NYC Taxi Trip Time Prediction

Lakshmi Keerthana , Tito Varghese

Data Science Trainees,
Alma Better , Banglore

Abstract

Taxi cabs are both loved and hated by New Yorkers. They serve as a quick and easy means of transportation across the city. The downside with having an abundance of cabs results in traffic. Taxicabs are operated by private companies and licensed by the New York City Taxi and Limousine Commission and they have provided us a dataset that contain pickup time, drop-off time, geo- coordinates, number of passengers, trip duration and other variables. We shall predict the trip duration of the taxi for both users and drivers and make it easier to understand the trip duration, pricing range and route to prefer. The dataset is based on the 2016 NYC Yellow Cab trip record data made available in Big Query on Google Cloud Platform. The data was originally published by the NYC Taxi and Limousine Commission (TLC).

Problem Statement

For predicting the trip duration of NYC taxi, we will have to analyse the dataset.Our objectives were to clean,extract,analyse and to build a regression model based on the 2016 NYC yellow cab data set given to us-

Based on individual trip attributes, we should predict the duration of each trip in the test set. The training set contains 1458644 trip records.

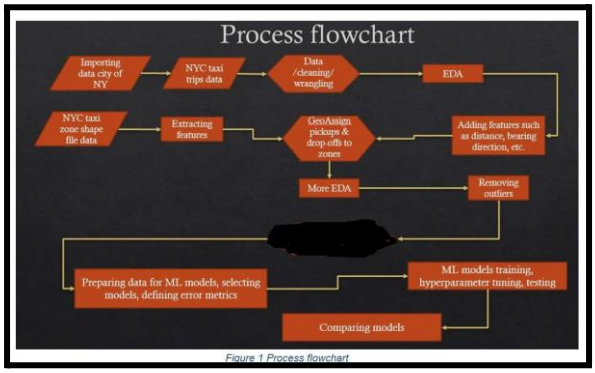


Figure 1 Process flowchart

Data fields

- id - a unique identifier for each trip
- vendor_id - a code indicating the provider associated with the trip record
- pickup_datetime - date and time when the meter was engaged
- dropoff_datetime - date and time when the meter was disengaged
- passenger_count - the number of passengers in the vehicle (driver entered value)
- pickup_longitude - the longitude where the meter was engaged
- pickup_latitude - the latitude where the meter was engaged
- dropoff_longitude - the longitude where the meter was disengaged
- dropoff_latitude - the latitude where the meter was disengaged
- store_and_fwd_flag - This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip
- trip_duration - duration of the trip in seconds

Introduction

This is our Regression Capstone Project, hence we will be looking into multiple regression models and try to come up with a best model at the end of this project. We are only focussing on all that algorithm which has been taught to us till now in our class. SVM,Time Series, Clustering and many more algos. still yet to be taught to us.

ML pipeline to be followed

1. Basic Dataset Understanding(Dimensionality,records,Data Types,5-point summary)
2. Data Cleaning
3. Exploratory Data Analysis
4. Feature Engineering
5. Feature Selection
6. Model Building
7. Evaluation
8. Hyperparameter tuning/cross Validation

1.Basic Dataset Understanding

We loaded the data from the given csv files. We checked the general information about data. We observethat the data contains 1458644 records and 11 features.

2. Data Cleaning

Data cleaning is one of the important parts. We remove the unwanted observations, fix the structural errors, manage the unwanted outliers and handle the missing data.

We had received a clean dataset without any missing values and duplicate rows.But handling Outliers was one of the main task inorder to fine tune our model accuracy.

One of the most important steps as part of data preprocessing is detecting and treating the outliers as they can negatively affect the analysis and the training process of a machine learning algorithm resulting in lower accuracy.

An outlier may occur due to the variability in the data, or due to experimental error/human error.

Box plots are a visual method to identify outliers. It is a data visualization plotting function. It shows the min, max, median, first quartile, and third quartile

The pickup latitudes and longitudes should be within the NYC boundary as defined-

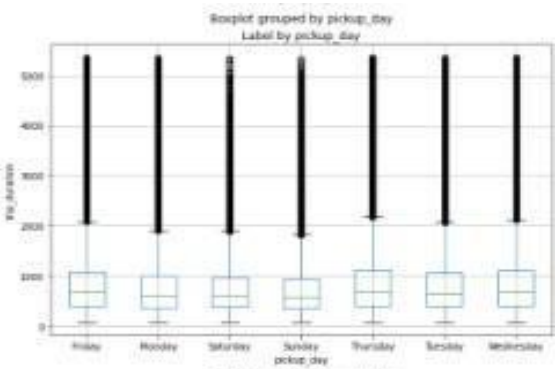
pickup latitude/longitude < **1th percentile value of the pickup latitude/longitude** and pickup latitude/longitude > **99.8th percentile value of the trip duration**

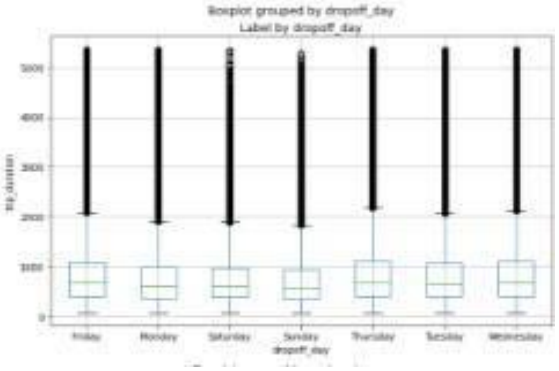
tripduration < **1th percentile value of the trip duration** and trip_duration > **99.8th percentile value of the trip duration**

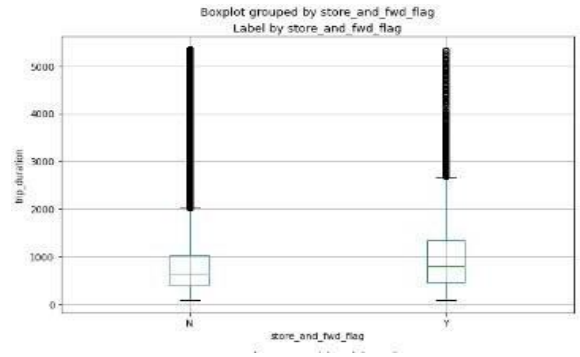
distance between the pickup and dropoff points is > **its 1th percentile value** and < its **99.8th percentile value.**

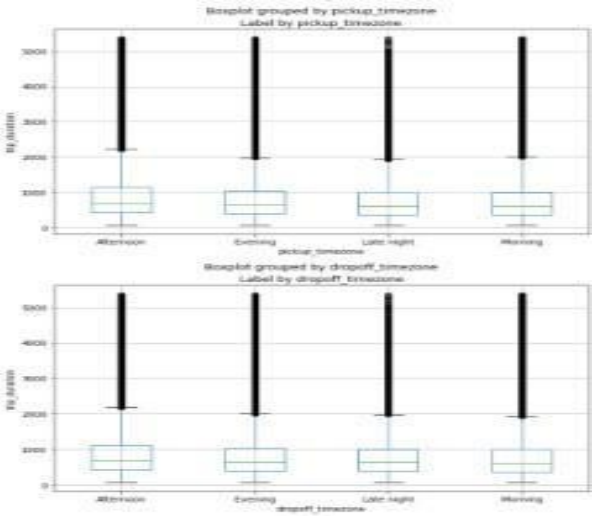
trip_speed between the pickup and dropoff points is > its **1th percentile value** and < its **99.8th percentile value.**

We have removed all the records which were not lies within these ranges.





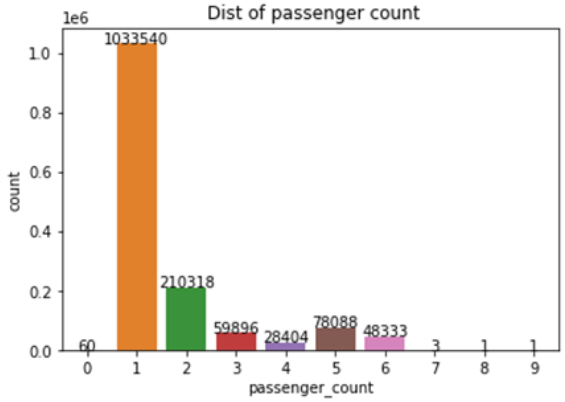




3.Exploratory Data Analysis

Univariate Analysis

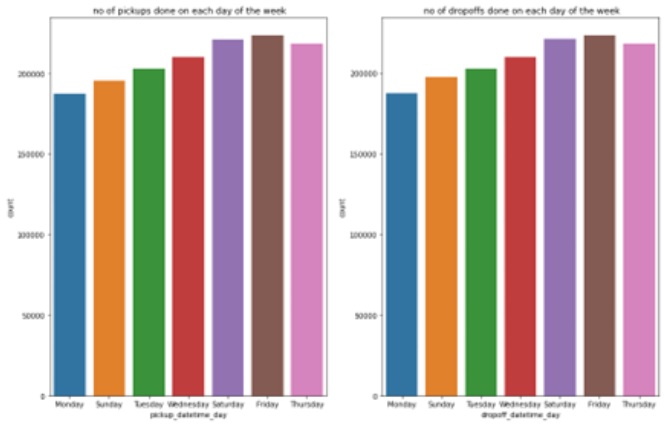
passenger count



The above plot tells us that the mostly the taxi trip passengers count is one.

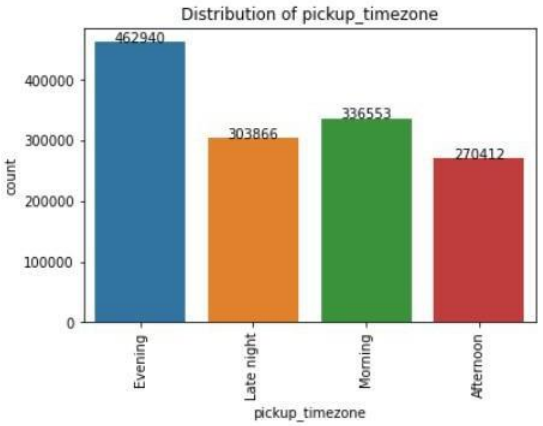
It depicts that taxi are mostly preferred by those who likes to travel alone.

pickup/dropoff day



The plot tells us that mostly on friday's we can see a high demand for taxi trip and the least number of taxi trip demand on Monday.

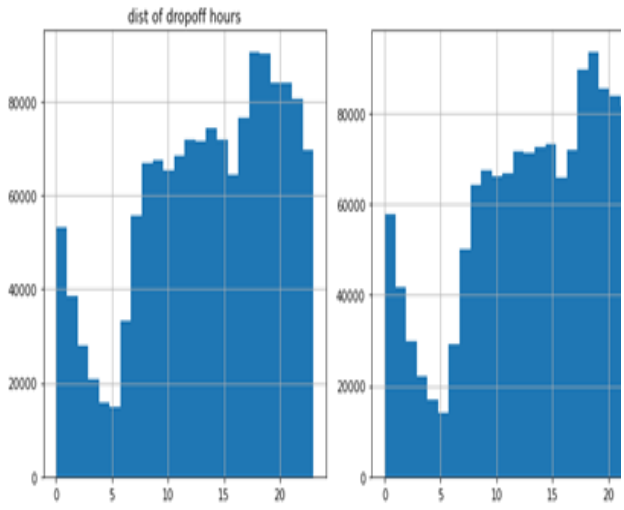
pickup_timezone

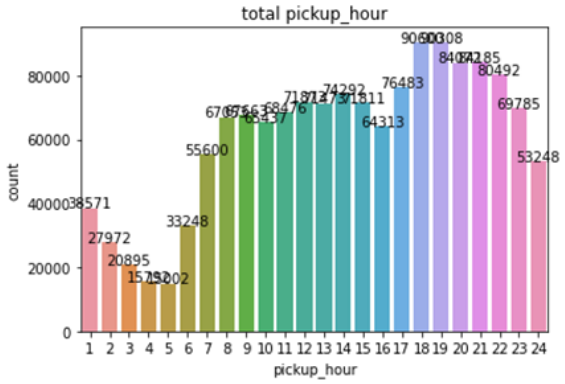


The high demand for taxi trip is in the evening timezone.

The least demand for taxi trip is in the afternoon timezone

pickup_hour and drop off hour

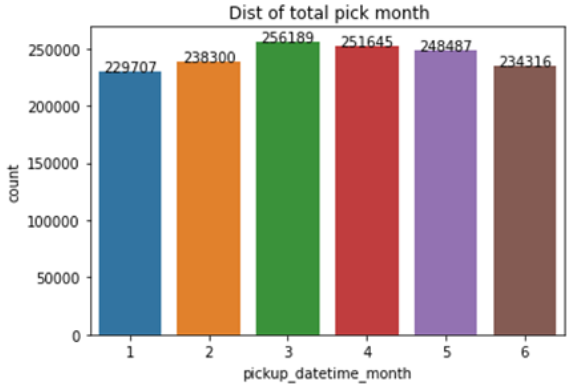


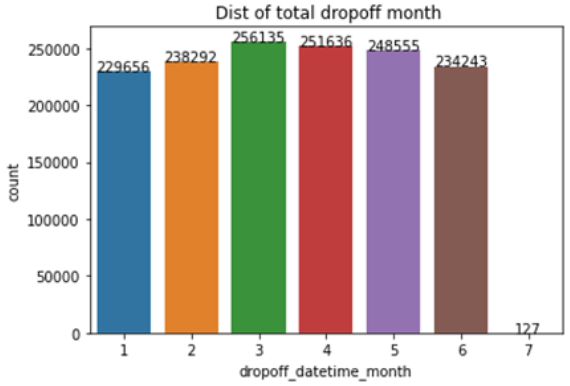


The most busy hours for taxi trip were between evening 18-22 hr

The least busy hours were between early morning 2-5 hr

pickup__month/Dropoff_month





The month of March has received the highest number of trips followed by April for both pickup/dropoff.

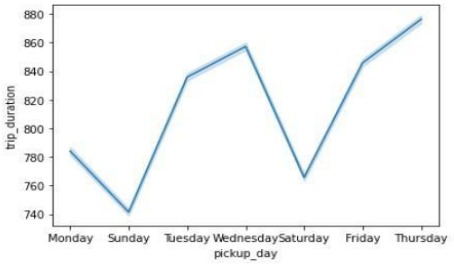
The least number of trips done in the month of January and July.

Bivariate Analysis

Trip Duration per different days

```
sns.lineplot(x='pickup_day',y='trip_duration',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f61048f7050>
```

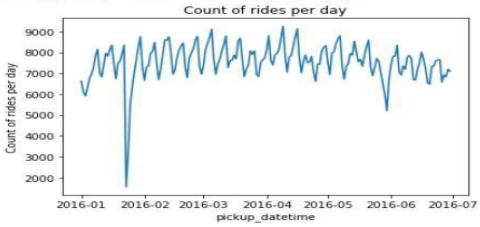


The trip duration is the maximum in Wednesday and lowest on Sunday

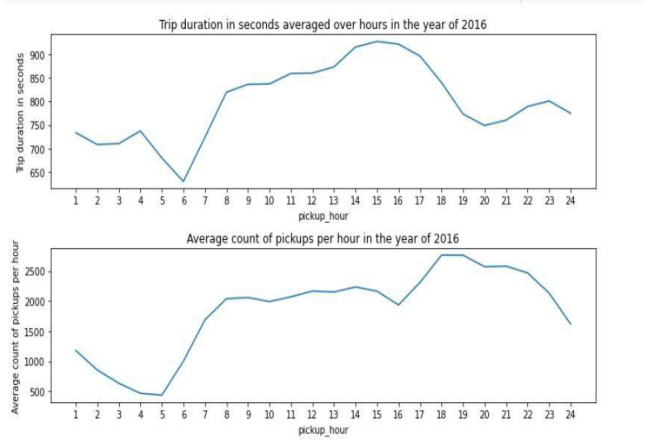
Number of trips per day

```
df.groupby(df.pickup_datetime.dt.date).size().plot()
plt.title('Count of rides per day')
plt.ylabel('Count of rides per day')
```

```
Text(0, 0.5, 'Count of rides per day')
```



The sudden dip in the taxi rides between the 20th and 26th Jan was because of the heavy snow that was observed during that period.



Rides increase steadily from 5am to 8am only to flatten out between 8am to 4pm and then again steeply increase from 4pm to 6pm to fall down later.

4. Feature Engineering

Step-1.Buliding New Features

We have derived few features from pickup/dropoff datetime columns after converting its datatype to datetime.

Few new features derived are-

pickup/dropoff month,day,weekday,hour,timezone and date.

We have derived a new feature (distance and trip_direction) using the available features in our dataset like geographical lattitude and longitude,

We have also derived new features like (time_diff_minutes) by subtracting dropoff_datetime and pickup_datetime which is highly correlated with our target variable.

```
df['time_diff_minutes']= df['dropoff_datetime']- df['pickup_datetime']
df['time_diff_minutes']= df['time_diff_minutes']/np.timedelta64(1,'m')
```

Another important feature derived trip_speed using trip_distance(km) and time_diff_minutes features. We have used 0.621 to convert the trip distance to miles and 0.00278 to convert seconds to hour.

```
#The trip_speed unit will be mph
def speed(x,y):
  z = (x*0.621)/(y*0.016667)
  return z
```

```
df['trip_speed']= df.apply(lambda x: speed(x['distance'],x['time_diff_minutes']),axis=1)
```

```
#Extract hour from pickup and dropoff datetime columns
df['pickup_hour']=df['pickup_datetime'].dt.hour
df['dropoff_hour']=df['dropoff_datetime'].dt.hour

#Extract day from pickup and dropoff datetime columns
df['pickup_day']=df['pickup_datetime'].dt.day_name()
df['dropoff_day']=df['dropoff_datetime'].dt.day_name()

#Extract date from pickup and dropoff datetime columns
df['pickup_date']=pd.DatetimeIndex(df['pickup_datetime']).day
df['dropoff_date']=pd.DatetimeIndex(df['dropoff_datetime']).day

#Extract month from pickup and dropoff datetime columns
df['pickup_month']=df['pickup_datetime'].dt.month
df['dropoff_month']=df['dropoff_datetime'].dt.month

#Extract weekday from pickup and dropoff datetime columns
df['pickup_weekday']=df['pickup_datetime'].dt.weekday
df['dropoff_weekday']=df['dropoff_datetime'].dt.weekday
```

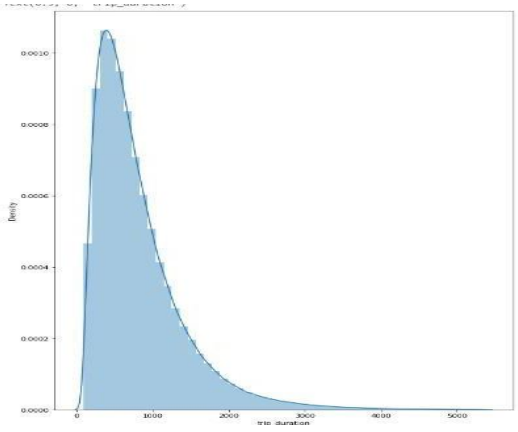
Step-3 Multivariate Normality check and Handling it with log transform

This method considers multiple variables in a data set to detect the skewness in the distribution. To check the distribution and handle the skewness if present.

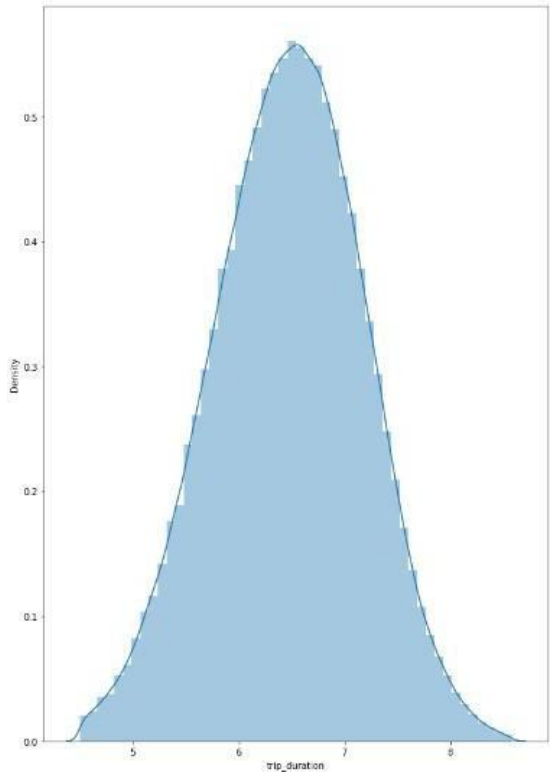
Data is skewed when its distribution curve is asymmetrical (as compared to a normal distribution curve that is perfectly symmetrical) and *skewness* is the measure of the asymmetry. We handle the skewness in numerical features using log Transform.

```
def time_zone(x):
    if x in range(6,12):
        return 'Morning'
    elif x in range(12,16):
        return 'Afternoon'
    elif x in range(16,22):
        return 'Evening'
    else:
        return 'Late night'
```

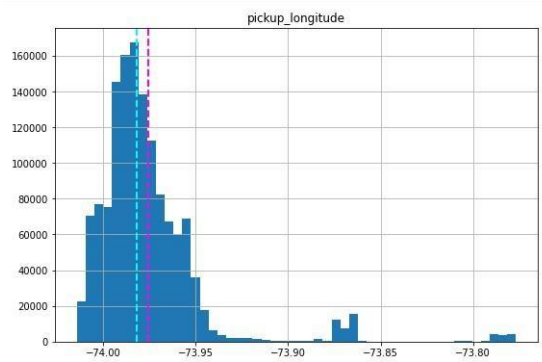
```
df['pickup_timezone']=df['pickup_hour'].apply(time_zone)
df['dropoff_timezone']=df['dropoff_hour'].apply(time_zone)
```

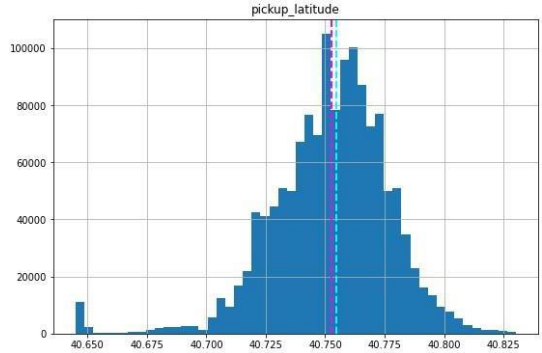


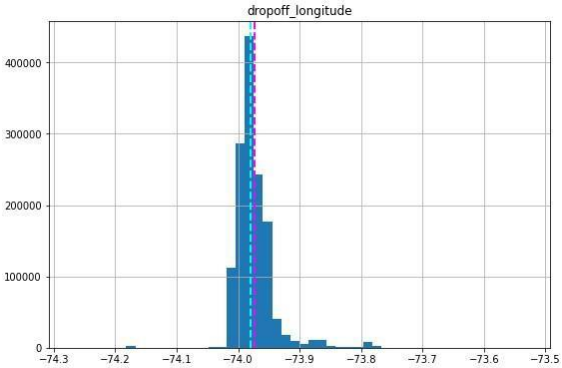
Using log transform to bring right skew distribution to normal transformation.

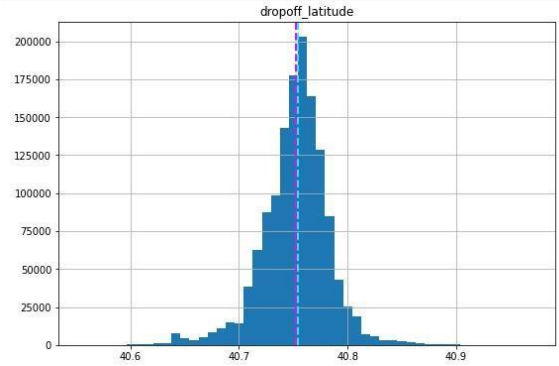


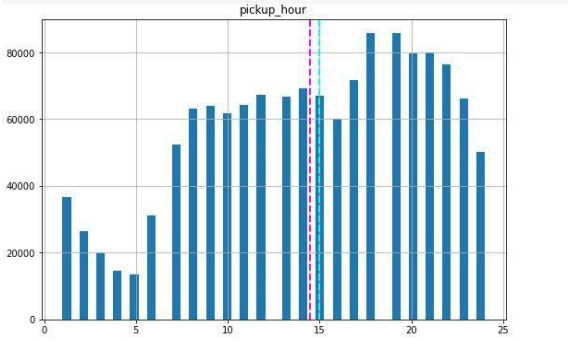
Plotting a bar plot for each numerical features

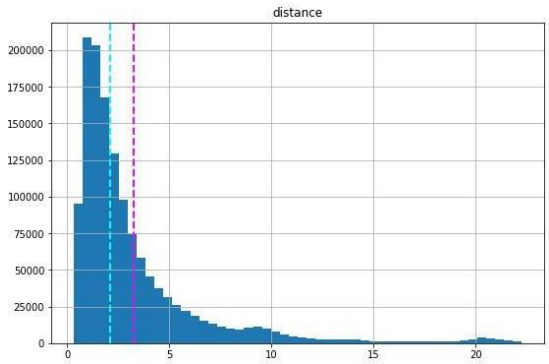






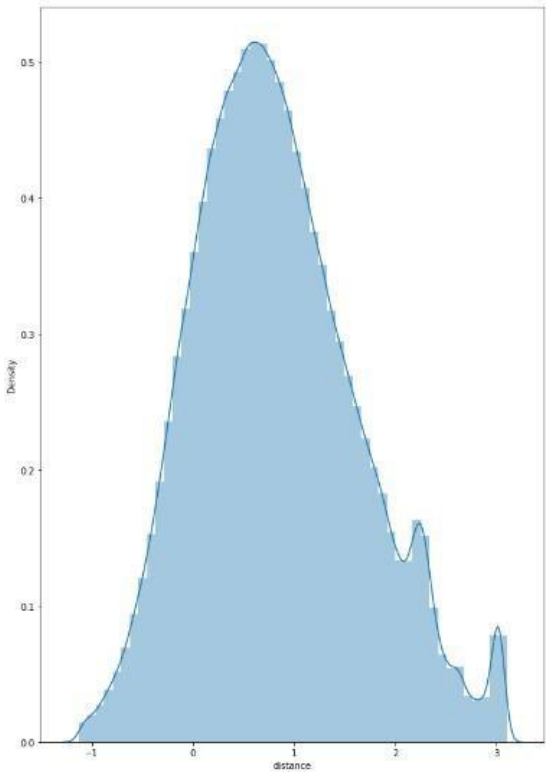






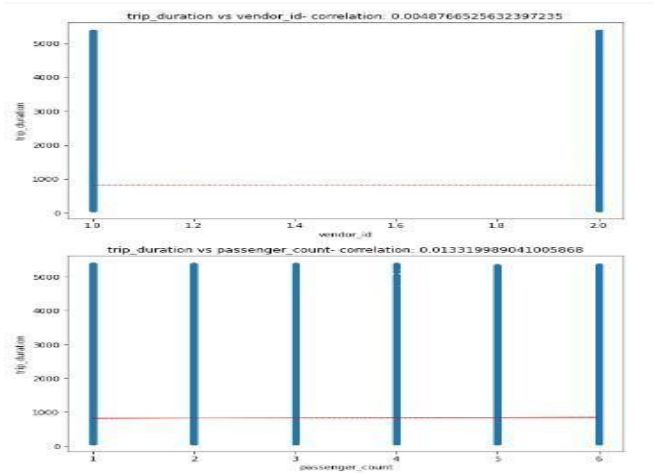
Treating the skewness in the feature using log transform

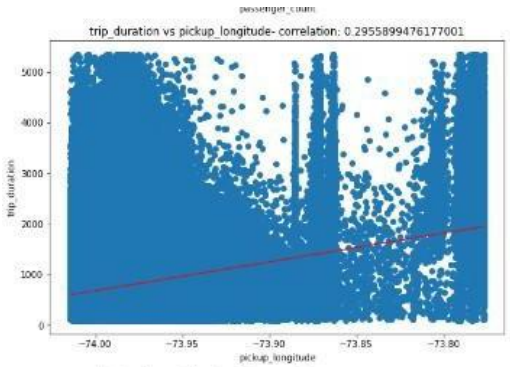
We can find positive skew in distance feature,therefore we will apply log transform to handle the skewness

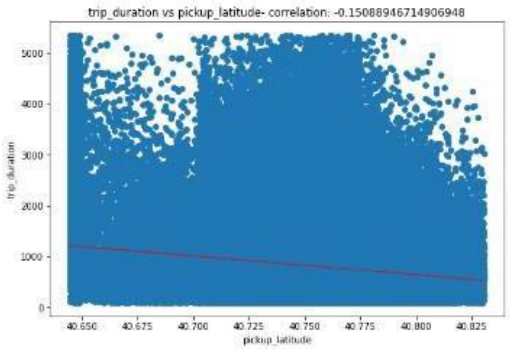


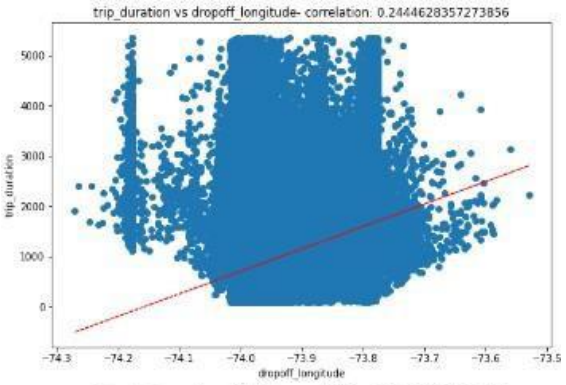
Step-4 Check for Linearity

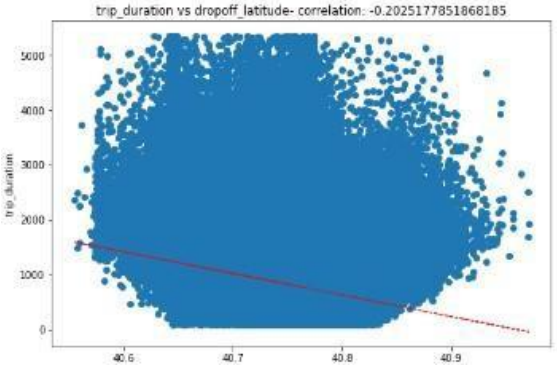
Linear regression needs the relationship between the independent and dependent variables to be linear. It is also important to check for outliers since linear regression is sensitive to outlier effects.

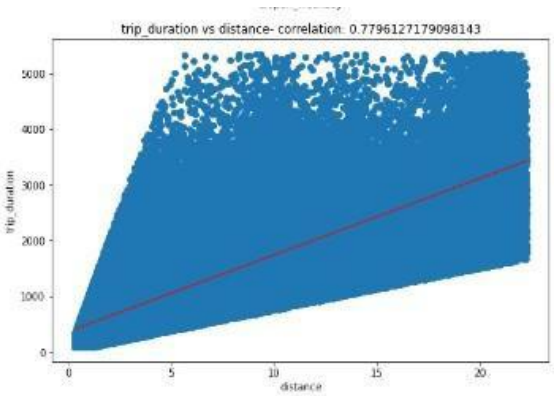


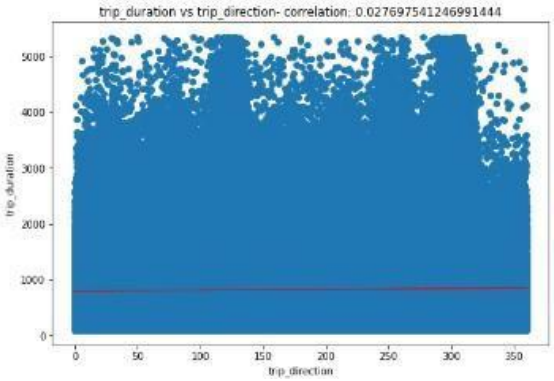












From above plot we can conclude that only distance were having a linear relationship with trip_duration.

Step-5 Encoding Categorical Features

One key challenge with most of the machine learning algorithm is the inability to work with categorical variables. They need the features to be converted to numeric form. The processing of transforming categorical features to numerical form is referred to as feature encoding. Feature encoding improves the performance of the model. It's a key step in machine learning modelling phase.

We have used One Hot Encoding in this project on features pickup_datetime_day and pickup_timezone.

```
# One hot encoding on pickup_timezone feature
df_copy = pd.get_dummies(df_copy,columns=["pickup_timezone"],prefix=["pickup_timezone"])
```

5. Feature Selection

Feature selection is nothing but a selection of required independent features. Selecting the important independent features which have more relation with the dependent feature will help to build a good model.

While building a machine learning model for real-life dataset, we come across a lot of features in the dataset and not all these features are important every time. Adding unnecessary features while training the model leads us to reduce the overall accuracy of the model, increase the complexity of the model and decrease the generalization capability of the model and makes the model biased.

Hence, feature selection is one of the important steps while building a machine learning model. Its goal is to find the best possible set of features for building a machine learning model. It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modelling problem.

```
features
```

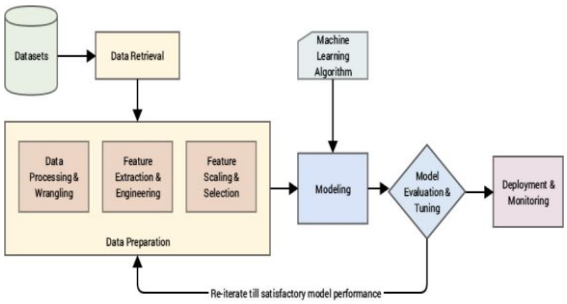
```
['vendor_id',
 'passenger_count',
 'pickup_longitude',
 'pickup_latitude',
 'dropoff_longitude',
 'dropoff_latitude',
 'pickup_weekday',
 'pickup_hour',
 'pickup_month',
 'pickup_date',
 'distance',
 'trip_direction',
 'pickup_timezone_Afternoon',
 'pickup_timezone_Evening',
 'pickup_timezone_Late night',
 'pickup_timezone_Morning']
```

6. Model Building

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, models are trained using labelled dataset, where

the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.



Steps Involved

- First Determine the type of training dataset
- Collect/Gather the labelled training data.
- Split the training dataset into training dataset, test dataset, and validation dataset.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

We have many variables such as the day of the month, weekday, hour of the day, bearing direction, zone_cluster, etc. which are not linear variables and can be difficult for a linear regression (LR) algorithm to model without first converting these variables to appropriate forms that can be fed to the LR models and understood by it. And, we also need to scale the data to prevent the

dominance of larger magnitude variables in LR models.

Another alternative is we can use non-linear methods such as decision trees to fit the data. Decision tree doesn't require us to convert the variables because it can split across any values of the variables and also, we don't need to scale the data when using trees because each variable is considered separately for calculating the gain at each branching.

But before moving onto decision trees or random forest let's check the performance of LR using only few significant variables such as the distance, hour and weekday. Since the hour and weekday are cyclical variables they were first transformed into sine and cosine terms using Fourier transform. Now, the LR can figure out that in reality 0th hour and 23rd hour are actually as much closer to each other as the 0th hour is to the 1st hour; same with the weekday. And we chose distance because based on the EDA and common intuition it can be assumed to be the most important variable in predicting the trip duration.

7. Regression Evaluation Metrics

Comparing these metrics:

MAE is the easiest to understand, because it's the average error.

MSE is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.

RMSE is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are loss functions, because we want to minimize them.

Linear Regression Model Evaluation Metrics Output

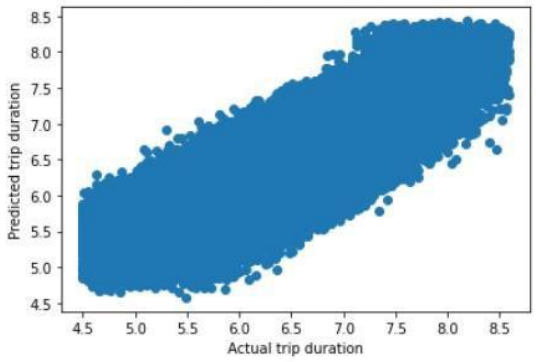
We have received r2score of 0.66 in train and test data for a linear regression model with few variable(distance,weekday and month).

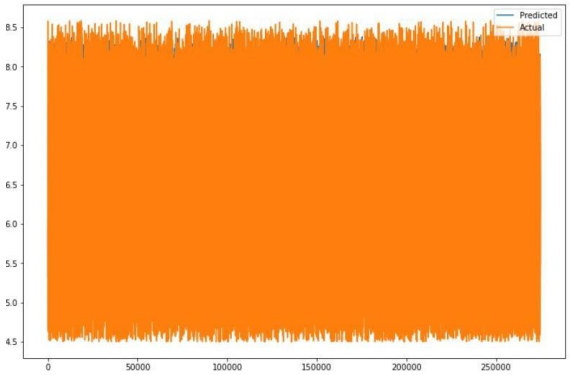
We consider this linear regression model has base model to compare other models.

Plotting a Scatter plot on Actual vs Predicted trip duration Values

```
plt.scatter((y_test), (y_pred_r))
plt.xlabel('Actual trip duration')
plt.ylabel('Predicted trip duration')
```

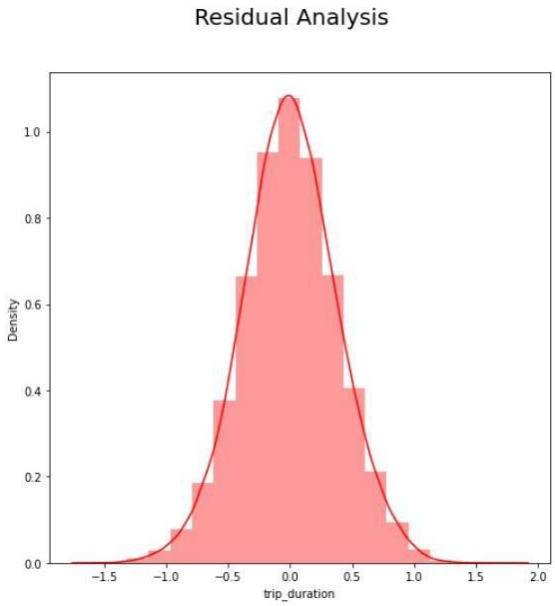
```
Text(0, 0.5, 'Predicted trip duration')
```





If we see above graph our prediction is quite good

Residuals

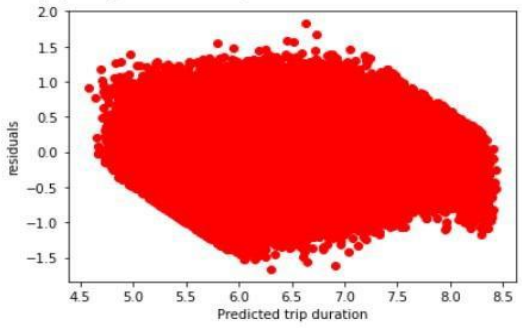


A residual is the vertical distance between a data point and the regression line. Each data point has one residual. They are positive if they are above the regression line and negative if they are below the regression line.

Heteroscedasticity

```
### Heteroscadacity
plt.scatter((y_pred_r),(y_test)-(y_pred_r),c='r')
plt.xlabel('Predicted trip duration')
plt.ylabel('residuals')
```

```
Text(0, 0.5, 'residuals')
```



Implementing Lasso regression

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters).

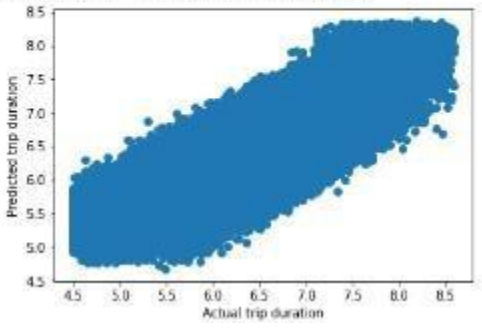
Lasso Regression Model Evaluation Metrics Output

```
MSE  = mean_squared_error((y_test), (y_pred_l))
print("MSE :" , MSE)

MAE=mean_absolute_error((y_test), (y_pred_l))
print("MAE :" ,MAE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

r2 = r2_score((y_test), (y_pred_l))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test), (y_pred_l)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

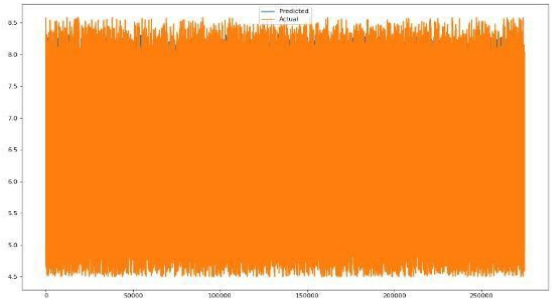
```
MSE : 0.14653434243345023
MAE : 0.3028767950018212
RMSE : 0.38279804392584116
R2 : 0.6991287349688717
Adjusted R2 :  0.6991112130380122
```

```
plt.scatter((y_test), (y_pred_l))
plt.xlabel('Actual trip duration')
plt.ylabel('Predicted trip duration')
```

```
Text(0, 0.5, 'Predicted trip duration')
```

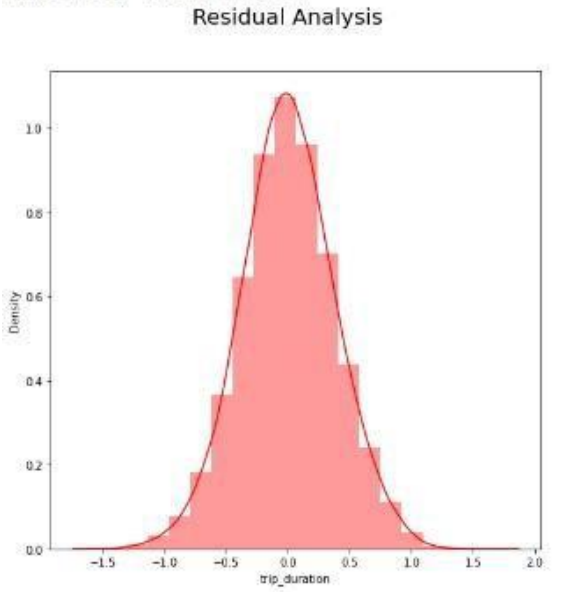



```
plt.figure(figsize=(15,10))
plt.plot((y_pred_l))
plt.plot(np.array((y_test)))
plt.legend(["Predicted","Actual"])
plt.show()
```




```
#Resuldual Analysis
fig=plt.figure(figsize=(8,8))

sns.distplot(((y_test)- (y_pred_l)),bins=20,color='r')

#Plot Label
fig.suptitle('Residual Analysis', fontsize = 20)
```

```
Text(0.5, 0.98, 'Residual Analysis')
```

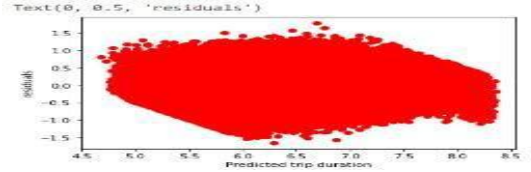


Heteroscedasticity:

Heteroscedasticity happens when the standard deviations of a predicted variable, monitored over different values of an independent variable or as related to prior time periods, are non-constant. With heteroscedasticity, the tell-tale sign upon visual inspection of the residual errors is that they will tend to fan out over time

```
### Heteroscadacity
plt.scatter((y_pred_l),(y_test)-(y_pred_l),c='r')
plt.xlabel('Predicted trip duration')
plt.ylabel('residuals')
```

```
Text(0, 0.5, 'residuals')
```



Ridge

Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.

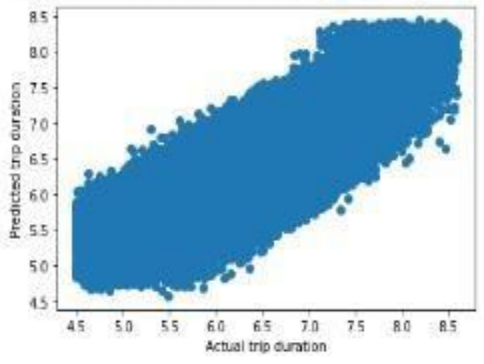
Ridge Regression Model Evaluation Metrics Output

```
MSE  = mean_squared_error((y_test), (y_pred_r))
print("MSE :" , MSE)

MAE=mean_absolute_error((y_test), (y_pred_r))
print("MAE :" ,MAE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

r2 = r2_score((y_test), (y_pred_r))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test), (y_pred_r)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

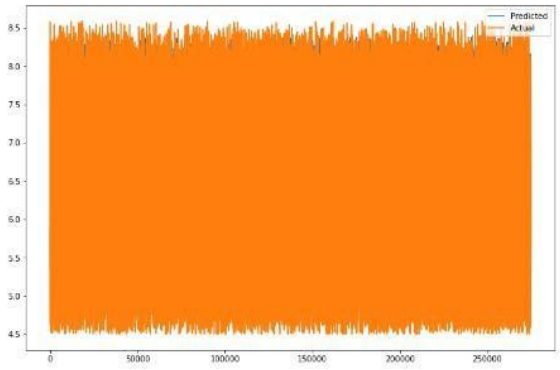
```
MSE : 0.1461178096529579
MAE : 0.3024749280616668
RMSE : 0.38225359338813545
R2 : 0.6999839798385212
Adjusted R2 :  0.6999665077148158
```

```
plt.scatter((y_test), (y_pred_r))
plt.xlabel('Actual trip duration')
plt.ylabel('Predicted trip duration')
```

```
Text(0, 0.5, 'Predicted trip duration')
```

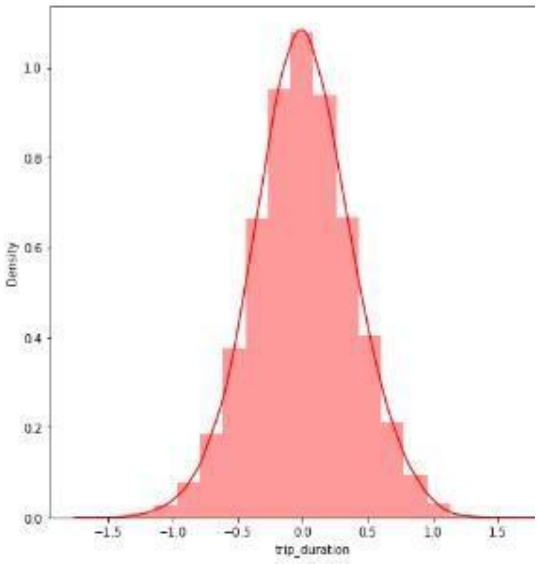



```
plt.figure(figsize=(12,8))
plt.plot((y_pred_r))
plt.plot((np.array(y_test)))
plt.legend(["Predicted","Actual"])
plt.show()
```



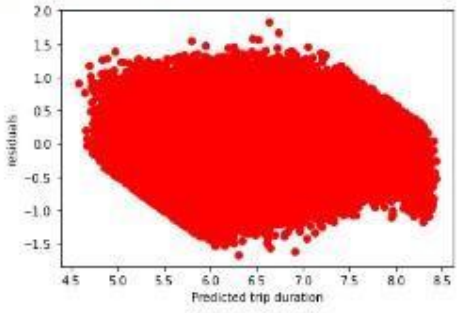

```
Text(0.5, 0.98, 'Residual Analysis')
```

Residual Analysis



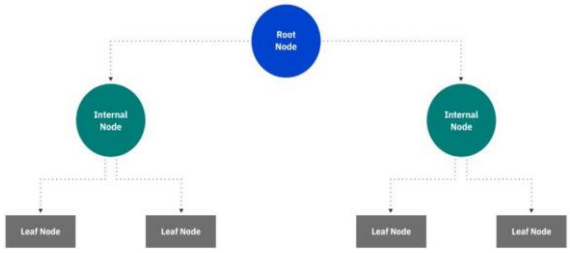

```
### Heteroscadacity
plt.scatter((y_pred_r),(y_test)-(y_pred_r),c='r')
plt.xlabel('Predicted trip duration')
plt.ylabel('residuals')
```

```
Text(0, 0.5, 'residuals')
```



Decision Tree:

A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.



As you can see from the diagram above, a decision tree starts with a root node, which does not have any incoming branches. The outgoing branches from the root node then feed into the internal nodes, also known as decision nodes. Based on the available features, both node types conduct evaluations to form homogenous subsets, which are denoted by leaf nodes, or terminal nodes. The leaf nodes represent all the possible outcomes within the dataset.

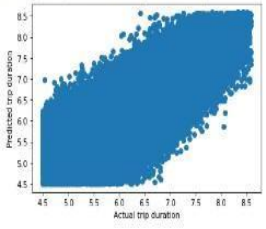
Decision Tree Regressor Model Evaluation Metrics Output

```
#Evaluating the model using regression metrics
MSE  = mean_squared_error((y_test), (y_pred_des))
print("MSE :" , MSE)

MAE=mean_absolute_error((y_test), (y_pred_des))
print("MAE :" ,MAE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

r2 = r2_score((y_test), (y_pred_des))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test), (y_pred_des)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shape[1]-1)))
```

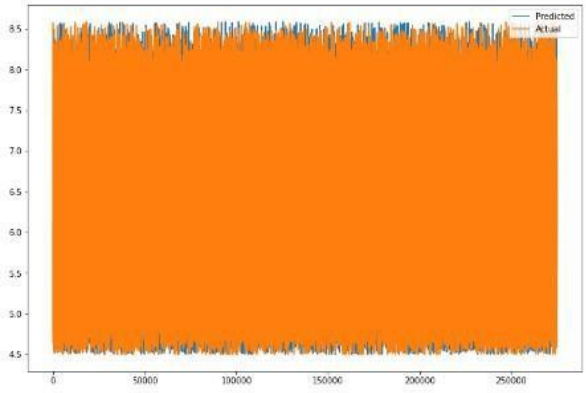
```
MSE : 0.16575605746694566
MAE : 0.3117404688876424
RMSE : 0.4071314989864401
R2 : 0.6596617975796237
Adjusted R2 :  0.6596419772007946
```

```
#Scatter plot vs Actual & Predicted trip duration Values
plt.scatter((y_test), (y_pred_des))
plt.xlabel('Actual trip duration')
plt.ylabel('Predicted trip duration')
```

```
Text(0, 0.5, 'Predicted trip duration')
```




```
plt.figure(figsize=(12,8))
plt.plot((y_pred_des))
plt.plot((np.array(y_test)))
plt.legend(["Predicted","Actual"])
plt.show()
```



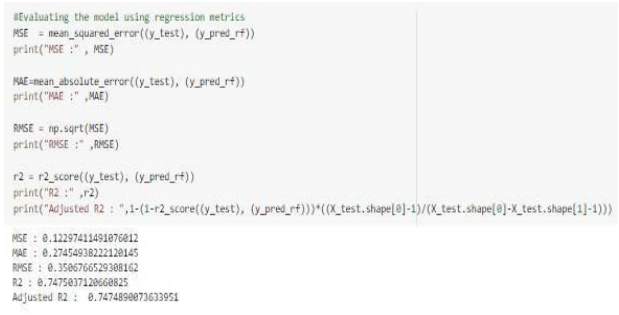
Random Forest

Random forest is a technique used in modeling predictions and behavior analysis and is built on decision trees. It contains many decision trees

representing a distinct instance of the classification of data input into the random forest. The random forest technique considers the instances individually, taking the one with the majority of votes as the selected prediction.

The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees.Using RandomForest Ensemble technique to predict the trip_duration after applying hyperparameter tuning with help of RandomizedSearchcv (cross-validation technique).

Random Forest Regressor Model Evaluation Metrics Output



The training R2 score was 0.74 and the test R2 score was 0.74 as wel, so we can say that the model is not overfitting the data.

Again, we can see considerable decrease in the error terms from the basemodel as well as the LR model. But the gain in accuracy comes with a loss in time complexity. It took around 90 minutes to tune the model and around 10 minutes to fit the best model on to the data set which is much slower than the LR (which only used 3 variables).

XGBOOST

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. It is an optimized distributed gradient boosting library.

XGBoost is well known to provide better solutions than other machine learning algorithms. Let's see if we get improved performance compared to the RF model

Using XGboost Ensemble technique to predict the trip_duration after applying hyperparameter tuning with help of RandomizedSearchcv (cross-validation technique)

Xgboost Regressor Model Evaluation Metrics Output

```
#Evaluating the model using regression metrics
MSE  = mean_squared_error((y_test), (y_pred_xg))
print("MSE :" , MSE)

MAE=mean_absolute_error((y_test), (y_pred_xg))
print("MAE :" ,MAE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

r2 = r2_score((y_test), (y_pred_xg))
print("R2 :" ,r2)
print("Adjusted R2 : ",1-(1-r2_score((y_test), (y_pred_xg)))*((X_test.shape[0]-1)/(X_test.shape[0]-X_test.shap
```

```
MSE : 0.07966303574784456
MAE : 0.21712430689740453
RMSE : 0.2822464096279075
R2 : 0.8364320749413443
Adjusted R2 :  0.8364225491866218
```

Our model isn't overfit.

We can see considerable decrease in the error terms from the baseline as well as the LR and RF models. Also, the predicted values are not limited to any trip duration range

Comparing negative mean square error in different models and plotting it on a box plot

Ridge

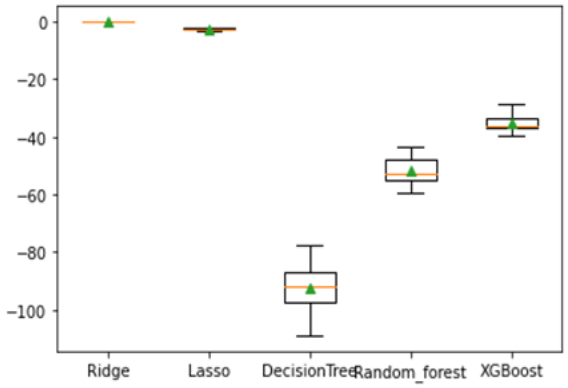
Lasso

Decision Tree

Random Forest

XGBoost

```
>Ridge -0.165 (0.013)
>Lasso -2.596 (0.238)
>DecisionTree -92.321 (7.021)
>Random_forest -51.579 (4.712)
>XGBoost -35.370 (2.778)
```



Based on negative mean square error metrics

We can say from looking at the boxplot that Ridge and Lasso Models were the Best Model.

The second best model is Xgboost model followed by Random Forest Model giving third least negative mean square error.

The Decision Tree Model has given the highest negative mean square error and hence underperformed model

8. HyperParameter Tuning

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.

There is another kind of parameter, known as ***Hyperparameters***, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are:

GridSearchCV

In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values.

RandomizedSearchCV

RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. This approach reduces unnecessary computation.

In this project,we have used RandomisedSearchCv for finding the best parameters for our regression model algorithm mainly used in RandomForest and Xgboost model to get the best max_depth parameter and n_estimator values.

The best parameter we got from RandomisedSearchCv for max_depth is 8 and the n_estimators value as 100 for both the models(RandomForest & Xgboost).

9. Conclusion

1. We can also use stacking algorithm over here to find the accuracy,but due to the high computation time we didn't used it.

2.We got the best model accuracy in Xgboost model,r2 score of 0.84 in training set and 0.83 in test data..The RMSE score of Xgboost model is 0.282

3.Our second best model is Random Forest based with a r2 score of 0.7485 accuracy in training and 0.7475 accuracy in test data. The RMSE score of random forest is 0.350

4.Our base model (Linear Regression) gave us a r2_score of 0.6499 in train and test data and keep this model accuracy in reference to compare other model.

5.And, the most important variables other than the distance between the pickup and dropoff locations, are:

- hour of the day
- weekday
- pickup and dropoff lat lons
- bearing (trip direction)
- even the day of the month

6. We can also do Model Explainability of random forest and XGboost model over here using SHAP, LIME or ELI5.But our team intention was not to make it lengthier hence we restricted ourselves till here.

10. References

- https://www.geeksforgeeks.org/feature-selection-techniques-in-machine-learning/
- https://scikit-learn.org/stable/index.html
- https://en.wikipedia.org/wiki/Taxis_of_New_York_City
- https://www.ibm.com/cloud/learn/random-forest
- https://www.geeksforgeeks.org/xgboost/

- https://www.ibm.com/cloud/learn/machine-learning

- https://www.simplilearn.com/tutorials/machine-learning-tutorial/supervised-machine-learning

Challenges

- Extracting new features from existing features were a bit tedious job to do.
- Handling Outliers in Independent features was bit lengthy process
- There were few irrelevant data records present in the dataset
- The Randomised Search cv took nearly more than a hour to run ,so it was really time consuming

Future Work

- Implement deep learning models to fit the data. Since we have millions of taxi trips in this data we can use multi layer neural network models to see if they give better performance compared to the other models tried in this notebook.
- Build a simple web application such that a user can input the pickup and dropoff latitudes and longitudes and the web app will throw out the estimated trip duration (using our best model).
- We can also do little more hyperparameter tuning using different parameters in GridSearchCv with a optimal value of crossvalidation ,inorder to fine tune the model and to improve the model accuracy.