High Volume For-Hire Vehicle

CIS 4130

Michelle Chow

michelle.chow@baruchmail.cuny.edu

Milestone 1 – Proposal

The dataset that I have chosen is from Kaggle and is called “NYC FHV (Uber/Lyft) Trip Data Expanded (2019-2022)”. This dataset includes data from every For-Hire Vehicle trip in New York City between 2019 and 2022. These trips include passenger trips from various rideshare companies including Uber, Lyft, Juno, and Via. Within each trip, there is detailed information about the specifics of each ride such as prices, tips, fees, distance of the trip, time of the trip, etc. From the dataset, we get a good sense of trends in the rides across various key factors. According to the pdf file, “data_dictionary_trip_records_hvfhs.pdf”, from the Kaggle dataset, “This data dictionary describes High Volume FHV trip data. Each row represents a single trip in an FHV dispatched by one of NYC’s licensed High Volume FHV bases.” The dataset from Kaggle can be found at this URL: https://www.kaggle.com/datasets/jeffsinsel/nyc-fhvhv-data. However, the original data can be found at this website: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

This NYC FHV dataset includes a large variety of attributes that describe the data in detail. According to the same pdf file as above, the attributes of this dataset include:

- **hvfhs_license_num** = TLC license number of the HVFHS base or business. As of September 2019, HVFHS licensees are: HV0002 = Juno, HV0003 = Uber, HV0004 = Via, HV0005 = Lyft
- **dispatching_base_num** = TLC Base License Number of the base that dispatched the trip
- **pickup_datetime** = Date and time of the trip pick-up
- **dropoff_datetime** = Date and time of the trip drop-off
- **PULocationID** = TLC Taxi Zone in which the trip began
- **DOLocationID** = TLC Taxi Zone in which the trip ended
- **originating_base_num** = Base number of the base that received the original trip request
- **request_datetime** = Date/time when passenger requested to be picked up
- **on_scene_datetime** = Date/time when driver arrived at the pick-up location (Accessible Vehicles only)
- **trip_miles** = Total miles for passenger trip
- **trip_time** = Total time in seconds for passenger trip
- **base_passenger_fare** = Base passenger fare before tolls, tips, taxes, and fees
- **tolls** = Total amount of all tolls paid in trip
- **bcf** = Total amount collected in trip for Black Car Fund
- **sales_tax** = Total amount collected in trip for NYS sales tax
- **congestion_surcharge** = Total amount collected in trip for NYS congestion surcharge
- **airport_fee** = $2.50 for both drop off and pick-up at Laguardia, Newark, and John F. Kennedy airports
- **tips** = total amount of tips received from passenger
- **driver_pay** = total driver pay (not including tolls or tips and net of commission, surcharges, or taxes)
- **shared_request_flag** = Did the passenger agree to a shared/pooled ride, regardless of whether they were matched? (Y/N)
- **shared_match_flag** = Did the passenger share the vehicle with another passenger who booked separately at any point during the trip? (Y/N)

- **access_a_ride_flag** = Was the trip administered on behalf of the Metropolitan Transportation Authority (MTA)? (Y/N)
- **wav_request_flag** = Did the passenger request a wheelchair-accessible vehicle (WAV)? (Y/N)
- **wav_match_flag** = Did the trip occur in a wheelchair-accessible vehicle (WAV)? (Y/N)

Based on the available data from this dataset, I intend to model the various trends across the various attributes. Some of the models that I can potentially model include the most popular rideshare company that is used, most common pick-up and drop off times across all rideshare companies, most popular pick-up and drop off locations, and trends in trip miles and trip times. Since there are many attributes within this dataset, there are many types of models that can be derived from this dataset. In addition, many of these attributes impact one another, so multiple connections can be made between one another. I intend to predict the tip amount for rides, and this can be based on the total number of miles. Tip amounts vary by passenger and are determined by a variety of factors of the ride experience, but I believe the factor that impacts the tip amount the most is the number of miles. For example, when putting in a request for an Uber, the prices are based on distance, so I would naturally believe that distance impacts tip amounts as well. I intend to forecast the optimal combination of ride attributes including rideshare company, trip miles, trip time, pick-up and drop off times, etc. that contribute to the highest tip amount. I can also evaluate the reasons behind why the trends across the dataset attributes are the way they are. For example, are there reasons behind why a certain ride got a lower tip than another. Throughout this project, I intend to analyze this dataset and build a model that evaluates the factors behind the data and see why certain decisions were made by passengers.

Milestone 2 - Data Acquisition

*Please see my code in Appendix 1 – Milestone 2 – Data Acquisition Code

<u>EC2 Instance Setup Process</u>:

1. I created a new EC2 instance specifically for my project titled: HVFHV Project Instance. I followed the steps from class and used the same instance settings such as t2.micro, Amazon Linux, 30GB, etc.

2. Once the instance was up and running, I connected to my HVFHV Project Instance, and I followed the steps on the lecture 6 PowerPoint (Cloud Computing - Virtual Machines) to configure an AWS Command Line Interface. The commands I used to configure the AWS CLI were:

    a. `aws configure`

    b. Input my access key and secret access key

    c. `us-east-2` for the default region name

    d. `json` for the default output format

    As a test, to make sure my AWS CLI was configured, I typed: `aws iam list-users` to see my list of users, and I saw my administrator user, so it was working correctly.

<u>"Automated" S3 Bucket Data Download Process</u>:

3. Since I had no existing buckets, to create a new S3 bucket, I searched for S3 on the AWS home page. From there, I clicked the "Buckets" tab. I then clicked the "Create Bucket" button, and named my bucket: `hvfhv-project-mc`

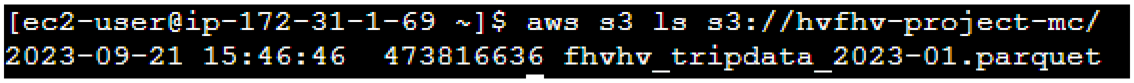
4. Since I am downloading data files from the web, I used the `curl` method. Also, to make this process easier, I used the `curl` command with two parts separated by "|". To download my files directly into my S3 project bucket without having them "land" on my EC2 instance hard drive, I used the following command:

```
curl -SL https://d37ci6vzurychx.cloudfront.net/trip-
data/fhvhv_tripdata_2023-01.parquet | aws s3 cp -
s3://hvfhv-project-mc/fhvhv_tripdata_2023-01.parquet
```

5. To check to make sure that the command downloaded the file correctly into my S3 project bucket, I used the command:

```
aws s3 ls s3://hvfhv-project-mc/
```

    This was the result of the above command:



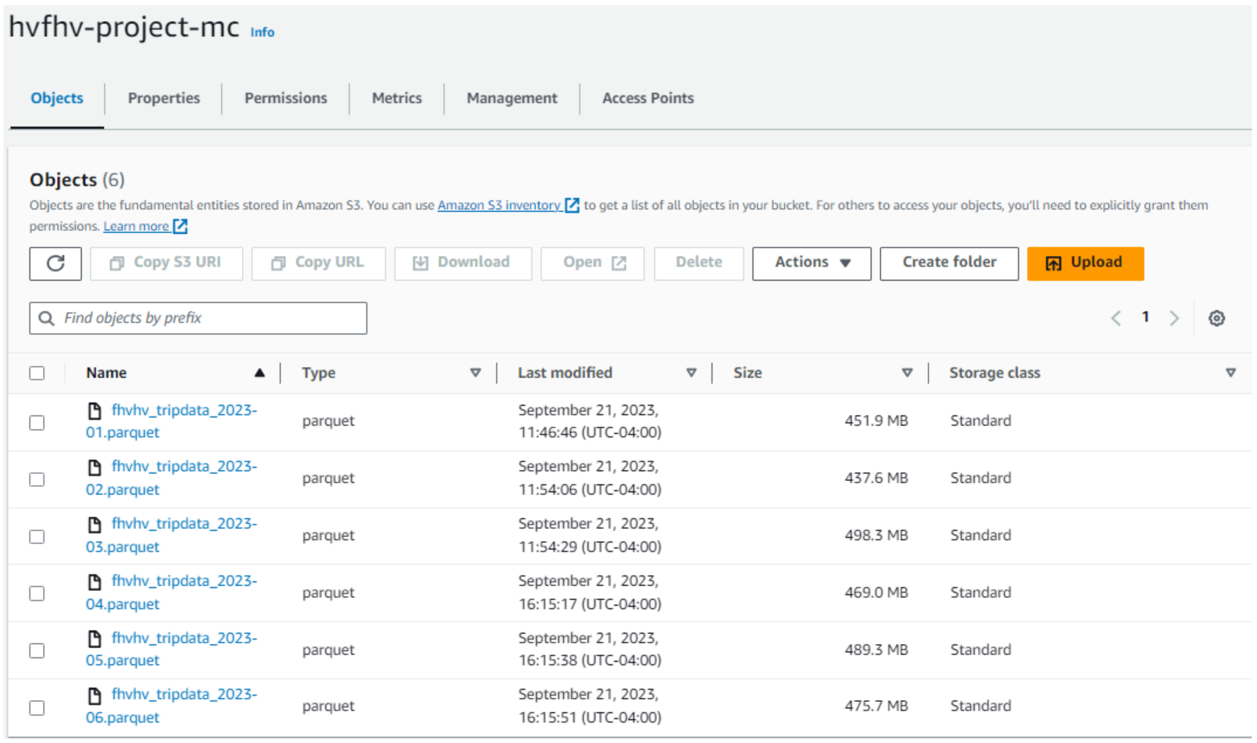
```
[ec2-user@ip-172-31-1-69 ~]$ aws s3 ls s3://hvfhv-project-mc/
2023-09-21 15:46:46  473816636 fhvhv_tripdata_2023-01.parquet
```

Since I see the name of my file in my S3 project bucket, I successfully downloaded my first data file into the bucket. I also went back to my S3 project bucket on the AWS website, and I was able to see my file there as well.

6. I used the command from step 4 to download five more files into my S3 project bucket, but I changed the URL and the name of the parquet files slightly since these are for different months data.

```
curl -SL https://d37ci6vzurychx.cloudfront.net/trip-
data/fhvhv_tripdata_2023-02.parquet | aws s3 cp -
s3://hvfhv-project-mc/fhvhv_tripdata_2023-02.parquet


curl -SL https://d37ci6vzurychx.cloudfront.net/trip-
data/fhvhv_tripdata_2023-03.parquet | aws s3 cp -
s3://hvfhv-project-mc/fhvhv_tripdata_2023-03.parquet


curl -SL https://d37ci6vzurychx.cloudfront.net/trip-
data/fhvhv_tripdata_2023-04.parquet | aws s3 cp -
s3://hvfhv-project-mc/fhvhv_tripdata_2023-04.parquet


curl -SL https://d37ci6vzurychx.cloudfront.net/trip-
data/fhvhv_tripdata_2023-05.parquet | aws s3 cp -
s3://hvfhv-project-mc/fhvhv_tripdata_2023-05.parquet


curl -SL https://d37ci6vzurychx.cloudfront.net/trip-
data/fhvhv_tripdata_2023-06.parquet | aws s3 cp -
s3://hvfhv-project-mc/fhvhv_tripdata_2023-06.parquet
```

7. I reviewed my S3 project bucket once again, and all 6 files that I downloaded were there.


hvfhv-project-mc Info
Objects | Properties | Permissions | Metrics | Management | Access Points

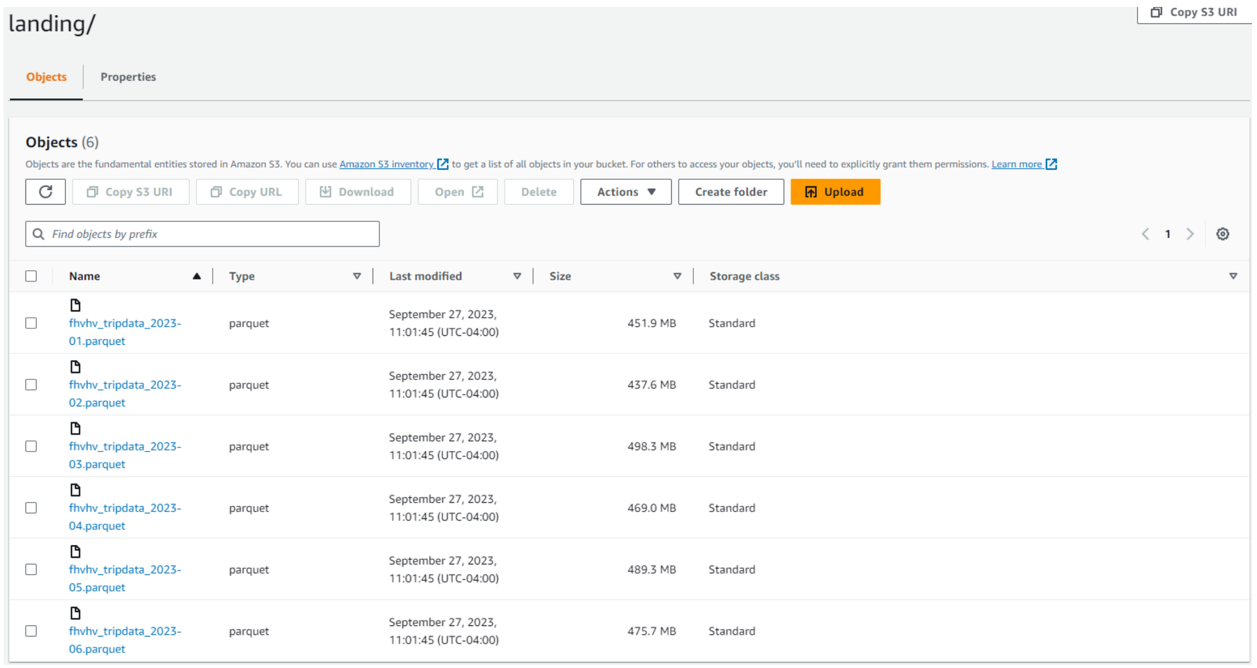
Objects (6)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload
Find objects by prefix

Name	Type	Last modified	Size	Storage class
fhvhv_tripdata_2023-01.parquet	parquet	September 21, 2023, 11:46:46 (UTC-04:00)	451.9 MB	Standard
fhvhv_tripdata_2023-02.parquet	parquet	September 21, 2023, 11:54:06 (UTC-04:00)	437.6 MB	Standard
fhvhv_tripdata_2023-03.parquet	parquet	September 21, 2023, 11:54:29 (UTC-04:00)	498.3 MB	Standard
fhvhv_tripdata_2023-04.parquet	parquet	September 21, 2023, 16:15:17 (UTC-04:00)	469.0 MB	Standard
fhvhv_tripdata_2023-05.parquet	parquet	September 21, 2023, 16:15:38 (UTC-04:00)	489.3 MB	Standard
fhvhv_tripdata_2023-06.parquet	parquet	September 21, 2023, 16:15:51 (UTC-04:00)	475.7 MB	Standard

8. Once all 6 files were in my S3 project bucket, I created a "landing" bucket within my project bucket, and then moved all 6 files into the "landing" bucket.


landing/
Copy S3 URI
Objects | Properties

Objects (6)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload
Find objects by prefix

Name	Type	Last modified	Size	Storage class
fhvhv_tripdata_2023-01.parquet	parquet	September 27, 2023, 11:01:45 (UTC-04:00)	451.9 MB	Standard
fhvhv_tripdata_2023-02.parquet	parquet	September 27, 2023, 11:01:45 (UTC-04:00)	437.6 MB	Standard
fhvhv_tripdata_2023-03.parquet	parquet	September 27, 2023, 11:01:45 (UTC-04:00)	498.3 MB	Standard
fhvhv_tripdata_2023-04.parquet	parquet	September 27, 2023, 11:01:45 (UTC-04:00)	469.0 MB	Standard
fhvhv_tripdata_2023-05.parquet	parquet	September 27, 2023, 11:01:45 (UTC-04:00)	489.3 MB	Standard
fhvhv_tripdata_2023-06.parquet	parquet	September 27, 2023, 11:01:45 (UTC-04:00)	475.7 MB	Standard

Alternative S3 Bucket Data Download Process:

9. Instead of directly downloading the data straight into the S3 project bucket, another way to download the data is to first download it on to the EC2 local file system. So, to copy a file on to the EC2 local file system, use this command:

```
curl -L -o fhvhv_tripdata_2023-01.parquet
https://d37ci6vzurychx.cloudfront.net/trip-
data/fhvhv_tripdata_2023-01.parquet
```

10. Use the following command to make sure the file was downloaded to the EC2 local file system:

```
ls -l
```

11. Once the file has been copied on to the EC2 local file system, copy the file over to the S3 project bucket using the following command:

```
aws s3 cp fhvhv_tripdata_2023-01.parquet s3://hvfhv-
project-mc/fhvhv_tripdata_2023-01.parquet
```

12. Use the same command from step 5 to check if the file is in the S3 bucket.

13. To remove the file from the EC2 local file system, use this command:

```
rm fhvhv_tripdata_2023-01.parquet
```

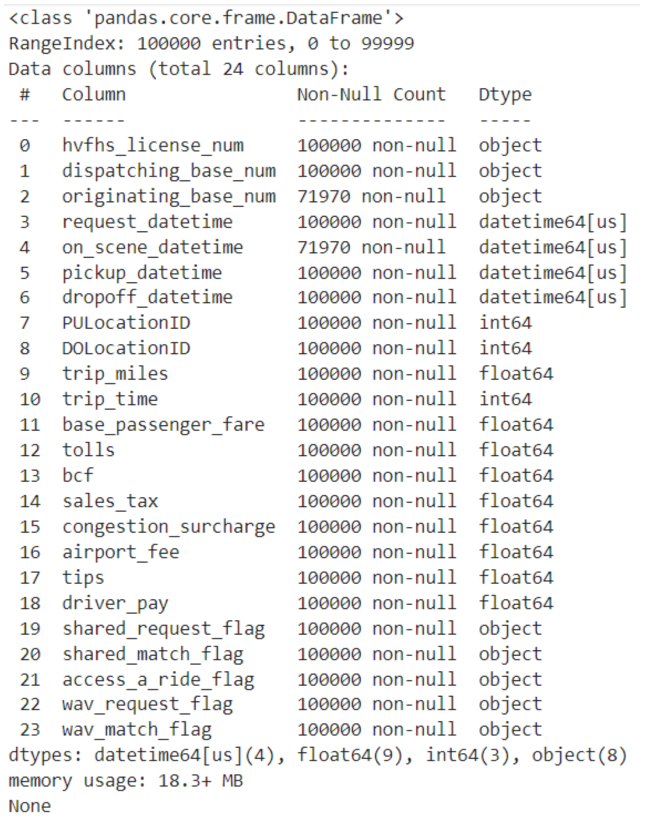
14. Repeat steps 9 to 13 for the remaining files and change the URL and name of the parquet files slightly to accommodate the correct file names.

Milestone 3 – Exploratory Data Analysis

Since my datasets are very large parquet files, I am unable to view the entirety of my file. Therefore, I am viewing a subset of my data to make my analysis easier. I have chosen to use 100,000 rows of my data, and there are 24 columns.

*Please see my code in Appendix 2 – Milestone 3 – Exploratory Data Analysis Code

To examine my data further, I used the df.info() to see the number of non-null values for each variable, as well as the data types for each. Based on this result (shown below), only 2 variables have null values, which are originating_base_num and on_scene_datetime. More specifically, both of these variables have 28,030 null values. I have a good variety of data types among my variables including object, datetime64, int64, and float64. Since the datetime variables already have the datetime data type, there is no need to parse them into actual datetime data types.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 24 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   hvfhs_license_num     100000 non-null  object
 1   dispatching_base_num  100000 non-null  object
 2   originating_base_num  71970 non-null   object
 3   request_datetime      100000 non-null  datetime64[us]
 4   on_scene_datetime     71970 non-null   datetime64[us]
 5   pickup_datetime       100000 non-null  datetime64[us]
 6   dropoff_datetime      100000 non-null  datetime64[us]
 7   PULocationID          100000 non-null  int64
 8   DOLocationID          100000 non-null  int64
 9   trip_miles            100000 non-null  float64
 10  trip_time             100000 non-null  int64
 11  base_passenger_fare   100000 non-null  float64
 12  tolls                 100000 non-null  float64
 13  bcf                   100000 non-null  float64
 14  sales_tax             100000 non-null  float64
 15  congestion_surcharge  100000 non-null  float64
 16  airport_fee           100000 non-null  float64
 17  tips                  100000 non-null  float64
 18  driver_pay            100000 non-null  float64
 19  shared_request_flag   100000 non-null  object
 20  shared_match_flag     100000 non-null  object
 21  access_a_ride_flag    100000 non-null  object
 22  wav_request_flag      100000 non-null  object
 23  wav_match_flag        100000 non-null  object
dtypes: datetime64[us](4), float64(9), int64(3), object(8)
memory usage: 18.3+ MB
None
```

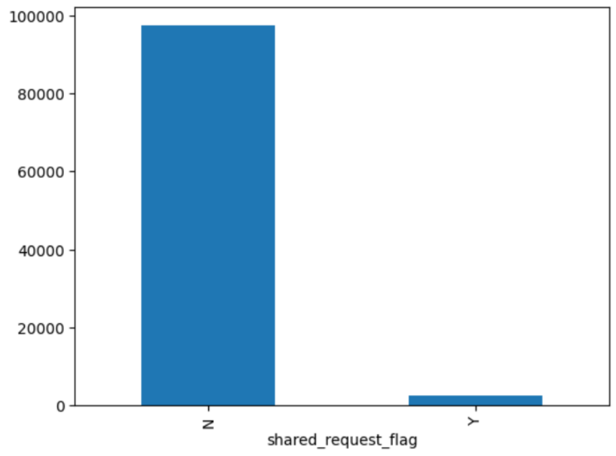
Considering that PULocationID and DOLocationID are mapped to a separate taxi zone lookup CSV file, I will merge this file with my parquet data file so that I can use the information in one designated location when I conduct my analysis and machine learning. This will also help with looking up LocationIDs to see which zone and borough each ID is linked to.

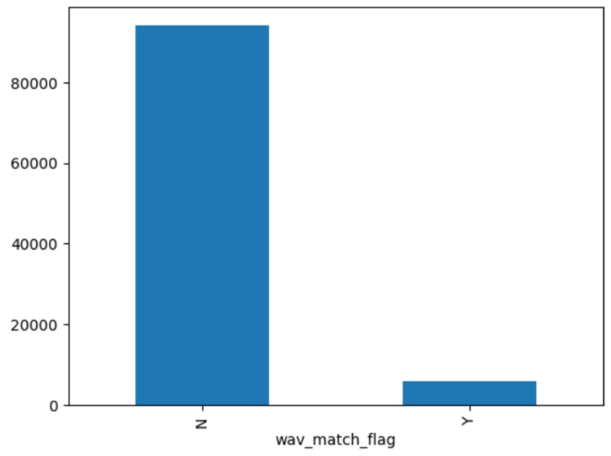
To see if there were any duplicated values within my dataset, I used df.duplicated().sum().sum(). This call outputs 0, meaning that I have no duplicates in my data.

By using the describe() method on the numeric variables, I was able to get statistical information such as the mean, standard deviation, minimum, 25%, 50%, 75%, and maximum. Based on these results, across some of the variables such as trip_miles, trip_time, base_passenger_fare, tolls, tips, and driver_pay, they seem to have some outliers because their maximum values are far away from their mean values. I also noticed that base_passenger_fare and driver_pay have a minimum of -36.76, which may indicate an error because it is not possible to have a negative monetary amount. To see the minimum and maximum values for the datetime variables, please see Appendix 2.

```
       trip_miles  trip_time  base_passenger_fare     tolls       bcf  \
count   100000.00  100000.00            100000.00 100000.00 100000.00
mean         4.81    1055.12                34.99      1.05      1.07
std          5.01     665.12                24.87      3.93      0.78
min          0.00       1.00               -36.76      0.00      0.00
25%          1.71     581.00                18.59      0.00      0.56
50%          3.21     900.00                28.80      0.00      0.88
75%          6.03    1363.00                44.20      0.00      1.36
max        112.87   12085.00               441.36     65.20     15.44

       sales_tax  congestion_surcharge  airport_fee      tips  driver_pay
count  100000.00             100000.00    100000.00 100000.00   100000.00
mean        2.92                  1.06         0.02      1.26       25.41
std         2.04                  1.33         0.23      3.64       15.35
min         0.00                  0.00         0.00      0.00      -36.46
25%         1.53                  0.00         0.00      0.00       14.75
50%         2.46                  0.00         0.00      0.00       22.72
75%         3.79                  2.75         0.00      0.00       32.95
max        29.74                  2.75         5.00     98.00      364.62
```

To visualize the distribution of N and Y in categorical variables, shared_request_flag, shared_match_flag, wav_request_flag, and wav_match_flag, I created bar graphs for each (which can be found in Appendix 2). In all of these graphs, they all have a significantly larger N category than Y category. This signifies that the majority of passengers did not agree to shared rides, did not share the vehicle with another passenger, and did not request or use a wheelchair-accessible vehicle (WAV). Two of these graphs are shown below.

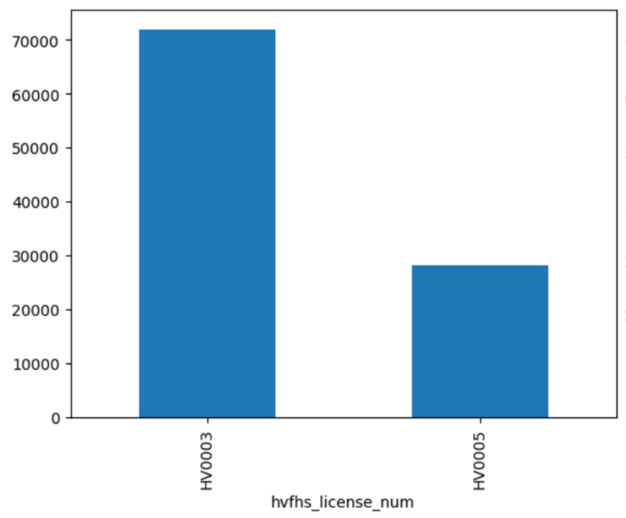


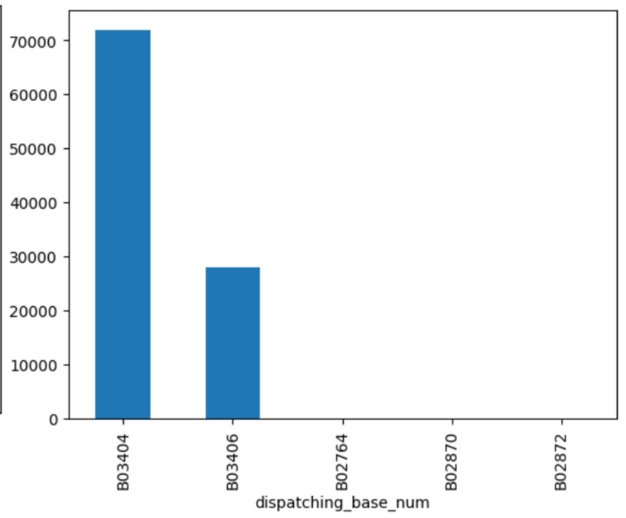


When graphing the access_a_ride_flag variable, there seemed to be issues with the data because instead of the expected Y and N categories, there was no Y category. Instead, it was a whitespace category because most of the values contained one whitespace (“ ”). Since this variable does not impact the analysis of my data, I will be excluding the access_a_ride_flag variable from the remaining analysis and milestones moving forward. The below screenshot is a frequency count that I generated.

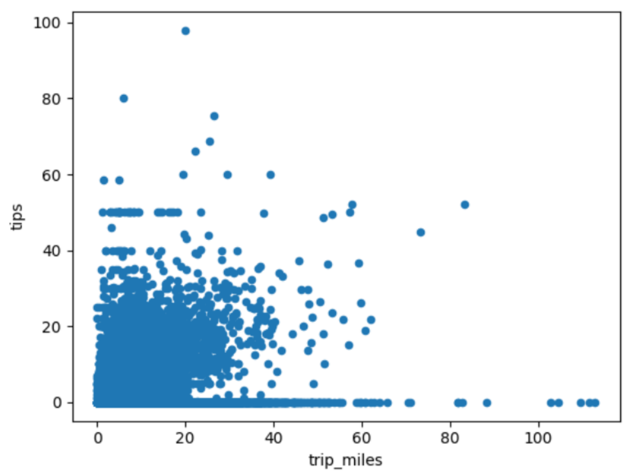
```
access_a_ride_flag
     71947
N    28053
dtype: int64
```

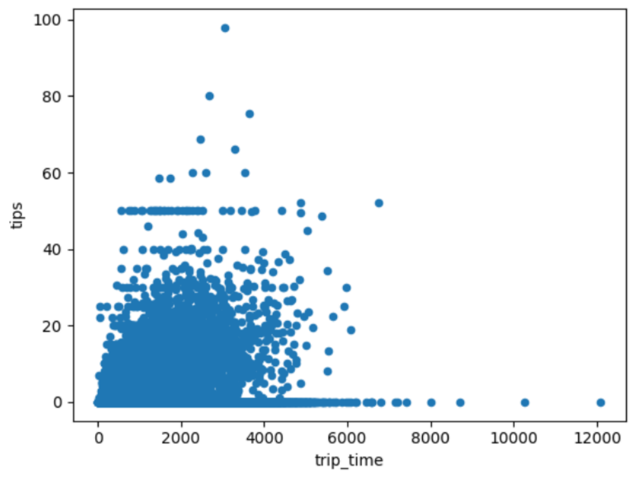
I also created graphs highlighting the frequency of the license plate types.

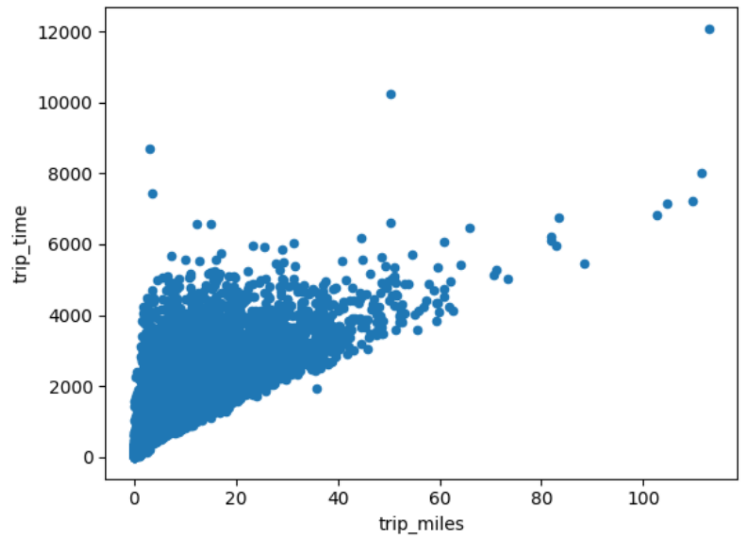




Finally, to see the relationship between a few continuous variables, I created scatter plots between trip_miles and tips, trip_time and tips, and trip_miles and trip_time. All three graphs demonstrate a positive correlation between each variable because as the variable on the x-axis increases, the variable on the y-axis also increases.







Challenges:

Evaluating the next steps, I do not see any challenges that will arise in cleaning my data and conducting feature engineering. The majority of my data does not require much cleaning because all of it is adequate and usable the way it is. I would only need to consider the null values for two of my variables. Overall, the variables and data that are included in my dataset are very useful in generating summary information about the dataset as a whole because they provide a lot of insight about trends and relationships between variables.

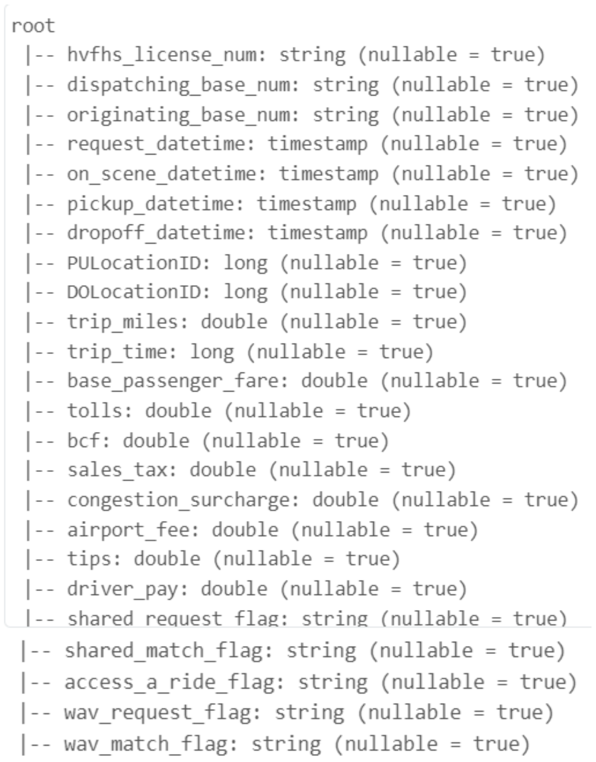
Milestone 4 – Feature Engineering and Modeling

*Please see my code in Appendix 3: Milestone 4 – Feature Engineering and Modeling Code

Data Cleaning Process:

In the first part of this milestone, I had to clean my data. Luckily, my data did not require much cleaning since most of it was usable as is. I completed two tasks to get my data to a clean state.

Before even cleaning my data, I had to read in one of my files from my landing folder using my AWS access key and secret access key. I then assigned the data to a spark dataframe so that I could use it in my later coding steps. After reading in one of my data files, I did not need to apply the schema to the data since my file types are parquet files, and already had pre-existing schemas. When I called `sdf.printSchema()`, there was already a schema assigned.

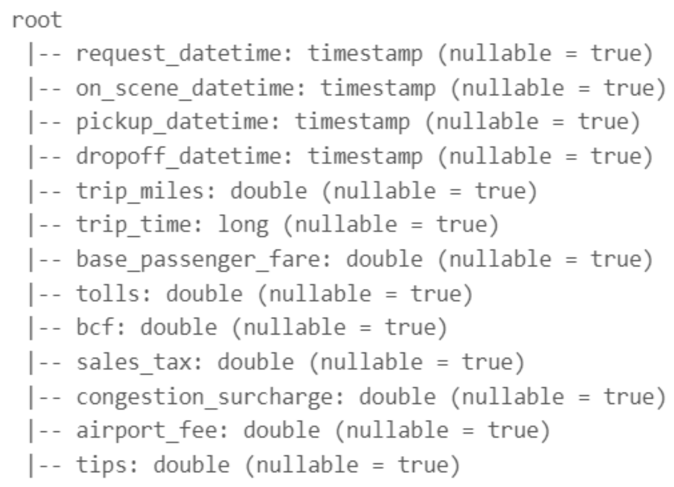


```
root
 |-- hvfhs_license_num: string (nullable = true)
 |-- dispatching_base_num: string (nullable = true)
 |-- originating_base_num: string (nullable = true)
 |-- request_datetime: timestamp (nullable = true)
 |-- on_scene_datetime: timestamp (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- PULocationID: long (nullable = true)
 |-- DOLocationID: long (nullable = true)
 |-- trip_miles: double (nullable = true)
 |-- trip_time: long (nullable = true)
 |-- base_passenger_fare: double (nullable = true)
 |-- tolls: double (nullable = true)
 |-- bcf: double (nullable = true)
 |-- sales_tax: double (nullable = true)
 |-- congestion_surcharge: double (nullable = true)
 |-- airport_fee: double (nullable = true)
 |-- tips: double (nullable = true)
 |-- driver_pay: double (nullable = true)
 |-- shared_request_flag: string (nullable = true)
 |-- shared_match_flag: string (nullable = true)
 |-- access_a_ride_flag: string (nullable = true)
 |-- wav_request_flag: string (nullable = true)
 |-- wav_match_flag: string (nullable = true)
```

The first data cleaning task I performed was to drop all unnecessary columns. The 11 columns I dropped were: "hvfhs_license_num", "dispatching_base_num", "originating_base_num", "PULocationID", "DOLocationID", "driver_pay", "shared_request_flag", "shared_match_flag", "wav_request_flag", "wav_match_flag", and "access_a_ride_flag". I dropped the "access_a_ride_flag" column because based on my Milestone 3 – Exploratory Data Analysis results, the Y category had an issue. Instead of the values having "Y", they had one whitespace. Since I determined that these columns would not impact my prediction results for my model, I decided to drop these columns using the following command:

```
dropped_cols_sdf = sdf.drop('hvfhs_license_num',
'dispatching_base_num', 'originating_base_num', 'PULocationID',
'DOLocationID', ‘driver_pay’, 'shared_request_flag',
'shared_match_flag', 'wav_request_flag', 'wav_match_flag',
'access_a_ride_flag')
```

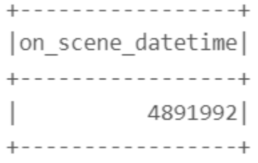
I assigned it a new name, dropped_cols_sdf, so that the next few steps would use this new spark dataframe without these columns. Then, to check and make sure the columns were dropped, I used `dropped_cols_sdf.printSchema()`, and my remaining 13 columns were listed.



```
root
 |-- request_datetime: timestamp (nullable = true)
 |-- on_scene_datetime: timestamp (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- trip_miles: double (nullable = true)
 |-- trip_time: long (nullable = true)
 |-- base_passenger_fare: double (nullable = true)
 |-- tolls: double (nullable = true)
 |-- bcf: double (nullable = true)
 |-- sales_tax: double (nullable = true)
 |-- congestion_surcharge: double (nullable = true)
 |-- airport_fee: double (nullable = true)
 |-- tips: double (nullable = true)
```

The second task I completed to clean my data even more, was to drop all null records. To check how many null records there were in the “on_scene_datetime” column, I used the following code:

```
dropped_cols_sdf.select([count(when(col(c).isNull(),
c)).alias(c) for c in ["on_scene_datetime"] ] ).show()
```



```
+-----------------+
|on_scene_datetime|
+-----------------+
|          4891992|
+-----------------+
```

Since only one variable had null values, I used the following code to drop all of the empty records:

```
clean_sdf =
dropped_cols_sdf.na.drop(subset=[“on_scene_datetime”])
```

After dropping the null rows, I assigned the dataframe a new name, clean_sdf, so that my feature engineering and modeling steps can use the fully cleaned data. Once I finished cleaning my data, I wrote my data to a parquet file and saved it in my raw folder using this code:

```
clean_sdf.write.parquet('s3://hvfhv-project-
mc/raw/cleaned_fhvhv_tripdata_2023-01.parquet')
```

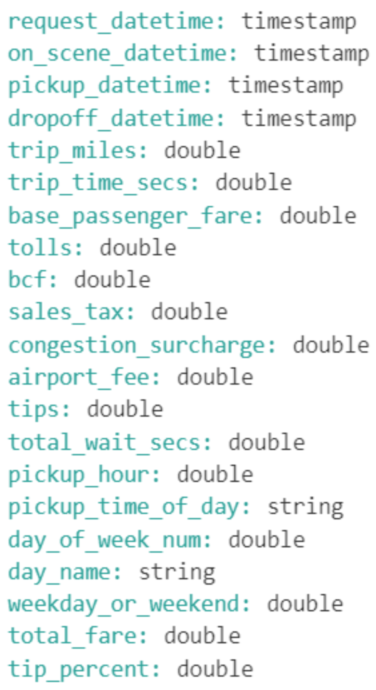
After successfully executing the above code, I went into my raw folder in my AWS S3 bucket, and the file was saved as a folder type. Within this folder were different partitions of the cleaned file that I wrote as a parquet file.

<u>Feature Engineering Process:</u>

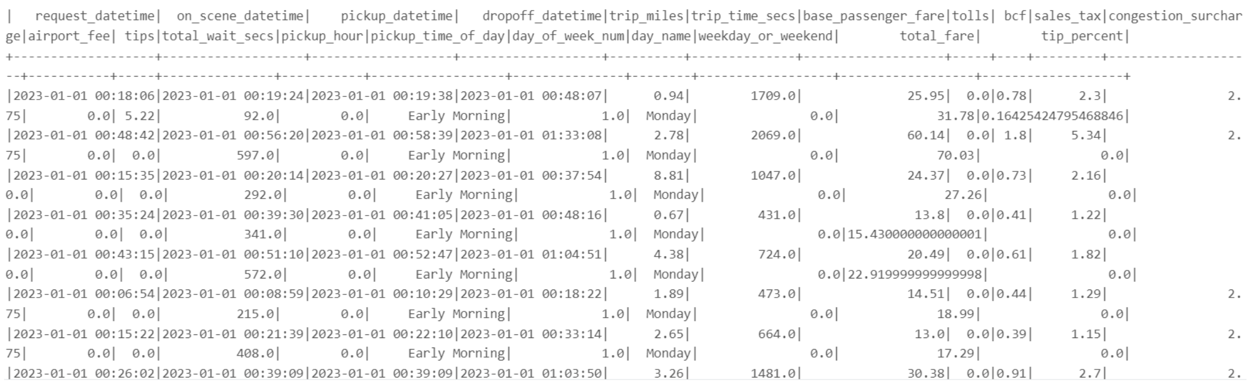
To start off my feature engineering process, I read in my cleaned data from my raw bucket. I then had to look at the columns that were already in the dataset and determine if any of these were usable as is for features. Only two of the columns were able to be used as features, and these were "trip_miles" and "trip_time". Since "trip_time" was a long data type, I converted this into a double using `.cast('double')`. In addition, considering that most of my columns were not able to be used as features as they were originally created, I had to extract features from those columns to create unique features more relevant to what I wanted to predict. I also had to take into account that before putting features into my VectorAssembler, the data types for the features had to be double or a vector.

The features that I engineered from the original columns are:

- **total_wait_secs** = The total time the passenger waited from after the request was put in, to when the driver picked them up
- **pickup_hour** = The hour that the passenger was picked up, which was extracted from the "pickup_datetime" timestamp
- **pickup_time_of_day** = The relative time of day the passenger was picked up, grouped into one of five categories, early morning, morning, noon, afternoon, or night
- **pickup_time_of_day_vector** = After "pickup_time_of_day" was indexed using StringIndexer, "pickup_time_of_day_index" was the resulting column. This was then converted to a vector using OneHotEncoder
- **day_of_week_num** = The day of the week converted to a number (Monday = 1, Tuesday = 2, Wednesday = 3, Thursday = 4, Friday = 5, Saturday = 6, Sunday = 7)
- **day_name** = The name of the day associated with the number from "day_of_week_num"
- **day_name_vector** = After "day_name" was indexed using StringIndexer, "day_name_index" was the resulting column. This was then converted to a vector using OneHotEncoder
- **weekday_or_weekend** = 0.0 = weekday and 1.0 = weekend
- **total_fare** = "base_passenger_fare" + "tolls" + "bcf" + "sales_tax" + "congestion_surcharge" + "airport_fee"
- **tip_percent** = "tips" / "total_fare"



```
request_datetime: timestamp
on_scene_datetime: timestamp
pickup_datetime: timestamp
dropoff_datetime: timestamp
trip_miles: double
trip_time_secs: double
base_passenger_fare: double
tolls: double
bcf: double
sales_tax: double
congestion_surcharge: double
airport_fee: double
tips: double
total_wait_secs: double
pickup_hour: double
pickup_time_of_day: string
day_of_week_num: double
day_name: string
weekday_or_weekend: double
total_fare: double
tip_percent: double
```



```
|   request_datetime|  on_scene_datetime|    pickup_datetime|   dropoff_datetime|trip_miles|trip_time_secs|base_passenger_fare|tolls| bcf|sales_tax|congestion_surchar
ge|airport_fee| tips|total_wait_secs|pickup_hour|pickup_time_of_day|day_of_week_num|day_name|weekday_or_weekend|        total_fare|        tip_percent|
+-------------------+-------------------+-------------------+-------------------+----------+--------------+-------------------+-----+----+---------+------------------
--+-----------+-----+---------------+-----------+------------------+---------------+--------+------------------+------------------+-------------------+
|2023-01-01 00:18:06|2023-01-01 00:19:24|2023-01-01 00:19:38|2023-01-01 00:48:07|      0.94|        1709.0|              25.95|  0.0|0.78|      2.3|                2.
75|        0.0| 5.22|           92.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             31.78|0.16425424795468846|
|2023-01-01 00:48:42|2023-01-01 00:56:20|2023-01-01 00:58:39|2023-01-01 01:33:08|      2.78|        2069.0|              60.14|  0.0| 1.8|     5.34|                2.
75|        0.0|  0.0|          597.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             70.03|                0.0|
|2023-01-01 00:15:35|2023-01-01 00:20:14|2023-01-01 00:20:27|2023-01-01 00:37:54|      8.81|        1047.0|              24.37|  0.0|0.73|     2.16|
0.0|        0.0|  0.0|          292.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             27.26|                0.0|
|2023-01-01 00:35:24|2023-01-01 00:39:30|2023-01-01 00:41:05|2023-01-01 00:48:16|      0.67|         431.0|               13.8|  0.0|0.41|     1.22|
0.0|        0.0|  0.0|          341.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|15.430000000000001|                0.0|
|2023-01-01 00:43:15|2023-01-01 00:51:10|2023-01-01 00:52:47|2023-01-01 01:04:51|      4.38|         724.0|              20.49|  0.0|0.61|     1.82|
0.0|        0.0|  0.0|          572.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|22.919999999999998|                0.0|
|2023-01-01 00:06:54|2023-01-01 00:08:59|2023-01-01 00:10:29|2023-01-01 00:18:22|      1.89|         473.0|              14.51|  0.0|0.44|     1.29|                2.
75|        0.0|  0.0|          215.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             18.99|                0.0|
|2023-01-01 00:15:22|2023-01-01 00:21:39|2023-01-01 00:22:10|2023-01-01 00:33:14|      2.65|         664.0|               13.0|  0.0|0.39|     1.15|                2.
75|        0.0|  0.0|          408.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             17.29|                0.0|
|2023-01-01 00:26:02|2023-01-01 00:39:09|2023-01-01 00:39:09|2023-01-01 01:03:50|      3.26|        1481.0|              30.38|  0.0|0.91|      2.7|                2.
```

Pipeline Creation Process:

When assembling my pipeline, I created a label, which is what I am predicting. Since I am predicting tips, I used the Binarizer to categorize a good tip versus a bad tip. I set the threshold to 0.01, so anything greater than 0.01 is considered a good tip, while anything less than 0.01 is considered a bad tip. I categorized the tip as a binary value, 1.0 = good tip (> 0.01), and 0.0 = bad tip (< 0.01). When determining what threshold, I wanted a more balanced set between the good and bad tip. Since there were far more 0.0s, I decided to make the threshold very low to create the most balance. Essentially, by setting it at 0.01, I am saying that any tip at all is considered a good tip, while no tip is a bad tip. Since predicting the exact amount of a tip is more challenging and may not be very accurate, I thought it would be better to predict the tip category instead.

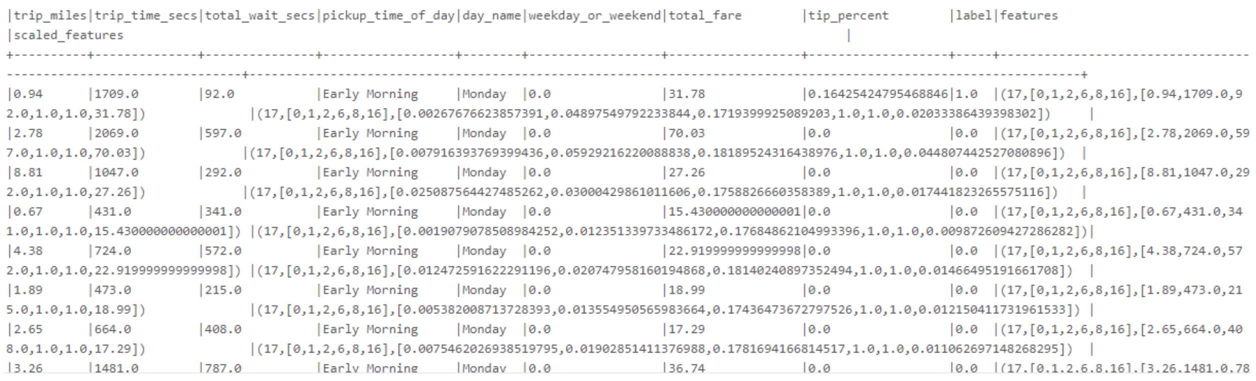
Next, since I had two features that were strings, "pickup_time_of_day" and "day_name", I used the StringIndexer to create two new features that were the indexes of these two columns. This changed the string features into double features and would allow it to be encoded in the next step. After indexing “pickup_time_of_day” and “day_name”, I used the OneHotEncoder to encode these columns into vectors. These would ultimately go into my VectorAssember.

Once I had all of my features engineered and in the correct data types, I called my VectorAssember. The features that I put into the assembler are: “trip_miles”, “trip_time_secs”, “total_wait_secs”, “pickup_time_of_day_vector”, “day_name_vector", "weekday_or_weekend”, and “total_fare”. I named the output column “features”. Then, since some of these features such as “trip_time_secs” and “total_wait_secs” are very large numbers that could be in the hundreds, I decided to scale my “features” column so that all of the numbers were in the range of 0.0 and 1.0. To do this, I used MinMaxScaler. The resulting vector showed all values in the range of 0.0 and 1.0.

My next step was to create my pipeline using all of the stages I put my data through. To assemble my pipeline, I used the following code:

```
hvfhv_pipe = Pipeline(stages=[indexer, encoder, assembler,
min_max_scaler])
```

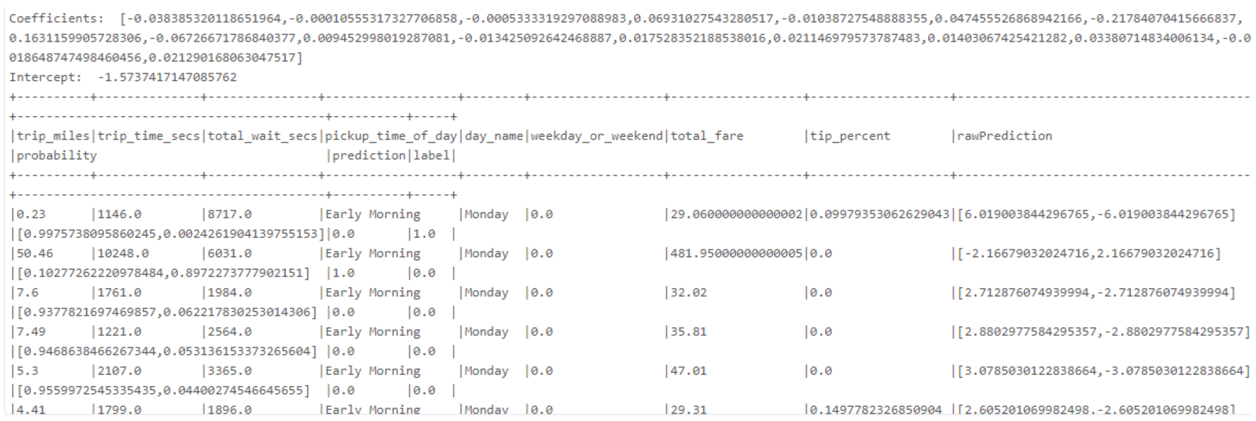
Then, to transform the data, I called `.fit()` on my hvfhv_pipe. Finally, to display this pipeline, I selected a few relevant features along with the label I created.



```
|trip_miles|trip_time_secs|total_wait_secs|pickup_time_of_day|day_name|weekday_or_weekend|total_fare        |tip_percent       |label|features
|scaled_features                                                                                                                   |
+----------+--------------+---------------+------------------+--------+------------------+------------------+------------------+-----+------------------------------------
------------------------------+------------------------------------------------------------------------------------------------------------------------+
|0.94      |1709.0        |92.0           |Early Morning     |Monday  |0.0               |31.78             |0.16425424795468846|1.0  |(17,[0,1,2,6,8,16],[0.94,1709.0,9
2.0,1.0,1.0,31.78])           |(17,[0,1,2,6,8,16],[0.00267676623857391,0.04897549792233844,0.1719399925089203,1.0,1.0,0.02033386439398302])     |
|2.78      |2069.0        |597.0          |Early Morning     |Monday  |0.0               |70.03             |0.0               |0.0  |(17,[0,1,2,6,8,16],[2.78,2069.0,59
7.0,1.0,1.0,70.03])          |(17,[0,1,2,6,8,16],[0.007916393769399436,0.05929216220088838,0.18189524316438976,1.0,1.0,0.044807442527080896])  |
|8.81      |1047.0        |292.0          |Early Morning     |Monday  |0.0               |27.26             |0.0               |0.0  |(17,[0,1,2,6,8,16],[8.81,1047.0,29
2.0,1.0,1.0,27.26])          |(17,[0,1,2,6,8,16],[0.025087564427485262,0.03000429861011606,0.1758826660358389,1.0,1.0,0.017441823265575116])   |
|0.67      |431.0         |341.0          |Early Morning     |Monday  |0.0               |15.430000000000001|0.0               |0.0  |(17,[0,1,2,6,8,16],[0.67,431.0,34
1.0,1.0,1.0,15.430000000000001]) |(17,[0,1,2,6,8,16],[0.0019079078508984252,0.012351339733486172,0.17684862104993396,1.0,1.0,0.009872609427286282])|
|4.38      |724.0         |572.0          |Early Morning     |Monday  |0.0               |22.919999999999998|0.0               |0.0  |(17,[0,1,2,6,8,16],[4.38,724.0,57
2.0,1.0,1.0,22.919999999999998]) |(17,[0,1,2,6,8,16],[0.012472591622291196,0.020747958160194868,0.18140240897352494,1.0,1.0,0.01466495191661708])  |
|1.89      |473.0         |215.0          |Early Morning     |Monday  |0.0               |18.99             |0.0               |0.0  |(17,[0,1,2,6,8,16],[1.89,473.0,21
5.0,1.0,1.0,18.99])           |(17,[0,1,2,6,8,16],[0.005382008713728393,0.013554950565983664,0.17436473672797526,1.0,1.0,0.012150411731961533]) |
|2.65      |664.0         |408.0          |Early Morning     |Monday  |0.0               |17.29             |0.0               |0.0  |(17,[0,1,2,6,8,16],[2.65,664.0,40
8.0,1.0,1.0,17.29])           |(17,[0,1,2,6,8,16],[0.0075462026938519795,0.01902851411376988,0.1781694166814517,1.0,1.0,0.011062697148268295])  |
|3.26      |1481.0        |787.0          |Early Morning     |Monday  |0.0               |36.74             |0.0               |0.0  |(17,[0,1,2,6,8,16],[3.26,1481.0,78
```

Model Specification Process:

It is now time to construct my model, so I decided to use a logistic regression model because since I am predicting a binary categorical value, this is the simplest model to use. I began by splitting up my data into a training and testing set, where 70% goes to the training set and 30% goes to the testing set. Then, when I went through the next few necessary steps to create my logistic regression estimator and test the model on the test data, I displayed the results and the confusion matrix.



```
Coefficients:  [-0.03838532011865196 4,-0.00010555317327706858,-0.0005333319297088983,0.0693102754328051 7,-0.01038727548888355,0.04745552686894216 6,-0.21784070415666837,
0.1631159905728306,-0.0672667178684037 7,0.00945299801928708 1,-0.01342509264246888 7,0.01752835218853801 6,0.02114697957378748 3,0.01403067425421282,0.0338071483400613 4,-0.0
01864874749846045 6,0.02129016806304751 7]
Intercept:  -1.5737417147085762
+----------+--------------+---------------+------------------+--------+------------------+------------------+------------------+----------------------------------------
+----------------------------------------+----------+-----+
|trip_miles|trip_time_secs|total_wait_secs|pickup_time_of_day|day_name|weekday_or_weekend|total_fare        |tip_percent       |rawPrediction
|probability                             |prediction|label|
+----------+--------------+---------------+------------------+--------+------------------+------------------+------------------+----------------------------------------
+----------------------------------------+----------+-----+
|0.23      |1146.0        |8717.0         |Early Morning     |Monday  |0.0               |29.060000000000002|0.0997935306262904 3|[6.019003844296765,-6.019003844296765]
|[0.9975738095860245,0.0024261904139755153]|0.0       |1.0  |
|50.46     |10248.0       |6031.0         |Early Morning     |Monday  |0.0               |481.95000000000005|0.0               |[-2.1667903202471 6,2.1667903202471 6]
|[0.1027726222097848 4,0.8972273777902151]  |1.0       |0.0  |
|7.6       |1761.0        |1984.0         |Early Morning     |Monday  |0.0               |32.02             |0.0               |[2.712876074939994,-2.712876074939994]
|[0.937782169746985 7,0.06221783025301430 6] |0.0       |0.0  |
|7.49      |1221.0        |2564.0         |Early Morning     |Monday  |0.0               |35.81             |0.0               |[2.880297758429535 7,-2.880297758429535 7]
|[0.946863846626734 4,0.05313615337326560 4] |0.0       |0.0  |
|5.3       |2107.0        |3365.0         |Early Morning     |Monday  |0.0               |47.01             |0.0               |[3.078503012283866 4,-3.078503012283866 4]
|[0.955997254533543 5,0.0440027454664565 5]  |0.0       |0.0  |
|4.41      |1799.0        |1896.0         |Early Morning     |Monday  |0.0               |29.31             |0.149778232685090 4|[2.605201069982498,-2.605201069982498]
```

```
+-----+-------+----+
|label|    0.0| 1.0|
+-----+-------+----+
|  0.0|3266899|9535|
|  1.0| 790598|5243|
+-----+-------+----+
```

<u>Model Evaluation Process:</u>

After running my model on the training and testing data, I calculated the accuracy, precision, recall, and F1 score to evaluate how well my model was at predicting correct predictions and outcomes. Generally, I would want all of the values to be high, which reflects a good model. The results are below, and based on these values, my accuracy is at about 0.80, which is relatively high meaning that across all predictions, 80% of predictions were correct. Considering that the other three values are low, my model is not the best at predicting positive predictions and positive outcomes. Overall, I would want high values in all four calculations to ensure a good model.

```
Accuracy, Precision, Recall, F1 Score
(0.8035169530544966, 0.3547841385843822, 0.006587999361681542, 0.012935793510884891)
```

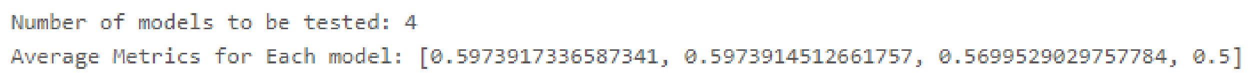
When I ran the cross-validator on my training data and used it to fit my training data, I evaluated the overall performance of my model over the three folds. The result is shown below:

```
cv.avgMetrics: [0.597391590869436]
```

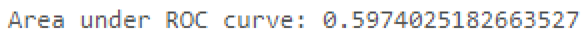
Then, I ran the cross-validator on my test data and the result is shown below:

```
evaluator.evaluate(cv.transform(testData)): 0.5967987430092285
```

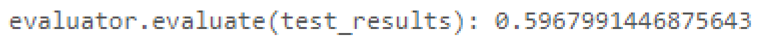
To begin the model optimization process, I created a parameter grid for different hyperparameters, and the average metrics for each of my four models are shown below. For regParam, I used [0.0, 1.0] so that my model does not have to test that many models. For elasticNetParam, I used [0.0, 1.0].


```
Number of models to be tested: 4
Average Metrics for Each model: [0.5973917336587341, 0.5973914512661757, 0.5699529029757784, 0.5]
```

After gathering the metrics and parameters of the model with the best average metrics, below is the value of the area under my ROC curve. My best model showed the area as about 0.60, which is not very good.


```
Area under ROC curve: 0.5974025182663527
```

Now that I evaluated what the best model is, I tested the model on the test data. The area under the ROC curve results is shown. This value is pretty much the same as the above value.


```
evaluator.evaluate(test_results): 0.5967991446875643
```

Across all of the code I ran to evaluate my model and the best model, all of the results appeared to be around 0.60. I want a number closer to 1.0, but based on my results, this most likely means that my model is not very good and accurate when making predictions.

After completing all of my feature engineering and modeling, my final step was to save my best model to my models folder, and save my feature engineered dataframe to my trusted folder in my S3 bucket.

Challenges:

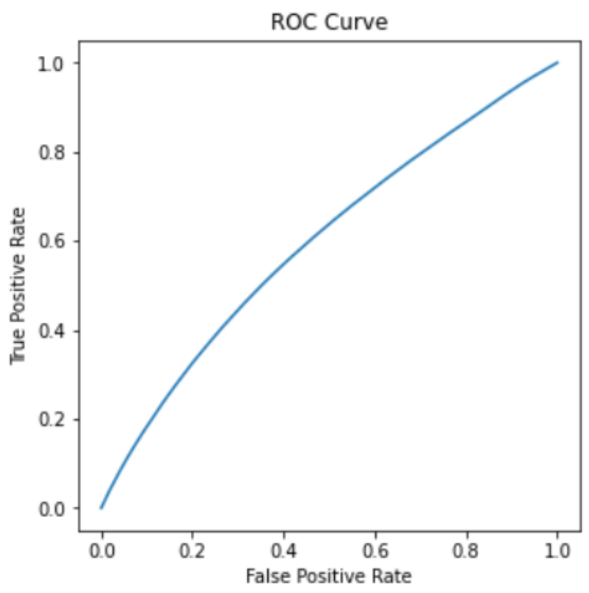
Throughout this milestone, when I was cleaning, feature engineering, building my model, and evaluating my model, I did not face many challenges. Since all of the code examples and resources were easily accessible and did not require many alterations, I was able to reuse most of the code from the class slides. The data cleaning process only required a few lines of code since my data was pretty clean and mostly usable as it was. However, it was a lot of trial and error during feature engineering because I had to figure out what new features I wanted to engineer, how to do that, and how to use indexers and encoders on those features. So, while this took a little bit of time to figure out, it did not pose much of a challenge. Another tricky aspect that I was wrangling with during my model specification was choosing the seed value for my logistic regression model. I tried many different values, some being very large and some being very small. Then, after each chosen seed, I would calculate the accuracy, precision, recall, and F1 score to see if there were any improvements. Overall, these calculations each round were very similar, so I decided to stick with the seed value of 42 that was shown on the class slide example. However, overall, as I evaluate the challenges of building an accurate model, this is a challenging task because it is difficult to engineer the right number of good features that will construct an excellent model.

Milestone 5 – Data Visualization

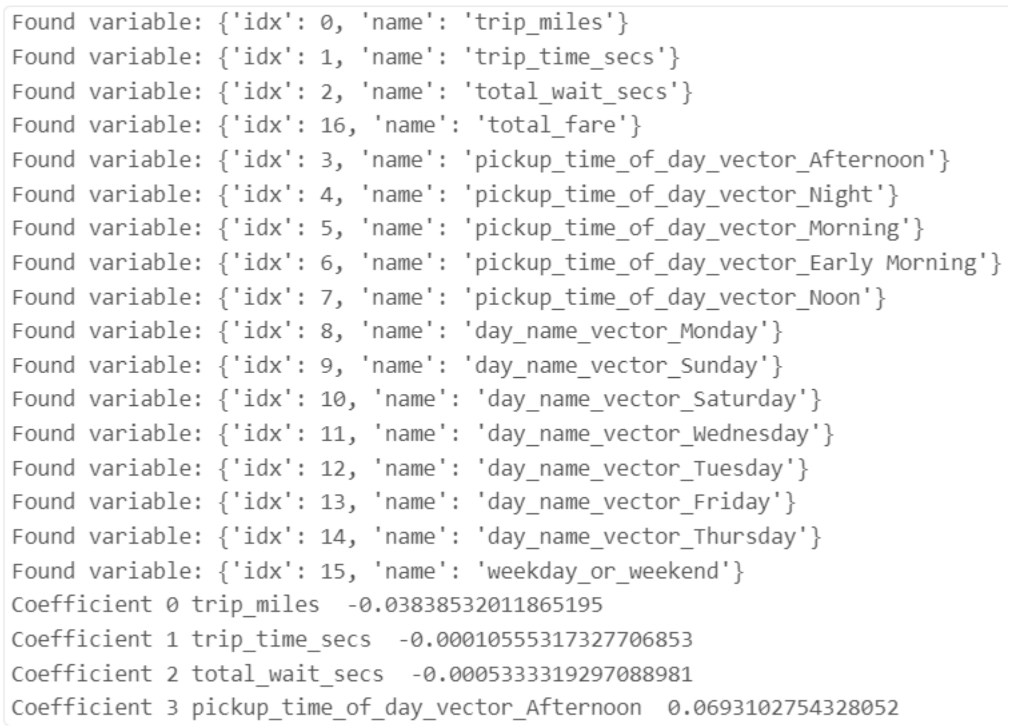
*Please see my code in Appendix 4: Milestone 5 – Data Visualization Code

After getting my best model from the previous milestone, I have yet to evaluate the best model. So, I ran some code to display the parameters for my best model, which was evaluated from the grid search. The results of my best model parameters are shown below, as well as the ROC curve from stage 4 of the pipeline. A good ROC curve would be more curved towards the top left of the graph to indicate a high true positive rate, but since mine is fairly straight, my model has a low true positive rate.

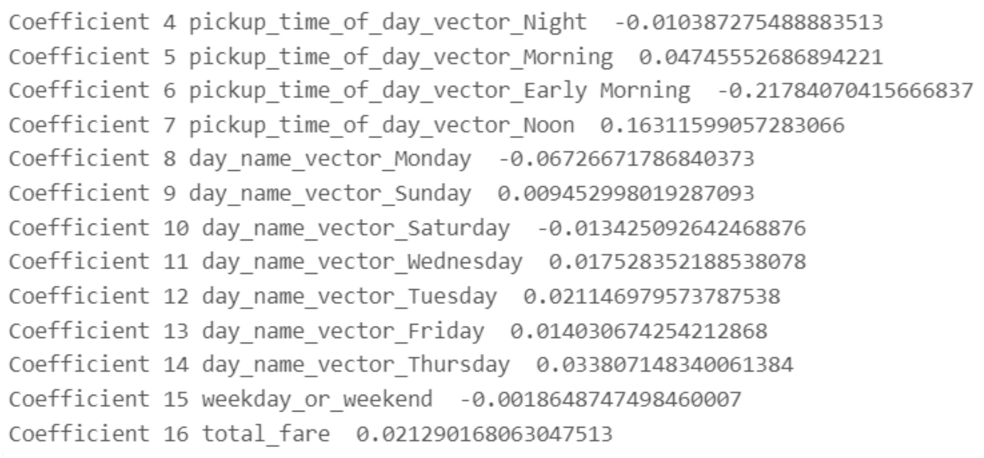
```
LogisticRegression_af08595af35e__aggregationDepth 2
LogisticRegression_af08595af35e__elasticNetParam 1.0
LogisticRegression_af08595af35e__family auto
LogisticRegression_af08595af35e__featuresCol features
LogisticRegression_af08595af35e__fitIntercept True
LogisticRegression_af08595af35e__labelCol label
LogisticRegression_af08595af35e__maxBlockSizeInMB 0.0
LogisticRegression_af08595af35e__maxIter 100
LogisticRegression_af08595af35e__predictionCol prediction
LogisticRegression_af08595af35e__probabilityCol probability
LogisticRegression_af08595af35e__rawPredictionCol rawPrediction
LogisticRegression_af08595af35e__regParam 0.0
LogisticRegression_af08595af35e__standardization True
LogisticRegression_af08595af35e__threshold 0.5
LogisticRegression_af08595af35e__tol 1e-06
```



After showing the parameters of my best model, I extracted the coefficients of each variable.

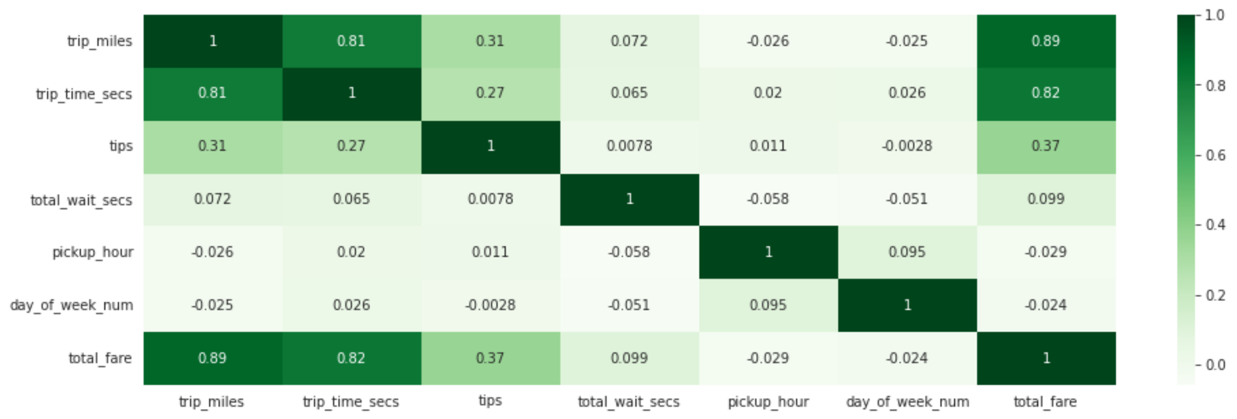


```
Found variable: {'idx': 0, 'name': 'trip_miles'}
Found variable: {'idx': 1, 'name': 'trip_time_secs'}
Found variable: {'idx': 2, 'name': 'total_wait_secs'}
Found variable: {'idx': 16, 'name': 'total_fare'}
Found variable: {'idx': 3, 'name': 'pickup_time_of_day_vector_Afternoon'}
Found variable: {'idx': 4, 'name': 'pickup_time_of_day_vector_Night'}
Found variable: {'idx': 5, 'name': 'pickup_time_of_day_vector_Morning'}
Found variable: {'idx': 6, 'name': 'pickup_time_of_day_vector_Early Morning'}
Found variable: {'idx': 7, 'name': 'pickup_time_of_day_vector_Noon'}
Found variable: {'idx': 8, 'name': 'day_name_vector_Monday'}
Found variable: {'idx': 9, 'name': 'day_name_vector_Sunday'}
Found variable: {'idx': 10, 'name': 'day_name_vector_Saturday'}
Found variable: {'idx': 11, 'name': 'day_name_vector_Wednesday'}
Found variable: {'idx': 12, 'name': 'day_name_vector_Tuesday'}
Found variable: {'idx': 13, 'name': 'day_name_vector_Friday'}
Found variable: {'idx': 14, 'name': 'day_name_vector_Thursday'}
Found variable: {'idx': 15, 'name': 'weekday_or_weekend'}
Coefficient 0 trip_miles  -0.03838532011865195
Coefficient 1 trip_time_secs  -0.00010555317327706853
Coefficient 2 total_wait_secs  -0.0005333319297088981
Coefficient 3 pickup_time_of_day_vector_Afternoon  0.0693102754328052
```

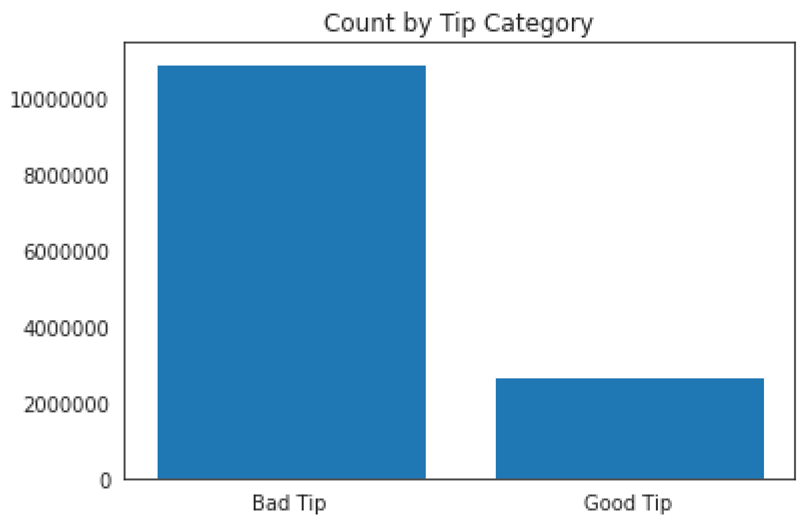


```
Coefficient 4 pickup_time_of_day_vector_Night  -0.010387275488883513
Coefficient 5 pickup_time_of_day_vector_Morning  0.04745552686894221
Coefficient 6 pickup_time_of_day_vector_Early Morning  -0.21784070415666837
Coefficient 7 pickup_time_of_day_vector_Noon  0.16311599057283066
Coefficient 8 day_name_vector_Monday  -0.06726671786840373
Coefficient 9 day_name_vector_Sunday  0.009452998019287093
Coefficient 10 day_name_vector_Saturday  -0.013425092642468876
Coefficient 11 day_name_vector_Wednesday  0.017528352188538078
Coefficient 12 day_name_vector_Tuesday  0.021146979573787538
Coefficient 13 day_name_vector_Friday  0.014030674254212868
Coefficient 14 day_name_vector_Thursday  0.033807148340061384
Coefficient 15 weekday_or_weekend  -0.0018648747498460007
Coefficient 16 total_fare  0.02129016806304751З
```

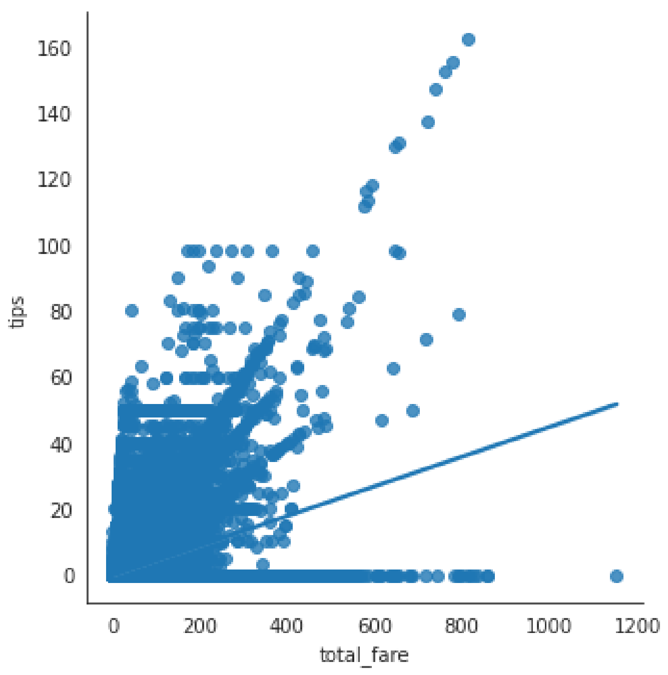
In the correlation matrix, it is a representation of how related all of my numeric columns are. Based on my result, ‘trip_miles’ and ‘trip_time_secs’ have a strong correlation of 0.81, ‘total_fare’ and ‘trip_time_secs’ have a strong correlation of 0.82, and ‘total_fare’ and ‘trip_miles’ have the strongest correlation of 0.89. These strong correlations seem reasonable because trip miles, trip time, and total fare all contribute to one another. For example, the greater the trip time or trip miles, the greater the total fare. Other than these three correlations, the remaining variables are not very correlated, which is good for my model.



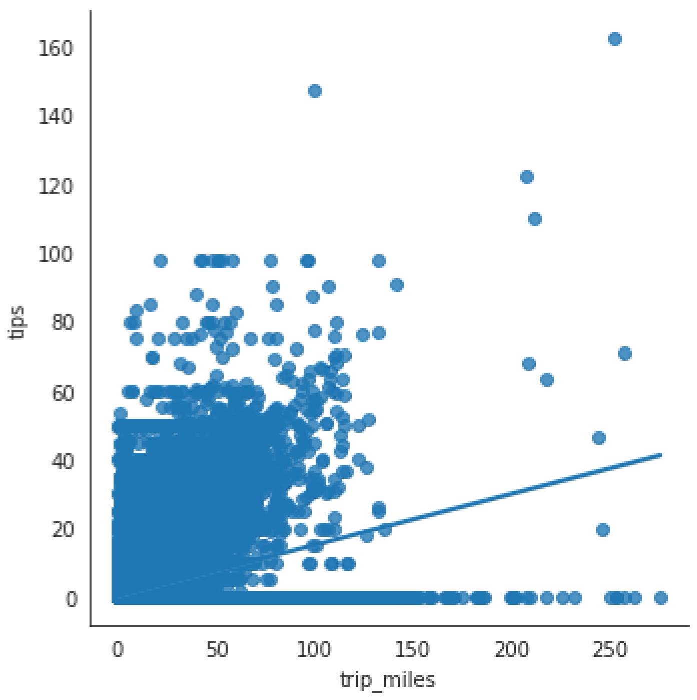
For my first visualization, I am showing my 'label' column, which is a 1.0 if the tip percent is > 0.01, and 0.0 if tip is < 0.01. I am using a bar chart to represent this because it is easy to see the distribution of the categories. Based on the bar chart, the majority of the tips are in the bad tip category, meaning that most passengers do not give a tip at all.



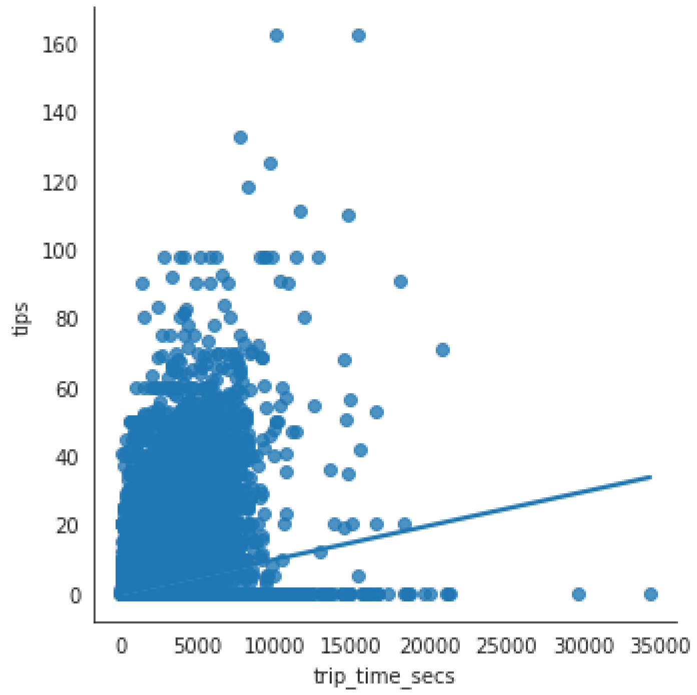
For my second visualization, I am using a relationship plot to show how the 'total_fare' and 'tips' columns relate to each other. The 'total_fare' is on the x-axis and 'tips' is on the y-axis because I am seeing how tips are impacted by total fare. This graph shows a positive correlation between total fare and tips, which makes sense because the greater the total fare, the more tip passengers typically give. Since the points are growing towards the top right of the graph, this shows a positive correlation. However, it is expected that the majority of points are concentrated towards the bottom left because passengers typically do not give that large of a tip. Furthermore, total fares are usually not very expensive either.



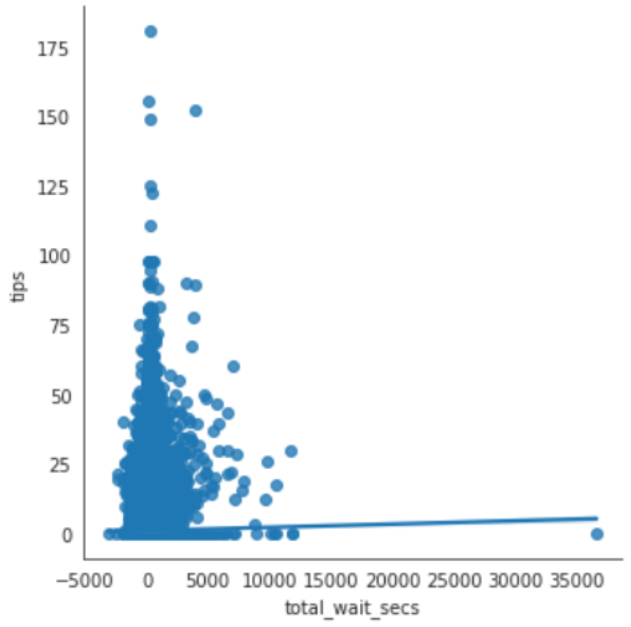
For my third visualization, I am using a relationship plot to show the relationship between ‘trip_miles’ and ‘tips’. This graph looks similar to the total fare and tips graph above. Similar to the above graph, the greater the trip miles, the more tips a passenger would usually give, which indicates a positive correlation. The same observations noted above also apply to this plot.



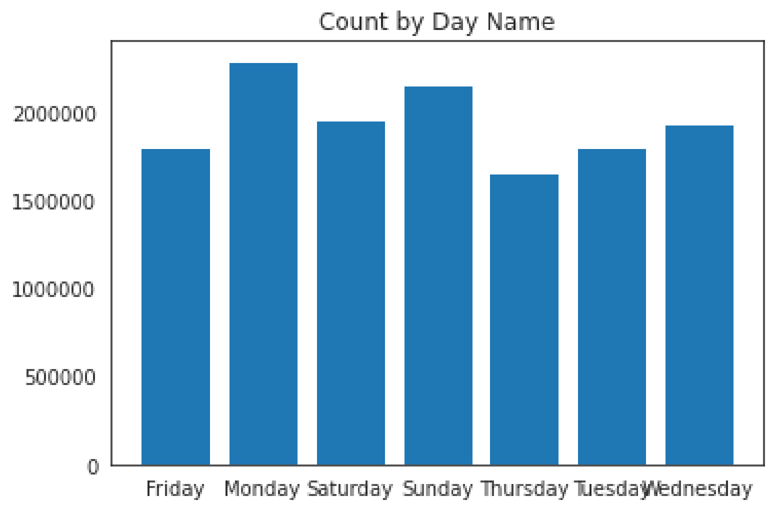
For my fourth visualization, I am using a relationship plot to show the relationship between ‘trip_time_secs’ and ‘tips’. This plot shows a positive correlation because usually, as the trip time increases, this indicates a longer ride, so more tips should be given. The same observations as the previous two plots apply here as well.



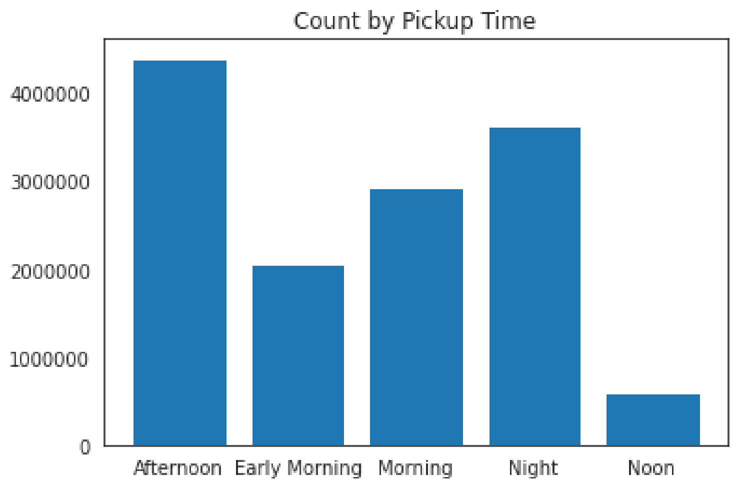
For my fifth visualization, I am showing the relationship between ‘total_wait_secs’ and ‘tips’ because I want to see if the amount of time a passenger waits for the for-hire vehicle after the request is input, impacts the amount of tips they give. Based on the relationship plot, most of the wait times are around 0 seconds, and show high tip amounts. This shows that shorter wait times lead to higher tip amounts, and therefore, the wait time does impact the tip amount.



For my sixth visualization, I am using a bar chart to visualize the ‘day_name’ column. This helps me get a sense of which days are more popular for passengers to ride in a for-hire vehicle. Overall, all of the days get a fair amount of ridership. However, Sunday and Monday get the most ridership. Even though Monday has the greatest ridership, the lowest ridership day is Thursday, but it is not too far behind, which shows that for-hire vehicles are ridden a fair amount across all days of the week.



For my seventh visualization, I am using a bar chart to show the ‘pickup_time_of_day’ column and what time of day is most popular for people to take a for-hire vehicle. Based on the results, people ride for-hire vehicles the most during the afternoon. This is reasonable because this is the most active time of day when people are up and around. The noon category is low as expected because it is not too often that riders take a for-hire vehicle at exactly 12pm. I also expect night to be the second highest because people usually take for-hire vehicles to go home when they are out late, but do not have their own car. The early morning and morning categories are third and fourth most common, which makes sense as well because people take for-hire vehicles to get to work, but there are also people who use other modes of transportation such as the MTA buses and subways, which makes these categories lower.



Milestone 6 – Summary and Conclusions

Completed Data Processing Pipeline:

My completed data processing pipeline consisted of multiple steps using AWS, Jupyter Notebook, and Databricks.

The first step was done in AWS to curl my hvfhv data from the NYC .gov website into my S3 bucket, which was done in an EC2 instance I created. To separate my data during the different milestones, the newly curled data was put into the landing folder.

The second step was to perform some exploratory data analysis to get a sense of my data and understand how many null values I had, how many columns and rows I had, and see the descriptive statistics for each column. This milestone was performed in Jupyter Notebook, but when completing this step, I found that it took a very long time to complete just one command. Often, my notebook would crash and stop, which required me to connect to my EC2 instance again.

Once I learned about my data, the third step was to clean my data. I used Databricks which proved to be much faster than Jupyter Notebook. During this step, I removed all null rows and dropped all of the columns that I did not need for my feature engineering step. So, after completing the data cleaning, I performed feature engineering, which extracted new features from the original columns. This was very helpful because I was finally able to see where my pipeline was going, and understand which features I needed to predict the tip category. Once I had all of my features engineered, I used VectorAssembler to assemble my pipeline. The following steps consisted of splitting my data into the 70% training set and 30% testing set. I decided to use a logistic regression model because I was predicting a binary category, and not the actual tip amounts. Furthermore, to learn about the performance of my model, I did multiple tests to determine the accuracy, precision, recall, f1-score, evaluation of predictions, and the ROC curve. Through these steps, I found that my model was not the best because these values were all relatively low, and therefore, not that great at making correct predictions.

In the final step, I created multiple visualizations to show my model results and see the relationship between different columns. I generated many scatter plots that included tips, and in all of these graphs, it was found that the wait time after a request is input to when the passenger is picked up, ride time, number of miles, and total fare of the trip directly relate to the amount of tip a passenger gives. These all showed a positive correlation. One way I could have improved my visualizations was to remove outliers because in a few of the graphs, they showed tip amounts greater than $100. Generally, the typical amount of tips given was far less than $100 because it is reasonable to assume that passengers do not give that much tip since the rides are not too expensive. The most common tip percentages usually fall between 10% and 20%. By the end of this step and going through all of the feature engineering, modeling, and evaluation steps, I have learned a great amount about my data. In addition, going through this entire project process has allowed me to get a sense of how machine learning pipelines work and how they are built.

Summary and Main Conclusions:

This project has taught me the process from start to finish of downloading data into a cloud computing resource to creating a machine learning model that could make predictions on data. By going through each step and taking the time to understand what I was doing at each command, I was learning the details of what it takes to build a machine learning model. Although my model was not the best at making predictions, it has shown me what needs to be improved and what it takes to build a good model. The main conclusions that I drew from the data were that there are a variety of factors that impact the amount of tips given. Since I made my tip category threshold 1% for a good tip, and most of the data showed up as less than 1%, this meant that the majority of passengers do not give any tip at all. This shocked me because I expected the results to be the opposite, where more people give some tip than none at all. Even though this project was aimed at learning about the data and model, I learned a lot about New Yorkers in general. I now understand the practices of tip giving among New Yorkers by looking at the tip amounts given after each ride. I am glad to have completed this project because I did not know that there was a way to process a very large amount of data using cloud computing resources. Furthermore, I am grateful to have been able to practice these skills using one of the most popular and useful resources out there using AWS.

Milestone 7 – Share the Project

The URL for my project on GitHub can be found here:

https://github.com/MLChow2/cis_4130_hvfhv_project.git

Appendices

Appendix 1: Milestone 2 - Data Acquisition Code

# Curl the files directly into my S3 project bucket, hvfhv-project-mc

```
curl -SL https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv_tripdata_2023-01.parquet | aws s3 cp - s3://hvfhv-project-mc/fhvhv_tripdata_2023-01.parquet

curl -SL https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv_tripdata_2023-02.parquet | aws s3 cp - s3://hvfhv-project-mc/fhvhv_tripdata_2023-02.parquet

curl -SL https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv_tripdata_2023-03.parquet | aws s3 cp - s3://hvfhv-project-mc/fhvhv_tripdata_2023-03.parquet

curl -SL https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv_tripdata_2023-04.parquet | aws s3 cp - s3://hvfhv-project-mc/fhvhv_tripdata_2023-04.parquet

curl -SL https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv_tripdata_2023-05.parquet | aws s3 cp - s3://hvfhv-project-mc/fhvhv_tripdata_2023-05.parquet

curl -SL https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv_tripdata_2023-06.parquet | aws s3 cp - s3://hvfhv-project-mc/fhvhv_tripdata_2023-06.parquet
```

# Download a copy of the 2023-01 files into my EC2 instance so I can work with it for the following milestones

```
curl -L -o fhvhv_tripdata_2023-01.parquet https://d37ci6vzurychx.cloudfront.net/trip-data/fhvhv_tripdata_2023-01.parquet
```

# Check to make sure the file was downloaded to my EC2

```
ls -l
```

Appendix 2: Milestone 3 - Exploratory Data Analysis Code

# Install python modules

```
pip3 install pandas numpy pyarrow fastparquet matplotlib
```

# Import multiple python libraries

```
import pandas as pd
from pyarrow.parquet import ParquetFile
import pyarrow as pa
import numpy as np
import matplotlib.pyplot as plt
```

# View a subset of the data from the 2023-01 parquet file and put the data into a dataframe (*Courtesy of Professor Holowczak)

# Number of rows to read from the parquet file
```
rows_to_read = 100000
```

# The file name to read
```
parquet_file_name = "fhvhv_tripdata_2023-01.parquet"
```

# Set up a pointer to the parquet file
```
pf = ParquetFile(parquet_file_name)
```

# Take a subset of the rows from the file
```
rows_subset = next(pf.iter_batches(batch_size = rows_to_read))
```

# Convert data to a Pandas dataframe
```
df = pa.Table.from_batches([rows_subset]).to_pandas()
```

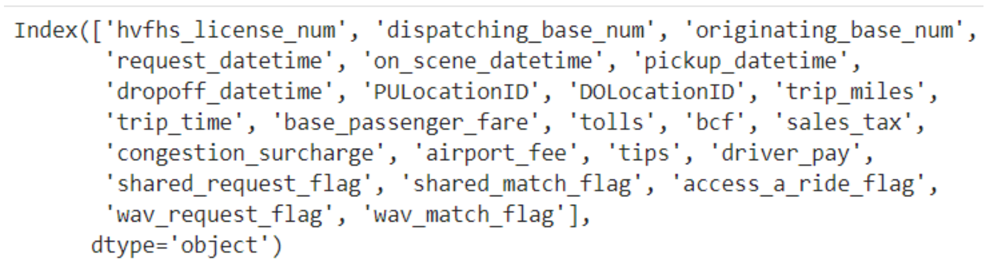
# Download the taxi zone lookup CSV file onto my EC2 instance

```
curl -L -o taxi+_zone_lookup.csv
https://d37ci6vzurychx.cloudfront.net/misc/taxi+_zone_lookup.csv
```

# Descriptive statistics techniques to learn about the data

# Show all column names

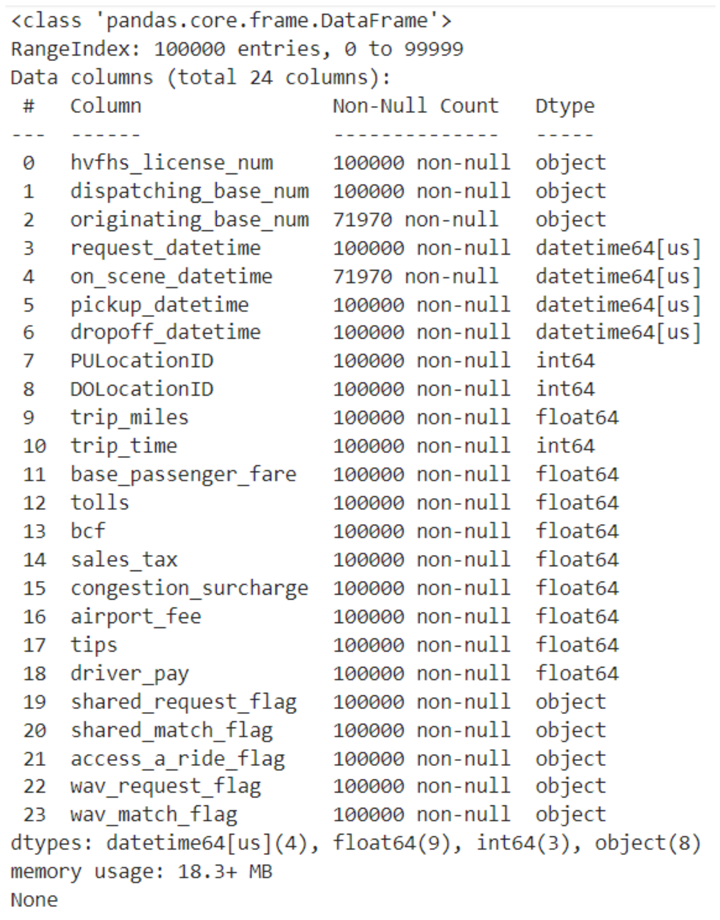
```
print(df.columns)
```



```
Index(['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'],
      dtype='object')
```

# Show the number of rows and columns
# Get information about the dataframe

```
print(df.info())
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 24 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   hvfhs_license_num     100000 non-null  object
 1   dispatching_base_num  100000 non-null  object
 2   originating_base_num  71970 non-null   object
 3   request_datetime      100000 non-null  datetime64[us]
 4   on_scene_datetime     71970 non-null   datetime64[us]
 5   pickup_datetime       100000 non-null  datetime64[us]
 6   dropoff_datetime      100000 non-null  datetime64[us]
 7   PULocationID          100000 non-null  int64
 8   DOLocationID          100000 non-null  int64
 9   trip_miles            100000 non-null  float64
 10  trip_time             100000 non-null  int64
 11  base_passenger_fare   100000 non-null  float64
 12  tolls                 100000 non-null  float64
 13  bcf                   100000 non-null  float64
 14  sales_tax             100000 non-null  float64
 15  congestion_surcharge  100000 non-null  float64
 16  airport_fee           100000 non-null  float64
 17  tips                  100000 non-null  float64
 18  driver_pay            100000 non-null  float64
 19  shared_request_flag   100000 non-null  object
 20  shared_match_flag     100000 non-null  object
 21  access_a_ride_flag    100000 non-null  object
 22  wav_request_flag      100000 non-null  object
 23  wav_match_flag        100000 non-null  object
dtypes: datetime64[us](4), float64(9), int64(3), object(8)
memory usage: 18.3+ MB
None
```

# Show the number of non-null values for each variable

```
print(df.count())
```

```
hvfhs_license_num       100000
dispatching_base_num    100000
originating_base_num     71970
request_datetime        100000
on_scene_datetime        71970
pickup_datetime         100000
dropoff_datetime        100000
PULocationID            100000
DOLocationID            100000
trip_miles              100000
trip_time               100000
base_passenger_fare     100000
tolls                   100000
bcf                     100000
sales_tax               100000
congestion_surcharge    100000
airport_fee             100000
tips                    100000
driver_pay              100000
shared_request_flag     100000
shared_match_flag       100000
access_a_ride_flag      100000
wav_request_flag        100000
wav_match_flag          100000
dtype: int64
```

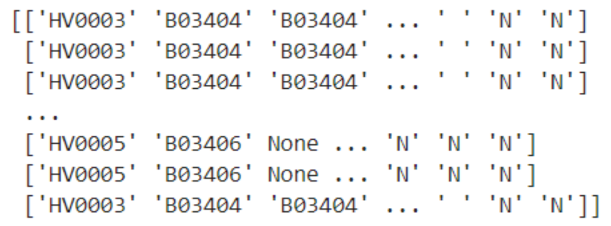
# Show the number of null values for each variable

```
print(df.isna().sum())
```

```
hvfhs_license_num           0
dispatching_base_num        0
originating_base_num    28030
request_datetime            0
on_scene_datetime       28030
pickup_datetime             0
dropoff_datetime            0
PULocationID                0
DOLocationID                0
trip_miles                  0
trip_time                   0
base_passenger_fare         0
tolls                       0
bcf                         0
sales_tax                   0
congestion_surcharge        0
airport_fee                 0
tips                        0
driver_pay                  0
shared_request_flag         0
shared_match_flag           0
access_a_ride_flag          0
wav_request_flag            0
wav_match_flag              0
dtype: int64
```

# Show the values for each row

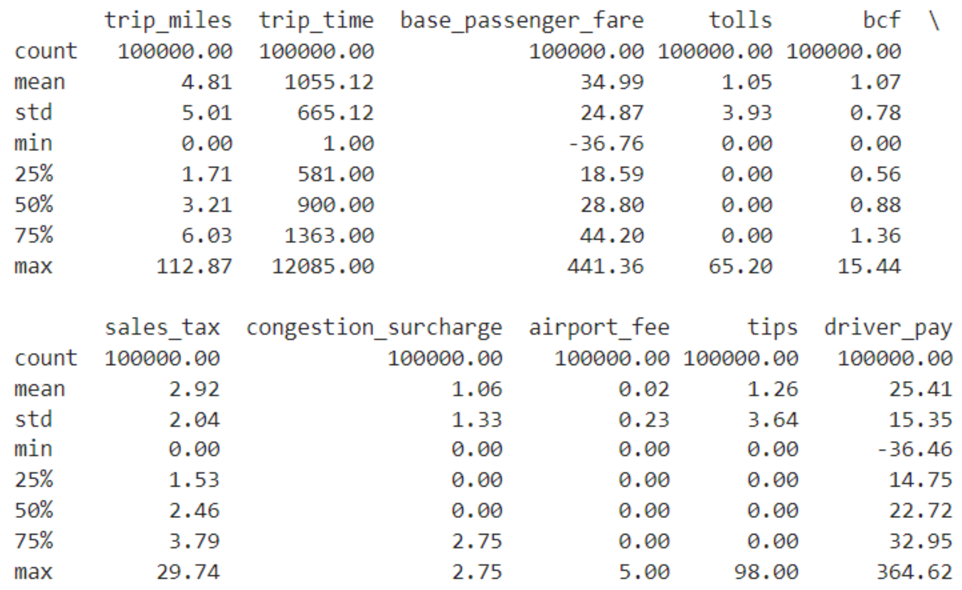
```
print(df.values)
```



```
[['HV0003' 'B03404' 'B03404' ... ' ' 'N' 'N']
 ['HV0003' 'B03404' 'B03404' ... ' ' 'N' 'N']
 ['HV0003' 'B03404' 'B03404' ... ' ' 'N' 'N']
 ...
 ['HV0005' 'B03406' None ... 'N' 'N' 'N']
 ['HV0005' 'B03406' None ... 'N' 'N' 'N']
 ['HV0003' 'B03404' 'B03404' ... ' ' 'N' 'N']]
```

# Show descriptive statistics summary of all numeric variables, rounded to 2 decimal places

```
pd.options.display.float_format = '{:.2f}'.format

print(df[['trip_miles','trip_time','base_passenger_fare','tolls'
,'bcf','sales_tax','congestion_surcharge','airport_fee','tips','
driver_pay']].describe())
```



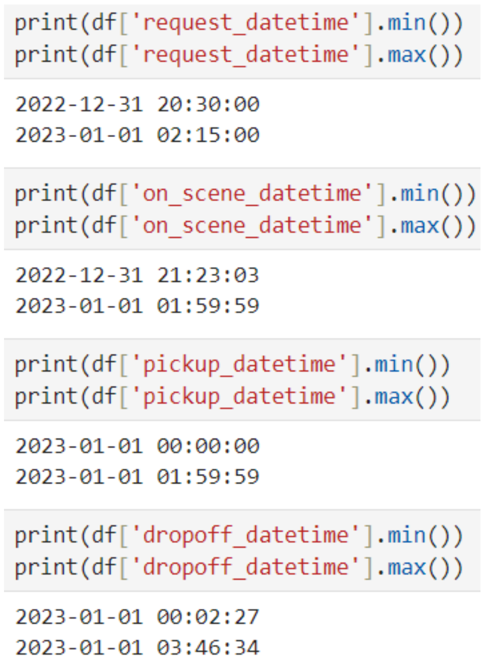
```
       trip_miles  trip_time  base_passenger_fare     tolls       bcf  \
count   100000.00  100000.00            100000.00 100000.00 100000.00
mean         4.81    1055.12                34.99      1.05      1.07
std          5.01     665.12                24.87      3.93      0.78
min          0.00       1.00               -36.76      0.00      0.00
25%          1.71     581.00                18.59      0.00      0.56
50%          3.21     900.00                28.80      0.00      0.88
75%          6.03    1363.00                44.20      0.00      1.36
max        112.87   12085.00               441.36     65.20     15.44

       sales_tax  congestion_surcharge  airport_fee      tips  driver_pay
count  100000.00             100000.00    100000.00 100000.00   100000.00
mean        2.92                  1.06         0.02      1.26       25.41
std         2.04                  1.33         0.23      3.64       15.35
min         0.00                  0.00         0.00      0.00      -36.46
25%         1.53                  0.00         0.00      0.00       14.75
50%         2.46                  0.00         0.00      0.00       22.72
75%         3.79                  2.75         0.00      0.00       32.95
max        29.74                  2.75         5.00     98.00      364.62
```

# Count the number of duplicated rows. Result was 0

```
df.duplicated().sum().sum()
```

# Min and max datetimes

```
print(df['request_datetime'].min())

print(df['request_datetime'].max())

print(df['on_scene_datetime'].min())

print(df['on_scene_datetime'].max())
print(df['pickup_datetime'].min())

print(df['pickup_datetime'].max())

print(df['dropoff_datetime'].min())

print(df['dropoff_datetime'].max())
```



```
print(df['request_datetime'].min())
print(df['request_datetime'].max())
```

```
2022-12-31 20:30:00
2023-01-01 02:15:00
```

```
print(df['on_scene_datetime'].min())
print(df['on_scene_datetime'].max())
```

```
2022-12-31 21:23:03
2023-01-01 01:59:59
```

```
print(df['pickup_datetime'].min())
print(df['pickup_datetime'].max())
```

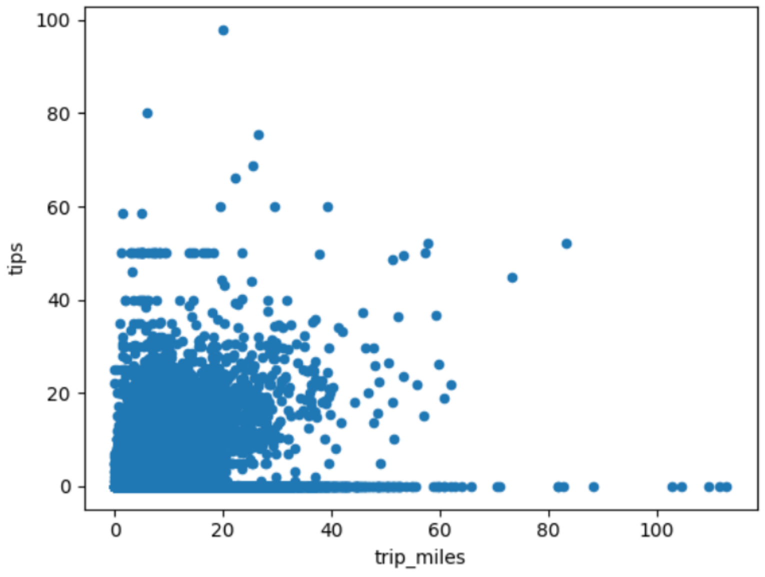
```
2023-01-01 00:00:00
2023-01-01 01:59:59
```

```
print(df['dropoff_datetime'].min())
print(df['dropoff_datetime'].max())
```

```
2023-01-01 00:02:27
2023-01-01 03:46:34
```

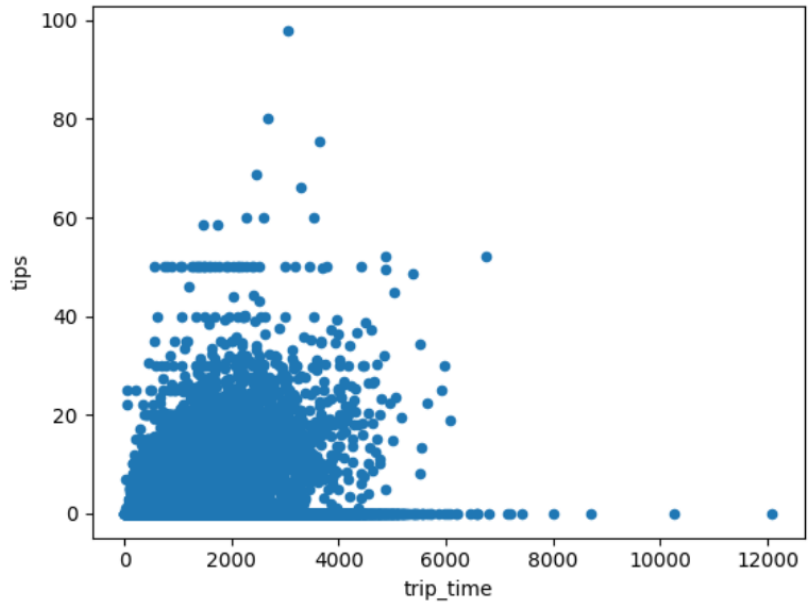
# Scatter plot showing how the number of miles impacts the tip amount

```
df.plot(kind='scatter', x='trip_miles', y='tips')

plt.show()
```



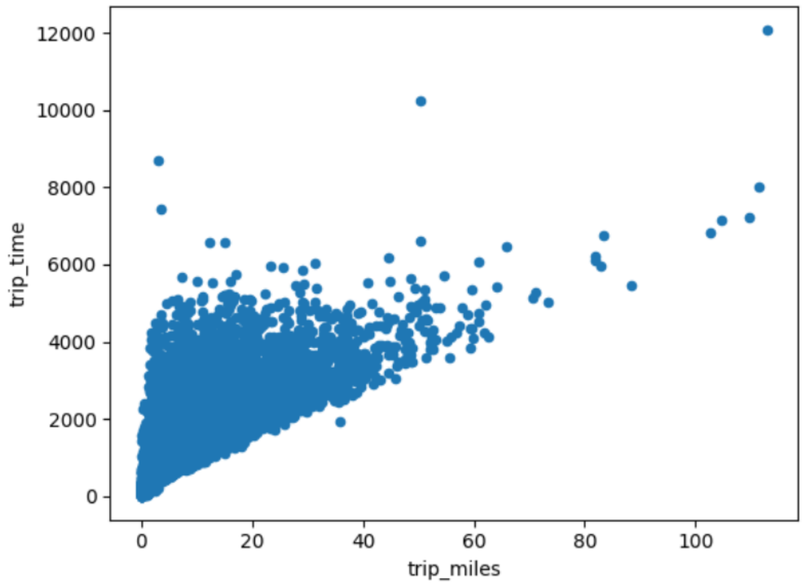
# Scatter plot showing how the amount of time impacts the tip amount

```
df.plot(kind='scatter', x='trip_time', y='tips')
plt.show()
```



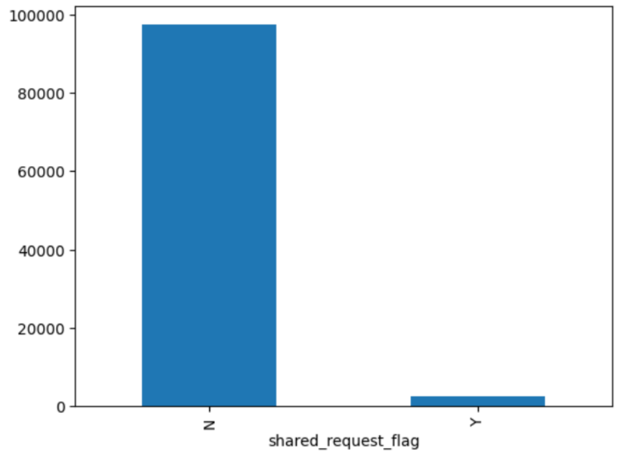
# Scatter plot showing how the number of miles impacts the trip time

```
df.plot(kind='scatter', x='trip_miles', y='trip_time')
plt.show()
```

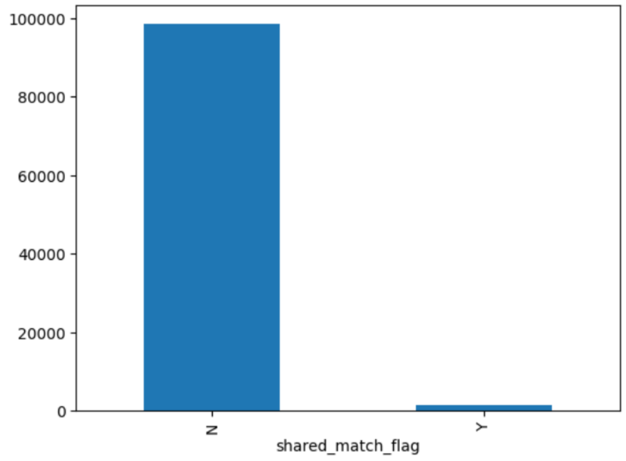


Bar graph of frequency counts for categorical variables

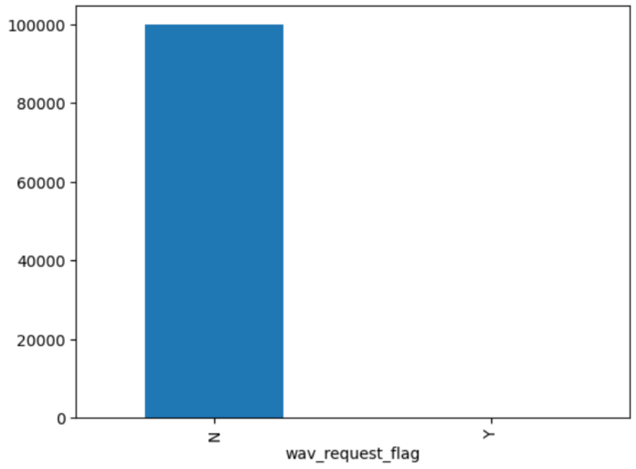
```
print(df['shared_request_flag'].value_counts().plot(kind='bar'))
```



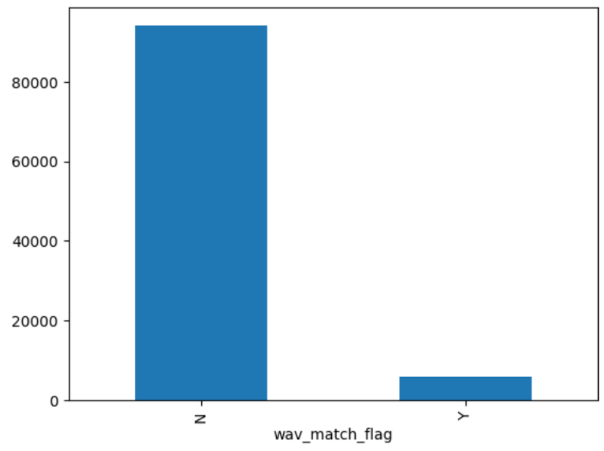
```
print(df['shared_match_flag'].value_counts().plot(kind='bar'))
```



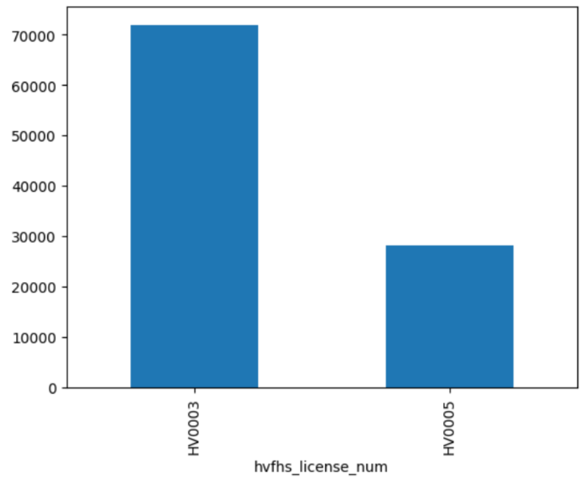
```
df['wav_request_flag'].value_counts().plot(kind='bar')
```



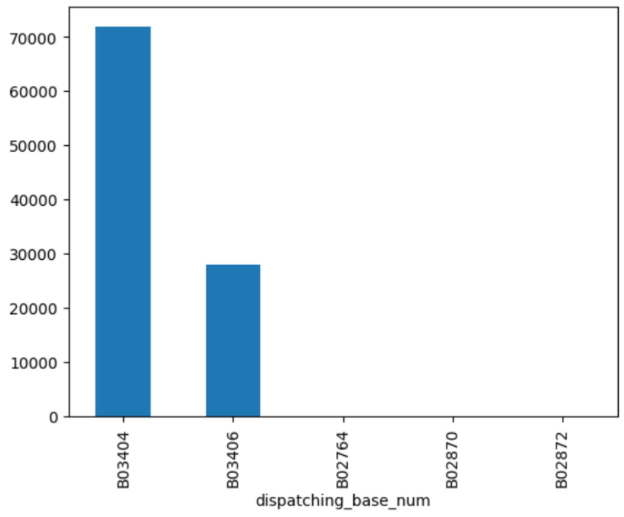
```
df['wav_match_flag'].value_counts().plot(kind='bar')
```



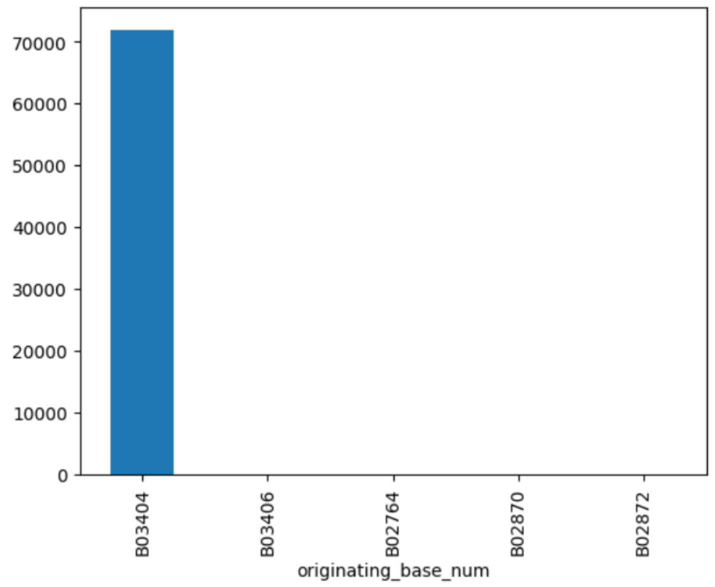
```
df['hvfhs_license_num'].value_counts().plot(kind='bar')
```



```
df['dispatching_base_num'].value_counts().plot(kind='bar')
```



```
df['originating_base_num'].value_counts().plot(kind='bar')
```



# Frequency counts of categorical variables

```
print(df.groupby('shared_request_flag').apply(len))
```

```
shared_request_flag
N    97438
Y     2562
dtype: int64
```

```
print(df.groupby('shared_match_flag').apply(len))
```

```
shared_match_flag
N    98499
Y     1501
dtype: int64
```

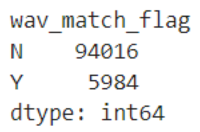
```
print(df.groupby('access_a_ride_flag').apply(len))
```

```
access_a_ride_flag
     71947
N    28053
dtype: int64
```

```
print(df.groupby('wav_request_flag').apply(len))
```

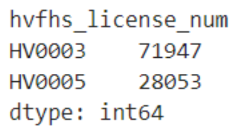
```
wav_request_flag
N    99949
Y       51
dtype: int64
```

```
print(df.groupby('wav_match_flag').apply(len))
```



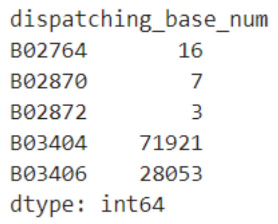
```
wav_match_flag
N    94016
Y     5984
dtype: int64
```

```
print(df.groupby('hvfhs_license_num').apply(len))
```



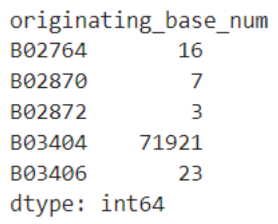
```
hvfhs_license_num
HV0003    71947
HV0005    28053
dtype: int64
```

```
print(df.groupby('dispatching_base_num').apply(len))
```



```
dispatching_base_num
B02764       16
B02870        7
B02872        3
B03404    71921
B03406    28053
dtype: int64
```

```
print(df.groupby('originating_base_num').apply(len))
```



```
originating_base_num
B02764       16
B02870        7
B02872        3
B03404    71921
B03406       23
dtype: int64
```

<u>Appendix 3: Milestone 4 – Feature Engineering and Modeling Code</u>

Appendix 3-1: Read data from my landing folder

(*Courtesy of Professor Holowczak)

```
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 = 'xxxxxxxxxxxxxx'

secret_key = 'xxxxxxxxxxxxxxxxxxxxxxxxxx'
```

# 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"
```

# Update the Spark options to work with our AWS Credentials

```
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")
```

# Read my parquet file from my landing folder

```
sdf = spark.read.parquet('s3a://hvfhv-project-
mc/landing/fhvhv_tripdata_2023-01.parquet')
```

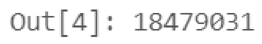
Appendix 3-2: Data Cleaning Code

# Import some functions we will need later on

```
from pyspark.sql.functions import *
```

# Count the number of rows in my file

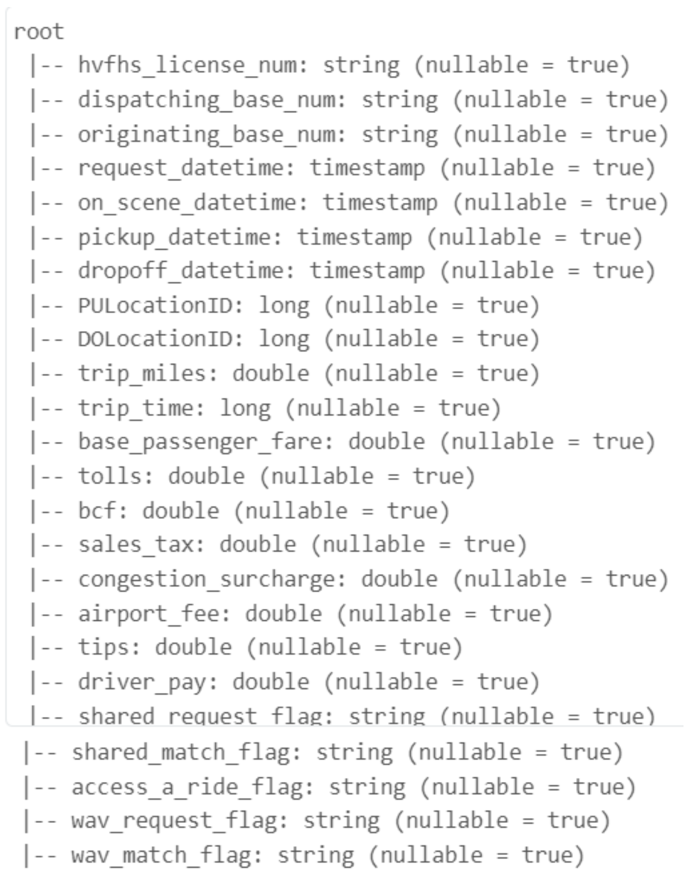
```
sdf.count()
```



```
Out[4]: 18479031
```

# Print the schema

```
sdf.printSchema()
```



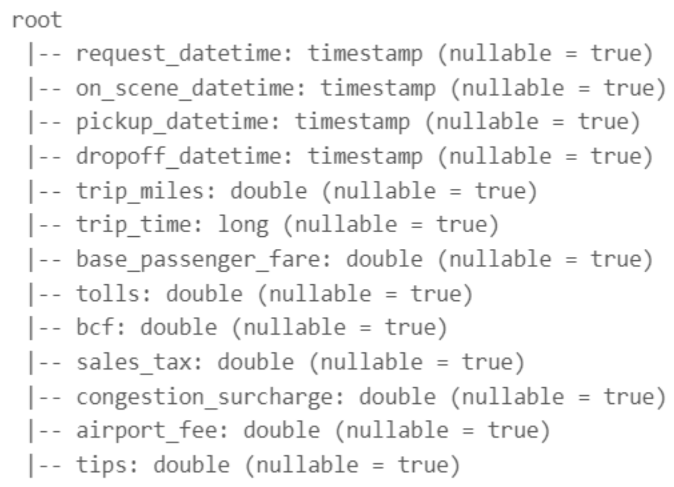
```
root
 |-- hvfhs_license_num: string (nullable = true)
 |-- dispatching_base_num: string (nullable = true)
 |-- originating_base_num: string (nullable = true)
 |-- request_datetime: timestamp (nullable = true)
 |-- on_scene_datetime: timestamp (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- PULocationID: long (nullable = true)
 |-- DOLocationID: long (nullable = true)
 |-- trip_miles: double (nullable = true)
 |-- trip_time: long (nullable = true)
 |-- base_passenger_fare: double (nullable = true)
 |-- tolls: double (nullable = true)
 |-- bcf: double (nullable = true)
 |-- sales_tax: double (nullable = true)
 |-- congestion_surcharge: double (nullable = true)
 |-- airport_fee: double (nullable = true)
 |-- tips: double (nullable = true)
 |-- driver_pay: double (nullable = true)
 |-- shared_request_flag: string (nullable = true)
 |-- shared_match_flag: string (nullable = true)
 |-- access_a_ride_flag: string (nullable = true)
 |-- wav_request_flag: string (nullable = true)
 |-- wav_match_flag: string (nullable = true)
```

# Drop all unnecessary columns, and call this dropped_cols_sdf

```
dropped_cols_sdf = sdf.drop('hvfhs_license_num',
'dispatching_base_num', 'originating_base_num', 'PULocationID',
'DOLocationID', ‘driver_pay’, 'shared_request_flag',
'shared_match_flag', 'wav_request_flag', 'wav_match_flag',
'access_a_ride_flag')
```

# Check that the columns have been dropped

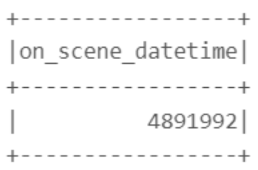
```
dropped_cols_sdf.printSchema()
```



```
root
 |-- request_datetime: timestamp (nullable = true)
 |-- on_scene_datetime: timestamp (nullable = true)
 |-- pickup_datetime: timestamp (nullable = true)
 |-- dropoff_datetime: timestamp (nullable = true)
 |-- trip_miles: double (nullable = true)
 |-- trip_time: long (nullable = true)
 |-- base_passenger_fare: double (nullable = true)
 |-- tolls: double (nullable = true)
 |-- bcf: double (nullable = true)
 |-- sales_tax: double (nullable = true)
 |-- congestion_surcharge: double (nullable = true)
 |-- airport_fee: double (nullable = true)
 |-- tips: double (nullable = true)
```

# Check how many records are null in the "on_scene_datetime" column

```
dropped_cols_sdf.select([count(when(col(c).isNull(), 
c)).alias(c) for c in ["on_scene_datetime"] ] ).show()
```



```
+-----------------+
|on_scene_datetime|
+-----------------+
|          4891992|
+-----------------+
```

# Drop the null records in the "on_scene_datetime" column, and call this clean_sdf

```
clean_sdf = 
dropped_cols_sdf.na.drop(subset=["on_scene_datetime"])
```

# Save the cleaned dataframe as a parquet file, and put it in the raw folder

```
clean_sdf.write.parquet('s3://hvfhv-project-
mc/raw/cleaned_fhvhv_tripdata_2023-01.parquet')
```

Appendix 3-3: Read clean data from my raw folder

(*Courtesy of Professor Holowczak)

```
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 = 'xxxxxxxxxxxxxx'
```

```
secret_key = 'xxxxxxxxxxxxxxxxxxxxxxxxxx'
```

# 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"
```

# Update the Spark options to work with our AWS Credentials

```
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")
```

# Read my cleaned parquet file from my raw folder

```
clean_sdf = spark.read.parquet('s3a://hvfhv-project-
mc/raw/cleaned_fhvhv_tripdata_2023-01.parquet')
```

Appendix 3-4: Feature Engineering Code

```
from pyspark.sql.functions import *
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
VectorAssembler, Binarizer, Bucketizer, MinMaxScaler
from pyspark.ml.stat import Correlation, ChiSquareTest,
Summarizer
from pyspark.ml import Pipeline
```

# Import the logistic regression model

```
from pyspark.ml.classification import LogisticRegression,
LogisticRegressionModel
```

# Import the evaluation module

```
from pyspark.ml.evaluation import *
```

# Import the model tuning module

```
from pyspark.ml.tuning import *
```

# Import other modeling modules

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

# Set the Spark logging level to only show errors

```
sc.setLogLevel("ERROR")
```

# Rename trip_time to trip_time_secs
# Convert trip_time_secs from long to double

```
trip_time_sec = clean_sdf.withColumnRenamed('trip_time',
'trip_time_secs')

trip_time_sec_sdf = trip_time_sec.withColumn('trip_time_secs',
col('trip_time_secs').cast('double'))
```

# Subtract request_datetime from pickup_datetime to get total wait time from after request was submitted to pickup time, in seconds. Convert to double.

```
total_wait_secs_sdf =
trip_time_sec_sdf.withColumn('total_wait_secs',
col('pickup_datetime').cast('double') -
col('request_datetime').cast('double'))
```

# Extract the hour from pickup_datetime to know what time of day the ride occurred. Convert to double.

```
pickup_hour_sdf = total_wait_secs_sdf.withColumn('pickup_hour',
hour(col('pickup_datetime')).cast('double'))
```

# Bucketize pickup_hour by 6-hour periods: <=6 = Early Morning, <12 = Morning, ==12 = Noon, <=18 = Afternoon, <=24 = Night

```
bucketized_pickup_hour_sdf =
pickup_hour_sdf.withColumn('pickup_time_of_day',
when(col('pickup_hour') <= 6, 'Early
Morning').when(col('pickup_hour') < 12,
"Morning").when(col('pickup_hour') == 12,
'Noon').when(col('pickup_hour') <= 18,
"Afternoon").when(col('pickup_hour') <= 24, "Night"))
```

# Extract the day of the week from pickup_datetime to know what day the ride occurred. Convert to double.

```
day_of_week_num_sdf =
bucketized_pickup_hour_sdf.withColumn('day_of_week_num',
dayofweek(col('pickup_datetime')).cast('double'))
```

# Create a column called day_name to specify the day name associated with day_of_week_num

```
bucketized_day_name_sdf =
day_of_week_num_sdf.withColumn('day_name',
when(col('day_of_week_num') == 1,
'Monday').when(col('day_of_week_num') == 2,
'Tuesday').when(col('day_of_week_num') == 3,
'Wednesday').when(col('day_of_week_num') == 4,
'Thursday').when(col('day_of_week_num') == 5,
'Friday').when(col('day_of_week_num') == 6,
'Saturday').when(col('day_of_week_num') == 7, 'Sunday'))
```

# Binarize day_of_week_num, where 0.0 = Weekday, 1.0 = Weekend

```
binarizer_day = Binarizer(threshold=5.0,
inputCol='day_of_week_num', outputCol='weekday_or_weekend')

weekday_or_weekend_sdf =
binarizer_day.transform(bucketized_day_name_sdf)
```

# Create a column called total_fare that includes all of the ride costs

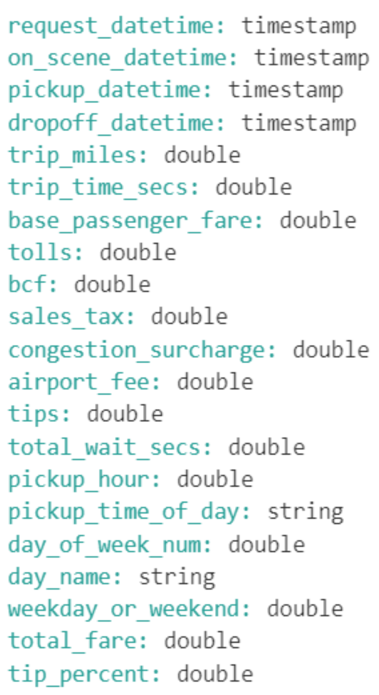
```
total_fare_sdf = weekday_or_weekend_sdf.withColumn('total_fare',
col('base_passenger_fare') + col('tolls') + col('bcf') +
col('sales_tax') + col('congestion_surcharge') +
col('airport_fee'))
```

# Drop all rows where total_fare is equal to 0.0 to avoid a divide by zero error
# Create a column called tip_percent that is tips divided by total_fare to get the tip percent

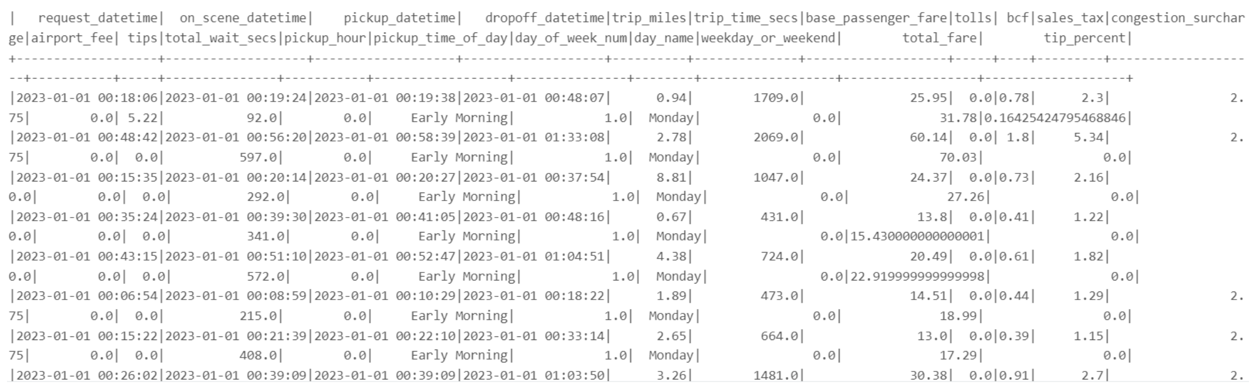
```
non_zero_fare = total_fare_sdf.where(col('total_fare') > 0.0)

tip_percent_sdf = non_zero_fare.withColumn('tip_percent',
col('tips') / col('total_fare'))
```

# Display the dataframe with the new features

```
tip_percent_sdf.show()
```



```
request_datetime: timestamp
on_scene_datetime: timestamp
pickup_datetime: timestamp
dropoff_datetime: timestamp
trip_miles: double
trip_time_secs: double
base_passenger_fare: double
tolls: double
bcf: double
sales_tax: double
congestion_surcharge: double
airport_fee: double
tips: double
total_wait_secs: double
pickup_hour: double
pickup_time_of_day: string
day_of_week_num: double
day_name: string
weekday_or_weekend: double
total_fare: double
tip_percent: double
```



```
|   request_datetime|  on_scene_datetime|    pickup_datetime|   dropoff_datetime|trip_miles|trip_time_secs|base_passenger_fare|tolls| bcf|sales_tax|congestion_surchar
ge|airport_fee| tips|total_wait_secs|pickup_hour|pickup_time_of_day|day_of_week_num|day_name|weekday_or_weekend|        total_fare|        tip_percent|
+-------------------+-------------------+-------------------+-------------------+----------+--------------+-------------------+-----+----+---------+------------------
--+-----------+-----+---------------+-----------+------------------+---------------+--------+------------------+------------------+-------------------+
|2023-01-01 00:18:06|2023-01-01 00:19:24|2023-01-01 00:19:38|2023-01-01 00:48:07|      0.94|        1709.0|              25.95|  0.0|0.78|      2.3|                2.
75|        0.0| 5.22|           92.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             31.78|0.16425424795468846|
|2023-01-01 00:48:42|2023-01-01 00:56:20|2023-01-01 00:58:39|2023-01-01 01:33:08|      2.78|        2069.0|              60.14|  0.0| 1.8|     5.34|                2.
75|        0.0|  0.0|          597.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             70.03|                0.0|
|2023-01-01 00:15:35|2023-01-01 00:20:14|2023-01-01 00:20:27|2023-01-01 00:37:54|      8.81|        1047.0|              24.37|  0.0|0.73|     2.16|                 
0.0|        0.0|  0.0|          292.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             27.26|                0.0|
|2023-01-01 00:35:24|2023-01-01 00:39:30|2023-01-01 00:41:05|2023-01-01 00:48:16|      0.67|         431.0|               13.8|  0.0|0.41|     1.22|                 
0.0|        0.0|  0.0|          341.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|15.430000000000001|                0.0|
|2023-01-01 00:43:15|2023-01-01 00:51:10|2023-01-01 00:52:47|2023-01-01 01:04:51|      4.38|         724.0|              20.49|  0.0|0.61|     1.82|                 
0.0|        0.0|  0.0|          572.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|22.919999999999998|                0.0|
|2023-01-01 00:06:54|2023-01-01 00:08:59|2023-01-01 00:10:29|2023-01-01 00:18:22|      1.89|         473.0|              14.51|  0.0|0.44|     1.29|                2.
75|        0.0|  0.0|          215.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             18.99|                0.0|
|2023-01-01 00:15:22|2023-01-01 00:21:39|2023-01-01 00:22:10|2023-01-01 00:33:14|      2.65|         664.0|               13.0|  0.0|0.39|     1.15|                2.
75|        0.0|  0.0|          408.0|        0.0|     Early Morning|            1.0|  Monday|               0.0|             17.29|                0.0|
|2023-01-01 00:26:02|2023-01-01 00:39:09|2023-01-01 00:39:09|2023-01-01 01:03:50|      3.26|        1481.0|              30.38|  0.0|0.91|      2.7|                2.
```

Appendix 3-5: Pipeline Creation Code

# Create a label where = 1 if > 0.01, = 0 otherwise

```
binarizer_tip = Binarizer(threshold=0.01,
inputCol='tip_percent', outputCol='label')
tip_label = binarizer_tip.transform(tip_percent_sdf)
```

# Create an indexer for the string-based columns

```
indexer = StringIndexer(inputCols=["pickup_time_of_day",
"day_name"], outputCols=["pickup_time_of_day_index",
"day_name_index"])
```

# Create an encoder for the two indexes

```
encoder = OneHotEncoder(inputCols=['pickup_time_of_day_index',
'day_name_index'], outputCols=['pickup_time_of_day_vector',
'day_name_vector'], dropLast=False)
```

# Create an assembler for the individual feature vectors and the double columns

```
assembler = VectorAssembler(inputCols=['trip_miles',
'trip_time_secs', 'total_wait_secs',
'pickup_time_of_day_vector', 'day_name_vector',
'weekday_or_weekend', 'total_fare'], outputCol='features')
```

# Scale the features column so the min = 0.0 and max = 1.0

```
min_max_scaler = MinMaxScaler(inputCol='features',
outputCol='scaled_features')
```

# Create the pipeline

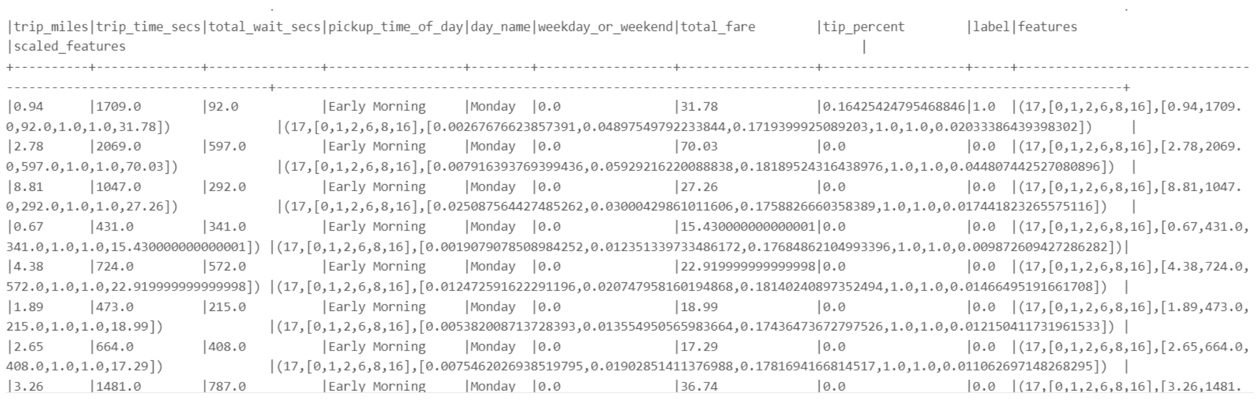
```
hvfhv_pipe = Pipeline(stages=[indexer, encoder, assembler,
min_max_scaler])
```

# Call .fit() to transform the data

```
transformed_sdf = hvfhv_pipe.fit(tip_label).transform(tip_label)
```

# Review the transformed features

```
transformed_sdf.select('trip_miles', 'trip_time_secs',
'total_wait_secs', 'pickup_time_of_day', 'day_name',
'weekday_or_weekend', 'total_fare', 'tip_percent', 'label',
'features', 'scaled_features').show(truncate=False)
```



```
|trip_miles|trip_time_secs|total_wait_secs|pickup_time_of_day|day_name|weekday_or_weekend|total_fare        |tip_percent        |label|features
|scaled_features                                                                                                             |
+----------+--------------+---------------+------------------+--------+------------------+------------------+-------------------+-----+------------------------------
----------------------------------+-------------------------------------------------------------------------------------------------------------+
|0.94      |1709.0        |92.0           |Early Morning     |Monday  |0.0               |31.78             |0.16425424795468846|1.0  |(17,[0,1,2,6,8,16],[0.94,1709.
0,92.0,1.0,1.0,31.78])            |(17,[0,1,2,6,8,16],[0.00267676623857391,0.04897549792233844,0.1719399925089203,1.0,1.0,0.02033386439398302])     |
|2.78      |2069.0        |597.0          |Early Morning     |Monday  |0.0               |70.03             |0.0                |0.0  |(17,[0,1,2,6,8,16],[2.78,2069.
0,597.0,1.0,1.0,70.03])           |(17,[0,1,2,6,8,16],[0.007916393769399436,0.05929216220088838,0.18189524316438976,1.0,1.0,0.044807442527080896])  |
|8.81      |1047.0        |292.0          |Early Morning     |Monday  |0.0               |27.26             |0.0                |0.0  |(17,[0,1,2,6,8,16],[8.81,1047.
0,292.0,1.0,1.0,27.26])           |(17,[0,1,2,6,8,16],[0.025087564427485262,0.03000429861011606,0.1758826660358389,1.0,1.0,0.017441823265575116])   |
|0.67      |431.0         |341.0          |Early Morning     |Monday  |0.0               |15.430000000000001|0.0                |0.0  |(17,[0,1,2,6,8,16],[0.67,431.0,
341.0,1.0,1.0,15.430000000000001]) |(17,[0,1,2,6,8,16],[0.0019079078508984252,0.012351339733486172,0.17684862104993396,1.0,1.0,0.009872609427286282])|
|4.38      |724.0         |572.0          |Early Morning     |Monday  |0.0               |22.919999999999998|0.0                |0.0  |(17,[0,1,2,6,8,16],[4.38,724.0,
572.0,1.0,1.0,22.919999999999998]) |(17,[0,1,2,6,8,16],[0.012472591622291196,0.020747958160194868,0.18140240897352494,1.0,1.0,0.01466495191661708])  |
|1.89      |473.0         |215.0          |Early Morning     |Monday  |0.0               |18.99             |0.0                |0.0  |(17,[0,1,2,6,8,16],[1.89,473.0,
215.0,1.0,1.0,18.99])             |(17,[0,1,2,6,8,16],[0.005382008713728393,0.013554950565983664,0.17436473672797526,1.0,1.0,0.012150411731961533]) |
|2.65      |664.0         |408.0          |Early Morning     |Monday  |0.0               |17.29             |0.0                |0.0  |(17,[0,1,2,6,8,16],[2.65,664.0,
408.0,1.0,1.0,17.29])             |(17,[0,1,2,6,8,16],[0.0075462026938519795,0.01902851411376988,0.1781694166814517,1.0,1.0,0.011062697148268295])  |
|3.26      |1481.0        |787.0          |Early Morning     |Monday  |0.0               |36.74             |0.0                |0.0  |(17,[0,1,2,6,8,16],[3.26,1481.
```

Appendix 3-6: Model Specification Code

# Split the data into 70% training and 30% test sets

```
trainingData, testData = transformed_sdf.randomSplit([0.7, 0.3],
seed=42)
```

# Create a LogisticRegression Estimator

```
lr = LogisticRegression()
```

# Fit the model to the training data

```
model = lr.fit(trainingData)
```

# Show model coefficients and intercept

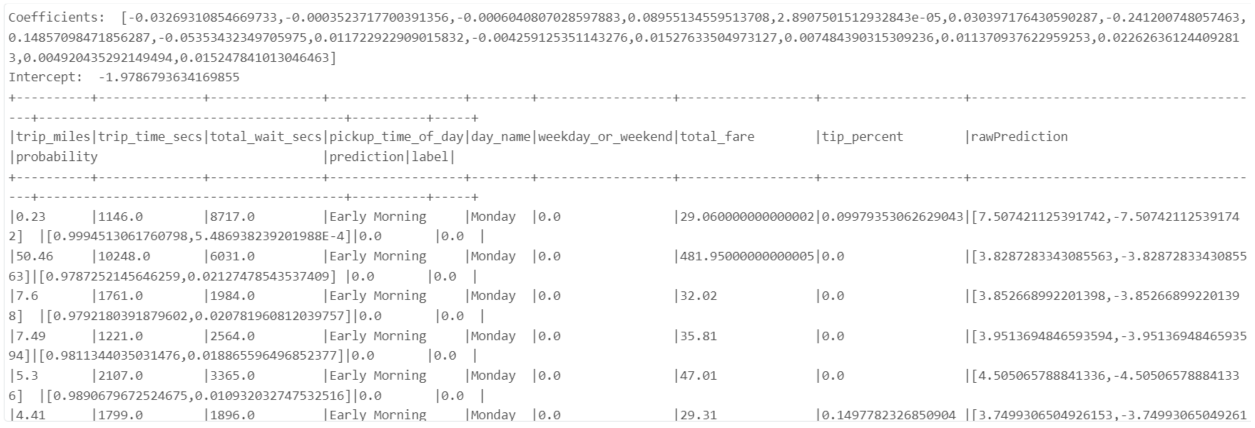
```
print("Coefficients: ", model.coefficients)
print("Intercept: ", model.intercept)
```

# Test the model on the testData

```
test_results = model.transform(testData)
```

# Show the test results

```
test_results.select('trip_miles', 'trip_time_secs',
'total_wait_secs', 'pickup_time_of_day', 'day_name',
'weekday_or_weekend', 'total_fare', 'tip_percent',
'rawPrediction','probability','prediction',
'label').show(truncate=False)
```

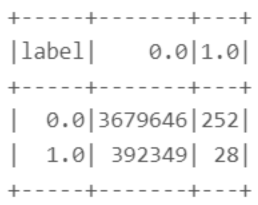


```
Coefficients:  [-0.03269310854669733,-0.0003523717700391356,-0.0006040807028597883,0.08955134559513708,2.8907501512932843e-05,0.030397176430590287,-0.241200748057463,
0.14857098471856287,-0.05353432349705975,0.011722922909015832,-0.004259125351143276,0.01527633504973127,0.007484390315309236,0.011370937622959253,0.022626361244092813,0.004920435292149494,0.015247841013046463]
Intercept:  -1.9786793634169855
+----------+--------------+---------------+------------------+--------+------------------+------------------+-------------------+----------------------------------------
---+----------------------------------------+----------+-----+
|trip_miles|trip_time_secs|total_wait_secs|pickup_time_of_day|day_name|weekday_or_weekend|total_fare        |tip_percent        |rawPrediction
|probability                              |prediction|label|
+----------+--------------+---------------+------------------+--------+------------------+------------------+-------------------+----------------------------------------
---+----------------------------------------+----------+-----+
|0.23      |1146.0        |8717.0         |Early Morning     |Monday  |0.0               |29.060000000000002|0.09979353062629043|[7.507421125391742,-7.50742112539174
2]  |[0.9994513061760798,5.486938239201988E-4]|0.0       |0.0  |
|50.46     |10248.0       |6031.0         |Early Morning     |Monday  |0.0               |481.95000000000005|0.0                |[3.8287283343085563,-3.82872833430855
63]|[0.978725214564625 9,0.02127478543537409] |0.0       |0.0  |
|7.6       |1761.0        |1984.0         |Early Morning     |Monday  |0.0               |32.02             |0.0                |[3.852668992201398,-3.85266899220139
8]  |[0.9792180391879602,0.020781960812039757]|0.0       |0.0  |
|7.49      |1221.0        |2564.0         |Early Morning     |Monday  |0.0               |35.81             |0.0                |[3.9513694846593594,-3.95136948465935
94]|[0.981134403503147 6,0.018865596496852377]|0.0       |0.0  |
|5.3       |2107.0        |3365.0         |Early Morning     |Monday  |0.0               |47.01             |0.0                |[4.505065788841336,-4.50506578884133
6]  |[0.9890679672524675,0.010932032747532516]|0.0       |0.0  |
|4.41      |1799.0        |1896.0         |Early Morning     |Monday  |0.0               |29.31             |0.1497782326850904 |[3.7499306504926153,-3.74993065049261
```

# Show the confusion matrix

```
test_results.groupby('label').pivot('prediction').count().sort('
label').show()
```

)

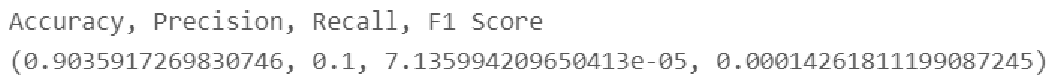


```
+-----+-------+---+
|label|    0.0|1.0|
+-----+-------+---+
|  0.0|3679646|252|
|  1.0| 392349| 28|
+-----+-------+---+
```

Appendix 3-7: Model Evaluation Code

# Calculate accuracy, precision, recall, and F1 score
# Save the confusion matrix

```
cm =
test_results.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 ) )
    return accuracy, precision, recall, f1_score

print('Accuracy, Precision, Recall, F1 Score')
print(calculate_recall_precision(cm))
```



```
Accuracy, Precision, Recall, F1 Score
(0.9035917269830746, 0.1, 7.135994209650413e-05, 0.00014261811199087245)
```

# Run cross-validator on training data
# Create a BinaryClassificationEvaluator to evaluate how well the model works

```
evaluator =
BinaryClassificationEvaluator(metricName="areaUnderROC")
```

# Create the parameter grid (empty for now)

```
grid = ParamGridBuilder().build()
```

# Create the CrossValidator

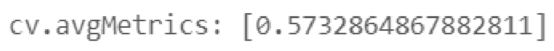
```
cv = CrossValidator(estimator=lr, estimatorParamMaps=grid,
evaluator=evaluator, numFolds=3 )
```

# Use the CrossValidator to fit the training data

```
cv = cv.fit(trainingData)
```

# Show the average performance over the three folds

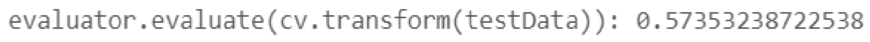
```
print('cv.avgMetrics:', cv.avgMetrics)
```



```
cv.avgMetrics: [0.5732864867882811]
```

# Evaluate the test data using the cross-validator model
# Reminder: We used Area Under the Curve

```
print('evaluator.evaluate(cv.transform(testData)):',
evaluator.evaluate(cv.transform(testData)))
```



```
evaluator.evaluate(cv.transform(testData)): 0.57353238722538
```

# Model Optimization - parameter grid search
# Create a grid to hold hyperparameters

```
grid = ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 1.0] )
grid = grid.addGrid(lr.elasticNetParam, [0, 1])
```

# Build the grid

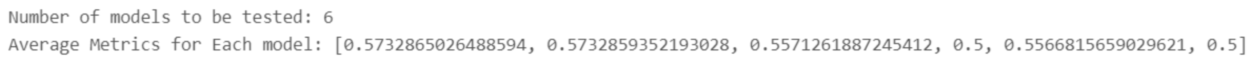
```
grid = grid.build()
print('Number of models to be tested:', len(grid))
```

# Create the CrossValidator using the new hyperparameter grid

```
cv = CrossValidator(estimator=lr, estimatorParamMaps=grid,
evaluator=evaluator)
```

# Call cv.fit() to create models with all of the combinations of parameters in the grid

```
all_models = cv.fit(trainingData)
print("Average Metrics for Each model:", all_models.avgMetrics)
```



```
Number of models to be tested: 6
Average Metrics for Each model: [0.5732865026488594, 0.5732859352193028, 0.5571261887245412, 0.5, 0.5566815659029621, 0.5]
```

# Characteristics of best model
# Gather the metrics and parameters of the model with the best average metrics

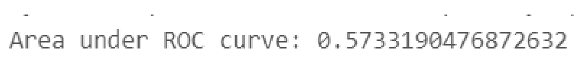
```
hyperparams =
all_models.getEstimatorParamMaps()[np.argmax(all_models.avgMetri
cs)]
```

# Print out the list of hyperparameters for the best model

```
for i in range(len(hyperparams.items())):
    print([x for x in hyperparams.items()][i])

# (Param(parent='LogisticRegression_2effdf339a6c',
name='regParam', doc='regularization parameter (>= 0).'), 0.4)
# (Param(parent='LogisticRegression_2effdf339a6c',
name='elasticNetParam', doc='the ElasticNet mixing parameter, in
range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For
alpha = 1, it is an L1 penalty.'), 0.0)
```

# Choose the best model

```
bestModel = all_models.bestModel
print("Area under ROC curve:", bestModel.summary.areaUnderROC)

# Area under ROC curve: 1.0
```



```
Area under ROC curve: 0.5733190476872632
```

# Test the best model on the test data
# Use the model 'bestModel' to predict the test set

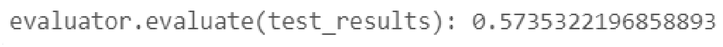
```
test_results = bestModel.transform(testData)
```

# Show the results

```
test_results.select('trip_miles', 'trip_time_secs',
'total_wait_secs', 'pickup_time_of_day', 'day_name',
'weekday_or_weekend', 'total_fare',
'tip_percent','probability','prediction',
'label').show(truncate=False)
```

# Evaluate the predictions. Area Under ROC curve

```
print('evaluator.evaluate(test_results):',
evaluator.evaluate(test_results))
```



```
evaluator.evaluate(test_results): 0.5735322196858893
```

Appendix 3-8: Save my best model and feature engineered dataframe to S3

# Save best model to models folder in my S3 bucket

```
bestModel.write().overwrite().save('s3://hvfhv-project-
mc/models/hvfhv_logistic_regression_model')
```

# Save transformed_sdf as a parquet file in my trusted folder

```
transformed_sdf.write.parquet('s3://hvfhv-project-
mc/trusted/feature_engineered_fhvhv_tripdata_2023-01.parquet')
```

<u>Appendix 4: Milestone 5 – Data Visualization Code</u>

Appendix 4-1: Read feature engineered data from my trusted folder, and my best model from my models folder

```
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 = 'xxxxxxxxxxxxxx'
secret_key = 'xxxxxxxxxxxxxxxxxxxxxxxxxx'
```

# 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"
```

# Update the Spark options to work with our AWS Credentials

```
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")

feature_engineered_sdf = spark.read.parquet('s3a://hvfhv-
project-mc/trusted/feature_engineered_fhvhv_tripdata_2023-
01.parquet')

my_model = LogisticRegressionModel.load('s3a://hvfhv-project-
mc/models/hvfhv_logistic_regression_model')
```

Appendix 4-2: Model Validation Code

```
from pyspark.sql.functions import *
from pyspark.ml.feature import StringIndexer, OneHotEncoder,
VectorAssembler, Binarizer, Bucketizer, MinMaxScaler
from pyspark.ml.stat import Correlation, ChiSquareTest,
Summarizer
from pyspark.ml import Pipeline
```

# Import the logistic regression model

```
from pyspark.ml.classification import LogisticRegression,
LogisticRegressionModel
```

# Import the evaluation module

```
from pyspark.ml.evaluation import *
```

# Import the model tuning module

```
from pyspark.ml.tuning import *
```

# Import other modeling modules

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

# Set the Spark logging level to only show errors

```
sc.setLogLevel("ERROR")
```

# Create a label. =1 if >= 0.01, =0 otherwise

```
binarizer_tip = Binarizer(threshold=0.01,
inputCol='tip_percent', outputCol='label')
tip_label = binarizer_tip.transform(tip_percent_sdf)
```

# Create an indexer for the string based columns

```
indexer = StringIndexer(inputCols=["pickup_time_of_day",
"day_name"], outputCols=["pickup_time_of_day_index",
"day_name_index"])
```

# Create an encoder for the two indexes

```
encoder = OneHotEncoder(inputCols=['pickup_time_of_day_index',
'day_name_index'], outputCols=['pickup_time_of_day_vector',
'day_name_vector'], dropLast=False)
```

# Create an assembler for the individual feature vectors and the float/double columns

```
assembler = VectorAssembler(inputCols=['trip_miles',
'trip_time_secs', 'total_wait_secs',
'pickup_time_of_day_vector', 'day_name_vector',
'weekday_or_weekend', 'total_fare'], outputCol='features')
```

# Scale the features columns so the min = 0.0 and max = 1.0

```
min_max_scaler = MinMaxScaler(inputCol='features',
outputCol='scaled_features')
```

# Create a LogisticRegression Estimator

```
lr = LogisticRegression()

hvfhv_pipe = Pipeline(stages=[indexer, encoder, assembler,
min_max_scaler, lr])
```

# Split the data into 70% training and 30% test sets

```
trainingData, testData = tip_label.randomSplit([0.7, 0.3],
seed=42)
```

# Create a grid to hold hyperparameters

```
grid = ParamGridBuilder()
grid = grid.addGrid(lr.regParam, [0.0, 1.0])
grid = grid.addGrid(lr.elasticNetParam, [0, 1])
```

# Build the parameter grid

```
grid = grid.build()
```

# How many models to be tested

```
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=hvfhv_pipe,
                    estimatorParamMaps=grid,
                    evaluator=evaluator,
                    numFolds=3)
```

# Train the models

```
cv = cv.fit(trainingData)
```

# Test the predictions

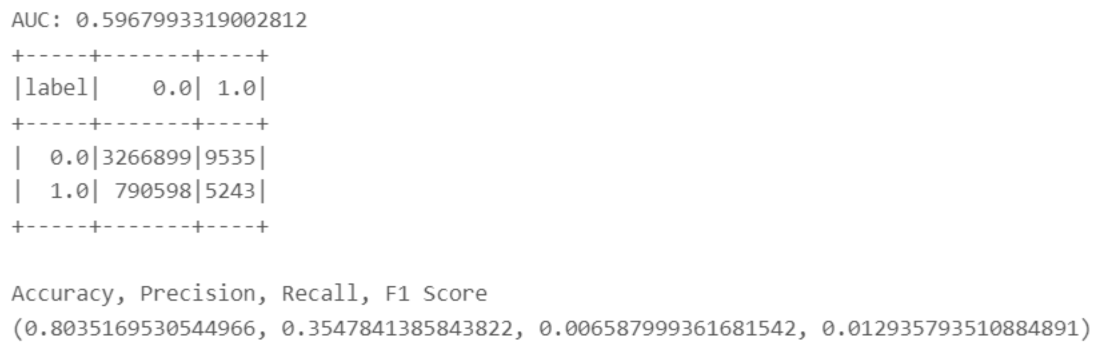
```
predictions = cv.transform(testData)
```

# Calculate AUC

```
auc = evaluator.evaluate(predictions)
print(f"AUC: {auc}")
```

# Create the confusion matrix

```
predictions.groupby('label').pivot('prediction').count().fillna(
0).show()
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 ) )
    return accuracy, precision, recall, f1_score

print("Accuracy, Precision, Recall, F1 Score")
print( calculate_recall_precision(cm) )
```



```
AUC: 0.5967993319002812
+-----+-------+----+
|label|    0.0| 1.0|
+-----+-------+----+
|  0.0|3266899|9535|
|  1.0| 790598|5243|
+-----+-------+----+

Accuracy, Precision, Recall, F1 Score
(0.8035169530544966, 0.3547841385843822, 0.006587999361681542, 0.012935793510884891)
```

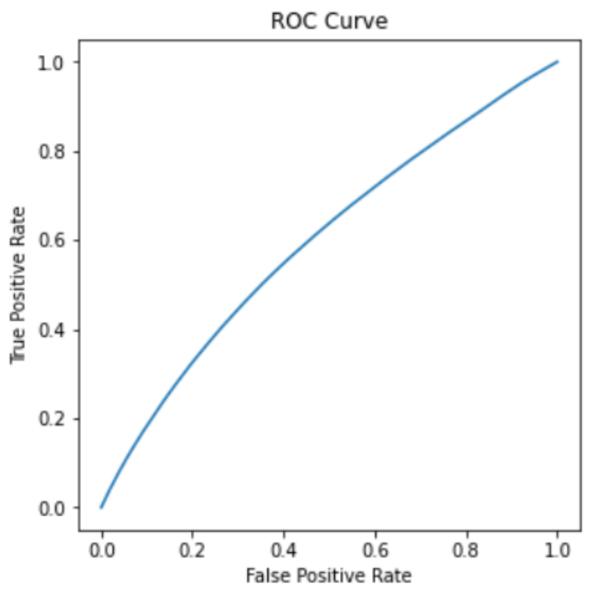
# Look at the parameters for the best model that was evaluated from the grid

```
parammap = cv.bestModel.stages[4].extractParamMap()

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

# Grab the model from Stage 4 of the pipeline

```
mymodel = cv.bestModel.stages[4]

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")
plt.show()
```

```
LogisticRegression_af08595af35e__aggregationDepth 2
LogisticRegression_af08595af35e__elasticNetParam 1.0
LogisticRegression_af08595af35e__family auto
LogisticRegression_af08595af35e__featuresCol features
LogisticRegression_af08595af35e__fitIntercept True
LogisticRegression_af08595af35e__labelCol label
LogisticRegression_af08595af35e__maxBlockSizeInMB 0.0
LogisticRegression_af08595af35e__maxIter 100
LogisticRegression_af08595af35e__predictionCol prediction
LogisticRegression_af08595af35e__probabilityCol probability
LogisticRegression_af08595af35e__rawPredictionCol rawPrediction
LogisticRegression_af08595af35e__regParam 0.0
LogisticRegression_af08595af35e__standardization True
LogisticRegression_af08595af35e__threshold 0.5
LogisticRegression_af08595af35e__tol 1e-06
```



# Extract the coefficients on each of the variables

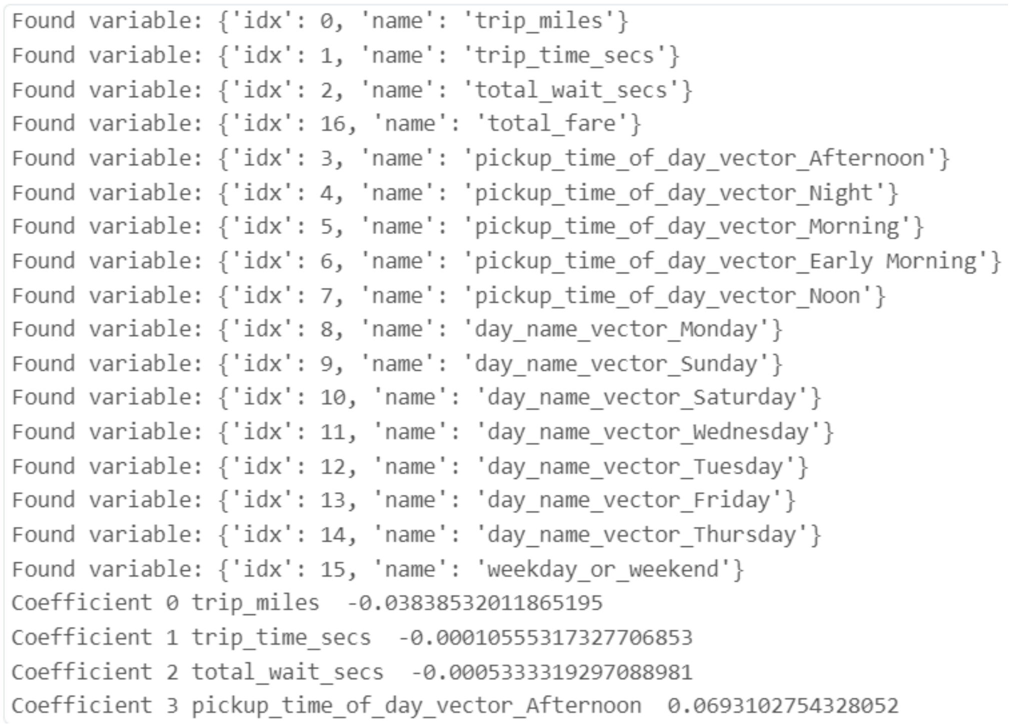
```
coeff = mymodel.coefficients.toArray().tolist()
```

# Loop through the features to extract the original column names. Store in the var_index dictionary

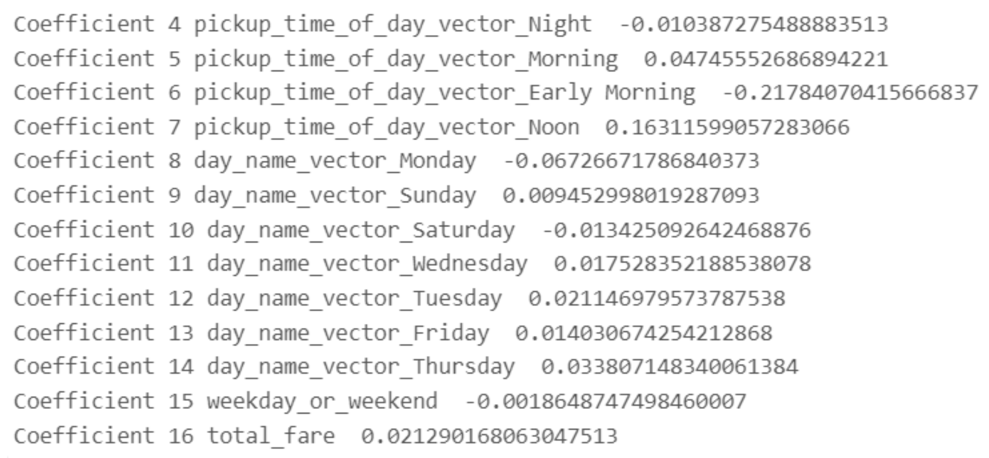
```
var_index = dict()
for variable_type in ['numeric', 'binary']:
    for variable in 
predictions.schema["features"].metadata["ml_attr"]["attrs"][vari
able_type]:
         print(f"Found variable: {variable}" )
         idx = variable['idx']
         name = variable['name']
         var_index[idx] = name      # Add the name to the 
dictionary
```

# Loop through all of the variables found and print out the associated coefficients

```
for i in range(len(var_index)):
    print(f"Coefficient {i} {var_index[i]}  {coeff[i]}")
```



```
Found variable: {'idx': 0, 'name': 'trip_miles'}
Found variable: {'idx': 1, 'name': 'trip_time_secs'}
Found variable: {'idx': 2, 'name': 'total_wait_secs'}
Found variable: {'idx': 16, 'name': 'total_fare'}
Found variable: {'idx': 3, 'name': 'pickup_time_of_day_vector_Afternoon'}
Found variable: {'idx': 4, 'name': 'pickup_time_of_day_vector_Night'}
Found variable: {'idx': 5, 'name': 'pickup_time_of_day_vector_Morning'}
Found variable: {'idx': 6, 'name': 'pickup_time_of_day_vector_Early Morning'}
Found variable: {'idx': 7, 'name': 'pickup_time_of_day_vector_Noon'}
Found variable: {'idx': 8, 'name': 'day_name_vector_Monday'}
Found variable: {'idx': 9, 'name': 'day_name_vector_Sunday'}
Found variable: {'idx': 10, 'name': 'day_name_vector_Saturday'}
Found variable: {'idx': 11, 'name': 'day_name_vector_Wednesday'}
Found variable: {'idx': 12, 'name': 'day_name_vector_Tuesday'}
Found variable: {'idx': 13, 'name': 'day_name_vector_Friday'}
Found variable: {'idx': 14, 'name': 'day_name_vector_Thursday'}
Found variable: {'idx': 15, 'name': 'weekday_or_weekend'}
Coefficient 0 trip_miles  -0.03838532011865195
Coefficient 1 trip_time_secs  -0.00010555317327706853
Coefficient 2 total_wait_secs  -0.0005333319297088981
Coefficient 3 pickup_time_of_day_vector_Afternoon  0.0693102754328052
```



```
Coefficient 4 pickup_time_of_day_vector_Night  -0.010387275488883513
Coefficient 5 pickup_time_of_day_vector_Morning  0.04745552686894221
Coefficient 6 pickup_time_of_day_vector_Early Morning  -0.21784070415666837
Coefficient 7 pickup_time_of_day_vector_Noon  0.16311599057283066
Coefficient 8 day_name_vector_Monday  -0.06726671786840373
Coefficient 9 day_name_vector_Sunday  0.009452998019287093
Coefficient 10 day_name_vector_Saturday  -0.013425092642468876
Coefficient 11 day_name_vector_Wednesday  0.017528352188538078
Coefficient 12 day_name_vector_Tuesday  0.021146979573787538
Coefficient 13 day_name_vector_Friday  0.014030674254212868
Coefficient 14 day_name_vector_Thursday  0.033807148340061384
Coefficient 15 weekday_or_weekend  -0.0018648747498460007
Coefficient 16 total_fare  0.021290168063047513
```

Appendix 4-3: Data Visualization Code

# Correlation matrix using Seaborn

# Convert the numeric values to vector columns

```
vector_column = "correlation_features"
```

# Choose the numeric (Double) columns

```
numeric_columns = ['trip_miles', 'trip_time_secs', 'tips',
'total_wait_secs', 'pickup_hour', 'day_of_week_num',
'total_fare']
assembler = VectorAssembler(inputCols=numeric_columns,
outputCol=vector_column)
sdf_vector =
assembler.transform(transformed_sdf).select(vector_column)
```

# Create the correlation matrix, then get just the values and convert to a list

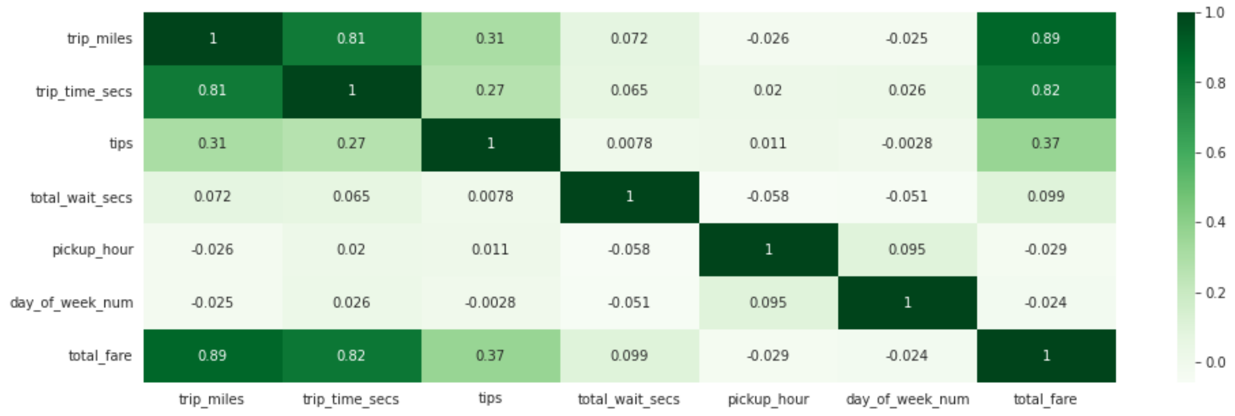
```
matrix = Correlation.corr(sdf_vector,
vector_column).collect()[0][0]
correlation_matrix = matrix.toArray().tolist()
```

# Convert the correlation to a Pandas dataframe

```
correlation_matrix_df = pd.DataFrame(data=correlation_matrix,
columns=numeric_columns, index=numeric_columns)

plt.figure(figsize=(16,5))
```

# Set the style for Seaborn plots

```
sns.set_style("white")
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")
```



```
tips_category = transformed_sdf.withColumn('tip_type',
when(transformed_sdf.label == 1.0, 'Good
Tip').when(transformed_sdf.label == 0.0, 'Bad Tip'))
```

# Show frequency of the tip 'label' column

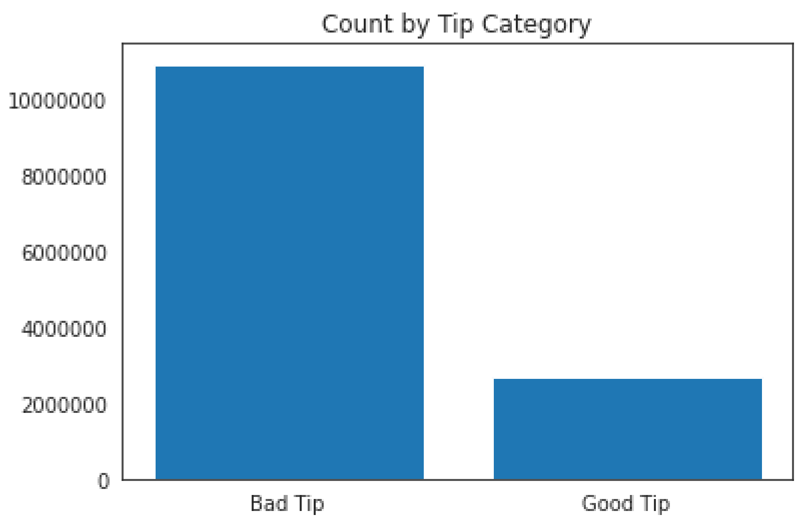
```
tip_category_df =
tips_category.groupby('tip_type').count().sort('tip_type').toPan
das()
```

# Set up a figure

```
fig = plt.figure(facecolor='white')
plt.ticklabel_format(style='plain')
```

# Bar plot of tip 'label' and count

```
plt.bar(tip_category_df['tip_type'],tip_category_df['count'] )
plt.title("Count by Tip Category")
plt.savefig("frequency_tip_category.png")
```



# Take a sample of the 'total_fare' and 'tips' columns and convert to a Pandas dataframe

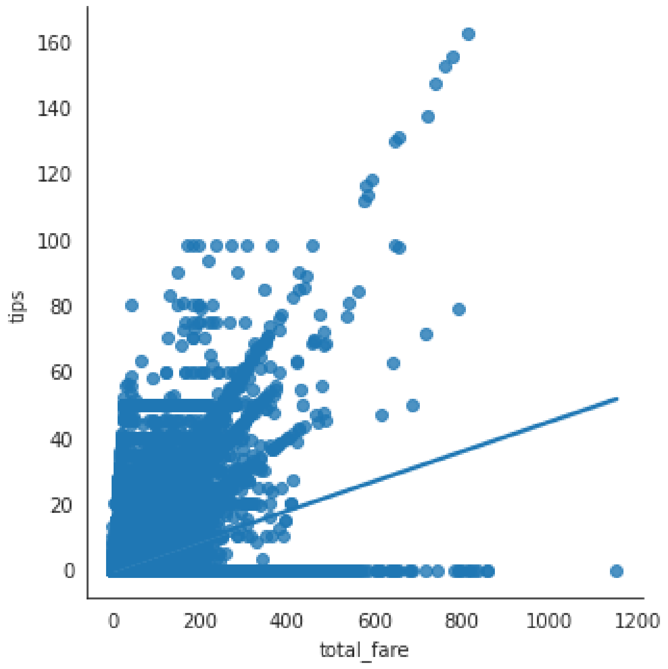
```
total_fare_tips_df = transformed_sdf.select('total_fare',
'tips').sample(False, 0.25).toPandas()
```

# Set the style for Seaborn plots

```
sns.set_style("white")
```

# Create the relationship plot

```
sns.lmplot(x='total_fare', y='tips', data=total_fare_tips_df)
```



# Take a sample of the 'trip_miles' and 'tips' columns and convert to a Pandas dataframe

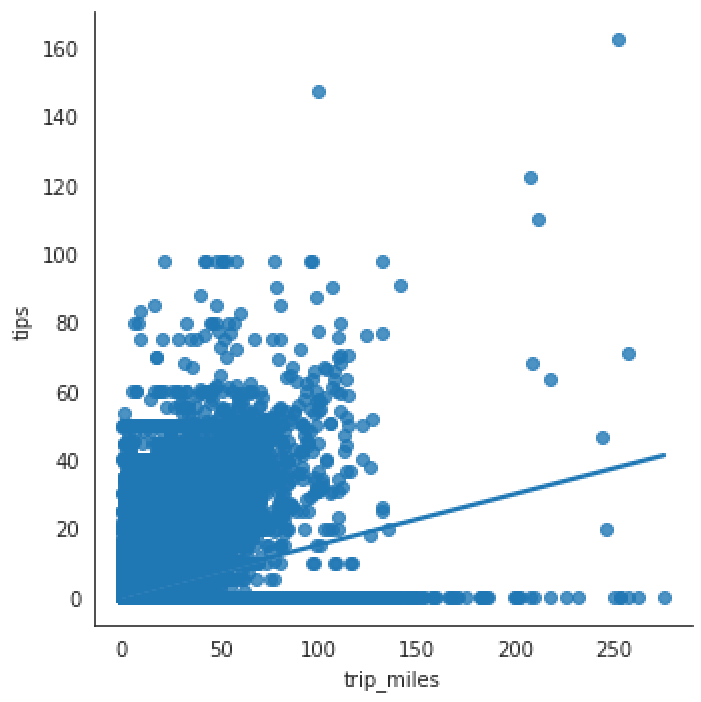
```
trip_miles_tips_df = transformed_sdf.select('trip_miles',
'tips').sample(False, 0.25).toPandas()
```

# Set the style for Seaborn plots

```
sns.set_style("white")
```

# Create the relationship plot

```
sns.lmplot(x='trip_miles', y='tips', data=trip_miles_tips_df)
```



# Take a sample of the 'trip_time_secs' and 'tips' columns and convert to a Pandas dataframe

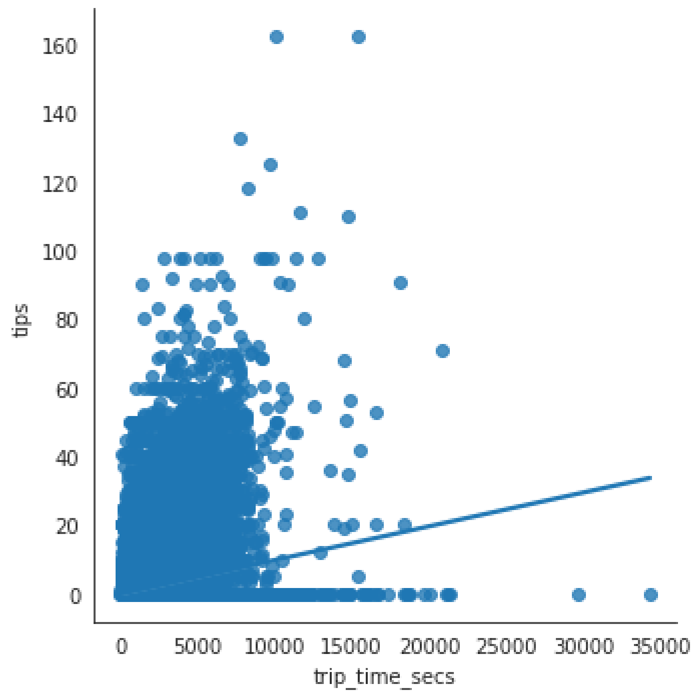
```
trip_time_secs_tips_df =
transformed_sdf.select('trip_time_secs', 'tips').sample(False,
0.25).toPandas()
```

# Set the style for Seaborn plots

```
sns.set_style("white")
```

# Create the relationship plot

```
sns.lmplot(x='trip_time_secs', y='tips',
data=trip_time_secs_tips_df)
```



# Take a sample of the 'total_wait_secs' and 'tips' columns and convert to a Pandas dataframe

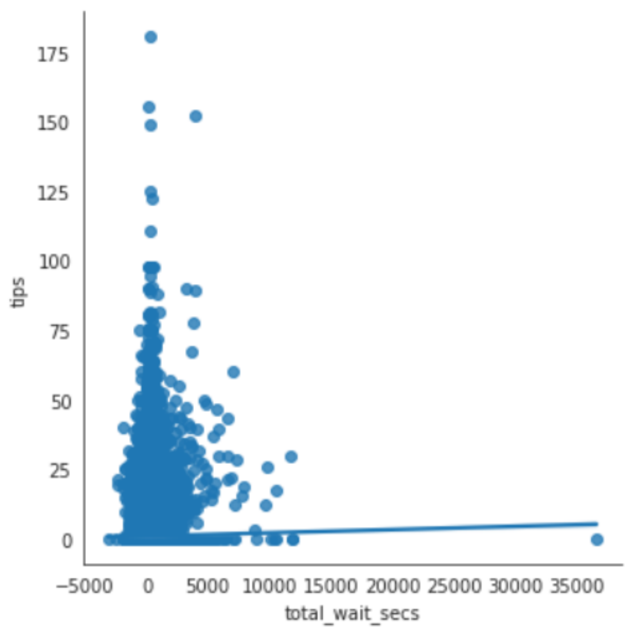
```
total_wait_secs_tips_df =
transformed_sdf.select('total_wait_secs', 'tips').sample(False,
0.25).toPandas()
```

# Set the style for Seaborn plots

```
sns.set_style("white")
```

# Create the relationship plot

```
sns.lmplot(x='total_wait_secs', y='tips',
data=total_wait_secs_tips_df)
```



# Show frequency of the 'day_name' column

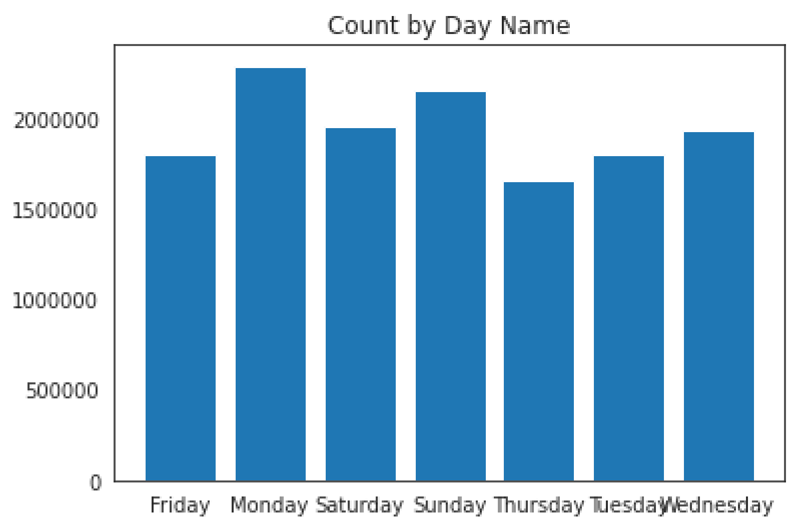
```
day_name_df =
transformed_sdf.groupby('day_name').count().sort('day_name').toP
andas()
```

# Set up a figure

```
fig = plt.figure(facecolor='white')
plt.ticklabel_format(style='plain')
```

# Bar plot of 'day_name' and count

```
plt.bar(day_name_df['day_name'],day_name_df['count'])
plt.title("Count by Day Name")
plt.savefig("frequency_day_name.png")
```



# Show frequency of the 'pickup_time_of_day' column

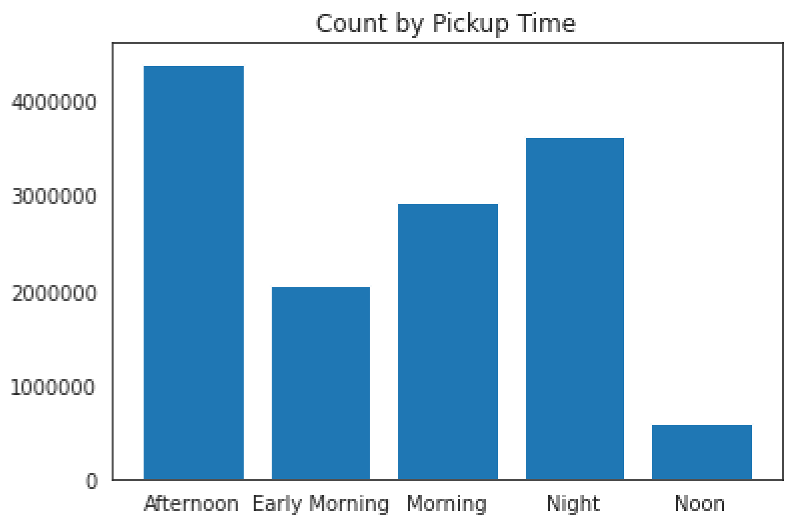
```
pickup_time_of_day_df =
transformed_sdf.groupby('pickup_time_of_day').count().sort('pick
up_time_of_day').toPandas()
```

# Set up a figure

```
fig = plt.figure(facecolor='white')
plt.ticklabel_format(style='plain')
```

# Bar plot of 'pickup_time_of_day' and count

```
plt.bar(pickup_time_of_day_df['pickup_time_of_day'],pickup_time_
of_day_df['count'])
plt.title("Count by Pickup Time")
plt.savefig("frequency_pickup_time_of_day.png")
```


Count by Pickup Time
Afternoon
Early Morning
Morning
Night
Noon