CAB FARE PREDICTION

- Poorna Sai Manikanta Chimirala
August -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 Problem Description

Number of attributes:-

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.

1.3 Data

The data is a Time-Series data but instead we will approach it as Regression Problem.Our task is to build a regression model which will predict the fare of the cab facility based on the customer attributes and all of the information during the journey and general information available to the company about them.

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
4.5	2009-06-15 17:26:21	-73.844311	40.721319	-73.841610	40.712278	1
16.9	2010-01-05 16:52:16	-74.016048	40.711303	-73.979268	40.782004	1
5.7	2011-08-18 00:35:00	-73.982738	40.761270	-73.991242	40.750562	2
7.7	2012-04-21 04:30:42	-73.987130	40.733143	-73.991567	40.758092	1
5.3	2010-03-09 07:51:00	-73.968095	40.768008	-73.956655	40.783762	1

1.4 Performance Metric

RMSE : 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. Also, Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors.

So, RMSE becomes more useful when large errors are particularly undesirable. So, Root Mean Square value seems like a perfect choice for our problem at hand.

Chapter 2

Methodology

2.1.1 Data Preparation and Cleaning

2.1.1.1 Missing Value Analysis

One of the most common problems I have faced in Data Cleaning/Exploratory Data Analysis is handling the missing values. Firstly, there is no good way to deal with missing data. But still missing value analysis helps address several concerns caused by incomplete data. If cases with missing values are systematically different from cases without missing values, the results can be misleading. Also, missing data may reduce the precision of calculated statistics because there is less information than originally planned.

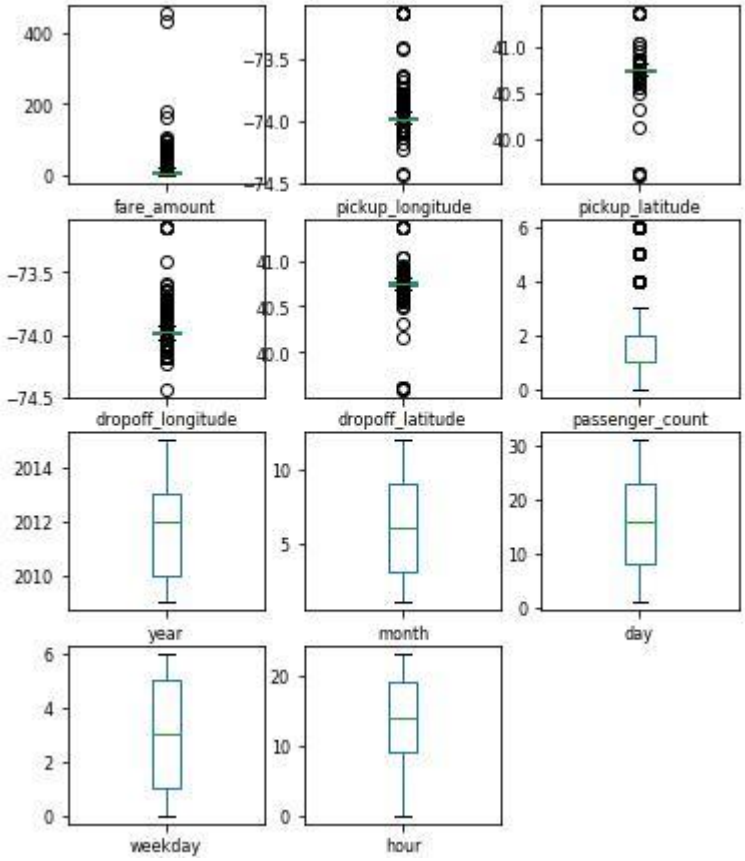
The missing value percentage in training data :

	Total	Percent
passenger_count	55	0.342317
fare_amount	25	0.155598
dropoff_latitude	0	0.000000
dropoff_longitude	0	0.000000
pickup_latitude	0	0.000000
pickup_longitude	0	0.000000
pickup_datetime	0	0.000000

Another concern is that the assumptions behind many statistical procedures are based on complete cases, and missing values can complicate the theory required. So, In our data, there are plenty of missing values available in different variables. So, after computing the percentage of missing data that is available to us in the dataset, it accounts to around 1% of the data. It is also important to note that, the missing value has been calculated after removing the missing values within the target variable. We impute the missing values in other columns using Mean imputation, because that fits the best after trying various other imputation techniques.

2.1.1.2 Outliers Analysis

In statistics, an outlier is an observation point that is distant from other observations.In layman terms, we can say that an outlier is something which is separated/different from the crowd. Also, Outlier analysis is very important because they affect the mean and median which in turn affects the error (absolute and mean) in any data set. When we plot the error we might get big deviations if outliers are in the data set. In Box plots analysis of individual features, we can clearly observe from these boxplots that, not every feature contains outliers and many of them even have very few outliers.



Also, given the constraint that, we have only 16k data-points and after removing the outliers, the data gets decreased by almost 15%. So, dropping the outliers is probably not the best idea. Instead we will try to visualise and find out the outliers using box plots and will fill them with NA, that means we have created ‘missing values’ in place of outliers within the data. Now, we can treat these outliers like missing values and impute them using standard imputation techniques. In our case, we use Mean imputation to impute these missing values.

2.1.1.3 Feature Scaling

Normalization rescales the values into a range of [0,1]. This might be useful in some cases where all parameters need to have the same positive scale. However, the outliers from the data set are lost.

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

In Linear Algebra, Normalization seems to refer to the dividing of a vector by its length.

2.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the first step in our data analysis process. We do this by taking a broad look at patterns, trends, outliers, unexpected results and so on in our existing data, using visual and quantitative methods to get a sense of the story this tells. To start with this process, we will first have a look at univariate analysis like plotting Box plot and whiskers for individual features, Histogram plots, Bar plots and Kernel Density Estimation for the same for the same.

Description of Datasets:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16067.000000	16067.000000	16067.000000	16067.000000	16012.000000
mean	-72.462787	39.914725	-72.462328	39.897906	2.625070
std	10.578384	6.826587	10.575062	6.187087	60.844122
min	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	-73.992156	40.734927	-73.991182	40.734651	1.000000
50%	-73.981698	40.752603	-73.980172	40.753567	1.000000
75%	-73.966838	40.767381	-73.963643	40.768013	2.000000
max	40.766125	401.083332	40.802437	41.366138	5345.000000

```
train_cab['fare_amount'].describe()
```

```
count     16043
unique      468
top         6.5
freq        759
Name: fare_amount, dtype: object
```

The columns 'fare_amount' and 'passenger_count' are having negative (-ve) values which need to be removed. Also Latitude and longitude values need to be in proper range.

How in a cab more than 6 passengers, can sit ?? Even 6 passengers are not possible. We will filter these excessive values in Outliers Analysis.

2.2.1 Data Visualisation

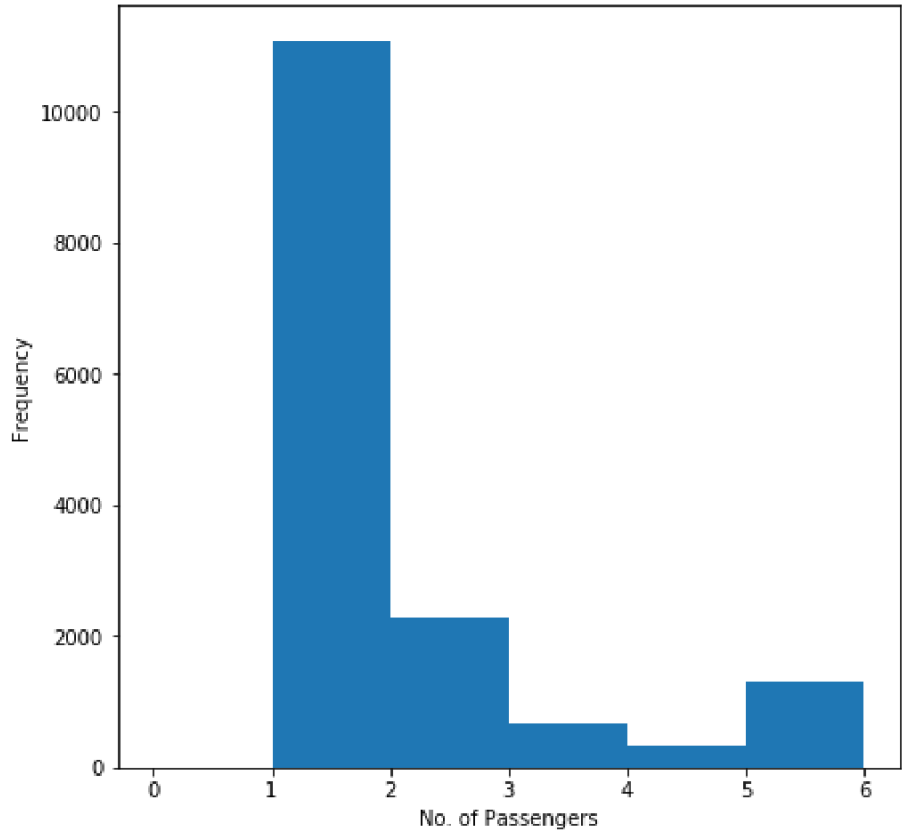
Data visualisation helps us to get better insights of the data. By visualising data, we can identify areas that need attention or improvement and also clarifies which factors influence fare of the cab and how the resources

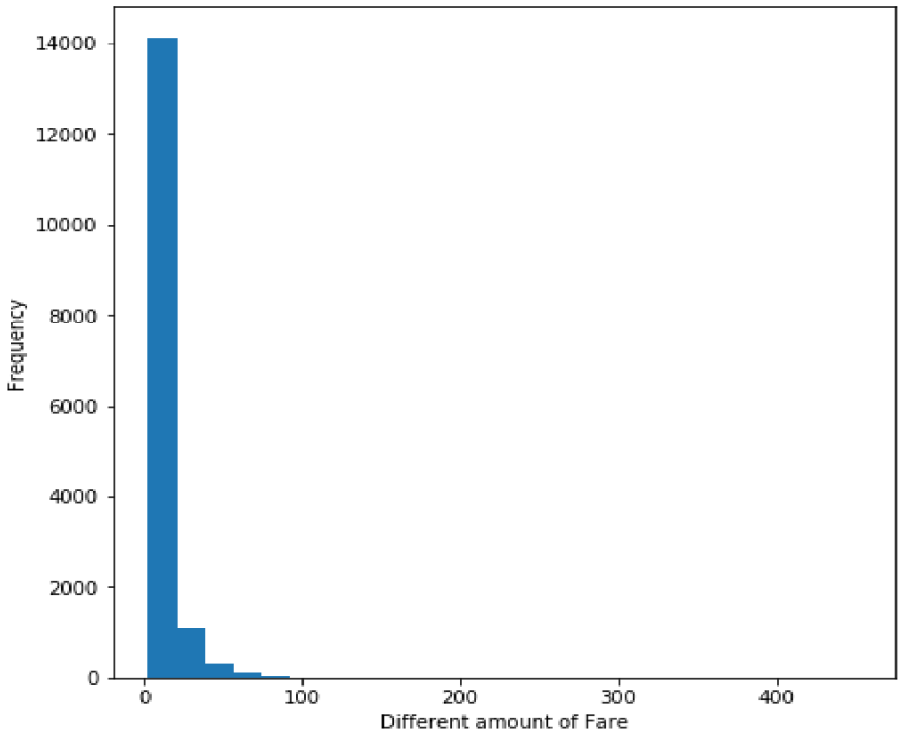
are used to determine it.

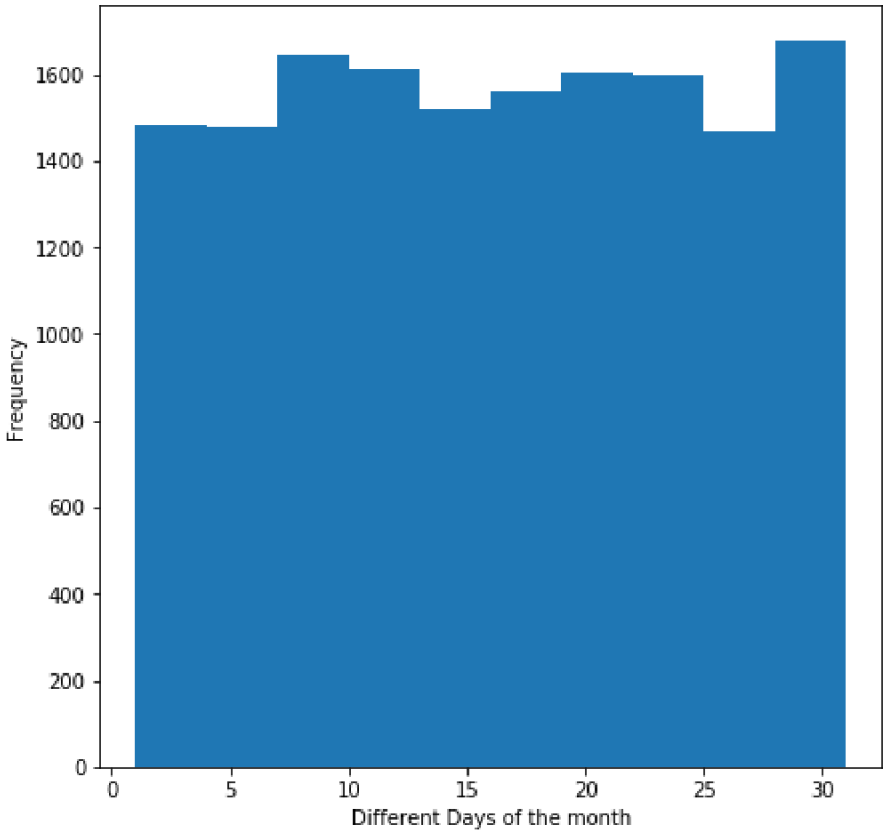
2.2.1.1 Univariate Analysis

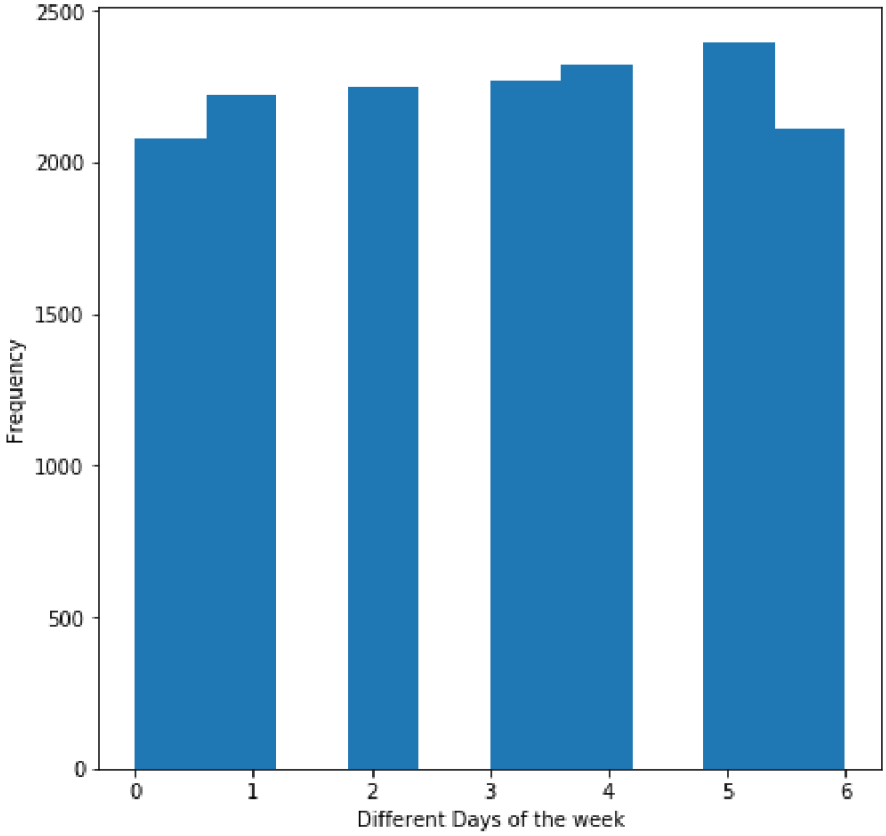
Univariate analysis is the simplest form of data analysis where the data being analysed contains only one variable. Since it's a single variable it doesn’t deal with causes or relationships. The main purpose of univariate analysis is to describe the data and find patterns that exist within it. So, Lets have a look at histogram plot, to identify the characteristic of the features and the data.

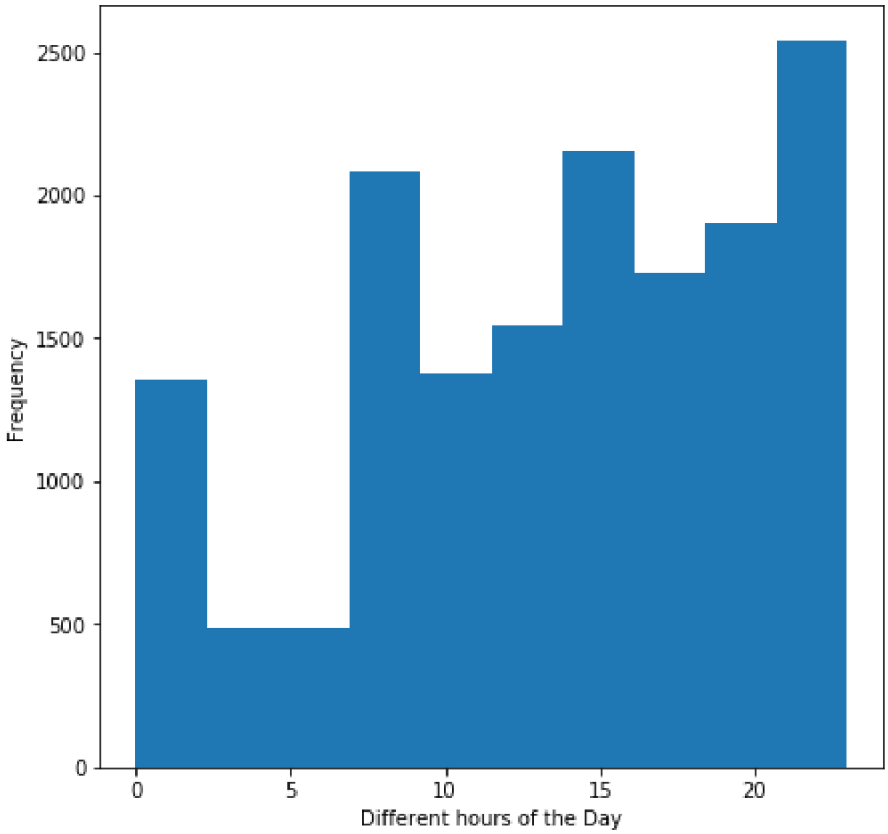
Histograms are constructed by binning the data and counting the number of observations in each bin. The objective of plotting Histogram plot is usually to visualise the shape of the distribution.The number of bins needs to be large enough to reveal interesting features and small enough not to be too noisy.

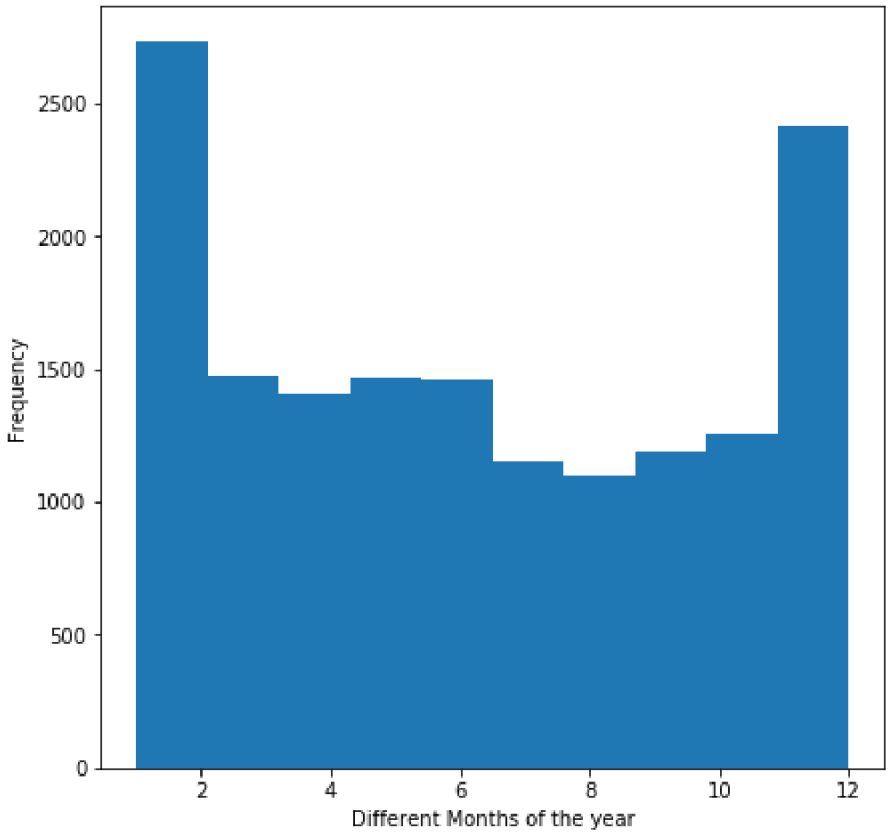


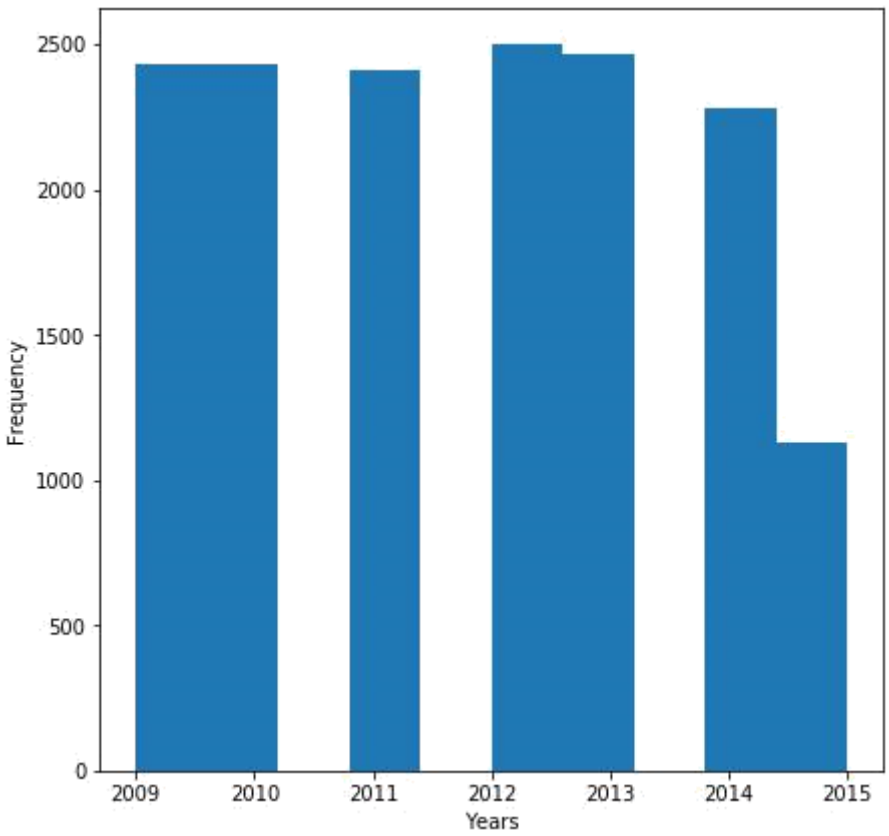



1600
1400
1200
1000
800
600
400
200
0
Frequency
0
5
10
15
20
25
30
Different Days of the month

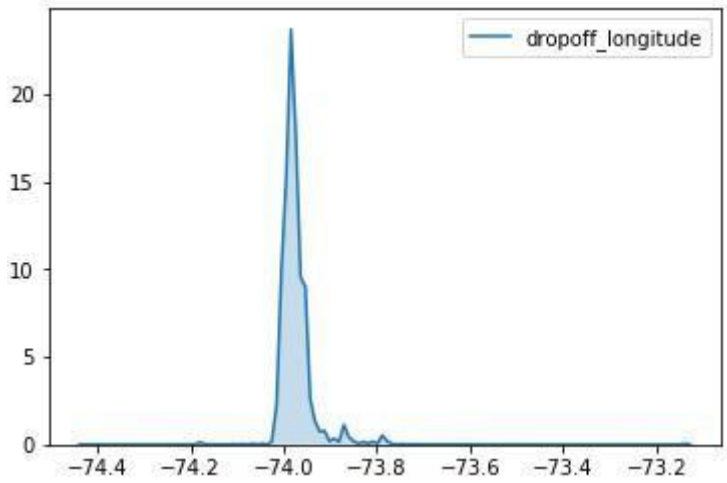

2500
2000
1500
1000
500
0
Frequency
0
1
2
3
4
5
6
Different Days of the week

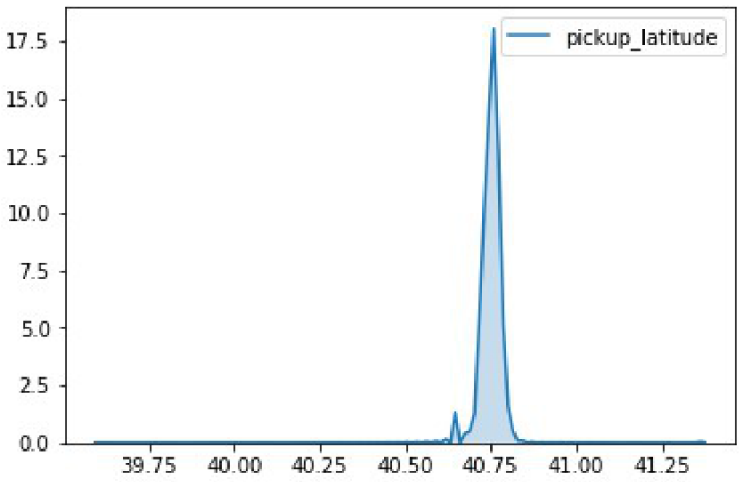

2500
2000
1500
1000
500
0
Frequency
0
5
10
15
20
Different hours of the Day

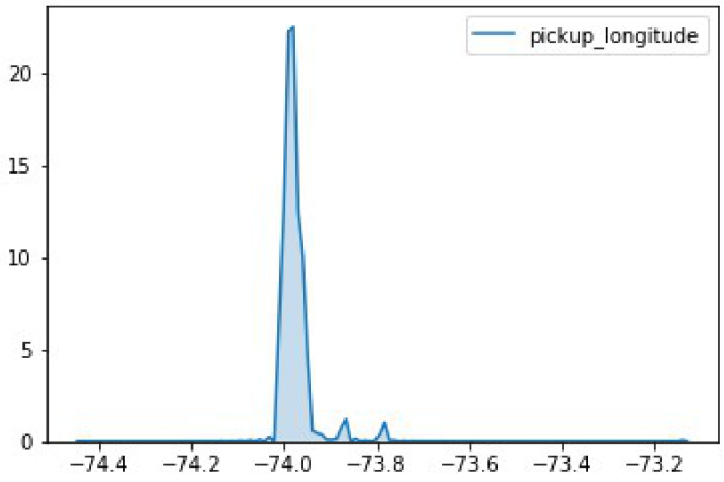

2500
2000
1500
1000
500
0
Frequency
2
4
6
8
10
12
Different Months of the year

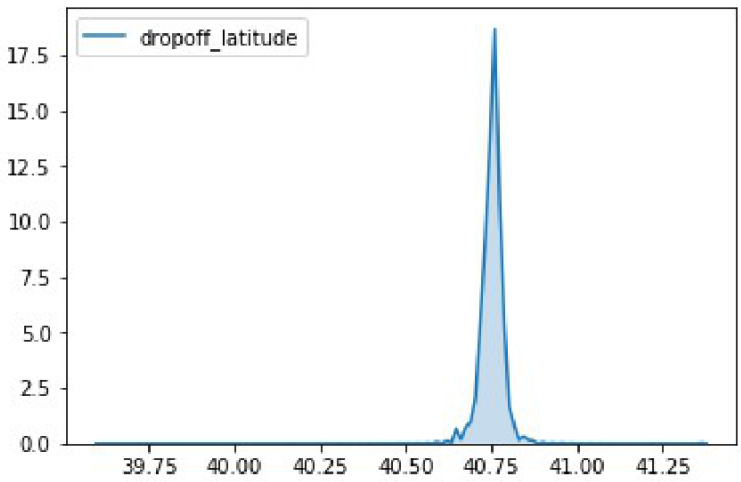


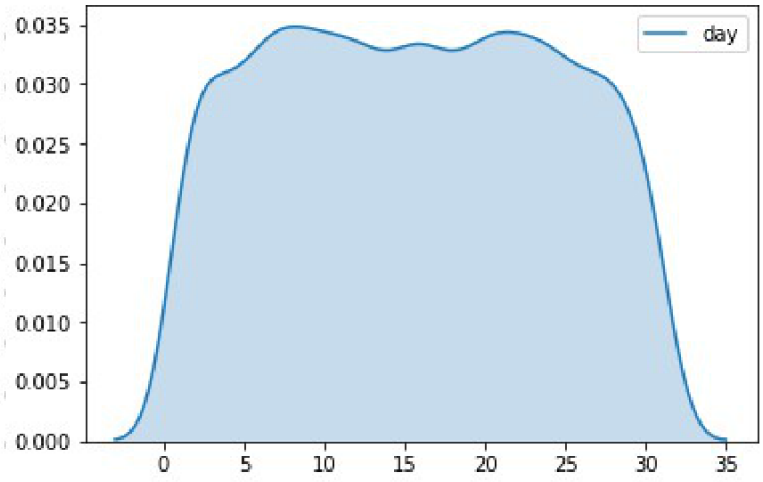
A Density Plot visualises the distribution of data over a continuous interval or time period. Density plots can be thought of as plots of smoothed histograms. An advantage Density Plots have over Histograms is that they're better at determining the distribution shape because they're not affected by the number of bins used.

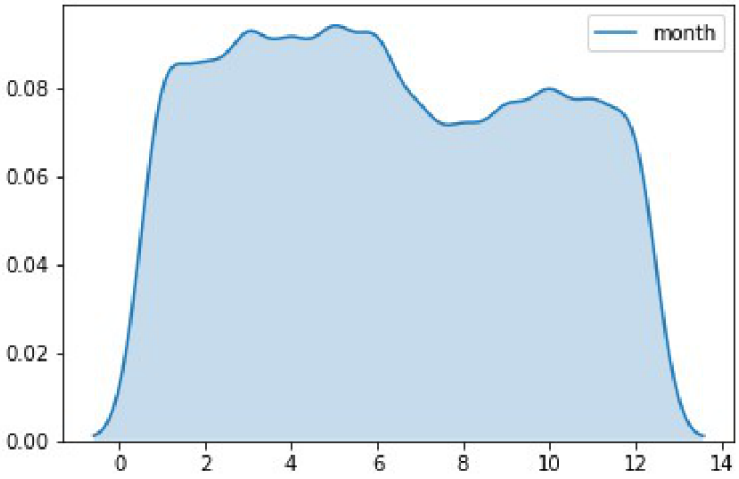


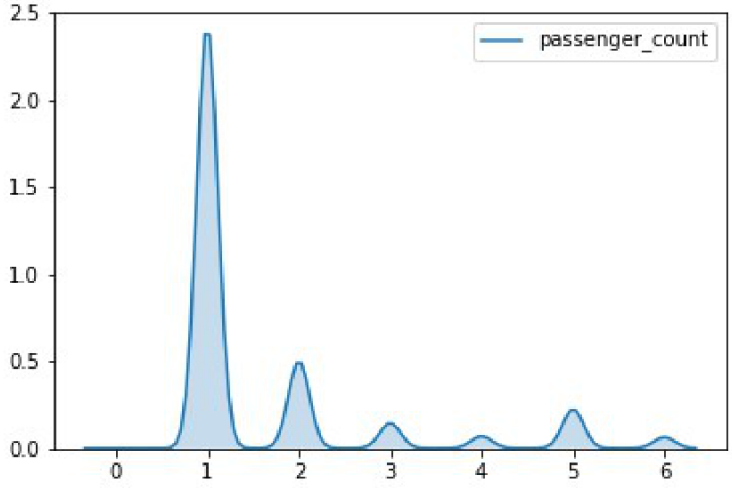

pickup_latitude
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
39.75
40.00
40.25
40.50
40.75
41.00
41.25

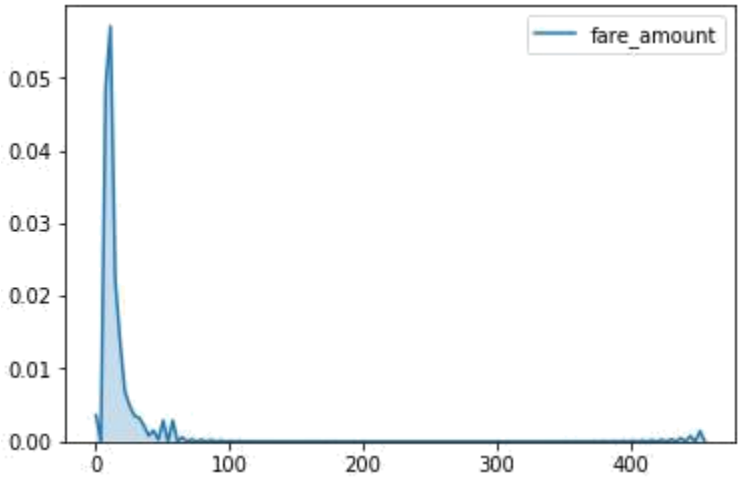

pickup_longitude
20
15
10
5
0
−74.4
−74.2
−74.0
−73.8
−73.6
−73.4
−73.2


dropoff_latitude
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
39.75
40.00
40.25
40.50
40.75
41.00
41.25

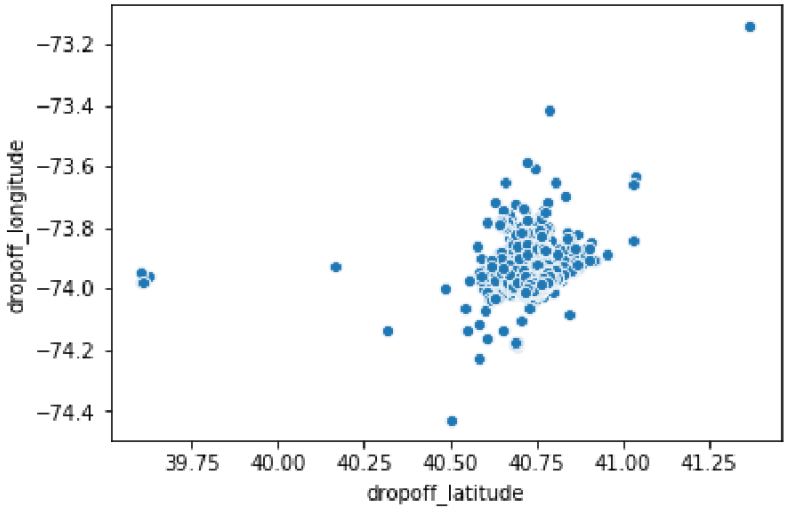

day
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
5
10
15
20
25
30
35

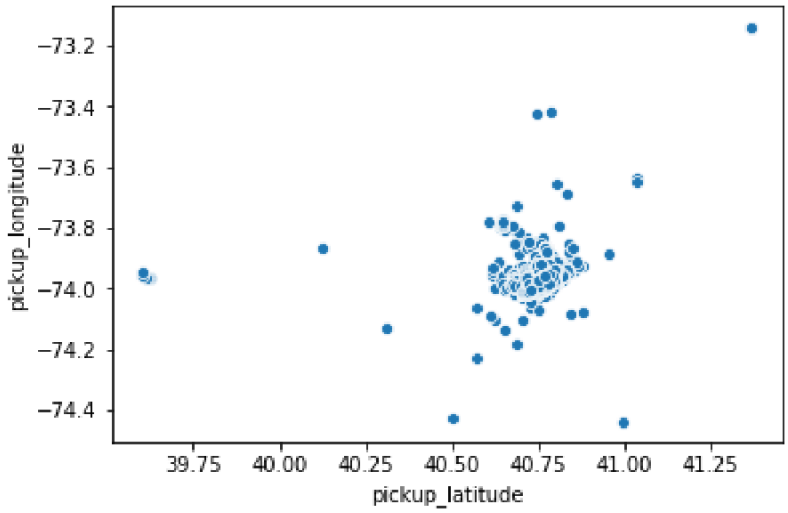

month
0.08
0.06
0.04
0.02
0.00
0
2
4
6
8
10
12
14

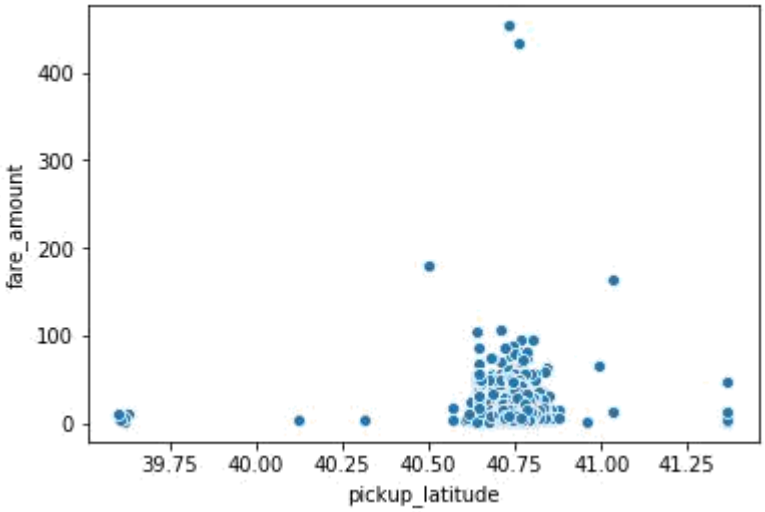

passenger_count
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6

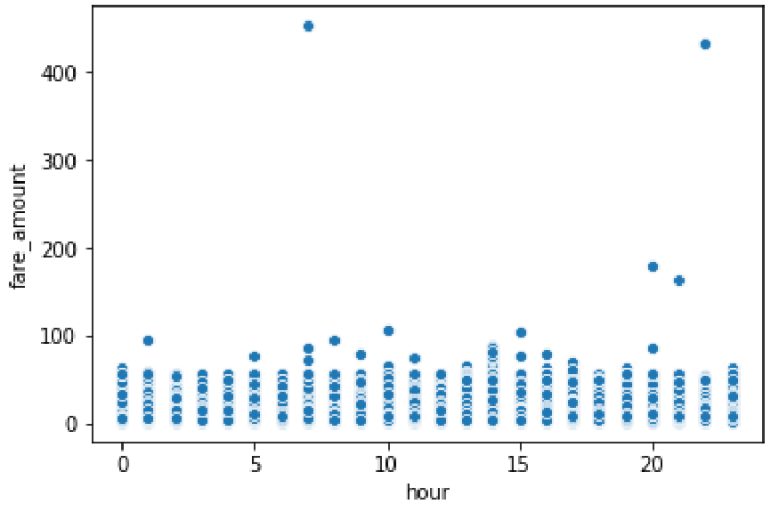


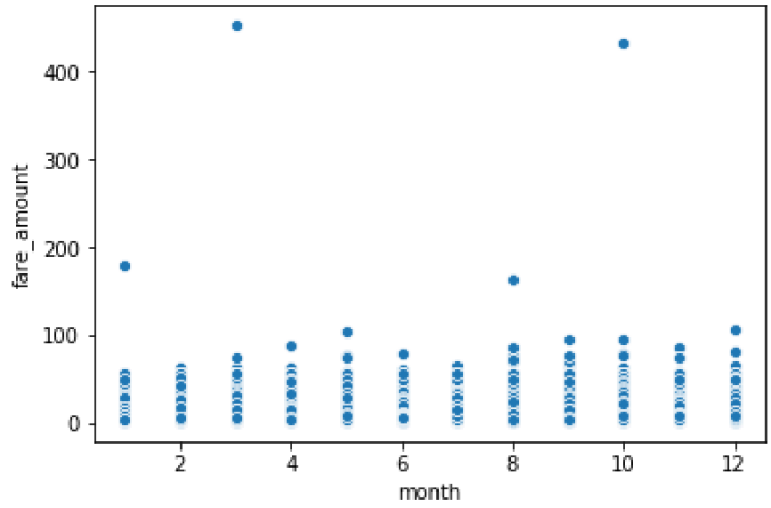
2.1.2.2 Bivariate Analysis

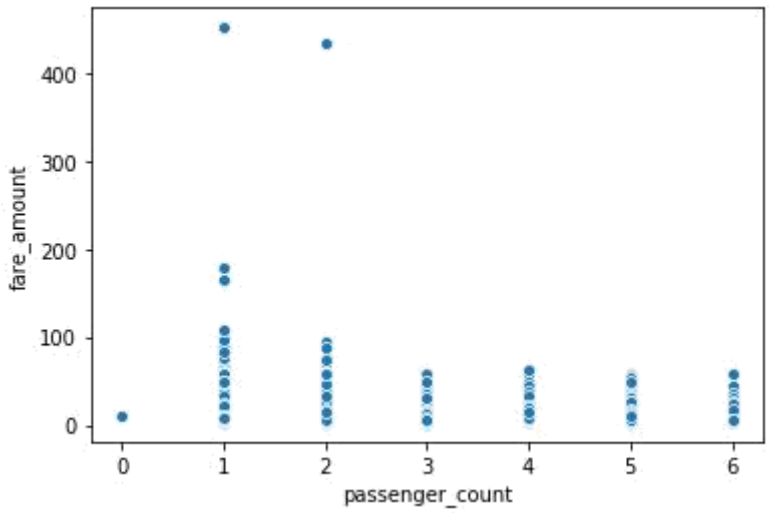





fare_amount
400
300
200
100
0
39.75
40.00
40.25
40.50
40.75
41.00
41.25
pickup_latitude


fare_amount
400
300
200
100
0
0
5
10
15
20
hour


fare_amount
400
300
200
100
0
2
4
6
8
10
12
month



2.1.2.3 Feature Selection

Correlation Analysis:

If the values of one column to other column are similar then it is said to be collinear. Therefore between predictor variables there should be less collinearity as compared to the collinearity among the predictors and variable.

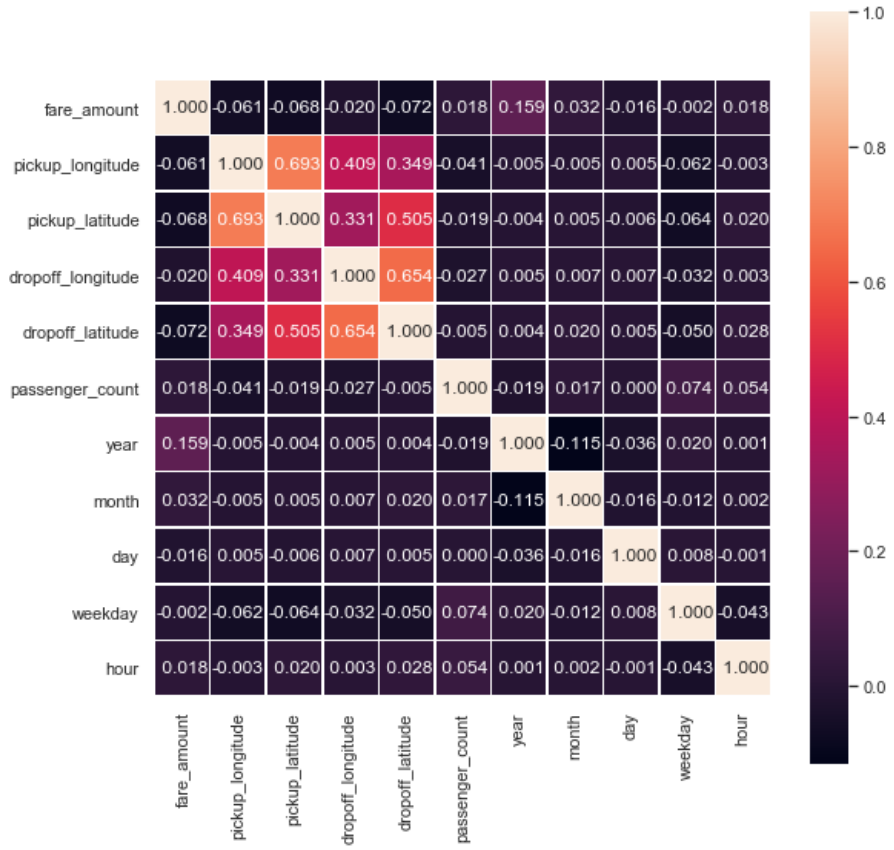
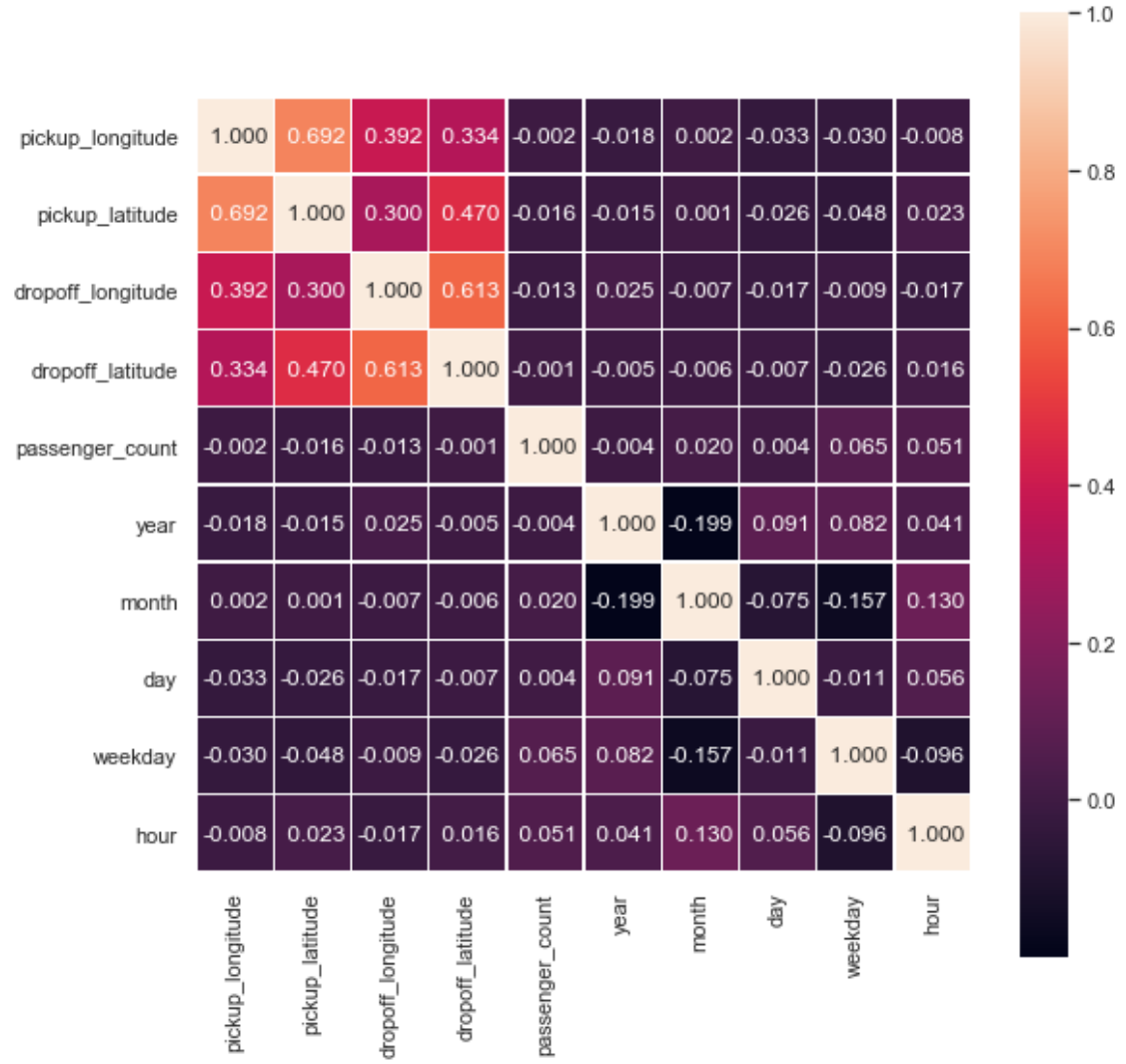


Fig: Collinearity test in train data

The value for collinearity is between -1 to 1. So, any value close to -1/1 will result in high collinearity.

It seems all right in our train and test data the situations nothing is more than 0.4 is positive direction and nothing is less than -0.1 is negative direction.

Therefore the datasets are free from collinearity problem.




```
No variable from the 10 input variables has collinearity problem.

The linear correlation coefficients ranges between:
min correlation ( pickup_longitude ~ Year ):  0.0001063857
max correlation ( pickup_latitude ~ pickup_longitude ):  0.6735994

---------- VIFs of the remained variables --------
           Variables      VIF
1        fare_amount 1.040557
2               Year 1.033507
3              Month 1.013227
4                Day 1.001641
5               Hour 1.006112
6   pickup_longitude 1.968744
7    pickup_latitude 2.075813
8  dropoff_longitude 1.648318
9   dropoff_latitude 1.767427
10   passenger_count 1.009395
> |
```

2.2 Modeling

We always start our model building from the most simplest to more complex. Therefore we start with KNN Regressor.

2.2.1 KNN Regression

KNN regression is one of the simplest algorithm in the whole of Machine learning. It gives a weighted average of the regression function in a local space (k nearest points to a given point). So, we first try to implement and fit KNN regression to our Data.

Python Code:

```
###### KNN Modelling ########

KNN_model = KNeighborsRegressor(n_neighbors= 40).fit(X_train , Y_train)
KNN_pred= KNN_model.predict(X_test)


df_results = pd.DataFrame({'actual': Y_test, 'pred': KNN_pred})
print(df_results.head())

MAPE(Y_test, KNN_pred)
RMSE(Y_test, KNN_pred)

#MAPE is:  0.3084418890595272
#MAE is:  2.470047013144009
#Mean Square :  10.781339613948111
#Root Mean Square :  3.283495030291368
```

RCode:

```
#### KNN ########################################
#Develop Model on training data

fit_knn = knnreg(fare_amount~ ., data =c(X_train,y_train))

#Lets predict for testing data
pred_knn_test = predict(fit_knn,X_test)

# Results
regr.eval(trues = y_test$fare_amount, preds = pred_knn_test, stats = c("mae","ms
df_KN = data.frame("actual"=y_test$fare_amount, "pred"=pred_knn_test)
head(df_KN)

#mae        mse       rmse       mape
#2.5410089 11.9405885  3.4555157  0.3069297
```

2.2.2 Linear Regression

Multiple linear regression is the most common form of linear regression analysis. As a predictive analysis, the 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.

Python Code:

```
dt_lnr_model = sm.OLS(train.iloc[:,0],train.iloc[:,1:]).fit()

#Summary of model
print(dt_lnr_model.summary())

#predict the  model

dt_predict_LR = dt_lnr_model.predict(test.iloc[:,1:])



MAPE(test.iloc[:,0], dt_predict_LR)
RMSE(test.iloc[:,0], dt_predict_LR)

#MAPE is:  0.4002251684131252
#MAE is:  2.9476151408555484
#Mean Square :  14.171074990188412
#Root Mean Square :  3.76444882953513
```

R Code:

```
################## Multiple Linear Regression ############################

set.seed(100)
#Develop Model on training data
fit_LR = lm(fare_amount~ ., data = train)
summary(fit_LR)

#Lets predict for testing data
pred_LR_test = predict(fit_LR,test[,-1])

# Results
regr.eval(trues = y_test$fare_amount, preds = pred_LR_test, stats = c("mae","mse"
df = data.frame("actual"=y_test$fare_amount, "pred"=pred_LR_test)
head(df)

#mae        mse       rmse       mape
#2.9642533 14.8734695  3.8566137  0.3968254

###### Decision Tree
```

2.2.3 Decision Tree Regression

Decision tree builds regression , models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes

```
##################################################Decision Tree#########################################################
dt_model = DecisionTreeRegressor(random_state=123).fit(train.iloc[:,1:], train.iloc[:,0])

print(dt_model)

dt_predictions = dt_model.predict(test.iloc[:,1:])


df_results = pd.DataFrame({'actual': test.iloc[:,0], 'pred': dt_predictions})
df_results.head()


MAPE(test.iloc[:,0], dt_predictions)
RMSE(test.iloc[:,0], dt_predictions)

#MAPE is:  0.2870557144870078
#MAE is:  2.248405252730194
#Mean Square :  10.056863408315259
#Root Mean Square :  3.1712558093467105
```

R Code:

```
###### Decision Tree#########################

#Develop Model on training data
fit_DT = rpart(fare_amount ~., data = train, method = 'anova')
pred_DT_test = predict(fit_DT,test[,-1])

# Results

regr.eval(trues = y_test$fare_amount, preds = pred_DT_test, stats = c("mae","mse"

df_dt = data.frame("actual"=y_test$fare_amount, "pred"=pred_DT_test)
head(df_dt)

#mae        mse       rmse       mape
#2.7778769 13.2811050  3.6443250  0.3665036
```

2.2.4 Gradient Boosting Decision Tree:

Regression Gradient boosting is a machine

learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalises them by allowing optimisation of an arbitrary differentiable loss function.

Python Code:

```
###### GBR Modelling ##########


gbr_model = GradientBoostingRegressor(max_depth= 2,learning_rate = 0.1).fit(train.iloc[:,1:],train.iloc[:,0])
gbr_pred= gbr_model.predict(test.iloc[:,1:])

MAPE(test.iloc[:,0], gbr_pred)
RMSE(test.iloc[:,0], gbr_pred)

#MAPE is:  0.3024254007844994
#MAE is:  2.1996787335105425
#Mean Square :  8.13426907872889
#Root Mean Square :  2.8520640032665625
```

RCode:

```
#################### GBDT############################

#Develop Model on training data
fit_GBDT = gbm(fare_amount ~., data = train, n.trees = 500, interaction.depth = 2

#Lets predict for testing data
pred_GBDT_test = predict(fit_GBDT,test[,-1], n.trees = 500)

# Results
regr.eval(trues = y_test$fare_amount, preds = pred_GBDT_test, stats = c("mae","ms
df_GB = data.frame("actual"=y_test$fare_amount, "pred"=pred_GBDT_test)
head(df_GB)

#mae       mse      rmse      mape
#2.0899910 8.4587235 2.9083885 0.2658953
```

2.2.5 Random Forest Regression

Random Forest Regression or Regression Trees are know to be very unstable, in other words, a small change in our data may drastically change your model. The Random Forest uses this instability as an advantage through bagging resulting on a very stable model.

Python Code:

```
##########################################Random Forest ##########################################

# Set best parameters given by random search
rf.set_params(max_features = 'auto',
              max_depth =8,
              n_estimators = 500,
             bootstrap = 'True'
               )

rf.fit(train.iloc[:,1:], train.iloc[:,0])

# Use the forest's predict method on the test data
dt_rfPredictions = rf.predict(test.iloc[:,1:])

df_results = pd.DataFrame({'actual': test.iloc[:,0], 'pred': dt_rfPredictions})
print(df_results.head())

MAPE(test.iloc[:,0], dt_rfPredictions)
RMSE(test.iloc[:,0], dt_rfPredictions)

#MAPE is:  0.2970395006180068
#MAE is:  2.1666677762006925
#Mean Square :  8.021622657590507
#Root Mean Square :  2.832246927368888
```

R Code :

```
####################Random Forest###################################
#Develop Model on training data
fit_RF = randomForest(fare_amount ~., data = train,ntree = 500 ,nodesize =8,impor
pred_RF_test = predict(fit_RF,test[,-1])

# Results
regr.eval(trues = y_test$fare_amount, preds = pred_RF_test, stats = c("mae","mse"
df_RF = data.frame("actual"=y_test$fare_amount, "pred"=pred_RF_test)
head(df_RF)

#mae       mse      rmse      mape
#1.9770044 7.6294763 2.7621507 0.2494035
```

Chapter 3

Conclusion 3.1

Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using any of the following criteria:

1. Predictive Performance

2. Interpretability

3. Computational Efficiency

In our case of Cab fare , the latter two, Interpretability and Computation Efficiency, do not hold much significance. Therefore we will use Predictive performance as the criteria to compare and evaluate models. Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure.

3.1.1 Root Mean Square Value

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. Also, Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. So, RMSE becomes more useful when large errors are particularly undesirable. So, Root Mean Square value seems like a perfect choice for our problem at hand.

3.2 Model Selection

We saw that both models Random Forest along with SVR and Ridge Regression perform comparatively on RMSE (Root Mean Square Error) , Random Forest gives the best results on the test data. Model comparison table is given below.

RMSE Results in R:

```
  rmse             Model
1  3.4               KNN
2  3.8 Linear Regression
3 3.64     Decision tree
4 2.76     Random forest
5 2.90               GBDT
> |
```

RMSE Results in Python:

```
   rmse             Model
0  3.28     KNN Regression
1  3.76  Linear Regression
2  3.17     Decision Trees
3  2.85               GBDT
4  2.83      Random Forest
```

So, It is obvious from above model performance comparison table Random Forest perform comparatively on RMSE (Root Mean Square Error) and can be used for deployment.

Chapter 4

Appendix A: R Code

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

#Set working directory
setwd("C:/Users/sai/Documents/New folder R")
getwd()


#import libraries

library(readr)
library(gridExtra)
library(corrgram)
library(caret)
library(tidyr)
library(rpart)
library(randomForest)
library(dplyr)
library(ggplot2)
library(data.table)
library(gbm)
library(usdm)
library(DMwR)

#Lets name our training dataset as df_train
df_train <- read_csv("train_cab.csv")


#Check the first 10 observations
View(df_train)

#Also  name our testing dataset as df_test
df_test <-read_csv("test.csv")

#Check the first 10 observations
View(df_test)

#Getting the column names of the dataset
colnames(df_train)
colnames(df_test)

#Getting the structure of the dataset
str(df_train)
str(df_test)

#Getting the number of variables and obervation in the datasets
dim(df_train)
dim(df_test)

#datatypes of train and test dataset
```

```
map(df_train, class)
map(df_test, class)

#summaries of train and test data's

summary(df_train)
summary(df_test)

#Clearly in the summary it can be seen there are so many anamolies:
# 1. fare amount is negative at few places and at a certain place it is
54343 which is not possible.
# 2. There are missing values in fare_amount and passenger_count
# 3. Number of passengers in few rows is 500 which is not possible for a
cab to carry.

###################################Missing Values
Analysis#################################################

#checking presence of Missing values in training set

missing_val = data.frame(apply(df_train,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage =
(missing_val$Missing_percentage/nrow(df_train)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]

View(missing_val)
## < - passenger_count, fare_amount is having missing values

#Plot these Missing values
ggplot(data = missing_val[1:3,], aes(x=reorder(Columns, -
Missing_percentage),y = Missing_percentage))+
  geom_bar(stat = "identity",fill = "grey")+xlab("Parameter")+
  ggtitle("Missing data percentage (Train)") + theme_bw()

#checking presence of Missing values in test set

missing_val1 = data.frame(apply(df_test,2,function(x){sum(is.na(x))}))
missing_val1$Columns = row.names(missing_val1)
names(missing_val1)[1] =  "Missing_percentage"
missing_val1$Missing_percentage =
(missing_val1$Missing_percentage/nrow(df_test)) * 100
missing_val1 = missing_val1[order(-missing_val1$Missing_percentage),]
row.names(missing_val1) = NULL
missing_val1 = missing_val1[,c(2,1)]
View(missing_val1)
## < - No missing values in test data

#Mean Method Imputation
df_train$passenger_count[is.na(df_train$passenger_count)] =
mean(df_train$passenger_count, na.rm = T)
df_train$fare_amount[is.na(df_train$fare_amount)] =
mean(df_train$fare_amount, na.rm = T)

#check the number of Missing values after Imputation
```

```
sum(is.na(df_train))

## < - now no missing values are present

#Now lets convert our pickup_datetime to numeric
df_train$pickup_datetime = gsub( " UTC", "",
as.character(df_train$pickup_datetime))
df_test$pickup_datetime = gsub( " UTC", "",
as.character(df_test$pickup_datetime))

Split_datetime = function(df){
  df = separate(df, "pickup_datetime", c("Date", "Time"), sep = " ")
  df = separate(df, "Date", c("Year", "Month", "Day"), sep = "-")
  df = separate(df, "Time", c("Hour"), sep = ":")
  print(sum(is.na(df)))
  df$Year = as.numeric(df$Year)
  df$Month = as.numeric(df$Month)
  df$Day = as.numeric(df$Day)
  df$Hour = as.numeric(df$Hour)
  df$Year[is.na(df$Year)] = mean(df$Year, na.rm = T)
  df$Month[is.na(df$Month)] = mean(df$Month, na.rm = T)
  df$Day[is.na(df$Day)] = mean(df$Day, na.rm = T)
  df$Hour[is.na(df$Hour)] = mean(df$Hour, na.rm = T)
  print(sum(is.na(df)))
  return(df)}
df_train = Split_datetime(df_train)
df_test = Split_datetime(df_test)

################################## Removing Anamolies
#############################################

#Remove unwanted values and bring our dataset in proper shape
df_train = df_train[((df_train['fare_amount'] >= 0) &
(df_train['fare_amount'] <=600)) & ((df_train['pickup_longitude']> -79)
&  (df_train['pickup_longitude'] < - 70)) &
((df_train['pickup_latitude']) > 36 & (df_train['pickup_latitude'] < 45))
& ((df_train['dropoff_longitude']) > -79 & (df_train['dropoff_longitude']
< -70)) & ((df_train['dropoff_latitude'] >= 36)  &
(df_train['dropoff_latitude'] < 45)) & ((df_train['passenger_count']  >=
1) & (df_train['passenger_count'] <= 7)),]

df_train$pickup_latitude = gsub( "0.00000", "0",
as.numeric(df_train$pickup_latitude))
df_train$dropoff_latitude = gsub( "0.00000", "0",
as.numeric(df_train$dropoff_latitude))
df_train$pickup_longitude= gsub( "0.00000", "0",
as.numeric(df_train$pickup_longitude))
df_train$dropoff_longitude = gsub( "0.00000", "0",
as.numeric(df_train$dropoff_longitude))

#Remove rows containing 0 as value
df_train = df_train[apply(df_train, 1, function(row) all(row !=0)),]
df_train <- data.frame(sapply(df_train, function(x)
as.numeric(as.character(x))))
sapply(df_train, class)

df_test = df_test[apply(df_test, 1, function(row) all(row !=0)),]
```

```
df_test <- data.frame(sapply(df_test, function(x)
as.numeric(as.character(x))))
sapply(df_test, class)

############################################ Outlier Analysis
############################################

## BoxPlots - Distribution and Outlier Check

numeric_index = sapply(df_train,is.numeric)

numeric_data = df_train[,numeric_index]

cnames = colnames(numeric_data)

for (i in 1:length(cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), group = 1),
data = subset(df_train))+
           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=cnames[i],X="count")+
           ggtitle(paste("Box plot for",cnames[i])))
}

gridExtra::grid.arrange(gn2,gn3,ncol=2)
gridExtra::grid.arrange(gn6,gn7,ncol=2)
gridExtra::grid.arrange(gn4,gn5,ncol=2)
gridExtra::grid.arrange(gn8,gn9,ncol=2)

for(i in cnames){
  print(i)
  val = df_train[,i][df_train[,i] %in% boxplot.stats(df_train[,i])$out]
  print(length(val))
  df_train[,i][df_train[,i] %in% val] = NA
}
df_train <- data.frame(sapply(df_train, function(x) ifelse(is.na(x),
mean( x, na.rm = TRUE),x)))

num = sapply(df_test,is.numeric)

num_data = df_test[,num]

cnamestest = colnames(num_data)

for(c in cnamestest){
  print(c)
  val = df_test[,c][df_test[,c] %in% boxplot.stats(df_test[,c])$out]
  print(length(val))
  df_test[,c][df_test[,c] %in% val] = NA
}
df_test <- data.frame(sapply(df_test, function(y) ifelse(is.na(y),
mean(y, na.rm = TRUE),y)))

df_train$passenger_count =  round(df_train$passenger_count)
```

```
df_test$passenger_count =  round(df_test$passenger_count)

ggplot(df_train, aes(x = fare_amount, y = pickup_latitude, group = 1)) +
geom_boxplot()
ggplot(df_train, aes(x = fare_amount, y = pickup_longitude  , group = 1))
+  geom_boxplot()
ggplot(df_train, aes(x = fare_amount, y = dropoff_longitude  , group =
1)) +  geom_boxplot()
ggplot(df_train, aes(x = fare_amount, y = dropoff_latitude  , group = 1))
+  geom_boxplot()
ggplot(df_train, aes(x = fare_amount, y = passenger_count  , group = 1))
+  geom_boxplot()

############################################ Univarate Analysis
##########################################

hist(df_train$fare_amount)
hist(df_train$pickup_latitude)
hist(df_train$pickup_longitude)
hist(df_train$dropoff_latitude)
hist(df_train$dropoff_longitude)
hist(df_train$passenger_count)
hist(df_train$Year)
hist(df_train$Month)
hist(df_train$Day)
hist(df_train$Hour)

######################################### Bivariate  Relationship
#########################################################
scat1 = ggplot(df_train, aes(x = fare_amount, y = pickup_latitude, group
= 1)) +  geom_point()
scat2 = ggplot(df_train, aes(x = fare_amount, y = Year, group = 1)) +
geom_point()
scat3 = ggplot(df_train, aes(x = fare_amount, y = pickup_longitude  ,
group = 1)) +  geom_point()
scat4 = ggplot(df_train, aes(x = fare_amount, y = dropoff_longitude  ,
group = 1)) +  geom_point()
scat5 = ggplot(df_train, aes(x = fare_amount, y = dropoff_latitude  ,
group = 1)) +  geom_point()
scat6 = ggplot(df_train, aes(x = fare_amount, y = passenger_count  ,
group = 1)) +  geom_point()
scat7 = ggplot(df_train, aes(x = fare_amount, y = Day  , group = 1)) +
geom_point()
scat8 = ggplot(df_train, aes(x = fare_amount, y = Hour  , group = 1)) +
geom_point()
scat9 = ggplot(df_train, aes(x = fare_amount, y = Month  , group = 1)) +
geom_point()

gridExtra::grid.arrange(scat1,scat2,scat3,scat4,scat5,scat6,scat7,scat8,s
cat9,ncol=3)
################################## Feature Selection
################################################
## Correlation Plot
corrgram(df_train[,numeric_index], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation
Plot")

vifcor(df_train)
```

```
##### Splitting the data into train and test
set.seed(12)
n = nrow(df_train)
trainIndex = sample(1:n, size = round(0.8*n), replace=FALSE)
train = df_train[trainIndex ,]
test = df_train[-trainIndex ,]
X_train = subset(train,select = -c(fare_amount))
y_train = subset(train,select = c(fare_amount))

X_test = subset(test,select = -c(fare_amount))
y_test = subset(test,select = c(fare_amount))

############################################## Scaling the data
#######################################
#Standardisation

for(i in colnames(X_train))
{
  print(i)
  X_train[,i] = (X_train[,i] - min(X_train[,i]))/(max(X_train[,i])-
min(X_train[,i]))
}
for(i in colnames(X_test))
{
  print(i)
  X_test[,i] = (X_test[,i] - min(X_test[,i]))/(max(X_test[,i])-
min(X_test[,i]))
}

######################################### Machine learning
model############################################

#### KNN ########################################
#Develop Model on training data

fit_knn = knnreg(fare_amount~ ., data =c(X_train,y_train))

#Lets predict for testing data
pred_knn_test = predict(fit_knn,X_test)

# Results
regr.eval(trues = y_test$fare_amount, preds = pred_knn_test, stats =
c("mae","mse","rmse","mape"))
df_KN = data.frame("actual"=y_test$fare_amount, "pred"=pred_knn_test)
head(df_KN)

#mae        mse       rmse       mape
#2.5410089 11.9405885  3.4555157  0.3069297



################# Multiple Linear Regression ############################

set.seed(100)
#Develop Model on training data
fit_LR = lm(fare_amount~ ., data = train)
summary(fit_LR)
```

```
#Lets predict for testing data
pred_LR_test = predict(fit_LR,test[,-1])

# Results
regr.eval(trues = y_test$fare_amount, preds = pred_LR_test, stats =
c("mae","mse","rmse","mape"))
df = data.frame("actual"=y_test$fare_amount, "pred"=pred_LR_test)
head(df)

#mae        mse       rmse       mape
#2.9642533 14.8734695  3.8566137  0.3968254

###### Decision Tree#########################

#Develop Model on training data
fit_DT = rpart(fare_amount ~., data = train, method = 'anova')
pred_DT_test = predict(fit_DT,test[,-1])

# Results

regr.eval(trues = y_test$fare_amount, preds = pred_DT_test, stats =
c("mae","mse","rmse","mape"))

df_dt = data.frame("actual"=y_test$fare_amount, "pred"=pred_DT_test)
head(df_dt)

#mae        mse       rmse       mape
#2.7778769 13.2811050  3.6443250  0.3665036

################### Random Forest###################################
#Develop Model on training data
fit_RF = randomForest(fare_amount ~., data = train,ntree = 500 ,nodesize
=8,importance=TRUE)
pred_RF_test = predict(fit_RF,test[,-1])

# Results
regr.eval(trues = y_test$fare_amount, preds = pred_RF_test, stats =
c("mae","mse","rmse","mape"))
df_RF = data.frame("actual"=y_test$fare_amount, "pred"=pred_RF_test)
head(df_RF)

#mae       mse      rmse      mape
#1.9770044 7.6294763 2.7621507 0.2494035

#################### GBDT############################

#Develop Model on training data
fit_GBDT = gbm(fare_amount ~., data = train, n.trees = 500,
interaction.depth = 2)

#Lets predict for testing data
pred_GBDT_test = predict(fit_GBDT,test[,-1], n.trees = 500)

# Results
regr.eval(trues = y_test$fare_amount, preds = pred_GBDT_test, stats =
c("mae","mse","rmse","mape"))
df_GB = data.frame("actual"=y_test$fare_amount, "pred"=pred_GBDT_test)
```

```
head(df_GB)

#mae       mse      rmse      mape
#2.0899910 8.4587235 2.9083885 0.2658953

df_rmse = data.frame("rmse" = c("3.4","3.8","3.64","2.76","2.90"
),"Model" = c("KNN","Linear Regression","Decision tree","Random
forest","GBDT"))
print(df_rmse)

#Prediction for given test data with best fit model random forest
pred_RF_test = predict(fit_RF,df_test)
pred_RF_test_R_DF = data.frame("fare_amount" = pred_RF_test )
write.csv(pred_RF_test_R_DF,"Test_Predictions_R.csv",row.names = FALSE)
```

Appendix B: Python code

```
#!/usr/bin/env python
# coding: utf-8

# In[38]:


#Importing the necessary packages
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split,RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from collections import Counter
import statsmodels.api as sm
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import GradientBoostingRegressor


# In[3]:


#Set working directory
os.chdir("C:/Users/sai/Documents/Python Scripts")
print(os.getcwd())


# In[4]:


# Loading the the train and test datasets associated with the Cab Fare:
df_train = pd.read_csv("train_cab.csv")
df_test = pd.read_csv("test.csv")
```

```
#Check the first 5 rows of our datasets
print(df_train.head(30))
print("\n")
print(df_test.head(5))


# In[5]:


#Data Types of the columns
print(df_train.dtypes)
print(df_test.dtypes)

#Converting the fare amount column into numeric data form
df_train["fare_amount"] =
pd.to_numeric(df_train["fare_amount"],errors='coerce')

print(df_train.dtypes)


# In[6]:


#description of our datatests
print(df_train.describe())
print(df_test.describe())


# In[9]:


########################################## MisssingValue Analysis
#####################################################

def missin_val(df):
    total = df.isnull().sum().sort_values(ascending=False)
    percent = (df.isnull().sum() * 100
/df.isnull().count()).sort_values(ascending=False)
    missing_data = pd.concat([total, percent], axis=1, keys=['Total',
'Percent'])
    return(missing_data)

print("The missing value percentage in training data :
\n\n",missin_val(df_train))
print("\n")
print("The missing value percentage in test data :
\n\n",missin_val(df_test))
print("\n")

df_train["passenger_count"] =
df_train["passenger_count"].fillna(df_train["passenger_count"].mean())
df_train["fare_amount"] =
df_train["fare_amount"].fillna(df_train["fare_amount"].mean())

print("Is there still any missing value in the training
data:\n\n",missin_val(df_train))
print("\n")
```

```
# In[10]:


######################### Proper Aligning the Dataset
#############################################################

## The fare amount column is having some neagative values, Lets Check it
print(Counter(df_train['fare_amount']<0))
print(Counter(df_train['fare_amount']>1000))
#Also there are values like more 6 persons in a cab, Lets cross check
print(Counter(df_train['passenger_count']>6))
print(Counter(df_train['passenger_count']<0))

#Originally, Latitudes range from -90 to 90.
#Originally, Longitudes range from -180 to 180.
#But our data is purely negative Longitudes and purely positive latitudes
#lets align our data in its respective minimum and maximum Longitudes
#and latitudes values, also removing fare amount,passenger count those
are negative and above optimum level.


df_train = df_train[((df_train['pickup_longitude'] > -79) &
(df_train['pickup_longitude'] < -70)) &
           ((df_train['dropoff_longitude'] > -79) &
(df_train['dropoff_longitude'] < -70)) &
           ((df_train['pickup_latitude'] > 36) &
(df_train['pickup_latitude'] < 45)) &
           ((df_train['dropoff_latitude'] > 36) &
(df_train['dropoff_latitude'] < 45)) &
           ((df_train['passenger_count'] > 0) &
(df_train['passenger_count'] < 7))  &
           ((df_train['fare_amount'] > 0)& (df_train['fare_amount'] <
1000))]


df_test = df_test[((df_test['pickup_longitude'] > -79) &
(df_test['pickup_longitude'] < -70)) &
           ((df_test['dropoff_longitude'] > -79) &
(df_test['dropoff_longitude'] < -70)) &
           ((df_test['pickup_latitude'] > 36) &
(df_test['pickup_latitude'] < 45)) &
           ((df_test['dropoff_latitude'] > 36) &
(df_test['dropoff_latitude'] < 45)) &
           (df_test['passenger_count'] > 0) ]


# In[11]:


#Split our Datetime into individual columns for ease of data processing
and modelling
def align_datetime(df):
    df["pickup_datetime"] = df["pickup_datetime"].map(lambda x: str(x)[:-
3])
    df["pickup_datetime"] = pd.to_datetime(df["pickup_datetime"],
format='%Y-%m-%d %H:%M:%S')
```

```
    df['year'] = df.pickup_datetime.dt.year
    df['month'] = df.pickup_datetime.dt.month
    df['day'] = df.pickup_datetime.dt.day
    df['weekday'] = df.pickup_datetime.dt.weekday
    df['hour'] = df.pickup_datetime.dt.hour
    return(df["pickup_datetime"].head())

align_datetime(df_train)
align_datetime(df_test)


#Remove the datetime column
df_train.drop('pickup_datetime', axis=1, inplace=True)
df_test.drop('pickup_datetime', axis=1, inplace=True)


#Checking NA in the fresh Dataset
print(df_train.isnull().sum())
df_train=df_train.fillna(df_train.mean())
print(df_train.isnull().sum())

print(df_train.head(5))
print("\n")
print(df_test.head(5))


# In[12]:


#Setting proper data type for each columns
print(df_train.dtypes)
print(df_test.dtypes)

df_train= df_train.astype({"passenger_count":int,"year":int,"month":int
,"day" :int,"weekday":int,"hour":int})
print(df_train.dtypes)


# In[14]:


################################################  Outlier Analysis
#########################################################################
######################
df_train.plot(kind='box', subplots=True, layout=(8,3), sharex=False,
sharey=False, fontsize=8)
plt.subplots_adjust(left=0.125, bottom=0.1, right=0.9, top= 3,wspace=0.2,
hspace=0.2)
plt.show()


##Detect and delete outliers from data
def outliers_analysis(df):
    for i in df.columns:
        print(i)
        q75, q25 = np.percentile(df.loc[:,i], [75 ,25])
        iqr = q75 - q25
```

```
        min = q25 - (iqr*1.5)
        max = q75 + (iqr*1.5)
        print(min)
        print(max)

        df = df.drop(df[df.loc[:,i] < min].index)
        df = df.drop(df[df.loc[:,i] > max].index)
    return(df)


def eliminate_rows_with_zero_value(df):
    df= df[df!= 0]
    df=df.fillna(df.mean())
    return(df)

df_train = outliers_analysis(df_train)
df_train = eliminate_rows_with_zero_value(df_train)

df_train.plot(kind='box', subplots=True, layout=(8,3), sharex=False,
sharey=False, fontsize=8)
plt.subplots_adjust(left=0.125, bottom=0.1, right=0.9, top= 3,wspace=0.2,
hspace=0.2)
plt.show()




# In[15]:


df_test.plot(kind='box', subplots=True, layout=(8,3), sharex=False,
sharey=False, fontsize=8)
plt.subplots_adjust(left=0.125, bottom=0.1, right=0.9, top= 3,wspace=0.2,
hspace=0.2)
plt.show()

df_test = outliers_analysis(df_test)
df_test = eliminate_rows_with_zero_value(df_test)

df_test.plot(kind='box', subplots=True, layout=(8,3), sharex=False,
sharey=False, fontsize=8)
plt.subplots_adjust(left=0.125, bottom=0.1, right=0.9, top= 3,wspace=0.2,
hspace=0.2)
plt.show()


# In[16]:


######################################## univarate analysis
#########################################################################
################################

#Histogram Plot of passenger_count Column
plt.figure(figsize=(7,7))
plt.hist(df_train['passenger_count'],bins = 6)
plt.xlabel('No. of Passengers')
```

```
plt.ylabel('Frequency')

#Histogram Plot of passenger_count Column
plt.figure(figsize=(7,7))
plt.hist(df_train['fare_amount'],bins=25)
plt.xlabel('Different amount of Fare')
plt.ylabel('Frequency')

#Histogram Plot of day Column
plt.figure(figsize=(7,7))
plt.hist(df_train['day'],bins=10)
plt.xlabel('Different Days of the month')
plt.ylabel('Frequency')

#Histogram Plot of weekday Column
plt.figure(figsize=(7,7))
plt.hist(df_train['weekday'],bins=10)
plt.xlabel('Different Days of the week')
plt.ylabel('Frequency')

#Histogram Plot of hour Column
plt.figure(figsize=(7,7))
plt.hist(df_train['hour'],bins=10)
plt.xlabel('Different hours of the Day')
plt.ylabel('Frequency')

#Histogram Plot of month Column
plt.figure(figsize=(7,7))
plt.hist(df_train['month'],bins=10)
plt.xlabel('Different Months of the year')
plt.ylabel('Frequency')

#Histogram Plot of year Column
plt.figure(figsize=(7,7))
plt.hist(df_train['year'],bins=10)
plt.xlabel('Years')
plt.ylabel('Frequency')

#Histogram Plot of year Column
plt.figure(figsize=(7,7))
plt.hist(df_train['dropoff_latitude'])
plt.xlabel('dropoff_latitude')
plt.ylabel('Frequency')
#Histogram Plot of year Column
plt.figure(figsize=(7,7))
plt.hist(df_train['dropoff_longitude'])
plt.xlabel('dropoff_longitude')
plt.ylabel('Frequency')

#Histogram Plot of year Column
plt.figure(figsize=(7,7))
plt.hist(df_train['dropoff_latitude'])
plt.xlabel('dropoff_latitude')
plt.ylabel('Frequency')
#Histogram Plot of year Column
plt.figure(figsize=(7,7))
plt.hist(df_train['pickup_latitude'])
plt.xlabel('pickup_latitude')
```

```
plt.ylabel('Frequency')


# In[17]:


################################################  Density Plots
#############################################################################
#########################
fig,x = plt.subplots(nrows=5,ncols=2)
fig.set_size_inches(10,12)
sns.kdeplot(df_train['fare_amount'], shade = True,ax=x[0][0])
sns.kdeplot(df_train['passenger_count'], shade = True,ax=x[0][1])
sns.kdeplot(df_train['month'], shade = True,ax=x[1][0])
sns.kdeplot(df_train['day'], shade = True,ax=x[1][1])
sns.kdeplot(df_train['weekday'], shade = True,ax=x[2][0])
sns.kdeplot(df_train['hour'], shade = True,ax=x[2][1])
sns.kdeplot(df_train['dropoff_latitude'], shade = True,ax=x[3][0])
sns.kdeplot(df_train['pickup_longitude'], shade = True,ax=x[3][1])
sns.kdeplot(df_train['pickup_latitude'], shade = True,ax=x[4][0])
sns.kdeplot(df_train['dropoff_longitude'], shade = True,ax=x[4][1])


# In[24]:


################################################ Bivariate Plots
#############################################################################
#########################
fig,x = plt.subplots(nrows=6,ncols=2)
fig.set_size_inches(10,12)

sns.scatterplot(x="passenger_count", y="fare_amount", data= df_train,
palette="Set2",ax=x[0][0])
sns.scatterplot(x="month", y="fare_amount", data= df_train,
palette="Set2",ax=x[0][1])
sns.scatterplot(x="weekday", y="fare_amount", data= df_train,
palette="Set2",ax=x[1][0])
sns.scatterplot(x="hour", y="fare_amount", data= df_train,
palette="Set2",ax=x[1][1])
sns.scatterplot(x="pickup_longitude", y="fare_amount", data= df_train,
palette="Set2",ax=x[2][0])
sns.scatterplot(x="pickup_latitude", y="fare_amount", data= df_train,
palette="Set2",ax=x[2][1])
sns.scatterplot(x="pickup_latitude", y="pickup_longitude", data=
df_train, palette="Set2",ax=x[3][0])
sns.scatterplot(x="dropoff_latitude", y="dropoff_longitude", data=
df_train, palette="Set2",ax=x[3][1])
sns.scatterplot(x="dropoff_longitude", y="fare_amount", data= df_train,
palette="Set2",ax=x[4][0])
sns.scatterplot(x="dropoff_latitude", y="fare_amount", data= df_train,
palette="Set2",ax=x[4][1])
sns.scatterplot(x="day", y="fare_amount", data= df_train,
palette="Set2",ax=x[5][0])
```

```
# In[25]:


############################ Feature Selection
##############################
##Correlation analysis
#Correlation plot
def Correlation(df):
    df_corr = df.loc[:,df.columns]
    sns.set()
    plt.figure(figsize=(9, 9))
    corr = df_corr.corr()
    sns.heatmap(corr, annot= True,fmt = " .3f", linewidths = 0.5,
            square=True)

Correlation(df_train)
Correlation(df_test)


# In[27]:


## Splitting DataSets######
train,test = train_test_split(df_train, test_size = 0.2,random_state =
123 )
X_train = train.loc[:,train.columns != 'fare_amount']
Y_train = train['fare_amount']
X_test = test.loc[:,test.columns != 'fare_amount']
Y_test = test['fare_amount']

print(train.head(5))
print(test.head(5))


# In[28]:


############################ Feature Scaling
##############################
# #Normalisation
def Normalisation(df):
    for i in df.columns:
        print(i)
        df[i] = (df[i] - df[i].min())/(df[i].max() - df[i].min())

Normalisation(X_train)
Normalisation(X_test)

print(X_train.head(5))
print(X_test.head(5))


# In[29]:


#Calculate MAPE
```

```
def MAPE(y_true, y_pred):
    MAE = np.mean(np.abs((y_true - y_pred)))
    mape = np.mean(np.abs((y_true - y_pred) / y_true))
    print("MAPE is: ",mape)
    print("MAE is: ",MAE)
    return mape

def RMSE(y_test,y_predict):
    mse = np.mean((y_test-y_predict)**2)
    print("Mean Square : ",mse)
    rmse=np.sqrt(mse)
    print("Root Mean Square : ",rmse)
    return rmse


# In[31]:


dt_lnr_model = sm.OLS(train.iloc[:,0],train.iloc[:,1:]).fit()

#Summary of model
print(dt_lnr_model.summary())

#predict the  model

dt_predict_LR = dt_lnr_model.predict(test.iloc[:,1:])



MAPE(test.iloc[:,0], dt_predict_LR)
RMSE(test.iloc[:,0], dt_predict_LR)

#MAPE is:  0.4002251684131252
#MAE is:  2.9476151408555484
#Mean Square :  14.171074990188412
#Root Mean Square :  3.76444882953513


# In[33]:


###### KNN Modelling ########

KNN_model = KNeighborsRegressor(n_neighbors= 40).fit(X_train , Y_train)
KNN_pred= KNN_model.predict(X_test)


df_results = pd.DataFrame({'actual': Y_test, 'pred': KNN_pred})
print(df_results.head())

MAPE(Y_test, KNN_pred)
RMSE(Y_test, KNN_pred)

#MAPE is:  0.3084418890595272
#MAE is:  2.470047013144009
#Mean Square :  10.781339613948111
#Root Mean Square :  3.283495030291368
```

```
# In[34]:


################################################Decision
Tree#################################################
dt_model = DecisionTreeRegressor(random_state=123).fit(train.iloc[:,1:],
train.iloc[:,0])

print(dt_model)

dt_predictions = dt_model.predict(test.iloc[:,1:])


df_results = pd.DataFrame({'actual': test.iloc[:,0], 'pred':
dt_predictions})
df_results.head()


MAPE(test.iloc[:,0], dt_predictions)
RMSE(test.iloc[:,0], dt_predictions)

#MAPE is:  0.2870557144870078
#MAE is:  2.248405252730194
#Mean Square :  10.056863408315259
#Root Mean Square :  3.1712558093467105


# In[46]:


############################################Random Forest
################################################

# Set best parameters given by random search
rf.set_params(max_features = 'auto',
               max_depth =8,
               n_estimators = 500,
              bootstrap = 'True'
                )

rf.fit(train.iloc[:,1:], train.iloc[:,0])

# Use the forest's predict method on the test data
dt_rfPredictions = rf.predict(test.iloc[:,1:])

df_results = pd.DataFrame({'actual': test.iloc[:,0], 'pred':
dt_rfPredictions})
print(df_results.head())

MAPE(test.iloc[:,0], dt_rfPredictions)
RMSE(test.iloc[:,0], dt_rfPredictions)

#MAPE is:  0.2970395006180068
#MAE is:  2.1666677762006925
#Mean Square :  8.021622657590507
#Root Mean Square :  2.832246927368888
```

```
# In[40]:


###### GBR Modelling ##########


gbr_model = GradientBoostingRegressor(max_depth= 2,learning_rate =
0.1).fit(train.iloc[:,1:],train.iloc[:,0])
gbr_pred= gbr_model.predict(test.iloc[:,1:])

MAPE(test.iloc[:,0], gbr_pred)
RMSE(test.iloc[:,0], gbr_pred)

#MAPE is:  0.3024254007844994
#MAE is:  2.1996787335105425
#Mean Square :  8.13426907872889
#Root Mean Square :  2.8520640032665625


# In[41]:


df_rmse = pd.DataFrame({"rmse":[3.28,3.76,3.17,2.85,2.83],
"Model" : ['KNN Regression' ,'Linear Regression','Decision Trees',
"GBDT", "Random Forest"]})
print(df_rmse)


# In[56]:


#output for given Test data with best model obtained
dt_rfPredictions_test = rf.predict(df_test)
dt_rfPredictions_test = pd.DataFrame(dt_rfPredictions_test, columns =
["fare_amount"])
dt_rfPredictions_test.to_csv("TestdataPrediction.csv",index=False)


# In[ ]:
```