Bike Rental Count Prediction

Vinayaka O

23rd January 2020

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.

Sample dataset (top 5 Observation)

	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

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 from Holiday 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:

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; there- fore, 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. Ap- proximately, data exploration, cleaning and preparation can take up to 70% of our total project time. This process is often called as Exploratory Data Analysis

2.2 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.

```
(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
```

From EDA we have concluded that there are 7 continuous variables and 9 categorical variables 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 variable

```
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 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

2.3 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 regress on, classification 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.

	Variables	Missing_percentage
0	season	0.0
1	year	0.0
2	month	0.0
3	holiday	0.0
4	weekday	0.0
5	workingday	0.0
6	weather	0.0
7	temprature	0.0
8	atemp	0.0
9	humidity	0.0
10	windspeed	0.0
11	count	0.0

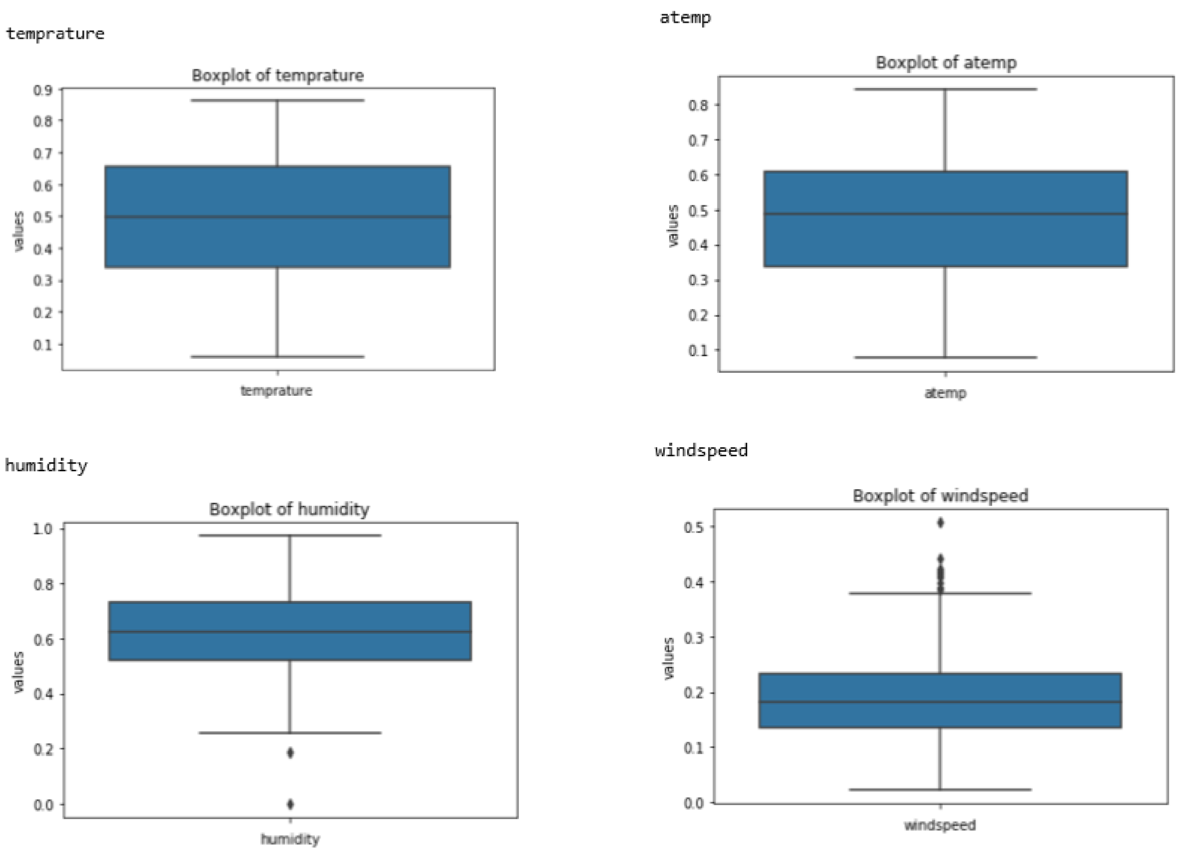
When come into our data set, observe the above picture there no missing value in our dataset, so no need impute missing values.

2.4 Outlier Analysis

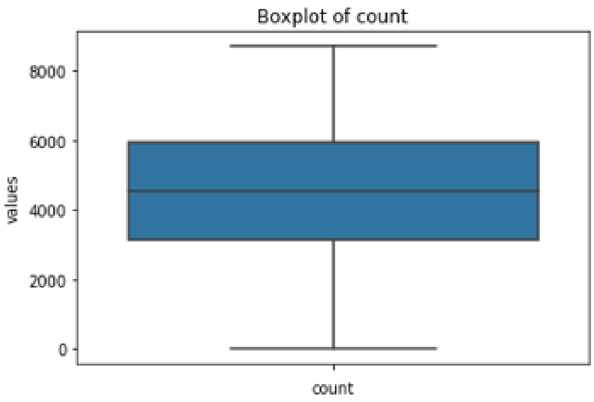
Outlier is an observation that appears far away and diverges from an overall pattern in a sample. These outliers occur in data due to many reasons like data entry errors, Measurement error, Experimental error, intentional errors etc.

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.

<u>Boxplot of continuous variables with outliers</u>



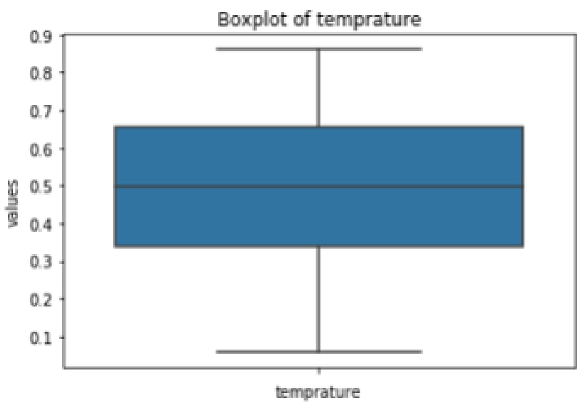
count



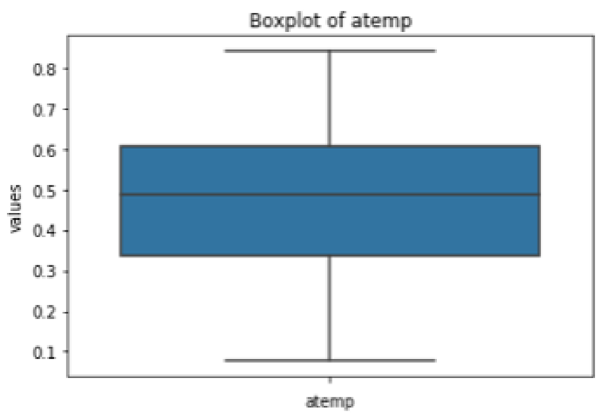
For our project we opted for capping method in which we are going to impute outlier with upper fence and lower fence value reason behind to opt this method is we don’t want to delete the observations with outliers as data collection is also a crucial step in data analytics for which client has to spend 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 treading or knn method or these outliers as missing values and impute them with mean or delete such observations.

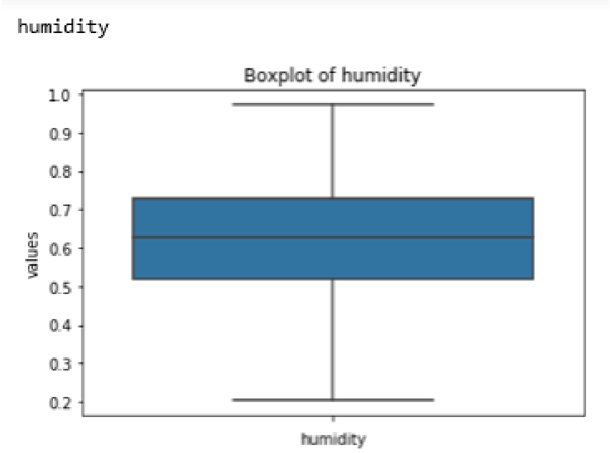
Boxplots of continuous variables after removing outliers

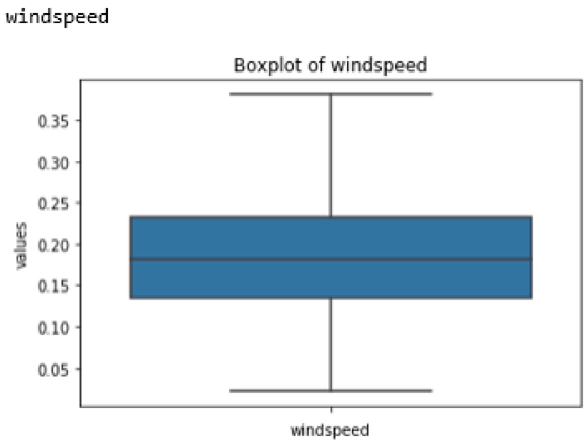
temprature

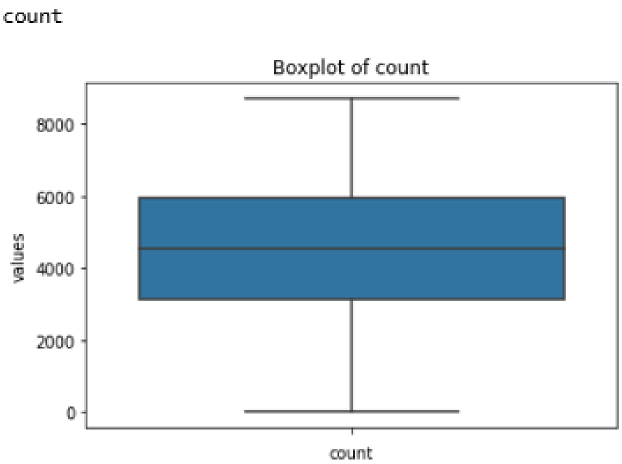


atemp





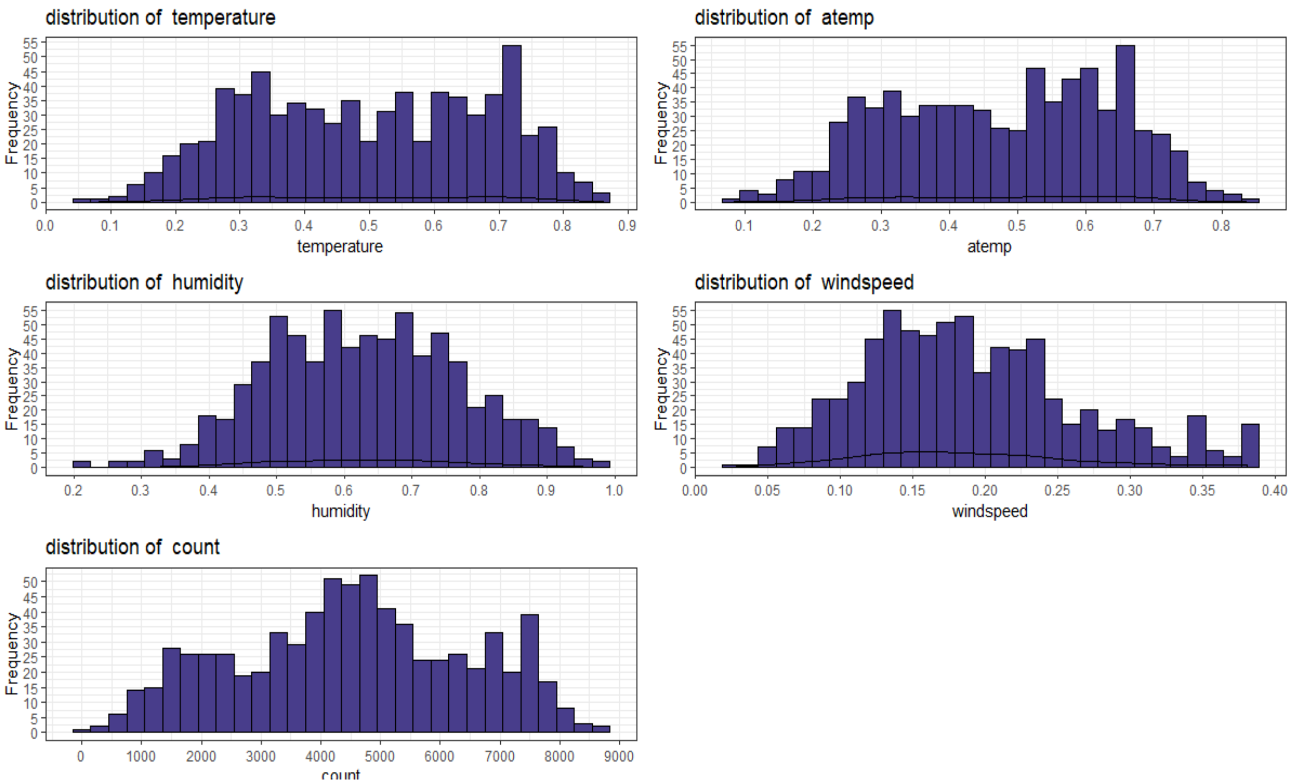




2.5 Data understanding using visualization

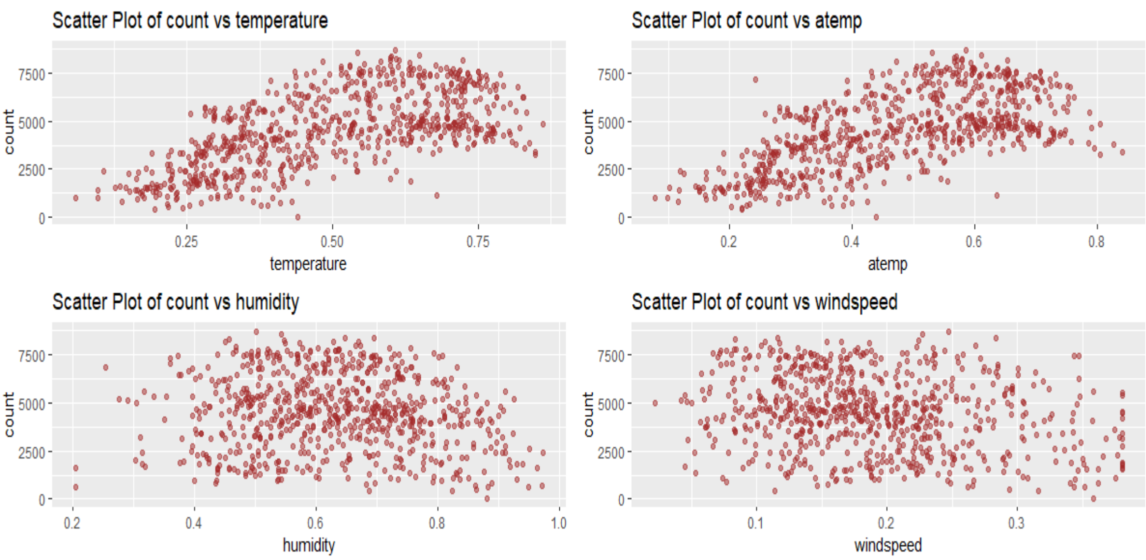
2.5.1 Distribution of continuous variables

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)



2.5.2 Effect of continuous variables wrt target variable

Let's check the impact of each continuous variable on target variable (count) using scatterplot. (In both R and Python)



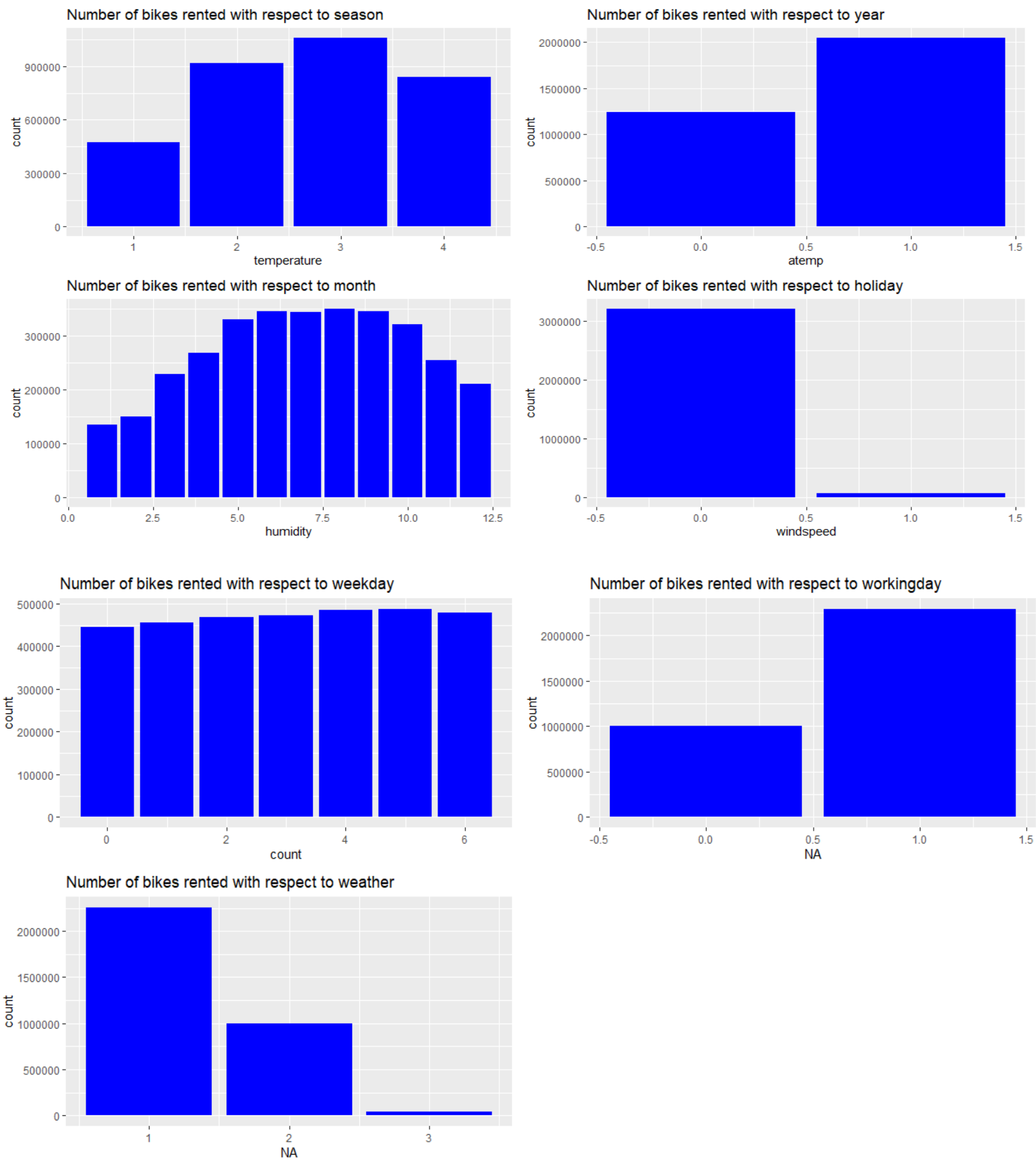
Count Vs Temperature: From the below plot we can say as temperature increase Bike rental count also increases.

Count Vs Humidity: From the below plot we can say humidity doesn't have any effect on bike rent count

Count Vs Windspeed: From the below plot we can say windspeed doesn't have any effect on bike rent count

2.5.3 Effect of categorical variables on target variable

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



Count VS Season: - Bike rent count is high in season 3(fall) and bike rent count is 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 i.e. 0

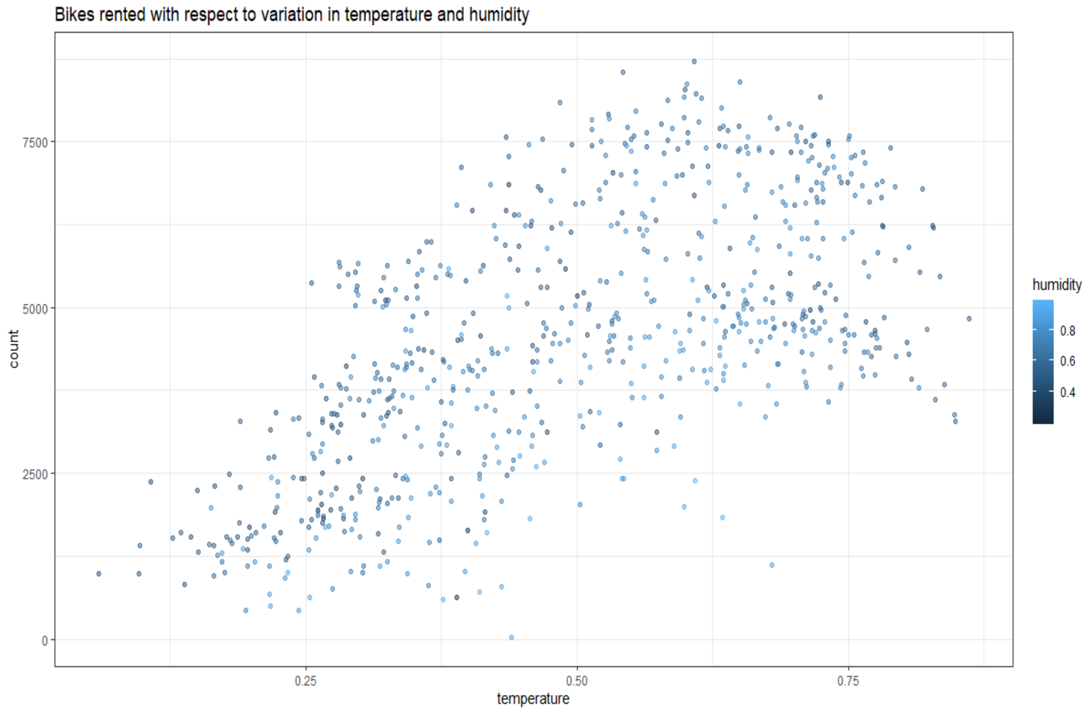
Count Vs weekday: - From bar plot we can see maximum bikes rented on 5th day and least bikes on day 0.

Count Vs working day: - Bike rent count is high on working day i.e. 1

Count Vs weather: - Bike rent count is high on weather 1 i.e. when the weather is Clear, Few clouds, partly cloudy, partly cloudy.

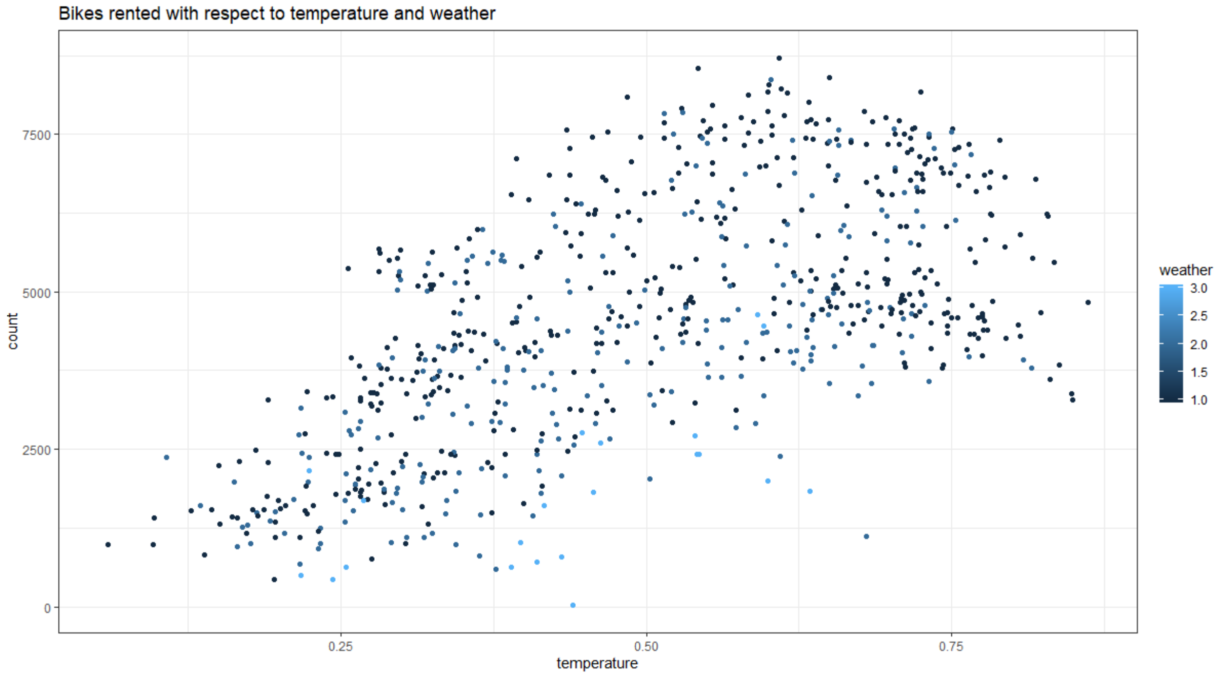
Bikes rented with respect to temp and humidity

Maximum bike rented between temp 0.50 to 0.75 and humidity 0.50 to 0.75

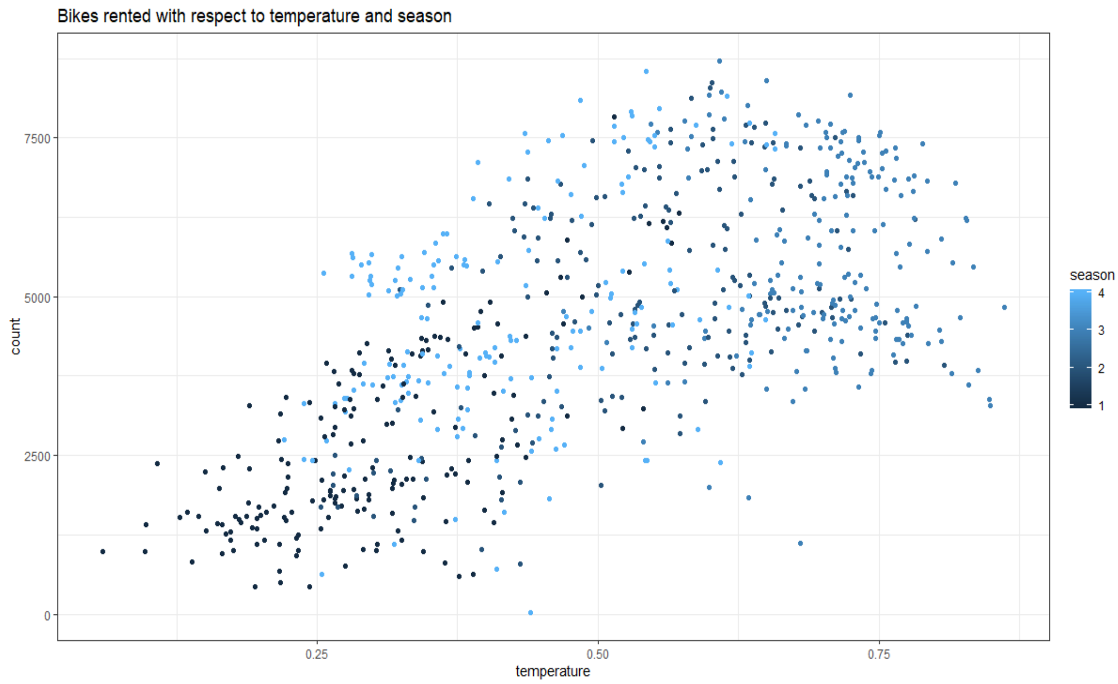


Bikes rented with respect to temp and windspeed:- maximum bike rented

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



2.6 Feature Engineering

Feature engineering is about **creating new input features** from our existing ones. In general, you can think have data cleaning as a process of subtraction and feature engineering as a process of addition.

This is often one of the most valuable tasks a data scientist can do to improve model performance, for 3 big reasons:

1. We can isolate and highlight key information, which helps your algorithms "focus" on what's important.
2. We can bring in our own **domain expertise**.
3. Most importantly, once we understand the "vocabulary" of feature engineering, can bring in other people's domain expertise

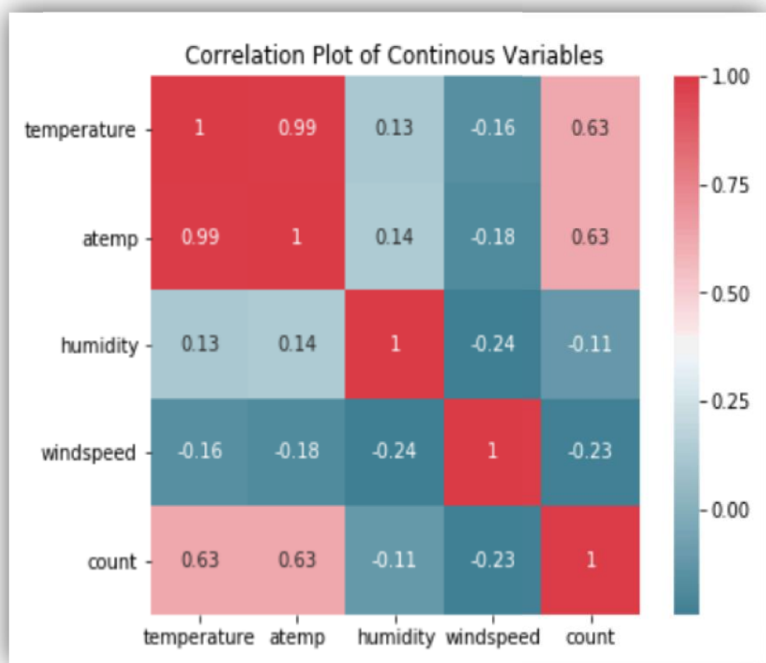
2.7 Feature Selection

Before performing any type of modelling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of 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 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 continues us variables



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

Result 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 continues variable and from ANOVA Analysis we found that in categorical variable Holyday, weekday, workingday have d P value >0.5. So we remove them in case of Categorical Variable.

After Correlation and ANOVA Analysis we have remaining variables are

Continuous variable in dataset

- temperature float64
- humidity float64
- windspeed float64
- count int64

Categorical variables in dataset

- season int6
- year int64
- month int64
- weather int64

Sample dataset after feature Selection

	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

2.8 Feature Scaling

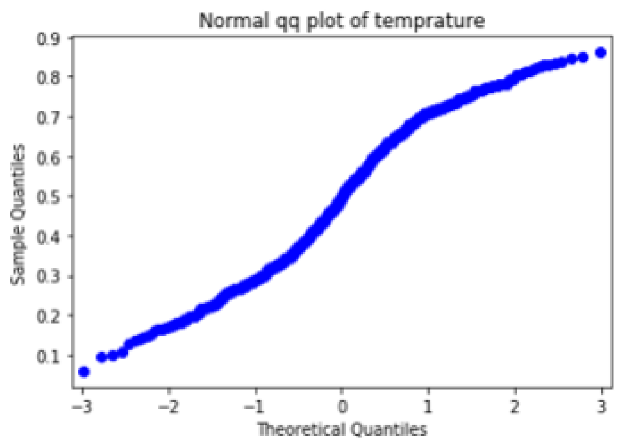
Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data pre-processing step. Since the range value of raw data varies widely, some machine learning algorithms, 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 no need to scale the data can check normality of data using normal qqplot, histogram plot and summary of plot.

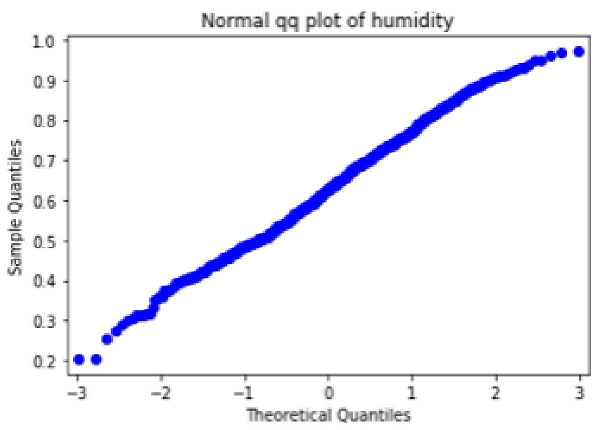
Since as it is mentioned in data dictionary the values of temp, humidity, windspeed variables are already normalized values so no need to go for feature scaling instead we will visualize the variables to see normality

Normal QQ Plots:

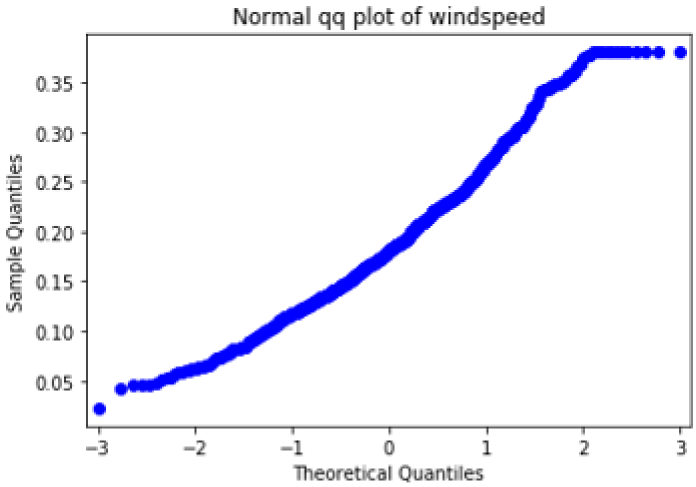
temprature



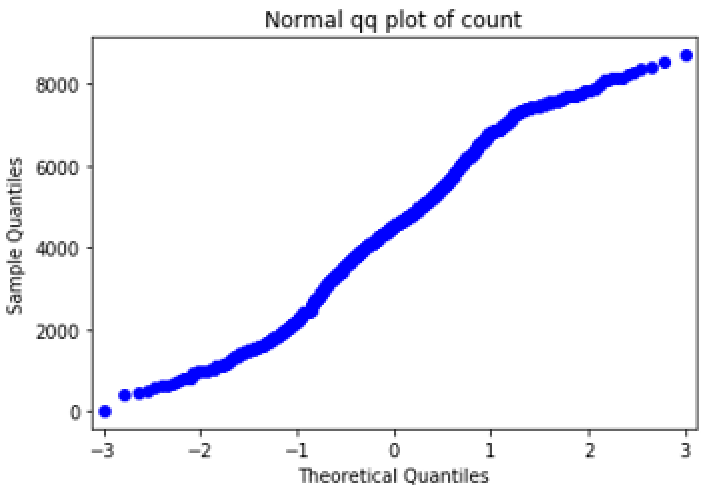
humidity



windspeed

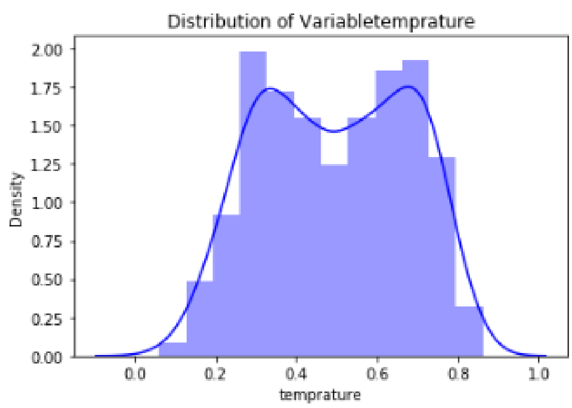


count

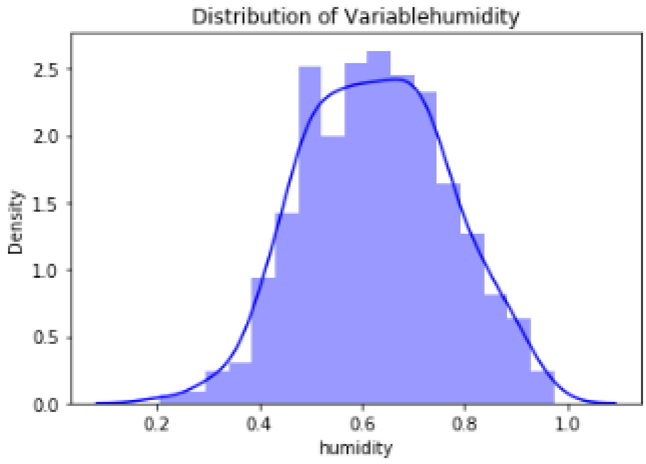


Histogram Plots:

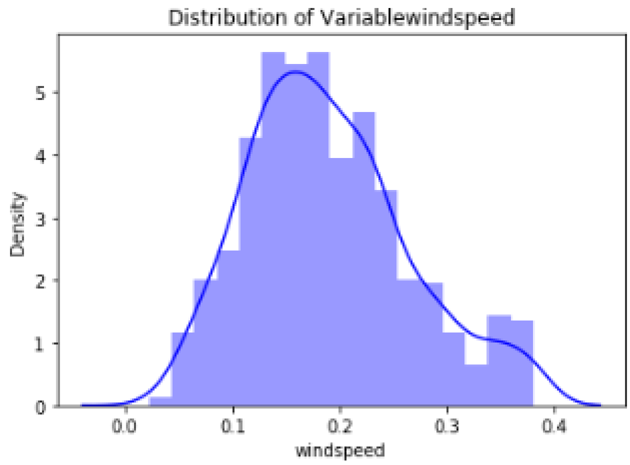
temprature



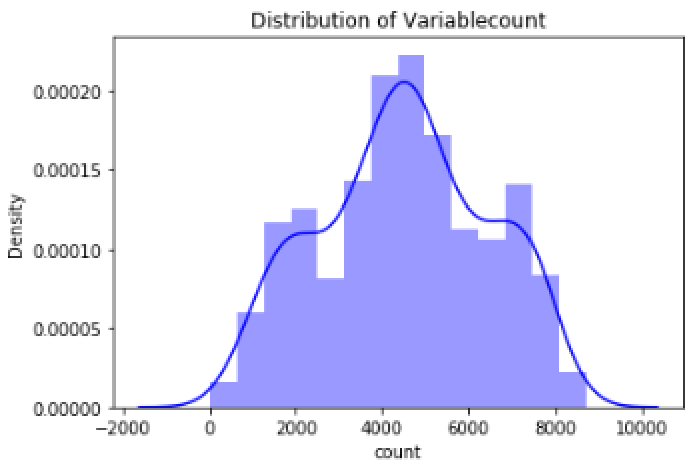
humidity



windspeed



count



Summary of each variable

	temprature	humidity	windspeed	count
count	731.000000	731.000000	731.000000	731.000000
mean	0.495385	0.628197	0.189846	4504.348837
std	0.183051	0.141320	0.075644	1937.211452
min	0.059130	0.204687	0.022392	22.000000
25%	0.337083	0.520000	0.134950	3152.000000
50%	0.498333	0.626667	0.180975	4548.000000
75%	0.655417	0.730209	0.233214	5956.000000
max	0.861667	0.972500	0.380611	8714.000000

2.9 Model Development

2.9.1 Model Selection

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.

2.9.2 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.

2.9.3 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.

2.9.4 Linear Regression

Linear Regression is one of the statistical methods 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 Regression.

3. Conclusion

In methodology we have done data cleaning and then applied 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, R square 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 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. **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 python:-

```
Final_Results
```

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Decision Tree	62.260133	36.948093	0.677583	0.646470	1080.381858	1226.219619
1	Random Forest	17.046471	20.787102	0.980456	0.879970	265.985987	714.495177
2	Linear Regression	43.773719	19.685303	0.837362	0.839261	767.301965	826.828014
3	Gradient Boosting	43.773719	18.430605	0.949290	0.869031	428.450622	746.344695

Final Result in R:-

	Model	MAPE_Train	MAPE_Test	Rsquare_Train	Rsquare_Test	Rmse_Train	Rmse_Test
1	Decision Tree for Regression	52.79857	19.93084	0.7961878	0.7583866	881.1670	923.3624
2	Random Forest	23.94825	13.99167	0.9683808	0.8580832	358.3914	706.3910
3	Linear Regression	44.63084	17.58232	0.8488937	0.7940854	758.7249	855.3072
4	Gradient Boosting	32.09771	14.83007	0.9091583	0.8665314	590.3581	683.0009

3.1.2 Model Selection

From the observation of all **MAPE, R-Squared** and **RMSE** Value we have concluded that,

Random Forest has minimum value of MAPE and its **R-Squared** Value is also maximum with RMSE**.** 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 differ a lot this implies that it is not the case of over fitting.

Insights about the Project:

- Temperature variable has major impact on bike rental count.
- In season 3 (fall September, October, November) there will be more bike rental counts.
- In the month of August bike rental count will be very high.
- Windspeed and humidity don’t have any impact on bike rental count and these two variables negative correlated with all variables.
- On Friday or Friday of week bike rental count is more.
- Even during working days bike rental count is more.

Appendix A – Python and R Codes

Python Code

Bike Rental

In [1]:

```
#importing required libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#import chi2_contigency for Chi square Test
from scipy import stats

from scipy.stats import chi2_contingency
from sklearn.ensemble import RandomForestClassifier
%matplotlib inline

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
#Set working directory
os.chdir("C:/Users/Hp/Desktop/Project2")
```

Data Pre-Processing

In [3]:

```
# reading data into dataframe
bike_rental= pd.read_csv("day.csv",sep=',')
```

In [4]:

```
bike_rental.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	1	0	1	0	0	0	2	0.363	0.353	0.696	0.24853	131	670	80

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
		-01-02								478	739	087	9			1
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

In [5]:

```
bike_rental.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 16 columns):
instant       731 non-null int64
dteday        731 non-null object
season        731 non-null int64
yr            731 non-null int64
mnth          731 non-null int64
holiday       731 non-null int64
weekday       731 non-null int64
workingday    731 non-null int64
weathersit    731 non-null int64
temp          731 non-null float64
atemp         731 non-null float64
hum           731 non-null float64
windspeed     731 non-null float64
casual        731 non-null int64
registered    731 non-null int64
cnt           731 non-null int64
dtypes: float64(4), int64(11), object(1)
memory usage: 91.5+ KB
```

In [6]:

```
bike_rental.shape
```

Out[6]:

```
(731, 16)
```

Exploratory Data Analysis¶

In [7]:

```
print(type(bike_rental))

print(bike_rental.shape)

print(bike_rental.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 [8]:

```
print(bike_rental.columns)

print(bike_rental.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 [9]:

```
#drop redudant variable

#drop instant variable

bike_rental=bike_rental.drop(['instant'],axis=1)


#drop dteday variable

bike_rental=bike_rental.drop(['dteday'],axis=1)


#drop casual variable

bike_rental=bike_rental.drop(['casual'],axis=1)


#drop registerd variable

bike_rental=bike_rental.drop(['registered'],axis=1)
```

In [10]:

```
print(bike_rental.shape)
```

```
(731, 12)
```

In [11]:

```
#rename variables in data set
```

```
bike_rental=bike_rental.rename(columns={'yr':'year','mnth':'month','weathersit':'we
ather','temp':'temprature','hum':'humidity',
                                        'cnt':'count'})


print(bike_rental.columns)
```

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

In [12]:

```
#seperate continues and categorical variable

#continues variable

cnames=['temprature','atemp','humidity','windspeed','count']
```

In [13]:

```
#categorical variable

cat_cnames = ['season','year','month','holiday','weekday','workingday','weather']
```

In [14]:

```
for i in cnames:

     print(bike_rental.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
```

Missing Value Analysis

In [15]:

```
#Create dataframe with missing percentage

missing_val = pd.DataFrame(bike_rental.isnull().sum())

missing_val = missing_val.reset_index()

missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missing_perce
ntage'})

missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(bike_ren
tal))*100

missing_val = missing_val.sort_values('Missing_percentage', ascending = False).rese
t_index(drop = True)

missing_val.to_csv("Miising_perc.csv", index = False)

missing_val
```

Out[15]:

	Variables	Missing_percentage
0	season	0.0
1	year	0.0
2	month	0.0
3	holiday	0.0
4	weekday	0.0
5	workingday	0.0
6	weather	0.0
7	temprature	0.0
8	atemp	0.0
9	humidity	0.0
10	windspeed	0.0
11	count	0.0

we concluded that from the above analysis, there is no missing valued variable in the data set that are given by the clients.

Outlier Analysis.

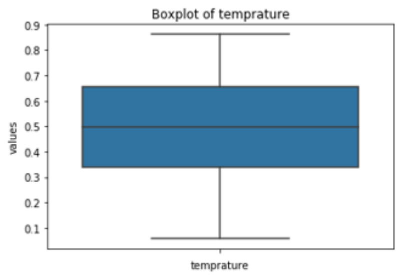
In [16]:

```
# Lets save copy of dataset before preprocessing

df = bike_rental.copy()

bike_rental = df.copy()
```

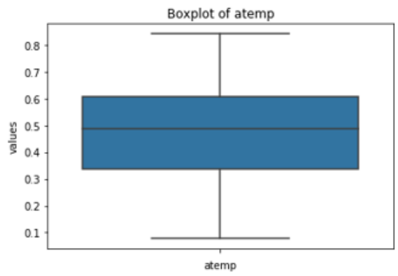
In [17]:

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


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

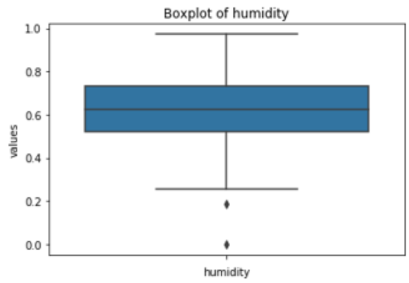
temprature



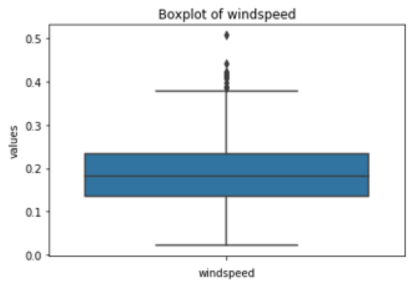
atemp



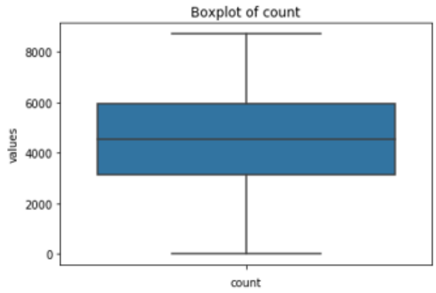
humidity



windspeed



count



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

In [18]:

```
# 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_rental[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_rental.loc[bike_rental[i] < LL ,i] = LL

   # For ioutliers
    bike_rental.loc[bike_rental[i] > UL ,i] = UL
```

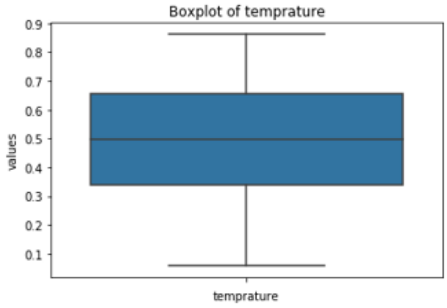
```
temprature
atemp
humidity
windspeed
count
```

In [19]:

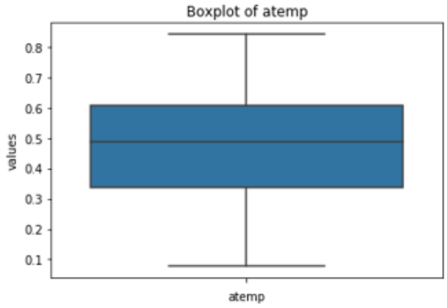
```
# Lets see our boxplots after removing outliers

for i in cnames:
    print(i)
    sns.boxplot(y=bike_rental[i])
    plt.xlabel(i)
    plt.ylabel("values")
```

```
    plt.title("Boxplot of "+i)
    plt.show()
```

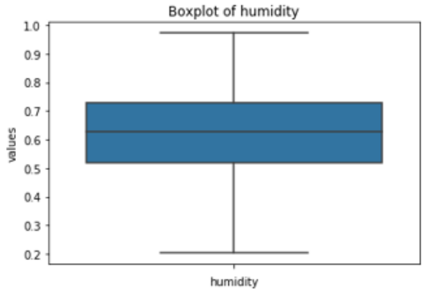
temprature



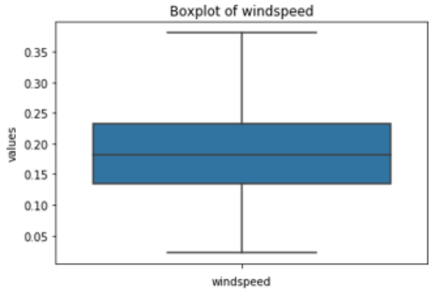
atemp



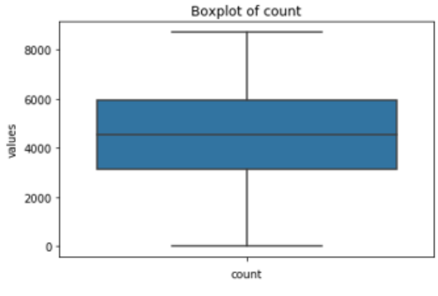
humidity



windspeed



count



Visualization

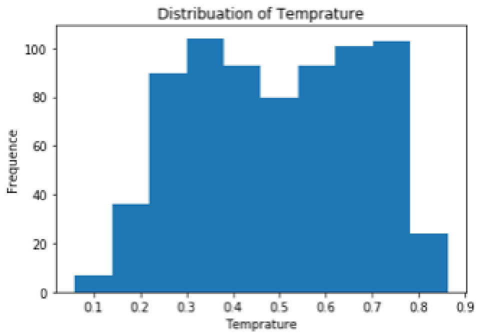
Univariate Analysis

In [20]:

```
# Histogram for all continuous variables to check distribution of each variable
# temperature
plt.hist(bike_rental['temprature'])
plt.xlabel("Temprature")
plt.ylabel("Frequence")
plt.title('Distribuation of Temprature')
```

Out[20]:

```
Text(0.5, 1.0, 'Distribuation of Temprature')
```



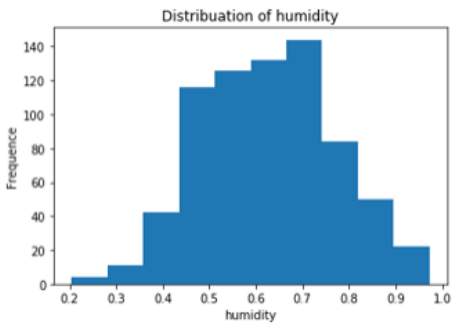
Normally distributed

In [21]:

```
#humidity
plt.hist(bike_rental['humidity'])
plt.xlabel("humidity")
plt.ylabel("Frequence")
plt.title('Distribuation of humidity')
```

Out[21]:

```
Text(0.5, 1.0, 'Distribuation of humidity')
```



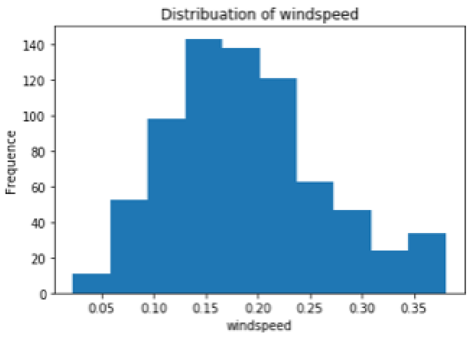
Normally distributed

In [22]:

```
#windspeed
plt.hist(bike_rental['windspeed'])
plt.xlabel("windspeed")
plt.ylabel("Frequence")
plt.title('Distribuation of windspeed')
```

Out[22]:

```
Text(0.5, 1.0, 'Distribuation of windspeed')
```



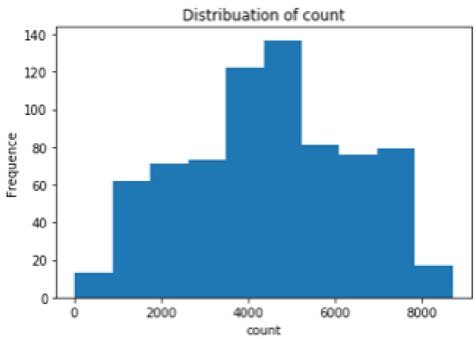
Normally distributed

In [23]:

```
#count
plt.hist(bike_rental['count'])
plt.xlabel("count")
plt.ylabel("Frequence")
plt.title('Distribuation of count')
```

Out[23]:

```
Text(0.5, 1.0, 'Distribuation of count')
```

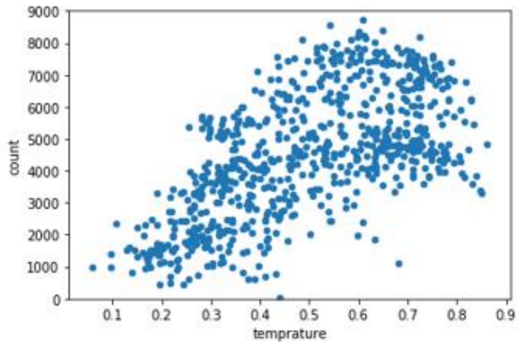


Normally distributed

Bivariate Analysis

In [24]:

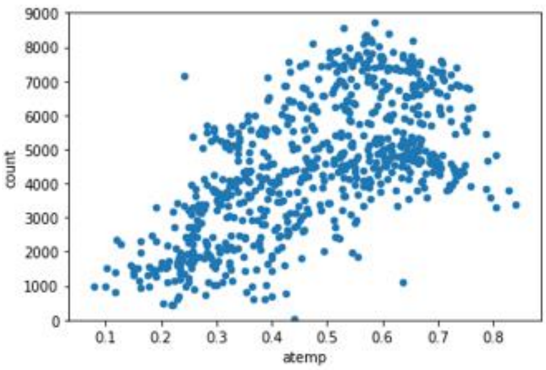
```
#relation between Numerical Variable 'temp' and target variable 'cnt'
bike_rental['temprature'].value_counts()

#Now draw scatter plot between 'temp' and 'cnt' variables
var = 'temprature'
data = pd.concat([bike_rental['count'], bike_rental[var]], axis=1)
data.plot.scatter(x=var, y='count', ylim=(0,9000));
```



As temperature increase Bike rent count also increases

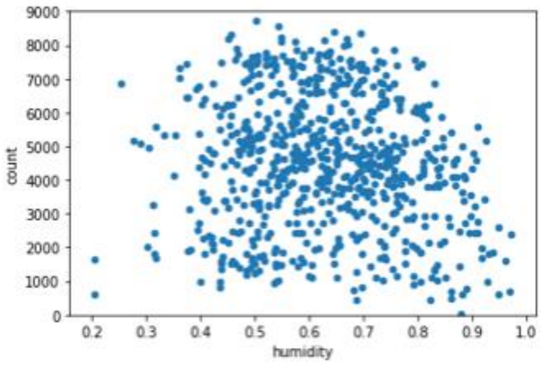
In [25]:

```
#relation between Numerical Variable 'atemp' and target variable 'cnt'
bike_rental['atemp'].value_counts()

#Now draw scatter plot between 'temp' and 'cnt' variables
var = 'atemp'
data = pd.concat([bike_rental['count'], bike_rental[var]], axis=1)
data.plot.scatter(x=var, y='count', ylim=(0,9000));
```



In [26]:

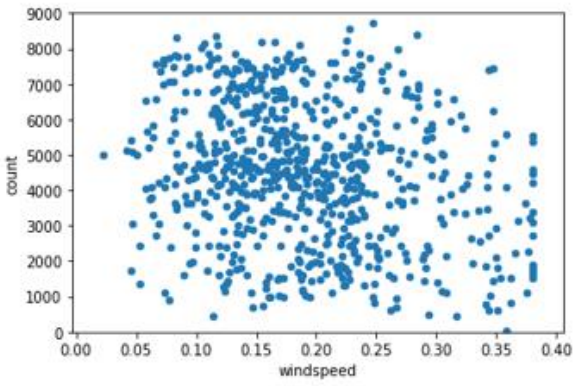
```
#relation between Numerical Variable 'hum' and target variable 'cnt'
bike_rental['humidity'].value_counts()

#Now draw scatter plot between 'hum' and 'cnt' variables
var = 'humidity'
data = pd.concat([bike_rental['count'], bike_rental[var]], axis=1)
data.plot.scatter(x=var, y='count', ylim=(0,9000));
```



humidity doesn't have any effect on bike rent count

In [27]:

```
#relation between Numerical Variable 'windspeed' and target variable 'cnt'
bike_rental['windspeed'].value_counts()

#Now draw scatter plot between 'windspeed' and 'cnt' variables
var = 'windspeed'
data = pd.concat([bike_rental['count'], bike_rental[var]], axis=1)
data.plot.scatter(x=var, y='count', ylim=(0,9000));
```

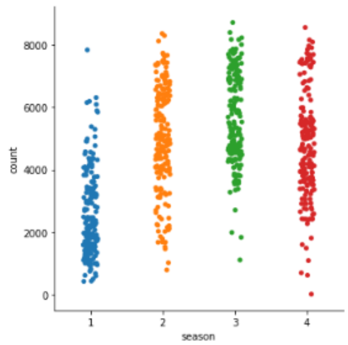


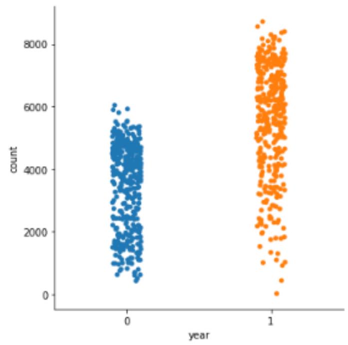
Windspeed doesn't have any effect on bike rent count

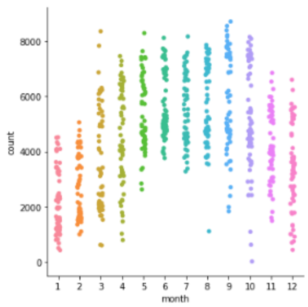
Data Visualization of categorical variable

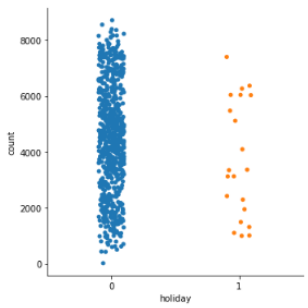
In [28]:

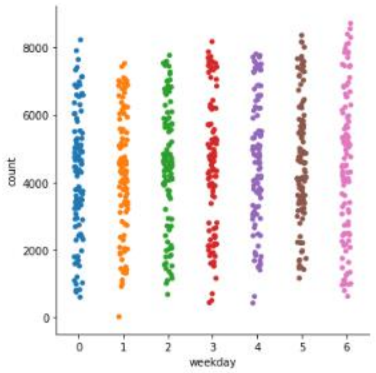
```
for i in cat_cnames:
    sns.catplot(x=i,y="count",data=bike_rental)
    fname=str(i)+'.pdf'
    plt.savefig(fname)
```

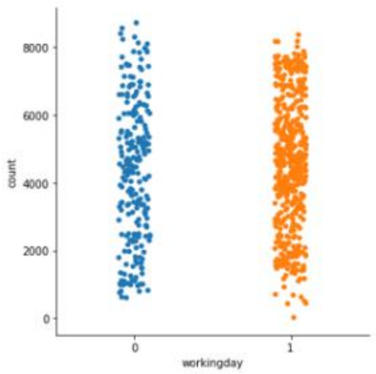


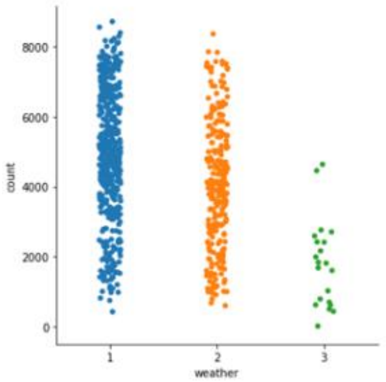




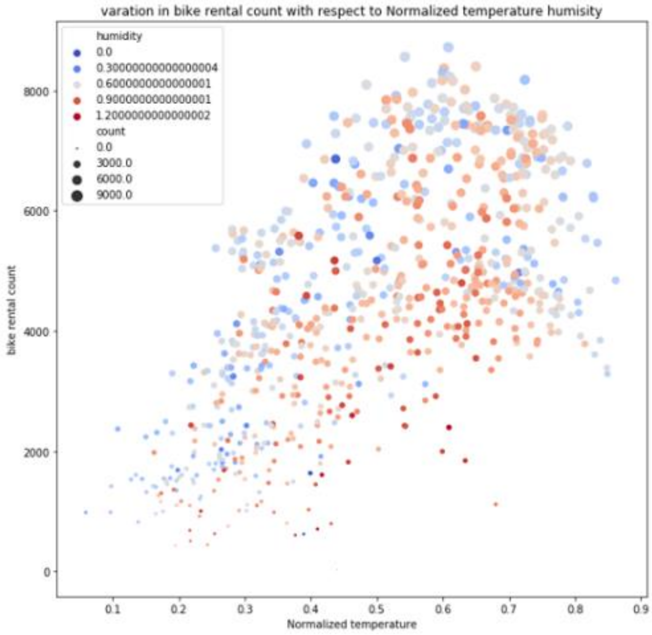






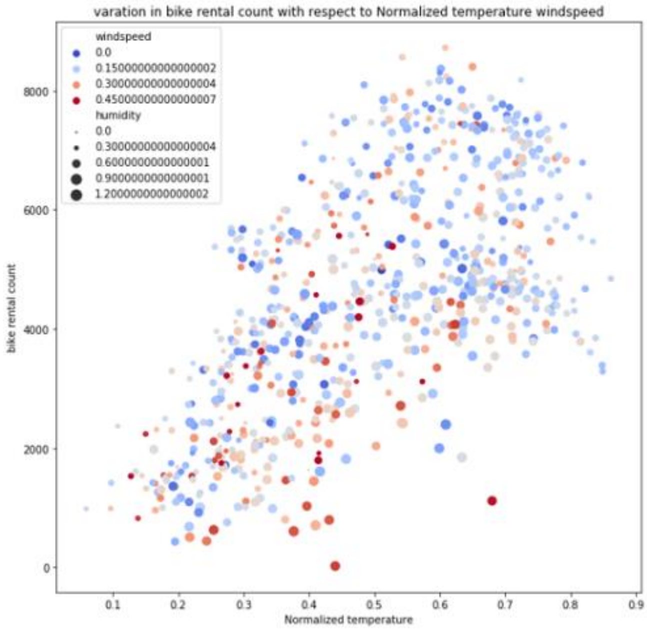



```
f,ax = plt.subplots(figsize=(10,10))

sns.scatterplot(x="temprature",y="count",hue="humidity",size="count",palette="coolw
arm",sizes=(1,100),linewidth=0,

                data=bike_rental,ax=ax)

plt.title("varation in bike rental count with respect to Normalized temperature hum
isity")

plt.ylabel("bike rental count")

plt.xlabel("Normalized temperature")

plt.savefig("bike_temp&humidity_plot.pdf")
```



In [30]:

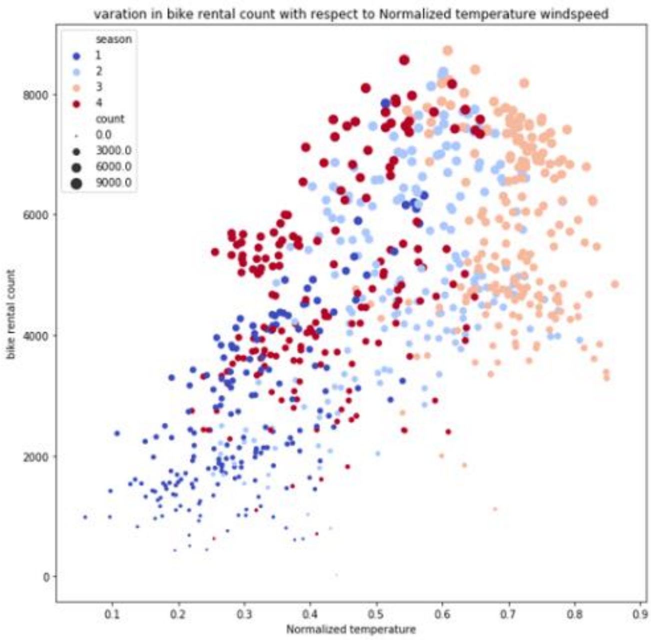
```
f,ax = plt.subplots(figsize=(10,10))

sns.scatterplot(x="temprature",y="count",hue="windspeed",size="humidity",palette="c
oolwarm",sizes=(1,100),linewidth=0,

                 data=bike_rental,ax=ax)

plt.title("varation in bike rental count with respect to Normalized temperature win
dspeed")

plt.ylabel("bike rental count")

plt.xlabel("Normalized temperature")

plt.savefig("bike_temp&windspeed_plot.pdf")
```



In [31]:

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

```
sns.scatterplot(x="temprature",y="count",hue="season",size="count",palette="coolwar
m",sizes=(1,100),linewidth=0,

                data=bike_rental,ax=ax)

plt.title("varation in bike rental count with respect to Normalized temperature win
dspeed")

plt.ylabel("bike rental count")

plt.xlabel("Normalized temperature")

plt.savefig("bike_temp&windspeed_plot.pdf")
```



Feature Engineering

In [32]:

```
#Converting redpective variables to required data format

bike_rental['season'] = bike_rental['season'].astype('category')

bike_rental['year'] = bike_rental['year'].astype('category')

bike_rental['month'] = bike_rental['month'].astype('category')

bike_rental['holiday'] = bike_rental['holiday'].astype('category')

bike_rental['weekday'] = bike_rental['weekday'].astype('category')

bike_rental['workingday'] = bike_rental['workingday'].astype('category')

bike_rental['weather'] = bike_rental['weather'].astype('category')


bike_rental['temprature'] = bike_rental['temprature'].astype('float')

bike_rental['atemp'] = bike_rental['atemp'].astype('float')

bike_rental['humidity'] = bike_rental['humidity'].astype('float')

bike_rental['windspeed'] = bike_rental['windspeed'].astype('float')

bike_rental['count'] = bike_rental['count'].astype('float')
```

Feature Selection

In [33]:

```
# Lets save dataset after outlier analysis
df =  bike_rental.copy()
bike_rental = df.copy()
```

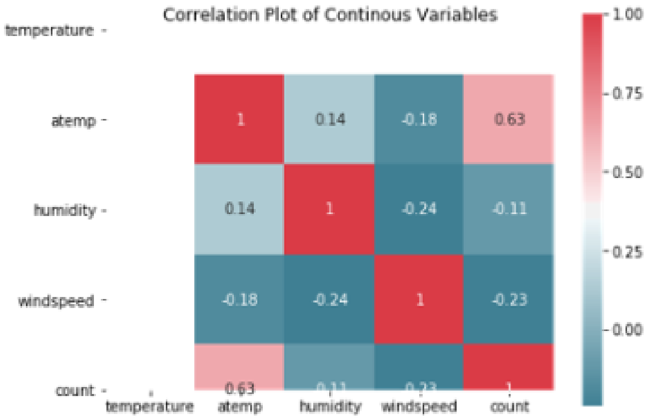
In [35]:

```
# 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_rental.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.d
iverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax,annot=True)


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

```
             temperature     atemp  humidity  windspeed     count
temperature          NaN       NaN       NaN        NaN       NaN
atemp                NaN  1.000000  0.139924  -0.182948  0.631066
humidity             NaN  0.139924  1.000000  -0.241160 -0.105664
windspeed            NaN -0.182948 -0.241160   1.000000 -0.233657
count                NaN  0.631066 -0.105664  -0.233657  1.000000
```

Out[35]:

Text(0.5, 1, 'Correlation Plot of Continous Variables')



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

In [36]:

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

In [37]:

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

# Anova analysis for categorical variable with target numeric variable

import statsmodels.api as sm
from statsmodels.formula.api import ols

for i in cat_cnames:
    mod = ols('count' + '~' + i, data = bike_rental).fit()
    aov_table = sm.stats.anova_lm(mod, typ = 2)
    print(aov_table)
```

```
                sum_sq     df           F        PR(>F)
season    9.505959e+08    3.0  128.769622  6.720391e-67
Residual  1.788940e+09  727.0         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     1.070192e+09   11.0  41.903703  4.251077e-70
Residual  1.669343e+09  719.0        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   1.765902e+07    6.0  0.782862  0.583494
Residual  2.721876e+09  724.0       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   2.716446e+08    2.0  40.066045  3.106317e-17
Residual  2.467891e+09  728.0        NaN           NaN
```

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 [38]:

```
# droping variables (feature selection)
bike_rental = bike_rental.drop(['atemp', 'holiday','weekday','workingday'], axis=1)
```

In [39]:

```
# Lets check dimensions after dimension reduction
bike_rental.shape
```

Out [39]:

```
(731, 8)
```

In [40]:

```
# Lets check column names after dimension reduction
bike_rental.columns
```

Out [40]:

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

In [41]:

```
bike_rental.head()
```

Out[41]:

	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 [42]:

```
# Lets update  continous and categorical variables after dimension reduction

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

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

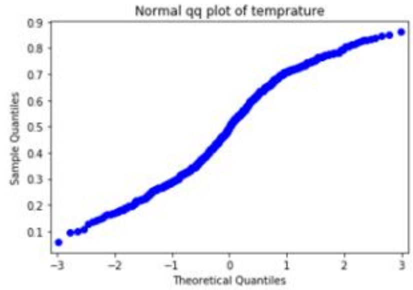
Feature Scaling

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

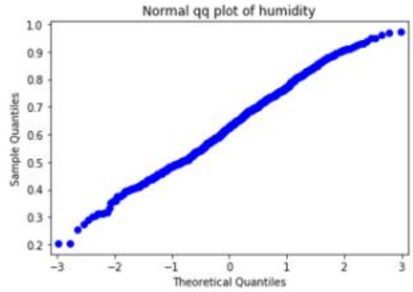
In [46]:

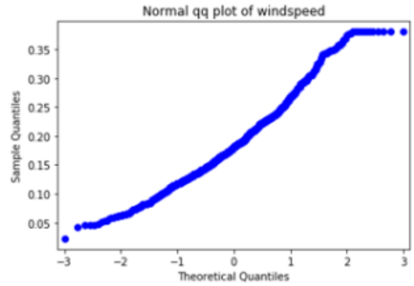
```
#Normality check
for i in cnames:
    print(i)
    sm.qqplot(bike_rental[i])
    plt.title("Normal qq plot of " +i)
    plt.show()
```

temprature

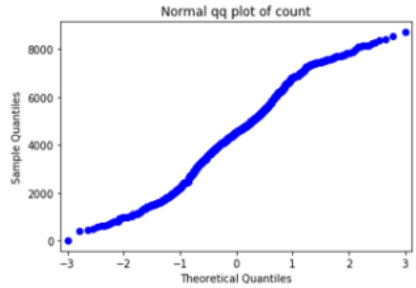


humidity





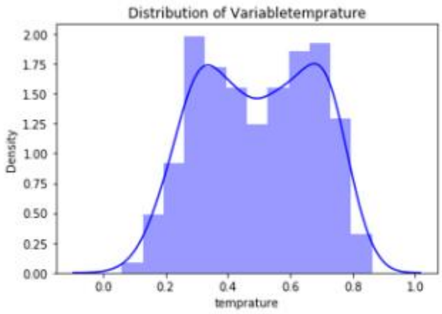
count



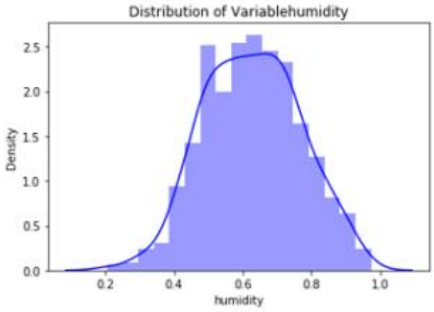
In [47]:

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

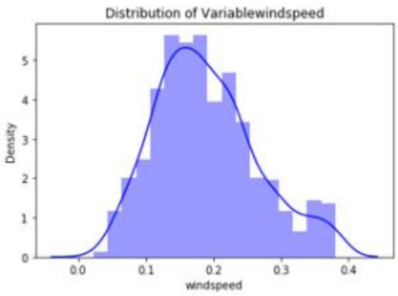
temprature



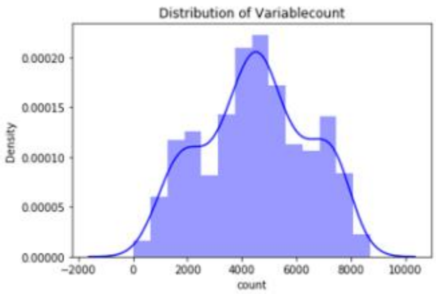
humidity



windspeed



count



In [48]:

```
bike_rental.describe()
```

Out[48]:

	temprature	humidity	windspeed	count
count	731.000000	731.000000	731.000000	731.000000
mean	0.495385	0.628197	0.189846	4504.348837
std	0.183051	0.141320	0.075644	1937.211452
min	0.059130	0.204687	0.022392	22.000000
25%	0.337083	0.520000	0.134950	3152.000000
50%	0.498333	0.626667	0.180975	4548.000000
75%	0.655417	0.730209	0.233214	5956.000000
max	0.861667	0.972500	0.380611	8714.000000

From distribution plot, normal qq plot and summary it is clear that data is already normalized.

From distribution plot,normal qq plot and summary it is clear that data is already normalized.

Model Development

Now all the required data preprocessing procedure finished. now i'm going to build the model with suitable machine learning algorithm.

Before going to apply machine learning Algorithm, i'm going to divide the data as test data and train data.

In [49]:

```
# Load Required libraries for model development
from sklearn.model_selection import train_test_split #used to split dataset into tr
ain 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
```

Lets convert all categorical variables ito dummy variables As we cant pass categorical variables directly in to regression problems

In [50]:

```
# Lets save our preprocessed data into ML data set
ML = bike_rental
bike_rental = ML
```

In [51]:

```
# Lets call Categorical varaibles after feature selection using ANOVA
cat_cnames = ['season', 'year', 'month','weather']
```

In [52]:

```
#  Create categorical variables to dummy variables-
bike_rental = pd.get_dummies(bike_rental,columns=cat_cnames)
```

In [53]:

```
bike_rental.shape
```

Out[53]:

```
(731, 25)
```

In [54]:

```
bike_rental.columns
```

Out[54]:

```
Index(['temprature', 'humidity', 'windspeed', 'count', 'season_1', 'season_2',
       'season_3', 'season_4', 'year_0', 'year_1', 'month_1', 'month_2',
       'month_3', 'month_4', 'month_5', 'month_6', 'month_7', 'month_8',
       'month_9', 'month_10', 'month_11', 'month_12', 'weather_1', 'weather_2',
       'weather_3'],
      dtype='object')
```

In [55]:

```
bike_rental.head()
```

Out[55]:

	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	month_11	month_12	weather_1	weather_2	weather_3
0	0.344167	0.805833	0.160446	985.0	1	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1	0
1	0.363478	0.696087	0.248539	801.0	1	0	0	0	1	0	...	0	0	0	0	0	0	0	0	1	0
2	0.196364	0.437273	0.248309	1349.0	1	0	0	0	1	0	...	0	0	0	0	0	0	0	1	0	0
3	0.200000	0.590435	0.160296	1562.0	1	0	0	0	1	0	...	0	0	0	0	0	0	0	1	0	0
4	0.226957	0.43695	0.186900	1600.	1	0	0	0	1	0	...	0	0	0	0	0	0	0	1	0	0

	temperature	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	month_11	month_12	weather_1	weather_2	weather_3
		7		0							.										

5 rows × 25 columns

In [56]:

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

In [57]:

```
# Divide data into train and test sets
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=.20,random_state=0)
```

In [58]:

```
# Function for Error metrics to calculate the performance of model
def MAPE(y_true,y_prediction):
    mape= np.mean(np.abs(y_true-y_prediction)/y_true)*100
    return mape
```

The given data set has dependent variable and also independent variable. So as per the data analysis this data set under the supervised data set. So here i am going to use and apply supervised machine learning algorithm. Those algorithms are

Decision tree regression,

Random Forest,

Linear regression,

Gradient Boosting.

I am going to check with these machine learning. Based on the accuracy and performance of the algorithm, i will select the suitable machine learning algorithm for this particular project.

Decision Tree regression

In [59]:

```
#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)
```

In [60]:

```
# 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
RMSE_test = np.sqrt(metrics.mean_squared_error(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))
print("RMSE for train data="+str(RMSE_train))
print("RMSE for test data="+str(RMSE_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
RMSE for train data=1080.3818579492188
RMSE for test data=1226.2196190864843
```

In [61]:

```
#applying
predict_DT = DT_model.predict(X_test)
```

In [62]:

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

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

In [63]:

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

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

In [64]:

```
# 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)
```

In [65]:

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

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

In [66]:

```
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))
```

```
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
RMSE for train data=1080.3818579492188
RMSE for test data=1226.2196190864843
```

In [67]:

```
Error_metrics_DT= {'Model Name': ['Decision Tree'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],'R-squared_Test':[r2_test],'RMSE_train':[RMSE_train],'RMSE_test':[RMSE_test]}

DT_Results = pd.DataFrame(Error_metrics_DT)
```

In [68]:

```
DT_Results
```

Out[68]:

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Decision Tree	62.260133	36.948093	0.677563	0.64647	1080.381858	1226.219619

Random Forest

In [69]:

```
# Import libraris
from sklearn.ensemble import RandomForestRegressor

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

# Prediction on train data
RF_train= RF_model.predict(X_train)

# Prediction on test data
RF_test= RF_model.predict(X_test)

# MAPE For train data
MAPE_train= MAPE(y_train,RF_train)

# MAPE For test data
MAPE_test= MAPE(y_test,RF_test)

# Rsquare  For train data
r2_train= r2_score(y_train,RF_train)

# Rsquare  For test data
r2_test=r2_score(y_test,RF_test)

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

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


print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage Error for test data="+str(MAPE_test))
print("R^2_score for train data="+str(r2_train))
print("R^2_score for test data="+str(r2_test))
print("RMSE for train data="+str (RMSE_train))
print("RMSE for test data="+str(RMSE_test))
```

```
Mean Absolute Precentage Error for train data=17.046471082559737
Mean Absolute Precentage Error for test data=20.78710156756213
R^2_score for train data=0.9804562488420521
R^2_score for test data=0.8799703233701455
RMSE for train data=265.9859865386361
RMSE for test data=714.4951770101378
```

In [70]:

```
RF_test= RF_model.predict(X_test)
```

In [71]:

```
# Lets print results of Randomforest random search
Error_metrics_RF= {'Model Name': ['Random Forest'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],
      'R-squared_Test':[r2_test],'RMSE_train':[RMSE_train],'RMSE_test':[RMSE_test]}
RF_Results = pd.DataFrame(Error_metrics_RF)
```

In [72]:

```
RF_Results
```

Out[72]:

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Random Forest	17.046471	20.787102	0.980456	0.87997	265.985987	714.495177

Linear Regression

In [73]:

```
# 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:                       0.837
Model:                            OLS   Adj. R-squared:                  0.832
Method:                 Least Squares   F-statistic:                     144.9
Date:                Thu, 23 Jan 2020   Prob (F-statistic):         6.64e-207
Time:                        15:02:28   Log-Likelihood:                -4708.1
No. Observations:                 584   AIC:                             9458.
Df Residuals:                     563   BIC:                             9550.
Df Model:                          20
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
temprature  4858.9463    470.970     10.317      0.000    3933.872    5784.020
humidity   -2037.1134    349.901     -5.822      0.000   -2724.384   -1349.843
windspeed  -3199.2377    478.946     -6.680      0.000   -4139.976   -2258.499
season_1     -98.6095    147.626     -0.668      0.504    -388.574     191.355
season_2     796.8599    147.746      5.393      0.000     506.659    1087.061
season_3     802.8059    167.798      4.784      0.000     473.220    1132.392
season_4    1449.2066    167.464      8.654      0.000    1120.276    1778.137
year_0       509.7013    150.788      3.380      0.001     213.526     805.877
year_1      2440.5616    149.453     16.330      0.000    2147.008    2734.115
month_1        3.8964    194.970      0.020      0.984    -379.061     386.854
month_2       86.7672    184.628      0.470      0.639    -275.877     449.412
month_3      536.1318    139.990      3.830      0.000     261.165     811.098
month_4      269.5716    171.747      1.570      0.117     -67.772     606.915
month_5      656.3467    180.601      3.634      0.000     301.612    1011.082
month_6      231.1045    177.474      1.302      0.193    -117.487     579.696
month_7     -239.2666    217.946     -1.098      0.273    -667.353     188.819
month_8      268.4678    204.104      1.315      0.189    -132.430     669.365
month_9      899.5437    171.554      5.243      0.000     562.579    1236.508
month_10     435.0144    185.201      2.349      0.019      71.244     798.784
month_11    -150.0238    192.217     -0.780      0.435    -527.573     227.526
month_12     -47.2906    166.144     -0.285      0.776    -373.629     279.047
weather_1   1673.2158     89.774     18.638      0.000    1496.883    1849.549
weather_2   1358.2012    109.310     12.425      0.000    1143.496    1572.906
weather_3    -81.1542    218.396     -0.372      0.710    -510.124     347.816
==============================================================================
Omnibus:                       99.070   Durbin-Watson:                   1.895
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              263.316
Skew:                          -0.850   Prob(JB):                     6.63e-58
Kurtosis:                       5.816   Cond. No.                     2.40e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly sp
ecified.
[2] The smallest eigenvalue is 2.06e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [74]:

```
# 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))
```

```
Mean Absolute Precentage Error for train data=43.77371916032853
Mean Absolute Precentage Error for test data=19.685303059218505
R^2_score for train data=0.8373616227857162
R^2_score for test data=0.8392613195741977
RMSE for train data=767.3019650844327
RMSE for test data=826.8280143941666
```

In [75]:

```
Error_Metrics = {'Model Name': ['Linear Regression'],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],
      'R-squared_Test':[r2_test],'RMSE_train':[RMSE_train],'RMSE_test':[RMSE_test]}

LR_Results = pd.DataFrame(Error_Metrics)
```

In [76]:

```
LR_Results
```

Out[76]:

	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

Gradient Boosting

In [77]:

```
# 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))



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))
```

```
Mean Absolute Precentage Error for train data=43.77371916032853
Mean Absolute Precentage Error for test data=18.43060471106612
R^2_score for train data=0.9492901918330808
R^2_score for test data=0.8690308728150022
RMSE for train data=428.4506224460579
RMSE for test data=746.3446949038457
```

In [78]:

```
# Lets print the result

Error_metrics_GB = {'Model Name': ['Gradient Boosting'],'MAPE_Train':[MAPE_train],'
MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],

      'R-squared_Test':[r2_test],'RMSE_train':[RMSE_train],'RMSE_test':[RMSE_test]}

GB_results = pd.DataFrame(Error_metrics_GB)
```

In [79]:

```
GB_results
```

Out[79]:

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Gradient Boosting	43.773719	18.430605	0.94929	0.869031	428.450622	746.344695

In [80]:

```
Final_Results = pd.concat([DT_Results,RF_Results,LR_Results,GB_results], ignore_ind
ex=True, sort =False)
```

In [81]:

```
Final_Results
```

Out[81]:

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Decision Tree	62.260133	36.948093	0.677563	0.646470	1080.381858	1226.219619
1	Random Forest	17.046471	20.787102	0.980456	0.879970	265.985987	714.495177
2	Linear Regression	43.773719	19.685303	0.837362	0.839261	767.301965	826.828014
3	Gradient Boosting	43.773719	18.430605	0.949290	0.869031	428.450622	746.344695

In [82]:

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

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


input = y_test.reset_index()

pred = pd.DataFrame(RF_test,columns = ['pred'])

Final_output = pred.join(input)
```

In [83]:

```
Final_output
```

Out[83]:

	pred	index	count
0	5070.24	196	5923.0
1	4628.00	187	4592.0
2	1526.02	14	1248.0
3	1258.88	31	1360.0
4	3524.18	390	4075.0

	pred	index	count
...	...	...	...
142	5409.30	566	5870.0
143	5413.67	688	5499.0
144	4231.39	266	5423.0
145	7391.34	504	8294.0
146	4741.77	239	4334.0

147 rows × 3 columns

In [84]:

```
Final_output.to_csv("RF_results.csv")
```

End Model

R Code

Project Name: Bike Rental Prediction

Clean the environment

rm(list=ls())

Set working directory

setwd("C:/Users/Hp/Desktop/Project2")

Check current working directory

getwd()

Load the required libraries for analysis of data

x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced","C50",

"dummies", "e1071", "Information", "MASS", "rpart", "gbm", "ROSE",

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

#install.packages(x)

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

rm(x)

Load the csv file

```
bike_rental= read.csv("day.csv",header = T,na.strings = c(""," ",NA))

# Explore the data
#Check the dimensions
dim(bike_rental)

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

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

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

#Check structure of dataset
str(bike_rental)

#Check summary of dataset
summary(bike_rental)

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

```
# Remove these variables
# instant variable
# casual and registered variable as count is sum of these two variables
# count = casual + registered
bike_rental = subset(bike_rental,select=-c(instant,dteday,casual,registered))

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

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

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

# EDA or Data Preprocessing
# Duplicate Values
duplicated(bike_rental)# No duplicates in dataset


# Missing Value anlysis
# Check missing values in dataset
sum(is.na(bike_rental))
# there is no missing values present in this dataset
# Outlier Analysis and treatment
# Lets save copy of dataset before preprocessing
df = bike_rental
```

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

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

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

  #Inter quartile range
  IQR = Q3-Q1
```

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

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

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

# Lets plot boxplots after removing outiers
for(i in 1:length(cnames))
{
  assign(paste0("gn",i),ggplot(aes_string(y=(cnames[i]),x = 'count'),
                              data=subset(bike_rental))+
          stat_boxplot(geom = "errorbar",width = 0.5) +
          geom_boxplot(outlier.color = "red",fill="grey",
                       outlier.shape = 18,outlier.size = 1,notch = FALSE)+
          theme(legend.position = "bottom")+
          labs(y = cnames[i],x='count')+
          ggtitle(paste("boxplot of count for",cnames[i])))
}

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

```
#visualization
# Continous Variables
cnames= c("temperature","atemp","humidity","windspeed","count")

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

# Univariate Analysis
# Histogram for continuous variables to check  distribution of each variable
for(i in 1:length(cnames))
{
  assign(paste0("h",i),ggplot(aes_string(x=(cnames[i])),
                              data=subset(bike_rental))+
           geom_histogram(fill="darkslateblue",colour = "black")+geom_density()+
           scale_y_continuous(breaks =pretty_breaks(n=10))+
           scale_x_continuous(breaks = pretty_breaks(n=10))+
           theme_bw()+xlab(cnames[i])+ylab("Frequency")+
           ggtitle(paste("distribution 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_rental))+
           geom_point(alpha=0.5,color="brown") +
```

```
        labs(title = "Scatter Plot of count vs", x = (cnames[i]), y = "count")+
         ggtitle(paste("Scatter Plot of count vs",cnames[i])))
}

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

# count vs temperature(atemp) : as temperature increase Bike rent count also increases
# count vs humidity : humidity doesnt have any effect on bikerent count
# count vs windspeed : windspeed doesnt have any effect on bikerent count
# count vs count : please ignore this plot as it is our target variable
options(scipen = 999)
# Let us check impact of categorical variables on count
for(i in 1:length(cat_cnames))
{
  assign(paste0("b",i),ggplot(aes_string(y='count',x = (cat_cnames[i])),
                              data=subset(bike_rental))+
         geom_bar(stat = "identity",fill = "blue") +
         labs(title = "Scatter Plot of count vs", x = (cnames[i]), y = "count")+
         ggtitle(paste("Number of bikes rented with respect to",cat_cnames[i])))+
    theme(axis.text.x = element_text( color="black", size=8))+
    theme(plot.title = element_text(face = "bold"))
}
# using library(gridExtra)
gridExtra::grid.arrange(b1,b2,b3,b4,ncol = 2)
gridExtra::grid.arrange(b5,b6,b7,ncol = 2)

# From barplot we can observe below points
# Season:Bike rent count is high in season 3(fall) and low in season 1(springer)
```

aggregate(count ~ season ,sum,data = bike_rental)

year : Bike rent count is high in year 1 (in 2012)

aggregate(count ~ year ,sum,data = bike_rental)

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

aggregate(count ~ month,sum,data = bike_rental)

holiday : Bike rent count is high on holidays ie 0

aggregate(count ~ holiday ,sum,data = bike_rental)

weekday :From bar plot we can see maximum bikes rented on 5th day and least bikes on day 0.

aggregate(count ~ weekday ,sum,data = bike_rental)

workingday : Bike rent count is high on working day ie 1

aggregate(count ~ workingday,sum,data = bike_rental)

weather : Bike rent count is high on weather 1: ie when the weather is

Clear, Few clouds, Partly cloudy, Partly cloudy

aggregate (count ~ weather,sum,data = bike_rental)

Bikes rented with respect to temp and humidity

ggplot(bike_rental,aes(temperature,count)) +

geom_point(aes(color=humidity),alpha=0.5) +

labs(title = "Bikes rented with respect to variation in temperature and humidity", x = "temperature")+ theme_bw()

maximum bike rented between temp 0.50 to 0.75 and humidity 0.50 to 0.75

#Bikes rented with respect to temp and weather

ggplot(bike_rental, aes(x = temperature, y = count))+

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

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

Bikes rented with respect to temp and season

```
ggplot(bike_rental, aes(x = temperature, y = count))+
  geom_point(aes(color=season))+
  labs(title = "Bikes rented with respect to temperature and season", x = "temperature")+
   theme(panel.background = element_rect("white"))+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme_bw()
```

From figure it is clear that maximum bike count is for season 2 and 3,when the temp between 0.5 to 0.7

Feature Selection

Lets save dataset after outlier analysis

```
df =  bike_rental
bike_rental = df
```

Using corrplot library we do correlation analysis for numeric variables

Let us derive our correlation matrix

```
Correlation_matrix = cor(bike_rental[,cnames])
Correlation_matrix
```

By looking at correlation matrix we can say temperature and atemp are highly correlated.

#Lets plot correlation plot using corrgram library

```
corrgram(bike_rental[,cnames],order = F,upper.panel = panel.pie,
```

```
    text.panel = panel.txt,main="Correlation plot for numeric variables")


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

# Lets find significant categorical variables usig ANOVA test

# Anova analysis for categorical variable with target numeric variable

 for(i in cat_cnames){

 print(i)

 Anova_result= summary(aov(formula = count~ bike_rental[,i],bike_rental))

 print(Anova_result)

}


# 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.


# Dimension reduction

bike_rental = subset(bike_rental,select = -c(atemp,holiday,weekday,workingday))


# Lets check dimensions after dimension reduction

dim(bike_rental)

head(bike_rental)


# Lets check column names after dimension reduction

names(bike_rental)


# Lets update  continous and categorical variables after dimension reduction

# Continuous variable

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


# Categorical variables
```

```
cat_cnames = c('season', 'year', 'month','weather')

# Feature Scaling
# normality
# Normality check using normal qq plot
for(i in cnames){
  print(i)
  qqplot= qqnorm(bike_rental[,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_rental[,i]))
}

# From normal qq plot,histplot and by looking at summary of
# numeric variables we can say data is normally distributed
# Model Development
# Let's clean R Environment, as it uses RAM which is limited
library(DataCombine)
rmExcept("bike_rental")

# 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_rental
bike_rental = df
```

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

# lets create dummy variables using dummies library
library(dummies)
bike_rental = dummy.data.frame(bike_rental,cat_cnames)
dim(bike_rental)
head(bike_rental)
# 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_rental$count, p = .80, list = FALSE)
train = bike_rental[ train.index,]
test  = bike_rental[-train.index,]

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

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

# Function for RMSE value
rmse = function(y,yhat){
```

```
  difference = y - yhat
  root_mean_square = sqrt(mean(difference^2))
  print(root_mean_square)
}

# 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)#52.7985

# MAPE For train data test data
DT_MAPE_Test = mape(test[,25],DT_test)#19.9308

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

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

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

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

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

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

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

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

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

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

# Rsquare For test data
```

```
RF_r2_test=rsquare(test[,25],RF_test)

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

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



# Random Forest
install.packages("randomForest")
library(randomForest)

# Lets Build random forest model on train data using randomForest library
RF_model= randomForest(count~.,train,importance=TRUE , ntree=100)

# 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)

# Linear Regression model
# Before building multiple linear regression model lets check the
# vif for multicollinearity
# Continuous variables after feature selection using correlation analysis
cnames= c("temperature","humidity","windspeed")
numeric_data= bike_rental[,cnames]

# VIF test  using usdm library
library(usdm)
vifcor(numeric_data,th=0.6)
# No variable from the 3 input variables has collinearity problem.
# Lets build multiple linear regression model on train data
# we will use the lm() function in the stats package
LR_Model = lm(count ~.,data = train)

# Check summary
summary(LR_Model)
```

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

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

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

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

# rmse For train data
```

LR_rmse_train = rmse(train[,25],LR_train)

rmse For test data

LR_rmse_test = rmse(test[,25],LR_test)

Gradient Boosting

library(gbm)

Lets build a Gradient Boosting model for regression problem

GB_model = gbm(count~., data = train,distribution = "gaussian", n.trees = 100, interaction.depth = 2)

Model Prediction on train data

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

Model Prediction on test data

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

Mape for train data

GB_MAPE_Train=mape(train[,25],GB_train)

Mape for test data

GB_MAPE_Test=mape(test[,25],GB_test)

Rsqure for train data

GB_r2_train=rsquare(train[,25],GB_train)

Rsquare for test data

GB_r2_test=rsquare(test[,25],GB_test)

rmse For train data

```
GB_rmse_train = rmse(train[,25],GB_train)


# rmse For test data

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


# Results

Model = c ('Decision Tree for Regression','Random Forest','Linear Regression','Gradient
Boosting')


MAPE_Train = c(DT_MAPE_Train, RF_MAPE_Train, LR_MAPE_Train,
               GB_MAPE_Train)


MAPE_Test = c(DT_MAPE_Test,RF_MAPE_Test,LR_MAPE_Test,
             GB_MAPE_Test)


Rsquare_Train = c(DT_r2_train,RF_r2_train,
                  LR_r2_train,GB_r2_train)


Rsquare_Test = c(DT_r2_test,RF_r2_test,
                  LR_r2_test,GB_r2_test)


Rmse_Train = c(DT_rmse_train,RF_rmse_train,
               LR_rmse_train,GB_rmse_train)


Rmse_Test = c(DT_rmse_test, RF_rmse_test,
              LR_rmse_test,GB_rmse_test)


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

Final_results
```

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

Lets save the output of finalized model (RF)

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

Exporting the output to hard disk for further use

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

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

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

write.csv(Final_output,"C:/Users/Hp/Desktop/Project2/RF_output_R.csv", row.names = FALSE)

Appendix C –References

1) edWisor Learning
2) https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/
3) https://towardsdatascience.com/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regression-metrics-d4a1a9ba3d74
4) https://towardsdatascience.com/how-to-select-the-right-evaluation-metric-for-
5) machine-learning-models-part-2-regression-metrics-d4a1a9ba3d74
6) https://medium.com/data-distilled/residual-plots-part-1-residuals-vs-fitted-plot-f069849616b1
7) https://medium.com/@TheDataGyan/day-8-data-transformation-skewness-normalization-and-much-more-4c144d370e55

Thank you