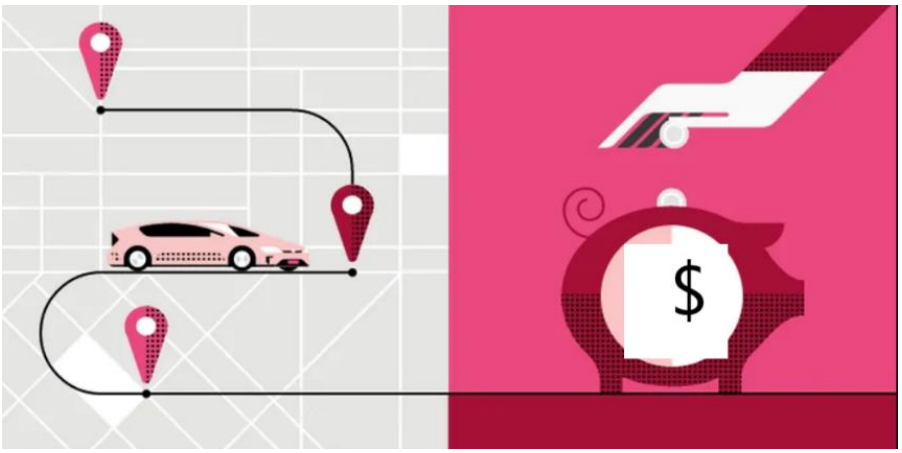
Project Report On



Cab fare Prediction

Created By :- Sushil Kumar

INDEX

CHAPTER 1

INTRODUCTION :-

1.1 Problem Statement

You are a cab rental start-up company. You have successfully run the pilot project & 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 Understanding

The Understanding of data should be the first and important step in the process of finding solution of any business problem. Here in our case our company has provided a data set with following features :-

1. pickup_datetime - timestamp value indicating when the cab ride started.
2. pickup_longitude - float for longitude coordinate of where the cab ride started.
3. pickup_latitude - float for latitude coordinate of where the cab ride started.
4. dropoff_longitude - float for longitude coordinate of where the cab ride ended.
5. dropoff_latitude - float for latitude coordinate of where the cab ride ended.
6. passenger_count - an integer indicating the number of passengers in the cab ride.
7. fare_amount - object indicating the fare for the cab ride.

Size of Dataset Provided: - 16067 rows, 7 Columns (including dependent variable)

Missing Values: Yes

Outliers Presented: Yes

CHAPTER 2

Methodology

➢ Pre-Processing

When we required to build a predictive model, we require to look and manipulate the data before we start modelling which includes multiple preprocessing steps such as exploring the data, cleaning the data as well as visualizing the data through graph and plots, all these steps is combined under one shed which is **Exploratory Data Analysis**, which includes following steps:

- Data exploration and Cleaning
- Missing values treatment
- Outlier Analysis
- Feature Selection
- Features Scaling
    - o Skewness and Log transformation
- Visualization

➢ Modelling

Once all the Pre-Processing steps has been done on our data set, we will now further move to our next step which is modelling. Modelling plays an important role to find out the good inferences from the data. Choice of models depends upon the problem statement and data set. As per our problem statement and dataset, we will try some models on our preprocessed data and post comparing the output results we will select the best suitable model for our problem. As per our data set following models need to be tested as our business problem is regression problem.

- Linear regression
- Decision Tree
- Random forest,
- Gradient Boosting

➢ Model Selection

The final step of our methodology will be the selection of the model based on the different output and results shown by different models. We have multiple parameters which we will study further in our report to test whether the model is suitable for our problem statement or not.

CHAPTER 3

Pre-Processing

3.1 Data exploration and cleaning

In the given project, we have training data set of cab fare. The data which we have is unorganized in nature so, here we need to spend more time for data understanding, data cleaning, and data visualization to figure out new features that will be the better predictors of cab fare**.**

```
In [7]: #Extract the data types of all the features in Dataset.
        df.dtypes

Out[7]: fare_amount          object
        pickup_datetime      object
        pickup_longitude    float64
        pickup_latitude     float64
        dropoff_longitude   float64
        dropoff_latitude    float64
        passenger_count     float64
        dtype: object
```

In the above Fig. we can see the data-types of the given attributes. Here we need to change our data-types because pickup-date time which is indicating when the trip started it should be in timestamp. fare_amount in dollar $ is nothing but our target variable it should be float. As we can see the target variable is not in test data set.

Below are our assumptions on the basis of our exploration of the dataset:-

a. Separate the combined variables.
b. As we know we have some negative values in fare amount so we have to remove those values.
c. Passenger count would be max 6 if it is a SUV vehicle not more than that. We have to remove the rows having passengers counts more than 6 and less than 1.
d. There are some outlier figures in the fare (like top 3 values) so we need to remove those.
e. Latitudes range from -90 to 90. Longitudes range from -180 to 180. We need to remove the rows if any latitude and longitude lies beyond the ranges

Some important features :-

1. **Pickup_datetime :-**

   As we see in the train dataset we have 6 independent and 1 target variables let's discuss one by one. **Pickup_datetime** telling us when the journey was stared like 2009-06-15 17:26:21 UTC so, what we can do is we differentiate the above attribute in year, month, day of month, hour, minute as shown below:-

```
In [16]: df.head()
```

Out[16]:

	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	pickup_year	pickup_month	pickup_date	pickup_hour	pickup_Dayofweek	pickup_minute
	40.721319	-73.841610	40.712278	1.0	2009	6	15	17	0	26
	40.711303	-73.979268	40.782004	1.0	2010	1	5	16	1	52
	40.761270	-73.991242	40.750562	2.0	2011	8	18	0	3	35
	40.733143	-73.991567	40.758092	1.0	2012	4	21	4	5	30
	40.768008	-73.956655	40.783762	1.0	2010	3	9	7	1	51

2. **Pickup and Dropout Location :-**

   In our data set pickup latitude, pickup longitude, drop-off latitude and drop-off longitude telling us the pickup and drop-off location. With the help these attributes here we can find the distance of trip using haversine formula. As we know the earth share is spherical or elliptical in nature so the 'haversine' formula help us to calculate the great-circle distance between two points that is, the shortest distance over the earth's surface distance. The formula to calculate the distance is shown below.

   $$a = \sin^2((\varphi B - \varphi A)/2) + \cos \varphi A \cdot \cos \varphi B \cdot \sin^2((\lambda B - \lambda A)/2)$$

   $$c = 2 * \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)})$$

   $$d = R \cdot c$$

   d = Haversine distance

   : Where,

   φ is latitude in radian, λ is longitude, R is earth's radius (6,371km)

The latitude and longitude point has to be entered within the range latitude: [−90, 90] longitude: [−180, 180]. But in the feature we can see latitude is 401.1 and longitude -73.9 also in some cases values of latlong is 0. It is not possible those values are nothing but outliers. So we can treat them.

3. **Fare_amount :-**

When we explore the values of fare amount we came to know few values are negative. The min value is -3$ and max 54343$ in dollar but, as we know the cost of the journey cannot be negative so it is nothing but the impure values we must filter it out from our existing data set.

4. **passenger_count :-**

In train data set we can see the min value of passenger in single trip is 0 and maximum is 5345. Practically it is not possible because cab has desire setting capacity. Suppose we have 7 sitter cabs so max capacity is 6 passengers and 1 driver so in our data set beyond 6 we will consider as an outlier. Above 6 and below1 we can filter those entries.

3.2 Missing Value and Outlier Analysis :-

Missing value analysis plays a vital role in data preparing. There are many reasons to occur missing values. In statistics while calculating missing values, if it is more than 30% we just drop the particular attribute because it does not carry much information to predict our target variables. As we can see in the below Fig. highest percentage of missing value is 0.0034 and it is negligible but, then also we impute this missing value with the help of central statistic method i.e. median.

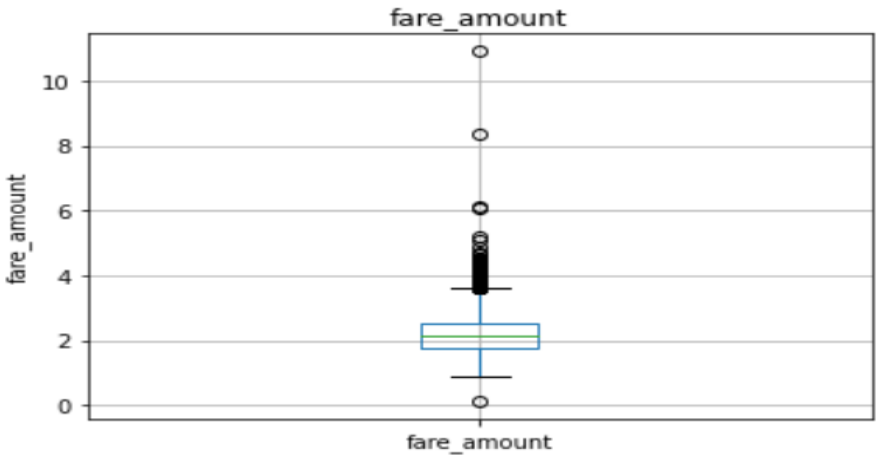
In [42]:
```
#list comprehensions is used to get the columns names which is having missing values
Missing_Val_Cols =[features for features in df.columns if df[features].isnull().sum()>1]
```

In [43]:
```
#Calculating the percentage of missing values present in the columns for above
for feature in Missing_Val_Cols :
    print(feature,"is having",np.round(df[feature].isnull().mean(),4), "% Missing Values")
```

```
fare_amount is having 0.0015 % Missing Values
passenger_count is having 0.0034 % Missing Values
```

Outliers:-

Outlier is the observation which is inconsistent related with all data set. The values of Outliers are the accurate but it is far away from the set of actual values and it heavily impact on the mean so that we consider it as an outlier. Here we have used box plot method to detect outliers as shown below Figs



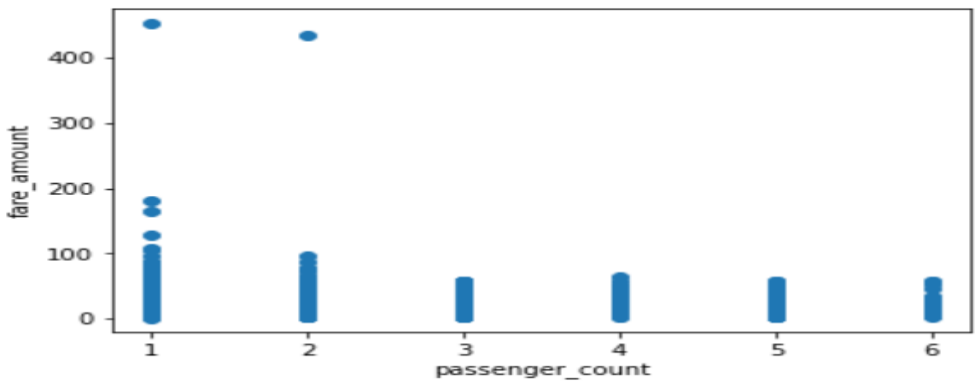
Here what we did is we remove some outliers manually as discussed in chapter 1. But in some cases we find the extreme values and if we consider the same it will impact on mean. So we must have to remove it from our train data set.

3.3 Data Visualization :-

By using visualization we can have a very clear picture of the data and we can actually get some important insights of the date ,

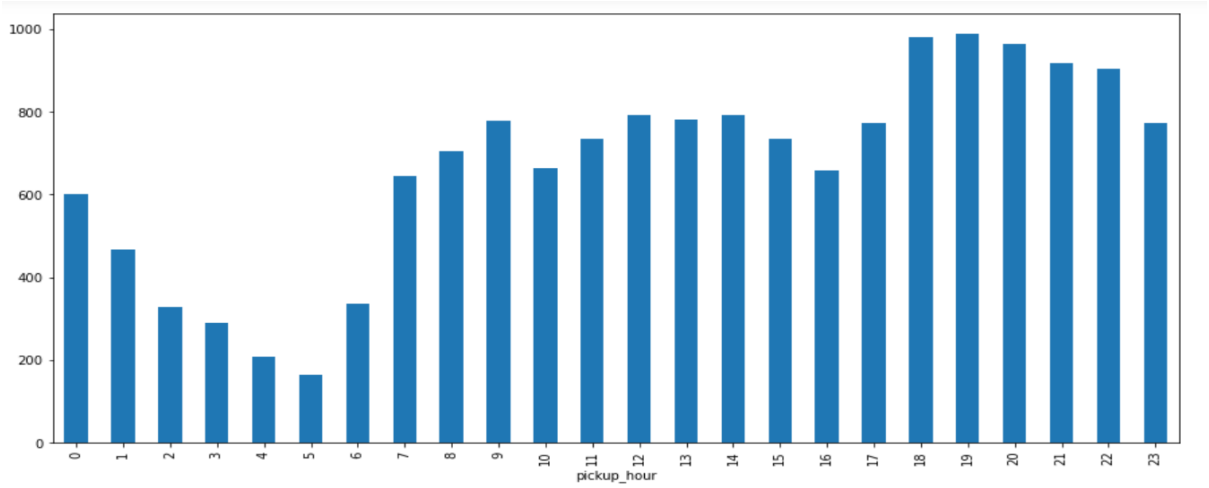
1. Affect of passenger count on fare amount :-

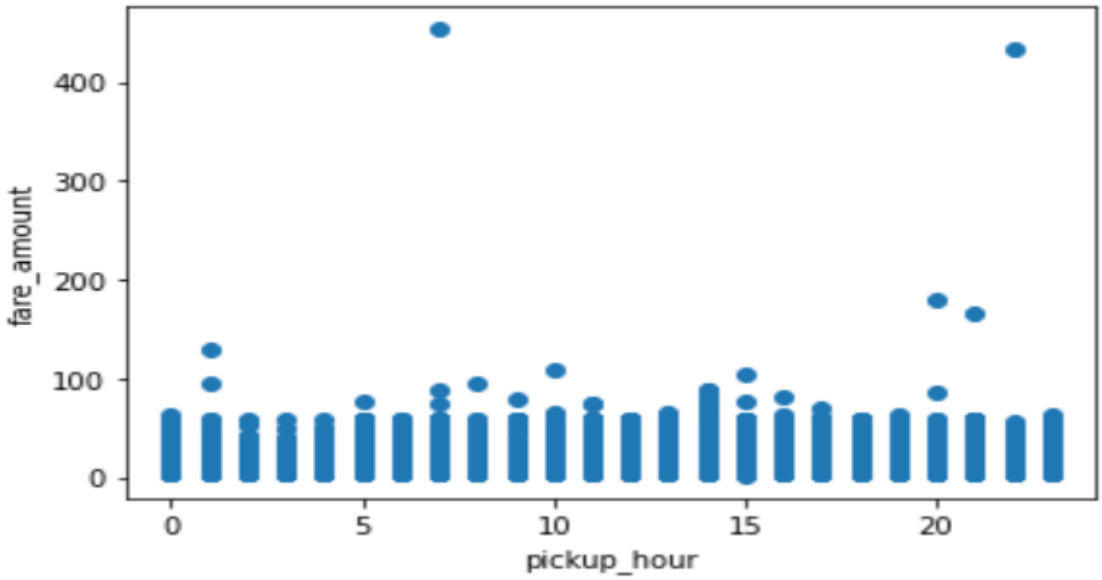
   → Single passenger frequency is High that means maximum trip were having Single passenger.
   → The highest fare also seems to come from cabs which carry 1 and 2 no of passengers.



2. Affect of time of pickup on fare amount :-

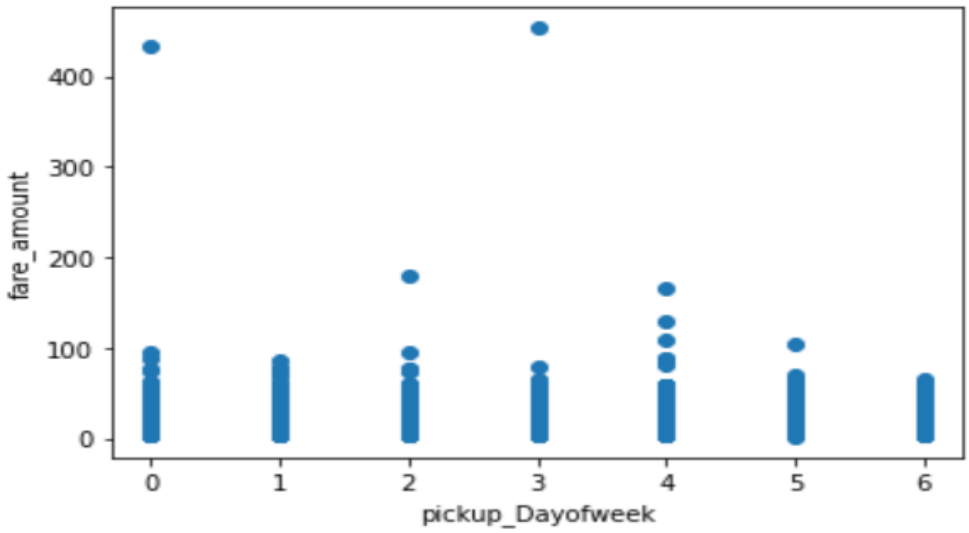
→ Maximum trips were in between 6pm to 10pm (Frequency of trips).
→ Minimum trips were at 5am.
→ The fares, however, seem to be high between 7AM and 10AM, between 2PM to 4PM and 8PM-9PM





3. Affect of day of the week on fare amount :-

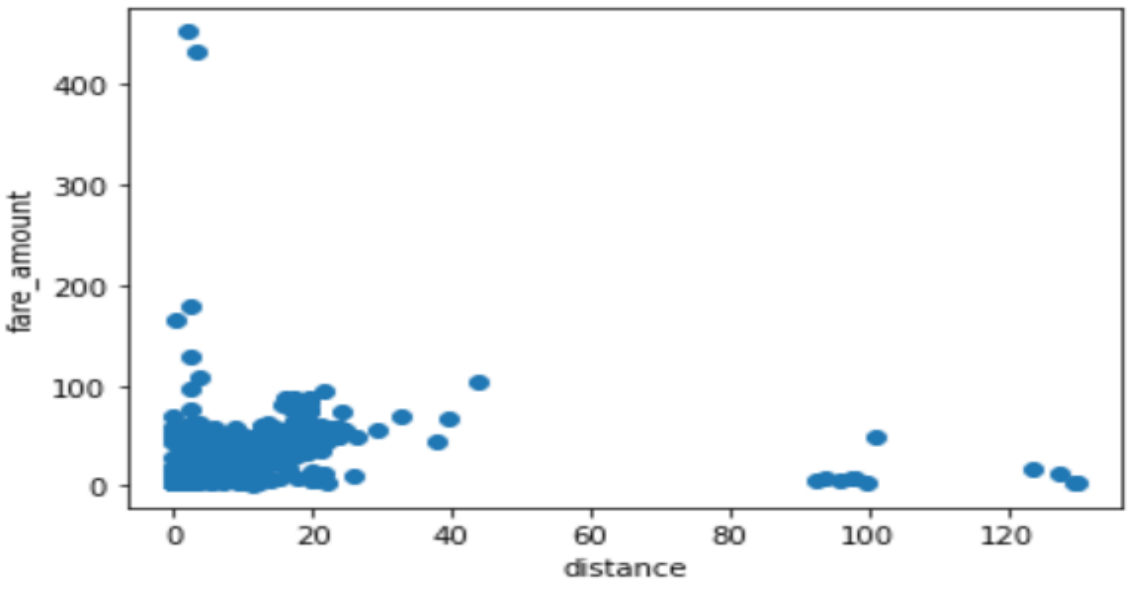
→ The frequency is uniform throughout the week.
→The highest fares seem to be on a Sunday and Monday, and the lowest on Wednesday and Saturday.



4. Affect of distance on fare amount :-

→ Maximum trips were in between 0.1-20KM (Frequency of distancen of trips).

➔ We can see in the graph, there is a linear co-relation between distance and fare amount,as distance will increace the fare will also increase.



3.4 Feature Selection :-

Now as we know that all below variables are of no use so we will drop the redundant variables:

- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude

Now only following variables we will use for further steps:-

	fare_amount	passenger_count	pickup_year	pickup_month	pickup_date	pickup_hour	pickup_Dayofweek	pickup_minute	distance
0	1.704748	1	2009	6	15	17	0	26	0.708412
1	2.884801	1	2010	1	5	16	1	52	2.246029
2	1.902108	2	2011	8	18	0	3	35	0.871095
3	2.163323	1	2012	4	21	4	5	30	1.334809
4	1.840550	1	2010	3	9	7	1	51	1.098331

Independent Variables :-

passenger_count, pickup_year, pickup_month, pickup_date, pickup_DayofWeek, pickup_hour, distance, pickup_minute

Dependent Variable :- fare_amount

Dividing the variables into two categories basis on their data types:-

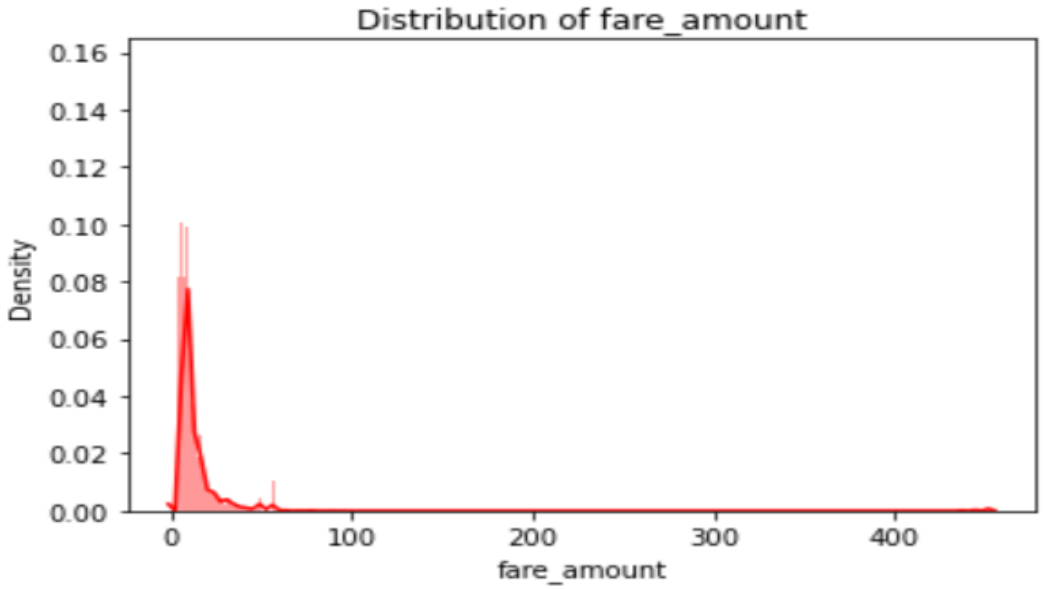
Continuous variables - 'fare_amount', 'distance'.

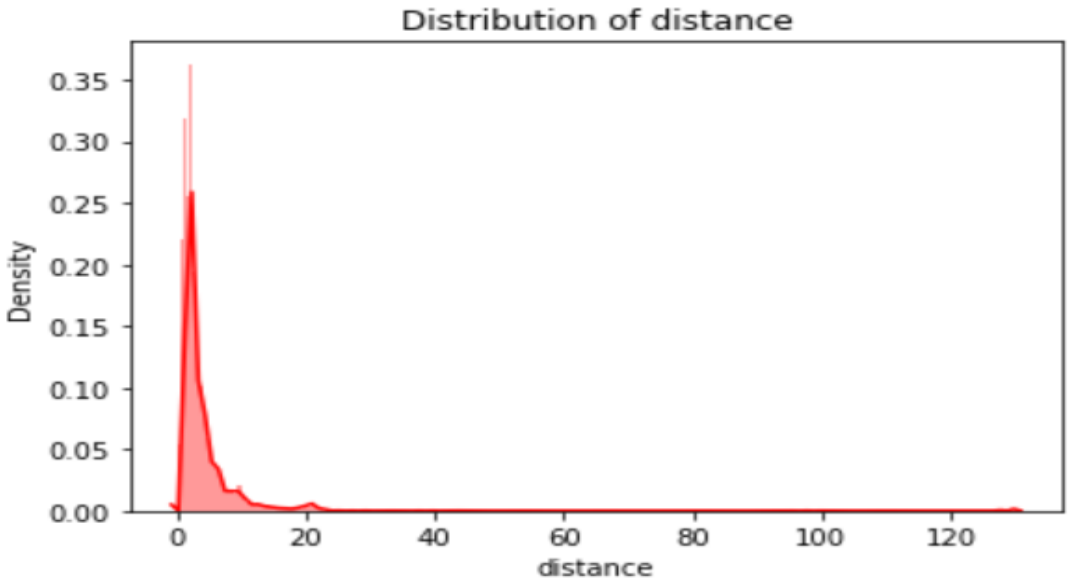
Categorical Variables - passenger_count, pickup_year, pickup_month, pickup_date, pickup_DayofWeek, pickup_hour, pickup_minute.

3.5 Feature Scaling:-

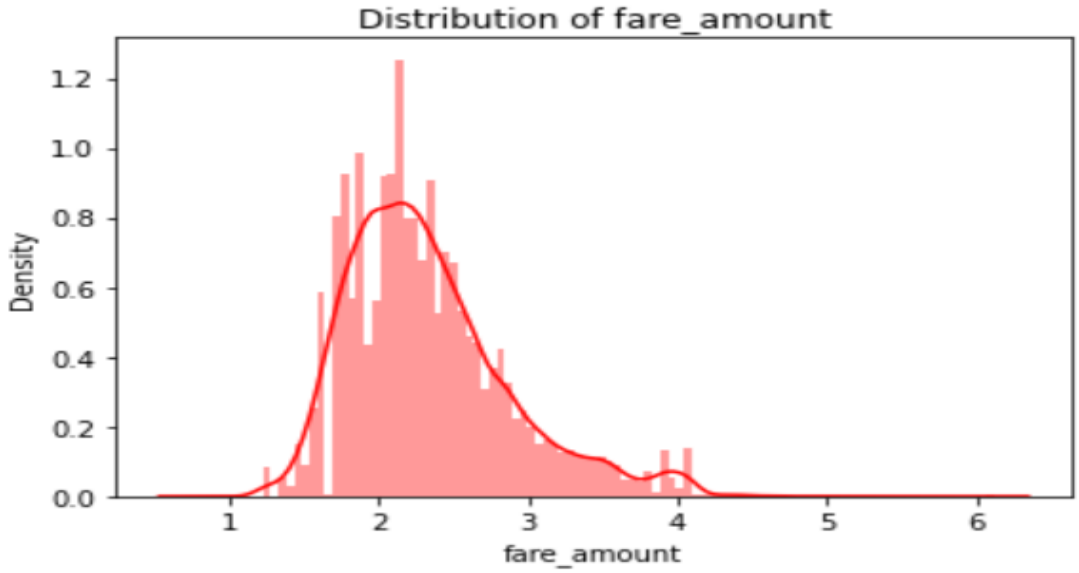
Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find that our target variable absenteeism in hours having is one sided skewed so by using **log transform** technique we tried to reduce the skewness of the same.

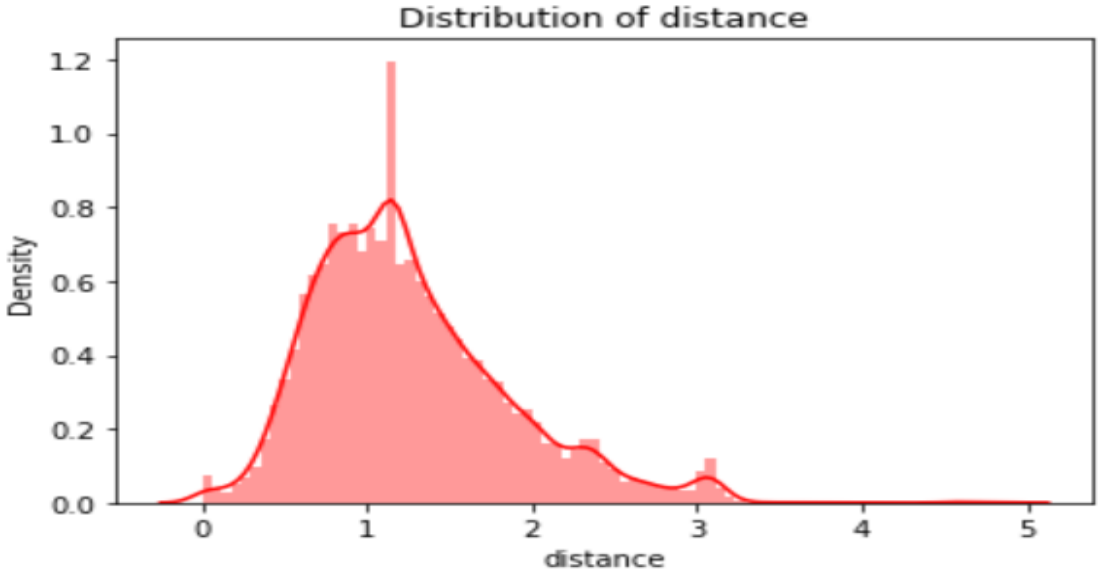
Below mentioned graphs shows the probability distribution plot to check distribution before log transformation:-





Below mentioned graphs shows the probability distribution plot to check distribution after log transformation:-





As our continuous variables appears to be normally distributed so we don't need to use feature scaling techniques like normalization and standardization for the same.

CHAPTER 4

Modelling:-

After data cleaning and exploratory data analysis phase, we finally arrived at the model building phase. In this chapter we will apply multiple machine learning algorithm to predict the test case. In cab fare prediction project our target variable i.e. fare amount is numeric (predicting and forecasting type of problem) so that here we are using regression models on structure data to predict test case.

The next step is to differentiate the train data into 2 parts i.e. train and test. The splitting of train data into 2 parts is very important factor to verify the model performance and to understand the problem of over-fitting and under-fitting. Over-fitting is the term where training error is low and testing error is high and under-fitting is the term where both training and testing error is high. Those are the common problem of complex model.

In this analysis, since we are predicting fare amount which is the numeric variable. So, we come to know that, our problem statement is predicting (forecasting) type. So, what we can do is we will apply supervise machine learning algorithms to predict our target variable. As we know our target variable is continuous in nature so, here we will build regression matrix model.
Root Mean Square Error (RMSE) to measures how much **error** there is between two data sets. In other words, it compares a predicted value and an observed or known value. The RMSE is directly interpretable in terms of measurement units, and so is a better measure of goodness of fit. So, in our case any model we build should have lower value of an **RMSE** and higher value of variance i.e**. R square.**

Before running any model, we will split our data into two parts which is train and test data. Here in our case we have taken 80% of the data as our train data. Below is the snipped image of the split of train test.

```
#train and test split , where 80% will be our train data and 20% is test set
X = df.iloc[:,1:]
Y = df['fare_amount']
X_train, X_test,y_train,y_test = train_test_split(X,Y,test_size = 0.2)
```

4.1 Linear Regression:-

Linear Regression is one of the statistical methods of prediction. It is used to find a linear relationship between the target and one or more predictors. It means the target variables should be continuous in nature. The main idea is to identify a line that best fits the data. To build any model we have some assumptions to put on data and model. This algorithm is not very flexible, and has a very high bias. Below we calculated RMSE and R^2 values using liner regression

Linear Regression

```
In [95]: #Model creation
         model_Linear = sm.OLS(y_train, X_train).fit()

In [96]: #prediction on train data
         prediction_train_OLS = model_Linear.predict(X_train)

In [97]: #prediction on test data
         prediction_test_OLS = model_Linear.predict(X_test)

In [98]: # R^2 for train data
         r2_score(y_train,prediction_train_OLS)

Out[98]: 0.7017848923676313

In [99]: ## R^2 for test data
         r2_score(y_test,prediction_test_OLS)

Out[99]: 0.7254010475435706

In [100]: # RMSE for train data
          RMSE_train_OLS = np.sqrt(mean_squared_error(y_train, prediction_train_OLS))

In [101]: RMSE_train_OLS

Out[101]: 0.2974607873184781

In [102]: # RMSE for test data
          RMSE_test_OLS = np.sqrt(mean_squared_error(y_test, prediction_test_OLS))

In [103]: RMSE_test_OLS

Out[103]: 0.2948419503146429
```

	Score
RMSE Train	0.2974607873184781
RMSE Test	0.2948419503146429
R^2 Test	0.7254010475435706

4.2 Random Forest :-

Random forest is the collection of multiple decision trees. In Random Forest, output is the average prediction by each of these trees. For this method to work, the baseline models must have a lower bias. The idea behind Random Forest is to build n number of trees to have more accuracy in dataset. Random Forest uses bagging method for predictions. It can handle large no of independent variables without variable deletion.

Random Forest Model :-

```
In [104]: #Model creation
          model_RF = RandomForestRegressor(n_estimators = 500).fit(X_train,y_train)

In [105]: #prediction on train data
          prediction_train_RF = model_RF.predict(X_train)

In [106]: #prediction on test data
          prediction_test_RF = model_RF.predict(X_test)

In [107]: # R^2 for train data
          r2_score(y_train, prediction_train_RF)

Out[107]: 0.9653160011792044

In [108]: # R^2 for train data
          r2_score(y_test, prediction_test_RF)

Out[108]: 0.7722229429167504

In [109]: # RMSE for train data
          RMSE_train_RF = np.sqrt(mean_squared_error(y_train, prediction_train_RF))

In [110]: RMSE_train_RF

Out[110]: 0.1014447284452336

In [111]: # RMSE for test data
          RMSE_test_RF = np.sqrt(mean_squared_error(y_test, prediction_test_RF))

In [112]: RMSE_test_RF

Out[112]: 0.26853124536206335
```

	Score
RMSE Train	0.1014447284452336
RMSE Test	0.26853124536206335
R^2 Test	0.7722229429167504

4.3 Decision tree :-

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

Decision tree

```
In [113]: #Model creation
          model_DT = DecisionTreeRegressor(max_depth=2).fit(X_train,y_train)

In [114]: #prediction on train data
          prediction_train_DT = model_DT.predict(X_train)

In [115]: #prediction on test data
          prediction_test_DT = model_DT.predict(X_test)

In [116]: # R^2 for train data
          r2_score(y_train, prediction_train_DT)

Out[116]: 0.6689488270768565

In [117]: # R^2 for test data
          r2_score(y_test, prediction_test_DT)

Out[117]: 0.6841042396184283

In [118]: #calculating RMSE for train data
          RMSE_train_DT = np.sqrt(mean_squared_error(y_train, prediction_train_DT))

In [119]: RMSE_train_DT

Out[119]: 0.31340972474800116

In [120]: #calculating RMSE for test data
          RMSE_test_DT = np.sqrt(mean_squared_error(y_test, prediction_test_DT))

In [121]: RMSE_test_DT

Out[121]: 0.3162363115084486
```

	Score
RMSE Train	0.31340972474800116
RMSE Test	0.3162363115084486
R^2 Test	0.6841042396184283

4.4 Gradient Boosting :-

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 generalizes them by allowing optimization of an arbitrary differentiable loss function.

Below is a screenshot of the model we build and its output:

Gradient Boosting :-

```
In [122]: #Model creation
          model_GB = GradientBoostingRegressor().fit(X_train,y_train)

In [123]: #prediction on train data
          prediction_train_GB = model_GB.predict(X_train)

In [124]: #prediction on test data
          prediction_test_GB = model_GB.predict(X_test)

In [125]: # R^2 for train data
          r2_score(y_train,prediction_train_GB)
Out[125]: 0.7898874413272966

In [126]: # R^2 for train data
          r2_score(y_test,prediction_test_GB)
Out[126]: 0.7909996476043311

In [127]: # RMSE for train data
          RMSE_train_GB = np.sqrt(mean_squared_error(y_train, prediction_train_GB))

In [128]: RMSE_train_GB
Out[128]: 0.24968411214002542

In [129]: # RMSE for test data
          RMSE_test_GB = np.sqrt(mean_squared_error(y_test, prediction_test_GB))

In [130]: RMSE_test_GB
Out[130]: 0.25722510099438073
```

	Score
RMSE Train	0.24968411214002542
RMSE Test	0.25722510099438073
R^2 Test	0.7909996476043311

CHAPTER 5

Conclusion

In above chapters we applied multiple preprocessing to frame our data into the structural format and different machine learning algorithm to check the performance of model. In this chapter we finalize one of them.

5.1 Model Evaluation

Above model help us to calculate the **Root Mean Square Error (RMSE)** and **R-Squared** Values. RMSE is the standard deviation of the 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. RMSE is an absolute measure of fit. R-squared is a relative measure of fit. R-squared is basically explains the degree to which input variable explain the variation of the output. In simple words R-squared tells how much variance of dependent variable explained by the independent variable. It is a measure if goodness of fit in regression line. Value of R-squared is between 0-1, where 0 means independent variable unable to explain the target variable and 1 means target variable is completely explained by the independent variable. So, Lower values of RMSE and higher value of R-Squared Value indicate better fit of model.

Model Name	RMSE		R Squared	
	Train	Test	Train	Test
`Linear Regression	0.29	0.29	0.70	0.72
Decision Tree	0.31	0.31	0.66	0.68
Random Forest model	0.10	0.26	0.96	0.77
Gradient Boosting	0.24	0.25	0.78	0.79

5.2 Model Selection

On the basis RMSE and R Squared results a good model should have least RMSE and max R Squared value. So, from above tables we can see:-

As we observed on all the model performance the **Gradient Boosting** gives us better outcomes as compare to other model. The RMSE of **Gradient Boosting** is minimum and the value of R^2 are also acceptable. The values difference of RMSE for train and test data is vary less so there is no any problem of model over-fitting. So, here we select **Gradient Boosting** is our final model for our given problem statement .

I have applied the same on test case data to predict fare amount. Results that I found are attached with my submissions.

Appendix

1. Python code is attached separately

2. R Code :-

```
rm(list=ls())

# Lode directory

setwd("C:/Users/User/Desktop/Project 3") getwd()

# loading Libraries

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest",
      "e1071","geosphere", "DataCombine", "pROC", "doSNOW", "class",
      "readxl","ROSE","dplyr", "plyr", "reshape","xlsx",
"pbapply", "unbalanced", "dummies", "MASS" , "gbm" ,"Information", "rpart",
      "tidyr", "miscTools")

# install.packages
#lapply(x, install.packages)

#load libraries
lapply(x, require, character.only = TRUE) rm(x)

#Loding the file
cab_train = data.frame(read.csv('train_cab.csv'))

#structure of data or data types
str(cab_train)

#Summary of data
summary(cab_train)

# Changing datatypes
cab_train$fare_amount=as.numeric(cab_train$fare_amount)

# split datetime into seprate attribute
      library(lubridate)
      library(dplyr)

cab_train$Date = as.Date(cab_train$pickup_datetime)
cab_train$month = month(cab_train$Date)
cab_train$year = year(cab_train$Date)
cab_train$day = day(cab_train$Date)
cab_train$hour = hour(cab_train$pickup_datetime)
```

```
#Now we can drop the column pickup_datetime and Date
cab_train = subset(cab_train, select = -c(pickup_datetime,Date))

 str(cab_train)

# calulating distance between latitude/longitude using the Haversine
    formula

lat1 = cab_train['pickup_latitude']
lat2 = cab_train['dropoff_latitude']
long1 = cab_train['pickup_longitude']
long2 = cab_train['dropoff_longitude']

##### Function to calculate distance ######

gcd_hf = function(long1, lat1, long2, lat2) {
 R = 6371.145 # radius of earth in KM
delta.long = (long2 - long1)
delta.lat = (lat2 - lat1)
a = sin(delta.lat/2)^2 + cos(lat1) * cos(lat2) * sin(delta.long/2)^2
c = 2 * atan2(sqrt(a),sqrt(1-a))
d = R * c
return(d) # Distance in km
}

#################### Let's apply to all variables ###############

for (i in 1:nrow(cab_train))
{
cab_train$Trip_distance_KM[i]= gcd_hf(cab_train$pickup_longitude[i],
    cab_train$pickup_latitude[i], cab_train$dropoff_longitude[i],
cab_train$dropoff_latitude[i])
}

############ Lets drop the columns of latitude/longitude.#######

    cab_train = subset(cab_train, select = -
    c(pickup_latitude,dropoff_latitude,pickup_longitude,dropoff_longitude))

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

missing_val =
data.frame(apply(cab_train,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"

#Calculating percentage missing value
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(cab_train)) *
```

```
# Sorting missing_val in Descending order
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL

# Reordering columns
missing_val =

missing_val[,c(2,1)]

missing_val

# Missing value plot
ggplot(data = missing_val[1:100,], aes(x=reorder(Columns, -
Missing_percentage),y = Missing_percentage))+
  geom_bar(stat = "identity",fill =
  "red")+xlab("Variables")+ ggtitle("Missing data
  percentage") + theme_bw()

# Median Method
cab_train$passenger_count[is.na(cab_train$passenger_c
ount)] = median(cab_train$passenger_count, na.rm = T)

sum(is.na(cab_train))

# as we know fare amount cannot be -ve
cab_train$fare_amount[cab_train$fare_amount<=0
] = NA
cab_train$fare_amount[cab_train$fare_amount>45
3] =NA


sum(is.na(cab_train))

 #removing passengers count more than 6
cab_train$passenger_count[cab_train$passenger_count<
1] = NA

 cab_train$passenger_count[cab_train$passenger_count
>6] =NA


sum(is.na(cab_train))

cab_train$passenger_count[is.na(cab_train$passenger_c
ount)] = median(cab_train$passenger_count, na.rm = T)

sum(is.na(cab_train))

summary(cab_train)
```

 # removing outliers in distance

```
str(cab_train)

cab_train$Trip_distance_KM[cab_train$Trip_distance_KM <=
0] = NA
cab_train$Trip_distance_KM[cab_train$Trip_distance_KM >
150] = NA

sum(is.na(cab_train))

cab_train$Trip_distance_KM[is.na(cab_train$Trip_distance
_KM)] = median(cab_train$Trip_distance_KM, na.rm = T)

sum(is.na(ca

b_train))

summary(ca

b_train)
```

 # Here we manually apply all preprocessing techniques i.e. (missing value, outliers, feature selection) on the top of our train data. So what we can do is we directly normalize our data

####### Feature Scaling ##########

```
cname = c('Trip_distance_KM')

catname = c( 'month', 'year' , 'hour' , 'passenger_count', 'day')
```

 # data visualisation to check normality

```
hist(cab_train$Trip_distance_KM,col="Red",main="Histogram of Trip

distance")
```

 # Here our data is not normally distributed so here we go for

Normalization

```
for(i in cname)
{
  print(i)
  cab_train[,i] = (cab_train[,i] - min(cab_train[,i]))/(max(cab_train[,i])-min(cab_train[,i]))
}
```

#Creating dummy variables for categorical variables

```
library(mlr)
df = dummy.data.frame(cab_train, catname)
```

#pf = df

#Clean the Environment-

```
rmExcept("df")
```

 # Divide data into train and test using stratified sampling method

```
library(caret)
set.seed(123)

train_index = createDataPartition(df$fare_amount, p = 0.8, list =
FALSE)
train = df[train_index,]
test  = df[-train_index,]
```

########## **Applying Supervises Machine Learning** ####################

##_____**Random Forest**_____##

```
set.seed(140)
library(randomForest) # Lode library for random forest.

library(inTrees)      # Lode library for intree transformation
```

#Develop Model on training data

```
fit_RF = randomForest(fare_amount~., data = train)
```

#Lets predict for training data

```
RF_train = predict(fit_RF, train)
RF_test = predict(fit_RF,test)
```

########################## **Error metrics to calculation** ################

```
print(postResample(pred = RF_train, obs = train$fare_amount)) # For Train

print(postResample(pred = RF_test, obs = test$fare_amount)) # For Test
```

############## **Visualization to check the model performance** ##################

########## **TRAIN** ##############

```
plot(train$ fare_amount
```

,type="l",lty=1.8,col="Green")
lines(RF_train,type="l",col="Blue")

########### **TEST** ###############
plot(test$ fare_amount
,type="l",lty=1.8,col="Red")
lines(RF_test,type="l",col="Blue")

Linear Regression

set.seed(123)

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

#Lets predict for training data
LR_train = predict(fit_LR, train)

#Lets predict for testing data
LR_test = predict(fit_LR,test)

----------**# Error metrics to calculation #**----------

print(postResample(pred = LR_train, obs = train$fare_amount)) # For Train

print(postResample(pred = LR_test, obs = test$fare_amount)) # For Test

#---------- XGBoost

#

set.seed(123)

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

#Lets predict for training data
XGB_train = predict(fit_XGB, train, n.trees = 500)

#Lets predict for testing data
XGB_test = predict(fit_XGB,test, n.trees = 500)

----------**# Error metrics to calculation #**----------

print(postResample(pred = XGB_train, obs = train$fare_amount)) # For train
print(postResample(pred = XGB_test, obs = test$fare_amount)) # For Test

#---------- Visualization Part to check performance of model

#

########### TRAIN ##############

plot(train$fare_amount,type="l",lty=1.8,col="
Red") lines(XGB_train,type="l",col="Blue")

########### **TEST** ###############
plot(test$fare_amount,type="l",lty=1.8,col="
Green") lines(XGB_test,type="l",col="Blue")

 # WE final XGboost model as our final model and apply on test data set (1st we need to preprocess and normalize all the test data using above pre-processing steps)

train = pf
test = data.frame(read.csv('test_normalize.csv')) **# preprocess and normalize data**

#XGboost
set.seed(123)

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

#lets predict for testing data
XGB_test1 = predict(fit_XGB,test, n.trees = 500)

 # predicting the fare amount for test case
test_final = data.frame(XGB_test1)

References

1. Data Cleaning, Model Development and Data Visualization
   https://edwisor.com/career-data-scientist

2. For Visualization using seaborn and other topics
   Krish Naik Youtube channel.

3. Haversine algorithm for distance calculation.
   https://www.movable-type.co.uk/scripts/latlong.html