Cab Fare Prediction

Astha Goyal

Edwisor

15th July, 2019

Table of Contents

Introduction

City taxi rides paint a vibrant picture of life in the city. The millions of rides taken each month can provide insight into traffic patterns, road blockage, or large-scale events that attract many People. With ridesharing apps gaining popularity, it is increasingly important for taxi companies to provide visibility to their estimated fare and ride duration, since the competing apps provide these metrics upfront. Predicting fare and duration of a ride can help passengers decide when is the optimal time to start their commute, or help drivers decide which of two potential rides will be more profitable, for example. Furthermore, this visibility into fare will attract customers during times when ridesharing services are implementing surge pricing.

In order to predict the fare, only data which would be available at the beginning of a ride was used. This includes pickup and drop off coordinates, the date, time and number of passengers. Linear regression, decision tree and random forest models were used to predict the fare amount.

PYTHON

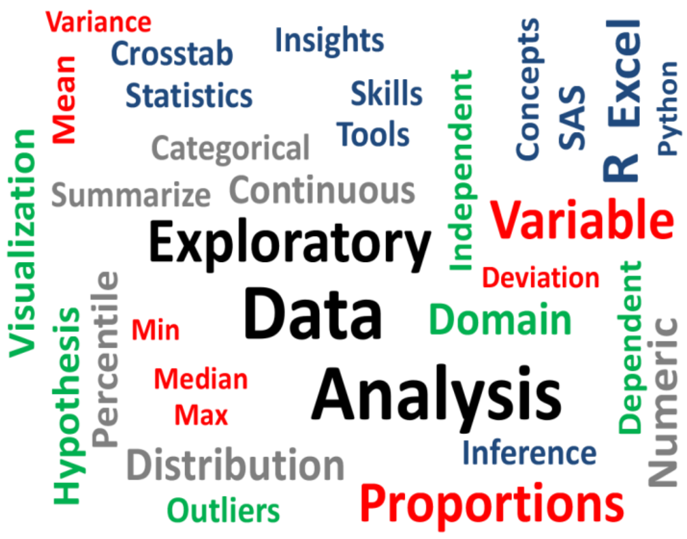
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import os


os.chdir(r"E:\MY\Project\Cab Fare Prediction")

Train = pd.read_csv("train.csv")

Test =  pd.read_csv("test.csv")
```

R

```
rm(list=ls())

setwd("E:/MY/Project/Cab Fare Prediction")

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information", "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees', "usdm", "party")

install.packages(x)

lapply(x, require, character.only = TRUE)

rm(x)


Train = read.csv("train.csv", header = T, na.strings = c(" ", "", "NA"))

Test= read.csv("test.csv", header = T, na.strings = c(" ", "", "NA"))
```

Exploratory Data Analysis

The train set consists of 16067 obser vations and 7 features containing the target. The missing percentage of “p assenger_count” is 34.23% while tha t of “fare_amoun”t is 15.55%. The te st data does not contain any missing values. There are only numeric featu res. No features seem to have substa ntial outliers. The data set is historic al dataset containing numeric featur e variables. Where, we tried to desig n a system that predicts the fare am ount for a cab ride in the city.



2.1 Data Pre Processing

1. Missing values Imputation

PYTHON

```
Train["fare_amount"] = pd.to_numeric (Train["fare_amount"],errors='coerce')

def missin_val(df): missin_val = pd.DataFrame(df.isnull().sum())

missin_val = missin_val.reset_index()

 missin_val = missin_val.rename(columns = {'index': 'Variables', 0: 'Missing_percentage'})

missin_val['Missing_percentage'] = (missin_val['Missing_percentage'] /len(df))*100

missin_val = missin_val.sort_values ('Missing_percentage', ascending = False).reset_index(drop = True)

 return(missin_val)

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

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

#Impute the missing values

Train["passenger_count"] = Train["passenger_count"].fillna(Train["passenger_count"].median())

Train["fare_amount"] = Train["fare_amount"].fillna(Train["fare_amount"].median())

print("Is there still any missing value in the training data:\n\n",Train.isnull().sum())
```

R

```
#covert factor to numeric
Train$fare_amount = as.numeric(Train$fare_amount)

#Missing Value Analysis
missing_val = data.frame(apply(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$Missing_percentage/nrow(Train)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
View(missing_val)

#Median Method
Train$fare_amount[is.na(Train$fare_amount)] = median(Train$fare_amount, na.rm = T)
Train$passenger_count[is.na(Train$passenger_count)] = median(Train$passenger_count, na.rm = T)
```

2. Data Alignment:

Python

```
#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(Train)
align_datetime(Test)

#Remove the datetime column
Train.drop('pickup_datetime', axis=1, inplace=True)
Test.drop('pickup_datetime', axis=1, inplace=True)
```

R

```
#covert Train datetime
Train$pickup_datetime1 = as.Date(Train$pickup_datetime)
Train$year = as.numeric(format(Train$pickup_datetime1, format = "%Y"))
Train$month = as.numeric(format(Train$pickup_datetime1, format = "%m"))
Train$day = as.numeric(format(Train$pickup_datetime1, format = "%d"))
Train$pickup_time <- sapply(strsplit(as.character(Train$pickup_datetime), " "), "[", 2)
Train$Hour <- sapply(strsplit(as.character(Train$pickup_time), ":"), "[", 1)
Train$pickup_datetime = NULL
Train$pickup_datetime1 =NULL
Train$pickup_time =NULL
View(Train)

#covert Test datetime

Test$pickup_datetime1 = as.Date(Test$pickup_datetime)
Test$year = as.numeric(format(Test$pickup_datetime1, format = "%Y"))
Test$month = as.numeric(format(Test$pickup_datetime1, format = "%m"))
Test$day = as.numeric(format(Test$pickup_datetime1, format = "%d"))
Test$pickup_time <- sapply(strsplit(as.character(Test$pickup_datetime), " "), "[", 2)
Test$Hour <- sapply(strsplit(as.character(Test$pickup_time), ":"), "[", 1)
Test$pickup_datetime = NULL
Test$pickup_datetime1 =NULL
Test$pickup_time =NULL
```

3. Convert data in their proper data type.

Python

```
#Setting proper data type for each columns
Train=
Train.astype({"fare_amount":float,"pickup_longitude":float,"pickup_latitude":float,"dropoff_longitude":fl
oat,"dropoff_latitude":float,"passenger_count":int,"year":int,"month":int                    ,"day"
:int,"weekday":int,"hour":int})
Train.dtypes
```

R

```
Train$Hour = as.numeric(Train$Hour)
Test$Hour = as.numeric(Test$Hour)
```

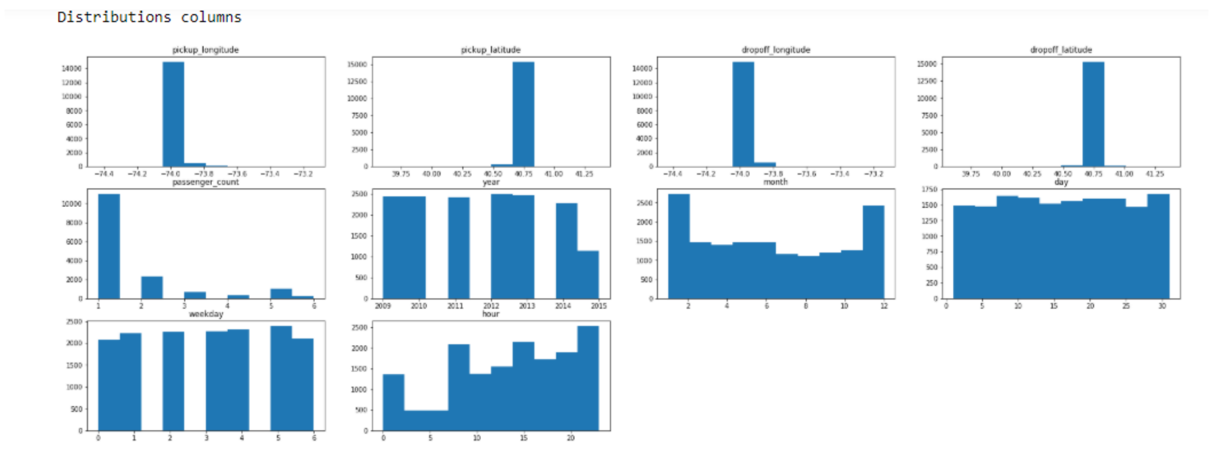
4. Individual Distribution:

Python

```
import gc
gc.collect();

print('Distributions columns')
plt.figure(figsize=(30, 185))
for i, col in enumerate(numerical_features):
    plt.subplot(50, 4, i + 1)
    plt.hist(Train[col])
    plt.title(col)
gc.collect();
```

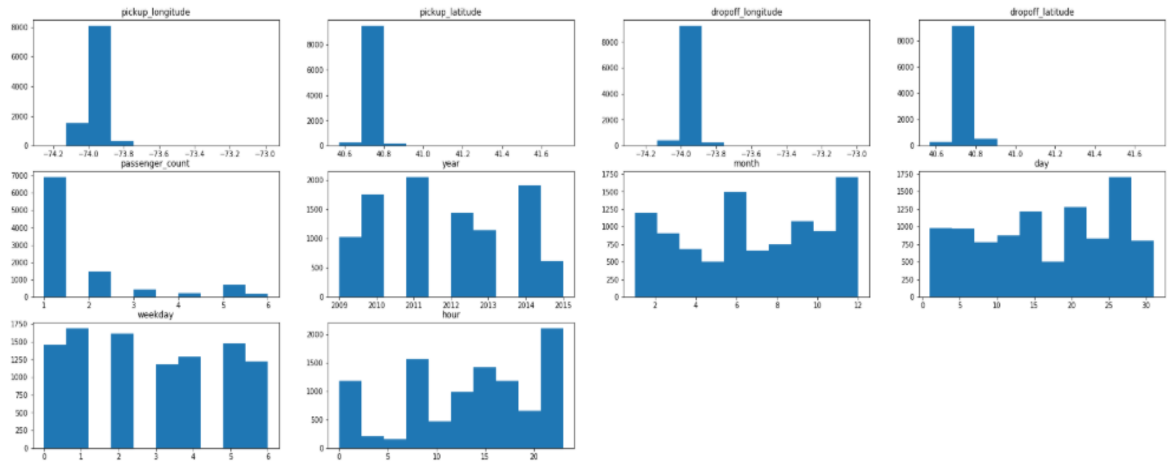
R

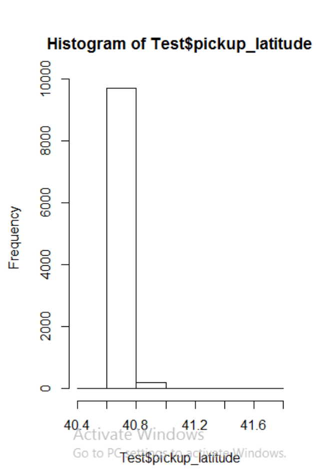
```
#Distribution of train attributes

 hist(Train$pickup_longitude, breaks=7)
 hist(Train$dropoff_longitude, breaks=7)
 hist(Train$pickup_latitude, breaks=7)
 hist(Train$dropoff_latitude, breaks=7)
 hist(Train$Hour, breaks=7)
 hist(Train$year, breaks=7)
 hist(Train$passenger_count)

 hist(Test$pickup_longitude, breaks=7)
 hist(Test$dropoff_longitude, breaks=7)
 hist(Test$pickup_latitude, breaks=7)
 hist(Test$dropoff_latitude, breaks=7)
 hist(Test$Hour, breaks=7)
 hist(Test$year, breaks=7)
 hist(Test$passenger_count)
```

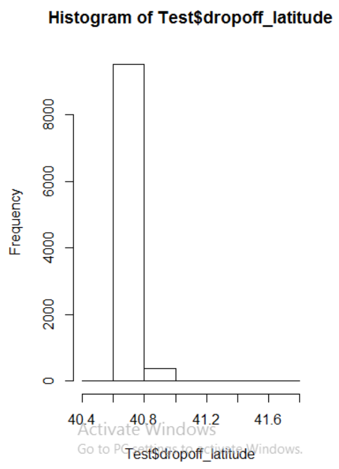


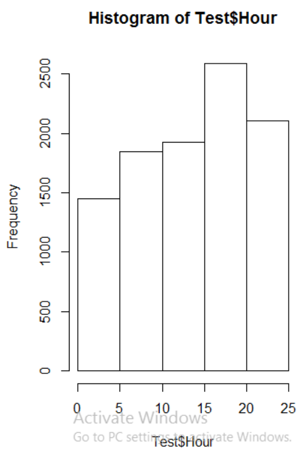
```
Distributions columns
```

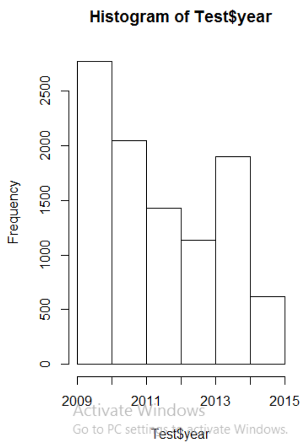
Distributions columns Test

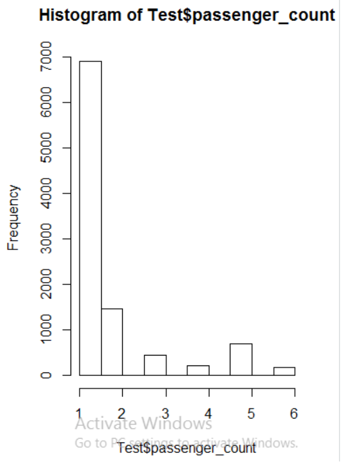


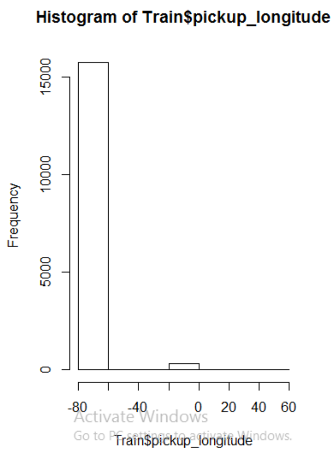


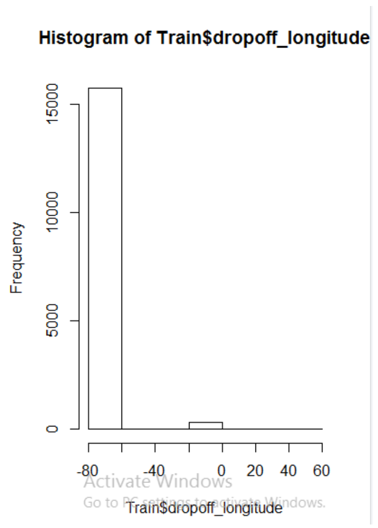


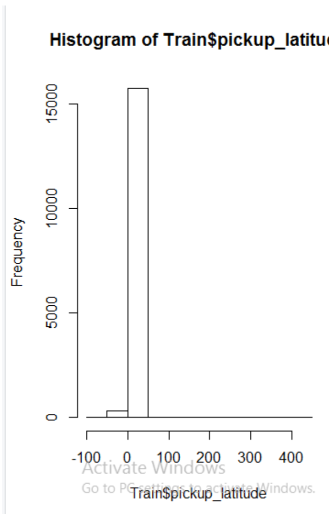


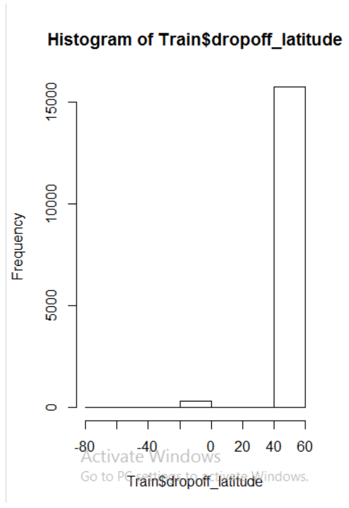


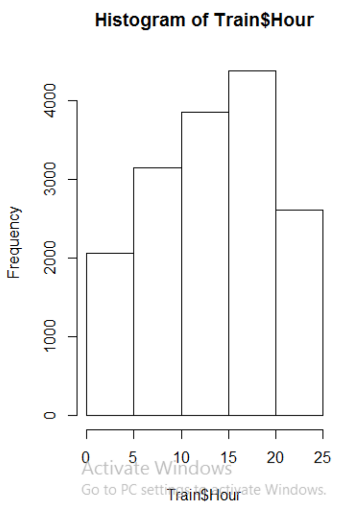


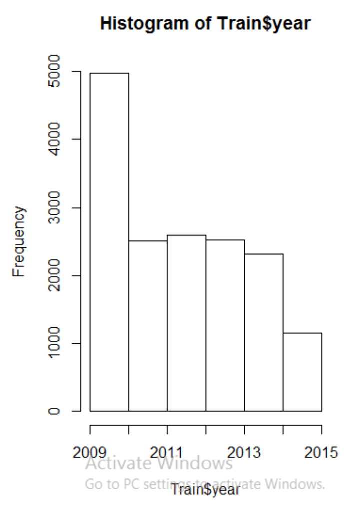


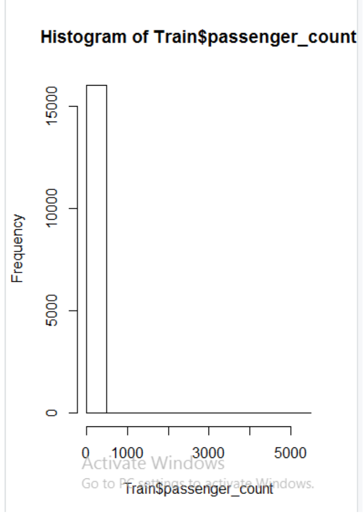

Histogram of Train$dropoff_longitude
Frequency
0
5000
10000
15000
-80
-40
0
20
40
60
Train$dropoff_longitude

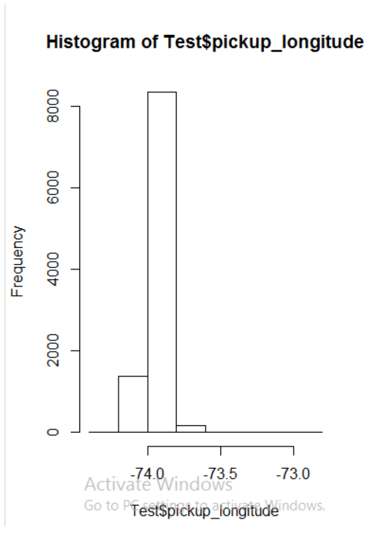

Histogram of Train$pickup_latitude
Frequency
0
5000
10000
15000
-100
0
100
200
300
400
Train$pickup_latitude

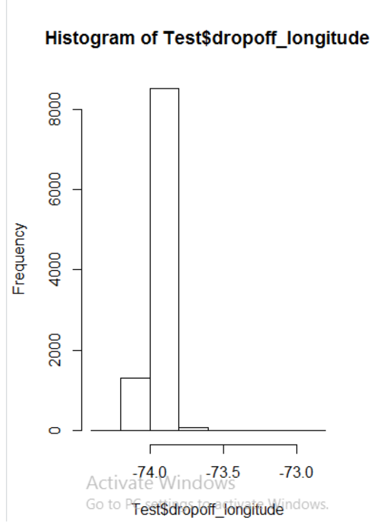

Histogram of Train$dropoff_latitude
Frequency
0
5000
10000
15000
-80
-40
0
20
40
60
Train$dropoff_latitude


Histogram of Train$Hour
Frequency
0
1000
2000
3000
4000
0
5
10
15
20
25
Train$Hour


Histogram of Train$year
Frequency
0
1000
2000
3000
4000
5000
2009
2011
2013
2015
Train$year


Histogram of Train$passenger_count
Frequency
0
5000
10000
15000
0
1000
3000
5000
Train$passenger_count

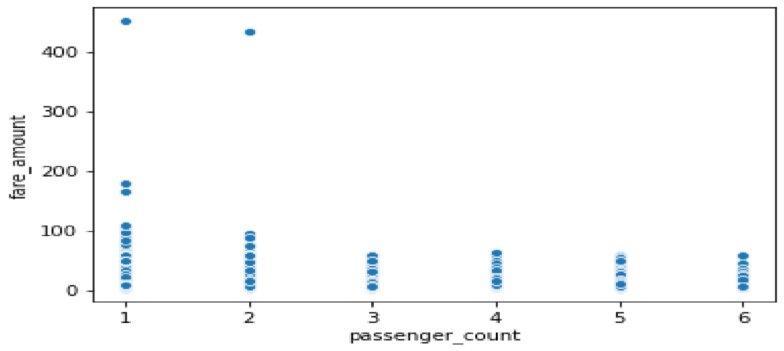

Histogram of Test$pickup_longitude
Frequency
0
2000
4000
6000
8000
-74.0
-73.5
-73.0
Test$pickup_longitude

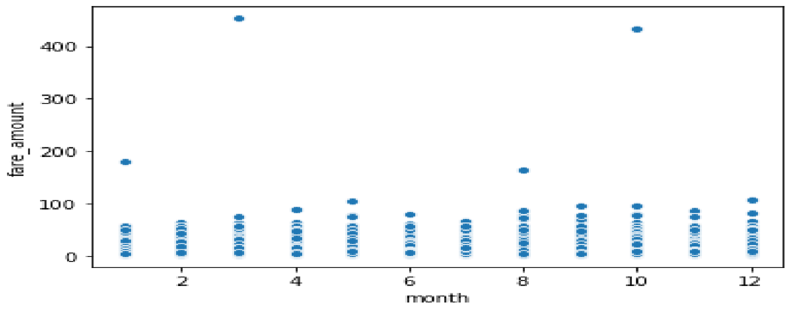

Histogram of Test$dropoff_longitude
Frequency
0
2000
4000
6000
8000
-74.0
-73.5
-73.0
Test$dropoff_longitude

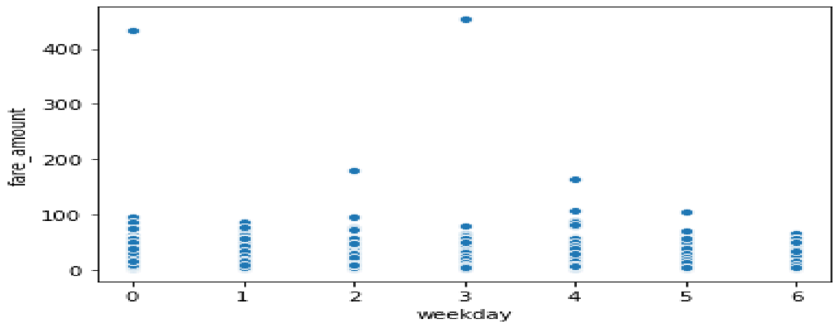
5. Scatter Plot analysis.

Python

```
#Scatter plots
cmap = sns.cubehelix_palette(dark=.3, light=.8, as_cmap=True)
sns.scatterplot(x="passenger_count", y="fare_amount", data= Train, palette="Set2")

sns.scatterplot(x="month", y="fare_amount", data= Train, palette="Set2")

sns.scatterplot(x="weekday", y="fare_amount", data= Train, palette="Set2")

sns.scatterplot(x="hour", y="fare_amount", data= Train, palette="Set2")
```







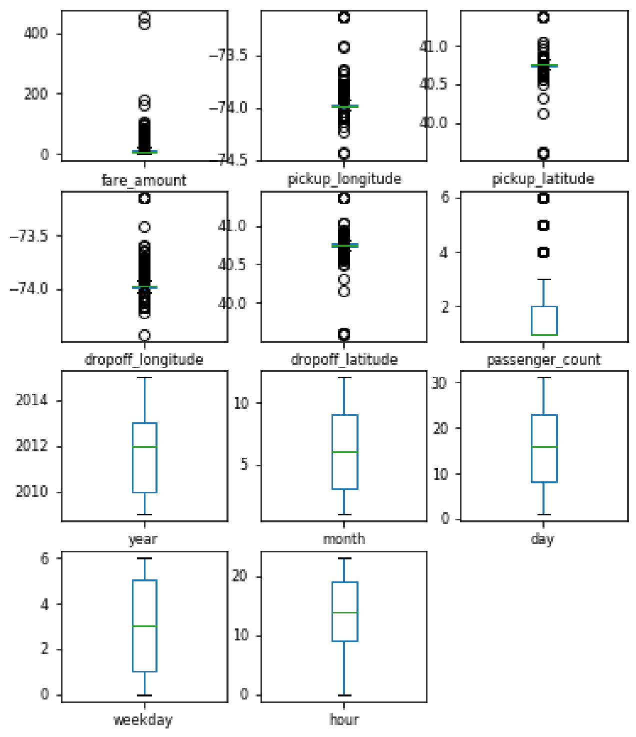
6. Outlier Analysis:

Python

```
#Outlier analysis

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()

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)

train = outliers_analysis(Train)
test = outliers_analysis(Test)
```

R

```
#remove outlier from test and Train
 for(i in cnames){
   print(i)
   val = Train[,i][Train[,i] %in% boxplot.stats(Train[,i])$out]
   #print(length(val))
   Train = Train[which(!Train[,i] %in% val),]
 }



for(i in cnames1){
  print(i)
  val = Test[,i][Test[,i] %in% boxplot.stats(Test[,i])$out]
  #print(length(val))
  Test = Test[which(!Test[,i] %in% val),]
}
```



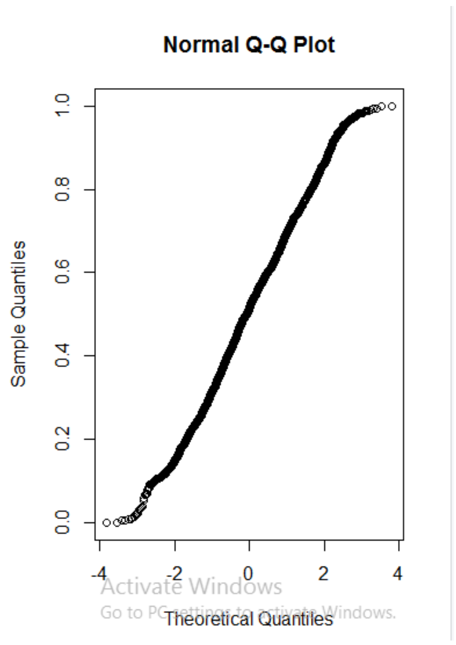
7. Feature scaling with the help of Normalisation:

Python

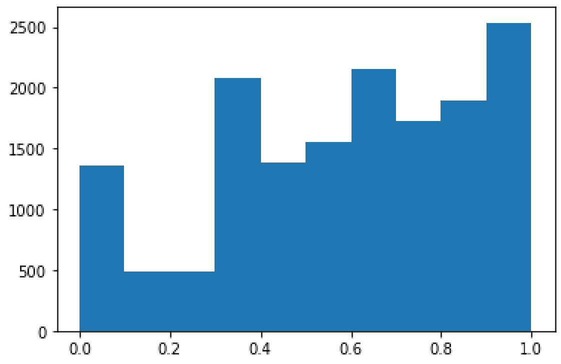
```
# #Normalisation
def Normalisation(df):
    for i in df.columns:
        df[i] = (df[i] - df[i].min())/(df[i].max() - df[i].min())

Normalisation(X_train)
Normalisation(Test)
```

R

```
#for Train
 for(i in cnames){
   print(i)
   Train[,i] = (Train[,i] - min(Train[,i]))/(max(Train[,i] - min(Train[,i])))
 }

 #for Test
 for(i in cnames1){
   print(i)
   Test[,i] = (Test[,i] - min(Test[,i]))/(max(Test[,i] - min(Test[,i])))
 }
```



```
<a list of 10 Patch objects>)
```



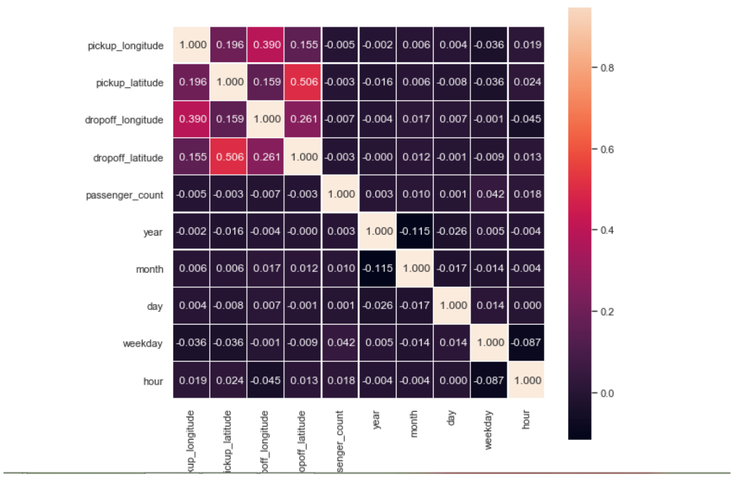
8. Feature selection using correlation analysis.

Python

```
#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(X_train)
Correlation(Test)
```

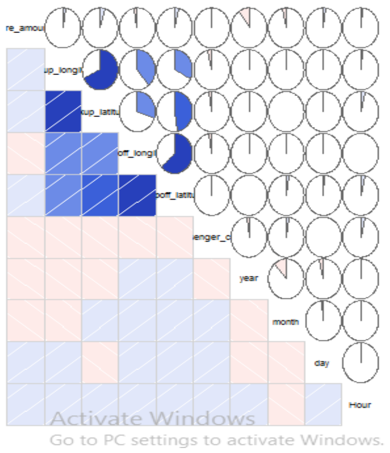
R

```
corrgram(Train[,numeric_index], order = F,
        upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot Train")


corrgram(Test[,numeric_index1], order = F,
        upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot Test")

## Dimension Reduction
Train = subset(Train, select = -c(Hour,day,month, year))
View(Train)
Test = subset(Test, select = -c(Hour,day,month, year))
View(Test)
```

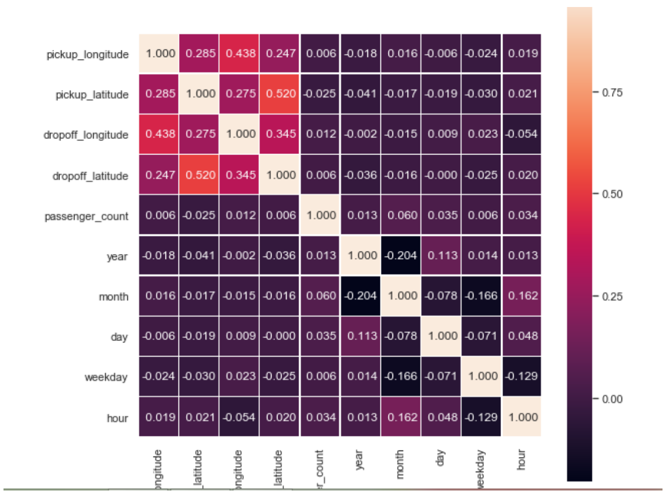


	kup_longitude	ickup_latitude	ooff_longitude	opoff_latitude	senger_count	year	month	day	weekday	hour
pickup_longitude	1.000	0.196	0.390	0.155	-0.005	-0.002	0.006	0.004	-0.036	0.019
pickup_latitude	0.196	1.000	0.159	0.506	-0.003	-0.016	0.006	-0.008	-0.036	0.024
dropoff_longitude	0.390	0.159	1.000	0.261	-0.007	-0.004	0.017	0.007	-0.001	-0.045
dropoff_latitude	0.155	0.506	0.261	1.000	-0.003	-0.000	0.012	-0.001	-0.009	0.013
passenger_count	-0.005	-0.003	-0.007	-0.003	1.000	0.003	0.010	0.001	0.042	0.018
year	-0.002	-0.016	-0.004	-0.000	0.003	1.000	-0.115	-0.026	0.005	-0.004
month	0.006	0.006	0.017	0.012	0.010	-0.115	1.000	-0.017	-0.014	-0.004
day	0.004	-0.008	0.007	-0.001	0.001	-0.026	-0.017	1.000	0.014	0.000
weekday	-0.036	-0.036	-0.001	-0.009	0.042	0.005	-0.014	0.014	1.000	-0.087
hour	0.019	0.024	-0.045	0.013	0.018	-0.004	-0.004	0.000	-0.087	1.000

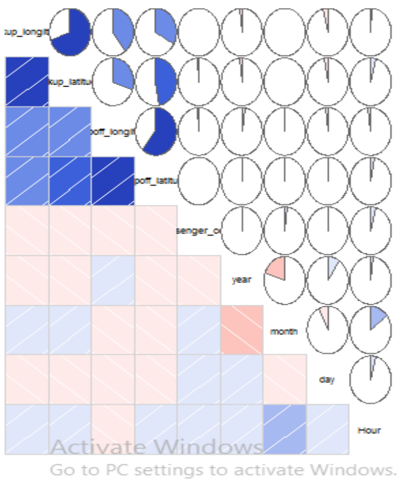
Correlation Plot



Correlation Plot Test



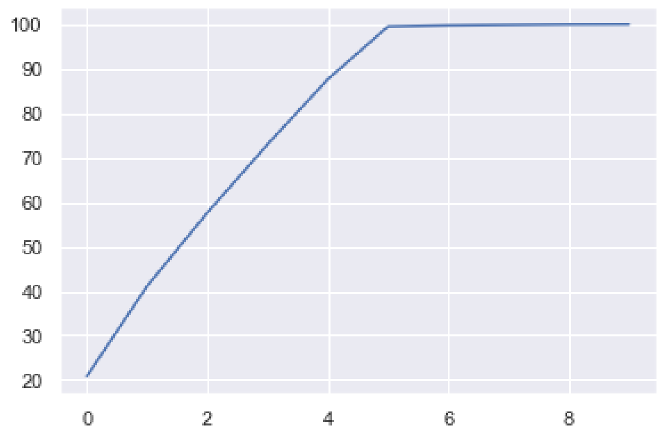
	ongitude	latitude	ongitude	latitude	r_count	year	month	day	weekday	hour
pickup_longitude	1.000	0.285	0.438	0.247	0.006	-0.018	0.016	-0.006	-0.024	0.019
pickup_latitude	0.285	1.000	0.275	0.520	-0.025	-0.041	-0.017	-0.019	-0.030	0.021
dropoff_longitude	0.438	0.275	1.000	0.345	0.012	-0.002	-0.015	0.009	0.023	-0.054
dropoff_latitude	0.247	0.520	0.345	1.000	0.006	-0.036	-0.016	-0.000	-0.025	0.020
passenger_count	0.006	-0.025	0.012	0.006	1.000	0.013	0.060	0.035	0.006	0.034
year	-0.018	-0.041	-0.002	-0.036	0.013	1.000	-0.204	0.113	0.014	0.013
month	0.016	-0.017	-0.015	-0.016	0.060	-0.204	1.000	-0.078	-0.166	0.162
day	-0.006	-0.019	0.009	-0.000	0.035	0.113	-0.078	1.000	-0.071	0.048
weekday	-0.024	-0.030	0.023	-0.025	0.006	0.014	-0.166	-0.071	1.000	-0.129
hour	0.019	0.021	-0.054	0.020	0.034	0.013	0.162	0.048	-0.129	1.000



9. Dimensionality Reduction using PCA:

Python

```
pca = PCA(n_components=10)
pca.fit(X_train)
var= pca.explained_variance_ratio_
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
plt.plot(var1)
plt.show()

#lets reduce our no. of variables to 5 as it explains 100% features of our Data
pca = PCA(n_components=5)
X = pca.fit(X_train).transform(X_train)
Test = pca.fit(Test).transform(Test)
```


100
90
80
70
60
50
40
30
20
0
2
4
6
8

Solution Methods:

We use 3 different methods to come to a solution: Logistic Regression, Decision Tree and Random Forest from Scikit Learn.

1 Logistic Regression:

Logistic regression is a relatively ’simple’ machine learning algorithm and we expect fast, but not great results. It tries to attach the best constant values to how features interact with the target, based on the train set, minimizing an error term. It then applies this same formula to the test data set. It is pursued here as a baseline in order to compare to more sophisticated models. For more details see (Bishop, 2006).

Python

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


import statsmodels.api as sm

# Train the model using the training sets
model = sm.OLS(y_train, X_train).fit()
model.summary()

predictions_LR = model.predict(X_test)

def rmse(y, y_pred):
   return np.sqrt(np.mean(np.square(y - y_pred)))

rmse(y_test, predictions_LR)
```

R

```
#############Linear Regression##################3


vif(Train[,-1])

vifcor(Train[,-1], th = 0.9)

#run regression model
lm_model = lm(fare_amount ~., data = train)

#Summary of the model
summary(lm_model)

#Predict
predictions_LR = predict(lm_model, test[,2:6])

RMSE = function(m, o){
  sqrt(mean((m - o)^2))
}

RMSE(test[,1],predictions_LR)
```

2 Decision Tree:

This section roughly describes what a decision tree is. The concept is fairly simple. If a prediction needs to be made, go from the top of the tree to the bottom. This tree is constructed by at each depth level greedily finding the best feature to split on, maximizing information gain via the Gini impurity. Let p0 be the proportion of observations that belong to class 0 and let p1 be the proportion of observations that belong to class 1 out of all observations.

$$\mathrm{GINI} = 1 - (P_0^2 + P_1^2)$$

This metric is an example of how to measure how pure a split is. It is more pure the lower it is with minimum 0 and maximum 0.5. A split is purer if in the branches the ratio of positive to negative examples is close to 0 or 1 or in other words the split is highly discriminatory. The maximum depth of this tree is set at 2 to keep it tractable and this effectively also keeps it from over fitting. A decision tree can otherwise perfectly match a training set, which does not generalize well.

PYTHON

```
###### DecisionTree Modelling ##########


from sklearn.tree import DecisionTreeRegressor
fit_DT = DecisionTreeRegressor(max_depth=2).fit(X_train, y_train)

predictions_DT = fit_DT.predict(X_test)

rmse(y_test, predictions_DT)
```

R

```
########Decion Tree#########3

fit = rpart(fare_amount ~ ., data = train, method = "anova")
predictions_DT = predict(fit, test[, -1])

#RMSE

RMSE(test[,1], predictions_DT)
```

3 Random Forest

The Random Forest model generates multiple decision trees. (Breiman, 2001) Only a subset of predictive features is now considered during each split that is randomly selected. The decision trees lead to one single prediction together by averaging all the predictions they give individually. The parameters of a Random Forest model are really important and need to be tuned appropriately. N_ESTIMATORS determines the number of trees in the forest. MAX_FEATURES controls the maximum random amount of features to consider when determining a best split during the algorithm. MAX_DEPTH limits the depth of a tree in the random forest. MIN_SAMPLES_SPLIT determines the minimum amount of observations that need to be in a node for it to be considered for splits. MIN_SAMPLES_LEAF constrains that leaf nodes have at least this number of observations. N_JOBS controls the number of processors. Trees in a random forest can be made in parallel, so the more cores working, the less computation time needed. CLASS_WEIGHT can be set to 'balanced' to deal with imbalanced datasets.

PYTHON

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

rf_model = RandomForestRegressor(max_depth= 5, n_estimators = 100).fit(X_train , y_train)
rf_pred_train = rf_model.predict(X_train)
rf_pred= rf_model.predict(X_test)

rmse(y_test, rf_pred)
```

R

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

rm_model <- randomForest(fare_amount ~., data = train)
print(rm_model)
predictions_RF = predict(rm_model, test[, -1])
RMSE(test[,1], predictions_RF)
```

Result

S.No	ALGORITHM	RMSE
1	Linear Regression	17.26
2	Decision Tree	12.87
3	Random Forest	12.83

Why RMSE?

When we fit linear regression model in time series and if in case we need to use dummy variable for observed but unpredictable spike(this is know as episodic events) then there could be possibility that model is over fit in forecast periods. It means that instead of switching off episodic events, our model also captured the error terms (this is the one of the difficult task to differentiate random error and episodic events). So in forecast, if we want to check the accuracy of the model and avoid over fit and any episodic event in future; we can square root the MSE value and due to square root, model avoids outliers (robust against extreme values) in the forecast period. As we cannot model spike in the forecast period. Also, RMSE more aggressively punishes big errors than small ones, whereas MAD/MAPE are more linear

Conclusion

Considering what is and what is not accounted for in the models built in this study, their predicting results are fairly accurate. To further improve the prediction accuracy, more variability's need to be considered and modelled. Although the rides in hour and average speed in hour work as proxies for traffic, more modelling on the effect of location is needed. These quantities could be calculated for different areas to further model local effects of traffic. Also, modelling traffic and the effect of location in between pickup and drop-off points should be considered as well as difference in drivers' speed. These further steps could be taken both by analysing larger sets of the data to infer relationships and effects of location and traffic at different times, as well as aggregation with other datasets, as data on traffic, speed limitations, etc.