PROJECT REPORT ON CAB FARE PREDICTION

SUBMITTED BY,

NAMAN SHARMA

Index

Chapter 1

Introduction

We have been given a dataset to predict the cab rental fare amount for the trips in test dataset. We have achieved this through variuos Exploratory Data Analysis techniques and using varioud model in order to get the best fitted model.

1.1 Problem Statement:

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected the historical data from your pilot project and now have a requirement to apply analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

For this we have been given with two datasets, test and train. Both the datasets contains same number of variables but different sizes. We have to analyse the train dataset and train a model that is able to predict the test data and with least error.

1.2 Data:

Both the dataset ontains 6 varaibles:

pickup_datetime - timestamp value indicating when the cab ride started.
pickup_longitude - float for longitude coordinate of where the cab ride started.
pickup_latitude - float for latitude coordinate of where the cab ride started.
dropoff_longitude - float for longitude coordinate of where the cab ride ended
dropoff_latitude - float for latitude coordinate of where the cab ride ended
passenger_count - an integer indicating the number of passengers in the cab ride
fare_amount [only in train dataset]- float indicating the price of each trip.

We have few missing values, and there are few outliers in the dataset which we deal further. As our traget variable is a countinous variable, that mean this is a **regression problem**.

Chapter 2 Brief Description of Each Step

Pre Processing

In the we have done univariatre analysis and tried to remove as much as outliers and Nan values from the data set. As some of the values seems relevant according to data, so we have kept it untill bivariate analysis to gather enough proofs to rmove those values.

We have done pre processing in few parts throughout our analysis. Removing all the unwanted variables and removing outliers in the starting of data is not the way that we have choosed to analyse thi dataset. Though we have removed NaN in the starting but have kept few variables and hav done feature engineering and variable scaling after EDA.

Overall steps that we have performed are:

- Data Exploration,knowing data variable types
- Outlier analysis
- Missing value treatment
- Univariate and Bivariate Analysis
- Feature Engineering
- Feature Selection
- Feature Scaling which includes log transformation
- Multicolinearity Check

Modeling

Modeling is simply a system for mapping inputs with outputs. Means we use the variables or features int the dataset as inputs and try to connect it with the output or the target variable. There are mmodels for regression type problems but we have used few most comman and most effective ones.

- Linear Regression
- Polynoial Regression
- Decision Tree
- Random Forest
- XGBoost
- Hyper-Parameter Tuning

Evaluation Matrix

We have at first used many matrices in order to see even a minute improvemnt in different model sets. But at last we have summarised to 3 Evaluation Matices:

- R^2 (R square)
- MAPE (Mean Absolute Percentage Error)
- RMSE (Root Mean Square Error)
- AIC as goodness of fit

Chapter 3: Data Pre Processing

3.1 Data Exploration

In this project we have two datasets ,test and train namely. We have to analyse the datasets and train our models as such that it redicts the fare amount of all the rides in the test dataset with least error. One dataset , train, contains our target variable ‘fare amount ’ and another dataset set dont and is considered as test dataset. We start data pre-processing by checking the shape of each dataset.

```
In [3]: train = pd.read_csv("train_cab.csv")
   ...: test = pd.read_csv("test.csv")

In [4]: train.shape
Out[4]: (16067, 7)

In [5]: test.shape
Out[5]: (9914, 6)
```

As we can see that our train data contains a shape of 16067 rows and 7 columns and test data contains a shape of 9914 rows and 6 columns. The one column that differs is our target variables.

Lets see the data types of variables in each dataset.

For Train data:

Variables	Data Type
fare_amount	object
pickup_datetime	object

pickup_longitude	float64
pickup_latitude	float64
dropoff_longitude	float64
dropoff_latitude	float64
passenger_count	float64

For Test Data:

Variables	Data Type
pickup_datetime	object
pickup_longitude	float64
pickup_latitude	float64
dropoff_longitude	float64
dropoff_latitude	float64
passenger_count	int

3.1.1 Dealing With Train data:

Let see some of the values in the dataset.

Index	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1.000000
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1.000000
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2.000000
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1.000000
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1.000000
5	12.1	2011-01-06 09:50:45 UTC	-74.000964	40.731630	-73.972892	40.758233	1.000000
6	7.5	2012-11-20 20:35:00 UTC	-73.980002	40.751662	-73.973802	40.764842	1.000000
7	16.5	2012-01-04 17:22:00 UTC	-73.951300	40.774138	-73.990095	40.751048	1.000000
8	nan	2012-12-03 13:10:00 UTC	-74.006462	40.726713	-73.993078	40.731628	1.000000
9	8.9	2009-09-02 01:11:00 UTC	-73.980658	40.733873	-73.991540	40.758138	2.000000
10	5.3	2012-04-08 07:30:50 UTC	-73.996335	40.737142	-73.980721	40.733559	1.000000

Changing Data types

Lets get to each variable one by one.

Here we can see that our fare_amount is object and pickup_datetime is object type. Rest all the variables are in float. lets look at the each variable one by one.So we first need to convert fare_amount in float and then convert passenger count in int as passenger count cannot be in

float. also there are special characters in the fare_amount feature, So we will replace them as well.
Also there are few negatice values in the fare_amount variable so we will remove them also.And at last we will reset the index

```
In [7]: train=train.drop(2486)
   ...: train=train.drop(2039)
   ...: train=train.drop(13032)
   ...: train.fare_amount = train.fare_amount.replace({"-":""},regex=True)
   ...: train.fare_amount = train.fare_amount.astype(float)
   ...: train = train.reset_index(drop=True)
```

Pickup_datetime variable is an object. So we need to convert it into datetime object After removing nan values.

Rest all variables are float,for longitude and latitudes its ok but not for passenger but as NA values are there in NA so we will cover this also under NA removal section.

Removing Missing Values

```
In [11]: train.isna().sum()
Out[11]:
fare_amount          24
pickup_datetime       1
pickup_longitude      0
pickup_latitude       0
dropoff_longitude     0
dropoff_latitude      0
passenger_count      55
```

Fare_amount: We can see that the total missing values in the dataset is less than 1% but we will not simply remove all of them as what all values we need to drop further we dont know. So we will try to keep as much values as we can.

There are some outliers in the fare amount but we are not removing them right now, a there might be a case that number of passenger count is high and distance travelled by the passenger is also high. As we are replacing NA values with mean of fare amount variable, we have seen that replacing 54343 alone with NAN and then replacing then comparing the mean with unremoved 54343 value, there is not much differenc in the mean. So we will replace NAN values with mean, and deal with outliers after further analysis.

Pickup_datetime missing values, we have simply replaced the missing value with the just before index value of the variable.

Passenger_count: There are no missing values for the passenger count.but there are few values which are impractical as the maximum number of passengers a SUV cab carry are 6. For all the values which are above 6 and less than 0 we will remove them.

Index	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
1146	8.000000	2014-03-27 08:05:01	-73.991098	40.770655	-73.976933	40.790070	5345.000000
293	6.100000	2011-01-18 23:48:00	-74.006642	40.738927	-74.010828	40.717907	5334.000000
8983	8.500000	2015-01-14 15:10:21	-73.955444	40.787605	-73.965561	40.798691	557.000000
971	10.100000	2010-11-21 01:41:00	-74.004500	40.742143	-73.994330	40.720412	554.000000
8504	11.300000	2010-05-23 20:06:37	-73.985424	40.738468	-74.001698	40.707758	537.000000
1200	9.700000	2011-08-16 09:29:00	-73.980487	40.741610	-73.980617	40.746868	536.000000
356	8.500000	2013-06-18 10:27:05	-73.992108	40.764203	-73.973000	40.762695	535.000000
8713	4.500000	2009-09-04 09:14:03	-73.977518	40.758480	-73.983252	40.749837	531.200000
263	4.900000	2010-07-12 09:44:33	-73.983249	40.734655	-73.991278	40.738918	456.000000
386	8.100000	2009-08-21 19:35:05	-73.960853	40.761557	-73.976335	40.748361	354.000000
1107	4.900000	2009-08-08 21:50:50	-73.988977	40.721068	-73.982368	40.732064	345.000000
233	8.500000	2011-07-24 01:14:35	0.000000	0.000000	0.000000	0.000000	236.000000
8569	12.500000	2011-12-03 03:21:00	-73.993718	40.762039	-73.977527	40.734024	87.000000
8443	5.700000	2009-03-28 22:00:00	-73.982413	40.751320	-73.971292	40.748502	58.000000
413	15.040000	2013-09-12 11:32:00	-73.982060	40.772705	-73.956213	40.771777	55.000000
1007	3.700000	2010-12-14 14:46:00	-73.969157	40.759000	-73.968763	40.764617	53.000000
8404	6.900000	2010-08-25 11:41:00	0.000000	0.000000	0.000000	0.000000	53.000000
1242	5.300000	2011-10-16 00:22:00	-73.981095	40.738160	-73.990587	40.740105	43.000000
8629	20.000000	2012-12-10 22:28:00	-73.955445	40.670232	-74.004795	40.731477	43.000000
1043	5.700000	2012-08-22 22:08:29	-73.973573	40.760184	-73.953564	40.767392	35.000000

```
In [2]: index_list = []
   ...: for i in range(len(train)):
   ...:         if train.passenger_count[i] > 6:
   ...:                 index_list.append(i)
   ...:
   ...: for i in range(len(train)):
   ...:         if train.passenger_count[i] <1:
   ...:                 index_list.append(i)
   ...:
   ...:
   ...: train = train.drop(index_list)
   ...: train = train.dropna()
   ...: train = train.reset_index(drop=True)
   ...: #rounding off the values in passenger count and removing NA
   ...: train.passenger_count = train.passenger_count.astype(int)
```

Logitude And Latitude: For the 4 variables i.e pickup_latitude,pickup_longitude, dropoff_latitude, dropoff_longitude, we need to check the maximum and minumum values. Latitudes range from -90 to 90 and longitudes range from -180 to 80. Any value above or below this need to be dropped. As we cannot replace these with any statistical method.

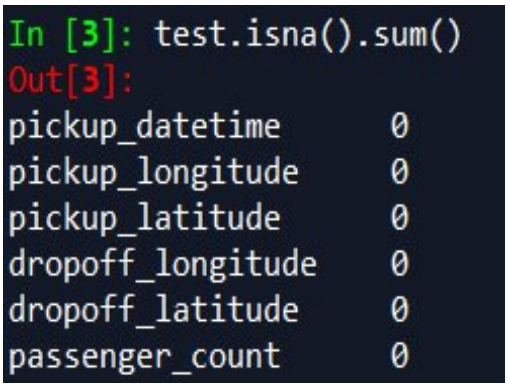
For some of the lat lon. when investigated are pointing towards artic ocean which is not relevant. After investigating further, I got to to know that for those indexes the values of latitudes and longitudes have been interchnageed. If we interchnage them again then we will get locations of manhattan or Queens which is much more relevant. So we will interchnage them.

3.1.2 Dealing With Test data:

Changing Data types

We can see from the above table that only pickup_datetime need to be converted to datetime object. Rest all the variable has right data type. We will convert the datetime variable same as the way we converted for test data.

Checking Missing Values.



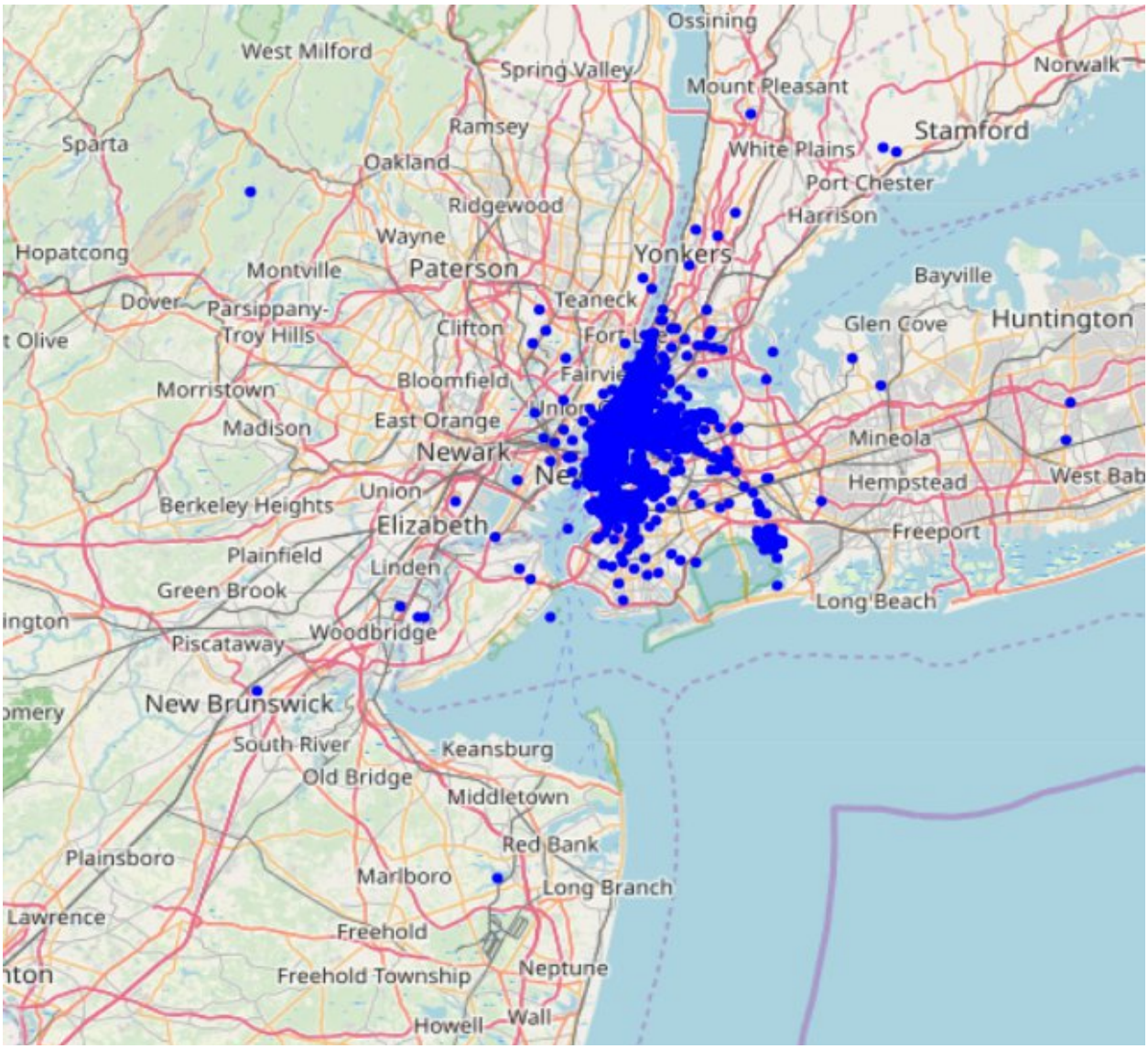
We cab see there is no NA value in the test data. So we can continue further.

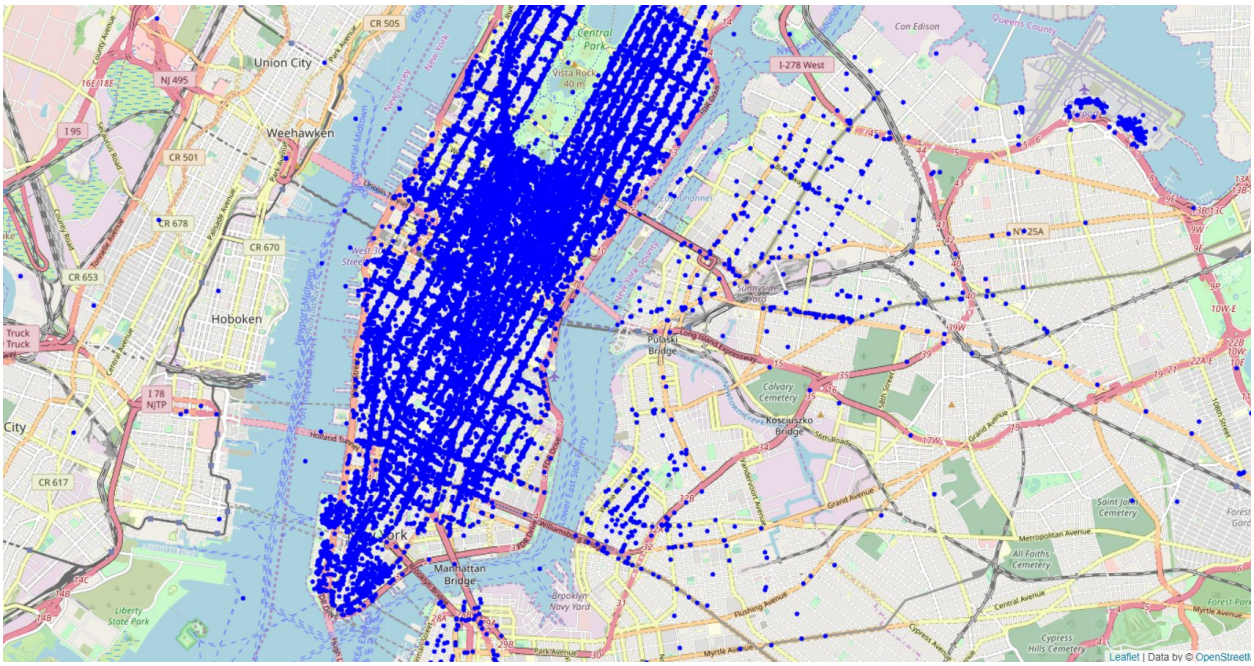
Chapter 4 Data Analysis

We will do data analysis in this section. Normally test and train data are analysed seperately but in this case we will join both the dataset and make a single dataset. As there is latitude and logitude in the dataset so, I have prefered to combine them to analyse better.

So we have combine the two datasets and have taken the fare_amount variable as seperately.

Before doing univariate and bivariate analysis we will first plot the longitudes and latitudes in map.

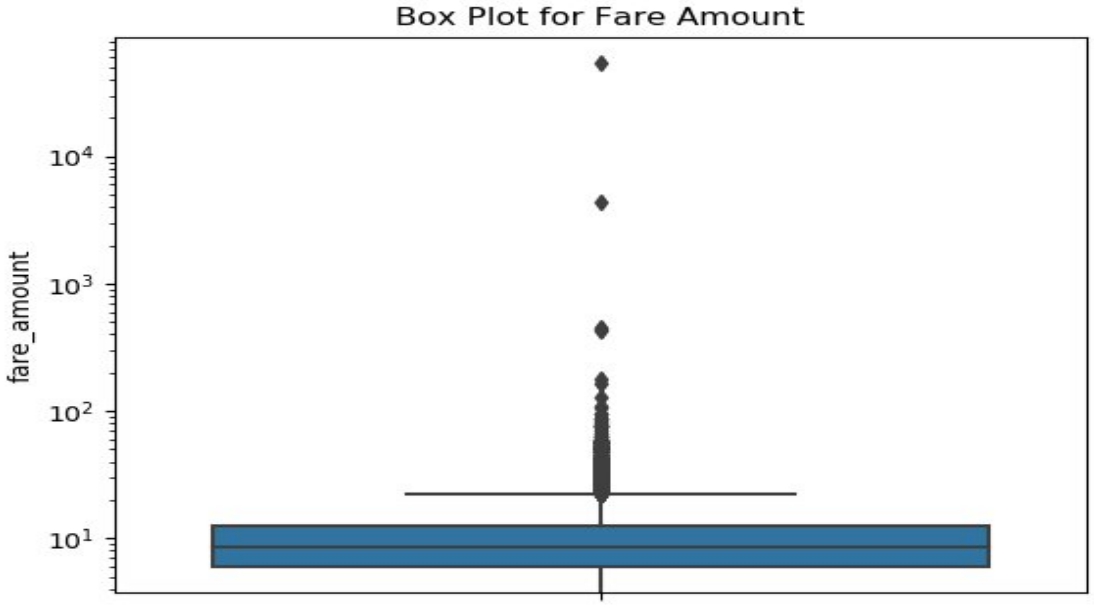




By the map plot that we get, we can see that this is the data of NewYork Ciy. The pickup points are majority from Manhattan and few from Queens and Broklyn. In this step we got few points that were pointing at the equator which are irrelevant. As except from those datapoint rest all are from this NYC. So we will remove them to get better analysis and fit.

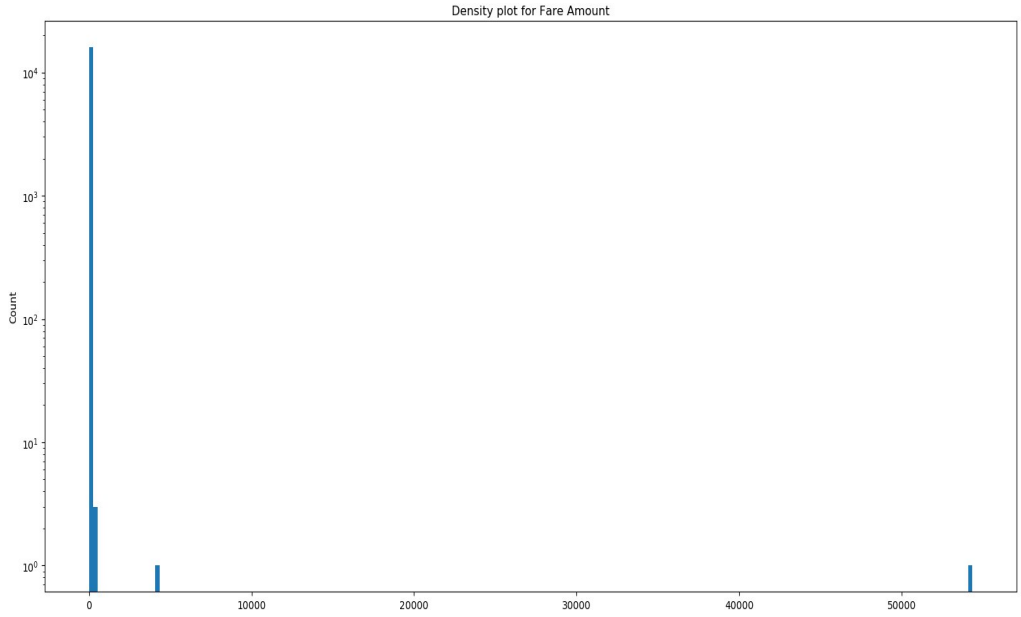
4.1 Univariate Analysis

Fare_amount: Lets see boxplot of fare_amount.



We can see that there are few outliers in the fare amount. We will keep them for now and will remove them after bivariate analysis.

Lets see a histogram of fare_amount

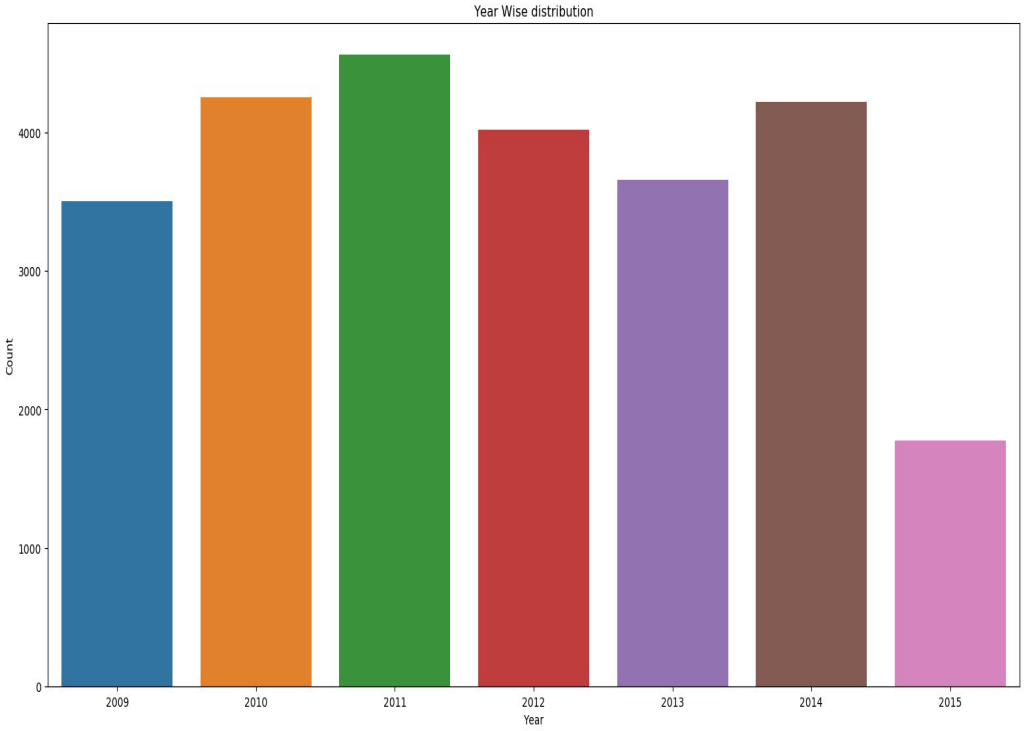


So we can see that most of the values are near to 20 and only few values are near to 100. We can also see that one is near to 4000 and another value is above 50000. We expect both of these values are outliers. But lets dig more deep

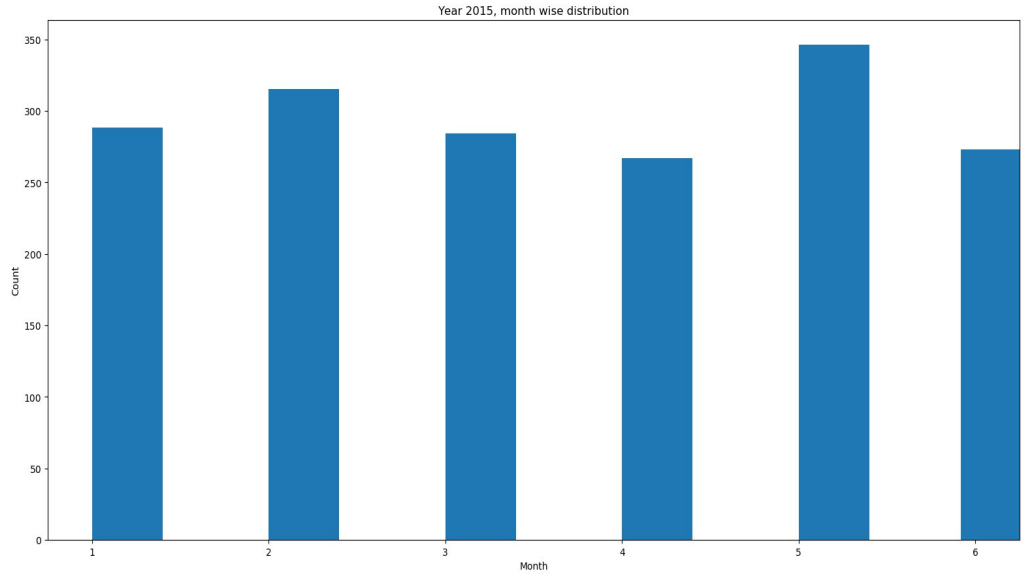
Pickup_datetime

Lets see some datetime analysis.

We will be first checking the count of trips in each year.We have a dataset of 7 years from 2009 to 2015. We got to know that the mximum number of rides were in the year 2012 and the minimum number of rides were in theyear 2015. Also we got to know that after 2012 the rides get decressed countinously and in 2015 thw rides were nearly half of the rest of the year

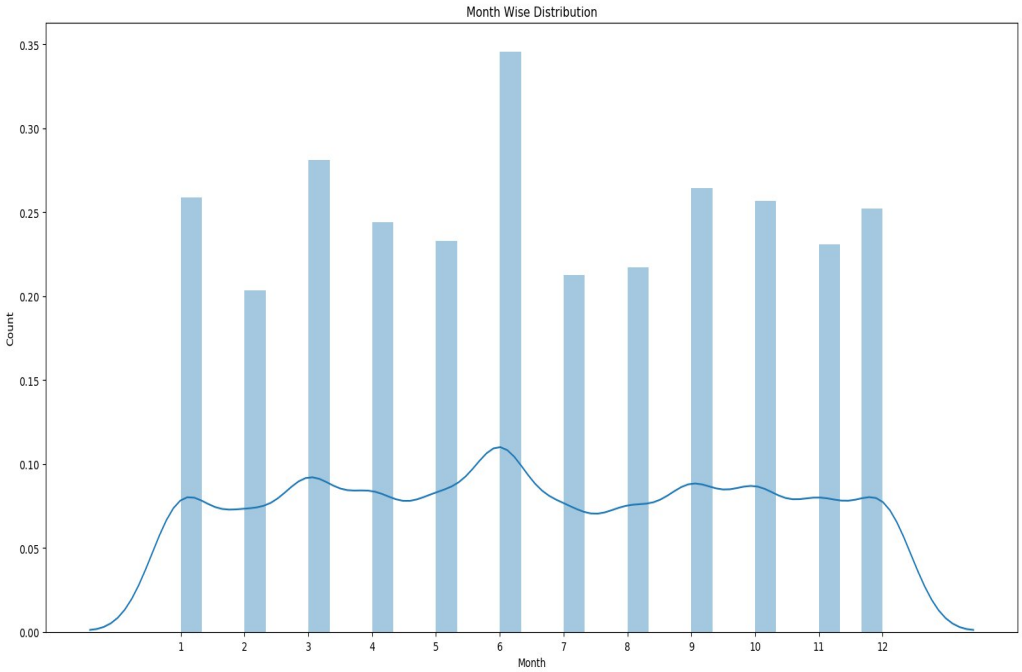


Lets see how much data we have for the year 2015



So we got to know that for the year 2015 we have data for only 6 months. That means the todat data starts from January 2009 till June 2015.

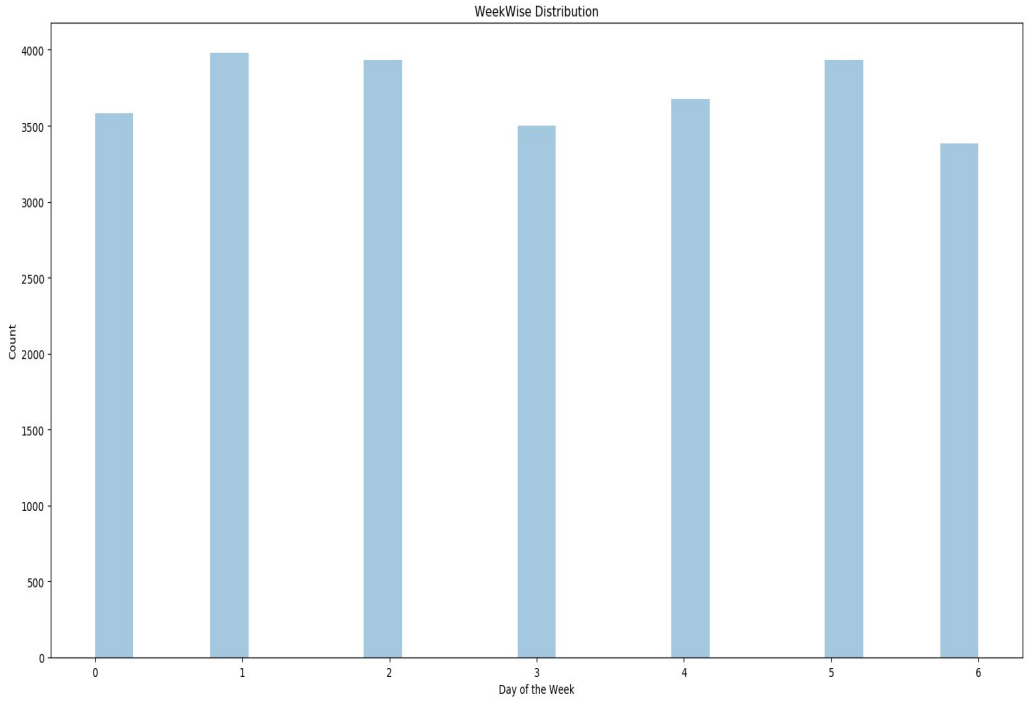
Lets see month wise dristribution of rides for all the years



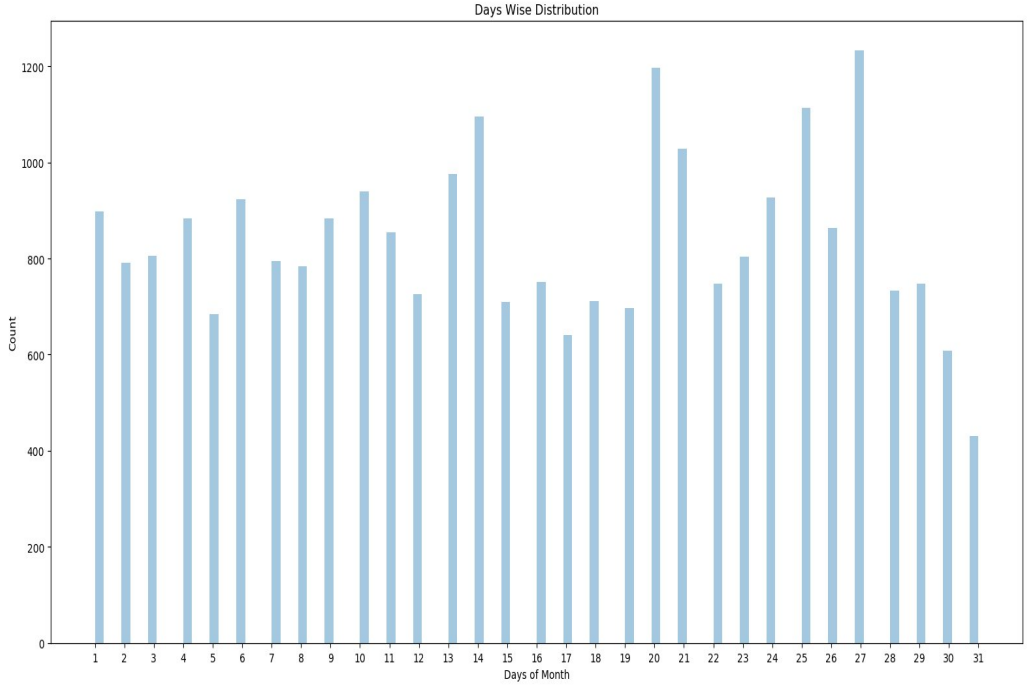
We can see that there is drop in the second month. This might be due to the snow fall in New York. And also there is a huge amount of rides in mid summer. That is the vacation time. So tourist visit increases the rides count. We can add month feature the feature enginnering.

Lets see week wise distribution

From the graph below we can see that there is huge amount of rides for the first two days i.e.Monday and Tuesday. But it decreases drastically on Wednusday. But it again starts increasing on friday and saturday but descrese on Sunday. That means majorly cab is used for office purpose as it has highest number of rides on Monday and Tuesday. And on weekends people dont use it. It can be a good factor in estimating the fare amount. So we will add this factor in the feature enginnering section.

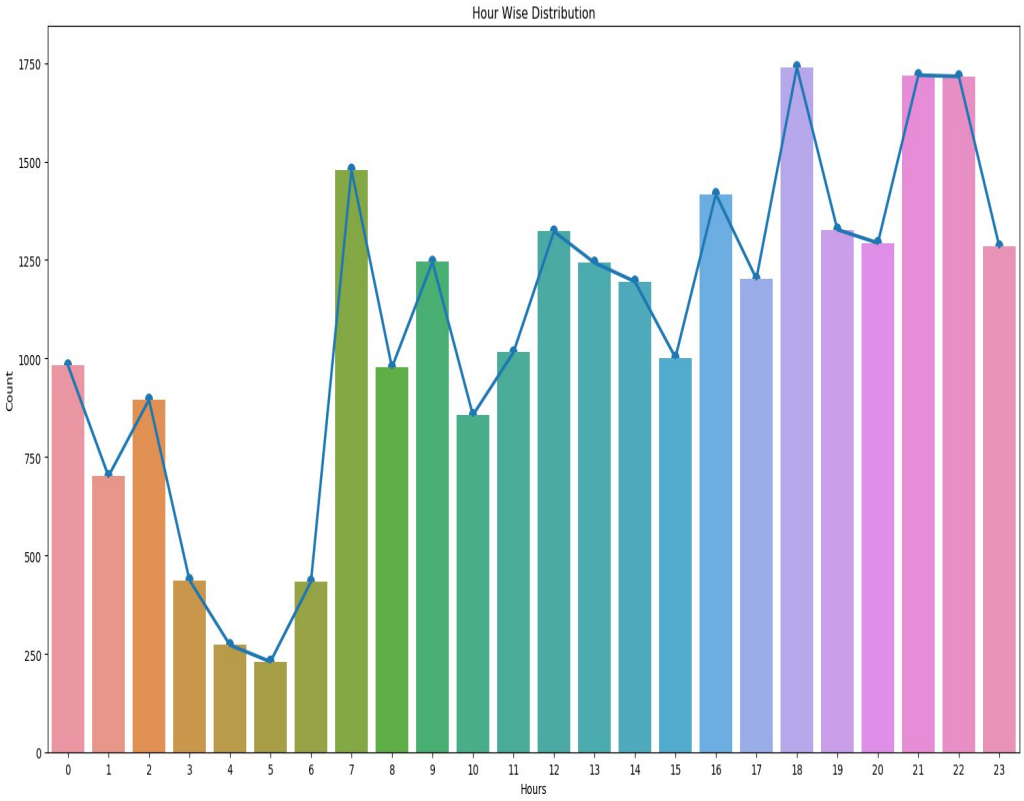


Lets again see day wise distribution. And check if we can find any pattern there



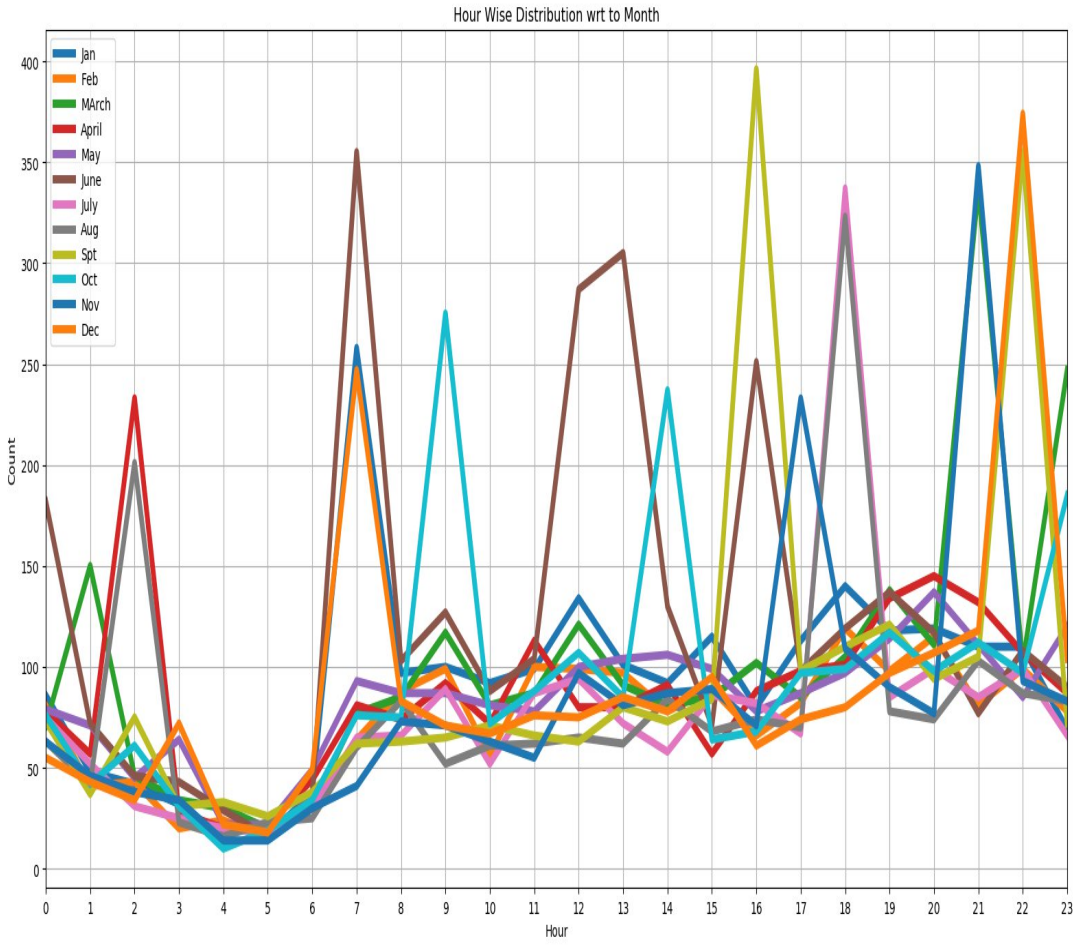
In this distribution we can see two drops, one at the middle of the month and another at the end of the month.

Lets check the hour wise distribution.



This tells us much about the usage of cab rides. In the early morning from 4 to 6 there are very less rides. Rides increases drastically from 7 AM to 12 PM with some ups and downs. Then after 12 PM the count of rides descreses steadily till 3 PM as its noon time and less people which travel. Then the rides count starts increasing from 4PM as the office shifts over. And the count remains at a higher count till midnight. It dreases over the time span of 00 AM till 3AM. This seems to be a very important visualization and we will include hours column in feature engineering.

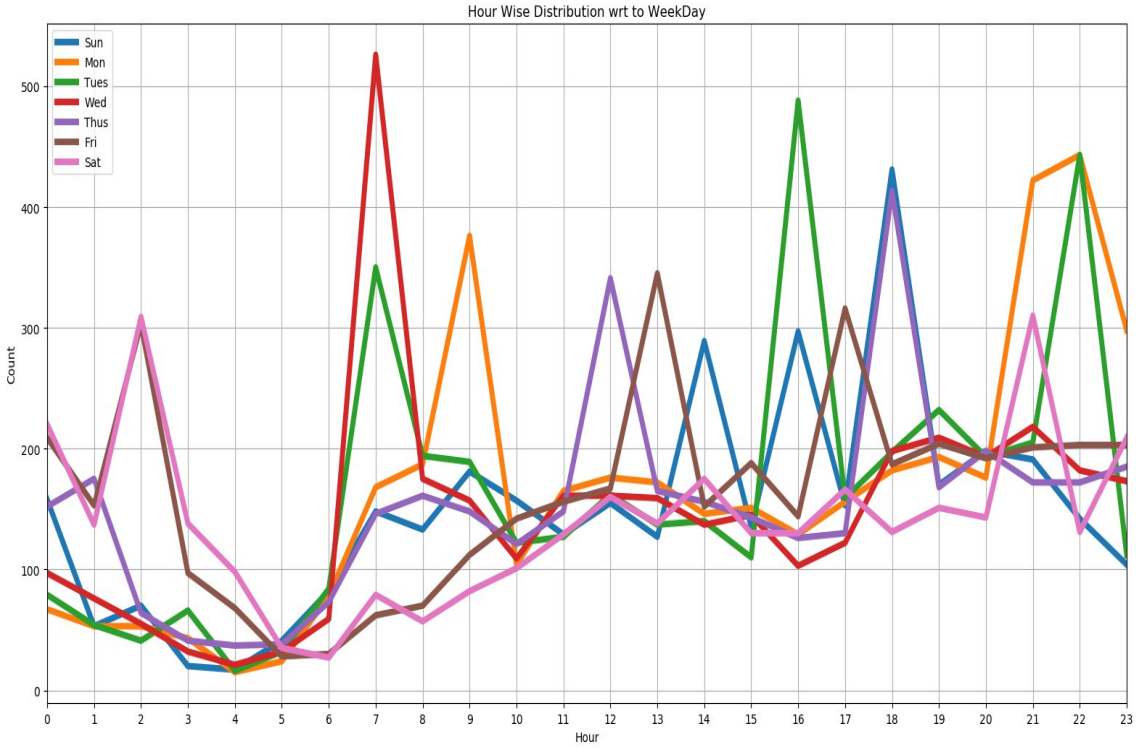
Comparing hour wise distribution wrt to Month.



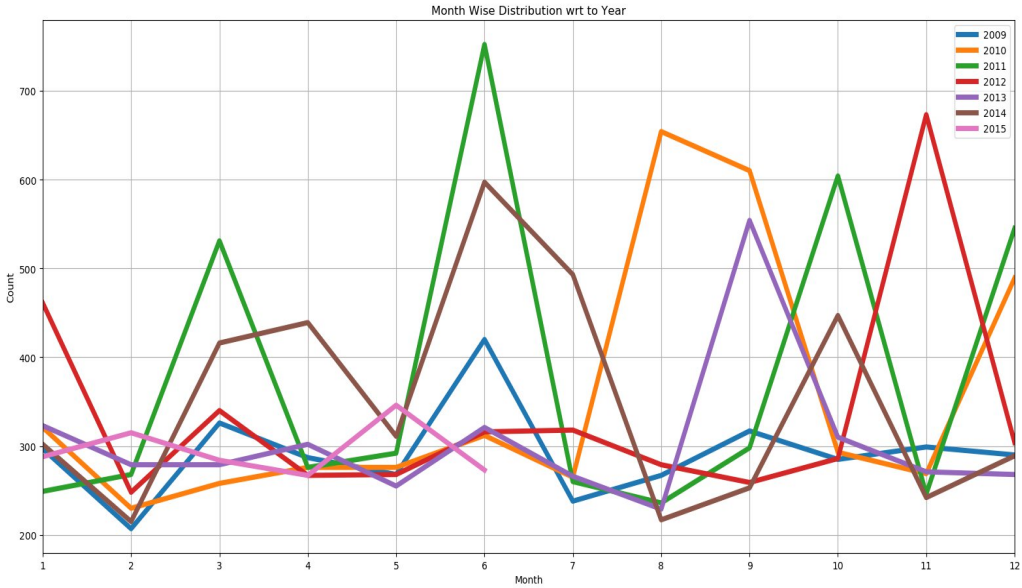
For all the month 3AM-6AM drop is common. There is certain peaks for almost all tof the months but at different hours of the day. This might be because of the weather conditions. As summer months has peaks from 11-19. Winter months has peaks in the late evenings. So weather is having a great impaact on the usage.

Comparing hour wise distribution wrt to Week days

Monday and uesday has peaks inearly mornings and late evenings that are the proper office timings Friday saturday has late night peaks. Sunday sare stable, just few samll peaks in the evening. Monday,Tue,Wed has good peaks in the morning hours. That means People are using cabs mostly for office purpose in the early mornings and in the late evenings. So timing are a major factor for fare amount prediction and will be included in feature engineering.

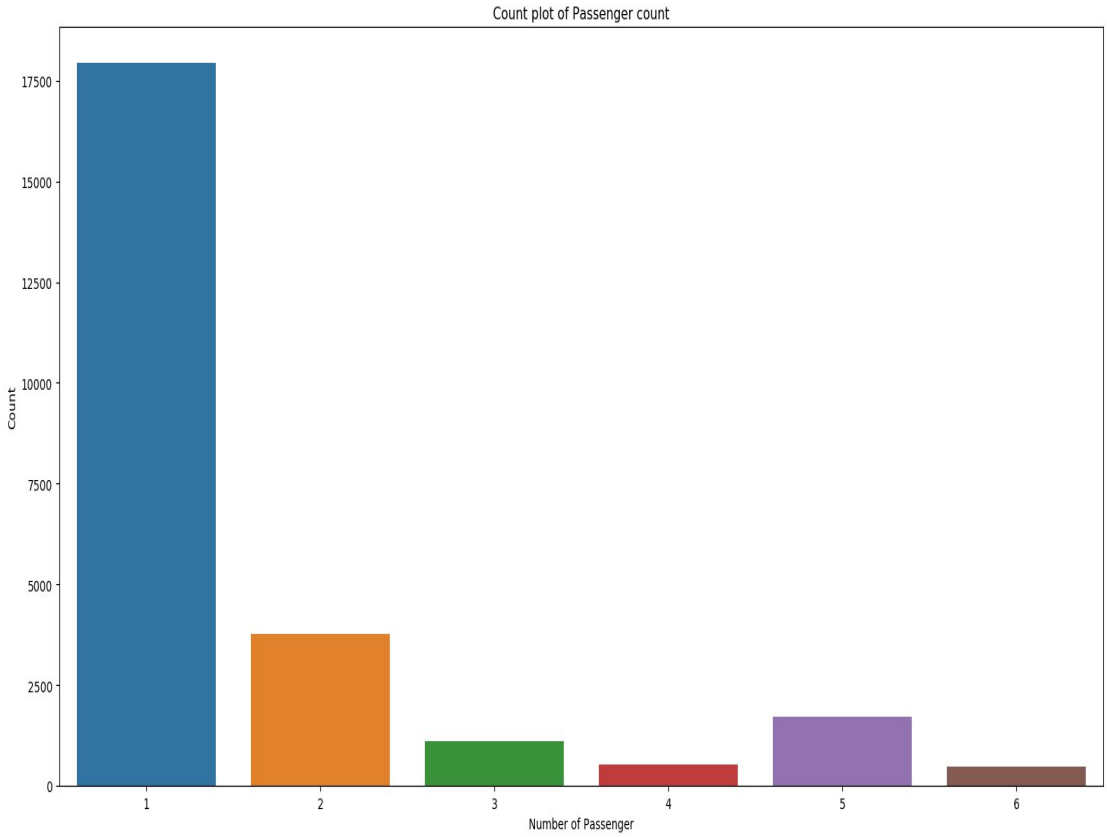


Comparing Month wise distribution wrt to Year



So from all the date time analysis we got to know few important features that needs to be present in the data for modeling an dwe will include thwm in feature enginnering section.Lets go further and analyse other variables.

Passenger Count visualisation

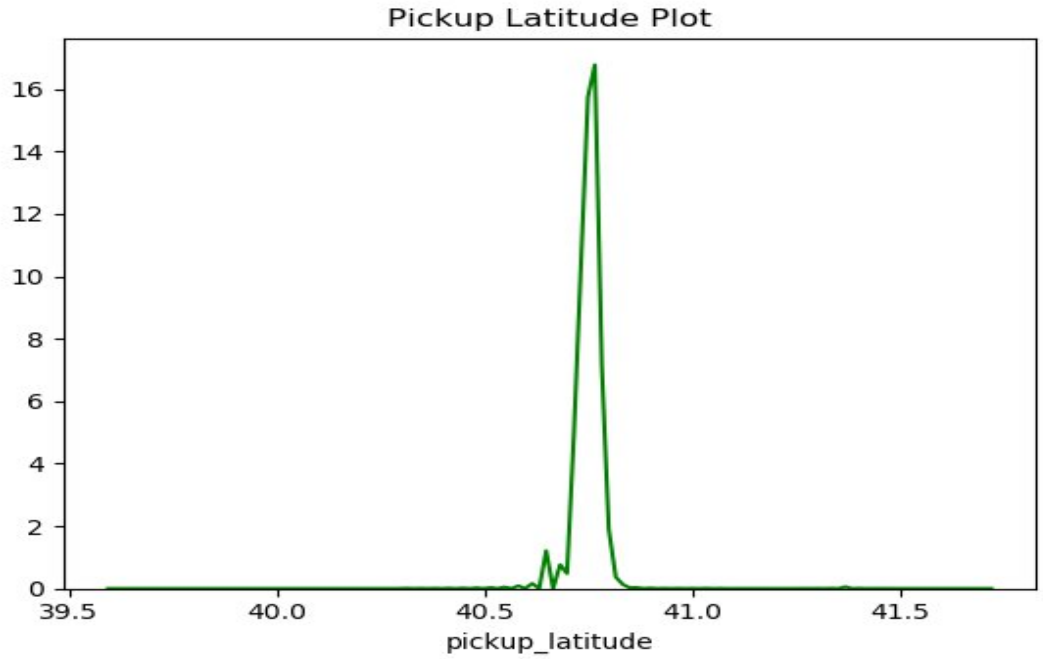


We can see that maximum count is for single passenger and then for 2 passsengers. Then the count decreasses but then increasses for 5 passenger.

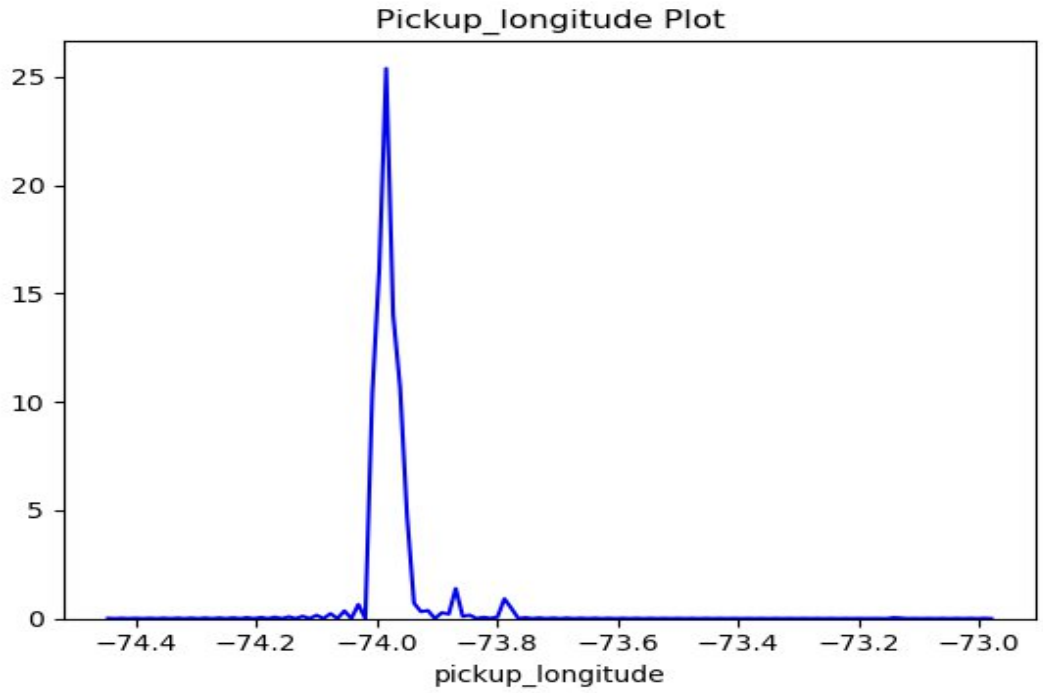
Lets check Longitude and Latitude visualisation.

We have alredy done our pre processing on the longitude and latitude. So our visualisation would be normal.Still we will check if we found any thing special in the visualization.

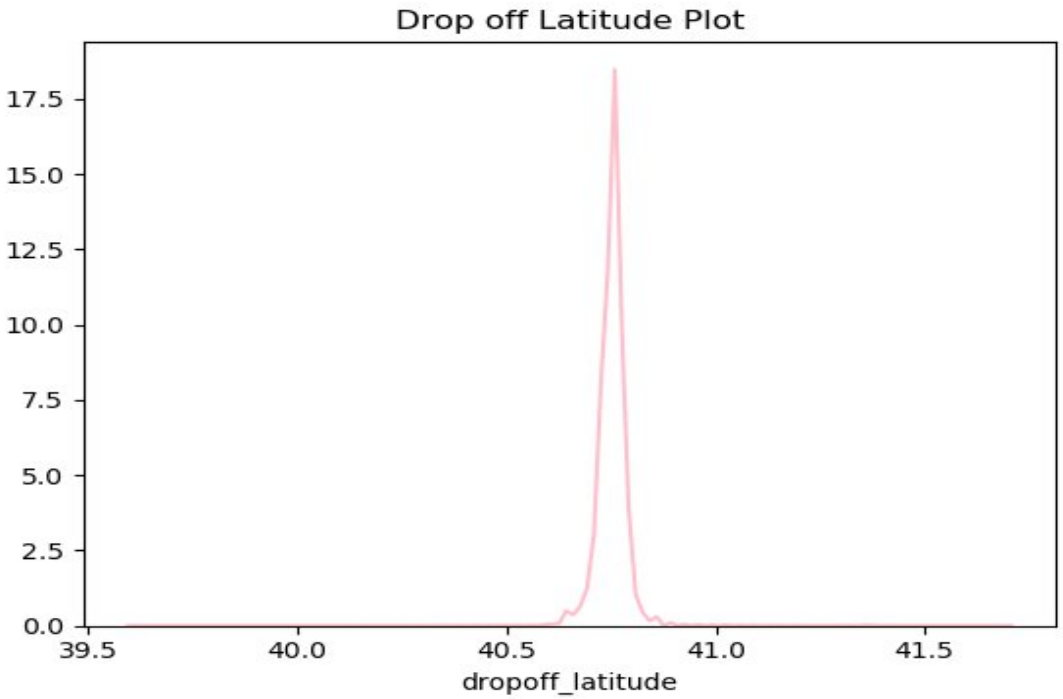
Pickup Latitude



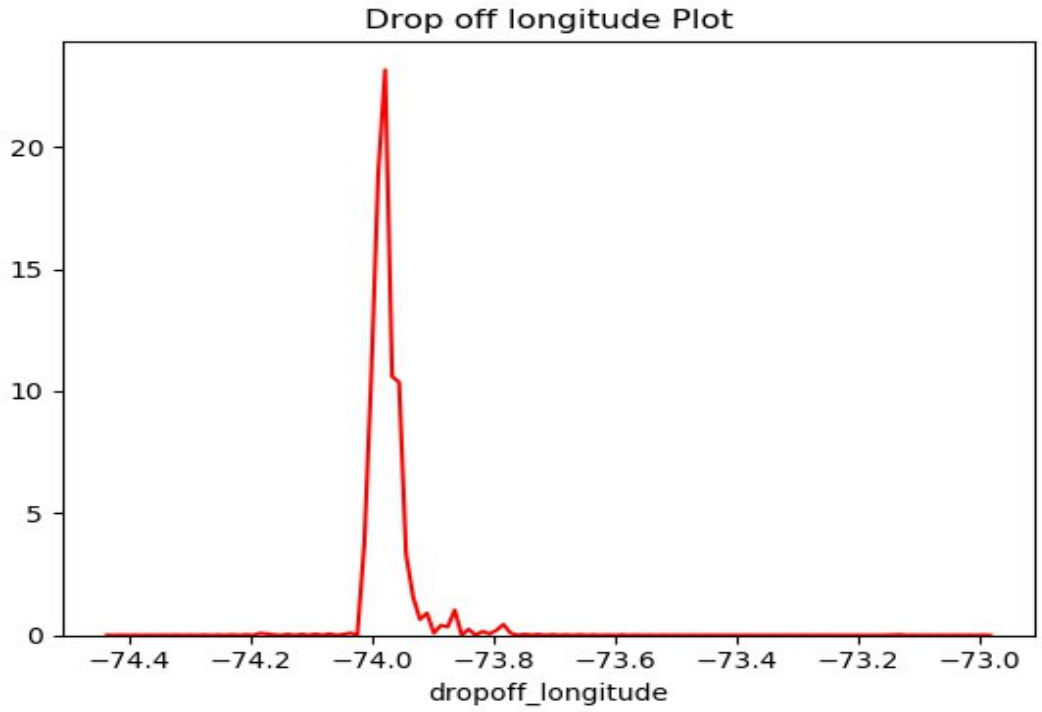
Pickup Longitude



Dropoff Latitude



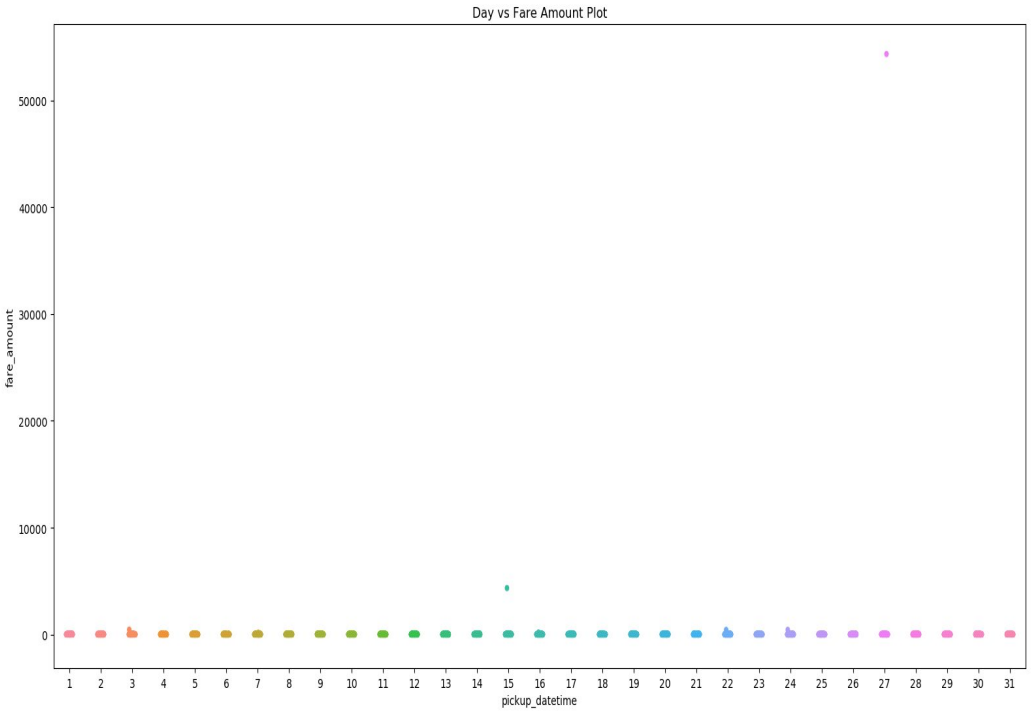
Dropoff Longitude



As we can see that our longitude and latitudes are in ranges. Latitudes range from -90 to 90 and longitudes range from -180 to 80. So no more processing in these variables and we can proceed further.

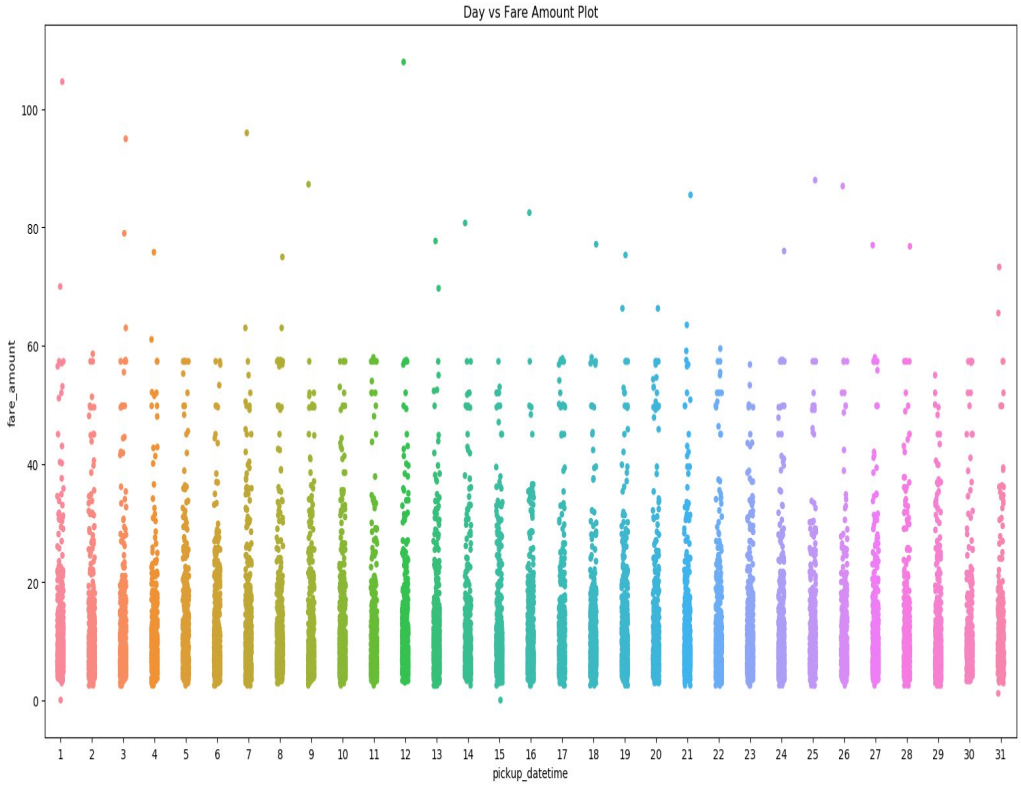
4.2 Bivariate Analysis

Day vs Fare Amount Plot



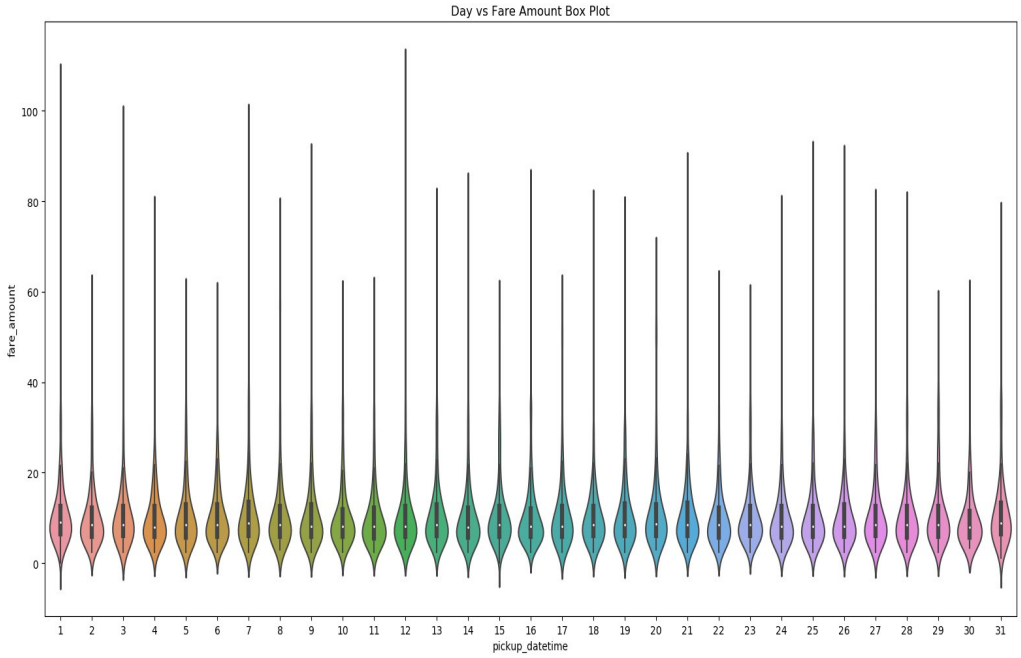
We can clearly see that the two values 54000 and 4000. So we will replace them with the mean of all values. We will first make them as NaN values and then replace them with mean. For mean we will take the mean of the same month and year to which these values belong.

After replacing them with mean we will again vusualise the plot



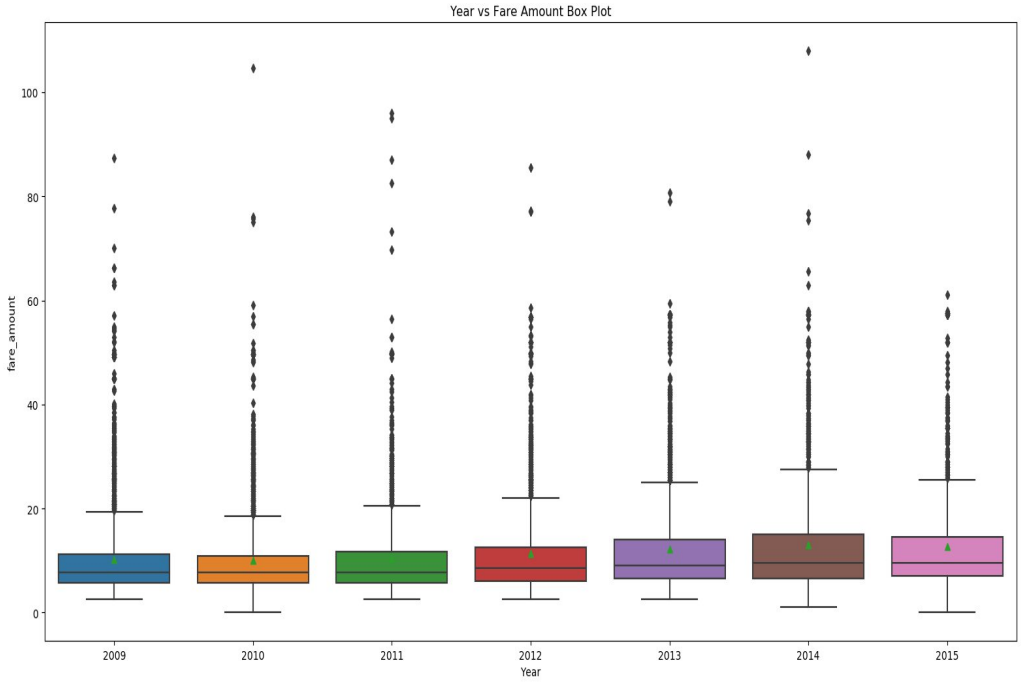
Now the plot is pretty stable. There is equal distribution of amount till fare 60 through out the month. After that the distribution beacomes a little unstable. We will not consider these values above 60 as outliers as these are present in almost each day of the month. That might be of high distance travelled.

Same can be observed from the violin plot below.



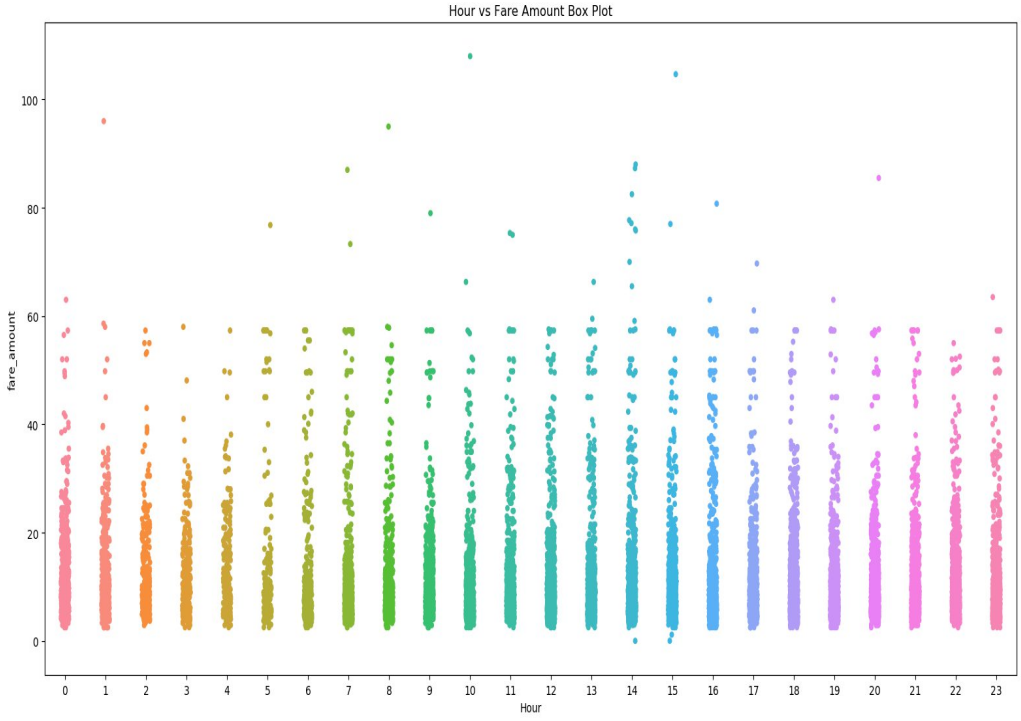
Year vs Fare amount

Lets see the yearly distribution of fare amount



Same we can see from this plot also. except for few values, rest all are under same range for each year. Median is same for all the years.

Hour vs Fare amount

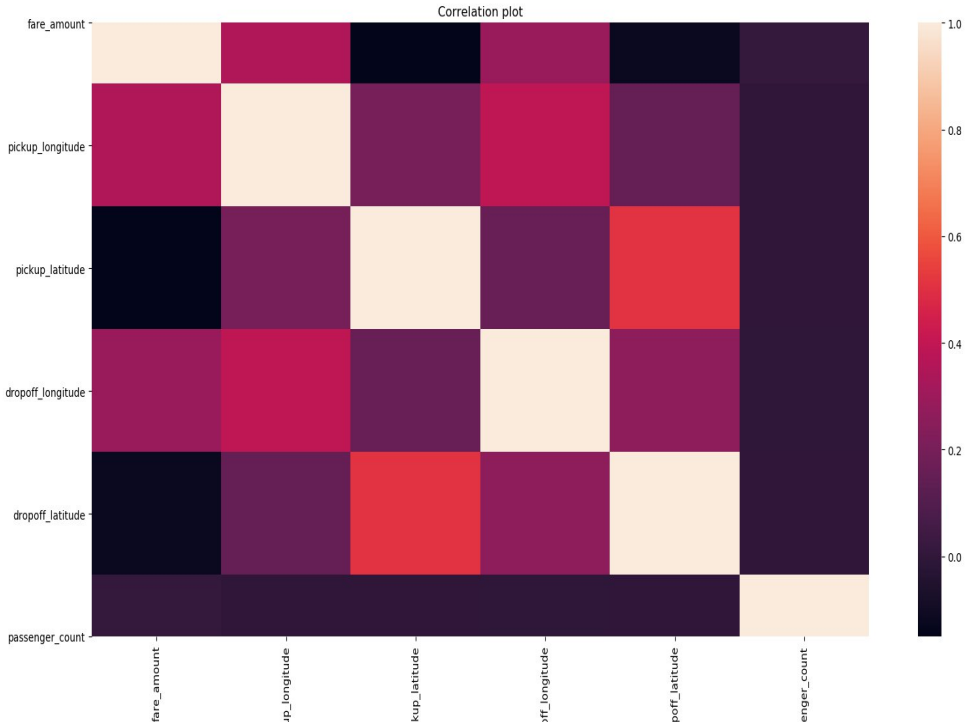


This is the boxplot of each in a day on x axis and fare amount values of each hour in y axis. We can see that there is a increse in prices in 15:00 hrs and even though the rides decreses in late night and early morning but still the fare amount is the same density.

Correlation Plot

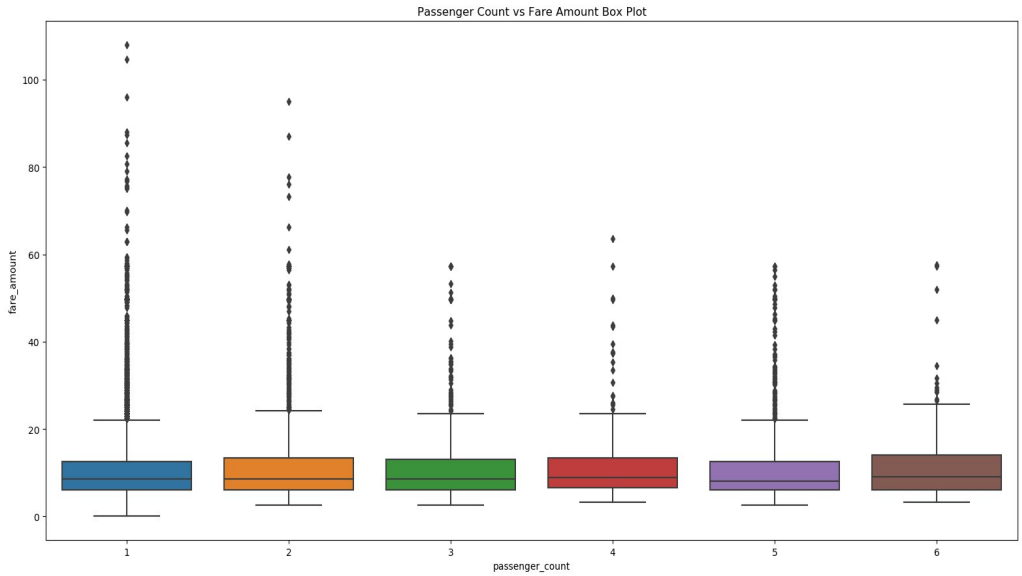
Lets check the correlation between the vairbales.

We know that our data has longitudes and latitudes which will be highly correlated with each other in the correlation plot. But we still plot it as any how we will be removing them afterwards.



There is no high correlation between different indepenent variables except longitudes and latitudes.

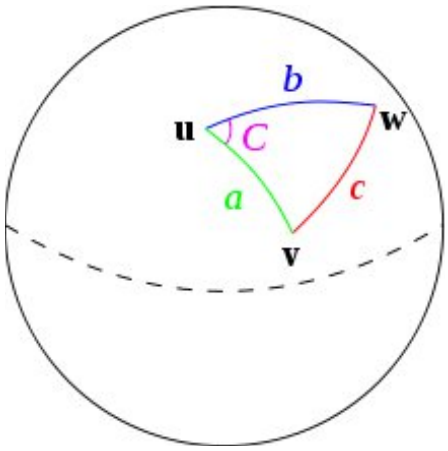
Passenger Count vs Fare Amount Box Plot



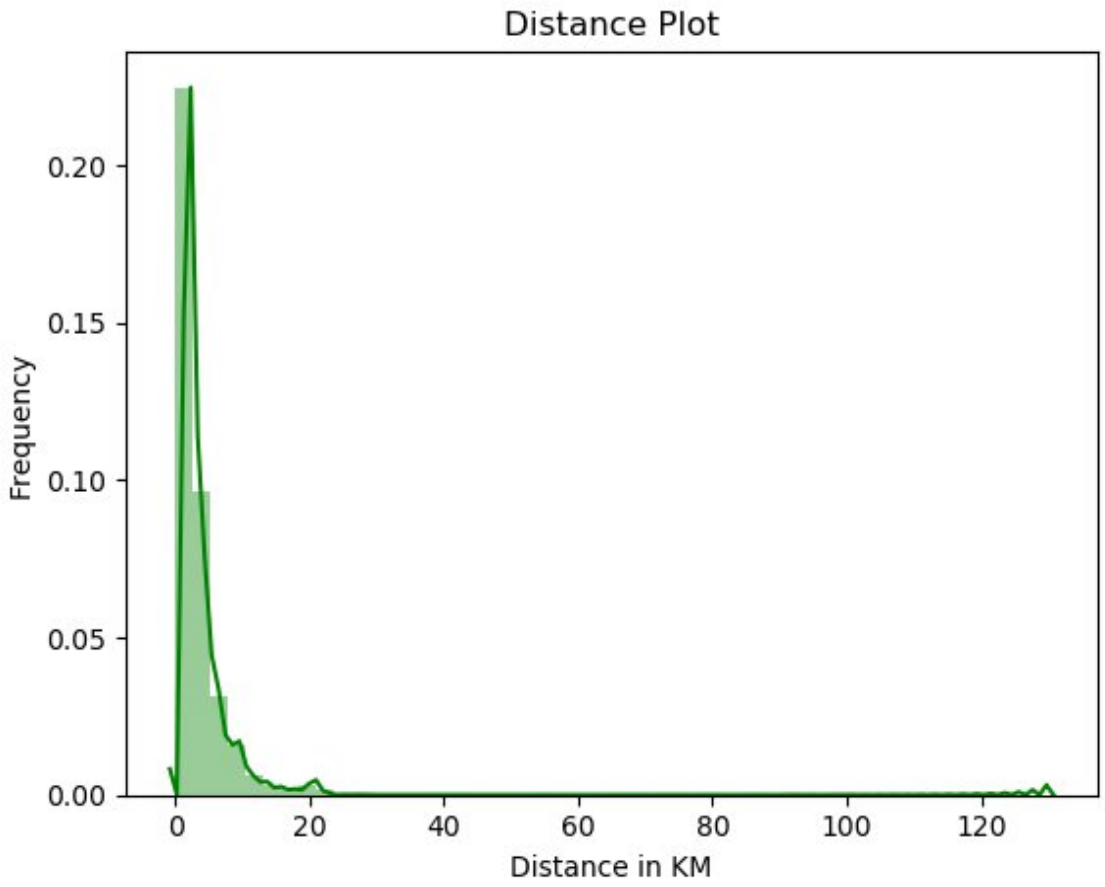
We can see that the maximum amount charged was when 1 or two passenger travel. There is no statistical difference in fare amount when the passenger count increases. That means number of passenger dose not make much difference in the fare amount.

Chapter 5 Feature engineering

The first feature we will be creating is calculating distance travelled by each ride with the help of longitude and latitude.
Inorder to calculate distance, we will be using using the Haversine formula which says:
The **Haversine** formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of Haversines, that relates the sides and angles of spherical triangles.



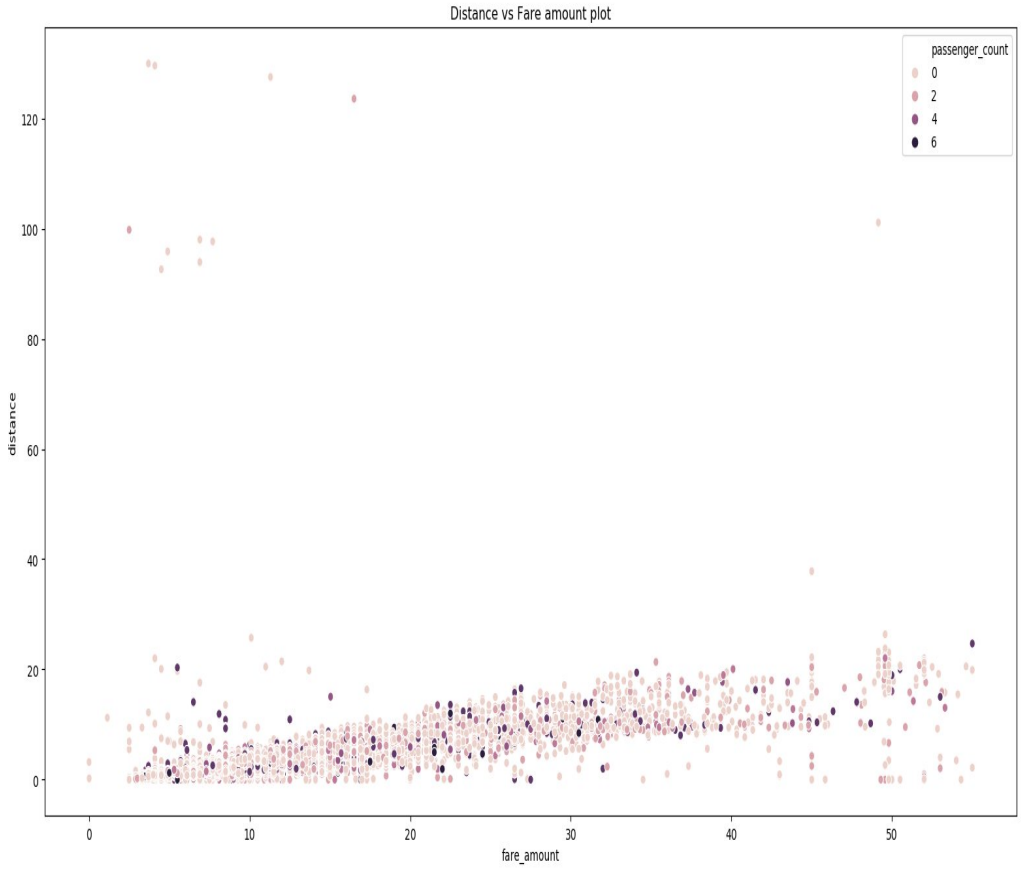
As we have created a new feature “Distance”, so lets do some analysis with that



We can see that most of the distance traveeled by the passenger is below 10 KM. Further there are a few peaks above 120 also. We can also see that the some of the distance is 0, so we need to remove them.

Visualization distance vs fare amount

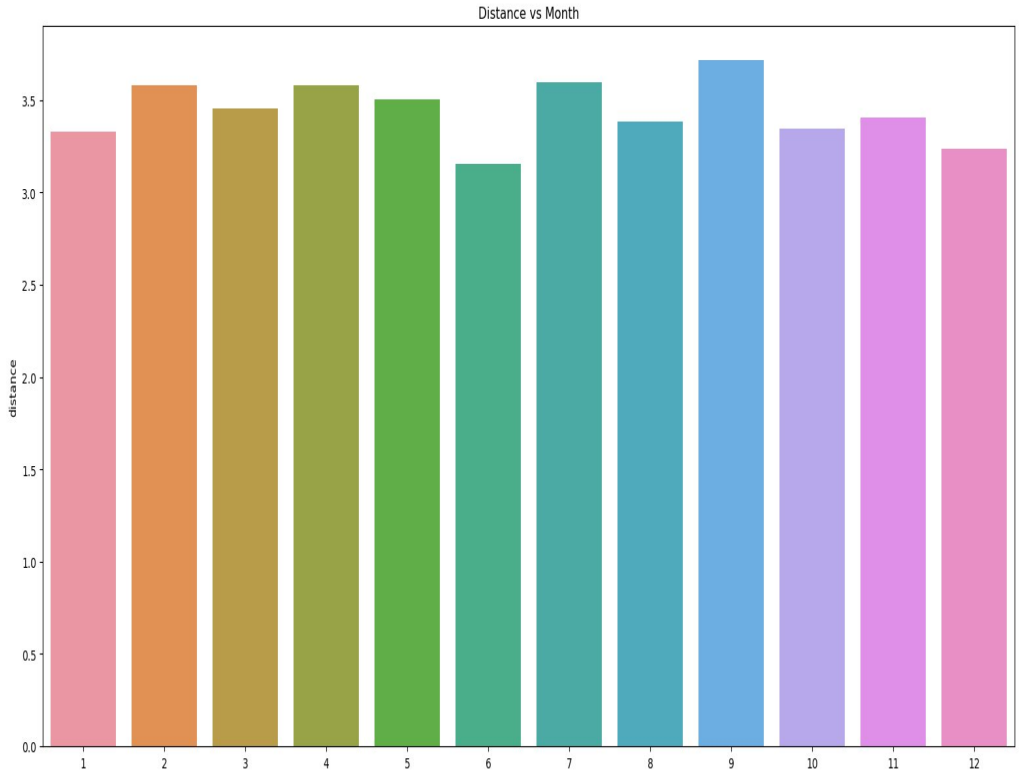
We have plot a scatter plot between distance travelled and fare amount with passenger count as hue. we can see from this plot that the distance and fare amount has a linear relation with one another. As the distance is increasing the fare amount also increases. So this is a very important feature for our model



AS we see from the scatter plot that distance and fare amount has some what linear relationship,but that dosenot actually appliea for the passenger count. Except few outlier values, where distance is high but the amount is very less, all other values has=ve linear relation. So we need to remove these outliers.

Month wise distribution of distance

Lets visualise month wise distance travelled by the rides



Now as per our analysis above we have found that certains features do count important and needs to be added in the final data. So we will create some more featues. We will be breaking the pickup datetime column into multiple columns. The columns would be Hour, Month, Day of the month, Day of Week, Year.

We will not do any analysis on this as we have already done this in above section

```
#Breaking Date Time and Year into different columns
data['year'] = data.pickup_datetime.dt.year
data['month']= data.pickup_datetime.dt.month
data['day'] = data.pickup_datetime.dt.day
data['dayofweek'] = data.pickup_datetime.dt.dayofweek
data['hour'] = data.pickup_datetime.dt.hour
data = data.drop('pickup_datetime',axis = 1)
```

Chapter 6 Model Pre Processing and Cleaning

6.1 Feature Selection

We will be dropping the cordinates column now as we already have used it in calculating the distance travelled and do not need more. Hence we will drop all of the 4 columns of the coordinates fro our dataset.

6.2 Checking Multi colinearity

Now we will check for Multi colinearity and will remove the variable which is highly corelted with others.For this we will be calculating Variable Inflation Factor of each variable with respect to others. We will set the threshold to 5. Any variable which has variable inflation factor above 5 with other variables will be removed.

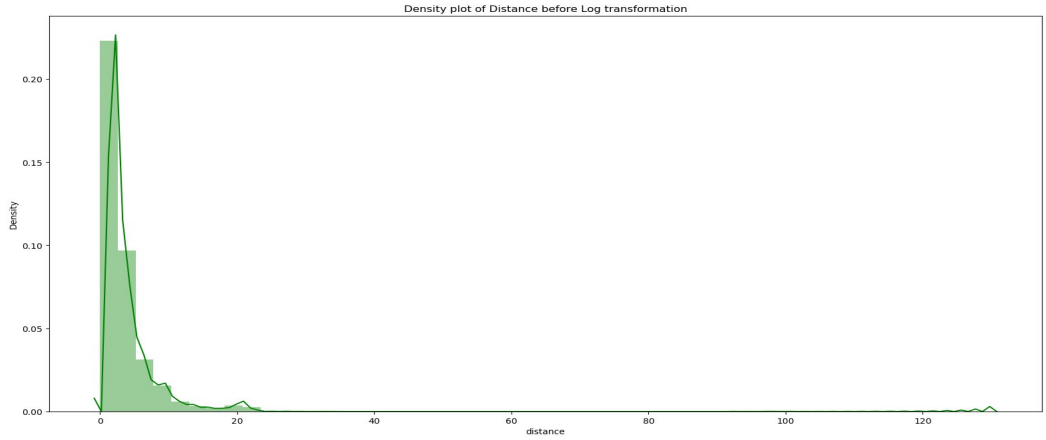
So in our analysis we found that the Year column is highly corelated with the other independent variable. So we will be dropping it.

6.3 Log transformation of variable Distance and Fare

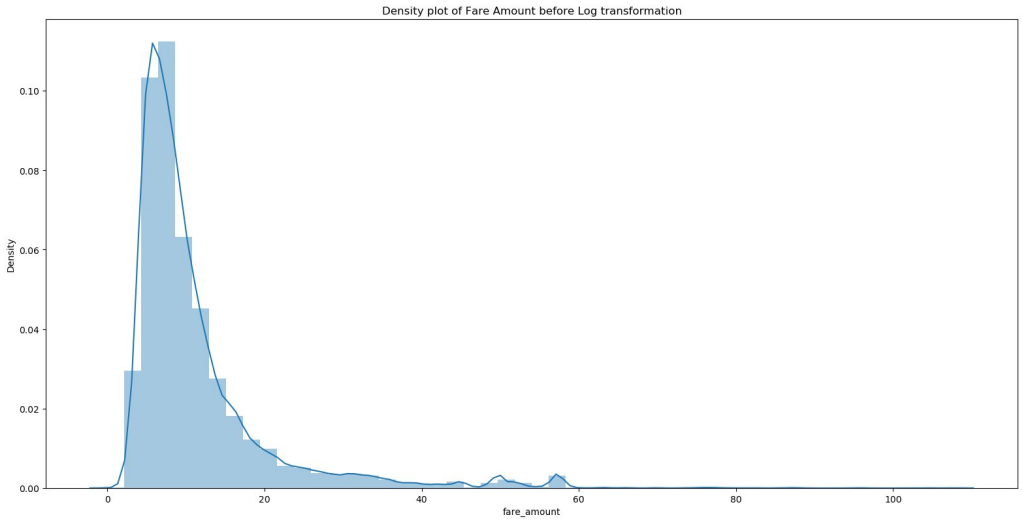
We have two countinous variables in our dataset. Distance and fare amount. As we have seen above in the density plot of distance and fare amount that our data is right skewed. So in order to fix that we have to do log transformation of both the variables.

Before Log transformation

Distance

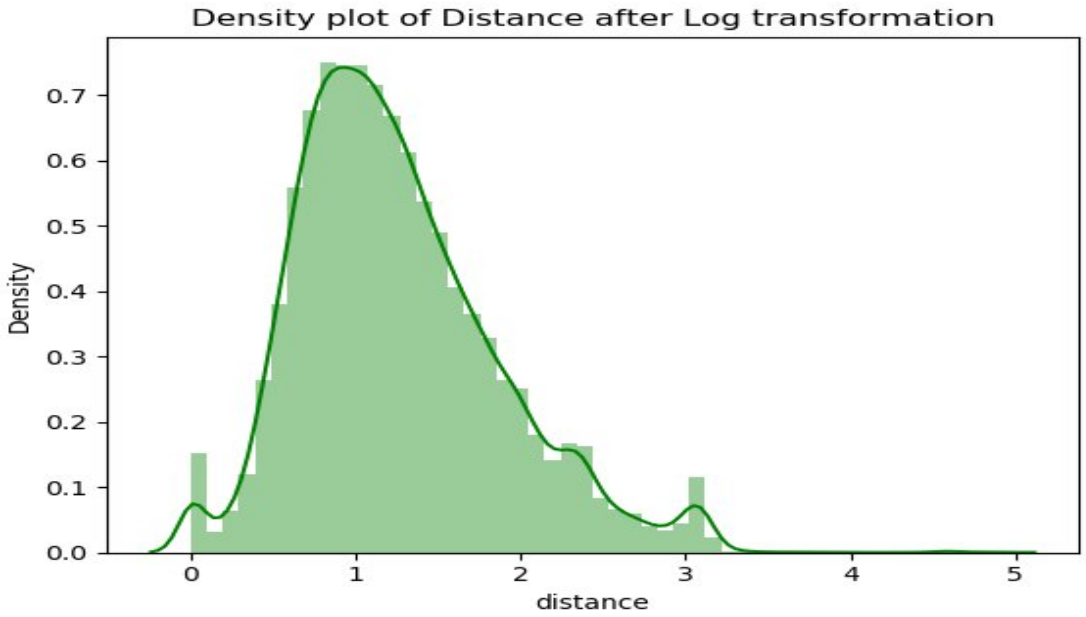


Fare Amount

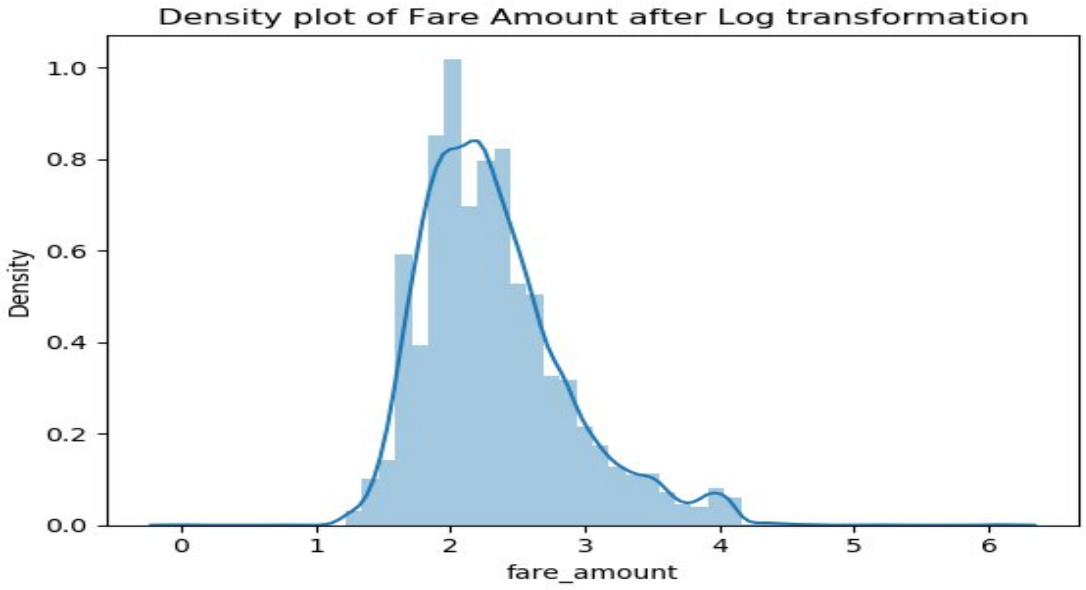


After Log Transformation

Distance


Density plot of Distance after Log transformation
Density
distance
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1
2
3
4
5

Fare Amount


Density plot of Fare Amount after Log transformation
Density
fare_amount
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6

6.4 Breaking Data into Test and Train again

As our analysis and feature engineering is done on the whole dataset, so now we wil break the data again into test and train and keep test data aside for file submission prediction.

So, final shape if our training dataset is 15450 rows and 7 columns and test data is of 9914 rows and 6 columns.

6.5 Column Categorising and Final Columns

So we will categorise the columns and give a list of final columns that are selected for the model.

Train Data			Test Data		
Column	Category	Data Type	Column	Category	Data Type
passenger_count	Categorical	int64	passenger_count	Categorical	int64
distance	Continuous	float64	distance	Continuous	float64
month	Categorical	int64	month	Categorical	int64
day	Categorical	int64	day	Categorical	int64
dayofweek	Categorical	int64	dayofweek	Categorical	int64
hour	Categorical	int64	hour	Categorical	int64
fare_amount	Continuous	float64			

Chapter 7 Model Building

We have done all the analysis and preprocessing for our dataset. And at last we have devied it again into test and train. We will further our process by fitting this data into many models and predict the fare amount. And after that we will evaluate each model. On the basis of our evaluation, we will select the best model for final prediction of our test data.

The models that we will use are

- Linear Regression
- Polynomial Regression
- Decission Tree
- Random Forest
- KNN
- XGBOOST
- Hyper Parameter Tuning
    - Grid Search [Random Forest and XGBOOST]

- Random Search [Random Forest XGBOOST]

We will evaluate each and every model on the basis of

- Root Mean Square Error
- R^2 Score

7.1 Splitting Train set into test and train for modeling

We will first split the dataset into test and train. We will keep 20% of the data for testing our model and rest 80% for training; This is done to best train our model.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test = train_test_split(x,y, test_size = 0.20,
                                                   random_state=0)
```

We will be predicting R^2 score and RMSE for train set also in order to check if our model is overfitted or not.

7.2 Linear Regression

Now we will first train our model with Multiple Linear Regression, Multiple linear regression is the most common form of linear regression analysis. Multiple regression is an extension of simple linear regression.

Following is the screenshot of the model that we train.

```
In [21]: from sklearn.linear_model import LinearRegression
    ...:
    ...: #Linear Regression
    ...: X_train,X_test,y_train,y_test=splitter(X,y)
    ...: regressor = LinearRegression()
    ...: regressor.fit(X_train,y_train)
    ...: y_pred=regressor.predict(X_test)
    ...: y_pred_train = regressor.predict(X_train)
    ...:
    ...:
    ...: y_pred_train = regressor.predict(X_train)
    ...:
    ...: #Evaluation Metric
    ...: rmse=np.sqrt(mean_squared_error(y_test,y_pred))
    ...: r2_sc= r2_score(y_test,y_pred)
    ...: rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
    ...: r2_sc_train = r2_score(y_train,y_pred_train)
    ...: coeff_df = pd.DataFrame(regressor.coef_,X.columns,columns=['Coefficient'])
    ...:
    ...: print("RMSE Score for test is : ", rmse)
    ...: print("R^2 Score for test is :", r2_sc)
    ...: print("RMSE Score for train is : ",rmse_train )
    ...: print("R^2 Score for train is :", r2_sc_train)
    ...: coeff_df
RMSE Score for test is :  0.26786078854953066
R^2 Score for test is : 0.7513812211858741
RMSE Score for train is :  0.2783553888710978
R^2 Score for train is : 0.7395048196286234
Out[21]:
                Coefficient
passenger_count     0.004327
distance            0.769521
month               0.004432
day                 0.000108
dayofweek          -0.003225
hour                0.000362
```

As we can see that our analysis was right and distance is giving maximum weightage to the output.

7.3 Polynomial Regression

Polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y.

```
In [18]: from sklearn.linear_model import LinearRegression
    ...: from sklearn.preprocessing import PolynomialFeatures
    ...:
    ...: #Polynomial Regression
    ...: regressor = LinearRegression()
    ...: regressor.fit(X_train,y_train)
    ...: y_pred=regressor.predict(X_test)
    ...:
    ...: poly_reg = PolynomialFeatures(degree = 4)
    ...: X = poly_reg.fit_transform(X)
    ...: poly_reg.fit(X,y)
    ...: X_train,X_test,y_train,y_test=splitter(X,y)
    ...: lin_reg_2 = LinearRegression()
    ...: lin_reg_2.fit(X_train, y_train)
    ...: y_pred = lin_reg_2.predict(X_test)
    ...:
    ...:
    ...: y_pred_train = regressor.predict(X_train)
    ...:
    ...: #Evaluation Metric
    ...: rmse=np.sqrt(mean_squared_error(y_test,y_pred))
    ...: r2_sc= r2_score(y_test,y_pred)
    ...: rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
    ...: r2_sc_train = r2_score(y_train,y_pred_train)
    ...:
    ...:
    ...: print("RMSE Score for test is : ", rmse)
    ...: print("R^2 Score for test is :", r2_sc)
    ...: print("RMSE Score for train is : ",rmse_train )
    ...: print("R^2 Score for train is :", r2_sc_train)
RMSE Score for test is :  0.2435018962959394
R^2 Score for test is : 0.794543285826862
RMSE Score for train is :  0.2547302851021791
R^2 Score for train is : 0.7818467995350316
```

7.4 Decision Tree

A decision tree algorithm is a tree based structure. Which starts from it roots and enlarges to n number of nodes. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making.

Below is the screenshot of Decision Model

```
In [23]: from sklearn.tree import DecisionTreeRegressor
    ...: #Decision Tree
    ...: X_train,X_test,y_train,y_test=splitter(X,y)
    ...: regressor = DecisionTreeRegressor(random_state=0)
    ...: regressor.fit(X_train,y_train)
    ...: y_pred = regressor.predict(X_test)
    ...:
    ...:
    ...: y_pred_train = regressor.predict(X_train)
    ...:
    ...: #Evaluation Metric
    ...: rmse=np.sqrt(mean_squared_error(y_test,y_pred))
    ...: r2_sc= r2_score(y_test,y_pred)
    ...: rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
    ...: r2_sc_train = r2_score(y_train,y_pred_train)
    ...:
    ...:
    ...: print("RMSE Score for test is : ", rmse)
    ...: print("R^2 Score for test is :", r2_sc)
    ...: print("RMSE Score for train is : ",rmse_train )
    ...: print("R^2 Score for train is :", r2_sc_train)
RMSE Score for test is :  0.360972586535292 46
R^2 Score for test is : 0.5484934181192291
RMSE Score for train is :  1.7180937182806568e-17
R^2 Score for train is : 1.0
```

7.5 Random Forest

Random Forest is the advance version of decision tree model. In the model many decision tree are an ensemble to form a method for classification, regression and other task. It output a class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees habit of overfitting to their training set.

Below is the screenshot of Random Forest model

```
In [24]: from sklearn.ensemble import RandomForestRegressor
    ...: #Random Forest
    ...: X_train,X_test,y_train,y_test=splitter(X,y)
    ...: regressor = RandomForestRegressor(n_estimators=200)
    ...: regressor.fit(X_train,y_train)
    ...: y_pred = regressor.predict(X_test)
    ...:
    ...:
    ...: y_pred_train = regressor.predict(X_train)
    ...:
    ...: #Evaluation Metric
    ...: rmse=np.sqrt(mean_squared_error(y_test,y_pred))
    ...: r2_sc= r2_score(y_test,y_pred)
    ...: rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
    ...: r2_sc_train = r2_score(y_train,y_pred_train)
    ...:
    ...:
    ...: print("RMSE Score for test is : ", rmse)
    ...: print("R^2 Score for test is :", r2_sc)
    ...: print("RMSE Score for train is : ",rmse_train )
    ...: print("R^2 Score for train is :", r2_sc_train)
RMSE Score for test is :  0.2482687154348595
R^2 Score for test is : 0.7864204647139373
RMSE Score for train is :  0.099069519158166 35
R^2 Score for train is : 0.9670025381322639
```

7.6 KNN (k-Nearest Neighbors)

KNN is an algorithm that works on the

Scrrenshot of KNN algorithm performed.

```
In [25]: from sklearn.neighbors import KNeighborsRegressor
    ...: #KNN
    ...: X_sc = StandardScaler()
    ...: X = X_sc.fit_transform(X)
    ...: y_sc = StandardScaler()
    ...: y=np.array(y).reshape(-1,1)
    ...: y = y_sc.fit_transform(y)
    ...: X_train,X_test,y_train,y_test=splitter(X,y)
    ...: regressor = KNeighborsRegressor(n_neighbors=5)
    ...: regressor.fit(X_train,y_train)
    ...: y_pred = regressor.predict(X_test)
    ...: y_pred_train = regressor.predict(X_train)
    ...:
    ...: #inverse transform
    ...: y_pred = y_sc.inverse_transform(y_pred)
    ...: y_test = y_sc.inverse_transform(y_test)
    ...: y_train = y_sc.inverse_transform(y_train)
    ...: y_pred_train = y_sc.inverse_transform(y_pred_train)
    ...:
    ...:
    ...: #Evaluation Metric
    ...: rmse=np.sqrt(mean_squared_error(y_test,y_pred))
    ...: r2_sc= r2_score(y_test,y_pred)
    ...: rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
    ...: r2_sc_train = r2_score(y_train,y_pred_train)
    ...:
    ...:
    ...: print("RMSE Score for test is : ", rmse)
    ...: print("R^2 Score for test is :", r2_sc)
    ...: print("RMSE Score for train is : ",rmse_train )
    ...: print("R^2 Score for train is :", r2_sc_train)
RMSE Score for test is :  0.2728818400150596
R^2 Score for test is : 0.7419731428773264
RMSE Score for train is :  0.23399262031377235
R^2 Score for train is : 0.8159207878002381
```

7.7 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.

Belwo is the screenshot of algorithm performed

```
In [4]: from xgboost import XGBRegressor
   ...:
   ...: regressor = XGBRegressor()
   ...: regressor.fit(X_train,y_train)
   ...: y_pred = regressor.predict(X_test)
   ...:
   ...: y_pred_train = regressor.predict(X_train)
   ...:
   ...: #Evaluation Metric
   ...: rmse=np.sqrt(mean_squared_error(y_test,y_pred))
   ...: r2_sc= r2_score(y_test,y_pred)
   ...: rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
   ...: r2_sc_train = r2_score(y_train,y_pred_train)
   ...:
   ...:
   ...: print("RMSE Score for test is : ", rmse)#0.24
   ...: print("R^2 Score for test is :", r2_sc)#0.78
   ...: print("RMSE Score for train is : ",rmse_train )#0.099
   ...: print("R^2 Score for train is :", r2_sc_train)#0.96
C:\Users\naman\Anaconda3\envs\deeplearning\lib\site-packages\xgboost\core.py:587:
FutureWarning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
[13:49:12] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated
in favor of reg:squarederror.
RMSE Score for test is :  0.2361428945945233
R^2 Score for test is : 0.8067740693208437
RMSE Score for train is :  0.24022351072901474
R^2 Score for train is : 0.8059867259041935
```

7.8 Hyper Parameter Tuning

A model hyperparameter is a configuration that is external to the model and whose value cannot be estimated from data.. These are set by the data analyst ahead of training and control

implementation aspects of the model. The weights learned during training of a linear regression model are parameters while the number of trees in a random forest is a model hyperparameter. Hyperparameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyperparameters for one particular dataset will not be the best across all datasets. The process of hyperparameter tuning (also called hyperparameter optimization) means finding the combination of hyperparameter values for a machine learning model that performs the best.We will do hyperparameter tuning in two models Random Forest and XGBOOST.

Here, we will be using two hyper parameter tuning techniques.

1. Grid Search
2. Random Search

7.8.1 Grid Search Hyperparameter Tuning

This algorithm set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, the model tries each and every combination of hyper parameters provieded to it. Due to this, this optimization technique is very slow. We will be performing Grid Search on both of our models Random Forest and XG Boost.

Grid Search on Random forest.

Screenshot of algorithm performed.

```
In [5]: from sklearn.model_selection import GridSearchCV
   ...: from sklearn.ensemble import RandomForestRegressor
   ...:
   ...: #Grid Search on Random Forest
   ...: """Defining parameters"""
   ...: n_estimators = list(range(20,35,1))
   ...: depth = list(range(5,14,2))
   ...:
   ...: param_grid = dict(n_estimators = n_estimators,
   ...:                   max_depth=depth)
   ...:
   ...: """Creating Grid dearch"""
   ...: grid = GridSearchCV(estimator = RandomForestRegressor(random_state=0),
   ...:                     param_grid = param_grid,
   ...:                     cv = 5)
   ...:
   ...: grid_result = grid.fit(X_train,y_train)
   ...: y_pred = grid_result.predict(X_test)
   ...: y_pred_train = regressor.predict(X_train)
   ...:
   ...: """Cehcking Best parameter"""
   ...: print("Best Parameters are: ", grid_result.best_params_)
   ...:
   ...: #Evaluation Metric
   ...: rmse=np.sqrt(mean_squared_error(y_test,y_pred))
   ...: r2_sc= r2_score(y_test,y_pred)
   ...: rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
   ...: r2_sc_train = r2_score(y_train,y_pred_train)
   ...:
   ...:
   ...: print("RMSE Score for test is : ", rmse)
   ...: print("R^2 Score for test is :", r2_sc)
   ...: print("RMSE Score for train is : ",rmse_train)
   ...: print("R^2 Score for train is :", r2_sc_train)
Best Parameters are:  {'max_depth': 5, 'n_estimators': 31}
RMSE Score for test is :  0.23953143482507294
R^2 Score for test is : 0.8011888786531356
RMSE Score for train is :  0.24022351072901474
R^2 Score for train is : 0.8059867259041935

In [6]: print("Best Parameters are: ", grid_result.best_params_)
Best Parameters are:  {'max_depth': 5, 'n_estimators': 31}
```

Grid search XGBoost

Screenshot of algorithm performed

```
from sklearn.model_selection import GridSearchCV
from xgboost import XGBRegressor


#Grid Search on XGBOOST
"""Defining parameters"""
n_estimators = list(range(20,50,1))
depth = list(range(5,20,2))

param_grid = dict(n_estimators = n_estimators,
                  max_depth=depth)

"""Creating Grid dearch"""
grid = GridSearchCV(estimator = XGBRegressor(objective='reg:squarederror',eval_metric='rmse') ,
                    param_grid = param_grid,
                    cv = 5)

grid_result = grid.fit(X_train,y_train)
y_pred = grid_result.predict(X_test)
y_pred_train = regressor.predict(X_train)

"""Checking Best parameter"""
print("Best Parameters are: ", grid_result.best_params_)

#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)
print("R^2 Score for test is :", r2_sc)
print("RMSE Score for train is : ",rmse_train)
print("R^2 Score for train is :", r2_sc_train )
```

```
RMSE Score for test is :  0.23768826752382785
R^2 Score for test is : 0.8042367649798339
RMSE Score for train is :  0.23346341020592673
R^2 Score for train is : 0.8167524928203433

In [10]: print("Best Parameters are: ", grid_result.best_params_)
Best Parameters are:  {'max_depth': 5, 'n_estimators': 49}
```

7.8.2 Random Search

This algorithm set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.
And hence this search s way faster than Grid search

Here we will perform Random Search on two models, Random Forest and XGBOOST

Random Search on Random Forest

Screenshot of algorithm performed

```
In [12]: from sklearn.model_selection import RandomizedSearchCV
    ...: from sklearn.ensemble import RandomForestRegressor
    ...:
    ...:
    ...: #Random Search on Random Forest
    ...: """Defining parameters"""
    ...: n_estimators = list(range(20,100,1))
    ...: depth = list(range(1,30,2))
    ...:
    ...: param_random = dict(n_estimators = n_estimators,
    ...:                     max_depth=depth)
    ...:
    ...: """Creating Grid dearch"""
    ...: random = RandomizedSearchCV(estimator =
RandomForestRegressor(random_state=0),
    ...:                         param_distributions = param_random,
    ...:                         cv = 5,
    ...:                         n_iter = 5)
    ...:
    ...: random_result = random.fit(X_train,y_train)
    ...: y_pred = random_result.predict(X_test)
    ...: y_pred_train = random_result.predict(X_train)
    ...:
    ...:
    ...: """Checking Best parameter"""
    ...: print("Best Parameters are: ", random_result.best_params_)
    ...:
    ...: #Evaluation Metric
    ...: rmse=np.sqrt(mean_squared_error(y_test,y_pred))
    ...: r2_sc= r2_score(y_test,y_pred)
    ...: rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
    ...: r2_sc_train = r2_score(y_train,y_pred_train)
    ...:
    ...:
    ...: print("RMSE Score for test is : ", rmse)
    ...: print("R^2 Score for test is :", r2_sc)
    ...: print("RMSE Score for train is : ",rmse_train)
    ...: print("R^2 Score for train is :", r2_sc_train )
Best Parameters are:  {'n_estimators': 82, 'max_depth': 7}
RMSE Score for test is :  0.23823713080736258
R^2 Score for test is : 0.8033316188384165
RMSE Score for train is :  0.23454443263690206
R^2 Score for train is : 0.81505155568848
```

Random Search on XGBOOST

Screenshot of algorithm performed

```
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBRegressor


#Random Search on XGBOOST
"""Defining parameters"""
n_estimators = list(range(20,100,1))
depth = list(range(5,100,1))

param_random = dict(n_estimators = n_estimators,
                    max_depth=depth)

"""Creating Random earch"""
random = RandomizedSearchCV(estimator = XGBRegressor(objective='reg:squarederror',eval_metric='rmse') ,
                    param_distributions = param_random,
                    cv = 5,
                    n_iter = 5)

random_result = random.fit(X_train,y_train)
y_pred = random_result.predict(X_test)
y_pred_train = random_result.predict(X_train)


"""Checking Best parameter"""
print("Best Parameters are: ", random_result.best_params_)
#Best Parameters are:  {'n_estimators': 53, 'max_depth': 61}
#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)
print("R^2 Score for test is :", r2_sc)
print("RMSE Score for train is : ",rmse_train)
print("R^2 Score for train is :", r2_sc_train )
```

```
RMSE Score for test is :  0.25824923723322646
R^2 Score for test is : 0.7689033032446105
RMSE Score for train is :  0.032477363588779 59
R^2 Score for train is : 0.9964538103324027
```

Chapter 8 Conclusion

8.1 Model Evaluation

We will be evaluating our model on the basis of

1. RMSE (Root Mean Square Error): is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled. RMSE is an absolute measure of fit. RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

2. R Squared(R^2): is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. In other words, we can say it explains as to how much of the variance of the target variable is explained. R-squared is basically explains the degree to which input variable explain the variation of the output. In simple words R-squared tells how much variance of dependent variable explained by the independent variable. It is a measure if goodness of fit in regression line.

3. We have shown both train and test data results, the main reason behind showing both the results is to check whether our data is overfitted or not.

We have also tried MAPE instead of R^2 score, but as our values for some of the cases is very less i.e. near to zero, so MAPE in some models was giving infinity value, whcih cannot be true as the error percentange cannot be infiinity. So we decided to R^2 score in place of MAPE

So basically, value of R-squared is between 0-1, where 0 means independent variable unable to explain the target variable and 1 means target variable is completely explained by the independent variable. So, Lower values of RMSE and higher value of R-Squared Value indicate better fit of model.

8.2 Model Selection

On the basis of RMSE and R Squared, we will select the best model whose RMSE value is least and R^2 value is maximum.

Table displaying all models values

	Test		Train	
Model	RMSE	R Squared	RMSE	R Squared
Linear Regression	0.26	0.75	0.27	0.74
Polynomial Regression	0.24	0.79	0.25	0.78
Decision Tree	0.36	0.54	1.71	1
Random Forest	0.25	0.79	0.09	0.96
KNN	0.27	0.74	0.23	0.81
XGBoost	0.236	0.806	0.24	0.805
GS Random Forest	0.234	0.801	0.24	0.8
GS XGB	0.237	0.804	0.23	0.8
RS Random Forest	0.238	0.803	0.23	0.81
RS XGB	0.258	0.76	0.032	0.99

On the basis of RMSE and R Squre we can concolude that

- Decision Tree is the least performing algorithm.
- There is not much difference in the score of top performing algorithms.
- Though Hyper parameter tunning seems to be of great help for our Random Forest Model but it seems to have no improvement in the case of XGBoost.
- XGBoost whithout the help of hyper parameter tunning is performing almost similar to that of after tuning.

So from the above table we ca clearly see that **XGBoost is the best performing algorithm**. And so we will finally use this model for our test case prediction. The RMSE of XGboost is minimum and the value of R^2 are also acceptable. The values difference of RMSE for train and test data is vary less so there is no any p roblem of model over-fitting.

Hence we have used XGBoost for the prediction of test case and have predicted the fare amount and saved the file with column “predicted_fare” in file “submission.csv” for submission.

References

1. For Data Cleaning and Model Development - https://edwisor.com/career-data-scientist
2. For other code related queries https://www.analyticsvidhya.com/blog/2016/03/practicalguide-principal-component-analysis- python/
3. For Visualization using seaborn. https://www.geeksforgeeks.org/plotting-graph-using-seaborn-python/

Appendix

Python Code:

```
#!/usr/bin/env python3

#Importing Libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import folium as fo
from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV
from geopy.distance import great_circle
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor

#Function to check MMulticolinearity
def calculate_vif_(X, thresh=5.0):
    variables = list(range(X.shape[1]))
    dropped = True
    while dropped:
        dropped = False
        vif = [variance_inflation_factor(X.iloc[:, variables].values, ix)
               for ix in range(X.iloc[:, variables].shape[1])]

        maxloc = vif.index(max(vif))
        if max(vif) > thresh:
            print('dropping \'' + X.iloc[:, variables].columns[maxloc] +
                  '\' at index: ' + str(maxloc))
```

```
            del variables[maxloc]
            dropped = True

    print('Remaining variables:')
    print(X.columns[variables])
    return X.iloc[:, variables]

#Train Test Split
def splitter(X,y):
      """fuction for test train spli"""
      X_train,X_test,y_train,y_test = train_test_split(X,y,
                                          test_size = 0.20,
                                          random_state=20)
      return (X_train,X_test,y_train,y_test)


#############################################################################
###########

"""Importing dataset"""
os.chdir('D:\MY PROGRAMMING DATA\edwisor\Project02')
train = pd.read_csv("train_cab.csv")
test = pd.read_csv("test.csv")

#############################################################################
###########

############################----DATA Exploration----
#################################

#############################################################################
#########

"""DATA Preprocessing"""
train.shape
#(16067, 7)
"""So our Train data has 16067 roes and 7 variables"""
test.shape
#(9914, 6)
"""The test data has shape of 9914 rows and 6 columns"""
"""So in this data set we have fare_amount as Dependent variable and rest 6 variables
as Independent variable"""

###Univariate Analysis
```

```
train.dtypes
#fare_amount          object
#pickup_datetime      object
#pickup_longitude     float64
#pickup_latitude      float64
#dropoff_longitude    float64
#dropoff_latitude     float64
#passenger_count      float64
"""Here we can see that our fare_amount is object  and pickup_datetime is object
type. Rest all the variables are in float. lets look at the each variable one by one.
So we first need to convert fare_amount in float and then convert passenger count in int
as passenger count cannot be in float. also there are special characters in the
fare_amount feature
, So we will replace them as well"""
#changing fare_amount in float and dropping negative values
train=train.drop(2486)
train=train.drop(2039)
train=train.drop(13032)
train.fare_amount = train.fare_amount.replace({"-":""},regex=True)
train.fare_amount = train.fare_amount.astype(float)
train = train.reset_index(drop=True)

#Changing pickup_datetime to date time format of train dataset
train.pickup_datetime[1327] = np.NAN
train.pickup_datetime = pd.to_datetime(train.pickup_datetime,format =
'%Y-%m-%d %H:%M:%S UTC')
"""Thoiugh we have put a nan value as the value on that index was of simple int. So
the conversion to datetime object of this variable is not fully and we will remove the nan
value
in removing NA section"""

"""Rest all variables are float,for longitude and latitudes its ok but not for passenger
but as NA values are there in NA so we will cover this also under NA removal section"""

########################Univariate Analysis
test.dtypes
#pickup_datetime      object
#pickup_longitude     float64
#pickup_latitude      float64
#dropoff_longitude    float64
#dropoff_latitude     float64
#passenger_count       int64
"""Here the Passenger count is in int which is acceptable but pickup_datetime is in
object so we need to convert it in date time object"""
```

```
#Changing pickup_datetime to date time format of test dataset
test.pickup_datetime = pd.to_datetime(test.pickup_datetime,format =
'%Y-%m-%d %H:%M:%S UTC')

########################Dealing with Outliers and NA
train.isna().sum()
#fare_amount         24
#pickup_datetime      1
#pickup_longitude     0
#pickup_latitude      0
#dropoff_longitude    0
#dropoff_latitude     0
#passenger_count      0

test.isna().sum()
#pickup_datetime      0
#pickup_longitude     0
#pickup_latitude      0
#dropoff_longitude    0
#dropoff_latitude     0
#passenger_count      0

#boxplot Fare_amount
fig1=sns.boxplot(x = train.fare_amount, orient = 'vertical')
plt.yscale('log')
plt.title("Box Plot for Fare Amount")
"""Though there are some values in the variable which can be an outlier but we cannot
say as that its an outlier or not as money can be high enough. So we will replace only
NA values
and reconsider this point after we include some features in feature engineering"""
#lets see sort this varibale into descending order and then see what are those high
values
train.sort_values(by='fare_amount',ascending=False).head(20)
"""We have two values 54343 and 4343 in fare_amount but we cannot say that this can
be an outlier.
Though 54343 can be an outlier but we cannot say with surerity"""
#Replacing
x = train.fare_amount
x[1015] = np.nan
x.mean() # 11.6
train.fare_amount.mean() #15.04
"""We can see that replacing 54343 alue with NAN and then replacing then comparing
the mean with
```

```
unremoved 54343 value, there is not much differenc in the mean. So we will replace
NAN values with mean"""
train.fare_amount.fillna(15.04,inplace = True)

#Replacing NA of pickup_datetime
train.pickup_datetime.fillna(method='ffill',inplace = True)

#boxplot Passenger_count
#lets see sort this varibale into descending order and then see what are those high
values
train.sort_values(by='passenger_count',ascending=False).head(20)
"""So we go few values ranging from 35 to 5345 which is not possible for a cab service
So we need to replace these outliers"""
index_list = []
for i in range(len(train)):
      if train.passenger_count[i] > 6:
            index_list.append(i)
for i in range(len(train)):
      if train.passenger_count[i] <1:
            index_list.append(i)

train = train.drop(index_list)
train = train.dropna()
train = train.reset_index(drop=True)
#rounding off the values in passenger count and removing NA
train.passenger_count = train.passenger_count.astype(int)
fig2=sns.countplot(x = train.passenger_count)
plt.title("Box Plot for Passenger Count")



#Checking Na values if left
train.isna().sum()
#fare_amount         0
#pickup_datetime      0
#pickup_longitude     0
#pickup_latitude      0
#dropoff_longitude    0
#dropoff_latitude     0
#passenger_count      0

#Checking Longitude an latitude. Latitudes range from -90 to 90 and longitudes range
from -180 to 80
#train
```

```
#Pickup
train.pickup_latitude.min() #-74.006893
train.pickup_latitude.max() # 401.083332
train.pickup_longitude.min() # -74.438233
train.pickup_longitude.max() # 40.766125
#Dropoff
train.dropoff_latitude.min() # -74.006377
train.dropoff_latitude.max() # 41.366138
train.dropoff_longitude.min() # --74.42933199999999
train.dropoff_longitude.max() # 40.802437
#train
#Pickup
test.pickup_latitude.min() #-40.573143
test.pickup_latitude.max() # 41.709555
test.pickup_longitude.min() # -74.252193
test.pickup_longitude.max() # -72.986532
#Dropoff
test.dropoff_latitude.min() # 40.568973
test.dropoff_latitude.max() # 41.696683
test.dropoff_longitude.min() # -74.263242
test.dropoff_longitude.max() # -72.990963

#deleting few cases where longitude nd latitude are irrelevant wrt to the data
train = train.drop(6927)# latitude 401
train = train.drop(1135) # lat approx to 0
train = train.drop(5782) #lon approx to -7
train = train.drop(5605)
train = train.drop(train[train['pickup_latitude']==0].index)
train = train.drop(train[train['dropoff_longitude']==0].index)
train = train.reset_index(drop=True)


"""For some of the lat lon. when investigated are pointing towards artic ocaen which is
not relevant.
After investigating further, I got to to know that for those indexes the values of latitudes
and longitudes
have been interchnageed. If we interchnage them again then we will get locations of
manhattan or Queens which
is much more relevant. So we will interchnage them"""
xyz = train[round(train['pickup_longitude']) == 41]
train.loc[xyz.index,['pickup_longitude','pickup_latitude']]=train.loc[xyz.index,['pickup_latitu
de','pickup_longitude']].values
train.loc[xyz.index,['dropoff_longitude','dropoff_latitude']]=train.loc[xyz.index,['dropoff_latit
ude','dropoff_longitude']].values
```

```
del xyz

"""Great we dont have any NA value now and our dataset is complete and now we can
proceed with
Data visualization"""


##############################################################################
###########

############################----DATA Visualization----
##############################

##############################################################################
#########

############################Univariate Visualization Analysis
"""Lets first Cobine our dataset test and train in order to better visualize the ups an
downs of the data"""
data = pd.concat([train[['pickup_datetime', 'pickup_longitude', 'pickup_latitude',
                    'dropoff_longitude', 'dropoff_latitude', 'passenger_count']],
      test])
data = data.reset_index(drop=True)


#creating Map of datapoints
map = fo.Map(location = [40.721319, -73.844311],zoom_start=6,tiles =
"OpenStreetMap")
fop = fo.FeatureGroup(name="Pickup")
for lt,ln in zip(data['pickup_latitude'], data['pickup_longitude']):
   fop.add_child(fo.CircleMarker(location=[float(lt),float(ln)], radius = 1, color = 'blue',
                                 popup=str(lt)+','+str(ln),fill=True, fill_opacity=0.7))
map.add_child(fop)
map.save("Cab_route_map.html")
"""We can see that all the trips is from NewYork. Major City is Manhattan and frw from
JFK Queens and Broklyn.
lets see the distribution"""


#Dataset Fare Amount
fig3 = plt.hist(train.fare_amount,bins=200)
plt.yscale('log')
plt.title("Density plot for Fare Amount")
plt.ylabel("Count")
```

"""So we can see that most of the values are near to 20 and on ly few values are near to 100.
We can also see that one is near to 4000 and another value is above 50000. We expect both of these
values are outliers. But lets dig more deep"""

#Dataset Pickup Datetime
"""Lets see some visualization on Date time"""
fig4 = sns.countplot(data.pickup_datetime.dt.year)
plt.ylabel("Count")
plt.xlabel("Year")
plt.title("Year Wise distribution")
"""We have a dataset of 7 years from 2009 to 2015. We got to know that the mximum number of rides were in
the year 2012 and the minimum number of rides were in theyear 2015. Also we got to know that after 2012
the rides get decressed countinously and in 2015 thw rides were nearly half of the rest of the year"""

"""Lets see how much data we have for the year2015"""
fig5 = plt.hist(data[data['pickup_datetime'].dt.year ==
2015].pickup_datetime.dt.month,bins=60,width=0.4)
plt.ylabel("Count")
plt.xlabel("Month")
plt.title("Year 2015, month wise distribution")
"""So we got to know that for the year 2015 we have data for only 6 months"""

"""Lets see month wise dristribution of rides for all the years"""
fig6=sns.distplot(data.pickup_datetime.dt.month)
plt.xticks(range(1,13))
plt.ylabel("Count")
plt.xlabel("Month")
plt.title("Month Wise Distribution")
""" We can see that there is drop in the second month. This might be due to the snow fall in New York.
And also there is a huge amount of rides in mid summer. That is the vacation time. So tourist visit increases
the rides count. We can add month feature in the feature enginnering"""

"""Lets see week wise distribution"""
fig7=sns.distplot(data.pickup_datetime.dt.dayofweek,kde=False)
plt.xticks(range(0,7))
plt.ylabel("Count")
plt.xlabel("Day of the Week")

```
plt.title("WeekWise Distribution")
"""From this we can see that there is huge ammount of rides for the first two days
i.e.Monday and Tuesday.
But it decreases drastically on Wednusday. But it again starts increasing on friday and
saturday but descrese
on Sunday. That means majorly cab is used for office purpose as it has highest number
of rides on Monday and
Tuesday. And on weekends people dont use it. It can be a good factor in estimating the
fare amount. So we will
add this factor in the feature enginnering section"""

"""Lets again see day wise distribution. And check if we can find any pattern there"""
fig8=sns.distplot(data.pickup_datetime.dt.day,bins = 100,kde=False)
plt.xticks(range(1,32))
plt.ylabel("Count")
plt.xlabel("Days of Month")
plt.title("Days Wise Distribution")
"""In this distribution we can see two drops, one at the middle of the month and another
at the end of the month"""

data['month'] = data.pickup_datetime.dt.month
data['hour'] = data.pickup_datetime.dt.hour

"""Lets check the hour wise distribution"""
fig9 = sns.countplot(data.pickup_datetime.dt.hour)
sns.pointplot(data.pickup_datetime.dt.hour.sort_values().unique(),

data['pickup_datetime'].groupby(by=data.pickup_datetime.dt.hour).count(),size=10)
plt.ylabel("Count")
plt.xlabel("Hours")
plt.title("Hour Wise Distribution")
"""This tells us much about the usage of cab rides. In the early morning from 4 to 6 there
are very less rides.
 Rides increases drastically from 7 AM to 12 PM with some ups and downs. Then after
12 PM the count of rides
 descreses steadily till 3 PM as its noon time and less people which travel. Then the
rides count starts increasing
 from 4PM as the office shifts over. And the count remains at a higher count till midnight.
It dreases over the time span of 00 AM till  3AM. This seems to be a very important
visualization and we will
include hours column in feature engineering."""

"""Lets see hour wise distribution wrt to Month"""
fig10=data.groupby([data.pickup_datetime.dt.hour,
```

```
        data.pickup_datetime.dt.month]).count()['hour'].unstack().plot(kind='line',
title="Hour Wise Distribution wrt to
Month",xticks=(range(0,24)),grid=True,legend=True,linewidth=5)
plt.xlabel("Hour")
plt.ylabel("Count")
plt.legend(["Jan","Feb","MArch","April",'May','June','July','Aug','Spt','Oct','Nov','Dec'])


"""For all the month 3AM-6AM drop is common
There is certain peaks for almost all tof the months but at different hours of the day
This might be because of the weather conditions. As summer months has peaks from
11-19.
Winetr months has peaks in the late evenings. So weather is having a great impaact on
the usage"""

"""Lets see hour wise distribution wrt to Week days"""
fig11=data.groupby([data.pickup_datetime.dt.hour,
        data.pickup_datetime.dt.dayofweek]).count()['hour'].unstack().plot(kind='line',
title="Hour Wise Distribution wrt to
WeekDay",xticks=(range(0,24)),grid=True,legend=True,linewidth=5)
plt.xlabel("Hour")
plt.ylabel("Count")
plt.legend(["Sun","Mon",'Tues','Wed','Thus','Fri','Sat'])
"""Monday and uesday has peaks inearly mornings and late evenings that are the proper
office timings
Friday saturday has late night peaks.
Sunday sare stable, just few samll peaks in the evening.
Monday,Tue,Wed has good peaks in the morning hours. That means People are using
cabs mostly for office
purpose in the early mornings and in the late evenings. So timing are a major factor for
fare amount prediction"""

"""Lets see Month wise distribution wrt to Year"""
fig12=data.groupby([data.pickup_datetime.dt.month,
        data.pickup_datetime.dt.year]).count()['month'].unstack().plot(kind='line',
title="Month Wise Distribution wrt to Year",xticks=(range(1,13)),grid=True,linewidth=5)
plt.xlabel("Month")
plt.ylabel("Count")
plt.legend()

#Passenger count visualization
fig13 = sns.countplot(data.passenger_count)
plt.xlabel("Number of Passenger")
plt.ylabel("Count")
```

```
plt.title("Count plot of Passenger count")
"""We can see that maximum count is for single passenger and then for 2 passsengers.
Then the count decreasses but then increasses for 5 passenger."""

data = data.drop(['month','hour'],axis=1)
"""Analysing longitudes and latitudes"""
#Pickup Latitue
fig14=sns.distplot(data.pickup_latitude,hist = False,color='green')
plt.title("Pickup Latitude Plot")


#Pickup Longitude
fig15=sns.distplot(data.pickup_longitude,hist = False,color='blue')
plt.title('Pickup_longitude Plot')
#Dropoff Latitue
fig16=sns.distplot(data.dropoff_latitude ,hist = False,color='pink')
plt.title('Drop off Latitude Plot')

#Dropoff Longitude
fig17=sns.distplot(data.dropoff_longitude,hist = False,color='red')
plt.title('Drop off longitude Plot')

###Biivariate Analysis
#Day vs Fare Amount Plot
fig18=sns.stripplot(x=train.pickup_datetime.dt.day,y=train.fare_amount)
plt.title("Day vs Fare Amount Plot")
"""We can clearly see that the two values 54000 and 4000. So we will replace them with
the mean of
all values"""
"""Replacing values with mee=an of that month"""

index_nan=[]
index_nan = list(train[train['fare_amount']>164].index)
for i in index_nan:
    train.fare_amount[i] = np.nan
    yr = train.pickup_datetime.dt.year[i]
    xyz=train[(train.pickup_datetime.dt.year)==yr]
    mn = xyz.pickup_datetime.dt.month[i]
    mean=xyz[(xyz.pickup_datetime.dt.month)==mn]['fare_amount'].mean()
    train.fare_amount[i] = mean

"""Lets again check the plot"""
fig19=sns.stripplot(x=train.pickup_datetime.dt.day,y=train.fare_amount)
plt.title("Day vs Fare Amount Plot")
```

```
fig20=sns.violinplot(x=train.pickup_datetime.dt.day,y=train.fare_amount)
plt.title("Day vs Fare Amount Box Plot")

#Year vs Fare amount
fig21 = sns.boxplot(x=
train.pickup_datetime.dt.year,y=train.fare_amount,showmeans=True)
plt.xlabel("Year")
plt.title("Year vs Fare Amount Box Plot")
"""Same we can see from this plot also. except for few values, rest all are under same
range for each
year. Median is same for all the years. As there are some 0's in the fare amount so the
IQR is high for
2010 and 2015"""

#Hour vs Fare amount
fig22 = sns.stripplot(x= train.pickup_datetime.dt.hour,y=train.fare_amount)
plt.xlabel("Hour")
plt.title("Hour vs Fare Amount Box Plot")

#Correlation Plot
fig23=sns.heatmap(train.corr())
plt.title("Correlation plot")
"""Hence there is almost no correlation between the variables"""

#Passenger Count vs Fare Amount Box Plot
fig24=sns.boxplot(x=train.passenger_count, y=train.fare_amount)
plt.title("Passenger Count vs Fare Amount Box Plot")
"""Also the maximum amount charged was when 1 or two passenger travel. There is no
statistical difference
in fare amount when the passenger count increases. That means number of passenger
dosenot make any difference
in the fare amount"""

##########################################################################
###########

############################----Feature Enginnering---
##############################

##########################################################################
#########

#Creating Column of distance from pickup and dropoff
```

```
"""Using Great Circle Distance"""
x=[]
for i in range(len(data)):
      d1 = (data.pickup_latitude[i],data.pickup_longitude[i])
      d2 = (data.dropoff_latitude[i],data.dropoff_longitude[i])
      x.append(great_circle(d1, d2).km)
data['distance'] = x

#Viausalizing distance
fig25=sns.distplot(data.distance,color='green')
plt.title("Distance Plot")
plt.xlabel("Distance in KM")
plt.ylabel("Frequency")
"""We cab see that most of the distance traveeled by the passenger is bwlow 10 KM.
Further there
are a few peaks  above 120 also. We can also see that the some of the distance is 0, so
we need to remove
thme"""

#Visualization distance vs fare amount
fig26=sns.scatterplot(y=data.distance[train.fare_amount.index],x=train.fare_amount,
            hue=train.passenger_count)
plt.title("Distance vs Fare amount plot")
"""AS we see from the scatter plot that distance and fare amount has some what linear
relationship.
but that dosenot actually appliea for the passenger count. Except few outlier values,
where distance is high but
the amount is very less, all other values has=ve linear relation. So we need to remove
these outliers"""

index_drop = list(data[data['distance'] ==0].index)

for index in index_drop:
      if index in train.index:
            train = train.drop(index)
            data = data.drop(index)

data = data.reset_index(drop=True)
train = train.reset_index(drop=True)


fig27=sns.barplot(x=data.pickup_datetime.dt.month.unique(),
              y=data.groupby(data.pickup_datetime.dt.month)['distance'].mean())
plt.title("Distance vs Month")
```

```
#Breaking Date Time and Year into different columns
data['year'] = data.pickup_datetime.dt.year
data['month']= data.pickup_datetime.dt.month
data['day'] = data.pickup_datetime.dt.day
data['dayofweek'] = data.pickup_datetime.dt.dayofweek
data['hour'] = data.pickup_datetime.dt.hour
data = data.drop('pickup_datetime',axis = 1)



#########################################################################
#########

############################----Model PreProcessing----
############################

#########################################################################
#########

#####Breaking up data into train and test again
#Dropping longitude and latitude
data= data.iloc[:,4:]

"""Checking Multi colinearity"""


data_vif=calculate_vif_(data)
"""So here Year varibale is dropped as it has vif of more than 5"""


#Log transformation of variable Distance and Fare
"""As We have seen above in the density plot of distance and fare amount that our data
is
left skweed. So in order to fix that we have to do log transformation of both the
variables."""
data_vif.distance = np.log(data_vif.distance+1)
train.fare_amount = np.log(train.fare_amount+1)
#visualization
fig28=sns.distplot(data_vif.distance,color='green')
plt.ylabel("Density")
plt.title("Density plot of Distance after Log transformation")
#Visualization
```

```
fig29=sns.distplot(train.fare_amount)
plt.ylabel("Density")
plt.title("Density plot of Fare Amount after Log transformation")


#Breakinf the data again into test and train
X = data_vif.iloc[train['fare_amount'].index,]
test = data_vif.iloc[test.index,]
y = train.fare_amount




#############################################################################
###########

############################----Model Building----
##############################

#############################################################################
#########
#Linear Regression
X_train,X_test,y_train,y_test=splitter(X,y)
regressor = LinearRegression()
regressor.fit(X_train,y_train)
y_pred=regressor.predict(X_test)
y_pred_train = regressor.predict(X_train)


y_pred_train = regressor.predict(X_train)

#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)
coeff_df = pd.DataFrame(regressor.coef_,X.columns,columns=['Coefficient'])

print("RMSE Score for test is : ", rmse)
print("R^2 Score for test is :", r2_sc)
print("RMSE Score for train is : ",rmse_train )
print("R^2 Score for train is :", r2_sc_train)
```

```
coeff_df
 #              Coefficient
#passenger_count     0.004074
#distance            0.768508
#month               0.004535
#day                0.000090
#dayofweek          -0.003499
#hour                0.000440
"""As expected we can see that the most weightage is of variables distance"""

##########################################################################
#########
#Polynomial Regression
regressor = LinearRegression()
regressor.fit(X_train,y_train)
y_pred=regressor.predict(X_test)

poly_reg = PolynomialFeatures(degree = 4)
X = poly_reg.fit_transform(X)
poly_reg.fit(X,y)
X_train,X_test,y_train,y_test=splitter(X,y)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_train, y_train)
y_pred = lin_reg_2.predict(X_test)


y_pred_train = regressor.predict(X_train)

#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)#0.24
print("R^2 Score for test is :", r2_sc)#0.79
print("RMSE Score for train is : ",rmse_train )#0.25
print("R^2 Score for train is :", r2_sc_train)#0.78



##########################################################################
#########
```

```
#Decision Tree
X = data_vif.iloc[train['fare_amount'].index,]
y = train.fare_amount
X_train,X_test,y_train,y_test=splitter(X,y)
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(X_train,y_train)
y_pred = regressor.predict(X_test)


y_pred_train = regressor.predict(X_train)

#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)
print("R^2 Score for test is :", r2_sc)
print("RMSE Score for train is : ",rmse_train )
print("R^2 Score for train is :", r2_sc_train)



#########################################################################
#########
#Random Forest
X_train,X_test,y_train,y_test=splitter(X,y)
regressor = RandomForestRegressor(n_estimators=200)
regressor.fit(X_train,y_train)
y_pred = regressor.predict(X_test)


y_pred_train = regressor.predict(X_train)

#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)#0.24
```

```
print("R^2 Score for test is :", r2_sc)#0.78
print("RMSE Score for train is : ",rmse_train )#0.099
print("R^2 Score for train is :", r2_sc_train)#0.96


#########################################################################
#########
#KNN
X_sc = StandardScaler()
X = X_sc.fit_transform(X)
y_sc = StandardScaler()
y=np.array(y).reshape(-1,1)
y = y_sc.fit_transform(y)
X_train,X_test,y_train,y_test=splitter(X,y)
regressor = KNeighborsRegressor(n_neighbors=5)
regressor.fit(X_train,y_train)
y_pred = regressor.predict(X_test)
y_pred_train = regressor.predict(X_train)

#inverse transform
y_pred = y_sc.inverse_transform(y_pred)
y_test = y_sc.inverse_transform(y_test)
y_train = y_sc.inverse_transform(y_train)
y_pred_train = y_sc.inverse_transform(y_pred_train)


#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)
print("R^2 Score for test is :", r2_sc)
print("RMSE Score for train is : ",rmse_train )
print("R^2 Score for train is :", r2_sc_train)

#########################################################################
#########
#XGBOOST
X = data_vif.iloc[train['fare_amount'].index,]
y = train.fare_amount
X_train,X_test,y_train,y_test=splitter(X,y)
```

```
regressor = XGBRegressor(objective='reg:squarederror',eval_metric='rmse')
regressor.fit(X_train,y_train)
y_pred = regressor.predict(X_test)

y_pred_train = regressor.predict(X_train)

#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)#0.26
print("R^2 Score for test is :", r2_sc)#0.80
print("RMSE Score for train is : ",rmse_train )#0.24
print("R^2 Score for train is :", r2_sc_train)#0.80

##########################################################################
#########
#Hyperparameter tuning
"""hyperparameter tuning is choosing a set of optimal hyperparameters for a learning
algorithm.
A hyperparameter is a parameter whose value is set before the learning process begins.
We will do hyperparameter tuning in two models Random Forest and XGBOOST. Also
we will use try
both typer of hyper parameter tuning i.e. Random Search CV and Grid Search CV"""
#Grid Search on Random Forest
"""Defining parameters"""
n_estimators = list(range(20,35,1))
depth = list(range(5,14,2))

param_grid = dict(n_estimators = n_estimators,
                  max_depth=depth)

"""Creating Grid search"""
grid = GridSearchCV(estimator = RandomForestRegressor(random_state=0),
                    param_grid = param_grid,
                    cv = 5)

grid_result = grid.fit(X_train,y_train)
y_pred = grid_result.predict(X_test)
y_pred_train = regressor.predict(X_train)
```

```
"""Checking Best parameter"""
print("Best Parameters are: ", grid_result.best_params_)

#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)
print("R^2 Score for test is :", r2_sc)
print("RMSE Score for train is : ",rmse_train)
print("R^2 Score for train is :", r2_sc_train)

##########################################################################
#########
#Grid Search on XGBOOST
"""Defining parameters"""
n_estimators = list(range(45,100,1))
depth = list(range(5,15,2))

param_grid = dict(n_estimators = n_estimators,
                  max_depth=depth)

"""Creating Grid search"""
grid = GridSearchCV(estimator =
XGBRegressor(objective='reg:squarederror',eval_metric='rmse') ,
                    param_grid = param_grid,
                    cv = 5)

grid_result = grid.fit(X_train,y_train)
y_pred = grid_result.predict(X_test)
y_pred_train = grid_result.predict(X_train)

"""Checking Best parameter"""
print("Best Parameters are: ", grid_result.best_params_)
#Best Parameters are:  {'max_depth': 5, 'n_estimators': 49}

#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)
```

```
print("RMSE Score for test is : ", rmse)
print("R^2 Score for test is :", r2_sc)
print("RMSE Score for train is : ",rmse_train)
print("R^2 Score for train is :", r2_sc_train )

##########################################################################
#########

#Random Search on Random Forest
"""Defining parameters"""
n_estimators = list(range(20,100,1))
depth = list(range(1,30,2))

param_random = dict(n_estimators = n_estimators,
                max_depth=depth)

"""Creating Random search"""
random = RandomizedSearchCV(estimator =
RandomForestRegressor(random_state=0),
                  param_distributions = param_random,
                  cv = 5,
                  n_iter = 5)

random_result = random.fit(X_train,y_train)
y_pred = random_result.predict(X_test)
y_pred_train = random_result.predict(X_train)


"""Checking Best parameter"""
print("Best Parameters are: ", random_result.best_params_)
#Best Parameters are:  {'n_estimators': 82, 'max_depth': 7}

#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)
print("R^2 Score for test is :", r2_sc)
print("RMSE Score for train is : ",rmse_train)
```

```
print("R^2 Score for train is :", r2_sc_train )

##########################################################################
#########

#Random Search on XGBOOST
"""Defining parameters"""
n_estimators = list(range(20,100,1))
depth = list(range(5,100,1))

param_random = dict(n_estimators = n_estimators,
               max_depth=depth)

"""Creating Random earch"""
random = RandomizedSearchCV(estimator =
XGBRegressor(objective='reg:squarederror',eval_metric='rmse') ,
                 param_distributions = param_random,
                 cv = 5,
                 n_iter = 5)

random_result = random.fit(X_train,y_train)
y_pred = random_result.predict(X_test)
y_pred_train = random_result.predict(X_train)


"""Checking Best parameter"""
print("Best Parameters are: ", random_result.best_params_)
#Best Parameters are:  {'n_estimators': 53, 'max_depth': 61}
#Evaluation Metric
rmse=np.sqrt(mean_squared_error(y_test,y_pred))
r2_sc= r2_score(y_test,y_pred)
rmse_train=np.sqrt(mean_squared_error(y_train,y_pred_train))
r2_sc_train = r2_score(y_train,y_pred_train)


print("RMSE Score for test is : ", rmse)
print("R^2 Score for test is :", r2_sc)
print("RMSE Score for train is : ",rmse_train)
print("R^2 Score for train is :", r2_sc_train )


##########################################################################
#########
```

```
######################--Model Selection and Predicting Test case--#############

#############################################################################
#########

"""We have choosed XGBOOST to be the best model and we will predict the test case
with this model only."""

#Building Model again
X = data_vif.iloc[train['fare_amount'].index,]
y = train.fare_amount
X_train,X_test,y_train,y_test=splitter(X,y)
regressor = XGBRegressor(objective='reg:squarederror',eval_metric='rmse')
regressor.fit(X_train,y_train)

#Predicting Test case
y_pred_test = regressor.predict(test)

#Adding prediction column to test file
test = pd.read_csv("test.csv")
test['predicted_fare'] = y_pred_test

#Writting the file
test.to_csv('submission.csv',header=True,index=False)
```