# Case Study 6.1 - NYC Taxi Trips

Note: If you close this notebook at any time, you will have to run all cells again upon re-opening it.

Note: You may get different numerical results running the notebook different times. This is to be expected, you can just report whatever results you get.

# BEGINNER PYTHON

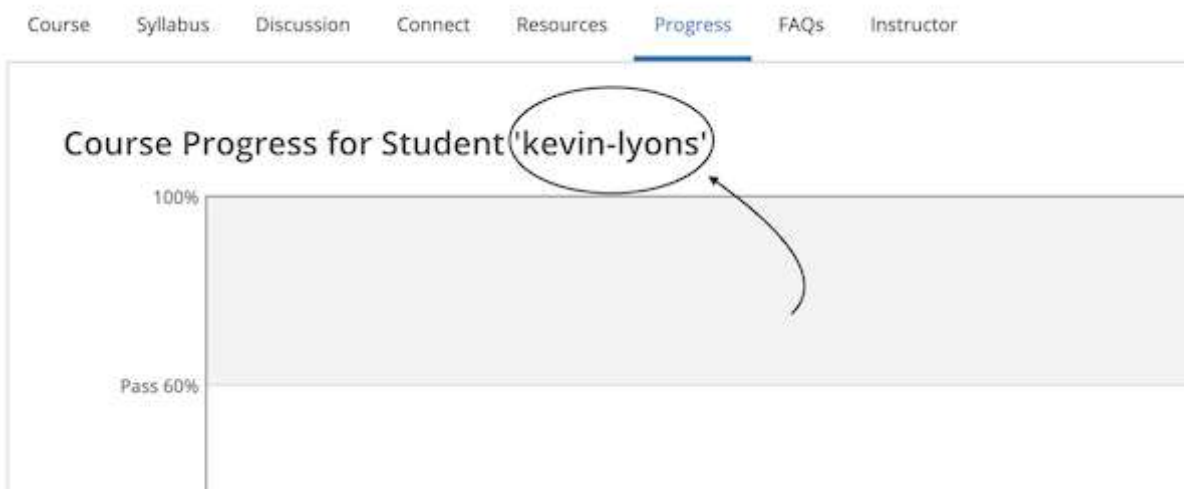
As this is a beginner version, we include a lot of code here to help you along the way.

**First, fill in your identification information below. Then, you only have to type in the answers to the questions we ask you. For the rest of the cells, just Run them by pressing the "Run" button above.**

# Identification Information

You can find your MITxPro username on this page and copying the username in quotations after it says "Student". For example, Kevin's username is "kevin-lyons".

**Please use the correct username and the e-mail address right next to it as it makes it much easier for course staff to identify your notebook.**



**YOUR NAME** = Leonardo Pacheco

**YOUR MITX PRO USERNAME** = leonardo-pacheco

**YOUR MITX PRO E-MAIL** = wleonardop@gmail.com

# Setup

Run these cells to install all the packages you need to complete the remainder of the case study. This may take a few minutes, so please be patient.

## Note: You may see red errors when you run the cell below. As long as you can run the Import cell 2 cells below and see "Install successful!" and "Import successful!", you can continue with the case study.

```
!pip install -q --upgrade pip
!pip install --upgrade featuretools
print('Install successful!')
```

```
     |████████████████████████████████| 1.5MB 7.3MB/s
Collecting featuretools
  Downloading featuretools-0.23.1-py3-none-any.whl (296 kB)
     |████████████████████████████████| 296 kB 7.3 MB/s
Requirement already satisfied: psutil>=5.4.8 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: pandas!=1.1.0,!=1.1.1,<2.0.0,>=0.24.1 in /usr/local
Requirement already satisfied: dask[dataframe]>=2.12.0 in /usr/local/lib/python3.6
Requirement already satisfied: scipy>=0.13.3 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: pyyaml>=3.12 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: click>=7.0.0 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: cloudpickle>=0.4.0 in /usr/local/lib/python3.6/dist
Collecting distributed>=2.12.0
  Downloading distributed-2021.2.0-py3-none-any.whl (675 kB)
     |████████████████████████████████| 675 kB 13.8 MB/s
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: tqdm>=4.32.0 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: toolz>=0.7.3 in /usr/local/lib/python3.6/dist-packa
Collecting fsspec>=0.6.0
  Downloading fsspec-0.8.5-py3-none-any.whl (98 kB)
     |████████████████████████████████| 98 kB 5.1 MB/s
Collecting partd>=0.3.10
  Downloading partd-1.1.0-py3-none-any.whl (19 kB)
Collecting distributed>=2.12.0
  Downloading distributed-2021.1.1-py3-none-any.whl (672 kB)
     |████████████████████████████████| 672 kB 17.8 MB/s
Collecting contextvars
  Downloading contextvars-2.4.tar.gz (9.6 kB)
Requirement already satisfied: zict>=0.1.3 in /usr/local/lib/python3.6/dist-packag
```

```
Collecting cloudpickle>=0.4.0
  Downloading cloudpickle-1.6.0-py3-none-any.whl (23 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: sortedcontainers!=2.0.0,!=2.0.1 in /usr/local/lib/p
Requirement already satisfied: tornado>=5 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: tblib>=1.6.0 in /usr/local/lib/python3.6/dist-packa
Collecting distributed>=2.12.0
  Downloading distributed-2021.1.0-py3-none-any.whl (671 kB)
     |████████████████████████████████| 671 kB 45.4 MB/s
  Downloading distributed-2020.12.0-py3-none-any.whl (669 kB)
     |████████████████████████████████| 669 kB 35.5 MB/s
  Downloading distributed-2.30.1-py3-none-any.whl (656 kB)
     |████████████████████████████████| 656 kB 58.2 MB/s
Requirement already satisfied: msgpack>=0.6.0 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.6/
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packa
Collecting locket
  Downloading locket-0.2.1-py2.py3-none-any.whl (4.1 kB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: heapdict in /usr/local/lib/python3.6/dist-packages
Collecting immutables>=0.9
  Downloading immutables-0.15-cp36-cp36m-manylinux1_x86_64.whl (100 kB)
     |████████████████████████████████| 100 kB 9.8 MB/s
Building wheels for collected packages: contextvars
  Building wheel for contextvars (setup.py) ... done
  Created wheel for contextvars: filename=contextvars-2.4-py3-none-any.whl size=76
  Stored in directory: /root/.cache/pip/wheels/41/11/53/911724983aa48deb94792432e1
Successfully built contextvars
Installing collected packages: locket, immutables, partd, fsspec, contextvars, clo
  Attempting uninstall: cloudpickle
```

Now, you have to restart the runtime by clicking Runtime > Restart Runtime at the top of the page. This will ensure our changes take effect.

# Import

Import the required tools into the notebook.

```
from google.colab import auth
auth.authenticate_user()

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
```

```
drive = GoogleDrive(gauth)
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
data_drop = drive.CreateFile({'id':'1c5KOUwB8m74of93FeXxIakenXwUnz_zc'})
data_drop.GetContentFile('dropoff_neighborhoods.csv')
data_pick = drive.CreateFile({'id':'1QqXoJl14w45ccHG5O9R_1Rb1hi2DhydQ'})
data_pick.GetContentFile('pickup_neighborhoods.csv')
data_trips = drive.CreateFile({'id':'12zOLS9C5TFoZRNUZ4hB7nGinI6LcvBZo'})
data_trips.GetContentFile('trips.pkl')
utils_file = drive.CreateFile({'id':'1-NBnqvEgYGzY0INoxwcQbOq28jH4O8kA'})
utils_file.GetContentFile('utils.py')
```

```
import featuretools as ft
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import utils
from featuretools.primitives import (Count, Day, Hour, Max, Mean, Median, Min,
                                     Minute, Month, Std, Sum, Week, Weekday)
from sklearn.ensemble import GradientBoostingRegressor
from utils import (compute_features, feature_importances, load_nyc_taxi_data,
                   preview)
%matplotlib inline

print('Import successful!')
```

```
Import successful!
```

# Data

Load the NYC taxi trip data. Note that this may take a minute or two, so please be patient.

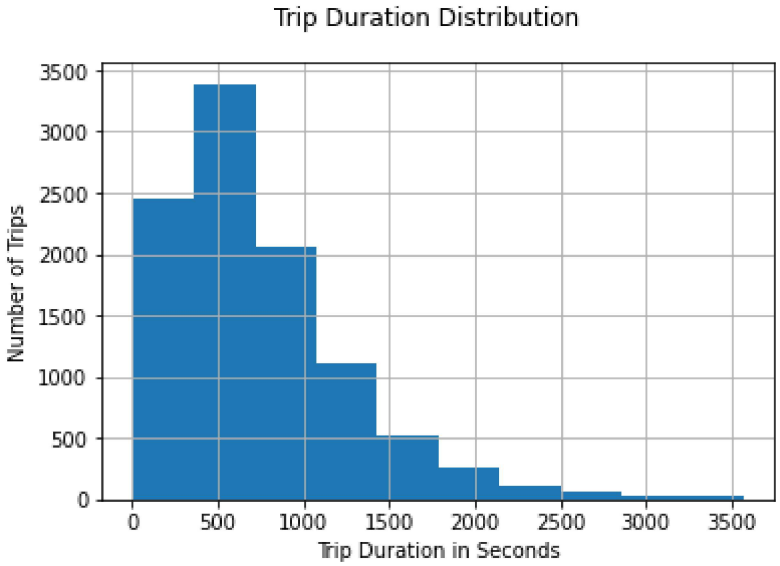
```
trips, pickup_neighborhoods, dropoff_neighborhoods = load_nyc_taxi_data()
print('Data load successful!')
preview(trips, 10)
```

```
Data load successful!
```

| | id | vendor_id | pickup_datetime | dropoff_datetime | passenger_count | trip_dista |
|---|---|---|---|---|---|---|
| **0** | 514030 | 2 | 2016-04-02 00:00:00 | 2016-04-02 00:17:00 | 1 | |
| **1** | 514031 | 1 | 2016-04-02 00:00:00 | 2016-04-02 00:24:00 | 2 | |
| **2** | 514032 | 1 | 2016-04-02 00:00:00 | 2016-04-02 00:19:00 | 1 | |
| **3** | 514033 | 2 | 2016-04-02 00:00:00 | 2016-04-02 00:01:00 | 1 | |
| **4** | 514034 | 1 | 2016-04-02 00:01:00 | 2016-04-02 00:58:00 | 2 | 1 |
| **6** | 514036 | 1 | 2016-04-02 | 2016-04-02 | 1 | |

We can also plot some aspects of the data to get a better sense of its distributions. For instance, here is the `trip_duration` variable we are going to try to predict.

```
trips.trip_duration.hist()
plt.xlabel('Trip Duration in Seconds')
plt.ylabel('Number of Trips')
plt.suptitle('Trip Duration Distribution')
plt.show()
print('Histogram generation successful!')
```




```
Histogram generation successful!
```

```
trips.shape[0]  # Tells us how many trips are in the dataset
```

```
10000
```

# QUESTION 1: DATA ANALYSIS

**Describe the dataset. How many trips are in the dataset? How would you describe the distribution of trip durations? Is there anything else we should observe? Make sure the histogram is visible in the notebook.**

- There are 10000 trips in the dataset.
- The distribution is skewed-left and single-peaked
- Most of the trips have a duration of 500 seconds.

# Entities and Relationships

```
entities = {
    "trips": (trips, "id", 'pickup_datetime'),
    "pickup_neighborhoods": (pickup_neighborhoods, "neighborhood_id"),
    "dropoff_neighborhoods": (dropoff_neighborhoods, "neighborhood_id"),
}

relationships = [("pickup_neighborhoods", "neighborhood_id", "trips", "pickup_neighborhood
                 ("dropoff_neighborhoods", "neighborhood_id", "trips", "dropoff_neighborho

print('Entities and relationships successful!')
```

```
Entities and relationships successful!
```

# Transform Primitives

```
trans_primitives = [Weekday]

# This may take some time to compute
features = ft.dfs(entities=entities,
                  relationships=relationships,
                  target_entity="trips",
                  trans_primitives=trans_primitives,
                  agg_primitives=[],
                  ignore_variables={"trips": ["pickup_latitude", "pickup_longitude",
                                              "dropoff_latitude", "dropoff_longitude", "tr
                  features_only=True)

print('Transform primitives successful!')
```

```
Transform primitives successful!
```

Here are the features that we just created.

```
print(f"Number of features: {len(features)}")
```

```
features
```

```
Number of features: 12
[<Feature: vendor_id>,
 <Feature: passenger_count>,
 <Feature: trip_distance>,
 <Feature: payment_type>,
 <Feature: pickup_neighborhood>,
 <Feature: dropoff_neighborhood>,
 <Feature: WEEKDAY(dropoff_datetime)>,
 <Feature: WEEKDAY(pickup_datetime)>,
 <Feature: pickup_neighborhoods.latitude>,
 <Feature: pickup_neighborhoods.longitude>,
 <Feature: dropoff_neighborhoods.latitude>,
 <Feature: dropoff_neighborhoods.longitude>]
```

Finally, we compute the feature matrix from these features.

```
feature_matrix = compute_features(features, entities, relationships)
preview(feature_matrix, 5)
```

```
Elapsed: 00:00 | Progress: 100%|          /usr/local/lib/python3.6/dist-packages/feat
  warnings.warn(msg)

Finishing computing...
```

| id | vendor_id | pickup_neighborhoods.longitude | dropoff_neighborhood = D | dropoff_nei |
|---|---|---|---|---|
| **514030** | 2 | -73.986446 | False | |
| **514031** | 1 | -73.919159 | False | |
| **514032** | 1 | -73.991595 | False | |
| **514033** | 2 | -73.987205 | False | |
| **514034** | 1 | -73.785073 | False | |

5 rows × 108 columns

# First Model

```
# Split data
X_train, y_train, X_test, y_test = utils.get_train_test_fm(feature_matrix, trips, .75)
y_train = np.log(y_train + 1)
y_test = np.log(y_test + 1)

print('Data split successful!')
```

```
Data split successful!
```

```
# This should train within a minute or so
```

```
# This should train within a minute or so
model = GradientBoostingRegressor(verbose=True)
model.fit(X_train, y_train)
print(model.score(X_test, y_test)) # This is the R^2 value of the prediction

print('Training successful!')
```

```
      Iter       Train Loss   Remaining Time
         1           0.4736            1.94s
         2           0.4148            1.89s
         3           0.3661            1.83s
         4           0.3266            1.81s
         5           0.2934            1.79s
         6           0.2665            1.76s
         7           0.2441            1.75s
         8           0.2257            1.72s
         9           0.2103            1.70s
        10           0.1973            1.68s
        20           0.1433            1.50s
        30           0.1307            1.32s
        40           0.1248            1.11s
        50           0.1214            0.91s
        60           0.1190            0.72s
        70           0.1165            0.54s
        80           0.1149            0.36s
        90           0.1138            0.18s
       100           0.1127            0.00s
0.7744673561948765
Training successful!
```

# QUESTION 2: FIRST MODEL

**Describe all the features that we added to the model. Do you think these improved the performance from a model that did not have these features? Why?**

- We added two boolean features for each Neighborhood, one corresponds to the pickup and the other to dropoff, this features specify if a trip starts or ends in a certain neighborhood.
- The WEEKDAY primitive was used to add two features, namely
  - WEEKDAY(pickup_datetime)
  - WEEKDAY(dropoff_datetime)
- Since the WEEKDAY primitive returns the day of a timestamp, the new features really improve the performance from a model that did not have these features. Indeed the mood of the passengers may be related to the day of the week, and consequently to the duration of the trips.

# More Transform Primitives

```
trans_primitives = [Minute, Hour, Day, Week, Month, Weekday]

features = ft.dfs(entities=entities,
                  relationships=relationships,
                  target_entity="trips",
                  trans_primitives=trans_primitives,
                  agg_primitives=[],
                  ignore_variables={"trips": ["pickup_latitude", "pickup_longitude",
                                              "dropoff_latitude", "dropoff_longitude", "tr
                  features_only=True)

print('Transform primitives successful!')
```

```
Transform primitives successful!
```

```
print(f"Number of features: {len(features)}")
features
```

```
Number of features: 22
[<Feature: vendor_id>,
 <Feature: passenger_count>,
 <Feature: trip_distance>,
 <Feature: payment_type>,
 <Feature: pickup_neighborhood>,
 <Feature: dropoff_neighborhood>,
 <Feature: DAY(dropoff_datetime)>,
 <Feature: DAY(pickup_datetime)>,
 <Feature: HOUR(dropoff_datetime)>,
 <Feature: HOUR(pickup_datetime)>,
 <Feature: MINUTE(dropoff_datetime)>,
 <Feature: MINUTE(pickup_datetime)>,
 <Feature: MONTH(dropoff_datetime)>,
 <Feature: MONTH(pickup_datetime)>,
 <Feature: WEEK(dropoff_datetime)>,
 <Feature: WEEK(pickup_datetime)>,
 <Feature: WEEKDAY(dropoff_datetime)>,
 <Feature: WEEKDAY(pickup_datetime)>,
 <Feature: pickup_neighborhoods.latitude>,
 <Feature: pickup_neighborhoods.longitude>,
 <Feature: dropoff_neighborhoods.latitude>,
 <Feature: dropoff_neighborhoods.longitude>]
```

```
feature_matrix = compute_features(features, entities, relationships)
preview(feature_matrix, 5)
```

```
Elapsed: 00:00 | Progress: 100%|          /usr/local/lib/python3.6/dist-packages/feat
  warnings.warn(msg)

Finishing computing...
```

| id | HOUR(dropoff_datetime) | WEEKDAY(pickup_datetime) | dropoff_neighborhoods.long |
|---|---|---|---|
| **514030** | 0 | 5 | -73.9 |

```
# Re-split data
X_train, y_train, X_test, y_test = utils.get_train_test_fm(feature_matrix, trips, .75)
y_train = np.log(y_train + 1)
y_test = np.log(y_test + 1)

print('Data split successful!')
```

```
Data split successful!
```

```
# This should train within a minute or so
model = GradientBoostingRegressor(verbose=True)
model.fit(X_train, y_train)
print(model.score(X_test, y_test)) # This is the R^2 value of the prediction

print('Training successful!')
```

```
      Iter       Train Loss   Remaining Time
         1           0.4736            2.34s
         2           0.4148            2.30s
         3           0.3661            2.25s
         4           0.3264            2.21s
         5           0.2930            2.17s
         6           0.2660            2.14s
         7           0.2432            2.11s
         8           0.2245            2.08s
         9           0.2090            2.09s
        10           0.1960            2.07s
        20           0.1362            1.84s
        30           0.1198            1.61s
        40           0.1124            1.91s
        50           0.1075            1.61s
        60           0.1044            1.27s
        70           0.1017            0.91s
        80           0.0999            0.61s
        90           0.0971            0.29s
       100           0.0936            0.00s
0.8008872871961774
Training successful!
```

# QUESTION 3: SECOND MODEL

**Describe the rest of the new features that we just added to the model. How did this affect performance? Did we have to sacrifice training time?**

- We just added 10 more features using the following transformation primitives:
  - MONTH,
  - WEEK,
  - DAY,
  - HOUR and
  - MINUTE.

  Each primitive was used on the timestamps pickup-time and dropoff-time.
- The training time was longer compared with the first model, this is because we increased the number of features and the problem was more complex to solve.

# Aggregation Primitives

```
trans_primitives = [Minute, Hour, Day, Week, Month, Weekday]
aggregation_primitives = [Count, Sum, Mean, Median, Std, Max, Min]

features = ft.dfs(entities=entities,
                  relationships=relationships,
                  target_entity="trips",
                  trans_primitives=trans_primitives,
                  agg_primitives=aggregation_primitives,
                  ignore_variables={"trips": ["pickup_latitude", "pickup_longitude",
                                              "dropoff_latitude", "dropoff_longitude", "tr
                  features_only=True)

print('Aggregation primitives successful!')
```

```
Aggregation primitives successful!
```

```
print(f"Number of features: {len(features)}")
features
```

```
Number of features: 60
[<Feature: vendor_id>,
 <Feature: passenger_count>,
 <Feature: trip_distance>,
 <Feature: payment_type>,
 <Feature: pickup_neighborhood>,
 <Feature: dropoff_neighborhood>,
 <Feature: DAY(dropoff_datetime)>,
 <Feature: DAY(pickup_datetime)>,
 <Feature: HOUR(dropoff_datetime)>,
 <Feature: HOUR(pickup_datetime)>,
 <Feature: MINUTE(dropoff_datetime)>,
 <Feature: MINUTE(pickup_datetime)>,
 <Feature: MONTH(dropoff_datetime)>,
 <Feature: MONTH(pickup_datetime)>,
 <Feature: WEEK(dropoff_datetime)>,
 <Feature: WEEK(pickup_datetime)>,
```

```
        <Feature: WEEKDAY(dropoff_datetime)>,
        <Feature: WEEKDAY(pickup_datetime)>,
        <Feature: pickup_neighborhoods.latitude>,
        <Feature: pickup_neighborhoods.longitude>,
        <Feature: dropoff_neighborhoods.latitude>,
        <Feature: dropoff_neighborhoods.longitude>,
        <Feature: pickup_neighborhoods.COUNT(trips)>,
        <Feature: pickup_neighborhoods.MAX(trips.passenger_count)>,
        <Feature: pickup_neighborhoods.MAX(trips.trip_distance)>,
        <Feature: pickup_neighborhoods.MAX(trips.vendor_id)>,
        <Feature: pickup_neighborhoods.MEAN(trips.passenger_count)>,
        <Feature: pickup_neighborhoods.MEAN(trips.trip_distance)>,
        <Feature: pickup_neighborhoods.MEAN(trips.vendor_id)>,
        <Feature: pickup_neighborhoods.MEDIAN(trips.passenger_count)>,
        <Feature: pickup_neighborhoods.MEDIAN(trips.trip_distance)>,
        <Feature: pickup_neighborhoods.MEDIAN(trips.vendor_id)>,
        <Feature: pickup_neighborhoods.MIN(trips.passenger_count)>,
        <Feature: pickup_neighborhoods.MIN(trips.trip_distance)>,
        <Feature: pickup_neighborhoods.MIN(trips.vendor_id)>,
        <Feature: pickup_neighborhoods.STD(trips.passenger_count)>,
        <Feature: pickup_neighborhoods.STD(trips.trip_distance)>,
        <Feature: pickup_neighborhoods.STD(trips.vendor_id)>,
        <Feature: pickup_neighborhoods.SUM(trips.passenger_count)>,
        <Feature: pickup_neighborhoods.SUM(trips.trip_distance)>,
        <Feature: pickup_neighborhoods.SUM(trips.vendor_id)>,
        <Feature: dropoff_neighborhoods.COUNT(trips)>,
        <Feature: dropoff_neighborhoods.MAX(trips.passenger_count)>,
        <Feature: dropoff_neighborhoods.MAX(trips.trip_distance)>,
        <Feature: dropoff_neighborhoods.MAX(trips.vendor_id)>,
        <Feature: dropoff_neighborhoods.MEAN(trips.passenger_count)>,
        <Feature: dropoff_neighborhoods.MEAN(trips.trip_distance)>,
        <Feature: dropoff_neighborhoods.MEAN(trips.vendor_id)>,
        <Feature: dropoff_neighborhoods.MEDIAN(trips.passenger_count)>,
        <Feature: dropoff_neighborhoods.MEDIAN(trips.trip_distance)>,
        <Feature: dropoff_neighborhoods.MEDIAN(trips.vendor_id)>,
        <Feature: dropoff_neighborhoods.MIN(trips.passenger_count)>,
        <Feature: dropoff_neighborhoods.MIN(trips.trip_distance)>,
        <Feature: dropoff_neighborhoods.MIN(trips.vendor_id)>,
        <Feature: dropoff_neighborhoods.STD(trips.passenger_count)>,
        <Feature: dropoff_neighborhoods.STD(trips.trip_distance)>,
        <Feature: dropoff_neighborhoods.STD(trips.vendor_id)>,
        <Feature: dropoff_neighborhoods.SUM(trips.passenger_count)>,
```

```
# This may take a bit longer to compute, so please be patient
feature_matrix = compute_features(features, entities, relationships)
preview(feature_matrix, 5)
```

```
Elapsed: 00:00 | Progress:   0%|          /usr/local/lib/python3.6/dist-packages/feat
  warnings.warn(msg)
Elapsed: 00:00 | Progress: 100%|██████████
Finishing computing...
```

| | pickup_neighborhoods.MEDIAN(trips.trip_distance) | dropoff_neighborhoods.MAX |
|---|---|---|

```
# Re-split data
X_train, y_train, X_test, y_test = utils.get_train_test_fm(feature_matrix, trips, .75)
y_train = np.log(y_train + 1)
y_test = np.log(y_test + 1)

print('Data split successful!')
```

```
Data split successful!
```

```
# This should train within a minute or so
model = GradientBoostingRegressor(verbose=True)
model.fit(X_train, y_train)
print(model.score(X_test, y_test)) # This is the R^2 value of the prediction

print('Training successful!')
```

```
      Iter       Train Loss   Remaining Time
         1           0.4736            4.23s
         2           0.4148            4.14s
         3           0.3661            4.25s
         4           0.3264            4.17s
         5           0.2930            4.14s
         6           0.2660            4.13s
         7           0.2432            4.08s
         8           0.2245            4.05s
         9           0.2090            4.00s
        10           0.1960            3.96s
        20           0.1363            3.54s
        30           0.1198            3.08s
        40           0.1114            2.60s
        50           0.1061            2.15s
        60           0.1027            1.71s
        70           0.1001            1.28s
        80           0.0982            0.85s
        90           0.0964            0.42s
       100           0.0928            0.00s
0.8013486211660491
Training successful!
```

# Evaluate on Test Data

```
y_pred = model.predict(X_test)
y_pred = np.exp(y_pred) - 1 # undo the log we took earlier

print('y_pred computation successful!')
```

```
y_pred computation successful!
```

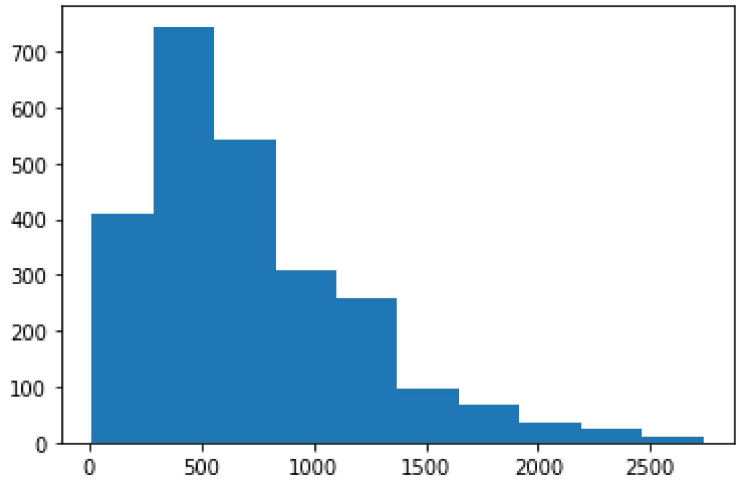
```
# Print the first 5 predictions
y_pred[:5]
```

```
array([438.04673321, 663.53356423, 662.35634254, 622.60224164,
       413.52728709])
```

```
# Create a histogram of all of them
matplotlib.pyplot.hist(y_pred)

print('Histogram generation successful!')
```

```
Histogram generation successful!
```




# QUESTION 4: MODEL PREDICTIONS

**Analyze the model predictions. Does the output distribution match the one you made earlier in the case study? What other features/strategies could we use to make our model even better, if we had more time?**

- Yes indeed the output distribution match the one we made earlier with the dataset.
  - The distribution is skewed-left and single-peaked.
  - Most of the trips have a duration of 500 seconds.
- To improve the model, we could make an inference analysis to drop some features that don't give relevant information.
- It may be useful to interview the taxi drivers too, in fact, they would give us some clue about new features that we could add to the model.

# Feature Importance

```
feature_importances(model, feature_matrix.columns, n=25)
```

```
1: Feature: trip_distance, 0.866
2: Feature: dropoff_neighborhoods.longitude, 0.028
3: Feature: HOUR(pickup_datetime), 0.027
4: Feature: dropoff_neighborhoods.latitude, 0.021
5: Feature: HOUR(dropoff_datetime), 0.013
6: Feature: pickup_neighborhoods.COUNT(trips), 0.004
7: Feature: MINUTE(dropoff_datetime), 0.003
8: Feature: dropoff_neighborhoods.COUNT(trips), 0.003
9: Feature: pickup_neighborhoods.longitude, 0.003
10: Feature: pickup_neighborhoods.latitude, 0.003
11: Feature: MINUTE(pickup_datetime), 0.002
12: Feature: dropoff_neighborhoods.SUM(trips.trip_distance), 0.002
13: Feature: pickup_neighborhoods.MEDIAN(trips.trip_distance), 0.002
14: Feature: pickup_neighborhoods.SUM(trips.trip_distance), 0.002
15: Feature: dropoff_neighborhoods.SUM(trips.passenger_count), 0.001
16: Feature: pickup_neighborhoods.MEAN(trips.trip_distance), 0.001
17: Feature: pickup_neighborhood = AU, 0.001
18: Feature: dropoff_neighborhood = AC, 0.001
19: Feature: dropoff_neighborhoods.SUM(trips.vendor_id), 0.001
20: Feature: pickup_neighborhoods.MEAN(trips.passenger_count), 0.001
21: Feature: dropoff_neighborhood = E, 0.001
22: Feature: dropoff_neighborhoods.MAX(trips.trip_distance), 0.001
23: Feature: dropoff_neighborhoods.MEAN(trips.vendor_id), 0.001
24: Feature: pickup_neighborhoods.STD(trips.trip_distance), 0.001
25: Feature: dropoff_neighborhoods.MEAN(trips.passenger_count), 0.001
```

# QUESTION 5: FEATURE IMPORTANCE

**Analyze the feature importance values you just computed above. Do they make sense? Are there any values you are surprised by? Give some brief explanations as to why these features are relevant in computing the `trip_duration` target variable.**

- The feature importance values make sense.
- The trip_distance is the most important, indeed there is a direct proportionality between this variable and the trip_duration.
- The second most important feature is the dropoff_neighborhoods.longitude, this is surprising for me. It seems that the destiny of the trips is "highly" related to the duration.
- The third most important feature is HOUR(pickup date-time). For me this was expected, depending on the start hour the passengers will make longer or shorter trips.
- The fourth most important feature is dropoff neighborhoods.latitude. This agrees with the above statement about the relation of destiny to the duration.
- The fifth most important feature is HOUR(dropoff_datetime), not only the start hour of the trip predicts the duration, the end hour too.
- The sixth most important feature is pickup_neighborhoods.COUNT(trips).

- Depending on the number of neighborhoods where a passanger started trips in the past the duration of a new of his trips will be.
- The other features have less importance than 0.004 an can be clasified in tree groups:
  - 0.003: MINUTE(dropoff_datetime), dropoff_neighborhoods.COUNT(trips), pickup_neighborhoods.longitude, pickup_neighborhoods.latitude
  - 0.002:MINUTE(pickup_datetime),dropoff_neighborhoods.SUM(trips.trip_distance), pickup_neighborhoods.MEDIAN(trips.trip_distance), pickup_neighborhoods.SUM(trips.trip_distance)
  - 0.001: dropoff_neighborhoods.SUM(trips.passenger_count),pickup_neighborhoods.MEAN(trips.trip_distance),pickup_neighborhood = AU, dropoff_neighborhood = AC, Feature: dropoff_neighborhoods.SUM(trips.vendor_id), pickup_neighborhoods.MEAN(trips.passenger_count), dropoff_neighborhood = E, dropoff_neighborhoods.MAX(trips.trip_distance), dropoff_neighborhoods.MEAN(trips.vendor_id), pickup_neighborhoods.STD(trips.trip_distance), dropoff_neighborhoods.MEAN(trips.passenger_count)

---

Great job! Now, make sure you check out the **Conclusion** section of the instruction manual to wrap up this case study properly.