              ␣
↪           1732                             433
2                                      9                            NaN              ␣
↪            246                             246
3                                      6                     883.883476              ␣
↪           1256                             628
4                                      6                       0.000000              ␣
↪              9                               3
5                                     12                      19.798990              ␣
↪             68                              34
```

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.

```
In [20]: from featuretools.variable_types import Datetime, Timedelta, Variable

In [21]: import pandas as pd

In [22]: def mean_sunday(numeric, datetime):
   ....:     '''
   ....:     Finds the mean of non-null values of a feature that occurred on Sundays
```

(continues on next page)

(continued from previous page)

```
   ....:     '''
   ....:     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]:
    MEAN_SUNDAY(log.value, datetime)  MEAN_SUNDAY(log.value_2, datetime)
id
0                                NaN                                 NaN
1                                NaN                                 NaN
2                                NaN                                 NaN
3                                2.5                                 1.0
4                                7.0                                 3.0
5                                NaN                                 NaN
```

3.2.4 Variable Types

A Variable is analogous to a column in a table in a relational database. When creating an Entity, Featuretools will attempt to infer the types of variables present. Featuretools also allows for explicitly specifying the variable types when creating the Entity.

It is important that datasets have appropriately defined variable types when using DFS because this will allow the correct primitives to be used to generate new features.

> Note: When using Dask Entities, users must explicitly specify the variable types for all columns in the Entity dataframe.

To understand the different variable types in Featuretools, let's first look at a graph of the variables:

```
[1]: from featuretools.variable_types import graph_variable_types
graph_variable_types()
```

```
[1]:
```

As we can see, there are multiple variable types and some have subclassed variable types. For example, ZIPCode is variable type that is child of Categorical type which is a child of Discrete type.

Let's explore some of the variable types and understand them in detail.

Discrete

A Discrete variable type can only take certain values. It is a type of data that can be counted, but cannot be measured. If it can be classified into distinct buckets, then it a discrete variable type.

There are 2 sub-variable types of Discrete. These are Categorical, and Ordinal. If the data has a certain ordering, it is of Ordinal type. If it cannot be ordered, then is a Categorical type.

Categorical

A Categorical variable type can take unordered discrete values. It is usually a limited, and fixed number of possible values. Categorical variable types can be represented as strings, or integers.

Some examples of Categorical variable types:

- Gender
- Eye Color
- Nationality
- Hair Color
- Spoken Language

Ordinal

A Ordinal variable type can take ordered discrete values. Similar to Categorical, it is usually a limited, and fixed number of possible values. However, these discrete values have a certain order, and the ordering is important to understanding the values. Ordinal variable types can be represented as strings, or integers.

Some examples of Ordinal variable types:

- Educational Background (Elementary, High School, Undergraduate, Graduate)
- Satisfaction Rating ("Not Satisfied", "Satisfied", "Very Satisfied")
- Spicy Level (Hot, Hotter, Hottest)
- Student Grade (A, B, C, D, F)
- Size (small, medium, large)

Categorical SubTypes (CountryCode, Id, SubRegionCode, ZIPCode)

There are also more distinctions within the Categorical variable type. These include CountryCode, Id, SubRegion-Code, and ZIPCode.

It is important to make this distinction because there are certain operations that can be applied, but they don't necessary apply to all Categorical types. For example, there could be a custom primitive that applies to the ZIPCode variable type. It could extract the first 5 digits of a ZIPCode. However, this operation is not valid for all Categorical variable types. Therefore it is approriate to use the ZIPCode variable type.

Datetime

A Datetime is a representation of a date and/or time. Datetime variable types can be represented as strings, or integers. However, they should be in a intrepretable format or properly cast before using DFS.

Some examples of Datetime include:

- transaction time
- flight departure time
- pickup time

DateOfBirth

A more distinct type of datetime is a DateOfBirth. This is an important distinction because it allows additional primitives to be applied to the data to generate new features. For example, having an DateOfBirth variable type, will allow the Age primitive to be applied during DFS, and lead to a new Numeric feature.

Text

Text is a long-form string, that can be of any length. It is commonly used with NLP operations, such as TF-IDF. Featuretools supports NLP operations with the nlp-primitives add-on.

LatLong

A LatLong represents an ordered pair (Latitude, Longitude) that tells the location on Earth. The order of the tuple is important. LatLongs can be represented as tuple of floating point numbers.

To make a LatLong in a dataframe do the following:

[2]:
```
import pandas as pd

data = pd.DataFrame()
data['latitude'] = [51.52, 9.93, 37.38]
data['longitude'] = [-0.17, 76.25, -122.08]
data['latlong'] = data[['latitude', 'longitude']].apply(tuple, axis=1)
data['latlong']
```

[2]:
```
0      (51.52, -0.17)
1       (9.93, 76.25)
2    (37.38, -122.08)
Name: latlong, dtype: object
```

List of Variable Types

We can also get all the variable types as a DataFrame.

[3]:
```
from featuretools.variable_types import list_variable_types
list_variable_types()
```

[3]:
```
                  name          type_string  \
0              Unknown              unknown
1             Discrete             discrete
2          Categorical          categorical
```

(continues on next page)

(continued from previous page)

```
3                  Id                   id
4             ZIPCode             zip_code
5         CountryCode         country_code
6       SubRegionCode      sub_region_code
7             Ordinal              ordinal
8             Boolean              boolean
9             Numeric              numeric
10   NumericTimeIndex   numeric_time_index
11              Index                index
12           Datetime             datetime
13  DatetimeTimeIndex  datetime_time_index
14        DateOfBirth        date_of_birth
15          TimeIndex           time_index
16          Timedelta            timedelta
17    NaturalLanguage     natural_language
18            LatLong             lat_long
19          IPAddress           ip_address
20           FullName            full_name
21       EmailAddress        email_address
22                URL                  url
23        PhoneNumber         phone_number
24           FilePath            file_path

                                          description
0                                                None
1   Superclass representing variables that take on...
2   Represents variables that can take an unordere...
3   Represents variables that identify another entity
4   Represents a postal address in the United Stat...
5   Represents an ISO-3166 standard country code.\...
6   Represents an ISO-3166 standard sub-region cod...
7   Represents variables that take on an ordered d...
8   Represents variables that take on one of two v...
9   Represents variables that contain numeric valu...
10    Represents time index of entity that is numeric
11  Represents variables that uniquely identify an...
12  Represents variables that are points in time\n...
13  Represents time index of entity that is a date...
14           Represents a date of birth as a datetime
15                    Represents time index of entity
16  Represents variables that are timedeltas\n\n  ...
17     Represents variables that are arbitary strings
18  Represents an ordered pair (Latitude, Longitud...
19  Represents a computer network address. Represe...
20  Represents a person's full name. May consist o...
21  Represents an email box to which email message...
22  Represents a valid web url (with or without ht...
23  Represents any valid phone number.\n    Can be...
24  Represents a valid filepath, absolute or relative
```

Defining Custom Variable Types

Users can define their own variable types. For example, to make a custom variable type called Age, run the following code:

[4]:
```
from featuretools.variable_types import Variable

class Age(Variable):
    _default_pandas_dtype = float
```

> The `_default_pandas_dtype` specifies the pandas dtype to use to represent the underlying data. A list of pandas dtypes can be found here.

Age can now be used as a variable type when creating an entity. For example, let's create an entity with a column called `customer_age`.

[5]:
```
import pandas as pd
import featuretools as ft

df = pd.DataFrame({"customer_id": [1, 2, 3, 4, 5],
                   "customer_age": [40, 50, 10, 20, 30]})

es = ft.EntitySet(id="customer_data")
es = es.entity_from_dataframe(entity_id="customers",
                              dataframe=df,
                              index="customer_id",
                              variable_types={
                                  'customer_age': Age
                              })
```

Age can also be used as a variable type to create a custom primitive. For example, let's create a transform primitive that returns a boolean if the age is greater than 100.

[6]:
```
from featuretools.variable_types import Boolean
from featuretools.primitives.base import TransformPrimitive

class AgeOver100(TransformPrimitive):
    name = "age_over_100"
    input_types = [Age]
    return_type = Boolean

    def get_function(self):
        def age_over_100(x):
            return x > 100
        return age_over_100
```

This primitive can now be passed to `ft.dfs` as one of the transform primitives. DFS will generate a feature which uses the custom primitive (**AgeOver100**) with the custom variable type (**Age**).

[7]:
```
feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_entity="customers",
                                      trans_primitives=[AgeOver100])
feature_defs
```

[7]:
```
[<Feature: AGE_OVER_100(customer_age)>]
```

[8]:
```
feature_matrix.head(5)
```

```
[8]:              AGE_OVER_100(customer_age)
customer_id
1                                    False
2                                    False
3                                    False
4                                    False
5                                    False
```

3.2.5 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.

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:

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

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:

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

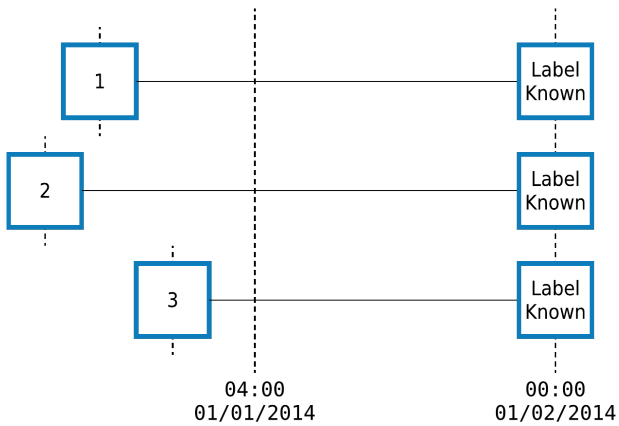
Here, we have two time columns, `join_date` and `date_of_birth`. While either column might be useful for making features, the `join_date` should be used as the time index because it indicates when that customer first became available in the dataset.

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.

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:

```
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],
   ...:                        cutoff_time_in_index=True)
   ...:

In [6]: fm
Out[6]:
                                          zip_code  COUNT(sessions) MODE(sessions.device)   NUM_
↪UNIQUE(sessions.device)  COUNT(transactions)  MAX(transactions.amount) ␣
↪MEAN(transactions.amount)  MIN(transactions.amount)  MODE(transactions.product_id) ␣
↪NUM_UNIQUE(transactions.product_id)  SKEW(transactions.amount)  STD(transactions.
↪amount)  SUM(transactions.amount)  DAY(date_of_birth)  DAY(join_date)  MONTH(date_
↪of_birth)  MONTH(join_date)  WEEKDAY(date_of_birth)  WEEKDAY(join_date)  YEAR(date_
↪of_birth)  YEAR(join_date)  MAX(sessions.COUNT(transactions))  MAX(sessions.
↪MEAN(transactions.amount))  MAX(sessions.MIN(transactions.amount))  MAX(sessions.
↪NUM_UNIQUE(transactions.product_id))  MAX(sessions.SKEW(transactions.amount)) ␣
↪MAX(sessions.STD(transactions.amount))  MAX(sessions.SUM(transactions.amount)) ␣
↪MEAN(sessions.COUNT(transactions))  MEAN(sessions.MAX(transactions.amount)) ␣
↪MEAN(sessions.MEAN(transactions.amount))  MEAN(sessions.MIN(transactions.amount)) ␣
↪MEAN(sessions.NUM_UNIQUE(transactions.product_id))  MEAN(sessions.SKEW(transactions.
↪amount))  MEAN(sessions.STD(transactions.amount))  MEAN(sessions.SUM(transactions.
↪amount))  MIN(sessions.COUNT(transactions))  MIN(sessions.MAX(transactions.amount))␣
↪ MIN(sessions.MEAN(transactions.amount))  MIN(sessions.NUM_UNIQUE(transactions.
↪product_id))  MIN(sessions.SKEW(transactions.amount))  MIN(sessions.
↪STD(transactions.amount))  MIN(sessions.SUM(transactions.amount))  MODE(sessions.
↪DAY(session_start))  MODE(sessions.MODE(transactions.product_id))  MODE(sessions.
↪MONTH(session_start))  MODE(sessions.WEEKDAY(session_start))  MODE(sessions.
↪YEAR(session_start))  NUM_UNIQUE(sessions.DAY(session_start))  NUM_UNIQUE(sessions.
↪MODE(transactions.product_id))  NUM_UNIQUE(sessions.MONTH(session_start))  NUM_
↪UNIQUE(sessions.WEEKDAY(session_start))  NUM_UNIQUE(sessions.YEAR(session_start)) ␣
↪SKEW(sessions.COUNT(transactions))  SKEW(sessions.MAX(transactions.amount)) ␣
↪SKEW(sessions.MEAN(transactions.amount))  SKEW(sessions.MIN(transactions.amount)) ␣
↪SKEW(sessions.NUM_UNIQUE(transactions.product_id))  SKEW(sessions.STD(transactions.
↪amount))  SKEW(sessions.SUM(transactions.amount))  STD(sessions.
```

(continues on next page)

(continued from previous page)

```
customer_id time                                                                                 ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪                                                                                                ␣
↪
1           2014-01-01 04:00:00    60091                4                tablet               ␣
↪                      3                    67                     139.23                     ␣
↪    74.002836                  5.81                                    4                     ␣
↪                      5                     -0.006928                 42.309717              ␣
↪          4958.19                  18              17                        7               ␣
↪        4                      0                     6                 1994                  ␣
↪    2011                            25                                  85.469167␣
↪                               8.74                                                          ␣
↪      5                             0.234349                                   46.
↪905665                              1613.93                                    16.75         ␣
↪                           135.0100                                 76.150425                ␣
↪                            6.905                                                            ␣
↪     5                              -0.126261                                    42.
↪393218                             1239.5475                                     12          ␣
↪                            129.00                                 64.557200                 ␣
↪                                                 5                                 -0.
↪830975                           39.825249                                        1025.
↪63                                1                                                  4␣
↪                                  1                                   2                      ␣
↪                       2014                                      1                           ␣
↪                                 3                                          1                ␣
↪                                      1                                        1             ␣
↪                       1.614843                                  -0.451371                   ␣
↪                     -0.233453                                   1.452325                    ␣
↪                                       0.0                                   1.235445␣
↪                            1.197406                             5.678908                    ␣
↪                      5.027226                                  10.426572                    ␣
↪                  1.285833                                                0.0                ␣
↪                          0.500353                               271.917637                  ␣
↪                       540.04                                  304.601700                    ␣
```

(continues on next page)

(continued from previous page)

```
2           2014-01-01 04:00:00    13244                4                desktop      ␣
↪                    2                   49                    146.81                ␣
↪    84.700000                 12.07                         4                       ␣
↪                    5                -0.134786                 39.289512            ␣
↪          4150.30                18               15                   8            ␣
↪         4                 0                     6                1986              ␣
↪    2012                            16                                 96.581000␣
↪                                  56.46                                            ␣
↪     5                                 0.295458                               47.
↪935920                                 1320.64                              12.25 ␣
↪                              142.3225                               85.197948    ␣
↪                               26.310                                              ␣
↪    5                                   0.011293                                39.
↪315685                                1037.5750                                8 ␣
↪                               138.38                              76.813125      ␣
↪                                                     5                         -0.
↪455197                              27.839228                                 634.
↪84                                   1                                          2␣
↪                                  1                                     2         ␣
↪                     2014                                       1                 ␣
↪                                 3                                         1      ␣
↪                                      1                                        1 ␣
↪                       -0.169238                                 0.459305         ␣
↪                       0.651941                                1.815491           ␣
↪                                    0.0                                 -0.966834␣
↪                             -0.823347                           3.862210         ␣
↪                     3.470527                                 8.983533            ␣
↪                 20.424007                                                  0.0    ␣
↪                          0.324809                             307.743859          ␣
↪                        569.29                              340.791792            ␣
↪                     105.24                                                   20   ␣
↪                           0.045171                              157.262738       ␣
↪                                2                          desktop                ␣
↪                              1                                           2       ␣
3           2014-01-01 04:00:00    13244                1                 tablet      ␣
↪                    1                   15                    146.31                ␣
↪    62.791333                  8.19                         1                       ␣
↪                    5                   0.618455               47.264797            ␣
↪           941.87                21               13                  11            ␣
↪         8                 4                     5                2003              ␣
↪    2011                            15                                 62.791333␣
↪                                   8.19                                            ␣
↪     5                                 0.618455                               47.
↪264797                                  941.87                              15.00 ␣
↪                              146.3100                               62.791333    ␣
↪                                8.190                                              ␣
↪    5                                   0.618455                                47.
↪264797                                 941.8700                                15 ␣
↪                               146.31                              62.791333      ␣
↪                                                     5                          0.
↪618455                              47.264797                                 941.
↪87                                   1                                          1␣
↪                                  1                                     2         ␣
↪                     2014                                       1                 ␣
↪                                 1                                         1      ␣
↪                                      1                                        1 ␣
↪                              NaN                                     NaN          ␣
↪                             NaN                                     NaN           ␣
↪                                     NaN                                     NaN␣
↪                                    NaN                                 NaN        ␣
↪                         NaN                                       NaN            ␣
↪                      NaN                                                 NaN      ␣
↪                              NaN                                    NaN           ␣
↪                        146.31                                 62.791333          ␣
```

(continues on next page)

(continued from previous page)

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 which includes one column for the instance id and another column for the corresponding cutoff time. These columns can be in any order, but they must be named properly. The column with the instance ids must either be named `instance_id` or have the same name as the target entity `index`. The column with the cutoff time values must either be named `time` or have the same name as the target entity `time_index`.

The column names for the instance ids and the cutoff time values should be unambiguous. Passing a dataframe that contains both a column with the same name as the target entity `index` and a column named `instance_id` will result in an error. Similarly, if the cutoff time dataframe contains both a column with the same name as the target entity `time_index` and a column named `time` an error will be raised.

Note: Only the columns corresponding to the instance ids and the cutoff times 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.

```
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 05:00',
   ...:                                      '2014-1-1 06:00',
   ...:                                      '2014-1-1 08:00'])
   ...:

In [10]: cutoff_times['label'] = [True, True, False, True]

In [11]: cutoff_times
Out[11]:
   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

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(sessions) MODE(sessions.device)  NUM_
↪UNIQUE(sessions.device)  COUNT(transactions)  MAX(transactions.amount) ␣
↪MEAN(transactions.amount)  MIN(transactions.amount)  MODE(transactions.product_id) ␣
↪NUM_UNIQUE(transactions.product_id)  SKEW(transactions.amount)  STD(transactions.
↪amount)  SUM(transactions.amount)  DAY(date_of_birth)  DAY(join_date)  MONTH(date_
↪of_birth)  MONTH(join_date)  WEEKDAY(date_of_birth)  WEEKDAY(join_date)  YEAR(date_
↪of_birth)  YEAR(join_date)  MAX(sessions.COUNT(transactions))  MAX(sessions.
↪MEAN(transactions.amount))  MAX(sessions.MIN(transactions.amount))  MAX(sessions.
↪NUM_UNIQUE(transactions.product_id))  MAX(sessions.SKEW(transactions.amount)) ␣
↪MAX(sessions.STD(transactions.amount))  MAX(sessions.SUM(transactions.amount)) ␣
```

(continues on next page)

(continued from previous page)

```
customer_id time                                                                   ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪                                                                                  ␣
↪
1           2014-01-01 04:00:00    60091                4                tablet    ␣
↪                      3                   67                    139.23            ␣
↪   74.002836                    5.81                                 4            ␣
↪                      5                    -0.006928                42.309717     ␣
↪          4958.19                18               17                      7       ␣
↪        4                      0                   6                 1994         ␣
↪   2011                                  25                          85.469167␣
↪                                 8.74                                             ␣
↪     5                                   0.234349                          46.
↪905665                                   1613.93                            16.75 ␣
↪                           135.01000                              76.150425       ␣
↪                            6.90500                                               ␣
↪    5                                    -0.126261                           42.
↪393218                                  1239.5475                            12 ␣
↪                             129.00                             64.557200         ␣
↪                                                5                           -0.
↪830975                                 39.825249                            1025.
↪63                                1                                          4␣
↪                                 1                                     2          ␣
↪                       2014                                        1              ␣
↪                                3                                        1        ␣
↪                                     1                                      1 ␣
↪                         1.614843                                -0.451371        ␣
↪                       -0.233453                                 1.452325         ␣
↪                                      0.0                                1.235445␣
↪                             1.197406                            5.678908         ␣
↪                       5.027226                              10.426572            ␣
↪                     1.285833                                           0.0       ␣
↪                            0.500353                             271.917637       ␣
↪                         540.04                              304.601700           ␣
```

(continues on next page)

(continued from previous page)

```
2          2014-01-01 05:00:00    13244                 5               desktop   
↪                 2                   62                    146.81              
↪    83.149355                 12.07                    4                       
↪                 5                  -0.121811              38.047944           
↪          5155.26               18              15                8            
↪        4                     0                   6                1986        
↪    2012                           16                             96.581000    
↪                                  56.46                                        
↪      5                           0.295458                              47.    
↪935920                             1320.64                               12.40 
↪                               137.62800                          83.619281    
↪                               25.41200                                        
↪     5                            -0.053949                               38.  
↪197555                            1031.0520                                 8  
↪                                 118.85                             76.813125  
↪                                                      5                     -0.
↪455197                               27.839228                             634.
↪84                                    1                                      2 
↪                                     1                                2        
↪                       2014                                    1               
↪                                    4                                     1    
↪                                         1                                  1  
↪                         -0.379092                              -1.814717      
↪                         1.082192                               1.959531       
↪                                          0.0                           -0.213518
↪                               -0.667256                          3.361547     
↪                      10.919023                                8.543351        
↪                   17.801322                                          0.0      
↪                             0.316873                           266.912832     
↪                           688.14                             418.096407       
↪                        127.06                                          25     
↪                            -0.269747                          190.987775      
↪                                  2                          desktop           
↪                                1                                    2   True
3          2014-01-01 06:00:00    13244                 4               desktop   
↪                 2                   44                    146.31              
↪    65.174773                  6.65                    1                       
↪                 5                   0.318315              40.349758           
↪          2867.69               21              13                11           
↪        8                     4                   5                2003        
↪    2011                           17                             91.760000    
↪                                  91.76                                        
↪      5                           0.618455                              47.    
↪264797                              944.85                               11.00 
↪                               123.26750                          72.742004    
↪                               31.66500                                        
↪     4                             0.286859                               39.  
↪712232                            716.9225                                  1  
↪                                 91.76                              55.579412  
↪                                                      1                     -0.
↪289466                               35.704680                              91.
↪76                                    1                                      1 
↪                                     1                                2        
↪                       2014                                    1               
↪                                    2                                     1    
↪                                         1                                  1  
↪                         -1.330938                              -1.060639      
↪                         0.201588                               1.874170       
↪                                    -2.0                               1.722323
↪                               -1.977878                          7.118052     
↪                      22.808351                                16.540757       
↪                   40.508892                                          2.0      
↪                             0.500999                           417.557763     
↪                           493.07                             290.968018       
```

(continues on next page)

(continued from previous page)

```
1          2014-01-01 08:00:00    60091               8                mobile    ␣
↪                   3                  126                   139.43            ␣
↪   71.631905                   5.81                          4                ␣
↪                   5                  0.019698               40.442059        ␣
↪          9025.62                18              17                  7        ␣
↪        4                  0                   6               1994           ␣
↪   2011                             25                           88.755625␣
↪                               26.36                                          ␣
↪     5                              0.640252                            46.
↪905665                              1613.93                               15.75 ␣
↪                          132.24625                          72.774140        ␣
↪                           9.82375                                            ␣
↪    5                               -0.059515                              39.
↪093244                             1128.2025                              12 ␣
↪                            118.90                           50.623125        ␣
↪                                                   5                        -1.
↪038434                            30.450261                                809.
↪97                                  1                                        4␣
↪                                  1                               2           ␣
↪                     2014                                    1                ␣
↪                                 4                                      1     ␣
↪                                      1                                    1  ␣
↪                        1.946018                          -0.780493           ␣
↪                      -0.424949                           2.440005            ␣
↪                                    0.0                           -0.312355␣
↪                            0.778170                       4.062019           ␣
↪                     7.322191                         13.759314               ␣
↪                  6.954507                                         0.0        ␣
↪                          0.589386                       279.510713           ␣
↪                       1057.97                        582.193117              ␣
↪                      78.59                                               40  ␣
↪                          -0.476122                        312.745952         ␣
↪                                1                     mobile                  ␣
↪                              1                                      3   True
```

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`.

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:

```
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) MODE(sessions.device)  NUM_
```

(continues on next page)

```
↪UNIQUE(sessions.device)  COUNT(transactions)  MAX(transactions.amount)  ␣
↪MEAN(transactions.amount)  MIN(transactions.amount)  MODE(transactions.product_id)  ␣
↪NUM_UNIQUE(transactions.product_id)  SKEW(transactions.amount)  STD(transactions.
↪amount)  SUM(transactions.amount)  DAY(date_of_birth)  DAY(join_date)  MONTH(date_
↪of_birth)  MONTH(join_date)  WEEKDAY(date_of_birth)  WEEKDAY(join_date)  YEAR(date_
↪of_birth)  YEAR(join_date)  MAX(sessions.COUNT(transactions))  MAX(sessions.
↪MEAN(transactions.amount))  MAX(sessions.MIN(transactions.amount))  MAX(sessions.
```

(continued from previous page)

```
customer_id time
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
1           2014-01-01 04:00:00    60091                2                desktop
↪                    2                   27                    139.09
↪    76.950370                 5.81                              4
↪                    5                 -0.187686                 43.772157
↪          2077.66               18               17                       7
↪        4                0                  6               1994
↪    2011                        15                                85.469167
↪                               6.78
↪      5                           0.226337                                46.
↪905665                          1052.03                         13.500000
↪                         135.905000                           77.802250
↪                          6.295000                                     5.
↪000000                            -0.302319                             43.
↪365457                       1038.830000                              12
↪                            132.72                          70.135333
↪                                             5                      -0.
↪830975                      39.825249                               1025.
↪63                             1                                        1
↪                              1                              2
↪                    2014                               1
↪                              2                                     1
↪                                  1                                    1
↪                          NaN                                 NaN
↪                         NaN                                 NaN
↪                                NaN                                   NaN
↪                               NaN                        2.121320
↪                   4.504270                         10.842650
↪                 0.685894                                  0.000000
↪                        0.747633                       18.667619
↪                      271.81                       155.604500
```

(continues on next page)

(continued from previous page)

```
2          2014-01-01 05:00:00    13244                   3               desktop     ␣
↪                  2                   31                     146.81              ␣
↪   84.051935                    12.07                           4                ␣
↪                  5                      -0.198611               36.077146       ␣
↪          2605.61                  18               15                 8         ␣
↪        4                       0                    6               1986        ␣
↪   2012                                  13                              96.581000␣
↪                                  56.46                                          ␣
↪     5                                 0.130019                            47.
↪935920                                 1004.96                          10.333333 ␣
↪                           134.680000                              84.413538     ␣
↪                           30.116667                                         5.
↪000000                                   -0.036670                           36.
↪500062                               868.536667                              8 ␣
↪                              118.85                                77.304615     ␣
↪                                                    5                      -0.
↪314918                               27.839228                              634.
↪84                                 1                                           1␣
↪                                  1                                      2       ␣
↪                      2014                                        1              ␣
↪                                 3                                          1     ␣
↪                                      1                                       1 ␣
↪                         0.585583                                -1.083626        ␣
↪                        1.659252                                 1.397956        ␣
↪                                0.000000                                 1.121470␣
↪                              -1.660092                           2.516611       ␣
↪                     14.342521                               10.587085           ␣
↪                  23.329038                                          0.000000    ␣
↪                           0.242542                              203.331699      ␣
↪                         404.04                              253.240615          ␣
↪                       90.35                                                15    ␣
↪                           -0.110009                               109.500185    ␣
↪                               2                                desktop          ␣
↪                             1                                           2   True
3          2014-01-01 06:00:00    13244                   3               desktop     ␣
↪                  1                   29                     128.26              ␣
↪   66.407586                     6.65                           1                ␣
↪                  5                      0.110145                37.130891       ␣
↪          1925.82                  21               13                 11        ␣
↪        8                       4                    5               2003        ␣
↪   2011                                  17                              91.760000␣
↪                                  91.76                                          ␣
↪     5                                 0.531588                            36.
↪167220                                   944.85                            9.666667 ␣
↪                           115.586667                              76.058895     ␣
↪                           39.490000                                         3.
↪666667                                   0.121061                            35.
↪935950                               641.940000                              1 ␣
↪                              91.76                                 55.579412     ␣
↪                                                    1                      -0.
↪289466                               35.704680                              91.
↪76                                 1                                           1␣
↪                                  1                                      2       ␣
↪                      2014                                        1              ␣
↪                                 2                                          1     ␣
↪                                      1                                       1 ␣
↪                        -0.722109                                -1.721498        ␣
↪                       -1.081879                                 1.566225        ␣
↪                                      -1.732051                               NaN␣
↪                              -1.705607                           8.082904       ␣
↪                     20.648490                               18.557570           ␣
↪                  45.761028                                          2.309401    ␣
↪                           0.580573                              477.281339      ␣
↪                         346.76                              228.176684          ␣
```

(continues on next page)

(continued from previous page)

```
1          2014-01-01 08:00:00    60091               3                mobile     ␣
↪                     2                   47                     139.43           ␣
↪    66.471277                 5.91                              4               ␣
↪                     5                   0.047120                  38.952172     ␣
↪          3124.15                 18                17                     7     ␣
↪        4                    0                   6                   1994         ␣
↪    2011                          16                                 88.755625␣
↪                                11.62                                          ␣
↪     5                               0.640252                              44.
↪354104                               1420.09                           15.666667 ␣
↪                          128.146667                                 66.328250 ␣
↪                           8.203333                                          5.
↪000000                                 -0.001146                             35.
↪709633                            1041.383333                                15 ␣
↪                            118.90                                 50.623125    ␣
↪                                             5                             -1.
↪038434                            30.450261                                 809.
↪97                                  1                                          1␣
↪                                   1                                2            ␣
↪                      2014                                     1                ␣
↪                                  3                                      1      ␣
↪                                       1                                    1 ␣
↪                       -1.732051                               0.846298          ␣
↪                       1.344879                               1.443486           ␣
↪                                 0.000000                                1.612576␣
↪                                1.606791                          0.577350       ␣
↪                    10.415432                            19.935229                ␣
↪                  3.016195                                    0.000000         ␣
↪                         0.906666                          330.655558            ␣
↪                       384.44                            198.984750              ␣
↪                     24.61                                                  15  ␣
↪                          -0.003438                           107.128899          ␣
↪                              1                           mobile                  ␣
↪                            1                                      2    True
```

We can see that that the counts for the same feature are lower after we shorten the training window:

```
In [16]: fm[["COUNT(transactions)"]]
Out[16]:
                                  COUNT(transactions)
customer_id 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

In [17]: window_fm[["COUNT(transactions)"]]
Out[17]:
                                  COUNT(transactions)
customer_id 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
```

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:

```
In [18]: es['sessions'].df['session_start'].head()
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
Name: session_start, dtype: datetime64[ns]

In [19]: es['sessions'].last_time_index.head()
Out[19]:
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
Name: last_time, dtype: datetime64[ns]
```

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.

Excluding data at cutoff times

The `cutoff_time` is the last point in time where data can be used for feature calculation. If you don't want to use the data at the cutoff time in feature calculation, you can exclude that data by setting `include_cutoff_time` to `False` in *`featuretools.dfs()`* or:func:*featuretools.calculate_feature_matrix*. If you set it to `True` (the default behavior), data from the cutoff time point will be used.

Setting `include_cutoff_time` to `False` also impacts how data at the edges of training windows are included or excluded. Take this slice of data as an example:

```
In [20]: df = es['transactions'].df

In [21]: df[df["session_id"] == 1].head()
Out[21]:
     transaction_id  session_id    transaction_time  amount  product_id
298             298           1 2014-01-01 00:00:00  127.64           5
2                 2           1 2014-01-01 00:01:05  109.48           2
308             308           1 2014-01-01 00:02:10   95.06           3
116             116           1 2014-01-01 00:03:15   78.92           4
371             371           1 2014-01-01 00:04:20   31.54           3
```

Looking at the data, transactions occur every 65 seconds. To check how `include_cutoff_time` effects training windows, we can calculate features at the time of a transaction while using a 65 second training window. This creates a training window with a transaction at both endpoints of the window. For this example, we'll find the sum of all transactions for session id 1 that are in the training window.

```
In [22]: from featuretools.primitives import Sum

In [23]: sum_log = ft.Feature(
   ....:     base=es['transactions']['amount'],
   ....:     parent_entity=es['sessions'],
   ....:     primitive=Sum,
   ....: )
   ....:

In [24]: cutoff_time = pd.DataFrame({
   ....:     'session_id': [1],
   ....:     'time': ['2014-01-01 00:04:20'],
   ....: }).astype({'time': 'datetime64[ns]'})
   ....:
```

With `include_cutoff_time=True`, the oldest point in the training window (`2014-01-01 00:03:15`) is excluded and the cutoff time point is included. This means only transaction 371 is in the training window, so the sum of all transaction amounts is 31.54

```
# Case1. include_cutoff_time = True
In [25]: actual = ft.calculate_feature_matrix(
   ....:     features=[sum_log],
   ....:     entityset=es,
   ....:     cutoff_time=cutoff_time,
   ....:     cutoff_time_in_index=True,
   ....:     training_window='65 seconds',
   ....:     include_cutoff_time=True,
   ....: )
   ....:

In [26]: actual
Out[26]:
                                SUM(transactions.amount)
session_id time
1          2014-01-01 00:04:20                     31.54
```

Whereas with `include_cutoff_time=False`, the oldest point in the window is included and the cutoff time point is excluded. So in this case transaction 116 is included and transaction 371 is exluded, and the sum is 78.92

```
# Case2. include_cutoff_time = False
In [27]: actual = ft.calculate_feature_matrix(
   ....:     features=[sum_log],
   ....:     entityset=es,
   ....:     cutoff_time=cutoff_time,
   ....:     cutoff_time_in_index=True,
   ....:     training_window='65 seconds',
   ....:     include_cutoff_time=False,
   ....: )
   ....:

In [28]: actual
Out[28]:
```

(continues on next page)

(continued from previous page)

```
                                         SUM(transactions.amount)
session_id time
1          2014-01-01 00:04:20                              78.92
```

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:

```
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.

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.

```
In [29]: es_flight = ft.demo.load_flight(nrows=100)
Downloading data ...

In [30]: es_flight
Out[30]:
Entityset: Flight Data
```

(continues on next page)

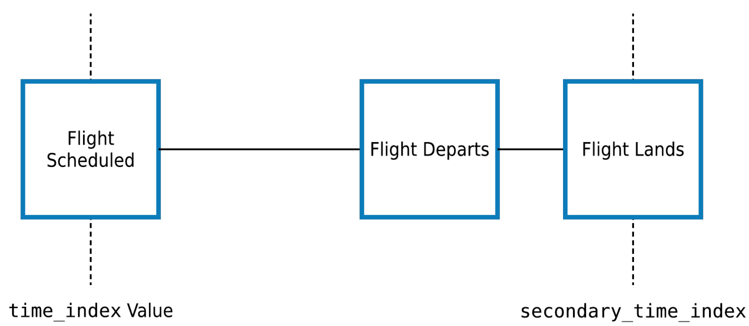
(continued from previous page)

```
  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 [31]: es_flight['trip_logs'].df.head(3)
Out[31]:
    trip_log_id        flight_id date_scheduled  scheduled_dep_time  scheduled_arr_
↪time              dep_time            arr_time  dep_delay  taxi_out  taxi_in  arr_
↪delay  scheduled_elapsed_time  air_time  distance  carrier_delay  weather_delay ␣
↪national_airspace_delay  security_delay  late_aircraft_delay  canceled  diverted
30           30  AA-494:RSW->CLT     2016-09-03 2017-01-01 13:14:00 2017-01-01 15:05:
↪00 2017-01-01 13:03:00 2017-01-01 14:53:00      -11.0      12.0      10.0      -12.0␣
↪          6660000000000      88.0     600.0            0.0            0.0           ␣
↪           0.0              0.0                    0.0       0.0       0.0
38           38  AA-495:ATL->PHX     2016-09-03 2017-01-01 11:30:00 2017-01-01 15:40:
↪00 2017-01-01 11:24:00 2017-01-01 15:41:00       -6.0      28.0       5.0        1.0␣
↪         15000000000000     224.0    1587.0            0.0            0.0           ␣
↪           0.0              0.0                    0.0       0.0       0.0
46           46  AA-495:CLT->ATL     2016-09-03 2017-01-01 09:25:00 2017-01-01 10:42:
↪00 2017-01-01 09:23:00 2017-01-01 10:39:00       -2.0      18.0       8.0       -3.0␣
↪          4620000000000      50.0     226.0            0.0            0.0           ␣
↪           0.0              0.0                    0.0       0.0       0.0
```

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:

```
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',
                    'taxi_in', 'taxi_out', 'air_time', 'dep_time']
```

(continues on next page)

(continued from previous page)

```
es.entity_from_dataframe('trip_logs',
                         data,
                         index='trip_log_id',
                         make_index=True,
                         time_index='date_scheduled',
                         secondary_time_index={'arr_time': arr_time_columns})
```

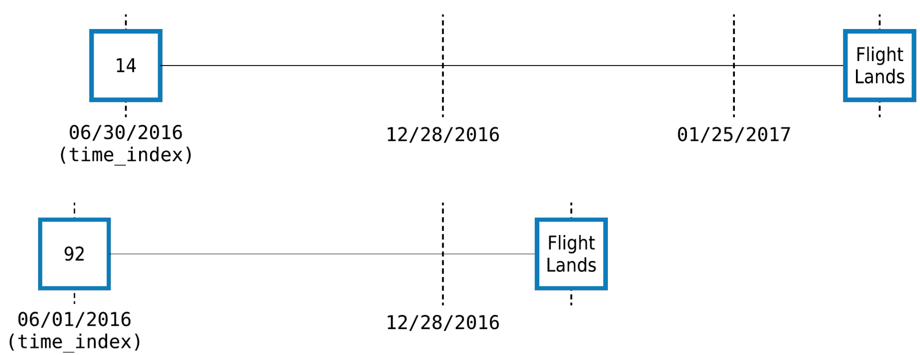
By setting a secondary time index, we can still use the delay information from a row, but only when it becomes known.

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.



Our cutoff time dataframe looks like this:

```
In [32]: ct_flight = pd.DataFrame()

In [33]: ct_flight['trip_log_id'] = [14, 14, 92]

In [34]: ct_flight['time'] = pd.to_datetime(['2016-12-28',
   ....:                                     '2017-1-25',
   ....:                                     '2016-12-28'])
   ....:

In [35]: ct_flight['label'] = [True, True, False]

In [36]: ct_flight
Out[36]:
   trip_log_id       time  label
0           14 2016-12-28   True
```

(continues on next page)

(continued from previous page)

```
1            14 2017-01-25   True
2            92 2016-12-28  False
```

Now, let's calculate the feature matrix:

```
In [37]: 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 [38]: fm[['flight_id', 'label', 'flights.MAX(trip_logs.arr_delay)',
↪'MONTH(scheduled_dep_time)']]
Out[38]:
                              flight_id  label  flights.MAX(trip_logs.arr_delay) ␣
↪MONTH(scheduled_dep_time)
trip_log_id time                                                                  ␣
↪
14          2016-12-28  AA-494:CLT->PHX   True                               NaN  ␣
↪                    1
            2017-01-25  AA-494:CLT->PHX   True                              33.0  ␣
↪                    1
92          2016-12-28  AA-496:PIT->DFW  False                               NaN  ␣
↪                    1
```

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.

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 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 [39]: cutoff_times
Out[39]:
   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 [40]: temporal_cutoffs = ft.make_temporal_cutoffs(cutoff_times['customer_id'],
   ....:                                              cutoff_times['time'],
   ....:                                              window_size='1h',
   ....:                                              num_windows=2)
   ....:

In [41]: temporal_cutoffs
Out[41]:
                 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 [42]: fm, features = ft.dfs(entityset=es,
   ....:                       target_entity='customers',
   ....:                       cutoff_time=temporal_cutoffs,
   ....:                       cutoff_time_in_index=True)
   ....:

In [43]: fm
Out[43]:
                                zip_code  COUNT(sessions) MODE(sessions.device)  NUM_
↪UNIQUE(sessions.device)  COUNT(transactions)  MAX(transactions.amount) ␣
↪MEAN(transactions.amount)  MIN(transactions.amount)  MODE(transactions.product_id) ␣
↪NUM_UNIQUE(transactions.product_id)  SKEW(transactions.amount)  STD(transactions.
↪amount)  SUM(transactions.amount)  DAY(date_of_birth)  DAY(join_date)  MONTH(date_
↪of_birth)  MONTH(join_date)  WEEKDAY(date_of_birth)  WEEKDAY(join_date)  YEAR(date_
↪of_birth)  YEAR(join_date)  MAX(sessions.COUNT(transactions))  MAX(sessions.
↪MEAN(transactions.amount))  MAX(sessions.MIN(transactions.amount))  MAX(sessions.
↪NUM_UNIQUE(transactions.product_id))  MAX(sessions.SKEW(transactions.amount)) ␣
↪MAX(sessions.STD(transactions.amount))  MAX(sessions.SUM(transactions.amount)) ␣
↪MEAN(sessions.COUNT(transactions))  MEAN(sessions.MAX(transactions.amount)) ␣
↪MEAN(sessions.MEAN(transactions.amount))  MEAN(sessions.MIN(transactions.amount)) ␣
↪MEAN(sessions.NUM_UNIQUE(transactions.product_id))  MEAN(sessions.SKEW(transactions.
↪amount))  MEAN(sessions.STD(transactions.amount))  MEAN(sessions.SUM(transactions.
↪amount))  MIN(sessions.COUNT(transactions))  MIN(sessions.MAX(transactions.amount))␣
↪ MIN(sessions.MEAN(transactions.amount))  MIN(sessions.NUM_UNIQUE(transactions.
↪product_id))  MIN(sessions.SKEW(transactions.amount))  MIN(sessions.
↪STD(transactions.amount))  MIN(sessions.SUM(transactions.amount))  MODE(sessions.
↪DAY(session_start))  MODE(sessions.MODE(transactions.product_id))  MODE(sessions.
↪MONTH(session_start))  MODE(sessions.WEEKDAY(session_start))  MODE(sessions.
↪YEAR(session_start))  NUM_UNIQUE(sessions.DAY(session_start))  NUM_UNIQUE(sessions.
```

(continues on next page)

(continued from previous page)

```
customer_id time
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
1           2014-01-01 03:00:00    60091                3                desktop
↪                     3                   55                     139.23
↪    71.501091                5.81                                  1
↪                     5                   0.140387                  42.769602
↪           3932.56                18               17                        7
↪         4                  0                     6                  1994
↪    2011                         25                                   84.440000
↪                                 8.74
↪      5                             0.234349                              46.
↪905665                              1613.93                             18.333333
↪                        133.650000                               73.044178
↪                         6.946667
↪     5                               0.108644                              43.
↪249208                           1310.853333                               15
↪                           129.00                                  64.557200
↪                                                  5                        -0.
↪134754                            40.187205                                 1052.
↪03                                  1                                        1
↪                                   1                                    2
↪                      2014                                       1
↪                                  3                                         1
↪                                       1                                       1
↪                        1.732051                                0.782152
↪                       1.173675                                1.552640
↪                                             0.0                              0.763052
↪                           0.685199                               5.773503
↪                      5.178021                                10.255607
↪                 1.571507                                          0.0
↪                         0.210827                                283.551883
↪                        400.95                                219.132533
```

(continues on next page)

(continued from previous page)

```
          2014-01-01 04:00:00    60091              4                tablet
↪                   3                  67                    139.23
↪    74.002836                  5.81                           4
↪                   5                -0.006928               42.309717
↪          4958.19                 18              17                7
↪        4                   0                   6                 1994
↪    2011                          25                              85.469167
↪                               8.74
↪      5                          0.234349                            46.
↪905665                           1613.93                         16.750000
↪                         135.010000                             76.150425
↪                          6.905000
↪     5                           -0.126261                           42.
↪393218                        1239.547500                            12
↪                            129.00                            64.557200
↪                                                5                      -0.
↪830975                          39.825249                           1025.
↪63                               1                                     4
↪                                1                                2
↪                     2014                                  1
↪                               3                                     1
↪                                   1                                    1
↪                        1.614843                         -0.451371
↪                      -0.233453                          1.452325
↪                                  0.0                                1.235445
↪                            1.197406                      5.678908
↪                     5.027226                         10.426572
↪                  1.285833                                         0.0
↪                          0.500353                        271.917637
↪                        540.04                         304.601700
↪                      27.62                                        20
↪                          -0.505043                         169.572874
↪                             1                         tablet
↪                           1                                     3
2         2014-01-01 04:00:00    13244              4               desktop
↪                   2                  49                    146.81
↪    84.700000                 12.07                           4
↪                   5                -0.134786               39.289512
↪          4150.30                 18              15                8
↪        4                   0                   6                 1986
↪    2012                          16                              96.581000
↪                               56.46
↪      5                          0.295458                            47.
↪935920                           1320.64                         12.250000
↪                         142.322500                             85.197948
↪                         26.310000
↪     5                            0.011293                           39.
↪315685                        1037.575000                             8
↪                            138.38                            76.813125
↪                                                5                      -0.
↪455197                          27.839228                            634.
↪84                               1                                     2
↪                                1                                2
↪                     2014                                  1
↪                               3                                     1
↪                                   1                                    1
↪                       -0.169238                           0.459305
↪                       0.651941                          1.815491
↪                                  0.0                                -0.966834
↪                           -0.823347                          3.862210
↪                     3.470527                            8.983533
↪                   20.424007                                        0.0
↪                          0.324809                         307.743859
↪                        569.29                          340.791792
```

(continues on next page)

(continued from previous page)

```
        2014-01-01 05:00:00    13244              5              desktop     ␣
↪              2                   62                    146.81              ␣
↪  83.149355                 12.07                      4                    ␣
↪              5                  -0.121811               38.047944          ␣
↪         5155.26               18              15                8          ␣
↪       4                  0                  6               1986           ␣
↪  2012                            16                                 96.581000␣
↪                            56.46                                           ␣
↪    5                           0.295458                                 47.
↪935920                          1320.64                          12.400000 ␣
↪                     137.628000                              83.619281     ␣
↪                     25.412000                                             ␣
↪   5                            -0.053949                                38.
↪197555                       1031.052000                                  8 ␣
↪                        118.85                               76.813125      ␣
↪                                      5                                  -0.
↪455197                         27.839228                                634.
↪84                                 1                                       2␣
↪                                  1                                 2       ␣
↪                   2014                                       1             ␣
↪                                 4                                    1     ␣
↪                                      1                                  1 ␣
↪                     -0.379092                                -1.814717     ␣
↪                     1.082192                                 1.959531      ␣
↪                                       0.0                        -0.213518␣
↪                           -0.667256                            3.361547    ␣
↪                   10.919023                                 8.543351       ␣
↪                 17.801322                                              0.0 ␣
↪                        0.316873                              266.912832    ␣
↪                      688.14                                418.096407      ␣
↪                   127.06                                               25  ␣
↪                        -0.269747                              190.987775   ␣
↪                             2                              desktop         ␣
↪                           1                                         2      ␣
3          2014-01-01 05:00:00    13244              2              desktop     ␣
↪              2                   32                    146.31              ␣
↪  58.960000                   6.65                     1                    ␣
↪              5                    0.637074              41.199361          ␣
↪         1886.72               21              13                11         ␣
↪       8                  4                  5               2003           ␣
↪  2011                            17                                 62.791333␣
↪                              8.19                                          ␣
↪    5                           0.618455                                 47.
↪264797                            944.85                          16.000000 ␣
↪                     136.525000                              59.185373     ␣
↪                       7.420000                                            ␣
↪   5                              0.575022                              41.
↪716008                         943.360000                                15 ␣
↪                        126.74                               55.579412      ␣
↪                                      5                                   0.
↪531588                         36.167220                                941.
↪87                                 1                                       1␣
↪                                  1                                 2       ␣
↪                   2014                                       1             ␣
↪                                 1                                    1     ␣
↪                                      1                                  1 ␣
↪                           NaN                                  NaN         ␣
↪                         NaN                                  NaN           ␣
↪                                       NaN                                NaN␣
↪                                     NaN                      1.414214      ␣
↪                   13.838080                                 3.099599       ␣
↪                 1.088944                                               0.0 ␣
↪                        0.061424                                2.107178    ␣
↪                      273.05                                118.370745      ␣
```

(continues on next page)

(continued from previous page)

```
          2014-01-01 06:00:00    13244              4               desktop      ␣
↪                    2                    44                     146.31          ␣
↪    65.174773                  6.65                                 1           ␣
↪                    5                     0.318315                 40.349758    ␣
↪          2867.69                  21               13                  11      ␣
↪        8                   4                    5                 2003         ␣
↪    2011                         17                                    91.760000␣
↪                               91.76                                            ␣
↪     5                             0.618455                                47.
↪264797                           944.85                               11.000000 ␣
↪                          123.267500                             72.742004      ␣
↪                          31.665000                                             ␣
↪    4                            0.286859                                  39.
↪712232                        716.922500                                     1 ␣
↪                              91.76                            55.579412       ␣
↪                                        1                              -0.
↪289466                        35.704680                                    91.
↪76                                 1                                        1␣
↪                                  1                                    2        ␣
↪                     2014                                      1                ␣
↪                                 2                                       1      ␣
↪                                      1                                     1  ␣
↪                       -1.330938                               -1.060639       ␣
↪                       0.201588                                1.874170        ␣
↪                                   -2.0                                1.722323␣
↪                             -1.977878                          7.118052        ␣
↪                     22.808351                              16.540737           ␣
↪                  40.508892                                           2.0       ␣
↪                            0.500999                           417.557763       ␣
↪                          493.07                             290.968018         ␣
↪                       126.66                                          16       ␣
↪                             0.860577                           119.136697      ␣
↪                                3                            desktop            ␣
↪                               1                                       2        ␣
1           2014-01-01 07:00:00    60091                7                tablet  ␣
↪                    3                   110                    139.43           ␣
↪    69.141182                  5.81                                 4           ␣
↪                    5                     0.149908                 41.018896    ␣
↪          7605.53                   18              17                   7      ␣
↪        4                   0                    6                 1994         ␣
↪    2011                         25                                    85.469167␣
↪                               26.36                                            ␣
↪     5                             0.640252                                46.
↪905665                         1613.93                                15.714286 ␣
↪                          133.122857                             70.491070      ␣
↪                           9.567143                                             ␣
↪    5                            0.080330                                  40.
↪060203                      1086.504286                                     12 ␣
↪                             118.90                            50.623125        ␣
↪                                        5                              -0.
↪830975                        30.450261                                   809.
↪97                                 1                                        1␣
↪                                  1                                    2        ␣
↪                     2014                                      1                ␣
↪                                 4                                       1      ␣
↪                                      1                                     1  ␣
↪                         1.927658                              -1.277394       ␣
↪                      -0.282093                                2.551326         ␣
↪                                          0.0                          -0.755846␣
↪                             1.377768                            4.386125       ␣
↪                     7.441648                               13.123365           ␣
↪                   7.470707                                            0.0      ␣
↪                            0.471955                           273.713405       ␣
↪                          931.86                             493.437492         ␣
```

(continues on next page)

(continued from previous page)

```
           2014-01-01 08:00:00    60091                8                mobile     ␣
↪                 3                  126                    139.43                 ␣
↪   71.631905                    5.81                     4                        ␣
↪                 5                   0.019698                40.442059            ␣
↪          9025.62                  18                17                 7         ␣
↪        4                        0                   6                 1994       ␣
↪   2011                               25                             88.755625␣
↪                               26.36                                              ␣
↪     5                              0.640252                                46.
↪905665                              1613.93                           15.750000 ␣
↪                          132.246250                                72.774140   ␣
↪                           9.823750                                               ␣
↪    5                               -0.059515                               39.
↪093244                           1128.202500                                 12 ␣
↪                             118.90                                 50.623125    ␣
↪                                            5                               -1.
↪038434                             30.450261                                809.
↪97                                  1                                         4␣
↪                                   1                                    2         ␣
↪                     2014                                        1                ␣
↪                                  4                                          1    ␣
↪                                       1                                        1 ␣
↪                      1.946018                                 -0.780493          ␣
↪                    -0.424949                                  2.440005           ␣
↪                                      0.0                               -0.312355␣
↪                             0.778170                            4.062019         ␣
↪                    7.322191                               13.759314              ␣
↪                 6.954507                                                0.0      ␣
↪                         0.589386                               279.510713        ␣
↪                      1057.97                              582.193117             ␣
↪                     78.59                                                   40   ␣
↪                         -0.476122                                312.745952       ␣
↪                                1                          mobile                 ␣
↪                              1                                      3
```

3.3 Guides

Guides on more advanced Featuretools functionality

3.3.1 Tuning Deep Feature Synthesis

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

```
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]
```

(continues on next page)

(continued from previous page)

```
    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
```

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=[expensive_purchase])
   ...:

In [6]: feature_matrix[['PERCENT_TRUE(transactions.amount > 125)']]
Out[6]:
             PERCENT_TRUE(transactions.amount > 125)
customer_id
5                                           0.227848
4                                           0.220183
1                                           0.119048
3                                           0.182796
2                                           0.129032
```

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

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=[])
   ...:
```

(continues on next page)

(continued from previous page)

```
In [9]: feature_matrix
Out[9]:
            zip_code  AVG_TIME_BETWEEN(sessions.session_start)  COUNT(sessions)  AVG_
↪TIME_BETWEEN(transactions.transaction_time)  COUNT(transactions)  AVG_TIME_
↪BETWEEN(sessions.session_start WHERE device = mobile)  AVG_TIME_BETWEEN(sessions.
↪session_start WHERE device = tablet)  AVG_TIME_BETWEEN(sessions.session_start WHERE␣
↪device = desktop)  COUNT(sessions WHERE device = mobile)  COUNT(sessions WHERE␣
↪device = tablet)  COUNT(sessions WHERE device = desktop)  AVG_TIME_
↪BETWEEN(transactions.sessions.session_start)  AVG_TIME_BETWEEN(transactions.
↪sessions.session_start WHERE sessions.device = desktop)  AVG_TIME_
↪BETWEEN(transactions.sessions.session_start WHERE sessions.device = tablet)  AVG_
↪TIME_BETWEEN(transactions.sessions.session_start WHERE sessions.device = mobile) ␣
↪AVG_TIME_BETWEEN(transactions.transaction_time WHERE sessions.device = desktop) ␣
↪AVG_TIME_BETWEEN(transactions.transaction_time WHERE sessions.device = tablet)  AVG_
↪TIME_BETWEEN(transactions.transaction_time WHERE sessions.device = mobile) ␣
↪COUNT(transactions WHERE sessions.device = desktop)  COUNT(transactions WHERE␣
↪sessions.device = tablet)  COUNT(transactions WHERE sessions.device = mobile)
customer_id                                                                          ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪                                                                                    ␣
↪
5              60091                             5577.000000                6        ␣
↪                                   363.333333                   79                  ␣
↪                 13942.500000                                                       ␣
↪        NaN                                                           9685.0        ␣
↪                                          3                                    1    ␣
↪                                     2                                      357.
↪500000                                   345.892857                                 ␣
↪                                             0.000000                               ␣
↪                                             796.714286                             ␣
↪                                               376.071429                           ␣
↪                                             65.000000                              ␣
↪                                        809.714286                                  ␣
↪                                            29                                      ␣
↪                         14                                               36
4              60091                             2516.428571                8        ␣
↪                                   168.518519                  109                  ␣
↪                  3336.666667                                                       ␣
↪        NaN                                                           4127.5        ␣
↪                                          4                                    1    ␣
↪                                     3                                      163.
↪101852                                   223.108108                                 ␣
↪                                             0.000000                               ␣
↪                                             192.500000                             ␣
↪                                               238.918919                           ␣
↪                                             65.000000                              ␣
↪                                        206.250000                                  ␣
↪                                            38                                      ␣
↪                         18                                               53
```

(continues on next page)

(continued from previous page)

```
1             60091                                  3305.714286                 8        ␣
↪                                   192.920000                126                          ␣
↪                   11570.000000                                                           ␣
↪     8807.5                                                          7150.0               ␣
↪                                            3                                         3   ␣
↪                                       2                                            185.
↪120000                                               275.000000                          ␣
↪                                                        419.404762                        ␣
↪                                                          420.727273                      ␣
↪                                                            302.500000                    ␣
↪                                                         442.619048                       ␣
↪                                                     438.454545                           ␣
↪                                                         27                               ␣
↪                       43                                                          56
3             13244                                  5096.000000                 6        ␣
↪                                   287.554348                 93                          ␣
↪                          NaN                                                             ␣
↪        NaN                                                          4745.0               ␣
↪                                            1                                         1   ␣
↪                                       4                                            276.
↪956522                                               233.360656                          ␣
↪                                                          0.000000                        ␣
↪                                                            0.000000                      ␣
↪                                                            251.475410                    ␣
↪                                                          65.000000                       ␣
↪                                                      65.000000                           ␣
↪                                                         62                               ␣
↪                       15                                                          16
2             13244                                  4907.500000                 7        ␣
↪                                   328.532609                 93                          ␣
↪                    1690.000000                                                           ␣
↪     5330.0                                                          6890.0               ␣
↪                                            2                                         2   ␣
↪                                       3                                            320.
↪054348                                               417.575758                          ␣
↪                                                        197.407407                        ␣
↪                                                           56.333333                      ␣
↪                                                            435.303030                    ␣
↪                                                         226.296296                       ␣
↪                                                      82.333333                           ␣
↪                                                         34                               ␣
↪                       28                                                          31
```

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*.

```
In [10]: feature_matrix[["COUNT(sessions WHERE device = tablet)", "AVG_TIME_
↪BETWEEN(sessions.session_start WHERE device = tablet)"]]
Out[10]:
             COUNT(sessions WHERE device = tablet)  AVG_TIME_BETWEEN(sessions.session_
↪start WHERE device = tablet)
customer_id                                                                            ␣
↪
5                                                1                                     ␣
↪            NaN
4                                                1                                     ␣
↪            NaN
```

(continues on next page)

(continued from previous page)

```
1                                                   3                               ␣
↪         8807.5
3                                                   1                               ␣
↪            NaN
2                                                   2                               ␣
↪         5330.0
```

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

Encoding categorical features

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

```
In [11]: feature_matrix, feature_defs = ft.dfs(entityset=es,
   ....:                                        target_entity="customers",
   ....:                                        agg_primitives=["mode"],
   ....:                                        max_depth=1)
   ....:

In [12]: feature_matrix
Out[12]:
            zip_code MODE(sessions.device)  DAY(date_of_birth)  DAY(join_date) ␣
↪MONTH(date_of_birth)  MONTH(join_date)  WEEKDAY(date_of_birth)  WEEKDAY(join_date) ␣
↪YEAR(date_of_birth)  YEAR(join_date)
customer_id                                                                         ␣
↪                                                                                   ␣
↪
5              60091                mobile                  28               17     ␣
↪              7                 7                       5                   5      ␣
↪         1984              2010
4              60091                mobile                  15                8     ␣
↪              8                 4                       1                   4      ␣
↪         2006              2011
1              60091                mobile                  18               17     ␣
↪              7                 4                       0                   6      ␣
↪         1994              2011
3              13244               desktop                  21               13     ␣
↪             11                 8                       4                   5      ␣
↪         2003              2011
2              13244               desktop                  18               15     ␣
↪              8                 4                       0                   6      ␣
↪         1986              2012
```

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(date_of_birth) = 15  DAY(date_of_birth) is unknown  DAY(join_date) = 17 ␣
↪DAY(join_date) = 15  DAY(join_date) = 13  DAY(join_date) = 8  DAY(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 ␣
```

(continues on next page)

(continued from previous page)

```
customer_id                                                                         ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪                                                                                   ␣
↪
5                        True             False                 False               ␣
↪           True                            False                           False   ␣
↪                  False                     True                     False         ␣
↪         False                         False                 True                ␣
↪False            False                 False                       False           ␣
↪           False                    True                  False                    ␣
↪             False            False                 False                  True␣
↪                        False                       False                        ␣
↪True                        False                       False                      ␣
↪       False                  False                   True                  False␣
↪                          False                       False                      ␣
↪False                         False                        False                   ␣
↪ True                             False                   False                  ␣
↪False                     True                     False
4                        True             False                 False               ␣
↪           True                            False                           False   ␣
↪                  False                    False                     False         ␣
↪          True                         False                False                ␣
↪False            False                    True                     False           ␣
↪             True                  False                  False                    ␣
↪             False                True                  False                  False␣
↪                        False                       False                       ␣
↪False                         False                        True                    ␣
↪        False                   False                 False                      ␣
↪True                              False                      True                  ␣
↪  False                           False                   False                    ␣
↪  False                               False                   True                 ␣
↪False                      False                     False
1                        True             False                 False               ␣
↪           True                            False                           False   ␣
↪                   True                    False                     False         ␣
↪         False                         False                 True                ␣
↪False            False                 False                       False           ␣
↪           False                    True                  False                    ␣
↪             False                True                  False                  False␣
↪                        False                        True                       ␣
↪False                         False                       False                    ␣
↪        False                     True                False                     ␣
↪False                            False                       False                 ␣
↪   False                           True                   False                    ␣
↪   False                              False                   True                 ␣
↪False                      False                     False
3                       False             True                  False               ␣
↪          False                             True                           False   ␣
↪                  False                    False                      True         ␣
↪         False                         False                False                ␣
↪False                   True                 False                       False      ␣
↪           False                   False                   True                    ␣
↪             False            False                   True                  False␣
↪                        False                       False                       ␣
```

(continues on next page)

(continued from previous page)

```
2                        False                True                 False                    ␣
↪          False                              True                               False  ␣
↪                   True                 False                     False                ␣
↪         False                         False                 False                   ␣
↪True                 False                False                     False              ␣
↪           True                 False                     False                        ␣
↪            False                 True                 False                    False ␣
↪                       False                      True                        False␣
↪                       False                     False                                ␣
↪ False                     True                 False                    False         ␣
↪                     False                    False                       False        ␣
↪                     False                     True                       False        ␣
↪                         False                False                   True             ␣
↪             False                        False
```

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)
[<Feature: zip_code = 60091>, <Feature: zip_code = 13244>, <Feature: zip_code is␣
↪unknown>, <Feature: MODE(sessions.device) = mobile>, <Feature: MODE(sessions.
↪device) = desktop>, <Feature: MODE(sessions.device) is unknown>, <Feature: DAY(date_
↪of_birth) = 18>, <Feature: DAY(date_of_birth) = 28>, <Feature: DAY(date_of_birth) =␣
↪21>, <Feature: DAY(date_of_birth) = 15>, <Feature: DAY(date_of_birth) is unknown>,
↪<Feature: DAY(join_date) = 17>, <Feature: DAY(join_date) = 15>, <Feature: DAY(join_
↪date) = 13>, <Feature: DAY(join_date) = 8>, <Feature: DAY(join_date) is unknown>,
↪<Feature: MONTH(date_of_birth) = 8>, <Feature: MONTH(date_of_birth) = 7>, <Feature:␣
↪MONTH(date_of_birth) = 11>, <Feature: MONTH(date_of_birth) is unknown>, <Feature:␣
↪MONTH(join_date) = 4>, <Feature: MONTH(join_date) = 8>, <Feature: MONTH(join_date)␣
↪= 7>, <Feature: MONTH(join_date) is unknown>, <Feature: WEEKDAY(date_of_birth) = 0>,
↪ <Feature: WEEKDAY(date_of_birth) = 5>, <Feature: WEEKDAY(date_of_birth) = 4>,
↪<Feature: WEEKDAY(date_of_birth) = 1>, <Feature: WEEKDAY(date_of_birth) is unknown>,
↪ <Feature: WEEKDAY(join_date) = 6>, <Feature: WEEKDAY(join_date) = 5>, <Feature:␣
↪WEEKDAY(join_date) = 4>, <Feature: WEEKDAY(join_date) is unknown>, <Feature:␣
↪YEAR(date_of_birth) = 2006>, <Feature: YEAR(date_of_birth) = 2003>, <Feature:␣
↪YEAR(date_of_birth) = 1994>, <Feature: YEAR(date_of_birth) = 1986>, <Feature:␣
↪YEAR(date_of_birth) = 1984>, <Feature: YEAR(date_of_birth) is unknown>, <Feature:␣
↪YEAR(join_date) = 2011>, <Feature: YEAR(join_date) = 2012>, <Feature: YEAR(join_
↪date) = 2010>, <Feature: YEAR(join_date) is unknown>]
```

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

3.3.2 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',
```

(continues on next page)

(continued from previous page)

```
   ...:                                             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>,
 <Feature: MODE(sessions.device_name)>,
 <Feature: MODE(sessions.device_type)>,
 <Feature: MODE(log.countrycode)>,
 <Feature: MODE(log.priority_level)>,
 <Feature: MODE(log.product_id)>,
 <Feature: MODE(log.subregioncode)>,
 <Feature: MODE(log.zipcode)>,
 <Feature: WEEKDAY(cancel_date)>,
 <Feature: WEEKDAY(date_of_birth)>,
 <Feature: WEEKDAY(signup_date)>,
 <Feature: WEEKDAY(upgrade_date)>,
 <Feature: cohorts.cohort_name>,
 <Feature: régions.language>,
 <Feature: MODE(sessions.MODE(log.countrycode))>,
 <Feature: MODE(sessions.MODE(log.priority_level))>,
 <Feature: MODE(sessions.MODE(log.product_id))>,
 <Feature: MODE(sessions.MODE(log.subregioncode))>,
 <Feature: MODE(sessions.MODE(log.zipcode))>,
 <Feature: MODE(log.sessions.customer_id)>,
 <Feature: MODE(log.sessions.device_name)>,
 <Feature: MODE(log.sessions.device_type)>,
 <Feature: cohorts.MODE(customers.cancel_reason)>,
 <Feature: cohorts.MODE(customers.engagement_level)>,
 <Feature: cohorts.MODE(customers.région_id)>,
 <Feature: cohorts.MODE(sessions.device_name)>,
 <Feature: cohorts.MODE(sessions.device_type)>,
 <Feature: cohorts.MODE(log.countrycode)>,
 <Feature: cohorts.MODE(log.priority_level)>,
 <Feature: cohorts.MODE(log.product_id)>,
 <Feature: cohorts.MODE(log.subregioncode)>,
 <Feature: cohorts.MODE(log.zipcode)>,
 <Feature: cohorts.WEEKDAY(cohort_end)>,
 <Feature: régions.MODE(customers.cancel_reason)>,
 <Feature: régions.MODE(customers.cohort)>,
 <Feature: régions.MODE(customers.engagement_level)>,
 <Feature: régions.MODE(sessions.device_name)>,
 <Feature: régions.MODE(sessions.device_type)>,
 <Feature: régions.MODE(log.countrycode)>,
 <Feature: régions.MODE(log.priority_level)>,
 <Feature: régions.MODE(log.product_id)>,
 <Feature: régions.MODE(log.subregioncode)>,
 <Feature: régions.MODE(log.zipcode)>]
```

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.

```
# 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'],
   ...:                                        trans_primitives=['weekday'],
   ...:                                        ignore_entities=['log', 'cohorts'],
   ...:                                        ignore_variables={
   ...:                                            'sessions': ['device_name'],
   ...:                                            'customers': ['date_of_birth']})
   ...:

In [6]: features_list
Out[6]:
[<Feature: cohort>,
 <Feature: age>,
 <Feature: région_id>,
 <Feature: loves_ice_cream>,
 <Feature: cancel_reason>,
 <Feature: engagement_level>,
 <Feature: MODE(sessions.device_type)>,
 <Feature: WEEKDAY(cancel_date)>,
 <Feature: WEEKDAY(signup_date)>,
 <Feature: WEEKDAY(upgrade_date)>,
 <Feature: régions.language>,
 <Feature: régions.MODE(customers.cancel_reason)>,
 <Feature: régions.MODE(customers.cohort)>,
 <Feature: régions.MODE(customers.engagement_level)>,
 <Feature: régions.MODE(sessions.device_type)>]
```

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.

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. Using the string primitive name or the primitive type will apply the options to all primitives of the same name. You can also set options for a specific instance of a primitive by using the primitive instance as a key in the `primitive_options` dictionary. Note, however, that specifying options for a specific instance will result in that instance ignoring any options set for the generic primitive through options with the primitive name or class as the key.

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.

```
# 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']},
   ...:                                            ('weekday', 'day'): {'include_
↪entities': ['customers']}
   ...:                                        })
   ...:

In [8]: features_list
Out[8]:
[<Feature: cohort>,
 <Feature: age>,
 <Feature: région_id>,
 <Feature: loves_ice_cream>,
 <Feature: cancel_reason>,
 <Feature: engagement_level>,
 <Feature: MODE(sessions.device_name)>,
 <Feature: MODE(sessions.device_type)>,
 <Feature: DAY(cancel_date)>,
 <Feature: DAY(date_of_birth)>,
 <Feature: DAY(signup_date)>,
 <Feature: DAY(upgrade_date)>,
 <Feature: WEEKDAY(cancel_date)>,
 <Feature: WEEKDAY(date_of_birth)>,
 <Feature: WEEKDAY(signup_date)>,
 <Feature: WEEKDAY(upgrade_date)>,
 <Feature: cohorts.cohort_name>,
 <Feature: régions.language>,
 <Feature: cohorts.MODE(customers.cancel_reason)>,
 <Feature: cohorts.MODE(customers.engagement_level)>,
 <Feature: cohorts.MODE(customers.région_id)>,
 <Feature: cohorts.MODE(sessions.device_name)>,
 <Feature: cohorts.MODE(sessions.device_type)>,
 <Feature: régions.MODE(customers.cancel_reason)>,
 <Feature: régions.MODE(customers.cohort)>,
 <Feature: régions.MODE(customers.engagement_level)>,
 <Feature: régions.MODE(sessions.device_name)>,
 <Feature: régions.MODE(sessions.device_type)>]
```

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.

```
# 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'],
   ...:
↪'customers': ['cancel_reason']}},
   ...:                                            'weekday': {'ignore_variables': {
↪'customers':
   ...:                                                                              ␣
↪  ['signup_date',
   ...:                                                                              ␣
↪   'cancel_date']}}})
   ...:

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.product_id)>,
 <Feature: MODE(log.zipcode)>,
 <Feature: WEEKDAY(date_of_birth)>,
 <Feature: WEEKDAY(upgrade_date)>,
 <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.product_id)>,
 <Feature: cohorts.MODE(log.zipcode)>,
 <Feature: cohorts.WEEKDAY(cohort_end)>,
 <Feature: régions.MODE(customers.cancel_reason)>,
 <Feature: régions.MODE(sessions.device_type)>,
 <Feature: régions.MODE(log.product_id)>,
 <Feature: régions.MODE(log.zipcode)>]
```

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',
   ....:                                        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_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: CUM_SUM(value) by session_id>,
 <Feature: CUM_SUM(value_2) by session_id>,
 <Feature: CUM_SUM(value_many_nans) by session_id>,
 <Feature: sessions.device_name>,
 <Feature: sessions.customer_id>,
 <Feature: sessions.device_type>,
 <Feature: products.rating>,
 <Feature: products.department>,
```

(continues on next page)

(continued from previous page)

```
 <Feature: sessions.customers.cohort>,
 <Feature: sessions.customers.age>,
 <Feature: sessions.customers.région_id>,
 <Feature: sessions.customers.loves_ice_cream>,
 <Feature: sessions.customers.cancel_reason>,
 <Feature: sessions.customers.engagement_level>,
 <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>,
 <Feature: CUM_SUM(products.rating) by session_id>,
 <Feature: CUM_SUM(products.rating) by sessions.customer_id>]
```

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.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`.

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.

```
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']}}]})
   ....:

In [14]: features_list
Out[14]:
[<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)>,
```

(continues on next page)

(continued from previous page)

```
<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.

3.3.3 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.

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*.

Parallel Feature Computation

Computational performance can often be improved by parallelizing the feature calculation process. There are several different approaches that can be used to perform parallel feature computation with Featuretools. An overview of the most commonly used approaches is provided below.

Computation with Dask and Koalas EntitySets (BETA)

Note: Support for Dask EntitySets and Koalas EntitySets is still in Beta. While the key functionality has been implemented, development is ongoing to add the remaining functionality.

All planned improvements to the Featuretools/Dask and Featuretools/Koalas integration are documented on Github (Dask issues, Koalas issues). If you see an open issue that is important for your application, please let us know by upvoting or commenting on the issue. If you encounter any errors using Dask or Koalas entities, or find missing functionality that does not yet have an open issue, please create a new issue on Github.

Dask or Koalas can be used with Featuretools to perform parallel feature computation with virtually no changes to the workflow required. Featuretools supports creating an `EntitySet` directly from Dask or Koalas dataframes instead of using pandas dataframes, enabling the parallel and distributed computation capabilities of Dask or Spark to be used. By creating an `EntitySet` directly from Dask or Koalas dataframes, Featuretools can be used to generate a larger-than-memory feature matrix, something that may be difficult with other approaches. When computing a feature matrix from an `EntitySet` created from Dask or Koalas dataframes, the resulting feature matrix will be returned as a Dask or Koalas dataframe depending on which type was used.

These methods do have some limitations in terms of the primitives that are available and the optional parameters that can be used when calculating the feature matrix. For more information on generating a feature matrix with this approach, refer to the guides *Using Dask EntitySets (BETA)* and *Using Koalas EntitySets (BETA)*.

Simple Parallel Feature Computation

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

```
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.

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`:

```
# 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:

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

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.

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`:

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

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 pandas `EntitySet` because the current parallel implementation sends the entire `EntitySet` to each worker which may exhaust the worker memory. Dask and Spark allow Featuretools to scale to multiple cores on a single machine or multiple machines on a cluster.

Note: Partitioning data is not necessary when using a Dask `EntitySet`, as the Dask dataframes that make up the `EntitySet` are already partitioned. Partitioning is only needed when working with pandas entities.

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.

3.3.4 Using Dask EntitySets (BETA)

Note: Support for Dask EntitySets is still in Beta. While the key functionality has been implemented, development is ongoing to add the remaining functionality.

All planned improvements to the Featuretools/Dask integration are documented on Github. If you see an open issue that is important for your application, please let us know by upvoting or commenting on the issue. If you encounter any errors using Dask entities, or find missing functionality that does not yet have an open issue, please create a new issue on Github.

Creating a feature matrix from a very large dataset can be problematic if the underlying pandas dataframes that make up the entities cannot easily fit in memory. To help get around this issue, Featuretools supports creating `Entity` and `EntitySet` objects from Dask dataframes. A Dask `EntitySet` can then be passed to `featuretools.dfs` or `featuretools.calculate_feature_matrix` to create a feature matrix, which will be returned as a Dask dataframe. In addition to working on larger than memory datasets, this approach also allows users to take advantage of the parallel and distributed processing capabilities offered by Dask.

This guide will provide an overview of how to create a Dask `EntitySet` and then generate a feature matrix from it. If you are already familiar with creating a feature matrix starting from pandas dataframes, this process will seem quite familiar, as there are no differences in the process. There are, however, some limitations when using Dask dataframes, and those limitations are reviewed in more detail below.

Creating Entities and EntitySets

For this example, we will create a very small pandas dataframe and then convert this into a Dask dataframe to use in the remainder of the process. Normally when using Dask, you would just read your data directly into a Dask dataframe without the intermediate step of using pandas.

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

In [2]: import pandas as pd

In [3]: import dask.dataframe as dd

In [4]: id = [0, 1, 2, 3, 4]

In [5]: values = [12, -35, 14, 103, -51]

In [6]: df = pd.DataFrame({"id": id, "values": values})

In [7]: dask_df = dd.from_pandas(df, npartitions=2)

In [8]: dask_df
Out[8]:
Dask DataFrame Structure:
                  id values
npartitions=2
0              int64  int64
3                ...    ...
4                ...    ...
Dask Name: from_pandas, 2 tasks
```

Now that we have our Dask dataframe, we can start to create the `EntitySet`. The current implementation does not support variable type inference for Dask entities, so we must pass a dictionary of variable types using the `variable_types` parameter when calling `es.entity_from_dataframe()`. Aside from needing to supply the variable types, the rest of the process of creating an `EntitySet` is the same as if we were using pandas dataframes.

```
In [9]: es = ft.EntitySet(id="dask_es")

In [10]: es = es.entity_from_dataframe(entity_id="dask_entity",
   ....:                               dataframe=dask_df,
   ....:                               index="id",
   ....:                               variable_types={"id": ft.variable_types.Id,
   ....:                                               "values": ft.variable_types.
↪Numeric})
   ....:

In [11]: es
Out[11]:
Entityset: dask_es
  Entities:
    dask_entity [Rows: Delayed('int-79d2621c-6394-4f55-b80b-255fda7c57b0'), Columns:␣
↪2]
  Relationships:
    No relationships
```

Notice that when we print our `EntitySet`, the number of rows for the `dask_entity` entity is returned as a Dask `Delayed` object. This is because obtaining the length of a Dask dataframe may require an expensive compute operation to sum up the lengths of all the individual partitions that make up the dataframe and that operation is not performed by default.

Running DFS

We can pass the `EntitySet` we created above to `featuretools.dfs` in order to create a feature matrix. If the `EntitySet` we pass to `dfs` is made of Dask entities, the feature matrix we get back will be a Dask dataframe.

```
In [12]: feature_matrix, features = ft.dfs(entityset=es,
   ....:                                   target_entity="dask_entity",
   ....:                                   trans_primitives=["negate"])
   ....:

In [13]: feature_matrix
Out[13]:
Dask DataFrame Structure:
              values -(values)     id
npartitions=2
0              int64     int64  int64
3                ...       ...    ...
4                ...       ...    ...
Dask Name: getitem, 21 tasks
```

This feature matrix can be saved to disk or computed and brought into memory, using the appropriate Dask dataframe methods.

```
In [14]: fm_computed = feature_matrix.compute()

In [15]: fm_computed
Out[15]:
   values  -(values)  id
0      12        -12   0
1     -35         35   1
2      14        -14   2
3     103       -103   3
4     -51         51   4
```

While this is a simple example to illustrate the process of using Dask dataframes with Featuretools, this process will also work with an `EntitySet` containing multiple entities, as well as with aggregation primitives.

Limitations

The key functionality of Featuretools is available for use with a Dask `EntitySet`, and work is ongoing to add the remaining functionality that is available when using a pandas `EntitySet`. There are, however, some limitations to be aware of when creating a Dask `Entityset` and then using it to generate a feature matrix. The most significant limitations are reviewed in more detail in this section.

Note: If the limitations of using a Dask `EntitySet` are problematic for your problem, you may still be able to compute a larger-than-memory feature matrix by partitioning your data as described in *Improving Computational Performance*.

Supported Primitives

When creating a feature matrix from a Dask `EntitySet`, only certain primitives can be used. Primitives that rely on the order of the entire dataframe or require an entire column for computation are currently not supported when using a Dask `EntitySet`. Multivariable and time-dependent aggregation primitives also are not currently supported.

To obtain a list of the primitives that can be used with a Dask `EntitySet`, you can call `featuretools.list_primitives()`. This will return a table of all primitives. Any primitive that can be used with a Dask `EntitySet` will have a value of `True` in the `dask_compatible` column.

```
In [16]: primitives_df = ft.list_primitives()

In [17]: dask_compatible_df = primitives_df[primitives_df["dask_compatible"] == True]

In [18]: dask_compatible_df.head()
Out[18]:
           name         type  dask_compatible  koalas_compatible                                    ␣
↪                 description
1    num_unique  aggregation             True               True  Determines the␣
↪number of distinct values, igno...
3           min  aggregation             True               True  Calculates the␣
↪smallest value, ignoring `NaN` ...
5          mean  aggregation             True               True         Computes the␣
↪average for a list of values.
6           any  aggregation             True              False       Determines if␣
↪any value is 'True' in a list.
7  percent_true  aggregation             True              False           Determines␣
↪the percent of `True` values.

In [19]: dask_compatible_df.tail()
Out[19]:
                name       type  dask_compatible  koalas_compatible                              ␣
↪                 description
74           absolute  transform             True               True             ␣
↪Computes the absolute value of a number.
75                age  transform             True              False  Calculates the␣
↪age in years as a floating poin...
76   subtract_numeric  transform             True              False              ␣
↪Element-wise subtraction of two lists.
77     divide_numeric  transform             True               True                 ␣
↪Element-wise division of two lists.
78        add_numeric  transform             True               True                 ␣
↪Element-wise addition of two lists.
```

Primitive Limitations

At this time, custom primitives created with `featuretools.primitives.make_trans_primitive()` or `featuretools.primitives.make_agg_primitive()` cannot be used for running deep feature synthesis on a Dask `EntitySet`. While it is possible to create custom primitives for use with a Dask `EntitySet` by extending the proper primitive class, there are several potential problems in doing so, and those issues are beyond the scope of this guide.

Entity Limitations

When creating a Featuretools `Entity` from Dask dataframes, variable type inference is not performed as it is when creating entities from pandas dataframes. This is done to improve speed as sampling the data to infer the variable types would require an expensive compute operation on the underlying Dask dataframe. As a consequence, users must define the variable types for each column in the supplied Dataframe. This step is needed so that the deep feature synthesis process can build the proper features based on the column types. A list of available variable types can be obtained by running `featuretools.variable_types.find_variable_types()`.

By default, Featuretools checks that entities created from pandas dataframes have unique index values. Because performing this same check with Dask would require an expensive compute operation, this check is not performed when creating an entity from a Dask dataframe. When using Dask dataframes, users must ensure that the supplied index values are unique.

When an `Entity` is created from a pandas dataframe, the ordering of the underlying dataframe rows is maintained. For a Dask `Entity`, the ordering of the dataframe rows is not guaranteed, and Featuretools does not attempt to maintain row order in a Dask `Entity`. If ordering is important, close attention must be paid to any output to avoid issues.

The `Entity.add_interesting_values()` method is not supported when using a Dask `Entity`. If needed, users can manually set `interesting_values` on entities by assigning them directly with syntax similar to this: `es["entity_name"]["variable_name"].interesting_values = ["Value 1", "Value 2"]`.

EntitySet Limitations

When creating a Featuretools `EntitySet` that will be made of Dask entities, all of the entities used to create the `EntitySet` must be of the same type, either all Dask entities or all pandas entities. Featuretools does not support creating an `EntitySet` containing a mix of Dask and pandas entities.

Additionally, the `EntitySet.add_interesting_values()` method is not supported when using a Dask `EntitySet`. Users can manually set `interesting_values` on entities, as described above.

DFS Limitations

There are a few key limitations when generating a feature matrix from a Dask `EntitySet`.

If a `cutoff_time` parameter is passed to `featuretools.dfs()` it should be a single cutoff time value, or a pandas dataframe. The current implementation will still work if a Dask dataframe is supplied for cutoff times, but a `.compute()` call will be made on the dataframe to convert it into a pandas dataframe. This conversion will result in a warning, and the process could take a considerable amount of time to complete depending on the size of the supplied dataframe.

Additionally, Featuretools does not currently support the use of the `approximate` or `training_window` parameters when working with Dask entitiysets, but should in future releases.

Finally, if the output feature matrix contains a boolean column with `NaN` values included, the column type may have a different datatype than the same feature matrix generated from a pandas `EntitySet`. If feature matrix column data types are critical, the feature matrix should be inspected to make sure the types are of the proper types, and recast as necessary.

Other Limitations

In some instances, generating a feature matrix with a large number of features has resulted in memory issues on Dask workers. The underlying reason for this is that the partition size of the feature matrix grows too large for Dask to handle as the number of feature columns grows large. This issue is most prevalent when the feature matrix contains a large number of columns compared to the dataframes that make up the entities. Possible solutions to this problem include reducing the partition size used when creating the entity dataframes or increasing the memory available on Dask workers.

Currently `featuretools.encode_features()` does not work with a Dask dataframe as input. This will hopefully be resolved in a future release of Featuretools.

The utility function `featuretools.make_temporal_cutoffs()` will not work properly with Dask inputs for `instance_ids` or `cutoffs`. However, as noted above, if a `cutoff_time` dataframe is supplied to `dfs`, the supplied dataframe should be a pandas dataframe, and this can be generated by supplying pandas inputs to `make_temporal_cutoffs()`.

The use of `featuretools.remove_low_information_features()` cannot currently be used with a Dask feature matrix.

When manually defining a `Feature`, the `use_previous` parameter cannot be used if this feature will be applied to calculate a feature matrix from a Dask `EntitySet`.

3.3.5 Using Koalas EntitySets (BETA)

Note: Support for Koalas EntitySets is still in Beta. While the key functionality has been implemented, development is ongoing to add the remaining functionality.

All planned improvements to the Featuretools/Koalas integration are documented on Github. If you see an open issue that is important for your application, please let us know by upvoting or commenting on the issue. If you encounter any errors using Koalas entities, or find missing functionality that does not yet have an open issue, please create a new issue on Github.

Creating a feature matrix from a very large dataset can be problematic if the underlying pandas dataframes that make up the entities cannot easily fit in memory. To help get around this issue, Featuretools supports creating `Entity` and `EntitySet` objects from Koalas dataframes. A Koalas `EntitySet` can then be passed to `featuretools.dfs` or `featuretools.calculate_feature_matrix` to create a feature matrix, which will be returned as a Koalas dataframe. In addition to working on larger than memory datasets, this approach also allows users to take advantage of the parallel and distributed processing capabilities offered by Koalas and Spark.

This guide will provide an overview of how to create a Koalas `EntitySet` and then generate a feature matrix from it. If you are already familiar with creating a feature matrix starting from pandas dataframes, this process will seem quite familiar, as there are no differences in the process. There are, however, some limitations when using Koalas dataframes, and those limitations are reviewed in more detail below.

Creating Entities and EntitySets

Koalas `EntitySets` require Koalas and PySpark. Both can be installed directly with `pip install featuretools[koalas]`. Java is also required for PySpark and may need to be installed, see the Spark documentation for more details. We will create a very small Koalas dataframe for this example. Koalas dataframes can also be created from pandas dataframes, Spark dataframes, or read in directly from a file.

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

In [2]: import databricks.koalas as ks

In [3]: id = [0, 1, 2, 3, 4]

In [4]: values = [12, -35, 14, 103, -51]

In [5]: koalas_df = ks.DataFrame({"id": id, "values": values})

In [6]: koalas_df
Out[6]:
   id  values
0   0      12
1   1     -35
2   2      14
3   3     103
4   4     -51
```

Now that we have our Koalas dataframe, we can start to create the `EntitySet`. The current implementation does not support variable type inference for Koalas entities, so we must pass a dictionary of variable types using the `variable_types` parameter when calling `es.entity_from_dataframe()`. Aside from needing to supply the variable types, the rest of the process of creating an `EntitySet` is the same as if we were using pandas dataframes.

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

In [8]: es = es.entity_from_dataframe(entity_id="koalas_entity",
   ...:                               dataframe=koalas_df,
   ...:                               index="id",
   ...:                               variable_types={"id": ft.variable_types.Id,
   ...:                                               "values": ft.variable_types.
↪Numeric})
   ...:

In [9]: es
Out[9]:
Entityset: koalas_es
  Entities:
    koalas_entity [Rows: 5, Columns: 2]
  Relationships:
    No relationships
```

Running DFS

We can pass the EntitySet we created above to featuretools.dfs in order to create a feature matrix. If the EntitySet we pass to dfs is made of Koalas entities, the feature matrix we get back will be a Koalas dataframe.

```
In [10]: feature_matrix, features = ft.dfs(entityset=es,
   ....:                                   target_entity="koalas_entity",
   ....:                                   trans_primitives=["negate"])
   ....:

In [11]: feature_matrix
Out[11]:
   values  -(values)  id
0      12        -12   0
1     -35         35   1
2     103       -103   3
3      14        -14   2
4     -51         51   4
```

This feature matrix can be saved to disk or converted to a pandas dataframe and brought into memory, using the appropriate Koalas dataframe methods.

While this is a simple example to illustrate the process of using Koalas dataframes with Featuretools, this process will also work with an EntitySet containing multiple entities, as well as with aggregation primitives.

Limitations

The key functionality of Featuretools is available for use with a Koalas EntitySet, and work is ongoing to add the remaining functionality that is available when using a pandas EntitySet. There are, however, some limitations to be aware of when creating a Koalas Entityset and then using it to generate a feature matrix. The most significant limitations are reviewed in more detail in this section.

Note: If the limitations of using a Koalas EntitySet are problematic for your problem, you may still be able to compute a larger-than-memory feature matrix by partitioning your data as described in *Improving Computational Performance*.

Supported Primitives

When creating a feature matrix from a Koalas EntitySet, only certain primitives can be used. Primitives that rely on the order of the entire dataframe or require an entire column for computation are currently not supported when using a Koalas EntitySet. Multivariable and time-dependent aggregation primitives also are not currently supported.

To obtain a list of the primitives that can be used with a Koalas EntitySet, you can call featuretools.list_primitives(). This will return a table of all primitives. Any primitive that can be used with a Koalas EntitySet will have a value of True in the koalas_compatible column.

```
In [12]: primitives_df = ft.list_primitives()

In [13]: koalas_compatible_df = primitives_df[primitives_df["koalas_compatible"] ==␣
↪True]

In [14]: koalas_compatible_df.head()
Out[14]:
```

(continues on next page)

(continued from previous page)

```
         name         type  dask_compatible  koalas_compatible                   ␣
↪                 description
1   num_unique  aggregation            True               True  Determines the␣
↪number of distinct values, igno...
3          min  aggregation            True               True  Calculates the␣
↪smallest value, ignoring `NaN` ...
5         mean  aggregation            True               True         Computes the␣
↪average for a list of values.
13         max  aggregation            True               True  Calculates the␣
↪highest value, ignoring `NaN` v...
14       count  aggregation            True               True  Determines the total␣
↪number of values, excludi...

In [15]: koalas_compatible_df.tail()
Out[15]:
                    name       type  dask_compatible  koalas_compatible            ␣
↪                          description
71                 weekday  transform            True               True  ␣
↪Determines the day of the week from a datetime.
72  greater_than_equal_to  transform            True               True  Determines␣
↪if values in one list are greater t...
74                absolute  transform            True               True       ␣
↪Computes the absolute value of a number.
77          divide_numeric  transform            True               True            ␣
↪  Element-wise division of two lists.
78             add_numeric  transform            True               True            ␣
↪  Element-wise addition of two lists.
```

Primitive Limitations

At this time, custom primitives created with `featuretools.primitives.make_trans_primitive()` or `featuretools.primitives.make_agg_primitive()` cannot be used for running deep feature synthesis on a Koalas `EntitySet`. While it is possible to create custom primitives for use with a Koalas `EntitySet` by extending the proper primitive class, there are several potential problems in doing so, and those issues are beyond the scope of this guide.

Entity Limitations

When creating a Featuretools `Entity` from Koalas dataframes, variable type inference is not performed as it is when creating entities from pandas dataframes. This is done to improve speed as sampling the data to infer the variable types could require expensive computation on the underlying Koalas dataframe. As a consequence, users must define the variable types for each column in the supplied Dataframe. This step is needed so that the deep feature synthesis process can build the proper features based on the column types. A list of available variable types can be obtained by running `featuretools.variable_types.find_variable_types()`.

By default, Featuretools checks that entities created from pandas dataframes have unique index values. Because performing this same check with Koalas could be computationally expensive, this check is not performed when creating an entity from a Koalas dataframe. When using Koalas dataframes, users must ensure that the supplied index values are unique.

When an `Entity` is created from a pandas dataframe, the ordering of the underlying dataframe rows is maintained. For a Koalas `Entity`, the ordering of the dataframe rows is not guaranteed, and Featuretools does not attempt to maintain row order in a Koalas `Entity`. If ordering is important, close attention must be paid to any output to avoid issues.

The `Entity.add_interesting_values()` method is not supported when using a Koalas `Entity`. If needed, users can manually set `interesting_values` on entities by assigning them directly with syntax similar to this: `es["entity_name"]["variable_name"].interesting_values = ["Value 1", "Value 2"]`.

EntitySet Limitations

When creating a Featuretools `EntitySet` that will be made of Koalas entities, all of the entities used to create the `EntitySet` must be of the same type, either all Koalas entities, all Dask entities, or all pandas entities. Featuretools does not support creating an `EntitySet` containing a mix of Koalas, Dask, and pandas entities.

Additionally, the `EntitySet.add_interesting_values()` method is not supported when using a Koalas `EntitySet`. Users can manually set `interesting_values` on entities, as described above.

DFS Limitations

There are a few key limitations when generating a feature matrix from a Koalas `EntitySet`.

If a `cutoff_time` parameter is passed to `featuretools.dfs()` it should be a single cutoff time value, or a pandas dataframe. The current implementation will still work if a Koalas dataframe is supplied for cutoff times, but a `.to_pandas()` call will be made on the dataframe to convert it into a pandas dataframe. This conversion will result in a warning, and the process could take a considerable amount of time to complete depending on the size of the supplied dataframe.

Additionally, Featuretools does not currently support the use of the `approximate` or `training_window` parameters when working with Koalas entitiysets, but should in future releases.

Finally, if the output feature matrix contains a boolean column with `NaN` values included, the column type may have a different datatype than the same feature matrix generated from a pandas `EntitySet`. If feature matrix column data types are critical, the feature matrix should be inspected to make sure the types are of the proper types, and recast as necessary.

Other Limitations

Currently `featuretools.encode_features()` does not work with a Koalas dataframe as input. This will hopefully be resolved in a future release of Featuretools.

The utility function `featuretools.make_temporal_cutoffs()` will not work properly with Koalas inputs for `instance_ids` or `cutoffs`. However, as noted above, if a `cutoff_time` dataframe is supplied to `dfs`, the supplied dataframe should be a pandas dataframe, and this can be generated by supplying pandas inputs to `make_temporal_cutoffs()`.

The use of `featuretools.remove_low_information_features()` cannot currently be used with a Koalas feature matrix.

When manually defining a `Feature`, the `use_previous` parameter cannot be used if this feature will be applied to calculate a feature matrix from a Koalas `EntitySet`.

3.3.6 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.

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.

```
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)
```

Now let's build some features definitions using DFS. Because we have categorical features, we also encode them with one hot encoding based on the values in the training data.

```
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)␣
↪ MODE(sessions.device) = mobile  MODE(sessions.device) = desktop  MODE(sessions.
↪device) is unknown  NUM_UNIQUE(sessions.device)  COUNT(transactions) ␣
↪MAX(transactions.amount)  MEAN(transactions.amount)  MIN(transactions.amount) ␣
↪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  NUM_UNIQUE(transactions.product_id) ␣
↪SKEW(transactions.amount)  STD(transactions.amount)  SUM(transactions.amount) ␣
↪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)␣
↪= 15  DAY(join_date) = 13  DAY(join_date) = 8  DAY(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  YEAR(date_of_birth) = 2006 ␣
↪YEAR(date_of_birth) = 2003  YEAR(date_of_birth) = 1994  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 ␣
↪MAX(sessions.COUNT(transactions))  MAX(sessions.MEAN(transactions.amount)) ␣
↪MAX(sessions.MIN(transactions.amount))  MAX(sessions.NUM_UNIQUE(transactions.
↪product_id))  MAX(sessions.SKEW(transactions.amount))  MAX(sessions.
↪STD(transactions.amount))  MAX(sessions.SUM(transactions.amount))  MEAN(sessions.
↪COUNT(transactions))  MEAN(sessions.MAX(transactions.amount))  MEAN(sessions.
↪MEAN(transactions.amount))  MEAN(sessions.MIN(transactions.amount))  MEAN(sessions.
↪NUM_UNIQUE(transactions.product_id))  MEAN(sessions.SKEW(transactions.amount)) ␣
↪MEAN(sessions.STD(transactions.amount))  MEAN(sessions.SUM(transactions.amount)) ␣
↪MIN(sessions.COUNT(transactions))  MIN(sessions.MAX(transactions.amount)) ␣
↪MIN(sessions.MEAN(transactions.amount))  MIN(sessions.NUM_UNIQUE(transactions.
```

(continues on next page)

(continued from previous page)

```
customer_id                                                                                                                          ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪                                                                                                                                    ␣
↪
5                               True                 False                      False                        6␣
↪                                    True                                        False                       ␣
↪          False                                     3                            79                         ␣
↪149.02                        80.375443                        7.55                                         ␣
↪      False                                         True                                        False       ␣
↪                                    False                                          False                    ␣
↪                                   5                      -0.025941                        44.095630        ␣
↪               6349.66                              False                         True                      ␣
↪         False                      False                                     False                         ␣
```

(continues on next page)

(continued from previous page)

```
4                    True             False             False           8␣
↪                       True                          False             ␣
↪         False                        3                 109            ␣
↪149.95                 80.070459                   5.73                ␣
↪     False                       False                            True ␣
↪                      False                               False        ␣
↪                     5               -0.036348              45.068765  ␣
↪            8727.68                   False                 False      ␣
↪       False                  True                   False             ␣
↪False                False              False              True        ␣
↪     False                   True                   False              ␣
↪False                         False                  True             False ␣
↪               False                    False                  False   ␣
↪                False                   False                  True    ␣
↪                     False                 False               False   ␣
↪            True                      False                    True    ␣
↪              False                   False                   False    ␣
↪              False                       False                True    ␣
↪          False                  False                    False        ␣
↪                18                        110.450000                   ␣
↪          54.83                                              5         ␣
↪               0.382868                             54.293903          ␣
↪            1351.46                    13.625000                       ␣
↪  144.748750                          81.207189                        ␣
↪  16.438750                                   4.625000                 ␣
↪             0.000346                        44.515729                 ␣
↪       1090.960000                          10                         ␣
↪    139.20                           70.638182                         ␣
↪                4                               -0.711744              ␣
↪      29.026424                          771.68                        ␣
↪      True                                 False                       ␣
↪                False                                         True     ␣
↪                                False                                  ␣
↪False                                        True                      ␣
↪           False                                   True                ␣
↪                    False                                   True       ␣
↪                          False                                   1    ␣
↪                                        5                          1␣
↪                                             1                         ␣
↪ 1                           0.282488                          0.027256 ␣
↪                            1.980948                          2.103510 ␣
↪                                   -0.644061                           -1.
↪065663                               -0.391805                    3.335416 ␣
↪                             3.514421                         13.027258 ␣
↪                         16.960575                                  0.
↪517549                          0.387884                          235.
↪992478                          1157.99                           649.
↪657515                           131.51
↪          37                                 0.002764                  ␣
↪356.125829                                      False                  ␣
↪            True                                     False             ␣
↪                 False                                     False       ␣
↪                             False                                True ␣
↪                                     False                             ␣
↪False                                         1                        ␣
↪        3
1                    True             False             False           8␣
↪                       True                          False             ␣
↪         False                        3                 126            ␣
↪139.43                 71.631905                   5.81                ␣
↪      True                       False                           False ␣
↪                      False                               False        ␣
↪                     5                0.019698              40.442059  ␣
↪            9025.62                    True                 False      ␣
```

(continues on next page)

(continued from previous page)

```
3                       False              True                False                6␣
↪                           False                              True                 ␣
↪            False                             3                 93                  ␣
↪149.15                 67.060430                        5.89                        ␣
↪     False                               False                               False  ␣
↪                            True                                    False           ␣
↪                          5                 0.418230                  43.683296     ␣
↪                  6236.62                    False                  False           ␣
↪        True                    False                        False                  ␣
↪False                  False               True                False                ␣
↪     False                    False                     False                       ␣
↪ True                           False                     False                True ␣
↪               False                        False                        False      ␣
↪                False                        True                        False      ␣
↪                       False                    False                       True    ␣
↪           False                         False                        False         ␣
↪               True                      False                        False         ␣
↪              False                          False                      True        ␣
↪          False                   False                        False                ␣
↪                18                              82.109444                           ␣
↪          20.06                                                       5             ␣
↪               0.854976                              50.110120                      ␣
↪             1477.97                        15.500000                               ␣
↪  141.271667                              67.539577                                 ␣
↪  11.035000                                         4.833333                        ␣
↪              0.381014                             42.883316                        ␣
↪        1039.436667                              11                                 ␣
↪    126.74                             55.579412                                    ␣
↪                 4                              -0.289466                           ␣
↪      35.704680                           889.21                                    ␣
↪      True                                  False                                   ␣
↪                 False                                            True              ␣
↪                                 False                                              ␣
↪False                                          True                                 ␣
↪           False                                      True                          ␣
↪                    False                                      True                 ␣
↪                          False                                            1        ␣
↪                                      4                                          1␣
↪                                          1                                         ␣
↪ 1                          -1.507217                                 -0.941078     ␣
↪                            0.678544                                  1.000771      ␣
↪                                   -2.449490                                  -0.
↪245703                                 2.246479                           2.428992  ␣
↪                            10.724241                                 11.174282     ␣
↪                          5.424407                                            0.
↪408248                                 0.429374                             219.
↪021420                                  847.63                              405.
↪237462                                   66.21                                      ␣
↪           29                               2.286086                                ␣
↪257.299895                                      False                               ␣
↪           False                                      True                          ␣
↪                 False                                     False                    ␣
↪                             False                                           False  ␣
↪                                      True                                          ␣
↪False                                               1                               ␣
↪        3                                                                           ␣
2                       False              True                False                7␣
↪                           False                              True                 ␣
↪            False                             3                 93                  ␣
↪146.81                 77.422366                        8.73                        ␣
↪     True                                False                               False  ␣
↪                            False                                   False           ␣
↪                          5                 0.098259                  37.705178     ␣
↪                  7200.28                    True                   False           ␣
```

(continues on next page)

(continued from previous page)

Now, we can use *featuretools.save_features()* to save a list features to a json file

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

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.

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

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

```
In [9]: feature_matrix = ft.calculate_feature_matrix(saved_features, es_test)

In [10]: feature_matrix
Out[10]: 
             zip_code = 60091  zip_code = 13244  zip_code is unknown  COUNT(sessions)␣
↪ MODE(sessions.device) = mobile  MODE(sessions.device) = desktop  MODE(sessions.
↪device) is unknown  NUM_UNIQUE(sessions.device)  COUNT(transactions) ␣
↪MAX(transactions.amount)  MEAN(transactions.amount)  MIN(transactions.amount) ␣
↪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  NUM_UNIQUE(transactions.product_id) ␣
↪SKEW(transactions.amount)  STD(transactions.amount)  SUM(transactions.amount) ␣
↪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)␣
↪= 15  DAY(join_date) = 13  DAY(join_date) = 8  DAY(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  YEAR(date_of_birth) = 2006 ␣
↪YEAR(date_of_birth) = 2003  YEAR(date_of_birth) = 1994  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 ␣
↪MAX(sessions.COUNT(transactions))  MAX(sessions.MEAN(transactions.amount)) ␣
↪MAX(sessions.MIN(transactions.amount))  MAX(sessions.NUM_UNIQUE(transactions.
↪product_id))  MAX(sessions.SKEW(transactions.amount))  MAX(sessions.
↪STD(transactions.amount))  MAX(sessions.SUM(transactions.amount))  MEAN(sessions.
↪COUNT(transactions))  MEAN(sessions.MAX(transactions.amount))  MEAN(sessions.
↪MEAN(transactions.amount))  MEAN(sessions.MIN(transactions.amount))  MEAN(sessions.
↪NUM_UNIQUE(transactions.product_id))  MEAN(sessions.SKEW(transactions.amount)) ␣
↪MEAN(sessions.STD(transactions.amount))  MEAN(sessions.SUM(transactions.amount)) ␣
↪MIN(sessions.COUNT(transactions))  MIN(sessions.MAX(transactions.amount)) ␣
↪MIN(sessions.MEAN(transactions.amount))  MIN(sessions.NUM_UNIQUE(transactions.
↪product_id))  MIN(sessions.SKEW(transactions.amount))  MIN(sessions.
↪STD(transactions.amount))  MIN(sessions.SUM(transactions.amount))  MODE(sessions.
↪DAY(session_start)) = 1  MODE(sessions.DAY(session_start)) is unknown ␣
↪MODE(sessions.MODE(transactions.product_id)) = 3  MODE(sessions.MODE(transactions.
↪product_id)) = 1  MODE(sessions.MODE(transactions.product_id)) = 4  MODE(sessions.
↪MODE(transactions.product_id)) is unknown  MODE(sessions.MONTH(session_start)) = 1 ␣
↪MODE(sessions.MONTH(session_start)) is unknown  MODE(sessions.WEEKDAY(session_
↪start)) = 2  MODE(sessions.WEEKDAY(session_start)) is unknown  MODE(sessions.
↪YEAR(session_start)) = 2014  MODE(sessions.YEAR(session_start)) is unknown  NUM_
↪UNIQUE(sessions.DAY(session_start))  NUM_UNIQUE(sessions.MODE(transactions.product_
↪id))  NUM_UNIQUE(sessions.MONTH(session_start))  NUM_UNIQUE(sessions.
↪WEEKDAY(session_start))  NUM_UNIQUE(sessions.YEAR(session_start))  SKEW(sessions.
↪COUNT(transactions))  SKEW(sessions.MAX(transactions.amount))  SKEW(sessions.
↪MEAN(transactions.amount))  SKEW(sessions.MIN(transactions.amount))  SKEW(sessions.
```

(continues on next page)

(continued from previous page)

```
customer_id
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
↪
1                    True             False                False                6
↪                       False                                   True
↪        False                          3                   73
↪147.64                    79.128904                  6.47
↪     True                           False                              False
↪                       False                                      False
↪                     5                 -0.173042                  41.998795
↪            5776.41                    False                    False
↪       False                    False                           True
```

(continues on next page)

(continued from previous page)

```
4                       False              True                 False                9␣
↪                           False                                True                 ␣
↪         False                            3                     126                  ␣
↪147.55                 80.781190                      6.19                           ␣
↪     False                                  False                              False ␣
↪                           False                                         True        ␣
↪                         5                    -0.179621               36.523849      ␣
↪              10178.43                      False                      False         ␣
↪       False                    False                         True                   ␣
↪False                   False                   False                   False        ␣
↪      True                        False                     False                    ␣
↪ True                               False                     False              False ␣
↪                True                          False                        False     ␣
↪                False                          True                        False     ␣
↪                       False                       False                  False      ␣
↪            True                         False                           False       ␣
↪              False                      False                           False       ␣
↪              False                           True                       False       ␣
↪          False                  False                         True                  ␣
↪                21                                104.565000                         ␣
↪          60.29                                                   5                  ␣
↪               0.417250                                41.627134                     ␣
↪            1650.65                       14.000000                                  ␣
↪  131.211111                             81.540322                                   ␣
↪  21.453333                                           4.777778                       ␣
↪            -0.199690                                35.499735                       ␣
↪        1130.936667                                  8                               ␣
↪    118.59                               69.665000                                   ␣
↪                 4                                        -0.624344                  ␣
↪      22.026552                                    557.32                            ␣
↪      True                                           False                           ␣
↪                 False                                                  True         ␣
↪                                 False                                               ␣
↪False                                            True                                ␣
↪           False                                        True                         ␣
↪                    False                                        True                ␣
↪                          False                                          1           ␣
↪                                      5                                            1␣
↪                                           1                                         ␣
↪ 1                             -0.086578                                  0.230847   ␣
↪                               1.078619                                  1.490781    ␣
↪                                      -1.619848                                    -1.
↪743267                                  -0.385392                              4.272002 ␣
↪                               11.457114                                 11.875823   ␣
↪                            17.851716                                              0.
↪440959                                    0.324894                                  333.
↪923377                                    1180.90                                   733.
↪862898                                     193.08                                        ␣
↪           43                                   -1.797214                            ␣
↪319.497611                                          False                            ␣
↪            True                                              False                  ␣
↪                 False                                              False            ␣
↪                             False                                             False ␣
↪                                      True                                           ␣
↪False                                                   1                            ␣
↪         3                                                                           ␣
3                        True                False                False                5␣
↪                            True                               False                 ␣
↪         False                            2                     64                   ␣
↪148.09                 82.171094                      10.66                          ␣
↪     False                                  True                               False ␣
↪                           False                                        False        ␣
↪                         5                    -0.081427               42.416322      ␣
↪              5258.95                       False                      False         ␣
```

(continues on next page)

(continued from previous page)

```
2                      False              True                False            8␣
↪                          False                             True               ␣
↪        False                            3                  129                ␣
↪148.34               76.571085                    8.00                          ␣
↪    False                                  False                              False ␣
↪                           True                                   False         ␣
↪                         5                   0.040395               39.913352   ␣
↪             9877.67                  False                 False               ␣
↪      False                  False                 True                        ␣
↪False                False             False               False               ␣
↪      True                 False               False                            ␣
↪False                           True                   False               False ␣
↪             False                        True                          False   ␣
↪              False                      False                          False   ␣
↪                      True                    True                     False    ␣
↪          False                       False                           False     ␣
↪             False                    False                           False     ␣
↪             False                         True                       False     ␣
↪          False                True                          False             ␣
↪                 21                              87.669412                      ␣
↪          40.88                                                     5           ␣
↪               0.454842                               43.950396                 ␣
↪             1690.97                       16.125000                            ␣
↪  137.602500                              76.964367                             ␣
↪  17.001250                                        4.875000                     ␣
↪            -0.010253                              39.477166                    ␣
↪        1234.708750                            8                                ␣
↪    120.06                            52.288421                                 ␣
↪                 4                             -0.522578                        ␣
↪      33.618728                              619.93                             ␣
↪      True                                     False                            ␣
↪                 False                                             True         ␣
↪                                 False                                          ␣
↪False                                         True                              ␣
↪          False                                     True                        ␣
↪                   False                                     True               ␣
↪                         False                                          1       ␣
↪                                     4                                        1␣
↪                                          1                                     ␣
↪ 1                             -1.158217                               -0.964539 ␣
↪                              -2.048650                               1.981423  ␣
↪                                      -2.828427                               -0.
↪403715                                  -0.576079                          4.086126 ␣
↪                               9.491736                               10.757169 ␣
↪                           10.517928                                          0.
↪353553                                   0.305345                            346.
↪152626                                   1100.82                             615.
↪714934                                    136.01                                ␣
↪          39                                  -0.082021                         ␣
↪315.817331                                       False                          ␣
↪          False                                           False                 ␣
↪                 True                                            False          ␣
↪                            False                                         False ␣
↪                                     True                                       ␣
↪False                                               1                           ␣
↪        3
5                       True              False               False            7␣
```

(continues on next page)

```
↪                          False                             True               ␣
↪        False                            3                  108                ␣
↪149.53               83.506852                    6.35                          ␣
↪    False                                  False                              False ␣
↪                          False                                    True         ␣
↪                         5                  -0.107234               37.514054   ␣
↪             9018.74                  False                 True                ␣
```

(continued from previous page)

As you can see above, we have the exact same features as before, but calculated using the test data.

Exporting Feature Matrix

Save as csv

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

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

We can also read it back in as follows:

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

In [13]: saved_fm
Out[13]:
              zip_code = 60091  zip_code = 13244  zip_code is unknown  COUNT(sessions)␣
↪ MODE(sessions.device) = mobile  MODE(sessions.device) = desktop  MODE(sessions.
↪device) is unknown  NUM_UNIQUE(sessions.device)  COUNT(transactions) ␣
↪MAX(transactions.amount)  MEAN(transactions.amount)  MIN(transactions.amount) ␣
↪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  NUM_UNIQUE(transactions.product_id) ␣
↪SKEW(transactions.amount)  STD(transactions.amount)  SUM(transactions.amount) ␣
↪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)␣
↪= 15  DAY(join_date) = 13  DAY(join_date) = 8  DAY(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  YEAR(date_of_birth) = 2006 ␣
↪YEAR(date_of_birth) = 2003  YEAR(date_of_birth) = 1994  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 ␣
↪MAX(sessions.COUNT(transactions))  MAX(sessions.MEAN(transactions.amount)) ␣
↪MAX(sessions.MIN(transactions.amount))  MAX(sessions.NUM_UNIQUE(transactions.
↪product_id))  MAX(sessions.SKEW(transactions.amount))  MAX(sessions.
↪STD(transactions.amount))  MAX(sessions.SUM(transactions.amount))  MEAN(sessions.
↪COUNT(transactions))  MEAN(sessions.MAX(transactions.amount))  MEAN(sessions.
↪MEAN(transactions.amount))  MEAN(sessions.MIN(transactions.amount))  MEAN(sessions.
↪NUM_UNIQUE(transactions.product_id))  MEAN(sessions.SKEW(transactions.amount)) ␣
↪MEAN(sessions.STD(transactions.amount))  MEAN(sessions.SUM(transactions.amount)) ␣
↪MIN(sessions.COUNT(transactions))  MIN(sessions.MAX(transactions.amount)) ␣
↪MIN(sessions.MEAN(transactions.amount))  MIN(sessions.NUM_UNIQUE(transactions.
↪product_id))  MIN(sessions.SKEW(transactions.amount))  MIN(sessions.
↪STD(transactions.amount))  MIN(sessions.SUM(transactions.amount))  MODE(sessions.
↪DAY(session_start)) = 1  MODE(sessions.DAY(session_start)) is unknown ␣
↪MODE(sessions.MODE(transactions.product_id)) = 3  MODE(sessions.MODE(transactions.
↪product_id)) = 1  MODE(sessions.MODE(transactions.product_id)) = 4  MODE(sessions.
↪MODE(transactions.product_id)) is unknown  MODE(sessions.MONTH(session_start)) = 1 ␣
↪MODE(sessions.MONTH(session_start)) is unknown  MODE(sessions.WEEKDAY(session_
↪start)) = 2  MODE(sessions.WEEKDAY(session_start)) is unknown  MODE(sessions.
↪YEAR(session_start)) = 2014  MODE(sessions.YEAR(session_start)) is unknown  NUM_
↪UNIQUE(sessions.DAY(session_start))  NUM_UNIQUE(sessions.MODE(transactions.product_
↪id))  NUM_UNIQUE(sessions.MONTH(session_start))  NUM_UNIQUE(sessions.
↪WEEKDAY(session_start))  NUM_UNIQUE(sessions.YEAR(session_start))  SKEW(sessions.
↪COUNT(transactions))  SKEW(sessions.MAX(transactions.amount))  SKEW(sessions.
↪MEAN(transactions.amount))  SKEW(sessions.MIN(transactions.amount))  SKEW(sessions.
↪NUM_UNIQUE(transactions.product_id))  SKEW(sessions.STD(transactions.amount)) ␣
```

(continues on next page)

(continued from previous page)

```
customer_id                                                                                ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪                                                                                          ␣
↪
1                         True              False                 False                   6␣
↪                            False                                 True                    ␣
↪          False                             3                    73                       ␣
↪147.64                  79.128904                     6.47                                ␣
↪      True                                 False                                 False    ␣
↪                            False                                          False          ␣
↪                        5                     -0.173042                   41.998795       ␣
↪              5776.41                        False                    False               ␣
↪         False                     False                        True                      ␣
```

(continues on next page)

(continued from previous page)

```
4                       False              True                 False              9␣
↪                           False                                True               ␣
↪          False                           3                     126                ␣
↪147.55                  80.781190                        6.19                      ␣
↪      False                              False                             False   ␣
↪                           False                                    True           ␣
↪                          5                  -0.179621                36.523849    ␣
↪              10178.43                    False                     False          ␣
↪        False                    False                        True                 ␣
↪False                    False                   False                False        ␣
↪       True                     False                        False                 ␣
↪ True                             False                     False               False ␣
↪                  True                       False                        False    ␣
↪                  False                       True                        False    ␣
↪                         False                    False                  False     ␣
↪              True                        False                         False      ␣
↪                False                     False                         False      ␣
↪                False                          True                     False      ␣
↪            False                    False                        True             ␣
↪                  21                              104.565000                       ␣
↪            60.29                                                   5              ␣
↪                 0.417250                               41.627134                  ␣
↪              1650.65                         14.000000                            ␣
↪  131.211111                                 81.540322                             ␣
↪  21.453333                                           4.777778                     ␣
↪              -0.199690                              35.499735                     ␣
↪          1130.936667                                8                             ␣
↪    118.59                              69.665000                                  ␣
↪                  4                             -0.624344                          ␣
↪      22.026552                                 557.32                             ␣
↪      True                                        False                            ␣
↪                   False                                            True           ␣
↪                                   False                                           ␣
↪False                                          True                                ␣
↪           False                                      True                         ␣
↪                    False                                     True                 ␣
↪                          False                                          1         ␣
↪                                       5                                          1␣
↪                                            1                                      ␣
↪ 1                           -0.086578                                0.230847     ␣
↪                             1.078619                                1.490781      ␣
↪                                    -1.619848                               -1.
↪743267                                -0.385392                               4.272002 ␣
↪                             11.457114                               11.875823     ␣
↪                          17.851716                                         0.
↪440959                                 0.324894                            333.
↪923377                                 1180.90                             733.
↪862898                                  193.08                                     ␣
↪           43                                -1.797214                             ␣
↪319.497611                                          False                          ␣
↪              True                                        False                    ␣
↪                   False                                        False              ␣
↪                             False                                          False  ␣
↪                                      True                                         ␣
↪False                                                1                             ␣
↪          3                                                                        ␣
3                        True              False                False              5␣
↪                               True                            False               ␣
↪          False                           2                     64                 ␣
↪148.09                  82.171094                        10.66                     ␣
↪      False                              True                              False   ␣
↪                           False                                    False          ␣
↪                          5                  -0.081427                42.416322    ␣
↪                5258.95                    False                     False         ␣
```

(continues on next page)

(continued from previous page)

```
2                    False              True                 False              8
↪                        False                           True
↪        False                          3                129
↪148.34              76.571085                     8.00
↪    False                              False                              False
↪                         True                                      False
↪                       5                    0.040395                 39.913352
↪             9877.67                    False                  False
↪      False                   False                     True
↪False               False              False               False
↪      True                    False                       False
↪False                            True                   False                  False
↪              False                       True                         False
↪               False                     False                         False
↪                       True                    True                   False
↪          False                          False                        False
↪             False                       False                        False
↪             False                            True                    False
↪         False                    True                       False
↪                 21                           87.669412
↪         40.88                                                  5
↪              0.454842                             43.950396
↪               1690.97                     16.125000
↪  137.602500                              76.964367
↪  17.001250                                         4.875000
↪            -0.010253                              39.477166
↪      1234.708750                                 8
↪    120.06                               52.288421
↪                 4                              -0.522578
↪      33.618728                                619.93
↪      True                                       False
↪                 False                                            True
↪                                 False
↪False                                          True
↪          False                                        True
↪                   False                                        True
↪                         False                                        1
↪                                       4                                       1
↪                                            1
↪ 1                         -1.158217                                  -0.964539
↪                          -2.048650                                   1.981423
↪                                  -2.828427                                   -0.
↪403715                              -0.576079                                4.086126
↪                            9.491736                                  10.757169
↪                        10.517928                                             0.
↪353553                               0.305345                                 346.
↪152626                               1100.82                                  615.
↪714934                               136.01
↪           39                              -0.082021
↪315.817331                                       False
↪          False                                         False
↪                 True                                         False
↪                            False                                             False
↪                                     True
↪False                                            1
↪        3
5                    True              False                 False              7
↪                        False                           True
↪        False                          3                108
↪149.53              83.506852                     6.35
↪    False                              False                              False
↪                        False                                      True
↪                       5                   -0.107234                 37.514054
↪             9018.74                    False                     True
```

(continues on next page)

(continued from previous page)

3.3.7 Advanced Custom Primitives Guide

Functions With Additional Arguments

One caveat with the make_primitive 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 `NaturalLanguage` input. Since `string` is not a feature, it needs to be a keyword argument to `string_count`.

```
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.

```
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.

```
In [3]: StringCount = make_trans_primitive(function=string_count,
   ...:                                    input_types=[NaturalLanguage],
   ...:                                    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.

```
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', 'MEAN(log.value)',
↪'MEAN(log.value_2)', 'MEAN(log.value_many_nans)', 'STD(log.value)', 'STD(log.value_
↪2)', 'STD(log.value_many_nans)', 'SUM(log.value)', 'SUM(log.value_2)', 'SUM(log.
↪value_many_nans)', 'customers.cohort', 'customers.age', 'customers.région_id',
↪'customers.loves_ice_cream', 'customers.cancel_reason', 'customers.engagement_level
↪', 'MEAN(log.STRING_COUNT(comments, "the"))', 'MEAN(log.products.rating)', 'STD(log.
↪STRING_COUNT(comments, "the"))', 'STD(log.products.rating)', 'SUM(log.STRING_
↪COUNT(comments, "the"))', 'SUM(log.products.rating)', 'customers.MEAN(log.value)',
↪'customers.MEAN(log.value_2)', 'customers.MEAN(log.value_many_nans)', 'customers.
↪STD(log.value)', 'customers.STD(log.value_2)', 'customers.STD(log.value_many_nans)',
↪ 'customers.SUM(log.value)', 'customers.SUM(log.value_2)', 'customers.SUM(log.value_
↪many_nans)', 'customers.STRING_COUNT(favorite_quote, "the")', 'customers.cohorts.
```

(continues on next page)

(continued from previous page)

```
In [8]: feature_matrix[['STD(log.STRING_COUNT(comments, "the"))', 'SUM(log.STRING_
↪COUNT(comments, "the"))', 'MEAN(log.STRING_COUNT(comments, "the"))']]
Out[8]:
    STD(log.STRING_COUNT(comments, "the"))  SUM(log.STRING_COUNT(comments, "the")) ␣
↪MEAN(log.STRING_COUNT(comments, "the"))
id                                                                                   ␣
↪
0                                 47.124304                                     209  ␣
↪                                  41.80
1                                 36.509131                                     109  ␣
↪                                  27.25
2                                       NaN                                      29  ␣
↪                                  29.00
3                                 49.497475                                      70  ␣
↪                                  35.00
4                                  0.000000                                       0  ␣
↪                                   0.00
5                                  1.414214                                       4  ␣
↪                                   2.00
```

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.

```
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.

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

In [11]: es = make_ecommerce_entityset()
```

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.

```
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]']]
Out[14]:
    customers.CASE_COUNT(favorite_quote)[0]  customers.CASE_COUNT(favorite_quote)[1]
id
0                                         1                                       44
1                                         1                                       44
2                                         1                                       44
3                                         1                                       41
4                                         1                                       41
5                                         1                                       57
```

3.4 Resources

Frequently asked questions and additional resources

3.4.1 Frequently Asked Questions

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

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

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.

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

[2]:
```
Entityset: transactions
  Entities:
    transactions [Rows: 500, Columns: 5]
    products [Rows: 5, Columns: 2]
    sessions [Rows: 35, Columns: 4]
```

(continues on next page)

(continued from previous page)

```
  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           1 2014-01-01 00:00:00  127.64           5
2                 2           1 2014-01-01 00:01:05  109.48           2
308             308           1 2014-01-01 00:02:10   95.06           3
116             116           1 2014-01-01 00:03:15   78.92           4
371             371           1 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

[6]:
```
es['transactions'].df.head()
```

[6]:
```
     transaction_id  session_id    transaction_time product_id  amount  \
298             298           1 2014-01-01 00:00:00          5  127.64
2                 2           1 2014-01-01 00:01:05          2  109.48
308             308           1 2014-01-01 00:02:10          3   95.06
116             116           1 2014-01-01 00:03:15          4   78.92
371             371           1 2014-01-01 00:04:20          3   31.54

     customer_id   device session_start zip_code           join_date  \
298            2  desktop    2014-01-01    13244 2012-04-15 23:31:04
2              2  desktop    2014-01-01    13244 2012-04-15 23:31:04
308            2  desktop    2014-01-01    13244 2012-04-15 23:31:04
116            2  desktop    2014-01-01    13244 2012-04-15 23:31:04
371            2  desktop    2014-01-01    13244 2012-04-15 23:31:04

    date_of_birth
298    1986-08-18
2      1986-08-18
308    1986-08-18
116    1986-08-18
371    1986-08-18
```

Notice the columns `session_id`, `session_start`, `join_date`, `device`, `customer_id`, and `zip_code`.

[7]:
```
es = es.normalize_entity(base_entity_id="transactions",
                         new_entity_id="sessions",
                         index="session_id",
                         make_time_index="session_start",
                         additional_variables=["join_date"],
                         copy_variables=["device", "customer_id", "zip_code","session_
↪start"])
```

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`.

[8]:
```
es['transactions'].df.head()
```

[8]:
```
     transaction_id  session_id    transaction_time product_id  amount  \
298             298           1 2014-01-01 00:00:00          5  127.64
2                 2           1 2014-01-01 00:01:05          2  109.48
308             308           1 2014-01-01 00:02:10          3   95.06
116             116           1 2014-01-01 00:03:15          4   78.92
371             371           1 2014-01-01 00:04:20          3   31.54

     customer_id   device session_start zip_code date_of_birth
298            2  desktop    2014-01-01    13244    1986-08-18
2              2  desktop    2014-01-01    13244    1986-08-18
308            2  desktop    2014-01-01    13244    1986-08-18
116            2  desktop    2014-01-01    13244    1986-08-18
371            2  desktop    2014-01-01    13244    1986-08-18
```

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 seen below.

[9]:
```
es['sessions'].df.head()
```

[9]:
```
   session_id           join_date   device  customer_id zip_code  \
1           1 2012-04-15 23:31:04  desktop            2    13244
2           2 2010-07-17 05:27:50   mobile            5    60091
3           3 2011-04-08 20:08:14   mobile            4    60091
4           4 2011-04-17 10:48:33   mobile            1    60091
5           5 2011-04-08 20:08:14   mobile            4    60091

        session_start
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
```

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.

[10]:
```
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.plot()
```

[10]:

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.

[11]:
```
es = es.normalize_entity(base_entity_id="transactions",
                         new_entity_id="sessions",
                         index="session_id",
                         make_time_index="session_start",
                         additional_variables=["session_start"])
es.plot()
```

[11]:

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.

[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()
```

[13]:
```
   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  \
0            2  desktop    2014-01-01    13244 2012-04-15 23:31:04
1            2  desktop    2014-01-01    13244 2012-04-15 23:31:04
2            2  desktop    2014-01-01    13244 2012-04-15 23:31:04
3            2  desktop    2014-01-01    13244 2012-04-15 23:31:04
4            2  desktop    2014-01-01    13244 2012-04-15 23:31:04

  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()
```

[14]:
```
  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.

[15]:
```
transactions_df['product_id_device'] = transactions_df['product_id'].astype(str) + '␣
↪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`.

[16]:
```
es = ft.EntitySet(id="customer_data")
es = es.entity_from_dataframe(entity_id="transactions",
                              dataframe=transactions_df,
                              index="transaction_id",
                              time_index="transaction_time",
                              variable_types={"product_id": ft.variable_types.
↪Categorical,
                                              "product_id_device": ft.variable_types.
↪Categorical,
                                              "zip_code": ft.variable_types.ZIPCode})

es = es.entity_from_dataframe(entity_id="products",
                              dataframe=products_df,
                              index="product_id")
es = es.normalize_entity(base_entity_id="transactions",
                         new_entity_id="sessions",
                         index="session_id",
                         additional_variables=["device", "product_id_device",
↪"customer_id"])
es = es.normalize_entity(base_entity_id="sessions",
                         new_entity_id="customers",
                         index="customer_id")
es
```

[16]:
```
Entityset: customer_data
  Entities:
    transactions [Rows: 500, Columns: 9]
    products [Rows: 5, Columns: 2]
    sessions [Rows: 35, Columns: 5]
    customers [Rows: 5, Columns: 2]
  Relationships:
    transactions.session_id -> sessions.session_id
    sessions.customer_id -> customers.customer_id
```

Now, we are ready to add our interesting values.

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

[17]:
```
interesting_values = transactions_df['product_id_device'].unique().tolist()
interesting_values
```

[17]:
```
['5 and desktop',
 '2 and desktop',
 '3 and desktop',
 '4 and desktop',
 '1 and desktop',
 '1 and tablet',
 '3 and tablet',
 '5 and tablet',
 '2 and tablet',
 '4 and tablet',
 '4 and mobile',
 '2 and mobile',
 '3 and mobile',
 '5 and mobile',
 '1 and mobile']
```

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.

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

Now we can run DFS.

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

[19]:
```
             COUNT(sessions)  COUNT(transactions)  \
customer_id
2                          7                   93
5                          6                   79
4                          8                  109
1                          8                  126
3                          6                   93

             COUNT(sessions WHERE product_id_device = 5 and tablet)  \
customer_id
2                                                          1.0
5                                                          0.0
4                                                          1.0
1                                                          0.0
3                                                          0.0

             COUNT(sessions WHERE product_id_device = 2 and desktop)  \
customer_id
2                                                          0.0
5                                                          0.0
4                                                          1.0
1                                                          1.0
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 = 2 and mobile)  \
customer_id
2                                                          1.0
5                                                          0.0
4                                                          2.0
1                                                          0.0
3                                                          0.0

             COUNT(sessions WHERE product_id_device = 1 and mobile)  \
customer_id
2                                                          0.0
```

(continues on next page)

(continued from previous page)

```
5                                                          1.0
4                                                          1.0
1                                                          0.0
3                                                          0.0

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

             COUNT(sessions WHERE product_id_device = 1 and desktop)  \
customer_id
2                                                          1.0
5                                                          0.0
4                                                          0.0
1                                                          0.0
3                                                          2.0

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

             COUNT(transactions WHERE sessions.product_id_device = 1 and mobile)  \
customer_id
2                                                          0.0
5                                                         18.0
4                                                         15.0
1                                                          0.0
3                                                          0.0

             COUNT(transactions WHERE sessions.product_id_device = 5 and tablet)  \
customer_id
2                                                         13.0
5                                                          0.0
4                                                         18.0
1                                                          0.0
3                                                          0.0

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

             COUNT(transactions WHERE sessions.product_id_device = 1 and desktop)  \
customer_id
2                                                          8.0
5                                                          0.0
```

(continues on next page)

(continued from previous page)

```
4                                                             0.0
1                                                             0.0
3                                                            33.0

             COUNT(transactions WHERE sessions.product_id_device = 5 and desktop)  \
customer_id
2                                                              16
5                                                              15
4                                                              10
1                                                              12
3                                                              29

             COUNT(transactions WHERE sessions.product_id_device = 4 and tablet)  \
customer_id
2                                                             0.0
5                                                            14.0
4                                                             0.0
1                                                            27.0
3                                                             0.0

             COUNT(transactions WHERE sessions.product_id_device = 2 and mobile)  \
customer_id
2                                                            13.0
5                                                             0.0
4                                                            23.0
1                                                             0.0
3                                                             0.0

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

             COUNT(transactions WHERE sessions.product_id_device = 4 and desktop)  \
customer_id
2                                                            10.0
5                                                            14.0
4                                                            18.0
1                                                             0.0
3                                                             0.0

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

[5 rows x 32 columns]
```

To better understand the `where` clause features, let's examine one of those features. The feature `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)**.

[20]:
```
feature_matrix[["COUNT(sessions WHERE product_id_device = 5 and tablet)"]]
```

[20]:
```
                COUNT(sessions WHERE product_id_device = 5 and tablet)
customer_id
2                                                            1.0
5                                                            0.0
4                                                            1.0
1                                                            0.0
3                                                            0.0
```

Can I create an `EntitySet` using Dask or Koalas dataframes? (BETA)

Support for Dask EntitySets and Koalas EntitySets is still in Beta - if you encounter any errors using either of these approaches, please let us know by creating a new issue on Github.

Yes! Featuretools supports creating an `EntitySet` from Dask dataframes or from Koalas dataframes. You can simply follow the same process you would when creating an `EntitySet` from pandas dataframes.

There are some limitations to be aware of when using Dask or Koalas dataframes. When creating an `Entity`, variable type inference is not performed as it is for pandas entities, so the user must supply a list of variable types during creation. Also, other quality checks are not performed, such as checking for unique index values. An `EntitySet` must be created entirely of one type of entity (Dask, Koalas, or pandas) - you cannot mix pandas entities, Dask entities, and Koalas entitites with each other in the same `EntitySet`.

For more information on creating an `EntitySet` from Dask dataframes or from Koalas dataframes, see the *Using Dask EntitySets* and the *Using Koalas EntitySets* guides.

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.

[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
```

[21]:
```
Entityset: customer_data
  Entities:
    transactions [Rows: 500, Columns: 11]
  Relationships:
    No relationships
```

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.

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

[22]:
```
[<Feature: session_id>,
 <Feature: product_id>,
 <Feature: amount>,
 <Feature: customer_id>,
 <Feature: device>,
 <Feature: zip_code>,
 <Feature: DAY(date_of_birth)>,
 <Feature: DAY(join_date)>,
 <Feature: DAY(session_start)>,
 <Feature: DAY(transaction_time)>,
 <Feature: MONTH(date_of_birth)>,
 <Feature: MONTH(join_date)>,
 <Feature: MONTH(session_start)>,
 <Feature: MONTH(transaction_time)>,
 <Feature: WEEKDAY(date_of_birth)>,
 <Feature: WEEKDAY(join_date)>,
 <Feature: WEEKDAY(session_start)>,
 <Feature: WEEKDAY(transaction_time)>,
 <Feature: YEAR(date_of_birth)>,
 <Feature: YEAR(join_date)>,
 <Feature: YEAR(session_start)>,
 <Feature: YEAR(transaction_time)>]
```

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.

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

[23]:
```
Entityset: customer_data
  Entities:
    transactions [Rows: 500, Columns: 11]
    products [Rows: 5, Columns: 2]
  Relationships:
    No relationships
```

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

Solution #2 - You can normalize an existing entity.

[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"])
es
```

[24]:
```
Entityset: customer_data
  Entities:
    transactions [Rows: 500, Columns: 7]
    products [Rows: 5, Columns: 2]
    sessions [Rows: 35, Columns: 6]
  Relationships:
    transactions.session_id -> sessions.session_id
```

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.

[25]:
```
feature_matrix, feature_defs = ft.dfs(entityset=es, target_entity="transactions")
feature_defs[:-10]
```

[25]:
```
[<Feature: session_id>,
 <Feature: product_id>,
 <Feature: amount>,
 <Feature: DAY(date_of_birth)>,
 <Feature: DAY(session_start)>,
 <Feature: DAY(transaction_time)>,
 <Feature: MONTH(date_of_birth)>,
 <Feature: MONTH(session_start)>,
 <Feature: MONTH(transaction_time)>,
 <Feature: WEEKDAY(date_of_birth)>,
 <Feature: WEEKDAY(session_start)>,
 <Feature: WEEKDAY(transaction_time)>,
 <Feature: YEAR(date_of_birth)>,
 <Feature: YEAR(session_start)>,
 <Feature: YEAR(transaction_time)>,
 <Feature: sessions.device>,
 <Feature: sessions.customer_id>,
 <Feature: sessions.zip_code>,
 <Feature: sessions.COUNT(transactions)>,
 <Feature: sessions.MAX(transactions.amount)>,
 <Feature: sessions.MEAN(transactions.amount)>,
 <Feature: sessions.MIN(transactions.amount)>,
 <Feature: sessions.MODE(transactions.product_id)>,
 <Feature: sessions.NUM_UNIQUE(transactions.product_id)>,
 <Feature: sessions.SKEW(transactions.amount)>]
```

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`.

```
# setting n_jobs to -1 will use all cores

feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_entity="customers",
                                      n_jobs=-1)


feature_matrix, feature_defs = ft.calculate_feature_matrix(entityset=es,
                                                           features=feature_defs,
                                                           n_jobs=-1)
```

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:

```
[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 that maps a primitive or a tuple of primitives to a dictionary containing options for the primitive(s). A primitive or tuple of primitives can also be mapped to a list of option dictionaries if the primitive(s) takes multiple inputs. The primitive keys can be the string names of the primitive, the primitive class, or specific instances of the primitive. 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:

```
[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"]}},
                                                         # For mode, ignore the
↪"sessions" entity and only include "brands" in the
                                                         # "products" entity and
↪"product_id" in the "transactions" entity
                                                         ("count", "mean"): {"include_
↪entities": ["sessions", "transactions"]}
                                                         # For count and mean, only␣
↪include the entities "sessions" and "transactions"
                                                         })
```

Note that if options are given for a specific instance of a primitive and for the primitive generally (either by string name or class), the instances with their own options will not use the generic options. For example, in this case:

```
special_mean = Mean()
options = {
    special_mean: {'include_entities: ['customers']},
    'mean': {'include_entities: ['sessions']}
```

the primitive `special_mean` will not use the entity `sessions` because it's options have it only include `customers`. Every other instance of the `Mean` primitive will use the `'mean'` options.

For 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`.

[28]:
```
es = ft.demo.load_mock_customer(return_entityset=True)
es['customers'].time_index
```

[28]:
```
'join_date'
```

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.

[29]:
```
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()
```

[29]:
```
                                 zip_code  COUNT(sessions)  \
customer_id time
1           2014-01-01 04:00:00    60091                1
2           2014-01-01 05:00:00    13244                1
3           2014-01-01 06:00:00    13244                2
1           2014-01-01 08:00:00    60091                1
```

(continues on next page)

(continued from previous page)

```
                                MODE(sessions.device)  \
customer_id time
1           2014-01-01 04:00:00                tablet
2           2014-01-01 05:00:00                tablet
3           2014-01-01 06:00:00               desktop
1           2014-01-01 08:00:00                mobile

                                 NUM_UNIQUE(sessions.device)  \
customer_id time
1           2014-01-01 04:00:00                            1
2           2014-01-01 05:00:00                            1
3           2014-01-01 06:00:00                            1
1           2014-01-01 08:00:00                            1

                                 COUNT(transactions)  \
customer_id time
1           2014-01-01 04:00:00                   12
2           2014-01-01 05:00:00                   13
3           2014-01-01 06:00:00                   12
1           2014-01-01 08:00:00                   16

                                 MAX(transactions.amount)  \
customer_id time
1           2014-01-01 04:00:00                    139.09
2           2014-01-01 05:00:00                    118.85
3           2014-01-01 06:00:00                    128.26
1           2014-01-01 08:00:00                    126.11

                                 MEAN(transactions.amount)  \
customer_id time
1           2014-01-01 04:00:00                  85.469167
2           2014-01-01 05:00:00                  77.304615
3           2014-01-01 06:00:00                  81.747500
1           2014-01-01 08:00:00                  88.755625

                                 MIN(transactions.amount)  \
customer_id time
1           2014-01-01 04:00:00                      6.78
2           2014-01-01 05:00:00                     21.82
3           2014-01-01 06:00:00                     20.06
1           2014-01-01 08:00:00                     11.62

                                 MODE(transactions.product_id)  \
customer_id time
1           2014-01-01 04:00:00                              4
2           2014-01-01 05:00:00                              1
3           2014-01-01 06:00:00                              3
1           2014-01-01 08:00:00                              4

                                 NUM_UNIQUE(transactions.product_id)  ...  \
customer_id time                                                      ...
1           2014-01-01 04:00:00                                    5  ...
2           2014-01-01 05:00:00                                    5  ...
3           2014-01-01 06:00:00                                    5  ...
1           2014-01-01 08:00:00                                    5  ...

                                 SUM(sessions.MEAN(transactions.amount))  \
```

(continues on next page)

(continued from previous page)

```
customer_id time
1           2014-01-01 04:00:00                                85.469167
2           2014-01-01 05:00:00                                77.304615
3           2014-01-01 06:00:00                               172.597273
1           2014-01-01 08:00:00                                88.755625

                                 SUM(sessions.MIN(transactions.amount))  \
customer_id time
1           2014-01-01 04:00:00                                    6.78
2           2014-01-01 05:00:00                                   21.82
3           2014-01-01 06:00:00                                  111.82
1           2014-01-01 08:00:00                                   11.62

                                 SUM(sessions.NUM_UNIQUE(transactions.product_id))  \
customer_id time
1           2014-01-01 04:00:00                                                  5
2           2014-01-01 05:00:00                                                  5
3           2014-01-01 06:00:00                                                  6
1           2014-01-01 08:00:00                                                  5

                                 SUM(sessions.SKEW(transactions.amount))  \
customer_id time
1           2014-01-01 04:00:00                                -0.830975
2           2014-01-01 05:00:00                                -0.314918
3           2014-01-01 06:00:00                                -0.289466
1           2014-01-01 08:00:00                                -1.038434

                                 SUM(sessions.STD(transactions.amount))  \
customer_id time
1           2014-01-01 04:00:00                               39.825249
2           2014-01-01 05:00:00                               33.725036
3           2014-01-01 06:00:00                               35.704680
1           2014-01-01 08:00:00                               32.324534

                                 MODE(transactions.sessions.customer_id)  \
customer_id time
1           2014-01-01 04:00:00                                        1
2           2014-01-01 05:00:00                                        2
3           2014-01-01 06:00:00                                        3
1           2014-01-01 08:00:00                                        1

                                 MODE(transactions.sessions.device)  \
customer_id time
1           2014-01-01 04:00:00                              tablet
2           2014-01-01 05:00:00                              tablet
3           2014-01-01 06:00:00                             desktop
1           2014-01-01 08:00:00                              mobile

                                 NUM_UNIQUE(transactions.sessions.customer_id)  \
customer_id time
1           2014-01-01 04:00:00                                              1
2           2014-01-01 05:00:00                                              1
3           2014-01-01 06:00:00                                              1
1           2014-01-01 08:00:00                                              1

                                 NUM_UNIQUE(transactions.sessions.device)  \
customer_id time
```

(continues on next page)

(continued from previous page)

```
1           2014-01-01 04:00:00                                          1
2           2014-01-01 05:00:00                                          1
3           2014-01-01 06:00:00                                          1
1           2014-01-01 08:00:00                                          1

                                 label
customer_id time
1           2014-01-01 04:00:00   True
2           2014-01-01 05:00:00   True
3           2014-01-01 06:00:00  False
1           2014-01-01 08:00:00   True

[4 rows x 78 columns]
```

Above, we ran DFS with `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:

```
[30]: transactions_df = ft.demo.load_mock_customer(return_single_table=True)

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

feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_entity="transactions",
                                      trans_primitives=['time_since', 'day', 'is_
↪weekend',
                                                        'cum_min', 'minute', 'weekday
↪',
                                                        'percentile', 'year', 'week',
                                                        'cum_mean'])
```

Before we examine the output, let's look at our original single table.

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

```
[31]:    transaction_id  session_id    transaction_time product_id  amount  \
0             298           1 2014-01-01 00:00:00          5  127.64
1              10           1 2014-01-01 00:09:45          5   57.39
2             495           1 2014-01-01 00:14:05          5   69.45
3             460          10 2014-01-01 02:33:50          5  123.19
4             302          10 2014-01-01 02:37:05          5   64.47

   customer_id   device        session_start  zip_code           join_date  \
0            2  desktop 2014-01-01 00:00:00     13244 2012-04-15 23:31:04
1            2  desktop 2014-01-01 00:00:00     13244 2012-04-15 23:31:04
2            2  desktop 2014-01-01 00:00:00     13244 2012-04-15 23:31:04
```

(continues on next page)

(continued from previous page)

```
3            2   tablet 2014-01-01 02:31:40    13244 2012-04-15 23:31:04
4            2   tablet 2014-01-01 02:31:40    13244 2012-04-15 23:31:04

  date_of_birth brand
0    1986-08-18     A
1    1986-08-18     A
2    1986-08-18     A
3    1986-08-18     A
4    1986-08-18     A
```

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()
```

[32]:
```
                session_id product_id  amount  customer_id   device zip_code  \
transaction_id
298                      1          5  127.64            2  desktop    13244
2                        1          2  109.48            2  desktop    13244
308                      1          3   95.06            2  desktop    13244
116                      1          4   78.92            2  desktop    13244
371                      1          3   31.54            2  desktop    13244

               brand  CUM_MEAN(amount)  CUM_MEAN(customer_id)  \
transaction_id
298                A        127.640000                    2.0
2                  B        118.560000                    2.0
308                B        110.726667                    2.0
116                B        102.775000                    2.0
371                B         88.528000                    2.0

                CUM_MEAN(session_id)  ...  WEEK(session_start)  \
transaction_id                        ...
298                              1.0  ...                    1
2                                1.0  ...                    1
308                              1.0  ...                    1
116                              1.0  ...                    1
371                              1.0  ...                    1

                WEEK(transaction_time)  WEEKDAY(date_of_birth)  \
transaction_id
298                                  1                       0
2                                    1                       0
308                                  1                       0
116                                  1                       0
371                                  1                       0

                WEEKDAY(join_date)  WEEKDAY(session_start)  \
transaction_id
298                              6                       2
2                                6                       2
308                              6                       2
116                              6                       2
371                              6                       2

                WEEKDAY(transaction_time)  YEAR(date_of_birth)  \
transaction_id
```

(continues on next page)

(continued from previous page)

```
298                                  2                1986
2                                    2                1986
308                                  2                1986
116                                  2                1986
371                                  2                1986

                YEAR(join_date)  YEAR(session_start)  YEAR(transaction_time)
transaction_id
298                        2012                 2014                    2014
2                          2012                 2014                    2014
308                        2012                 2014                    2014
116                        2012                 2014                    2014
371                        2012                 2014                    2014

[5 rows x 44 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:

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

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

[33]:
```
from featuretools.autonormalize import autonormalize as an
es = an.normalize_entity(es)
es.plot()
```

```
100%|| 10/10 [00:01<00:00,  5.05it/s]
```

[33]:

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.

[34]:
```
feature_matrix, feature_defs = ft.dfs(entityset=es,
                                      target_entity="transaction_id",
                                      trans_primitives=[])
feature_matrix.head()
```

[34]:
```
                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
```

(continues on next page)

(continued from previous page)

```
                session_id.COUNT(transaction_id)  \
transaction_id
298                                           16
2                                             16
308                                           16
116                                           16
371                                           16

                session_id.MAX(transaction_id.amount)  \
transaction_id
298                                              141.66
2                                                141.66
308                                              141.66
116                                              141.66
371                                              141.66

                session_id.MEAN(transaction_id.amount)  \
transaction_id
298                                            76.813125
2                                              76.813125
308                                            76.813125
116                                            76.813125
371                                            76.813125

                session_id.MIN(transaction_id.amount)  ...  \
transaction_id                                         ...
298                                               20.91  ...
2                                                 20.91  ...
308                                               20.91  ...
116                                               20.91  ...
371                                               20.91  ...

                session_id.customer_id.zip_code  \
transaction_id
298                                        13244
2                                          13244
308                                        13244
116                                        13244
371                                        13244

                product_id.COUNT(transaction_id)  \
transaction_id
298                                          104
2                                             92
308                                           96
116                                          106
371                                           96

                product_id.MAX(transaction_id.amount)  \
transaction_id
298                                              149.02
2                                                149.95
308                                              148.31
116                                              146.46
371                                              148.31

                product_id.MEAN(transaction_id.amount)  \
```

(continues on next page)

(continued from previous page)

```
transaction_id
298                                         76.264904
2                                           76.319891
308                                         73.001250
116                                         76.311038
371                                         73.001250

                product_id.MIN(transaction_id.amount)  \
transaction_id
298                                             5.91
2                                               5.73
308                                             5.89
116                                             5.81
371                                             5.89

               product_id.MODE(transaction_id.session_id)  \
transaction_id
298                                                     4
2                                                      28
308                                                     1
116                                                    29
371                                                     1

                product_id.NUM_UNIQUE(transaction_id.session_id)  \
transaction_id
298                                                           34
2                                                             34
308                                                           35
116                                                           34
371                                                           35

                product_id.SKEW(transaction_id.amount)  \
transaction_id
298                                          0.098248
2                                            0.151934
308                                          0.223938
116                                         -0.132077
371                                          0.223938

                product_id.STD(transaction_id.amount)  \
transaction_id
298                                        42.131902
2                                          46.336308
308                                        38.871405
116                                        42.492501
371                                        38.871405

                product_id.SUM(transaction_id.amount)
transaction_id
298                                          7931.55
2                                            7021.43
308                                          7008.12
116                                          8088.97
371                                          7008.12

[5 rows x 25 columns]
```

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()
```

```
[35]:    customer_id  age gender         signup_date  labels
0             1   40      m 2014-01-01 01:41:50    True
1             2   50      f 2014-01-01 02:06:50   False
2             3   10      m 2014-01-01 02:31:50    True
3             4   20      f 2014-01-01 02:56:50   False
4             5   30      f 2014-01-01 03:21:50    True
```

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
```

```
[36]: 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
```

```
[37]:              age gender  labels  DAY(signup_date)  MONTH(signup_date)  \
customer_id
1             40      m    True                 1                   1
```

(continues on next page)

(continued from previous page)

```
2             50     f   False                1                   1
3             10     m    True                1                   1
4             20     f   False                1                   1
5             30     f    True                1                   1

             WEEKDAY(signup_date)  YEAR(signup_date)
customer_id
1                               2               2014
2                               2               2014
3                               2               2014
4                               2               2014
5                               2               2014
```

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()
```

```
[38]:              age  gender = f  gender = m  gender is unknown  labels  \
      customer_id
      1             40       False        True              False    True
      2             50        True       False              False   False
      3             10       False        True              False    True
      4             20        True       False              False   False
      5             30        True       False              False    True

                   DAY(signup_date) = 1  DAY(signup_date) is unknown  \
      customer_id
      1                            True                        False
      2                            True                        False
      3                            True                        False
      4                            True                        False
      5                            True                        False

                   MONTH(signup_date) = 1  MONTH(signup_date) is unknown  \
      customer_id
      1                              True                          False
      2                              True                          False
      3                              True                          False
      4                              True                          False
      5                              True                          False

                   WEEKDAY(signup_date) = 2  WEEKDAY(signup_date) is unknown  \
      customer_id
      1                                True                            False
      2                                True                            False
      3                                True                            False
      4                                True                            False
      5                                True                            False

                   YEAR(signup_date) = 2014  YEAR(signup_date) is unknown
      customer_id
      1                                True                         False
      2                                True                         False
      3                                True                         False
      4                                True                         False
```

(continues on next page)

(continued from previous page)

```
5                                True                          False
```

Notice how the the whole feature matrix only inclues numeric values now.

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

[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")

# Use the feature definitions from earlier
feature_matrix_enc_test = ft.calculate_feature_matrix(features=features_enc,
                                                      entityset=es_test_data)

feature_matrix_enc_test.head()
```

[39]:
```
             age  gender = f  gender = m  gender is unknown  labels  \
customer_id
6             20        True       False              False     NaN
7             25       False        True              False     NaN
8             55       False        True              False     NaN
9             22       False        True              False     NaN
10            35       False        True              False     NaN

             DAY(signup_date) = 1  DAY(signup_date) is unknown  \
customer_id
6                            True                        False
7                            True                        False
8                            True                        False
9                            True                        False
10                           True                        False

             MONTH(signup_date) = 1  MONTH(signup_date) is unknown  \
customer_id
6                              True                          False
7                              True                          False
8                              True                          False
9                              True                          False
10                             True                          False

             WEEKDAY(signup_date) = 2  WEEKDAY(signup_date) is unknown  \
customer_id
6                                True                            False
7                                True                            False
8                                True                            False
9                                True                            False
```

(continues on next page)

(continued from previous page)

```
10                              True                           False

             YEAR(signup_date) = 2014  YEAR(signup_date) is unknown
customer_id
6                               True                        False
7                               True                        False
8                               True                        False
9                               True                        False
10                              True                        False
```

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.

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()
```

[40]:
```
   customer_id                time
0            1 2014-01-01 01:41:50
1            2 2014-01-01 02:06:50
2            3 2014-01-01 02:31:50
3            4 2014-01-01 02:56:50
4            5 2014-01-01 03:21:50
```

[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')})

es_train_test_data = ft.EntitySet(id="customer_train_test_data")
es_train_test_data = es_train_test_data.entity_from_dataframe(entity_id="customers",
                                                              dataframe=train_test_
↪data,
                                                              index="customer_id",
                                                              time_index="signup_date
↪")

feature_matrix_train_test, features = ft.dfs(entityset=es_train_test_data,
                                             target_entity="customers",
                                             cutoff_time=cutoff_times,
                                             cutoff_time_in_index=True)
feature_matrix_train_test.head()
```

```
[41]:                                  age  gender  DAY(signup_date)  \
customer_id time
1           2014-01-01 01:41:50   20      f                 1
2           2014-01-01 02:06:50   25      m                 1
3           2014-01-01 02:31:50   55      m                 1
4           2014-01-01 02:56:50   22      m                 1
5           2014-01-01 03:21:50   35      m                 1

                                 MONTH(signup_date)  WEEKDAY(signup_date)  \
customer_id time
1           2014-01-01 01:41:50                   1                     4
2           2014-01-01 02:06:50                   1                     4
3           2014-01-01 02:31:50                   1                     4
4           2014-01-01 02:56:50                   1                     4
5           2014-01-01 03:21:50                   1                     4

                                 YEAR(signup_date)
customer_id time
1           2014-01-01 01:41:50               2010
2           2014-01-01 02:06:50               2010
3           2014-01-01 02:31:50               2010
4           2014-01-01 02:56:50               2010
5           2014-01-01 03:21:50               2010
```

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.

```
[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
```

```
[43]:             zip_code  TIME_SINCE_PREVIOUS(join_date)
customer_id
5              60091                             NaN
4              60091                      22948824.0
1              60091                        744019.0
3              13244                      10212841.0
2              13244                      21282510.0
```

Now, let's use the primitive object.

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

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

(continues on next page)

(continued from previous page)

```
                                              target_entity="customers",
                                              agg_primitives=[],
                                              trans_primitives=[TimeSincePrevious])
feature_matrix
```

[44]:

```
               zip_code  TIME_SINCE_PREVIOUS(join_date)
customer_id
5                 60091                             NaN
4                 60091                      22948824.0
1                 60091                        744019.0
3                 13244                      10212841.0
2                 13244                      21282510.0
```

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).

[45]:

```
from featuretools.primitives import TimeSincePrevious

time_since_previous_in_hours = TimeSincePrevious(unit='hours')

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

[45]:

```
               zip_code  TIME_SINCE_PREVIOUS(join_date, unit=hours)
customer_id
5                 60091                                         NaN
4                 60091                                 6374.673333
1                 60091                                  206.671944
3                 13244                                 2836.900278
2                 13244                                 5911.808333
```

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.

[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():
```

(continues on next page)

(continued from previous page)

```
        features_with_amount.append(x)
features_with_amount[0:5]
```

[46]:
```
[<Feature: MAX(transactions.amount)>,
 <Feature: MEAN(transactions.amount)>,
 <Feature: MIN(transactions.amount)>,
 <Feature: SKEW(transactions.amount)>,
 <Feature: STD(transactions.amount)>]
```

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

[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]
```

[47]:
```
[<Feature: COUNT(sessions)>,
 <Feature: MODE(sessions.device)>,
 <Feature: NUM_UNIQUE(sessions.device)>,
 <Feature: COUNT(transactions)>,
 <Feature: MAX(transactions.amount)>]
```

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.

[48]:
```
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]
```

[48]:
```
[<Feature: MAX(sessions.COUNT(transactions))>,
 <Feature: MAX(sessions.MEAN(transactions.amount))>,
 <Feature: MAX(sessions.MIN(transactions.amount))>,
 <Feature: MAX(sessions.NUM_UNIQUE(transactions.product_id))>,
 <Feature: MAX(sessions.SKEW(transactions.amount))>]
```

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

[49]:
```
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]
```

[49]:
```
[<Feature: COUNT(sessions)>,
 <Feature: NUM_UNIQUE(sessions.device)>,
 <Feature: COUNT(transactions)>,
 <Feature: MAX(transactions.amount)>,
 <Feature: MEAN(transactions.amount)>]
```

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.

[50]:
```
feature_matrix = ft.calculate_feature_matrix(entityset=es,
                                             features=features_with_amount) # change␣
↪to your specific feature list
feature_matrix.head()
```

[50]:
```
             MAX(transactions.amount)  MEAN(transactions.amount)  \
customer_id
5                              149.02                  80.375443
4                              149.95                  80.070459
1                              139.43                  71.631905
3                              149.15                  67.060430
2                              146.81                  77.422366

             MIN(transactions.amount)  SKEW(transactions.amount)  \
customer_id
5                                7.55                  -0.025941
4                                5.73                  -0.036348
1                                5.81                   0.019698
3                                5.89                   0.418230
2                                8.73                   0.098259

             STD(transactions.amount)  SUM(transactions.amount)  \
customer_id
5                           44.095630                   6349.66
4                           45.068765                   8727.68
1                           40.442059                   9025.62
3                           43.683296                   6236.62
2                           37.705178                   7200.28

             MAX(sessions.MEAN(transactions.amount))  \
customer_id
5                                          94.481667
4                                         110.450000
1                                          88.755625
3                                          82.109444
2                                          96.581000

             MAX(sessions.MIN(transactions.amount))  \
customer_id
5                                            20.65
4                                            54.83
1                                            26.36
3                                            20.06
2                                            56.46

             MAX(sessions.SKEW(transactions.amount))  \
customer_id
5                                           0.602209
4                                           0.382868
1                                           0.640252
3                                           0.854976
2                                           0.755711

             MAX(sessions.STD(transactions.amount))  ...  \
customer_id                                          ...
5                                         51.149250  ...
```

(continues on next page)

(continued from previous page)

```
4                                   54.293903  ...
1                                   46.905665  ...
3                                   50.110120  ...
2                                   47.935920  ...

             STD(sessions.MAX(transactions.amount))  \
customer_id
5                                           7.928001
4                                           3.514421
1                                           7.322191
3                                          10.724241
2                                          17.221593

             STD(sessions.MEAN(transactions.amount))  \
customer_id
5                                           11.007471
4                                           13.027258
1                                           13.759314
3                                           11.174282
2                                           11.477071

             STD(sessions.MIN(transactions.amount))  \
customer_id
5                                           4.961414
4                                          16.960575
1                                           6.954507
3                                           5.424407
2                                          15.874374

             STD(sessions.SKEW(transactions.amount))  \
customer_id
5                                            0.415426
4                                            0.387884
1                                            0.589386
3                                            0.429374
2                                            0.509798

             STD(sessions.SUM(transactions.amount))  \
customer_id
5                                         402.775486
4                                         235.992478
1                                         279.510713
3                                         219.021420
2                                         251.609234

             SUM(sessions.MAX(transactions.amount))  \
customer_id
5                                             839.76
4                                            1157.99
1                                            1057.97
3                                             847.63
2                                             931.63

             SUM(sessions.MEAN(transactions.amount))  \
customer_id
5                                          472.231119
4                                          649.657515
```

(continues on next page)

(continued from previous page)

```
1                                         582.193117
3                                         405.237462
2                                         548.905851

             SUM(sessions.MIN(transactions.amount))  \
customer_id
5                                            86.49
4                                           131.51
1                                            78.59
3                                            66.21
2                                           154.60

             SUM(sessions.SKEW(transactions.amount))  \
customer_id
5                                         0.014384
4                                         0.002764
1                                        -0.476122
3                                         2.286086
2                                        -0.277640

             SUM(sessions.STD(transactions.amount))
customer_id
5                                       259.873954
4                                       356.125829
1                                       312.745952
3                                       257.299895
2                                       258.700528

[5 rows x 37 columns]
```

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`.

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

[51]:
```
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.

[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`.

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

[53]:
```
             zip_code  COUNT(sessions)  COUNT(transactions)  \
customer_id
5               60091                6                   79
4               60091                8                  109
1               60091                8                  126
3               13244                6                   93
2               13244                7                   93

             COUNT(sessions WHERE device = mobile)  \
customer_id
5                                                3
4                                                4
1                                                3
3                                                1
2                                                2

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

             COUNT(sessions WHERE device = tablet)  \
customer_id
5                                                1
4                                                1
1                                                3
3                                                1
2                                                2

             COUNT(sessions WHERE customers.zip_code = 13244)  \
customer_id
5                                                         0.0
4                                                         0.0
1                                                         0.0
3                                                         6.0
2                                                         7.0

             COUNT(sessions WHERE customers.zip_code = 60091)  \
customer_id
5                                                         6.0
4                                                         8.0
```
(continues on next page)

(continued from previous page)

```
1                                                            8.0
3                                                            0.0
2                                                            0.0

             COUNT(transactions WHERE sessions.device = desktop)  \
customer_id
5                                                              29
4                                                              38
1                                                              27
3                                                              62
2                                                              34

             COUNT(transactions WHERE sessions.device = mobile)  \
customer_id
5                                                              36
4                                                              53
1                                                              56
3                                                              16
2                                                              31

             COUNT(transactions WHERE sessions.device = tablet)
customer_id
5                                                              14
4                                                              18
1                                                              43
3                                                              15
2                                                              28
```

We have now created some useful features. One example of a useful feature is the `COUNT(sessions WHERE device = tablet)`. This feature tells us how many sessions a customer completed on a tablet.

[54]:
```
feature_matrix[["COUNT(sessions WHERE device = tablet)"]]
```

[54]:
```
             COUNT(sessions WHERE device = tablet)
customer_id
5                                                  1
4                                                  1
1                                                  3
3                                                  1
2                                                  2
```

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`.

[55]:
```
import pandas as pd
import featuretools as ft

df = pd.DataFrame({
    "id": [1, 2, 3, 4, 5, 6],
```

(continues on next page)

(continued from previous page)

```
    "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()
```

[55]:

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             5
2      b    1             6
3      a   10            16
4      c   20            36
5      a    6            42
6      b   23            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=[],
                                      groupby_trans_primitives=["cum_sum"])

feature_matrix
```

[58]:
```
   group  val  CUM_SUM(val) by group
id
1      a    5                    5.0
2      b    1                    1.0
3      a   10                   15.0
4      c   20                   20.0
5      a    6                   21.0
6      b   23                   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.

[59]:
```
feature_matrix, feature_defs = ft.dfs(target_entity="observations",
                                      entityset=es,
                                      agg_primitives=["sum"],
                                      trans_primitives=[],
                                      cutoff_time_in_index=True,
                                      groupby_trans_primitives=[])

feature_matrix
```

[59]:
```
                              group  val  groups.SUM(observations.val)
id time
1  2020-10-19 16:53:34.719162     a    5                            21
2  2020-10-19 16:53:34.719162     b    1                            24
3  2020-10-19 16:53:34.719162     a   10                            21
4  2020-10-19 16:53:34.719162     c   20                            20
5  2020-10-19 16:53:34.719162     a    6                            21
6  2020-10-19 16:53:34.719162     b   23                            24
```

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).

[60]:
```
cutoff_time = df[["id", "time_index"]]
cutoff_time
```

[60]:
```
   id time_index
0   1 2019-01-01
1   2 2019-01-02
2   3 2019-01-03
3   4 2019-01-04
```

(continues on next page)

(continued from previous page)

```
4   5 2019-01-05
5   6 2019-01-06
```

[61]:
```
feature_matrix, feature_defs = ft.dfs(target_entity="observations",
                                      entityset=es,
                                      agg_primitives=["sum"],
                                      trans_primitives=[],
                                      groupby_trans_primitives=[],
                                      cutoff_time_in_index=True,
                                      cutoff_time=cutoff_time)

feature_matrix
```

[61]:
```
              group  val  groups.SUM(observations.val)
id time
1  2019-01-01     a    5                             5
2  2019-01-02     b    1                             1
3  2019-01-03     a   10                            15
4  2019-01-04     c   20                            20
5  2019-01-05     a    6                            21
6  2019-01-06     b   23                            24
```

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, and if the primitive can be used with entitysets created from Dask dataframes. You can also visit primitives.featurelabs.com to obtain a list of all available primitives.

[62]:
```
df_primitives = ft.list_primitives()
df_primitives.head()
```

[62]:
```
              name         type  dask_compatible  koalas_compatible  \
0  time_since_last  aggregation            False              False
1             mean  aggregation             True               True
2              any  aggregation             True              False
3     percent_true  aggregation             True              False
4            first  aggregation            False              False

                                         description
0  Calculates the time elapsed since the last dat...
1         Computes the average for a list of values.
2       Determines if any value is 'True' in a list.
3           Determines the percent of `True` values.
4              Determines the first value in a list.
```

[63]:
```
df_primitives.tail()
```

[63]:
```
                                name       type  dask_compatible  \
74                         haversine  transform            False
75                           cum_sum  transform            False
76  scalar_subtract_numeric_feature  transform             True
77                             equal  transform             True
78                            second  transform             True

    koalas_compatible                                        description
```

(continues on next page)

(continued from previous page)

```
74              False  Calculates the approximate haversine distance ...
75              False                    Calculates the cumulative sum.
76               True  Subtract each value in the list from a given s...
77               True  Determines if values in one list are equal to ...
78               True        Determines the seconds value of a datetime.
```

What primitives can I use when creating a feature matrix from a Dask `EntitySet`? (BETA)

Support for Dask EntitySets is still in Beta - if you encounter any errors using this approach, please let us know by creating a new issue on Github.

When creating a feature matrix from a Dask `EntitySet`, only certain primitives can be used. Computation of certain features is quite expensive in a distributed environment, and as a result only a subset of Featuretools primitives are currently supported when using a Dask `EntitySet`.

The table returned by `featuretools.list_primitives()` will contain a column labeled `dask_compatible`. Any primitive that has a value of `True` in this column can be used safely when computing a feature matrix from a Dask `EntitySet`.

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:

[64]:
```
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_
↪first],
                                      trans_primitives=[time_since, time_since_
↪previous])
```

Above, we changed the units to the following: - minutes for `TimeSince` - hours for `TimeSincePrevious` - days for `TimeSinceLast` - years for `TimeSinceFirst`.

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

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

[65]:
```
            zip_code  TIME_SINCE_FIRST(sessions.session_start, unit=years)  \
customer_id
5              60091                                          6.803772
```

(continues on next page)

(continued from previous page)

```
4             60091                                            6.803752
1             60091                                            6.803721
3             13244                                            6.803616
2             13244                                            6.803805

             TIME_SINCE_LAST(sessions.session_start, unit=days)  \
customer_id
5                                                 2483.369099
4                                                 2483.480441
1                                                 2483.405210
3                                                 2483.339758
2                                                 2483.363080

             TIME_SINCE_FIRST(transactions.transaction_time, unit=years)  \
customer_id
5                                                    6.803772
4                                                    6.803752
1                                                    6.803721
3                                                    6.803616
2                                                    6.803805

             TIME_SINCE_LAST(transactions.transaction_time, unit=days)  \
customer_id
5                                                 2483.363832
4                                                 2483.473670
1                                                 2483.393925
3                                                 2483.328474
2                                                 2483.354052

             TIME_SINCE(date_of_birth, unit=minutes)  \
customer_id
5                                      1.905509e+07
4                                      7.458774e+06
1                                      1.381061e+07
3                                      8.895894e+06
2                                      1.797365e+07

             TIME_SINCE(join_date, unit=minutes)  \
customer_id
5                                  5.396366e+06
4                                  5.013885e+06
1                                  5.001485e+06
3                                  4.831271e+06
2                                  4.476563e+06

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

             TIME_SINCE_FIRST(transactions.sessions.session_start, unit=years)  \
customer_id
5                                                    6.803772
4                                                    6.803752
```

(continues on next page)

(continued from previous page)

```
1                                                       6.803721
3                                                       6.803616
2                                                       6.803805

             TIME_SINCE_LAST(transactions.sessions.session_start, unit=days)
customer_id
5                                                    2483.369099
4                                                    2483.480441
1                                                    2483.405210
3                                                    2483.339758
2                                                    2483.363080
```

There are now features where time unit is different from the default of seconds, such as `TIME_SINCE_LAST(sessions.session_start, unit=days)`, and `TIME_SINCE_FIRST(sessions.session_start, unit=years)`.

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 **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.

[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]})
train_data.head()
```

[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 `EntitySet` for the train data, and create our features. To prevent label leakage, we will use cutoff times (see *earlier question*).

[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",
                                        cutoff_time=cutoff_times,
```

(continues on next page)

(continued from previous page)

```
                                        cutoff_time_in_index=True)
feature_matrix_train.head()
```

[67]:

```
                                 age gender  labels  DAY(signup_date)  \
customer_id time
1           2014-01-01 01:41:50   20      f   False                 1
2           2014-01-01 02:06:50   25      m    True                 1
3           2014-01-01 02:31:50   55      m    True                 1
4           2014-01-01 02:56:50   22      m   False                 1
5           2014-01-01 03:21:50   35      m   False                 1

                                 MONTH(signup_date)  WEEKDAY(signup_date)  \
customer_id time
1           2014-01-01 01:41:50                   1                     4
2           2014-01-01 02:06:50                   1                     4
3           2014-01-01 02:31:50                   1                     4
4           2014-01-01 02:56:50                   1                     4
5           2014-01-01 03:21:50                   1                     4

                                 YEAR(signup_date)
customer_id time
1           2014-01-01 01:41:50               2010
2           2014-01-01 02:06:50               2010
3           2014-01-01 02:31:50               2010
4           2014-01-01 02:56:50               2010
5           2014-01-01 03:21:50               2010
```

We will also encode our feature matrix to compatible for machine learning algorithms.

[68]:

```
feature_matrix_train_enc, feature_enc = ft.encode_features(feature_matrix_train,␣
↪features)
feature_matrix_train_enc.head()
```

[68]:

```
                                 age  gender = m  gender = f  \
customer_id time
1           2014-01-01 01:41:50   20       False        True
2           2014-01-01 02:06:50   25        True       False
3           2014-01-01 02:31:50   55        True       False
4           2014-01-01 02:56:50   22        True       False
5           2014-01-01 03:21:50   35        True       False

                                 gender is unknown  labels  \
customer_id time
1           2014-01-01 01:41:50              False   False
2           2014-01-01 02:06:50              False    True
3           2014-01-01 02:31:50              False    True
4           2014-01-01 02:56:50              False   False
5           2014-01-01 03:21:50              False   False

                                 DAY(signup_date) = 1  \
customer_id time
1           2014-01-01 01:41:50                  True
2           2014-01-01 02:06:50                  True
3           2014-01-01 02:31:50                  True
4           2014-01-01 02:56:50                  True
5           2014-01-01 03:21:50                  True

                                 DAY(signup_date) is unknown  \
```

(continues on next page)

(continued from previous page)

```
customer_id time
1           2014-01-01 01:41:50                        False
2           2014-01-01 02:06:50                        False
3           2014-01-01 02:31:50                        False
4           2014-01-01 02:56:50                        False
5           2014-01-01 03:21:50                        False

                                 MONTH(signup_date) = 1  \
customer_id time
1           2014-01-01 01:41:50                    True
2           2014-01-01 02:06:50                    True
3           2014-01-01 02:31:50                    True
4           2014-01-01 02:56:50                    True
5           2014-01-01 03:21:50                    True

                                 MONTH(signup_date) is unknown  \
customer_id time
1           2014-01-01 01:41:50                          False
2           2014-01-01 02:06:50                          False
3           2014-01-01 02:31:50                          False
4           2014-01-01 02:56:50                          False
5           2014-01-01 03:21:50                          False

                                 WEEKDAY(signup_date) = 4  \
customer_id time
1           2014-01-01 01:41:50                      True
2           2014-01-01 02:06:50                      True
3           2014-01-01 02:31:50                      True
4           2014-01-01 02:56:50                      True
5           2014-01-01 03:21:50                      True

                                 WEEKDAY(signup_date) is unknown  \
customer_id time
1           2014-01-01 01:41:50                            False
2           2014-01-01 02:06:50                            False
3           2014-01-01 02:31:50                            False
4           2014-01-01 02:56:50                            False
5           2014-01-01 03:21:50                            False

                                 YEAR(signup_date) = 2010  \
customer_id time
1           2014-01-01 01:41:50                      True
2           2014-01-01 02:06:50                      True
3           2014-01-01 02:31:50                      True
4           2014-01-01 02:56:50                      True
5           2014-01-01 03:21:50                      True

                                 YEAR(signup_date) is unknown
customer_id time
1           2014-01-01 01:41:50                         False
2           2014-01-01 02:06:50                         False
3           2014-01-01 02:31:50                         False
4           2014-01-01 02:56:50                         False
5           2014-01-01 03:21:50                         False
```

[69]:
```
from sklearn.model_selection import train_test_split
```

(continues on next page)

(continued from previous page)

```
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.

[70]:
```
train_data = pd.DataFrame({"customer_id": [1, 2, 3, 4, 5],
                           "product_purchased": ["coke zero", "car", "toothpaste",
↪"coke zero", "car"]})

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
```

[70]:
```
            product_purchased
customer_id
1                   coke zero
2                         car
3                  toothpaste
4                   coke zero
5                         car
```

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

[71]:
```
feature_matrix_train_encoded, features_encoded = ft.encode_features(feature_matrix_
↪train,
                                                                    features)
feature_matrix_train_encoded.head()
```

[71]:
```
             product_purchased = coke zero  product_purchased = car  \
customer_id
1                                     True                    False
2                                    False                     True
3                                    False                    False
4                                     True                    False
5                                    False                     True

             product_purchased = toothpaste  product_purchased is unknown
customer_id
1                                     False                         False
```

(continues on next page)

(continued from previous page)

```
2                                    False                         False
3                                     True                         False
4                                    False                         False
5                                    False                         False
```

Now lets imagine we have some test data that has 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**).

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

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()
```

[72]:
```
             product_purchased = coke zero  product_purchased = car  \
customer_id
6                                     True                    False
7                                    False                     True
8                                     True                    False
9                                     True                    False
10                                   False                    False

             product_purchased = toothpaste  product_purchased is unknown
customer_id
6                                     False                         False
7                                     False                         False
8                                     False                         False
9                                     False                         False
10                                    False                          True
```

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.

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.

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

[73]:
```
   id  rating
0   1     3.5
1   2     4.0
2   3     4.5
3   4     1.5
4   4     5.0
```

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.

```
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 <module>
      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,␣
↪entity_id, dataframe, index, variable_types, make_index, time_index, secondary_time_
↪index, already_sorted)
    486             secondary_time_index=secondary_time_index,
    487             already_sorted=already_sorted,
--> 488             make_index=make_index)
    489         self.entity_dict[entity.id] = entity
    490         self.reset_data_description()

~/featuretools/featuretools/entityset/entity.py in __init__(self, id, df, entityset,␣
↪variable_types, index, time_index, secondary_time_index, last_time_index, already_
↪sorted, make_index, verbose)
     79
     80         self.df = df[[v.id for v in self.variables]]
---> 81         self.set_index(index)
     82
     83         self.time_index = None

~/featuretools/featuretools/entityset/entity.py in set_index(self, variable_id,␣
↪unique)
    450         self.df.index.name = None
    451         if unique:
--> 452             assert self.df.index.is_unique, "Index is not unique on dataframe␣
↪(Entity {})".format(self.id)
    453
    454         self.convert_variable_type(variable_id, vtypes.Index, convert_
↪data=False)

AssertionError: Index is not unique on dataframe (Entity products)
```

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

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

[74]:
```
product_df = pd.DataFrame({'id': [1, 2, 3, 4, 5],
```

(continues on next page)

(continued from previous page)

```
                                'rating': [3.5, 4.0, 4.5, 1.5, 5.0]})
product_df
```

[74]:

```
   id  rating
0   1     3.5
1   2     4.0
2   3     4.5
3   4     1.5
4   5     5.0
```

Notice how we now have a unique index column called `id`.

[75]:

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

[75]:

```
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
```

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.

[76]:

```
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
```

[76]:

```
   product_id  id  rating
0           0   1     3.5
1           1   2     4.0
2           2   3     4.5
3           3   4     1.5
4           4   4     5.0
```

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 `your_entityset.add_last_time_indexes()` after you create your `EntitySet`. This will remove the warning.

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

Now we can run DFS without getting the warning.

[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:

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

You should see a version greater than `0.4.1`

3.4.2 Help

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

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.

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.

3.4.3 Limitations

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.

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.

3.4.4 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.

3.4.5 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.

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.

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

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.

3.5 API Reference

3.5.1 Demo Datasets

`load_retail`([id, nrows, return_single_table])	Returns the retail entityset example.
`load_mock_customer`([n_customers, . . .])	Return dataframes of mock customer data
`load_flight`([month_filter, . . .])	Download, clean, and filter flight data from 2017.

featuretools.demo.load_retail

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

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

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%|xxxxxxxxxxxxxxxxxxxxxxxxx| 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
```

3.5.2 Deep Feature Synthesis

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

featuretools.dfs

`featuretools.dfs`(*entities=None*, *relationships=None*, *entityset=None*, *target_entity=None*, *cutoff_time=None*, *instance_ids=None*, *agg_primitives=None*, *trans_primitives=None*, *groupby_trans_primitives=None*, *allowed_paths=None*, *max_depth=2*, *ignore_entities=None*, *ignore_variables=None*, *primitive_options=None*, *seed_features=None*, *drop_contains=None*, *drop_exact=None*, *where_primitives=None*, *max_features=- 1*, *cutoff_time_in_index=False*, *save_progress=None*, *features_only=False*, *training_window=None*, *approximate=None*, *chunk_size=None*, *n_jobs=1*, *dask_kwargs=None*, *verbose=False*, *return_variable_types=None*, *progress_callback=None*, *include_cutoff_time=True*)

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, dict[str -> Variable])]*) – 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).
- **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 the features for each instance. The resulting feature matrix will use data up to and including the cutoff_time. Can either be a DataFrame or a single value. If a DataFrame is passed the instance ids for which to calculate features must be in a column with the same name as the target entity index or a column named *instance_id*. The cutoff time values in the DataFrame must be in a column with the same name as the target entity time index or a column named *time*. If the DataFrame has more than two columns, any additional columns will be added to the resulting feature matrix. If a single value is passed, this value will be used for all instances.
- **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.

  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”]

- **groupby_trans_primitives** (list[str or *primitives.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 feautres 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 entityt 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 blacklist 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:

  [“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
- **include_cutoff_time** (*bool*) – Include data at cutoff times in feature calculations. Defaults to `True`.

Returns The list of generated feature defintions, and the feature matrix. If `features_only` is `True`, the feature matrix will not be generated.

Return type list[`FeatureBase`], pd.DataFrame

Examples

```
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)
```

3.5.3 Wrappers

Scikit-learn (BETA)

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

featuretools.wrappers.DFSTransformer

class featuretools.wrappers.**DFSTransformer**(*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*)

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

```
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 examle data
In [6]: n_customers = 3
```

(continues on next page)

(continued from previous page)

```
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=2)),
   ...:     ('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(steps=[('ft',
                 <featuretools_sklearn_transformer.transformer.DFSTransformer␣
↪object at 0x7f4518ec1a58>),
                ('et', ExtraTreesClassifier())])

In [11]: pipeline.predict_proba([4,5]) # predict probability of each class on␣
↪last 2
Out[11]:
array([[0., 1.],
       [0., 1.]])

In [12]: pipeline.predict([4,5]) # predict on last 2
Out[12]: 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(steps=[('ft',
                 <featuretools_sklearn_transformer.transformer.DFSTransformer␣
↪object at 0x7f4518ec1a58>),
                ('et', ExtraTreesClassifier())])

In [17]: pipeline.predict_proba(ct.tail(2))
Out[17]:
array([[0.57, 0.43],
       [0.  , 1.  ]])

In [18]: pipeline.predict(ct.tail(2))
Out[18]: array([False,  True])
```

Methods

`__init__`([entities, relationships, ...])	Creates Transformer
`fit`(cuttof_time_ids[, y])	Wrapper for DFS
`fit_transform`(X[, y])	Fit to data, then transform it.
`get_params`([deep])	
`transform`(cuttof_time_ids)	Wrapper for calculate_feature_matix

3.5.4 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: >>> three_observations_log = Timedelta(3, "observations") >>> three_observations_log.get_name() '3 Observations'

__init__(*value*, *unit=None*, *delta_obj=None*)

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])	**param value** Value of timedelta, string providing
`check_value`(value, unit)	
`fix_units`()	
`from_dictionary`(dictionary)	
`get_arguments`()	
`get_name`()	
`get_unit_type`()	
`get_units`()	
`get_value`([unit])	
`has_multiple_units`()	
`has_no_observations`()	
`is_absolute`()	
`lower_readable_times`()	
`make_singular`(s)	

3.5.5 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.

featuretools.make_temporal_cutoffs

featuretools.**make_temporal_cutoffs**(*instance_ids*, *cutoffs*, *window_size=None*, *num_windows=None*, *start=None*)

Makes a set of equally spaced cutoff times prior to a set of input cutoffs and instance ids.

If window_size and num_windows are provided, then num_windows of size window_size will be created prior to each cutoff time

If window_size and a start list is provided, then a variable number of windows will be created prior to each cutoff time, with the corresponding start time as the first cutoff.

If num_windows and a start list is provided, then num_windows of variable size will be created prior to each cutoff time, with the corresponding start time as the first cutoff

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

3.5.6 Feature Primitives

A list of all Featuretools primitives can be obtained by visiting primitives.featurelabs.com.

Primitive Types

TransformPrimitive()	Feature for entity that is a based off one or more other features in that entity.
AggregationPrimitive()	

featuretools.primitives.TransformPrimitive

class featuretools.primitives.**TransformPrimitive**

Feature for entity that is a 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
compatibility
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.
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
compatibility
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

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, default_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

```
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*) – 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 primitive. If no name is provided, the name of *function* will be used.
- **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'.
- **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

```
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*.
`Mean`([skipna])	Computes the average for a list of values.
`Sum`()	Calculates the total addition, ignoring *NaN*.
`Min`()	Calculates the smallest value, ignoring *NaN* values.
`Max`()	Calculates the highest value, ignoring *NaN* values.
`Std`()	Computes the dispersion relative to the mean value, ignoring *NaN*.
`Median`()	Determines the middlemost number in a list of values.
`Mode`()	Determines the most commonly repeated value.
`AvgTimeBetween`([unit])	Computes the average number of seconds between consecutive events.
`TimeSinceLast`([unit])	Calculates the time elapsed since the last datetime (default in seconds).
`TimeSinceFirst`([unit])	Calculates the time elapsed since the first datetime (in seconds).
`NumUnique`()	Determines the number of distinct values, ignoring *NaN* values.
`PercentTrue`()	Determines the percent of *True* values.
`All`()	Calculates if all values are 'True' in a list.
`Any`()	Determines if any value is 'True' in a list.
`First`()	Determines the first value in a list.
`Last`()	Determines the last value in a list.

continues on next page

Table 14 – continued from previous page

Skew()	Computes the extent to which a distribution differs from a normal distribution.
Trend()	Calculates the trend of a variable over time.
Entropy([dropna, base])	Calculates the entropy for a categorical variable

featuretools.primitives.Count

class featuretools.primitives.**Count**

Determines the total number of values, excluding *NaN*.

Examples

```
>>> 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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

```
>>> mean = Mean()
>>> mean([1, 2, 3, 4, 5, None])
3.0
```

We can also control the way *NaN* values are handled.

```
>>> 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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

```
>>> 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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time

featuretools.primitives.Min

class featuretools.primitives.**Min**

Calculates the smallest value, ignoring *NaN* values.

Examples

```
>>> 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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

class featuretools.primitives.**Max**
Calculates the highest value, ignoring *NaN* values.

Examples

```
>>> max = Max()
>>> max([1, 2, 3, 4, 5, None])
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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

class featuretools.primitives.**Std**

Computes the dispersion relative to the mean value, ignoring *NaN*.

Examples

```
>>> std = Std()
>>> round(std([1, 2, 3, 4, 5, None]), 3)
1.414
```

__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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

class featuretools.primitives.**Median**
Determines the middlemost number in a list of values.

Examples

```
>>> median = Median()
>>> median([5, 3, 2, 1, 4])
3.0
```

NaN values are ignored.

```
>>> median([5, 3, 2, 1, 4, None])
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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

class featuretools.primitives.**Mode**

Determines the most commonly repeated value.

Description: Given a list of values, return the value with the highest number of occurences. If list is empty, return *NaN*.

Examples

```
>>> mode = Mode()
>>> mode(['red', 'blue', 'green', 'blue'])
'blue'
```

__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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

```
>>> 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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
stack_on

continues on next page

Table 32 – continued from previous page

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

```
>>> 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, 11, 57, 30)]
>>> time_since_last(times, time=cutoff_time)
150.0
```

```
>>> 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, 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])	Initialize self.
generate_name(base_feature_names, …)	
generate_names(base_feature_names, …)	
get_args_string()	
get_arguments()	
get_filepath(filename)	
get_function([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time

featuretools.primitives.TimeSinceFirst

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([agg_type])`	

Attributes

`base_of`
`base_of_exclude`
`commutative`
`compatibility`
`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

```
>>> 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`([agg_type])	

Attributes

`base_of`
`base_of_exclude`
`commutative`
`compatibility`
`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

```
>>> percent_true = PercentTrue()
>>> percent_true([True, False, True, True, None])
0.6
```

`__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([agg_type])`	

Attributes

`base_of`
`base_of_exclude`
`commutative`
`compatibility`
`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

```
>>> all = All()
>>> all([False, False, False, 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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time

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

```
>>> any = Any()
>>> any([False, False, False, True])
True
```

__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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

```
>>> 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`([agg_type])	

Attributes

`base_of`
`base_of_exclude`
`commutative`
`compatibility`
`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.Last

class featuretools.primitives.**Last**
Determines the last value in a list.

Examples

```
>>> 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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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.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

```
>>> skew = Skew()
>>> skew([1, 10, 30, None])
1.0437603722639681
```

__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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

```
>>> 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, 11, 12),
...          datetime(2010, 1, 1, 11, 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()	
get_arguments()	
get_filepath(filename)	
get_function([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
stack_on
stack_on_exclude
stack_on_self
uses_calc_time

featuretools.primitives.Entropy

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

```
>>> 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([agg_type])	

Attributes

base_of
base_of_exclude
commutative
compatibility
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

IsIn([list_of_outputs])	Determines whether a value is present in a provided list.
And()	Element-wise logical AND of two lists.
Or()	Element-wise logical OR of two lists.
Not()	Negates a boolean value.

featuretools.primitives.IsIn

class featuretools.primitives.**IsIn**(*list_of_outputs=None*)

Determines whether a value is present in a provided list.

Examples

```
>>> items = ['string', 10.3, False]
>>> is_in = IsIn(list_of_outputs=items)
>>> is_in(['string', 10.5, False]).tolist()
[True, False, True]
```

__init__(*list_of_outputs=None*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([list_of_outputs])	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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time

continues on next page

Table 57 – continued from previous page

uses_full_entity

featuretools.primitives.And

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

```
>>> _and = And()
>>> _and([False, True, False], [True, True, False]).tolist()
[False, 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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.Or

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

```
>>> _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
compatibility
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

```
>>> not_func = Not()
>>> not_func([True, True, False]).tolist()
[False, False, True]
```

__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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

General Transform Primitives

Absolute()	Computes the absolute value of a number.
Percentile()	Determines the percentile rank for each value in a list.
TimeSince([unit])	Calculates time from a value to a specified cutoff datetime.

featuretools.primitives.Absolute

class featuretools.primitives.**Absolute**

Computes the absolute value of a number.

Examples

```
>>> 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)	
generate_names(base_feature_names)	
get_args_string()	
get_arguments()	
get_filepath(filename)	
get_function()	

Attributes

base_of
base_of_exclude
commutative
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.Percentile

class featuretools.primitives.**Percentile**

Determines the percentile rank for each value in a list.

Examples

```
>>> percentile = Percentile()
>>> percentile([10, 15, 1, 20]).tolist()
[0.5, 0.75, 0.25, 1.0]
```

Nan values are ignored when determining rank

```
>>> 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

`__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`
`compatibility`
`default_value`
`input_types`
`max_stack_depth`
`name`
`number_output_features`
`uses_calc_time`
`uses_full_entity`

featuretools.primitives.TimeSince

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

```
>>> 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(times, time=cutoff_time)
>>> list(map(int, values))
[0, -1, -120]
```

Change output to nanoseconds

```
>>> 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(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
compatibility
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

```
>>> 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]
```

__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`
`compatibility`
`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

```
>>> 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.
`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`
`compatibility`
`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

```
>>> 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

`__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`
`compatibility`
`default_value`
`input_types`
`max_stack_depth`
`name`
`number_output_features`
`uses_calc_time`
`uses_full_entity`

featuretools.primitives.IsWeekend

class featuretools.primitives.**IsWeekend**

Determines if a date falls on a weekend.

Examples

```
>>> 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__`()	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`
`compatibility`
`default_value`
`input_types`
`max_stack_depth`
`name`
`number_output_features`
`uses_calc_time`
`uses_full_entity`

featuretools.primitives.Hour

class featuretools.primitives.**Hour**

Determines the hour value of a datetime.

Examples

```
>>> 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`
`compatibility`
`default_value`
`input_types`
`max_stack_depth`
`name`
`number_output_features`
`uses_calc_time`
`uses_full_entity`

featuretools.primitives.Day

class featuretools.primitives.**Day**

Determines the day of the month from a datetime.

Examples

```
>>> 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

`__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`
`compatibility`
`default_value`
`input_types`
`max_stack_depth`
`name`
`number_output_features`
`uses_calc_time`
`uses_full_entity`

featuretools.primitives.Week

class featuretools.primitives.**Week**

Determines 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

```
>>> 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`
`compatibility`
`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

```
>>> 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`
`compatibility`
`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

```
>>> 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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

Cumulative Transform Primitives

Diff()	Compute the difference between the value in a list and the previous value in that list.
TimeSincePrevious([unit])	Compute the time since the previous entry in a list.
CumCount()	Calculates the cumulative count.
CumSum()	Calculates the cumulative sum.
CumMean()	Calculates the cumulative mean.
CumMin()	Calculates the cumulative minimum.
CumMax()	Calculates the cumulative maximum.

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

```
>>> diff = Diff()
>>> values = [1, 10, 3, 4, 15]
>>> diff(values).tolist()
[nan, 9.0, -7.0, 1.0, 11.0]
```

__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`
`compatibility`
`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

```
>>> from datetime import datetime
>>> time_since_previous = TimeSincePrevious()
>>> dates = [datetime(2019, 3, 1, 0, 0, 0),
...          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`
`compatibility`
`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
compatibility
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 caluclation, they're ignored.

Examples

```
>>> 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()	
get_filepath(filename)	
get_function()	

Attributes

base_of
base_of_exclude
commutative
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

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 caluclation, they're treated as 0.

Examples

```
>>> 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.
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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

featuretools.primitives.CumMin

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 caluclation, they're ignored.

Examples

```
>>> 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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

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 caluclation, they're ignored.

Examples

```
>>> 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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

NaturalLanguage 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

```
>>> 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
compatibility
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

```
>>> 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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

Location Transform Primitives

Latitude()	Returns the first tuple value in a list of LatLong tuples.
Longitude()	Returns the second tuple value in a list of LatLong tuples.
Haversine([unit])	Calculates the approximate haversine distance between two LatLong variable types.

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

```
>>> 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

__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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time

continues on next page

Table 112 – continued from previous page

uses_full_entity

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

```
>>> 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
compatibility
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

__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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time

continues on next page

Table 116 – continued from previous page

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.

DiversityScore()	Calculates the overall complexity of the text based on the total
LSA()	Calculates the Latent Semantic Analysis Values of Text Input
MeanCharactersPerWord()	Determines the mean number of characters per word.
PartOfSpeechCount()	Calculates the occurences of each different part of speech.
PolarityScore()	Calculates the polarity of a text on a scale from -1 (negative) to 1 (positive)
PunctuationCount()	Determines number of punctuation characters in a string.
StopwordCount()	Determines number of stopwords in a string.
TitleWordCount()	Determines the number of title words in a string.
UniversalSentenceEncoder()	Transforms a sentence or short paragraph to a vector using [tfhub model](https://tfhub.dev/google/universal-sentence-encoder/2)
UpperCaseCount()	Calculates the number of upper case letters in text.

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

```
>>> 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

`__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`
`compatibility`
`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

```
>>> 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.

```
>>> 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
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

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

```
>>> 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
compatibility
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 occurences 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

```
>>> 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, 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, 0.0], [0.0, 0.0], [1.
↪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
compatibility
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 functions returns a score for every given piece of text. If a string is missing, return 'NaN'

Examples

```
>>> 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
compatibility
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

```
>>> 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__()	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
compatibility
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

```
>>> x = ['This is a test string.', 'This is second string', 'third string']
>>> stopword_count = StopwordCount()
>>> stopword_count(x).tolist()
[3, 2, 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
compatibility
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`
`compatibility`
`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

```
>>> 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)	
`get_function`()	

Attributes

`base_of`
`base_of_exclude`
`commutative`
`compatibility`
`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

```
>>> 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()	

Attributes

base_of
base_of_exclude
commutative
compatibility
default_value
input_types
max_stack_depth
name
number_output_features
uses_calc_time
uses_full_entity

Feature methods

`FeatureBase.rename`(name)	Rename Feature, returns copy
`FeatureBase.get_depth`([stop_at])	Returns depth of feature

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

3.5.7 Feature calculation

`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*, *include_cutoff_time=True*)
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 times at which to 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 or a single value. If a DataFrame is passed the instance ids for which to calculate features must be in a column with the same name as the target entity index or a column named *instance_id*. The cutoff time values in the DataFrame must be in a column with the same name as the target entity time index or a column named *time*. If the DataFrame has more than two columns, any additional columns will be added to the resulting feature matrix. If a single value is passed, this value will be used for all instances.
- **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, dict[str -> Variable])]*) – 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).
- **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
- **include_cutoff_time** (*bool*) – Include data at cutoff times in feature calculations. Defaults to `True`.

Returns The feature matrix.

Return type pd.DataFrame

3.5.8 Feature visualization

graph_feature(feature[, to_file])	Generates a feature lineage graph for the given feature

featuretools.graph_feature

featuretools.**graph_feature**(*feature*, *to_file=None*)

Generates a feature lineage graph for the given feature

Parameters

- **feature** (*FeatureBase*) – Feature to generate lineage graph for
- **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

3.5.9 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

- **feature_matrix** (*pd.DataFrame*) – Dataframe of features.
- **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

```
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,
   ...:                                              instance_ids=ids)
   ...:

In [7]: fm_encoded, f_encoded = ft.encode_features(feature_matrix,
   ...:                                            features)
   ...:

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'])
   ....:
```

(continues on next page)

(continued from previous page)

```
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>]
```

3.5.10 Saving and Loading Features

save_features(features[, location, profile_name])	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.
load_features(features[, profile_name])	Loads the features from a filepath, S3 path, URL, an open file, or a JSON formatted string.

featuretools.save_features

featuretools.**save_features** (*features*, *location=None*, *profile_name=None*)

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

```
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()

featuretools.load_features

featuretools.**load_features**(*features*, *profile_name=None*)

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** (*been*) –
- **a JSON formatted** (*or*) –
- **string** –
- **a readable file handle where the features have been saved.** (*or*) –
- **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

```
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()

3.5.11 EntitySet, Entity, Relationship, Variable Types

Constructors

EntitySet([id, entities, relationships])	Stores all actual data for a entityset
Entity(id, df, entityset[, variable_types, ...])	Represents an entity in a Entityset, and stores relevant metadata and data
Relationship(parent_variable, child_variable)	Class to represent an relationship between entities

featuretools.EntitySet

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, dict[str -> Variable])]*) – dictionary of entities. Entries take the format {entity id -> (dataframe, id column, (time_index), (variable_types), (make_index))}. Note that time_index, variable_types and make_index 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

```
entities = {
    "cards" : (card_df, "id"),
    "transactions" : (transactions_df, "id", "transaction_time")
}

relationships = [("cards", "id", "transactions", "card_id")]

ft.EntitySet("my-entity-set", entities, relationships)
```

Methods

`__init__`([id, entities, relationships])	Creates EntitySet
`add_interesting_values`([max_values, verbose])	Find interesting values for categorical variables, to be used to generate "where" clauses
`add_last_time_indexes`([updated_entities])	Calculates the last time index values for each entity (the last time an instance or children of that instance were observed). Used when calculating features using training windows :param updated_entities: List of entity ids to update last_time_index for (will update all parents of those entities as well) :type updated_entities: list[str].
`add_relationship`(relationship)	Add a new relationship between entities in the entityset
`add_relationships`(relationships)	Add multiple new relationships to a entityset
`concat`(other[, inplace])	Combine entityset with another to create a new entityset with the combined data of both entitysets.
`entity_from_dataframe`(entity_id, dataframe)	Load the data for a specified entity from a Pandas DataFrame.
`find_backward_paths`(start_entity_id, . . .)	Generator which yields all backward paths between a start and goal entity.
`find_forward_paths`(start_entity_id, . . .)	Generator which yields all forward paths between a start and goal entity.
`get_backward_entities`(entity_id[, deep])	Get entities that are in a backward relationship with entity
`get_backward_relationships`(entity_id)	get relationships where entity "entity_id" is the parent.
`get_forward_entities`(entity_id[, deep])	Get entities that are in a forward relationship with entity
`get_forward_relationships`(entity_id)	Get relationships where entity "entity_id" is the child
`has_unique_forward_path`(start_entity_id, . . .)	Is the forward path from start to end unique?
`normalize_entity`(base_entity_id, . . . [, . . .])	Create a new entity and relationship from unique values of an existing variable.
`plot`([to_file])	Create a UML diagram-ish graph of the EntitySet.
`reset_data_description`()	
`to_csv`(path[, sep, encoding, engine, . . .])	Write entityset to disk in the csv format, location specified by *path*.
`to_dictionary`()	

continues on next page

Table 145 – continued from previous page

`to_parquet`(path[, engine, compression, . . .])	Write entityset to disk in the parquet format, location specified by *path*.
`to_pickle`(path[, compression, profile_name])	Write entityset in the pickle format, location specified by *path*.

Attributes

`entities`	
`metadata`	Returns the metadata for this EntitySet.

featuretools.Entity

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 a 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 -> type/str/dict[str -> type]]*) – An entity's variable_types dict maps string variable ids to types (`Variable`) or type_string (str) 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

`__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.
`set_secondary_time_index`(secondary_time_index)	
`set_time_index`(variable_id[, already_sorted])	
`update_data`(df[, already_sorted, ...])	Update entity's internal dataframe, optionaly making sure data is sorted, reference indexes to other entities are consistent, and last_time_indexes are consistent.

Attributes

`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

__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/str], optional`*) – Keys are of variable ids and values are variable types or type_strings. 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"]
Out[6]:
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
Out[7]:
```

(continues on next page)

(continued from previous page)

```
   id  session_id  amount    transaction_time  fraud
1   1           1  100.40 2020-10-19 10:00:00   True
2   2           2   20.63 2020-10-19 10:00:10  False
3   3           1   33.32 2020-10-19 10:00:20   True
4   4           3   13.12 2020-10-19 10:00:30  False
5   5           4   67.22 2020-10-19 10:00:40   True
6   6           5    1.00 2020-10-19 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(path[, sep, encoding, . . .])	Write entityset to disk in the csv format, location specified by *path*.
EntitySet.to_pickle(path[, compression, . . .])	Write entityset in the pickle format, location specified by *path*.
EntitySet.to_parquet(path[, engine, . . .])	Write entityset to disk in the parquet format, location specified by *path*.

featuretools.entityset.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

EntitySet.__getitem__(entity_id)	Get entity instance from entityset
EntitySet.find_backward_paths(…)	Generator which yields all backward paths between a start and goal entity.
EntitySet.find_forward_paths(…)	Generator which yields all forward paths between a start and goal entity.
EntitySet.get_forward_entities(entity_id[, deep])	Get entities that are in a forward relationship with entity
EntitySet.get_backward_entities(entity_id[, …])	Get entities that are in a backward relationship with entity

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 *Entity*

featuretools.entityset.EntitySet.find_backward_paths

`EntitySet.find_backward_paths(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

- **start_entity_id** (*str*) – Id of entity to start the search from.
- **goal_entity_id** (*str*) – Id of entity to find backward path to.

See also:

`BaseEntitySet.find_forward_paths()`

featuretools.entityset.EntitySet.find_forward_paths

`EntitySet.find_forward_paths(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

- **start_entity_id** (*str*) – id of entity to start the search from
- **goal_entity_id** (*str*) – if of entity to find forward path to

See also:

`BaseEntitySet.find_backward_paths()`

featuretools.entityset.EntitySet.get_forward_entities

`EntitySet.get_forward_entities(entity_id, deep=False)`

Get entities that are in a forward relationship with entity

Parameters

- **entity_id** (*str*) – Id entity of entity to search from.
- **deep** (*bool*) – if True, recursively find forward entities.

Yields a tuple of (descendent_id, path from entity_id to descendant).

featuretools.entityset.EntitySet.get_backward_entities

`EntitySet.get_backward_entities`(*entity_id*, *deep=False*)
Get entities that are in a backward relationship with entity

Parameters

- **entity_id** (*str*) – Id entity of entity to search from.
- **deep** (*bool*) – if True, recursively find backward entities.

Yields a tuple of (descendent_id, path from entity_id to descendant).

EntitySet visualization

EntitySet.plot([to_file])	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

```
>>> 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*)

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

Relationship.parent_variable	Instance of variable in parent entity
Relationship.child_variable	Instance of variable in child entity
Relationship.parent_entity	Parent entity object
Relationship.child_entity	Child entity object

featuretools.entityset.Relationship.parent_variable

property `Relationship.`**`parent_variable`**

Instance of variable in parent entity

featuretools.entityset.Relationship.child_variable

property `Relationship.`**`child_variable`**

Instance of variable in child entity

featuretools.entityset.Relationship.parent_entity

property `Relationship.`**`parent_entity`**

Parent entity object

featuretools.entityset.Relationship.child_entity

property Relationship.**child_entity**
Child entity object

Variable types

Index(id, entity[, name])	Represents variables that uniquely identify an instance of an entity
Id(id, entity[, name, categories])	Represents variables that identify another entity
TimeIndex(id, entity[, name])	Represents time index of entity
DatetimeTimeIndex(id, entity[, name, format])	Represents time index of entity that is a datetime
NumericTimeIndex(id, entity[, name, range, . . .])	Represents time index of entity that is numeric
Datetime(id, entity[, name, format])	Represents variables that are points in time
Numeric(id, entity[, name, range, . . .])	Represents variables that contain numeric values
Categorical(id, entity[, name, categories])	Represents variables that can take an unordered discrete values
Ordinal(id, entity[, name])	Represents variables that take on an ordered discrete value
Boolean(id, entity[, name, true_values, . . .])	Represents variables that take on one of two values
NaturalLanguage(id, entity[, name])	Represents variables that are arbitary strings
LatLong(id, entity[, name])	Represents an ordered pair (Latitude, Longitude) To make a latlong in a dataframe do data['latlong'] = data[['latitude', 'longitude']].apply(tuple, axis=1)
ZIPCode(id, entity[, name, categories])	Represents a postal address in the United States.
IPAddress(id, entity[, name])	Represents a computer network address.
FullName(id, entity[, name])	Represents a person's full name.
EmailAddress(id, entity[, name])	Represents an email box to which email message are sent.
URL(id, entity[, name])	Represents a valid web url (with or without http/www)
PhoneNumber(id, entity[, name])	Represents any valid phone number.
DateOfBirth(id, entity[, name, format])	Represents a date of birth as a datetime
CountryCode(id, entity[, name, categories])	Represents an ISO-3166 standard country code.
SubRegionCode(id, entity[, name, categories])	Represents an ISO-3166 standard sub-region code.
FilePath(id, entity[, name])	Represents a valid filepath, absolute or relative

featuretools.variable_types.variable.Index

class featuretools.variable_types.variable.**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

featuretools.variable_types.variable.Id

class featuretools.variable_types.variable.**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.variable.TimeIndex

class featuretools.variable_types.variable.**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.variable.DatetimeTimeIndex

class featuretools.variable_types.variable.**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`
`entityset`
`interesting_values`
`name`
`series`
`type_string`

featuretools.variable_types.variable.NumericTimeIndex

class `featuretools.variable_types.variable.`**`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`
`entityset`
`interesting_values`
`name`
`series`
`type_string`

featuretools.variable_types.variable.Datetime

class `featuretools.variable_types.variable.`**`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.variable.Numeric

class `featuretools.variable_types.variable.`**`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.variable.Categorical

class featuretools.variable_types.variable.**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.variable.Ordinal

class featuretools.variable_types.variable.**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.variable.Boolean

class featuretools.variable_types.variable.**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`
`entityset`
`interesting_values`
`name`
`series`
`type_string`

featuretools.variable_types.variable.NaturalLanguage

class `featuretools.variable_types.variable.`**`NaturalLanguage`**(*id*, *entity*, *name=None*)

Represents variables that are arbitary strings

`__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.variable.LatLong

class featuretools.variable_types.variable.**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.

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.variable.ZIPCode

class featuretools.variable_types.variable.**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`()	

Attributes

`dtype`
`entityset`
`interesting_values`
`name`
`series`
`type_string`

featuretools.variable_types.variable.IPAddress

class `featuretools.variable_types.variable.`**`IPAddress`**(*id*, *entity*, *name=None*)
Represents a computer network address. Represented in dotted-decimal notation. IPv4 and IPv6 are supported.

`__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.variable.FullName

class `featuretools.variable_types.variable.`**`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`
`entityset`
`interesting_values`
`name`
`series`
`type_string`

featuretools.variable_types.variable.EmailAddress

class featuretools.variable_types.variable.**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

`dtype`
`entityset`
`interesting_values`
`name`
`series`
`type_string`

featuretools.variable_types.variable.URL

class featuretools.variable_types.variable.**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])	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.variable.PhoneNumber

class featuretools.variable_types.variable.**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])	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.variable.DateOfBirth

class featuretools.variable_types.variable.**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
entityset
interesting_values
name
series
type_string

featuretools.variable_types.variable.CountryCode

class featuretools.variable_types.variable.**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

`dtype`
`entityset`
`interesting_values`
`name`
`series`
`type_string`

featuretools.variable_types.variable.SubRegionCode

class `featuretools.variable_types.variable.`**`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`
`entityset`
`interesting_values`
`name`
`series`
`type_string`

featuretools.variable_types.variable.FilePath

class featuretools.variable_types.variable.**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

dtype
entityset
interesting_values
name
series
type_string

Variable Utils Methods

find_variable_types()	Retrieves all Variable types as a dictionary where key is type_string
list_variable_types()	Retrieves all Variable types as a dataframe, with the column headers
graph_variable_types([to_file])	Create a UML diagram-ish graph of all the Variables.

featuretools.variable_types.utils.find_variable_types

featuretools.variable_types.utils.**find_variable_types**()

Retrieves all Variable types as a dictionary where key is type_string of Variable, and value is a Variable object.

Parameters **None** –

Returns

Return type variable_types (dict)

featuretools.variable_types.utils.list_variable_types

`featuretools.variable_types.utils.list_variable_types()`

Retrieves all Variable types as a dataframe, with the column headers of name, type_string, and description.

Parameters **None** –

Returns

a DataFrame with column headers of name, type_strings, and description.

Return type variable_types (pd.DataFrame)

featuretools.variable_types.utils.graph_variable_types

`featuretools.variable_types.utils.graph_variable_types(to_file=None)`

Create a UML diagram-ish graph of all the Variables.

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

Feature Selection

remove_low_information_features(feature_matrix)	Select features that have at least 2 unique values and that are not all null
remove_highly_correlated_features(feature_matrix)	Removes columns in feature matrix that are highly correlated with another column.
remove_highly_null_features(feature_matrix)	Removes columns from a feature matrix that have higher than a set threshold of null values.
remove_single_value_features(feature_matrix)	Removes columns in feature matrix where all the values are the same.

featuretools.selection.remove_low_information_features

`featuretools.selection.remove_low_information_features(feature_matrix, features=None)`

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)

featuretools.selection.remove_highly_correlated_features

`featuretools.selection.`**`remove_highly_correlated_features`**(*feature_matrix*, *features=None*, *pct_corr_threshold=0.95*, *features_to_check=None*, *features_to_keep=None*)

Removes columns in feature matrix that are highly correlated with another column.

Note: We make the assumption that, for a pair of features, the feature that is further right in the feature matrix produced by `dfs` is the more complex one. The assumption does not hold if the order of columns in the feature matrix has changed from what `dfs` produces.

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.
- **pct_corr_threshold** (*float*) – The correlation threshold to be considered highly correlated. Defaults to 0.95.
- **features_to_check** (*list[str], optional*) – List of column names to check whether any pairs are highly correlated. Will not check any other columns, meaning the only columns that can be removed are in this list. If null, defaults to checking all columns.
- **features_to_keep** (*list[str], optional*) – List of colum names to keep even if correlated to another column. If null, all columns will be candidates for removal.

Returns The feature matrix and the list of generated feature definitions. Matches dfs output. If no feature list is provided as input, the feature list will not be returned. For consistent results, do not change the order of features outputted by dfs.

Return type pd.DataFrame, list[`FeatureBase`]

featuretools.selection.remove_highly_null_features

`featuretools.selection.`**`remove_highly_null_features`**(*feature_matrix*, *features=None*, *pct_null_threshold=0.95*)

Removes columns from a feature matrix that have higher than a set threshold of null values.

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.
- **pct_null_threshold** (*float*) – If the percentage of NaN values in an input feature exceeds this amount, that feature will be considered highly-null. Defaults to 0.95.

Returns The feature matrix and the list of generated feature definitions. Matches dfs output. If no feature list is provided as input, the feature list will not be returned.

Return type pd.DataFrame, list[`FeatureBase`]

featuretools.selection.remove_single_value_features

`featuretools.selection.remove_single_value_features`(*feature_matrix*, *features=None*, *count_nan_as_value=False*)

Removes columns in feature matrix where all the values are the same.

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.
- **count_nan_as_value** – If True, missing values will be counted as their own unique value. If set to False, a feature that has one unique value and all other data missing will be removed from the feature matrix. Defaults to False.

3.6 Release Notes

Future Release

- Enhancements
- Fixes
- **Changes**
  - Restrict koalas version to below 1.3.0 (GH#1192)
  - Keep koalas requirements in separate file (GH#1195)
- **Documentation Changes**
  - Added footer to the documentation (GH#1189)
- **Testing Changes**
  - Add `pyspark` and `koalas` to automated dependency checks (GH#1191)

Thanks to the following people for contributing to this release: @rwedge, @thehomebrewnerd, @jeff-hernandez

v0.20.0 Sep 30, 2020

> **Warning:** The Text variable type has been deprecated and been replaced with the NaturalLanguage variable type. The Text variable type will be removed in a future release.

- **Fixes**
  - Allow FeatureOutputSlice features to be serialized (GH#1150)
  - Fix duplicate label column generation when labels are passed in cutoff times and approximate is being used (GH#1160)
  - Determine calculate_feature_matrix behavior with approximate and a cutoff df that is a subclass of a pandas DataFrame (GH#1166)
- **Changes**
  - Text variable type has been replaced with NaturalLanguage (GH#1159)
- **Documentation Changes**

    - Update release doc for clarity and to add Future Release template (GH#1151)
    - Use the PyData Sphinx theme (GH#1169)
- **Testing Changes**
    - Stop requiring single-threaded dask scheduler in tests (GH#1163, GH#1170)

Thanks to the following people for contributing to this release: @gsheni, @rwedge, @tamargrey, @tuethan1999

v0.19.0 Sept 8, 2020

- **Enhancements**
    - Support use of Koalas DataFrames in entitysets (GH#1031)
    - Add feature selection functions for null, correlated, and single value features (GH#1126)
- **Fixes**
    - Fix `encode_features` converting excluded feature columns to a numeric dtype (GH#1123)
    - Improve performance of unused primitive check in dfs (GH#1140)
- **Changes**
    - Remove the ability to stack transform primitives (GH#1119, GH#1145)
    - Sort primitives passed to `dfs` to get consistent ordering of features* (GH#1119)
- **Documentation Changes**
    - Added return values to dfs and calculate_feature_matrix (GH#1125)
- **Testing Changes**
    - Better test case for normalizing from no time index to time index (GH#1113)

* When passing multiple instances of a primitive built with `make_trans_primitive` or `maxe_agg_primitive`, those instances must have the same relative order when passed to `dfs` to ensure a consistent ordering of features.

Thanks to the following people for contributing to this release: @frances-h, @gsheni, @rwedge, @tamargrey, @thehomebrewnerd, @tuethan1999

Breaking Changes

- `ft.dfs` will no longer build features from Transform primitives where one of the inputs is a Transform feature, a GroupByTransform feature, or a Direct Feature of a Transform / GroupByTransform feature. This will make some features that would previously be generated by `ft.dfs` only possible if explicitly specified in `seed_features`.

v0.18.1 Aug 12, 2020

- **Fixes**
    - Fix `EntitySet.plot()` when given a dask entityset (GH#1086)
- **Changes**
    - Use `nlp-primitives[complete]` install for `nlp_primitives` extra in `setup.py` (GH#1103)
- **Documentation Changes**
    - Fix broken downloads badge in README.md (GH#1107)
- **Testing Changes**

  - Use CircleCI matrix jobs in config to trigger multiple runs of same job with different parameters (GH#1105)

Thanks to the following people for contributing to this release: @gsheni, @systemshift, @thehomebrewnerd

v0.18.0 July 31, 2020

- **Enhancements**
  - Warn user if supplied primitives are not used during dfs (GH#1073)
- **Fixes**
  - Use more consistent and uniform warnings (GH#1040)
  - Fix issue with missing instance ids and categorical entity index (GH#1050)
  - Remove warnings.simplefilter in feature_set_calculator to un-silence warnings (GH#1053)
  - Fix feature visualization for features with '>' or '<' in name (GH#1055)
  - Fix boolean dtype mismatch between encode_features and dfs and calculate_feature_matrix (GH#1082)
  - Update primitive options to check reversed inputs if primitive is commutative (GH#1085)
  - Fix inconsistent ordering of features between kernel restarts (GH#1088)
- **Changes**
  - Make DFS match `TimeSince` primitive with all `Datetime` types (GH#1048)
  - Change default branch to `main` (GH#1038)
  - Raise TypeError if improper input is supplied to `Entity.delete_variables()` (GH#1064)
  - Updates for compatibility with pandas 1.1.0 (GH#1079, GH#1089)
  - Set pandas version to pandas>=0.24.1,<2.0.0. Filter pandas deprecation warning in Week primitive. (GH#1094)
- **Documentation Changes**
  - Remove benchmarks folder (GH#1049)
  - Add custom variables types section to variables page (GH#1066)
- **Testing Changes**
  - Add fixture for `ft.demo.load_mock_customer` (GH#1036)
  - Refactor Dask test units (GH#1052)
  - Implement automated process for checking critical dependencies (GH#1045, GH#1054, GH#1081)
  - Don't run changelog check for release PRs or automated dependency PRs (GH#1057)
  - Fix non-deterministic behavior in Dask test causing codecov issues (GH#1070)

Thanks to the following people for contributing to this release: @frances-h, @gsheni, @monti-python, @rwedge, @systemshift, @tamargrey, @thehomebrewnerd, @wsankey

v0.17.0 June 30, 2020

- **Enhancements**

- Add `list_variable_types` and `graph_variable_types` for Variable Types (GH#1013)
- Add `graph_feature` to generate a feature lineage graph for a given feature (GH#1032)

- **Fixes**
  - Improve warnings when using a Dask dataframe for cutoff times (GH#1026)
  - Error if attempting to add entityset relationship where child variable is also child index (GH#1034)
- **Changes**
  - Remove `Feature.get_names` (GH#1021)
  - Remove unnecessary `pd.Series` and `pd.DatetimeIndex` calls from primitives (GH#1020, GH#1024)
  - Improve cutoff time handling when a single value or no value is passed (GH#1028)
  - Moved `find_variable_types` to Variable utils (GH#1013)
- **Documentation Changes**
  - Add page on Variable Types to describe some Variable Types, and util functions (GH#1013)
  - Remove featuretools enterprise from documentation (GH#1022)
  - Add development install instructions to contributing.md (GH#1030)
- **Testing Changes**
  - Add `required` flag to CircleCI codecov upload command (GH#1035)

Thanks to the following people for contributing to this release: @frances-h, @gsheni, @kmax12, @rwedge, @thehomebrewnerd, @tuethan1999

Breaking Changes

- Removed `Feature.get_names`, `Feature.get_feature_names` should be used instead

v0.16.0 June 5, 2020

- **Enhancements**
  - Support use of Dask DataFrames in entitysets (GH#783)
  - Add `make_index` when initializing an EntitySet by passing in an `entities` dictionary (GH#1010)
  - Add ability to use primitive classes and instances as keys in primitive_options dictionary (GH#993)
- **Fixes**
  - Cleanly close tqdm instance (GH#1018)
  - Resolve issue with `NaN` values in `LatLong` columns (GH#1007)
- **Testing Changes**
  - Update tests for numpy v1.19.0 compatability (GH#1016)

Thanks to the following people for contributing to this release: @Alex-Monahan, @frances-h, @gsheni, @rwedge, @thehomebrewnerd

v0.15.0 May 29, 2020

- **Enhancements**
  - Add get_default_aggregation_primitives and get_default_transform_primitives (GH#945)
  - Allow cutoff time dataframe columns to be in any order (GH#969, GH#995)
  - Add Age primitive, and make it a default transform primitive for DFS (GH#987)
  - Add include_cutoff_time arg - control whether data at cutoff times are included in feature calculations (GH#959)
  - Allow variables_types to be referenced by their type_string for the entity_from_dataframe function (GH#988)
- **Fixes**
  - Fix errors with Equals and NotEquals primitives when comparing categoricals or different dtypes (GH#968)
  - Normalized type_strings of Variable classes so that the find_variable_types function produces a dictionary with a clear key to name transition (GH#982, GH#996)
  - Remove pandas.datetime in test_calculate_feature_matrix due to deprecation (GH#998)
- **Documentation Changes**
  - Add python 3.8 support for docs (GH#983)
  - Adds consistent Entityset Docstrings (GH#986)
- **Testing Changes**
  - Add automated tests for python 3.8 environment (GH#847)
  - Update testing dependencies (GH#976)

Thanks to the following people for contributing to this release: @ctduffy, @frances-h, @gsheni, @jeff-hernandez, @rightx2, @rwedge, @sebrahimi1988, @thehomebrewnerd, @tuethan1999

Breaking Changes

- Calls to featuretools.dfs or featuretools.calculate_feature_matrix that use a cutoff time dataframe, but do not label the time column with either the target entity time index variable name or as time, will now result in an AttributeError. Previously, the time column was selected to be the first column that was not the instance id column. With this update, the position of the column in the dataframe is no longer used to determine the time column. Now, both instance id columns and time columns in a cutoff time dataframe can be in any order as long as they are named properly.
- The type_string attributes of all Variable subclasses are now a snake case conversion of their class names. This changes the type_string of the Unknown, IPAddress, EmailAddress, SubRegionCode, FilePath, LatLong, and ZIPcode classes. Old saved entitysets that used these variables may load incorrectly.

v0.14.0 Apr 30, 2020

- **Enhancements**
  - ft.encode_features - use less memory for one-hot encoded columns (GH#876)
- **Fixes**
  - Use logger.warning to fix deprecated logger.warn (GH#871)
  - Add dtype to interesting_values to fix deprecated empty Series with no dtype (GH#933)
  - Remove overlap in training windows (GH#930)

- Fix progress bar in notebook (GH#932)
- **Changes**
  - Change premium primitives CI test to Python 3.6 (GH#916)
  - Remove Python 3.5 support (GH#917)
- **Documentation Changes**
  - Fix README links to docs (GH#872)
  - Fix Github links with correct organizations (GH#908)
  - Fix hyperlinks in docs and docstrings with updated address (GH#910)
  - Remove unused script for uploading docs to AWS (GH#911)

Thanks to the following people for contributing to this release: @frances-h, @gsheni, @jeff-hernandez, @rwedge

Breaking Changes

- Using training windows in feature calculations can result in different values than previous versions. This was done to prevent consecutive training windows from overlapping by excluding data at the oldest point in time. For example, if we use a cutoff time at the first minute of the hour with a one hour training window, the first minute of the previous hour will no longer be included in the feature calculation.

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, @thehomebrewnerd

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, @rwedge, @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-hernandez, @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, @kkleidal, @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 unecessary 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

  ```
  from featuretools.primitives import NUnique
  ```

  New behavior

  ```
  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, @minkvsky, @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, @jrkinley, @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

```
ft.dfs(entityset=es,
       target_entity='customers',
       trans_primitives=["cum_mean"])
```

  New behavior

```
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

```
ft.Feature([base_feature, groupby_feature],␣
↪primitive=CumulativePrimitive)
```

  New behavior

```
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 fullargsspec (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 mising 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

CHAPTER FOUR

OTHER LINKS

- genindex
- search

INDEX

Symbols

A

M

N

O

P

R

S

T

U

V

W

Y

Z