**PREDICTING BIKE RENTAL COUNT**

**Mukesh Kumar sahu**

**25/1/2020**

**Contents:**

1. **[INTRODUCTION 3](#_Toc30426252)** 
   1. **[PROBLEM STATEMENT 3](#_Toc30426252)**
   2. **[DATA OVERVIEW 3](#_Toc30426252)**
2. **[METHODOLOGY 4](#_Toc30426252)**

**[2.1 PRE- PROCESSING 4](#_Toc30426252)**

**[2.1.1 DATA EXPLORATION 4](#_Toc30426252)**

**[2.1.2 MISSING VALUE ANALYSIS 4](#_Toc30426252)**

**[2.1.3 OUTLIER ANALYSIS 5](#_Toc30426252)**

**[2.1.4 DATA VISUALIZATION 5](#_Toc30426252)**

**[2.1.4.1 DISTRIBUTION OF THE NUMERIC VARIABLE 6](#_Toc30426252)**

**[2.1.4.2 DISTRIBUTION OF CONTINUOUS VARIABLES WITH RESPECT TO TARGET VARIABLE 6](#_Toc30426252)**

**[2.1.4.3 IMPACT OF THE CATEGORICAL VARIABLE WITH RESPECT TO TARGET VARIABLE 7](#_Toc30426252)**

**[A) BIKE RENTED WITH RESPECTED TO TEMPERATURE AND HUMIDITY](#_Toc30426252) 9**

**B) BIKE RENTED WITH RESPECTED TO TEMPERATURE AND WINDSPEED 10**

**C) BIKE RENTED WITH RESPECT TO TEMPERATURE AND SEASON 10**

**2.1.5 FEATURE SELECTION 11**

**2.1.5.1 CORRELATION MATRIX AND PLOT 11**

**2.1.5.2 ANALYSIS OF VARIANCE 12**

**2.1.5.3 DIMENSION REDUCTION 12**

**2.1.6 FEATURE SCALING 12**

**2.2 MODEL DEVELOPMENT 13**

**2.2.1 LINEAR REGRESSION 13**

**2.2.2 DECISION TREE 14  
 2.2.3 RANDOM FOREST 14**

**2.3 HYPERPARAMETER TUNING 14**

**2.3.1 TUNING PARAMETERS 14**

* + - 1. **RANDOM SEARCH 14**

**2.3.1.2 Grid Search 15**

**3. MODEL EVALUATION 16**

**3.1 EVALUATION METRICS 16**

### MAPE (Mean Absolute Percentage Error) 16

### R-SQUARED 16

### RMSE ( Root Mean Square Error) 16

### 

**3.2 MODEL SELECTION 17**

**APPENDIX A - PYTHON CODE 18**

**APPENDIX B – R CODE 37**

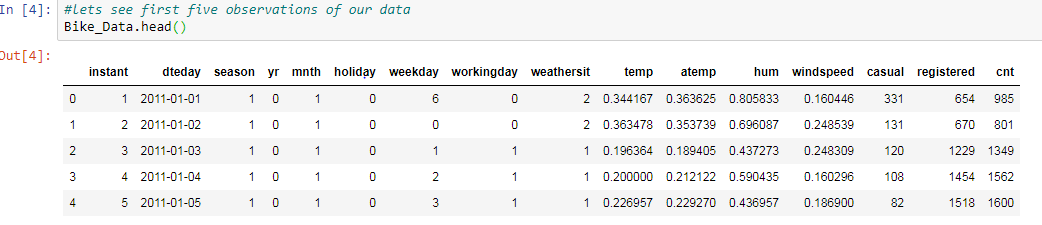
1. **INTRODUCTION**
2. **PROBLEM STATEMENT:**

In our project we need to predict bike rental count on daily basis based on the season and environmental settings.

**1.2 DATA OVERVIEW:**

We have 16 variables and 731 observations. In that 13 variables are independent and 3 dependent variables.

Lets have a look at the first 5 observation of the data:

Here casual, registered and count are our dependent variables.

COUNT = CASUAL+REGISTERED

Remaining all are independent variables.

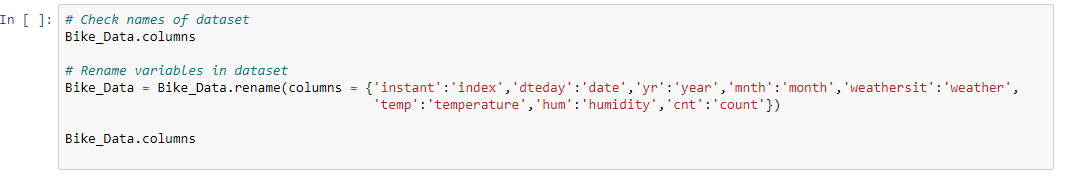
**2. METHODOLOGY**

**2.1 PRE-PROCESSING:**

Data preprocessing is a data mining techniques which transforms raw data into an understandable format .Data goes through series of steps during preprocessing. They are data cleaning, Data visualization, Data transformation, Data reduction.

**2.1.1 DATA EXPLORATION:**

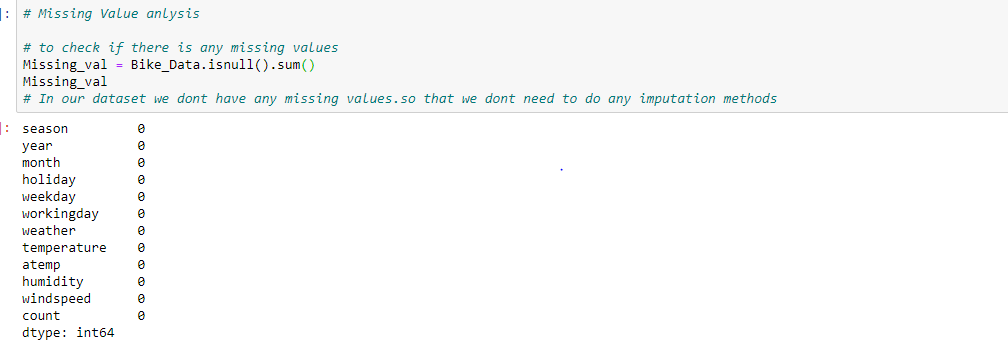
We need to check dimensions of the data, data types of the data, summary of the data .so that we can get good understandings about the data and also identify the target variable.



For our convenience, I changed some shortcut variable name into understandable format. The above picture is self-explanatory.

**2.1.2 MISSING VALUE ANALYSIS:**

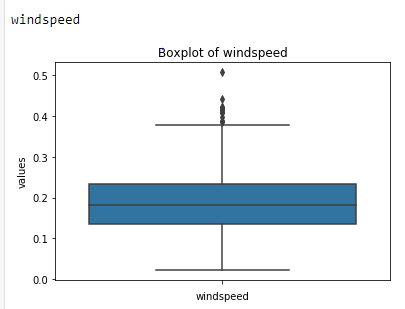
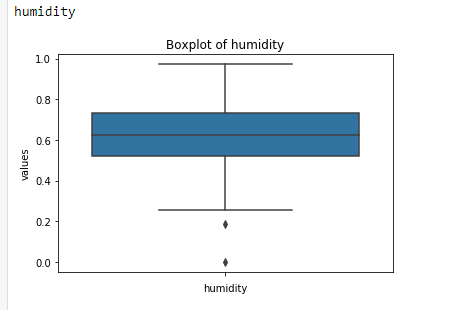
Missing values are the data which is not present in the particular variable or observations. It may happen due to human error, or it may mark as an optional during the survey. If the data set contains missing values which is above 30%, either we need to drop the column or that particular observation. In our dataset we don’t have any missing values but in real world problems there is always some missing values. We need to impute those missing values either it is classification or regression problems.



**2.1.3 OUTLIER ANALYSIS:**

Basically outliers are the values which are lying far away from the remaining variables which may lead biased towards the higher value which results in the performance of our model. So that we need to treat the outliers.

Here outliers are detected using boxplot method. We have inliers in humidity and outliers in wind speed other than that we don’t have any outliers.so, In our case we saved minimum value to the inliers and maximum values to the outliers.so that we no need to loss the data and also we can increase the performance of our model. How much data we feed that much accuracy our model will be.

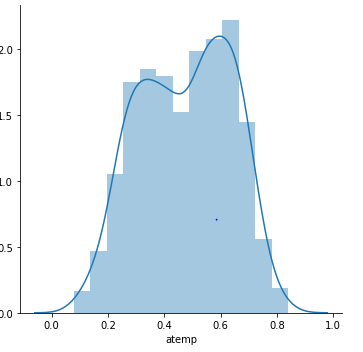
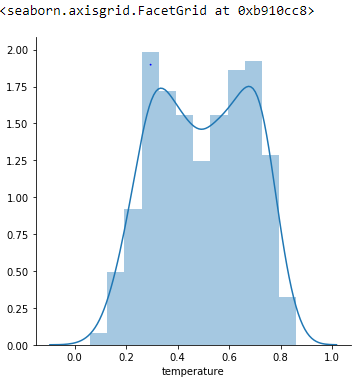


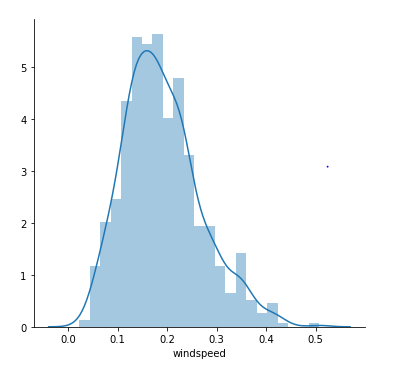
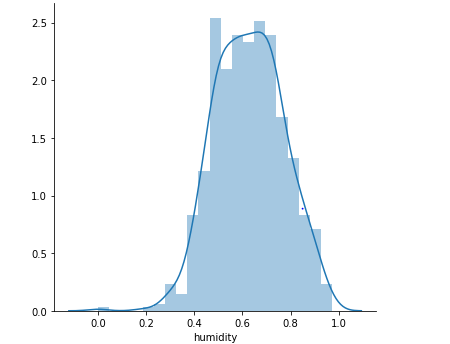
**2.1.4 DATA VISUALIZATION:**

Data visualization is the easy method to understand the data. It will gives clear idea of our data and also impact of dependent variables.

**2.1.4.1 DISTRIBUTION OF THE NUMERIC VARIABLE:**

Distribution plot helps us to know the distribution of the data and makes us easily understandable. So that it used in the both R and Python languages. Here we used histogram for our visualization.



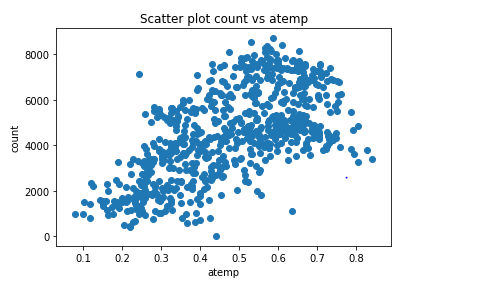
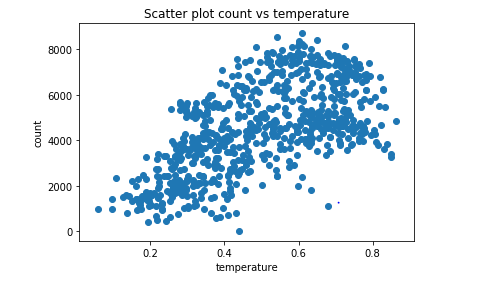


Based on the above graphs, we can clearly say that our temperature and atemp variables are carrying same information. So that we are going to drop any one of the variables for our further analysis.

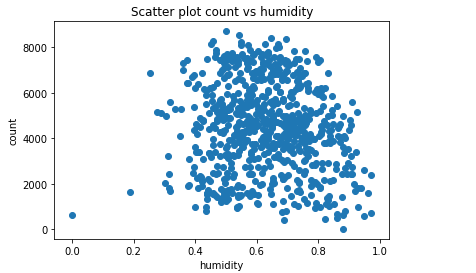
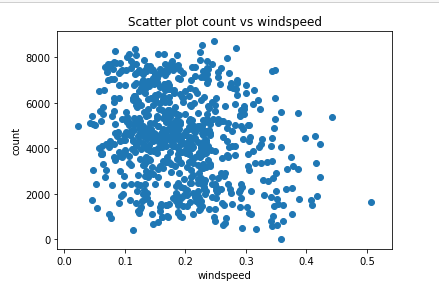
**2.1.4.2 DISTRIBUTION OF CONTINUOUS VARIABLES WITH RESPECT TO TARGET VARIABLE:**

Here we used scatterplot for our visualization. First plot is between temperature and count variables. Both the temperature and atemp variables are mostly similar to each other.

From the below plot, we say that the temperature increases our bike rental count also increases.



Second plot is between wind speed and humidity with respect to the count.

  
From the above plot we say that bike Rental count is not affected by humidity and windspeed.

**2.1.4.3 IMPACT OF THE CATEGORICAL VARIABLE WITH RESPECT TO TARGET VARIABLE:**

* **Count vs Season:-** Bike rental is higher in the season 3 which is fall and low in season 1 which is spring.
* **Count vs Year :** Bike rental is higher in the year 1 which is 2012.

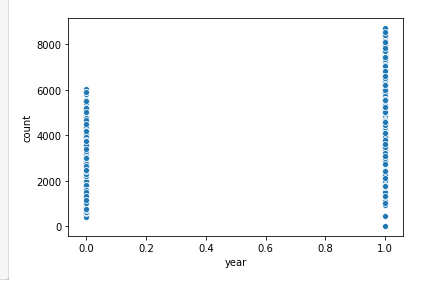
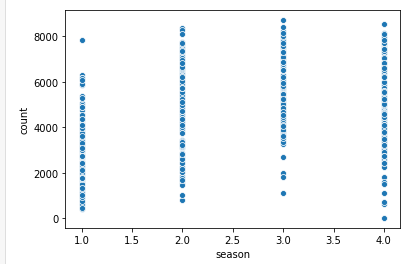
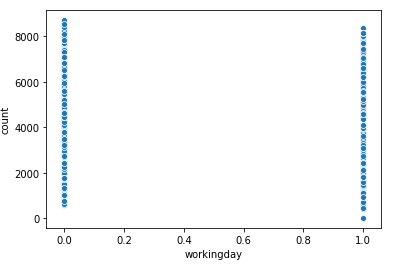
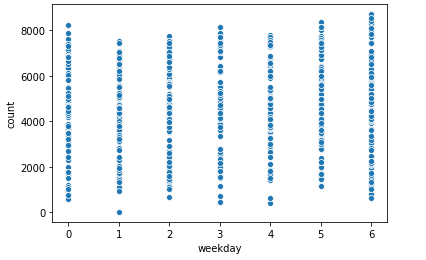
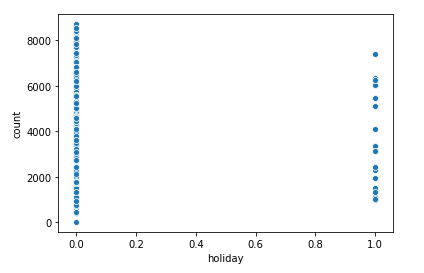
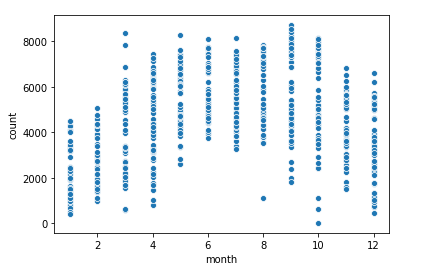


Fig:- Count VS season Fig:- Count VS Year

* **COUNT VS WEEKDAY:** Bike rental count is high in 5 which is Friday and low in 1 which is Sunday.
* **COUNT VS WORKING DAY:** Bike rental count is high in 1 which is working day and low in 0 which is holiday.

   
  
 Fig:- Count VS weekday Fig:- Count VS working day

* **COUNT VS MONTH**: Bike rental is higher in the month of 8 which is in august and low in 1 which is in January.
* **COUNT VS HOLIDAY:**Bike rental count is higher in 0 which is holiday and low in 1 which is working day.

   
Fig:- Count VS Month Fig:- Count VS Holiday

* **COUNT VS WEATHER:**Bike rental count is high in 1 which is clear, few clouds, partly cloudy and there is no bikes rental in 4.

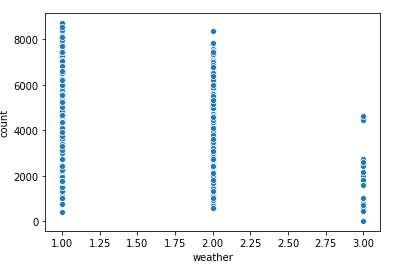
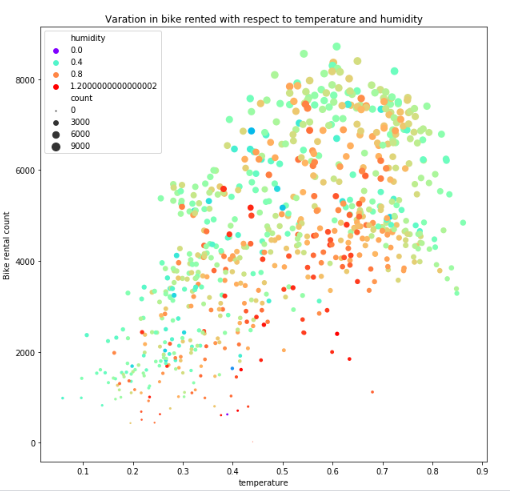


Fig:- Count VS Weather

### BIKE RENTED WITH RESPECTED TO TEMPERATURE AND HUMIDITY:



From the above plot, we can say that bike rental count is higher when the

* temperature is between 0.4 to 0.8
* humidity less than 0.8

### BIKE RENTED WITH RESPECT TO TEMPERATURE AND WINDSPEED:

### 

* From this above plot, we say that bike rental is higher when the
  + - * + temperature is between 0.4 to 0.8
        + humidity is less than 0.8
        + windspeed is less than 0.2

### BIKE RENTED WITH RESPECT TO TEMPERATURE AND SEASON:

### 

* From the plot we say that, bike rental count is higher when the
  + - * + temperature is between 0.4 to 0.8
        + season was 2 and 3
        + weather was from 1 and 2

### FEATURE SELECTION:

We can use correlation analysis for numerical variables and Analysis of Variance for categorical variables. It shows correlation between the two variables. So that if two variables carrying same information can be removed.

### 2.1.5.1 CORRELATION MATRIX AND PLOT

### 

### 

From the above plot, we saw that temperature and atemp variables are carrying same information. Hence there is no need to continue with both variables. So we need to remove atemp variable. So we need to drop any one of the variables. Here I am dropping atemp variables.

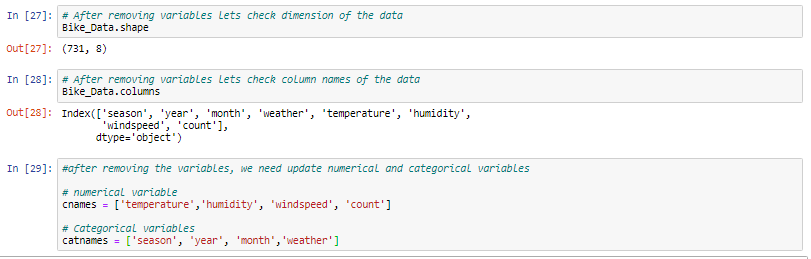
### 2.1.5.2 ANALYSIS OF VARIANCE:

### 

From the above diagram, holiday, weekday, and working day these variables has p-value which is higher than 0.05.so that we need to drop these variables.

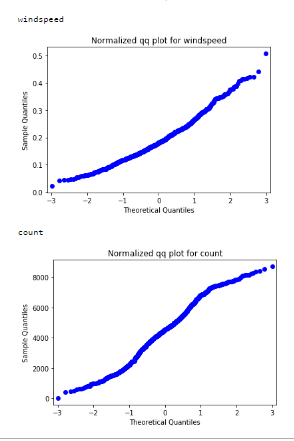
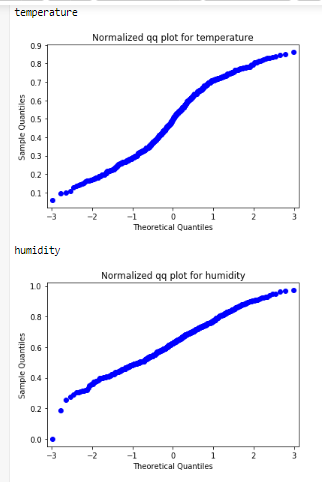
### 2.1.5.3 DIMENSION REDUCTION:

After the feature selection, we have only these 8 variables. They are mentioned in the below diagram.

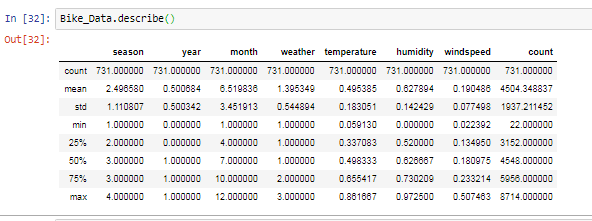


### FEATURE SCALING:

In our dataset, all our continuous variables are already normalized. So we don’t need to need any scaling methods to scale the data. Though we can use qqplot, summary, distribution of the data to see the normality.



* Summary of the data after feature selection and dimension reduction.



### MODEL DEVELOPMENT:

Next we need to split the data into train and test data and build a model using train data to predict the output using test data. Different models to be built and the model which gives more accurate values must be selected.



### LINEAR REGRESSION:

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things:

* + - 1. Does a set of predictor variables do a good job in predicting an outcome (dependent) variable?
      2. Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates– impact the outcome variable?

These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. We trained our model in both R and Python and predicted in these languages using test data.

### DECISION TREE:

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes .The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

### RANDOM FOREST:

### A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees, which involves training each decision tree on a different data sample where sampling is done with replacement. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. The higher no of trees in the random forest will give higher no of accuracy, so in random forest we can go for multiple trees. It can handle large no of independent variables without variable deletion and it will give the estimates that what variables are important.

### HYPERPARAMETER TUNING

In statistics, hyperparameter is a parameter from a prior distribution; it captures the prior belief before data is observed. In any machine learning algorithm, these parameters need to be initialized before training a model. Choosing appropriate hyperparameters plays a crucial role in the success of good model. Since it makes a huge impact on the learned model.

### TUNING PARAMETERS:

We will explore two different methods for optimizing hyperparameters:

* Grid Search
* Random Search

### 2.3.1.1 RANDOM SEARCH:

Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. In this search pattern, random combinations of parameters are considered in every iteration. The chances of finding the optimal parameter are comparatively higher in random search because of the random search pattern where the model might end up being trained on the optimised parameters without any aliasing.

### 2.3.1.2 Grid Search

Grid search is a technique which tends to find the right set of hyperparameters for the particular model. Hyperparameters are not the model parameters and it is not possible to find the best set from the training data. In this tuning technique, we simply build a model for every combination of various hyperparameters and evaluate each model. The model which gives the highest accuracy will be selected.

## 

## **3. MODEL EVALUATION**

**3.1 EVALUATION METRICS:**

In regression problems, we have three important metrics, they are

* MAPE(Mean Absolute Percentage Error)
* R- SQUARED
* RMSE(Root Mean Square Error)

### MAPE (Mean Absolute Percentage Error)

MAPE is a measure of prediction accuracy of a forecasting method. It measures accuracy in terms of percentage. Lower value of MAPE indicates better fit.

### R-SQUARED

R-squared is basically explains the degree to which input variable explain the variation of the output. In simple words Rsquared tells how much variance of dependent variable explained by the independent variable. It is a measure if goodness of fit in regression line. Higher values of R-square indicate better fit.

### RMSE (Root Mean Square Error)

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors).Residuals are a measure of how far from the regression line data points are, RMSE is a measure of how spread out these residuals are. As the square root of a variance, RMSE can be interpreted as the standard deviation of the unexplained variance and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate better fit.

### FINAL PREDICTED MODELS IN PYTHON:

### 



### MODEL SELECTION:

From the predicted output in R and Python, the Random Forest model can have explained almost 87% of the predictor matches with the target variable. The values of the random forest model is mentioned below.

### MAPE = 16.755

* **R-SQUARED =0.85**
* **RMSE = 717.37**

**APPENDIX A - PYTHON CODE**

**PREDICTING BIKE RENTAL COUNT**

**# Load the required libraries for analysis of data**

**import os**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**# Set working directory**

**os.chdir("D:/mukesh")**

**# lets Check working directory**

**os.getcwd()**

**# Load the data**

**Bike\_Data = pd.read\_csv("day.csv")**

# Explore the data

*# Check the dimensions(no of rows and no of columns)*

Bike\_Data.shape

# Check names of dataset

Bike\_Data.columns

# Rename variables in dataset

Bike\_Data = Bike\_Data.rename(columns = {'instant':'index','dteday':'date','yr':'year','mnth':'month','weathersit':'weather',

'temp':'temperature','hum':'humidity','cnt':'count'})

Bike\_Data.columns

**#lets see first five observations of our data**

**Bike\_Data.head()**

**# Lets see the datatypes of the given data**

**Bike\_Data.dtypes**

**# lets Check summary of the dataset**

**Bike\_Data.describe()**

**# Variable Identification**

**Bike\_Data['count'].dtypes**

**#lets drop some variables because it doesnot carry any useful information**

**Bike\_Data = Bike\_Data.drop(['casual','registered','index','date'],axis=1)**

**# Lets check dimensions of data after removing some variables**

**Bike\_Data.shape**

**# Continous Variables**

**cnames= ['temperature', 'atemp', 'humidity', 'windspeed', 'count']**

**# Categorical variables-**

**cat\_cnames=['season', 'year', 'month', 'holiday', 'weekday', 'workingday','weather']**

# EDA or Data Preprocessing

*# Missing Value anlysis*

​

*# to check if there is any missing values*

Missing\_val **=** Bike\_Data.isnull().sum()

Missing\_val

*# In our dataset we dont have any missing values.so that we dont need to do any imputation methods*

# Outlier Analysis

# Lets save copy of dataset before preprocessing

df = Bike\_Data.copy()

Bike\_Data = df.copy()

# Using seaborn library, we can viualize the outliers by plotting box plot

for i in cnames:

print(i)

sns.boxplot(y=Bike\_Data[i])

plt.xlabel(i)

plt.ylabel("values")

plt.title("Boxplot of "+i)

plt.show()

# From boxplot we can see inliers in humidity and outliers in windspeed

# Lets detect and remove outliers

for i in cnames:

print(i)

# Quartiles and IQR

q25,q75 = np.percentile(Bike\_Data[i],[25,75])

IQR = q75-q25

# Lower and upper limits

Minimum = q25 - (1.5 \* IQR)

print(Minimum)

Maximum = q75 + (1.5 \* IQR)

print(Maximum)

Minimum = Bike\_Data.loc[Bike\_Data[i] < Minimum ,i]

Maximum = Bike\_Data.loc[Bike\_Data[i] > Maximum ,i]

#we substituted minimum values for inliers and maximum values for outliers.

#from that we removed all the outliers.

# after replacing the outliers,let us plot boxplot for understanding

for i in cnames:

print(i)

sns.boxplot(y=Bike\_Data[i])

plt.xlabel(i)

plt.ylabel("values")

plt.title("Boxplot of "+i)

plt.show()

# Univariate Analysis

# temperature

sns.FacetGrid(Bike\_Data , height = 5).map(sns.distplot,'temperature').add\_legend()

#normally distributed

# humidity

sns.FacetGrid(Bike\_Data , height = 5).map(sns.distplot,'humidity').add\_legend()

#normally distributed

# windspeed

sns.FacetGrid(Bike\_Data , height = 5).map(sns.distplot,'windspeed').add\_legend()

#normally distributed

#atemp

sns.FacetGrid(Bike\_Data , height = 5).map(sns.distplot,'atemp').add\_legend()

#normally distributed

# count

sns.FacetGrid(Bike\_Data , height = 5).map(sns.distplot,'count').add\_legend()

#normally distributed

# Lets check impact of continous variables on target variable

# count vs temperatur

plt.scatter(x='temperature',y='count',data=Bike\_Data)

plt.title('Scatter plot count vs temperature')

plt.ylabel('count')

plt.xlabel('temperature')

plt.show()

#temperature is directly proportional to each other

#as temperature increases bike rental count also increases

**# count vs atemp**

**plt.scatter(x='atemp',y='count',data=Bike\_Data)**

**plt.title('Scatter plot count vs atemp')**

**plt.ylabel('count')**

**plt.xlabel('atemp')**

**plt.show()**

**#as atemp increases bike rental count also increases**

**# count vs humidity**

**plt.scatter(x='humidity',y='count',data=Bike\_Data)**

**plt.title('Scatter plot count vs humidity')**

**plt.ylabel('count')**

**plt.xlabel('humidity')**

**plt.show()**

**# Apart from humidity,Bike rental count does not get affected**

**# count vs windspeed**

**plt.scatter(x='windspeed',y='count',data=Bike\_Data)**

**plt.title('Scatter plot count vs windspeed')**

**plt.ylabel('count')**

**plt.xlabel('windspeed')**

**plt.show()**

**# Apart from windspeed,Bike rental count does not get affected**

**#for categorical variables**

**# SEASON**

**print(Bike\_Data.groupby(['season'])['count'].sum())**

**#based on the season, bike rental count is high in season 3 which is fall and low in season 1 which is spring**

**#lets visualize the count using scatterplot**

**sns.scatterplot(x='season',y='count',data = Bike\_Data)**

**# YEAR**

**print(Bike\_Data.groupby(['year'])['count'].sum())**

**#based on the year, bike rental count is high in the year 1 which is 2012**

**#lets visualize the count using scatterplot**

**sns.scatterplot(x='year',y='count',data = Bike\_Data)**

**# MONTH**

**print(Bike\_Data.groupby(['month'])['count'].sum())**

**#Based on the month, Bike rental count is high in 8 which is in august and low in 1 which is in january**

**#lets visualize the count using scatterplot**

**sns.scatterplot(x='month',y='count',data = Bike\_Data)**

**#HOLIDAY**

**print(Bike\_Data.groupby(['holiday'])['count'].sum())**

**#Based on the holiday, bike rental count is high in 0 which is holiday and low in 1 which is working day**

**#lets visualize the count using scatterplot**

**sns.scatterplot(x='holiday',y='count',data = Bike\_Data)**

**# WEAKDAY**

**print(Bike\_Data.groupby(['weekday'])['count'].sum())**

**#Based on the weakday, bike rental count is high in 5 which is friday and low in 0 which is sunday**

**#lets visualize the count using scatterplot**

**sns.scatterplot(x='weekday',y='count',data = Bike\_Data)**

**# WORKINGDAY**

**print(Bike\_Data.groupby(['workingday'])['count'].sum())**

**#Based on the workingday, Bike rental count is high in 1 which is working day and low in 0 which is hoiday**

**#lets visualize the count using scatterplot**

**sns.scatterplot(x='workingday',y='count',data = Bike\_Data)**

**#WEATHER**

**print(Bike\_Data.groupby(['weather'])['count'].sum())**

**#Based n the weather bike rental count is higher in 1 which clear,few clouds,partly cloudy and there is no bikes rental in 4**

**#lets visualize the count using scatterplot**

**sns.scatterplot(x='weather',y='count',data = Bike\_Data)**

**# Bike rented with respected to tempeature and humidity**

**f, ax = plt.subplots(figsize=(10, 10))**

**sns.scatterplot(x="temperature", y="count",**

**hue="humidity", size="count",**

**palette="rainbow",sizes=(1, 100), linewidth=0,**

**data=Bike\_Data,ax=ax)**

**plt.title("Varation in bike rented with respect to temperature and humidity")**

**plt.ylabel("Bike rental count")**

**plt.xlabel("temperature")**

**# based on the below plot we know that bike rental is higher when the**

**#temperature is between 0.4 to 0.8**

**#humidity less than 0.8**

**#Bikes rented with respect to temperature and windspeed**

**f, ax = plt.subplots(figsize=(10,10))**

**sns.scatterplot(x="temperature", y="count",**

**hue="windspeed", size="humidity",**

**palette="rainbow",sizes=(1, 100), linewidth=0,**

**data=Bike\_Data,ax=ax)**

**plt.title("Varation in bike rented with respect to temperature and windspeed")**

**plt.ylabel("Bike rental count")**

**plt.xlabel("temperature")**

**#based on the below plot we know that bike rental is higher when the**

**#temperature is between 0.4 to 0.8**

**#humidity is less than 0.8**

**#windspeed is less than 0.2**

**# Bikes rented with respect to temperature and season**

**f, ax = plt.subplots(figsize=(10,10))**

**sns.scatterplot(x="temperature", y="count",**

**hue="season", size="count",style= "weather",**

**palette="rainbow",sizes=(1, 100), linewidth=0,**

**data=Bike\_Data,ax=ax)**

**plt.title("Varation in bike rented with respect to temperature and season")**

**plt.ylabel("Bike rental count")**

**plt.xlabel("Normalized temperature")**

**#based on the below plot we know that bike rental is higher when the**

**#temperature is between 0.4 to 0.8**

**#season was 2 and 3**

**#weather was from 1 and 2**

# Feature Selection

*# Lets save dataset after outlier analysis*

df **=** Bike\_Data.copy()

Bike\_Data **=** df.copy()

# Correlation analysis

# Correlation matrix continuous variables

Bike\_corr= Bike\_Data.loc[:,cnames]

# Generate correlation matrix

corr\_matrix = Bike\_corr.corr()

(print(corr\_matrix))

# Set the width and hieght of the plot

f, ax = plt.subplots(figsize=(15,15))

#Plot using seaborn library

sns.heatmap(corr\_matrix, mask=np.zeros\_like(corr\_matrix, dtype=np.bool), cmap=sns.diverging\_palette(220, 10, as\_cmap=True),

square=True, ax=ax,annot=True)

plt.title("Correlation Plot For Numeric or Continous Variables")

#from the below plot,we came to know that both temperature and atemp variables are carrying almost same information

#hence there is no need to continue with both variables.

#so we need to drop any one of the variables

#here I am dropping atemp variables

import statsmodels.api as sm

from statsmodels.formula.api import ols

# ANOVA test for categorical variables

for i in cat\_cnames:

mod = ols('count' + '~' + i, data = Bike\_Data).fit()

aov\_table = sm.stats.anova\_lm(mod, typ = 2)

print(aov\_table)

# Removing the variables which has p-value > 0.05 and correlated variable

Bike\_Data = Bike\_Data.drop(['atemp', 'holiday','weekday','workingday'], axis=1)

# After removing variables lets check dimension of the data

Bike\_Data.shape

#after removing the variables, we need update numerical and categorical variables

# numerical variable

cnames = ['temperature','humidity', 'windspeed', 'count']

# Categorical variables

catnames = ['season', 'year', 'month','weather']

# Feature scaling

*#based on the details of the attributes given, all the numerical variables are normalised*

*#lets visualise the numerical variables to see normality*

**for** i **in** cnames:

print(i)

sm.qqplot(Bike\_Data[i])

plt.title("Normalized qq plot for " **+**i)

plt.show()

Bike\_Data.describe()

#we confirmed the normalized data based on the qqplot and summary of the data

# Model Development

*# Load Required libraries for model development*

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.metrics **import** mean\_squared\_error

**from** sklearn.metrics **import** r2\_score

**from** sklearn.linear\_model **import** LinearRegression

**from** sklearn.tree **import** DecisionTreeRegressor

**from** sklearn.ensemble **import** RandomForestRegressor

**from** sklearn **import** metrics

*#In Regression problems, we can't pass directly categorical variables.*

*#so we need to convert all categorical variables into dummy variables*

​

df **=** Bike\_Data

Bike\_Data **=** df

​

*# Converting categorical variables to dummy variables*

Bike\_Data **=** pd.get\_dummies(Bike\_Data,columns**=**catnames)

​

Bike\_Data.shape

Bike\_Data.columns

# Lets Divide the data into train and test set

X= Bike\_Data.drop(['count'],axis=1)

y= Bike\_Data['count']

# Divide data into train and test sets

X\_train,X\_test,y\_train,y\_test= train\_test\_split(X,y,test\_size=.20)

# Function for Error metrics to calculate the performance of model

def MAPE(y\_true,y\_prediction):

mape= np.mean(np.abs(y\_true-y\_prediction)/y\_true)\*100

return mape

# Linear Regression model

# Import libraries

import statsmodels.api as sm

# Linear Regression model

LinearRegression\_model= sm.OLS(y\_train,X\_train).fit()

print(LinearRegression\_model.summary())

# Model prediction on train data

LinearRegression\_train= LinearRegression\_model.predict(X\_train)

# Model prediction on test data

LinearRegression\_test= LinearRegression\_model.predict(X\_test)

# Model performance on train data

MAPE\_train= MAPE(y\_train,LinearRegression\_train)

# Model performance on test data

MAPE\_test= MAPE(y\_test,LinearRegression\_test)

# r2 value for train data

r2\_train= r2\_score(y\_train,LinearRegression\_train)

# r2 value for test data-

r2\_test=r2\_score(y\_test,LinearRegression\_test)

# RMSE value for train data

RMSE\_train = np.sqrt(metrics.mean\_squared\_error(y\_train,LinearRegression\_train))

# RMSE value for test data

RMSE\_test = np.sqrt(metrics.mean\_squared\_error(y\_test,LinearRegression\_test))

print("Mean Absolute Precentage Error for train data="+str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="+str(MAPE\_test))

print("R^2\_score for train data="+str(r2\_train))

print("R^2\_score for test data="+str(r2\_test))

print("RMSE for train data="+str (RMSE\_train))

print("RMSE for test data="+str(RMSE\_test))

Error\_MetricsLT = {'Model Name': ['Linear Regression'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

LinearRegression\_Results = pd.DataFrame(Error\_MetricsLT)

LinearRegression\_Results

#Decision tree model

# Lets Build decision tree model on train and test data

from sklearn.tree import DecisionTreeRegressor

# Decision tree for regression

DecisionTree\_model= DecisionTreeRegressor(max\_depth=2).fit(X\_train,y\_train)

# Model prediction on train data

DecisionTree\_train= DecisionTree\_model.predict(X\_train)

# Model prediction on test data

DecisionTree\_test= DecisionTree\_model.predict(X\_test)

# Model performance on train data

MAPE\_train= MAPE(y\_train,DecisionTree\_train)

# Model performance on test data

MAPE\_test= MAPE(y\_test,DecisionTree\_test)

# r2 value for train data

r2\_train= r2\_score(y\_train,DecisionTree\_train)

# r2 value for test data

r2\_test=r2\_score(y\_test,DecisionTree\_test)

# RMSE value for train data

RMSE\_train = np.sqrt(metrics.mean\_squared\_error(y\_train,DecisionTree\_train))

# RMSE value for test data

RMSE\_test = np.sqrt(metrics.mean\_squared\_error(y\_test,DecisionTree\_test))

print("Mean Absolute Precentage Error for train data="+str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="+str(MAPE\_test))

print("R^2\_score for train data="+str(r2\_train))

print("R^2\_score for test data="+str(r2\_test))

print("RMSE for train data="+str(RMSE\_train))

print("RMSE for test data="+str(RMSE\_test))

Error\_MetricsDT = {'Model Name': ['Decision Tree'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

DecisionTree\_Results = pd.DataFrame(Error\_MetricsDT)

DecisionTree\_Results

# Random Search CV In Decision Tree

# Import libraries

from sklearn.model\_selection import RandomizedSearchCV

RandomDecisionTree = DecisionTreeRegressor(random\_state = 0)

depth = list(range(1,20,2))

random\_search = {'max\_depth': depth}

# Lets build a model using above parameters on train data

RandomDecisionTree\_model= RandomizedSearchCV(RandomDecisionTree,param\_distributions= random\_search,n\_iter=5,cv=5)

RandomDecisionTree\_model= RandomDecisionTree\_model.fit(X\_train,y\_train)

# Lets look into best fit parameters

best\_parameters = RandomDecisionTree\_model.best\_params\_

print(best\_parameters)

# Again rebuild decision tree model using randomsearch best fit parameter ie

# with maximum depth = 7

RDT\_best\_model = RandomDecisionTree\_model.best\_estimator\_

print(RDT\_best\_model)

# Prediction on train data

RDT\_train = RDT\_best\_model.predict(X\_train)

# Prediction on test data

RDT\_test = RDT\_best\_model.predict(X\_test)

# Lets check Model performance on both test and train using error metrics of regression like mape,rsquare value

# MAPE for train data

MAPE\_train= MAPE(y\_train,RDT\_train)

# MAPE for test data

MAPE\_test= MAPE(y\_test,RDT\_test)

# Rsquare for train data

r2\_train= r2\_score(y\_train,RDT\_train)

# Rsquare for test data

r2\_test=r2\_score(y\_test,RDT\_test)

# RMSE value for train data

RMSE\_train = np.sqrt(metrics.mean\_squared\_error(y\_train,RDT\_train))

# RMSE value for test data

RMSE\_test = np.sqrt(metrics.mean\_squared\_error(y\_test,RDT\_test))

# Lets print the results

print("Best Parameter="+str(best\_parameters))

print("Best Model="+str(RDT\_best\_model))

print("Mean Absolute Precentage Error for train data="+str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="+str(MAPE\_test))

print("R^2\_score for train data="+str(r2\_train))

print("R^2\_score for test data="+str(r2\_test))

print("RMSE for train data="+str (RMSE\_train))

print("RMSE for test data="+str(RMSE\_test))

rror\_MetricsRDT = {'Model Name': ['Random Search CV Decision Tree'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

RandomDecisionTree\_Results = pd.DataFrame(Error\_MetricsRDT)

RandomDecisionTree\_Results

# Grid Search CV in Decision Tree

# Import libraries

from sklearn.model\_selection import GridSearchCV

GridDecisionTree= DecisionTreeRegressor(random\_state=0)

depth= list(range(1,20,2))

grid\_search= {'max\_depth':depth}

# Lets build a model using above parameters on train data

GridDecisionTree\_model= GridSearchCV(GridDecisionTree,param\_grid=grid\_search,cv=5)

GridDecisionTree\_model= GridDecisionTree\_model.fit(X\_train,y\_train)

# Lets look into best fit parameters from gridsearch cv DT

best\_parameters = GridDecisionTree\_model.best\_params\_

print(best\_parameters)

# Again rebuild decision tree model using gridsearch best fit parameter ie

# with maximum depth = 7

GDT\_best\_model = GridDecisionTree\_model.best\_estimator\_

# Prediction on train data

GDT\_train = GDT\_best\_model.predict(X\_train)

# Prediction on train data test data-

GDT\_test = GDT\_best\_model.predict(X\_test)

# Lets check Model performance on both test and train using error metrics of regression like mape,rsquare value

# MAPE for train data

MAPE\_train= MAPE(y\_train,GDT\_train)

# MAPE for test data

MAPE\_test= MAPE(y\_test,GDT\_test)

# Rsquare for train data

r2\_train= r2\_score(y\_train,GDT\_train)

# Rsquare for train data

r2\_test=r2\_score(y\_test,GDT\_test)

# RMSE value for train data

RMSE\_train = np.sqrt(metrics.mean\_squared\_error(y\_train,GDT\_train))

# RMSE value for test data

RMSE\_test = np.sqrt(metrics.mean\_squared\_error(y\_test,GDT\_test))

print("Best Parameter="+str(best\_parameters))

print("Best Model="+str(GDT\_best\_model))

print("Mean Absolute Precentage Error for train data="+str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="+str(MAPE\_test))

print("R^2\_score for train data="+str(r2\_train))

print("R^2\_score for test data="+str(r2\_test))

print("RMSE for train data="+str (RMSE\_train))

print("RMSE for test data="+str(RMSE\_test))

Error\_MetricsGDT = {'Model Name': ['Grid Search CV Decision Tree'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

GridDecisionTree\_Results = pd.DataFrame(Error\_MetricsGDT)

GridDecisionTree\_Results

# Random Forest

# Import libraris

from sklearn.ensemble import RandomForestRegressor

# Random Forest for regression

RF\_model= RandomForestRegressor(n\_estimators=100).fit(X\_train,y\_train)

# Prediction on train data

RF\_train= RF\_model.predict(X\_train)

# Prediction on test data

RF\_test= RF\_model.predict(X\_test)

# MAPE For train data

MAPE\_train= MAPE(y\_train,RF\_train)

# MAPE For test data

MAPE\_test= MAPE(y\_test,RF\_test)

# Rsquare For train data

r2\_train= r2\_score(y\_train,RF\_train)

# Rsquare For test data

r2\_test=r2\_score(y\_test,RF\_test)

# RMSE value for train data

RMSE\_train = np.sqrt(metrics.mean\_squared\_error(y\_train,RF\_train))

# RMSE value for test data

RMSE\_test = np.sqrt(metrics.mean\_squared\_error(y\_test,RF\_test))

print("Mean Absolute Precentage Error for train data="+str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="+str(MAPE\_test))

print("R^2\_score for train data="+str(r2\_train))

print("R^2\_score for test data="+str(r2\_test))

print("RMSE for train data="+str (RMSE\_train))

print("RMSE for test data="+str(RMSE\_test))

Error\_MetricsRF = {'Model Name': ['Random Forest'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

RandomForest\_Results = pd.DataFrame(Error\_MetricsRF)

RandomForest\_Results

# Random Search CV in Random Forest

# Import libraries

from sklearn.model\_selection import RandomizedSearchCV

RandomRandomForest = RandomForestRegressor(random\_state = 0)

n\_estimator = list(range(1,100,2))

depth = list(range(1,20,2))

random\_search = {'n\_estimators':n\_estimator, 'max\_depth': depth}

# Lets build a model using above parameters on train data

RandomRandomForest\_model= RandomizedSearchCV(RandomRandomForest,param\_distributions= random\_search,n\_iter=5,cv=5)

RandomRandomForest\_model= RandomRandomForest\_model.fit(X\_train,y\_train)

# Best parameters for model

best\_parameters = RandomRandomForest\_model.best\_params\_

print(best\_parameters)

# Again rebuild random forest model using gridsearch best fit parameter

RRF\_best\_model = RandomRandomForest\_model.best\_estimator\_

# Prediction on train data

RRF\_train = RRF\_best\_model.predict(X\_train)

# Prediction on test data

RRF\_test = RRF\_best\_model.predict(X\_test)

# Lets check Model performance on both test and train using error metrics of regression like mape,rsquare value

# MAPE for train data

MAPE\_train= MAPE(y\_train,RRF\_train)

# MAPE for test data

MAPE\_test= MAPE(y\_test,RRF\_test)

# Rsquare for train data

r2\_train= r2\_score(y\_train,RRF\_train)

# Rsquare for test data

r2\_test=r2\_score(y\_test,RRF\_test)

# RMSE value for train data

RMSE\_train = np.sqrt(metrics.mean\_squared\_error(y\_train,RRF\_train))

# RMSE value for test data

RMSE\_test = np.sqrt(metrics.mean\_squared\_error(y\_test,RRF\_test))

print("Best Parameter="+str(best\_parameters))

print("Best Model="+str(RRF\_best\_model))

print("Mean Absolute Precentage Error for train data="+str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="+str(MAPE\_test))

print("R^2\_score for train data="+str(r2\_train))

print("R^2\_score for test data="+str(r2\_test))

print("RMSE for train data="+str (RMSE\_train))

print("RMSE for test data="+str(RMSE\_test))

Error\_MetricsRSRF = {'Model Name': ['Random Search CV Random Forest'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

RandomSearchRandomForest\_Results = pd.DataFrame(Error\_MetricsRSRF)

RandomSearchRandomForest\_Results

# Grid search CV in Random Forest

# Import libraries

from sklearn.model\_selection import GridSearchCV

GridRandomForest= RandomForestRegressor(random\_state=0)

n\_estimator = list(range(1,20,2))

depth= list(range(1,20,2))

grid\_search= {'n\_estimators':n\_estimator, 'max\_depth': depth}

# Lets build a model using above parameters on train data using random forest grid search cv

GridRandomForest\_model= GridSearchCV(GridRandomForest,param\_grid=grid\_search,cv=5)

GridRandomForest\_model= GridRandomForest\_model.fit(X\_train,y\_train)

# Best fit parameters for model

best\_parameters\_GRF = GridRandomForest\_model.best\_params\_

print(best\_parameters\_GRF)

# Again rebuild random forest model using gridsearch best fit parameter

GRF\_best\_model = GridRandomForest\_model.best\_estimator\_

# Prediction on train data

GRF\_train = GRF\_best\_model.predict(X\_train)

# Prediction on test data

GRF\_test = GRF\_best\_model.predict(X\_test)

# Lets check Model performance on both test and train using error metrics of regression like mape,rsquare value

# MAPE for train data

MAPE\_train= MAPE(y\_train,GRF\_train)

# MAPE for test data

MAPE\_test= MAPE(y\_test,GRF\_test)

# Rsquare for train data

r2\_train= r2\_score(y\_train,GRF\_train)

# Rsquare for test data

r2\_test=r2\_score(y\_test,GRF\_test)

# RMSE value for train data

RMSE\_train = np.sqrt(metrics.mean\_squared\_error(y\_train,GRF\_train))

# RMSE value for test data

RMSE\_test = np.sqrt(metrics.mean\_squared\_error(y\_test,GRF\_test))

print("Best Parameter="+str(best\_parameters))

print("Best Model="+str(GRF\_best\_model))

print("Mean Absolute Precentage Error for train data="+str(MAPE\_train))

print("Mean Absolute Precentage Error for test data="+str(MAPE\_test))

print("R^2\_score for train data="+str(r2\_train))

print("R^2\_score for test data="+str(r2\_test))

print("RMSE for train data="+str (RMSE\_train))

print("RMSE for test data="+str(RMSE\_test))

Error\_MetricsGSRF = {'Model Name': ['Grid search CV Random Forest'],

'MAPE\_Train':[MAPE\_train],

'MAPE\_Test':[MAPE\_test],

'R-squared\_Train':[r2\_train],

'R-squared\_Test':[r2\_test],

'RMSE\_train':[RMSE\_train],

'RMSE\_test':[RMSE\_test]}

GridSearchRandomForest\_Results = pd.DataFrame(Error\_MetricsGSRF)

GridSearchRandomForest\_Results

Final\_Results = pd.concat([LinearRegression\_Results,

DecisionTree\_Results,

RandomDecisionTree\_Results,

GridDecisionTree\_Results,

RandomForest\_Results,

RandomSearchRandomForest\_Results,

GridSearchRandomForest\_Results,], ignore\_index=True, sort =False)

# From above results Random Forest model have optimum values and this

# algorithm is good for our data

# Lets save the out put of finalized model (RF)

input = y\_test.reset\_index()

pred = pd.DataFrame(RF\_test,columns = ['pred'])

Final\_output = pred.join(input)

Final\_output

Final\_output.to\_csv("Final\_results\_py.csv")

**APPENDIX B - R CODE**

**PREDICTING BIKE RENTAL COUNT**

**##################################################**

**######## PREDICTING BIKE RENTAL COUNT ############**

**##################################################**

**#lets clean the R environment**

**rm(list = ls())**

**#setting working directory**

**setwd("D:/Data Science/Assignments/Project")**

**getwd()**

**# Load libraries**

**x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced",**

**"Information", "MASS", "rpart", "ROSE",**

**'sampling', 'DataCombine', 'inTrees',"scales","psych","gplots")**

**#install.packages(x)**

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

**rm(x)**

**#lets load the data**

**Bike\_Data = read.csv("day.csv")**

**#explore the data**

**dim(Bike\_Data)**

**names(Bike\_Data)**

**#rename the shortcut values for our understanding**

**names(Bike\_Data)[1] = 'index'**

**names(Bike\_Data)[2] = 'date'**

**names(Bike\_Data)[4] = 'year'**

**names(Bike\_Data)[5] = 'month'**

**names(Bike\_Data)[9] = 'weather'**

**names(Bike\_Data)[10] = 'temperature'**

**names(Bike\_Data)[12] = 'humidity'**

**names(Bike\_Data)[16] = 'count'**

**#lets check column names after renamed**

**names(Bike\_Data)**

**#lets see top 5 observations in the dataset**

**head(Bike\_Data)**

**#lets check last 5 observations in our dataset**

**tail(Bike\_Data)**

**#lets check structure of each variable**

**str(Bike\_Data)**

**#lets see summary of the dataset**

**summary(Bike\_Data)**

**#in our dataset we have 16 variables out of which all are independent variable except last variable**

**str(Bike\_Data['count'])**

**#in our dataset some vaiables has no usefull information for our prediction**

**#so it is better to remove those variables.so it helps us to make useful inferences**

**#lets drop unnecessary variables**

**Bike\_Data = subset(Bike\_Data,select = -c(index,date,casual,registered))**

**#lets divide categorical variables and numerical variables**

**#numerical variables**

**cnames = c("temperature",'atemp','windspeed','humidity','count')**

**#categorical variables**

**catnames = c('season','year','month','holiday','weekday','workingday','weather')**

**#Data preprocessing**

**missing\_val = sum(is.na(Bike\_Data))**

**missing\_val**

**#there is no missing values in our dataset**

**#outlier analysis**

**for (i in 1:length(cnames))**

**{**

**assign(paste0("gn",i), ggplot(aes\_string(y = (cnames[i]), x = "count"), data = subset(Bike\_Data))+**

**stat\_boxplot(geom = "errorbar", width = 0.5) +**

**geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,**

**outlier.size=1, notch=FALSE) +**

**theme(legend.position="bottom")+**

**labs(y=cnames[i],x="count")+**

**ggtitle(paste("Box plot of count for",cnames[i])))**

**}**

**#plotting boxplot**

**library(gridExtra)**

**gridExtra::grid.arrange(gn1,gn2,ncol=2)**

**gridExtra::grid.arrange(gn3,gn4,ncol=2)**

**gridExtra::grid.arrange(gn5,ncol=1)**

**#lets remove outliers using boxplot**

**df = Bike\_Data**

**for(i in cnames){**

**print(i)**

**outliers = Bike\_Data[,i][Bike\_Data[,i] %in% boxplot.stats(Bike\_Data[,i])$out]**

**print(length(outliers))**

**Bike\_Data = Bike\_Data[which(!Bike\_Data[,i] %in% outliers),]**

**}**

**#lets plot boxplot after removing outliers**

**for (i in 1:length(cnames))**

**{**

**assign(paste0("gn",i), ggplot(aes\_string(y = (cnames[i]), x = "count"), data = subset(Bike\_Data))+**

**stat\_boxplot(geom = "errorbar", width = 0.5) +**

**geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,**

**outlier.size=1, notch=FALSE) +**

**theme(legend.position="bottom")+**

**labs(y=cnames[i],x="count")+**

**ggtitle(paste("Box plot of count for",cnames[i])))**

**}**

**#plotting Boxplot after removing outliers**

**gridExtra::grid.arrange(gn1,gn2,ncol=2)**

**gridExtra::grid.arrange(gn3,gn4,ncol=2)**

**gridExtra::grid.arrange(gn5,ncol=1)**

**#data visualization**

**#univariate analysis**

**#lets see distribution of the variables.**

**for(i in 1:length(cnames))**

**{**

**assign(paste0("h",i),ggplot(aes\_string(x=(cnames[i])),**

**data=subset(Bike\_Data))+**

**geom\_histogram(fill="green",colour = "red")+geom\_density()+**

**scale\_y\_continuous(breaks =pretty\_breaks(n=10))+**

**scale\_x\_continuous(breaks = pretty\_breaks(n=10))+**

**theme\_bw()+xlab(cnames[i])+ylab("Frequency")+**

**ggtitle(paste("distribution plot for ",cnames[i])))**

**}**

**#lets see distribution plot**

**gridExtra::grid.arrange(h1,h2,ncol=1)**

**gridExtra::grid.arrange(h3,h4,ncol=1)**

**gridExtra::grid.arrange(h5,ncol=1)**

**#bivariate analysis**

**#lets check distribution between target and continuous variables**

**for(i in 1:length(cnames))**

**{**

**assign(paste0("s",i),ggplot(aes\_string(y='count',x = (cnames[i])),**

**data=subset(Bike\_Data))+**

**geom\_point(alpha=0.5,color="green") +**

**ggtitle(paste("Scatter Plot between count vs ",cnames[i])))**

**}**

**#lets plot between continuous and target variables.**

**gridExtra::grid.arrange(s1,s2,ncol=1)**

**gridExtra::grid.arrange(s3,s4,ncol=1)**

**gridExtra::grid.arrange(s5,ncol=1)**

**#from the above graphs,we can see that as temperature increases and rental count also increases**

**#apart from temperature,windspeed and humidity doesnot impact on rental count**

**#lets check categorical variables**

**for(i in 1:length(catnames))**

**{**

**assign(paste0("b",i),ggplot(aes\_string(y='count',x = (catnames[i])),**

**data=subset(Bike\_Data))+**

**geom\_bar(stat = "identity",fill = "green") +**

**ggtitle(paste("Number of bikes rented with respect to",catnames[i])))+**

**theme(axis.text.x = element\_text( color="red", size=8))+**

**theme(plot.title = element\_text(face = "old"))**

**}**

**#lets plot between categorical and target variables**

**gridExtra::grid.arrange(b1,b2,ncol=1)**

**gridExtra::grid.arrange(b3,b4,ncol=1)**

**gridExtra::grid.arrange(b5,b6,ncol=1)**

**gridExtra::grid.arrange(b7,ncol=1)**

**# Based on the plot,we can the observe the below inferences**

**aggregate(count ~ season ,sum,data = Bike\_Data)**

**#Bike rental count is high in season 3 which is fall and low in season 1**

**aggregate(count ~ year ,sum,data = Bike\_Data)**

**#Bike rental count is high in year 1 which is 2012**

**aggregate(count ~ month,sum,data = Bike\_Data)**

**#Bike rental count is high in the month of august and low in january**

**aggregate(count ~ holiday ,sum,data = Bike\_Data)**

**#Bike rental count is high on holidays which is 0 and low in working day**

**aggregate(count ~ weekday ,sum,data = Bike\_Data)**

**#bike rental count is high in 5 which is friday and low in 0 which is sunday**

**aggregate(count ~ workingday,sum,data = Bike\_Data)**

**#Bike rental count is high in 1 which is working day**

**#Bike rental count is high in weather 1 which is Clear, Few clouds, Partly cloudy, Partly cloudy**

**#and there is no bike rented on 4**

**aggregate(count ~ weather,sum,data = Bike\_Data)**

**# Bikes rented with respect to temperature and humidity**

**ggplot(Bike\_Data,aes(temperature,count)) +**

**geom\_point(aes(color=humidity),alpha=0.8) +**

**labs(title = "Variations of bikes rented with respect to temperature and humidity",**

**x = "temperature")+ theme\_bw()**

**#based on the below plot we know that bike rental is higher when the**

**#temperature is between 0.5 to 0.75**

**#humidity less than 0.6**

**#Bikes rented with respect to temperature and windspeed**

**ggplot(Bike\_Data, aes(x = temperature, y = count))+**

**geom\_point(aes(color=windspeed))+**

**labs(title = "Variations of bikes rented with respect to temperature and windspeed",**

**x = "temperature")+**

**theme(plot.title = element\_text(hjust = 0.5, face = "bold"))+**

**theme\_bw()**

**#based on the below plot we know that bike rental is higher when the**

**#temperature is between 0.5 to 0.75**

**#windspeed is less than 0.2**

**# Bikes rented with respect to temperature and season**

**ggplot(Bike\_Data, aes(x = temperature, y = count))+**

**geom\_point(aes(color=season))+**

**labs(title = "Variations of bikes rented with respect to temperature and season",**

**x = "temperature")+**

**theme(plot.title = element\_text(hjust = 0.5, face = "bold"))+**

**theme\_bw()**

**#based on the below plot we know that bike rental is higher when the**

**#temperature is between 0.5 to 0.75**

**#season was 2 and 3**

**#FEATURE SELECTION**

**#lets find correlation matrix using corrplot and correlation plot using corrgram library**

**#FOR NUMERICAL VARIABLES**

**#lets save dataset after outlier analysis**

**df = Bike\_Data**

**#correlation matrix**

**correlation\_matrix = cor(Bike\_Data[,cnames])**

**correlation\_matrix**

**#correlation plot**

**corrgram(Bike\_Data[,cnames],order = F,upper.panel = panel.pie,**

**text.panel = panel.txt,main = 'Correlation plot')**

**#From the correlation plot,we see that temperature and atemp variables are correlated to each other**

**#so we need to remove atemp variable.**

**#lets see annova test for categorical variables**

**for (i in catnames) {**

**print(i)**

**anova = summary(aov(formula = count~Bike\_Data[,i],Bike\_Data))**

**print(anova)**

**}**

**#based on the anova result, we are going to drop three variables namely,**

**#HOLIDAY**

**#WEEKDAY**

**#WORKINGDAY**

**#because these variables having the p-value > 0.05**

**#Dimension reduction**

**Bike\_Data = subset(Bike\_Data,select = -c(holiday,weekday,workingday,atemp))**

**#lets check after dimension reduction**

**dim(Bike\_Data)**

**head(Bike\_Data)**

**#lets update continous and categorical variables after dimension reduction**

**cnames = c('temperature','humidity','windspeed','count')**

**catnames = c('season','year','month','weather')**

**#FEATURE SCALING**

**#lets check normality between the varaibles**

**for (i in cnames){**

**print(i)**

**normality = qqnorm(Bike\_Data[,i])**

**}**

**#already we plotted distrution between these variables,lets recall it**

**for(i in 1:length(cnames))**

**{**

**assign(paste0("h",i),ggplot(aes\_string(x=(cnames[i])),**

**data=subset(Bike\_Data))+**

**geom\_histogram(fill="green",colour = "red")+geom\_density()+**

**scale\_y\_continuous(breaks =pretty\_breaks(n=10))+**

**scale\_x\_continuous(breaks = pretty\_breaks(n=10))+**

**theme\_bw()+xlab(cnames[i])+ylab("Frequency")+**

**ggtitle(paste("distribution plot for ",cnames[i])))**

**}**

**gridExtra::grid.arrange(h1,h2,h3,h4,ncol = 2)**

**#summary of the data**

**for (i in cnames) {**

**print(i)**

**print(summary(Bike\_Data[,i]))**

**}**

**#Based on the above inferences and plots,we can see that the variables are normalised**

**#as mentioned in problem statement**

**#MODEL DEVELOPMENT**

**#lets clean our environment except preprocessed dataset**

**rmExcept('Bike\_Data')**

**#we can pass categorical variables to regression problems**

**#lets convert categorical variables into dummy variables**

**#save our preprocessed data**

**df = Bike\_Data**

**#create dummies**

**library(dummies)**

**catnames = c('season','year','month','weather')**

**Bike\_Data = dummy.data.frame(Bike\_Data,catnames)**

**#we succesfully created dummies,lets check dimension and top 5 observations**

**dim(Bike\_Data)**

**head(Bike\_Data)**

**#divide the data into train and test**

**set.seed(1234)**

**train\_index = sample(1:nrow(df), 0.8 \* nrow(df))**

**train\_data = Bike\_Data[train\_index,]**

**test\_data = Bike\_Data[-train\_index,]**

**#linear regression**

**#check multicollearity**

**library(usdm)**

**cnames = c('temperature','humidity','windspeed')**

**vif(Bike\_Data[,cnames])**

**vifcor(Bike\_Data[,cnames], th = 0.9)**

**#No variable from the 3 input variables has collinearity problem.**

**#The linear correlation coefficients ranges between:**

**# min correlation ( humidity ~ temperature ): 0.1267216**

**#max correlation ( windspeed ~ humidity ): -0.2411599**

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

**# Variables VIF**

**#1 temperature 1.034137**

**#2 humidity 1.070959**

**#3 windspeed 1.080362**

**#lets run regression model**

**lm\_model = lm(count~. ,data = Bike\_Data)**

**#lets check performance of our modedl**

**summary(lm\_model)**

**#Residual standard error: 787.1 on 710 degrees of freedom**

**#Multiple R-squared: 0.8394, Adjusted R-squared: 0.8349**

**#F-statistic: 185.6 on 20 and 710 DF, p-value: < 2.2e-16**

**# Function for Error metrics to calculate the performance of model**

**#lets build function for MAPE**

**#calculate MAPE**

**MAPE = function(y, y1){**

**mean(abs((y - y1)/y))**

**}**

**# Function for r2 to calculate the goodness of fit of model**

**rsquare=function(y,y1){**

**cor(y,y1)^2**

**}**

**# Function for RMSE value**

**RMSE = function(y,y1){**

**difference = y - y1**

**root\_mean\_square = sqrt(mean(difference^2))**

**}**

**#lets predict for train and test data**

**Predictions\_LR\_train = predict(lm\_model,train\_data[,-25])**

**Predictions\_LR\_test = predict(lm\_model,test\_data[,-25])**

**#let us check performance of our model**

**#mape calculation**

**LR\_train\_mape = MAPE(Predictions\_LR\_train,train\_data[,25])**

**LR\_test\_mape = MAPE(test\_data[,25],Predictions\_LR\_test)**

**#Rsquare calculation**

**LR\_train\_r2 = rsquare(train\_data[,25],Predictions\_LR\_train)**

**LR\_test\_r2 = rsquare(test\_data[,25],Predictions\_LR\_test)**

**#rmse calculation**

**LR\_train\_rmse = RMSE(train\_data[,25],Predictions\_LR\_train)**

**LR\_test\_rmse = RMSE(test\_data[,25],Predictions\_LR\_test)**

**print(LR\_train\_mape)#0.16**

**print(LR\_test\_mape)#0.17**

**print(LR\_train\_r2)#0.825**

**print(LR\_test\_r2)#0.893**

**print(LR\_train\_rmse)#804.9**

**print(LR\_test\_rmse)#648.9**

**#Decision tree regression**

**library(rpart)**

**DT\_model = rpart(count ~ ., data = train\_data, method = "anova")**

**DT\_model**

**# Lets predict for train and test data**

**predictions\_DT\_train= predict(DT\_model,train\_data[,-25])**

**predictions\_DT\_test= predict(DT\_model,test\_data[,-25])**

**# MAPE calculation**

**DT\_train\_mape = MAPE(train\_data[,25],predictions\_DT\_train)**

**DT\_test\_mape = MAPE(test\_data[,25],predictions\_DT\_test)**

**# Rsquare calculation**

**DT\_train\_r2= rsquare(train\_data[,25],predictions\_DT\_train)**

**DT\_test\_r2 = rsquare(test\_data[,25],predictions\_DT\_test)**

**# RMSE calculation**

**DT\_train\_rmse = RMSE(train\_data[,25],predictions\_DT\_train)**

**DT\_test\_rmse = RMSE(test\_data[,25],predictions\_DT\_test)**

**print(DT\_train\_mape)#0.536**

**print(DT\_test\_mape)#0.269**

**print(DT\_train\_r2)#0.806**

**print(DT\_test\_r2)#0.834**

**print(DT\_train\_rmse)#846.85**

**print(DT\_test\_rmse)#805.67**

**#Random search CV in decision tree**

**df = Bike\_Data**

**#setting parameters for training using caret library**

**control = trainControl(method="repeatedcv", number=5, repeats=1,search='random')**

**maxdepth = c(1:30)**

**tunegrid = expand.grid(.maxdepth=maxdepth)**

**# Lets build a model using above parameters on train data**

**RDT\_model = caret::train(count~., data=train\_data, method="rpart2",trControl=control,tuneGrid= tunegrid)**

**print(RDT\_model)**

**#lets look best parameter**

**best\_parameter = RDT\_model$bestTune**

**print(best\_parameter)**

**#maximum depth = 10**

**#again build a decsion tree using best parameters**

**RDT\_bestmodel = rpart(count~.,train\_data,method = 'anova',maxdepth=10)**

**print(RDT\_bestmodel)**

**#lets predict for train and test data**

**predictions\_RDT\_train = predict(RDT\_bestmodel,train\_data[1:24])**

**predictions\_RDT\_test = predict(RDT\_bestmodel,test\_data[1:24])**

**#model performance**

**# MAPE calculation**

**RDT\_train\_mape = MAPE(train\_data[,25],predictions\_RDT\_train)**

**RDT\_test\_mape = MAPE(test\_data[,25],predictions\_RDT\_test)**

**# Rsquare calculation**

**RDT\_train\_r2= rsquare(train\_data[,25],predictions\_RDT\_train)**

**RDT\_test\_r2 = rsquare(test\_data[,25],predictions\_RDT\_test)**

**# RMSE calculation**

**RDT\_train\_rmse = RMSE(train\_data[,25],predictions\_RDT\_train)**

**RDT\_test\_rmse = RMSE(test\_data[,25],predictions\_RDT\_test)**

**print(RDT\_train\_mape)#0.522**

**print(RDT\_test\_mape)#0.243**

**print(RDT\_train\_r2)#0.811**

**print(RDT\_test\_r2)#0.798**

**print(RDT\_train\_rmse)#833.48**

**print(RDT\_test\_rmse)#885.59**

**#Grid search CV decision tree**

**#setting parameters for training using caret library**

**control = trainControl(method="repeatedcv", number=5, repeats=2,search='grid')**

**maxdepth = c(6:30)**

**tunegrid = expand.grid(maxdepth=maxdepth)**

**# Lets build a model using above parameters on train data**

**GDT\_model = caret::train(count~., data=train\_data, method="rpart2",trControl=control,tuneGrid= tunegrid)**

**print(GDT\_model)**

**#lets look best parameter**

**best\_parameter = GDT\_model$bestTune**

**print(best\_parameter)**

**#maximum depth = 10**

**#again build a decsion tree using best parameters**

**GDT\_bestmodel = rpart(count~.,train\_data,method = 'anova',maxdepth=10)**

**print(GDT\_bestmodel)**

**#lets predict for train and test data**

**predictions\_GDT\_train = predict(GDT\_bestmodel,train\_data[1:24])**

**predictions\_GDT\_test = predict(GDT\_bestmodel,test\_data[1:24])**

**#model performance**

**# MAPE calculation**

**GDT\_train\_mape = MAPE(train\_data[,25],predictions\_GDT\_train)**

**GDT\_test\_mape = MAPE(test\_data[,25],predictions\_GDT\_test)**

**# Rsquare calculation**

**GDT\_train\_r2= rsquare(train\_data[,25],predictions\_GDT\_train)**

**GDT\_test\_r2 = rsquare(test\_data[,25],predictions\_GDT\_test)**

**# RMSE calculation**

**GDT\_train\_rmse = RMSE(train\_data[,25],predictions\_GDT\_train)**

**GDT\_test\_rmse = RMSE(test\_data[,25],predictions\_GDT\_test)**

**print(GDT\_train\_mape)#0.522**

**print(GDT\_test\_mape)#0.243**

**print(GDT\_train\_r2)#0.811**

**print(GDT\_test\_r2)#0.798**

**print(GDT\_train\_rmse)#833.48**

**print(GDT\_test\_rmse)#885.59**

**#RANDOM FOREST**

**#lets build the random forest model**

**RF\_model = randomForest(count~.,data = train\_data,n.trees = 500)**

**print(RF\_model)**

**#lets predict for both train and test data**

**predictions\_RF\_train = predict(RF\_model,train\_data[-25])**

**predictions\_RF\_test = predict(RF\_model,test\_data[-25])**

**#MAPE calculation**

**RF\_train\_mape = MAPE(predictions\_RF\_train,train\_data[,25])**

**RF\_test\_mape = MAPE(predictions\_RF\_test,test\_data[,25])**

**#Rsquare calculation**

**RF\_train\_r2 = rsquare(predictions\_RF\_train,train\_data[,25])**

**RF\_test\_r2 = rsquare(predictions\_RF\_test,test\_data[,25])**

**#RMSE calculation**

**RF\_train\_rmse = RMSE(train\_data[,25],predictions\_RF\_train)**

**RF\_test\_rmse = RMSE(test\_data[,25],predictions\_RF\_test)**

**print(RF\_train\_mape)#0.07**

**print(RF\_test\_mape)#0.11**

**print(RF\_train\_r2)#0.965**

**print(RF\_test\_r2)#0.912**

**print(RF\_train\_rmse)#371.06**

**print(RF\_test\_rmse)#586.72**

**#Random search CV random forest**

**#setting parameters for training using caret library**

**control = trainControl(method="repeatedcv", number=5, repeats=3,search='random')**

**maxdepth = c(1:30)**

**tunegrid = expand.grid(maxdepth=maxdepth)**

**#lets build Random forest model using the above parameters**

**RRF\_model = caret::train(count~.,data=train\_data,method ='rf',trcontrol=control,tunegrid=tunegrid)**

**print(RRF\_model)**

**best\_parameter = RRF\_model$bestTune**

**print(best\_parameter)**

**#mtry = 13**

**#lets again build the random forest by above paremeters**

**RRF\_bestmodel = randomForest(count~.,data = train\_data,method = 'rf',mtry = 13,importance = TRUE)**

**print(RRF\_bestmodel)**

**#lets predict for both train and test data**

**prediction\_RRF\_train = predict(RRF\_bestmodel,train\_data[-25])**

**prediction\_RRF\_test = predict(RRF\_bestmodel,test\_data[-25])**

**#MAPE calculation**

**RRF\_train\_mape = MAPE(train\_data[,25],prediction\_RRF\_train)**

**RRF\_test\_mape = MAPE(test\_data[,25],prediction\_RRF\_test)**

**#Rsquare calculation**

**RRF\_train\_r2 = rsquare(train\_data[,25],prediction\_RRF\_train)**

**RRF\_test\_r2 = rsquare(test\_data[,25],prediction\_RRF\_test)**

**#RMSE calculation**

**RRF\_train\_rmse = RMSE(train\_data[,25],prediction\_RRF\_train)**

**RRF\_test\_rmse = RMSE(test\_data[,25],prediction\_RRF\_test)**

**print(RRF\_train\_mape)#0.241**

**print(RRF\_test\_mape)#0.159**

**print(RRF\_train\_r2)#0.971**

**print(RRF\_test\_r2)#0.907**

**print(RRF\_train\_rmse)#333.513**

**print(RRF\_test\_rmse)#602.26**

**#GRID SEARCH CV RANDOM FOREST**

**#lets set require parameters using caret library**

**control = trainControl(method="repeatedcv", number=5, repeats=4,search='grid')**

**maxdepth = c(6:30)**

**tunegrid = expand.grid(maxdepth=maxdepth)**

**#lets build Random forest model using the above parameters**

**GRF\_model = caret::train(count~.,data=train\_data,method ='rf',trcontrol=control,tunegrid=tunegrid)**

**print(GRF\_model)**

**best\_parameter = GRF\_model$bestTune**

**print(best\_parameter)**

**#mtry = 13**

**#lets again build the same model using bestparameter**

**GRF\_bestmodel = randomForest(count~.,data = train\_data,mtry =13,importance = TRUE,method='rf')**

**print(GRF\_bestmodel)**

**#lets predict on train and test data,**

**predictions\_GRF\_train = predict(GRF\_bestmodel,train\_data[-25])**

**predictions\_GRF\_test = predict(GRF\_bestmodel,test\_data[-25])**

**#MAPE calculation**

**GRF\_train\_mape = MAPE(predictions\_GRF\_train,train\_data[,25])**

**GRF\_test\_mape = MAPE(predictions\_GRF\_test,test\_data[,25])**

**#Rsquare calculation**

**GRF\_train\_r2 = rsquare(predictions\_GRF\_train,train\_data[,25])**

**GRF\_test\_r2 = rsquare(predictions\_GRF\_test,test\_data[,25])**

**#RMSE calculation**

**GRF\_train\_rmse = RMSE(predictions\_GRF\_train,train\_data[,25])**

**GRF\_test\_rmse = RMSE(predictions\_GRF\_test,test\_data[,25])**

**print(GRF\_train\_mape)#0.06**

**print(GRF\_test\_mape)#0.12**

**print(GRF\_train\_r2)#0.972**

**print(GRF\_test\_r2)#0.90**

**print(GRF\_train\_rmse)#335.18**

**print(GRF\_test\_rmse)#597.59**

**#MODEL SELECTION**

**Model\_name = c('Linear regression',**

**'Decision tree','Random search CV decision tree','Grid search CV decision tree',**

**'Random forest','Random search CV random forest','Grid search CV random forest')**

**MAPE\_train = c(LR\_train\_mape,DT\_train\_mape,RDT\_train\_mape,GDT\_train\_mape,**

**RF\_train\_mape,GRF\_train\_mape,GRF\_train\_mape)**

**MAPE\_test = c(LR\_test\_mape,DT\_test\_mape,RDT\_test\_mape,GDT\_test\_mape,**

**RF\_test\_mape,GRF\_test\_mape,GRF\_test\_mape)**

**Rsquare\_train = c(LR\_train\_r2,DT\_train\_r2,RDT\_train\_r2,GDT\_train\_r2,**

**RF\_train\_r2,GRF\_train\_r2,GRF\_train\_r2)**

**Rsquare\_test = c(LR\_test\_r2,DT\_test\_r2,RDT\_test\_r2,GDT\_test\_r2,**

**RF\_test\_r2,GRF\_test\_r2,GRF\_test\_r2)**

**RMSE\_train = c(LR\_train\_rmse,DT\_train\_rmse,RDT\_train\_rmse,GDT\_train\_rmse,**

**RF\_train\_rmse,GRF\_train\_rmse,GRF\_train\_rmse)**

**RMSE\_test = c(LR\_test\_rmse,DT\_test\_rmse,RDT\_test\_rmse,GDT\_test\_rmse,**

**RF\_test\_rmse,RRF\_test\_rmse,GRF\_test\_rmse)**

**FINAL\_RESULTS = data.frame(Model\_name,MAPE\_train,MAPE\_test,Rsquare\_train,Rsquare\_test,**

**RMSE\_train,RMSE\_test)**

**print(FINAL\_RESULTS)**

**#Index Model\_name MAPE\_train MAPE\_test Rsquare\_train Rsquare\_test RMSE\_train RMSE\_test**

**#1 Linear regression 0.15497164 0.1829289 0.8311816 0.8671739 789.6785 717.2833**

**#2 Decision tree 0.52210598 0.2438791 0.8119266 0.7986807 833.4855 885.5906**

**#3 Random search CV decision tree 0.52210598 0.2438791 0.8119266 0.7986807 833.4855 885.5906**

**#4 Grid search CV decision tree 0.52210598 0.2438791 0.8119266 0.7986807 833.4855 885.5906**

**#5 Random forest 0.07220796 0.1200109 0.9652279 0.9132608 371.1086 585.0001**

**#6 Random search CV random forest 0.06441047 0.1222848 0.9718028 0.9077133 332.6579 593.6970**

**#7 Grid search CV random forest 0.06441047 0.1222848 0.9718028 0.9077133 332.6579 601.4685**

**# Based on the above inferences,we came to know that Random forest performs very well in our dataset**

**#lets finalise that model.**