Big Data Technologies Semester Project On NYC Taxi Drivers

CIS 4130

Tenzing Palden
tenzing.palden@baruchmail.cuny.edu

CIS 4130
Tenzing Palden
tenzing.palden@baruchmail.cuny.edu

Milestone 1

New York City is a concrete jungle run by bustling cars and trains. A few decades ago, when you thought of New York, a stark image of a yellow taxi cab was imagined along with the statue of the library. The data I am going to be looking at is the New York City Taxi and Limousine Commission (TLC) Trip Record Data. This OpenData dataset is already hosted on amazon s3, so I dont have to do much work downloading or editing the data. It contains the records counting, trip price, trip duration, trip length and many more statistics for the past 13 years of operational TLCs in NYC. I want to understand how strong the economy of pre-Uber dominated markets of yellow taxis were, and chart the timeline of growth by For-Hire Vehicles (Lyft, Uber, Curb). I can visualize hotspots of Taxi pickups, calculate averages for price and ride length, and understand the patterns that people inevitably and unconsciously make.

My goal for this project is to map out the prices by location , and forecast what prices might be in the future. Taking in account to taxi zones, zip codes and frequency, the relationship between price, trip length, and location can be calculated. The data for this topic is separated into years, and then months and then by type of TLC. It is broken down into Yellow cab trip records, Green cab trip records, For-hire vehicle trip records and finally High-volume For-hire trip records. The difference in price vs trip length for these categories can also be investigated. I am also curious to see what type of service a regular customer uses depending on their needs. Is a yellow cab called when the trip is long? Is a green cab called when the trip is short? These are some questions I would like to answer with the analysis of the data.

CIS 4130
Tenzing Palden
tenzing.palden@baruchmail.cuny.edu

Milestone 2

My data set (New York City Taxi and Limousine Commission (TLC) Trip Record Data) is already hosted on a s3 bucket. I started an amazon Ec2 instance, and used Aws CLI to enter “aws s3 ls s3://nyc-tlc/” to see the paths and directories inside the bucket. I also used “aws s3 ls --recursive s3://nyc-tlc/” in order to see every single file in the directory. The files I want to work with were inside a path called “s3://nyc-tlc/trip data/”. It contains the monthly datasets from 2009-2022 for Yellow Cab data, Green Cab data and For-Hire Vehicles.

Link for database is down below:

https://registry.opendata.aws/nyc-tlc-trip-records-pds/

CIS 4130
Tenzing Palden
tenzing.palden@baruchmail.cuny.edu

Milestone 3

In the Open dataset for NYC taxi trips there are 795 files within the entire bucket that total up to 373.8 GB. It is safe to say that this data pile is huge. Analyzing each file would prove to be a challenge since each month of a year is separated into its individual CSV file. The data is also available in CSV file format and also Parquet file format. They are also separated even further with different types of tacos being offered in different files. Green taxis, yellow taxis, and pay for service cars all have different files. What is very strange is that the features of the tables do not remain constant throughout the years. I only looked only into 7 years worth of Green taxi and I noticed this. The data types observed were objects, int64 and float64.

Year, data	Number of Columns
2015, Green	21
2016, Green	21
2017, Green	19
2018, Green	19
2019, Green	20
2020, Green	20
2021, Green	20
2022, Green	20

The most common features are VendorID, lpep_pickup_datetime, lpep_dropoff_datetime, RatecodeID,PULocationID, DOLocationID, passenger_count, trip_distance, fare_amount, Extra, mta_tax, tip_amount, tolls_amount, egail_fee, improvement_surcharge, total_amount, payment_type, trip_type, congestion_surcharge. The oldest date in this dataset is from january

2009 and the newest to the dataset is from May 2022. The only data analysis I collected was that in 2022, so far, the average pay per Green taxi cab was 16.23 dollars.

This was all done using python.

CODE:

```python
import boto3
import botocore
import pandas as pd
from IPython.display import display, Markdown

s3 = boto3.client('s3',
aws_access_key_id="AKIAWNUHHLMR5HQNRZEU",
    aws_secret_access_key="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"
)
s3_resource = boto3.resource('s3',
aws_access_key_id="AKIAWNUHHLMR5HQNRZEU",
    aws_secret_access_key="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"
)

def list_bucket_contents(bucket, match='', size_mb=0):
    bucket_resource = s3_resource.Bucket(bucket)
    total_size_gb = 0
    total_files = 0
    match_size_gb = 0
    match_files = 0
    for key in bucket_resource.objects.all():
        key_size_mb = key.size/1024/1024
        total_size_gb += key_size_mb
        total_files += 1
        list_check = False
        if not match:
            list_check = True
        elif match in key.key:
            list_check = True
        if list_check and not size_mb:
            match_files += 1
            match_size_gb += key_size_mb
            print(f'{key.key} ({key_size_mb:3.0f}MB)')
        elif list_check and key_size_mb <= size_mb:
            match_files += 1
            match_size_gb += key_size_mb
            print(f'{key.key} ({key_size_mb:3.0f}MB)')

    if match:
        print(f'Matched file size is {match_size_gb/1024:3.1f}GB with {match_files}
files')

    print(f'Bucket {bucket} total size is {total_size_gb/1024:3.1f}GB with
{total_files} files')


list_bucket_contents(bucket='nyc-tlc', match='', size_mb="")

def preview_csv_dataset(bucket, key, rows=10):
    data_source = {
            'Bucket': bucket,
            'Key': key
        }
    # Generate the URL to get Key from Bucket
    url = s3.generate_presigned_url(
        ClientMethod = 'get_object',
        Params = data_source
    )

    if "csv" in key:
        data = pd.read_csv(url, nrows=rows)
        return data

df_2015 = preview_csv_dataset(bucket='nyc-tlc',
key='csv_backup/green_tripdata_2015-01.csv', rows=100)
df_2016 = preview_csv_dataset(bucket='nyc-tlc',
key='csv_backup/green_tripdata_2016-01.csv', rows=100)
df_2017 = preview_csv_dataset(bucket='nyc-tlc',
key='csv_backup/green_tripdata_2017-01.csv', rows=100)
df_2018 = preview_csv_dataset(bucket='nyc-tlc',
key='csv_backup/green_tripdata_2018-01.csv', rows=100)
df_2019 = preview_csv_dataset(bucket='nyc-tlc',
key='csv_backup/green_tripdata_2019-01.csv', rows=100)
df_2020 = preview_csv_dataset(bucket='nyc-tlc',
key='csv_backup/green_tripdata_2020-01.csv', rows=100)
df_2021 = preview_csv_dataset(bucket='nyc-tlc',
key='csv_backup/green_tripdata_2021-01.csv', rows=100)
df_2022 = preview_csv_dataset(bucket='nyc-tlc',
key='csv_backup/green_tripdata_2022-01.csv', rows=100)

df_2018.head()
df_2015.info()
df_2016.info()
df_2017.info()
df_2018.info()
df_2019.info()
```

```
df_2020.info()
df_2021.info()
df_2022.info()
df_2022.describe()
```

CIS 4130
Tenzing Palden
tenzing.palden@baruchmail.cuny.edu

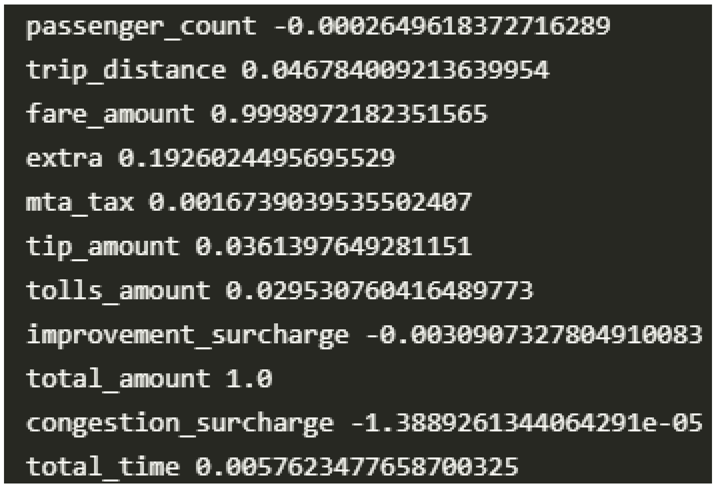
Milestone 4

To process my data (NYC-TLC), I wrote a Pyspark program in Python to extract and load the files and perform some regression analysis on the data. The main challenged I faced was with the data not loading properly. The 149 parquet files did not load properly from the directory and thus giving me alot of hard time starting the entire pipeline process itself. Possible explanations of the issue might be from different schemas from different files, but also could be explained with local issues on my computer as other students were able to complete this task successfully while I struggled on it. After exhausting many options such as launching the code on EC2 instance and factory resetting my hardware, I decided to look at a specific time frame instead of looking at the broader picture of the data. I chose 2019 as the year I would analyze because it was right before the covid pandemic and it would be interesting to examine the taxi industry before this virus hit our shores.

The main parts of the pipeline are transformation of the data, the analysis of said data and the regression models built. The pipeline begins by importing the necessary libraries and creating a SparkSession object, which is used to read in the data stored in Parquet format in multiple files on an S3 bucket. The data is then cleaned and transformed by casting all double data type columns to integer. This was done to retain a create format dataframe for regression models down the file. Then more columns were added that contained more easier to understand figures such as calculating the total time of each ride in minutes by finding the difference between the "start_time" and "end_time" columns.This helps to ensure that the analysis is based on high-quality, representative data.

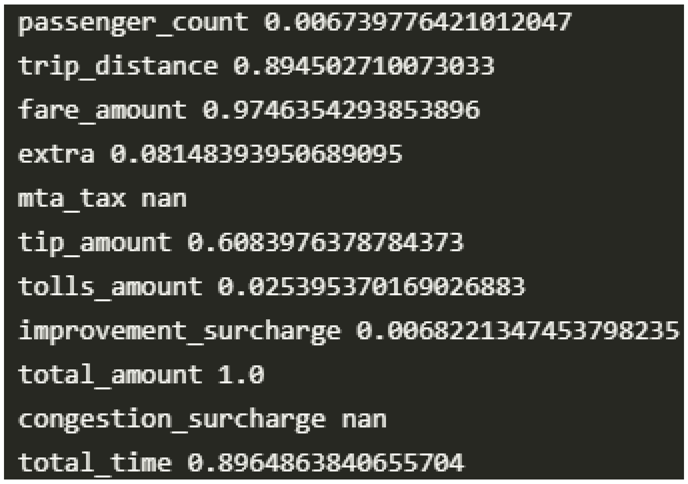
The most important part of our code was the removal of outliers from our dataset. I define outlier in this project as any data point that shares the characteristics of an error in the dataset, for example there were thousands of rows of negative values in columns where that should have been impossible. This was a massive undertaking as our dataset was so massive that finding a perfect function that removed all outliers and impossible data was not possible. I chose a method to remove all datapoints that did not fall into 3 standard deviations of the mean. To test the before and after of this function, I found code online that looped through all columns and found its correlation with the target column.The images below are the output that represents the before and after of the outlier removal process. Before the rmeoval, the correlation of the fare of the taxi ride and the distance was less than 5 percent. This should not be true as the taxi business relies of distance to calculate the total fare. After the removal, the correlation of the same target and column jumped to an estimated 90 percent correlation.

Before



```
passenger_count -0.0002649618372716289
trip_distance 0.046784009213639954
fare_amount 0.9998972182351565
extra 0.1926024495695529
mta_tax 0.0016739039535502407
tip_amount 0.0361397649281151
tolls_amount 0.029530760416489773
improvement_surcharge -0.0030907327804910083
total_amount 1.0
congestion_surcharge -1.3889261344064291e-05
total_time 0.005762347765870325
```

After



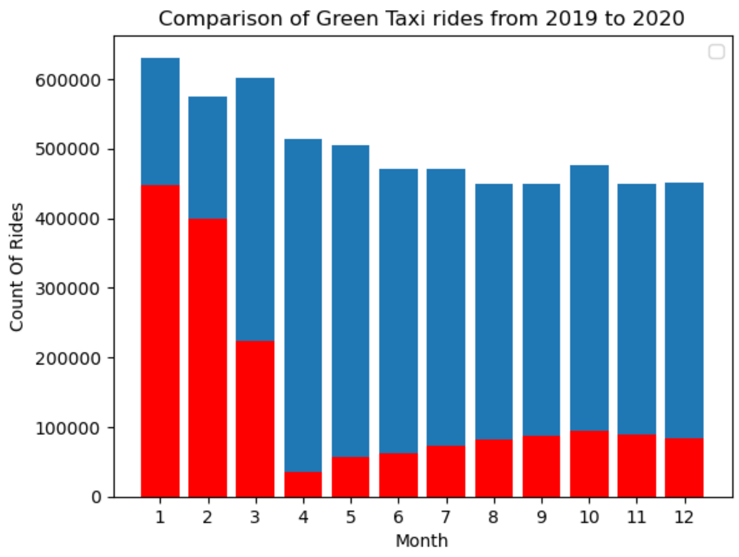
```
passenger_count 0.006739776421012047
trip_distance 0.894502710073033
fare_amount 0.9746354293853896
extra 0.08148393950689095
mta_tax nan
tip_amount 0.6083976378784373
tolls_amount 0.025395370169026883
improvement_surcharge 0.006822134745379823S
total_amount 1.0
congestion_surcharge nan
total_time 0.8964863840655704
```

The code is the following:

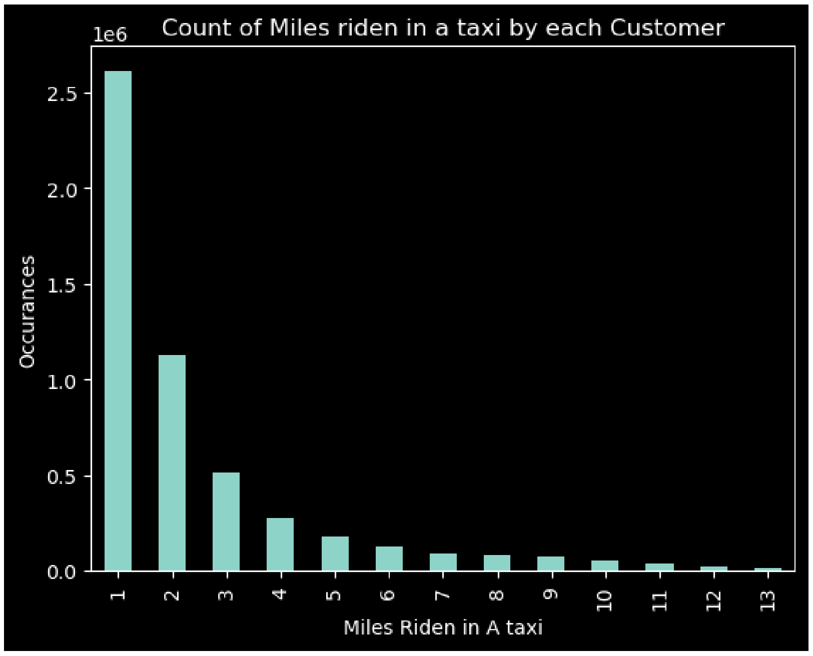
```
import six
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.sql.functions import unix_timestamp, round, stddev, avg, mean, expr, col
from pyspark.ml.regression import LinearRegression
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.evaluation import RegressionEvaluator
import matplotlib.pyplot as plt
import matplotlib.style
import matplotlib as mpl

if __name__ == "__main__":
    spark = SparkSession.builder \
        .appName("single_csv") \
        .getOrCreate()

    sc = spark.sparkContext
    sc.setLogLevel("ERROR")
    #get version of pyspark for future reference
    print('Apache Spark Version :'+spark.version)
    print('Apache Spark Version :'+spark.sparkContext.version)

    file_list =  ['s3a://nyc-tlc/trip data/yellow_tripdata_2021-01.parquet',
                  's3a://nyc-tlc/trip data/yellow_tripdata_2021-02.parquet',
                  's3a://nyc-tlc/trip data/yellow_tripdata_2021-03.parquet',
                  's3a://nyc-tlc/trip data/yellow_tripdata_2021-04.parquet',
                  's3a://nyc-tlc/trip data/yellow_tripdata_2021-05.parquet',
                  's3a://nyc-tlc/trip data/yellow_tripdata_2021-06.parquet'
                 ]

    df = spark.read.parquet(*file_list)

    df.printSchema()
    print("Number of rows:", df.count())
    print("Number of columns:", len(df.columns))

    # creating a list of column names
    cols = df.columns

    # iterate through the list of column names
    for col in cols:
        # get the number of missing values for each column
        missing_count = df.filter(df[col].isNull()).count()
        # print the column name and number of missing values
        print(f"Column {col}: {missing_count} missing values")

    # Calculate the difference between the "start_time" and "end_time" columns in seconds and
then making them secondds

    df = df.withColumn("total_time", unix_timestamp(df["tpep_dropoff_datetime"]) -
unix_timestamp(df["tpep_pickup_datetime"]))
    df = df.withColumn("total_time", round(df["total_time"]/60, 2))

    #finding the columns with the datatype double and converting them into int
    double_cols=["passenger_count", "trip_distance", "RatecodeID", "fare_amount", "extra",
"mta_tax", "tip_amount", "tolls_amount","improvement_surcharge",
"total_amount","congestion_surcharge", "total_time"]
    new_df = df
    for col in double_cols:
        new_df = new_df.withColumn(col,new_df[col].cast('integer'))
    new_df=new_df.drop("tpep_pickup_datetime", "tpep_dropoff_datetime", "VendorID",
"payment_type")
    #check new datatypes
    new_df.printSchema()

    int_cols = ["passenger_count", "trip_distance", "fare_amount", "extra", "mta_tax",
"tip_amount", "tolls_amount","improvement_surcharge", "total_amount","congestion_surcharge",
"total_time"]
    new_df.describe(int_cols).show()

#/////////////////////////////////////////////////////////////////////////////////////////////remvi
ng outlier portion /

    #making a df with only integrs
    int_only_df = df.select(int_cols)

    cols = int_only_df.columns

    #corrrelation before outlier removal. Good to see how much the data was effected
    #this code is copied from online. Finding correlations from features and total_amount
    for i in int_only_df.columns:
        if not( isinstance(int_only_df.select(i).take(1)[0][0], six.string_types)):
            print( "Correlation to total_amount for ", i,
int_only_df.stat.corr('total_amount',i))

    #removing outliers from data
    for col in cols:
        col_mean = int_only_df.select(mean(int_only_df[col])).first()[0]
        col_std = int_only_df.select(stddev(int_only_df[col])).first()[0]

        # Filter out the outliers using the mean and standard deviation
        int_only_df = int_only_df.filter((int_only_df[col] > col_mean - 3 * col_std) &
(int_only_df[col] < col_mean + 3 * col_std))

    #this code is copied from online. Checkinig correlation again. HUGE DIFFERENCE
    for i in int_only_df.columns:
        if not( isinstance(int_only_df.select(i).take(1)[0][0], six.string_types)):
            print( "Correlation to total_amount for ", i,
int_only_df.stat.corr('total_amount',i))

#///////////////////////////////////////////////////////////////////////////////////////////lin
portion /
    featureassembler = VectorAssembler(inputCols=["trip_distance","tip_amount", "total_time"],
outputCol= "Independant Features")
    output= featureassembler.transform(int_only_df)
    output.select("Independant Features").show()
    finalised_data = output.select("Independant Features", "total_amount")

    train_set, test_set = finalised_data.randomSplit([0.8, 0.2])
    reg = LinearRegression(featuresCol="Independant Features", labelCol= "total_amount")
    lr_model  = reg.fit(train_set)

    print("Coefficients: " + str(lr_model.coefficients))
    print("Intercept: " + str(lr_model.intercept))

    trainingSummary = lr_model.summary
    print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
    print("r2: %f" % trainingSummary.r2)

    lr_predictions = lr_model.transform(test_set)
    lr_predictions.show()

    lr_evaluator = RegressionEvaluator(predictionCol="prediction", \
                     labelCol="total_amount",metricName="r2")

    print("R Squared (R2) on test data = %g" % lr_evaluator.evaluate(lr_predictions))


#/////////////////////////////////////////////////////////////////////////////////////////////log
portion /
    int_only_df = int_only_df.withColumn("tip_binary", expr("CASE WHEN tip_amount > '0' THEN '1'
" +
                   "WHEN tip_amount <= '0' THEN '0' WHEN tip_amount IS NULL THEN ''" +
                   "ELSE tip_amount END"))
    int_only_df = int_only_df.withColumn("tip_binary",int_only_df["tip_binary"].cast('integer'))

    featureassembler = VectorAssembler(inputCols=["passenger_count", "trip_distance",
"fare_amount", "tolls_amount", "congestion_surcharge"], outputCol= "features")
    output= featureassembler.transform(int_only_df)
    log_moel_df= output.select("features", "tip_binary")
    train_set, test_set = log_moel_df.randomSplit([0.7, 0.3])

    log_reg = LogisticRegression(labelCol="tip_binary").fit(train_set)
    log_results = log_reg.evaluate(train_set).predictions
    results = log_reg.evaluate(test_set).predictions
    tp = results[(results.tip_binary == 1) & (results.prediction == 1)].count()
    tn = results[(results.tip_binary == 0) & (results.prediction == 0)].count()
    fp = results[(results.tip_binary == 0) & (results.prediction == 1)].count()
    fn = results[(results.tip_binary == 1) & (results.prediction == 0)].count()
    print(f"The TP is {tp}")
    print(f"The TN is {tn}")
    print(f"The FP is {fp}")
    print(f"The FN is {fn}")

    print(f"The accuracy of this model is {float((tp+tn)/results.count())}")
    print(f"The recall of this model is {float(tn)/(tp+tn)}")
```

CIS 4130
Tenzing Palden
tenzing.palden@baruchmail.cuny.edu

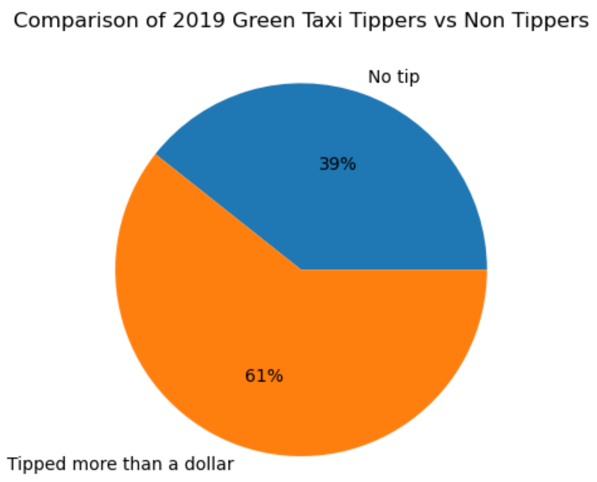
Milestone 5



Charting the comparisons between 2019 and 2020 ridership.As you can see this became a massive hit to our taxi industry. On the height of covid it was squashed down to 1/10th of its original size.
(For code see appendix A)

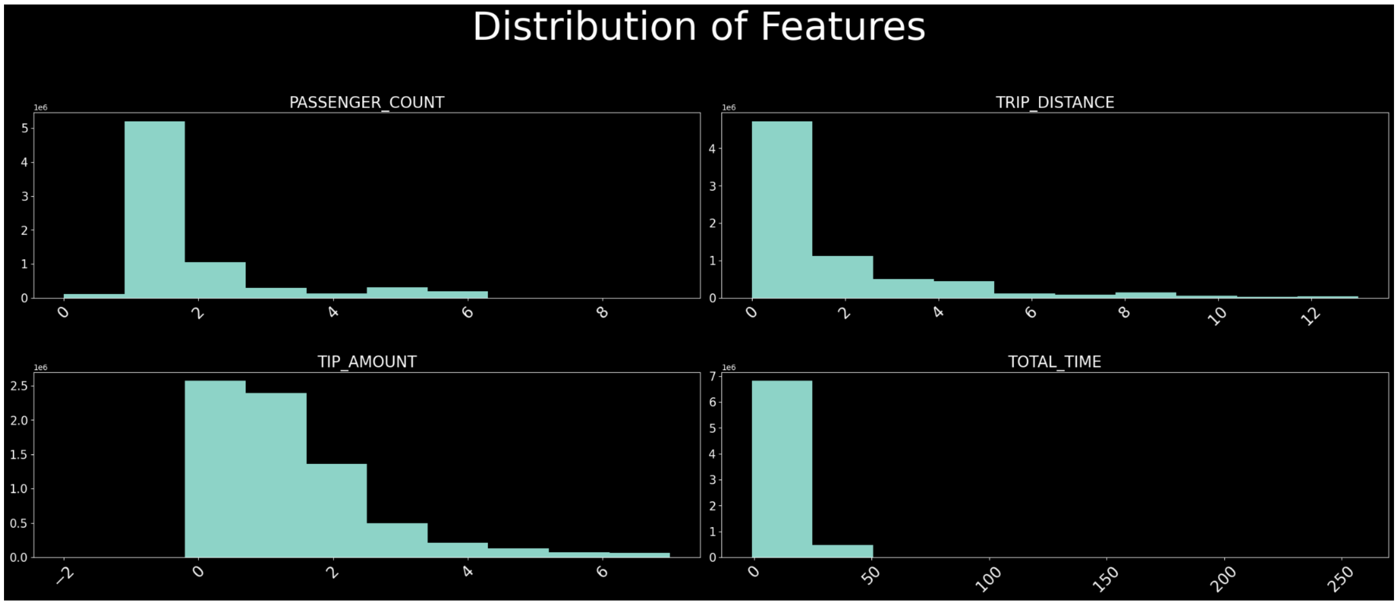


Bar chart of the Value counts of miles driven in the dataset.Most people use taxis to ride short distances.
(For code see appendix B)

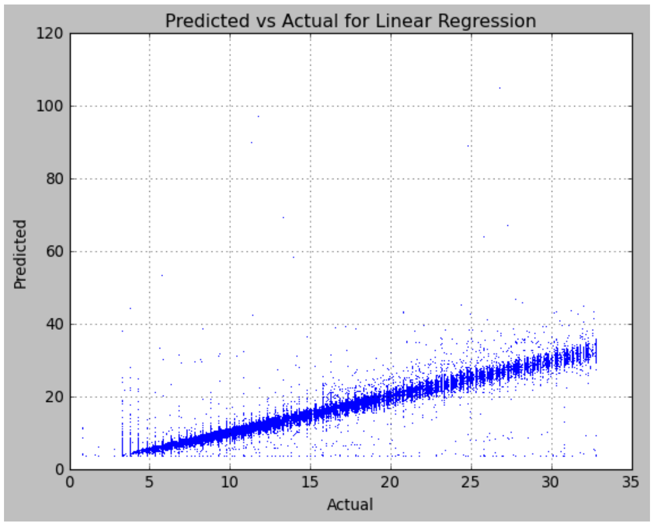


Pie chart showing percentages of Tippers vs non-Tippers. The logical model is about if we can predict tippers or not based on rid length and other factors.
(For code see appendix)

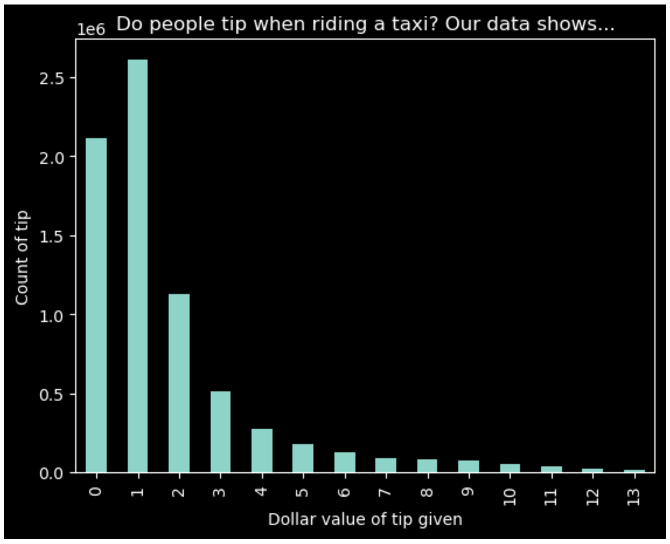
(For code see appendix D)



Predicted Vs Actual for Linreg model(Appendix E)



Count of Tips given when riding taxi (Appendix F)



CIS 4130
Tenzing Palden
tenzing.palden@baruchmail.cuny.edu

Milestone 6

This project involved building a data processing pipeline that uses a dataset of NYC taxi logs to gain insights into various aspects of the industry. This data was sourced from Amazon's open data. The data processing pipeline included tasks such as sourcing and loading the data, cleaning the raw data, conducting exploratory data analysis to identify patterns, and building regression models to try and inquiries about the large taxi business.

The main conclusions I have derived from insights into the this data set are the main reasons why people tip a taxi driver. It seems as if the column "tolls_amount" has the highest strength on influencing weather or not the customer of the taxi will tip or not. This was discovered through the logcical regression model where specific columns were tested together to see which one had the highest weight. In my opinion, I think the psychological effect of driving across a bridge or tunnel (or any area where tolls happen) makes a ride feel longer and the tip more desrevred towards the driver. Our visualzations show that 40% of people do not tip the drivers, but at the same time an overwhelming amount of rides in a taxi cab are less than one mile. New Yorkers tip a majority amount of the time but in the situations they dont, I feel as if it is because they think that one mile rides do not deserve tip.

Another conclusion I formulated for this report is that Taxi drivers are not the best when it comes to reporting data. Much of the data was very irregular with thousands and thousands of rows containing negative values for things like miels driven. Data preprocessing has been a very strong component of the code.

Future plans for this project can include a more indepth analysis on specific conditions that sets the grounds on when or when not a person tips during a car ride. This analysis is interesting because many drivers could make more money if they were to receive tip, so they should understand if there are any factors that make a person tip during a ride.

Thank you for reading.

This concludes the NYC-TLC project by Tenzing Palden.

The appendix follows this page.

Appendix

(Appendix A)

```
plt.bar(by_month_2019.index, by_month_2019["Count of Rides"])
plt.bar(by_month_2020.index, by_month_2020["Count of Rides"], color="red")
plt.xlabel('Month')
plt.ylabel('Count Of Rides')
plt.title('Comparison of Green Taxi rides from 2019 to 2020')
plt.xticks(by_month_2019.index)
plt.legend()
plt.show()
```

(Appendix B)

```
trip_dist= feature_viz[feature_viz["trip_distance"] > 0]
trip_dist["trip_distance"].value_counts().sort_index().plot(kind="bar", xlabel =
"Miles Riden in A taxi", ylabel="Occurances", title= "Count of Miles riden in a
taxi by each Customer")
```

(Appendix C)

```
plt.pie(percent_tip_2019, labels = labels, autopct='%1.0f%%')
myexplode = [0.2, 0, 0, 0]
plt.title('Comparison of 2019 Green Taxi Tippers vs Non Tippers')
```

(Appendix D)

```
from matplotlib import cm
fig = plt.figure(figsize=(25,15)) ## Plot Size
st = fig.suptitle("Distribution of Features", fontsize=50,
                  verticalalignment='center') # Plot Main Title

for col,num in zip(feature_viz.describe().columns, range(1,11)):
    ax = fig.add_subplot(3,2,num)
    ax.hist(feature_viz[col])
    plt.grid(False)
    plt.xticks(rotation=45,fontsize=20)
    plt.yticks(fontsize=15)
    plt.title(col.upper(),fontsize=20)
    plt.style.use('dark_background')
plt.tight_layout()
```

```
st.set_y(0.95)
fig.subplots_adjust(top=0.85,hspace = 0.4)
plt.show()
```

(Appendix E)

```
plt.plot(lr_predict_pd['total_amount'],lr_predict_pd['prediction'], 'b,')
plt.grid(True)
plt.title("Predicted vs Actual for Linear Regression")
plt.xlabel("Actual")
plt.ylabel("Predicted")
```

(Appendix F)

```
trip_dist= feature_viz[feature_viz["tip_amount"] > -1]
trip_dist["trip_distance"].value_counts().sort_index().plot(kind="bar", xlabel =
"Dollar value of tip given", ylabel="Count of tip", title= "Do people tip when
riding a taxi? Our data shows...")
```