CIS 4130- Big Data Technologies

Naziha Malik

naziha.malik@baruchmail.cuny.edu

Milestone 1- Introduction/ Proposal

For the semester-long project, I am proposing to use a set that describes the Uber and Lyft data in NYC from the years 2019- 2022. This data set includes very granular information about each ride, such as the TLC license plate, the date, the pick-up and drop off-times, total miles, total time of trip, and much more. With this data, I anticipate on determining and analyzing the aggregate data to learn more about the current trend of ride share services in New York City. While there are many routes to take when predicting future decisions with this data, I will mainly be focusing on predicting the tip amount based on the other factors that are involved with this table, such as the time, location, and distance of the trip.

Due to the fact that this data set contains so much information, even down to whether or not the passenger needed a wheelchair assisted vehicle or if the passenger agreed to having a shared car with other passengers, I anticipate learning more about the average behaviors of NYC TLC drivers and passengers. This data could also pair well with information regarding traffic patterns or vehicle collisions, or even any data that pertains to customer complaints on these rideshare apps.

The data is separated by month and year, and is stored as a parquet file, which can be used with pandas. The link to the data can be found below:

Uber/ Lyft (19 GB of data)-

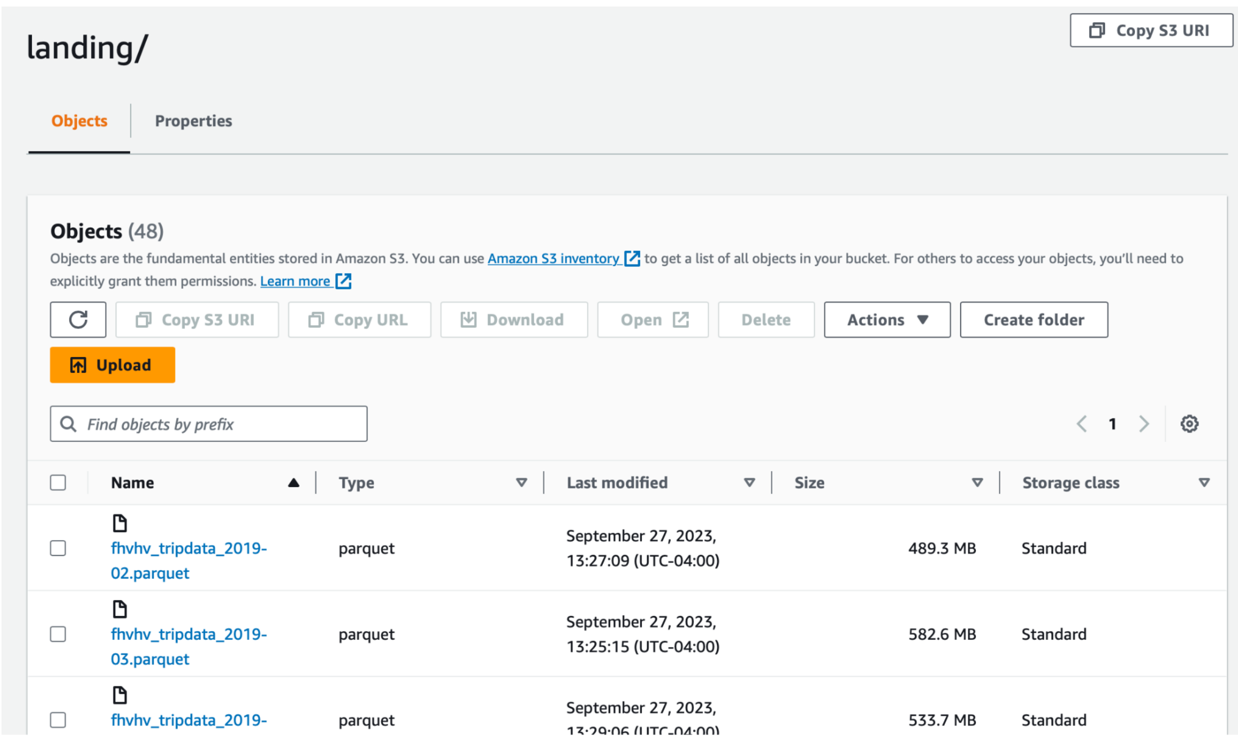
https://www.kaggle.com/datasets/jeffsinsel/nyc-fhvhv-data?select=fhvhv_tripdata_2020-06.parquet

https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Milestone 2- Data Acquisition

Since I planned to download my data set from Kaggle, I installed the Kaggle CLI and input my username and key. The steps I used to download the files can be found in Appendix A. I repeated the code to download each of the individual files onto my EC2 instance and store the file onto my S3 “landing” bucket, replacing the bolded text with the appropriate file name. I also had to use curl to retrieve some of the more recent files directly from the NYC data repository.

My Amazon S3 landing bucket can be seen below:



Milestone 3- Exploratory Data Analysis

The code for milestone 3 can be found in Appendix B, and the outputs of the code can be seen below:

```
#info on the fhv_df1 file
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18479031 entries, 0 to 18479030
Data columns (total 24 columns):
 #   Column                Dtype
---  ------                -----
 0   hvfhs_license_num     object
 1   dispatching_base_num  object
 2   originating_base_num  object
 3   request_datetime      datetime64[ns]
 4   on_scene_datetime     datetime64[ns]
 5   pickup_datetime       datetime64[ns]
 6   dropoff_datetime      datetime64[ns]
 7   PULocationID          int64
 8   DOLocationID          int64
 9   trip_miles            float64
 10  trip_time             int64
 11  base_passenger_fare   float64
 12  tolls                 float64
 13  bcf                   float64
 14  sales_tax             float64
 15  congestion_surcharge  float64
 16  airport_fee           float64
 17  tips                  float64
```

```
 18  driver_pay            float64
 19  shared_request_flag   object
 20  shared_match_flag     object
 21  access_a_ride_flag    object
 22  wav_request_flag      object
 23  wav_match_flag        object
dtypes: datetime64[ns](4), float64(9), int64(3), object(8)
memory usage: 3.3+ GB
```

 # describe the fhv_df1 file

	request_datetime	on_scene_datetime	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	trip_miles	trip_time	base_passenger_fare	tolls	bcf
count	18479031	13587039	18479031	18479031	1.847903e+07	1.847903e+07	1.847903e+07	1.847903e+07	1.847903e+07	1.847903e+07	1.847903e+07
mean	2023-01-17 02:32:09.458061824	2023-01-17 06:15:32.572615936	2023-01-17 02:36:36.490860032	2023-01-17 02:54:51.087476736	1.393105e+02	1.426835e+02	4.870138e+00	1.094634e+03	2.155565e+01	1.034738e+00	6.819760e-01
min	2022-12-31 20:30:00	2022-12-31 21:23:03	2023-01-01 00:00:00	2023-01-01 00:02:27	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	-1.463400e+02	0.000000e+00	0.000000e+00
25%	2023-01-09 15:34:10	2023-01-09 19:35:55	2023-01-09 15:37:59	2023-01-09 15:58:02	7.500000e+01	7.600000e+01	1.550000e+00	5.750000e+02	1.043000e+01	0.000000e+00	3.100000e-01
50%	2023-01-17 09:35:41	2023-01-17 15:57:34	2023-01-17 09:39:34	2023-01-17 09:59:16	1.400000e+02	1.420000e+02	2.897000e+00	9.060000e+02	1.626000e+01	0.000000e+00	4.900000e-01
75%	2023-01-24 17:46:14	2023-01-24 20:15:07	2023-01-24 17:50:30	2023-01-24 18:10:02	2.110000e+02	2.200000e+02	6.015000e+00	1.404000e+03	2.617000e+01	0.000000e+00	8.100000e-01
max	2023-02-01 00:15:00	2023-01-31 23:59:53	2023-01-31 23:59:59	2023-02-01 01:47:23	2.650000e+02	2.650000e+02	4.075630e+02	3.535900e+04	1.455120e+03	1.843700e+02	6.471000e+01
std	NaN	NaN	NaN	NaN	7.510532e+01	7.803905e+01	5.655499e+00	7.453304e+02	1.801144e+01	3.757672e+00	6.096692e-01

 # count of each variable in the fhv_df1 file

```
Out[4]: hvfhs_license_num       18479031
dispatching_base_num    18479031
originating_base_num    13587039
request_datetime        18479031
on_scene_datetime       13587039
pickup_datetime         18479031
dropoff_datetime        18479031
PULocationID            18479031
DOLocationID            18479031
trip_miles              18479031
trip_time               18479031
base_passenger_fare     18479031
```

```
tolls                   18479031
bcf                     18479031
sales_tax               18479031
congestion_surcharge    18479031
airport_fee             18479031
tips                    18479031
driver_pay              18479031
shared_request_flag     18479031
shared_match_flag       18479031
access_a_ride_flag      18479031
wav_request_flag        18479031
wav_match_flag          18479031
dtype: int64
```

 # list the variable names of the fhv_df1 file

```
['hvfhs_license_num', 'dispatching_base_num', 'originating_base_num',
'request_datetime', 'on_scene_datetime', 'pickup_datetime', 'dropoff_datetime',
'PULocationID', 'DOLocationID', 'trip_miles', 'trip_time',
'base_passenger_fare', 'tolls', 'bcf', 'sales_tax', 'congestion_surcharge',
'airport_fee', 'tips', 'driver_pay', 'shared_request_flag',
'shared_match_flag', 'access_a_ride_flag', 'wav_request_flag',
'wav_match_flag']
```

 # count the number of rows with null values in the fhv_df2 file

```
>> Rows with null values: 4891992
```

______

#info on the fhv_df2 file

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20159102 entries, 0 to 20159101
Data columns (total 24 columns):
 #   Column                Dtype
```

```
---  ------                 -----
 0   hvfhs_license_num      object
 1   dispatching_base_num   object
 2   originating_base_num   object
 3   request_datetime       datetime64[ns]
 4   on_scene_datetime      datetime64[ns]
 5   pickup_datetime        datetime64[ns]
 6   dropoff_datetime       datetime64[ns]
 7   PULocationID           int64
 8   DOLocationID           int64
 9   trip_miles             float64
 10  trip_time              int64
 11  base_passenger_fare    float64
 12  tolls                  float64
 13  bcf                    float64
 14  sales_tax              float64
 15  congestion_surcharge   float64
 16  airport_fee            object
 17  tips                   float64
 18  driver_pay             float64
 19  shared_request_flag    object
 20  shared_match_flag      object
 21  access_a_ride_flag     object
 22  wav_request_flag       object
 23  wav_match_flag         object
dtypes: datetime64[ns](4), float64(8), int64(3), object(9)
memory usage: 3.6+ GB
None
```

```
# describe the fhv_df2 file
```

	request_datetime	on_scene_datetime	pickup_datetime	dropoff_datetime	PULocationID	DOLocationID	trip_miles	trip_time	base_passenger_fare	tolls	bcf
count	20050204	13505053	20159102	20159102	2.015910e+07	2.015910e+07	2.015910e+07	2.015910e+07	2.015910e+07	2.015910e+07	2.015910e+07
mean	2019-02-15 02:09:02.991858432	2019-02-15 01:57:07.700399104	2019-02-15 00:36:08.897504256	2019-02-15 00:54:53.959078656	1.393497e+02	1.418818e+02	4.660525e+00	1.117965e+03	1.570783e+01	7.686299e-01	3.988125e-01
min	2019-01-31 23:19:44	2019-01-31 23:48:32	2019-02-01 00:00:00	2019-02-01 00:02:09	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	-1.632800e+02	0.000000e+00	0.000000e+00
25%	2019-02-08 08:28:40	2019-02-08 02:15:14	2019-02-08 06:07:44	2019-02-08 06:27:09	7.500000e+01	7.600000e+01	1.560000e+00	5.790000e+02	6.690000e+00	0.000000e+00	1.700000e-01
50%	2019-02-15 01:19:22.500000	2019-02-15 00:11:01	2019-02-14 23:29:07	2019-02-14 23:47:53	1.410000e+02	1.420000e+02	2.880000e+00	9.280000e+02	1.074000e+01	0.000000e+00	2.600000e-01
75%	2019-02-22 10:34:11	2019-02-22 15:12:38	2019-02-22 09:45:07	2019-02-22 10:04:03.750000128	2.110000e+02	2.190000e+02	5.670000e+00	1.453000e+03	1.898000e+01	0.000000e+00	4.800000e-01
max	2019-02-28 23:58:52	2019-02-28 23:59:50	2019-02-28 23:59:59	2019-03-01 06:02:46	2.650000e+02	2.650000e+02	4.692600e+02	8.384700e+04	1.097290e+03	1.710800e+02	2.793000e+01
std	NaN	NaN	NaN	NaN	7.521032e+01	7.743277e+01	5.415590e+00	7.730596e+02	1.612579e+01	3.185317e+00	4.517552e-01

```
# count of each variable in the fhv_df2 file
```

```
Out[5]: hvfhs_license_num       20159102

dispatching_base_num    20158697

originating_base_num    14483914

request_datetime        20050204

on_scene_datetime       13505053

pickup_datetime         20159102

dropoff_datetime        20159102

PULocationID            20159102

DOLocationID            20159102

trip_miles              20159102

trip_time               20159102

base_passenger_fare     20159102

tolls                   20159102

bcf                     20159102

sales_tax               20159102

congestion_surcharge    19646061

airport_fee                    0

tips                    20159102

driver_pay              20159102

shared_request_flag     20159102

shared_match_flag       20159102
```

```
access_a_ride_flag      20159102
wav_request_flag        20159102
wav_match_flag                 0
dtype: int64
```

 # list the variable names of the fhv_df2 file

```
['hvfhs_license_num', 'dispatching_base_num', 'originating_base_num',
'request_datetime', 'on_scene_datetime', 'pickup_datetime', 'dropoff_datetime',
'PULocationID', 'DOLocationID', 'trip_miles', 'trip_time',
'base_passenger_fare', 'tolls', 'bcf', 'sales_tax', 'congestion_surcharge',
'airport_fee', 'tips', 'driver_pay', 'shared_request_flag',
'shared_match_flag', 'access_a_ride_flag', 'wav_request_flag',
'wav_match_flag']
```

 # count the number of rows with null values in the fhv_df2 file

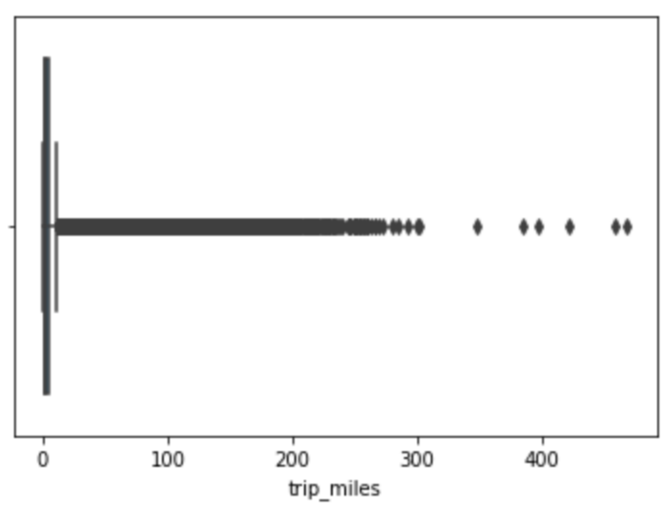
```
Rows with null values: 20159102
```

 # convert the dtype of the “trip_miles” variable to an integer, and ensure that this change was properly made

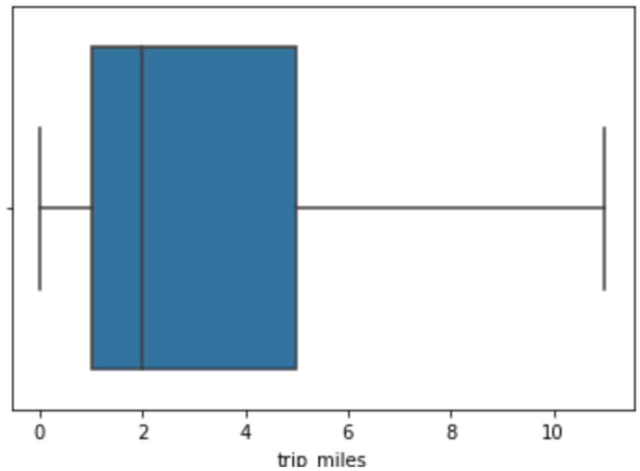
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20159102 entries, 0 to 20159101
Data columns (total 24 columns):
 #   Column                Dtype
---  ------                -----
 0   hvfhs_license_num     object
 1   dispatching_base_num  object
 2   originating_base_num  object
 3   request_datetime      datetime64[ns]
 4   on_scene_datetime     datetime64[ns]
 5   pickup_datetime       datetime64[ns]
 6   dropoff_datetime      datetime64[ns]
 7   PULocationID          int64
```

```
 8   DOLocationID          int64
 9   trip_miles            int64
 10  trip_time             int64
 11  base_passenger_fare   float64
 12  tolls                 float64
 13  bcf                   float64
 14  sales_tax             float64
 15  congestion_surcharge  float64
 16  airport_fee           object
 17  tips                  float64
 18  driver_pay            float64
 19  shared_request_flag   object
 20  shared_match_flag     object
 21  access_a_ride_flag    object
 22  wav_request_flag      object
 23  wav_match_flag        object
dtypes: datetime64[ns](4), float64(7), int64(4), object(9)
memory usage: 3.6+ GB
```

 # create a boxplot showing the trip_miles for the fhv_df2 file



 # create the same boxplot as above but remove the outliers for fhv_df2



From looking at the data, the sheer number of rows shows how many Uber and Lyft rides NYC uses in a given month- just looking at the February 2019 data and January 2023 data, the number of rides range from 18 million to 20 million. With this in mind, it did not surprise me to see how much variability can be in a single column. I chose to look at the “trip_miles” column, which showcases the length of the Uber/ Lyft ride in miles. Only using a boxplot on the February 2019 data, it is hard to even see the data due to the amount of outliers that don’t fit the normal distribution of the column. When taking out the outliers, we can see that the normal distribution of trip miles ranges from 0-11 miles, which is miles away from the maximum of 469 miles. Due to the variability of the data, I’m expecting to deal with difficulties regarding visualizing the data in a way that it is easy for others to decipher. In addition, the amount of null values were concerning; the February 2019 data set had roughly 20 million rows with null values while the January 2023 data set had about more than 4 million rows with null values. The null values may be explained by the fact that much of the numerical data has been given the data type as float, so hopefully by converting the appropriate columns to integers would help to decrease the null values.

Milestone 4- Feature Engineering and Modeling

Cleaning Data → Code in Appendix C

- The fhv data was originally in a parquet file with predefined schema, so my main focus was on removing the null values. After looking at the data, I realized that much of the null values came from the timestamp columns so I opted to drop the null from the request_datetime, on_scene_datetime, pickup_datetime, and dropoff_datetime columns. I also dropped the trip_mile outliers that I had discovered through my EDA in the previous milestone, meaning I dropped the records where the trip_miles exceeded 15 miles. Lastly, I converted trip_time to minutes, as the data dictionary had listed the time as seconds. The schema and head of my dataframe can be seen below:

```
dec21:  pyspark.sql.dataframe.DataFrame
  hvfhs_license_num: string
  dispatching_base_num: string
  originating_base_num: string
  request_datetime: timestamp
  on_scene_datetime: timestamp
  pickup_datetime: timestamp
  dropoff_datetime: timestamp
  PULocationID: long
  DOLocationID: long
  trip_miles: double
  trip_time: double
  base_passenger_fare: double
  tolls: double
  bcf: double
  sales_tax: double
  congestion_surcharge: double
  airport_fee: double
  tips: double
  driver_pay: double
  shared_request_flag: string
  shared_match_flag: string
  access_a_ride_flag: string
```

```
|hvfhs_license_num|dispatching_base_num|originating_base_num|   request_datetime|  on_scene_datetime|    pickup_datetime|   dropoff_datetime|PULocationID|
DOLocationID|trip_miles|         trip_time|base_passenger_fare|tolls| bcf|sales_tax|congestion_surcharge|airport_fee|tips|driver_pay|shared_request_flag|s
hared_match_flag|access_a_ride_flag|wav_request_flag|wav_match_flag|
+-----------------+--------------------+--------------------+-------------------+-------------------+-------------------+-------------------+------------+
------------+----------+------------------+-------------------+-----+----+---------+--------------------+-----------+----+----------+-------------------+-
----------------+------------------+----------------+--------------+
|           HV0003|              B03404|              B03404|2021-12-01 00:02:58|2021-12-01 00:05:38|2021-12-01 00:06:05|2021-12-01 00:19:05|          80|
112|      2.39|              13.0|              13.27|  0.0| 0.4|     1.18|                 0.0|        0.0| 0.0|      9.41|                  N|
N|                 |               N|             N|
|           HV0003|              B03404|              B03404|2021-12-01 00:20:22|2021-12-01 00:21:26|2021-12-01 00:22:45|2021-12-01 00:43:47|         112|
189|      4.91|21.033333333333335|              22.59|  0.0|0.68|      2.0|                 0.0|        0.0|5.05|     16.23|                  N|
N|                 |               N|             N|
|           HV0003|              B03404|              B03404|2021-12-01 00:47:40|2021-12-01 00:48:50|2021-12-01 00:50:51|2021-12-01 00:59:20|          49|
225|      1.59| 8.483333333333333|               9.58|  0.0|0.29|     0.85|                 0.0|        0.0| 0.0|      6.26|                  N|
N|                 |               N|             N|
|           HV0003|              B03404|              B03404|2021-12-01 00:25:31|2021-12-01 00:29:07|2021-12-01 00:29:12|2021-12-01 00:37:22|         239|
263|      1.78| 8.166666666666666|               9.15|  0.0|0.21|     0.63|                2.75|        0.0| 7.0|      6.07|                  N|
N|                 |               N|             N|
```

Feature Engineering → Code in Appendix D

- The main goal of feature engineering was to make the data as digestible as possible before encoding it into a vector for my regression model, so there was much tweaking to be done. The specific changes I made are listed below:
    - Converting columns that had less than 5 standard values into flags (tolls, congestion_charge, etc.)
    - Converting the taxi codes to the respective taxi companies
    - Converting the taxi zones to their respective boroughs
    - Drilling down request_datetime into it's day of month, day type (weekend or weekday), month, year, and hour
    - Deriving how long the passenger waited for their taxi ride
    - Drop unnecessary columns
- The schema and head of the dataframe after feature engineering can be seen below:

```
jan23:  pyspark.sql.dataframe.DataFrame
  trip_miles: double
  trip_time: double
  base_passenger_fare: double
  tips: double
  driver_pay: double
  shared_match_flag: string
  congestion_surcharge_flag: string
  tolls_flag: string
  airport_fee_flag: string
  hvfhs_company: string
  PULocationBorough: string
  DOLocationBorough: string
  day_of_month: integer
  day_type: string
  month: integer
  year: integer
  request_hour: integer
  wait_minutes: double
```

trip_miles	trip_time	base_passenger_fare	tips	driver_pay	shared_match_flag	congestion_surcharge_flag	tolls_flag	airport_fee_flag	hvfhs_company	PULocationBorough	DOLocationBorough	day_of_month	day_type	month	year	request_hour	wait_minutes
0.94	28.483333333333334	25.95	5.22	27.83	N	Y	N	N	Uber	Manhattan	Manhattan	1	Weekend	1	2023	0	1.3
2.78	34.483333333333334	60.14	0.0	50.15	N	Y	N	N	Uber	Manhattan	Manhattan	1	Weekend	1	2023	0	7.633333333333334
8.81	17.45	24.37	0.0	20.22	N	N	N	N	Uber	Queens	Queens	1	Weekend	1	2023	0	4.65
0.67	7.183333333333334	13.8	0.0	7.9	N	N	N	N	Uber	Queens	Queens	1	Weekend	1	2023	0	4.1
4.38	12.066666666666666	20.49	0.0	16.48	N	N	N	N	Uber	Queens	Queens	1	Weekend	1	2023	0	7.916666666666667

Modeling → Code in Appendix E

- To model my data, I combined all of the 2020 parquet files, as this was the year with the least FHV rides. Using an Indexer, Encoder, and Vector Assembler, I was able to create a pipeline that would predict whether or not the taxi driver had received a "good" tip, which I classified as 20% of the base_passenger_fare. After creating my model, I was able to test it on my testData (a random 30%) and validate this model across 3 folds.
- The AUC for a base tip of 20% or more was 0.692. This shows that the model was getting slightly better at predicting whether the driver would get a "good tip" as the threshold got higher, but the increase in AUC isn't too substantial.
- My confusion matrix for a base tip of 20% can be seen below:

    ```
    +-----+--------+----+
    |label|     0.0| 1.0|
    +-----+--------+----+
    |  0.0|17601491| 306|
    |  1.0|  832613|2301|
    +-----+--------+----+
    accuracy:  0.94732520217838
    precision:  0.9090909090909091
    recall:  0.00010316191262186001
    f1 score:  0.0002063004146638335
    (0.94732520217838, 0.9090909090909091, 0.00010316191262186001, 0.0002063004146638335)
    ```

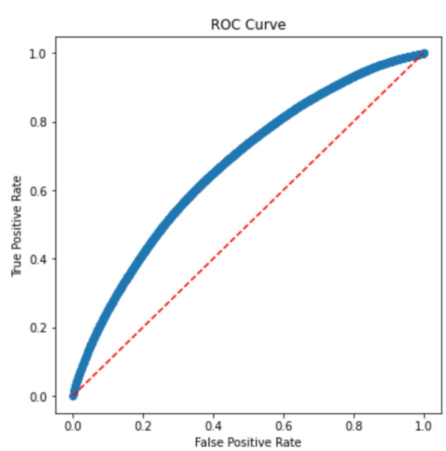
    -
    - The model was much better at determining true negatives (that someone left a "poor" tip) compared to true positives (leaving a "good" tip) for all tip thresholds. The F1 score is also very low across the thresholds, which is likely attributed to the low recall of the model.

Milestone 5- Data Visualizing

In an effort to visualize the accuracy of the models that I had created, I was able to develop ROC curves, which were able to tell me that the best model had an elasticNetParam of 0.0 and a regParam of 0.0 despite the different thresholds for tips. Through the ROC curves, we are able to see the tradeoff between accuracy and precision. The ROC curves are generally the same across the different thresholds. The code I used for data visualization can be found in Appendix F.

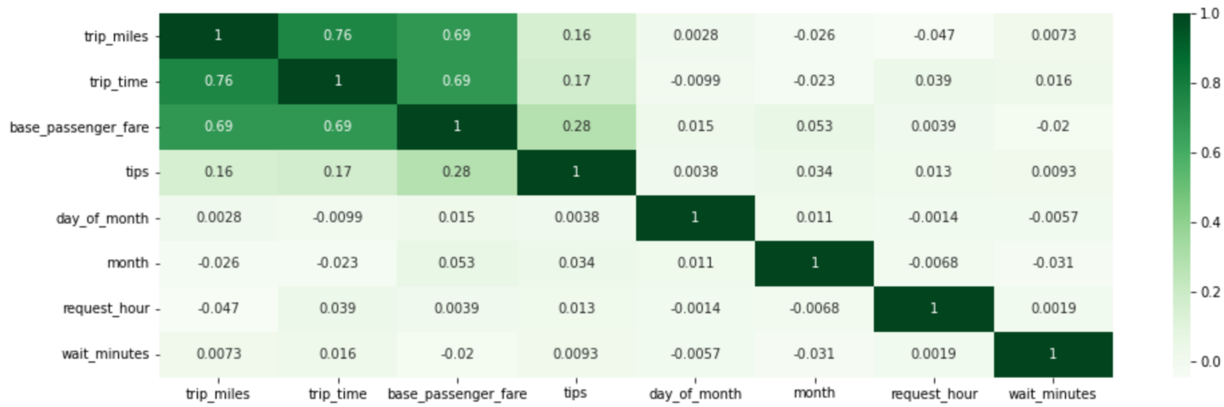
```
LogisticRegression_acc3ee1e7660__aggregationDepth 2
LogisticRegression_acc3ee1e7660__elasticNetParam 0.0
LogisticRegression_acc3ee1e7660__family auto
LogisticRegression_acc3ee1e7660__featuresCol features
LogisticRegression_acc3ee1e7660__fitIntercept True
LogisticRegression_acc3ee1e7660__labelCol label
LogisticRegression_acc3ee1e7660__maxBlockSizeInMB 0.0
LogisticRegression_acc3ee1e7660__maxIter 100
LogisticRegression_acc3ee1e7660__predictionCol prediction
LogisticRegression_acc3ee1e7660__probabilityCol probability
LogisticRegression_acc3ee1e7660__rawPredictionCol rawPrediction
LogisticRegression_acc3ee1e7660__regParam 0.0
LogisticRegression_acc3ee1e7660__standardization True
LogisticRegression_acc3ee1e7660__threshold 0.5
LogisticRegression_acc3ee1e7660__tol 1e-06
```

-
- The ROC curve for the 20% tip threshold can be seen below:

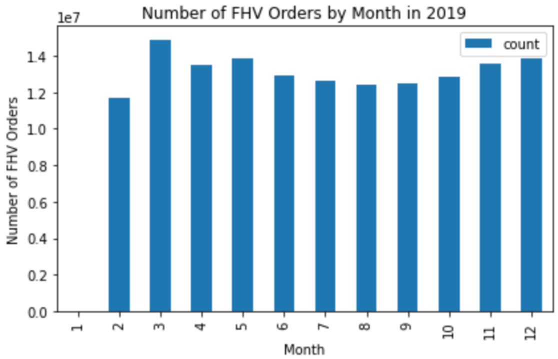


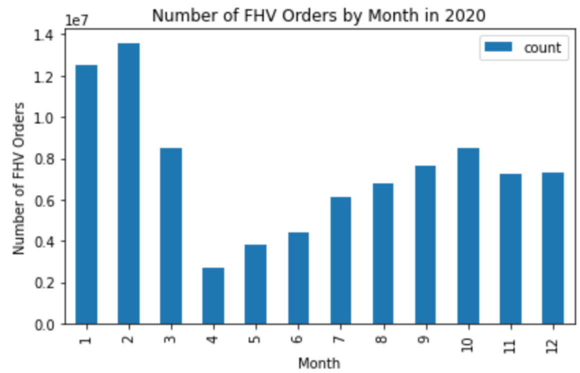
-

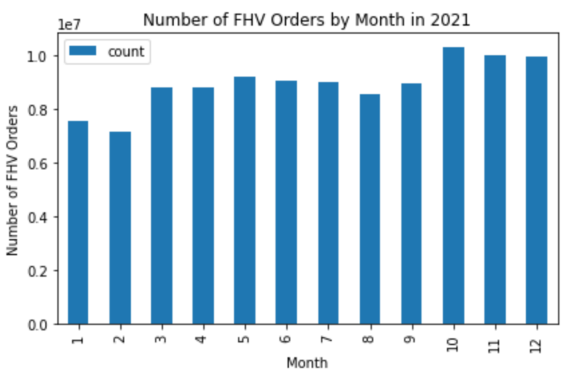
I had also developed a correlation matrix using all of the numeric values in the dataframe. From this correlation matrix, we are able to see that the day_of_month, month, request_hour, and wait_minutes have little to no correlation to the other variables. The factor that most affected "tips" would be the base_passenger_fare, though it is a very weak correlation. There are strong correlations between trip_miles, trip_time, and base_passenger_fare, which is to be expected as these variables are dependent on one another. The correlation matrix can be seen below:

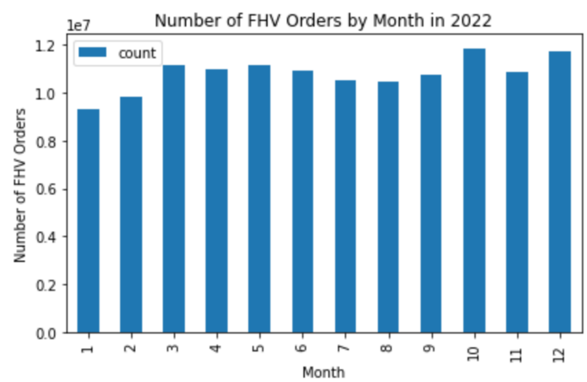


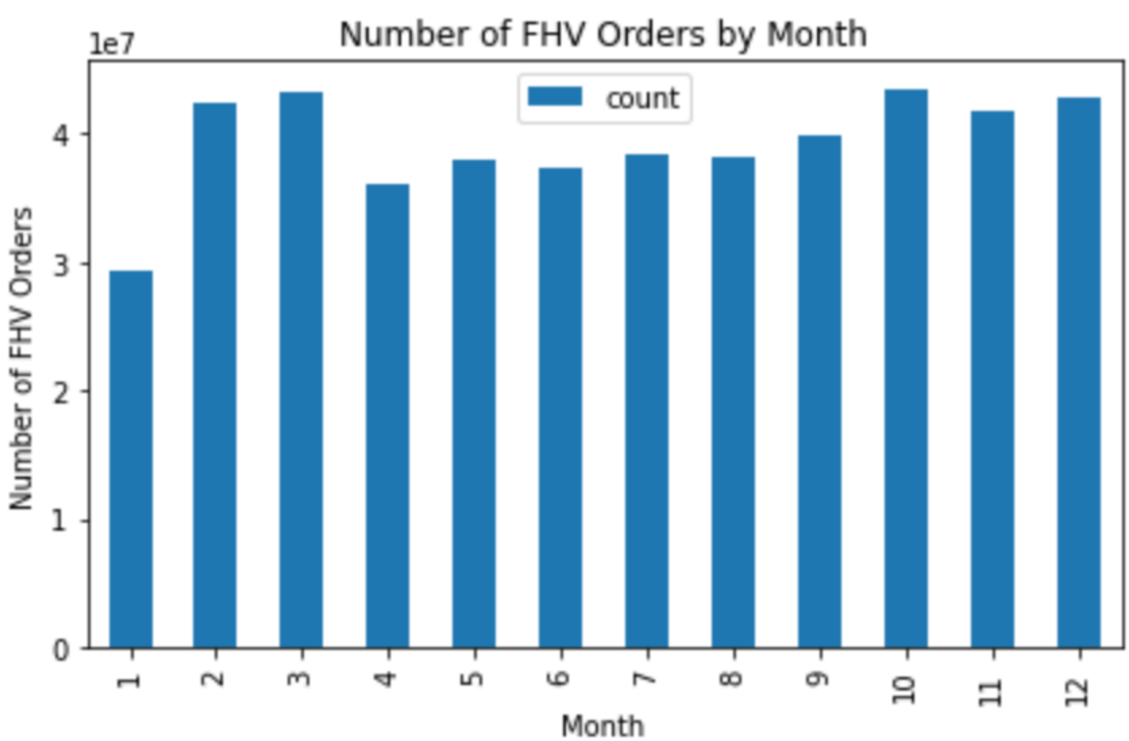
In addition, I was also able to visualize the number of rides that were requested by month, which I did per year as well as a cumulative total. We are able to see how FHV rides were greatly affected by the pandemic, as it was until October of 2021 that they were able to make pre-pandemic numbers again. By looking at the cumulative bar graph, we are able to see that February, March, October, and December would bring in the most rides from these FHV apps, whereas January and April bring in the lowest. The bar graphs can be seen below:


Number of FHV Orders by Month in 2019
1e7
count
Number of FHV Orders
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1 2 3 4 5 6 7 8 9 10 11 12
Month

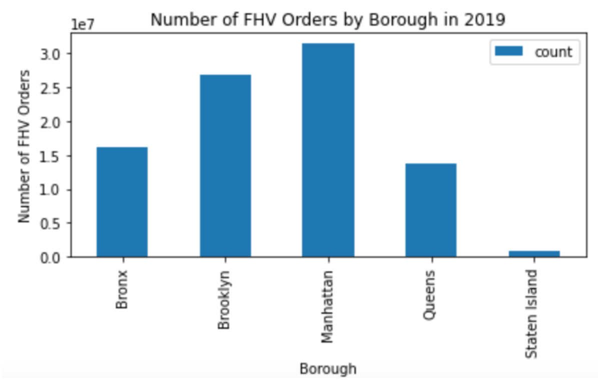

Number of FHV Orders by Month in 2020
1e7
count
Number of FHV Orders
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1 2 3 4 5 6 7 8 9 10 11 12
Month

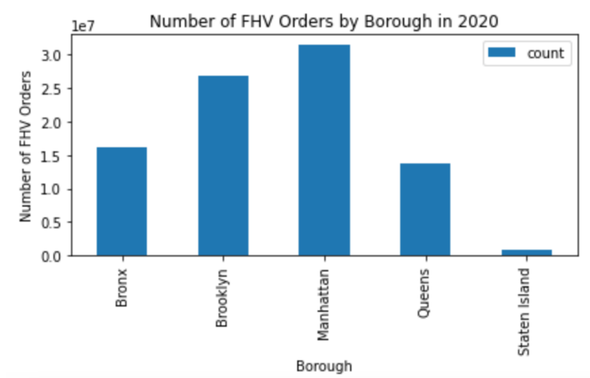

Number of FHV Orders by Month in 2021
1e7
count
Number of FHV Orders
0.0
0.2
0.4
0.6
0.8
1.0
1 2 3 4 5 6 7 8 9 10 11 12
Month

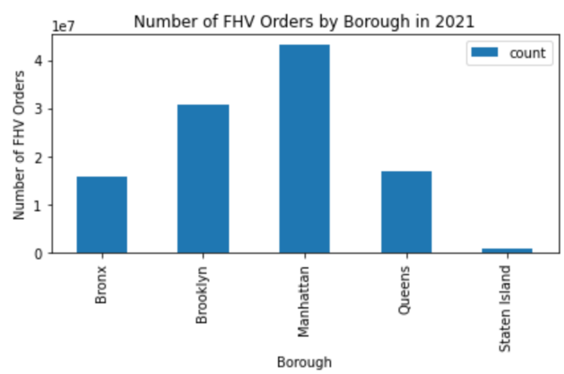

Number of FHV Orders by Month in 2022
1e7
count
Number of FHV Orders
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1 2 3 4 5 6 7 8 9 10 11 12
Month

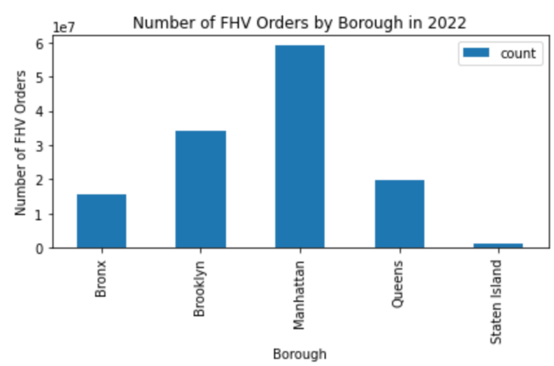

Number of FHV Orders by Month
1e7
count
Number of FHV Orders
0
1
2
3
4
1 2 3 4 5 6 7 8 9 10 11 12
Month

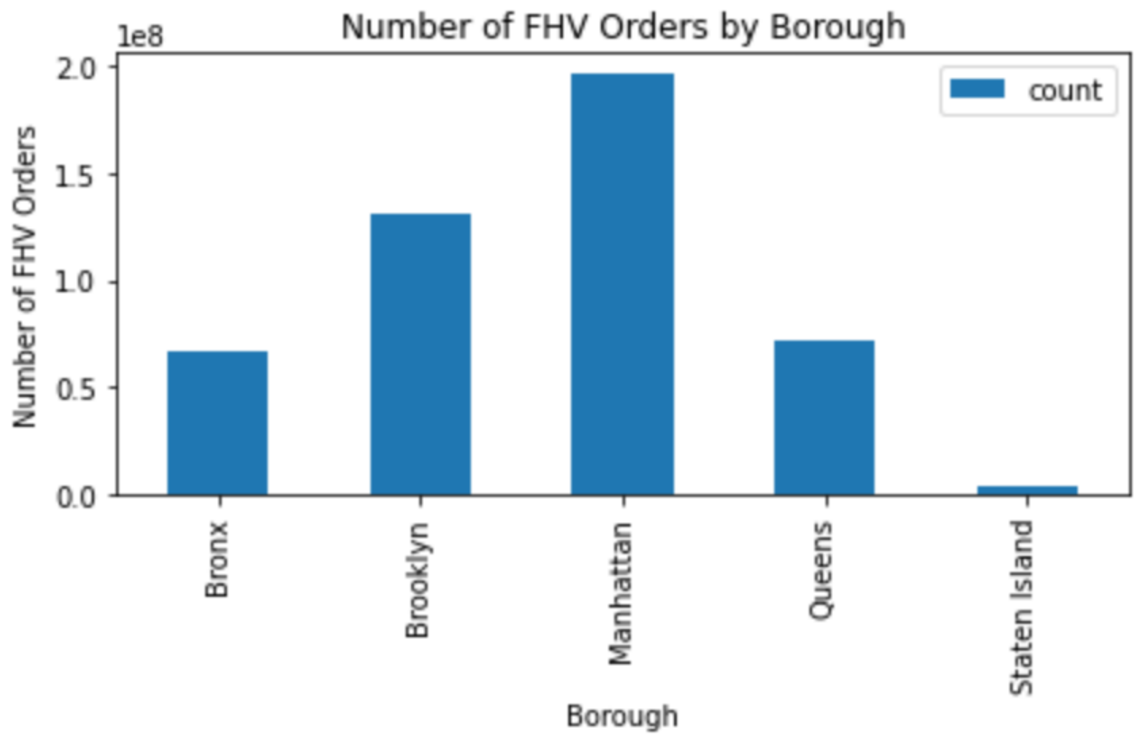
Lastly, I was able to develop bar graphs that showed the number of FHV rides per borough. Once again, I was able to show the bar graphs per year, as well as a cumulative view of the results. In all of the bar graphs, it is clear that Manhattan is the borough with the most FHV rides, which is to be expected due to the high population density and tourism that exists there. Brooklyn seems to be the next borough that has the highest number of FHV rides, followed by either the Bronx or Queens, tie for third place in the cumulative graph:










Number of FHV Orders by Borough
1e8
2.0
1.5
1.0
0.5
0.0
Number of FHV Orders
count
Bronx
Brooklyn
Manhattan
Queens
Staten Island
Borough

Milestone 6- Summary and Conclusions

This machine learning project has helped me to understand the concept of big data, as well as learn the foundations of sources such as AWS, spark, and Databricks. A summary of my machine learning process can be seen below:

- Utilized Amazon EC2 to download and zip the parquet files from Kaggle, and load these files onto my S3 bucket.
- Showcased descriptive statistics using Databricks.
- Cleaned up my data in order to remove outliers and produce more meaningful columns.
- Performed feature engineering on Databricks, utilizing StringIndexer, OneHotEncoder, and VectorAssembler.
- Tested my data using a 70-30 split.
- Created visualizations to showcase my model performance, as well as visualizations to show the distribution of my data.

After completing my machine learning pipeline, I was able to draw conclusions about my model, as well as about the state of FHV rides. My model did not have the best overall performance, despite my best efforts at cleaning the data and removing outliers. However, I think this is due to the sheer amount of people who hadn't tipped, as this could result in heavily skewed data. This can be backed up by the fact that "tips" had a weak correlation to "base_passenger_fare." In addition, it didn't seem like any of the numerical variables had a strong correlation to the "tips" variable, which could also explain the poor model performance. If I were to redo this project, I would likely opt for a linear regression model instead of a logistic regression model and see if it changes the model performance.

https://github.com/nazihamalik/fhv-tip-prediction/tree/main

Appendix A- Code Used for Data Acquisition

```
#look at the individual data files in the Kaggle data set

kaggle datasets files jeffsinsel/nyc-fhvhv-data

#download individual data files onto EC2

kaggle datasets download -d jeffsinsel/nyc-fhvhv-data -f
fhvhv_tripdata_2019-06.parquet

#unzip file:

unzip fhvhv_tripdata_2019-06.parquet

#copy file onto Amazon S3 Bucket named "landing":

aws s3 cp fhvhv_tripdata_2019-06.parquet
s3://my-project-nm/landing/fhvhv_tripdata_2019-06.parquet

#remove the downloaded file off of EC2 once confirming it was uploaded onto my
bucket:

rm fhvhv_tripdata_2019-06.parquet

rm fhvhv_tripdata_2019-06.parquet.zip


#download individual data files onto EC2

kaggle datasets download -d jeffsinsel/nyc-fhvhv-data -f
fhvhv_tripdata_2019-02.parquet

#unzip file:

unzip fhvhv_tripdata_2019-02.parquet

#copy file onto Amazon S3 Bucket named "landing":

aws s3 cp fhvhv_tripdata_2019-02.parquet
s3://my-project-nm/landing/fhvhv_tripdata_2019-02.parquet

#remove the downloaded file off of EC2 once confirming it was uploaded onto my
bucket:

rm fhvhv_tripdata_2019-02.parquet
```

rm **fhvhv_tripdata_2019-02.parquet.zip**

#download individual data files onto EC2

kaggle datasets download -d jeffsinsel/nyc-fhvhv-data -f **fhvhv_tripdata_2019-03.parquet**

#unzip file:

unzip **fhvhv_tripdata_2019-03.parquet**

#copy file onto Amazon S3 Bucket named "landing":

aws s3 cp **fhvhv_tripdata_2019-03.parquet** s3://my-project-nm/landing/**fhvhv_tripdata_2019-03.parquet**

#remove the downloaded file off of EC2 once confirming it was uploaded onto my bucket:

rm **fhvhv_tripdata_2019-03.parquet**

rm **fhvhv_tripdata_2019-03.parquet.zip**

#download the data set using curl, and with the pipe, I was able to upload it onto my S3 bucket:

curl -SL **https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv_tripdata_2022-01.parquet** | aws s3 cp - s3://my-project-nm/landing/**fhvhv_tripdata_2022-01.parquet**

#download the data set using curl, and with the pipe, I was able to upload it onto my S3 bucket:

curl -SL **https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv_tripdata_2020-12.parquet** | aws s3 cp - s3://my-project-nm/landing/**fhvhv_tripdata_2020-12.parquet**

Appendix B- Code Used for Exploratory Data Analysis

```
!pip install fsspec s3fs boto3
!pip install pyarrow fastparquet
!pip install seaborn

pip install --upgrade pandas

import pandas as pd
import pyarrow
import fastparquet
import seaborn as sns

fhv_df1 =
pd.read_parquet("https://my-project-nm.s3.us-east-2.amazonaws.com/landing/fhvhv
_tripdata_2023-01.parquet")

# column names and data types for each column
print(fhv_df1.info())

# summary statistics of each column
fhv_df1.describe()

# number of observations
fhv_df1.count()

# list of variable/ column names
variable_list = list(fhv_df1)
print(variable_list)

# null values?
# Which columns have nulls or NaN ?
fhv_df1.columns[fhv_df1.isnull().any()].tolist()
# How many rows have nulls?
print("Rows with null values:", fhv_df1.isnull().any(axis=1).sum())
```

```python
fhv_df2 =
pd.read_parquet("https://my-project-nm.s3.us-east-2.amazonaws.com/landing/fhvhv
_tripdata_2019-02.parquet")

# column names and data types for each column
print(fhv_df2.info())

# summary statistics of each column
fhv_df2.describe()

# number of observations
fhv_df2.count()

# list of variable/ column names
variable_list = list(fhv_df2)
print(variable_list)

# null values?
# Which columns have nulls or NaN ?
fhv_df2.columns[fhv_df2.isnull().any()].tolist()
# How many rows have nulls?
print("Rows with null values:", fhv_df2.isnull().any(axis=1).sum())

# convert variable trip_miles from float to integer
fhv_df2['trip_miles'] = fhv_df2['trip_miles'].fillna(0).astype(int)

# make sure that the changed dtype is stored
print(fhv_df2.info())

# make a boxplot showing the trip_miles
sns.boxplot(x=fhv_df2["trip_miles"])

# make a boxplot without outliers for trip_miles
sns.boxplot(x=fhv_df2["trip_miles"], showfliers= False)
```

Appendix C- Code Used for Cleaning and Normalizing the Data

```
# To work with Amazon S3 install boto3,  s3fs
%pip install "boto3>=1.28" "s3fs>=2023.3.0"
# If your files are in Parquet format, install pyarrow and fastparquet
%pip install pyarrow fastparquet
# For visualizations, install seaborn
%pip install seaborn

Spark

import os
# To work with Amazon S3 storage, set the following variables using your AWS
Access Key and Secret Key
# Set the Region to where your files are stored in S3.
access_key = '####################'
secret_key = '######################################'
# Set the environment variables so boto3 can pick them up later
os.environ['AWS_ACCESS_KEY_ID'] = access_key
os.environ['AWS_SECRET_ACCESS_KEY'] = secret_key encoded_secret_key =
secret_key.replace("/", "%2F")
aws_region = "us-east-2"

sc._jsc.hadoopConfiguration().set("fs.s3a.access.key", access_key)
sc._jsc.hadoopConfiguration().set("fs.s3a.secret.key", secret_key)
sc._jsc.hadoopConfiguration().set("fs.s3a.endpoint", "s3." + aws_region +
".amazonaws.com")

from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import col, lit, when, date_format, expr, to_date,
to_timestamp, date_format, split, hour, dayofweek, dayofmonth, month, year

### DATA CLEANING
# read the data from my landing folder
dec21 =
spark.read.parquet("s3://my-project-nm/landing/fhvhv_tripdata_2021-12.parquet")
```

```
# drop necessary null values
columns_to_drop_null = ["request_datetime", "on_scene_datetime",
"pickup_datetime", "dropoff_datetime" ]
dec21 = dec21.na.drop(subset=columns_to_drop_null)

# convert trip_time from seconds to minutes
dec21 = dec21.withColumn('trip_time', (F.col('trip_time') /60))

# drop records where trip_miles is an outlier based on previous EDA
trip_miles_condition = col("trip_miles") <= 15
dec21 = dec21.where(trip_miles_condition)

dec21.show(5)

# write the parquet to my S3 Raw Bucket
dec21.write.parquet("s3://my-project-nm/raw/cleaned_fhvhv_tripdata_2021-12.parq
uet")
```

Appendix D- Code Used for Feature Engineering

```
###FEATURE ENGINEERING
jan23 =
spark.read.parquet("s3://my-project-nm/raw/cleaned_fhvhv_tripdata_2023-01.parqu
et")

### convert congestion_surcharge to a flag
jan23 = jan23.withColumn("congestion_surcharge_flag",
when(jan23.congestion_surcharge >0, "Y")
.otherwise("N")
)

### convert tolls to a flag
jan23 = jan23.withColumn("tolls_flag",
when(jan23.tolls >0, "Y")
.otherwise("N")
)

### convert airport_fee to a flag
jan23 = jan23.withColumn("airport_fee_flag",
when(jan23.airport_fee>0, "Y")
.otherwise("N")
)

### convert hvfhs_license_num to the respected fhv company
jan23 = jan23.withColumn("hvfhs_company",
when(jan23.hvfhs_license_num== "HV0005", "Lyft")
.when(jan23.hvfhs_license_num== "HV0002", "Juno")
.when(jan23.hvfhs_license_num== "HV0003", "Uber")
.when(jan23.hvfhs_license_num == "HV0004", "Via")
.otherwise("N/A")
)

### convert PULocationID to the respected borough
jan23 = jan23.withColumn("PULocationBorough",
when(jan23.PULocationID.isin(Bronx), "Bronx")
.when(jan23.PULocationID.isin(Brooklyn), "Brooklyn")
```

```
.when(jan23.PULocationID.isin(Queens), "Queens")
.when(jan23.PULocationID.isin(Manhattan), "Manhattan")
.when(jan23.PULocationID.isin(Staten_Island), "Staten Island")
.otherwise("N/A")
)
# drop records if the PULocationBorough isn't registered
PULocationCondition = col("PULocationBorough") != "N/A"
jan23 = jan23.where(PULocationCondition)

### convert DOLocation ID to the respected borough
jan23 = jan23.withColumn("DOLocationBorough",
when(jan23.DOLocationID.isin(Bronx), "Bronx")
.when(jan23.DOLocationID.isin(Brooklyn), "Brooklyn")
.when(jan23.DOLocationID.isin(Queens), "Queens")
.when(jan23.DOLocationID.isin(Manhattan), "Manhattan")
.when(jan23.DOLocationID.isin(Staten_Island), "Staten Island")
.otherwise("N/A")
)
# drop records if the DOLocationBorough isn't registered
DOLocationCondition = col("DOLocationBorough") != "N/A"
jan23 = jan23.where(DOLocationCondition)

### separate request_datetime into date and time columns
jan23 = jan23.withColumn("request_date", to_date("request_datetime",
"yyyy-MM-dd"))
jan23 = jan23.withColumn("day_of_month", dayofmonth(col("request_date")))
jan23 = jan23.withColumn("day_of_week", dayofweek(col("request_date")))
def is_weekday(day):
  return "Weekday" if day in range(2, 7) else "Weekend"
spark.udf.register("is_weekday", is_weekday)
jan23 = jan23.withColumn("day_type",
col("day_of_week").cast("int").alias("day_type"))
jan23 = jan23.withColumn("day_type", expr("is_weekday(day_type)"))

jan23 = jan23.withColumn("month", month(col("request_datetime")))
jan23 = jan23.withColumn("year", year(col("request_datetime")))

split_col = split(jan23['request_datetime'], ' ')
```

```
jan23 = jan23.withColumn('request_time', split_col.getItem(1))
jan23 = jan23.withColumn("request_hour", hour(col("request_time")))

## create a new column to show the wait time of a passenger
jan23 = jan23.withColumn("wait_seconds",
(col("on_scene_datetime").cast("long")- col("request_datetime").cast("long")))
jan23 = jan23.withColumn("wait_minutes", col("wait_seconds") / 60)

# drop unnecessary columns
columns_to_drop= ["access_a_ride_flag", "wav_request_flag", "wav_match_flag",
"bcf", "sales_tax", "shared_request_flag", "dispatching_base_num",
"originating_base_num", "DOLocationID", "PULocationID", "hvfhs_license_num",
"airport_fee", "tolls", "congestion_surcharge", "request_datetime",
"on_scene_datetime", "pickup_datetime", "dropoff_datetime", "wait_seconds",
"request_time", "request_date", "day_of_week"]
jan23 = jan23.drop(*columns_to_drop)

jan23.show(5, truncate=False)

# write the parquet to my S3 Trusted Bucket
jan23.write.parquet("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2023-01.
parquet")
```

Appendix E- Code Used for Modeling Pipeline

```
# read 2020 files
parquet_2020 =
[("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-01.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-02.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-03.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-04.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-05.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-06.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-07.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-08.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-09.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-10.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-11.parquet"),
("s3://my-project-nm/trusted/trusted_fhvhv_tripdata_2020-12.parquet")]

fhv2020 = [spark.read.parquet(p) for p in parquet_2020]
combined_2020 = fhv2020[0]
for p in fhv2020[1:]:
   combined_2020 = combined_2020.union(p)

# create new columns to show our tip label
combined_2020 = combined_2020.withColumn("label20",
(combined_2020.base_passenger_fare*0.20))

# COMPARING TIPS THAT ARE 20%
# Create a label. =1 if good tip, =0 otherwise
combined_2020 = combined_2020.withColumn("label", when(combined_2020.tips >=
combined_2020.label20, 1.0).otherwise(0.0))

# Create an indexer for the string based columns
indexer = StringIndexer(inputCols=["shared_match_flag",
"congestion_surcharge_flag", "tolls_flag", "airport_fee_flag", "hvfhs_company",
"PULocationBorough", "DOLocationBorough", "day_type"],
                        outputCols=["shared_matchIndex", "congestionIndex",
"tollsIndex", "airportIndex", "hvfhsIndex", "PUIndex", "DOIndex",
"daytypeIndex"])
```

```
# Create an encoder for the indexes and the integer columns.
encoder = OneHotEncoder(inputCols=["shared_matchIndex", "congestionIndex",
"tollsIndex", "airportIndex", "hvfhsIndex", "PUIndex", "DOIndex",
"daytypeIndex", "day_of_month",  "month", "year", "request_hour"],
                       outputCols=["shared_matchVector", "congestionVector",
"tollsVector", "airportVector", "hvfhsVector", "PUVector", "DOVector",
"daytypeVector", "daymonthVector", "monthVector", "yearVector",
"requesthourVector"], dropLast=True, handleInvalid="keep")

# Create an assembler for the individual feature vectors and the float/double
columns
assembler = VectorAssembler(inputCols=["shared_matchVector",
"congestionVector", "tollsVector", "airportVector", "hvfhsVector", "PUVector",
"DOVector", "daytypeVector", "daymonthVector", "monthVector", "yearVector",
"requesthourVector", "trip_miles", "trip_time", "base_passenger_fare",
"wait_minutes"], outputCol="features")

# Create a LogisticRegression Estimator
lr = LogisticRegression()

# split the data into two subsets
trainingData, testData = combined_2020.randomSplit([0.7, 0.3])
# create the pipeline
fhv_pipe = Pipeline(stages=[indexer, encoder, assembler, lr])

# Create the parameter grid
grid = ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 0.5, 1.0])
grid = grid.addGrid(lr.elasticNetParam, [0, 1])
grid = grid.build()

print("number of models to be tested: ", len(grid))
# Create a BinaryClassificationEvaluator to evaluate how well the model works
evaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")
# Create the CrossValidator using the hyperparameter grid
cv = CrossValidator(estimator=fhv_pipe,
                   estimatorParamMaps=grid,
```

```
                    evaluator=evaluator,
                    numFolds=3)
# Train the models
cv = cv.fit(trainingData)
predictions = cv.transform(testData)
auc = evaluator.evaluate(predictions)
print("auc:", auc)
```

Appendix F- Code Used for Data Visualization

```
# Show the confusion matrix
predictions.groupby('label').pivot('prediction').count().sort('label').show()

# Save the confusion matrix
cm =
predictions.groupby('label').pivot('prediction').count().fillna(0).collect()
def calculate_recall_precision(cm):
   tn = cm[0][1] #true negative
   fp = cm[0][2] #false positive
   fn = cm[1][1] #false negative
   tp = cm[1][2] #true positive
   precision = tp / ( tp + fp )
   recall = tp / ( tp + fn )
   accuracy = ( tp + tn ) / ( tp + tn + fp + fn )
   f1_score = 2 * ( ( precision * recall ) / ( precision + recall ) )
   print("accuracy: ", accuracy)
   print("precision: ", precision)
   print("recall: ", recall)
   print("f1 score: ", f1_score)
   return accuracy, precision, recall, f1_score

print( calculate_recall_precision(cm) )


## SHOW ROC CURVES
# Look at the parameters for the best model that was evaluated from the grid
parammap = cv.bestModel.stages[3].extractParamMap()

for p, v in parammap.items():
   print(p, v)

# Grab the model from Stage 3 of the pipeline
mymodel = cv.bestModel.stages[3]

import matplotlib.pyplot as plt
plt.figure(figsize=(5,5))
plt.plot(mymodel.summary.roc.select('FPR').collect(),
        mymodel.summary.roc.select('TPR').collect())
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")
plt.savefig("roc1.png")

parammap = cv.bestModel.stages[3].extractParamMap()

for p, v in parammap.items():
   print(p, v)

# Grab the model from Stage 3 of the pipeline
mymodel = cv.bestModel.stages[3]
plt.figure(figsize=(6,6))
plt.plot([0, 1], [0, 1], 'r--')
x = mymodel.summary.roc.select('FPR').collect()
y = mymodel.summary.roc.select('TPR').collect()
plt.scatter(x, y)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC Curve")
plt.savefig("reviews_roc.png")

## CORRELATION MATRIX
correlation_columns = ['trip_miles', 'trip_time', 'base_passenger_fare',
'tips', 'day_of_month', 'month', 'request_hour', 'wait_minutes']
numeric_all = spark.read.parquet(*parquet_all).select(correlation_columns)

# Convert the numeric values to vector columns
vector_column = "correlation_features"

# Make a list of all of the numeric columns
numeric_columns = ['trip_miles', 'trip_time', 'base_passenger_fare', 'tips',
'day_of_month', 'month', 'request_hour', 'wait_minutes']

# Use a vector assembler to combine all of the numeric columns together
assembler = VectorAssembler(inputCols=numeric_columns, outputCol=vector_column)
sdf_vector = assembler.transform(numeric_all).select(vector_column)
```

```
# Create the correlation matrix, then get just the values and convert to a list
matrix = Correlation.corr(sdf_vector, vector_column).collect()[0][0]
correlation_matrix = matrix.toArray().tolist()

# Convert the correlation to a Pandas dataframe
correlation_matrix_df = pd.DataFrame(data=correlation_matrix,
columns=numeric_columns, index=numeric_columns)

# Crate the plot using Seaborn
plt.figure(figsize=(16,5))
sns.heatmap(correlation_matrix_df,
           xticklabels=correlation_matrix_df.columns.values,
           yticklabels=correlation_matrix_df.columns.values,
           cmap="Greens",
           annot=True)
plt.savefig("correlation_matrix.png")

## BAR GRAPHS OF NUMBER OF FHV ORDERS BY MONTH
# select the columns that are needed
month_columns = ["month", "request_hour", "tips"]

# create a sdf with the specified parquet files and columns
monthrequesthourtips = spark.read.parquet(*parquet_all).select(month_columns)

# Use groupby to get a count by date. Then convert to pandas dataframe
month = monthrequesthourtips.groupby("month").count().sort("month").toPandas()

# Using Pandas built-in plotting functions
# Create a bar plot using the columns order_date and count
monthplot = month.plot.bar('month','count')
# Set the x-axis and y-axis labels
monthplot.set(xlabel='Month', ylabel='Number of FHV Orders')
# Set the title
monthplot.set(title='Number of FHV Orders by Month')
monthplot.figure.set_tight_layout('tight')
# Save the plot as a PNG file
monthplot.get_figure().savefig("order_rides_by_month.png")
```

```
## BAR GRAPH OF NUMBER OF FHV RIDES BY BOROUGH
# select the columns that are needed
borough_columns = ["PULocationBorough", "DOLocationBorough", "tips"]

# create a sdf with the specified parquet files and columns
boroughtips = spark.read.parquet(*parquet_all).select(borough_columns)

# Use groupby to get a count by date. Then convert to pandas dataframe
borough =
boroughtips.groupby("PULocationBorough").count().sort("PULocationBorough").toPa
ndas()

# Using Pandas built-in plotting functions
# Create a bar plot using the columns order_date and count
boroughplot = borough.plot.bar('PULocationBorough','count')
# Set the x-axis and y-axis labels
boroughplot.set(xlabel='Borough', ylabel='Number of FHV Orders')
# Set the title
boroughplot.set(title='Number of FHV Orders by Borough')
boroughplot.figure.set_tight_layout('tight')
# Save the plot as a PNG file
boroughplot.get_figure().savefig("order_rides_by_borough.png")
```