Project Report

On

Cab fare Prediction

-Harsh Jain

Project - Cab Fare Prediction

Contents

Project - Cab Fare Prediction

Chapter 1

Introduction

Now a day's cab rental services are expanding with the multiplier rate. The ease of using the services and flexibility gives their customer a great experience with competitive prices.

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.

Data

The given data set are :-
Train_cab.csv
Test.csv

Understanding of data is the very 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, we need to go through each and every variable of it to understand and for better functioning.

- Size of Dataset Provided: - 16067 rows, 7 Columns
- Missing Values: Yes
- Outliers Presented: Yes

Below mentioned is a list of all the variable names with their meanings:

Variables	Description
fare_amount	Fare amount
pickup_datetime	Cab pickup date with time
pickup_longitude	Pickup location longitude
pickup_latitude	Pickup location latitude
dropoff_longitude	Drop location longitude
dropoff_latitude	Drop location latitude
passenger_count	Number of passengers sitting in the cab

Project - Cab Fare Prediction

Chapter 2

Pre-Processing

The work starts with data preprocessing, means looking at the data to get insights. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**.

These includes following steps:

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

To start this process we will first try and look at all the distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the distributions of the variable by QQ-normality graph or histogram. Histogram is the best chart to represent the data distribution, later the data is normalized or standardized, according to the model needs.

And we check for distribution of the target variable with respect to its independent features.

Data preprocessing is the tedious task, we need to focus more on this part to reduce the model complexity. Before feeding the data to model, we must preprocess the data, which has various stages like missing value analysis, impute the missing values, outlier check, normality check, sampling, weather the dataset is imbalanced are not. Then the data is subjected to model training and prediction.

After completing the model prediction, the machine learning prediction system is ready for deployment. Project deployment refers to , preparing the machine learning environment to the front end users.

Project - Cab Fare Prediction

Data Cleaning and Outlier

We are analysing the data of cab rental startup, whose features are comprised of fare amount, passenger count and geometrical coordinates like pickup latitude,longitude and drop off longitude and latitude and find out value which is out of rannge

So keeping in mind that these variables play an important role in the analysis, we keep checking the minimum and maximum values in this features.

Exploring the data, which means looking at all the features and knowing their characters. According to our problem statement:-

- We are analysing the data of cab rental startup, whose features are comprised of fare amount, passenger count and geometrical coordinates like pickup latitude,longitude and drop off longitude and latitude.
- So keeping in mind that these variables play an important role in the analysis, we keep checking the minimum and maximum values in this features.
- There were a lot of incorrect information in this feature, like observation showed up passenger count as 0 and 55 , 5000...and few showed decimal values 1.3
- And fare amount ranged from 1 to 400+ and its normal, few showed high as 4000 and 5000...etc , which is not possible.
- Latitudes range from -90 to 90.Longitudes range from -180 to 180. So Removing those features, which does not satisfy these ranges. Many showed 0 .
- So these residual information must be removed from data set, this is known as data cleaning.
- Many observations nearly 400 to 600 has this wrong information, so considering this as OUTLIERS in the data and removing it.
- Deriving the new features **year,day, date, month,min,hour** by timestamp variable pickup_datetime.
- Deriving the new feature **distance** from the given coordinates with the help of haversine function.

Creating some new variables from the given variables.

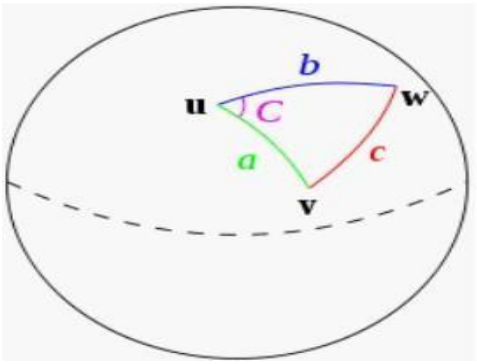
Here in our data set our variable name pickup_datetime contains date and time for pickup. So we tried to extract some important variables from pickup_datetime:

- Year
- Month
- Date
- Day of Week
- Hour

- Minute

Also, we tried to find out the distance using the haversine formula which says:

The **haversine formula** determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles



So our new extracted variables are:

- fare_amount
- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- passenger_count
- year
- Month
- Date
- Day of Week
- Hour
- Minute
- Distance

Project - Cab Fare Prediction

Selection of variables

Now as we know that all above variables are of now use so we will drop the redundant variables:

- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- Minute

	fare_amount	passenger_count	year	Month	Date	Day of Week	Hour	distance
0	4.5	1.0	2009.0	6.0	15.0	0.0	17.0	1.030764
1	16.9	1.0	2010.0	1.0	5.0	1.0	16.0	8.450134
2	5.7	2.0	2011.0	8.0	18.0	3.0	0.0	1.389525
3	7.7	1.0	2012.0	4.0	21.0	5.0	4.0	2.799270
4	5.3	1.0	2010.0	3.0	9.0	1.0	7.0	1.999157
5	12.1	1.0	2011.0	1.0	6.0	3.0	9.0	3.787239
6	7.5	1.0	2012.0	11.0	20.0	1.0	20.0	1.555807
8	8.9	2.0	2009.0	9.0	2.0	2.0	1.0	2.849627
9	5.3	1.0	2012.0	4.0	8.0	6.0	7.0	1.374577
10	5.5	3.0	2012.0	12.0	24.0	0.0	11.0	0.000000

Now only following variables we will use for further steps:

Data type of variable:-

VariableNames	Variable DataTypes
fare_amount	float64
passenger_count	object
year	object
Month	object
Date	object
Day of Week	object
Hour	object
distance	float64

Project - Cab Fare Prediction

Missing Value Analysis :-

In this step we look for missing values in the dataset like empty row column cell which was left after removing special characters and punctuation marks. Some missing values are in form of NA. missing values left behind after outlier analysis; missing values can be in any form.
Unfortunately, in this dataset we have found some missing values. Therefore, we will do some missing value analysis. Before imputed we selected random row no-1000 and made it NA, so that we will compare original value with imputed value and choose best method which will impute value closer to actual value.

index	variable	
0	fare_amount	22
1	pickup_datetime	1
2	pickup_longitude	0
3	pickup_latitude	0
4	dropoff_longitude	0
5	dropoff_latitude	0
6	passenger_count	55

We will impute values for fare_amount and passenger_count both of them has missing values 22 and 55 respectively. We will drop 1 value in pickup_datetime i.e it will be an entire row to drop.

Below are the missing value percentage for each variable:

Variables	Missing_percentage	
0	passenger_count	0.351191
1	fare_amount	0.140476
2	pickup_datetime	0.006385
3	pickup_longitude	0.000000
4	pickup_latitude	0.000000
5	dropoff_longitude	0.000000
6	dropoff_latitude	0.000000

Project - Cab Fare Prediction

Nearly 77 missing values, but these missing values are omitted, as they are in very low composition.

The reason for deleting the missing values is , while cleaning the data we encountered many wrong data points nearly 500 to 600 data points, which doesn't make sense for this particular problem statement.

As the missing values is in very low proportion ,decided to delete it.

Some more data exploration:-

In this report we are trying to predict the fare prices of a cab rental company. So here we have a data set of 16067 observations with 8 variables including one dependent variable.

Below are the names of Independent variables:
passenger_count, year, Month, Date, Day of Week, Hour, distance
Our Dependent variable is: fare_amount

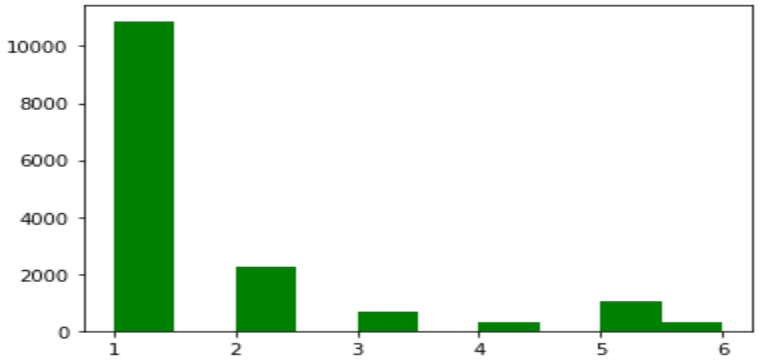
Uniqueness in Variable

We need to look at the unique number in the variables which help us to decide whether the variable is categorical or numeric. So, by using python script 'nunique' we tried to find out the unique values in each variable. We have also added the table below

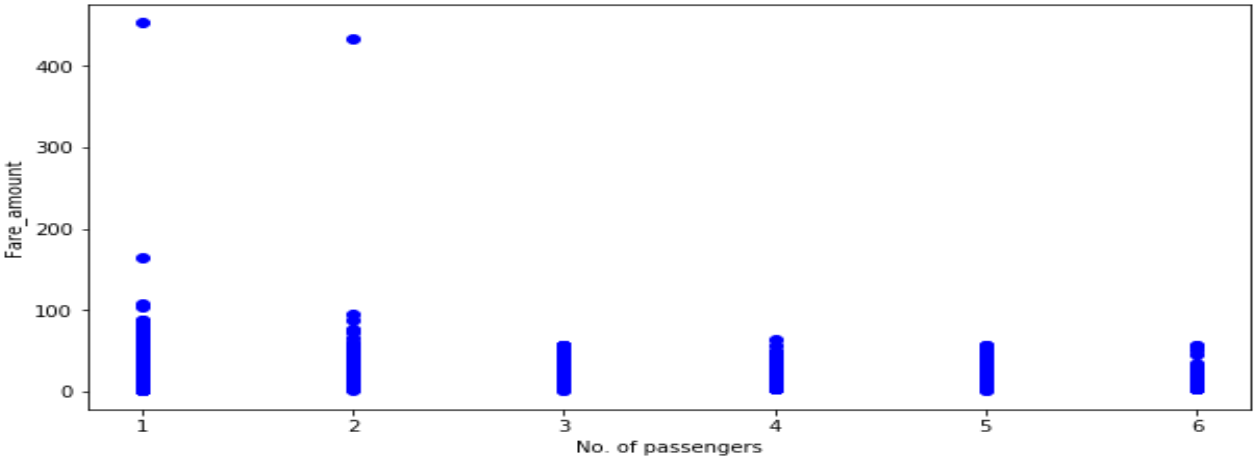
Variable Name	Unique Counts
fare_amount	450
passenger_count	7
year	7
Month	12
Date	31
Day of Week	7
Hour	24
distance	15424

Project - Cab Fare Prediction

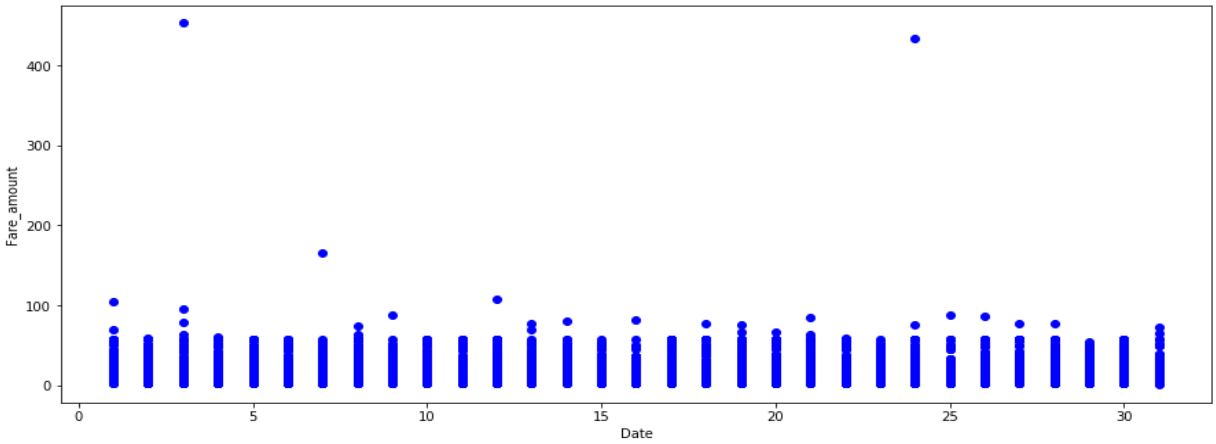
Data Visualization.



- There are many single travelling passengers, followed by 2,5,3,4,6.

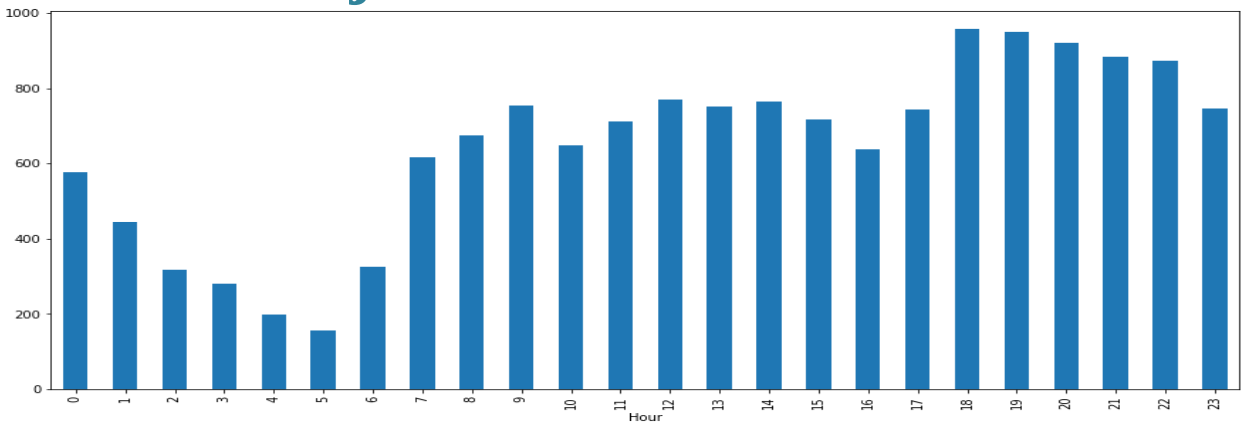


- Single travelling passengers contribute a lot to the fare amount.

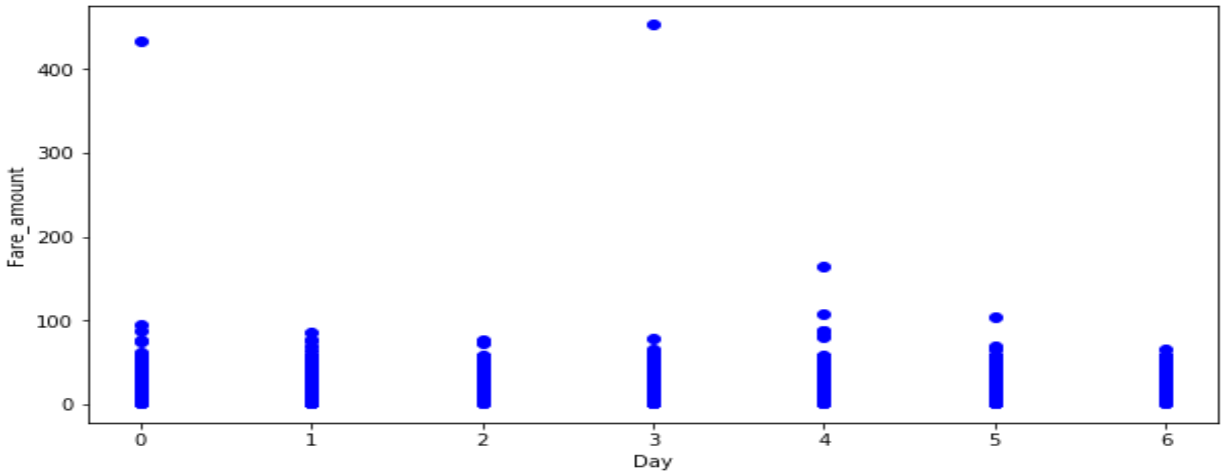


- All dates have high demand of cabs, as it is common.
- Commute is common!!!

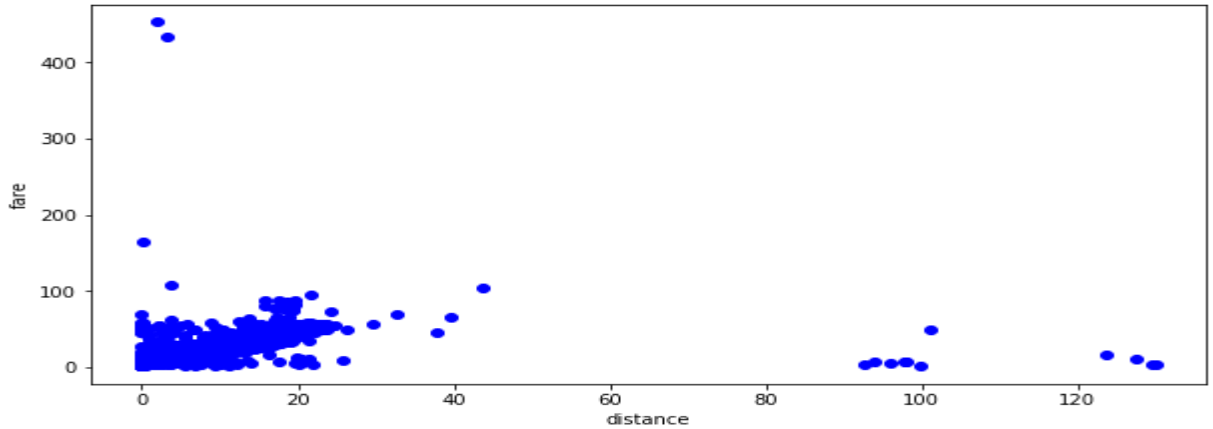
Project - Cab Fare Prediction



- There is a high demand for cabs in the early morning and from night to midnight.



- There is a rise in fare amount on Monday,Thursday and Friday.
- As monday is the starting day of the week and thursday and friday are weekend rush.



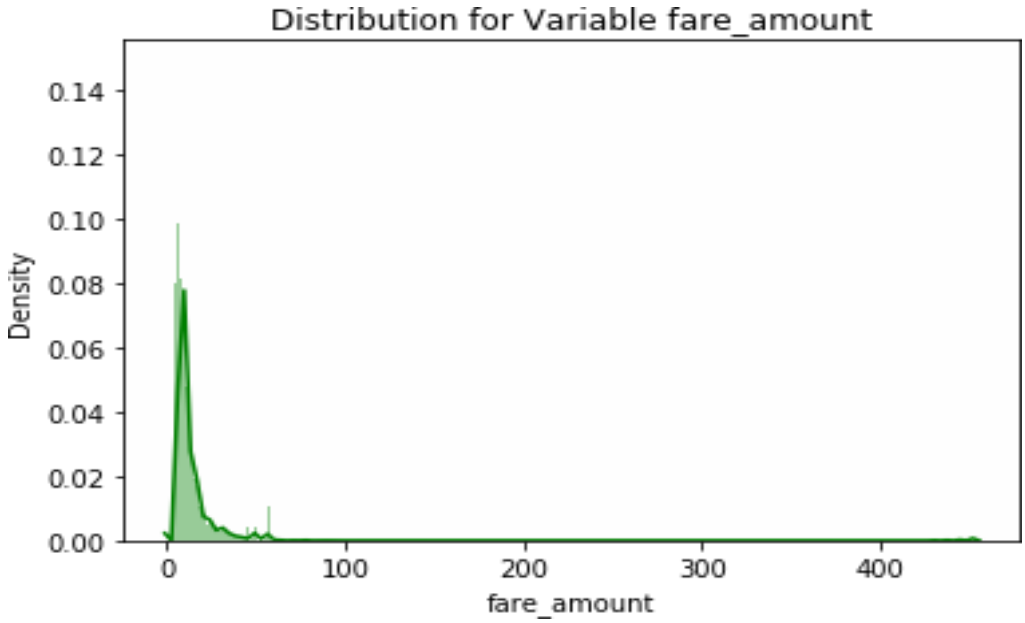
- It's quite obvious, that increase in distance = increases in fare amount

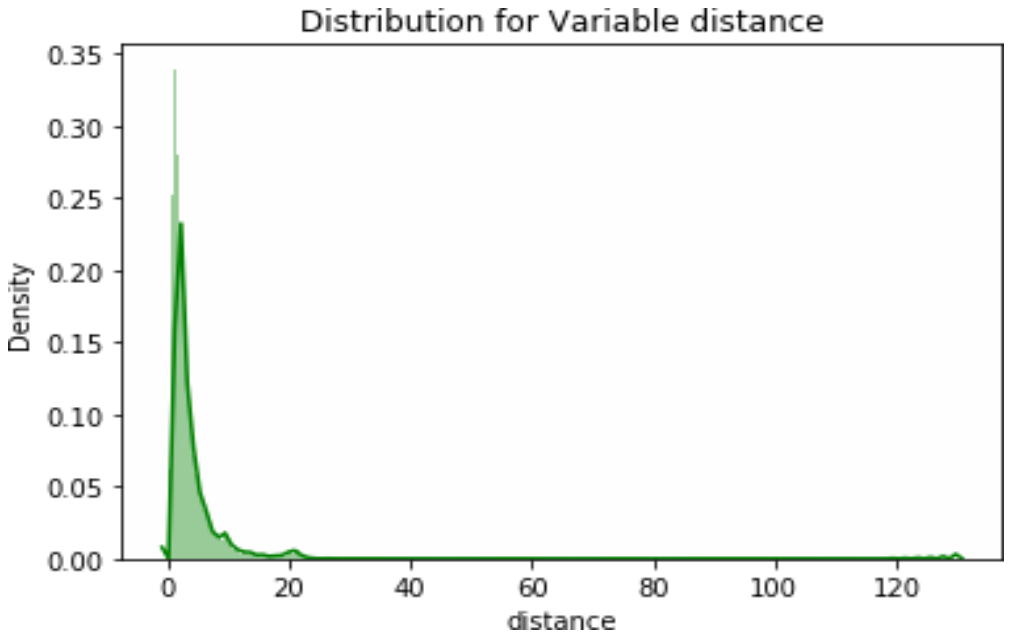
Project - Cab Fare Prediction

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.

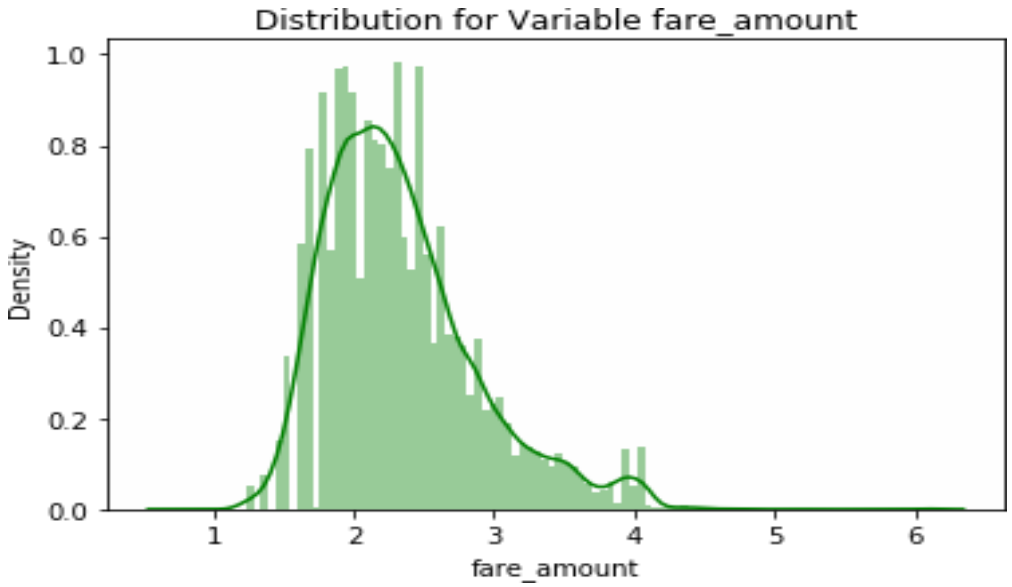
Below mentioned graphs shows the probability distribution plot to check distribution before log transformation:

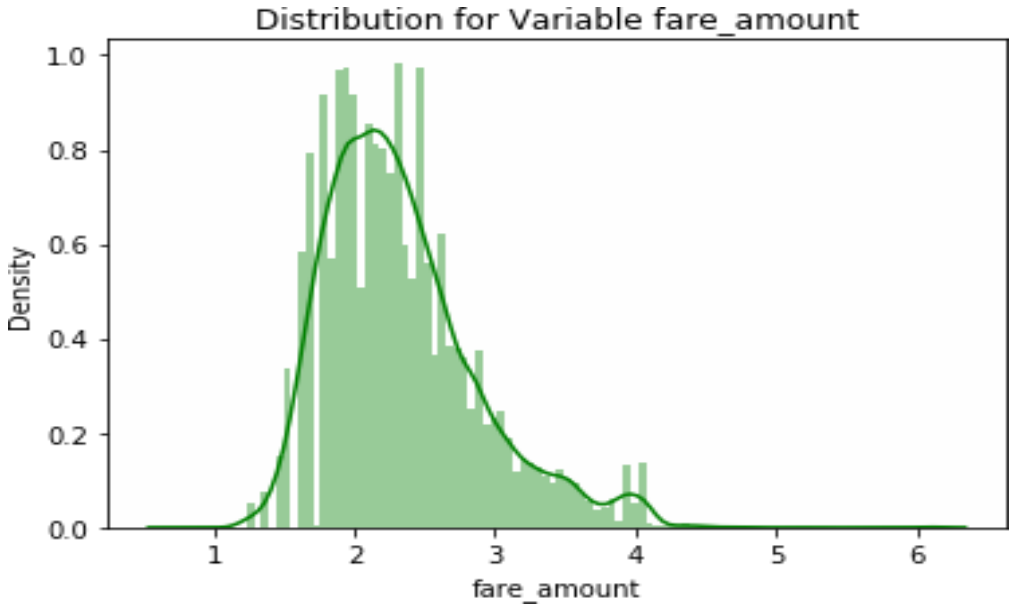




Project - Cab Fare Prediction

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

Project - Cab Fare Prediction

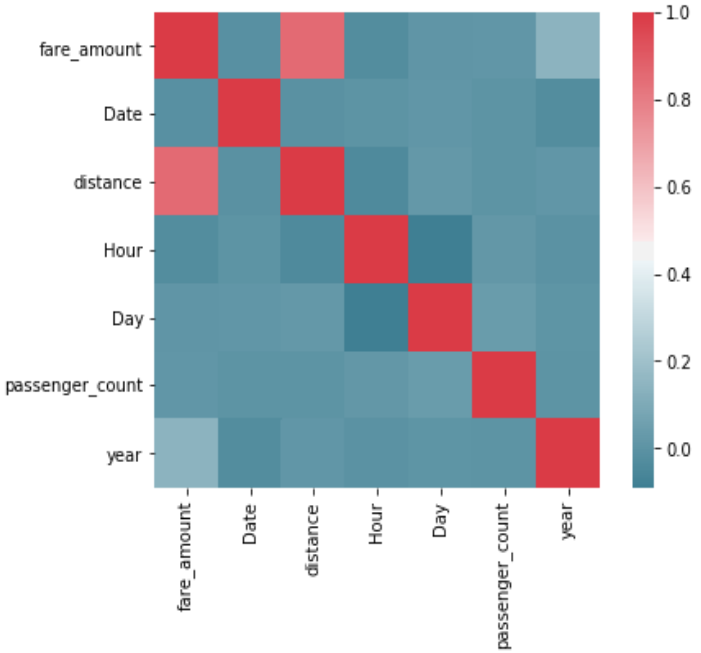
Feature Selection

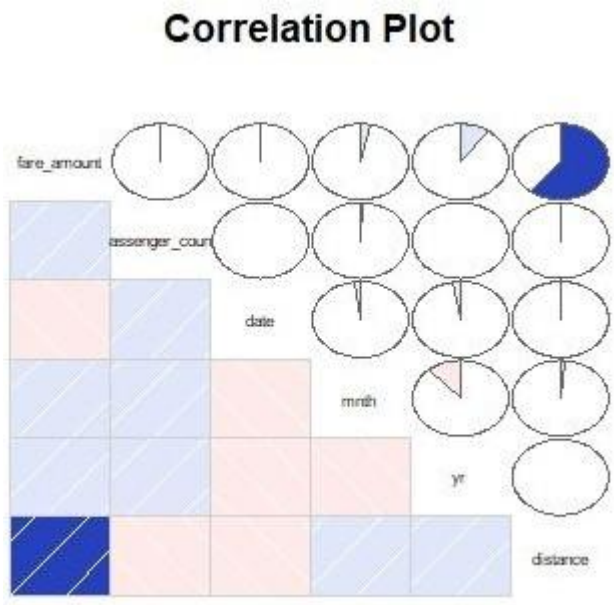
Before performing any type of modelling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction.

For all the numerical variable present in the training data set.

The objective of this selection is to neglect the features with high correlation.

There are several methods of doing that. Below we have used the **correlation plot** and **VIF** for checking multicollinearity and level 1 regularization (Lasso Regression) Here we use correlation plot.





#No variable from the 6 input variables has collinearity problem.

#The linear correlation coefficients ranges between:

#min correlation (distance ~ passenger_count): 0.001499388

#max correlation (month ~ year): -0.124351

#--------- VIFs of the remained variables --------

#	Variables	VIF
#1	passenger_count	1.001601
#2	distance	1.001827
#3	year	1.017753
#4	month	1.017362
#5	date	1.002016
#6	hour	1.002634

Chapter 3

Model Building

Based on the target variable or feature we select the model. Here in our case cab rentals data set, the target variable is comprised of continuous distribution, so the model will be the regression model, to predict continuous outcomes.

Regression Model

In statistical modeling, regression analysis is a set of statistical processes for estimating the relationships between a dependent variable (often called the 'outcome variable') and one or more independent variables (often called 'predictors', 'covariates', or 'features').

Regression analysis is primarily used for two conceptually distinct purposes. First, regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning. Second, in some situations regression analysis can be used to infer causal relationships between the independent and dependent variables. Importantly, regressions by themselves only reveal relationships between a dependent variable and a collection of independent variables in a fixed dataset. To use regressions for prediction or to infer causal relationships, respectively, a researcher must carefully justify why existing relationships have predictive power for a new context or why a relationship between two variables has a causal interpretation. The latter is especially important when a researcher hopes to estimate causal relationships using observational data.

Linear Regression model

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models.[3] Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.

Project - Cab Fare Prediction

Dep. Variable:	fare_amount	**R-squared (uncentered):**	0.986
Model:	OLS	**Adj. R-squared (uncentered):**	0.986
Method:	Least Squares	**F-statistic:**	1.254e+05
Date:	Thu, 30 Jul 2020	**Prob (F-statistic):**	0.00
Time:	22:39:32	**Log-Likelihood:**	-1892.9
No. Observations:	12339	**AIC:**	3800.
Df Residuals:	12332	**BIC:**	3852.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
passenger_count	0.0047	0.002	2.335	0.020	0.001	0.009
year	0.0007	5.77e-06	114.675	0.000	0.001	0.001
Month	0.0037	0.001	5.062	0.000	0.002	0.005
Date	-0.0002	0.000	-0.546	0.585	-0.001	0.000
Day	-0.0019	0.001	-1.450	0.147	-0.004	0.001
Hour	0.0005	0.000	1.352	0.176	-0.000	0.001
distance	0.7694	0.004	184.062	0.000	0.761	0.778

Omnibus:	6069.957	**Durbin-Watson:**	1.995
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	518935.332
Skew:	1.467	**Prob(JB):**	0.00
Kurtosis:	34.635	**Cond. No.**	3.31e+03

Lasso Regression model

In statistics and machine learning, lasso (least absolute shrinkage and selection operator) (also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.

Lasso is able to achieve both of these goals by forcing the sum of the absolute value of the regression coefficients to be less than a fixed value, which forces certain coefficients to be set to zero, effectively choosing a simpler model that does not include those coefficients. This idea is similar to ridge regression, in which the sum of the squares of the coefficients is forced to be less than a fixed value, though in the case of ridge regression, this only shrinks the size of the coefficients, it does not set any of them to zero.

Lasso(alpha=0.005, copy_X=True, fit_intercept=True, max_iter=1000,
normalize=False, positive=False, precompute=False, random_state=0,
selection='cyclic', tol=0.0001, warm_start=False)

Decision Tree Regressor

The decision trees is used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve.

We can see that if the maximum depth of the tree (controlled by the max_depth parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e. they overfit.

Decision Tree Regression:

Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented by a discrete, known set of numbers or values.

Discrete output example: A weather prediction model that predicts whether or not there'll be rain in a particular day.

Continuous output example: A profit prediction model that states the probable profit that can be generated from the sale of a product.

DecisionTreeRegressor(criterion='mse', max_depth=10, max_features=None,
max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
presort=False, random_state=None, splitter='best')

Random Forest Regression

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Approach :

- Pick at random K data points from the training set.
- Build the decision tree associated with those K data points.
- Choose the number Ntree of trees you want to build and repeat step 1 & 2.
- For a new data point, make each one of your Ntree trees predict the value of Y for the data point, and assign the new data point the average across all of the predicted Y values.

RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=20,
n_jobs=None, oob_score=False, random_state=None,
verbose=0, warm_start=False)

Chapter 4

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

Evaluating a model is a core part of building an effective machine learning model There are several evaluation metrics, like confusion matrix, cross-validation, AUC-ROC curve, etc. Different evaluation metrics are used for different kinds of problems.

MAPE

Here in regression model the evaluation is done by MAPE.

The mean absolute percentage error (MAPE), also known as mean absolute percentage deviation (MAPD), is a measure of prediction accuracy of a forecasting method in statistics, for example in trend estimation, also used as a loss function for regression problems in machine learning. It usually expresses accuracy as a percentage, by Multiplying 100 to percentage error.

For R

```
mape=function(av,pv){
    mean(abs((av-pv)/av))*100   #av=actual value and pv= predicted value
}
```

For python

```
#mape                                  #av= actual value and pv= predicted value
def mape(av, pv):
        mape = np.mean(np.abs((av - pv) / av))*100
        return mape
```

From above we can define the mape function in both R and Python. The internal

mape operation is

$$\frac{\sum_{t=1}^{n}\left|\frac{(Y_t - \hat{Y}_t)}{Y_t}(100)\right|}{n}$$

regr.eval function available in R

Calculate Some Standard Regression Evaluation Statistics

This function is able to calculate a series of regression evaluation statistics given two vectors: one with the true target variable values, and the other with the predicted target variable values.

```
regr.eval(trues, preds, stats = if (is.null(train.y))
 c("mae","mse","rmse","mape") else
c("mae","mse","rmse","mape"), train.y = NULL)
```

The regression evaluation statistics calculated by this function belong to two different groups of measures: absolute and relative. The former include "mae", "mse", and "rmse" and are calculated as follows:

"mae": mean absolute error, which is calculated as sum(|t_i - p_i|)/N, where t's are the true values and p's are the predictions, while N is supposed to be the size of both vectors.

"mse": mean squared error, which is calculated as sum((t_i - p_i)^2)/N "rmse": root

mean squared error that is calculated as sqrt(mse)

The remaining measures ("mape", "nmse" and "nmae") are relative measures, the two later comparing the performance of the model with a baseline. They are unit-less measures with values always greater than 0. In the case of "nmse" and "nmae" the values are expected to be in the interval [0,1] though occasionally scores can overcome 1, which means that your model is performing worse than the baseline model. The baseline used in our

implementation is a constant model that always predicts the average target variable value, estimated using the values of this variable on the training data (data used to obtain the model that generated the predictions), which should be given in the parameter train.y. The relative error measure "mape" does not require a baseline. It simply calculates the average percentage difference between the true values and the predictions.

MAPE TEST

The function created in python.

####MODEL EVALUATION #####

```
#mape                                                    #av= actual value
and pv= predicted value def
mape(av, pv):
        mape = np.mean(np.abs((av - pv) / av))*100 return mape
```

performance of linear regression model.

mape(ytest,predictionLR) **### Accuracy= 92.0 %**

error =8.0 %

performance of lasso regression model.

mape(ytest,predict_lasso) **### Accuracy= 92.4 %**

error= 7.6 %

performance of decision tree regression model.

mape(ytest,prediction_DTR) **### Accuracy= 92.0 %**

error= 8.0 %

performance of random forest regression model.

mape(ytest,RFprediction)

Accuracy= 92.5 %

error= 7.5 %

Standard Regression Evaluation Statistics

The function created in R:

performance of linear regression model.

```
mape(Test_set[,1],predict_lm)
# 7.500
regr.eval(Test_set[,1],predict_lm)
#        mae             mse              rmse              mape
#  0.17297582    0.07128912      0.26700023      0.07500190
```

performance of decision tree regression model.

```
mape(Test_set[,1],predictions_tree)
# 8.09
regr.eval(Test_set[,1],predictions_tree)
#       mae              mse            rmse               mape
#   0.18633188     0.07226311    0.26881799        0.0809881
```

performance of random forest regression model.

```
#model evalution
mape(Test_set[,1],RF_Predictions)
# 7.19

regr.eval(Test_set[,1],RF_Predictions)
#       mae              mse            rmse            mape
#   0.16310940     0.05805191    0.24093963     0.0719228
```

Observations

- As we can observe that the error rate is less in models like random forest regression models , while comparing to other models.
- Each and every model used here lasso, linear , decision trees , random forest regression models are good at predicting the numerical outcomes.
- We select the model with less errors and high accuracy.

Model Selection

ALL MODELS PERFORM WELL
NOTICABALLY **RANDOM FOREST** PERFORM VERY GOOD.....

R Code

```
rm(list=ls())

# set working directory
setwd("C:/Users/Guest/Desktop/harsh jain/Project -2")
getwd()

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

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

# loading the data
train=read.csv("train_cab.csv",header=TRUE)
test=read.csv("test.csv",header=TRUE)

# EXPLORING DATA
#viewing the data
head(train,5)
head(test,5)

#structure of data or data types
str(train)
str(test)

#Summary of data
summary(train)
summary(test)

#dimension of data
dim(train)
dim(test)

#unique value of each count
apply(train, 2,function(x) length(table(x)))
apply(test, 2,function(x) length(table(x)))

# converting the features in the required data types.
train$fare_amount = as.numeric(as.character(train$fare_amount))
train$passenger_count=round(train$passenger_count)
```

```
# DATA CLEANING & OUTLIER
# remove irregular data point in data set
sort(train$fare_amount,decreasing = TRUE)

# fare amount cannot be less than one
# considring fare amount 453 as max
# removing all the fare amount greater than 453 and less than 1.

# count no. of row which is outside the condition
nrow(train[which(train$fare_amount < 1 ),])
nrow(train[which(train$fare_amount >453 ),])

#remove those data point in fare_amount
train = train[-which(train$fare_amount < 1 ),]
train = train[-which(train$fare_amount >453 ),]

sort(train$passenger_count,decreasing = TRUE)
# passenger count cannot be Zero
# even if we consider suv max seat is 6
# so removing passenger count greater than 6 and less than 1.
# count no. of row which is outside the condition
nrow(train[which(train$passenger_count < 1 ),])
nrow(train[which(train$passenger_count >6 ),])

#remove those data point in passengger_count
train=train[-which(train$passenger_count < 1 ),]
train=train[-which(train$passenger_count >6 ),]

# Latitudes range from -90 to 90.
# Longitudes range from -180 to 180.
# Removing which does not satisfy these ranges.

print(paste('pickup_longitude above 180=',nrow(train[which(train$pickup_longitude >180 ),])))
print(paste('pickup_longitude above -180=',nrow(train[which(train$pickup_longitude < -180 ),])))
print(paste('pickup_latitude above 90=',nrow(train[which(train$pickup_latitude > 90 ),])))
print(paste('pickup_latitude above -90=',nrow(train[which(train$pickup_latitude < -90 ),])))
print(paste('dropoff_longitude above 180=',nrow(train[which(train$dropoff_longitude > 180 ),])))
print(paste('dropoff_longitude above -180=',nrow(train[which(train$dropoff_longitude < -180 ),])))
print(paste('dropoff_latitude above -90=',nrow(train[which(train$dropoff_latitude < -90 ),])))
print(paste('dropoff_latitude above 90=',nrow(train[which(train$dropoff_latitude > 90 ),])))

# removing one data point
train = train[-which(train$pickup_latitude > 90),]

# Also we will see if there are any values equal to 0.
nrow(train[which(train$pickup_longitude == 0 ),])
```

```
nrow(train[which(train$pickup_latitude == 0 ),])
nrow(train[which(train$dropoff_longitude == 0 ),])
nrow(train[which(train$dropoff_latitude == 0 ),])

# removing those data points.
train=train[-which(train$pickup_longitude == 0 ),]
train=train[-which(train$dropoff_longitude == 0),]

# calculate distance from the given coordinates of latitude and longitude.
deg_to_rad = function(deg){
  (deg * pi) / 180
}
haversine = function(long1,lat1,long2,lat2){

  #long1rad = deg_to_rad(long1)
  phi1 = deg_to_rad(lat1)

  #long2rad = deg_to_rad(long2)
  phi2 = deg_to_rad(lat2)

  delphi = deg_to_rad(lat2 - lat1)
  dellamda = deg_to_rad(long2 - long1)

  a = sin(delphi/2) * sin(delphi/2) + cos(phi1) * cos(phi2) *
    sin(dellamda/2) * sin(dellamda/2)

  c = 2 * atan2(sqrt(a),sqrt(1-a))
  R = 6371e3
  R * c / 1000
}


train$distance =
haversine(train$pickup_longitude,train$pickup_latitude,train$dropoff_longitude,train$dropoff_latitude)
test$distance =
haversine(test$pickup_longitude,test$pickup_latitude,test$dropoff_longitude,test$dropoff_latitude)

# distance variable is obtained from co-ordinates points ,so there is no further use of pickup & dropoff
points.
# hence we drop these varible from data set
train = subset(train,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
test = subset(test,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))

# considering the distance 130 as max and greater than 0,considering rest as outlier.
nrow(train[which(train$distance ==0 ),])
nrow(test[which(test$distance==0 ),])
```

```
nrow(train[which(train$distance >130 ),])
nrow(test[which(test$distance >130 ),])

# removing the data points by considering the above conditions.
train=train[-which(train$distance ==0 ),]
train=train[-which(train$distance >130 ),]
test=test[-which(test$distance ==0 ),]

str(train)
str(test)

summary(train)
summary(test)

# Convert pickup_datetime from factor to date time
# new features will be year,month,date,hour

train$pickup_datetime = as.POSIXct(train$pickup_datetime,format='%Y-%m-%d %H:%M:%S UTC')
train$year =as.integer(format(train$pickup_datetime,"%Y"))
train$month =as.integer(format(train$pickup_datetime,"%m"))
train$date = as.integer(format(train$pickup_datetime,"%d"))
train$hour = as.integer(format(train$pickup_datetime,"%H"))

# for test data set.
test$pickup_datetime = as.POSIXct(test$pickup_datetime,format='%Y-%m-%d %H:%M:%S UTC')
test$year =as.integer(format(test$pickup_datetime,"%Y"))
test$month =as.integer(format(test$pickup_datetime,"%m"))
test$date = as.integer(format(test$pickup_datetime,"%d"))
test$hour = as.integer(format(test$pickup_datetime,"%H"))

#Drop the variable Pickup-datetime
train = subset(train,select = -c(pickup_datetime))
test = subset(test,select = -c(pickup_datetime))

str(train)
str(test)

# MISSINNG VALUE ANALYSIS
# checking missing data
apply(train,2, function(x) {sum(is.na(x))})

#Creating dataframe with missing values present in each variable
miss_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
miss_val$Columns = row.names(miss_val)
row.names(miss_val) = NULL
names(miss_val)[1] = "miss_percentage"
```

```
#Calculating percentage missing value
miss_val$miss_percentage = (miss_val$miss_percentage/nrow(train )) * 100

# Sorting null_val in Descending order
miss_val = miss_val[order(-miss_val$miss_percentage),]

# Reordering columns
miss_val = miss_val[,c(2,1)]

#We have seen that null values are very less in our data set i.e. less than 1%.
#So we can delete the columns having missing values
train =DropNA(train)

#Verifying missing values after deletion
sum(is.na(train))
names(train)

# OUTLIER ANALYSIS = I DID IT MANNULLY IN ABOVE CODING #

# FEATURE SELECTION
#selecting only numeric
numeric_index = sapply(train,is.numeric)
numeric_data = train[,numeric_index]
cnames = colnames(numeric_data)

#Correlation analysis for numeric variables
corrgram(train[,numeric_index],upper.panel=panel.pie, main = "Correlation Plot")

# hence all variable are important.

# FEATURE SCALING
truehist(train$fare_amount)
lines(density(train$fare_amount))

d=density(train$fare_amount)
plot(d,main="distribution")
polygon(d,col="green",border="red")

D=density(train$distance)
plot(D,main="distribution")
polygon(D,col="yellow",border = "red")

A=density(test$distance)
plot(A,main="distribution")
polygon(A,col="black",border="red")
```

```
#Normalisation
# log transformation.
train$fare_amount=log1p(train$fare_amount)
test$distance=log1p(test$distance)
train$distance=log1p(train$distance)

# checking back features after transformation.
d=density(train$fare_amount)
plot(d,main="distribution")
polygon(d,col="green",border="red")

D=density(train$distance)
plot(D,main="distribution")
polygon(D,col="red",border="black")

A=density(test$distance)
plot(A,main="distribution")
polygon(A,col="black",border="red")

###check multicollearity
vif(train[,-1])
vifcor(train[,-1], th = 0.8)

#No variable from the 6 input variables has collinearity problem.
#The linear correlation coefficients ranges between:
#min correlation ( distance ~ passenger_count ):  0.001499388
#max correlation ( month ~ year ):  -0.124351

#--------- VIFs of the remained variables --------
# Variables      VIF
#1 passenger_count 1.001601
#2        distance 1.001827
#3            year 1.017753
#4           month 1.017362
#5            date 1.002016
#6            hour 1.002634

# MODEL BUILDING
#Divide data into trainset and testset
set.seed(1234)
Train.index = sample(1:nrow(train), 0.8 * nrow(train))
Train_set =  train[ Train.index,]
Test_set =  train[-Train.index,]

## accuracy check
```

```
#defining the function (to find the error percentage)
mape=function(av,pv){
  mean(abs((av-pv)/av))*100
}
#av = actual value
#pv = predicted value

# LINEAR REGRESSION #

#Develop Model on training data
linear_model=lm(fare_amount~.,data=Train_set)
summary(linear_model)

#Lets predict for testset data
predict_lm=predict(linear_model,Test_set[,2:7])
# For test data
predict_test=predict(linear_model,test)

# model evalution
mape(Test_set[,1],predict_lm)
# 7.500
regr.eval(Test_set[,1],predict_lm)
#       mae       mse       rmse       mape
#  0.17297582 0.07128912 0.26700023 0.07500190

# DECISION TREE #

#Develop Model on training data
DT=rpart(fare_amount~.,data=Train_set,method="anova")

#Lets predict for testset data
predictions_tree=predict(DT,Test_set[,2:7])

# For test data
predictions_test=predict(DT,test)

# model evalution
mape(Test_set[,1],predictions_tree)
# 8.09
regr.eval(Test_set[,1],predictions_tree)
#       mae       mse       rmse       mape
#   0.18633188 0.07226311 0.26881799 0.0809881

# RANDOM FOREST

#Develop Model on training data
```

```
rf_model = randomForest(fare_amount~ ., Train_set, importance = TRUE, ntree = 500)

#Extract rules fromn random forest
treeList = RF2List(rf_model)

#Extract rules
rules= extractRules(treeList, Train_set[,2:7])

#Visualize some rules
rules[1:2,]

#Make rules more readable:
readrules = presentRules(rules, colnames(Train_set))
readrules[1:2,]

#Lets predict for testset data
RF_Predictions = predict(rf_model, Test_set[,2:7])

# For test data
RF_test = predict(rf_model, test)

#model evalution
mape(Test_set[,1],RF_Predictions)
# 7.19

regr.eval(Test_set[,1],RF_Predictions)
#       mae        mse       rmse       mape
#   0.16310940 0.05805191 0.24093963 0.07192287

# from above result it state that random forest is best model
# And accuracy of model in order of - Random forest > Linear regression > Decision tree

# Hence we use random forest model for this data set:-
test$predicted_fare = with(test,RF_test)

# save this predicted value in data set
write.csv(test,'Predicted_value_in_R.csv',row.names = FALSE)

## END OF PROJECT #
```

Project - Cab Fare Prediction