Capstone 1 Final Report - Lyft Baywheels Analysis

Table of Contents

Context

This summer Lyft rebranded its Ford GoBikes to create a fully Lyft branded suite of share bikes, black with the quintessentially Lyft-like pink rims. I started to see them all over the place in SF and the East Bay and began to wonder about them.

Doing some digging, I realized that these bikes comes out of a longstanding lawsuit with the city of San Francisco and is an effort by Lyft to re-establish control over the bikeshare market in SF. The issue was that Uber's JUMP bikes had encroached on a supposed 10 year agreement that Lyft had, stating Lyft had sole operational privilege of share bikes in the city.

In what may be an effort of good will towards data-savvy San Franciscans, Lyft opened up their anonymized Baywheels trip data, showing 2.5 years of rides as well as API access to real-time trip data - the locations of every bike. I discovered the historical trip data and wanted to dive in to explore a number of questions.

Problem statement

In looking at these data, I hope to answer the following questions:

1. Who is riding Bay Wheels (distributions for variables like age, gender, and rider type)?
2. Can you predict how long someone will ride based on their age, gender, rider type, etc.?
3. Can you predict if a rider is a subscriber or just a customer based on age, gender, etc.?

By answering the above three questions and the second one in particular, Lyft would be able to predict how much revenue they might expect from a ride and a rider type. By scaling this, one could expect them to be able to predict how much revenue can be expected from the system as a whole.

Stakeholders

The key stakeholders would be Lyft, people involved in the share economy more generally, and perhaps data-savvy bike-share users as well. Perhaps this analysis could show to the city of San Francisco that with enough investment and enablement, Baywheels could provide the city's bike sharing needs without competitors. The main reason the city of SF is fighting back on the lawsuit was that they wanted more bikesharing offerings. Maybe it’s about volume, or maybe it’s about a fair and evenly distributed market.

The Data

Lyft has made available a directory of data hosted on AWS, illustrating bike-share rides over the past two and a half years. Details on these data and the link to the real-time data feed is available on their Systems Data page. I wrote a Python script and scraped all historical trip data, concatenating them together into 3.4M rows of rides. The script can be seen below.

Acquiring the Data

Once I acquired a list of all the filenames from the directory, I downloaded all the zip folders containing them and unzipped each locally. The whole step for acquiring the main data is outlined below:

```
import os
import requests
from bs4 import BeautifulSoup
import re

# Move to the data folder for this project
os.chdir('C:/Users/riley/Documents/Coding/DSC/datas/baywheels-data/')

# Get the main URL and turn it into XML soup
r = requests.get("https://s3.amazonaws.com/baywheels-data/")
data = r.text
soup = BeautifulSoup(data, "xml")

# The data file URLs are in the <Key> node so find that
keys = soup.find_all('Key')

key_list = []

# Grab the name of each file by taking the text attribute of the key node
for key in keys:
    key_list.append(key.text)

# Slice off the trailing file, index.html
key_list_final = key_list[:-1]
```

```
# Downloads all files in the list of filenames.
# Deal with extracting them in the next cell.

import zipfile, urllib.request, shutil

for key in key_list_final:

    url = 'https://s3.amazonaws.com/baywheels-data/' + key
    file_name = key

    with urllib.request.urlopen(url) as response, open(file_name, 'wb') as out_file:
        shutil.copyfileobj(response, out_file)
```

```python
# For all items ending in .zip in the dir_name,
# extract their contents to dir_name and delete the original zip folder.

extension = ".zip"
dir_name = 'C:/Users/riley/Documents/Coding/DSC/datas/baywheels-data'

for item in os.listdir(dir_name): # loop through items in dir
    if item.endswith(extension): # check for ".zip" extension
        file_name = os.path.abspath(item) # get full path of files
        zip_ref = zipfile.ZipFile(file_name) # create zipfile object
        zip_ref.extractall(dir_name) # extract file to dir
        zip_ref.close() # close file
        os.remove(file_name) # delete zipped file
```

```python
# Create a list of all CSVs in the directory to iterate over and make
# dataframes from.

dir_name = 'C:/Users/riley/Documents/Coding/DSC/datas/baywheels-data/'

# List of all .csv filenames to be read into dataframes
csv_list = os.listdir(dir_name)

# Build a list of dataframe names by removing .csv
df_names = []

for csv in csv_list:
    df_name = csv.replace('.csv', '')
    df_names.append(df_name)
```

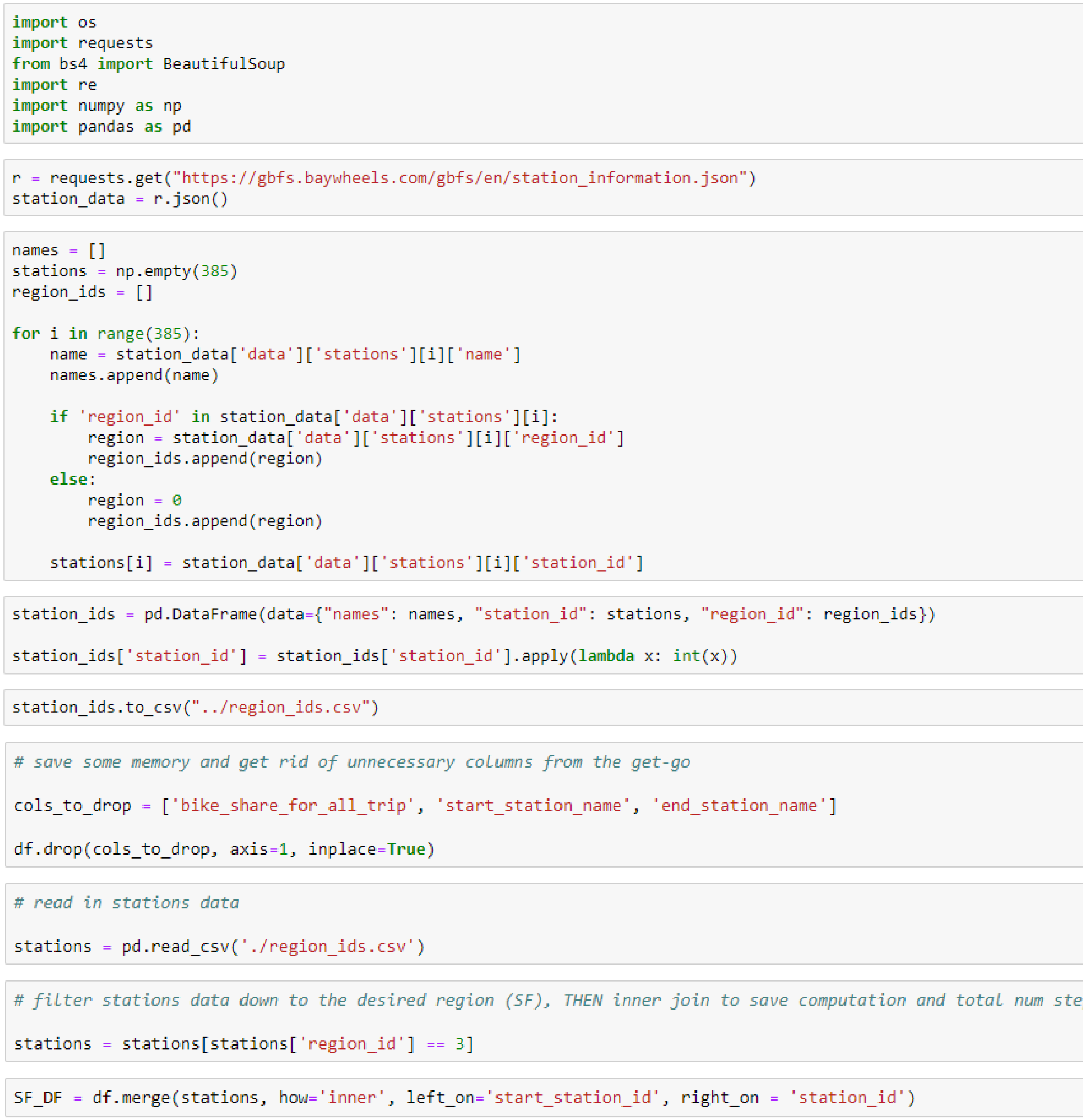
```python
# Names of DFs and the filenames
df_dict_names = dict(zip(df_names, csv_list))
```

```python
import pandas as pd

# Create a dictionary of dataframes to then concatenate into one dataframe

dict_of_dataframes = {}
for i in range(len(df_names)):
    dict_of_dataframes[df_names[i]] = pd.read_csv(csv_list[i])
```

```python
master_df = pd.concat(dict_of_dataframes.values())
```

The Master Dataframe

The above step assembled the master dataframe. I grabbed the station and region IDs from the real-time system data API, joined those on station IDs in the master dataframe, and then filtered by the SF region ID because the dataframe was unnecessarily large for modeling. It originally contained SF, the East Bay, and San Jose bike rides. To create the master dataframe I chunked the different dataframes and that allowed me to build it. After that I used the region IDs dataframe I got from the API. The code for grabbing the station region IDs data is below:



```python
import os
import requests
from bs4 import BeautifulSoup
import re
import numpy as np
import pandas as pd
```

```python
r = requests.get("https://gbfs.baywheels.com/gbfs/en/station_information.json")
station_data = r.json()
```

```python
names = []
stations = np.empty(385)
region_ids = []

for i in range(385):
    name = station_data['data']['stations'][i]['name']
    names.append(name)

    if 'region_id' in station_data['data']['stations'][i]:
        region = station_data['data']['stations'][i]['region_id']
        region_ids.append(region)
    else:
        region = 0
        region_ids.append(region)

    stations[i] = station_data['data']['stations'][i]['station_id']
```

```python
station_ids = pd.DataFrame(data={"names": names, "station_id": stations, "region_id": region_ids})

station_ids['station_id'] = station_ids['station_id'].apply(lambda x: int(x))
```

```python
station_ids.to_csv("../region_ids.csv")
```

```python
# save some memory and get rid of unnecessary columns from the get-go

cols_to_drop = ['bike_share_for_all_trip', 'start_station_name', 'end_station_name']

df.drop(cols_to_drop, axis=1, inplace=True)
```

```python
# read in stations data

stations = pd.read_csv('./region_ids.csv')
```

```python
# filter stations data down to the desired region (SF), THEN inner join to save computation and total num ste

stations = stations[stations['region_id'] == 3]
```

```python
SF_DF = df.merge(stations, how='inner', left_on='start_station_id', right_on = 'station_id')
```

Note: I did not include all data from the original repository because 2019 is only up until the current month on a rolling basis. If I did any analysis on the year or month, the 2019 data would be underrepresented in the sample and thus would make it difficult to draw inferences from.

Cleaning the Data

The final dataframe is known as 'SF_DF' henceforth. For cleaning the data, I decided to remove the names of the stations and the bike_share_for_all_trip columns because I already have station latitudes and longitudes so the names are distracting/unhelpful. The bike_share_for_all_trip column is a marketing campaign that was a very small subset of the data and I didn't care to analyze it. The steps I took are below for data cleaning as well as creating new features like the cost of ride. The logic for deriving that value is outlined in the code below:

```
import numpy as np

# Get min from seconds column
SF_DF['duration_min'] = np.round(SF_DF['duration_sec'] / 60, 0)
```

```
# Use pd.to_datetime to convert the time strings to datetime objects
# for easier time series analyses

SF_DF['start_time'] = pd.to_datetime(SF_DF['start_time'])
SF_DF['end_time'] = pd.to_datetime(SF_DF['end_time'])
```

```
# Encode the different categorical genders to integers
genders = {'Male': 0, 'Female':1, 'Other':2}

# Use replace to replace each key in genders dict with corresponding value
SF_DF['member_gender'].replace(genders, inplace=True);
```

```
cols_to_drop = ['region_id', 'names', 'duration_sec']

SF_DF.drop(cols_to_drop, axis=1, inplace=True)
```

```
import datetime

# Will now interpolate the member_birth_year with the mean birth year
# Also, I'm gonna get their age by subtracting now() from their birth year
SF_DF['member_birth_year'].fillna(np.mean(SF_DF['member_birth_year']), inplace=True)

# That got years as floats so I'm changing it to int:
SF_DF['member_birth_year'] = SF_DF['member_birth_year'].apply(np.int64)

# Now let's get the current year and subtract that from birth year to get age.
now = datetime.datetime.now()
SF_DF['age'] = now.year - SF_DF['member_birth_year']
```

```
# Apply the price calculator to separate dfs, one for each customer type
# this is pretty fast computationally

import math as m


customers = SF_DF[SF_DF['user_type'] == 'Customer']

cost_calc = lambda minute: 2 + m.ceil(max((minute - 30), 0) / 15) * 3

customers['cost_to_ride'] = customers['duration_min'].apply(cost_calc)

subscribers = SF_DF[SF_DF['user_type'] == 'Subscriber']

cost_calc = lambda minute: m.ceil(max((minute - 45), 0) / 15) * 3

subscribers['cost_to_ride'] = subscribers['duration_min'].apply(cost_calc)

SF_DF = pd.concat([subscribers, customers])
```

Glancing at the data I saw that rides were in seconds, so I added a column duration_min, which converted it to minutes - this seemed to make more sense and be more useful. I also converted the start and end times to pandas datetime objects since they were strings initially. This was so I could do Time Series analysis.

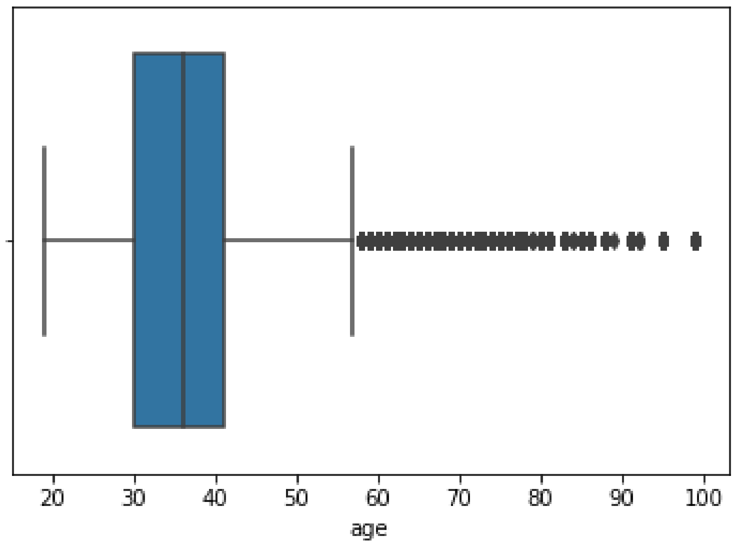
I encoded the gender columns as floats because these process faster than strings. I was also able to interpolate the age of the riders from their member_birth_year column.

I created columns for day, hour, etc. from the datetime objects previously created. I wanted to look at a heatmap of rides by day and hour. However, because not all months of the year have 31 days, I removed the 31st day. This was hard to add into the main dataframe since it didn’t make sense that each row corresponds to a ride but would also have a frequency of rides by day column. As such this sits in a different dataframe: days.

Exploratory Data Analysis (EDA)

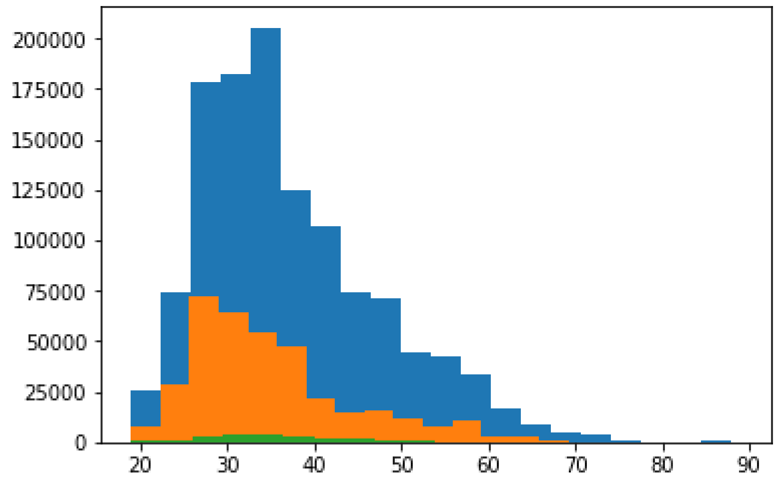
Distributions

In order to understand what the data looks like, I looked at the distributions. We can see that overall age was quite spread out. We can see this easily if we take a look at the box and whisker plot for it below:



The median age is about 38, but there were many single instances of riders whose ages were greater than about 58. I decided based on this to go back to data cleaning for a second and filter out those who were older than 70. I figured this would get rid of some fake ages too and clean up the data.

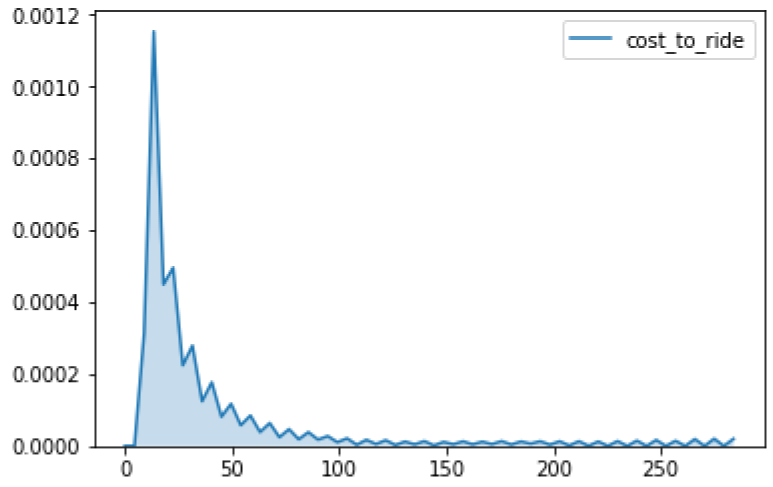
Age by gender histogram (blue=male, orange=female, green=other):



From the above chart we can glean several things:

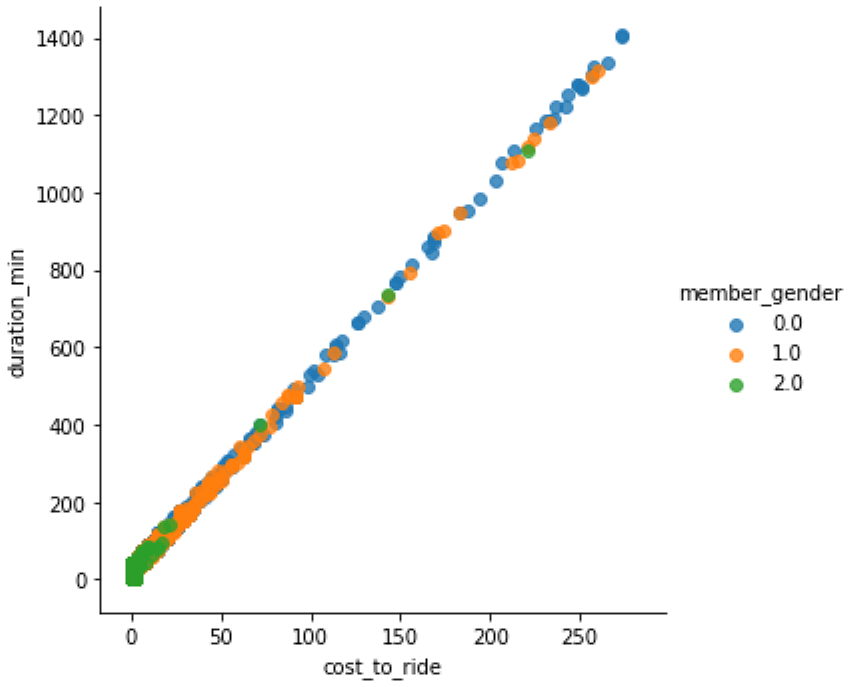
1. The age distributions for all genders appears to be right skewed
2. The median age seems to be somewhere between 30 and 35
3. There are more men than any other gender in these data
4. There are about half as many women than men in these data
5. Other is the least common gender type

The revenue gains from rides appears to be exponentially distributed (and is highly correlated with the duration of the ride since that determines the cost of the ride):



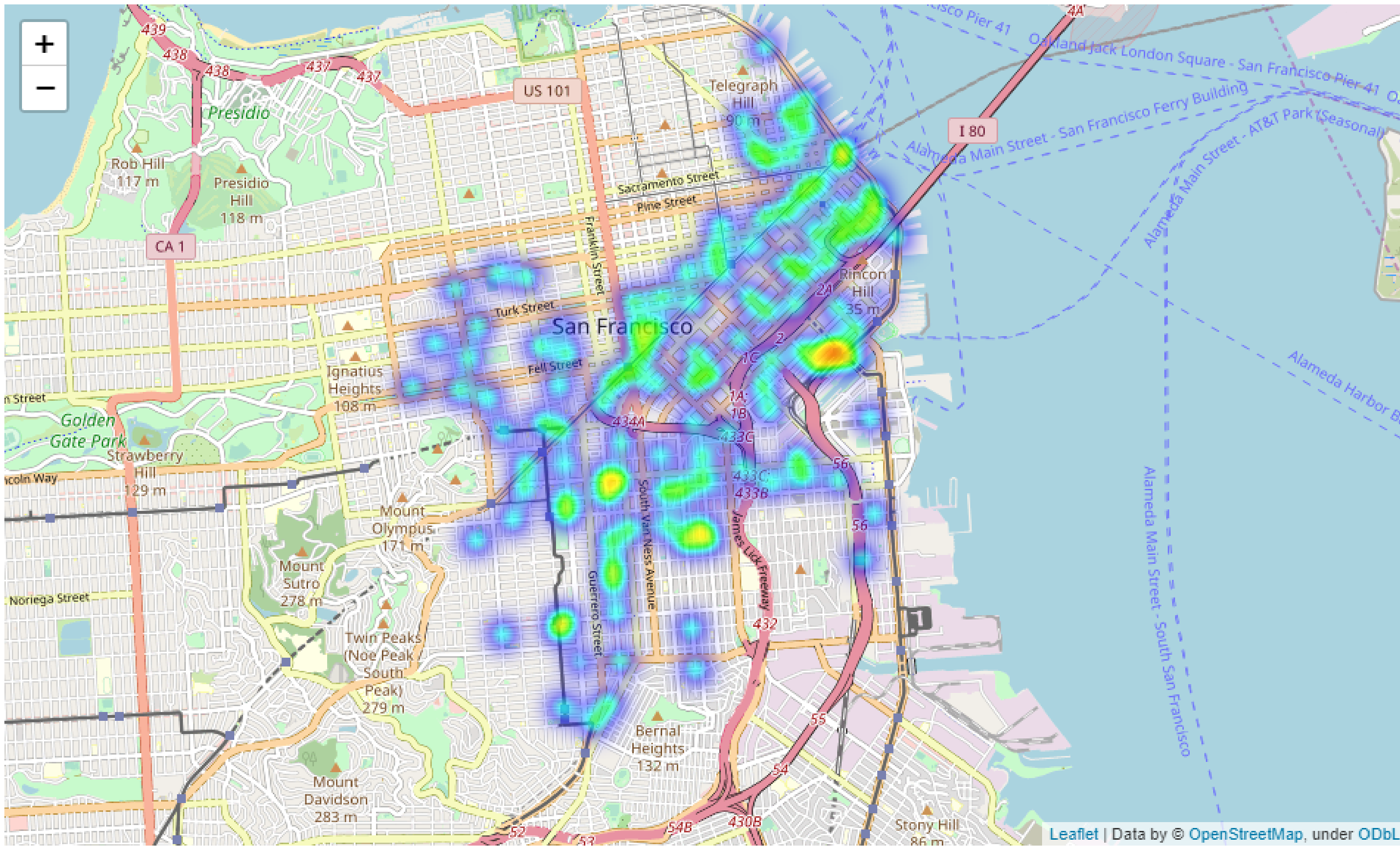
The majority of costs are below $25 it appears for a given ride.

If we look at the cost of rides and duration of rides split by gender, we can see the perfect correlation between them as cost is derived from duration. But what’s interesting too is that there are more men (member_gender = 0.0) taking longer rides than women (member_gender = 1.0):



If we look at the locations from which riders start their rides, we can see a pattern as well, with more rides happening in the financial district, south of market, and the mission districts of San Francisco. These tend to be tech-savvy and popular areas, and are probably where many of the bikes are located. Walking around in the area confirms this at a cursory glance.

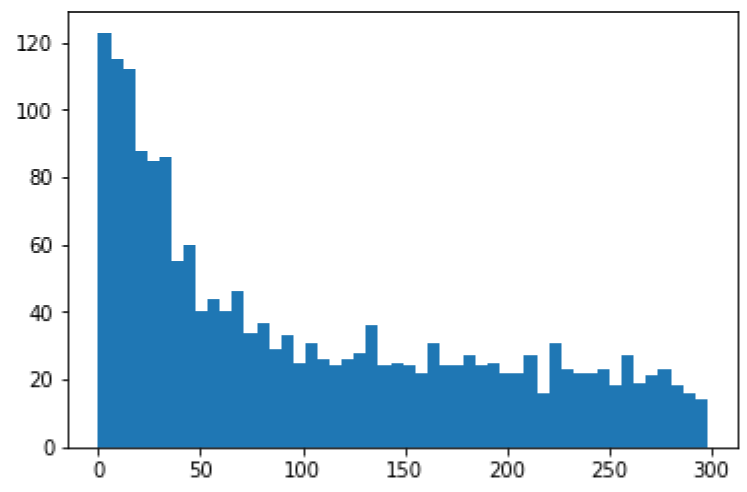
Map of ride start locations, with green and yellow indicating higher volumes



Interesting findings

If we had assumed the revenue generated by bikes was normally distributed, then the number of people who took bikes and the duration of rides would also be normally distributed. Each bike would also have a normally distributed likelihood of being selected. This was in fact not the case (and of course none of the above variables would be normally distributed because bikes could be anywhere in the city, and riders are likely concentrated in the financial district, south of market, and similar areas):

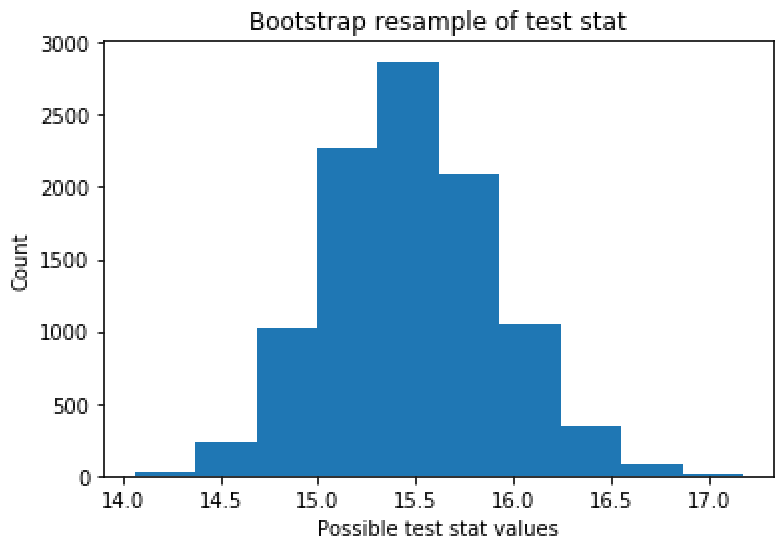
Revenue by Bike ID (Y-axis = total money made):



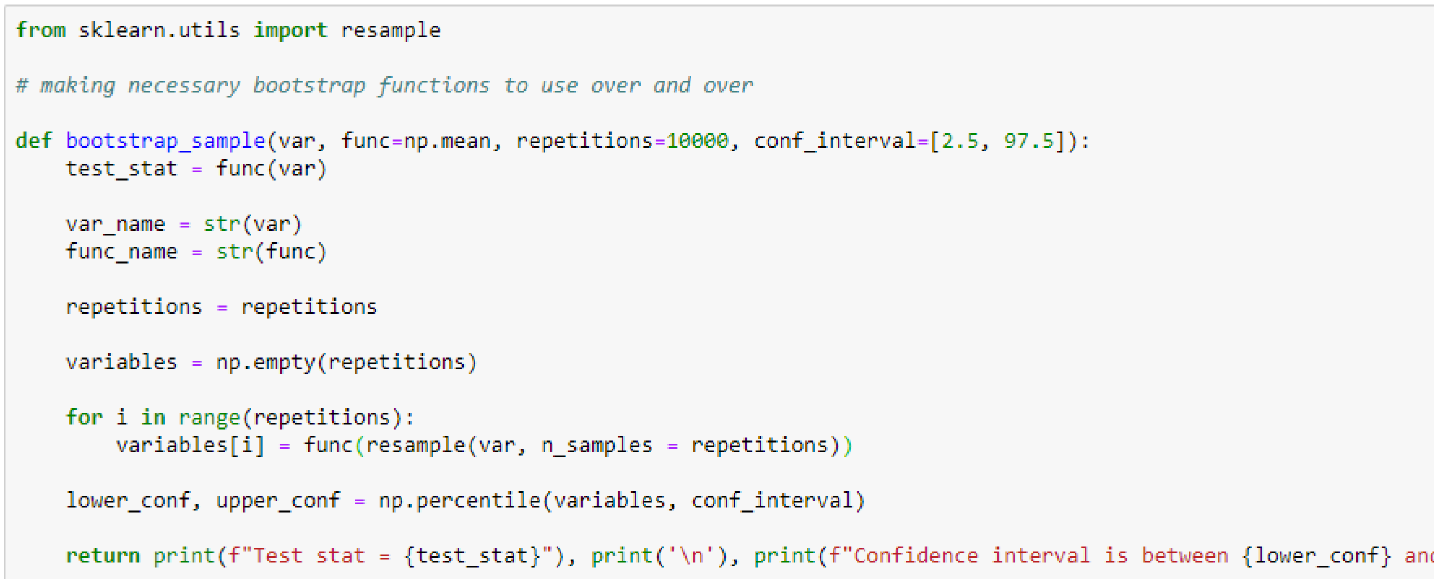
There is a long tail of bikes that make well over $25 total from rides in these data.

Statistics

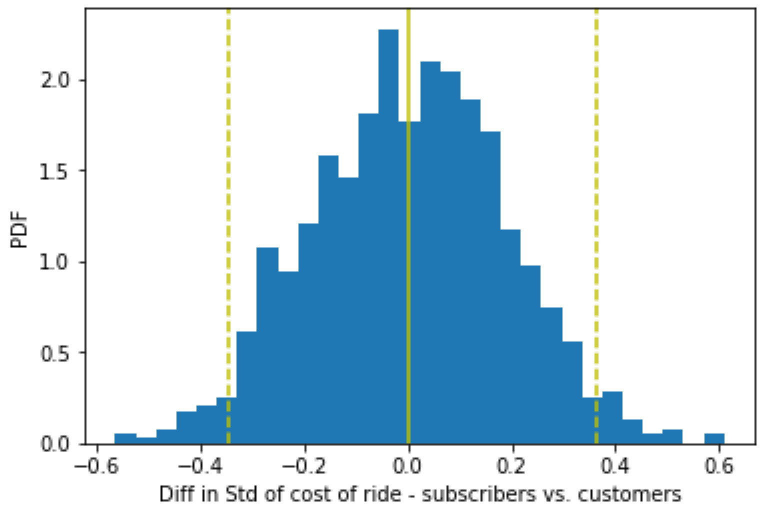
I performed bootstrap resamples of test statistics to see that I could be 95% confident they came from the population, and that was indeed the case as shown below for the average duration in minutes of rides:



I was also curious about whether or not there was a difference between customers and subscribers and how much they spent on a per-ride basis. The average difference in the standard deviations of their costs of rides did not differ significantly either. Yellow dashed lines indicate the cut-off points of the 95% confidence interval. The function that made this possible is shown in the code below and the plot resulting from calling it is below that:



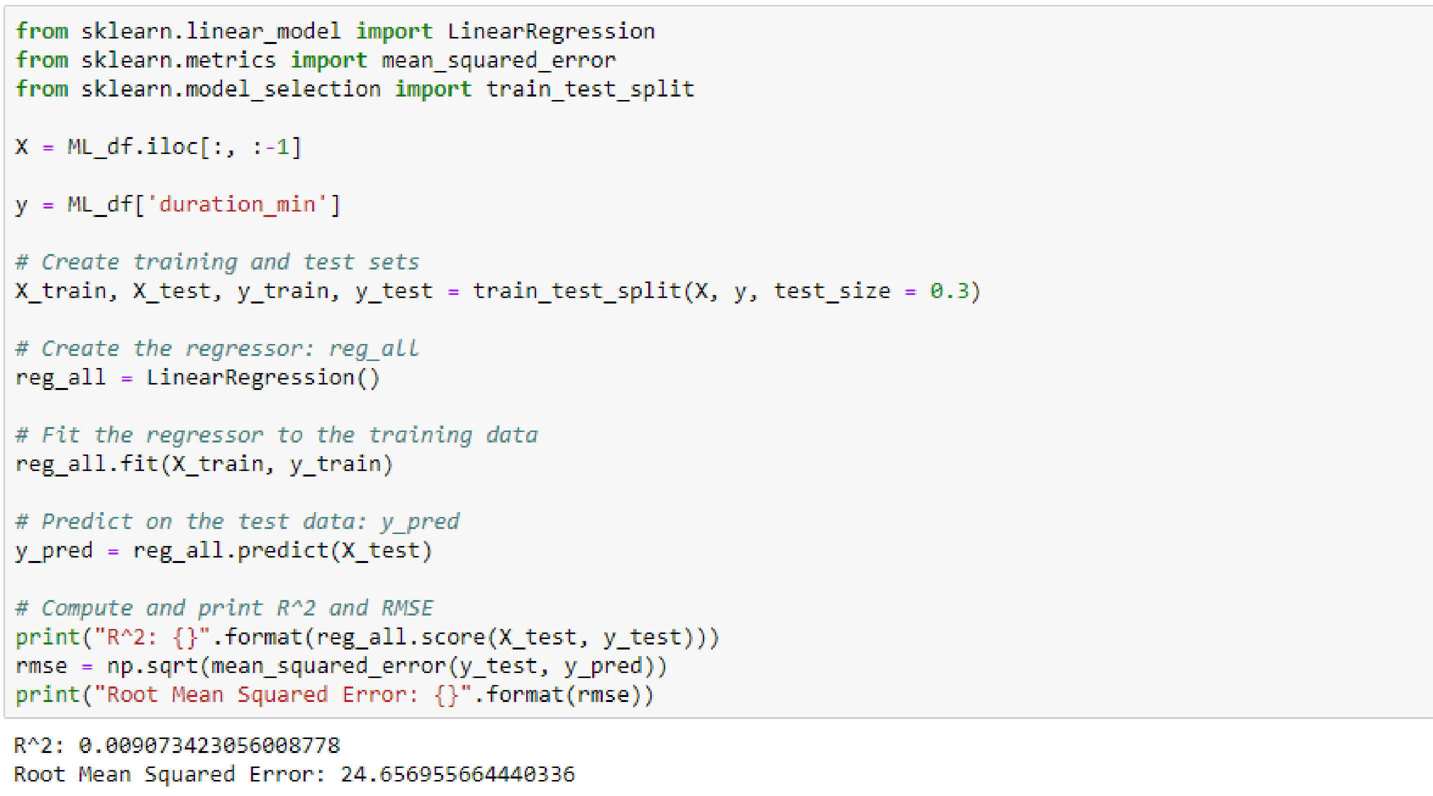
```
from sklearn.utils import resample

# making necessary bootstrap functions to use over and over

def bootstrap_sample(var, func=np.mean, repetitions=10000, conf_interval=[2.5, 97.5]):
    test_stat = func(var)

    var_name = str(var)
    func_name = str(func)

    repetitions = repetitions

    variables = np.empty(repetitions)

    for i in range(repetitions):
        variables[i] = func(resample(var, n_samples = repetitions))

    lower_conf, upper_conf = np.percentile(variables, conf_interval)

    return print(f"Test stat = {test_stat}"), print('\n'), print(f"Confidence interval is between {lower_conf} an
```



Machine Learning

Given the state of the data and the variables and nuances discovered in the cleaning and EDA steps, I can perform regression and classification modeling.

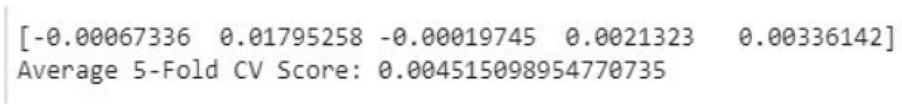
My focus was thus on doing linear regression to predict the duration of the ride based on three dependent variables: the user type, age, and gender. From these I wanted to find how long they rode. I ran a regular linear regression model from sklearn and found a strong performance with a low RMSE. The code and those scores are below:



```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

X = ML_df.iloc[:, :-1]

y = ML_df['duration_min']

# Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

# Create the regressor: reg_all
reg_all = LinearRegression()

# Fit the regressor to the training data
reg_all.fit(X_train, y_train)

# Predict on the test data: y_pred
y_pred = reg_all.predict(X_test)

# Compute and print R^2 and RMSE
print("R^2: {}".format(reg_all.score(X_test, y_test)))
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error: {}".format(rmse))
```

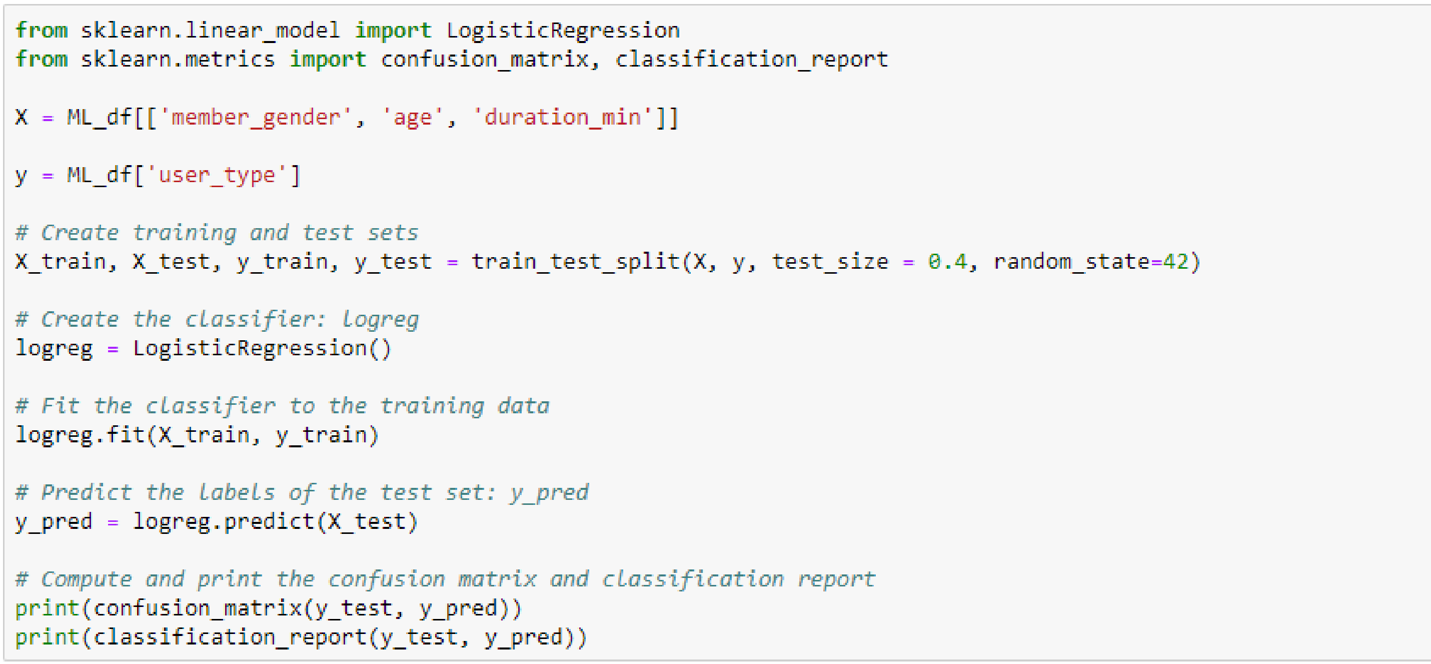
```
R^2: 0.009073423056008778
Root Mean Squared Error: 24.656955664440336
```

I also tried out 5-Fold Cross validation on the linear regression model and X and y variables and got the following result:



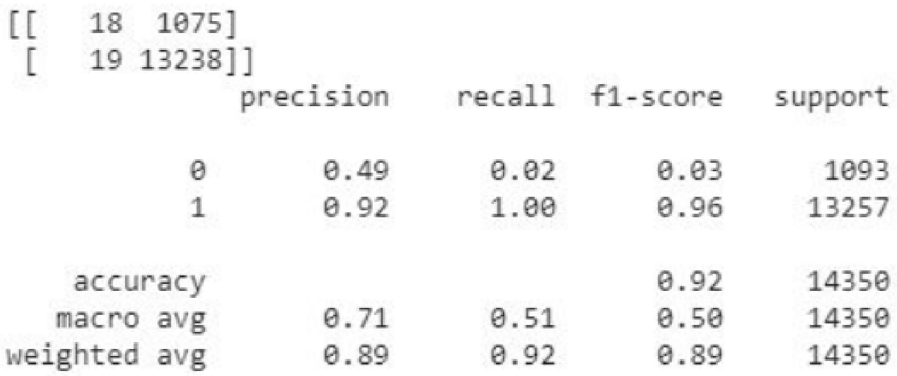
```
[-0.00067336  0.01795258 -0.00019745  0.0021323   0.00336142]
Average 5-Fold CV Score: 0.004515098954770735
```

I also wanted to do another algorithm but after EDA and all that I realized there isn't much I can do with these data in the end. As such, I used a standard logistic regression model to see if I might predict what the user type is (either customer or subscriber) based on duration of the ride, gender, and age. It turns out I can quite well. The code for the logistic regression model is below:



```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report

X = ML_df[['member_gender', 'age', 'duration_min']]

y = ML_df['user_type']

# Create training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4, random_state=42)

# Create the classifier: logreg
logreg = LogisticRegression()

# Fit the classifier to the training data
logreg.fit(X_train, y_train)

# Predict the labels of the test set: y_pred
y_pred = logreg.predict(X_test)

# Compute and print the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

The model was looking at the binary user type class, where 0 = ‘customer’ (rode at least once on the fly) and 1 = ‘subscriber’ (rides and pays monthly). The results of the confusion matrix and classification report show the good performance that was achieved for true positive classifications.



```
[[   18  1075]
 [   19 13238]]
              precision    recall  f1-score   support

           0       0.49      0.02      0.03      1093
           1       0.92      1.00      0.96     13257

    accuracy                           0.92     14350
   macro avg       0.71      0.51      0.50     14350
weighted avg       0.89      0.92      0.89     14350
```

This means that the model correctly predicting 92% of the time that the user type was subscriber when it was in fact subscriber. It’s worth noting here however that the data are skewed quite significantly in favor of the Subscriber user type. As such, in the future I would need to figure out a way to undersample that class, or resample and oversample the underrepresented class.

With these results in mind and in light of the EDA I've done throughout, I think it's safe to say that these data really do tell something about the kinds of rides that happen and what the riders are like. It looks like they can be used to figure out future examples of ride durations or user types based on the selected X variables above.

Conclusion & Future Directions

The Lyft Baywheels dataset has important implications for understanding how people get around in San Francisco on the ground, outside of ride-shares like Uber and Lyft cars and public transit. Bike-share programs are one possible future of urban mobility, and many companies are vying for market share, with Lyft and Uber being the main players.

Some future steps that could be taken are to redo the distribution of the predictor variables in the classification problem, since there is oversampling there. Another thing that would be cool would be to have the data piped into a data visualization front-end web app in real time to see where the bikes are and how they move throughout the day. This would be a great way to visualize the flow of users moment by moment.