CAB FARE PREDICTION.

PRUDHVI KUMAR K

07th November, 2019

Contents

Chapter-1 Introduction

1.1 Problem Statement

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected 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.

The objective of this project is to predict the Cab fare amount for the new test case , by analysing the given Historical Data.

1.2 Data

The given data attributes are

The ,

Train_cab.csv

Test.csv

The datasets features are comprised of dependant and independant features.

Dependant Features:- fare_amont, which exists only in train data set.

Independent Features:- pickup_datetime, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, passenger_count. And this features are common for both the dat sets.

Longitude and Latitude are the geometrical coordinates.

Chapter 2

Methodology

2.1 .1 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**.

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.

The Incorrect information.

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 .

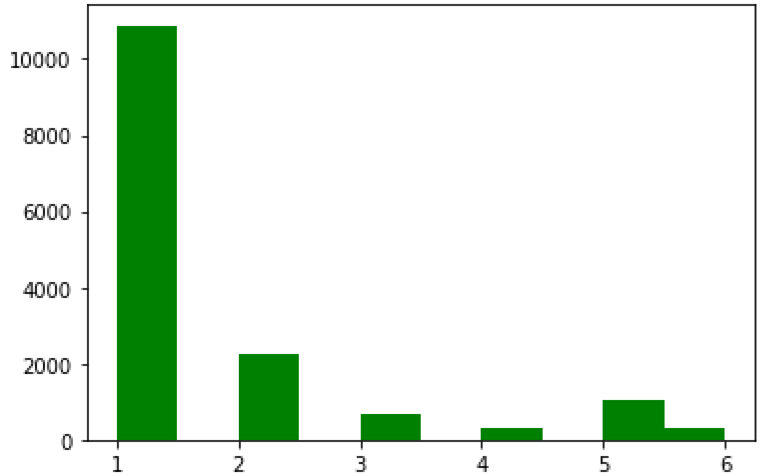
So these residual information must be removed from data set, this is known as data cleaning.

Many observations nearly 400 to 600 has this wrong information, so considering this as OUTLIERS in the data and removing it.

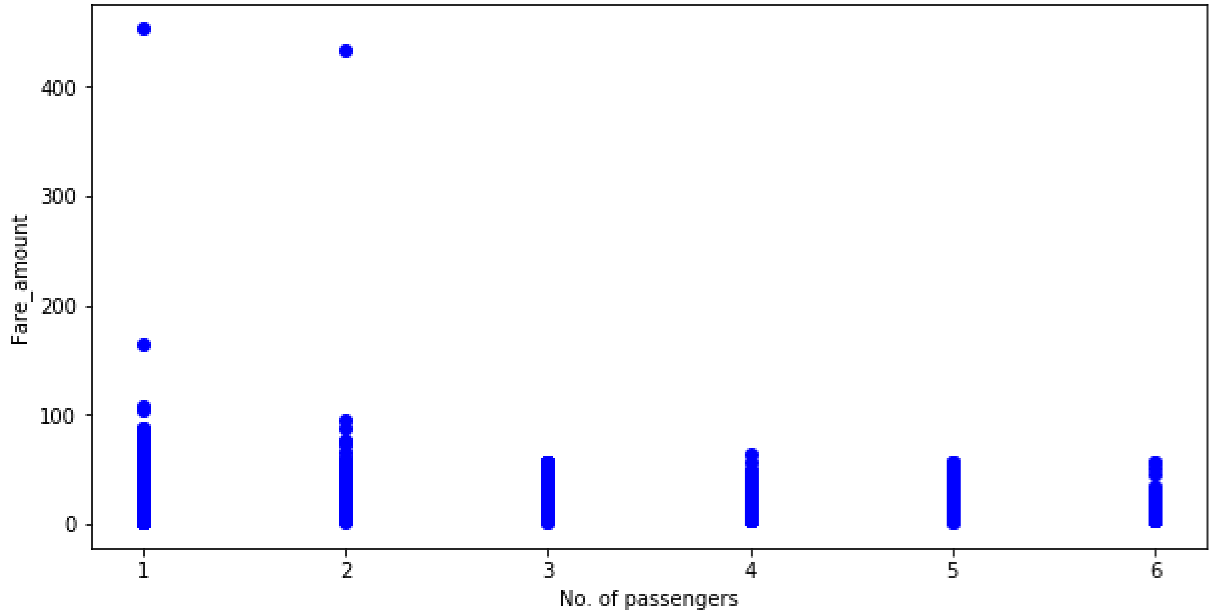
Deriving the new features **year,day, date, month,min,hour** by timestamp variable pickup_datetime.

Deriving the new feature **distance** from the given coordinates with the help of haversine function.

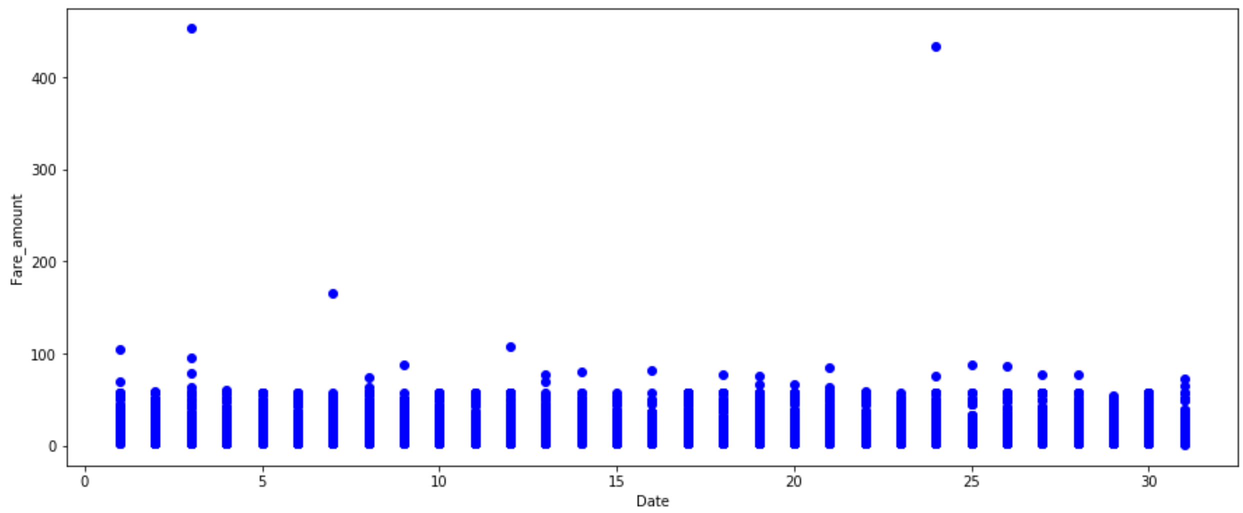
Data Visualization.



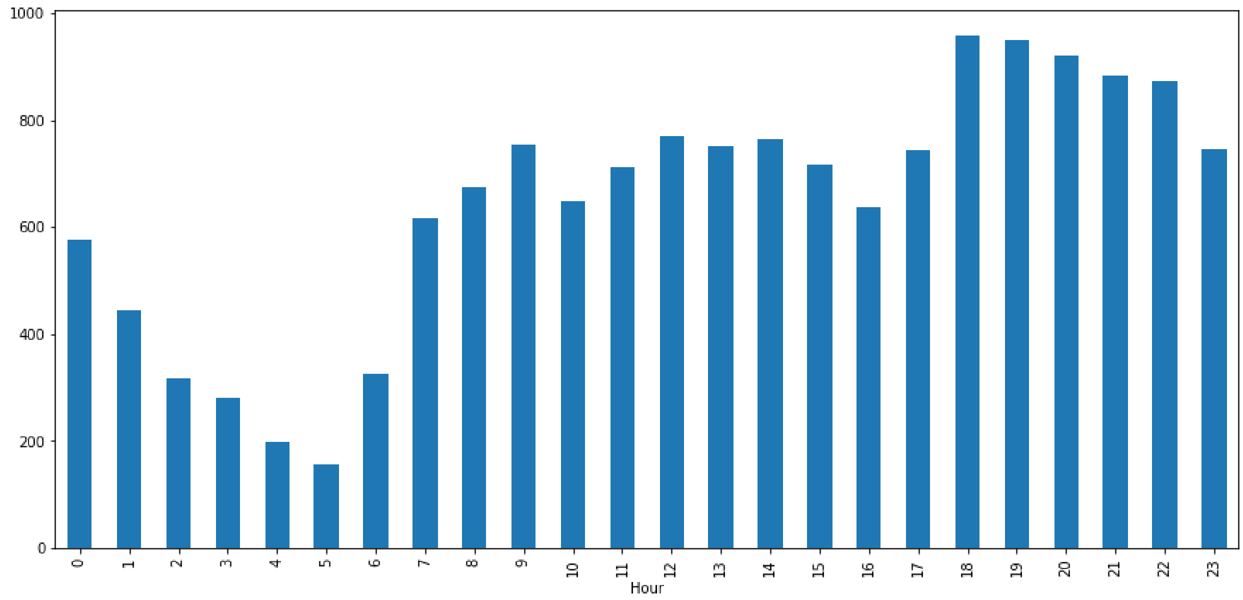
- There are many single travelling passengers, followed by 2,5,3,4,6.



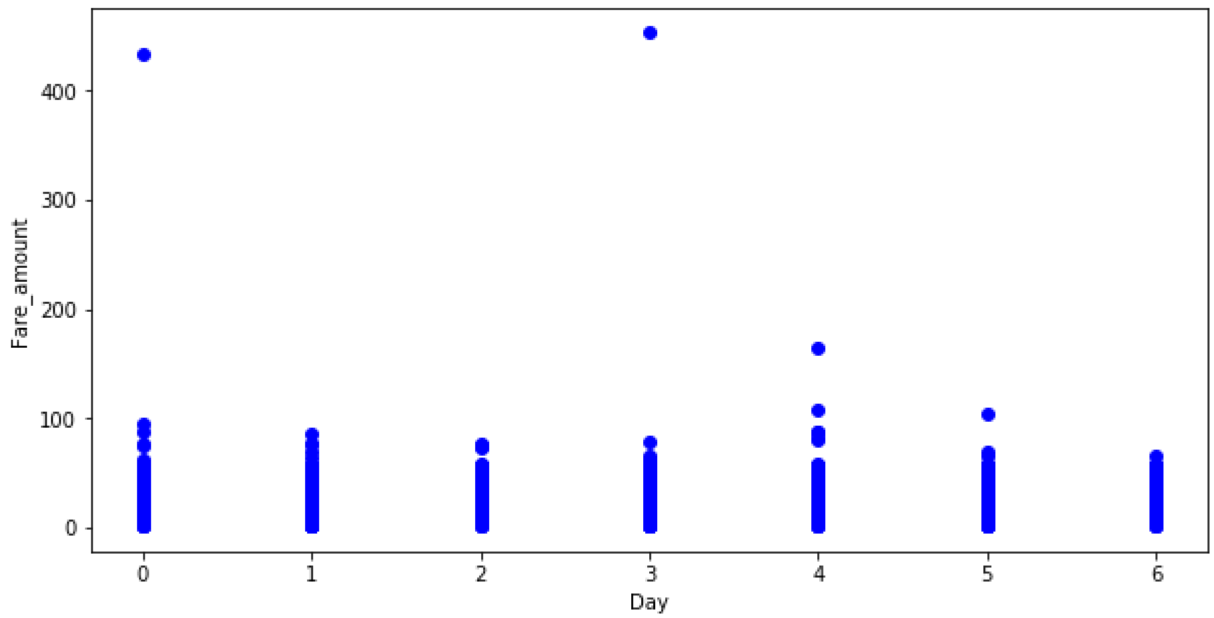
- Single travelling passengers contribute a lot to the fare amount.



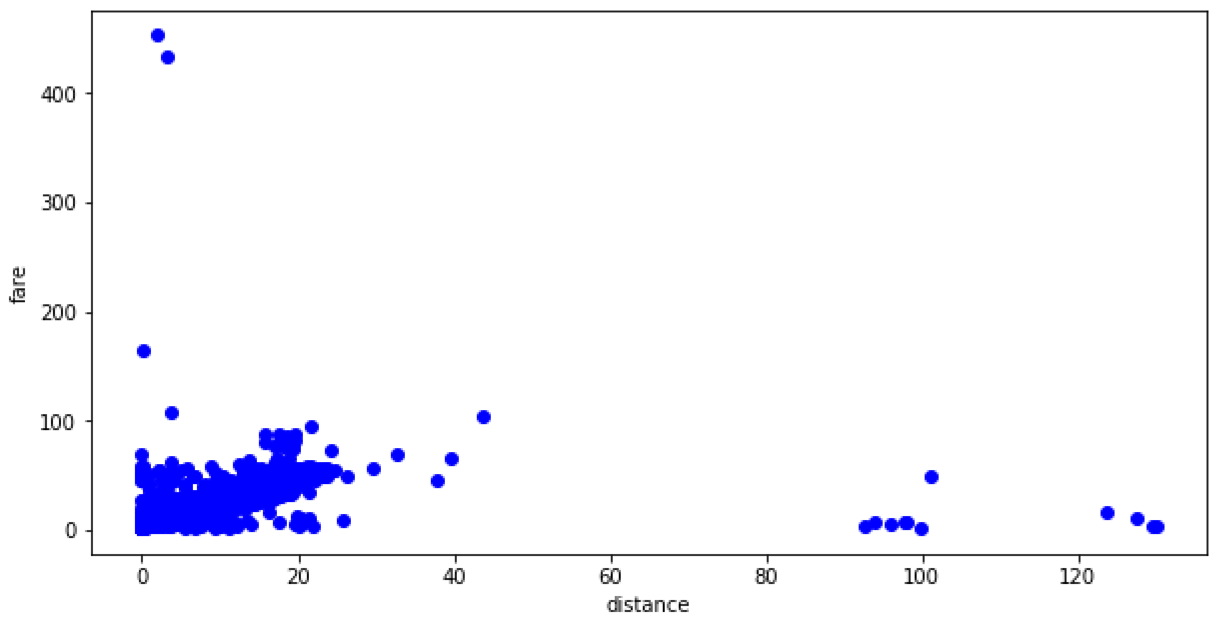
- All dates have high demand of cabs, as it is common.
- Commute is common!!!



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



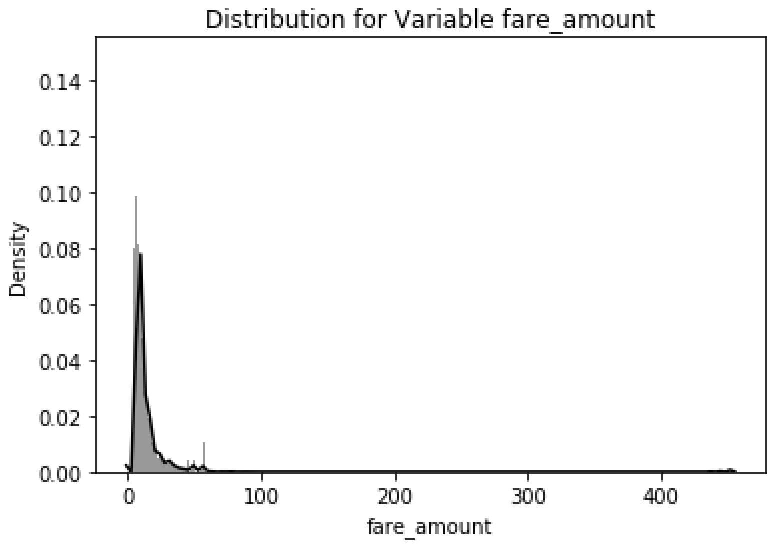
- There is a rise in fare amount on Monday,Thursday and Friday.
- As monday is the starting day of the week and thursday and friday are weekend rush.

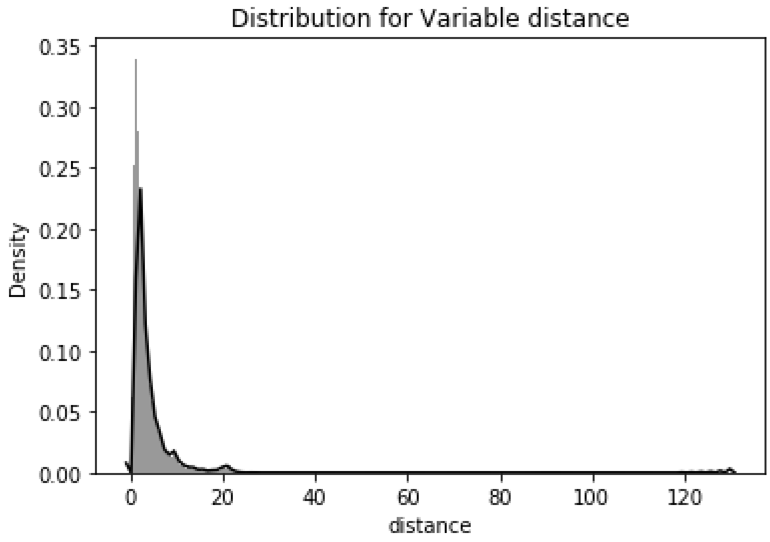


- It's quite obvious, that increase in distance = increases in fare amount.

Distribution of the features.

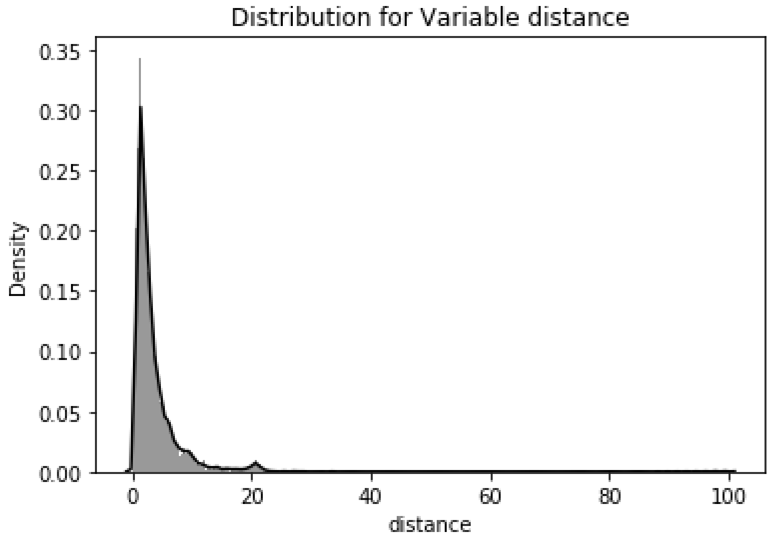
Let's check the distribution of the features.





The information in fare amount and distance of train dataset are skewed.

We found the skewed data in distance of training dataset.



Observation

- There are few missing values, nearly 77 NA values.
- With respect to our problem statement, we observed many incorrect data in our features and deleted/cleaned it.(That is in form of outlier.)
- With timestamp features and coordinates we created new features, like year, month,day,hour,date,distance.
- The above data visualizations shows the characters of the features in their respective domain.
- The features like distance in both the datasets and fare amount in the training set must we normalized with log transformation or any other normalization method.
- The dependent feature is Fare_amount.
- The type of machine learning model required to this problem is a REGRESSION MODEL, which can predict the continuous outcome.

Few of the regression model are

- Linear regression
- Lasso regression
- Decision tree regression
- Random forest regressor

2.1.2 Feature Engineering.

It is a part of data pre-processing, after cleaning the data.

The term Feature Engineering refers to transforming the features available in the dataset.

Transforming in sense, preparing the features ready to use for model building . This includes

Missing values.

There are few missing values in train dataset.

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.

Outliers

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 ...even if we consider suv or sumo max seater is 6.

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.

Feature Scaling

Feature scaling is the process of dealing with skewed data, making the feature to set with in a scale of 0-10, by applying the log transformation.

Data Transformation :-

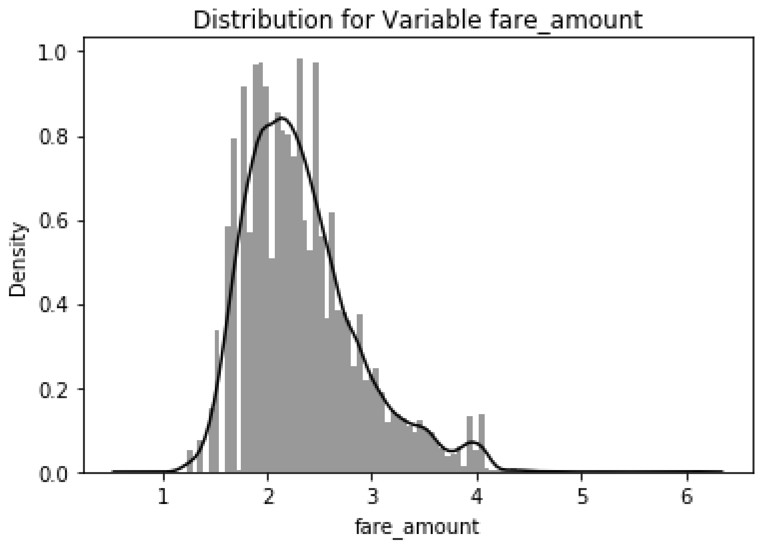
In statistics, data transformation is the application of a deterministic mathematical function to each point in a data set — that is, each data point z_i is replaced with the transformed value $y_i = f(z_i)$, where f is a function. Transforms are usually applied so that the data appear to more closely meet the assumptions of a statistical inference procedure that is to be applied, or to improve the interpretability or appearance of graphs.

Log Transformation:-

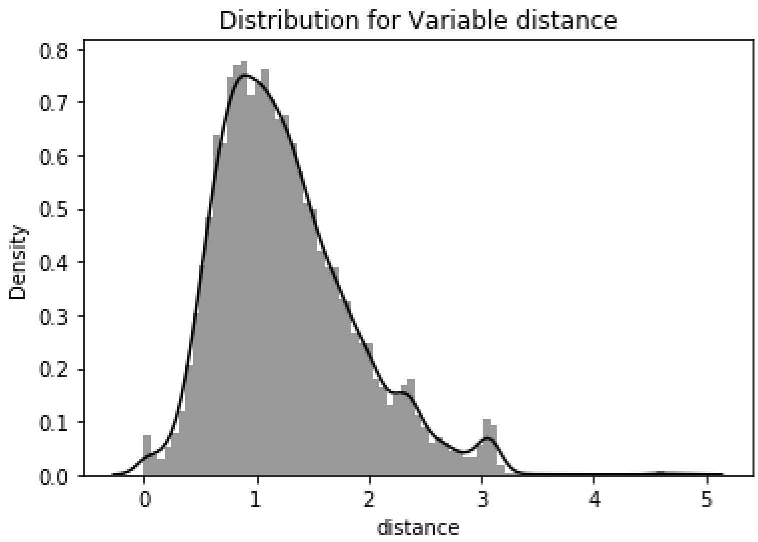
The log transformation can be used to make highly skewed distributions less skewed. This can be valuable both for making patterns in the data more interpretable and for helping to meet the assumptions of inferential statistics.

After applying normalization, the data representation is here below

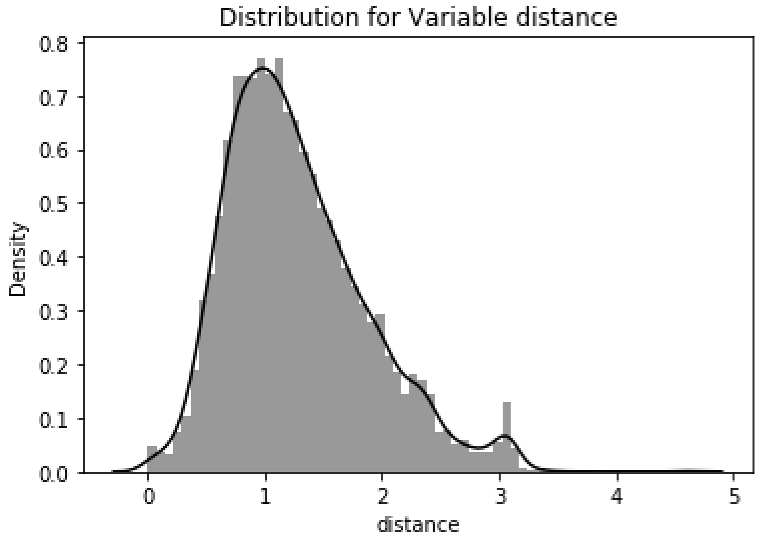
Fare_amount feature in train set



Distance feature in train set



Distance feature in test set



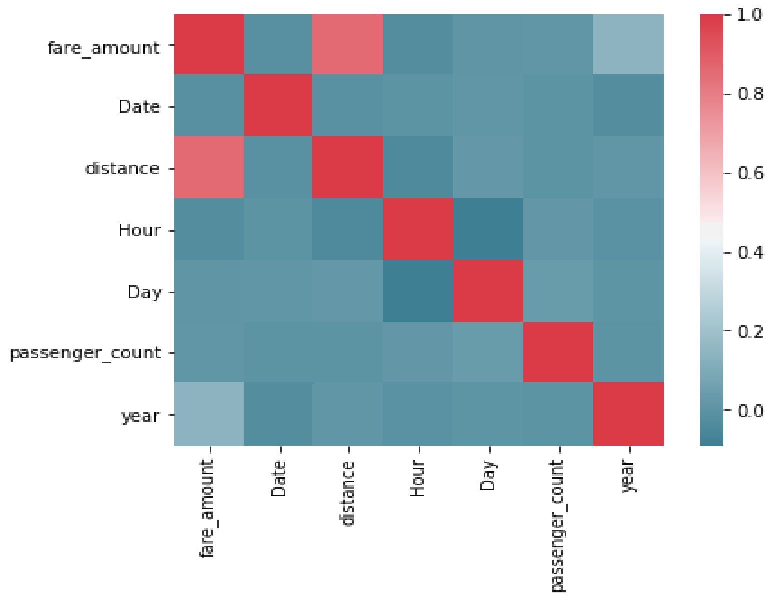
2.1.3 Feature selection

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.



VIF

```
vif(train[,-1])
        Variables      VIF
1 passenger_count 1.000732
2            mnth 1.012996
3              yr 1.012502
4        distance 1.000287
>  vifcor(train[,-1], th = 0.9)
No variable from the 4 input variables has collinearity
problem.

The linear correlation coefficients ranges between:
min correlation ( distance ~ yr ):  -0.001162428
max correlation ( yr ~ mnth ):  -0.1230008

---------- VIFs of the remained variables --------
        Variables      VIF
1 passenger_count 1.000572
2            mnth 1.015883
3              yr 1.015749
4        distance 1.000445
```

What is Regularization?

Regularization is a way to avoid overfitting by penalizing high-valued regression coefficients. In simple terms, it reduces parameters and shrinks (simplifies) the model. This more streamlined, more parsimonious model will likely perform better at predictions. Regularization adds penalties to more complex models and then sorts potential models from least overfit to greatest; The model with the lowest "overfitting" score is usually the best choice for predictive power.

Level 1 regularization.

L1 regularization adds an L1 penalty equal to the absolute value of the magnitude of coefficients. In other words, it limits the size of the coefficients. L1 can yield sparse models (i.e. models with few coefficients); Some coefficients can become zero and eliminated. Lasso regression uses this method.

```
SelectFromModel(estimator=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),
                max_features=None, norm_order=1,
prefit=False, threshold=None)
```

Chapter 3

3.1 Data Modelling

3.1.1 Model selection

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.

3.1.2 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.

Dep. Variable:	fare_amount	**R-squared (uncentered):**	0.987
Model:	OLS	**Adj. R-squared (uncentered):**	0.987
Method:	Least Squares	**F-statistic:**	1.695e+05
Date:	Mon, 04 Nov 2019	**Prob (F-statistic):**	0.00
Time:	19:55:23	**Log-Likelihood:**	-1936.0
No. Observations:	13880	**AIC:**	3884.
Df Residuals:	13874	**BIC:**	3929.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
passenger_count	0.0046	0.002	2.487	0.013	0.001	0.008
year	0.0007	4.91e-06	134.848	0.000	0.001	0.001
Month	0.0039	0.001	5.761	0.000	0.003	0.005
Day	-0.0029	0.001	-2.403	0.016	-0.005	-0.001
Hour	0.0005	0.000	1.279	0.201	-0.000	0.001
distance	0.7686	0.004	198.136	0.000	0.761	0.776

Omnibus:	5376.673	**Durbin-Watson:**	2.015
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	495029.631
Skew:	0.930	**Prob(JB):**	0.00
Kurtosis:	32.198	**Cond. No.**	3.30e+03

3.1.3 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)
predict_lasso=lasso_model.predict(xtest)
```

3.1.4 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=3, 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')
```

3.1.5 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.

Chapter 4

Conclusion

4.1.1Model 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.

4.1.2 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

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

4.1.3 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","nmse","nmae"), 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.These measures are calculated as follows:

"mape": sum(|(t_i - p_i) / t_i|)/N

"nmse": sum((t_i - p_i)^2) / sum((t_i - AVG(Y))^2), where AVG(Y) is the average of the values provided in vector train.y

"nmae": sum(|t_i - p_i|) / sum(|t_i - AVG(Y)|)

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= 92.1 %**

error =7.9 %

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= 92.2 %**

error= 7.8 %

Regr.eval function from R

```
# linear regression model
mape(Test[,1],predict_lm)
# 7.8
regr.eval(Test[,1],predict_lm)
# mae        mse        rmse       mape
#0.18035915 0.08013439 0.28308018 0.07892880
# decision tree
mape(Test[,1],predictions_tree)
# 8.8
regr.eval(Test[,1],predictions_tree)
# mae        mse        rmse       mape
#0.20133555 0.08520206 0.29189392 0.08854953

# random_forest
mape(Test[,1],RF_Predictions)
# 9.8
regr.eval(Test[,1],RF_Predictions)
# mae        mse        rmse       mape
#0.22070787 0.09726554 0.31187423 0.09823838
```

4.1.4 Observations

- As we can observe that the error rate is less in models like lasso and 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

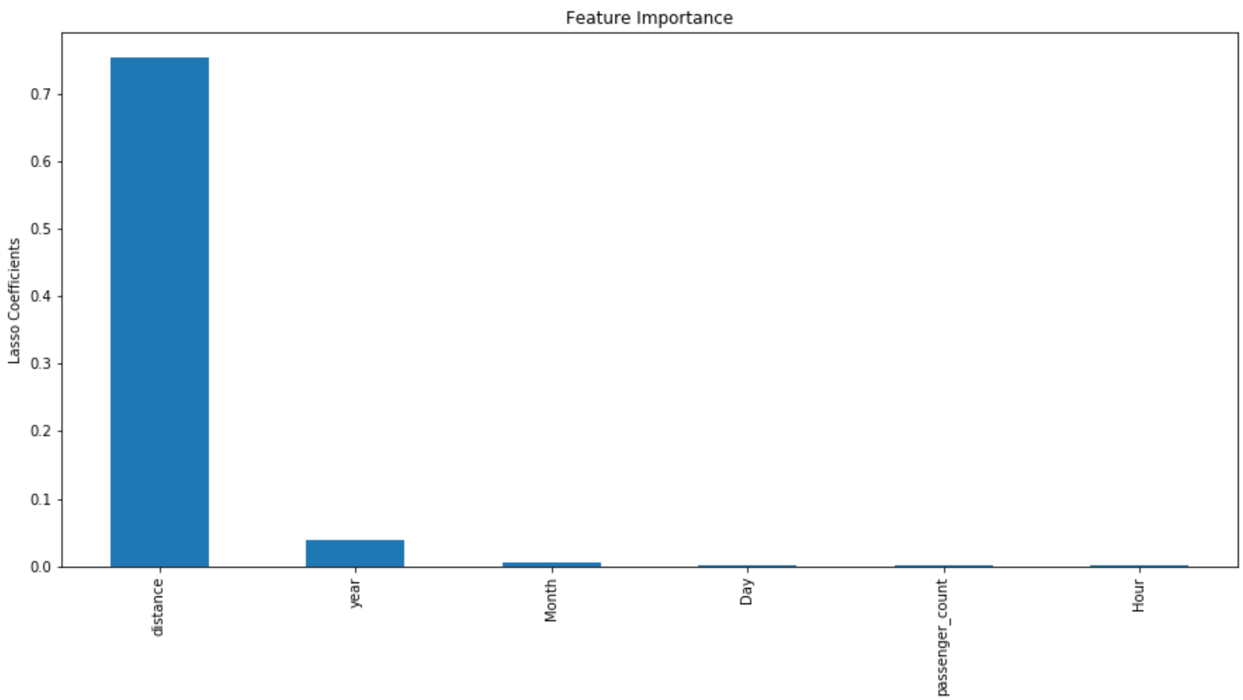
 # NOTICABALLY LASSO AND RANDOM FOREST PERFORM VERY GOOD.....

Appendix A - EXTRA

Variable/ Feature importance.

These are the features explaining more about the target variable.

The below graph shows the strength of the independent features used for predicting fare_amount.

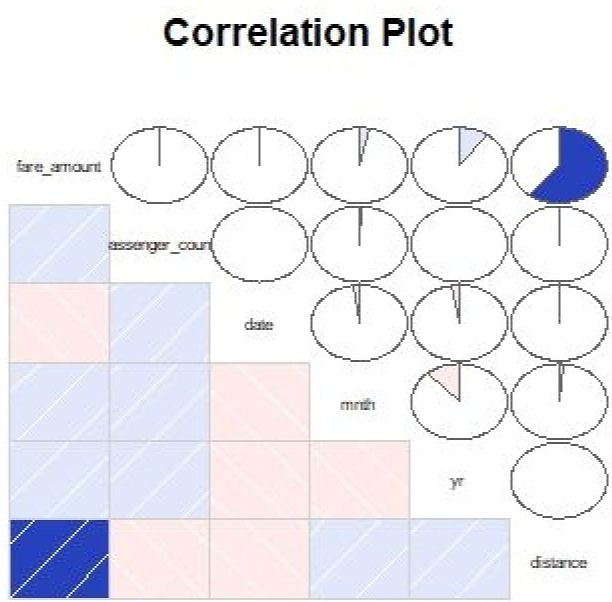


These important features are further tuned and used as an important parameter in model production.

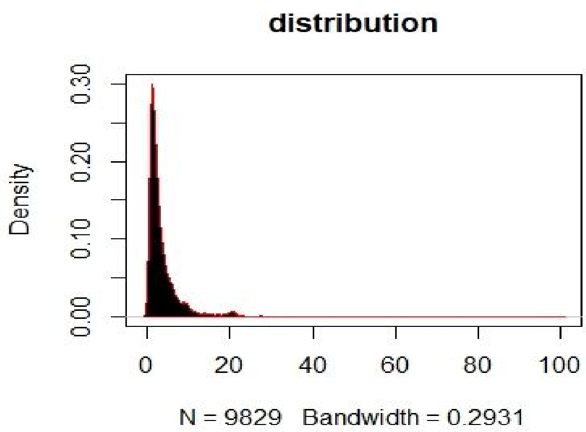
Extra Figures.

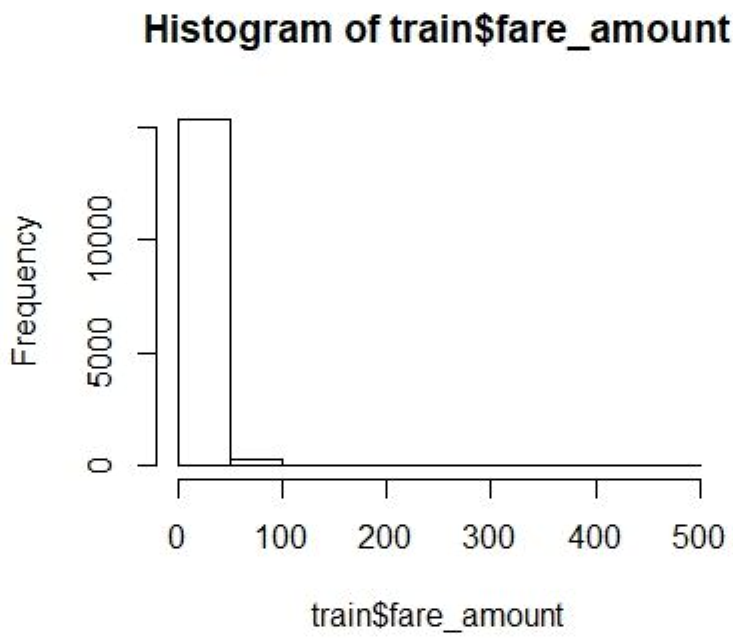
Pasting all the graphs created from R studio.

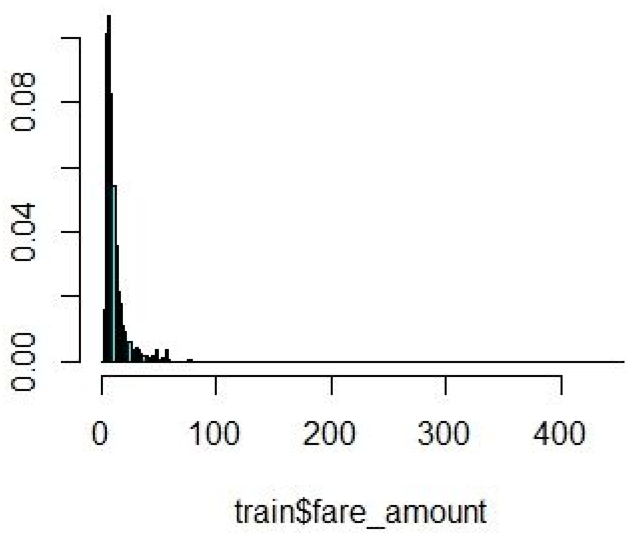
Correlation Plot

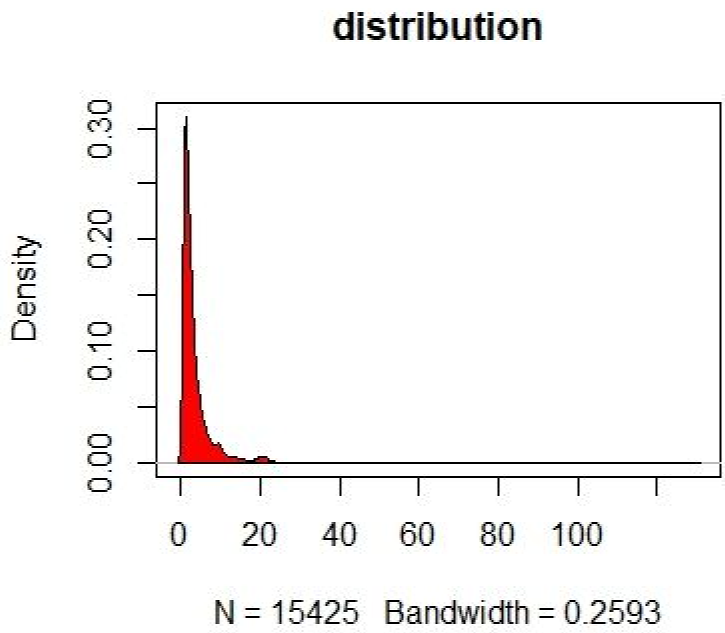


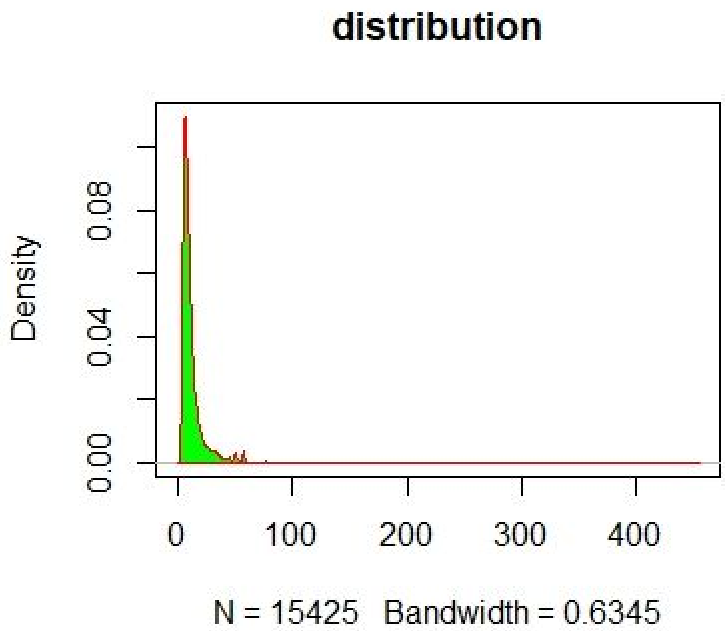
Histograms and density plots



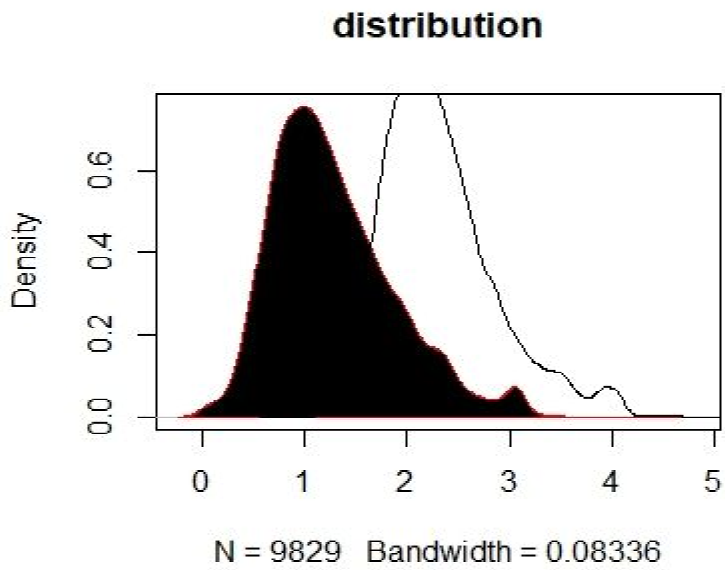

Histogram of train$fare_amount
Frequency
0
5000
10000
0
100
200
300
400
500
train$fare_amount


0.00
0.04
0.08
0
100
200
300
400
train$fare_amount

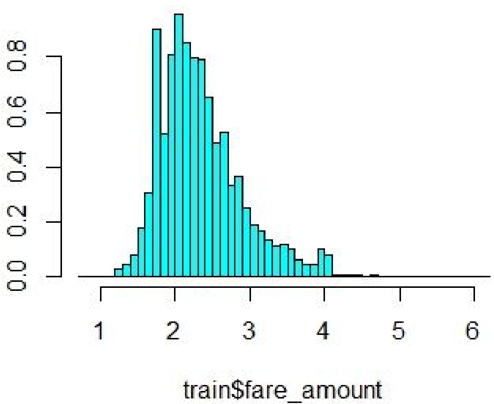

distribution
Density
0.00
0.10
0.20
0.30
0
20
40
60
80
100
N = 15425   Bandwidth = 0.2593

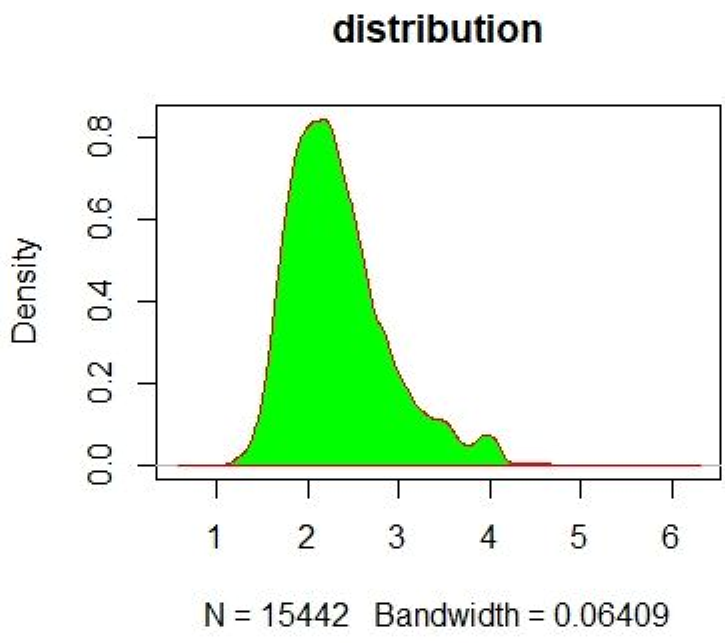

distribution
Density
0.00
0.04
0.08
0
100
200
300
400
N = 15425   Bandwidth = 0.6345

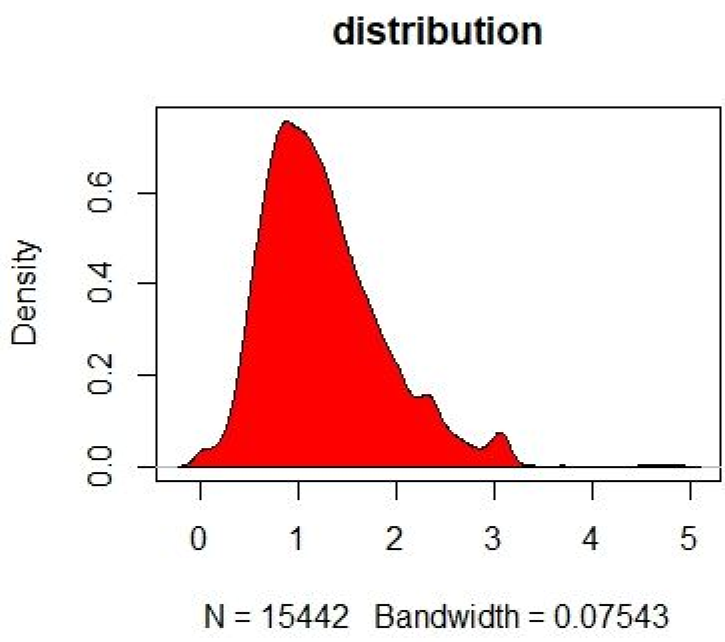
Normalized feature plots

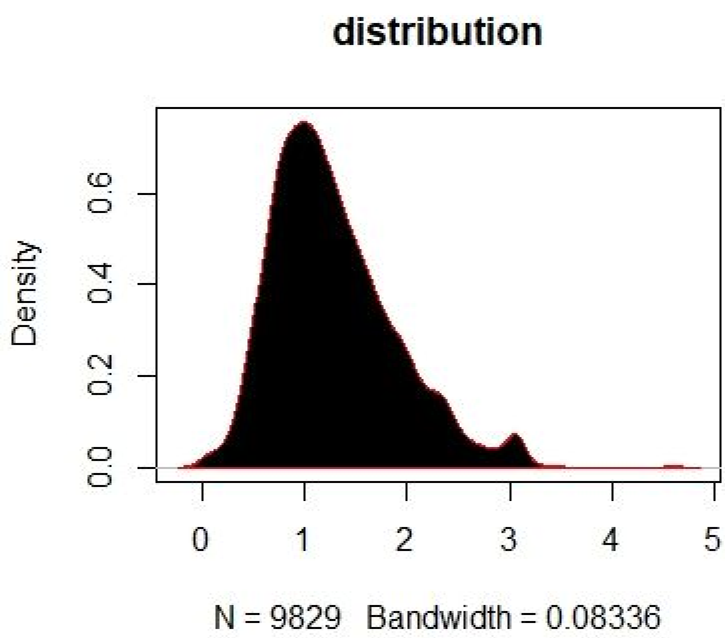


DENSITY and LINE PLOT

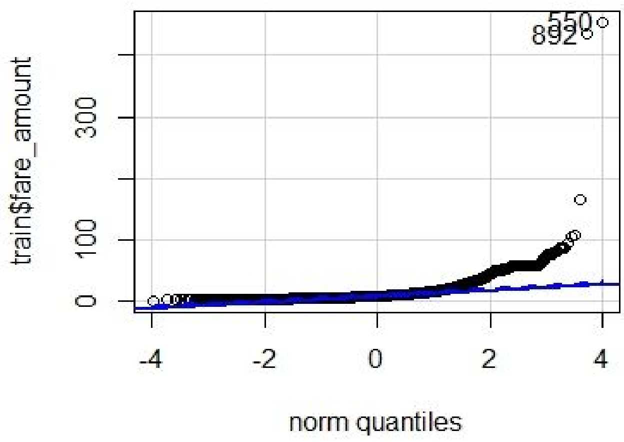



distribution
Density
0.0
0.2
0.4
0.6
0.8
1
2
3
4
5
6
N = 15442 Bandwidth = 0.06409

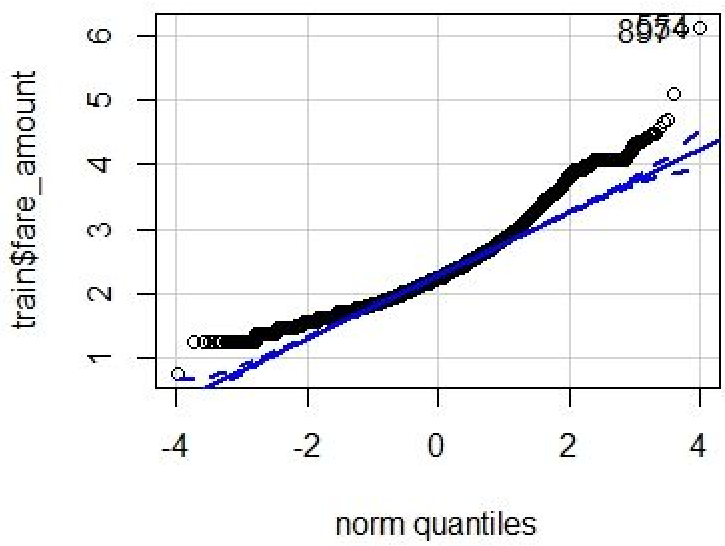

distribution
Density
0.0
0.2
0.4
0.6
0
1
2
3
4
5
N = 15442 Bandwidth = 0.07543



QQplot before Normalization



QQplot after Normalization



PYTHON CODE

```
# importing all necessary libraries.
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import datetime
import scipy.stats
import sklearn
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.ensemble import RandomForestRegressor
# to display all the columns of the dataframe in the
notebook
#pd.pandas.set_option('display.max_columns', None)
from sklearn.linear_model import Lasso
from sklearn.feature_selection import SelectFromModel
import statsmodels.api as sm
from sklearn.tree import DecisionTreeRegressor
# setting up the working directory.
os.chdir("D:/R and PYTHON files/data set/project 3")
os.getcwd()
# loading the data in python environment.
A=pd.read_csv("train_cab.csv",sep=',',dtype={'fare_amount':
np.float},na_values={'fare_amount':'430-'}) # A= train
data.
B=pd.read_csv("test.csv",sep=',')
# B= test data.
data=[A,B]
```

```
for i in data:

i['pickup_datetime']=pd.to_datetime(i['pickup_datetime'],er
rors='coerce')
# Exploratory Data analysis
A.shape, B.shape
A.head(10)
B.head(10)
A.describe()
B.describe()
# getting the details about the data sets.
A.info(), B.info()
# Data cleaning.
A['passenger_count'].value_counts(),B['passenger_count'].va
lue_counts()
A['fare_amount'].sort_values(ascending=False)
sum(A['fare_amount']>453) # considring the fare amount
above 543 as outlier dropping out the observation.
sum(A['fare_amount']==0) # at the same time fare amount
cannot be 0.
# fare amount cannot be less than 1,the passenger count
maxiumum is 6 if considring an SUV,passengr count cannot be
less than one.
sum(A['fare_amount']<1),sum(A['passenger_count']>6),sum(A['
passenger_count']<1) # so filtering out those observation
which satisfies the above condition.
# the filtered observation which are not wanting.
A[A['fare_amount']<1],A[A['passenger_count']>6],A[A['passen
ger_count']<1],A[A['fare_amount']>453]
# Latitudes range from -90 to 90.Longitudes range from -180
to 180. Removing which does not satisfy these ranges
print('pickup_longitude above
```

```
180={}'.format(sum(A['pickup_longitude']>180)))
print('pickup_longitude below
-180={}'.format(sum(A['pickup_longitude']<-180)))
print('pickup_latitude above
90={}'.format(sum(A['pickup_latitude']>90)))
print('pickup_latitude below
-90={}'.format(sum(A['pickup_latitude']<-90)))
print('dropoff_longitude above
180={}'.format(sum(A['dropoff_longitude']>180)))
print('dropoff_longitude below
-180={}'.format(sum(A['dropoff_longitude']<-180)))
print('dropoff_latitude below
-90={}'.format(sum(A['dropoff_latitude']<-90)))
print('dropoff_latitude above
90={}'.format(sum(A['dropoff_latitude']>90)))
# latitude and longitude cannot be comprised of ero value,
so filtering up the values.
for i in
['pickup_longitude','pickup_latitude','dropoff_longitude','
dropoff_latitude']:
    print(i,'equal to 0={}'.format(sum(A[i]==0)))
# Data cleaning. # by above experiments we can say that
most of the data is corrupted and we need to clean it.
A=A.drop(A[A['fare_amount']<1].index,axis=0)
A=A.drop(A[A['fare_amount']>453].index,axis=0)
A=A.drop(A[A['passenger_count']>6].index,axis=0)
A=A.drop(A[A['passenger_count']<1].index,axis=0)
A=A.drop(A[A['pickup_latitude']>90].index,axis=0)
for i in
['pickup_longitude','pickup_latitude','dropoff_longitude','
dropoff_latitude']:
    A=A.drop(A[A[i]==0].index,axis=0)
```

```
# after removing all the wrong data points.
# checking the data after cleaning.
sum(A['fare_amount']<1),sum(A['passenger_count']>6),sum(A['
passenger_count']<1),sum(A['fare_amount']>453)
# checking the data after cleaning.
for i in
['pickup_longitude','pickup_latitude','dropoff_longitude','
dropoff_latitude']:
    print(i,'equal to 0={}'.format(sum(A[i]==0)))
# number of observations reduced from 16067 to 15659
A.shape   # nearly dropped 408 observations.
# FEATURE ENGINEERING = TRANSFORMING THE DATA READY FOR
MODEL.()
# checking for missing values.
print(A.isnull().sum()),print(B.isnull().sum())
missing_val=pd.DataFrame(A.isnull().sum())
missing_val=missing_val.reset_index()
missing_val
index    0
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
missing_val=missing_val.rename(columns={'index':'features',
0:'missing_percentage'})
missing_val
missing_val['missing_percentage']=(missing_val['missing_per
```

```
centage']/len(A))
missing_val
missing_val=missing_val.sort_values('missing_percentage',
ascending = False).reset_index(drop = True)
missing_val
A.head(2)
# removing the NA observations.(as they are very less in
count to impute.)# nearly 77 missing values.
A = A.drop(A[A['fare_amount'].isnull()].index, axis=0)
A = A.drop(A[A['passenger_count'].isnull()].index, axis=0)
#removing datetime missing values rows
A = A.drop(A[A['pickup_datetime'].isnull()].index, axis=0)
print(A.shape)
print(A['pickup_datetime'].isnull().sum())
(15581, 7)
0
A.isnull().sum(),A.shape # now there are Zero missing
values.
A.describe()
B.describe()
A['passenger_count'].unique()
sum(A['passenger_count']==1.3)
A=A.drop(A[A['passenger_count']==1.3].index,axis=0) #
passenger count cannot be 1.3 so dropping it out.
# MAKING NEW FEATURES.
A['pickup_datetime'] = pd.to_datetime(A['pickup_datetime'],
format='%Y-%m-%d %H:%M:%S UTC')  # note:- A=train, B=test.
B['pickup_datetime'] = pd.to_datetime(B['pickup_datetime'],
format='%Y-%m-%d %H:%M:%S UTC')  # note:- A=train, B=test.
# separating the Pickup_datetime column into separate field
like year, month, day of the week, etc
```

```
A['year'] = A['pickup_datetime'].dt.year
A['Month'] = A['pickup_datetime'].dt.month
A['Date'] = A['pickup_datetime'].dt.day
A['Day'] = A['pickup_datetime'].dt.dayofweek
A['Hour'] = A['pickup_datetime'].dt.hour
A['Minute'] = A['pickup_datetime'].dt.minute
# lets do same for B dataset which is test dataset.
# note:- A=train, B=test.
B['year'] = B['pickup_datetime'].dt.year
B['Month'] = B['pickup_datetime'].dt.month
B['Date'] = B['pickup_datetime'].dt.day
B['Day'] = B['pickup_datetime'].dt.dayofweek
B['Hour'] = B['pickup_datetime'].dt.hour
B['Minute'] =B['pickup_datetime'].dt.minute
A.info(),A.shape   # note:- A=train
B.info(),B.shape    # note:-  B=test.
A['year'].unique(),B['year'].unique()
A['Month'].unique(),B['Month'].unique()
A['Date'].unique(),B['Date'].unique()
A['Day'].unique(),A['Day'].value_counts(),B['Day'].unique()
,B['Day'].value_counts()  # note:- A=train, B=test.
A['Hour'].unique(),A['Hour'].value_counts(),B['Hour'].uniqu
e(),B['Hour'].value_counts()  # note:- A=train, B=test.
A['Minute'].unique(),A['Minute'].value_counts(),B['Minute']
.unique(),B['Minute'].value_counts()  # note:- A=train,
B=test.
#As we know that we have given pickup longitute and
latitude values and same for drop.
#So we need to calculate the distance Using the haversine
formula and we will create a new variable called distance
from math import radians, cos, sin, asin, sqrt
```

```
def haversine(a):
    lon1=a[0]
    lat1=a[1]
    lon2=a[2]
    lat2=a[3]
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1,
lon2, lat2])

    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) *
sin(dlon/2)**2
    c =  2 * asin(sqrt(a))
    # Radius of earth in kilometers is 6371
    km = 6371* c
    return km
A['distance'] =
A[['pickup_longitude','pickup_latitude','dropoff_longitude'
,'dropoff_latitude']].apply(haversine,axis=1)   # note:-
A=train, B=test.
B['distance'] =
B[['pickup_longitude','pickup_latitude','dropoff_longitude'
,'dropoff_latitude']].apply(haversine,axis=1)    # note:-
A=train, B=test.
A['distance'].sort_values(ascending=False),B['distance'].so
rt_values(ascending=False) # note:- A=train, B=test.
```

```
# distance travelled cannot be Zero , so removing those
observation with ZERO distance travelled.
# at the same time how a person can travel 4000 to 5000 km
in cab, thats not possible and removing it.
sum(A['distance']==0),sum(A['distance']>130)


sum(B['distance']==0),sum(B['distance']>130)
A=A.drop(A[A['distance']==0].index,axis=0)
A=A.drop(A[A['distance']>130].index,axis=0)
B=B.drop(B[B['distance']==0].index,axis=0)
A.describe()
B.describe()
# deleting the features.
deletingthefeatures = ['pickup_datetime',
'pickup_longitude', 'pickup_latitude','dropoff_longitude',
'dropoff_latitude','Minute']
A = A.drop(deletingthefeatures, axis = 1)
deleting_the_features = ['pickup_datetime',
'pickup_longitude', 'pickup_latitude','dropoff_longitude',
'dropoff_latitude','Minute']
B = B.drop(deleting_the_features, axis = 1)
A.head(), A.shape
B.head(), B.shape
# converting the data in required data type.
A['passenger_count'] = A['passenger_count'].astype('int64')
A['year'] = A['year'].astype('int64')
A['Month'] = A['Month'].astype('int64')
A['Date'] = A['Date'].astype('int64')
A['Day'] = A['Day'].astype('int64')
A['Hour'] = A['Hour'].astype('int64')
# converting the data in required data type.
```

```
B['passenger_count'] = B['passenger_count'].astype('int64')
B['year'] = B['year'].astype('int64')
B['Month'] = B['Month'].astype('int64')
B['Date'] = B['Date'].astype('int64')
B['Day'] = B['Day'].astype('int64')
B['Hour'] = B['Hour'].astype('int64')

A.dtypes
B.dtypes
green
# DATA VISUALIZATIONS.
plt.hist(A['passenger_count'],color='green') # there are
lot of single passenger travellers, followed by 2,5,3,4,6.
5
# Count plot on passenger count
plt.figure(figsize=(10,5))
sns.countplot(x="passenger_count", data=A)
test data.
# Count plot on passenger count
plt.figure(figsize=(10,5))
sns.countplot(x="passenger_count", data=B)  # passenger
count for test data.
# relationship between passenger count and fare amount.
plt.figure(figsize=(10,5))
plt.scatter(x="passenger_count",y="fare_amount",
data=A,color='blue')
plt.xlabel('No. of passengers')
plt.ylabel('Fare_amount')
plt.show()
6
# relationship between date and fare amount.
plt.figure(figsize=(15,6))
```

```
plt.scatter(x="Date",y="fare_amount", data=A,color='blue')
plt.xlabel('Date')
plt.ylabel('Fare_amount')
plt.show()
ours.  .
# number of cabs with respect to hours.  .
plt.hist(A["Hour"])
s..
# number of cabs with respect to hours..
plt.figure(figsize=(15,7))
A.groupby(A["Hour"])['Hour'].count().plot(kind="bar")
plt.show()
hour
# realationship between fare and hour
plt.figure(figsize=(10,5))
plt.scatter(x="Hour",y="fare_amount", data=A,color='blue')
plt.xlabel('Hour')
plt.ylabel('Fare_amount')
plt.show()
# realationship between fare and day
plt.figure(figsize=(10,5))
plt.scatter(x="Day",y="fare_amount", data=A,color='blue')
plt.xlabel('Day')
plt.ylabel('Fare_amount')
plt.show()
# realationship between fare and distance
plt.figure(figsize=(10,5))
plt.scatter(x="distance",y="fare_amount",
data=A,color='blue')
plt.xlabel('distance')
plt.ylabel('fare')
plt.show()
```

```
# checking the distribution of features...(fare_amount and
Distance), rest of the features are date, time, year ,
hour...
#Normality check of training data is uniformly distributed
or not-

for i in ['fare_amount', 'distance']:
    print(i)
    sns.distplot(A[i],bins='auto',color='black')
    plt.title("Distribution for Variable "+i)
    plt.ylabel("Density")
    plt.show()
fare_amount
plt.hist(A['fare_amount'])
plt.hist(A['distance'])
plt.hist(A['distance'])
#since skewness of target variable is high, apply log
transform to reduce the skewness-
A['fare_amount'] = np.log1p(A['fare_amount'])

#since skewness of distance variable is high, apply log
transform to reduce the skewness-
A['distance'] = np.log1p(A['distance'])
A.head()
fare_amount    passenger_count    year Month    Date Day
Hour distance
0    1.704748 1    2009 6    15    0    17    0.708412
1    2.884801 1    2010 1    5     1    16    2.246029
2    1.902108 2    2011 8    18    3    0     0.871095
3    2.163323 1    2012 4    21    5    4     1.334809
4    1.840550 1    2010 3    9     1    7     1.098331
for i in ['fare_amount', 'distance']:
```

```
    print(i)
    sns.distplot(A[i],bins='auto',color='black')
    plt.title("Distribution for Variable "+i)
    plt.ylabel("Density")
    plt.show()
#Normality check for test data is uniformly distributed or
not-

sns.distplot(B['distance'],bins='auto',color='black')
plt.title("Distribution for Variable "+i)
plt.ylabel("Density")
plt.show()
#since skewness of distance variable is high, apply log
transform to reduce the skewness-
B['distance'] = np.log1p(B['distance'])
B.head()
B.head()
sns.distplot(B['distance'],bins='auto',color='black')
plt.title("Distribution for Variable "+i)
plt.ylabel("Density")
plt.show()
numerical_val=['fare_amount','Date','distance','Hour','Day'
,'passenger_count','year']
plot ##
# FEATURE SELECTION     #### FILTER METHOD ####    ##
pearson correlation plot ##
A_corr=A.loc[:,numerical_val]
f, ax = plt.subplots(figsize=(7, 5))
correlation_matrix=A_corr.corr()
#correlation plot
sns.heatmap(correlation_matrix,mask=np.zeros_like(correlati
on_matrix,dtype=np.bool),cmap=sns.diverging_palette(220,10,
```

```
as_cmap=True),square=True,ax=ax).get_figure().savefig('pyth
onheat_map.png')
 it is highly nevetively correlated with other features.
# by above observation removing feature "Date" it is highly
nevetively correlated with other features.
y=A['fare_amount']
x=A.drop(["fare_amount"],axis=1)
# splitting the train data set for model building and
finding accuracy.
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.
1)
## LASSO REGRESSION SELECTION ## ## level 1 ##
#### FEATURE SELECTION BY EMBEDDED METHOD   ####
## LASSO REGRESSION SELECTION ## ## level 1 ##
sel_ = SelectFromModel(Lasso(alpha=0.005, random_state=0))
# remember to set the seed, the random state in this
function
sel_.fit(xtrain, ytrain)
sel_.get_support()
# let's print the number of total and selected features

# this is how we can make a list of the selected features
selected_feat = xtrain.columns[(sel_.get_support())]

# let's print some stats
print('total features: {}'.format((xtrain.shape[1])))
print('selected features: {}'.format(len(selected_feat)))
print('features with coefficients shrank to zero:
{}'.format(
    np.sum(sel_.estimator_.coef_ == 0)))
...SELECTION.
selected_feat   ## the below features are selected by
```

```
EMBEDDED METHOD...SELECTION.
pd.Series(selected_feat).to_csv('selected_features.csv',
index=False)
features = pd.read_csv('selected_features.csv',
header=None)
features = [x for x in features[0]]
# reduce the train and test set to the desired features

xtrain = xtrain[features]
xtest = xtest[features]
# preparing the new test case data...
B=B.drop(['Date'],axis=1)
Btest=B
.shape
Btest.shape
,ytrain.shape,ytest.shape
xtrain.shape,xtest.shape,ytrain.shape,ytest.shape
REGRESSION ##
### MODEL BUILDING ###
## LINEAR REGRESSION ##
LRmodel=sm.OLS(ytrain,xtrain).fit()
LRmodel.summary()
predictionLR
predictionLR=LRmodel.predict(xtest)
predictionLR.head()
newpredB
newpredB=LRmodel.predict(Btest)
newpredB.head()
### LASSO REGRESSION ###
### LASSO REGRESSION ###
lasso_model = Lasso(alpha=0.005, random_state=0)
lasso_model.fit(xtrain, ytrain)
```

```
predict_lasso
predict_lasso=lasso_model.predict(xtest)
predict_lasso
ytest.head()
predict_lassoB=lasso_model.predict(Btest)
predict_lassoB
10
### DECISION TREE REGRESSOR ###
DTR=DecisionTreeRegressor(max_depth=10).fit(xtrain,ytrain)
DTR
DTR
prediction_DTR=DTR.predict(xtest)
prediction_DTR
prediction_DTRB
prediction_DTRB=DTR.predict(Btest)
prediction_DTRB
20
### RANDOM FOREST REGRESSOR ###
RF=RandomForestRegressor(n_estimators = 20).fit(xtrain,
ytrain)
RFprediction=RF.predict(xtest)
RFprediction
RFpredictionB=RF.predict(Btest)
RFpredictionB
##### 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

.1
```

```
## performance of linear regression model.
mape(ytest,predictionLR)                   ### Accuracy=
92.1
.4
## performance of lasso regression model.
mape(ytest,predict_lasso)                  ### Accuracy=
92.4
2.0
## performance of decision tree regression model.
mape(ytest,prediction_DTR)                         ###
Accuracy= 92.0
2
## performance of random forest regression model.
mape(ytest,RFprediction)                           ###
Accuracy= 92.2
#### MODEL SELECTION ####
 ## ALL MODELS PERFORM WELL
    # NOTICABALLY LASSO AND RANDOM FOREST PERFORM VERY
GOOD.....
n is further used for model deployment .
#  let's look at the feature importance (best model=
lasso), so selecting lasso co-efficients

importance = pd.Series(np.abs(lasso_model.coef_.ravel()))
importance.index = features
importance.sort_values(inplace=True, ascending=False)
importance.plot.bar(figsize=(15,7))
plt.ylabel('Lasso Coefficients')
plt.title('Feature Importance')             ##### this
section is further used for model deployment .
#### writing back best prediction (lasso regression
results) results to the TEST data set (B)
```

```
B['fare_amount']=predict_lassoB
B.head()
plt.scatter
#### lets visualize the predicted fare amount in the test
data.
# realationship between fare and distance
plt.figure(figsize=(10,5))
plt.scatter(x="distance",y="fare_amount",
data=B,color='red')
plt.xlabel('distance')
plt.ylabel('fare')
plt.show()
```

R CODE

```
rm(list=ls())
setwd("D:/R and PYTHON files/data set/project 3")
getwd()

# loading all required librares.
install.packages (c("ggplot2", "corrgram", "DMwR", "caret", "randomForest",
"unbalanced", "C50", "dummies", "e1071", "Information",
                    "MASS", "rpart", "gbm", "ROSE", 'sampling',
'DataCombine', 'inTrees'))

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


# exploring the data.
str(train)
str(test)
summary(train)
summary(test)
head(train,5)
head(train,5)

# 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  #############
# fare amount cannot be less than one
# considring fare amount 453 as max and removing all the fare amount
greater than 453, as chances are
# very less of fare amount having 4000 and 5000 ...etc
train[which(train$fare_amount < 1 ),]
nrow(train[which(train$fare_amount < 1 ),]) # to show the count i.e.,5
train = train[-which(train$fare_amount < 1 ),]  # removing those values.
train[which(train$fare_amount>453),]
nrow(train[which(train$fare_amount >453 ),]) # to show the count i.e., 2
```

```
train = train[-which(train$fare_amount >453 ),]  # removing those values.
# passenger count cannot be Zero
# even if we consider suv max seat is 6, so removing passenger count
greater than 6.
train[which(train$passenger_count < 1 ),]
nrow(train[which(train$passenger_count < 1 ),]) # to show count, that is 58
train=train[-which(train$passenger_count < 1 ),] # removing the values
train[which(train$passenger_count >6 ),]
nrow(train[which(train$passenger_count >6 ),]) # to show count, that is 20
train=train[-which(train$passenger_count >6 ),] # removing the values
# Latitudes range from -90 to 90.Longitudes range from -180 to 180.
# Removing which does not satisfy these ranges.
print(paste('pickup_longitude above
180=',nrow(train[which(train$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$dropoff_latitude > 90 ),])))
train = train[-which(train$pickup_latitude > 90),] # removing one data
point
# 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$dropoff_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
# removing those data points.
train=train[-which(train$pickup_longitude == 0 ),]
train=train[-which(train$dropoff_longitude == 0),]

# checking for missing values.
sum(is.na(train))
```

```
sum(is.na(test))
train=na.omit(train) # we have removed the missing values...as they are
less,,..likely 50 to 60 missing values.
sum(is.na(train))

# deriving the new features using pickup_datetime and coordinated provided.
# new features will be year,month,day_of_week,hour
# Convert pickup_datetime from factor to date time
train$pickup_datetime=as.Date(train$pickup_datetime)
pickup_time = strptime(train$pickup_datetime,format='%Y-%m-%d %H:%M:%S
UTC')
train$date = as.integer(format(train$pickup_date,"%d"))# Monday = 1
train$mnth = as.integer(format(train$pickup_date,"%m"))
train$yr = as.integer(format(train$pickup_date,"%Y"))
#train$min = as.integer(format(train$pickup_date,"%M"))
#train$day=as.integer(as.POSIXct(train$pickup_datetime),abbreviate=F)

# for test data set.
test$pickup_datetime=as.Date(test$pickup_datetime)
pickup_time = strptime(test$pickup_datetime,format='%Y-%m-%d %H:%M:%S UTC')
test$date = as.integer(format(test$pickup_date,"%d"))# Monday = 1
test$mnth = as.integer(format(test$pickup_date,"%m"))
test$yr = as.integer(format(test$pickup_date,"%Y"))
# outlier
#library(ggplot2)
#pl1 = ggplot(train,aes(x = factor(passenger_count),y = fare_amount))
#pl1 + geom_boxplot(outlier.colour="red", fill = "grey"
,outlier.shape=18,outlier.size=1, notch=FALSE)+ylim(0,100)

# deriving the new feature, distance from the given coordinates.
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 #1000 is used to convert to meters
}
train$distance =
haversine(train$pickup_longitude,train$pickup_latitude,train$dropoff_longit
ude,train$dropoff_latitude)
test$distance =
haversine(test$pickup_longitude,test$pickup_latitude,test$dropoff_longitude
,test$dropoff_latitude)


# removing the features, which were used to create new features.
train = subset(train,select =
-c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude,pick
up_datetime))
test = subset(test,select =
-c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude,pick
up_datetime))
str(train)
summary(train)
nrow(train[which(train$distance ==0 ),])
nrow(test[which(test$distance==0 ),])
nrow(train[which(train$distance >130 ),]) # considering the distance 130 as
max and considering rest as outlier.
nrow(test[which(test$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 ),]

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

#Correlation analysis for numeric variables
library(corrgram)
```

```
corrgram(train[,numeric_index],upper.panel=panel.pie, main = "Correlation
Plot")

#removing date
# pickup_weekdat has p value greater than 0.05
train = subset(train,select=-date)
#remove from test set
test = subset(test,select=-date)


## feature scaling ##
library(car)
library(MASS)
qqPlot(train$fare_amount) # qqPlot, it has a x values derived from gaussian
distribution, if data is distributed normally then the sorted data points
should lie very close to the solid reference line
truehist(train$fare_amount) # truehist() scales the counts to give an
estimate of the probability density.
lines(density(train$fare_amount)) # lines() and density() functions to
overlay a density plot on histogram

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="green",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")
```

```
#print('fare_amount')
#train[,'fare_amount'] = (train[,'fare_amount'] -
min(train[,'fare_amount']))/
 # (max(train[,'fare_amount'] - min(train[,'fare_amount'])))
#train[,'distance'] = (train[,'distance'] - min(train[,'distance']))/
 # (max(train[,'distance'] - min(train[,'distance'])))
#test[,'distance'] = (test[,'distance'] - min(test[,'distance']))/
 # (max(test[,'distance'] - min(test[,'distance'])))

###check multicollearity
 library(usdm)
 vif(train[,-1])
 vifcor(train[,-1], th = 0.9)
 #No variable from the 4 input variables has collinearity problem.
 #The linear correlation coefficients ranges between:
 #min correlation ( mnth ~ passenger_count ):  -0.001868147
 #max correlation ( yr ~ mnth ):  -0.1091115

# ---------- VIFs of the remained variables --------
#   Variables      VIF
# 1 passenger_count 1.000583
# 2            mnth 1.012072
# 3              yr 1.012184
# 4        distance 1.000681

 ## to make sure that we dont have any missing values
 sum(is.na(train))
 train=na.omit(train)
```

```
# model building
# preparing the data
set.seed(1200)
Train.index = sample(1:nrow(train), 0.9 * nrow(train))
Train = train[ Train.index,]
Test  = train[-Train.index,]
#head(Test[,2:5],5)
TestData=test
# linear regression
linear_model=lm(fare_amount~.,data=Train)
summary(linear_model)
predict_lm=predict(linear_model,Test[,2:5])
predict_test=predict(linear_model,TestData)

# decision tree regressor
library(rpart)
DT=rpart(fare_amount~.,data=Train,method="anova")
predictions_tree=predict(DT,Test[,2:5])
predictions_test=predict(DT,TestData)
summary(DT)

# random forest regressor
library(randomForest)
random_model = randomForest(fare_amount~ ., Train, importance = TRUE, ntree
= 500)

#Extract rules fromn random forest
#transform rf /object to an inTrees' format
library(inTrees)
treeList = RF2List(random_model)

#Extract rules
rules= extractRules(treeList, Train[,2:5])

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

#Predict test data using random forest model
```

```
RF_Predictions = predict(random_model, Test[,2:5])
RF_test=predict(random_model, TestData)

# saving the results in hard disk
#write(capture.output(summary(random_model)),"RF_summary.txt")

## accuracy check
#defining the function (to find the error percentage)
mape=function(av,pv){
  mean(abs((av-pv)/av))*100 #av=actual value and pv= predicted value
}
library(DMwR)
# linear regression model
mape(Test[,1],predict_lm)
# 7.8
regr.eval(Test[,1],predict_lm)
# mae        mse        rmse       mape
#0.18035915 0.08013439 0.28308018 0.07892880
# decision tree
mape(Test[,1],predictions_tree)
# 8.8
regr.eval(Test[,1],predictions_tree)
# mae        mse        rmse       mape
#0.20133555 0.08520206 0.29189392 0.08854953

# random_forest
mape(Test[,1],RF_Predictions)
# 9.8
regr.eval(Test[,1],RF_Predictions)
# mae        mse        rmse       mape
#0.22070787 0.09726554 0.31187423 0.09823838
```