automated-feature-engineering-tutorial

September 22, 2018

1 Automated Feature Engineering - Exercise (for me)

source:https://www.kaggle.com/willkoehrsen/automated-feature-engineering-tutorial

```
In [2]: import pandas as pd
        import numpy as np

        # featuretools for automated feature engineering
        import featuretools as ft

        # ignore warnings from pandas
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [3]: # Read in the data
        clients = pd.read_csv('./data/clients.csv', parse_dates = ['joined'])
        loans = pd.read_csv('./data/loans.csv', parse_dates = ['loan_start', 'loan_end'])
        payments = pd.read_csv('./data/payments.csv', parse_dates = ['payment_date'])
```

1.1 Manual Feature Engineering

1.1.1 transformation feature primitives

because they act on column in a single table

```
In [9]: # Create a month column
        clients['join_month'] = clients['joined'].dt.month

        # Create a log of income column
        clients['log_income'] = np.log(clients['income'])

        clients.head()
```

Out[9]:

	client_id	joined	income	credit_score	join_month	log_income
0	46109	2002-04-16	172677	527	4	12.059178
1	49545	2007-11-14	104564	770	11	11.557555
2	41480	2013-03-11	122607	585	3	11.716739
3	46180	2001-11-06	43851	562	11	10.688553
4	25707	2006-10-06	211422	621	10	12.261611

1.1.2 aggregation feature primitive

because we using multiple tables in a one-to-many relationship to calculate aggregation figures

```
In [13]: print(loans.shape)
         loans.head()
```

```
(443, 8)
```

```
Out[13]:    client_id loan_type  loan_amount  repaid  loan_id loan_start   loan_end  \
         0      46109      home        13672       0    10243 2002-04-16 2003-12-20
         1      46109    credit         9794       0    10984 2003-10-21 2005-07-17
         2      46109      home        12734       1    10990 2006-02-01 2007-07-05
         3      46109      cash        12518       1    10596 2010-12-08 2013-05-05
         4      46109    credit        14049       1    11415 2010-07-07 2012-05-21

            rate
         0  2.15
         1  1.25
         2  0.68
         3  1.24
         4  3.13
```

```
In [14]: # Groupby client id and calculate mean, max, min previous loan size
         stats = loans.groupby('client_id')['loan_amount'].agg(['mean', 'max', 'min'])
         stats.columns = ['mean_loan_amount', 'max_loan_amount', 'min_loan_amount']
         stats.head()
```

```
Out[14]:            mean_loan_amount  max_loan_amount  min_loan_amount
         client_id
         25707           7963.950000            13913             1212
         26326           7270.062500            13464             1164
         26695           7824.722222            14865             2389
         26945           7125.933333            14593              653
         29841           9813.000000            14837             2778
```

```
In [15]: # Merge with the clients dataframe
         clients.merge(stats, left_on = 'client_id', right_index=True, how = 'left').head(10)
```

```
Out[15]:    client_id     joined  income  credit_score  join_month  log_income  \
         0      46109 2002-04-16  172677           527           4   12.059178
         1      49545 2007-11-14  104564           770          11   11.557555
         2      41480 2013-03-11  122607           585           3   11.716739
         3      46180 2001-11-06   43851           562          11   10.688553
         4      25707 2006-10-06  211422           621          10   12.261611
         5      39505 2011-10-14  153873           610          10   11.943883
         6      32726 2006-05-01  235705           730           5   12.370336
         7      35089 2010-03-01  131176           771           3   11.784295
```

```
8      35214 2003-08-08   95849          696           8   11.470529
9      48177 2008-06-09  190632          769           6   12.158100

   mean_loan_amount  max_loan_amount  min_loan_amount
0       8951.600000            14049              559
1      10289.300000            14971             3851
2       7894.850000            14399              811
3       7700.850000            14081             1607
4       7963.950000            13913             1212
5       7424.050000            14575              904
6       6633.263158            14802              851
7       6939.200000            13194              773
8       7173.555556            14767              667
9       7424.368421            14740              659
```

We could go further and include information about payments in the clients dataframe. To do so, we would have to group payments by the loan_id, merge it with the loans, group the resulting dataframe by the client_id, and then merge it into the clients dataframe. This would allow us to include information about previous payments for each client.

Clearly, this process of manual feature engineering can grow quite tedious with many columns and multiple tables and I certainly don't want to have to do this process by hand! Luckily, feature tools can automatically perform this entire process and will create more features than we would have ever thought of. Although I love pandas, there is only so much manual data manipulation I'm willing to stand!

2 Feature Tools

The concept of Deep Feature Synthesis is to use basic building blocks known as feature primitives (like the transformations and aggregations done above) that can be stacked on top of each other to form new features. The depth of a "deep feature" is equal to the number of stacked primitives.

The first part of Feature Tools to understand is an **entity**. This is simply a table, or in pandas, a DataFrame. We corral multiple entities into a single object called an **EntitySet**. This is just a large data structure composed of many individual entities and the relationships between them

2.0.1 EntitySet

```
In [19]: es = ft.EntitySet(id = 'clients')
         es
```

```
Out[19]: Entityset: clients
           Entities:
           Relationships:
             No relationships
```

2.0.2 Entities

An entity is simply a table, which is represented in Pandas as a dataframe. Each entity must have a uniquely identifying column, known as an index. For the clients dataframe, this is the client_id because each id only appears once in the clients data. In the loans dataframe, client_id is not an index because each id might appear more than once. The index for this dataframe is instead loan_id.

When we create an entity in feature tools, we have to identify which column of the dataframe is the index. If the data does not have a unique index we can tell feature tools to make an index for the entity by passing in make_index = True and specifying a name for the index. If the data also has a uniquely identifying time index, we can pass that in as the time_index parameter.

Feature tools will automatically infer the variable types (numeric, categorical, datetime) of the columns in our data, but we can also pass in specific datatypes to override this behavior. As an example, even though the repaid column in the loans dataframe is represented as an integer, we can tell feature tools that this is a categorical feature since it can only take on two discrete values. This is done using an integer with the variables as keys and the feature types as values.

In the code below we create the three entities and add them to the EntitySet. The syntax is relatively straightforward with a few notes: for the payments dataframe we need to make an index, for the loans dataframe, we specify that repaid is a categorical variable, and for the payments dataframe, we specify that missed is a categorical feature.

```
In [21]: clients.head()
```

```
Out[21]:    client_id     joined  income  credit_score  join_month  log_income
         0      46109 2002-04-16  172677           527           4   12.059178
         1      49545 2007-11-14  104564           770          11   11.557555
         2      41480 2013-03-11  122607           585           3   11.716739
         3      46180 2001-11-06   43851           562          11   10.688553
         4      25707 2006-10-06  211422           621          10   12.261611
```

```
In [22]: # Create an entity from the client dataframe
         # This dataframe already has an index and a time index
         es = es.entity_from_dataframe(entity_id = 'clients', dataframe = clients,
                                       index = 'client_id', time_index = 'joined')
         es
```

```
Out[22]: Entityset: clients
           Entities:
             clients [Rows: 25, Columns: 6]
           Relationships:
             No relationships
```

```
In [24]: loans.head()
```

```
Out[24]:    client_id loan_type  loan_amount  repaid  loan_id loan_start   loan_end  \
         0      46109      home        13672       0    10243 2002-04-16 2003-12-20
         1      46109    credit         9794       0    10984 2003-10-21 2005-07-17
         2      46109      home        12734       1    10990 2006-02-01 2007-07-05
         3      46109      cash        12518       1    10596 2010-12-08 2013-05-05
         4      46109    credit        14049       1    11415 2010-07-07 2012-05-21

            rate
         0  2.15
```

```
1  1.25
2  0.68
3  1.24
4  3.13
```

In [25]:
```
# Create an entity from the loans dataframe
# This dataframe already has an index and a time index
es = es.entity_from_dataframe(entity_id = 'loans', dataframe = loans,
                              variable_types = {'repaid': ft.variable_types.Categorica
                              index = 'loan_id',
                              time_index = 'loan_start')
es
```

Out[25]:
```
Entityset: clients
  Entities:
    clients [Rows: 25, Columns: 6]
    loans [Rows: 443, Columns: 8]
  Relationships:
    No relationships
```

In [26]:
```
payments.head()
```

Out[26]:

	loan_id	payment_amount	payment_date	missed
0	10243	2369	2002-05-31	1
1	10243	2439	2002-06-18	1
2	10243	2662	2002-06-29	0
3	10243	2268	2002-07-20	0
4	10243	2027	2002-07-31	1

In [28]:
```
# Create an entity from the payments dataframe
# This does not yet have a unique index
es = es.entity_from_dataframe(entity_id = 'payments',
                              dataframe = payments,
                              variable_types = {'missed': ft.variable_types.Categorica
                              make_index = True,
                              index = 'payment_id',
                              time_index = 'payment_date')
es
```

Out[28]:
```
Entityset: clients
  Entities:
    clients [Rows: 25, Columns: 6]
    loans [Rows: 443, Columns: 8]
    payments [Rows: 3456, Columns: 5]
  Relationships:
    No relationships
```

In [29]:
```
es['loans']
```

Out[29]: Entity: loans
```
Entity: loans
  Variables:
    loan_id (dtype: index)
    client_id (dtype: numeric)
    loan_type (dtype: categorical)
    loan_amount (dtype: numeric)
    loan_start (dtype: datetime_time_index)
    loan_end (dtype: datetime)
    rate (dtype: numeric)
    repaid (dtype: categorical)
  Shape:
    (Rows: 443, Columns: 8)
```

2.1 Relationships

In [30]:
```
# Relationship between clients and previous loans
r_client_previous = ft.Relationship(es['clients']['client_id'],
                                    es['loans']['client_id'])

# Add the relationship to the entity set
es = es.add_relationship(r_client_previous)
```

this is a parent to child relationship because for each client_id in the parent client dataframe, there may be multiple entries of the same client_id in the child loans dataframe.

In [32]:
```
es
```

Out[32]:
```
Entityset: clients
  Entities:
    clients [Rows: 25, Columns: 6]
    loans [Rows: 443, Columns: 8]
    payments [Rows: 3456, Columns: 5]
  Relationships:
    loans.client_id -> clients.client_id
```

The second relationship is between the loans and payments. These two entities are related by the loan_id variable.

In [33]:
```
# Relationship between previous loans and previous payments
r_payments = ft.Relationship(es['loans']['loan_id'],
                             es['payments']['loan_id'])

# Add the relationship to the entity set
es = es.add_relationship(r_payments)

es
```

Out[33]:
```
Entityset: clients
  Entities:
```

```
  clients [Rows: 25, Columns: 6]
  loans [Rows: 443, Columns: 8]
  payments [Rows: 3456, Columns: 5]
Relationships:
  loans.client_id -> clients.client_id
  payments.loan_id -> loans.loan_id
```

---

3 Feature Primitives

A feature primitive a at a very high-level is an operation applied to data to create a feature. These represent very simple calculations that can be stacked on top of each other to create complex features. Feature primitives fall into two categories:

- **Aggregation**: function that groups together child datapoints for each parent and then calculates a statistic such as mean, min, max, or standard deviation. An example is calculating the maximum loan amount for each client. An aggregation works across multiple tables using relationships between tables.
- **Transformation**: an operation applied to one or more columns in a single table. An example would be extracting the day from dates, or finding the difference between two columns in one table.

Let's take a look at feature primitives in feature tools. We can view the list of primitives:

In [34]:
```
primitives = ft.list_primitives()
pd.options.display.max_colwidth = 100
primitives[primitives['type'] == 'aggregation'].head(10)
```

Out[34]:
```
             name         type  \
0           trend  aggregation
1             all  aggregation
2   n_most_common  aggregation
3             min  aggregation
4            mode  aggregation
5            last  aggregation
6           count  aggregation
7    percent_true  aggregation
8             sum  aggregation
9      num_unique  aggregation

                                                 description
0  Calculates the slope of the linear trend of variable overtime.
1                                 Test if all values are 'True'.
2      Finds the N most common elements in a categorical feature.
3         Finds the minimum non-null value of a numeric feature.
4           Finds the most common element in a categorical feature.
5                                        Returns the last value.
```

```
6                              Counts the number of non null values.
7          Finds the percent of 'True' values in a boolean feature.
8                    Sums elements of a numeric or boolean feature.
9               Returns the number of unique categorical variables.
```

In [36]:
```python
primitives[primitives['type'] == 'transform'].head(10)
```

Out[36]:
```
          name       type  \
19     cum_min  transform
20        year  transform
21    numwords  transform
22      divide  transform
23     cum_max  transform
24      months  transform
25     is_null  transform
26        diff  transform
27    multiply  transform
28  characters  transform


19  Calculates the min of previous values of an instance for each value in a time-depe
20                                                          Transform a Datetime feature i
21                                              Returns the words in a given string by countin
22                                                   Creates a transform feature that divides
23  Calculates the max of previous values of an instance for each value in a time-depe
24                                             Transform a Timedelta feature into the numb
25                                         For each value of base feature, return 'True' if v
26                    Compute the difference between the value of a base feature and the pr
27                                             Creates a transform feature that multplies
28                                                                Return the characters in a
```

In [42]:
```python
print(primitives.shape)
print()
print(primitives['type'].value_counts())
```

```
(62, 3)

transform      43
aggregation    19
Name: type, dtype: int64
```

Using primitives is surprisingly easy using the ft.dfs function (which stands for deep feature synthesis). In this function, we specify the entityset to use; the target_entity, which is the dataframe we want to make the features for (where the features end up); the agg_primitives which are the aggregation feature primitives; and the trans_primitives which are the transformation primitives to apply.

In the following example, we are using the EntitySet we already created, the target entity is the clients dataframe because we want to make new features about each client, and then we specify a few aggregation and transformation primitives.

```
In [43]: # Create new features using specified primitives
         features, feature_names = ft.dfs(entityset = es, target_entity = 'clients',
                                          agg_primitives = ['mean', 'max', 'percent_true', 'la
                                          trans_primitives = ['years', 'month', 'subtract', 'di
```

```
In [44]: pd.DataFrame(features['MONTH(joined)'].head())
```

```
Out[44]:            MONTH(joined)
         client_id
         25707                 10
         26326                  5
         26695                  8
         26945                 11
         29841                  8
```

```
In [49]: len(feature_names)
```

```
Out[49]: 797
```

```
In [57]: features.shape
```

```
Out[57]: (25, 94)
```

```
In [50]: features.head()
```

```
Out[50]:            income  credit_score  join_month  log_income  \
         client_id
         25707      211422           621          10   12.261611
         26326      227920           633           5   12.336750
         26695      174532           680           8   12.069863
         26945      214516           806          11   12.276140
         29841       38354           523           8   10.554614

                    MEAN(loans.loan_amount)  MEAN(loans.rate)  MAX(loans.loan_amount)  \
         client_id
         25707                  7963.950000          3.477000                   13913
         26326                  7270.062500          2.517500                   13464
         26695                  7824.722222          2.466111                   14865
         26945                  7125.933333          2.855333                   14593
         29841                  9813.000000          3.445000                   14837

                    MAX(loans.rate) LAST(loans.loan_type)  LAST(loans.loan_amount)  \
         client_id
         25707                 9.44                  home                     2203
         26326                 6.73                credit                     5275
         26695                 6.51                 other                    13918
         26945                 5.65                  cash                     9249
         29841                 6.76                  home                     7223
```

```
                                    ...                               \
client_id                           ...
25707                               ...
26326                               ...
26695                               ...
26945                               ...
29841                               ...

           LAST(loans.rate) / MEAN(loans.rate)  \
client_id
25707                                 2.128271
26326                                 0.575968
26695                                 0.364947
26945                                 1.001634
29841                                 1.477504

           MEAN(loans.loan_amount) / income - log_income  \
client_id
25707                                           0.037671
26326                                           0.031899
26695                                           0.044836
26945                                           0.033221
29841                                           0.255924

           income - log_income / income  \
client_id
25707                          0.999942
26326                          0.999946
26695                          0.999931
26945                          0.999943
29841                          0.999725

           log_income - income / join_month - credit_score  \
client_id
25707                                           346.006118
26326                                           362.910292
26695                                           259.702277
26945                                           269.816005
29841                                            74.453292

           credit_score - log_income / log_income - join_month  \
client_id
25707                                               269.161353
26326                                                84.596484
26695                                               164.116107
26945                                               621.972593
29841                                               200.596006
```

```
            MAX(loans.rate) / credit_score - income  \
client_id
25707                                     -0.000045
26326                                     -0.000030
26695                                     -0.000037
26945                                     -0.000026
29841                                     -0.000179

            income - join_month / credit_score - log_income  \
client_id
25707                                              347.295331
26326                                              367.212011
26695                                              261.290800
26945                                              270.251420
29841                                               74.829438

            log_income - credit_score / join_month - log_income  \
client_id
25707                                                  269.161353
26326                                                   84.596484
26695                                                  164.116107
26945                                                  621.972593
29841                                                  200.596006

            credit_score - join_month / MAX(loans.loan_amount)  \
client_id
25707                                                   0.043916
26326                                                   0.046643
26695                                                   0.045207
26945                                                   0.054478
29841                                                   0.034711

            join_month - credit_score / credit_score - log_income
client_id
25707                                                   -1.003715
26326                                                   -1.011821
26695                                                   -1.006093
26945                                                   -1.001608
29841                                                   -1.004985

[5 rows x 797 columns]
```

In [51]:
```
pd.DataFrame(features['MEAN(payments.payment_amount)'].head())
```

Out[51]:
```
            MEAN(payments.payment_amount)
client_id
25707                         1178.552795
26326                         1166.736842
```

```
26695                   1207.433824
26945                   1109.473214
29841                   1439.433333
```

---

4 Deep Feature Synthesis

While feature primitives are useful by themselves, the main benefit of using feature tools arises when we stack primitives to get deep features. The depth of a feature is simply the number of primitives required to make a feature. So, a feature that relies on a single aggregation would be a deep feature with a depth of 1, a feature that stacks two primitives would have a depth of 2 and so on. The idea itself is lot simpler than the name "deep feature synthesis" implies. (I think the authors were trying to ride the way of deep neural network hype when they named the method!)

```
In [52]: # Show a feature with a depth of 1
         pd.DataFrame(features['MEAN(loans.loan_amount)'].head(10))
```

```
Out[52]:            MEAN(loans.loan_amount)
         client_id
         25707                  7963.950000
         26326                  7270.062500
         26695                  7824.722222
         26945                  7125.933333
         29841                  9813.000000
         32726                  6633.263158
         32885                  9920.400000
         32961                  7882.235294
         35089                  6939.200000
         35214                  7173.555556
```

As well scroll through the features, we see a number of features with a depth of 2. For example, the LAST(loans.(MEAN(payments.payment_amount)) has depth = 2 because it is made by stacking two feature primitives, first an aggregation and then a transformation. This feature represents the average payment amount for the last (most recent) loan for each client.

```
In [53]: # Show a feature with a depth of 2
         pd.DataFrame(features['LAST(loans.MEAN(payments.payment_amount))'].head(10))
```

```
Out[53]:            LAST(loans.MEAN(payments.payment_amount))
         client_id
         25707                                     293.500000
         26326                                     977.375000
         26695                                    1769.166667
         26945                                    1598.666667
         29841                                    1125.500000
         32726                                     799.500000
         32885                                    1729.000000
```

```
32961                                          282.600000
35089                                          110.400000
35214                                         1410.250000
```

We can create features of arbitrary depth by stacking more primitives. **However, when I have used feature tools I've never gone beyond a depth of 2!** After this point, the features become very convoluted to understand. I'd encourage anyone interested to experiment with increasing the depth (maybe for a real problem) and see if there is value to "going deeper".

5 Automated Deep Feature Synthesis

In addition to manually specifying aggregation and transformation feature primitives, we can let feature tools automatically generate many new features. We do this by making the same ft.dfs function call, but without passing in any primitives. We just set the max_depth parameter and feature tools will automatically try many all combinations of feature primitives to the ordered depth.

When running on large datasets, this process can take quite a while, but for our example data, it will be relatively quick. For this call, we only need to specify the entityset, the target_entity (which will again be clients), and the max_depth.

```
In [55]: # Perform deep feature synthesis without specifying primitives
         features, feature_names = ft.dfs(entityset=es, target_entity='clients',
                                          max_depth = 2)
```

```
In [56]: features.shape
```

```
Out[56]: (25, 94)
```

```
In [58]: features.iloc[:, 4:].head()
```

```
Out[58]:           SUM(loans.loan_amount)  SUM(loans.rate)  STD(loans.loan_amount)  \
         client_id
         25707                     159279            69.54             4149.486062
         26326                     116321            40.28             4393.666631
         26695                     140845            44.39             4196.462499
         26945                     106889            42.83             4543.621769
         29841                     176634            62.01             4209.224171

                   STD(loans.rate)  MAX(loans.loan_amount)  MAX(loans.rate)  \
         client_id
         25707            2.484186                   13913             9.44
         26326            2.057142                   13464             6.73
         26695            1.561659                   14865             6.51
         26945            1.619717                   14593             5.65
         29841            2.122904                   14837             6.76

                   SKEW(loans.loan_amount)  SKEW(loans.rate)  MIN(loans.loan_amount)  \
         client_id
```

```
25707              -0.186352      0.735470                   1212
26326               0.149658      1.181651                   1164
26695               0.168879      0.896574                   2389
26945               0.174492     -0.002227                    653
29841              -0.232215      0.055321                   2778

           MIN(loans.rate)             ...               \
client_id                              ...
25707                 0.33             ...
26326                 0.50             ...
26695                 0.22             ...
26945                 0.13             ...
29841                 0.26             ...

           NUM_UNIQUE(loans.WEEKDAY(loan_end))  \
client_id
25707                                        6
26326                                        5
26695                                        6
26945                                        6
29841                                        7

           MODE(loans.MODE(payments.missed))  MODE(loans.DAY(loan_start))  \
client_id
25707                                      0                           27
26326                                      0                            6
26695                                      0                            3
26945                                      0                           16
29841                                      1                            1

           MODE(loans.DAY(loan_end))  MODE(loans.YEAR(loan_start))  \
client_id
25707                              1                          2010
26326                              6                          2003
26695                             14                          2003
26945                              1                          2002
29841                             15                          2005

          MODE(loans.YEAR(loan_end))  MODE(loans.MONTH(loan_start))  \
client_id
25707                           2007                              1
26326                           2005                              4
26695                           2005                              9
26945                           2004                             12
29841                           2007                              3

           MODE(loans.MONTH(loan_end))  MODE(loans.WEEKDAY(loan_start))  \
client_id
```

```
25707                                    8                                3
26326                                    7                                5
26695                                    4                                1
26945                                    5                                0
29841                                    2                                5

           MODE(loans.WEEKDAY(loan_end))
client_id
25707                                  0
26326                                  2
26695                                  1
26945                                  1
29841                                  1

[5 rows x 90 columns]
```

Deep feature synthesis has created 90 new features out of the existing data! While we could have created all of these manually, I am glad to not have to write all that code by hand. The primary benefit of feature tools is that it creates features without any subjective human biases. Even a human with considerable domain knowledge will be limited by their imagination when making new features (not to mention time). Automated feature engineering is not limited by these factors (instead it's limited by computation time) and **provides a good starting point for feature creation**. This process likely will not remove the human contribution to feature engineering completely because a human can still use domain knowledge and machine learning expertise to select the most important features or build new features from those suggested by automated deep feature synthesis.

5.1 Testing with Iris dataset

```
In [70]: from sklearn import datasets
         from sklearn.decomposition import PCA

         # import some data to play with
         iris = datasets.load_iris()
         X = pd.DataFrame(iris.data)  # we only take the first two features.
         X.columns = iris.feature_names
         y = pd.DataFrame(iris.target)
```

```
In [71]: X.head()
```

```
Out[71]:    sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
         0                5.1               3.5                1.4               0.2
         1                4.9               3.0                1.4               0.2
         2                4.7               3.2                1.3               0.2
         3                4.6               3.1                1.5               0.2
         4                5.0               3.6                1.4               0.2
```

```
In [72]: iris.feature_names
```

```
Out[72]: ['sepal length (cm)',
          'sepal width (cm)',
          'petal length (cm)',
          'petal width (cm)']

In [86]: X['index'] = X.index

In [87]: es = ft.EntitySet(id = 'target')
         es

Out[87]: Entityset: target
           Entities:
           Relationships:
             No relationships

In [88]: # https://stackoverflow.com/questions/50145953/how-to-apply-deep-feature-synthesis-to

         es = ft.EntitySet('Transactions')

         es.entity_from_dataframe(dataframe=X,
                                  entity_id='log',
                                  index='index')

Out[88]: Entityset: Transactions
           Entities:
             log [Rows: 150, Columns: 5]
           Relationships:
             No relationships

In [89]: fm, features = ft.dfs(entityset=es,
                               target_entity='log',
                               trans_primitives=['diff'])

In [ ]:

In [ ]:

In [92]: # https://docs.featuretools.com/loading_data/using_entitysets.html#the-raw-data

In [91]: data = ft.demo.load_mock_customer()

In [93]: transactions_df = data["transactions"].merge(data["sessions"]).merge(data["customers"]

In [94]: transactions_df.head()

Out[94]:    transaction_id  session_id    transaction_time product_id  amount  \
         0             352           1 2014-01-01 00:00:00          4    7.39
         1             186           1 2014-01-01 00:01:05          4  147.23
         2             319           1 2014-01-01 00:02:10          2  111.34
         3             256           1 2014-01-01 00:03:15          4   78.15
```

```
4              449          1 2014-01-01 00:04:20          3   33.93

   customer_id   device session_start zip_code  join_date
0            1  desktop    2014-01-01    60091 2008-01-01
1            1  desktop    2014-01-01    60091 2008-01-01
2            1  desktop    2014-01-01    60091 2008-01-01
3            1  desktop    2014-01-01    60091 2008-01-01
4            1  desktop    2014-01-01    60091 2008-01-01
```

```
In [95]: products_df = data["products"]
```

```
In [96]: products_df
```

```
Out[96]:   product_id brand
         0          1     B
         1          2     B
         2          3     C
         3          4     A
         4          5     C
```

```
In [97]: es = ft.EntitySet(id="transactions")
```

```
In [98]: 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.Categori
         es
```

```
Out[98]: Entityset: transactions
           Entities:
             transactions [Rows: 500, Columns: 10]
           Relationships:
             No relationships
```

```
In [99]: es["transactions"].variables
```

```
Out[99]: [<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: zip_code (dtype = categorical)>,
          <Variable: join_date (dtype: datetime, format: None)>,
          <Variable: product_id (dtype = categorical)>]
```

```
In [101]: feature_matrix, feature_defs = ft.dfs(entityset=es,
                                                target_entity="transactions")
```

In [103]: feature_defs

```
Out[103]: [<Feature: session_id>,
 <Feature: amount>,
 <Feature: customer_id>,
 <Feature: device>,
 <Feature: zip_code>,
 <Feature: product_id>,
 <Feature: DAY(transaction_time)>,
 <Feature: DAY(session_start)>,
 <Feature: DAY(join_date)>,
 <Feature: YEAR(transaction_time)>,
 <Feature: YEAR(session_start)>,
 <Feature: YEAR(join_date)>,
 <Feature: MONTH(transaction_time)>,
 <Feature: MONTH(session_start)>,
 <Feature: MONTH(join_date)>,
 <Feature: WEEKDAY(transaction_time)>,
 <Feature: WEEKDAY(session_start)>,
 <Feature: WEEKDAY(join_date)>]
```

In [104]: transactions_df.columns

```
Out[104]: Index(['transaction_id', 'session_id', 'transaction_time', 'product_id',
       'amount', 'customer_id', 'device', 'session_start', 'zip_code',
       'join_date'],
      dtype='object')
```

In [109]:
```
# Perform deep feature synthesis without specifying primitives
feature_matrix, feature_defs = ft.dfs(entityset=es, target_entity='transactions',
                                      max_depth = 3)
```

In [112]:
```
print(feature_matrix.shape)
print()
feature_matrix.head()
```

```
(500, 18)
```

```
Out[112]:                 session_id  amount  customer_id   device zip_code product_id  \
transaction_id
1                       15   37.48            1   mobile    60091          5
2                       16  108.48            3   mobile    02139          4
3                       21   17.27            4  desktop    60091          3
4                       24  125.72            2  desktop    02139          3
5                       16  121.07            3   mobile    02139          4

                DAY(transaction_time)  DAY(session_start)  DAY(join_date)  \
```

```
transaction_id
1                                    1                    1                1
2                                    1                    1               10
3                                    1                    1               30
4                                    1                    1               20
5                                    1                    1               10

                YEAR(transaction_time)  YEAR(session_start)  YEAR(join_date)  \
transaction_id
1                                 2014                 2014             2008
2                                 2014                 2014             2008
3                                 2014                 2014             2008
4                                 2014                 2014             2008
5                                 2014                 2014             2008

                MONTH(transaction_time)  MONTH(session_start)  \
transaction_id
1                                     1                     1
2                                     1                     1
3                                     1                     1
4                                     1                     1
5                                     1                     1

                MONTH(join_date)  WEEKDAY(transaction_time)  \
transaction_id
1                              1                          2
2                              4                          2
3                              5                          2
4                              2                          2
5                              4                          2

                WEEKDAY(session_start)  WEEKDAY(join_date)
transaction_id
1                                    2                   1
2                                    2                   3
3                                    2                   4
4                                    2                   2
5                                    2                   3
```

In [111]:
```
es
```

Out[111]:
```
Entityset: transactions
  Entities:
    transactions [Rows: 500, Columns: 10]
  Relationships:
    No relationships
```

In [ ]: