Bike Rental Count Prediction

Laxmi S Kallimath

10th August

Contents

1. Introduction .....

1.1 Problem Statement

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

1.2 Data

We have to predict bike rental count on daily based on the environmental and seasonal settings. and since our target variable 'count'('cnt') is a continuous Variable therefore this problem comes under supervised machine learning Regression problem. Dataset has 16 variables in which 15 variables are independent and 1 ('cnt') is dependent variable and there are 731 observations.

Table 1.1 Sample data

	instant	dteday	season	year	month	holiday	weekday	workingday	weather
0	1	2011-01-01	1	0	1	0	6	0	2
1	2	2011-01-02	1	0	1	0	0	0	2
2	3	2011-01-03	1	0	1	0	1	1	1
3	4	2011-01-04	1	0	1	0	2	1	1
4	5	2011-01-05	1	0	1	0	3	1	1

temperature	atemp	humidity	windspeed	casual	registered	count
0.344167	0.363625	0.805833	0.160446	331	654	985
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349
0.200000	0.212122	0.590435	0.160296	108	1454	1562
0.226957	0.229270	0.436957	0.186900	82	1518	1600

Data Dictionary: - The details of data attributes in the dataset are as follows let's understand each attribute in detail

The details of data attributes in the dataset are as follows -

1) instant: Record index
2) dteday: Date
3) season: Season (1:springer, 2:summer, 3:fall, 4:winter)
4) yr: Year (0: 2011, 1:2012)
5) mnth: Month (1 to 12)
6) hr: Hour (0 to 23)
7) holiday: weather day is holiday or not (extracted fromHoliday Schedule)
8) weekday: Day of the week
9) workingday: If day is neither weekend nor holiday is 1, otherwise is 0.

10) weathersit: (extracted fromFreemeteo)
    1: Clear, Few clouds, Partly cloudy, Partly cloudy
    2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
    4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
11) temp: Normalized temperature in Celsius. The values are derived via
    (t-t_min)/(t_max-t_min),
    t_min=-8, t_max=+39 (only in hourly scale)
12) atemp: Normalized feeling temperature in Celsius. The values are derived via
    (t-t_min)/(t_maxt_
    min), t_min=-16, t_max=+50 (only in hourly scale)
13) hum: Normalized humidity. The values are divided to 100 (max)
14) windspeed: Normalized wind speed. The values are divided to 67 (max)
15) casual: count of casual users
16) registered: count of registered users
17) cnt: count of total rental bikes including both casual and registered

2.Methodology .....

2.1 Data Pre –Processing/Exploratory Data Analysis

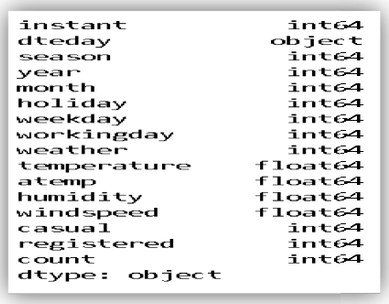
Data preprocessing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn; therefore, it is extremely important that we preprocess our data before feeding it into our model. As we already know the quality of our inputs decide the quality of our output. So, once we have got our business hypothesis ready, it makes sense to spend lot of time and efforts here. Approximately, data exploration, cleaning and preparation can take up to 70% of our total project time. This process is often called as Exploratory Data Analysis

Below are data preprocessing techniques used for this Project

1. Data Exploration and Cleaning
2. Missing Value Analysis
3. Outlier analysis
4. Data understanding using visualization
5. Feature Selection
6. Feature Scaling

2.1.1 Data Exploration and Cleaning

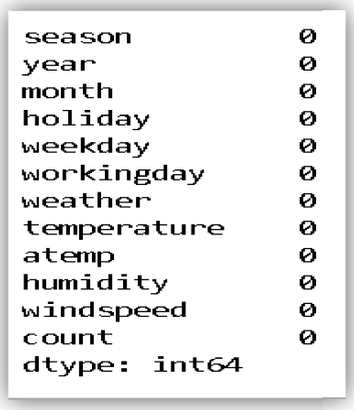
In this step we first imported the data into R and Python environments ,explored the data by l ooking its dimensions,datastructures,variable names, summary of data ,to have a glance at our data we checked first and last rows of the train dataset .renamed variables for more understa nding of the data,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



```
instant              int64
dteday              object
season               int64
year                 int64
month                int64
holiday              int64
weekday              int64
workingday           int64
weather              int64
temperature        float64
atemp              float64
humidity           float64
windspeed          float64
casual               int64
registered           int64
count                int64
dtype: object
```

2.1.2 Missing Value Analysis

Missing data or missing values occur when no data value is stored for the variable in an observation. In any real world dataset there are always few null values. It doesn't really matter whether it is a regression, classfication or any other kind of problem, no model can handle these NULL or NaN values on its own so we need to intervene. They are often encoded as NaNs, blanks or any other placeholders. If columns have more than 30% of data as missing value either we ignore the entire column or we ignore those observations. 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

Outlier is an observation that appears far away and diverges from an overall pattern in a sample. These outliers occur in data due to many reasons like data entry errors, Measurment error, Experimental error, intentional errors etc.

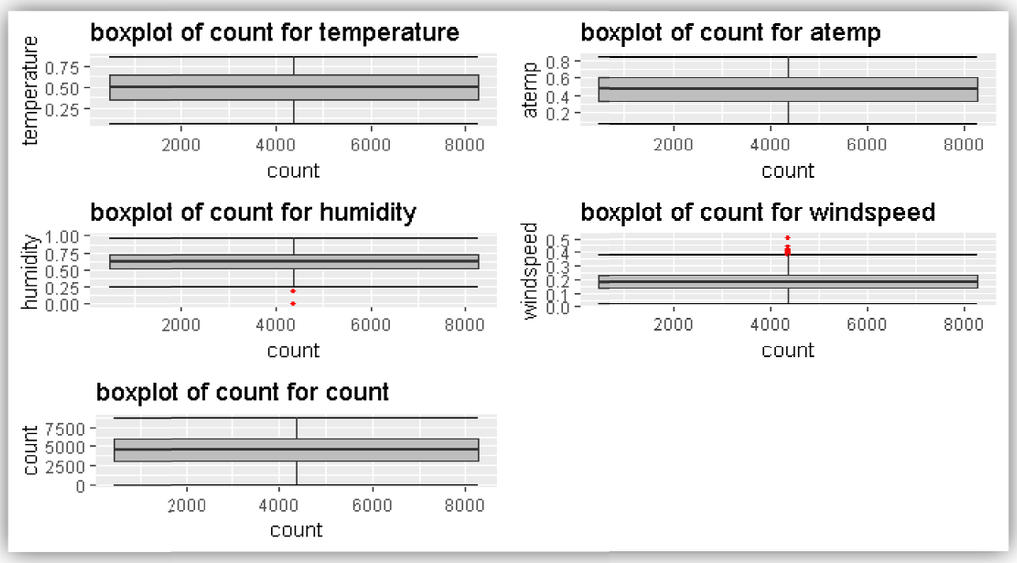
For our data we used box plots to visualize and detect the outliers , used summary descriptive statistics to check range of each numeric variable and Sorted the variables.

From the boxplot almost all the variables **except "windspeed" and "humidity"** does not have outliers .

	instant	season	year	month	holiday	weekday	workingday	weather
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
mean	366.000000	2.496580	0.500684	6.519836	0.028728	2.997264	0.683995	1.395349
std	211.165812	1.110807	0.500342	3.451913	0.167155	2.004787	0.465233	0.544894
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000
25%	183.500000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000
50%	366.000000	3.000000	1.000000	7.000000	0.000000	3.000000	1.000000	1.000000
75%	548.500000	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	2.000000
max	731.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000

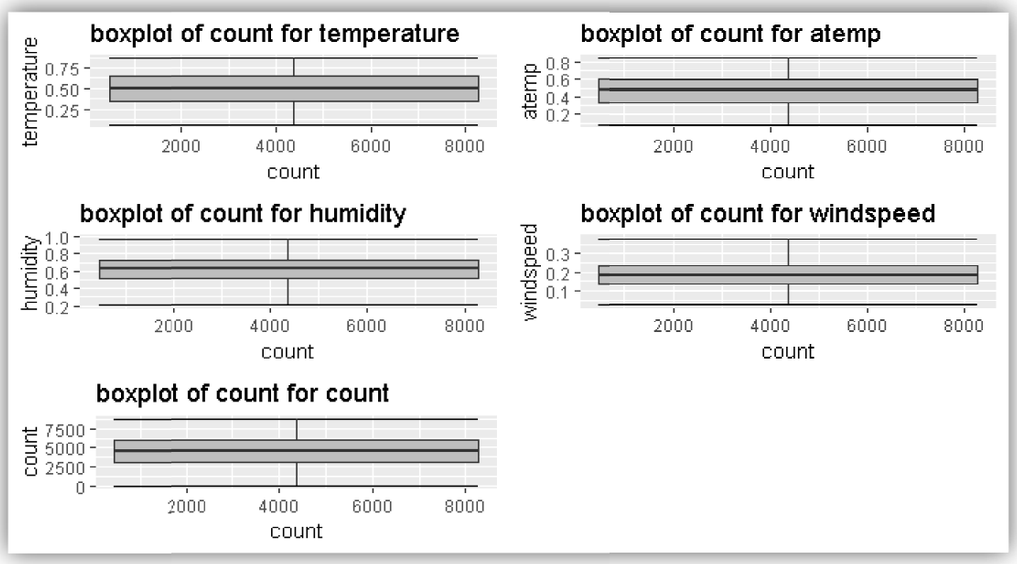
atemp	humidity	windspeed	casual	registered	count
731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
0.474354	0.627894	0.190486	848.176471	3656.172367	4504.348837
0.162961	0.142429	0.077498	686.622488	1560.256377	1937.211452
0.079070	0.000000	0.022392	2.000000	20.000000	22.000000
0.337842	0.520000	0.134950	315.500000	2497.000000	3152.000000
0.486733	0.626667	0.180975	713.000000	3662.000000	4548.000000
0.608602	0.730209	0.233214	1096.000000	4776.500000	5956.000000
0.840896	0.972500	0.507463	3410.000000	6946.000000	8714.000000

Boxplot of continous variables with outliers



For our project we opted for capping method in which we are going to impute outlier with upper fence and lower fence value reason behind to opt this method is we don't want to delete the observations with outliers as data collection is also a crucial step in data analytics for which client has to spend more money if don't have any past data specially for startup companies .by keeping this point into consideration we tried to retain the data wherever possible in the preprocessing. Boxplot stat method is one of the method to remove outlier and we also learnt we can also treat these outliers as missing values and impute them with mean or median or knn method or delete such observations

Boxplots of continous variables after removing outliers

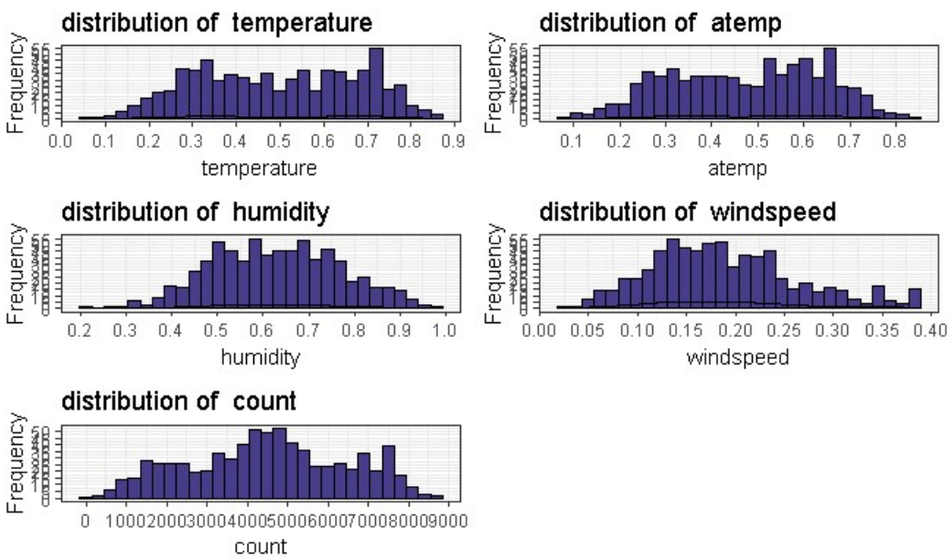


2.1.4 Data understanding using visualization .....

2.1.4.1 Distribution of each continuous variables

To check distribution of each continuous variable we plotted histogram for each variable Both in R and Python ,we can also check distribution using summary or describe function.

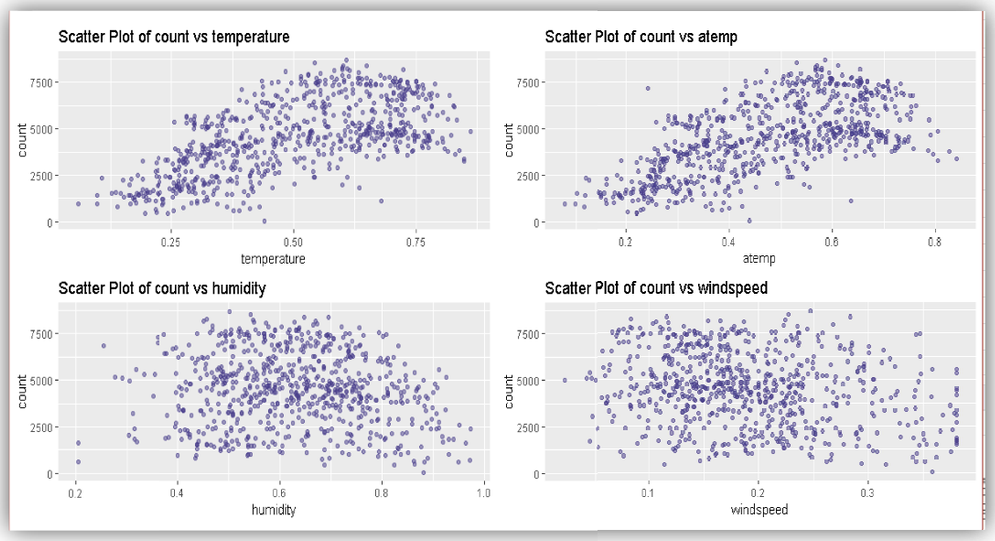
From the below plot we can say predictor and target variables are normally distributed.(temp and atemp are same ,atemp is removed in further analysis)



2.1.5.2 Effect of continous variables wrt target variable

Lets check the impact of each continous variables on target variable using scatterplot

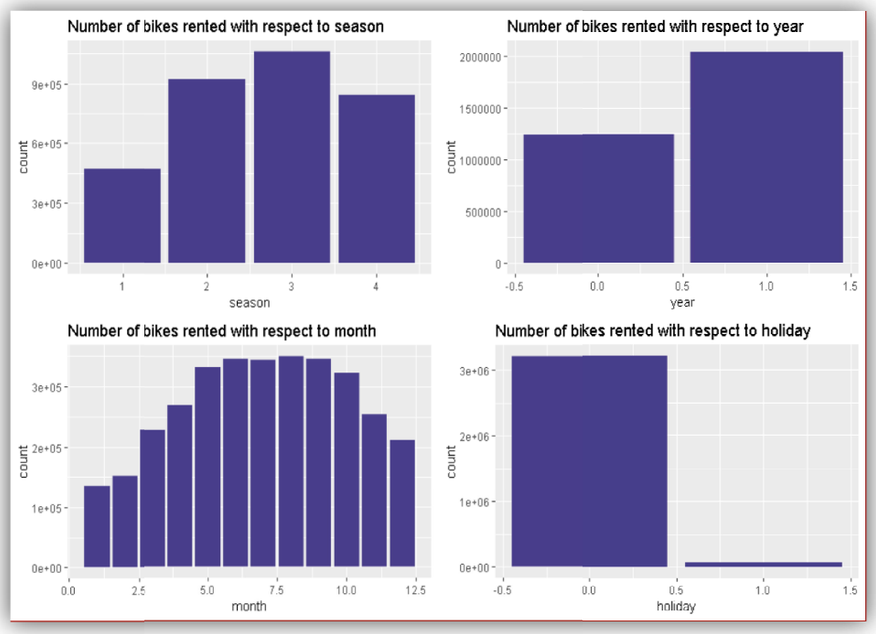
Count Vs Temperature: From the below plot we can say as temperature increase Bike rental count also increases.
Count Vs Humidity: From the below plot we can say humidity doesnt have any effect on bike rent count
Count Vs Windspeed: From the below plot we can say windspeed doesnt have any effect on bike rent count

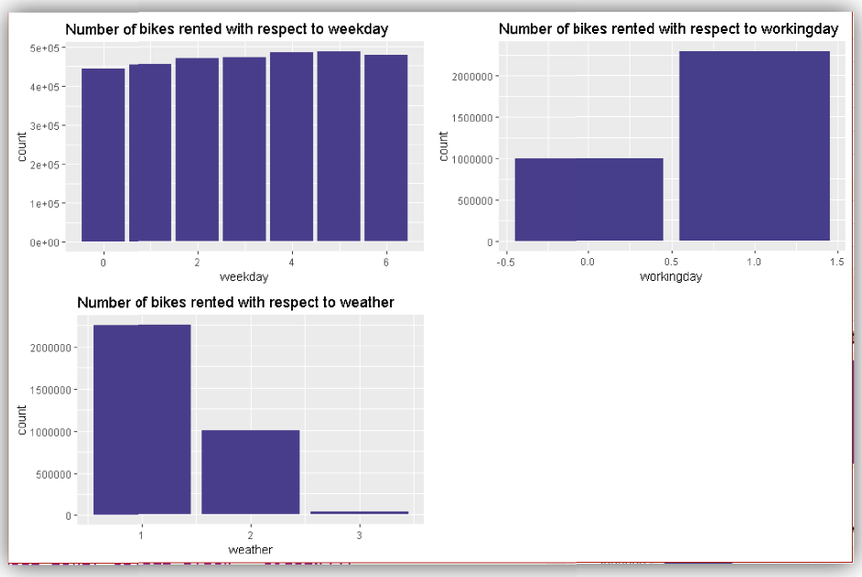


2.1.5.3 Effect of categorical variables on target variable

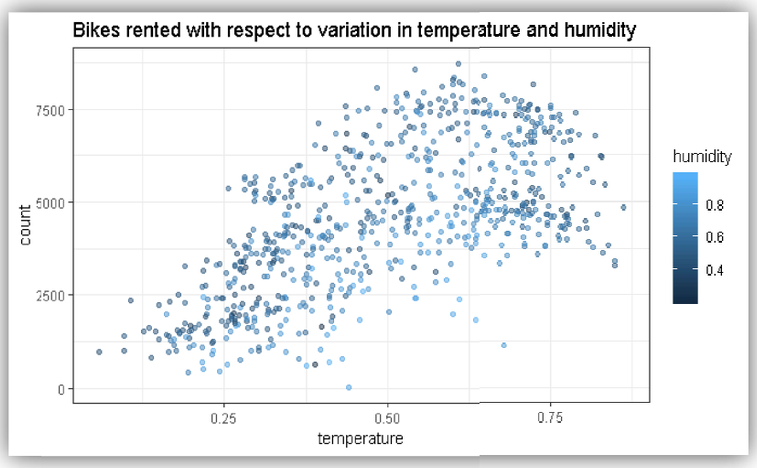
To check distribution of each categorical variable with respect to target variable we used barplot and we can also look at the frequency table

Count Vs season:- Bike rent count is high in season 3(fall) and low in season 1(springer)
Count Vs year :- Bike rent count is high in year 1 (in 2012)
Count Vs month :- Bike rent count is high in month of august and low in jan
Count Vs holiday:- Bike rent count is high on holidays ie 0
Count Vs weekday : - From bar plot we can see maximum bikes rented on 5th day and least bikes on day 0.
Count Vs workingday : - Bike rent count is high on working day ie 1
Count Vs weather : - Bike rent count is high on weather 1: ie when the weather is Clear, Few clouds, Partly cloudy, Partly cloudy

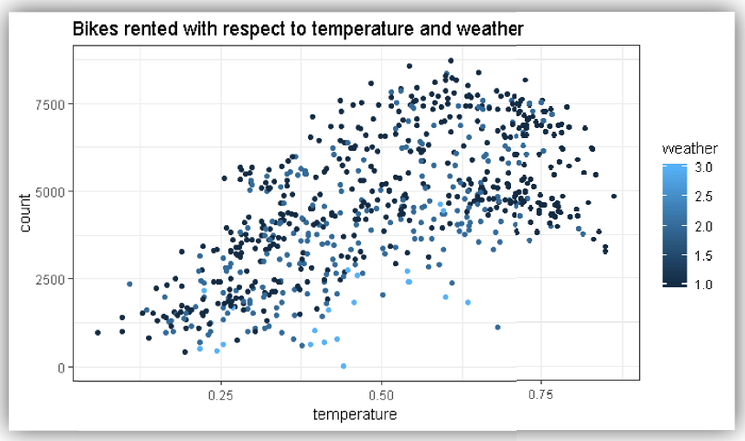

Number of bikes rented with respect to season
count
season
Number of bikes rented with respect to year
count
year
Number of bikes rented with respect to month
count
month
Number of bikes rented with respect to holiday
count
holiday


Number of bikes rented with respect to weekday
count
weekday
Number of bikes rented with respect to workingday
count
workingday
Number of bikes rented with respect to weather
count
weather

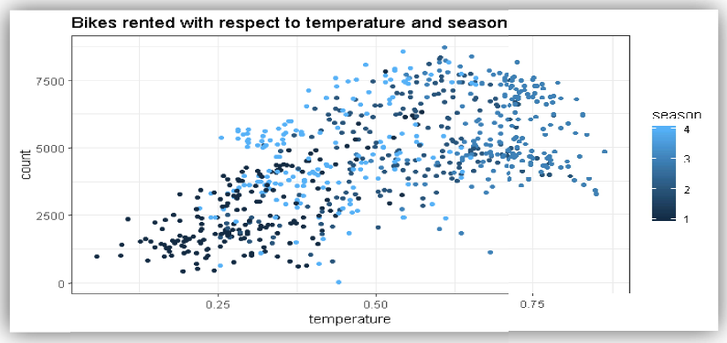
Bikes rented with respect to temp and humidity :- maximum bike rented between temp 0.50 to 0.75 and humidity 0.50 to 0.75



Bikes rented with respect to temp and windspeed:- 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 :-From figure it is clear that maximum bike-count is for season 2 and 3, when the temp between 0.5 to 0.7



2.1.5 Feature Selection

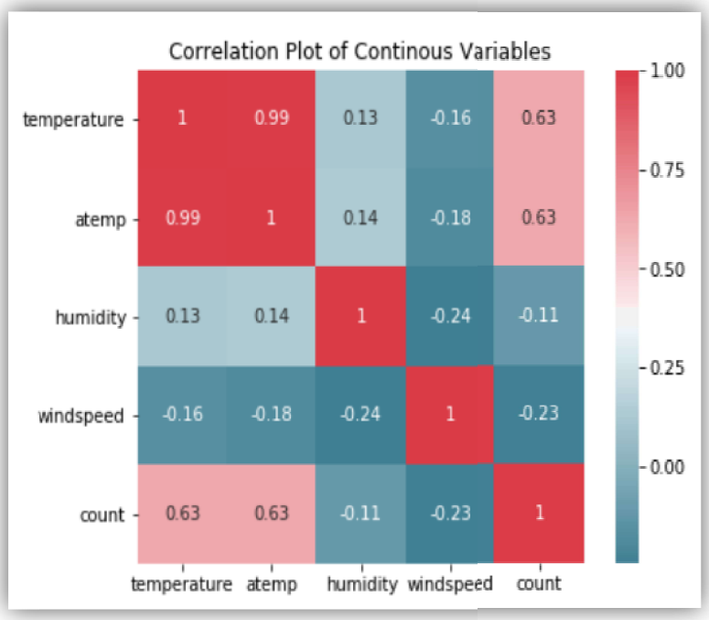
Before performing any type of modeling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of prediction. Selecting subset of relevant columns for the model construction is known as Feature Selection. We cannot use all the features because some features may be carrying the same information or irrelevant information which can increase overhead. To reduce overhead we adopt feature selection technique to extract meaningful features out of data. This in turn helps us to avoid the problem of multi collinearity. In this project we have selected

Correlation Analysis for continous variable and **ANOVA** (Analysis of variance) for categorical variables.

Correlation Matrix

```
> Correlation_matrix
             temperature     atemp   humidity  windspeed      count
temperature   1.0000000 0.9917016  0.1267216 -0.1569155  0.6274940
atemp         0.9917016 1.0000000  0.1399240 -0.1829480  0.6310657
humidity      0.1267216 0.1399240  1.0000000 -0.2411599 -0.1056645
windspeed    -0.1569155 -0.1829480 -0.2411599  1.0000000 -0.2336573
count         0.6274940 0.6310657 -0.1056645 -0.2336573  1.0000000
```

Correlation plot for all continuous variables



Results of ANOVA Test

```
[1] "season"
                Df    Sum Sq   Mean Sq F value Pr(>F)
Bike_Rent[, i]   1 4.518e+08 451797359     144 <2e-16 ***
Residuals      729 2.288e+09   3138187
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "year"
                Df    Sum Sq   Mean Sq F value Pr(>F)
Bike_Rent[, i]   1 8.798e+08 879828893   344.9 <2e-16 ***
Residuals      729 1.860e+09   2551038
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "month"
                Df    Sum Sq   Mean Sq F value   Pr(>F)
Bike_Rent[, i]   1 2.147e+08 214744463   62.01 1.24e-14 ***
Residuals      729 2.525e+09   3463362
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "holiday"
                Df    Sum Sq  Mean Sq F value Pr(>F)
Bike_Rent[, i]   1 1.280e+07 12797494   3.421 0.0648 .
Residuals      729 2.727e+09  3740381
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
[1] "weekday"
                Df    Sum Sq  Mean Sq F value Pr(>F)
Bike_Rent[, i]   1 1.246e+07 12461089   3.331 0.0684 .
Residuals      729 2.727e+09  3740843
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "workingday"
                Df    Sum Sq  Mean Sq F value Pr(>F)
Bike_Rent[, i]   1 1.025e+07 10246038   2.737 0.0985 .
Residuals      729 2.729e+09  3743881
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "weather"
                Df    Sum Sq   Mean Sq F value Pr(>F)
Bike_Rent[, i]   1 2.423e+08 242288753   70.73 <2e-16 ***
Residuals      729 2.497e+09   3425578
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

From correlation analysis we have found that **temperature** and **atemp** has high correlation (>0.9), so we remove the **atemp** variable ,in case of continous variables

and from ANOVA analysis we have found that in categorical variables **Holiday, weekday and working day** have the pr(>0.05), so we remove them in case of categorical variables.

After Correlation and ANOVA Analysis we have remaining variables are

Continuous variables in dataset

- temprature float64
- humidity float64
- windspeed float64
- count int64

Categorical variables in dataset

- season int64
- year int64
- month int64
- weather int64

Sample data after feature selection

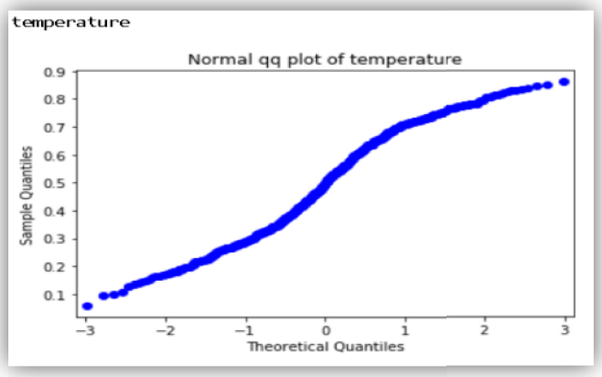
```
> head(Bike_Rent)
  season year month weather temperature humidity windspeed count
1      1    0     1       2    0.344167 0.805833 0.1604460   985
2      1    0     1       2    0.363478 0.696087 0.2485390   801
3      1    0     1       1    0.196364 0.437273 0.2483090  1349
4      1    0     1       1    0.200000 0.590435 0.1602960  1562
5      1    0     1       1    0.226957 0.436957 0.1869000  1600
6      1    0     1       1    0.204348 0.518261 0.0895652  1606
```

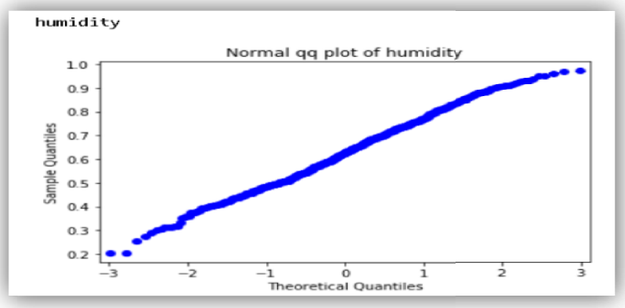
2.1.6 Feature Scaling

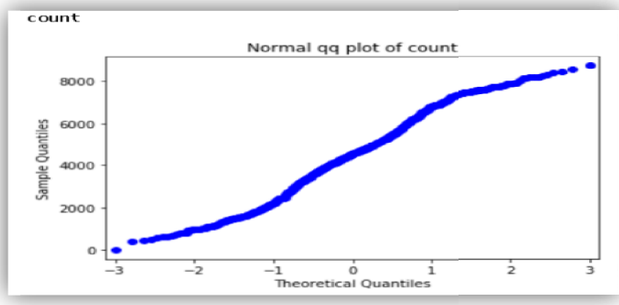
Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step. Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. Widely used feature scaling methods are min- max scaling and Standardization.

Since as in given dataset for continuous variables data is already Normalized, so we do not need to scale the data.We can check normality of data using normal qqplot ,histogram plot and summary of variables.

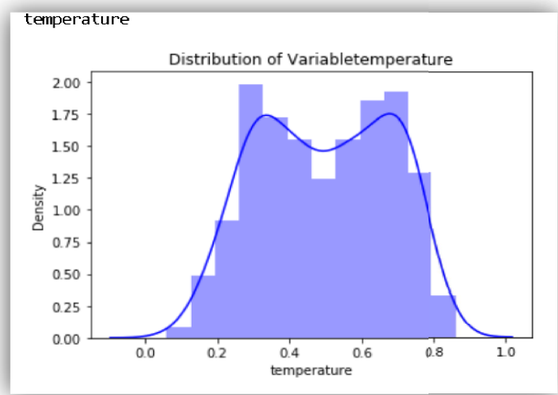
Normal QQ Plots :

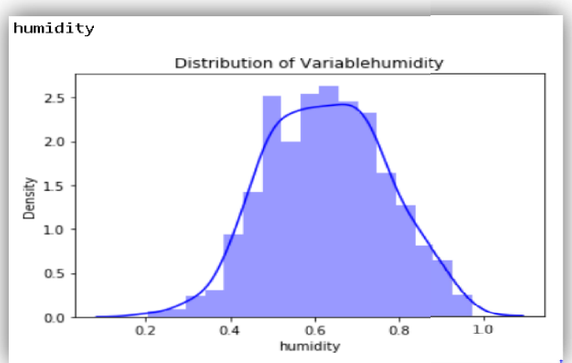


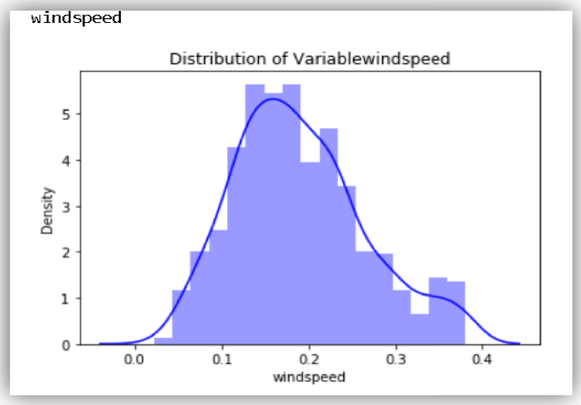


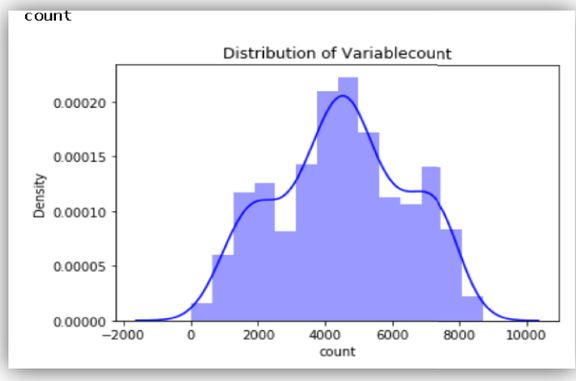


Histogram Plots :









Summary of each variables

	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	0.628197	0.189846	4504.348837
std	1.110807	0.500342	3.451913	0.544894	0.183051	0.141320	0.075644	1937.211452
min	1.000000	0.000000	1.000000	1.000000	0.059130	0.204687	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	0.380611	8714.000000

2.2 Predictive Modeling / Model Development

After Data pre-processing the next step is to develop a model using a train or historical data Which can perform to predict accurate result on test data or new data. Here we have tried with different model and will choose the model which will provide the most accurate values.

2.2.1 Multiple Linear Regression

Multiple linear regression is the most common form of linear regression analysis. Multiple linear regression is used to explain the relationship between one continuous dependent variable and two or more independent variables. The independent variables can be continuous or categorical. We trained/fitted linear regression model on train data in both R and Python

2.2.2 Decision Tree

Decision Tree is a supervised machine learning algorithm, which is used to predict the data for classification and regression. It accepts both continuous and categorical variables A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. 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

Random Forest is an ensemble technique that consists of many decision trees. The idea behind Random Forest is to build N number of trees to have more accuracy in dataset. It is called random forest as we are building n no. of trees randomly. In other words, to build the decision trees it selects randomly N no of variables and n no of observations. It means to build each decision tree on random forest we are not going to use the same data. 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 produce 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.

2.2.25 Hyper parameter 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. For example, if the learning rate is too low, the model will miss the important patterns in the data. If it is high, it may have collisions.

We used two techniques of Hyperparameter for our models

- Random Search
- Grid Search

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.

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. 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 the 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 of parameters which give the top accuracy is considered to be the best.

We used these two techniques for Decision tree,Random Forest ,Gredient boosting models

3.Conclusion .....

In methodology we have done data cleaning and then applied different-different machine learning algorithms on the data set to check the performance of each model, now in conclusion we will finalize the model of Bike Rental Count.

3.1 Model Evaluation

3.1.1 RMSE,R-squared, MAPE

In the previous chapter we have applied four algorithms on our dataset and calculated RMSE ,Rsquare and the Mean absolute percentage error Value for all the models.

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. In other words, it tells you how concentrated the data is around the line of best fit. Whereas R-squared is a relative measure of fit, RMSE is an absolute measure of fit. 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.

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.

We also said about MAPE value for all the three models which is a measure of prediction accuracy of a forecasting method. It measures accuracy in terms of percentage lower values of MAPE indicate better fit

Final Result in R:-

	Model	MAPE_Train	MAPE_Test	Rsquare_Train	Rsquare_Test	Rmse_Train	Rmse_Test
1	Linear Regression	46.48080	19.30459	0.8389979	0.8371418	778.1350	778.9873
2	Decision Tree for Regression	55.91186	28.09475	0.8119475	0.7269897	840.9667	1010.7788
3	Random Search in Decision Tree	55.91186	28.09475	0.8119475	0.7269897	840.9667	1010.7788
4	Gird Search in Decision Tree	55.91186	28.09475	0.8119475	0.7269897	840.9667	1010.7788
5	Random Forest	28.91363	19.79689	0.9648079	0.8673564	373.2392	701.9936
6	Random Search in Random Forest	25.06342	19.57239	0.9694855	0.8714193	348.5147	691.2888
7	Grid Search in Random Forest	25.28408	19.44296	0.9711066	0.8693046	338.7313	696.6711
8	Gradient Boosting	34.80372	16.74061	0.9032066	0.8778029	604.6178	672.5727
9	Random Search in Gradient Boosting	27.16694	19.65012	0.9661663	0.8711714	338.7313	696.6711
10	Grid Search in Gradient Boosting	26.19758	19.54139	0.9671025	0.8708002	362.9703	693.9289

Final Result in python :-

Final_Results

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Linear Regression	43.773719	19.685303	0.837362	0.839261	767.301965	826.828014
1	Decision Tree	62.260133	36.948093	0.677563	0.646470	1080.381858	1226.219619
2	Decision Tree Random Search CV	14.033006	22.953146	0.875127	0.811933	672.339115	894.356646
3	Decision Tree Grid Search CV	14.033006	22.953146	0.875127	0.811933	672.339115	894.356646
4	Random Forest	15.828286	20.471133	0.981456	0.883159	259.091491	704.940016
5	Random Forest Random Search CV	21.529458	20.146195	0.966812	0.883646	346.615114	703.471104
6	Random Forest Grid Search CV	17.211393	20.531874	0.974122	0.881434	306.067200	710.124689
7	Gradient Boosting	17.211393	18.858040	0.949290	0.867707	428.450622	750.106146
8	Gradient Boosting Random Search CV	43.199311	21.837795	0.851629	0.831375	767.301965	826.828014
9	Gradient Boosting Grid Search CV	14.959465	25.317824	0.960632	0.838236	377.506431	829.460512

3.1 Model Selection

From the observation of all **MAPE** , **R-Squared** and **RMSE** Value we have concluded that **Random Forest** has minimum value of MAPE (**20.42%)** and it's **R-Squared** Value is also maximum (**0.88**),with RMSE **(699.83).**Means, By Random forest algorithm predictor are able to explain 88% to the target variable on the test data. The MAPE value of Test data and Train does not differs a lot this implies that it is not the case of overfitting.

Insights about the Project :

1) Temperature variable has major impact on bike rental count
2) In season 3 (fall sept,oct,novem)there will be more bike rental counts
3) When the weather is 1 means when weather is Clear, Few clouds, Partly cloudy then there will be more bike rental counts
4) In the month of August bike rental count will be very high
5) Windspeed and humidity don't have any impact on bike rental count and these two variables negative correlated with all variables
6) On Friday or fifthday of week bike rental count is more
7) Even during working days bike rental count is more

4. Appendix A – R And Python Codes……

R Code :-

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

# Clean the environment ------------------------------------------
rm(list=ls())

# Set working directory ------------------------------------------
setwd("E:/EDWISOR/Project_BikeRent")

# Cross Check current working directory -----|--------------------
getwd()


# Load the required libraries for analysis of data----------------
x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unb
      "dummies", "e1071", "Information", "MASS", "rpart", "gbm", "
      'sampling', 'DataCombine', 'inTrees',"scales","psych","gplot
#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

# Check names of dataset,in names we can see shortcuts like, hum f
#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
```

```
# casual and registered variable as count is sum of these two vari
# cnt = casual + registered

Bike_Rent = subset(Bike_Rent,select=-c(instant,dteday,casual,regis

# 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","working


# 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))
```

```
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
           theme(legend.position = "bottom")+
           labs(y = cnames[i],x='count')+
           ggtitle(paste("boxplot of count for",cnames[i])))
}
```

```r
# Loop to remove outliers by capping upperfence and lower fence va
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
```

```r
# 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
           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
```

```
# Univariate Analysis ----------------------------------------
# Histogram for continuous variables to check  distribution of eac
for(i in 1:length(cnames))
{
  assign(paste0("h",i),ggplot(aes_string(x=(cnames[i])),
                              data=subset(Bike_Rent))+
           geom_histogram(fill="darkslateblue",colour = "black")+g
           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[
           ggtitle(paste("Scatter Plot of count vs",cnames[i])))
}
```

```
# count vs windspeed : windspeed doesnt have any effect on bikeren
# count vs count : please ignore this plot as it is our target var

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
                              data=subset(Bike_Rent))+
           geom_bar(stat = "identity",fill = "DarkSlateBlue") +
           # labs(title = "Scatter Plot of count vs", x = (cnames[
           ggtitle(paste("Number of bikes rented with respect to",
           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 seas
aggregate(count ~ season ,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 da
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 weat
# 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 temperat
# maximum bike rented between temp 0.50 to 0.75 and humidity 0.50

#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 weatl
```

```
# 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 seas
   # 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
# 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 va
# Let us derive our correlation matrix
Correlation_matrix = cor(Bike_Rent[,cnames])
Correlation_matrix
# By looking at correlation matrix we can say temperature and atem
# are highly correlated (>0.99)

#Lets plot correlation plot using corrgram library
```

```r
# Lets find significant categorical variables usig ANOVA test
# Anova analysis for categorical variable with target numeric var
for(i in cat_cnames){
  print(i)
  Anova_result= summary(aov(formula = count~ Bike_Rent[,i],Bike_R
  print(Anova_result)
}

# From the anova result, we can observe working day,weekday and h
# has p value > 0.05, so delete this variable not consider in mod

# Dimension reduction  ----------------------------------------
Bike_Rent = subset(Bike_Rent,select = -c(atemp,holiday,weekday,wo


# 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 d
# Continuous variable
```

```r
# Feature Scaling  ----------------------------------------------
# Since as it is mentioned in data dictionary the values of
# temp,humidity,windspeed variables are already normalized values
# need to go for feature scaling instead we will visualize the va
# 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

```

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

# Lets call Categorical varaibles after feature selection using AI
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 =
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 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 a
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
```

```
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
```

```
########################################################

# 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)#55.91

# MAPE For train data test data
DT_MAPE_Test = mape(test[,25],DT_test)#28.09

# Rsquare  For train data
DT_r2_train= rsquare(train[,25],DT_train)#0.81
```

```
476
477 # rmse For test data
478 DT_rmse_test = rmse(test[,25],DT_test)
479
480
481 # Random Search CV In Decision Tree ----------------------------
482 # Lets set parameters to pass into our decision tree model
483 # Lets use caret package for the same
484 control = trainControl(method="repeatedcv", number=5, repeats=1,se
485 maxdepth = c(1:30)
486 tunegrid = expand.grid(.maxdepth=maxdepth)
487
488 # Lets build a model using above parameters on train data
489 RDT_model = caret::train(count~., data=train, method="rpart2",trC
490 print(RDT_model)
491
492 # Lets look into best fit parameters
493 best_fit_parameters = RDT_model$bestTune
494 print(best_fit_parameters)
495
496 # Again rebuild decision tree model using randomsearch best fit pa
497 # with maximum depth = 9
498 RDT_best_model = rpart(count~.,train,method = "anova",maxdepth=9)
499
500 # Prediction on train data
501 RDT_train = predict(RDT_best_model,train[-25])
```

```
508 # MAPE for train data
509 RDT_MAPE_Train =mape(train[,25],RDT_train)
510
511 # MAPE for test data
512 RDT_MAPE_Test =mape(test[,25],RDT_test)
513
514 # Rsquare for train data
515 RDT_r2_train =  rsquare(train[,25],RDT_train)
516
517 # Rsquare for test data
518 RDT_r2_test = rsquare(test[,25],RDT_test)
519
520 # rmse For train data
521 RDT_rmse_train = rmse(train[,25],RDT_train)
522
523 # rmse For test data
524 RDT_rmse_test = rmse(test[,25],RDT_test)
525
526
527 # Grid Search CV in Decision Tree ------------------------------
528 control =trainControl(method="repeatedcv", number=5, repeats=2, s
529 tunegrid = expand.grid(.maxdepth=c(6:20))
530
531 # Lets build a model using above parameters on train data
532 GDT_model = caret::train(count~.,train, method="rpart2", tuneGrid
533 print(GDT_model)
```

```
# with maximum depth = 9
GDT_best_model = rpart(count ~ .,train, method = "anova", maxdept
# 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)
```

```
# 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  ----------------------------
```

```
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", r

# 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)
```

```
# 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, s
tunegrid = expand.grid(.mtry=c(6:20))

# Lets build a model using above parameters on train data using ra
GRF_model= caret::train(count~.,train, method="rf", tuneGrid=tune
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", r

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

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

```
# 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.

# 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_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,s
#maxdepth = c(1:30)
#tunegrid = expand.grid(.maxdepth=maxdepth)

# Model devlopment on train dat
RGB_model = caret::train(count~., data=train, method="gbm",trCont

print(RGB_model)

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

```
# 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)


# Grid Search CV in Gradient Boosting ----------------------------
control = trainControl(method="repeatedcv", number=5, repeats=2, s
tunegrid = expand.grid(n.trees = seq(2565,2575, by = 2),
                       interaction.depth = c(2:4),
```

```

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

# Build model based on best fit
GGB_best_model = randomForest(count ~ .,train, method = "anova", r
                          interaction.depth = 4,shrinkage = 0.01,n.

# Prediction on train data
GGB_train= predict(GGB_best_model,train[-25])

# Prediction on test data
GGB_test= predict(GGB_best_model,test[-25])

# Mape calculation of train data
GGB_MAPE_Train = mape(train[,25],GGB_train)

# Mape for test data
GGB_MAPE_Test = mape(test[,25],GGB_test)

# Rsquare for train data
GGB_r2_train= rsquare(train[,25],GGB_train)

# Rsquare for test data
GGB_r2_test=rsquare(test[,25],GGB_test)
```

```
# Results ------------------------------------------------------

Model = c('Linear Regression','Decision Tree for Regression',
        'Random Search in Decision Tree','Gird Search in Decision
        'Random Forest','Random Search in Random Forest',
        'Grid Search in Random Forest','Gradient Boosting',
        'Random Search in Gradient Boosting',
        'Grid 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_MA
               GB_MAPE_Train,RGB_MAPE_Train,GGB_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_
               GB_MAPE_Test,RGB_MAPE_Test,GGB_MAPE_Test)

Rsquare_Train = c(LR_r2_train,DT_r2_train,RDT_r2_train,GDT_r2_tra
                  RF_r2_train,RRF_r2_train,GRF_r2_train,GB_r2_trai
                  RGB_r2_train,GGB_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,GGB_r2_test)

Rmse_Train = c(LR_rmse_train,DT_rmse_train,RDT_rmse_train,GDT_rms

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

Final_results

# From above results Random Forest model have optimum values and t
```

Python Code :-

```
In [1]: # Bike Rental Prediction ------------------------------------------------

In [2]: # 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
        #from fancyimpute import KNN
        from random import randrange,uniform
        from ggplot import *
        from sklearn.metrics import r2_score
        from scipy import stats

In [3]: # Set working directory ------------------------------------------------
        os.chdir("E:\EDWISOR\Project_BikeRent")

        # Cross Check current working directory ----------------------------------
        os.getcwd()

Out[3]: 'E:\\EDWISOR\\Project_BikeRent'

In [4]: # Load the data ----------------------------------------------------------
        Bike_Rent = pd.read_csv("day.csv")
```

In [6]:
```
# Check names of dataset
Bike_Rent.columns

# Variables names in shortcuts like, hum for humidity  yr for year ,mnth for month,cnt for count

# Rename variables in dataset
Bike_Rent = Bike_Rent.rename(columns = {'yr':'year','mnth':'month','weathersit':'weather',
                                        'temp':'temperature','hum':'humidity','cnt':'count'})

Bike_Rent.columns
```

Out[6]:
```
Index(['instant', 'dteday', 'season', 'year', 'month', 'holiday', 'weekday',
       'workingday', 'weather', 'temperature', 'atemp', 'humidity',
       'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

In [7]:
```
# Check top(first) rows of dataset
Bike_Rent.head()
```

Out[7]:

	instant	dteday	season	year	month	holiday	weekday	workingday	weather	temperature	atemp	humidity	win
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.

In [8]:
```
# Check bottom(last) rows of dataset
Bike_Rent.tail()
```

Out[8]:

	instant	dteday	season	year	month	holiday	weekday	workingday	weather	temperature	atemp	humidity	win
726	727	2012-12-27	1	1	12	0	4	1	2	0.254167	0.226642	0.652917	0.
727	728	2012-12-28	1	1	12	0	5	1	2	0.253333	0.255046	0.590000	0.
728	729	2012-12-29	1	1	12	0	6	0	2	0.253333	0.242400	0.752917	0.
729	730	2012-12-30	1	1	12	0	0	0	1	0.255833	0.231700	0.483333	0.
730	731	2012-12-31	1	1	12	0	1	1	2	0.215833	0.223487	0.577500	0.

In [9]:
```
# Check structure of dataset(data structure of each variable)
Bike_Rent.dtypes
```

Out[9]:
```
instant          int64
dteday          object
season           int64
year             int64
month            int64
holiday          int64
weekday          int64
workingday       int64
weather          int64
temperature    float64
```

In [10]:
```
# Check summary of dataset
Bike_Rent.describe()
```

Out[10]:

	instant	season	year	month	holiday	weekday	workingday	weather	temperature	
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	73
mean	366.000000	2.496580	0.500684	6.519836	0.028728	2.997264	0.683995	1.395349	0.495385	
std	211.165812	1.110807	0.500342	3.451913	0.167155	2.004787	0.465233	0.544894	0.183051	
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.059130	
25%	183.500000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000	0.337083	
50%	366.000000	3.000000	1.000000	7.000000	0.000000	3.000000	1.000000	1.000000	0.498333	
75%	548.500000	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	2.000000	0.655417	
max	731.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000	0.861667	

In [11]:
```
# Variable Identification
Bike_Rent['count'].dtypes
```

Out[11]:
```
dtype('int64')
```

In [12]:
```
# 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
```

In [13]:
```
# Make Seperate categorical and numerical variables dataframe

# Continous Variables
cnames= ['temperature', 'atemp', 'humidity', 'windspeed', 'count']

# Categorical variables-
cat_cnames=['season', 'year', 'month', 'holiday', 'weekday', 'workingday','weather']
```

In [14]:
```
# EDA or Data Preprocessing  ----------------------------------------------
```

In [15]:
```
# Missing Value anlysis ---------------------------------------------------
# Check missing values in dataset
Bike_Rent.isnull().sum()

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

Out[15]:
```
season         0
year           0
month          0
holiday        0
weekday        0
workingday     0
weather        0
temperature    0
atemp          0
humidity       0
windspeed      0
count          0
```

In [16]:
```
# Outlier Analysis and treatment ------------------------------------
# Lets save copy of dataset before preprocessing
df = Bike_Rent.copy()
Bike_Rent = df.copy()

# Lets use boxplot to detect and visulaze the outliers using sns librray

for i in cnames:
    print(i)
    sns.boxplot(y=Bike_Rent[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
```

In [17]:
```
# Lets cap outliers and inliers with upper fence and lower fence values
for i in cnames:
    print(i)
    # Quartiles and IQR
    q25,q75 = np.percentile(Bike_Rent[i],[25,75])
    IQR = q75-q25

    # Lower and upper limits
    LL = q25 - (1.5 * IQR)
    UL = q75 + (1.5 * IQR)

    # Capping with ul for maxmimum values
    # For inliers
    Bike_Rent.loc[Bike_Rent[i] < LL ,i] = LL

   # For ioutliers
```

In [18]:
```
# Lets see our boxplots after removing outliers

for i in cnames:
    print(i)
    sns.boxplot(y=Bike_Rent[i])
    plt.xlabel(i)
    plt.ylabel("values")
    plt.title("Boxplot of "+i)
    plt.show()
```

In [19]:
```
# Visualization ----------------------------------------------------------
```

In [20]:
```
# Lets look at numeric and categorical variables
# Continous Variables
cnames= ['temperature', 'atemp', 'humidity', 'windspeed', 'count']

# Categorical variables-
cat_cnames=['season', 'year', 'month', 'holiday', 'weekday', 'workingday','weather']
```

In [21]:
```
# Univariate Analysis ----------------------------------------------------

# Histogram for all continuous variables to check  distribution of each variable

# temperature
```

In [22]:
```
# humidity
ggplot(Bike_Rent,aes(x='humidity')) + geom_histogram(fill="darkslateblue",colour = "black")+\
   geom_density()+\
   theme_bw() + xlab("humidity") + ylab("Frequency") +ggtitle("distrubution of humidity")+\
   theme(text=element_text(size=20))
# normally distributed
```

In [23]:
```
# windspeed
ggplot(Bike_Rent,aes(x='windspeed')) + geom_histogram(fill="darkslateblue",colour = "black")+\
   geom_density()+\
   theme_bw() + xlab("windspeed") + ylab("Frequency") +ggtitle("distrubution of windspeed")+\
   theme(text=element_text(size=20))
# normally distributed
```

In [24]:
```
# count
ggplot(Bike_Rent,aes(x='count')) + geom_histogram(fill="darkslateblue",colour = "black")+\
   geom_density()+\
   theme_bw() + xlab("count") + ylab("Frequency") +ggtitle("distrubution of count")+\
   theme(text=element_text(size=20))
# normally distributed
```

In [25]:
```
# Bivariate Analysis ----------------------------------------------------
# Lets check impact of continous variables on target variable

# count vs temperature
ggplot(aes(x="temperature",y="count"),data=Bike_Rent)+\
  geom_point(alpha=1,size=50,color="DarkSlateBlue")+theme_bw()+ylab("count")+xlab("temperature")
# as temperature increase Bike rent count also increases
```

In [27]:
```
# count vs windspeed

ggplot(aes(x="windspeed",y="count"),data=Bike_Rent)+\
  geom_point(alpha=1,size=50,color="DarkSlateBlue")+theme_bw()+ylab("count")+xlab("windspeed")+g

# windspeed doesnt have any effect on bikerent count
```

In [28]:
```
# Let us check impact of categorical variables on count
# Season
print(Bike_Rent.groupby(['season'])['count'].sum())


ggplot(Bike_Rent,aes(x='season',y='count'))+\
    geom_bar(fill="DarkSlateBlue")+\
    scale_color_brewer(type = 'diverging',palette=4)+\
    xlab("season")+ylab("count") + ggtitle("Bike Rent Count vs season")+ theme_bw()

# Bike rent count is high in season 3(fall) and low in season 1(springer)
```

In [29]:
```
# Year
print(Bike_Rent.groupby(['year'])['count'].sum())
```

In [30]:
```
# month
print(Bike_Rent.groupby(['month'])['count'].sum())


ggplot(Bike_Rent,aes(x='month',y='count'))+\
    geom_bar(fill="DarkSlateBlue",stat='sum')+\
    scale_color_brewer(type = 'diverging',palette=4)+\
    xlab("month")+ylab("count") + ggtitle("Bike Rent Count vs month")+ theme_bw()

# Bike rent count is high in month of august and low in jan
```

In [31]:
```
# holiday
print(Bike_Rent.groupby(['holiday'])['count'].sum())


ggplot(Bike_Rent,aes(x='holiday',y='count'))+\
    geom_bar(fill="DarkSlateBlue")+\
    scale_color_brewer(type = 'diverging',palette=4)+\
    xlab("holiday")+ylab("count") + ggtitle("Bike Rent count vs holiday")+ theme_bw()

#  Bike rent count is high in holiday ie 0
```

In [32]:
```
# weekday
print(Bike_Rent.groupby(['weekday'])['count'].sum())
```

In [33]:
```
# workingday
print(Bike_Rent.groupby(['workingday'])['count'].sum())


ggplot(Bike_Rent,aes(x='workingday',y='count'))+\
    geom_bar(fill="DarkSlateBlue")+\
    scale_color_brewer(type = 'diverging',palette=4)+\
    xlab("workingday")+ylab("count") + ggtitle("Bike Rent count vs workingday")+ theme_bw()

# Bike rent count is high on working day  ie 1
```

In [34]:
```
# weather
print(Bike_Rent.groupby(['weather'])['count'].sum())


ggplot(Bike_Rent,aes(x='weather',y='count'))+\
    geom_bar(fill="DarkSlateBlue")+\
    scale_color_brewer(type = 'diverging',palette=4)+\
    xlab("weather")+ylab("count") + ggtitle("Bike Rent count vs weather")+ theme_bw()

# Bike rent count is high on weather 1: ie when the weather is
# Clear, Few clouds, Partly cloudy, Partly cloudy
```

In [35]:
```
# Bikes rented with respect to temp and humidity--------------
f, ax = plt.subplots(figsize=(10, 10))
sns.scatterplot(x="temperature", y="count",
                hue="humidity", size="count",
                palette="coolwarm", sizes=(1, 100), linewidth=0
```

In [36]:
```
#Bikes rented with respect to temp and windspeed------------------
f, ax = plt.subplots(figsize=(8, 8))
sns.scatterplot(x="temperature", y="count",
                hue="windspeed", size="humidity",
                palette="coolwarm",sizes=(1, 100), linewidth=0,
                data=Bike_Rent,ax=ax)
plt.title("Varation in bike rental count with respect to  temperature and windspeed")
plt.ylabel("Bike rental count")
plt.xlabel("temperature")
plt.savefig('bike_temp & windspeed_plot.png')

#From the above plot we can see bike count is maximum between temp 0.5 to 0.7, windspped below 0
```

In [37]:
```
# Bikes rented with respect to temp and season--------------------
f, ax = plt.subplots(figsize=(8, 8))
sns.scatterplot(x="temperature", y="count",
                hue="season", size="count",style= "weather",
                palette="coolwarm",sizes=(1, 100), linewidth=0,
                data=Bike_Rent,ax=ax)
plt.title("Varation in bike rental count with respect to stemperature and season")
plt.ylabel("Bike rental count")
plt.xlabel("Normalized temperature")
plt.savefig('bike_temp&season_plot.pdf')

# From figure it is clear that maximum bike count is for season 2 and 3, when the temp between 0
```

In [40]:
```
# Correlation analysis
# Using corrplot library we do correlation analysis for numeric variables
# Lets recall numeric variabls and derive correlation matrix and plot

# Continous Variables
cnames= ['temperature', 'atemp', 'humidity', 'windspeed', 'count']

# Correlation matrix
# Extract only numeric variables in dataframe for correlation
df_corr= Bike_Rent.loc[:,cnames]

# Generate correlation matrix
corr_matrix = df_corr.corr()
(print(corr_matrix))

# Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(7, 5))

#Plot using seaborn library
sns.heatmap(corr_matrix, mask=np.zeros_like(corr_matrix, dtype=np.bool), cmap=sns.diverging_pale
            square=True, ax=ax,annot=True)

plt.title("Correlation Plot of Continous Variables")

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

In [41]:
```
# Categorical variables-
cat_cnames=['season', 'year', 'month', 'holiday', 'weekday', 'workingday','weather']
```

In [42]:
```
# Lets find significant categorical variables usig ANOVA test
```

In [43]:
```
# From the anova result, we can observe working day,weekday and holiday
# has p value > 0.05, so delete this variable not consider in model.
```

In [44]:
```
# Dimension reduction  ------------------------------------------------
Bike_Rent = Bike_Rent.drop(['atemp', 'holiday','weekday','workingday'], axis=1)
```

In [45]:
```
# Lets check dimensions after dimension reduction
Bike_Rent.shape
```

Out[45]:
```
(731, 8)
```

In [46]:
```
# Lets check column names after dimension reduction
Bike_Rent.columns
```

Out[46]:
```
Index(['season', 'year', 'month', 'weather', 'temperature', 'humidity',
       'windspeed', 'count'],
      dtype='object')
```

In [47]:
```
# Lets define/update  continous and categorical variables after dimension reduction

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

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

In [48]:
```
# 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
```

In [50]:
```
for i in cnames:
    print(i)
    sns.distplot(Bike_Rent[i],bins='auto',color='blue')
    plt.title("Distribution of Variable"+i)
    plt.ylabel("Density")
    plt.show()
```

In [51]:
```
Bike_Rent.describe()

# from distribution plot,normal qq plot  and summary  it is clear that data is already normalize
```

In [52]:
```
# Model Development ----------------------------------------------------------
```

In [53]:
```
# Load Required libraries for model development
from sklearn.model_selection import train_test_split #used to split dataset into train and test
from sklearn.metrics import mean_squared_error # used to calculate MSE
from sklearn.metrics import r2_score # used to calculate r square
from sklearn.linear_model import LinearRegression # For linear regression
from sklearn.tree import DecisionTreeRegressor # For Decision Tree
from sklearn.ensemble import RandomForestRegressor # For RandomForest
from sklearn import metrics
```

In [54]:
```
# 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
```

In [55]:
```
Bike_Rent.shape
```

Out[55]: (731, 25)

In [56]:
```
Bike_Rent.head()
```

Out[56]:

	temperature	humidity	windspeed	count	season_1	season_2	season_3	season_4	year_0	year_1	...	month_6	mo
0	0.344167	0.805833	0.160446	985.0	1	0	0	0	1	0	...	0	
1	0.363478	0.696087	0.248539	801.0	1	0	0	0	1	0	...	0	
2	0.196364	0.437273	0.248309	1349.0	1	0	0	0	1	0	...	0	
3	0.200000	0.590435	0.160296	1562.0	1	0	0	0	1	0	...	0	
4	0.226957	0.436957	0.186900	1600.0	1	0	0	0	1	0	...	0	

5 rows × 25 columns

In [57]:
```
# Lets Divide the data into train and test set

# Split data for predictor and target seperatly
X= Bike_Rent.drop(['count'],axis=1)
y= Bike_Rent['count']
```

In [58]:
```
# Divide data into train and test sets
```

In [60]:
```
# Linear Regression model ------------------------------------------------
```

In [61]:
```
# Import libraries
import statsmodels.api as sm

# Linear Regression model for regression
LR_model= sm.OLS(y_train,X_train).fit()
```

In [62]:
```
# Model prediction on  on train data
LR_train= LR_model.predict(X_train)

# Model prediction on test data
LR_test= LR_model.predict(X_test)

# Model performance on train data
MAPE_train= MAPE(y_train,LR_train)

# Model performance on test data
MAPE_test= MAPE(y_test,LR_test)

# r2 value for train data
r2_train= r2_score(y_train,LR_train)

# r2 value for test data-
r2_test=r2_score(y_test,LR_test)

# RMSE value for train data
RMSE_train = np.sqrt(metrics.mean_squared_error(y_train,LR_train))

# RMSE value for test data
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test,LR_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))
```

In [65]:
```
# Lets build some more models using different ml algorithms for more accuracy
# and less prediction error
```

In [66]:
```
# Desicision Tree ------------------------------------------------------
# Lets Build decision tree model on train data
# Import libraries
from sklearn.tree import DecisionTreeRegressor

# Decision tree for regression
DT_model= DecisionTreeRegressor(max_depth=2).fit(X_train,y_train)

# Model prediction on train data
DT_train= DT_model.predict(X_train)

# Model prediction on test data
DT_test= DT_model.predict(X_test)

# Model performance on train data
MAPE_train= MAPE(y_train,DT_train)

# Model performance on test data
MAPE_test= MAPE(y_test,DT_test)

# r2 value for train data
r2_train= r2_score(y_train,DT_train)

# r2 value for test data
r2_test=r2_score(y_test,DT_test)

# RMSE value for train data
RMSE_train = np.sqrt(metrics.mean_squared_error(y_train,DT_train))

# RMSE value for test data
```

In [68]:
```
DT_Results
```

In [69]:
```
# Random Search CV In Decision Tree ---------------------------------------
```

In [70]:
```
# 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
RDT_model= RandomizedSearchCV(RandomDecisionTree,param_distributions= random_search,n_iter=5,cv=
RDT_model= RDT_model.fit(X_train,y_train)
```

In [71]:
```
# Lets look into best fit parameters
best_parameters = RDT_model.best_params_
print(best_parameters)
```

```
{'max_depth': 7}
```

In [72]:
```
# Again rebuild decision tree model using randomsearch best fit parameter ie
# with maximum depth = 5
RDT best model = RDT model.best estimator
```

In [74]:
```
# Lets check Model performance on both test and train using error metrics of regression like map

# 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))
```

. . .

In [77]:
```
# Grid Search CV in Decision Tree ----------------------------------------
```

In [78]:
```
# 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
GDT_model= GridSearchCV(GridDecisionTree,param_grid=grid_search,cv=5)
GDT_model= GDT_model.fit(X_train,y_train)
```

In [79]:
```
# Lets look into best fit parameters from gridsearch cv DT
best_parameters = GDT_model.best_params_
print(best_parameters)
```

. . .

In [80]:
```
# Again rebuild decision tree model using gridsearch best fit parameter ie
# with maximum depth = 5
GDT_best_model = GDT_model.best_estimator_
```

In [82]:
```
# Lets check Model performance on both test and train using error metrics of regression like map

# 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))
```

In [85]:
```
# Random Forest ----------------------------------------
```

In [86]:
```
# 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))
```

In [87]:
```
# Lets print results of Randomforest random search
Error_metrics_RF= {'Model Name': ['Random Forest'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_t
      'R-squared_Test':[r2_test],'RMSE_train':[RMSE_train],'RMSE_test':[RMSE_test]}
RF_Results = pd.DataFrame(Error_metrics_RF)
```

In [88]:
```
RF_Results
```

In [89]:
```
# Random Search CV in Random Forest  ------------------------------------
```

In [90]:
```
# 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
RRF model= RandomizedSearchCV(RandomRandomForest,param distributions= random search,n iter=5,cv=
```

In [91]:
```
# Best parameters for model
best_parameters = RRF_model.best_params_
print(best_parameters)
```

In [92]:
```
# Again rebuild random forest  model using gridsearch best fit parameter ie {'n_estimators': 91,
RRF_best_model = RRF_model.best_estimator_
```

In [93]:
```
# Prediction on train data
RRF_train = RRF_best_model.predict(X_train)
```

In [94]:
```
# Lets check Model performance on both test and train using error metrics of regression like map

# 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))
```

In [97]:
```
# Grid search CV in Random Forest  ---------------------------------------
```

In [98]:
```
# 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}
```

In [99]:
```
# Lets build a model using above parameters on train data using random forest grid search cv
GRF_model= GridSearchCV(GridRandomForest,param_grid=grid_search,cv=5)
GRF_model= GRF_model.fit(X_train,y_train)
```

In [100]:
```
# Best fit parameters for model
best_parameters_GRF = GRF_model.best_params_
print(best_parameters_GRF)
```

In [102]:
```
# Prediction on train data
GRF_train = GRF_best_model.predict(X_train)

# Prediction on test data
GRF_test = GRF_best_model.predict(X_test)
```

In [103]:
```
# Lets check Model performance on both test and train using error metrics of regression like map

# 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))
```

In [105]:
```
GRF_results
```

In [106]:
```
# Gradient Boosting ------------------------------------------------------
```

In [107]:
```
# Import libraries
from sklearn.ensemble import GradientBoostingRegressor

# Lets build a Gradient Boosting model for regression problem
GB_model = GradientBoostingRegressor().fit(X_train, y_train)

# Model prediction on train data
GB_train= GB_model.predict(X_train)

# Model prediction on test data
GB_test= GB_model.predict(X_test)

# Model performance on train data
MAPE_test= MAPE(y_train,GB_train)

# Model performance on test data
MAPE_test= MAPE(y_test,GB_test)

# Rsquare value for train data
r2_train= r2_score(y_train,GB_train)

# Rsquare value for test data
r2_test=r2_score(y_test,GB_test)

# RMSE value for train data
RMSE_train = np.sqrt(metrics.mean_squared_error(y_train,GB_train))

# RMSE value for test data
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test,GB_test))
```

In [108]:
```
# Lets print the result
Error_metrics_GB = {'Model Name': ['Gradient Boosting'],'MAPE_Train':[MAPE_train],'MAPE_Test':[M
      'R-squared_Test':[r2_test],'RMSE_train':[RMSE_train],'RMSE_test':[RMSE_test]}
GB_results = pd.DataFrame(Error_metrics_GB)
```

In [109]:
```
GB_results
```

In [110]:
```
# Random Search CV in Gradient Boosting ----------------------------------
```

In [111]:
```
# Import libraries
from sklearn.model_selection import RandomizedSearchCV

RandomGradientBoosting = GradientBoostingRegressor(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}
```

In [112]:
```
# Lets build a model using above parameters on train data
RGB_model= RandomizedSearchCV(RandomGradientBoosting,param_distributions= random_search,n_iter=5
RGB_model= RGB_model.fit(X_train,y_train)
```

In [113]:
```
# Best parameters for model
best_parameters = RGB_model.best_params_
print(best_parameters)
```

```
{'n_estimators': 69, 'max_depth': 7}
```

In [114]: # Again rebuild random forest model using gridsearch best fit parameter {'n_estimators': 67, 'ma

In [116]:
```
# Lets check Model performance on both test and train using error metrics of regression like map

# MAPE for train data
MAPE_train= MAPE(y_train,RGB_train)

# MAPE for test data
MAPE_test= MAPE(y_test,RGB_test)

# Rsquare for train data
r2_train= r2_score(y_train,RGB_train)

# Rsquare for test data
r2_test=r2_score(y_test,RGB_test)

# RMSE value for train data
RMSE_train = np.sqrt(metrics.mean_squared_error(y_train,LR_train))

# RMSE value for test data
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test,LR_test))

print("Best Parameter="+str(best_parameters))
print("Best Model="+str(RGB_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))
```

In [119]:
```
# Grid Search CV in Gradient Boosting ------------------------------------
```

In [120]:
```
# Import libraries
from sklearn.model_selection import GridSearchCV

GridGradientBoosting= GradientBoostingRegressor(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}
```

In [121]:
```
# Lets build a model using above parameters on train data(Grind Random Forest)
GGB_model= GridSearchCV(GridGradientBoosting,param_grid=grid_search,cv=5)
GGB_model= GGB_model.fit(X_train,y_train)
```

In [122]:
```
# Best parameters for model
best_parameters = GGB_model.best_params_
print(best_parameters)
```

```
{'max_depth': 7, 'n_estimators': 19}
```

In [123]:
```
# Again rebuild random forest model using gridsearch best fit parameter {'n_estimators': 67, 'ma
GGB_best_model = GGB_model.best_estimator_
```

In [125]:
```
# Lets check Model performance on both test and train using error metrics of regression like map

# MAPE for train data
MAPE_train= MAPE(y_train,GGB_train)

# MAPE for test data
MAPE_test= MAPE(y_test,GGB_test)

# Rsquare value for train data
r2_train= r2_score(y_train,GGB_train)

# Rsquare value for test data
r2_test=r2_score(y_test,GGB_test)

# RMSE value for train data
RMSE_train = np.sqrt(metrics.mean_squared_error(y_train,GGB_train))

# RMSE value for test data
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test,GGB_test))

print("Best Parameter="+str(best_parameters))
print("Best Model="+str(GGB_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))
```

In [126]:
```
Error_metrics_GGB = {'Model Name': ['Gradient Boosting Grid Search CV'],'MAPE_Train':[MAPE_train
      'R-squared_Train':[r2_train],'R-squared_Test':[r2_test],'RMSE_train':[RMSE_train],'RMSE_te
GGB_Results = pd.DataFrame(Error_metrics_GGB)
```

In [129]:
```
Final_Results
```

Out[129]:

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Linear Regression	43.773719	19.685303	0.837362	0.839261	767.301965	826.828014
1	Decision Tree	62.260133	36.948093	0.677563	0.646470	1080.381858	1226.219619
2	Decision Tree Random Search CV	8.426494	21.320865	0.937985	0.842193	473.807532	819.253889
3	Decision Tree Grid Search CV	14.033006	22.953146	0.875127	0.811933	672.339115	894.356646
4	Random Forest	13.495810	20.324684	0.980794	0.883042	263.677930	705.293131
5	Random Forest Random Search CV	19.501021	20.501396	0.979519	0.884690	272.288350	700.305764
6	Random Forest Grid Search CV	17.211393	20.531874	0.974122	0.881434	306.067200	710.124889
7	Gradient Boosting	17.211393	18.766969	0.949290	0.867347	428.450622	751.126323
8	Gradient Boosting Random Search CV	1.732120	21.125833	0.998500	0.872410	767.301965	826.828014
9	Gradient Boosting Grid Search CV	14.959465	25.317824	0.960632	0.838236	377.506431	829.460512

Appendix C – References.....

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

Thank You