Cab Fare Prediction

Prakash B

20th Jun 2019

Contents

Chapter 1

Introduction

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.

1.2 Data

We have to predict fare_amount for our test data and since our target variable is a continuous variable therefore this a Regression problem.

The details of data attributes in the dataset are as follows -

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

Table 1.1: Cab Fare of Sample Data (Columns 1-7)

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
<fct>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
4.5	2009-06-15 17:26:21 UTC	-73.84431	40.72132	-73.84161	40.71228	1
16.9	2010-01-05 16:52:16 UTC	-74.01605	40.71130	-73.97927	40.78200	1
5.7	2011-08-18 00:35:00 UTC	-73.98274	40.76127	-73.99124	40.75056	2
7.7	2012-04-21 04:30:42 UTC	-73.98713	40.73314	-73.99157	40.75809	1
5.3	2010-03-09 07:51:00 UTC	-73.96810	40.76801	-73.95665	40.78376	1

As you can see in the table below, we have the following 6 variables, using which we have to correctly predict the fare amount of the cab:

Table 1.2: Predictor variables

Sl.No	Variables
1	pickup_datetime
2	pickup_longitude
3	dropoff_longitude
4	pickup_latitude
5	dropoff_latitude
6	passenger_count

Table 1.3: Target variable

Sl.No	Variables
1	fare_amount

Chapter 2

Methodology

2.1 Data Pre-Processing

Any predictive modelling requires that we look at the data before we start modelling. However, in data mining terms looking at data refers to so much more than just looking, looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process, we will first try and look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable.

2.1.1 Missing Value Analysis

Missing data or missing values occur when no data value is stored for the variable in an observation. Missing values are a common occurrence in data analysis. These values can have a significant impact on the results or conclusions that would be drawn from these data. If a variable has more than 30% of its values missing, then those values can be ignored, or the column itself is ignored.

For the given dataset we can see that we have only two variables with missing data and too the percentage of missing value is very less, so for this case we would be removing the observations which have missing values.

Variables	Missing_Values_Count	Missing_Val_percentage
<chr>	<int>	<dbl>
passenger_count	55	0.3423165
fare_amount	25	0.1555984
pickup_datetime	0	0.0000000
pickup_longitude	0	0.0000000
pickup_latitude	0	0.0000000
dropoff_longitude	0	0.0000000
dropoff_latitude	0	0.0000000

Fig 2.1: Missing value analysis

2.1.2 Outlier Analysis

It can be observed from the distribution of variables that almost all of the variables are normally distributed except few variables because of the skewed information, The skew in these distributions can be explained by the presence of outliers and extreme values in the data. One of the steps in pre-processing involves the detection and removal of such outliers. In this project, we use boxplot to visualize and remove outliers by manually using test data feature conditions.

Outliers can be removed using the Boxplot stats method, wherein the Inter Quartile Range (IQR) is calculated and the minimum and maximum value are calculated for the variables. Any value ranging outside the minimum and maximum value are discarded

Variables windspeed and humidity contain outliers, the below figure shows the boxplot representation of the variables with outliers.

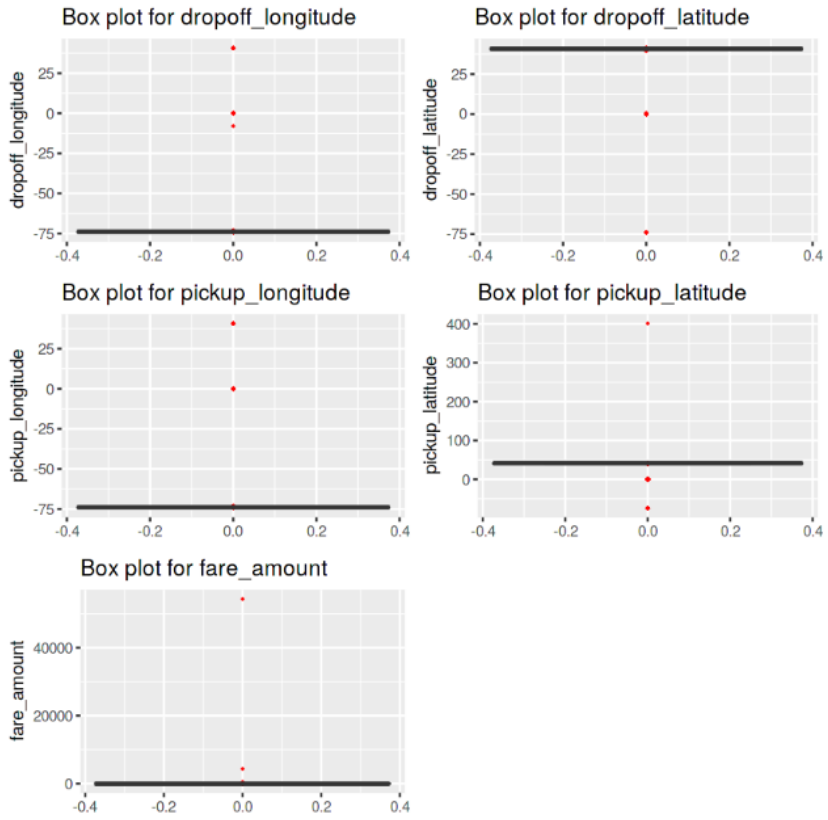
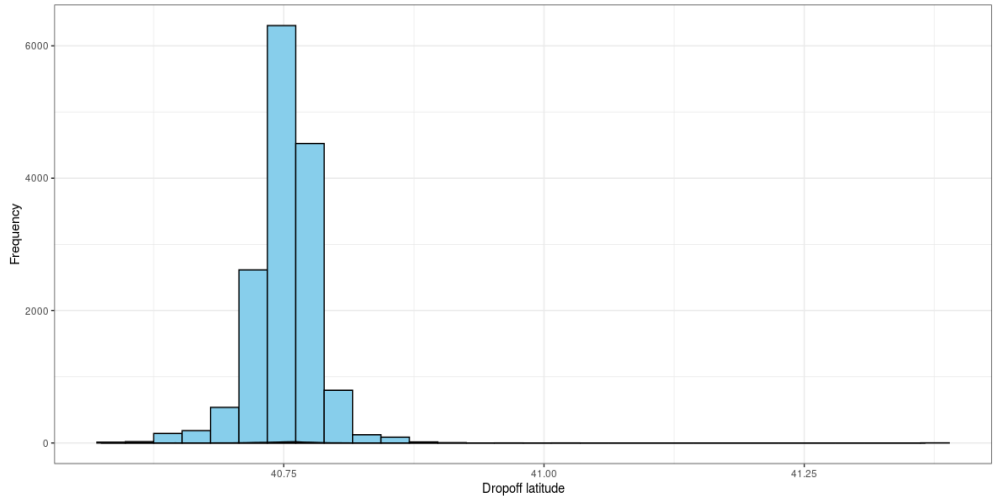


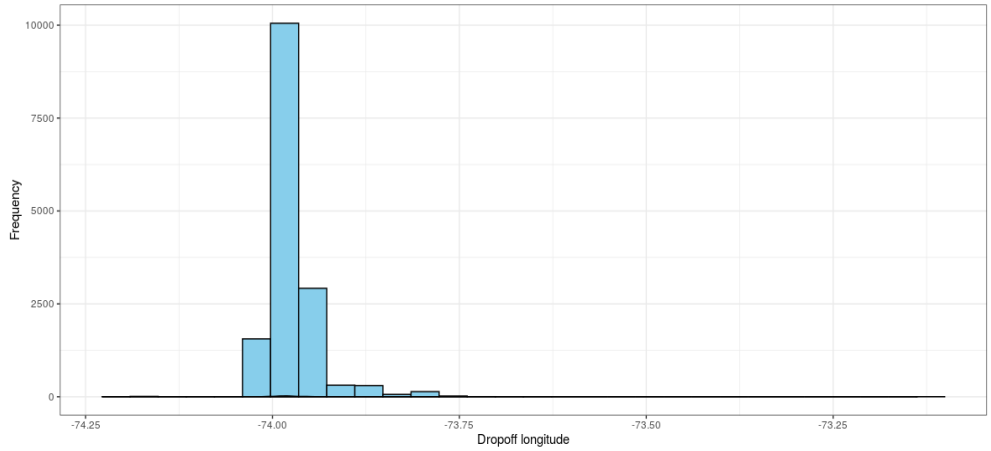
Fig 2.2: Boxplot of continuous variables with outliers

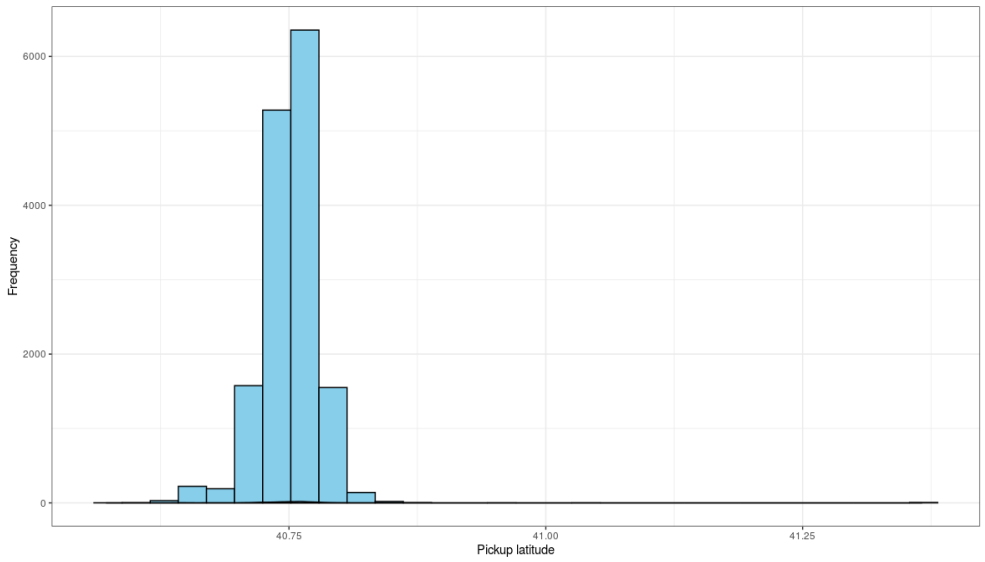
From the above plot we can observe that all columns have outliers, these outliers are removed manually based on the given test case feature conditions are also called ranges of the test data variables.

2.1.3 Distribution of Continuous Variables

From the below distributions it can be observed that fare amount and distance lightly skewed, whereas rest of the variables normally distributed. The skewness is likely because of the presence of more information or huge data in those variables.







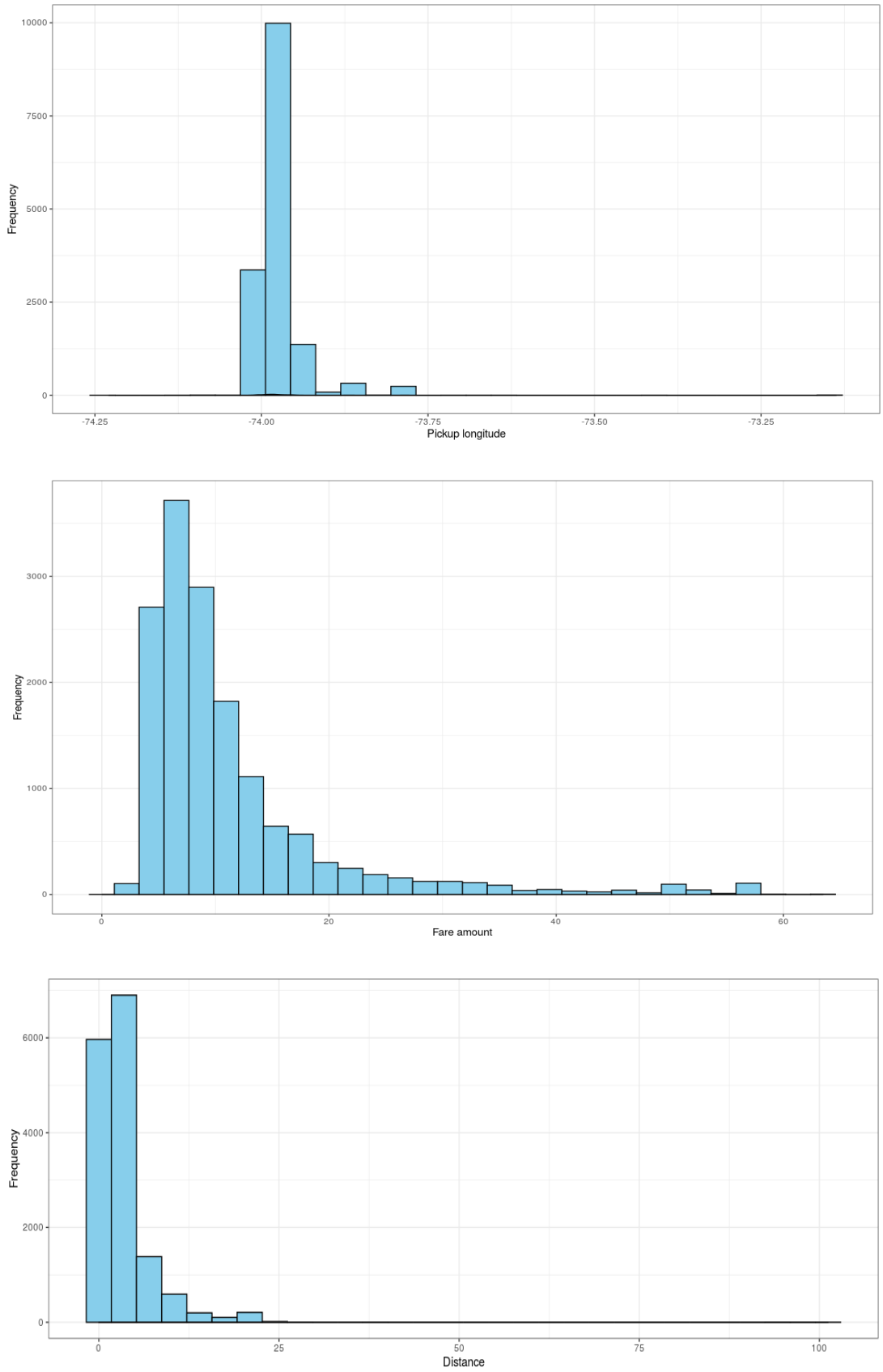
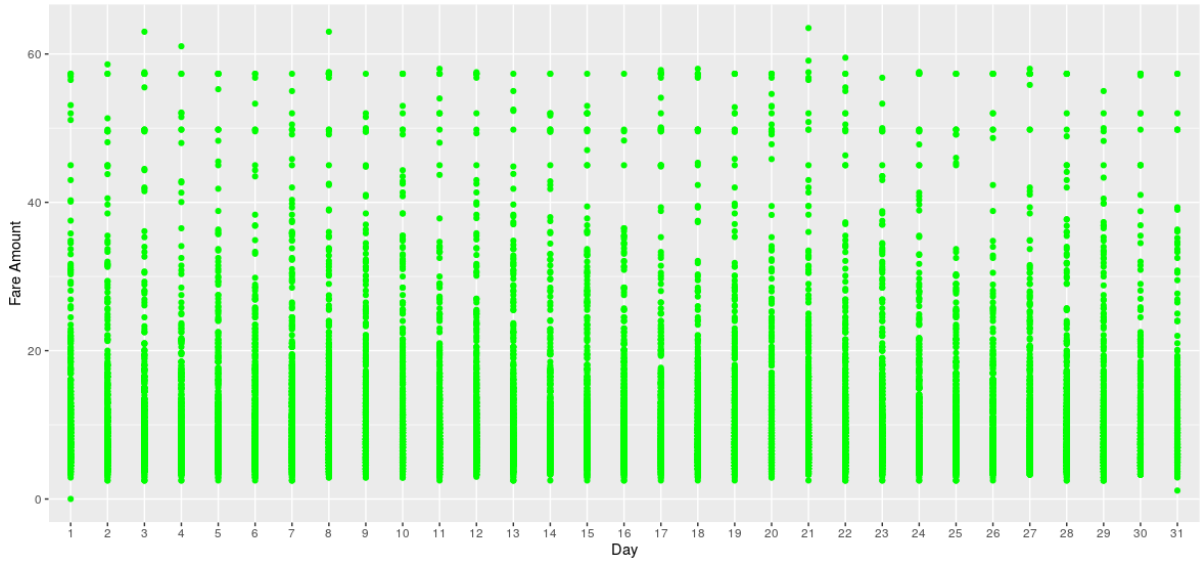


Fig 2.3: Distribution of continuous variables using histograms

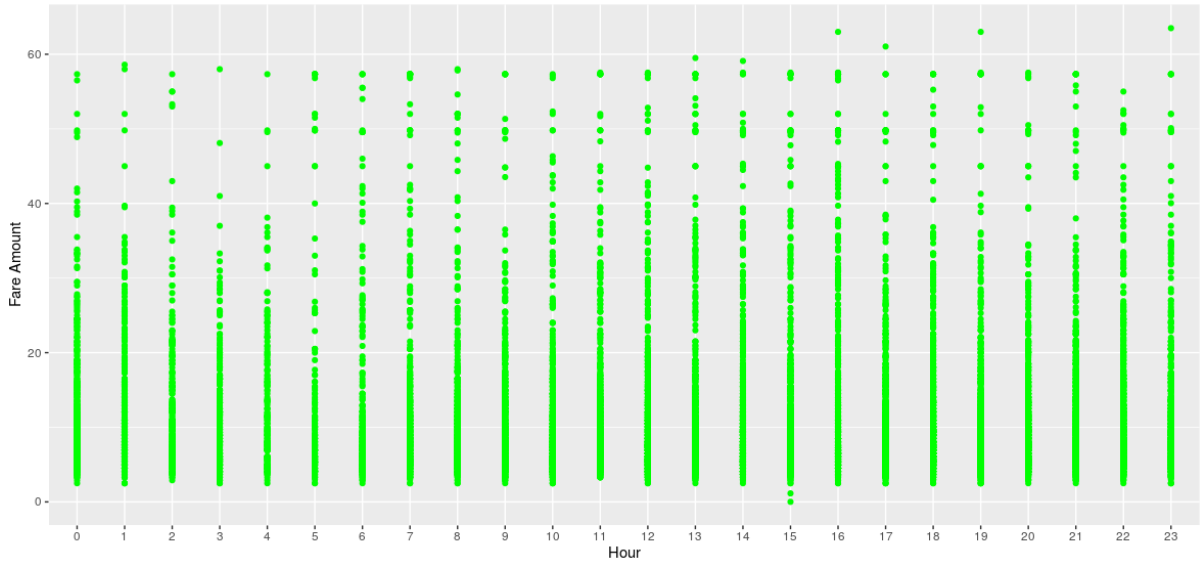
2.1.4 Relationship of Continuous Variable against target Variable

The Below figure shows the relationship between the continuous variables against target variable (fare amount) using Scatter plot, from the figure we can see that how the data is scattered, and there exists a linear positive relationship between the variables distance and against the target variable.

- **Check the pickup date and time affect the fare or not**

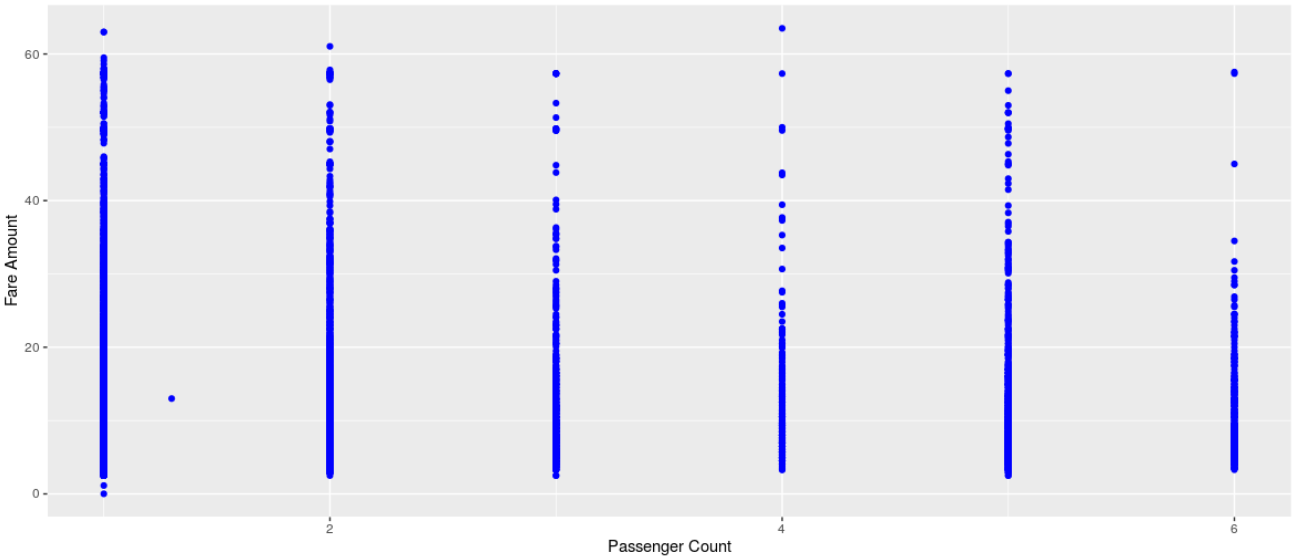


The fares throught the month mostly seem uniform



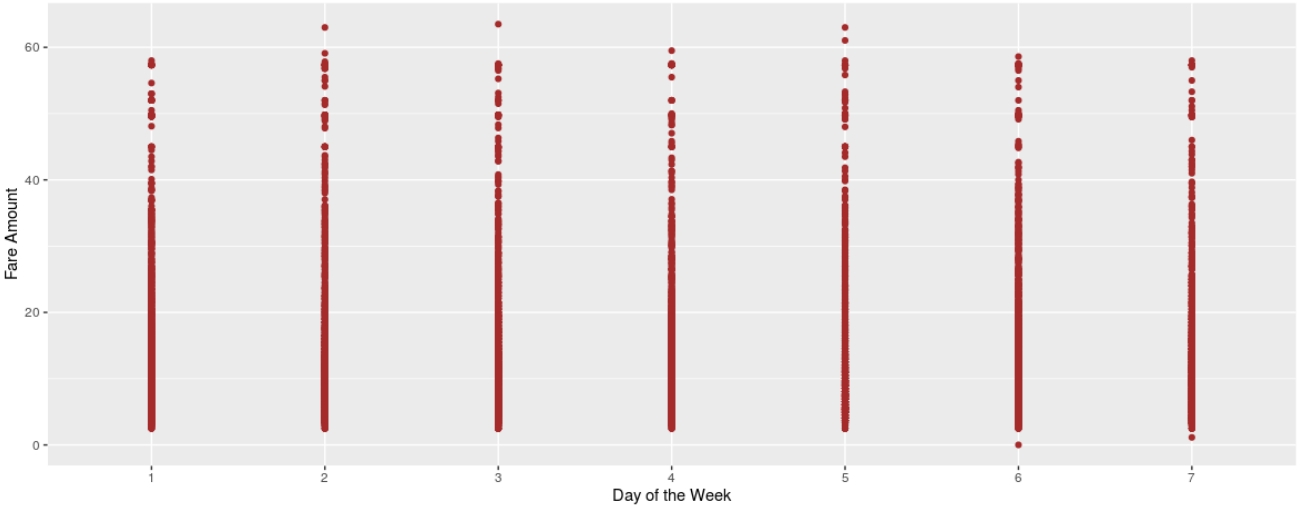
The time of day definitely plays an important role. The frequency of cab rides seems to be the lowest at 5AM

- **Number of Passengers v/s Fare**



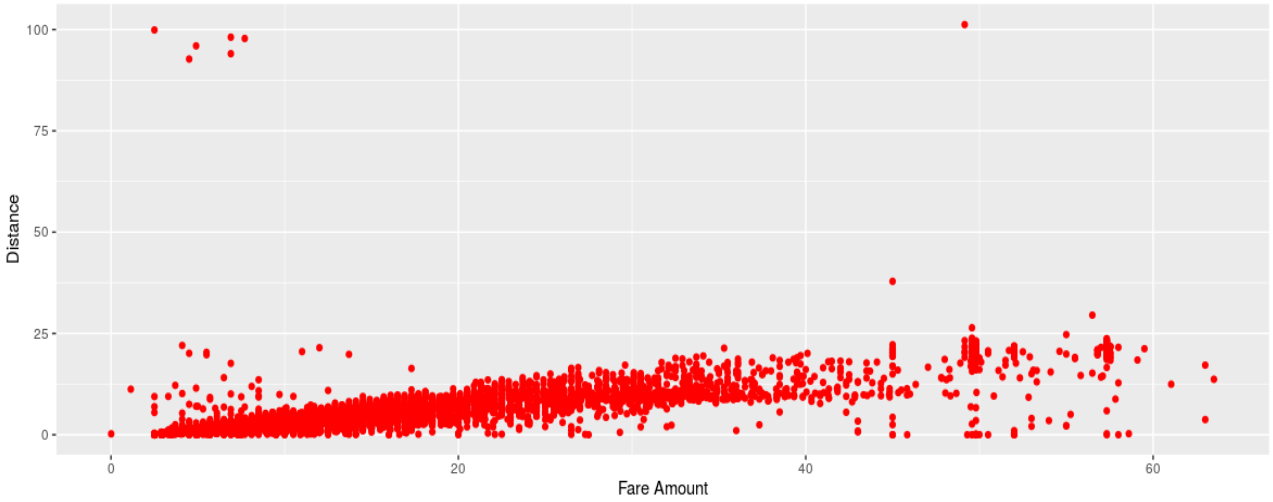
single passengers are the most frequent travellers, and the highest fare also seems to come from cabs which carry just 1 passenger.

- **Does the day of the week affect the fare?**



Day of the week doesn't seem to have the effect on the number of cab rides

- **Between distance and fare**



From the above figures we have observed that how the Fare amount varies across the different distances.

Here we can observe that more number of passengers travelled with in 1 to 25 km of distance

- **Checking the year against fare**

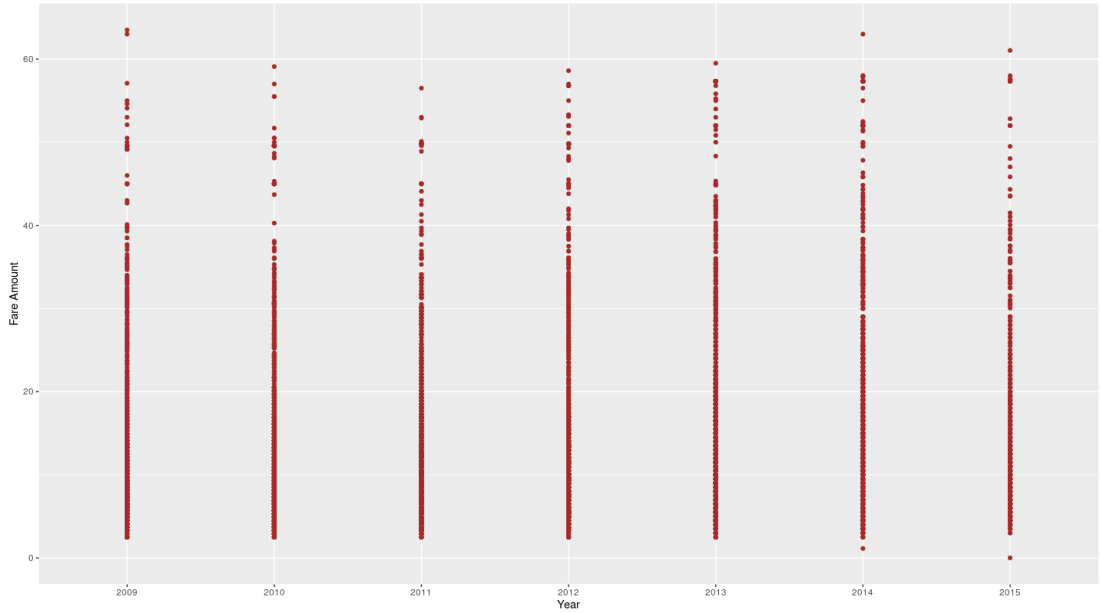


Fig 2.4: Relationship between the variables against Fare Amount

From the above figures we have observed that how the Fare amount varies across the Years.

2.1.5 Feature Selection

Feature Selection reduces the complexity of a model and makes it easier to interpret. It also reduces overfitting. Features are selected based on their scores in various statistical tests for their correlation with the outcome variable. Correlation plot is used to find out if there is any multicollinearity between variables. The highly collinear variables are dropped and then the model is executed.

From the correlation plot we can observe that there is no multicollinearity occurred in the variables, and we have observed that distance variable highly correlated with the target variable, it means that distance variable carries more information to predict the target variable.

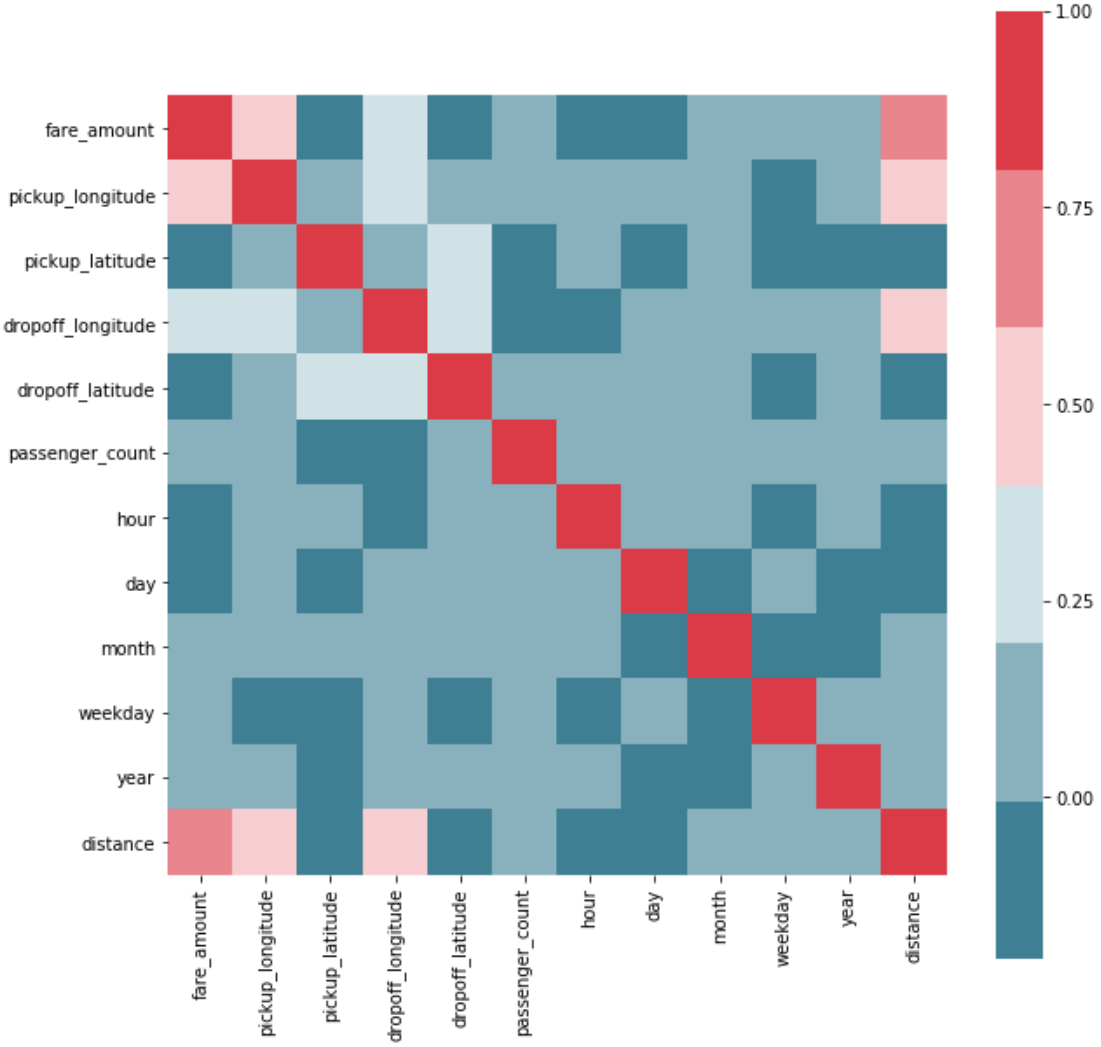


Fig 2.5: Correlation plot of Continuous variables

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	hour	day	month	weekday	year	distance
fare_amount	1.000000	-0.001076	0.000778	-0.001053	0.000995	-0.000372	0.004208	0.010042	-0.009871	0.004775	0.016426	0.000370
pickup_longitude	-0.001076	1.000000	-0.893827	0.963554	-0.951690	-0.000330	-0.009888	0.001243	-0.003849	0.002276	-0.011781	0.132801
pickup_latitude	0.000778	-0.893827	1.000000	-0.862755	0.881747	0.000221	0.009315	0.001982	0.005016	0.001102	0.006084	-0.112894
dropoff_longitude	-0.001053	0.963554	-0.862755	1.000000	-0.977943	-0.000318	-0.009367	-0.000701	-0.005544	0.003455	-0.011090	0.120548
dropoff_latitude	0.000995	-0.951690	0.881747	-0.977943	1.000000	0.000293	0.010661	-0.002080	0.006169	-0.003322	0.007697	-0.097488
passenger_count	-0.000372	-0.000330	0.000221	-0.000318	0.000293	1.000000	0.002131	0.008767	-0.012086	-0.004151	0.002143	-0.000828
hour	0.004208	-0.009888	0.009315	-0.009367	0.010661	0.002131	1.000000	-0.002498	-0.005136	-0.087651	-0.004404	0.002790
day	0.010042	0.001243	0.001982	-0.000701	-0.002080	0.008767	-0.002498	1.000000	-0.015100	0.013308	-0.026128	0.018061
month	-0.009871	-0.003849	0.005016	-0.005544	0.006169	-0.012086	-0.005136	-0.015100	1.000000	-0.015325	-0.118048	-0.014950
weekday	0.004775	0.002276	0.001102	0.003455	-0.003322	-0.004151	-0.087651	0.013308	-0.015325	1.000000	0.006553	-0.001147
year	0.016426	-0.011781	0.006084	-0.011090	0.007697	0.002143	-0.004404	-0.026128	-0.118048	0.006553	1.000000	0.018946
distance	0.000370	0.132801	-0.112894	0.120548	-0.097488	-0.000828	0.002790	0.018061	-0.014950	-0.001147	0.018946	1.000000

The above figure shows the multicollinearity values of all the variables.

Let us analyse for continuous variables using VIF (Variance inflation Factor):

Checking VIF for continuous variables:

Below figure shows the multicollinearity representation using VIF

```
const               1.921785e+07
fare_amount         2.899409e+00
pickup_longitude    1.461893e+00
pickup_latitude     1.267572e+00
dropoff_longitude   1.324546e+00
dropoff_latitude    1.249912e+00
passenger_count     1.002682e+00
hour                1.014675e+00
day                 1.001724e+00
month               1.018722e+00
weekday             1.014930e+00
year                1.057942e+00
distance            3.143368e+00
dtype: float64
```

We have good value of VIF, and don't have multicollinearity problem, distance variable highly associated with target variable(fare_amount) and const is not part of our dataset, it is was added as it required to calculate VIF.

Before feature selection we have added one more variable which is '**distance'** in km and it calculated by the latitude and the longitude of the points in the data, also derived new variables are **Day**, **Year**, **Month**, **Dayofweek**, from the **pickup_datetime** variable.

Following is the haversine mathematical distance formula to calculate distance between two geographical points.

R = 6373.0

dlon = radians(lon2) - radians(lon1)

dlat = radians(lat2) - radians(lat1)

a = (sin(dlat/2))2 + cos(radians(lat1)) * cos(radians(lat2)) * (sin(dlon/2))**2**

c = 2 * atan2(sqrt(a), sqrt(1-a))

distance = R * c

2.1.6 Feature Scaling

Feature scaling is a method used to normalize/standardize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data pre-processing step. Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance.

If one of the features has a broad range of values, the distance will be governed by this particular feature. therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

Since most of our data is uniformly distributed, here data is almost mostly normalized based on the test case feature conditions and we are considering this data for modelling without applying standardization.

2.2 Modelling

2.2.1 Model Selection

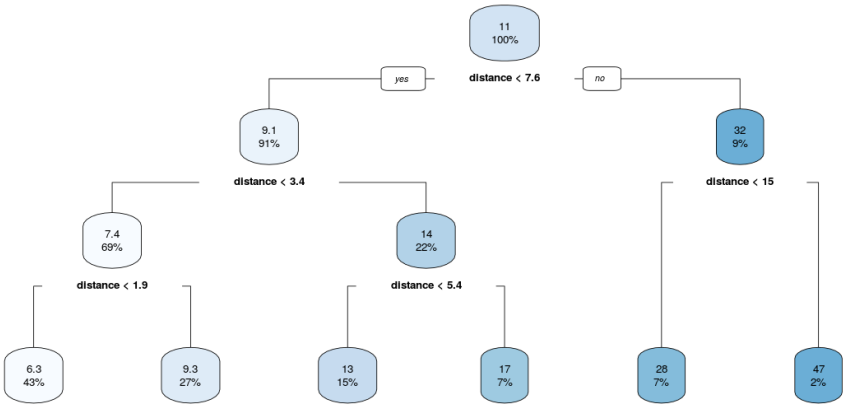
After a thorough pre-processing, we will be using some regression models on our processed data to predict the target variable. The target variable in our model is a continuous variable i.e. **fare_amount** (Cab fare), Hence the models that we choose are Linear Regression, Decision Tree and Random Forest. The error metric chosen for the given problem statement is Root Mean Squared Error(RMSE) and R^2(R-Squared).

2.2.2 Decision Tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Decision trees are used for both classification and regression problems.
A decision tree is a tree where each node represents a feature (attribute), each link (branch) represents a decision (rule) and each leaf represents an outcome (categorical or continues value). The general motive of using Decision Tree is to create a training model which can use to predict class or value of target variables by learning decision rules inferred from prior data (training data).

The RMSE values and R^2 values for the given project in R and Python are:

Decision Tree	**RMSE**	**R^2**
R	**4.47**	**0.75**
PYTHON	**5.33**	**0.65**



Using decision tree in R, we can predict the value of bike count. RMSE and R2 for this model is 5.33 and 0.65, The MAPE for this decision tree is 29.08 %. Hence the accuracy for this model is 70.92%.

2.2.3 Multiple Linear Regression

Multiple linear regression is the most common form of linear regression analysis. Multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables. The independent variables can be continuous or categorical.

Multiple Linear Regression	**RMSE**	**R^2**
R	**5.15**	**0.63**
PYTHON	**5.30**	**0.66**

The below figure shows the complete summary of the model:

```
Call:
lm(formula = fare_amount ~ ., data = train)

Residuals:
     Min       1Q   Median       3Q      Max
-148.427   -2.206   -0.669    1.226   51.105

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)       5.178e+03  2.046e+02  25.314  < 2e-16 ***
pickup_longitude  7.730e+00  1.562e+00   4.948 7.60e-07 ***
pickup_latitude  -4.820e+01  1.828e+00 -26.372  < 2e-16 ***
dropoff_longitude 2.212e+01  1.611e+00  13.731  < 2e-16 ***
dropoff_latitude -2.462e+01  1.727e+00 -14.253  < 2e-16 ***
passenger_count   6.904e-02  3.743e-02   1.844 0.065169 .
year2010          2.452e-01  1.706e-01   1.437 0.150701
year2011          4.596e-01  1.713e-01   2.683 0.007316 **
year2012          1.160e+00  1.700e-01   6.821 9.49e-12 ***
year2013          2.216e+00  1.710e-01  12.959  < 2e-16 ***
year2014          2.710e+00  1.736e-01  15.610  < 2e-16 ***
year2015          3.194e+00  2.187e-01  14.602  < 2e-16 ***
month2            3.498e-01  2.278e-01   1.535 0.124726
month3            5.748e-01  2.207e-01   2.605 0.009210 **
month4            6.779e-01  2.224e-01   3.048 0.002309 **
month5            6.652e-01  2.207e-01   3.014 0.002586 **
month6            8.197e-01  2.218e-01   3.696 0.000220 ***
month7            3.865e-01  2.368e-01   1.632 0.102685
month8            4.060e-01  2.403e-01   1.690 0.091114 .
month9            1.204e+00  2.364e-01   5.093 3.57e-07 ***
month10           1.281e+00  2.313e-01   5.537 3.15e-08 ***
month11           1.314e+00  2.342e-01   5.610 2.07e-08 ***
month12           1.136e+00  2.326e-01   4.886 1.04e-06 ***
day2             -4.007e-01  3.812e-01  -1.051 0.293248
day3             -1.599e-01  3.873e-01  -0.413 0.679674
day4             -3.377e-01  3.896e-01  -0.867 0.386117
day5              3.777e-01  3.834e-01   0.985 0.324565
day6             -2.875e-01  3.761e-01  -0.765 0.444509
day7              3.985e-01  3.746e-01   1.064 0.287398
day8              8.693e-02  3.722e-01   0.234 0.815343
day9              8.077e-02  3.742e-01   0.216 0.829107
day10            -8.431e-03  3.689e-01  -0.023 0.981765
day11            -1.712e-01  3.786e-01  -0.452 0.651112
day12            -2.857e-01  3.756e-01  -0.761 0.446905
day13            -9.569e-02  3.763e-01  -0.254 0.799266
day14             2.934e-01  3.875e-01   0.757 0.448984
day15            -2.779e-01  3.782e-01  -0.735 0.462400
day16            -4.115e-01  3.715e-01  -1.108 0.267964
day17             1.594e-01  3.787e-01   0.421 0.673859
day18            -1.388e-01  3.864e-01  -0.359 0.719425
day19             3.509e-01  3.777e-01   0.929 0.352991
day20             3.347e-01  3.771e-01   0.888 0.374756
day21             1.999e-01  3.715e-01   0.538 0.590495
```

```
day21          1.999e-01  3.715e-01   0.538 0.590495
day22          8.113e-02  3.769e-01   0.215 0.829571
day23          3.047e-01  3.807e-01   0.800 0.423458
day24          1.877e-01  3.771e-01   0.498 0.618633
day25         -6.015e-01  3.823e-01  -1.573 0.115711
day26          9.255e-03  3.886e-01   0.024 0.981002
day27          4.053e-01  3.843e-01   1.055 0.291640
day28         -2.015e-01  3.829e-01  -0.526 0.598824
day29          4.442e-01  3.993e-01   1.112 0.265987
day30         -4.149e-01  3.970e-01  -1.045 0.295963
day31          2.979e-01  4.395e-01   0.678 0.497903
dayOfWeek2     1.964e-01  1.871e-01   1.049 0.293997
dayOfWeek3     2.833e-01  1.852e-01   1.530 0.126004
dayOfWeek4     9.528e-02  1.850e-01   0.515 0.606426
dayOfWeek5     3.819e-01  1.821e-01   2.097 0.035980 *
dayOfWeek6     4.174e-01  1.823e-01   2.290 0.022028 *
dayOfWeek7    -3.041e-02  1.790e-01  -0.170 0.865069
hour1         -1.405e-01  3.730e-01  -0.377 0.706509
hour2         -4.181e-02  4.144e-01  -0.101 0.919629
hour3          7.322e-02  4.302e-01   0.170 0.864847
hour4         -7.488e-02  4.921e-01  -0.152 0.879051
hour5          7.208e-01  5.317e-01   1.356 0.175274
hour6          2.913e-01  4.098e-01   0.711 0.477298
hour7          3.542e-01  3.449e-01   1.027 0.304462
hour8          1.173e+00  3.392e-01   3.458 0.000547 ***
hour9          1.163e+00  3.309e-01   3.515 0.000442 ***
hour10         8.618e-01  3.387e-01   2.544 0.010963 *
hour11         1.468e+00  3.344e-01   4.388 1.15e-05 ***
hour12         1.502e+00  3.233e-01   4.645 3.44e-06 ***
hour13         1.359e+00  3.286e-01   4.135 3.58e-05 ***
hour14         1.390e+00  3.271e-01   4.249 2.16e-05 ***
hour15         1.436e+00  3.335e-01   4.306 1.67e-05 ***
hour16         1.908e+00  3.420e-01   5.578 2.48e-08 ***
hour17         1.250e+00  3.301e-01   3.786 0.000154 ***
hour18         1.266e+00  3.132e-01   4.041 5.34e-05 ***
hour19         2.057e-01  3.147e-01   0.654 0.513354
hour20         1.581e-01  3.136e-01   0.504 0.614276
hour21         3.619e-01  3.164e-01   1.144 0.252671
hour22         2.353e-01  3.172e-01   0.742 0.458244
hour23        -1.732e-01  3.292e-01  -0.526 0.598778
distance       1.529e+00  1.528e-02 100.121  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.265 on 12218 degrees of freedom
Multiple R-squared:  0.6411,    Adjusted R-squared:  0.6387
F-statistic: 266.2 on 82 and 12218 DF,  p-value: < 2.2e-16
```

As you can see the Adjusted R-squared value, we can explain 63.87% of the data using our multiple linear regression model, this model explains the data very well and is considered to be good.

Even after removing the non-significant variables, the accuracy, Adjusted R-squared and F-statistic do not change by much, RMSE is calculated and found to be 5.30 and accuracy is 69.39% along with MAPE is 30.61.

2.2.4 Random Forest

Random Forest is a supervised learning algorithm. Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. It can be used for both classification and regression problems. The method of combining trees is known as an ensemble method. Assembling is nothing but a combination of weak learners (individual trees) to produce a strong learner.
The number of decision trees used for prediction in the **forest** is 10.
(Observed parameter from the hyperparametertuning) Number of decision trees used for prediction in the **forest** is 800.

Random Forest	**RMSE**	**R^2**
R	**3.99**	**0.85**
PYTHON	**3.97**	**0.81**

Using Random forest for prediction analysis in this case the number of decision trees used for prediction in the forest is 10. RMSE for this model is 3.97.

Using random forest, the MAPE was found to be 22.14 %, Hence the accuracy is 77.86%

Variable Importance of random forest model

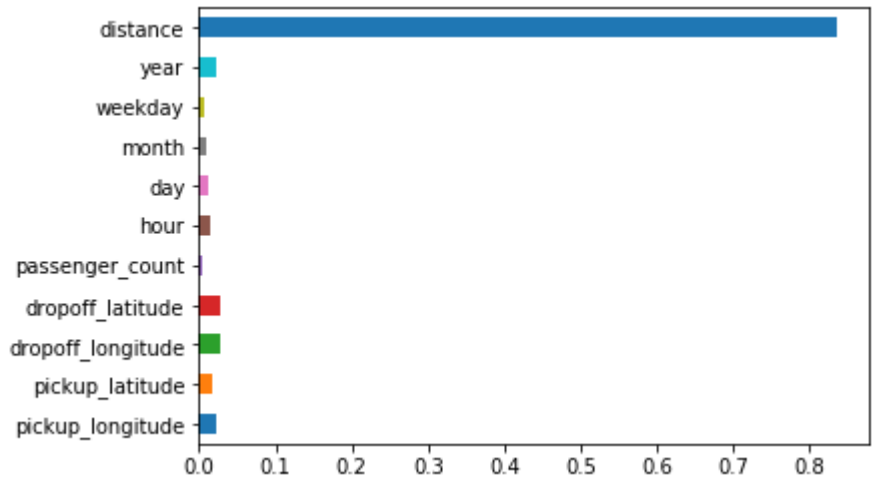


Fig 2.6: Feature Importance table

From the above plot explains the which variable carries highest information to predict the target variable, the highest information is called feature importance, from the above plot we can observe that distance and year have high feature importance.

2.2.5 HyperParameter Tuning

Now, we will tune our model i.e. Random Forest, we will tune our model for whole dataset i.e. Cab data. With the help of hyperparameter tuning we would find optimum values for parameter used in function and would increase our accuracy.

Base model performance

```
<---Model Performance--->
R-Squared Value = 0.81
RMSE = 3.97
MAPE = 22.14
Accuracy = 77.86%.
```

Hyperparameter tuned model performance

```
<---Model Performance--->
R-Squared Value = 0.83
RMSE = 3.74
MAPE = 19.98
Accuracy = 80.02%.
```

Parameters of the Hyperparameter tuned model

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=30,
           max_features='sqrt', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=2, min_samples_split=8,
           min_weight_fraction_leaf=0.0, n_estimators=800, n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

Analysis:

From above result (on tuned parameter), we have increased our model accuracy from 77.86% to 80.02%. Also we can observe that previously it was giving R^2 0.81 and now it is giving 0.83, also the RMSE also reduced from 3.97 to 3.74, On the test dataset we have slightly increased accuracy. In the parameters we can observed that n_estimators (tree numbers) changed from 10 to 800.

So, with the help of hyperparameter tuning we have increased performance of model.

Chapter 3

Conclusion

3.1 Model Evaluation

In the previous chapter we have seen the Root Mean Square Error (RMSE) and R-Squared Value of different models. Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are, 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. Whereas R-squared is a relative measure of fit, RMSE is an absolute measure of fit. As the square root of a variance, RMSE can be interpreted as the standard deviation of the unexplained variance and has the useful property of being in the same units as the response variable. Lower values of RMSE and higher value of R-Squared Value indicate better fit.

3.1.1 RMSE and R-Squared

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are, 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. Root mean square error is commonly used in climatology, forecasting, and regression analysis to verify experimental results.

$$\textbf{RMSE} = \sqrt{\overline{(\textbf{predicted values} - \textbf{observed values})^2}}$$

R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model. Whereas correlation explains the strength of the relationship between an independent and dependent variable, R-squared explains to what extent the variance of one variable explains the variance of the second variable. So, if the R2 of a model is 0.50, then approximately half of the observed variation can be explained by the model's inputs.

3.1.2 Model selection

Multiple Linear Regression: RMSE = 5.30, R^2 = 0.66

Decision Tree: RMSE = 5.33, R^2 = 0.65

Random Forest: RMSE = 3.97, R^2 = 0.81

Based on the above error metrics, Random Forest is the better model for our analysis.

Hence Random Forest is chosen as the model for Cab Fare Prediction.

Appendix A

Extra Figures

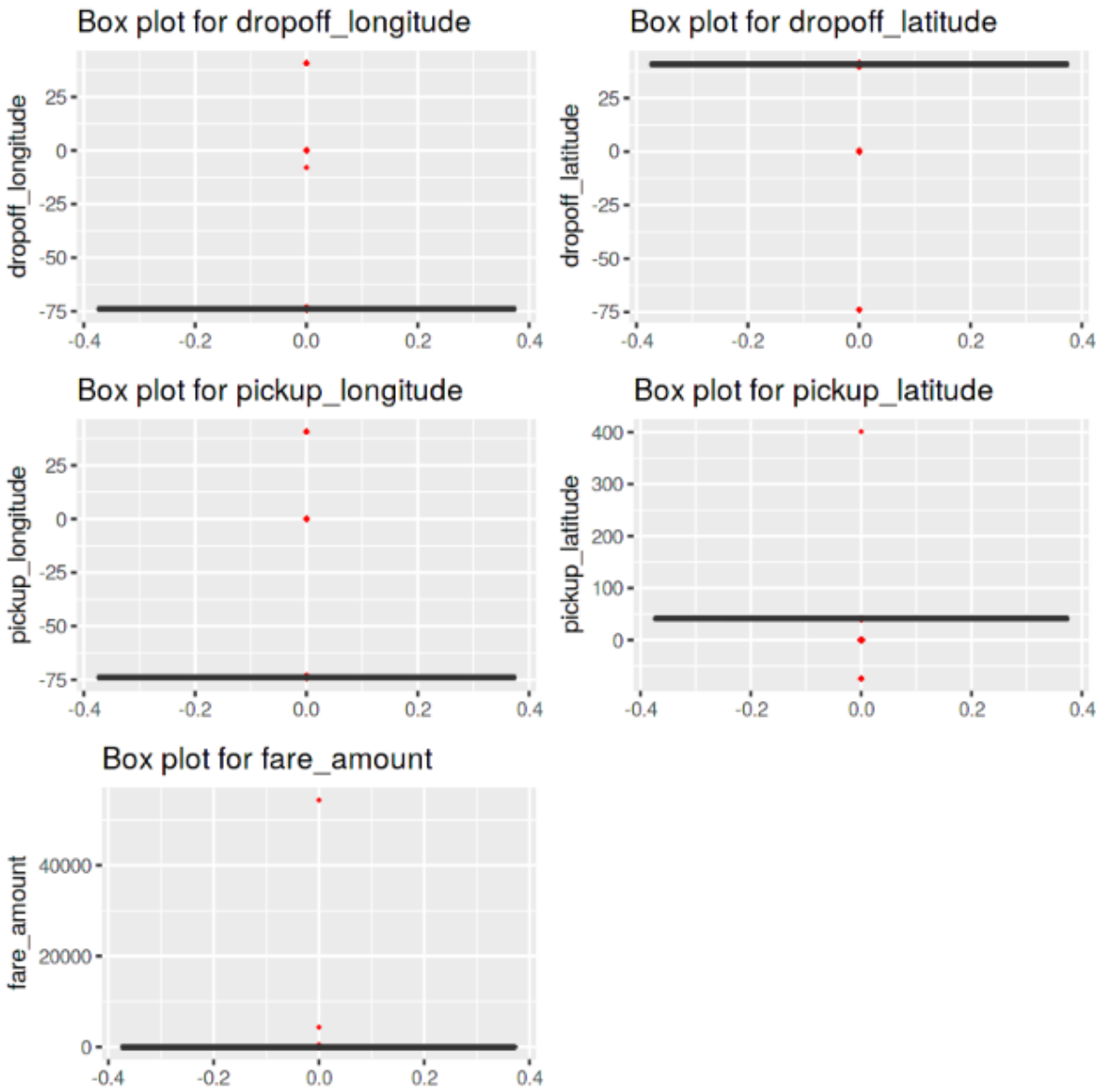
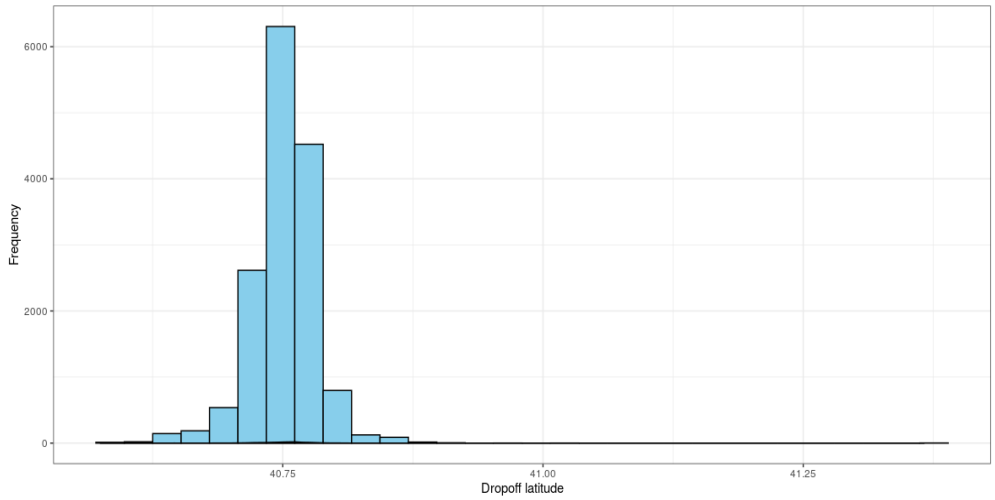
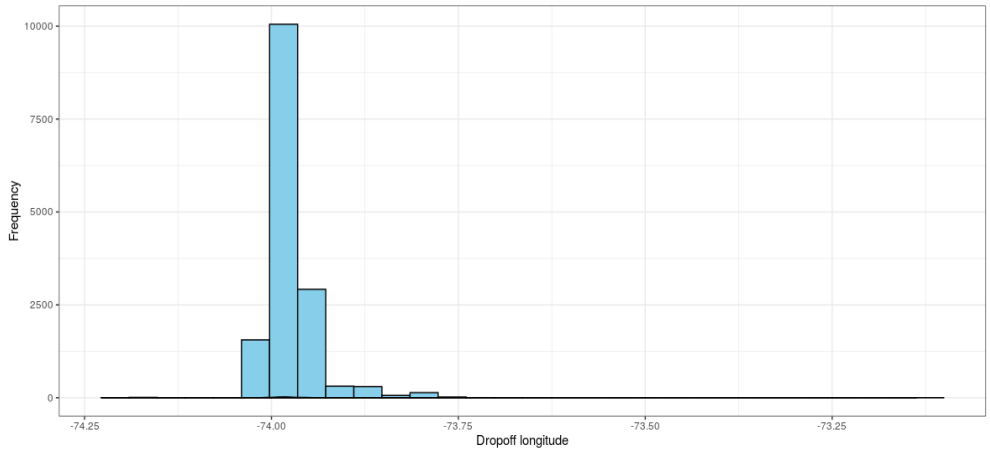
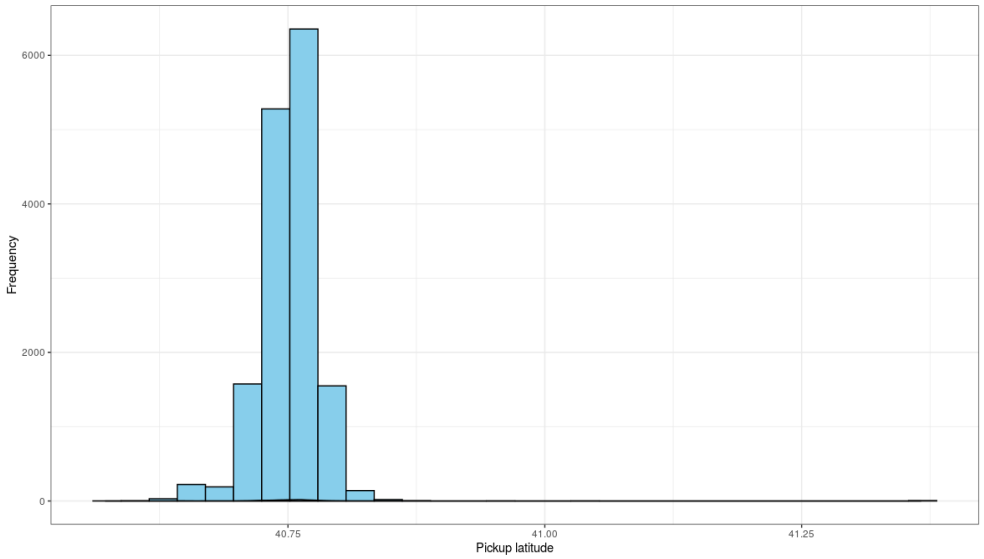


Fig 2.1: Boxplot of continuous variables with outliers


6000
4000
2000
0
Frequency
40.75
41.00
41.25
Dropoff latitude


10000
7500
5000
2500
0
Frequency
-74.25
-74.00
-73.75
-73.50
-73.25
Dropoff longitude


6000
4000
2000
0
Frequency
40.75
41.00
41.25
Pickup latitude

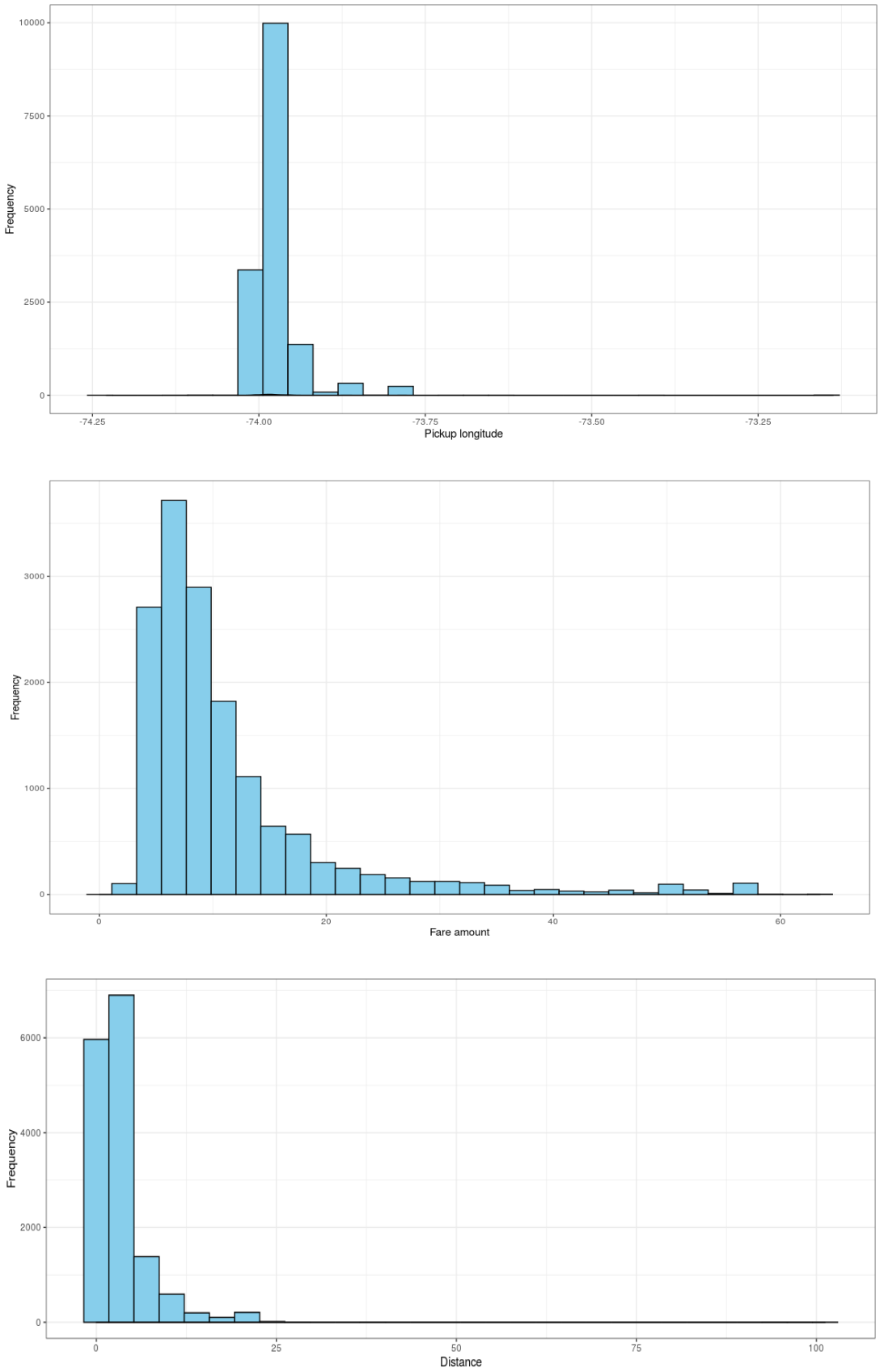
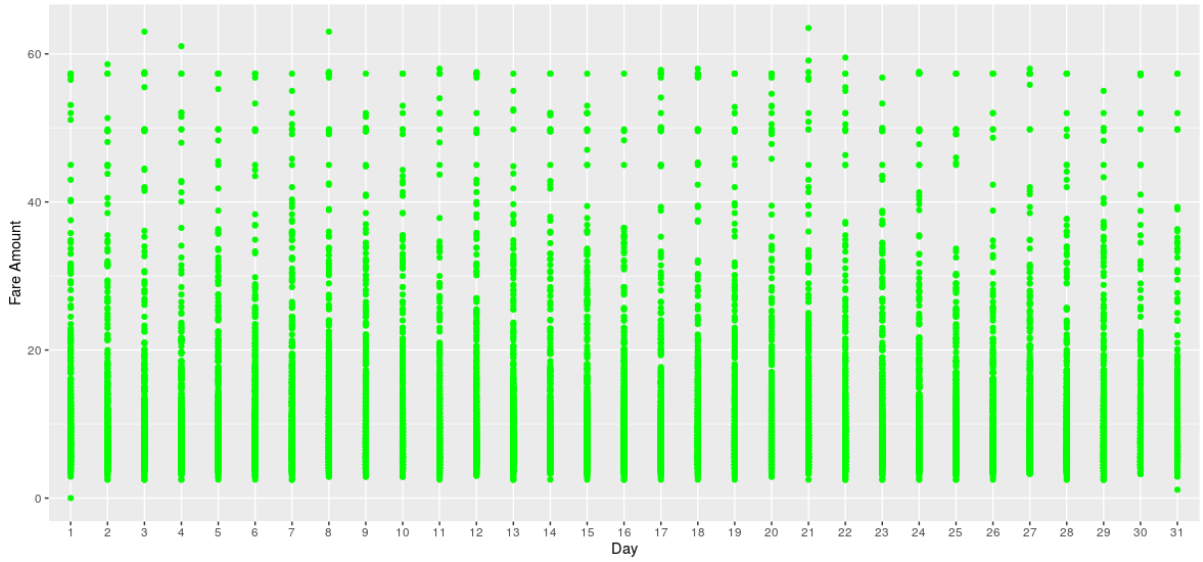
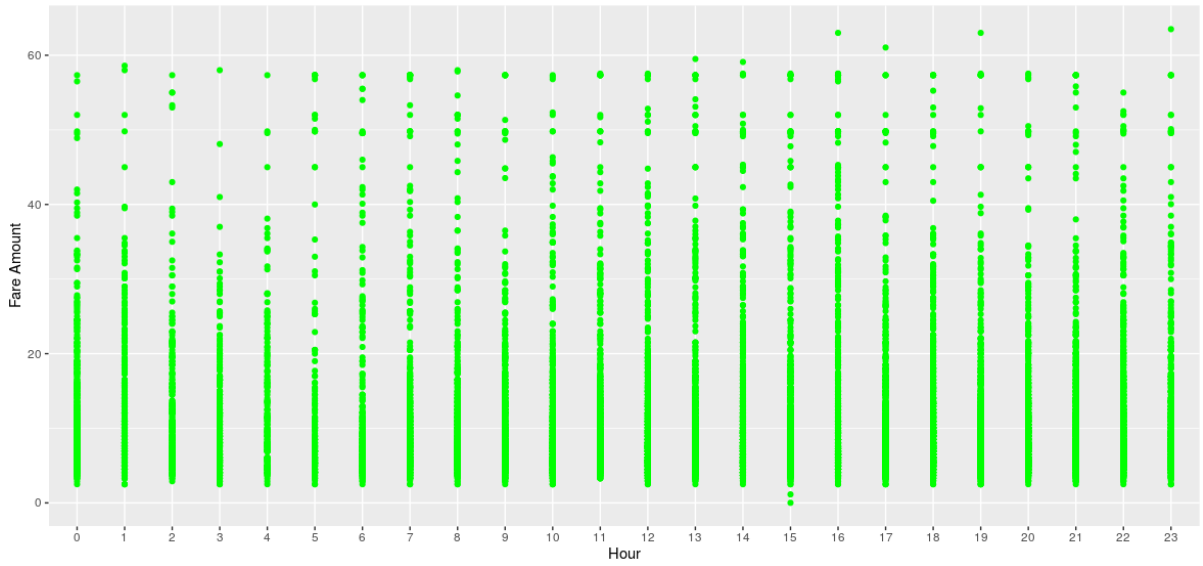
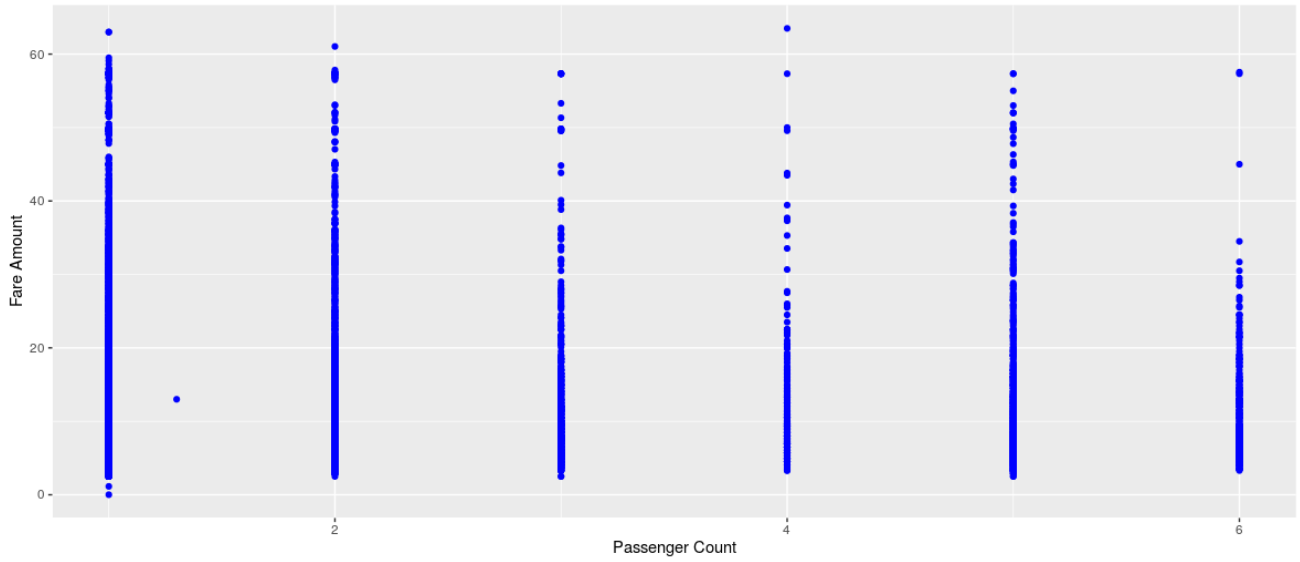


Fig 2.2: Distribution of continuous variables using histograms


Fare Amount
Day


Fare Amount
Hour


Fare Amount
Passenger Count

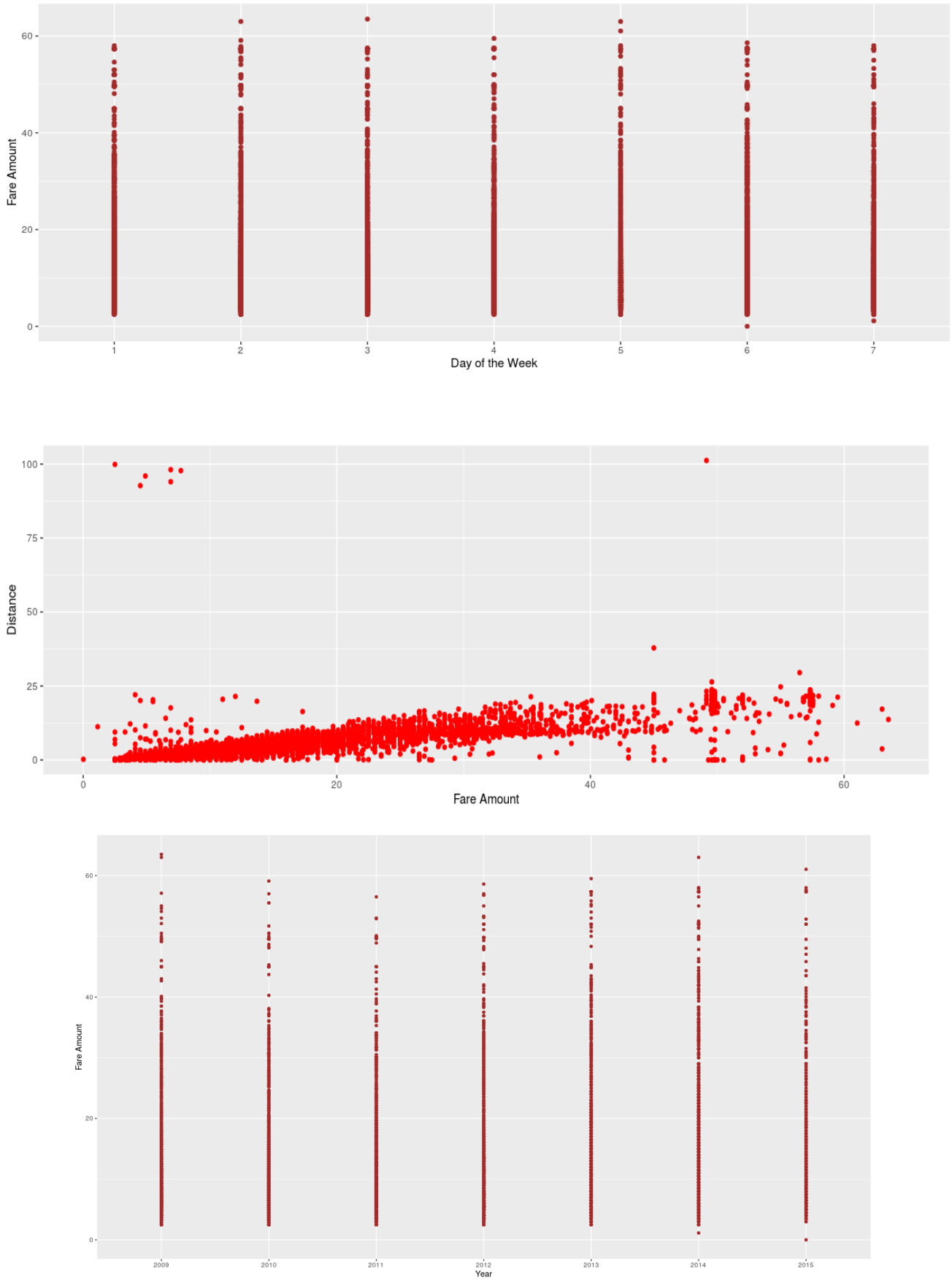


Fig 2.3: Relationship between the variables against Fare Amount

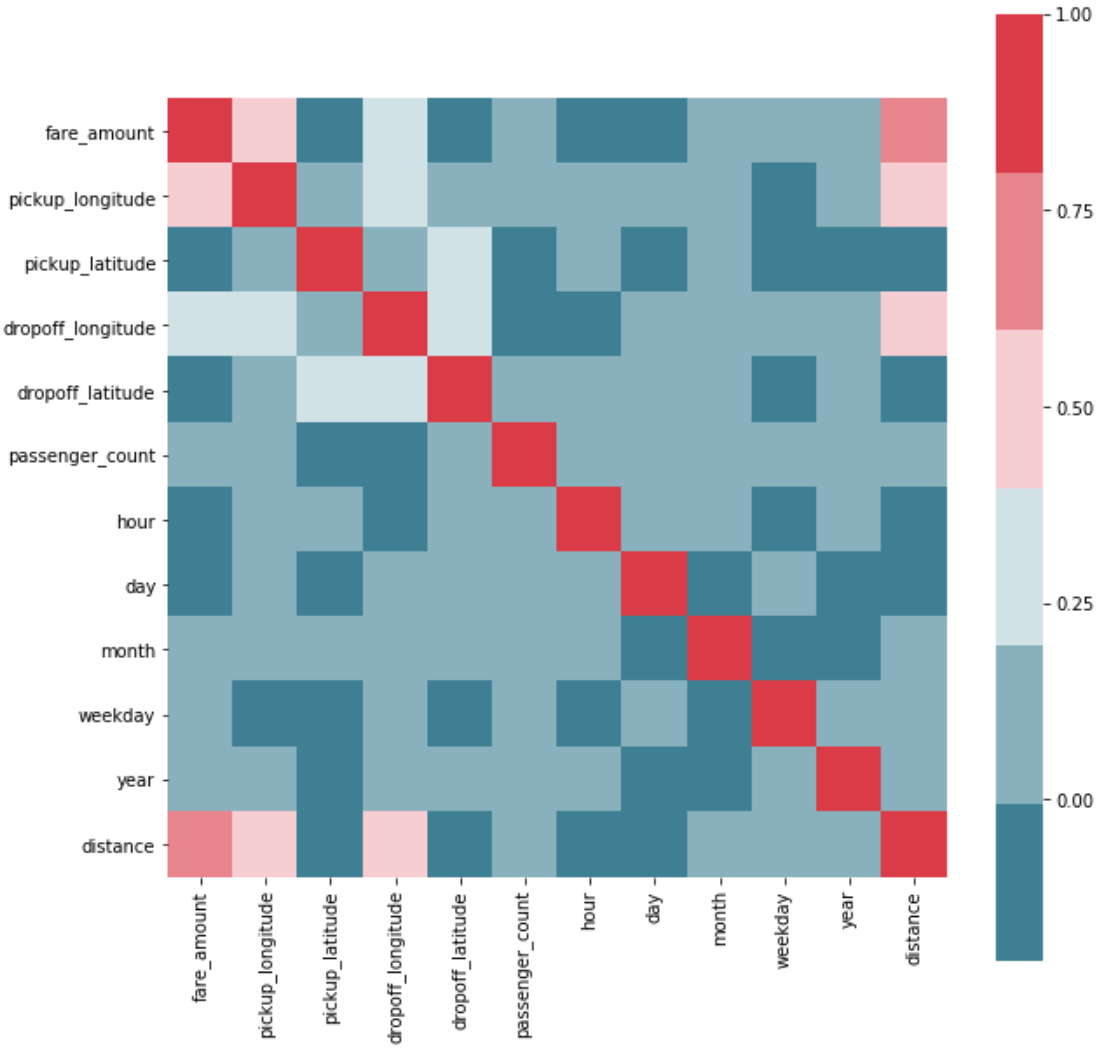


Fig 2.4: Correlation plot of Continuous variables

	pickup_datetime	Predicted_Fare
0	2015-01-27 13:08:24	10.001796
1	2015-01-27 13:08:24	9.931756
2	2011-10-08 11:53:44	4.880131
3	2012-12-01 21:12:12	8.760334
4	2012-12-01 21:12:12	15.648714
5	2012-12-01 21:12:12	10.451380
6	2011-10-06 12:10:20	5.146584
7	2011-10-06 12:10:20	48.712475
8	2011-10-06 12:10:20	11.828461
9	2014-02-18 15:22:20	6.382178

Fig 2.5: Predictions of given test data

Appendix B

R – Code

```
#Clean the environment
rm(list = ls())


#Loading Libraries
#loading libraries
library(rpart)
library(randomForest)
library(DataCombine)
library(caret)
library(dplyr)
library(tidyr)
library(ggplot2)
library(lubridate)
library(datetime)
library(tidyverse)
library(VIF)
library(purrr)
library(geosphere)
library(rlist)
library(usdm)
#Loding the dataset
train_data = read.csv('train_cab.csv')
test_data = read.csv('test.csv')


#Checking Data
str(train_data)
head(train_data,4)
```

#Shape of the data

dim(train_data)

#------EDA----------#

#Converting data type of fare amount to float

train_data$fare_amount = as.numeric(as.character(train_data$fare_amount))

#-------Missing Value Analysis----------#

missing_val = data.frame(sapply(train_data, function(x){sum(is.na(x))}))

missing_val$Variables = row.names(missing_val)

names(missing_val)[1] = "Missing_Values_Count"

missing_val = missing_val[,c(2,1)]

missing_val$Missing_percentage = (missing_val$Missing_Values_Count/nrow(train_data)) * 100

missing_val = missing_val[order(-missing_val$Missing_percentage),]

row.names(missing_val) = NULL

missing_val

#deleting the rows containing NA

train_data = train_data %>% drop_na()

#Checking the Data

dim(train_data)

str(train_data)

#---------- Feature Engineering ----------#

#Extracting Year, Month, dayOfweek, hour, day from pickup_datetime Column for Train data

```
train_data <- mutate(train_data,
                     pickup_datetime = ymd_hms(pickup_datetime),
                     year = as.factor(year(pickup_datetime)),
                     month = as.factor(month(pickup_datetime)),
                     day = as.factor(day(pickup_datetime)),
                     dayOfWeek = as.factor(wday(pickup_datetime)),
                     hour = hour(pickup_datetime),
                     hour = as.factor(hour(pickup_datetime))
)
```

#Extracting Year, Month, dayOfweek, hour, day from pickup_datetime Column for Test data

```
test_data <- mutate(test_data,
                    pickup_datetime = ymd_hms(pickup_datetime),
                    year = as.factor(year(pickup_datetime)),
                    month = as.factor(month(pickup_datetime)),
                    day = as.factor(day(pickup_datetime)),
                    dayOfWeek = as.factor(wday(pickup_datetime)),
                    hour = hour(pickup_datetime),
                    hour = as.factor(hour(pickup_datetime))
)
```

#Check the data

```
str(test_data)
head(test_data,5)

str(train_data)
```

```
head(train_data,5)

#----- Outlier Analysis ----------#

#Box plot of continuous data

continous_var =
c("dropoff_longitude","dropoff_latitude","pickup_longitude","pickup_latitude","fare_am
ount")

for (i in 1:length(continous_var))
{
  assign(paste0("gn",i), ggplot(aes_string(y = continous_var[i]), data = train_data)+
         stat_boxplot(geom = "errorbar", width = 0.5) +
         geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                      outlier.size=1, notch=FALSE) +
         theme(legend.position="bottom")+
         labs(y=continous_var[i])+
         ggtitle(paste("Box plot for",continous_var[i])))
}
gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,ncol=2)

#----------Removing Outliers based on the Test data feature conditions ----------#

#Analysing train_data for selecting feature conditions
summary(train_data)

#Analysing Test_data for extracting feature conditions
summary(test_data)

#feature 1: Finding min and max of longitude
```

```
lon_min = min(min(test_data$pickup_longitude),min(test_data$dropoff_longitude))
lon_max = max(max(test_data$pickup_longitude),max(test_data$dropoff_longitude))

#feature 2: Finding min and max of latitude
lat_min = min(min(test_data$pickup_latitude),min(test_data$dropoff_latitude))
lat_max = max(max(test_data$pickup_latitude),max(test_data$dropoff_latitude))

cat("Longitude min is : ",lon_min, "and max is: ",lon_max, "")
cat("Latitude min is : ",lat_min, "and max is: ",lat_max, "")

#feature 3: Finding min and max of fareamount and Removing -ve values from the
fare_amount variable and
#max range selected based on the data distribution we found that max is 65
fare_max = 65
fare_min = 1

#feature 4: Finding min and max of passenger count

pass_min = min(test_data$passenger_count)
pass_max = max(test_data$passenger_count)
cat("Passemger count min is : ",pass_min, "and max is: ",pass_max, "")

#Applying all the feature conditions on train data

#Taking copy of the data
new_train_data = train_data

#train_data = new_train_data

train_data <- filter(train_data,
```

```
            train_data$pickup_longitude >= lon_min &
             train_data$pickup_longitude <= lon_max &

             train_data$dropoff_longitude >= lon_min &
             train_data$dropoff_longitude <= lon_max &

             train_data$pickup_latitude >= lat_min &
             train_data$pickup_latitude <= lat_max &

             train_data$dropoff_latitude >= lat_min &
             train_data$dropoff_latitude <= lat_max &

             train_data$fare_amount >0 &
             train_data$fare_amount < 65 &

             train_data$passenger_count >=1 &
             train_data$passenger_count <= 6
)

#Checking the data
summary(train_data)
head(train_data,5)



#Deriving New variable distance for train data and test data


#Here pickup and drop locations are related to fare_amont of the data,
#So we need to find the distance using pickup and drop location coordinates by using "Haversine distance formula"
```

#Loading the libararies purrr,geosphere,rlist for calculating ditance using latitudes and logitudes,

#library(purrr)

#library(geosphere)

#library(rlist)

#Haversine distance formula for finding the distance

#Function for adding a new feature distance

```
get_geo_distance = function(long1, lat1, long2, lat2) {
 loadNamespace("purrr")
 loadNamespace("geosphere")
 longlat1 = purrr::map2(long1, lat1, function(x,y) c(x,y))
 longlat2 = purrr::map2(long2, lat2, function(x,y) c(x,y))
 distance_list = purrr::map2(longlat1, longlat2, function(x,y) geosphere::distHaversine(x, y))
 distance_m = list.extract(distance_list, position = 1)
 #if (units == "km") {
 distance = distance_m / 1000.0;
 distance
}
```

#Applying **distance formula for train data**

```
for(i in (1:nrow(train_data)))
{
 train_data$distance[i]= get_geo_distance(train_data$pickup_longitude[i],train_data$pickup_latitude[i],train_data$dropoff_longitude[i],train_data$dropoff_latitude[i])
}
```

#Applying distance formula for test data

```
for(i in (1:nrow(test_data)))
{
  test_data$distance[i]=
get_geo_distance(test_data$pickup_longitude[i],test_data$pickup_latitude[i],test_data$dropoff_longitude[i],test_data$dropoff_latitude[i])
}
```

#Removing observations those have distance of 0 Km in train data

```
train_data_d = train_data
train_data <- filter(train_data,
                     train_data$distance>0)
```

#Checking the train data

```
summary(train_data)
head(train_data,4)
```

#Checking the test data

```
summary(test_data)
head(test_data,4)
```

#Removing generated null values

```
train_data = train_data %>% drop_na()
```

#checking the train data

```
summary(train_data)
```

```
#-------Checking the Distribution of the data ----------------------#
continuous_variables =
c('year','month','fare_amount','passenger_count','pickup_longitude',
                  'pickup_latitude','dropoff_longitude','dropoff_latitude','distance')

#histogram of continuous variables
#fare amount
ggplot(train_data, aes_string(x = train_data$fare_amount)) +
  geom_histogram(fill="skyblue", colour = "black") + geom_density() +
  theme_bw() + xlab("Fare amount") + ylab("Frequency")

#pickup latitude
ggplot(train_data, aes_string(x = train_data$pickup_latitude)) +
  geom_histogram(fill="skyblue", colour = "black") + geom_density() +
  theme_bw() + xlab("Pickup latitude") + ylab("Frequency")

#pickup longitude
ggplot(train_data, aes_string(x = train_data$pickup_longitude)) +
  geom_histogram(fill="skyblue", colour = "black") + geom_density() +
  theme_bw() + xlab("Pickup longitude") + ylab("Frequency")

#dropoff latitude
ggplot(train_data, aes_string(x = train_data$dropoff_latitude)) +
  geom_histogram(fill="skyblue", colour = "black") + geom_density() +
  theme_bw() + xlab("Dropoff latitude") + ylab("Frequency")

#dropoff longitude
ggplot(train_data, aes_string(x = train_data$dropoff_longitude)) +
  geom_histogram(fill="skyblue", colour = "black") + geom_density() +
```

```
 theme_bw() + xlab("Dropoff longitude") + ylab("Frequency")
```

#distance

```
ggplot(train_data, aes_string(x = train_data$distance)) +
 geom_histogram(fill="skyblue", colour = "black") + geom_density() +
 theme_bw() + xlab("Distance") + ylab("Frequency")
```

#Scatter plot between distance and fare

```
ggplot(train_data, aes(x = fare_amount, y =distance))+geom_point(color='red')+xlab('Fare Amount')+ylab('Distance')
```

#Hypothesis Assumptions

#1 - Check the pickup date and time affect the fare or not

```
ggplot(train_data, aes(x = day, y =fare_amount))+geom_point(color='green')+xlab('Day')+ylab('Fare Amount')

ggplot(train_data, aes(x = hour, y =fare_amount))+geom_point(color='green')+xlab('Hour')+ylab('Fare Amount')
```

#2 - Number of Passengers vs Fare

```
ggplot(train_data, aes(x = passenger_count, y =fare_amount))+xlim(1,6)+geom_point(color='green')+xlab('Passenger Count')+ylab('Fare Amount')
```

#3 - Does the day of the week affect the fare?

```
ggplot(train_data, aes(x = dayOfWeek, y =fare_amount))+geom_point(color='green')+xlab('Day of the Week')+ylab('Fare Amount')
```

#-------Feature Selection/Feature Scaling----------#

#Checking Correlation

```
con= c('fare_amount', 'pickup_longitude', 'pickup_latitude',
     'dropoff_longitude', 'dropoff_latitude','distance')


#Correlation Plot
corrgram(train_data[,continuous_variables], order = F,
       upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")

#Removing pickupdate variable as it is no longer required

#Checking multicollinearity using VIF
con= c('fare_amount', 'pickup_longitude', 'pickup_latitude',
     'dropoff_longitude', 'dropoff_latitude','distance')

df_vif = train_data[,con]
vifcor(df_vif)

#Selected variables for Model Building
continuous_variables

#---------- Model Development ----------#
#Divide data into train and test using stratified sampling method
set.seed(123)
train.index = sample(1:nrow(train_data), 0.8 * nrow(train_data))
train = train_data[ train.index,-2]
test  = train_data[-train.index,-2]
```

#------Linear Regression----------#

#Building the Model

LR = lm(fare_amount ~ ., data = train)

#predicting for test data

y_pred = predict(LR,test[,names(test) != "fare_amount"])

For testing data

print(postResample(pred = y_pred, obs =test$fare_amount))

RMSE Rsquared MAE

5.1525245 0.7325277 2.8235090

#------- Decision tree---------#

#Develop Model on training data

DT = rpart(fare_amount ~., data = train, method = "anova")

#predicting for test data

y_pred = predict(DT,test[,names(test) != "fare_amount"])

For testing data

print(postResample(pred = y_pred, obs =test$fare_amount))

RMSE Rsquared MAE

4.4755224 0.7938271 2.5698910

```
#---------Random Forest----------#
#Developing model on train data
RF = randomForest(fare_amount~., data = train,ntree=200)

#predicting for test data
y_pred = predict(RF,test[,names(test) != "fare_amount"])

# For testing data
print(postResample(pred = y_pred, obs = test$fare_amount))

#   RMSE      Rsquared      MAE
# 3.9960223 0.8590887   2.2500168

#-------HyperParameter Tuning for Random Forest Using GridSearchCv-------

# Tuning Random Forest
control <- trainControl(method = 'repeatedcv',
                        number = 10,
                        repeats = 3,
                        search = 'grid')
#create tunegrid
tunegrid <- expand.grid(.mtry = c(sqrt(ncol(train))))
modellist <- list()

#train with different ntree parameters
for (ntree in c(300,500,600,800)){
  set.seed(123)
```

```
 fit <- train(fare_amount~.,
              data = train,
              method = 'rf',
              metric = 'rmse',
              tuneGrid = tunegrid,
              trControl = control,
              ntree = ntree)
 key <- toString(ntree)
 modellist[[key]] <- fit
}

#Compare results
results <- resamples(modellist)
summary(results)

#Modelling with Hyperparameters
#---------Random Forest----------#
#Developing model on train data
RF = randomForest(fare_amount~., data = train,ntree=800)

#predicting for test data
y_pred = predict(RF,test_data[,names(test_data) != "pickup_datetime"])

# Sample predicted data
print(y_pred)
```

Python – Code

 # Importing packages for the model development and data processing

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

#Loading the dataset

```
train_data = pd.read_csv("train_cab.csv", sep=",")
test_data  = pd.read_csv("test.csv", sep=",")
```

#Missing Value Analysis

```
missing_val = pd.DataFrame(train_data.isnull().sum())
missing_val = missing_val.reset_index()
missing_val = missing_val.rename(columns={'index':'variables',0:'Missing_values'})
missing_val['Missing_Value_Percentage'] = (missing_val.Missing_values/len(train_data))*100
missing_val =
missing_val.sort_values('Missing_Value_Percentage',ascending=False).reset_index(drop=True)
missing_val
```

#Dropping Missing values(NA values) very few viables have missing values and its better remove them.

```
train_data.drop(train_data[train_data.fare_amount.isnull()==True].index,axis=0,inplace=True)
train_data.drop(train_data[train_data.passenger_count.isnull()==True].index,axis=0,inplace=True)
```

#Rechecking Missing Value

```
missing_val = pd.DataFrame(train_data.isnull().sum())
missing_val = missing_val.reset_index()
missing_val = missing_val.rename(columns={'index':'variables',0:'Missing_values'})
missing_val
```

```
#Checking the Data
train_data.head()
#Checking the datatypes
train_data.dtypes

#Reordering incurrect datatypes of Variables
train_data['fare_amount'] = pd.to_numeric(train_data['fare_amount'],errors='coerce')
train_data['pickup_datetime'] =
pd.to_datetime(train_data['pickup_datetime'],infer_datetime_format=True,errors='coerce')
train_data['passenger_count'] = train_data['passenger_count'].astype('int')
train_data.dtypes
#Feature Engineering
from math import radians, cos, sin, asin, sqrt
def distance(pickup_lat, pickup_lon, dropoff_lat, dropoff_lon):
    """
    Return distance along great radius between pickup and dropoff coordinates.
    """
    #Define earth radius (km)
    R_earth = 6371

    #Convert degrees to radians
    pickup_lat, pickup_lon, dropoff_lat, dropoff_lon = map(np.radians,
                                             [pickup_lat, pickup_lon,
                                              dropoff_lat, dropoff_lon])
    #Compute distances along lat, lon dimensions
    dlat = dropoff_lat - pickup_lat
    dlon = dropoff_lon - pickup_lon
```

```
    #Compute haversine distance
    a = np.sin(dlat/2.0)**2 + np.cos(pickup_lat) * np.cos(dropoff_lat) * np.sin(dlon/2.0)**2

    return 2 * R_earth * np.arcsin(np.sqrt(a))

def date_time_info(data):
    data['pickup_datetime'] = pd.to_datetime(data['pickup_datetime'], format="%Y-%m-%d
%H:%M:%S UTC")

    data['hour'] = data['pickup_datetime'].dt.hour
    data['day']  = data['pickup_datetime'].dt.day
    data['month'] = data['pickup_datetime'].dt.month
    data['weekday'] = data['pickup_datetime'].dt.weekday
    data['year']    = data['pickup_datetime'].dt.year

    return data

#Applying on train_data
train_data = date_time_info(train_data)
train_data['distance'] = distance(train_data['pickup_latitude'],
                                  train_data['pickup_longitude'],
                                  train_data['dropoff_latitude'] ,
                                  train_data['dropoff_longitude'])
#Preprocessing on test data
#Applying distance and date_time_info function on test_data
test_data = date_time_info(test_data)
test_data['distance'] = distance(test_data['pickup_latitude'],
                                  test_data['pickup_longitude'],
                                  test_data['dropoff_latitude'] ,
                                  test_data['dropoff_longitude'])
```

```
test_key = pd.DataFrame({'key_date':test_data['pickup_datetime']})
test_data = test_data.drop(columns=['pickup_datetime'],axis=1)

train_data.head()
#weekday starts from 0 to 6
train_data.describe()

#Checking data distribution Before Outlier Analysis
continuous_variables = ['year','month','fare_amount','passenger_count','pickup_longitude',
                        'pickup_latitude','dropoff_longitude','dropoff_latitude','distance']
for i in continuous_variables:
    plt.hist(train_data[i],bins=18)
    plt.title("Checking Distribution for Variable "+str(i))
    plt.ylabel("Density")
    plt.xlabel(i)
    plt.show()
#Checking data types
train_data.dtypes
#Outlier Analysis
#Outlier Visualizations
col = ['fare_amount', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude']
for i in col:
    sns.boxplot(y=train_data[i])
    fig=plt.gcf()
    fig.set_size_inches(5,5)
    plt.show()
#Fare_amount data distribution by using Scatter plot for all the observations
plt.scatter(x=train_data.fare_amount,y=train_data.index)
plt.ylabel('Index')
plt.xlabel('fare_amount')
```

```
plt.show()
```

#Fare_amount data distribution by using Scatter plot for selected observations (of x lim range from 1 to 70)

#because from 70 onwards all observations are extreme outliers.

```
plt.scatter(x=train_data.fare_amount,y=train_data.index)
plt.ylabel('Index')
plt.xlim(1,70)
plt.xlabel('fare_amount')
plt.show()
```

#passenger_count data distribution by using Scatter plot for all the observations,

#because from 70 onwards all observations are extreme outliers.

```
plt.scatter(x=train_data.passenger_count,y=train_data.index)
plt.ylabel('Index')
plt.xlabel('passenger_count')
plt.show()
```

#passenger_count data distribution by using Scatter plot for sected observations (x lim range #from 1 to 10)

#because from 70 onwards all observations are extreme outliers.

```
plt.scatter(x=train_data.passenger_count,y=train_data.index)
plt.ylabel('Index')
plt.xlim(1,10)
plt.xlabel('passenger_count')
plt.show()
```

#Scatter plot for distance variable for all the observations

```
plt.scatter(x=train_data.distance,y=train_data.index)
plt.ylabel('Index')
#plt.xlim(1,10)
plt.xlabel('ditsance')
plt.show()
```

#Scatter plot for distance variable for selected index range of observations

```
plt.scatter(x=train_data.distance,y=train_data.index)
plt.ylabel('Index')
plt.xlim(1,30)
plt.xlabel('ditsance')
plt.show()
```

Manualy removing outliers based on the test data feature conditions

#Finding the longitude min and max of test data

```
lon_min=min(test_data.pickup_longitude.min(),test_data.dropoff_longitude.min())
lon_max=max(test_data.pickup_longitude.max(),test_data.dropoff_longitude.max())
print(lon_min,',',lon_max)
```

#Finding the latitude min and max of test data

```
lat_min=min(test_data.pickup_latitude.min(),test_data.dropoff_latitude.min())
lat_max=max(test_data.pickup_latitude.max(),test_data.dropoff_latitude.max())
print(lat_min,',',lat_max)
```

```
train_data.describe()
```

#1 - Removing -ve values from the fare_amount variable

```
train_data_new = train_data
train_data=train_data.drop(train_data[(train_data.fare_amount<=0) |
(train_data.fare_amount>=65)].index,axis=0)
```

#2 - Removing null values from passenger count

```
#From the the test data passenger count lies between min is 1 and max is 6
train_data=train_data.drop(train_data[(train_data.passenger_count<=0) |
(train_data.passenger_count>6)].index,axis=0)
```

#4 - Removing pickup_latitude,dropoff_latitude, pickup_longitude, and dropoff_longitude

```
train_data=train_data.drop(train_data[(train_data.pickup_latitude <lat_min)|
(train_data.pickup_latitude >lat_max)].index,axis=0)
```

```
train_data=train_data.drop(train_data[(train_data.dropoff_latitude <lat_min) |
(train_data.dropoff_latitude >lat_max)].index,axis=0)

train_data=train_data.drop(train_data[(train_data.pickup_longitude <lon_min) |
(train_data.pickup_longitude >lon_max)].index,axis=0)

train_data=train_data.drop(train_data[(train_data.dropoff_longitude <lon_min) |
(train_data.dropoff_longitude >lon_max)].index,axis=0)
```

#4 - Removing Outliers in the Distance variable outliers

```
train_data=train_data.drop(train_data[(train_data.distance <=0)].index,axis=0)
```

#checking Null values

```
train_data.isna().sum()

train_data.dropna(axis=0,inplace=True)

train_data.isnull().sum()

train_data.describe()
```

#Checking data distribution after Outlier Analysis

```
continuous_variables = ['year','month','fare_amount','passenger_count','pickup_longitude',
                        'pickup_latitude','dropoff_longitude','dropoff_latitude','distance']
for i in continuous_variables:
   plt.hist(train_data[i],bins=18)
   plt.title("Checking Distribution for Variable "+str(i))
   plt.ylabel("Density")
   plt.xlabel(i)
   plt.show()
```

#Plot for fare_amount variation across distance

```
plt.scatter(y=train_data['distance'],x=train_data['fare_amount'])

plt.xlabel('fare')

plt.ylabel('distance')

plt.show()

sns.countplot(train_data['passenger_count'])
```

#Asumptions for Hypothesis

1)Check the pickup date and time affect the fare or not

```
plt.figure(figsize=(15,7))
plt.scatter(x=train_data['day'], y=train_data['fare_amount'], s=1.5)
plt.xlabel('Day')
plt.ylabel('Fare')
plt.show()
plt.figure(figsize=(15,7))
plt.scatter(x=train_data['hour'], y=train_data['fare_amount'], s=1.5)
plt.xlabel('Hour')
plt.ylabel('Fare')
plt.show()
```

#2 - Number of Passengers vs Fare

```
plt.figure(figsize=(15,7))
plt.hist(train_data['passenger_count'], bins=15)
plt.xlabel('No. of Passengers')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(15,7))
plt.scatter(x=train_data['passenger_count'], y=train_data['fare_amount'], s=1.5)
plt.xlabel('No. of Passengers')
plt.ylabel('Fare')
plt.show()
```

#Does the day of the week affect the fare?

```
plt.figure(figsize=(15,7))
plt.hist(train_data['weekday'], bins=100)
plt.xlabel('Day of Week')
plt.ylabel('Frequency')
plt.show()
```

```
#Feature Scaling

#Normality check

#%matplotlib inline

plt.hist(train_data['fare_amount'], bins='auto')

#train_data = train_data_df.copy()

#correlation between numerical variables

num = pd.DataFrame(train_data.select_dtypes(include=np.number))

cor = num.corr()

cor

#Coorelation Plot to check the Coorelation

continuous_variables =['fare_amount', 'pickup_datetime', 'pickup_longitude', 'pickup_latitude',

    'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'hour',

    'day', 'month', 'weekday', 'year', 'distance']


df_cor = train_data.loc[:,continuous_variables]

f, ax = plt.subplots(figsize=(10,10))


#Generate correlation matrix

cor_mat = df_cor.corr()

#Plot using seaborn library

sns.heatmap(cor_mat, mask=np.zeros_like(cor_mat, dtype=np.bool),
cmap=sns.diverging_palette(220, 10, as_cmap=False),

        square=True, ax=ax)

plt.plot()


#VIF to check the Correlation

#pick_up date is correlated to its extracted columns i.e day, year, month, weekday

from statsmodels.stats.outliers_influence import variance_inflation_factor

from statsmodels.tools.tools import add_constant

X = add_constant(num)

pd.Series([variance_inflation_factor(X.values, i) for i in range(X.shape[1])], index=X.columns)
```

```
#Feature Selection
#Removing variable 'Pickup datetime' beacause day,year,month carries all the information  from it
del train_data['pickup_datetime']

#Selected variables for model building
train_data.columns

"""
Index(['fare_amount', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count', 'hour',
       'day', 'month', 'weekday', 'year', 'distance'],
      dtype='object')
"""
#Model Building for train Data
#Splitting data into test and train
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

#-> For Train data -> train_data

y = train_data['fare_amount']
X = train_data.drop(columns=['fare_amount'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

def evaluate(model, test_features, test_actual):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_actual)
    rmse = np.sqrt(mean_squared_error(test_actual,predictions))
    mape = 100 * np.mean(errors / test_actual)
    accuracy = 100 - mape
    rsquared = r2_score(test_actual, predictions)
```

```
    df_pred = pd.DataFrame({'actual':test_actual,'predicted':predictions})
    print('<---Model Performance--->')
    print('R-Squared Value = {:0.2f}'.format(rsquared))
    print('RMSE = {:0.2f}'.format(rmse))
    print('MAPE = {:0.2f}'.format(mape))
    print('Accuracy = {:0.2f}%.'.format(accuracy))
    return rmse
```

#Linear regression

```
from sklearn.linear_model import LinearRegression
model_lr = LinearRegression().fit(X_train,y_train)
```

#predicting and testing on train data

```
evaluate(model_lr, X_test, y_test)
```

#Decision Tree

```
from sklearn.tree import DecisionTreeRegressor
model_dt = DecisionTreeRegressor(random_state = 123).fit(X_train,y_train)
```

#predicting and testing on train data

```
evaluate(model_dt, X_test, y_test)
```

#Random Forest

```
from sklearn.ensemble import RandomForestRegressor
model_rf = RandomForestRegressor(n_estimators=500,random_state=123).fit(X_train,y_train)
```

#predicting and testing on train data

```
evaluate(model_rf, X_test, y_test)
```

#Parameters of base model

```
model_rf.get_params()
```

```
#Printing Feature importance of the model
feat_importances = pd.Series(model_rf.feature_importances_, index=X_train.columns)
feat_importances.plot(kind='barh')

#HyperParameter Tuning
#Hyperparameter tuning using GridSearchCV
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
   'bootstrap': [True],
   'max_depth': [20, 25, 30],
   'max_features': ['auto', 'sqrt'],
   'min_samples_leaf': [2,3],
   'min_samples_split': [8, 10, 12],
   'n_estimators': [300,500,800]
}
# Create a base model
rf = RandomForestRegressor()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2)
grid_search.fit(X_train,y_train)
grid_search.best_params_
best_grid = grid_search.best_estimator_

#Applying gridsearchcsv to test data
grid_accuracy = evaluate(best_grid,X_test, y_test)
#Getting the best Parameters
#grid_search.best_params_
#or
grid_search.best_estimator_
```

#Printing Feature importance by visualizations

```
feat_importances_hyp = pd.Series(grid_search.best_estimator_.feature_importances_,
index=X_train.columns)

feat_importances_hyp.plot(kind='barh')
```

#Applying hyperparameters tuned base model on test data

#Building Random Forest model with hypertuned parameters

```
from sklearn.ensemble import RandomForestRegressor

model_rf = RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=25,

        max_features='sqrt', max_leaf_nodes=None,

        min_impurity_decrease=0.0, min_impurity_split=None,

        min_samples_leaf=2, min_samples_split=8,

        min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=None,

        oob_score=False, random_state=None, verbose=0, warm_start=False).fit(X_train,y_train)
```

#predicting and testing on train data

```
evaluate(model_rf, X_test, y_test)
```

#Prediction on given test data

#Checking the given test data

```
test_data.describe()


predictions = model_rf.predict(test_data)

predicted_test =
pd.DataFrame({'pickup_datetime':test_key['key_date'],'Predicted_Fare':predictions})

predicted_test.head(10)
```