BIKE RENTAL COUNT

DATA SCIENCE PROJECT

MAYUR SHARMA

WWW.EDWISOR.COM

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 Dataset

Sample Dataset-

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
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

temp	atemp	hum	windspeed	casual	registered	cnt
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

Dataset has 16 variables in which 15 variables are independent and 1 ('cnt') is dependent variable. And we have to prepare a model to predict the count of bikes on daily basis based on environmental. In the dataset target variable is continuous in nature, so this is a regression problem.

Attribute Information:

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. holiday: weather day is holiday or not (extracted fromHoliday Schedule)
7. weekday: Day of the week
8. workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
9.weathersit: (extracted fromFreemeteo)
i: Clear, Few clouds, Partly cloudy, Partly cloudy
ii: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
iii: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
iv: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
10. 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)
11. 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)
12. hum: Normalized humidity. The values are divided to 100 (max)

13. windspeed: Normalized wind speed. The values are divided to 67 (max)

14. casual: count of casual users

15. registered: count of registered users

16. cnt: count of total rental bikes including both casual and registered

1.3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an approach to analysing data sets to summarize their main characteristics. In the given data set there are 16 variables and data types of all variables are object, float64 or int64. There are 731 observations and 16 columns in our data set.

- **instant int64**
- **dteday object**
- **season int64**
- **yr int64**
- **mnth int64**
- **holiday int64**
- **weekday int64**
- **workingday int64**
- **weathersit int64**
- **temp float64**
- **atemp float64**
- **hum float64**
- **windspeed float64**
- **casual int64**
- **registered int64**
- **cnt int64**
- **dtype: object**

From EDA we have concluded that there are 7 continuous variable and 9 categorical variable in nature.

Continuous variables in dataset-

- temp float64
- atemp float64
- hum float64
- windspeed float64
- casual int64
- registered int64
- cnt int64

Categorical variables in dataset-

- **instant int64**
- **dteday object**
- **season int64**
- **yr int64**
- **mnth int64**
- **holiday int64**
- **weekday int64**
- **workingday int64**
- **weathersit int64**

From EDA we have concluded the number of unique values in each variables.

- instant 731
- dteday 731
- season 4
- yr 2
- mnth 12
- holiday 2
- weekday 7
- workingday 2
- weathersit 3
- temp 499
- atemp 690
- hum 595
- windspeed 650
- casual 606
- registered 679
- cnt 696

In EDA we have seen that some of variables are not important for proceed further as these are irrelevant variable in our dataset so we will remove them before processing the data. we have dropped variable **'Instant'** as it is index in dataset, also removed **'dteday'** variable as it is not Time-Series data, so we dropped it, also there are two variables **'casual'** and **'registered',** because these two variables sum is our target variable, so these are not of our use. so we dropped them.

In EDA we rename some of variables in our dataset before proceeding further, for better understanding the dataset. After renaming of variables the updated variables name are as-

- season
- year
- month
- holiday
- weekday
- workingday
- weather
- temperature
- humidity
- windspeed
- count

1.4 Data Understanding

For better understanding of data, here we have plotted some visualization for the variables.

1. From season plot in figure-1.4.1 we can see that season 2,3 and 4 have more bike count as compare to season 1. the daily bike count for these season was between 4000 to 8000.

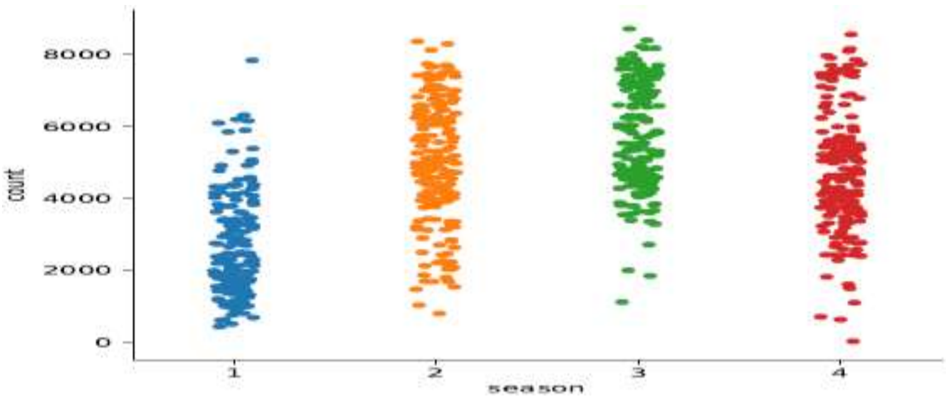


figure-1.4.1

2. Below plot figure-1.4.2 is for month wise count of bikes, so this tells us that the bike counts are higher between month 4 to month 10 as comapre to other months.

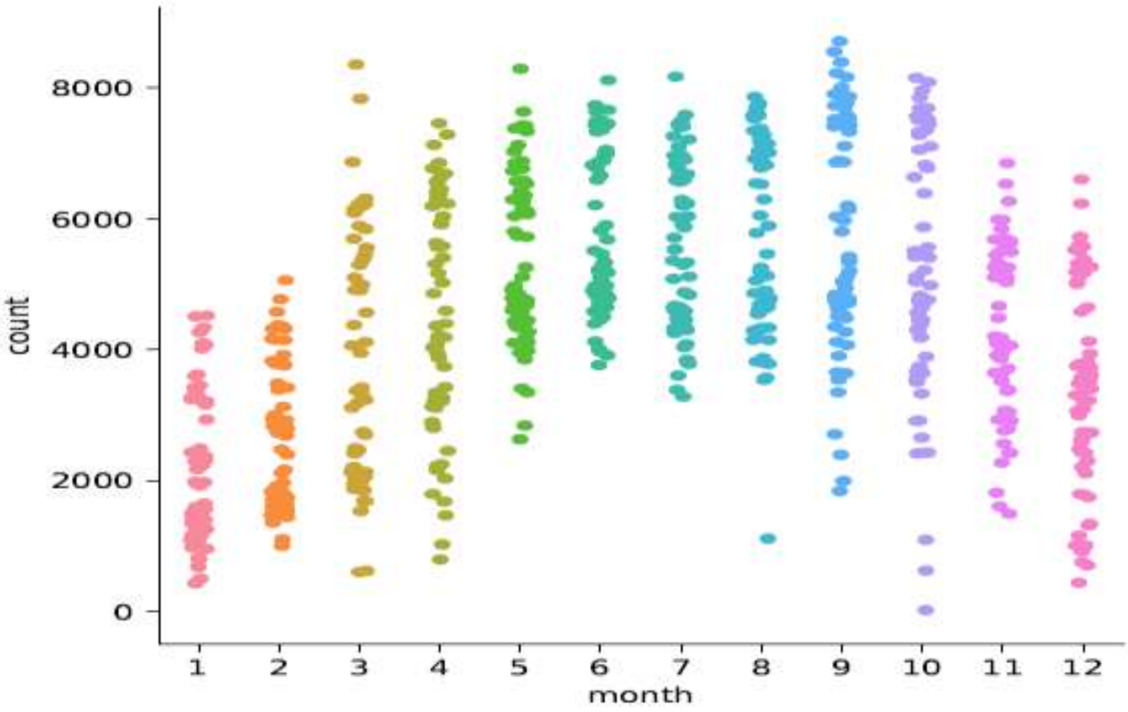


figure-1.4.2

3. Below Plot Figure-1.4.3 is between holiday and count, from this plot we can clearly say count of rented bikes are higher on holiday.

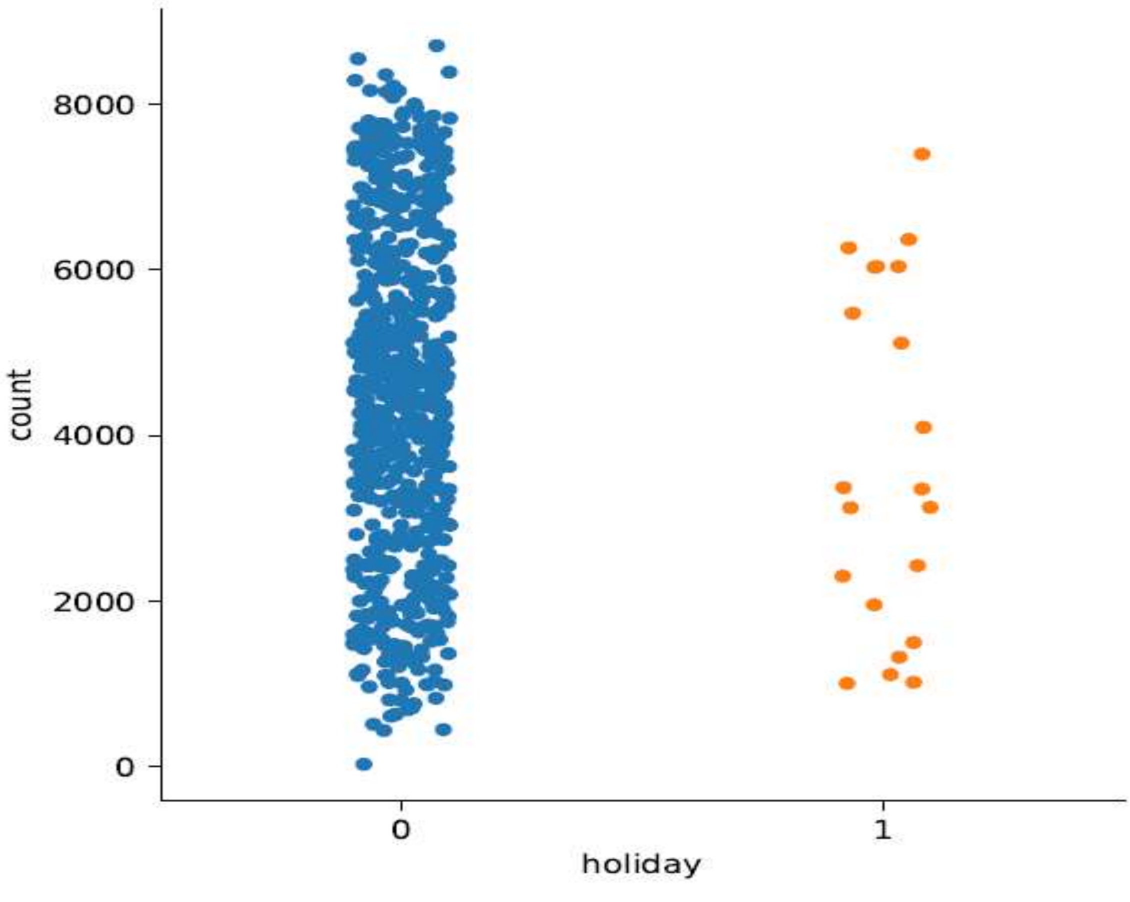


figure-1.4.3

4. In weather-1 in figure-1.4.4 the count of bikes is good as compare to other weather.

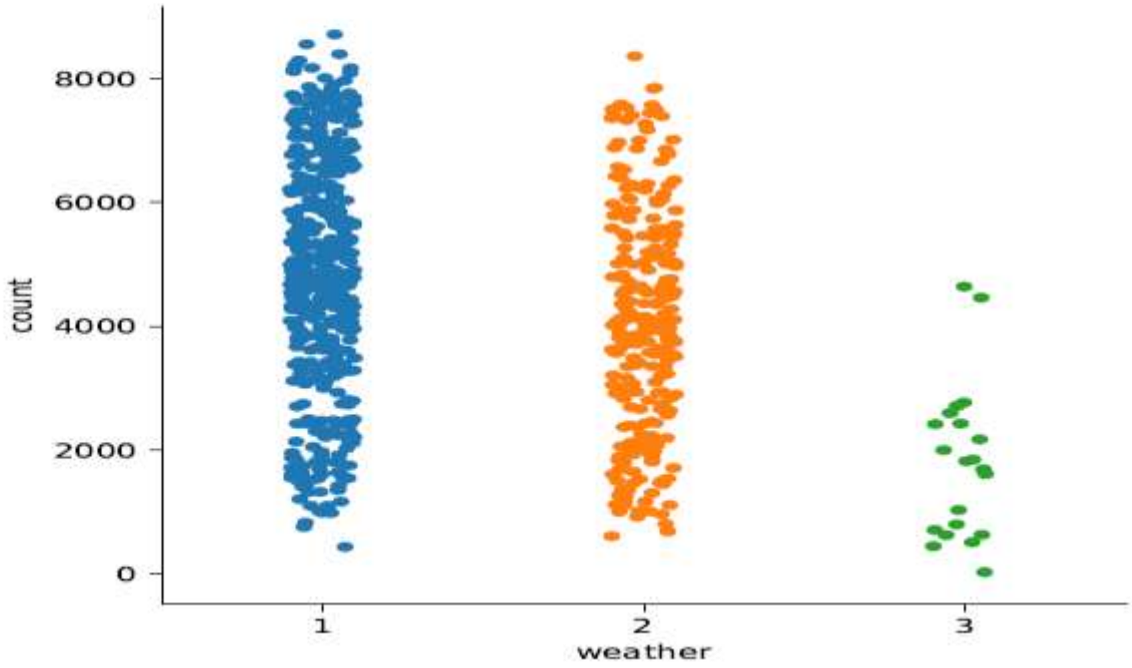


figure-1.4.4

5. Below plot figure-1.4.5 is for count bike with respect to normalized temperature and normalized humidity, from this we can see that count is maximum when temperature 0.4 to 0.7 and humidity below 0.75

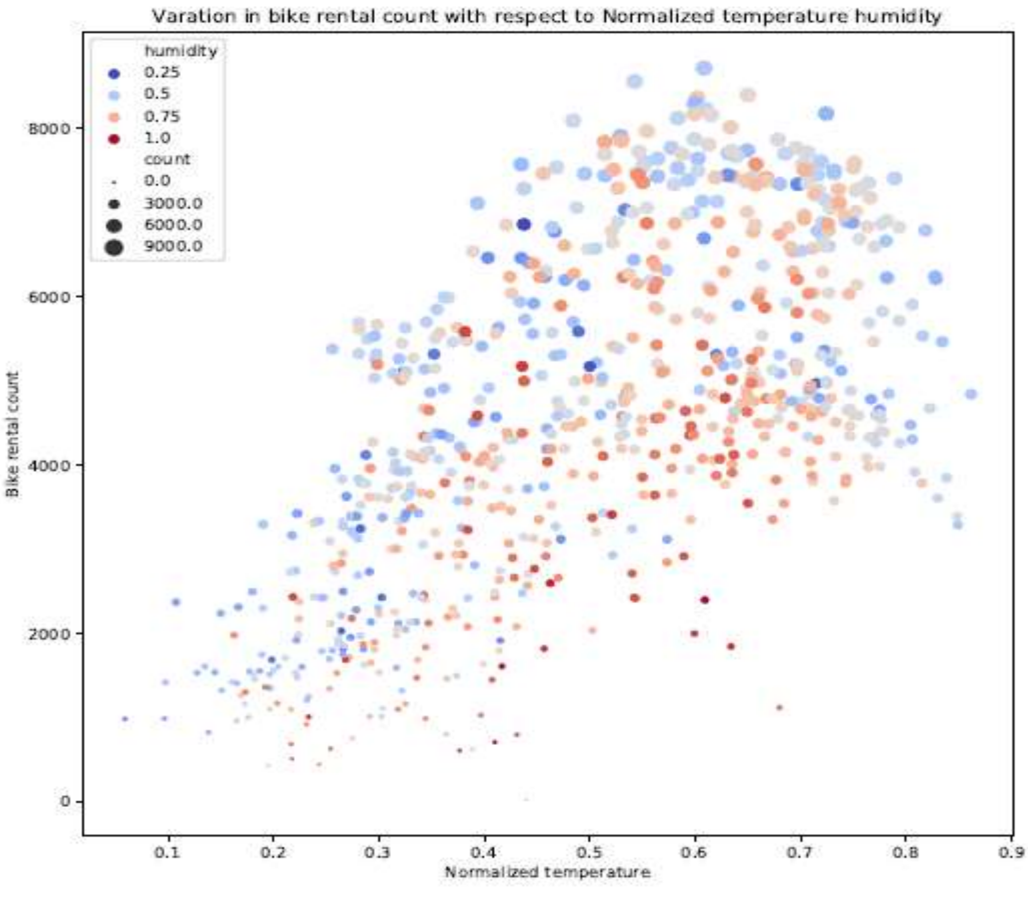


figure-1.4.5

6. The below plot figure-1.4.6 is for bike count with respect to Normalized Temperature and Normalized Humidity, from this plot it is clear that count is higher when the temp is 0.5 to 0.7 and windspeed below 0.15 and humidity less than 0.75

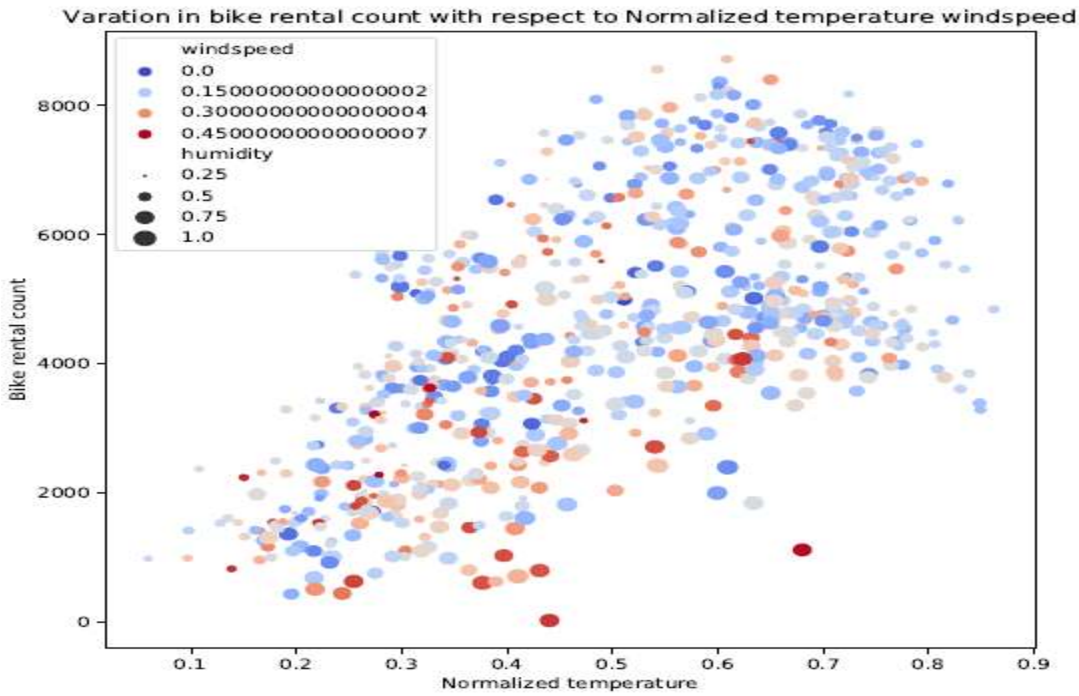


figure-1.4.6

7. Below Plot figure-1.4.7 is plotted for count of bikes with respect to temperature, weather and humidity, and we have found that the count is maximum when temperature is between 0.5 to 0.7, and in season 2 and 3.

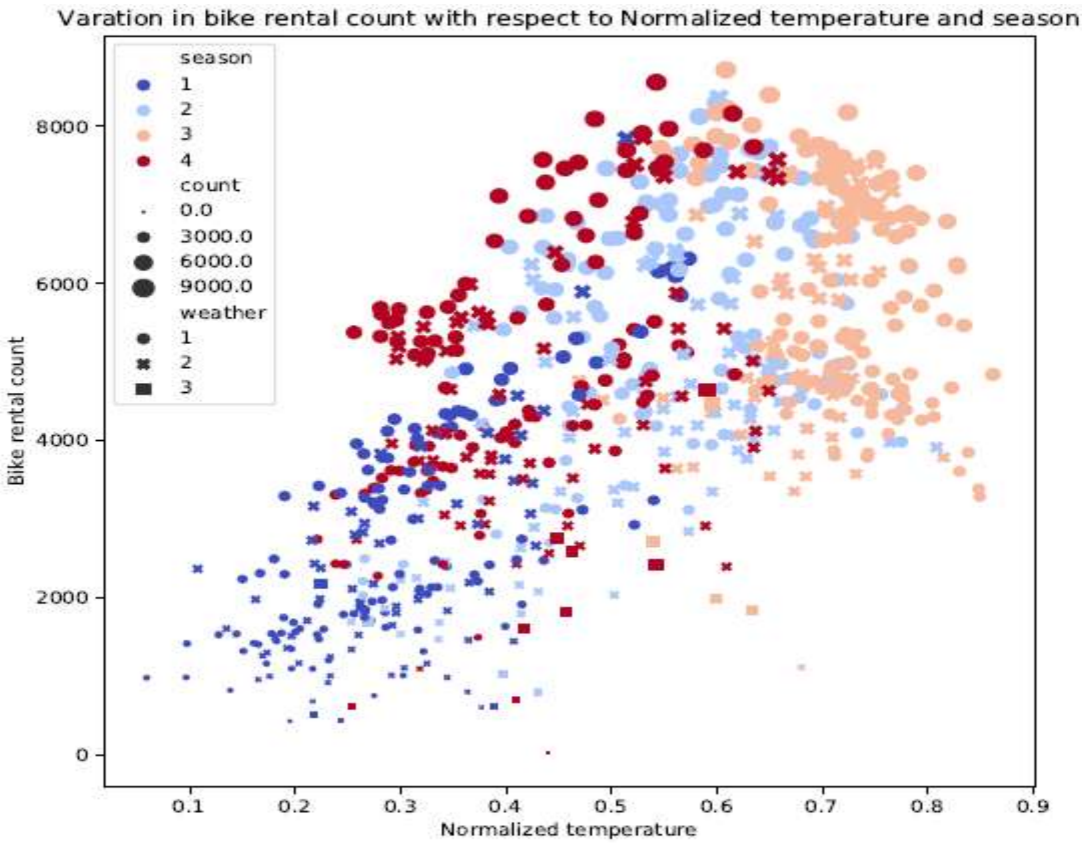


figure-1.4.7

2. Methodology

Before feeding the data to the model we need to clean the data and convert it to a proper format. It is the most crucial part of data science. In this we have to apply different pre-processing techniques to clean the data and to convert it into proper format.

2.1 Data Pre Processing

Any predictive modelling requires that we look at the data before we start modelling. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process we will first try and look at all the probability distributions of the variables. Most analysis like regression, require the data to be normally distributed. We can visualize in the below figure 2.1.1 and figure 2.1.2 the probability distributions or probability density functions of the variables.

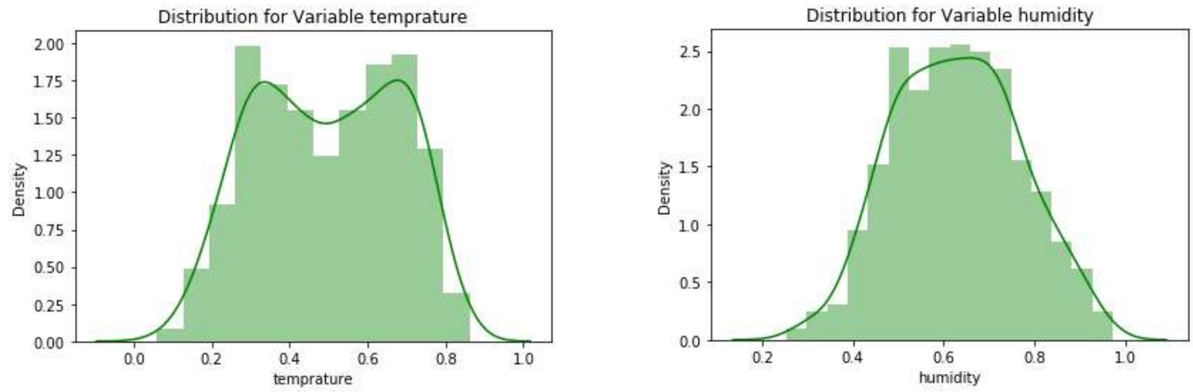


figure-2.1.1

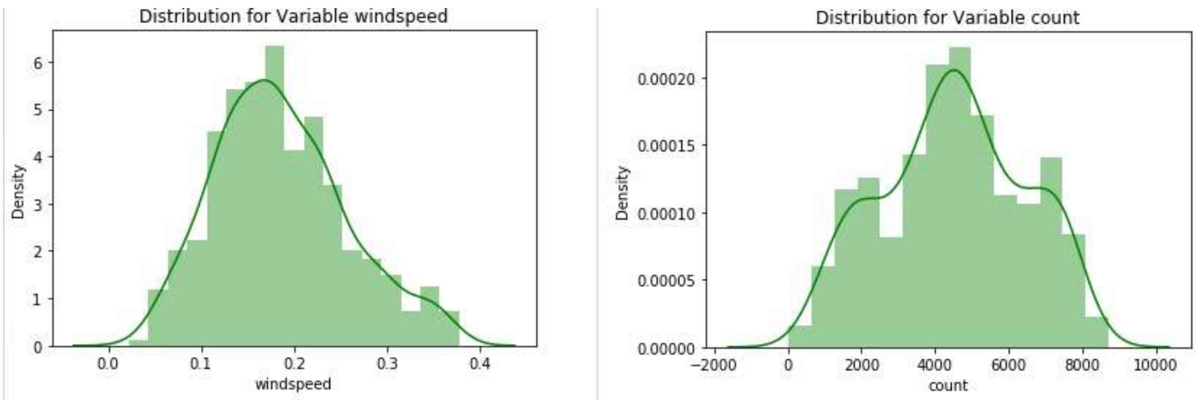


figure-2.1.2

2.2.1 Missing Value Analysis

In statistics, *missing data*, or *missing values*, occur when no *data value* is stored for the variable in an observation. If a columns has 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.

Missing Values in Dataset-

- `season        0`
- `year          0`
- `month         0`
- `holiday       0`
- `weekday       0`
- `workingday    0`
- `weather       0`
- `temprature    0`

- `atemp`   0
- `humidity`   0
- `windspeed`   0
- `count`   0

2.1.2 Outlier Analysis

One of the other steps of pre-processing is to check the presence of outliers. Outliers are those values which are present in the dataset with a abnormal distance from most part of values. The issue of outlier occurs only in Continuous variables. Here to check the outlier in our dataset, we used a classic approach to visualize outliers, that is Boxplot Method.

In figure 2.1.2.1 and figure 2.1.2.2 we have plotted the boxplots of the continuous variables with respect to target variable **count**, and detect the outliers by visualization.

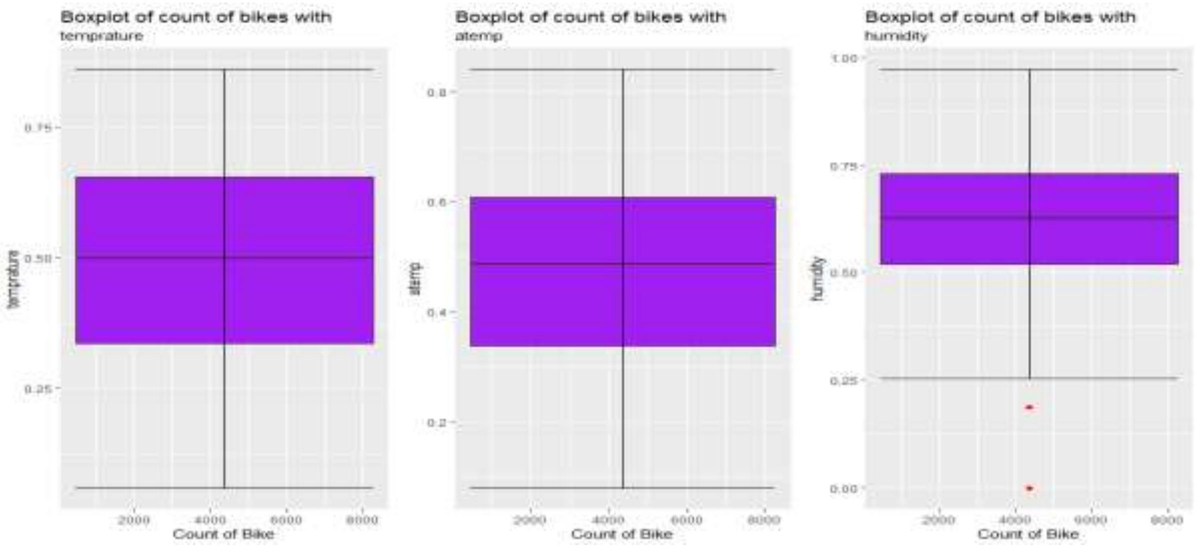


figure-2.1.2.1

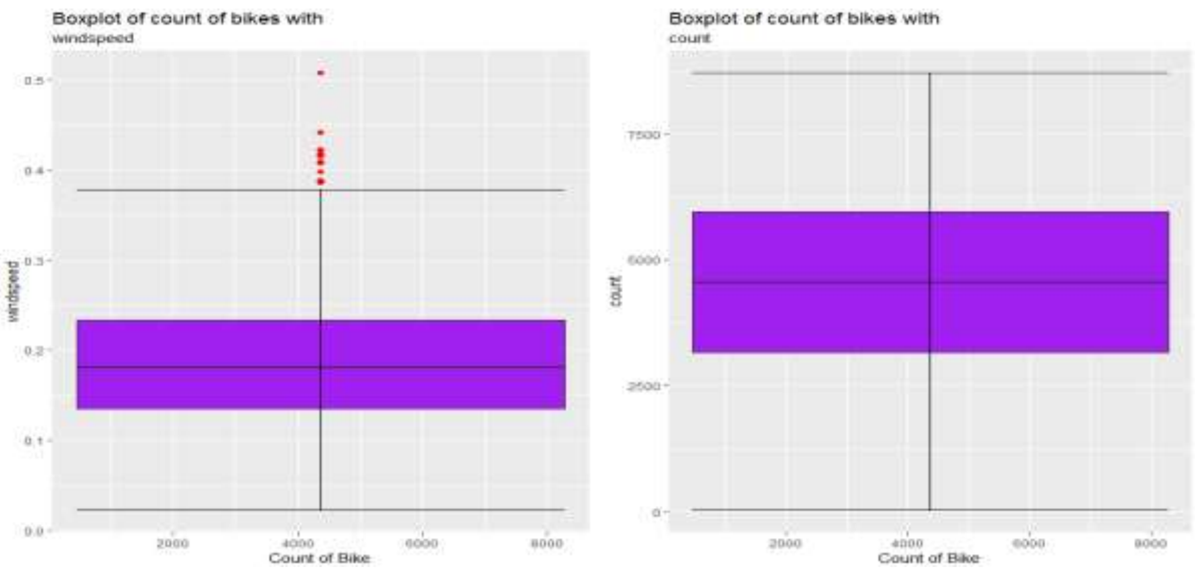


figure-2.1.2.2

From the boxplot almost all the variables **except "windspeed" and "humidity"** does not have outliers. From the boxplot visualization .We have converted the outliers (data beyond minimum and maximum values) as NA i.e. missing values and fill them by **Median** imputation method.

2.1.3 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 numerical variable and **ANOVA** (Analysis of variance) for categorical variables.

Correlation Analysis plot (figure-2.1.3.1) for continuous variables-

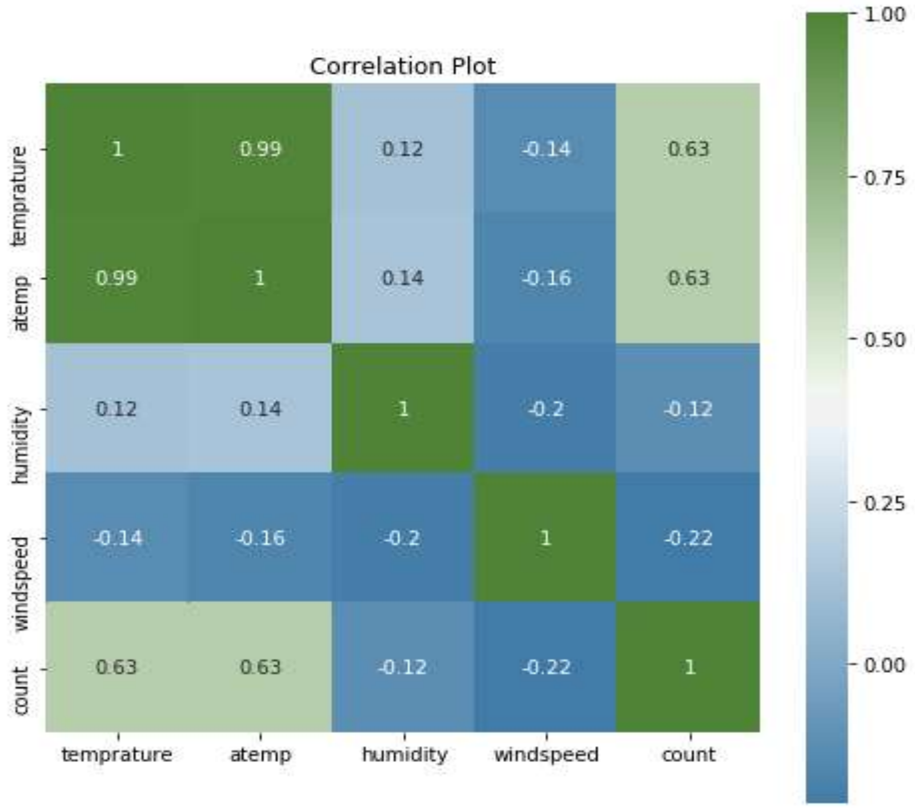


figure-2.1.3.1

ANOVA Analysis for categorical variables-

	sum_sq	df	F	PR(>F)
season	4.52E+08	1	143.967653	2.13E-30
Residual	2.29E+09	729	NaN	NaN

	sum_sq	df	F	PR(>F)
year	8.80E+08	1	344.890586	2.48E-63
Residual	1.86E+09	729	NaN	NaN

	sum_sq	df	F	PR(>F)
month	2.15E+08	1	62.004625	1.24E-14
Residual	2.52E+09	729	NaN	NaN

	sum_sq	df	F	PR(>F)
holiday	1.28E+07	1	3.421441 0	0.064759
Residual	2.73E+09	729	NaN	NaN

	sum_sq	df	F	PR(>F)
weekday	1.25E+07	1	3.331091 0	0.068391
Residual	2.73E+09	729	NaN	NaN

	sum_sq	df	F	PR(>F)

	sum_sq	df	F	PR(>F)
workingday	1.02E+07	1	0 2.736742	0.098495
Residual	2.73E+09	729	0 NaN	NaN

	sum_sq	df	F	PR(>F)
weather	2.42E+08	1	70.729298	2.15E-16
Residual	2.50E+09	729	NaN	NaN

From correlation analysis we have found that **temperature** and **atemp** has high correlation (>0.9), so we have excluded the **atemp** column, and from ANOVA analysis we have found that in categorical variables **Holiday, weekday and working day** have the pr(>0.05), so we excluded them.

After Correlation 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**

2.1.4 Feature Scaling

Feature scaling is a method used to standardize 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. In some machine learning algorithms, objective functions will not work properly without normalization. For example, the majority of classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance. Since as in given dataset for continuous variables data is already Normalized, so we do not need to scale the data.

	temperature	humidity	windspeed
count	731.000000	731.000000	731.000000
mean	0.495385	0.629354	0.186257
std	0.183051	0.139566	0.071156
min	0.059130	0.254167	0.022392
25%	0.337083	0.522291	0.134950
50%	0.498333	0.627500	0.178802
75%	0.655417	0.730209	0.229786
max	0.861667	0.972500	0.378108

2.2 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 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.

2.2.2 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 in our model-

- 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.

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. The RMSE value and R^2 value for our project in R and Python are –

2.2.4 Liner Regression

Linear Regression is one of the statistical method of prediction. It is most common predictive analysis algorithm. It uses only for regression, means if the target variable is continuous than we can use linear regression machine learning algorithm.

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.

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

In the previous chapter we have applied four algorithms on our dataset and calculate the Mean absolute percentage error (MAPE) and R-Squared Value for all the models. MAPE is a measure of prediction accuracy of a forecasting method. It measures accuracy in terms of percentage. R-squared is basically explains the degree to which input variable explain the variation of the output. In simple words R-squared tells how much variance of dependent variable explained by the independent variable. It is a measure if goodness of fit in regression line. Value of R-squared between 0-1, where 0 means independent variable unable to explain the target variable and 1 means target variable is completely explained by the independent variable. So, lower values of MAPE and higher value of R-Squared Value indicate better fit of model.

Here, the result of each model in Python and R as-

Python Result-

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test
0	Decision Tree	62.260133	36.948093	0.677563	0.646470
1	Decision Tree Random Search CV	14.180789	23.419816	0.874435	0.809361
2	Decision Tree Grid Search CV	14.180789	23.419816	0.874435	0.809361
3	Random Forest	16.776997	20.426067	0.979178	0.881801
4	Random Forest Random Search CV	21.445350	21.029355	0.978219	0.878929
5	Random Forest Grid Search CV	21.320742	20.567325	0.964826	0.875335
6	Linear Regression	44.444512	18.800696	0.832760	0.841110
7	Gradient Boosting	44.444512	19.899341	0.945385	0.864595
8	Gradient Boosting Random Search CV	1.732620	21.730096	0.998236	0.866549
9	Gradient Boosting Grid Search CV	18.833448	25.485646	0.922114	0.833746

R-Result-

Model	MAPE_Train	MAPE_Test	R.Squared_Train	R.Squared_Test
Decision Tree for Regression	56.30014552	23.70970208	0.793974257	0.752194789
Random Search in Decision Tree	56.30014552	23.70970208	0.793974257	0.752194789
Gird Search in Decision Tree	56.30014552	23.70970208	0.793974257	0.752194789
Random Forest	23.31578346	17.41229633	0.967674325	0.866706395
Random Search in Random Forest	25.44288679	17.52099336	0.96787762	0.865370081
Grid Search in Random Forest	24.88492838	17.61271901	0.964533357	0.866318841
Linear Regression	47.40023298	16.87102913	0.900758595	0.851246285
Gradient Boosting	37.02665525	17.24280795	0.900758595	0.851246285
Random Search in Gradient Boosting	25.00204653	17.60370181	0.968267213	0.863821245
Grid Search in Gradient Boosting	25.64630592	17.40282785	0.964533357	0.866318841

3.2 Model Selection

From the observation of all **MAPE** and **R-Squared** 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**).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.

4. Coding

In this section we are attaching the coding of R and Python which we developed for our model.

4.1.R Coding

```
#Clear Environment-
rm(list=ls())
#Set working directory-
setwd("D:/R-programming/2.Project- Bike Rental-R_File")
#Check working directory-
getwd()

#load data-
data= read.csv("day.csv")

#-------------------------------Exploratory Data Analysis------------------------------
class(data)
dim(data)
head(data)
names(data)
str(data)
summary(data)

#Remove the instant variable, as it is index in dataset.
data= subset(data,select=-(instant))
#Remove date variable as we have to predict count on seasonal basis not date basis-
data= subset(data,select=-(dteday))
#Remove casual and registered variable as count is sum of these two variables-
data= subset(data,select=-c(casual,registered))

#check the remaining variables-
names(data)

#Rename the variables-
names(data)[2]="year"
names(data)[3]="month"
names(data)[7]="weather"
names(data)[8]="temprature"
names(data)[10]="humidity"
names(data)[12]="count"
#Seperate categorical and numeric variables-
names(data)

#numeric variables-
cnames= c("temprature","atemp","humidity","windspeed","count")

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

#=================================Data Pre-processing==================================

#-------------------------------Missing Vlaue Analysis------------------------------
#Check missing values in dataset-
sum(is.na(data))
#Missing value= 0
#No Missing values in data.

#-----------------------------------Outlier Analysis----------------------------------
df=data
data=df

#create Box-Plot for outlier analysis-
library(ggplot2)    #Library for visulization-
for(i in 1:length(cnames)){
  assign(paste0("AB",i),ggplot(aes_string(x="count",y=(cnames[i])),d=subset(data))+
           geom_boxplot(outlier.color = "Red",outlier.shape = 18,outlier.size = 2,
                        fill="Purple")+theme_get()+
           stat_boxplot(geom = "errorbar",width=0.5)+
           labs(x="Count of Bike",y=cnames[i])+
           ggtitle("Boxplot of count of bikes with",cnames[i]))
}

gridExtra::grid.arrange(AB1,AB2,AB3,ncol=3)
gridExtra::grid.arrange(AB4,AB5,ncol=2)
```

```
#Replace outliers with NA-
for(i in cnames){
  print(i)
  outlier= data[,i][data[,i] %in% boxplot.stats(data[,i])$out]
  print(length(outlier))
  data[,i][data[,i] %in% outlier]=NA
}

sum(is.na(data))

#Impute outliers by median method-
data$humidity[is.na(data$humidity)]=median(data$humidity,na.rm=TRUE)
data$windspeed[is.na(data$windspeed)]=median(data$windspeed,na.rm=TRUE)

sum(is.na(data))

#---------------------------------Data Understanding-----------------------------------

#Barplot of bike rented with respect to working days-
ggplot(data, aes(x = reorder(weekday,-count), y = count))+
  geom_bar(stat = "identity",fill = "aquamarine3")+
  labs(title = "Number of bikes rented with respect to days", x = "Days of the week")+
  theme(panel.background = element_rect("antiquewhite"))+
  theme(plot.title = element_text(face = "bold"))

#->from bar plot we can see maximum bikes rented on day 5 least bikes on day 0.

#Bikes rented with respect to temp and humidity-
ggplot(data,aes(temprature,count)) +
  geom_point(aes(color=humidity),alpha=0.5) +
  labs(title = "Bikes rented with respect to variation in temperature and hunidity", x 
  scale_color_gradientn(colors=c(dark blue,blue,light blue,light green,yellow,orange,re
  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 windspeed-
ggplot(data, aes(x = temprature, y = count))+
  geom_point(aes(color=weather))+
  labs(title = "Bikes rented with respect to temperature and weathersite", x = "Normali
  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

#Bikes rented with respect to temp and season-
ggplot(data, aes(x = temprature, y = count))+
  geom_point(aes(color=season))+
  labs(title = "Bikes rented with respect to temperature and season", x = "Normalized t
  #  theme(panel.background = element_rect("white"))+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme_bw()
#->maximum bike rented in season4

#---------------------------------Feature Selection-----------------------------------
df=data
data=df

#correlation analysis for numeric variables-
library(corrgram)
corrgram(data[,cnames],order=FALSE,upper.panel = panel.pie,
         text.panel = panel.txt,
         main= "Correlation plot for numeric variables")
```

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

#Dimension Reduction-
data = subset(data,select=-c(atemp,holiday,weekday,workingday))

#--------------------------------Feature Scaling------------------------------------
df=data
data=df

#update numeric variables after dimension reduction-
cnames= c("temprature","humidity","windspeed","count")

#skewness test for continuous variables-
library(propagate)
for(i in cnames){
  print(i)
  skew= skewness(data[,i])
  print(skew)
}
#No skewness in dataset.

#Normality check using histogram plot-
hist(data$temprature,col="Green",xlab="Temprature",ylab="Frequency",
     main="Histogram of Temprature")
hist(data$humidity,col="Red",xlab="Humidity",ylab="Frequency",
     main="Histogram of Humidity")
hist(data$windspeed,col="Purple",xlab="Windspeed",ylab="Frequency",
     main="Histogram of Windspeed")

#check summary of continuous variable to check the scaling-
for(i in cnames){
  print(summary(data[,i]))
}
#as from summary, the data is already normalized, so no need for scaling.

#save the pre-processed data in drive-
write.csv(data,"Bike_Rental_count.csv",row.names=FALSE)

#====================================Model Devlopment==================================

#Clean the Environment-
library(DataCombine)
rmExcept("data")

#Data Copy for refrance-
df=data
data=df

#Function for Error metrics to calculate the performance of model-
mape= function(y,y1){
  mean(abs((y-y1)/y))*100
}

#Function for r2 to calculate the goodness of fit of model-
rsquare=function(y,y1){
  cor(y,y1)^2
}

#convert categorical variables into dummy variable-
#Recall categorical variables-
cat_cnames= c("season","year","month","weather")
```

```r
library(dummies)
data= dummy.data.frame(data,cat_cnames)

#divide the data into traina nd test-
set.seed(123)
train_index= sample(1:nrow(data),0.8*nrow(data))
train= data[train_index,]
test= data[-train_index,]

#------------------------------Decision Tree for Regression----------------------------
#Model devlopment on train data-
library(rpart)

DT_model= rpart(count~.,train,method = "anova")
DT_model

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

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

#Mape calculation of train data-
DT_MAPE_Train = mape(train[,25],DT_train)
#mape= 56.30%

#Mape calculation of test data-
DT_MAPE_Test = mape(test[,25],DT_test)
#mape=23.70%

#r2 calculation for train data-
DT_r2_train= rsquare(train[,25],DT_train)
#r2_test= 0.79

#r2 calculation for test data-
DT_r2_test=rsquare(test[,25],DT_test)
#r2_test= 0.75

##############Random Search CV in Decision Tree#######################################

#set parameters-
library(caret)

control = trainControl(method="repeatedcv", number=5, repeats=1,search=random)
maxdepth = c(1:30)
tunegrid = expand.grid(.maxdepth=maxdepth)

#model devlopment on train data-
RDT_model = caret::train(count~., data=train, method="rpart2",trControl=control,tuneGri

print(RDT_model)

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

#build model based on best fit-
RDT_model = rpart(count ~ .,train, method = "anova", maxdepth =7)

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

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

#Mape calculation of train data-
RDT_MAPE_Train = mape(train[,25],RDT_train)
#mape= 56.30%
```

```
#Mape calculation of test data-
RDT_MAPE_Test = mape(test[,25],RDT_test)
#mape=23.70%

#r2 calculation for train data-
RDT_r2_train= rsquare(train[,25],RDT_train)
#r2_test= 0.79

#r2 calculation for test data-
RDT_r2_test=rsquare(test[,25],RDT_test)
#r2_test= 0.75

##############Grid Search CV in Decision Tree##########################################

control = trainControl(method="repeatedcv", number=5, repeats=2, search="grid")
tunegrid = expand.grid(.maxdepth=c(6:18))

#model devlopment on train data-
GDT_model= caret::train(count~.,train, method="rpart2", tuneGrid=tunegrid, trControl=co
print(GDT_model)

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

#build model based on best fit-
GDT_model = rpart(count ~ .,train, method = "anova", maxdepth =7)

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

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

#Mape calculation of train data-
GDT_MAPE_Train = mape(train[,25],GDT_train)
#mape= 56.30%

#Mape calculation of test data-
GDT_MAPE_Test = mape(test[,25],GDT_test)
#mape=23.70%

#r2 calculation for train data-
GDT_r2_train= rsquare(train[,25],GDT_train)
#r2_test= 0.79

#r2 calculation for test data-
GDT_r2_test=rsquare(test[,25],GDT_test)
#r2_test= 0.75

#--------------------------------Random Forest for Regression--------------------------
#Model devlopment on train data-
library(randomForest)
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 calculation of train data-
RF_MAPE_Train=mape(train[,25],RF_train)
#mape= 23.31%

#Mape calculation of test data-
RF_MAPE_Test=mape(test[,25],RF_test)
```

```
#mape= 17.41%

#r2 calculation for train data-
RF_r2_train=rsquare(train[,25],RF_train)
#r2_test= 0.96

#r2 calculation for test data-
RF_r2_test=rsquare(test[,25],RF_test)
#r2_test= 0.87

##############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)

#model devlopment on train data-
RGB_model = caret::train(count~., data=train, method="rf",trControl=control,tuneLength=

print(RGB_model)

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

#build model based on best fit-
RGB_model = randomForest(count ~ .,train, method = "anova", mtry=8,importance=TRUE)

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

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

#Mape calculation of train data-
RGB_MAPE_Train = mape(train[,25],RGB_train)
#mape= 25.44%

#Mape calculation of test data-
RGB_MAPE_Test = mape(test[,25],RGB_test)
#mape=17.52%

#r2 calculation for train data-
RGB_r2_train= rsquare(train[,25],RGB_train)
#r2_test= 0.96

#r2 calculation for test data-
RGB_r2_test=rsquare(test[,25],RGB_test)
#r2_test= 0.86

##############Grid Search CV in Random Forest##########################################

control = trainControl(method="repeatedcv", number=5, repeats=2, search="grid")
tunegrid = expand.grid(.mtry=c(6:18))

#model devlopment on train data-
GRF_model= caret::train(count~.,train, method="rf", tuneGrid=tunegrid, trControl=contro
print(GRF_model)

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

#build model based on best fit-
GRF_model = randomForest(count ~ .,train, method = "anova", mtry=7)

#Prediction on train data-
GRF_train= predict(GRF_model,train[-25])
```

```
#Prediction on test data-
GRF_test= predict(GRF_model,test[-25])

#Mape calculation of train data-
GRF_MAPE_Train = mape(train[,25],GRF_train)
#mape= 24.88%

#Mape calculation of test data-
GRF_MAPE_Test = mape(test[,25],GRF_test)
#mape=17.61%

#r2 calculation for train data-
GRF_r2_train= rsquare(train[,25],GRF_train)
#r2_test= 0.96

#r2 calculation for test data-
GRF_r2_test=rsquare(test[,25],GRF_test)
#r2_test= 0.87

#---------------------------------Linear Regression-----------------------------------

#Recall numeric variables to check the VIF for multicollinearity-
cnames= c("temprature","humidity","windspeed")
numeric_data= data[,cnames]

#VIF test-
library(usdm)
vifcor(numeric_data,th=0.7)

#Model devlopment on train data-
LR_model= lm(count~.,train)
summary(LR_model)

#prediction on train data-
LR_train= predict(LR_model,train[-25])
#prediction on test data-
LR_test= predict(LR_model,test[-25])

#Mape calculation of train data-
LR_MAPE_Train=mape(train[,25],LR_train)
#mape= 47.40%

#Mape calculation of test data-
LR_MAPE_Test=mape(test[,25],LR_test)
#mape= 16.87%

#r2 calculation for train data-
LR_r2_train=rsquare(train[,25],LR_train)
#r2_test= 0.84

#r2 calculation for test data-
LR_r2_test=rsquare(test[,25],LR_test)
#r2_test= 0.83

#---------------------------------Gradient Boosting------------------------------------
library(gbm)

#Develop Model
GB_model = gbm(count~., data = train, n.trees = 100, interaction.depth = 2)

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

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

#Mape calculation of train data-
GB_MAPE_Train=mape(train[,25],GB_train)
#mape= 37.02%
```

```
#Mape calculation of test data-
GB_MAPE_Test=mape(test[,25],GB_test)
#mape= 17.24%

#r2 calculation for train data-
GB_r2_train=rsquare(train[,25],GB_train)
#r2_test= 0.90

#r2 calculation for test data-
GB_r2_test=rsquare(test[,25],GB_test)
#r2_test= 0.85

##############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 data-
RGB_model = caret::train(count~., data=train, method="gbm",trControl=control,tuneLength

print(RGB_model)

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

#build model based on best fit-
RGB_model = randomForest(count ~ .,train, method = "anova", n.trees=15,
                          interaction.depth=4,shrinkage=0.433567,n.minobsinnode=20)

#Prediction on train data-
RGB_train= predict(RGB_model,train[-25])
#Prediction on test data-
RGB_test= predict(RGB_model,test[-25])

#Mape calculation of train data-
RGB_MAPE_Train = mape(train[,25],RGB_train)
#mape= 25.00%

#Mape calculation of test data-
RGB_MAPE_Test = mape(test[,25],RGB_test)
#mape=17.60%

#r2 calculation for train data-
RGB_r2_train= rsquare(train[,25],RGB_train)
#r2_test= 0.97

#r2 calculation for test data-
RGB_r2_test=rsquare(test[,25],RGB_test)
#r2_test= 0.86

##############Grid Search CV in Gradient Boosting######################################

control = trainControl(method="repeatedcv", number=5, repeats=2, search="grid")
tunegrid = expand.grid(n.trees = seq(2565,2575, by = 2),
                        interaction.depth = c(2:4),
                        shrinkage = c(0.01,0.02),
                        n.minobsinnode = seq(18,22, by = 2))

#model devlopment on train data-
GGB_model= caret::train(count~.,train, method="gbm", tuneGrid=tunegrid, trControl=contr
print(GGB_model)

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

```
#build model based on best fit-
GGB_model = randomForest(count ~ .,train, method = "anova", n.trees = 2569,
                         interaction.depth = 4,shrinkage = 0.01,n.minobsinnode = 20)

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

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

#Mape calculation of train data-
GGB_MAPE_Train = mape(train[,25],GGB_train)
#mape= 25,64%

#Mape calculation of test data-
GGB_MAPE_Test = mape(test[,25],GGB_test)
#mape=17.40%

#r2 calculation for train data-
GGB_r2_train= rsquare(train[,25],GGB_train)
#r2_test= 0.97

#r2 calculation for test data-
GGB_r2_test=rsquare(test[,25],GGB_test)
#r2_test= 0.86

#===================================Result==================================================#

Result= data.frame(Model=c('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',
                           'Linear Regression','Gradient Boosting','Random Search in Gradient Boosting',
                           'Grid Search in Gradient Boosting'),'MAPE_Train'=c(DT_MAPE_Train,
                           RDT_MAPE_Train,GDT_MAPE_Train,RF_MAPE_Train,RRF_MAPE_Train,
                           GRF_MAPE_Train,LR_MAPE_Train,GB_MAPE_Train,RGB_MAPE_Train,GGB_MAPE_Train),
                  'MAPE_Test'=c(DT_MAPE_Test,RDT_MAPE_Test,GDT_MAPE_Test,RF_MAPE_Test,RRF_MAPE_Test,
                                GRF_MAPE_Test,LR_MAPE_Test,GB_MAPE_Test,RGB_MAPE_Test,GGB_MAPE_Test),
                  'R-Squared_Train'=c(DT_r2_train,RDT_r2_train,GDT_r2_train,RF_r2_train,RRF_r2_train,
                                      GRF_r2_train,LR_r2_train,GB_r2_train,RGB_r2_train,GRF_r2_train),
                  'R-Squared_Test'=c(DT_r2_test,RDT_r2_test,GDT_r2_test,RF_r2_test,RRF_r2_test,
                                     GRF_r2_test,LR_r2_test,GB_r2_test,RGB_r2_test,GRF_r2_test))


write.csv(Result,"Result.csv",row.names=FALSE)

######################################Thank You####################################################
```

4.2 Python Coding

In [1]:
```
#load libraries-
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
```

```
C:\Users\Mayur Sharma\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the second argument of issu
bdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
C:\Users\Mayur Sharma\Anaconda3\lib\site-packages\ggplot\utils.py:81: FutureWarning: pandas.tslib is deprecated and will be rem
oved in a future version.
You can access Timestamp as pandas.Timestamp
  pd.tslib.Timestamp,
C:\Users\Mayur Sharma\Anaconda3\lib\site-packages\statsmodels\compat\pandas.py:56: FutureWarning: The pandas.core.datetools mod
ule is deprecated and will be removed in a future version. Please use the pandas.tseries module instead.
  from pandas.core import datetools
```

In [2]:
```
#set working directory-
os.chdir("D:/Python-programming/2.Project- Bike Rental-Python")

#check current working directory-
os.getcwd()
```

Out[2]:
```
'D:\\Python-programming\\2.Project- Bike Rental-Python'
```

In [3]:
```
#load data-
data= pd.read_csv("day.csv")
```

In [4]:
```
data.head()
```

Out[4]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600

Exploratory Data Analysis-

In [5]:
```
print(type(data))
print(data.shape)
print(data.dtypes)
```

```
<class 'pandas.core.frame.DataFrame'>
(731, 16)
instant         int64
dteday         object
season          int64
yr              int64
mnth            int64
holiday         int64
weekday         int64
workingday      int64
weathersit      int64
temp          float64
atemp         float64
hum           float64
windspeed     float64
casual          int64
registered      int64
cnt             int64
dtype: object
```

In [6]:
```
print(data.columns)
print(data.nunique())
```

```
Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed',
       'casual', 'registered', 'cnt'],
      dtype='object')
instant       731
dteday        731
season          4
yr              2
mnth           12
holiday         2
weekday         7
workingday      2
weathersit      3
temp          499
atemp         690
hum           595
windspeed     650
casual        606
registered    679
cnt           696
dtype: int64
```

In [7]:
```
#drop redudant variable-
#drop'instant' variable as it is index in dataset-
data= data.drop(['instant'],axis=1)

#drop 'dteday' variable as we have to predict count on seasonal basis not date basis-
data= data.drop(['dteday'],axis=1)

#drop 'casual' and 'registered' variable as traget variable is sum of these two variables-
data= data.drop(['casual','registered'],axis=1)

print(data.shape)
```

```
(731, 12)
```

In [8]:
```
#rename variables in dataset-
data= data.rename(columns={'yr':'year','mnth':'month','weathersit':'weather','temp':'temprature',
                          'hum':'humidity','cnt':'count'})

print(data.columns)
```

```
Index(['season', 'year', 'month', 'holiday', 'weekday', 'workingday',
       'weather', 'temprature', 'atemp', 'humidity', 'windspeed', 'count'],
      dtype='object')
```

In [9]:
```
#seperate continuous and categorical variables-
#continuous variable-
cnames= ['temprature', 'atemp', 'humidity', 'windspeed', 'count']

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

In [10]:
```
for i in cnames:
    print(data.loc[:,i].describe())
```

```
count    731.000000
mean       0.495385
std        0.183051
min        0.059130
25%        0.337083
50%        0.498333
75%        0.655417
max        0.861667
Name: temprature, dtype: float64
count    731.000000
mean       0.474354
std        0.162961
min        0.079070
25%        0.337842
50%        0.486733
75%        0.608602
max        0.840896
Name: atemp, dtype: float64
count    731.000000
mean       0.627894
std        0.142429
min        0.000000
25%        0.520000
50%        0.626667
75%        0.730209
max        0.972500
Name: humidity, dtype: float64
count    731.000000
mean       0.190486
std        0.077498
min        0.022392
25%        0.134950
50%        0.180975
75%        0.233214
max        0.507463
Name: windspeed, dtype: float64
count     731.000000
mean     4504.348837
std      1937.211452
min        22.000000
25%      3152.000000
50%      4548.000000
75%      5956.000000
max      8714.000000
Name: count, dtype: float64
```

Data Pre-processing-

Missing Value Analysis-

In [11]:
```
#check the missing values in dataset-
data.isnull().sum()
```

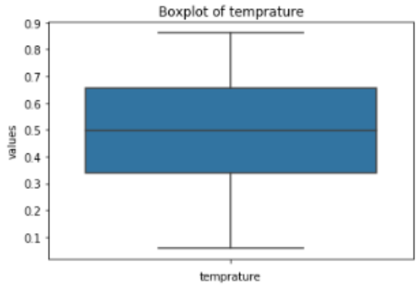
Out[11]:
```
season        0
year          0
month         0
holiday       0
weekday       0
workingday    0
weather       0
temprature    0
atemp         0
humidity      0
windspeed     0
count         0
dtype: int64
```

data does not have any missing values.

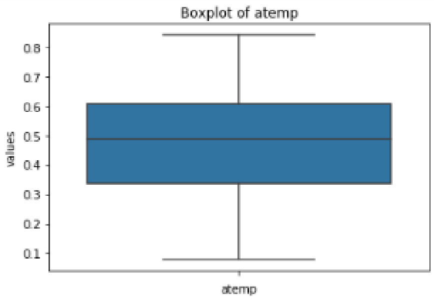
Outier Analysis-

In [12]:
```
##Plot boxplot to visulazie outliers-
for i in cnames:
    print(i)
    sns.boxplot(y=data[i])
    plt.xlabel(i)
    plt.ylabel("values")
    plt.title("Boxplot of "+i)
    plt.show()
```

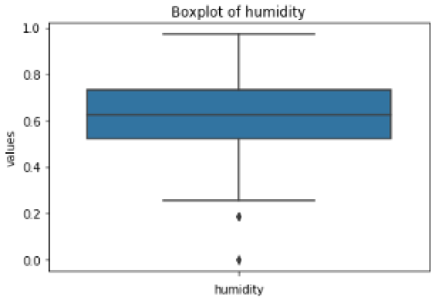
```
temprature
```



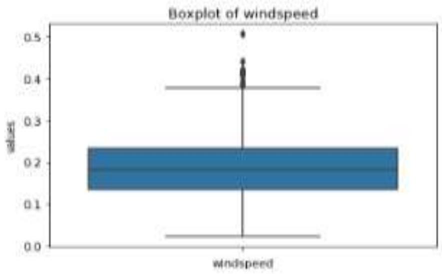

```
atemp
```



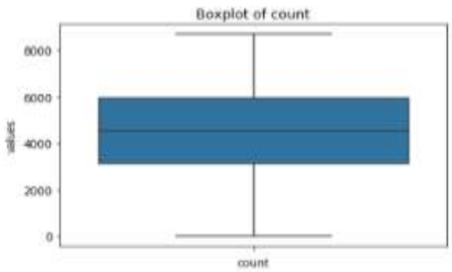

```
humidity
```




```
windspeed
```




```
count
```



from boxplot it is clear that two variables humidity and windspeed having outliers.

In [13]:
```python
#calculate iqr, lower fence and upper fence-
for i in cnames:
    print(i)
    q75,q25= np.percentile(data.loc[:,i],[75,25])
    iqr= q75-q25
    minimum= q25-(iqr*1.5)
    maximum= q75+(iqr*1.5)
    print("min= "+str(minimum))
    print("max= "+str(maximum))
    print("IQR= "+str(iqr))

#replace outliers with NA-
    data.loc[data[i]<minimum,i]=np.nan
    data.loc[data[i]>maximum,i]=np.nan
```

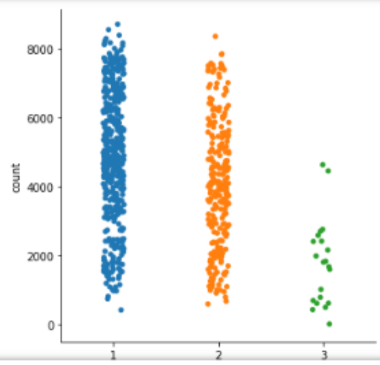
```
temprature
min= -0.14041600000000015
max= 1.1329160000000003
IQR= 0.3183330000000001
atemp
min= -0.06829675000000018
max= 1.0147412500000002
IQR= 0.2707595000000001
humidity
min= 0.20468725
max= 1.0455212500000002
IQR= 0.21020850000000002
windspeed
min= -0.012446750000000034
max= 0.38061125
IQR= 0.0982645
count
min= -1054.0
max= 10162.0
IQR= 2804.0
```

In [14]:
```python
#impute NA with median-
data['humidity']=data['humidity'].fillna(data['humidity'].median())
data['windspeed']=data['windspeed'].fillna(data['windspeed'].median())

#check NA in data-
print(data.isnull().sum())
```

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

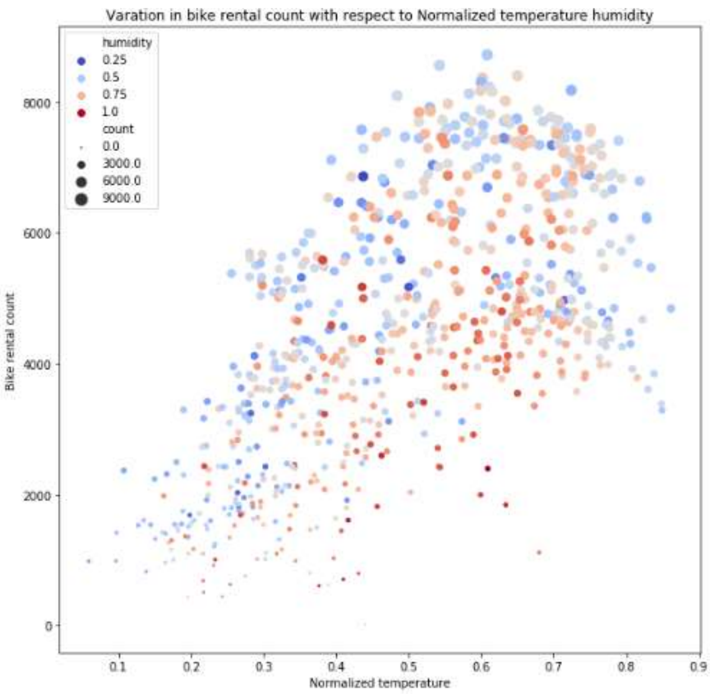
Data Understanding-

In [15]:
```python
for i in cat_cnames:
    sns.catplot(x=i,y="count",data=data)
    fname = str(i)+'.pdf'
    plt.savefig(fname)
```



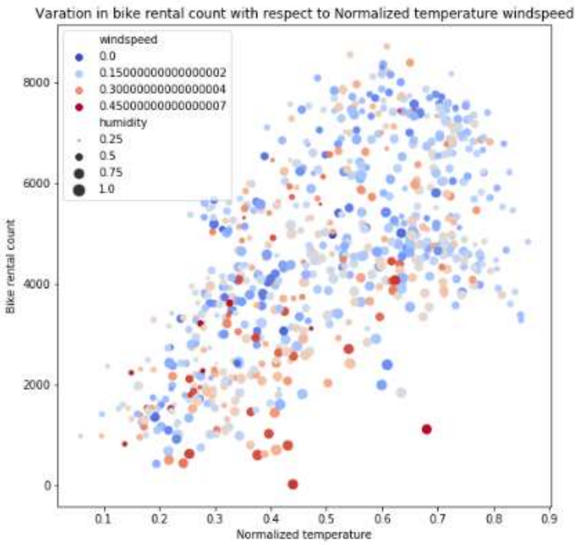
From Fisrt plot we can see that season 2,3 and 4 have more bike count as comapre to season 1. the daily bike count for these season was between 4000 to 8000. From year plot we can see that bike count is increased in 2012 as compared to 2011. *From month plot we can see the bike count maximum between 4 to 10 month.* From holiday the bike count is maximum as comapre to non holiday. *Bike count is maximum for day 0,5 and 6 as per weekday varaible.* FOr weather 1 the count of bike is maximum, after that for weather 2.

In [16]:
```python
f, ax = plt.subplots(figsize=(10, 10))
sns.scatterplot(x="temprature", y="count",
                hue="humidity", size="count",
                palette="coolwarm",sizes=(1, 100), linewidth=0,
                data=data,ax=ax)
plt.title("Varation in bike rental count with respect to Normalized temperature humidity")
plt.ylabel("Bike rental count")
plt.xlabel("Normalized temperature")
plt.savefig('bike_temp&humidity_plot.pdf')
```



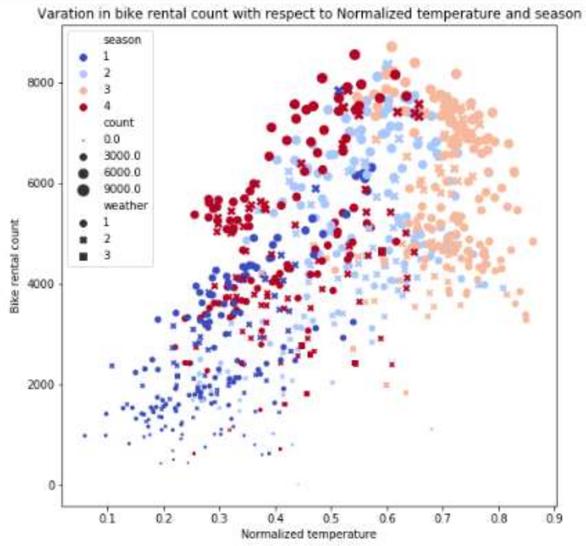
*From the plot we can see that count is maximum when temprature 0.4 to 0.7 and humidity below 0.75.

In [17]:
```
f, ax = plt.subplots(figsize=(8, 8))
sns.scatterplot(x="temprature", y="count",
                hue="windspeed", size="humidity",
                palette="coolwarm",sizes=(1, 100), linewidth=0,
                data=data,ax=ax)
plt.title("Varation in bike rental count with respect to Normalized temperature windspeed")
plt.ylabel("Bike rental count")
plt.xlabel("Normalized temperature")
plt.savefig('bike_temp&windspeed_plot.pdf')
```



*From the above plot we can see bike count is maximum between temp 0.5 to 0.7, windspped below 0.15 and humidity less than 0.75

In [18]:
```python
f, ax = plt.subplots(figsize=(8, 8))
sns.scatterplot(x="temprature", y="count",
                hue="season", size="count",style= "weather",
                palette="coolwarm",sizes=(1, 100), linewidth=0,
                data=data,ax=ax)
plt.title("Varation in bike rental count with respect to Normalized temperature 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 0.5 to 0.7, and weather was 1 and 2

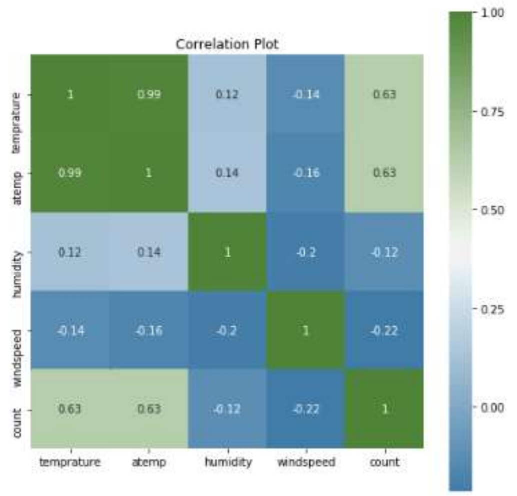
Feature Selection-

In [19]:
```python
#correlation analysis for numeric varialles-

#extract only numeric variables in dataframe for correlation-
df_corr= data.loc[:,cnames]

#generate correlation matrix-
corr_matrix= df_corr.corr()
(print(corr_matrix))
```

```
             temprature     atemp  humidity  windspeed     count
temprature     1.000000  0.991702  0.123723  -0.138937  0.627494
atemp          0.991702  1.000000  0.137312  -0.164157  0.631066
humidity       0.123723  0.137312  1.000000  -0.200237 -0.121454
windspeed     -0.138937 -0.164157 -0.200237   1.000000 -0.215203
count          0.627494  0.631066 -0.121454  -0.215203  1.000000
```

In [20]:
```python
#correlation plot-
f,ax= plt.subplots(figsize=(8,8))

#plot-
sns.heatmap(corr_matrix,mask=np.zeros_like(corr_matrix,dtype=np.bool),cmap=sns.diverging_palette(240,120,as_cmap=True),
            square=True,ax=ax,annot=True)
plt.title("Correlation Plot")
```

Out[20]: Text(0.5,1,'Correlation Plot')



From correlation plot we can see temprature and atemp are highly correlated, so we can remove atemp variable under dimension redution.

In [22]:
```
#Anova test for categorical predictor and numeric target variable-
import statsmodels.api as sm
from statsmodels.formula.api import ols

label = 'count'
for  i in cat_cnames:
    frame = label + ' ~ ' + i
    model = ols(frame,data=data).fit()
    anova = sm.stats.anova_lm(model, typ=2)
    print(anova)
```

```
                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 anova test we can see varaibles- holiday,weekday,and workingday have pr>0.05, so we can drop them in dimension reduction.

In [23]:
```
#Dimension Reduction-
data= data.drop(["atemp","holiday","weekday","workingday"],axis=1)

print(data.shape)
```

```
(731, 8)
```

In [25]:
```
data.head()
```

Out[25]:

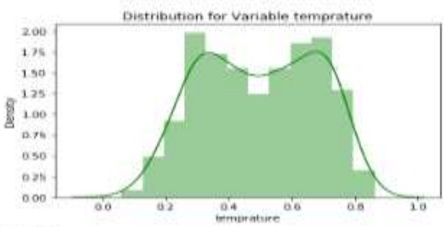
	season	year	month	weather	temprature	humidity	windspeed	count
0	1	0	1	2	0.344167	0.805833	0.160446	985.0
1	1	0	1	2	0.363478	0.696087	0.248539	801.0
2	1	0	1	1	0.196364	0.437273	0.248309	1349.0
3	1	0	1	1	0.200000	0.590435	0.160296	1562.0
4	1	0	1	1	0.226957	0.436957	0.186900	1600.0

In [26]:
```
#update numeric and categorical variable after dimension reduction-

#continuous variable-
cnames= ['temprature','humidity', 'windspeed', 'count']

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

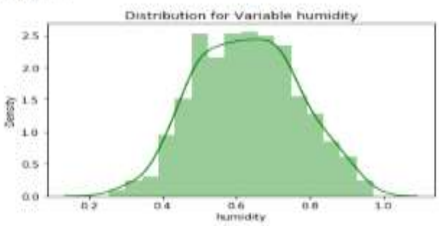
Feature Scaling-

In [27]:
```
#distribution check to check data is uniformly distributed or not-

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

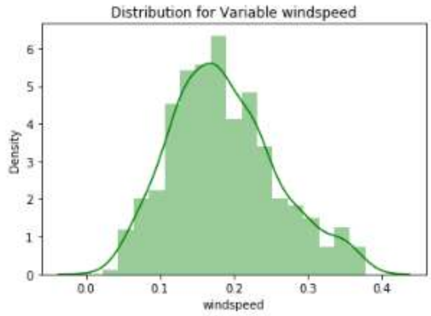
temprature



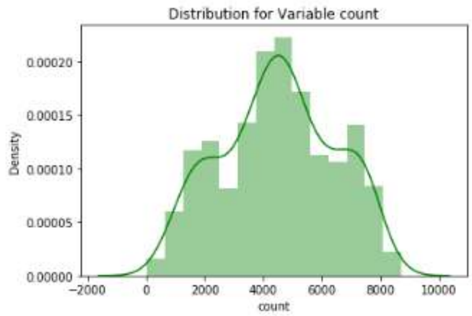
humidity



windspeed



count



from distribution plot it is clear that data is already normalized.

In [28]:
```
data.describe()
```

Out[28]:

	season	year	month	weather	temprature	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	0.629354	0.186257	4504.348837
std	1.110807	0.500342	3.451913	0.544894	0.183051	0.139566	0.071156	1937.211452
min	1.000000	0.000000	1.000000	1.000000	0.059130	0.254167	0.022392	22.000000
25%	2.000000	0.000000	4.000000	1.000000	0.337083	0.522291	0.134950	3152.000000
50%	3.000000	1.000000	7.000000	1.000000	0.498333	0.627500	0.178802	4548.000000
75%	3.000000	1.000000	10.000000	2.000000	0.655417	0.730209	0.229786	5956.000000
max	4.000000	1.000000	12.000000	3.000000	0.861667	0.972500	0.378108	8714.000000

Here, we can see data all numeric varaibles are already normalized, so we do not need to scale them.

Machine Learning Model Devlopment-

Train-Test Split-

In [151]:
```
#df=data
data=df
```

In [152]:
```
#create categorical variables to dummy variables-
data= pd.get_dummies(data,columns=cat_cnames)

data.shape
```

Out[152]: (731, 25)

In [98]:
```
data.head()
```

Out[98]:

	temprature	humidity	windspeed	count	season_1	season_2	season_3	season_4	year_0	year_1	...	month_6	month_7	month_8	month_9	month_10
0	0.344167	0.805833	0.160446	985.0	1	0	0	0	1	0	...	0	0	0	0	0
1	0.363478	0.696087	0.248539	801.0	1	0	0	0	1	0	...	0	0	0	0	0
2	0.196364	0.437273	0.248309	1349.0	1	0	0	0	1	0	...	0	0	0	0	0
3	0.200000	0.590435	0.160296	1562.0	1	0	0	0	1	0	...	0	0	0	0	0
4	0.226957	0.436957	0.186900	1600.0	1	0	0	0	1	0	...	0	0	0	0	0

5 rows × 25 columns

In [99]:
```
#Import libraries-
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import train_test_split
from sklearn.metrics import mean_squared_error
```

In [153]:
```
#error metrics function-
def MAPE(y_true,y_prediction):
    mape= np.mean(np.abs(y_true-y_prediction)/y_true)*100
    return mape
```

In [154]:
```
#split data for predictor and target seperatly-
X= data.drop(['count'],axis=1)
y= data['count']
```

In [155]:
```
#divide data into train and test part-
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=.20,random_state=0)
```

Decision Tree Model-

In [156]:
```
#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)

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))
```

```
Mean Absolute Precentage Error for train data=62.26013293672567
Mean Absolute Precentage Error for test data=36.94809301452646
R^2_score for train data=0.6775629218593628
R^2_score for test data=0.6464697716428666
```

In [157]:
```
df1= {'Model Name': ['Decision Tree'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],
      'R-squared_Test':[r2_test]}
result1= pd.DataFrame(df1)
```

Random Search CV in Decision Tree-

In [159]:
```
#import Libraries-
from sklearn.model_selection import RandomizedSearchCV

RandomDecisionTree = DecisionTreeRegressor(random_state = 0)
depth = list(range(1,20,2))
random_search = {'max_depth': depth}

#Random Decision Tree model-
RDT_model= RandomizedSearchCV(RandomDecisionTree,param_distributions= random_search,n_iter=5,cv=5)
RDT_model= RDT_model.fit(X_train,y_train)

#Best parameters for model-
best_parameters = RDT_model.best_params_

#Best model-
best_model = RDT_model.best_estimator_

#Model prediction on train data-
RDT_train = best_model.predict(X_train)

#Model prediction on test data-
RDT_test = best_model.predict(X_test)

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

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

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

#r2 value for test data-
r2_test=r2_score(y_test,RDT_test)
print("Best Parameter="+str(best_parameters))
print("Best Model="+str(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))
```

```
Best Parameter={'max_depth': 5}
Best Model=DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=0, splitter='best')
Mean Absolute Precentage Error for train data=14.180789128346541
Mean Absolute Precentage Error for test data=23.419815797374792
R^2_score for train data=0.8744351993110204
R^2_score for test data=0.8093605658476156
```

In [160]:
```
df2= {'Model Name': ['Decision Tree Random Search CV'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_tr
      'R-squared_Test':[r2_test]}
result2= pd.DataFrame(df2)
```

In [161]:
```
result= result1.append(result2)
```

Grid Search CV in Decision Tree-

In [162]:
```
#import Libraries-
from sklearn.model_selection import GridSearchCV

GridDecisionTree= DecisionTreeRegressor(random_state=0)
depth= list(range(1,20,2))
grid_search= {'max_depth':depth}

#Grid Decision Tree model-
GDT_model= GridSearchCV(GridDecisionTree,param_grid=grid_search,cv=5)
GDT_model= GDT_model.fit(X_train,y_train)
```

```
#Best parameters for model-
best_parameters = GDT_model.best_params_

#Best model-
best_model = GDT_model.best_estimator_

#Model prediction on train data-
GDT_train = best_model.predict(X_train)

#Model prediction on test data-
GDT_test = best_model.predict(X_test)

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

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

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

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

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

```
Best Parameter={'max_depth': 5}
Best Model=DecisionTreeRegressor(criterion='mse', max_depth=5, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=0, splitter='best')
Mean Absolute Precentage Error for train data=14.180789128346541
Mean Absolute Precentage Error for test data=23.419815797374792
R^2_score for train data=0.8744351993110204
     R^2_score for test data=0.8093605658476156
```

In [163]:
```
df3= {'Model Name': ['Decision Tree Grid Search CV'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_trai
      'R-squared_Test':[r2_test]}
result3= pd.DataFrame(df3)
```

In [164]:
```
result= result.append(result3)
```

Random Forest Model-

In [165]:
```
#import libraris-
from sklearn.ensemble import RandomForestRegressor

#Random Forest for regression-
RF_model= RandomForestRegressor(n_estimators=100).fit(X_train,y_train)

#model prediction on train data-
RF_train= RF_model.predict(X_train)

#model prediction on test data-
RF_test= RF_model.predict(X_test)

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

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

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

#r2 value for test data-
r2_test=r2_score(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))
```

```
Mean Absolute Precentage Error for train data=16.77699090980825
Mean Absolute Precentage Error for test data=20.42606722936694
R^2_score for train data=0.9791775521075736
R^2_score for test data=0.8818011050780997
```

In [166]:
```
df4= {'Model Name': ['Random Forest'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],
      'R-squared_Test':[r2_test]}
result4= pd.DataFrame(df4)
```

In [167]:
```
result= result.append(result4)
```

Random Search CV in Random Forest-

In [168]:
```
#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}

#Random Random Forest model-
RRF_model= RandomizedSearchCV(RandomRandomForest,param_distributions= random_search,n_iter=5,cv=5)
RRF_model= RRF_model.fit(X_train,y_train)

#Best parameters for model-
best_parameters = RRF_model.best_params_
```

```
#Best model-
best_model = RRF_model.best_estimator_

#Model prediction on train data-
RRF_train = best_model.predict(X_train)

#Model prediction on test data-
RRF_test = best_model.predict(X_test)

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

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

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

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

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

```
Best Parameter={'n_estimators': 81, 'max_depth': 15}
Best Model=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=15,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=81, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
Mean Absolute Precentage Error for train data=21.445350458942634
Mean Absolute Precentage Error for test data=21.02935544953941
R^2_score for train data=0.9782192685232404
R^2_score for test data=0.8789293277536785
```

In [170]:
```
df5= {'Model Name': ['Random Forest Random Search CV'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_tr
      'R-squared_Test':[r2_test]}
result5= pd.DataFrame(df5)
```

In [171]:
```
result= result.append(result5)
```

Grid Search CV in Random Forest-

In [172]:
```
#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}

#Grid Random Forest model-
GRF_model= GridSearchCV(GridRandomForest,param_grid=grid_search,cv=5)
GRF_model= GRF_model.fit(X_train,y_train)

#Best parameters for model-
best_parameters = GRF_model.best_params_

#Best model-
best_model = GRF_model.best_estimator_

#Model prediction on train data-
GRF_train = best_model.predict(X_train)

#Model prediction on test data-
GRF_test = best_model.predict(X_test)

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

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

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

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

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

```
Best Parameter={'max_depth': 9, 'n_estimators': 17}
Best Model=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=9,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=17, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
Mean Absolute Precentage Error for train data=21.320741945462775
Mean Absolute Precentage Error for test data=20.567325463456854
R^2_score for train data=0.9648262675118173
R^2_score for test data=0.8753352940976121
```

In [173]:
```
df6= {'Model Name': ['Random Forest Grid Search CV'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_trai
      'R-squared_Test':[r2_test]}
result6= pd.DataFrame(df6)
```

In [174]:
```
result= result.append(result6)
```

Linear Regression Model-

In [175]:
```python
#import libraries-
import statsmodels.api as sm

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

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  count   R-squared:                       0.833
Model:                            OLS   Adj. R-squared:                  0.827
Method:                 Least Squares   F-statistic:                     140.2
Date:                Sat, 06 Apr 2019   Prob (F-statistic):          1.63e-203
Time:                        17:25:53   Log-Likelihood:                -4716.2
No. Observations:                 584   AIC:                             9474.
Df Residuals:                     563   BIC:                             9566.
Df Model:                          20
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
temprature  4807.6605    477.418     10.070      0.000    3869.923    5745.398
humidity   -1840.0359    351.762     -5.231      0.000   -2530.963   -1149.109
windspeed  -2692.7145    509.781     -5.282      0.000   -3694.019   -1691.410
season_1    -160.8963    149.431     -1.077      0.282    -454.407     132.615
season_2     735.4147    149.261      4.927      0.000     442.239    1028.591
season_3     756.5640    170.170      4.446      0.000     422.319    1090.809
season_4    1424.2811    170.259      8.365      0.000    1089.860    1758.702
year_0       409.9681    152.821      2.683      0.008     109.799     710.137
year_1      2345.3954    151.325     15.499      0.000    2048.166    2642.625
month_1       -1.9341    197.841     -0.010      0.992    -390.531     386.663
month_2       45.1383    186.947      0.241      0.809    -322.060     412.337
month_3      510.8770    141.897      3.600      0.000     232.166     789.588
month_4      233.3586    174.311      1.339      0.181    -109.021     575.738
month_5      659.7195    183.392      3.597      0.000     299.503    1019.936
month_6      250.5066    180.098      1.391      0.165    -103.239     604.252
month_7     -222.2685    220.988     -1.006      0.315    -656.331     211.794
month_8      271.1265    207.045      1.310      0.191    -135.548     677.801
month_9      888.8861    173.978      5.109      0.000     547.161    1230.611
month_10     382.5832    187.383      2.042      0.042      14.528     750.639
month_11    -183.6576    194.752     -0.943      0.346    -566.188     198.873
month_12     -78.9721    168.303     -0.469      0.639    -409.550     251.606
weather_1   1643.7280     90.978     18.067      0.000    1465.030    1822.426
weather_2   1302.9232    110.447     11.797      0.000    1085.985    1519.862
weather_3   -191.2876    221.771     -0.863      0.389    -626.886     244.311
==============================================================================
Omnibus:                       97.249   Durbin-Watson:                   1.897
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              248.035
Skew:                          -0.849   Prob(JB):                     1.38e-54
Kurtosis:                       5.704   Cond. No.                     1.46e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.57e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [176]:
```python
#model prediction 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)
```

```python
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))
```

```
Mean Absolute Precentage Error for train data=44.444512312552895
Mean Absolute Precentage Error for test data=18.800696038206944
R^2_score for train data=0.8327600660988469
R^2_score for test data=0.8411102698142885
```

In [177]:
```python
df7= {'Model Name': ['Linear Regression'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],
      'R-squared_Test':[r2_test]}
result7= pd.DataFrame(df7)
```

In [178]:
```python
result= result.append(result7)
```

Gradient Boosting Model-

In [179]:
```python
#import libraries-
from sklearn.ensemble import GradientBoostingRegressor

#Gradient Boosting for regression-
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)

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

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

```
Mean Absolute Precentage Error for train data=44.444512312552895
Mean Absolute Precentage Error for test data=19.899340749272547
R^2_score for train data=0.9453851786306906
R^2_score for test data=0.8645945042011559
```

In [180]:
```python
df8= {'Model Name': ['Gradient Boosting'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],
      'R-squared_Test':[r2_test]}
result8= pd.DataFrame(df8)
```

In [181]:
```python
result= result.append(result8)
```

Random Search CV in Gradient Boosting-

In [182]:
```python
#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}

#Random Gradient Boosting model-
RGB_model= RandomizedSearchCV(RandomGradientBoosting,param_distributions= random_search,n_iter=5,cv=5)
RGB_model= RGB_model.fit(X_train,y_train)

#Best parameters for model-
best_parameters = RGB_model.best_params_

#Best model-
best_model = RGB_model.best_estimator_
```

```python
#Model prediction on train data-
RGB_train = best_model.predict(X_train)

#Model prediction on test data-
RGB_test = best_model.predict(X_test)

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

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

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

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

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

```
Best Parameter={'n_estimators': 73, 'max_depth': 7}
Best Model=GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=7, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=73, presort='auto', random_state=0,
             subsample=1.0, verbose=0, warm_start=False)
Mean Absolute Precentage Error for train data=1.732620030029962
Mean Absolute Precentage Error for test data=21.73009586735038
R^2_score for train data=0.9982358399869793
R^2_score for test data=0.8665491553293638
```

In [183]:
```python
df9= {'Model Name': ['Gradient Boosting Random Search CV'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],
     'R-squared_Train':[r2_train],'R-squared_Test':[r2_test]}
result9= pd.DataFrame(df9)
```

In [184]:
```python
result= result.append(result9)
```

Grid Search CV in Gradient Boosting-

In [185]:
```python
#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}

#Grid Random Forest model-
GGB_model= GridSearchCV(GridGradientBoosting,param_grid=grid_search,cv=5)
GGB_model= GGB_model.fit(X_train,y_train)

#Best parameters for model-
best_parameters = GGB_model.best_params_

#Best model-
best_model = GGB_model.best_estimator_

#Model prediction on train data-
GGB_train = best_model.predict(X_train)

#Model prediction on test data-
GGB_test = best_model.predict(X_test)

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

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

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

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

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

```
Best Parameter={'max_depth': 5, 'n_estimators': 19}
Best Model=GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=5, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=19, presort='auto', random_state=0,
             subsample=1.0, verbose=0, warm_start=False)
Mean Absolute Precentage Error for train data=18.833447720956062
Mean Absolute Precentage Error for test data=25.485646188467083
R^2_score for train data=0.9221138936676011
R^2_score for test data=0.8337462222690681
```

In [186]:
```
df10= {'Model Name': ['Gradient Boosting Grid Search CV'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],
       'R-squared_Train':[r2_train],'R-squared_Test':[r2_test]}
result10= pd.DataFrame(df10)
```

In [187]:
```
result= result.append(result10)
```

In [188]:
```
result= result.reset_index(drop=True)
```

In [189]:
```
#Final result of all the model with MAPE and r-Squared-
result
```

Out[189]:

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test
0	Decision Tree	62.260133	36.948093	0.677563	0.646470
1	Decision Tree Random Search CV	14.180789	23.419816	0.874435	0.809361
2	Decision Tree Grid Search CV	14.180789	23.419816	0.874435	0.809361
3	Random Forest	16.776997	20.426067	0.979178	0.881801
4	Random Forest Random Search CV	21.445350	21.029355	0.978219	0.878929
5	Random Forest Grid Search CV	21.320742	20.567325	0.964826	0.875335
6	Linear Regression	44.444512	18.800696	0.832780	0.841110
7	Gradient Boosting	44.444512	19.899341	0.945385	0.864595
8	Gradient Boosting Random Search CV	1.732620	21.730096	0.998236	0.866549
9	Gradient Boosting Grid Search CV	18.833448	25.485646	0.922114	0.833746

Thank You

References-

1. For Data Cleaning and Model Development - https://edwisor.com/career-data-scientist

2. For Visualization – https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/

THANK YOU