Predicting Duration of New York City Taxi Rides

1. INTRODUCTION

The purpose of this project is to use data collected by taxi services in New York to predict the duration of rides. To do this, it will be necessary to employ a number of analytical techniques. After preprocessing, I will use visualizations to derive insights from the data, such as what patterns and relationships might exist, and then perform feature engineering to aid in fitting a model that will be able to make accurate predictions.

2. DATA

2.1 Overview

The data used for this comes from a dataset created for a competition on kaggle.com. The data is a subset of a larger dataset of New York City Taxi trips, and consists of aproximately 1.5 million rows. Each row represents a taxi trip fromthe months of January, March, April, May, June, and July of 2016.

Note: The dataset was made available for use outside of the competition, provided it not be used for commercial purposes.

2.2 Features

id: A unique id for each entry

vendor_id: A numerical value assigned to the vendor who handled the trip.

pickup_datetime: The exact time at which the driver started the meter.

dropoff_datetime: The exact time at which the stopped the meter.

passenger_count: The number of passengers on the trip recorded by the driver.

pickup_longitude: The 8-digit longitude where the trip began.

pickup_latitude: The 8-digit latitude where the trip began.

dropoff_longitude: The 8-digit longitude where the trip ended.

dropoff_latitude: The 8-digit latitude where the trip ended.

store_and_fwd_flag: A Y/N variable that indicates if the trip was stored in the vehicle's memory due to lack of connection to the server.

trip_duration: The total duration of the trip in seconds.

3. METHODOLOGY

3.1 Exploratory Analysis

After reading the data into a pandas dataframe, we can begin exploration. I start by removing rows with null values and checking the datatypes of the columns.

```
df_train.dtypes
```

```
id                     object
vendor_id               int64
pickup_datetime        object
dropoff_datetime       object
passenger_count         int64
pickup_longitude      float64
pickup_latitude       float64
dropoff_longitude     float64
dropoff_latitude      float64
store_and_fwd_flag     object
trip_duration           int64
dtype: object
```

The datatype of the datetime columns is 'object', so they will need to be converted to datetime to make them practical for analysis.

```
#Convert pickup_datetime and dropoff_datetime columns to datetime datatype
df_train[["pickup_datetime", "dropoff_datetime"]] = df_train[["pickup_datetime", "dropoff_datetime"]].apply(pd.to_datetime)
```

```
df_train.dtypes
```

```
id                            object
vendor_id                      int64
pickup_datetime       datetime64[ns]
dropoff_datetime      datetime64[ns]
passenger_count                int64
pickup_longitude             float64
pickup_latitude              float64
dropoff_longitude            float64
dropoff_latitude             float64
store_and_fwd_flag            object
trip_duration                  int64
dtype: object
```

Next I plotted all of the latitude and longitude points. The resulting graph was a lot of empty space with a small blue blob roughly in the middle of the right-hand third, indicating that there were some extreme outliers in the coordinates data. To deal with these outliers, I selected latitude and longitude values that approximated the boundaries of New York City and limited the latitude and longitude data to values within those boundaries.

```
#setting coordinate boundaries
xlim = [-74.05, -73.75]
ylim = [40.60, 40.90]

df_train = df_train[(df_train.pickup_longitude > xlim[0]) & (df_train.pickup_longitude < xlim[1])]
df_train = df_train[(df_train.dropoff_longitude > xlim[0]) & (df_train.dropoff_longitude < xlim[1])]
df_train = df_train[(df_train.pickup_latitude > ylim[0]) & (df_train.pickup_latitude < ylim[1])]
df_train = df_train[(df_train.dropoff_latitude > ylim[0]) & (df_train.dropoff_latitude < ylim[1])]
```

I then plotted the new coordinates **(Fig. 1)**.

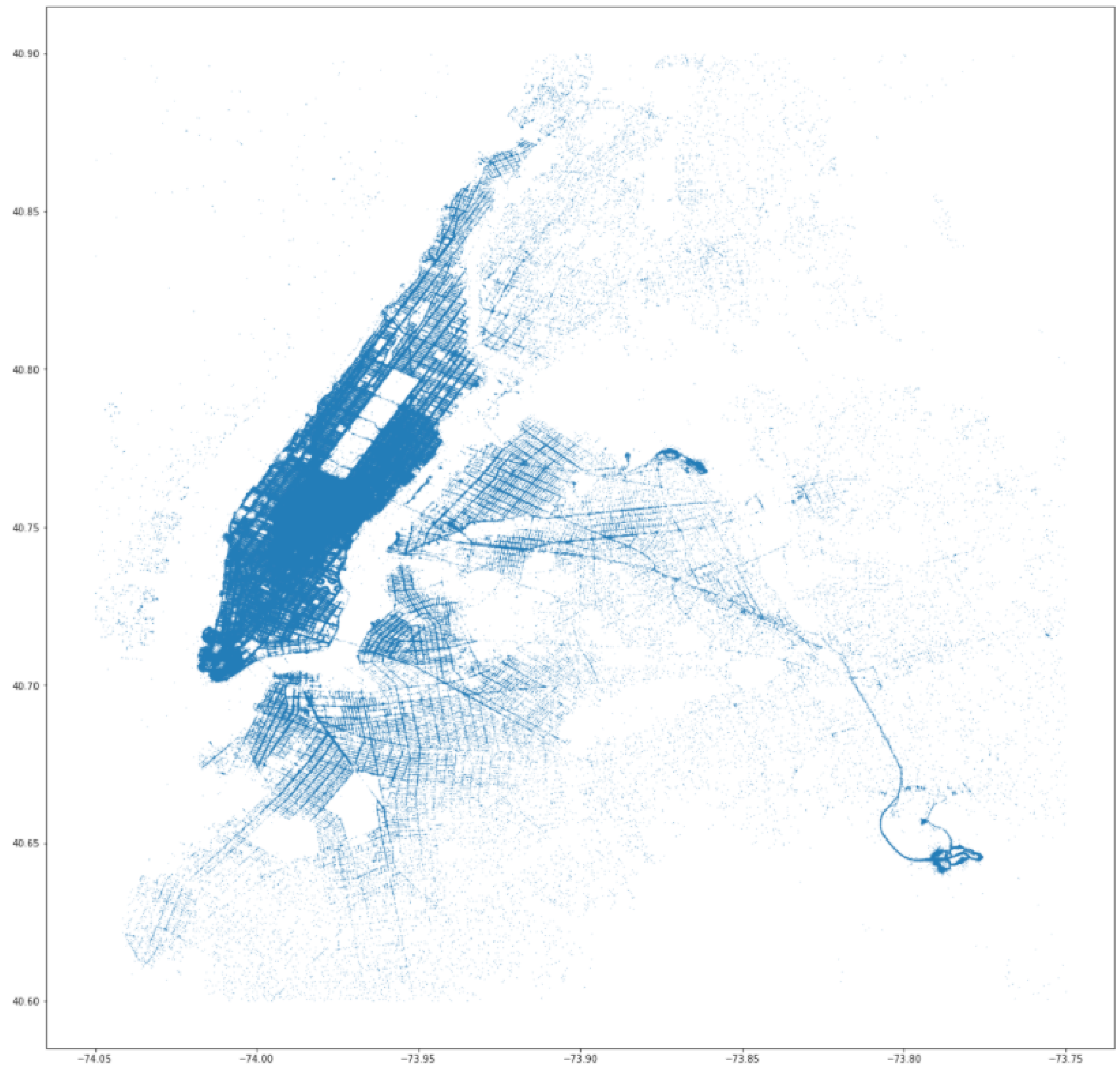


Fig. 1 Plot of pickup latitude and longitude coordinates.

We can already make a few interesting observations based on this graph. Particularly noteworthy is the high density of trips with start or end points in Manhattan and at JFK International Airport, in the lower right of the graph.

Next, I determined that knowing the distance of the trip would be a useful feature, and so I used a function to calculate the distance between the pickup and dropoff coordinates. As it is impossible to know the exact route taken, to accomplish this, I made use of the Haversine formula, which is used to calculate the distance between two points on a sphere.

Haversine formula:

$$\operatorname{hav}(\theta) = \operatorname{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2 - \lambda_1)$$

$$\theta = \frac{d}{r}$$

where:
$\varphi 1$, $\varphi 2$ are the latitude of point 1 and latitude of point 2,
$\lambda 1$, $\lambda 2$ are the longitude of point 1 and longitude of point 2,
θ is the central angle between the two points,
d is the distance, and r is the radius of the sphere.

```python
from math import radians, sin, cos, sqrt, asin
```

```python
#Calculate distance in metres between pickup and dropoff coordinate points, add new column to df_train

def haversine(columns):
    lat1, lon1, lat2, lon2 = columns
    R = 6372.8 # Earth radius in kilometers

    dLat = radians(lat2 - lat1)
    dLon = radians(lon2 - lon1)
    lat1 = radians(lat1)
    lat2 = radians(lat2)

    a = sin(dLat/2)**2 + cos(lat1)*cos(lat2)*sin(dLon/2)**2   #Haversine Formula
    c = 2*asin(sqrt(a))

    return (R * c) * 1000

cols = ['pickup_latitude','pickup_longitude','dropoff_latitude','dropoff_longitude']
distances = df_train[cols].apply(lambda x: haversine(x),axis = 1)
df_train['distance_m'] = distances.copy()
df_train['distance_m'] = round(df_train.distance_m,0)
```

Though this will only be the distance 'as the crow flies', and will not be an accurate representation of the distance travelled by the taxi.

I then used the distance to calculate the average speed of each trip. Again, this will not be an accurate calculation of the actual speed of the taxi, but should be useful for analysis nonetheless.

```python
#calculate average speep over the duration of each trip in m/s
df_train['av_speed'] = df_train.distance_m/df_train.trip_duration
```

```python
print("The average speed of all trips is", df_train.av_speed.mean(),"m/s.")
```

```
The average speed of all trips is 3.980839870495266 m/s.
```

```python
df_train.av_speed.describe()
```

```
count    1.451343e+06
mean     3.980840e+00
std      3.161351e+00
min      0.000000e+00
25%      2.531085e+00
50%      3.546028e+00
75%      4.935533e+00
max      2.577143e+03
Name: av_speed, dtype: float64
```

The max value returned by the describe function is unrealistically high.

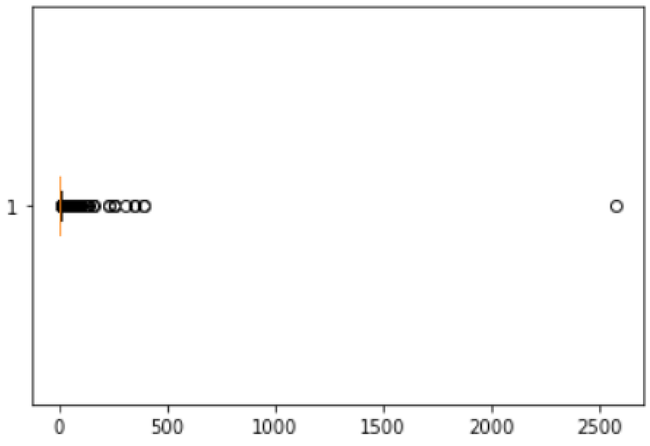


Fig. 2 Boxplot of average speed per trip.

From the boxplot **(Fig. 2)**, we can see a number of outliers with unrealistically high values, even if we assume a generous speed limit and no impediments such as traffic or stoplights. I removed these outliers by defining an upper and lower quantile.

```
#remove outliers
q_hi  = df_train["av_speed"].quantile(0.99)
q_lo = df_train["av_speed"].quantile(0.01)

df_train = df_train[(df_train["av_speed"] < q_hi) & (df_train["av_speed"] > q_lo)]

df_train.av_speed.describe()
```

```
count    1.422315e+06
mean     3.923450e+00
std      1.911855e+00
min      4.270613e-01
25%      2.551639e+00
50%      3.546028e+00
75%      4.896642e+00
max      1.117253e+01
Name: av_speed, dtype: float64
```

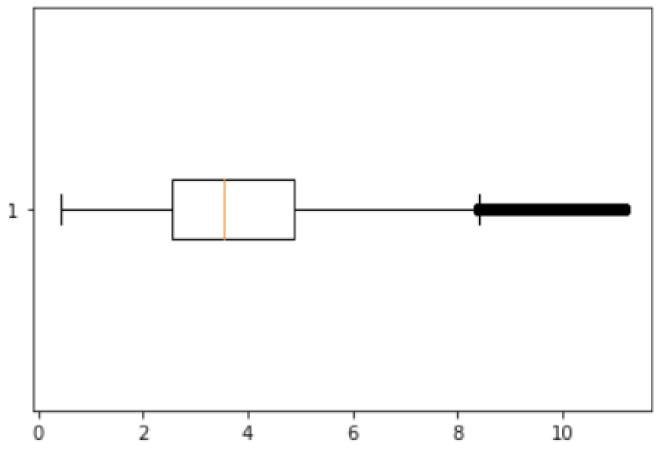


Fig. 3 Boxplot of average speed per trip after filtering outliers.

The resulting boxplot shows a more reasonable speed range **(Fig. 3)**.

I expected that there would be a strong relationship between distance and trip duration

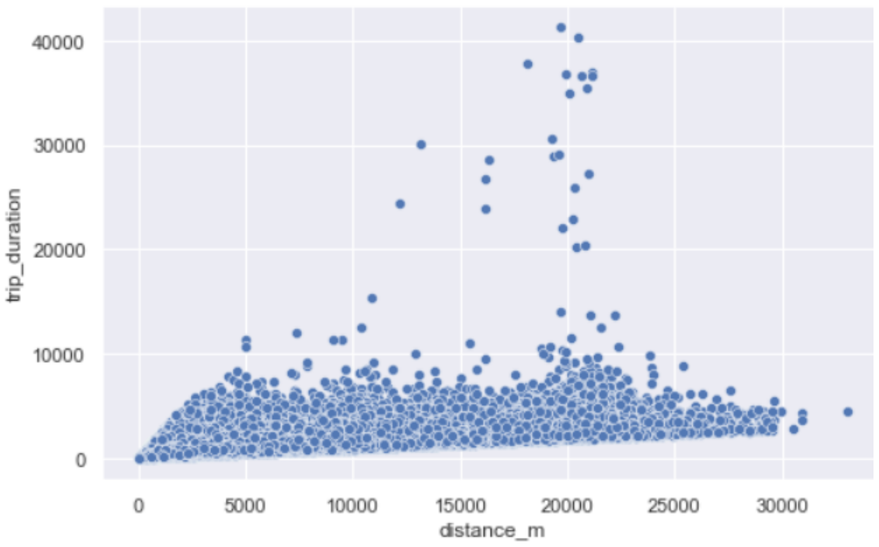


Fig. 4 Scatter plot of distance and distance.

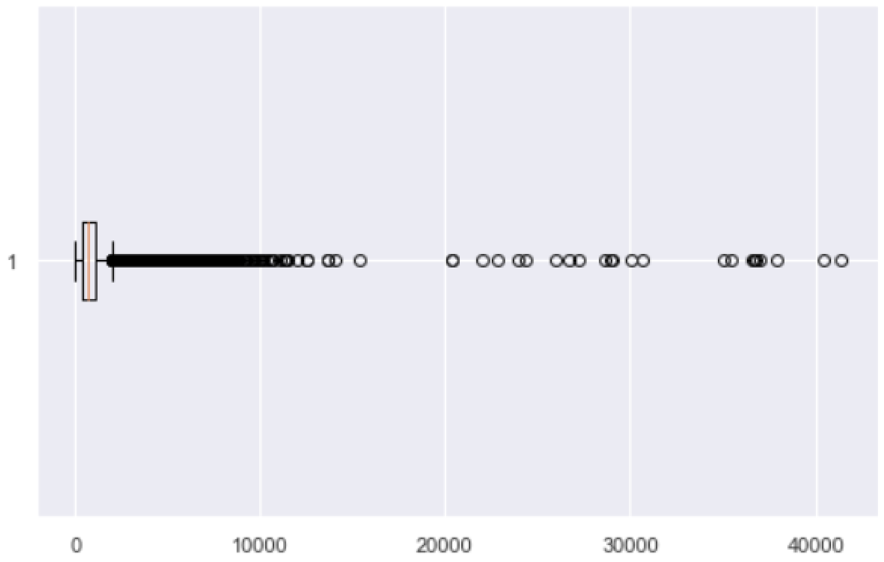


Fig. 5 Boxplot of trip_duration.

A scatterplot of distance vs trip duration **(Fig. 4)** reveals a noticeable positive slope, but with many outlying points indicating durations of several hours or more. A boxplot of trip duration **(Fig. 5)** gives a better idea of the distribution of these outliers in relation to the Google Maps indicates that the distance from JFK International Airport to Central Park is aproximately 16.9 miles, or 27 200 meters (rounded), and so trips in the 30 000 meter range, as shown in the scatterplot, seem reasonable. However, trips with extremely long duration seem likely to be the result of some extreme circumstance, so I will limit the results to trips with a duration of less than 2 hours, or 7200 seconds **(Fig. 6)**. This limit seems generous, even taking into account heavy traffic.

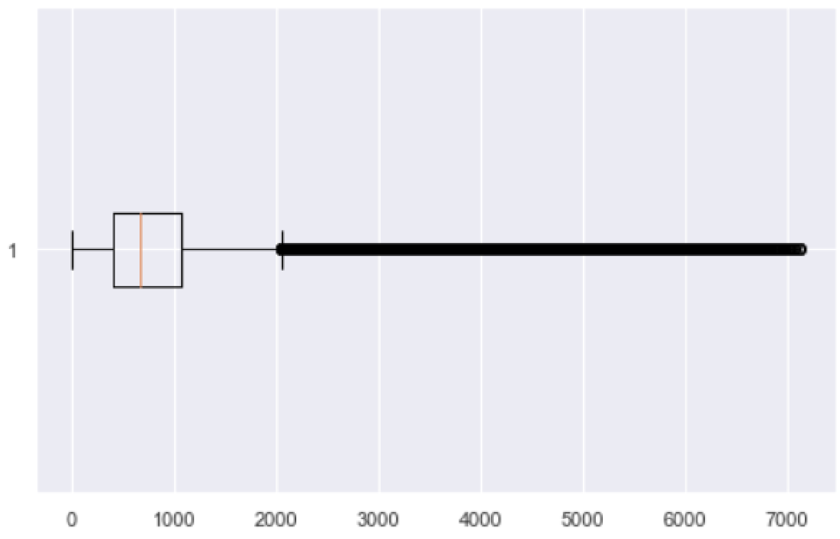


Fig. 6 Boxplot of trip_duration after setting maximum limit.

I then generated a new scatterplot of distance vs duration with a line of best fit **(Fig. 7)**.

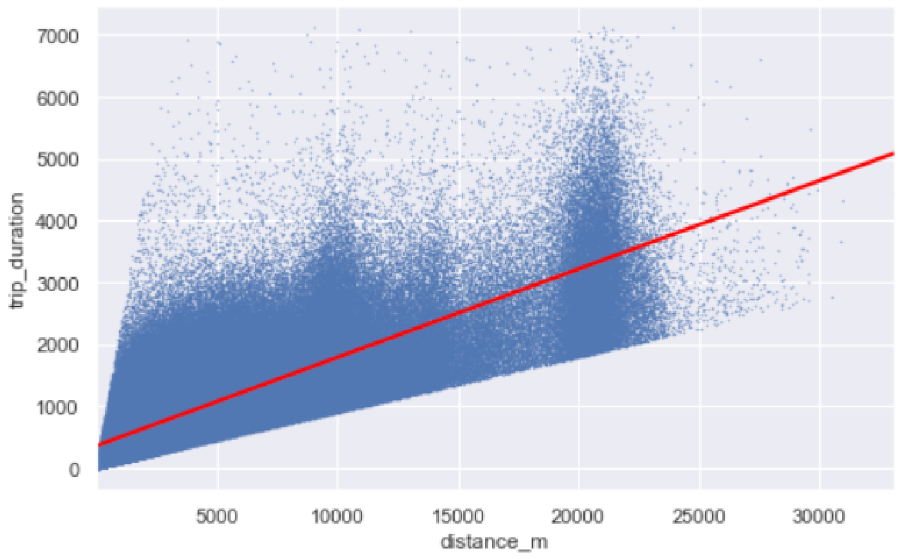
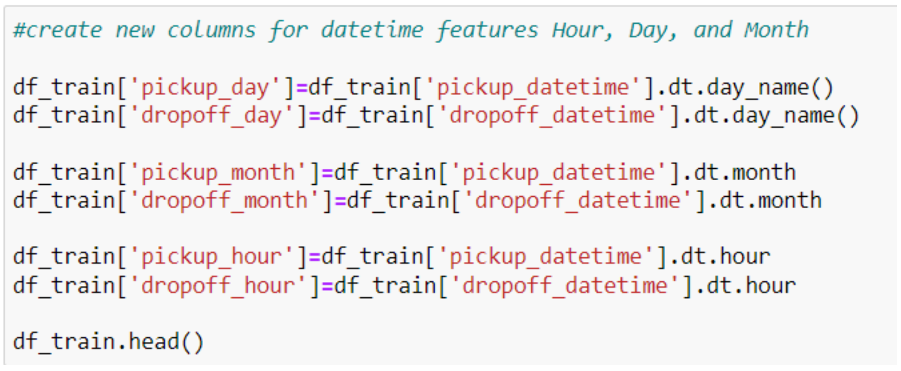


Fig. 7 Regression plot of distance and trip duration.

Next I used the pickup and dropoff datetime features to create new separate columns for the pickup and dropoff month, day, and hour. I then generated a bar graph showing the distribution of trips by hour **(Fig. 8)**.



```
#create new columns for datetime features Hour, Day, and Month

df_train['pickup_day']=df_train['pickup_datetime'].dt.day_name()
df_train['dropoff_day']=df_train['dropoff_datetime'].dt.day_name()

df_train['pickup_month']=df_train['pickup_datetime'].dt.month
df_train['dropoff_month']=df_train['dropoff_datetime'].dt.month

df_train['pickup_hour']=df_train['pickup_datetime'].dt.hour
df_train['dropoff_hour']=df_train['dropoff_datetime'].dt.hour

df_train.head()
```

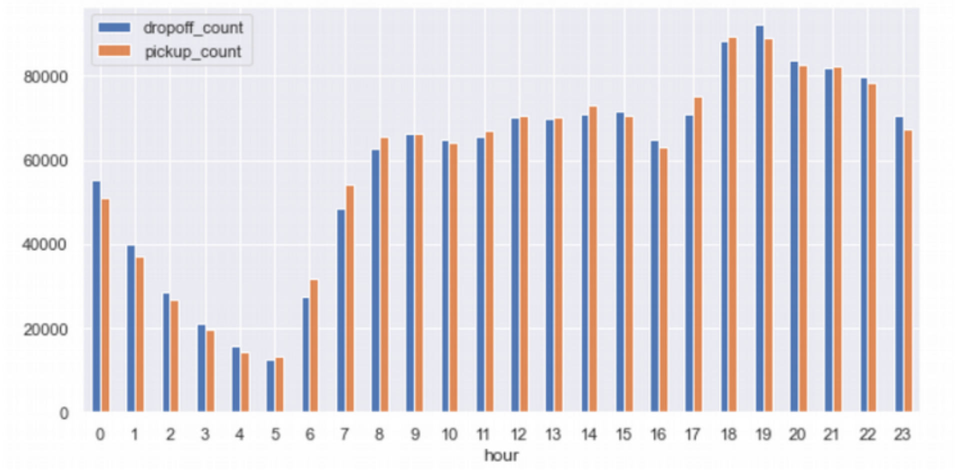


Fig. 8 Count of pickups and dropoffs by hour.

As one would expect, the count of pickups compared to dropoffs, as they correspond to the hour, are very similar. We can also see that number of trips peaks at 6 and 7 pm, while the lowest point is around 5 am.

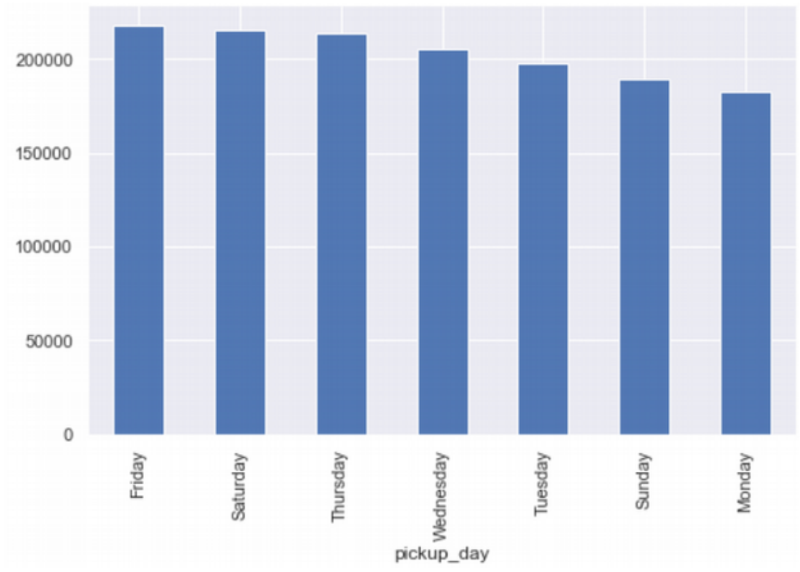


Fig. 9 Count of pickups by day.

If we look at the count by pickup day **(Fig. 9)**, however, we see relatively little variance throughout the week; though trips peak on Friday and Saturday, and then fall off on Sunday and Monday.

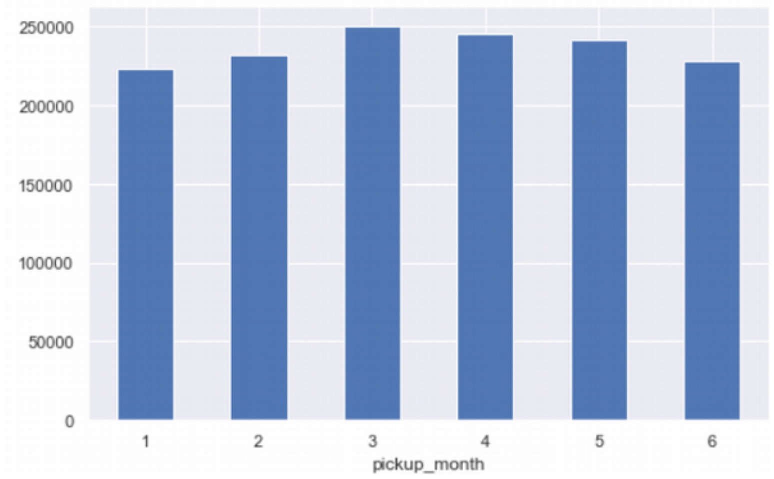


Fig. 10 Count of pickups by month.

Again, looking at the count by month **(Fig. 10)**, there is not a very high amount of variance, at least not from January through June. Perhaps one would expect a higher number of trips in colder weather; however the number of trips in January is roughly equal to (in fact, slightly lower than) the number of trips in June.

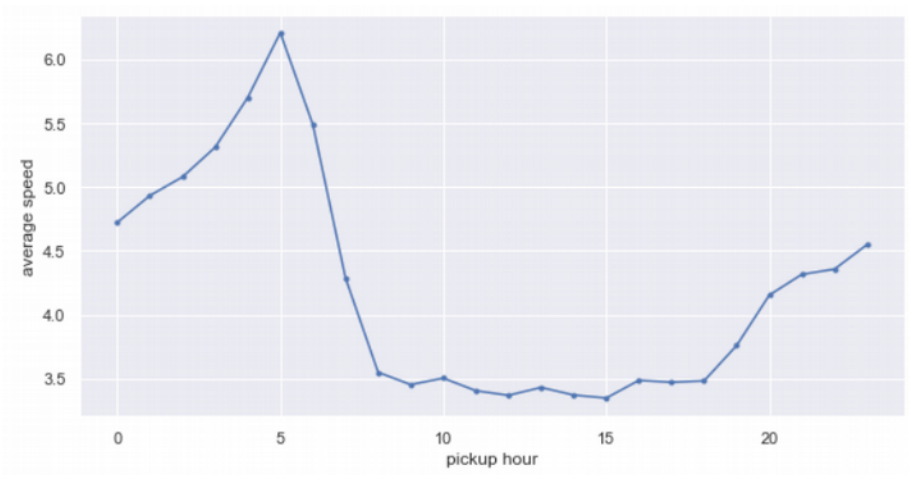


Fig. 11 Plot of average trip speed by pickup hour.

Above is a plot of the average trip speed (meters/second) by pickup hour **(Fig. 11)**. The average speed peaks at 5 am and falls precipitously until around 8 am, where it remains consistently low until 6 pm when it begins steadily climbing again. It is notable that the hour at which average speed is highest coincides with the lowest number of trips.

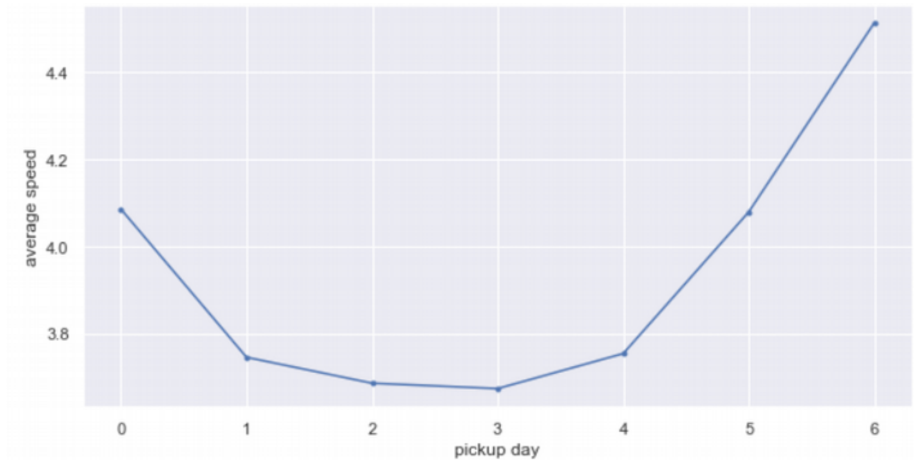


Fig. 12 Plot of average trip speed by pickup day.

We can also see how the average speed changes throughout the week **(Fig. 12)**. Note that the days of the week have been converted to numbers as follows:
0 = Monday
1 = Tuesday
2 = Wednesday
3 = Thursday
4 = Friday
5 = Saturday
6 = Sunday
Therefore we can see that the average trip speed is relatively low Tuesdays through Fridays, and peaks on Sunday.

Finally, I plotted the average trip duration by hour **(Fig. 13)**.

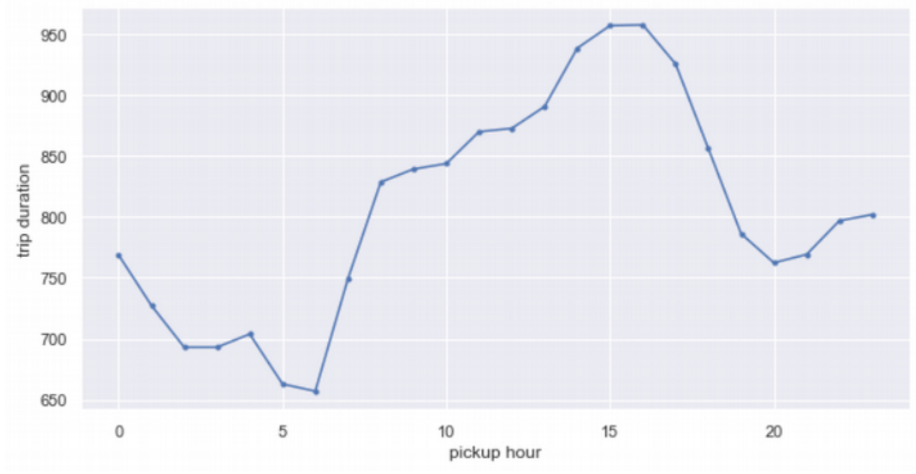
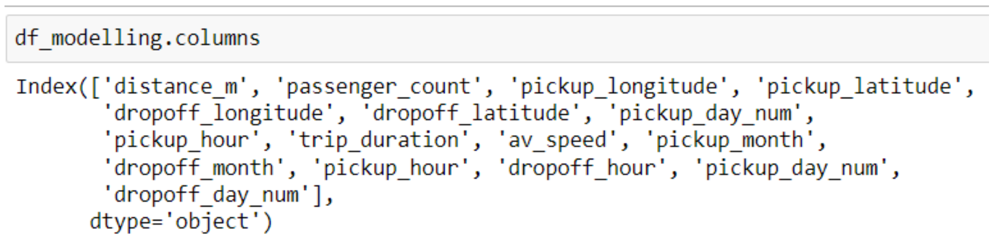


Fig. 13 Plot of average trip duration by pickup hour.

Here again, we see the point at which trip duration is shortest coincides roughly with the fewest trips and highest average trip speed.

3.2 Modelling

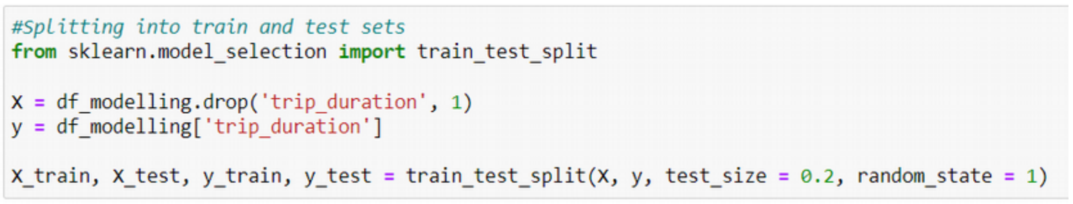
The next step is to split the data into training and testing sets. I created a new dataframe with all of the columns I wanted to include.



```
df_modelling.columns
```

```
Index(['distance_m', 'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'pickup_day_num',
       'pickup_hour', 'trip_duration', 'av_speed', 'pickup_month',
       'dropoff_month', 'pickup_hour', 'dropoff_hour', 'pickup_day_num',
       'dropoff_day_num'],
      dtype='object')
```

I then applied the train-test-split function.

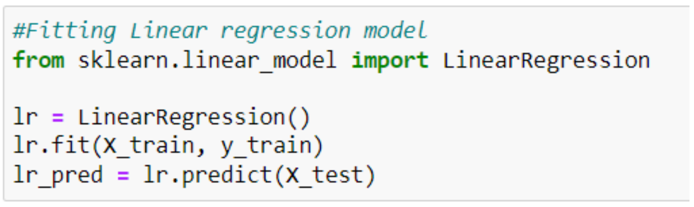


```
#Splitting into train and test sets
from sklearn.model_selection import train_test_split

X = df_modelling.drop('trip_duration', 1)
y = df_modelling['trip_duration']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

First I attempted to make a linear regression model, with the feature trip_duration as the target variable (y) and all of the other features in the dataframe as predictor variables.

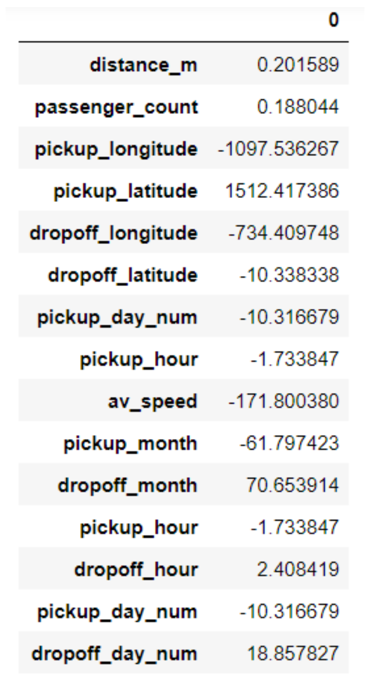


```
#Fitting Linear regression model
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
```

I applied several evaluation metrics which produced the following results:
Mean squared error: 69629.23649078414
Root mean squared error: 263.87352366386466
Mean absolute error: 168.921
R^2 value: 0.7998985423677614
Train score: 0.8337178380693502
Test score: 0.8335289615735982

I then generated a bar chart of the model coefficients **(Fig. 14)**.



	0
distance_m	0.201589
passenger_count	0.188044
pickup_longitude	-1097.536267
pickup_latitude	1512.417386
dropoff_longitude	-734.409748
dropoff_latitude	-10.338338
pickup_day_num	-10.316679
pickup_hour	-1.733847
av_speed	-171.800380
pickup_month	-61.797423
dropoff_month	70.653914
pickup_hour	-1.733847
dropoff_hour	2.408419
pickup_day_num	-10.316679
dropoff_day_num	18.857827

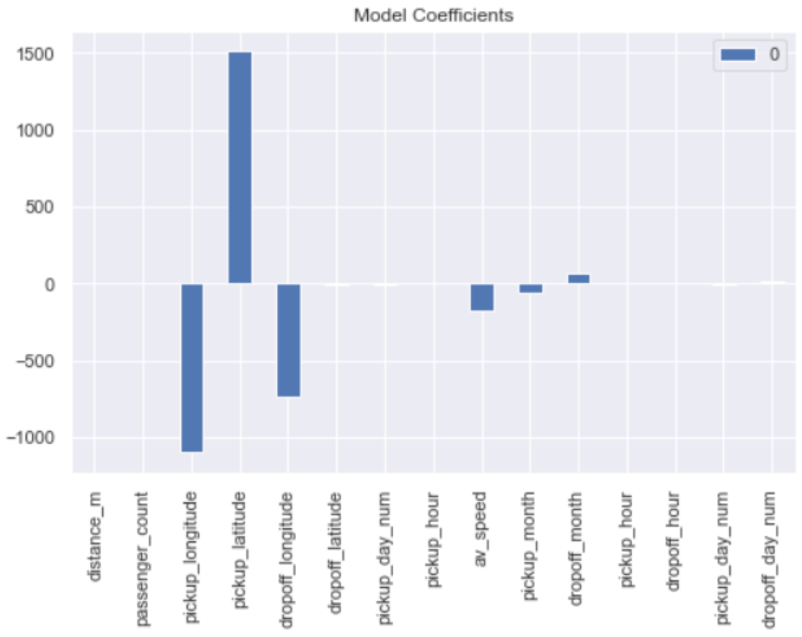


Fig. 14 Plot of linear regression model coefficients

Apparently the most significant factors are the pickup coordinates. whereas factors that I hypothesized would have the most influence on the target variable appear to be of relatively minor significance.

The odd coefficient values combined with the high error scores raise concerns over the accuracy of the model, therefore to improve predictive accuracy and to control potential for over-fitting, I fit a Random Forest Regression model.

```
Mean squared error: 5.843911192568555
Root mean squared error: 2.3752525579515953
Mean absolute error: 0.853
R^2 value: 0.9991823000067694
Train score: 0.9999894061627267
Test score: 0.9999238997237208
```

As with the linear regression model, I plotted a bar graph of the gini-importance scores of the features **(Fig. 15)**. The results show that the random forest model determined that the most significant features for predicting the trip duration are distance and average speed.

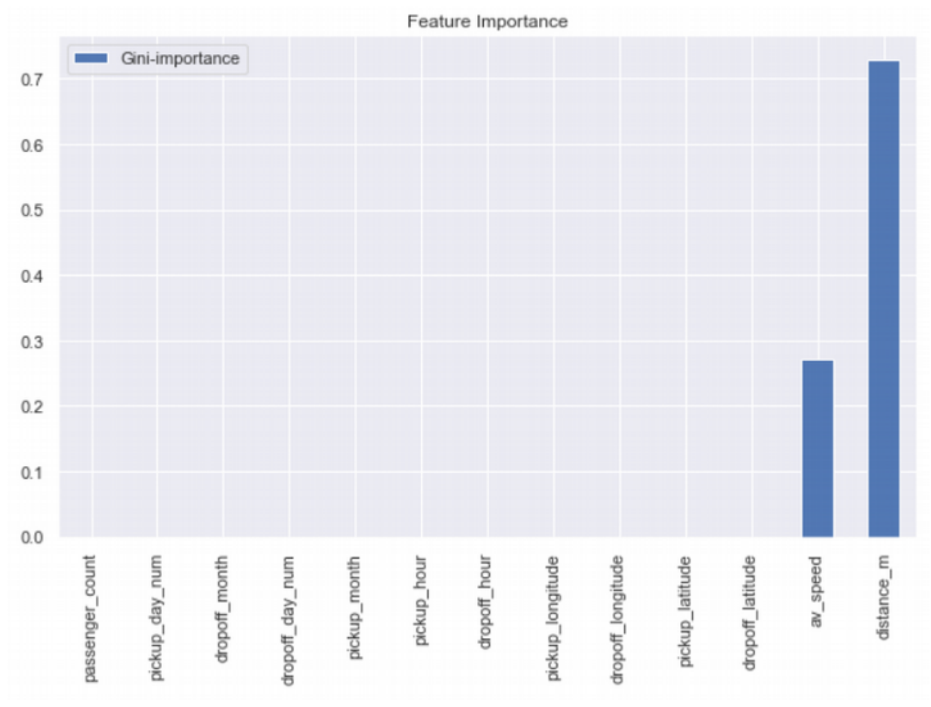


Fig. 15 Plot of gini-importances of Random Forest model features

For the sake of comparison, I also fit a gradient boost regressor model, and performed the same evaluation metrics which produced the following scores:

Mean squared error: 37.401723392653246
Root mean squared error: 6.115694841361303
Mean absolute error: 22.397
R^2 value: 0.9965143953345249
Train Score: 0.9967794014319582
Test Score: 0.9966555070587637

I then calculated and plotted the feature importances for the gradient boost model **(Fig. 16)**.

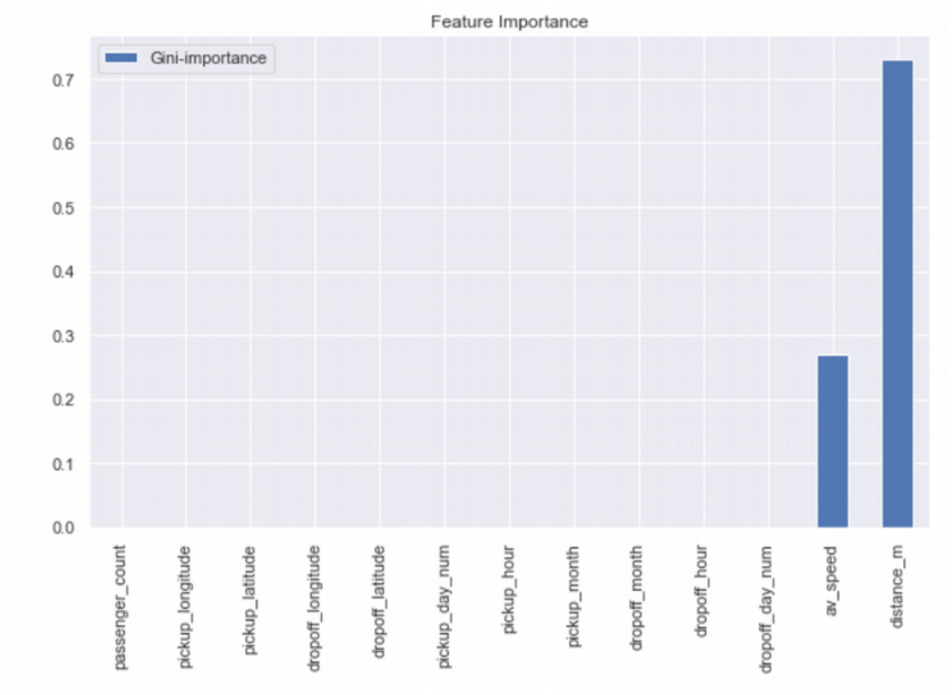


Fig. 16 Plot of gini-importances of Gradient Boost model features

We can see that the gradient boost model gave similar results for the feature importances as the random forest model.

5. DISCUSSION

As this project shows, there is a lot of preparation that must be done before we begin applying our machine learning tools. Though, in this case, the data did not require a lot of cleaning, it was necessary to fix some datatypes and remove outliers from sevreal features. I also made extensive use of visualizations to get a better understanding of the data, which helped make clear a number of interesting patterns in the data. During the feature engineering process, I was able to derive several new features, notably distance and average speed, which proved to be important predictors.

The results obtained by the evaluation metrics illustrate the importance of feature selection in a model's predictive accuracy. As the random forest regressor's method of averaging the predictions of multiple decision trees improves predictive accuracy and reduces the potential problem of overfitting, when compared to the linear regression model , both the random forest model and gradient boost regression model produced a lower mean squared error, a lower mean absolute error, and a higher r-squared value.

Furthermore, although the random forest model is unable to extrapolate from the data, unlike a linear model, the data we are using is bound by a number of limits. For example, distances are limited to the geographical boundaries of New York City, and trip duration has been capped at 2 hours; as these values are not beyond what exists in the training data, we do not have to worry about the random forest extrapolation problem.

6. CONCLUSION

From looking at the error values and scores for the different models, shown in section 3.2, both the gradient boost regression model and the random forest regression model performed well compared to the linear regression model. Of the three models, the random forest model had the best results; the lowest error scores and R^2 value closest to 1. Also, in contrast to the linear regression model, the gradient boost model and random forest model determined the most important predictors of trip duration were distance and average speed.