EDWISOR.COM

Bike Rental Count Prediction

The objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings

Saloni Mishra

2/23/2020

TABLE OF CONTENTS

1. INTRODUCTION

1.1 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.

Let's have a look at the data:

```
In [7]: #Lets see first five observations of our data
        Bike_Data.head()
```

Out[7]:

	index	date	season	year	month	holiday	weekday	workingday	weather	temperature	atemp	humidity	windspeed	casual	registered	count
0	1	01-01-2011	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	02-01-2011	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	03-01-2011	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	04-01-2011	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	05-01-2011	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600

```
In [8]: # lets see last five observations of our data
        Bike_Data.tail()
```

Out[8]:

	index	date	season	year	month	holiday	weekday	workingday	weather	temperature	atemp	humidity	windspeed	casual	registered	count
726	727	27-12-2012	1	1	12	0	4	1	2	0.254167	0.226642	0.652917	0.350133	247	1867	2114
727	728	28-12-2012	1	1	12	0	5	1	2	0.253333	0.255046	0.590000	0.155471	644	2451	3095
728	729	29-12-2012	1	1	12	0	6	0	2	0.253333	0.242400	0.752917	0.124383	159	1182	1341
729	730	30-12-2012	1	1	12	0	0	0	1	0.255833	0.231700	0.483333	0.350754	364	1432	1796
730	731	31-12-2012	1	1	12	0	1	1	2	0.215833	0.223487	0.577500	0.154846	439	2290	2729

Here casual,registered and count are our dependent variables

COUNT = CASUAL+REGISTERED

Remaining all are independent variables.

3

2. METHODOLOGY

2.1 DATA 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:

In this step we first imported the data into R and Python environments,explored the data by looking its dimensions, data structures, variable names, summary of data ,to prediction like what are the independent variables and target variable so our target variable is cnt(count) and predictors areas below in the image .have a glance at our data we checked first and last rows of the train dataset. Renamed variables for more understanding of the data, Identified the variables for Bike Rent count

```
# 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
```

```
Index(['index', 'date', 'season', 'year', 'month', 'holiday', 'weekday',
       'workingday', 'weather', 'temperature', 'atemp', 'humidity',
       'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

For our convineance, 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.

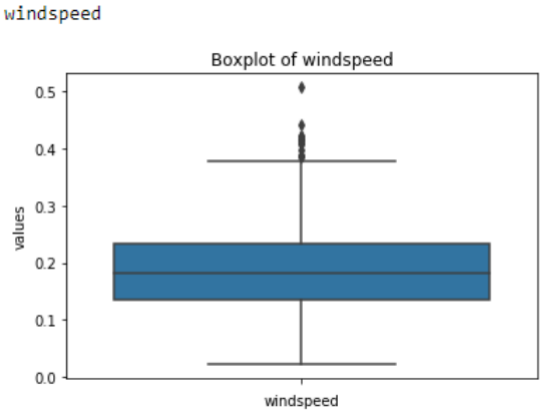
In the given data there is no any missing value. So we do not need to impute missing values.

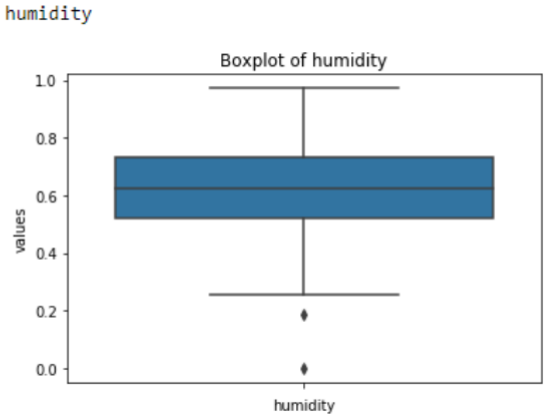
```
season         0
year           0
month          0
holiday        0
weekday        0
workingday     0
weather        0
temperature    0
atemp          0
humidity       0
windspeed      0
count          0
dtype: int64
```

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. We have inliers in humidity and outliers in windspeed 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 the of our model. How much data we feed is that much accuracy to our model.



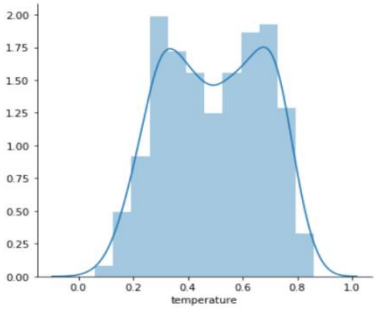


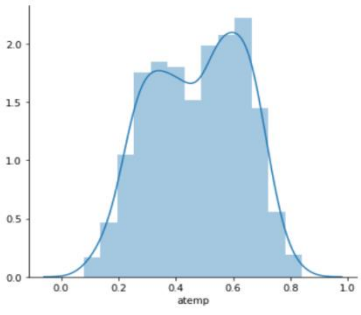
2.1.4 DATA VISUALIZATION:

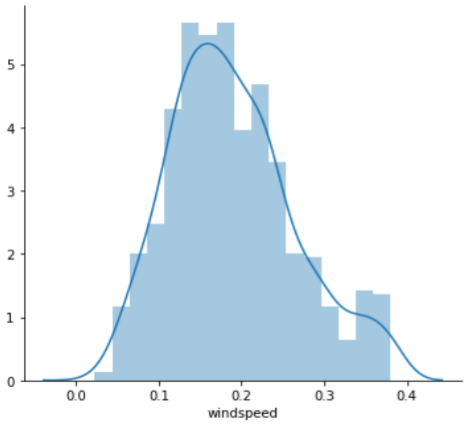
Data visualization is the easy method to understand the data. It will gives clear idea of our data and also impact of dependent variables.

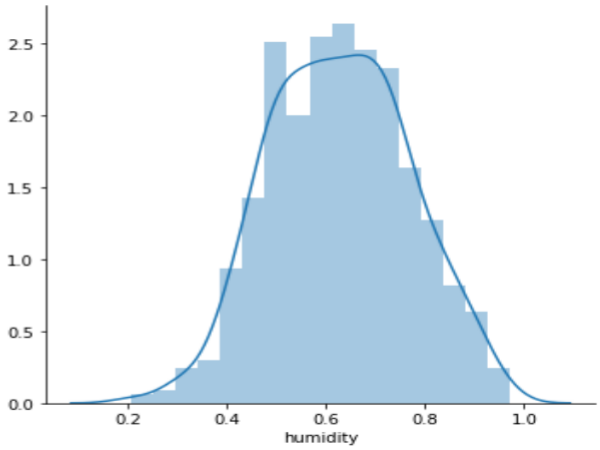
2.1.4a:DISTRIBUTION OF THE NUMERIC VARIABLE:

Distributution plot helps us to know the distribution of the data and makes us easily understandable. So that it used in the bothR and Python languages.here we used histogram for our visualization.







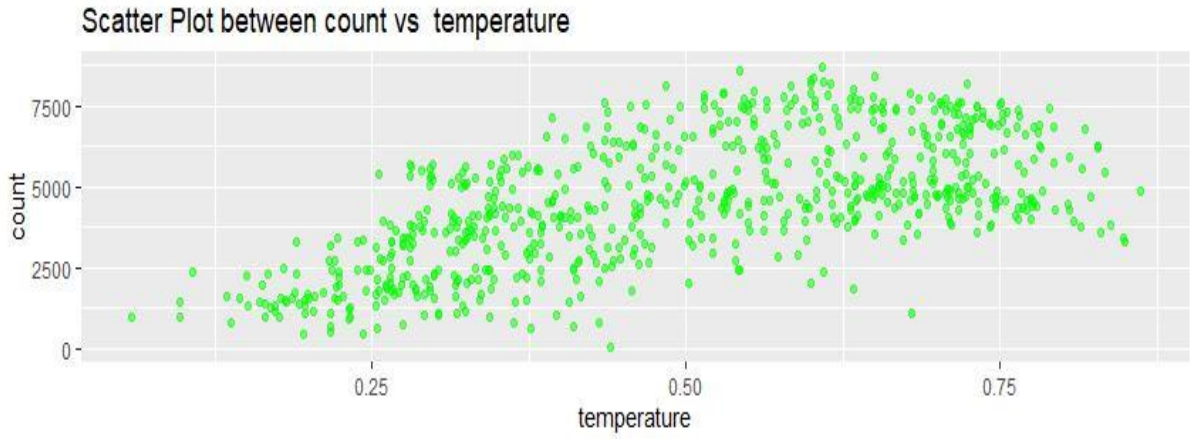


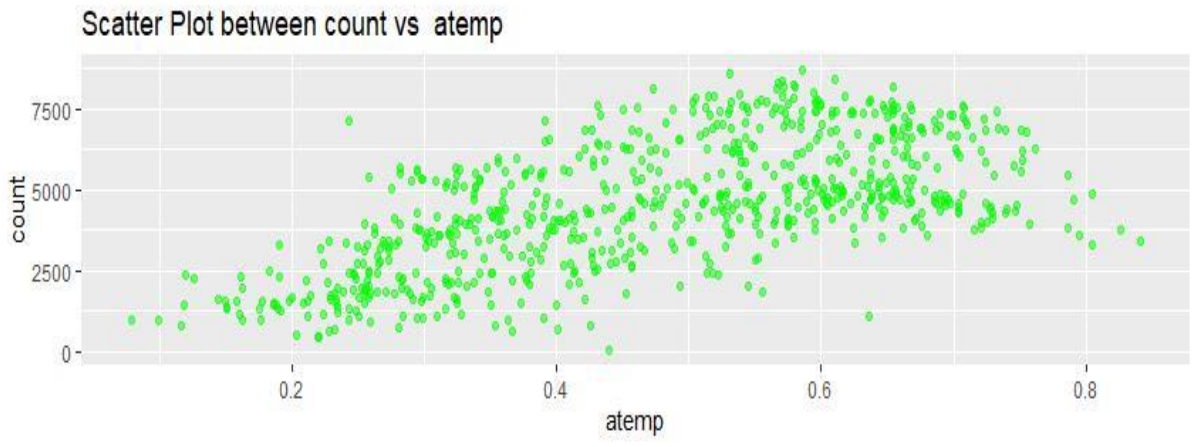
Based on the above graphs, we can clearly says that our temperature and atemp variables are carrying same information. So that we are going to drop any one of the variables for our further analysis.

2.1.4b:DISTRIBUTION OF CONTINUOUS VARIABLES WITH RESPECT TO TARGET VARIABLE:

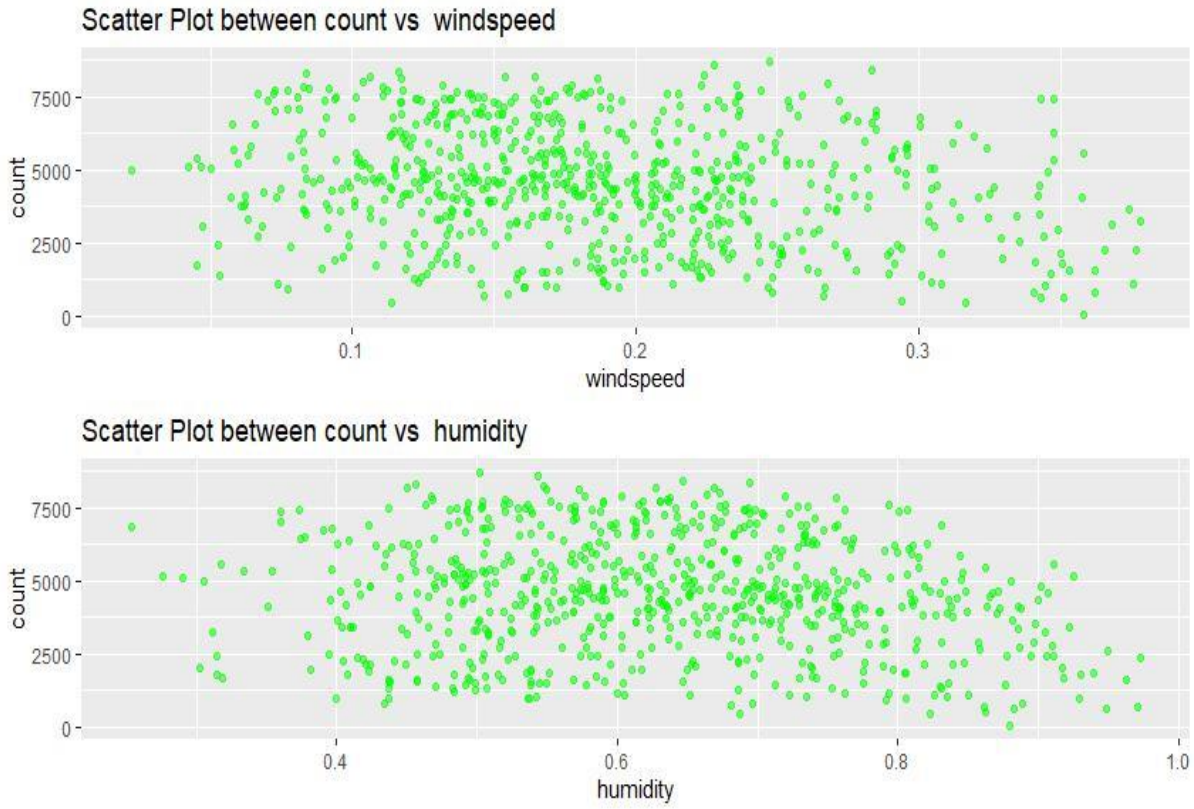
Here we used scatterplot our visualization. First plot is between temperature and atemp variables. Both the temperature and atemp variables are mostly similar to each other.

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





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



Count Vs Temperature: From the below plot we can say as temperature increase Bike rental count also increases.

Count Vs Humi dity: From the below plot we can s ay humidity doesnt have any effect on bike rent cou nt

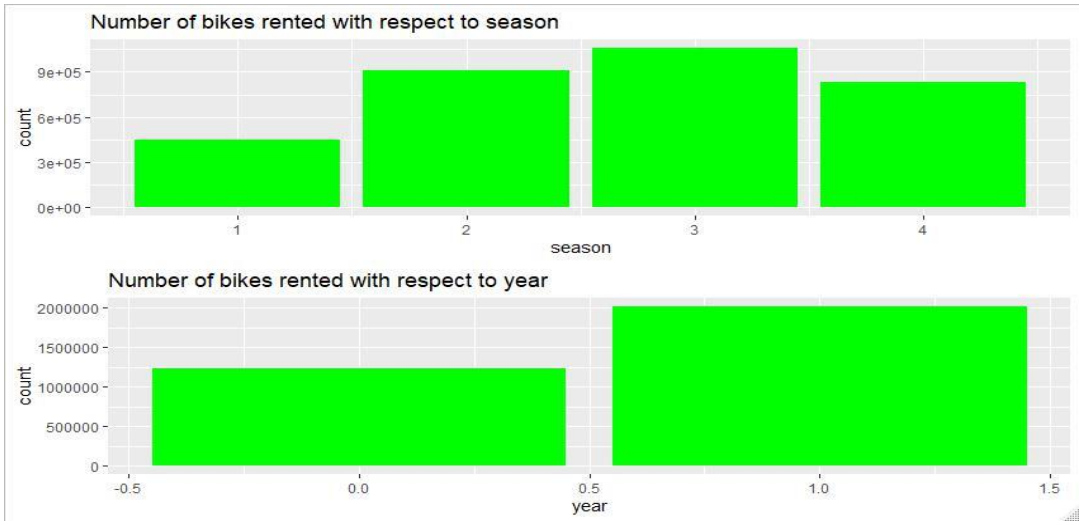
Count Vs Windspeed: From the below plot we can say windspeed doesnt have any effect on bike rent count

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

2.1.4c: IMPACT OF THE CATEGORICAL VARIABLE WITH RESPECT TO TARGET VARIABLE:

COUNT VS SEASON: Bike rental is higher in the season 3 which is 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



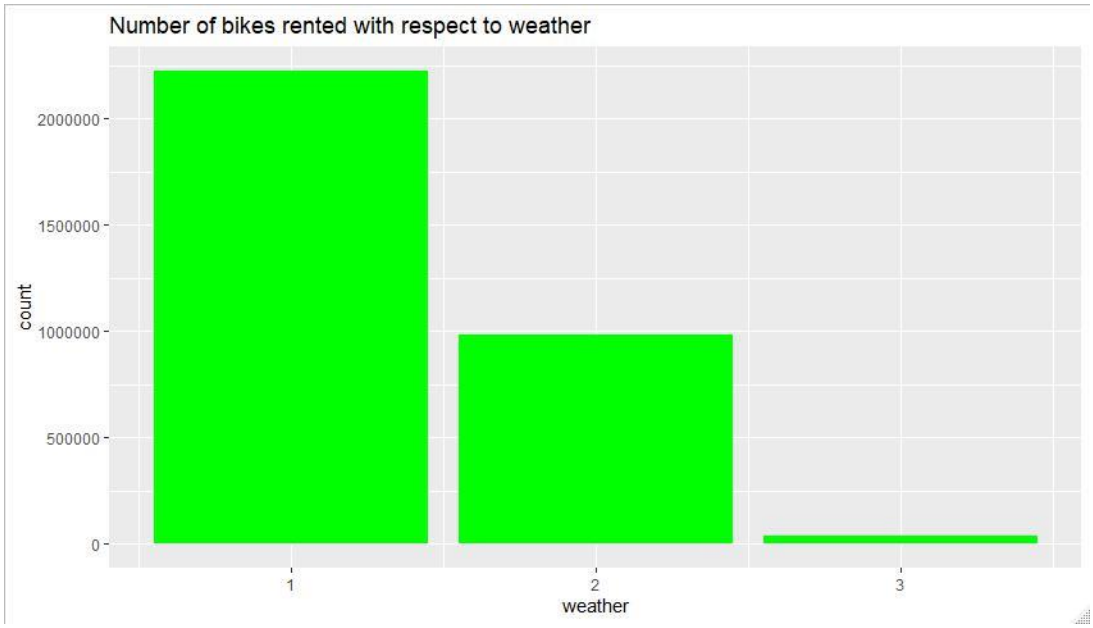
COUNT VS MONTH: Bike rental is higher in the month of 8 which is in august and low in 1 which is in January

COUNT VS HOLIDAY: Bike rental count is higher in 0 which is holiday and low in workingday

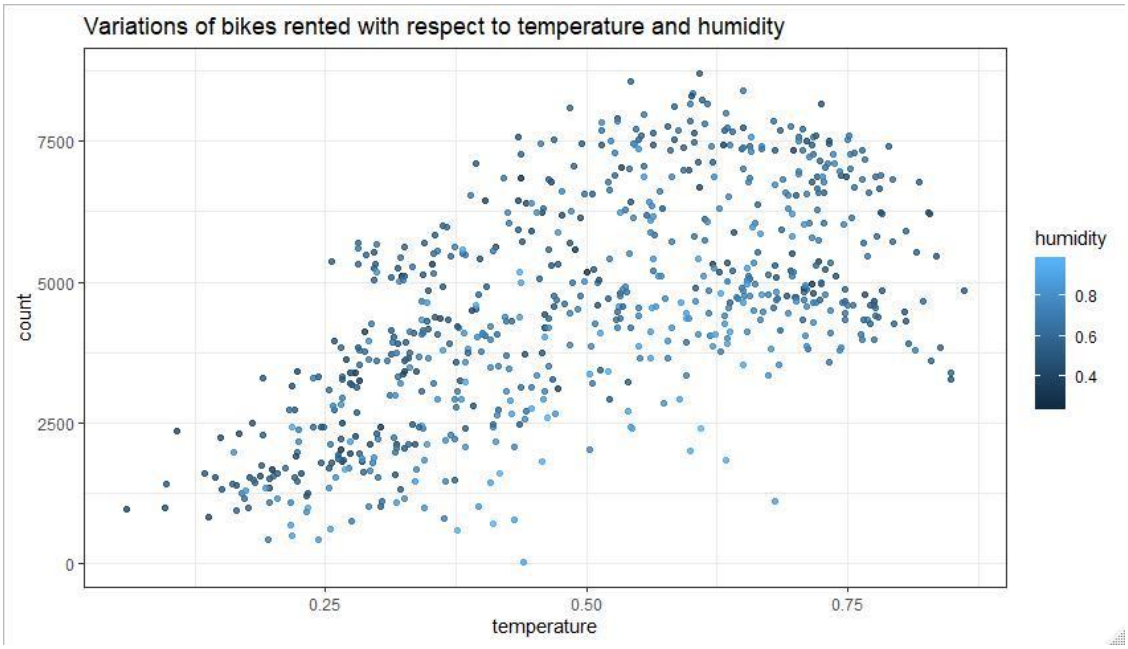
COUNT VS WEEKDAY: Bike rental count is high in 5 which is friday and low in 0 which is sunday

COUNT VS WORKING DAY: Bike rental count is high in 1 which is working day and low in 0 which is holiday

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



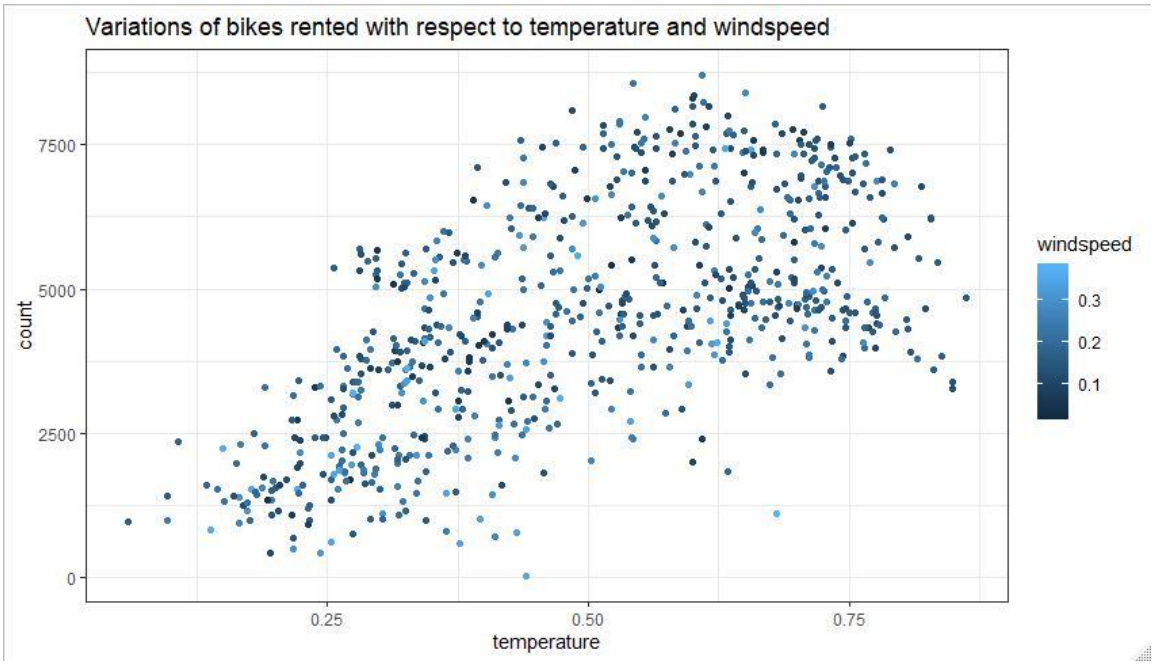
A) BIKE RENTED WITH RESPECTED TO TEMPERATURE AND HUMIDITY:



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

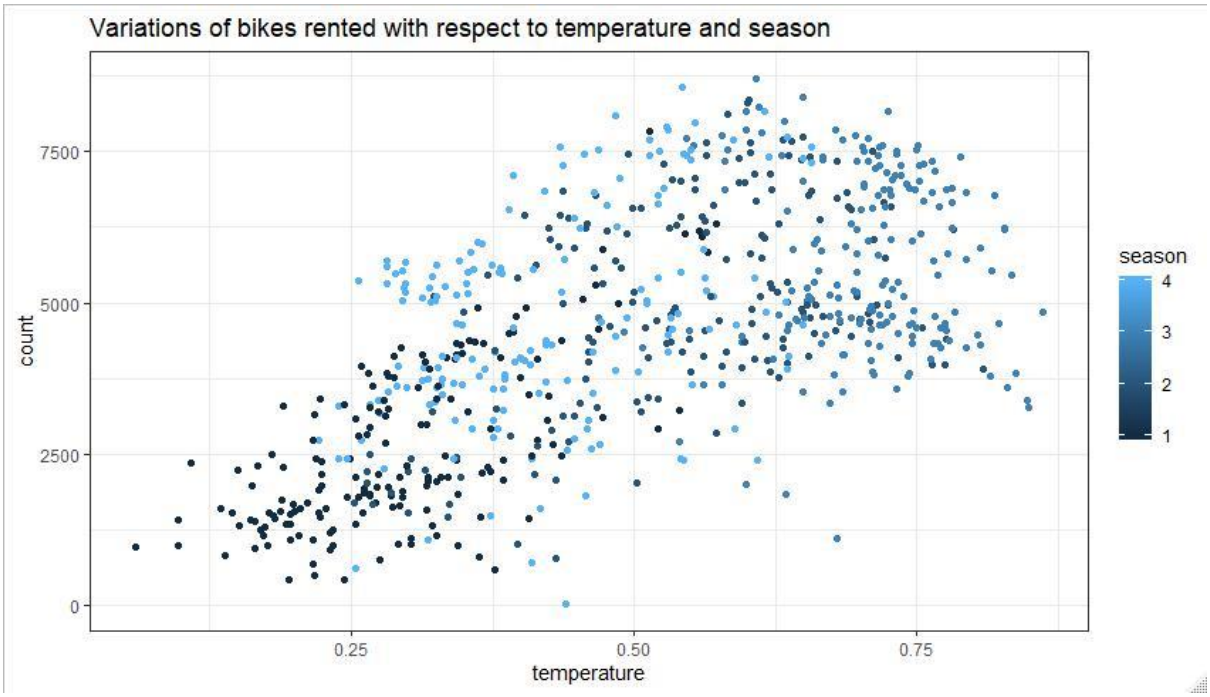
B) BIKE RENTED WITH RESPECT TO TEMPERATURE AND WINDSPEED:



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

C) BIKE RENTED WITH RESPECT TO TEMPERATURE AND SEASON



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

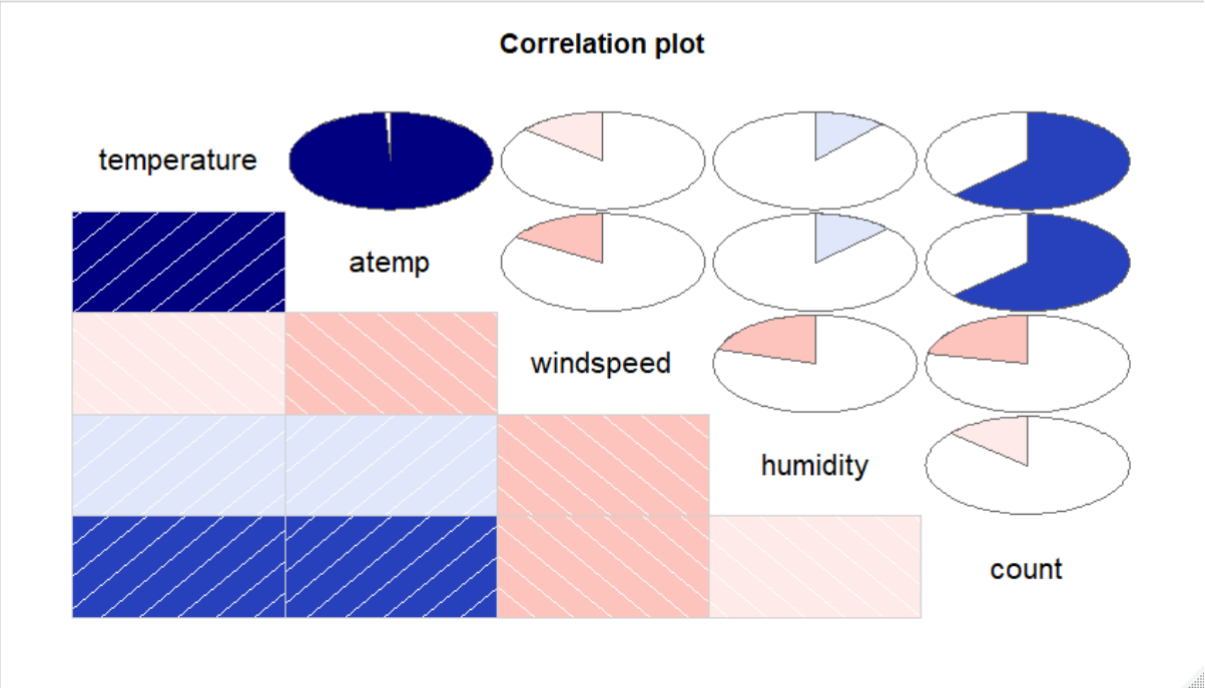
- temperature is between 0.4 to 0.8
- season was 2 and 3
- weather was from 1 and 2

2.1.5 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.

2.1.5a: CORRELATION MATRIX AND PLOT

```
             temperature       atemp   windspeed
temperature    1.0000000   0.9917378 -0.1401690
atemp          0.9917378   1.0000000 -0.1660383
windspeed     -0.1401690  -0.1660383  1.0000000
humidity       0.1141910   0.1265874 -0.2044964
count          0.6258917   0.6292045 -0.2161933
              humidity       count
temperature  0.1141910   0.6258917
atemp        0.1265874   0.6292045
windspeed   -0.2044964  -0.2161933
humidity     1.0000000  -0.1366214
count       -0.1366214   1.0000000
> #correlation plot
```



From the above plot, we say that temperature and atemp variables are carrying same information. So we need to remove atemp variable.

2.1.5b: ANALYSIS OF VARIANCE:

```
                sum_sq     df           F        PR(>F)
season    4.517974e+08    1.0  143.967653  2.133997e-30
Residual  2.287738e+09  729.0         NaN           NaN
                sum_sq     df           F        PR(>F)
year      8.798289e+08    1.0  344.890586  2.483540e-63
Residual  1.859706e+09  729.0         NaN           NaN
                sum_sq     df          F        PR(>F)
month     2.147445e+08    1.0  62.004625  1.243112e-14
Residual  2.524791e+09  729.0        NaN           NaN
                sum_sq     df         F    PR(>F)
holiday   1.279749e+07    1.0  3.421441  0.064759
Residual  2.726738e+09  729.0       NaN       NaN
                sum_sq     df         F    PR(>F)
weekday   1.246109e+07    1.0  3.331091  0.068391
Residual  2.727074e+09  729.0       NaN       NaN
                  sum_sq     df         F    PR(>F)
workingday  1.024604e+07    1.0  2.736742  0.098495
Residual    2.729289e+09  729.0       NaN       NaN
                sum_sq     df          F        PR(>F)
weather   2.422888e+08    1.0  70.729298  2.150976e-16
Residual  2.497247e+09  729.0        NaN           NaN
```

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.

2.1.5c DIMENSION REDUCTION:

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

```
# 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
```

```
(731, 8)
```

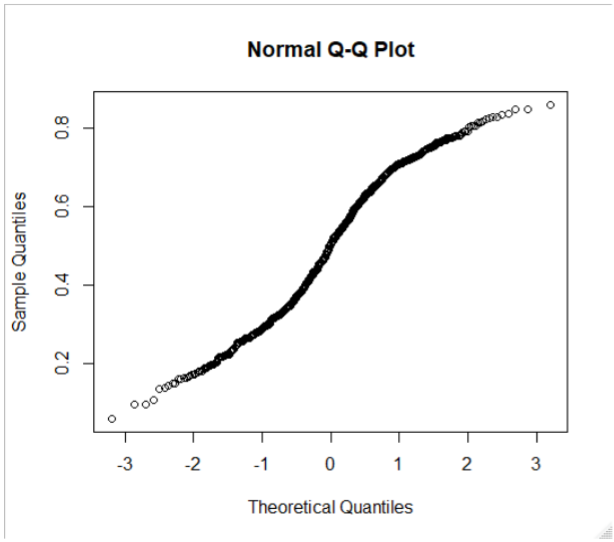
```
# After removing variables lets check column names of the data
Bike_Data.columns
```

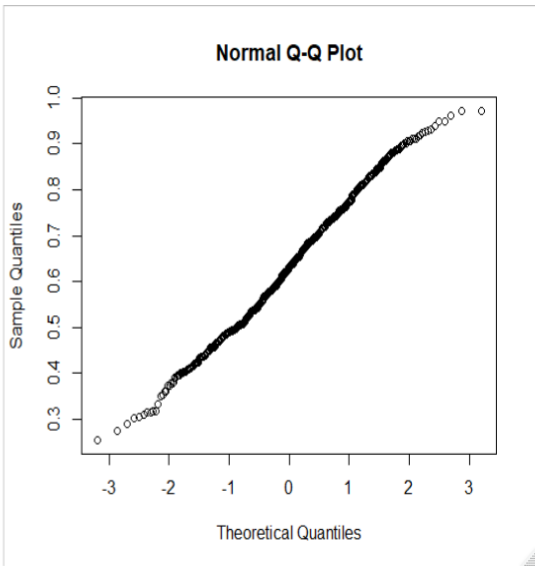
```
Index(['season', 'year', 'month', 'weather', 'temperature', 'humidity',
       'windspeed', 'count'],
      dtype='object')
```

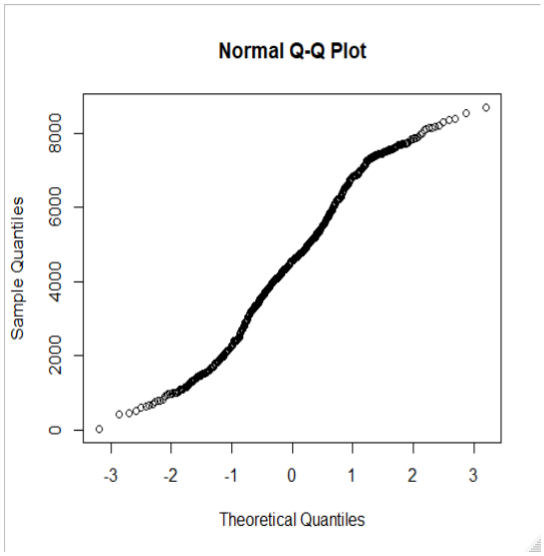
```
#after removing the variables, we need update numerical and categorical variables

# numerical variable
cnames = ['temperature','humidity', 'windspeed', 'count']

# Categorical variables
catnames = ['season', 'year', 'month','weather']
```

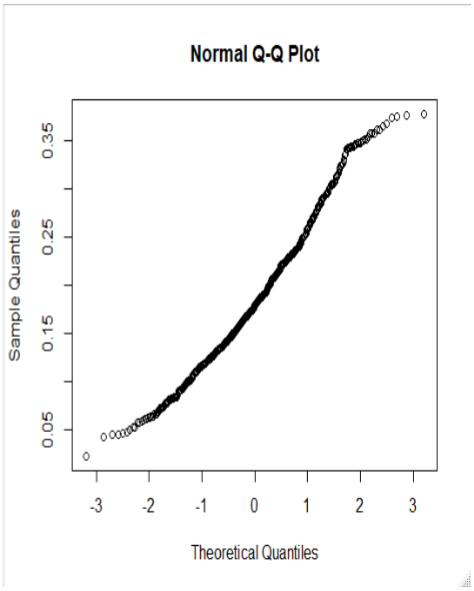
2.1.6 FEATURE SCALING:

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.

```
Bike_Data.describe()
```

	season	year	month	weather	temperature	humidity	windspeed	count
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
mean	2.496580	0.500684	6.519836	1.395349	0.495385	0.627894	0.190486	4504.348837
std	1.110807	0.500342	3.451913	0.544894	0.183051	0.142429	0.077498	1937.211452
min	1.000000	0.000000	1.000000	1.000000	0.059130	0.000000	0.022392	22.000000
25%	2.000000	0.000000	4.000000	1.000000	0.337083	0.520000	0.134950	3152.000000
50%	3.000000	1.000000	7.000000	1.000000	0.498333	0.626667	0.180975	4548.000000
75%	3.000000	1.000000	10.000000	2.000000	0.655417	0.730209	0.233214	5956.000000
max	4.000000	1.000000	12.000000	3.000000	0.861667	0.972500	0.507463	8714.000000

2.2 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.

2.2.1 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.

2.2.2 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. Each branch connects nodes with “and” and multiple branches are connected by “or”. Extremely easy to understand by the business users. It provides its output in the form of rule, which can easily understood by a non-technical person also.

2.2.3. 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.

2.2.4 Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, It produces a prediction model in the form of an ensemble of weak learner models and pro-duce a strong learner with less misclassification and higher accuracy. It feed the error from one decision tree to another decision tree and generates a strong classifier or Regressor.

2.3 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.

2.3.1 TUNING PARAMETERS:

We will explore two different methods for optimizing hyperparameters:

- ✓ Grid Search
- ✓ Random Search

2.3.1A). 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.B) Grid Search

Grid search is a technique which tends to find the right set of hyperparameters for the par-ticular model. Hyperparameters are not the model parameters and it is not possible to find the best set from the training data. Model parameters are learned during training when we optimise a loss function using something like a gradient descent. 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 wins. The pattern followed here is similar to the grid, where all th e values are placed in the form of a matrix. Each set of parameters is taken into consideration and the accuracy is noted. Once all the combinations are evaluated, the model with the set o f parameters which give the top ac curacy is considered to be the best.

We used these two techniques for Decision tree,Random Fo rest ,Gredient boosting models.

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)

3.1.1 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.

3.1.2 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.

3.1.3 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.

PREDICTED MODELS IN PYTHON:

```
Final_Results
```

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Linear Regression	45.480011	17.528674	0.835720	0.847770	776.486700	782.728203
1	Decision Tree	81.433513	29.346978	0.669888	0.725704	1100.706423	1050.678261
2	Random Search CV Decision Tree	8.834388	19.961435	0.929061	0.799082	510.252314	899.227765
3	Grid Search CV Decision Tree	8.834388	19.961435	0.929061	0.799082	510.252314	899.227765
4	Random Forest	16.726797	17.056622	0.981439	0.871775	261.001514	718.368952
5	Random Search CV Random Forest	17.740802	17.042779	0.981037	0.874703	263.810977	710.117599
6	Grid search CV Random Forest	21.536863	17.406332	0.964378	0.872677	361.574910	715.836573

PREDICTED OUTPUT IN R:

	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

3.2 MODEL SELECTION:

From the predicted output in R and Python, the random forest model can have explained almost 90% of the predictor matches with the target variable.the values of the random forest model is mentioned below.

- **MAPE = 0.83**
- **R-SQUARED =0.87**
- **RMSE = 718.36**

4. R-CODE

```
#************************************************************************#
# Bike Rental Prediction ----------------------------------------------
#************************************************************************ #

# Clean the environment

rm(list=ls())

# Set working directory

setwd("D:/001. Data sc/Project/Bike_Rental")

# Cross Check current working directory
getwd()


# Load the required libraries for analysis of data
x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced","C50",
      "dummies", "e1071", "Information", "MASS", "rpart", "gbm", "ROSE",
      'sampling', 'DataCombine', 'inTrees',"scales","psych","gplots")
#install.packages(x)
lapply(x, require, character.only = TRUE)
rm(x)

# Load the data
Bike_Rent = read.csv("day.csv",header = T,na.strings = c("","  ",NA))


# Explore the data

#Check the dimensions(no of rows and no of columns)
dim(Bike_Rent)

# Check names of dataset,in names we can see shortcuts like, hum for humidity
#yr for year ,mnth for month,cnt for count
names(Bike_Rent)

#Rename the variables-
names(Bike_Rent)[4]  = "year"
names(Bike_Rent)[5]  = "month"
names(Bike_Rent)[9]  = "weather"
names(Bike_Rent)[10] = "temperature"
names(Bike_Rent)[12] = "humidity"
names(Bike_Rent)[16] = "count"

#Check top(first) rows of dataset
head(Bike_Rent)

#Check bottom(last) rows of dataset
tail(Bike_Rent)

#Check structure of dataset(data structure of each variable)
str(Bike_Rent)

#Check summary of dataset
summary(Bike_Rent)

# Variable Identification
# In this dataset cnt is our target variable and it is continous variable
str(Bike_Rent$count)

# Remove these variables
# instant variable, as it is index in dataset
# date variable as we have to predict count on seasonal basis not date basis
# casual and registered variable as count is sum of these two variables
# cnt = casual + registered

Bike_Rent = subset(Bike_Rent,select=-c(instant,dteday,casual,registered))
```

```
# Lets check dimensions of data after removing some variables
dim(Bike_Rent)

# Make Seperate categorical and numerical variables dataframe
# Continous Variables
cnames= c("temperature","atemp","humidity","windspeed","count")

# Categorical varibles-
cat_cnames= c("season","year","month","holiday","weekday","workingday","weather")
```

 # EDA or Data Preprocessing

```
# Duplicate Values
# duplicated(Bike_Rent)# No duplicates in dataset
```

 # Missing Value anlysis

```
# Check missing values in dataset
sum(is.na(Bike_Rent))

# there is no missing values present in this dataset
```

 # Outlier Analysis and treatment

```
# Lets save copy of dataset before preprocessing
df = Bike_Rent
Bike_Rent = df

# Lets use boxplot to detect the outliers
# We use ggplot library to plot boxplot for each numeric variable
for(i in 1:length(cnames))
{
  assign(paste0("gn",i),ggplot(aes_string(y=(cnames[i]),x = 'count'),
                       data=subset(Bike_Rent))+
         stat_boxplot(geom = "errorbar",width = 0.5) +
         geom_boxplot(outlier.color = "red",fill="grey",
                      outlier.shape = 18,outlier.size = 1,notch = FALSE)+
         theme(legend.position = "bottom")+
         labs(y = cnames[i],x='count')+
         ggtitle(paste("boxplot of count for",cnames[i])))
}

# using library(gridExtra)
gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,ncol = 2)


# Loop to remove outliers by capping upperfence and lower fence values
for(i in cnames){
  print(i)
  #Quartiles
  Q1 = quantile(Bike_Rent[,i],0.25)
  Q3 = quantile(Bike_Rent[,i],0.75)

  #Inter quartile range
  IQR = Q3-Q1

  # Upperfence and Lower fence values
  UL = Q3 + (1.5*IQR(Bike_Rent[,i]))
  LL = Q1 - (1.5*IQR(Bike_Rent[,i]))

  # No of outliers and inliers in variables
  No_outliers = length(Bike_Rent[Bike_Rent[,i] > UL,i])
  No_inliers = length(Bike_Rent[Bike_Rent[,i] < LL,i])

  # Capping with upper and inner fence values
  Bike_Rent[Bike_Rent[,i] > UL,i] = UL
  Bike_Rent[Bike_Rent[,i] < LL,i] = LL

}

# Lets plot boxplots after removing outiers
for(i in 1:length(cnames))
```

```
{
  assign(paste0("gn",i),ggplot(aes_string(y=(cnames[i]),x = 'count'),
                         data=subset(Bike_Rent))+
         stat_boxplot(geom = "errorbar",width = 0.5) +
         geom_boxplot(outlier.color = "red",fill="grey",
                      outlier.shape = 18,outlier.size = 1,notch = FALSE)+
         theme(legend.position = "bottom")+
         labs(y = cnames[i],x='count')+
         ggtitle(paste("boxplot of count for",cnames[i])))
}

# using library(gridExtra)
gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,ncol = 2)


# Data understanding using visualization

# Lets look at numeric and categorical variables
# Continous Variables
cnames= c("temperature","atemp","humidity","windspeed","count")

# Categorical varibles-
cat_cnames= c("season","year","month","holiday","weekday","workingday","weather")

# Univariate Analysis
# Histogram for continuous variables to check  distribution of each variable
for(i in 1:length(cnames))
{
  assign(paste0("h",i),ggplot(aes_string(x=(cnames[i])),
                         data=subset(Bike_Rent))+
         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 of ",cnames[i])))
}

# using library(gridExtra)
gridExtra::grid.arrange(h1,h2,h3,h4,h5,ncol = 2)

# Bivariate Analysis
# Lets check impact of continous variables on target variable
for(i in 1:length(cnames))
{
  assign(paste0("s",i),ggplot(aes_string(y='count',x = (cnames[i])),
                         data=subset(Bike_Rent))+
         geom_point(alpha=0.5,color="DarkSlateBlue") +
         # labs(title = "Scatter Plot of count vs", x = (cnames[i]), y = "count")+
         ggtitle(paste("Scatter Plot of count vs",cnames[i])))
}

# using library(gridExtra)
gridExtra::grid.arrange(s1,s2,s3,s4,s5,ncol = 2)

# count vs temperature(atemp) : as temperature increase Bike rent count also increases
# count vs humidity : humidity doesnt have any effect on bikerent count
# count vs windspeed : windspeed doesnt have any effect on bikerent count
# count vs count : please ignore this plot as it is our target variable

options(scipen = 999)
# Let us check impact of categorical variables on count

for(i in 1:length(cat_cnames))
{
  assign(paste0("b",i),ggplot(aes_string(y='count',x = (cat_cnames[i])),
                         data=subset(Bike_Rent))+
         geom_bar(stat = "identity",fill = "DarkSlateBlue") +
         # labs(title = "Scatter Plot of count vs", x = (cnames[i]), y = "count")+
         ggtitle(paste("Number of bikes rented with respect to",cat_cnames[i])))+
    theme(axis.text.x = element_text( color="black", size=8))+
    theme(plot.title = element_text(face = "bold"))
}

# using library(gridExtra)
gridExtra::grid.arrange(b1,b2,b3,b4,ncol = 2)
```

```
gridExtra::grid.arrange(b5,b6,b7,ncol = 2)

# From barplot we can observe below points
# Season:Bike rent count is high in season 3(fall) and low in season 1(springer)
aggregate(count ~ season ,sum,data = Bike_Rent)

# year : Bike rent count is high in year 1 (in 2012)
aggregate(count ~ year ,sum,data = Bike_Rent)

# month : Bike rent count is high in month of august and low in jan
aggregate(count ~ month,sum,data = Bike_Rent)

# holiday : Bike rent count is high on holidays ie 0
aggregate(count ~ holiday ,sum,data = Bike_Rent)

# weekday :From bar plot we can see maximum bikes rented on 5th day and least bikes on
day 0.
aggregate(count ~ weekday ,sum,data = Bike_Rent)

# workingday : Bike rent count is high on working day  ie 1
aggregate(count ~ workingday,sum,data = Bike_Rent)

# weather : Bike rent count is high on weather 1: ie when the weather is
# Clear, Few clouds, Partly cloudy, Partly cloudy
aggregate(count ~ weather,sum,data = Bike_Rent)


# Bikes rented with respect to temp and humidity
ggplot(Bike_Rent,aes(temperature,count)) +
  geom_point(aes(color=humidity),alpha=0.5) +
  labs(title = "Bikes rented with respect to variation in temperature and humidity", x =
"temperature")+ theme_bw()
# maximum bike rented between temp 0.50 to 0.75 and humidity 0.50 to 0.75

#Bikes rented with respect to temp and weather
ggplot(Bike_Rent, aes(x = temperature, y = count))+
  geom_point(aes(color=weather))+
  labs(title = "Bikes rented with respect to temperature and weather", x = "temperature")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme_bw()

# maximum bike rented with windspeed and normalized temp between 0.50 to 0.75
# and when the weathersite is 1

# Bikes rented with respect to temp and season
ggplot(Bike_Rent, aes(x = temperature, y = count))+
  geom_point(aes(color=season))+
  labs(title = "Bikes rented with respect to temperature and season", x = "temperature")+
  # theme(panel.background = element_rect("white"))+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme_bw()
# From figure it is clear that maximum bike count is for season 2 and 3,
# when the temp between 0.5 to 0.7
```

 # Feature Selection

```
# Lets save dataset after outlier analysis
df =  Bike_Rent
Bike_Rent = df

# Using corrplot library we do correlation analysis for numeric variables
# Let us derive our correlation matrix
Correlation_matrix = cor(Bike_Rent[,cnames])
Correlation_matrix
# By looking at correlation matrix we can say temperature and atemp
# are highly correlated (>0.99)

#Lets plot correlation plot using corrgram library
corrgram(Bike_Rent[,cnames],order = F,upper.panel = panel.pie,
         text.panel = panel.txt,main="Correlation plot for numeric variables")

# From correlation analysis temp and atemp variables are highly correlated
# so delete atemp variable

# Lets find significant categorical variables usig ANOVA test
```

```
# Anova analysis for categorical variable with target numeric variable
for(i in cat_cnames){
  print(i)
  Anova_result= summary(aov(formula = count~ Bike_Rent[,i],Bike_Rent))
  print(Anova_result)
}


#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_Rent = subset(Bike_Rent,select = -c(atemp,holiday,weekday,workingday))


# Lets check dimensions after dimension reduction
dim(Bike_Rent)

head(Bike_Rent)

# Lets check column names after dimension reduction
names(Bike_Rent)

# Lets define/update  continous and categorical variables after dimension reduction
# Continuous variable
cnames= c('temperature','humidity', 'windspeed', 'count')

# Categorical variables
cat_cnames = c('season', 'year', 'month','weather')


# Feature Scaling
# Since as it is mentioned in data dictionary the values of
# temp,humidity,windspeed variables are already normalized values so no
# need to go for feature scaling instead we will visualize the variables
# to see normality

# Normality check using normal qq plot
for(i in cnames){
  print(i)
  qqplot= qqnorm(Bike_Rent[,i])
}

# Normality check using histogram plot(we already plotted hist in
# data understanding)
gridExtra::grid.arrange(h1,h2,h3,h4,h5,ncol = 2)

#check summary of continuous variable to check the scaling-
for(i in cnames){
  print(i)
  print(summary(Bike_Rent[,i]))
}

# From normal qq plot,histplot and by looking at summary of
# numeric variables we can say data is normally distributed


# Model Development
# Let's clean R Environment, as it uses RAM which is limited
library(DataCombine)
rmExcept("Bike_Rent")

# Lets convert all categorical variables ito dummy variables
# As we cant pass categorical variables directly in to regression problems
# Lets save our preprocessed data into df data set
df = Bike_Rent
Bike_Rent = df

# Lets call Categorical varaibles after feature selection using ANOVA
cat_cnames= c("season","year","month","weather")
```

```
# lets create dummy variables using dummies library
library(dummies)
Bike_Rent = dummy.data.frame(Bike_Rent,cat_cnames)
dim(Bike_Rent)
head(Bike_Rent)
# we can see dummy variables are created in Bike rent dataset

# Divide data into train and test sets
set.seed(1234)
train.index = createDataPartition(Bike_Rent$count, p = .80, list = FALSE)
train = Bike_Rent[ train.index,]
test  = Bike_Rent[-train.index,]

# Function for Error metrics to calculate the performance of model
mape= function(y,yhat){
  mean(abs((y-yhat)/y))*100
}

# Function for r2 to calculate the goodness of fit of model
rsquare=function(y,yhat){
  cor(y,yhat)^2
}

# Function for RMSE value
rmse = function(y,yhat){
  difference = y - yhat
  root_mean_square = sqrt(mean(difference^2))
  print(root_mean_square)
}
```

 # Linear Regression model

```
# Before building multiple linear regression model lets check the
# vif for multicolinearity
# continous variables after feature selection using correlation analysis
cnames= c("temperature","humidity","windspeed")
numeric_data= Bike_Rent[,cnames]

# VIF test  using usdm library
library(usdm)
vifcor(numeric_data,th=0.6)
# No variable from the 3 input variables has collinearity problem.

# Lets build multiple linear regression model on train data
# we will use the lm() function in the stats package
LR_Model = lm(count ~.,data = train)

# Check summary
summary(LR_Model) # Adjus.Rsquare = 0.833

# Lets check the assumptins of ols regression
# 1) Error should follow normal distribution - Normal qqplot
# 2) No heteroscadacity - Residual plot
par(mfrow = c(2, 2))# Change the panel layout to 2 x 2
plot(LR_Model)
# 3) No multicolinearity between Independent variables
# 4) No autocorrelation between errors
library(car)
dwt(LR_Model)
# All Asumptions of regression are satisfied

# Lets predict on train data
LR_train = predict(LR_Model,train[,-25])
# Now Lets predict on test data
LR_test= predict(LR_Model,test[-25])

# Lets check performance of model
# MAPE For train data
LR_MAPE_Train =mape(train[,25],LR_train)
# MAPE For test data
LR_MAPE_Test=mape(test[,25],LR_test)

# Rsquare For train data
```

```
LR_r2_train=rsquare(train[,25],LR_train)
# Rsquare For test data
LR_r2_test=rsquare(test[,25],LR_test)

# rmse For train data
LR_rmse_train = rmse(train[,25],LR_train)
# rmse For test data
LR_rmse_test = rmse(test[,25],LR_test)


print(LR_MAPE_Train)
print(LR_MAPE_Test)
print(LR_r2_train)
print(LR_r2_test)
print(LR_rmse_train)
print(LR_rmse_test)


########################################################################################
#
# Lets build some more models using different ml algorithms for more accuracy
# and less prediction error
########################################################################################
###
```

 # Desicision Tree

```
# Lets Build decision tree model on train data using rpart library
DT_model= rpart(count~.,train,method = "anova")
DT_model

# Lets plot the decision tree model using rpart.plot library
library(rpart.plot)
rpart.plot(DT_model,type=4,digits=3,fallen.leaves=T,tweak = 2)

# Prediction on train data
DT_train= predict(DT_model,train[-25])


# Prediction on test data
DT_test= predict(DT_model,test[-25])

# MAPE For train data
DT_MAPE_Train = mape(train[,25],DT_train)

# MAPE For train data test data
DT_MAPE_Test = mape(test[,25],DT_test)

# Rsquare  For train data
DT_r2_train= rsquare(train[,25],DT_train)

# Rsquare For test data
DT_r2_test= rsquare(test[,25],DT_test)

# rmse For train data
DT_rmse_train = rmse(train[,25],DT_train)

# rmse For test data
DT_rmse_test = rmse(test[,25],DT_test)
```

 # Random Search CV In Decision Tree

```
# Lets set parameters to pass into our decision tree model
# Lets use caret package for the same
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, method="rpart2",trControl=control,tuneGrid=
tunegrid)
print(RDT_model)
```

```
# Lets look into best fit parameters
best_fit_parameters = RDT_model$bestTune
print(best_fit_parameters)

# Again rebuild decision tree model using randomsearch best fit parameter ie
# with maximum depth = 9
RDT_best_model = rpart(count~.,train,method = "anova",maxdepth=9)

# Prediction on train data
RDT_train = predict(RDT_best_model,train[-25])

# Prediction on test data
RDT_test = predict(RDT_best_model,test[-25])

# Lets check Model performance on both test and train

# MAPE for train data
RDT_MAPE_Train =mape(train[,25],RDT_train)

# MAPE for test data
RDT_MAPE_Test =mape(test[,25],RDT_test)

# Rsquare for train data
RDT_r2_train =  rsquare(train[,25],RDT_train)

# Rsquare for test data
RDT_r2_test = rsquare(test[,25],RDT_test)

# rmse For train data
RDT_rmse_train = rmse(train[,25],RDT_train)

# rmse For test data
RDT_rmse_test = rmse(test[,25],RDT_test)
```

 # Grid Search CV in Decision Tree

```
control =trainControl(method="repeatedcv", number=5, repeats=2, search="grid")
tunegrid = expand.grid(.maxdepth=c(6:20))

# Lets build a model using above parameters on train data
GDT_model = caret::train(count~.,train, method="rpart2", tuneGrid=tunegrid,
trControl=control)
print(GDT_model)

# Lets look into best fit parameters from gridsearch cv DT
best_parameter = GDT_model$bestTune
print(best_parameter)

# Again rebuild decision tree model using gridsearch best fit parameter ie
# with maximum depth = 9
GDT_best_model = rpart(count ~ .,train, method = "anova", maxdepth =9)

# Prediction on train data
GDT_train = predict(GDT_best_model,train[-25])

# Prediction on test data
GDT_test = predict(GDT_best_model,test[-25])

# Mape for train data using gridsearch cv  DT
GDT_MAPE_Train = mape(train[,25],GDT_train)

# Mape for test data using gridsearch cv  DT
GDT_MAPE_Test = mape(test[,25],GDT_test)

# Rsquare for train data
GDT_r2_train= rsquare(train[,25],GDT_train)

# Rsquare for test data
GDT_r2_test=rsquare(test[,25],GDT_test)

# rmse For train data
GDT_rmse_train = rmse(train[,25],GDT_train)
```

```
# rmse For test data
GDT_rmse_test = rmse(test[,25],GDT_test)


# Random Forest
# Lets Build random forest model on train data using randomForest library
RF_model= randomForest(count~.,train,ntree=100,method="anova")

# Prediction on train data
RF_train= predict(RF_model,train[-25])

# Prediction on test data
RF_test = predict(RF_model,test[-25])

# MAPE For train data
RF_MAPE_Train = mape(train[,25],RF_train)

# MAPE For test data
RF_MAPE_Test = mape(test[,25],RF_test)

# Rsquare  For train data
RF_r2_train=rsquare(train[,25],RF_train)

# Rsquare For test data
RF_r2_test=rsquare(test[,25],RF_test)

# rmse For train data
RF_rmse_train = rmse(train[,25],RF_train)

# rmse For test data
RF_rmse_test = rmse(test[,25],RF_test)


# Random Search CV in Random Forest
control = trainControl(method="repeatedcv", number=5, repeats=1,search='random')
maxdepth = c(1:30)
tunegrid = expand.grid(.maxdepth=maxdepth)

# Lets build modelon train data using random search
RRF_model = caret::train(count~., data=train,
method="rf",trControl=control,tuneLength=10)
print(RRF_model)

# Best fit parameters
best_parameter = RRF_model$bestTune

print(best_parameter)

# Lets build model based on best fit parameters
RRF_best_model = randomForest(count ~ .,train, method = "anova",
mtry=9,importance=TRUE)

# Prediction on train data
RRF_train= predict(RRF_best_model,train[-25])

# Prediction on test data
RRF_test= predict(RRF_best_model,test[-25])

# Mape for train data
RRF_MAPE_Train = mape(train[,25],RRF_train)

# Mape for test data
RRF_MAPE_Test = mape(test[,25],RRF_test)

# Rsquare for train data
RRF_r2_train = rsquare(train[,25],RRF_train)

# Rsquare for test data
RRF_r2_test = rsquare(test[,25],RRF_test)

# rmse For train data
RRF_rmse_train = rmse(train[,25],RRF_train)
```

```
# rmse For test data
RRF_rmse_test = rmse(test[,25],RRF_test)
```

 # Grid search CV in Random Forest

```
control = trainControl(method="repeatedcv", number=5, repeats=2, search="grid")
tunegrid = expand.grid(.mtry=c(6:20))

# Lets build a model using above parameters on train data using random forest grid search
cv
GRF_model= caret::train(count~.,train, method="rf", tuneGrid=tunegrid, trControl=control)
print(GRF_model)

# Best fit parameters
best_parameter = GRF_model$bestTune
print(best_parameter)

# Build model based on Best fit parameters
GRF_best_model = randomForest(count ~ .,train, method = "anova", mtry=10)

# Prediction for train data
GRF_train= predict(GRF_best_model,train[-25])

# Prediction for test data
GRF_test= predict(GRF_best_model,test[-25])

# Mape for train data
GRF_MAPE_Train = mape(train[,25],GRF_train)

# Mape calculation of test data
GRF_MAPE_Test = mape(test[,25],GRF_test)

# Rsquare for train data
GRF_r2_train= rsquare(train[,25],GRF_train)

# Rsquare for test data
GRF_r2_test=rsquare(test[,25],GRF_test)

# rmse For train data
GRF_rmse_train = rmse(train[,25],GRF_train)

# rmse For test data
GRF_rmse_test = rmse(test[,25],GRF_test)
```

 # Gradient Boosting

```
library(gbm)

# Lets build a Gradient Boosting model for regression problem
GB_model = gbm(count~., data = train,distribution = "gaussian", n.trees = 100,
interaction.depth = 2)

# Model Prediction on train data
GB_train = predict(GB_model, train[-25], n.trees = 100)

# Model Prediction on test data
GB_test = predict(GB_model, test[-25], n.trees = 100)

# Mape for train data
GB_MAPE_Train=mape(train[,25],GB_train)
# Mape for test data
GB_MAPE_Test=mape(test[,25],GB_test)

# Rsqure for train data
GB_r2_train=rsquare(train[,25],GB_train)

# Rsquare for test data
GB_r2_test=rsquare(test[,25],GB_test)

# rmse For train data
GB_rmse_train = rmse(train[,25],GB_train)

# rmse For test data
```

```
GB_rmse_test = rmse(test[,25],GB_test)


# Random Search CV in Gradient Boosting
control = trainControl(method="repeatedcv", number=5, repeats=1,search="random")
#maxdepth = c(1:30)
#tunegrid = expand.grid(.maxdepth=maxdepth)

# Model devlopment on train dat
RGB_model = caret::train(count~., data=train,
method="gbm",trControl=control,tuneLength=10)

print(RGB_model)

# Best fit parameters
best_parameter = RGB_model$bestTune
print(best_parameter)

# Build model based on best fit
RGB_best_model = randomForest(count ~ .,train, method = "anova", n.trees=50,
                              interaction.depth=5,shrinkage=0.1,n.minobsinnode=10)

# Prediction on train data
RGB_train= predict(RGB_best_model,train[-25])

# Prediction on test data
RGB_test= predict(RGB_best_model,test[-25])

# Mape calculation of train data
RGB_MAPE_Train = mape(train[,25],RGB_train)

# Mape calculation of test data
RGB_MAPE_Test = mape(test[,25],RGB_test)

# Rsquare calculation for train data
RGB_r2_train= rsquare(train[,25],RGB_train)

# Rsquare calculation for test data
RGB_r2_test=rsquare(test[,25],RGB_test)

# rmse For train data
RGB_rmse_train = rmse(train[,25],GRF_train)

# rmse For test data
RGB_rmse_test = rmse(test[,25],GRF_test)

# Results

Model = c('Linear Regression','Decision Tree for Regression',
          'Random Search in Decision Tree','Gird Search in Decision Tree',
          'Random Forest','Random Search in Random Forest',
          'Grid Search in Random Forest','Gradient Boosting',
          'Random Search in Gradient Boosting')
MAPE_Train = c(LR_MAPE_Train,DT_MAPE_Train,RDT_MAPE_Train,
               GDT_MAPE_Train,RF_MAPE_Train,RRF_MAPE_Train,GRF_MAPE_Train,
               GB_MAPE_Train,RGB_MAPE_Train)

MAPE_Test = c(LR_MAPE_Test,DT_MAPE_Test,RDT_MAPE_Test,
              GDT_MAPE_Test,RF_MAPE_Test,RRF_MAPE_Test,GRF_MAPE_Test,
              GB_MAPE_Test,RGB_MAPE_Test)

Rsquare_Train = c(LR_r2_train,DT_r2_train,RDT_r2_train,GDT_r2_train,
                  RF_r2_train,RRF_r2_train,GRF_r2_train,GB_r2_train,
                  RGB_r2_train)

Rsquare_Test = c(LR_r2_test,DT_r2_test,RDT_r2_test,GDT_r2_test,
                 RF_r2_test,RRF_r2_test,GRF_r2_test,GB_r2_test,
                 RGB_r2_test)

Rmse_Train = c(LR_rmse_train,DT_rmse_train,RDT_rmse_train,GDT_rmse_train,
               RF_rmse_train,RRF_rmse_train,GRF_rmse_train,GB_rmse_train,
               RGB_rmse_train)

Rmse_Test = c(LR_rmse_test,DT_rmse_test,RDT_rmse_test,GDT_rmse_test,
```

```
        RF_rmse_test,RRF_rmse_test,GRF_rmse_test,GB_rmse_test,
        RGB_rmse_test)

Final_results = data.frame(Model,MAPE_Train,MAPE_Test,Rsquare_Train,
                  Rsquare_Test,Rmse_Train,Rmse_Test)
```

Final_results

```
# 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)
Pred_count_RF_test = predict(RF_model,test[-25])

# Exporting the output to hard disk for further use
test <- as.data.frame(cbind(test,Pred_count_RF_test))

Final_output <- as.data.frame(cbind(test$count,Pred_count_RF_test))

names(Final_output)[1] <- "Actual_Bike_Rent_Count"

write.csv(Final_output,"D:/001. Data sc/Project/Bike_Rental/RF_output_R.csv",
      row.names = FALSE)

# -------------------------------END-----------------------------------------#
```

5. PYTHON CODE

Bike Rental Prediction

```
# 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:/001. Data sc/Project/Bike_Rental")

# 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','weath
ersit':'weather',
                    'temp':'temperature','hum':'humidity','cnt':'count'})

Bike_Data.columns


#lets see first five observations of our data
Bike_Data.head()

# lets see last five observations of
our data Bike_Data.tail()
```

```
# 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_Dat
  a[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_Dat
  a[i]) plt.xlabel(i)
  plt.ylabel("values")
  plt.title("Boxplot of "+i)
  plt.show()
```

Visualization

```
# 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

# Bivariate Analysis ----------------------------------------------------
# Lets check impact of continous variables on target variable

# count vs temperature
sns.violinplot(x='count',y='temperature',data=Bike_Data)

#temperature is directly proportional to each other
#as temperature increases bike rental count also increases

# count vs humidity

sns.violinplot(x='count',y='humidity',data=Bike_Data)

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

# count vs windspeed

sns.violinplot(x='count',y='windspeed',data=Bike_Data)

# 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 bik es 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 sam e 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)

#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

#  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 variables lets check column names of the
data Bike_Data.columns

#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()

for i in cnames:
    print(i)
    sns.distplot(Bike_Data[i],bins='auto',color='blue')
    plt.title("Distribution plot for "+i)
    plt.ylabel("Density")
    plt.show()

Bike_Data.describe()

#we confirmed the normalized data based on the qqplot,distribution plot 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= rando
m_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,r square 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))

Error_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 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,r square 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
# 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= ran
dom_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,r square 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,r square 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
```

Final_Results

 # From results Random Forest model have optimum values and this algorithm is good for our data

```
Final_Results
```

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Linear Regression	45.480011	17.528674	0.835720	0.847770	776.486700	782.728203
1	Decision Tree	81.433513	29.346978	0.669888	0.725704	1100.706423	1050.678261
2	Random Search CV Decision Tree	8.834388	19.961435	0.929061	0.799082	510.252314	899.227765
3	Grid Search CV Decision Tree	8.834388	19.961435	0.929061	0.799082	510.252314	899.227765
4	Random Forest	16.726797	17.056622	0.981439	0.871775	261.001514	718.368952
5	Random Search CV Random Forest	17.740802	17.042779	0.981037	0.874703	263.810977	710.117599
6	Grid search CV Random Forest	21.536863	17.406332	0.964378	0.872677	361.574910	715.836573

REFERENCES

Edwisor Learning
https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/
https://www.geeksforgeeks.org/haversine-formula-to-find-distance-between-two-points-on-a-sphere/
https://towardsdatascience.com/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regression-metrics-d4a1a9ba3d74
https://towardsdatascience.com/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-2-regression-metrics-d4a1a9ba3d74
https://medium.com/data-distilled/residual-plots-part-1-residuals-vs-fitted-plot-f069849616b1
Machine Learning made easy using R by udemy
https://medium.com/@TheDataGyan/day-8-data-transformation-skewness-normalization-and-much-more-4c144d370e55