BIKE RENTAL COUNT

DATA SCIENCE PROJECT

JAYRAM S

21-03-2020

Contents

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

1.2 Data

There are 16 variables in our data in which 15 are independent variables and 1 (cnt) is dependent variable. Since our target variable is continuous in nature, this is a regression problem.

Variables Information:

instant: Record index

dteday: Date

season: Season (1:springer, 2:summer, 3:fall, 4:winter)

yr: Year (0: 2011, 1:2012)

mnth: Month (1 to 12)

holiday: weather day is holiday or not (extracted fromHoliday Schedule)

weekday: Day of the week

workingday: If day is neither weekend nor holiday is 1, otherwise is 0.

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

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)

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)

hum: Normalized humidity. The values are divided to 100 (max)

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

casual: count of casual users

registered: count of registered users

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

1.3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a method to analysing data sets to summarize their main features. In the given data set, there are 16 variables and data types of all variables are either float64 or int64. There are 731 observations and 16 columns in our data set. Missing value is not present in our data. There are 731

List of columns and their number of unique values -

```
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
```

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

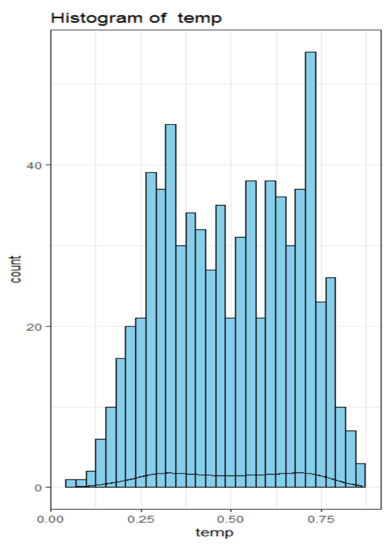
Chapter 2

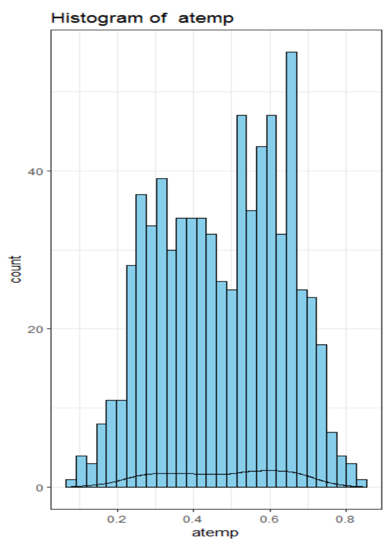
Methodology

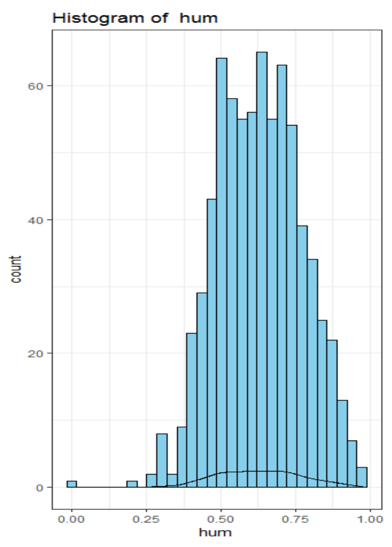
Before feeding the data to the model we need to clean the data and convert it to a proper format. It is the most crucial part of data science project we spend almost 80% of time in it.

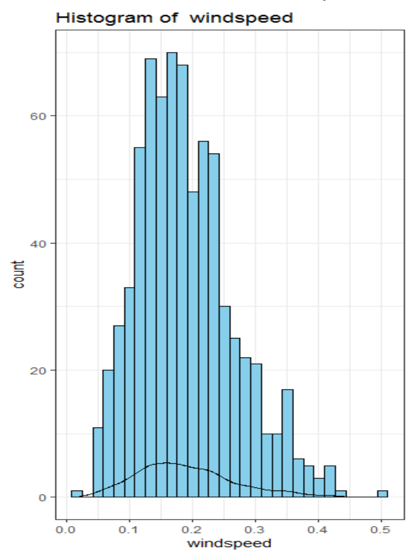
2.1 Pre Processing

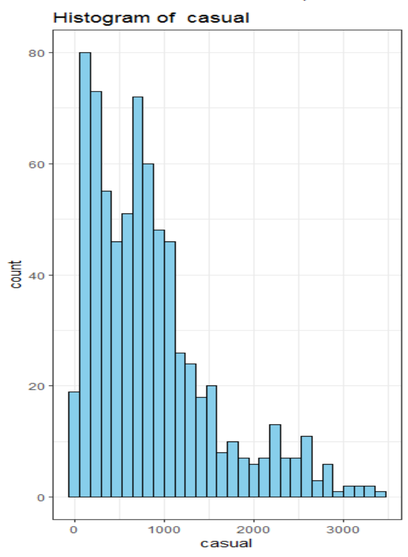
Any predictive modelling requires that we look at the data before we start modelling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is frequently called as Exploratory Data Analysis. To flinch this process, we will first try and look at all the probability distributions of the variables. Most analysis like the regression, require the data to be normally distributed. We can visualize in a glance by looking at the probability distributions or probability density functions of the variable.

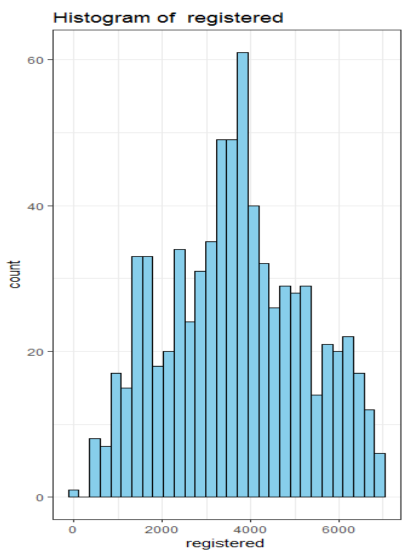












2.2.1 Missing Value Analysis

In statistics, *missing data*, or *missing values*, occur when no *data value* is stored for the variable in an observation. *Missing data* are a common occurrence and can have a significant effect on the conclusions that can be drawn from the *data*. If a column has more than 30% of data as missing value either we ignore the entire column or we ignore those observations But in this problem, we don't have any missing value.

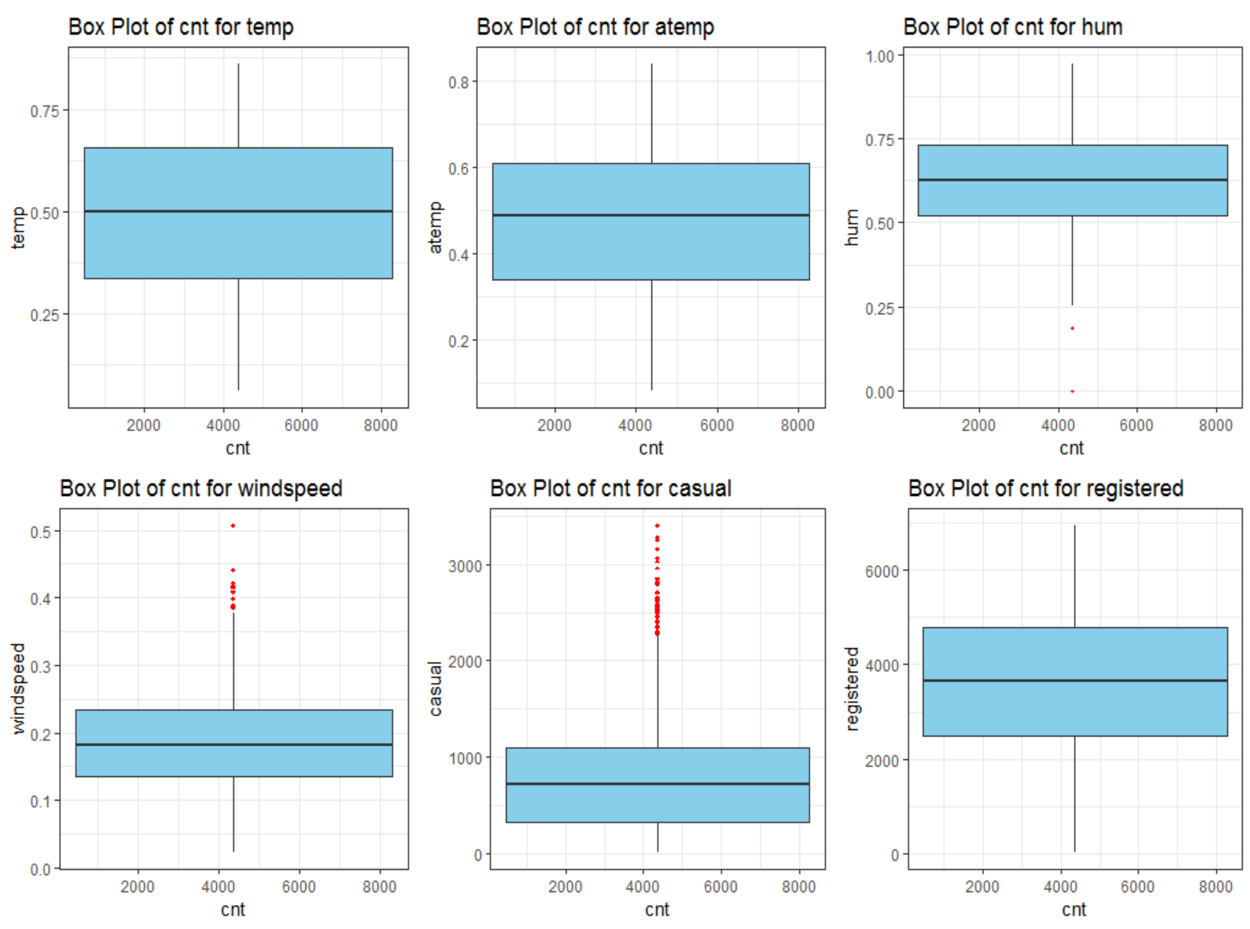
```
> Missing_Val   # ----------------------> There is no missing value found.

instant                                   0
dteday                                    0
season                                    0
yr                                        0
mnth                                      0
holiday                                   0
weekday                                   0
workingday                                0
weathersit                                0
temp                                      0
atemp                                     0
```

```
hum                                     0
windspeed                               0
casual                                  0
registered                              0
cnt
```

2.1.2 <u>Outlier Analysis</u>

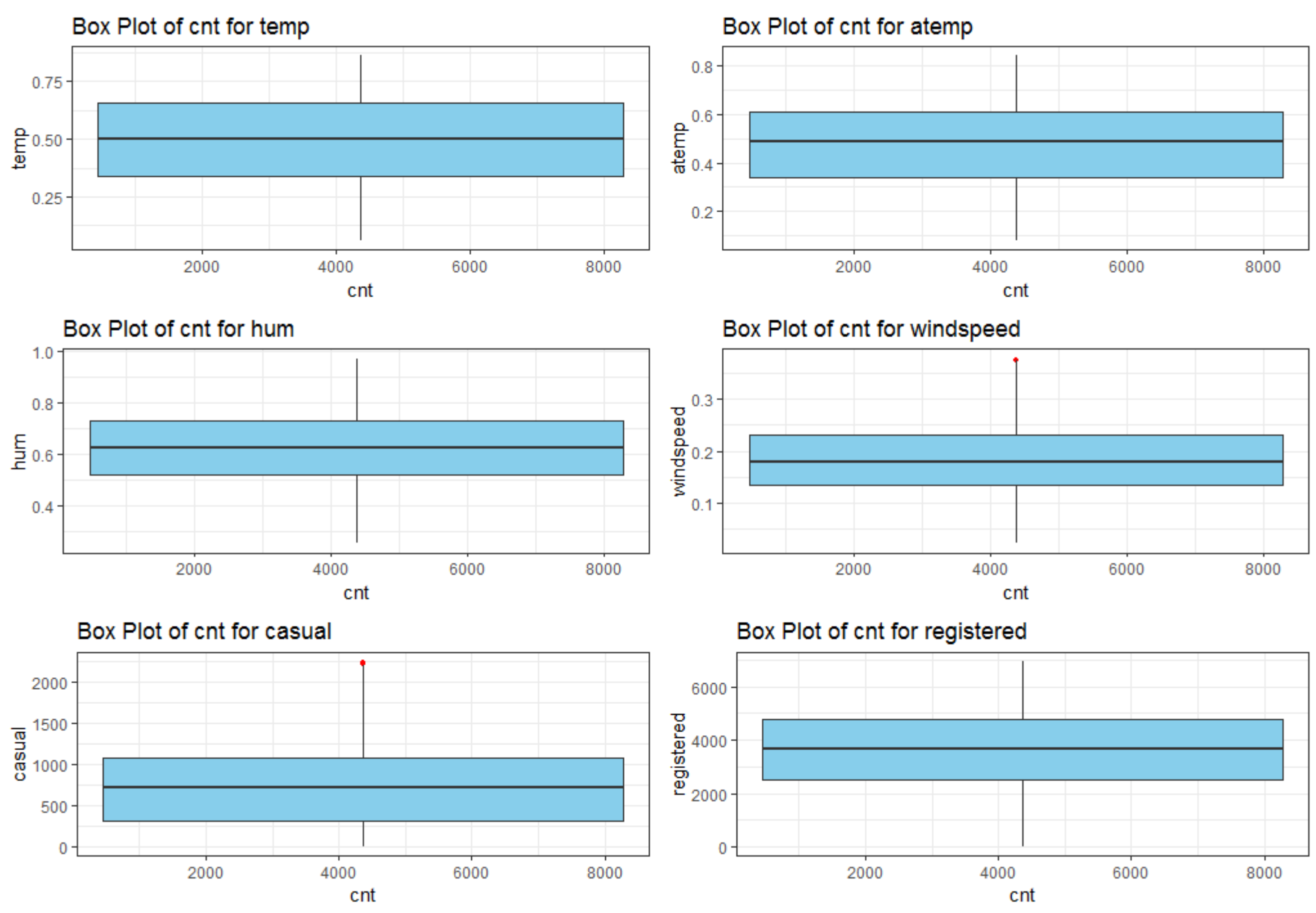
We can clearly observe from these probability distributions that most of the variables are skewed. The skew in these distributions can be most likely explained by the presence of outliers and extreme values in the data. One of the other steps of pre-processing apart from checking for normality is the presence of outliers. In this case we use a classic approach of removing outliers. We visualize the outliers using boxplots.

In figure we have plotted the boxplots of the 6 predictor variables with respect to **cnt**. A lot of useful inferences can be made from these plots. First as we can see, we have a lot of outliers and extreme values in each of the data set.



From the boxplot “hum”, “windspeed”, and “casual” consists of outliers. We have converted the outliers (data beyond minimum and maximum values) as NA i.e. missing values and fill them by **KNN** imputation method in case of R and mean method in case of Python

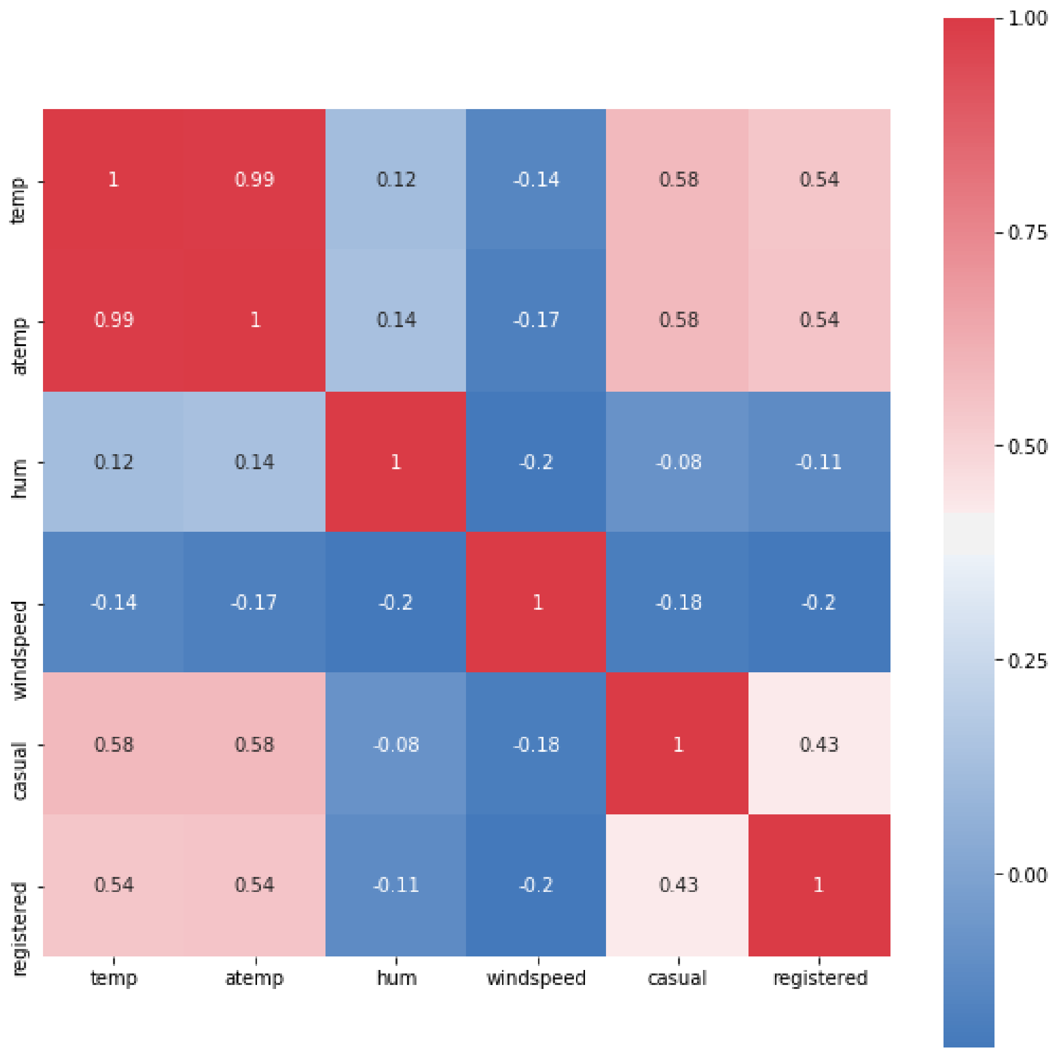
Below we can see new Boxplot to make sure that we have almost removed outliers:



2.1.3 <u>Feature Selection</u>

Before performing any type of modeling we need to assess the importance of each of the predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class and the regression prediction. Selecting the subset of relevant columns as 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   inappropriate information which can increase above. 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, **ANOVA** (Analysis of variance) for categorical variable and in case of R we have also used Backward Elimination Method by lm().

Correlation plot:



	temp	atemp	hum	windspeed	casual	registered
temp	1	0.99	0.12	-0.14	0.58	0.54
atemp	0.99	1	0.14	-0.17	0.58	0.54
hum	0.12	0.14	1	-0.2	-0.08	-0.11
windspeed	-0.14	-0.17	-0.2	1	-0.18	-0.2
casual	0.58	0.58	-0.08	-0.18	1	0.43
registered	0.54	0.54	-0.11	-0.2	0.43	1

From correlation analysis we have found that **temp** and **atemp** has high correlation (=0.99), so we have excluded the **atemp** column both in **R and PYTHON**

ANNOVA IN PYTHON:

```
P value for variable instant is 0.0
P value for variable season is 0.0
P value for variable yr is 0.0
P value for variable mnth is 0.0
P value for variable holiday is 0.0
P value for variable weekday is 0.0
```

```
P value for variable workingday is 0.0
P value for variable weathersit is 0.0
```

Here P value of **every variable** is 0.0 which is less than the **threshold value** 0.05.

My Hypothesis is-

H0= Independent does not variable explain our Target variable. hello

H1= Independent variable does not explain our Target variable.

If p> 0.05 then Reject the **NULL Hypothesis (H0)**, that means this particular Independent variable is not going to explain my **Target Variable**.

Now, in my case -

P value for every variable, p < 0.05 that means every categorical variable explain my target variables

Conclusion-

we have not Removed any categorical variables from ANOVA. But

we have removed "atemp" correlation Plot Analysis.

we will also remove 'instant' and "dteday" whcih is irrelevant for model learning & prediction.

And finally I have 13 variables remaining in Python –

```
Index(['season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday',
       'weathersit', 'temp', 'hum', 'windspeed', 'casual', 'registered'
       'cnt']
```

Dummy variables:

I have converted categorical variables ("season", "mnth", "weekday", and "weathersit") into dummy variables. Because the categories in these variables Nominal data. The factor (1,2,3,4….) which represent these categories can be understood as ordinal data by Machine learning Model because of these order (1,2,3,4..). Therefore, to avoid this confusion sometimes it is compulsory to convert these variables into dummy variables where each categories will become a feature with 0/1 representation.
1 represents presence of categories and 0 represents absence of categories.

And after all have remaining 31 features with dummy variables.

ANNOVA IN R:

```
instant       1 1.083e+09 1.083e+09   476.8 <2e-16 ***
season        1 4.518e+08 451797359     144 <2e-16 ***
yr            1 8.798e+08 879828893   344.9 <2e-16 ***
mnth          1 2.147e+08 214744463   62.01 1.24e-14 ***
```

```
holiday       1 1.280e+07 12797494   3.421 0.0648 .
weekday       1 1.246e+07 12461089   3.331 0.0684 .
workingday    1 1.025e+07 10246038   2.737 0.0985 .
weathersit    1 2.423e+08 242288753   70.73 <2e-16 ***
```

But here I have only deleted “holiday” whose p>0.05 because the other two variables (“weekday” and “workingday”) contribute to explain my target variable (cnt).

Backward elimination in R:

Backward elimination method is used to delete the irrelevant variables one by one by keeping In mind that p value of each variable must be less than 0.05. And also the best possible value of R^2 and adjusted R^2. Here, first we include all the variables and try to get the best possible team of Features.

```
Call:
lm(formula = cnt ~ ., data = dataset_deleted)

Residuals:
    Min      1Q  Median      3Q     Max
-353.93 -120.99  -34.17   44.25 1783.12

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   36.52445   74.27285   0.492  0.62304
instant       -0.49723    1.08896  -0.457  0.64809
season        -6.49720   17.36816  -0.374  0.70845
yr           221.07385  402.06259   0.550  0.58259
mnth          12.03869   33.59409   0.358  0.72018
holiday      -19.06951   59.85767  -0.319  0.75014
weekday       14.63003    4.83612   3.025  0.00257 **  → Good predictor
workingday  -269.70513   36.39110  -7.411 3.53e-13 *** → Good predictor
weathersit    21.49505   25.07165   0.857  0.39154
temp         214.62734   85.24629   2.518  0.01203 *   → Good predictor
hum          -54.19694   97.64473  -0.555  0.57904
windspeed    -68.40549  145.57847  -0.470  0.63858
casual         0.95419    0.03452  27.639  < 2e-16 *** → Good predictor
registered     1.03449    0.01513  68.385  < 2e-16 *** → Good predictor
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 257.3 on 717 degrees of freedom
Multiple R-squared:  0.9827,   Adjusted R-squared:  0.9824
F-statistic:  3128 on 13 and 717 DF,  p-value: < 2.2e-16
```

From Backward elimination I have erased one by one these variables (dteday,atemp,holiday,season,windspeed,yr) whose P value is >0.05

Finally I received these predictors which contribute much to describe my target variables.

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   21.90354   55.97710   0.391  0.69570
instant        0.10481    0.08058   1.301  0.19377
mnth          -7.82514    3.38035  -2.315  0.02090 *    → Good predictor
weekday       14.79016    4.79622   3.084  0.00212 **   → Good predictor
workingday  -266.38390   35.01198  -7.608 8.69e-14 *** → Good predictor
weathersit    20.36923   24.37128   0.836  0.40355
temp         204.81049   83.83195   2.443  0.01480 *    → Good predictor
hum          -44.82217   93.05406  -0.482  0.63018
casual         0.95727    0.03401  28.146  < 2e-16 *** → Good predictor
registered     1.03457    0.01384  74.738  < 2e-16 *** → Good predictor
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 256.7 on 721 degrees of freedom
Multiple R-squared:  0.9827,  Adjusted R-squared:  0.9824
F-statistic:  4538 on 9 and 721 DF,  p-value: < 2.2e-16
```

And after all I got only 10 predictor variable in R –

```
"instant"    "mnth"       "weekday"    "workingday" "weathersit"
"temp"       "hum"        "casual"     "registered" "cnt"
```

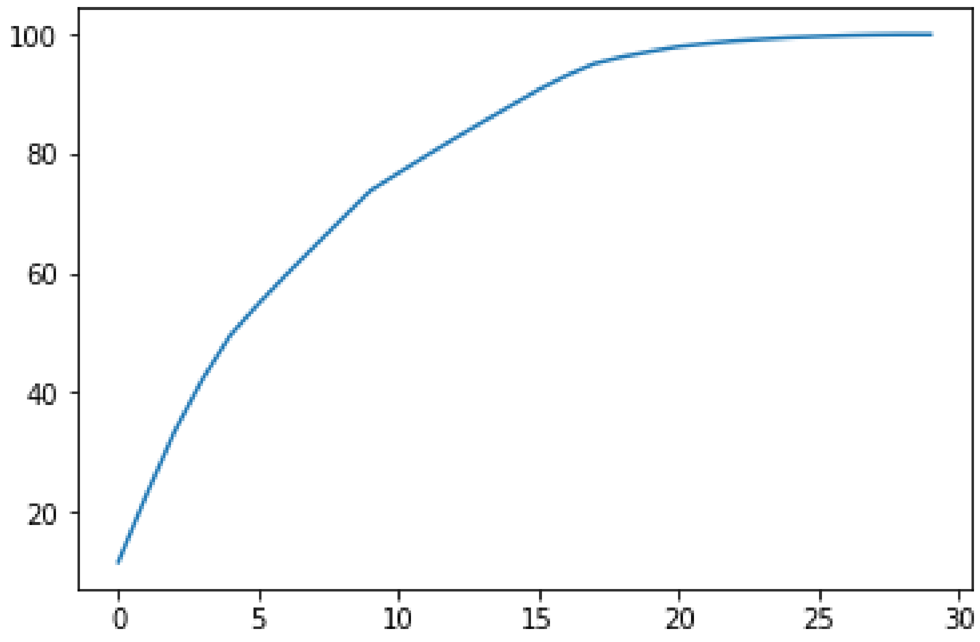
2.2.4 Feature Scaling

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 pre-processing step. Since the range of values of raw data varies widely, 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 our data is not uniformly distributed as we have seen in (**2.1 Pre-Processing**) we have used **Normalization** as Feature Scaling Method.

5 Principal Component Analysis

Principal component analysis is a method of extracting important variables (in form of components) from a large set of variables available in a data set. It extracts low dimensional set of features from a high dimensional data set with a motive to capture as much information as possible. With fewer variables, visualization also becomes much more meaningful. PCA is more useful when dealing with 3 or higher dimensional data. After creating dummy variable of categorical variables the shape of our data became 31 columns and 731 observations, this high number of columns leads to bad accuracy.

We have used PCA only in Python



We have applied PCA algorithm on our data and from the above graph. we have determined that 25 variables out of 31 describes more than 90% of data. So we have selected only those 25 variables to feed our models.

2.2 Modeling

After a thorough preprocessing we will be using some regression models on our processed data to predict the target variable. Following are the models which we have built –

2.2.1 <u>Liner Regression</u>

Linear Regression is one of the statistical methods of prediction. It is applicable only on continuous data. To build any model we have some assumptions to put on data and model. Here are the assumptions to the linear regression model.

Linear Regression	R	PYTHON
RMSE Train	265.3598716	348.6541043531216
RMSE Test	212.2170503	387.8544256571429
R^2 Test	0.9807779	0.9646306127764841

2.2.2 <u>Decision Tree</u>

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". It can be used for classification and regression. It is a supervised machine learning algorithm. Accept continuous and categorical variables as independent variables. Extremely easy to understand by the business users. Split of decision tree is seen in the below tree. The RMSE value and R^2 value for our project in R and Python are –

Decision Tree	R	PYTHON
RMSE Train	503.005411	0.0
RMSE Test	598.3854972	872.348507402487
R^2 Test	0.9821495	0.8210753906837198

2.2.3 <u>Random Forest</u>

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 to build each decision tree. It means to build each decision tree on random forest we are not going to use the same data. The RMSE value and R^2 value for our project in R and Python are –

Random Forest	R	PYTHON
RMSE Train	129.4719764	232.90808643404776
RMSE Test	240.1007197	629.230001844388
R^2 Test	0.9865681	0.9069087468979179

2.2.4 <u>Support vector Regression (SVR)</u>

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

<u>SVR</u>	R	PYTHON
RMSE Train	218.790003	1896.7030417458407
RMSE Test	221.1328272	2055.676382291139
R^2 Test	0.9880324	0.006427584876783965

2.2.5 <u>Gradient boosting</u>

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Gradient boosting	PYTHON
RMSE Train	218.7014471159227
RMSE Test	517.4852151253407
R^2	0.9381141770882626

Chapter 3

Conclusion

In this chapter we are going to evaluate our models, select the best model for our dataset to predict the target variable with good accuracy.

3.1 Model Evaluation

In the previous chapter we have seen the **Root Mean Square Error** (RMSE) and **R-Squared** Value of different 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** and higher value of **R-Squared Value** indicate better fit.

3.2 Model Selection

In case of Python:

From the observation of all **RMSE Value** and **R-Squared** Value we have concluded that **Linear Regression Model** has minimum value of RMSE test and it's **R-Squared** Value is also maximum (i.e. 0.9646) in Python.

The RMSE value of Testing data and Training does not differs a lot this implies that it is not the case of overfitting.

Therefore Multiple Linear Regression is good model in Python.

In case of R:

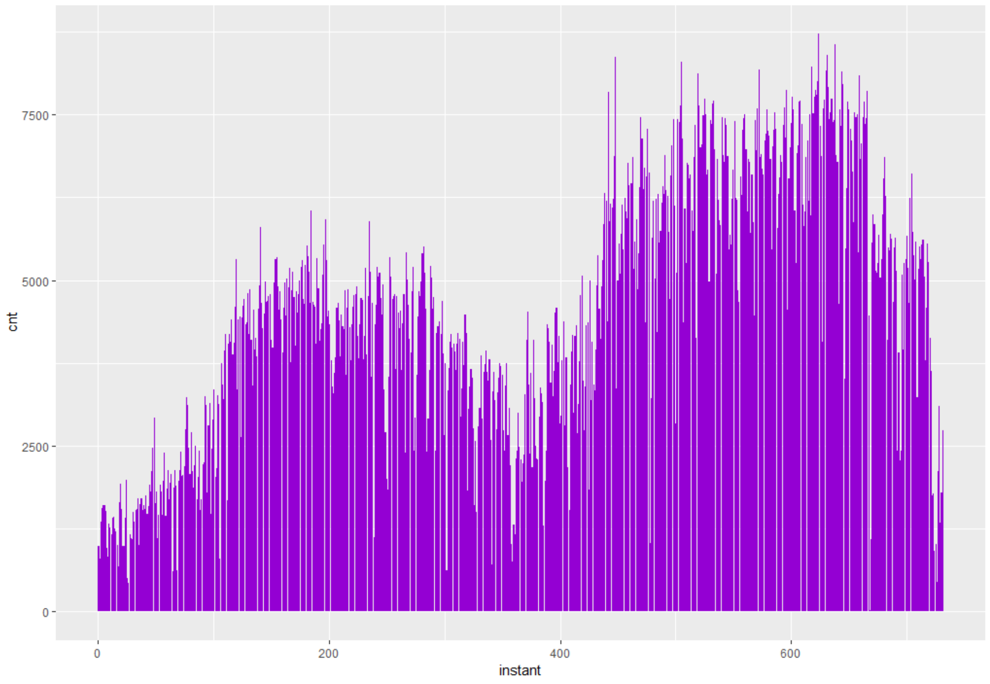
Support Vector Regression has 2nd minimum RMSE test value (221.13) after Linear Regression RMSE test value (212.21) .
But RMSE train value in case of Linear Regression is 265.35, seems like model is under fitting.

And RMSE train value in case of SVR is 218.79 ,which seems like model is not under fitting as well as the RMSE value of Testing data and Training does not differs a lot this implies that it is not the case of overfitting. Its R^2 values is 0.9880324 which is maximum.
Therefore, SVR is good model in R.

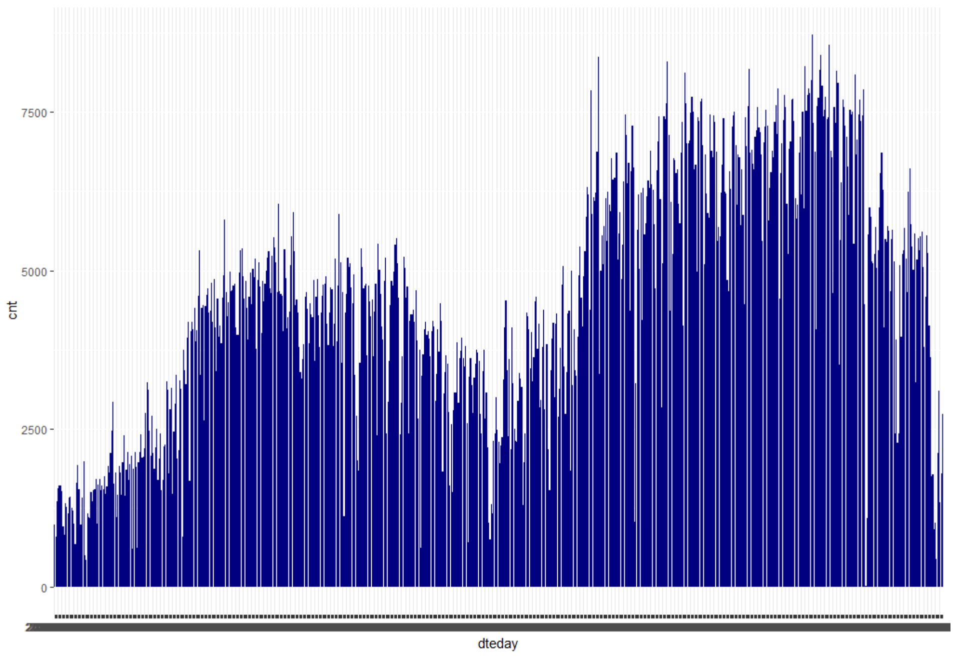
Appendix

Extra Figures

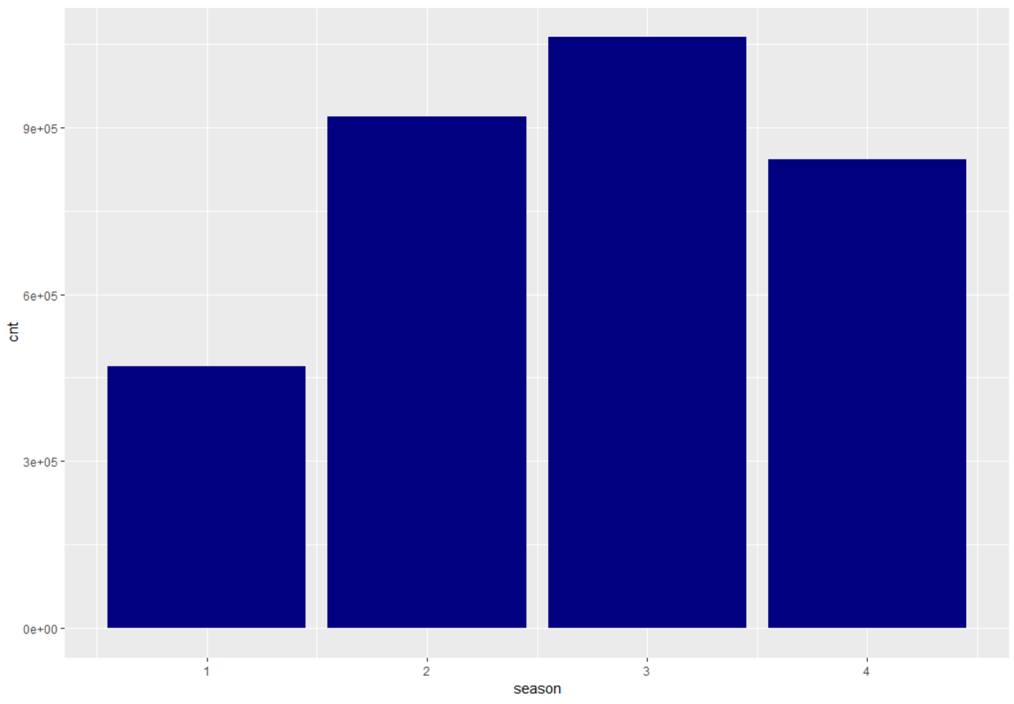
1. **From below graph we can conclude that in 2012 customer take more bike in RENT as compared to 2011 data.**



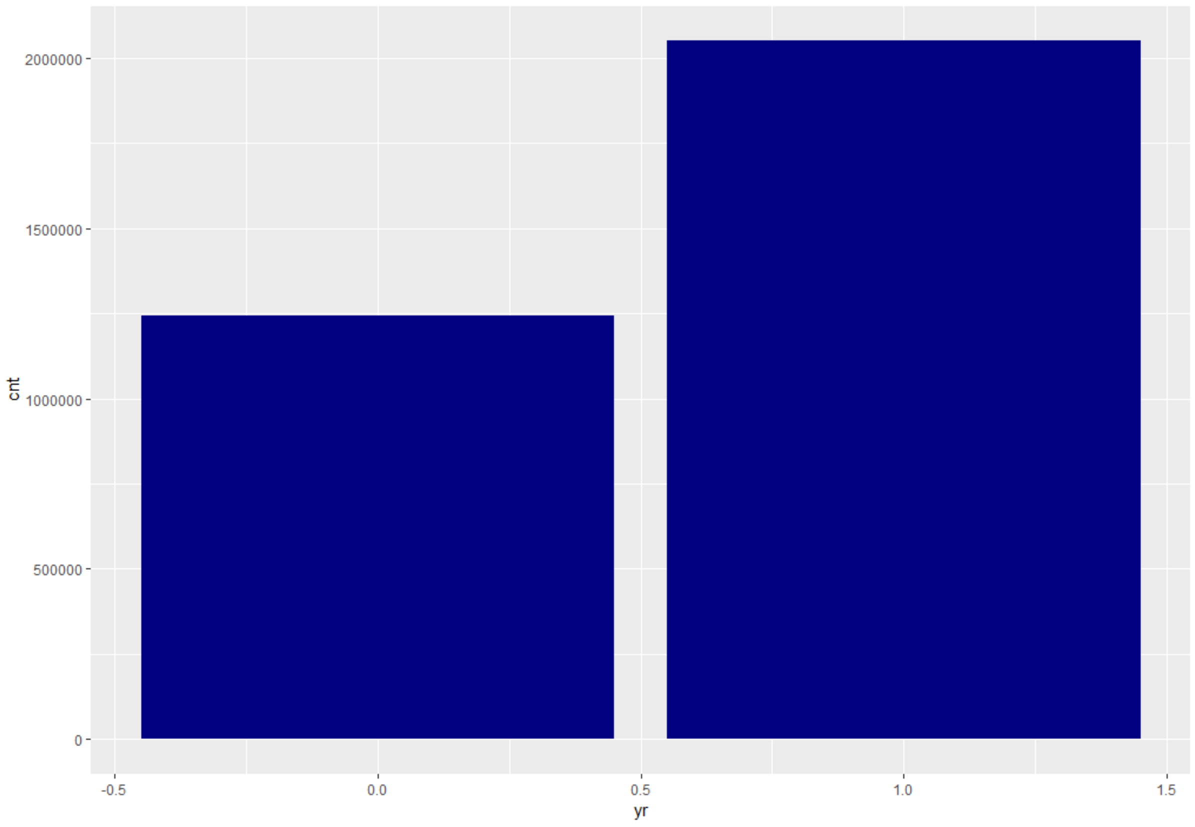
2. **Below graph is similar like above but this is a bar graph of each unique date with cnt.**



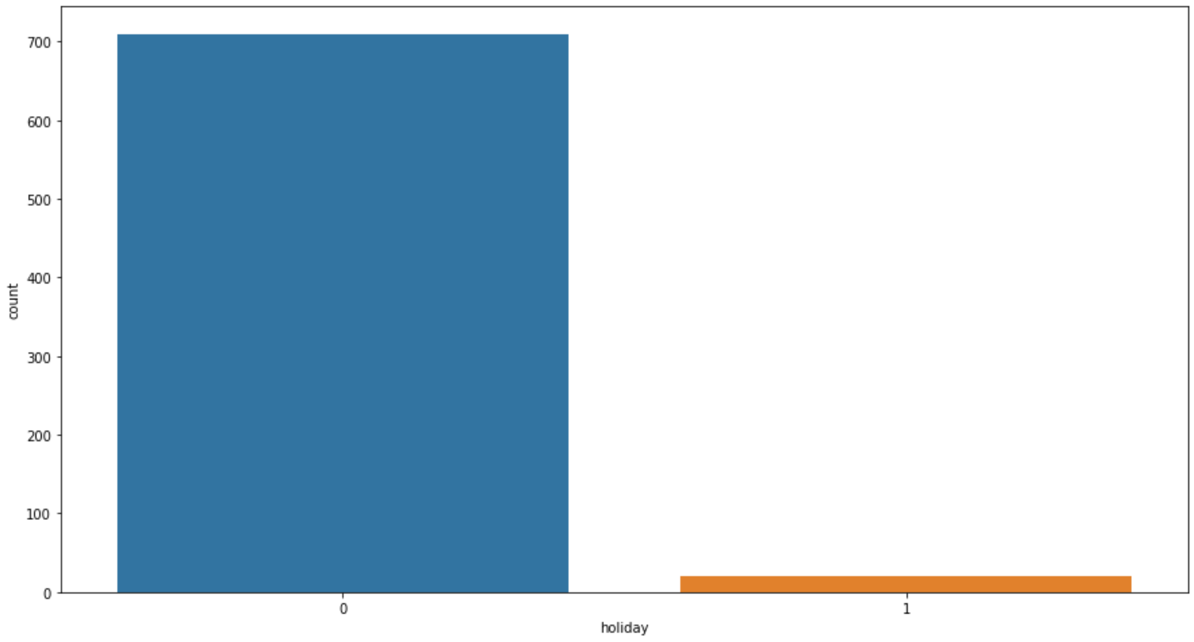
3. _Season (1:springer, 2:summer, 3:fall, 4:winter)
From below graph we can see that in fall (Rainy) season more bikes have been rented.



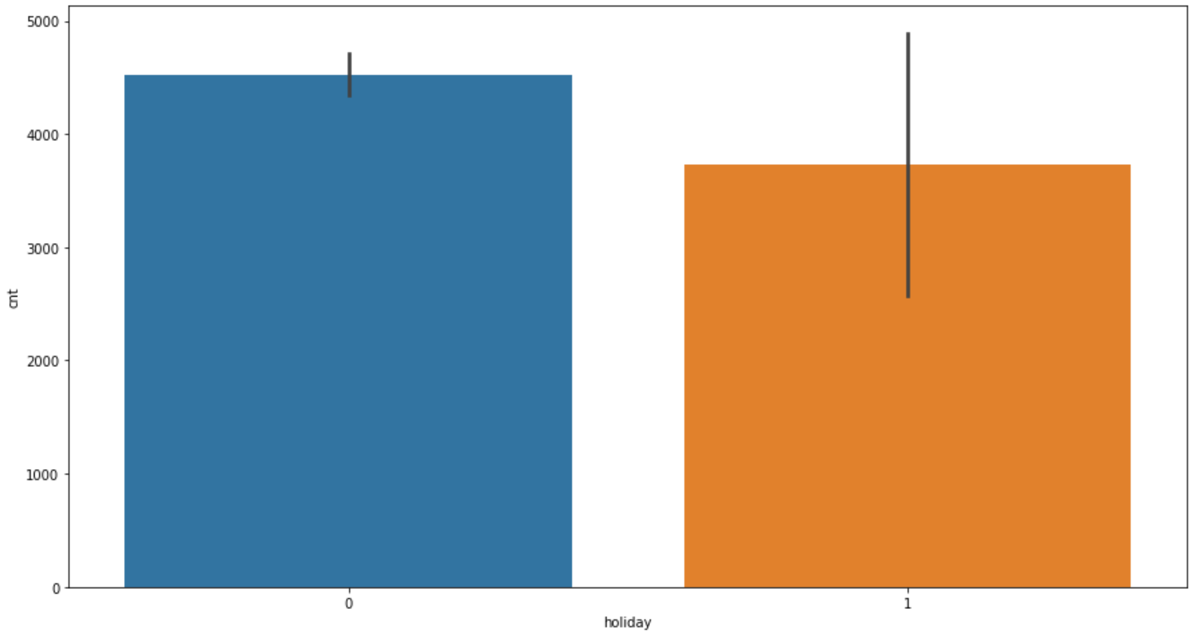
4. **In 2012** **more bikes have been rented.**



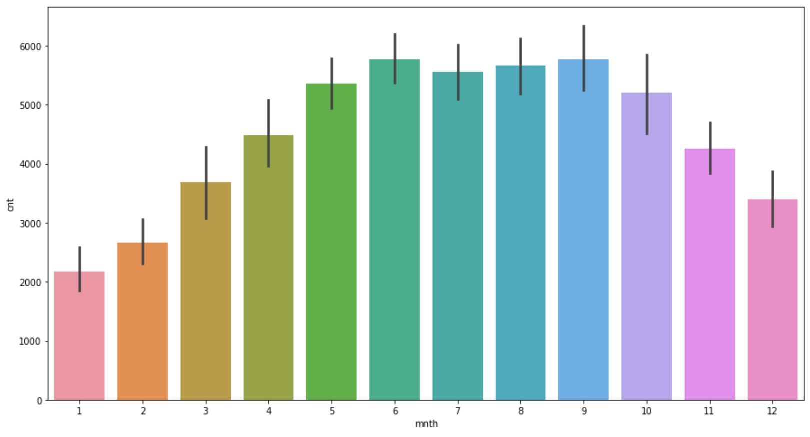
5.



By comparing above and below graph there I much less holiday in one year but from below bar graph in those holidays customer take bike in Rent more as compare to non holiday in Ratio.

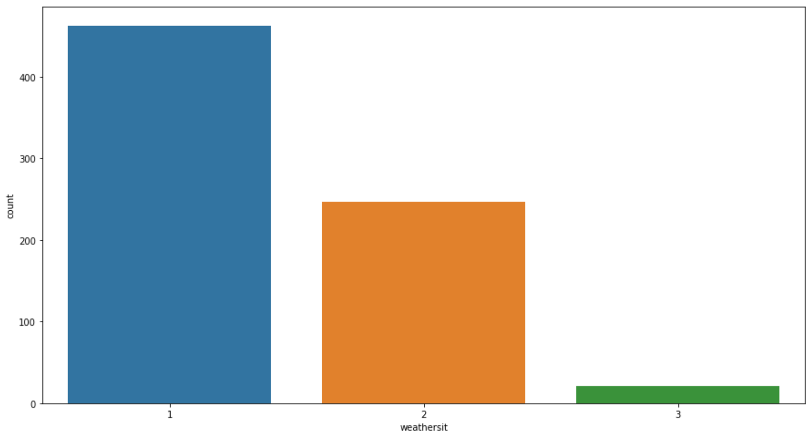


6. In mid year from July to October customer take more bike in Rent.

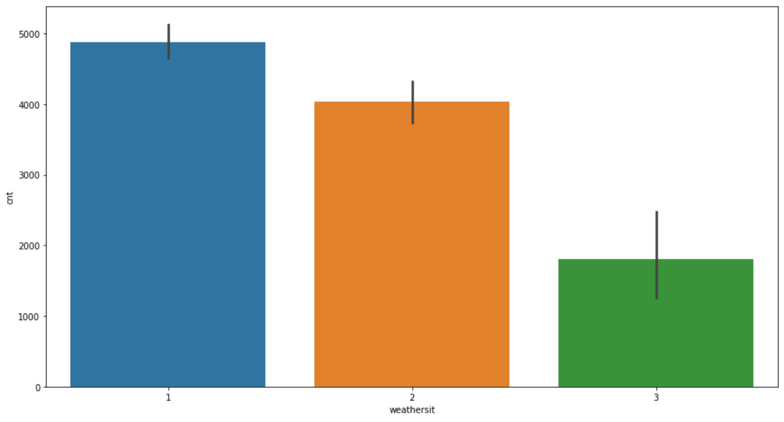


7. 1: Clear, Few clouds, Partly cloudy, Partly cloudy
   2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

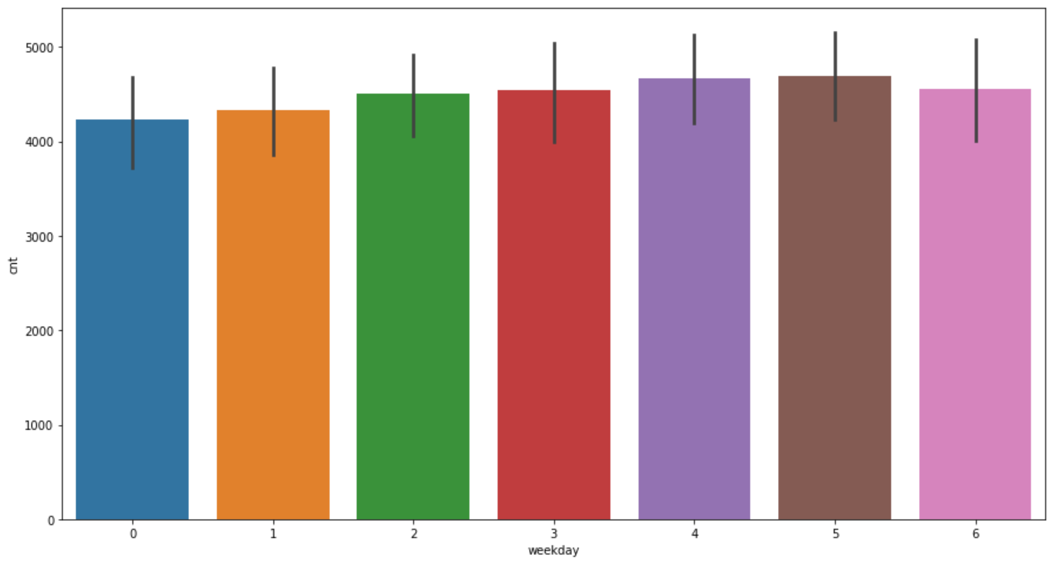
   3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds



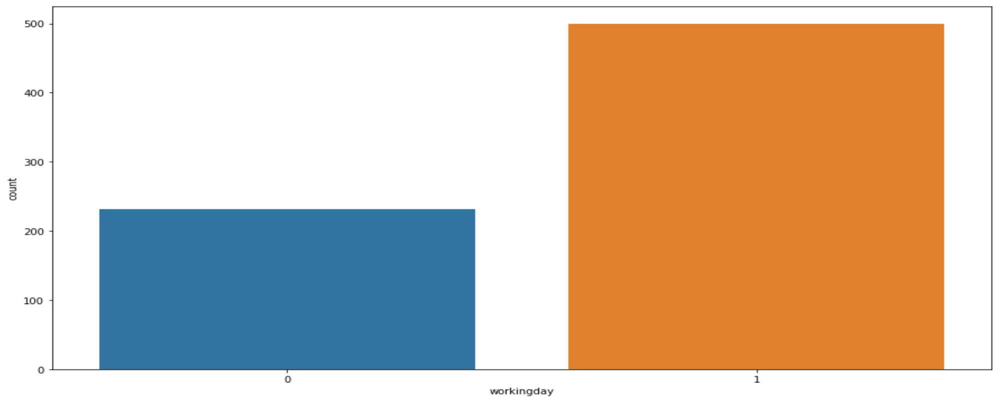
In clear sky with few clouds more bike is rented

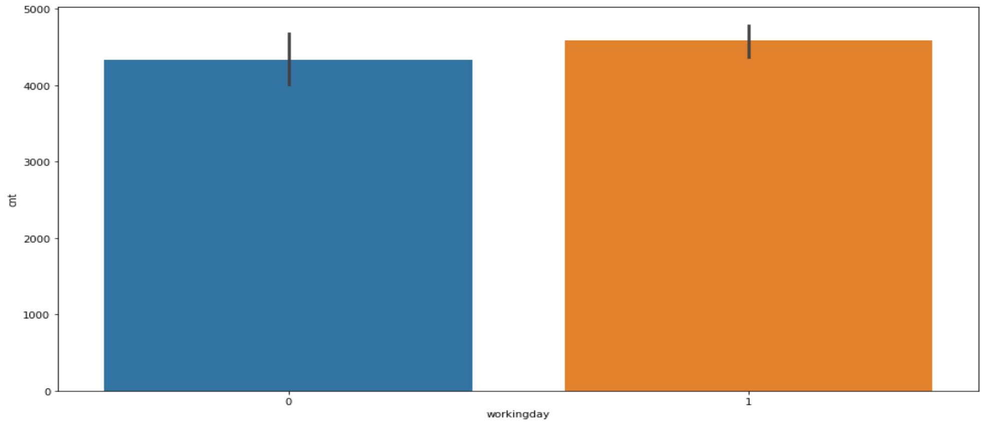


8. **In Saturday more bike is rented.**

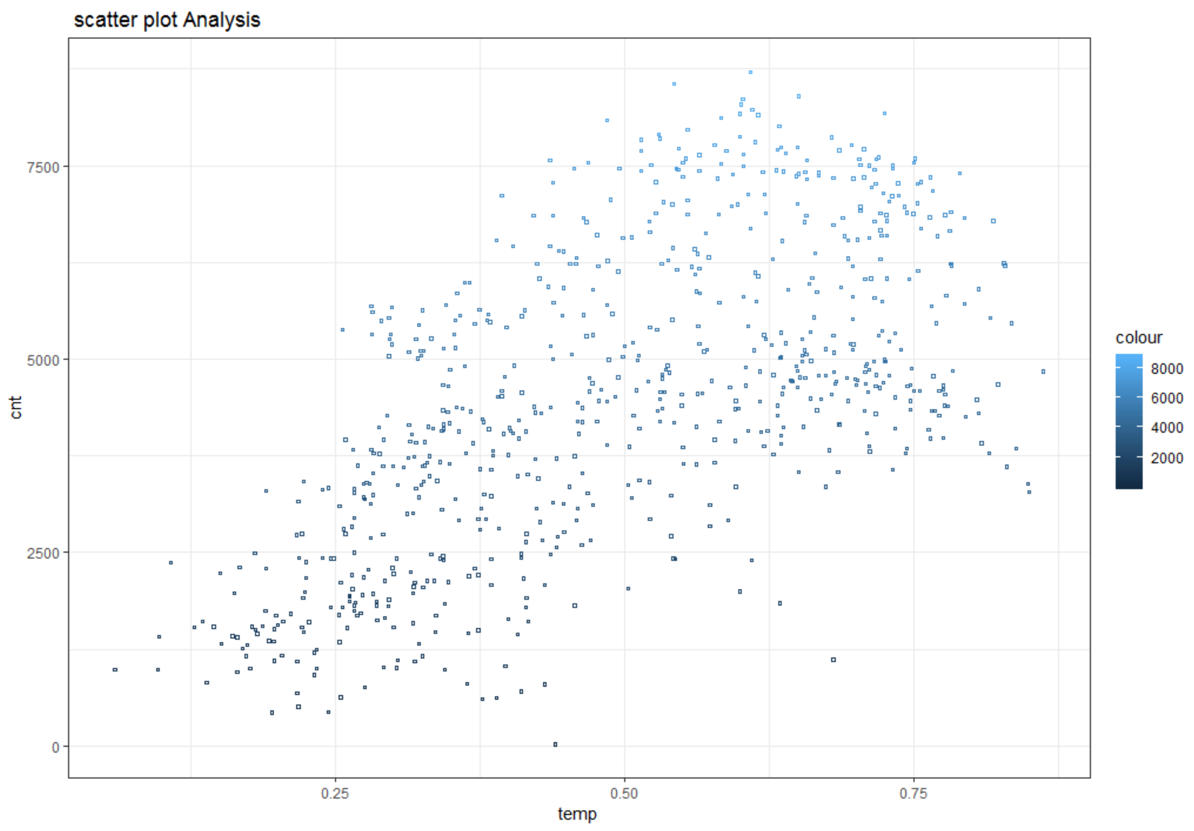


9. In working day is more in a year but bike is rented more in holiday in Ratio by comparing the below two graph.

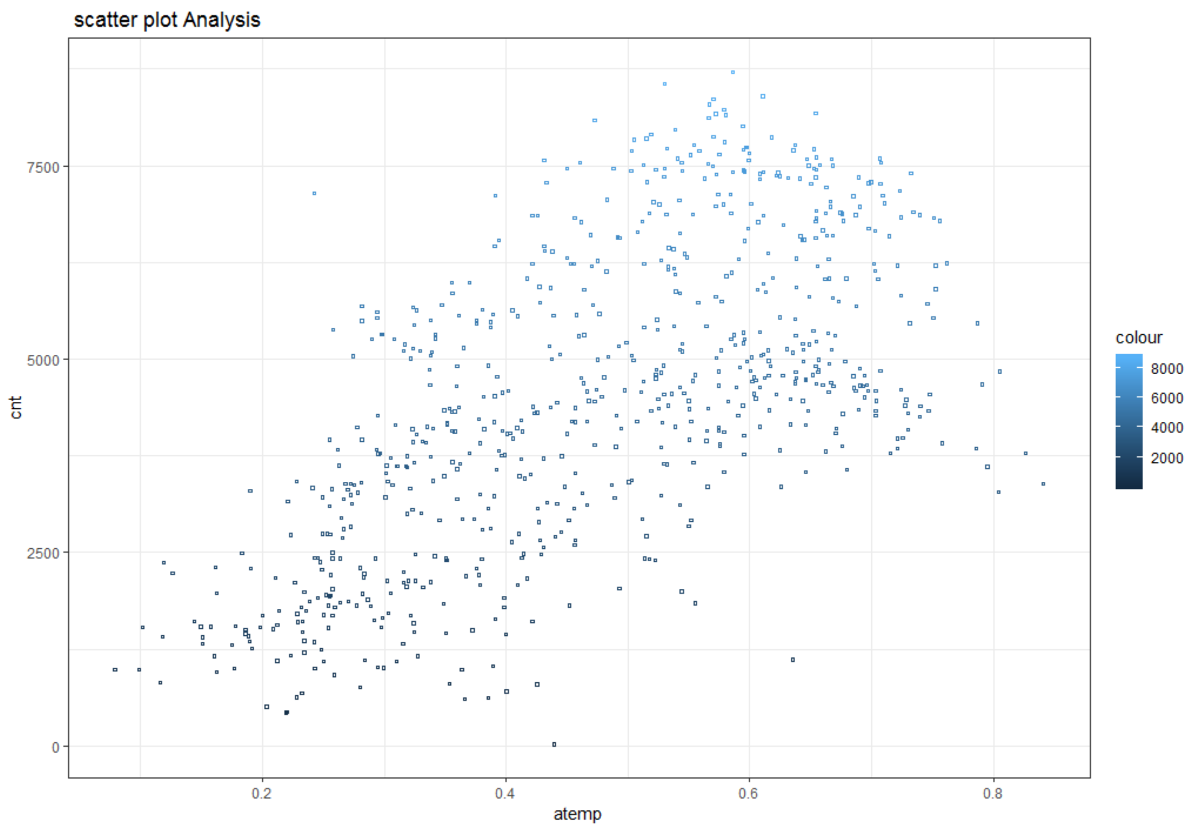




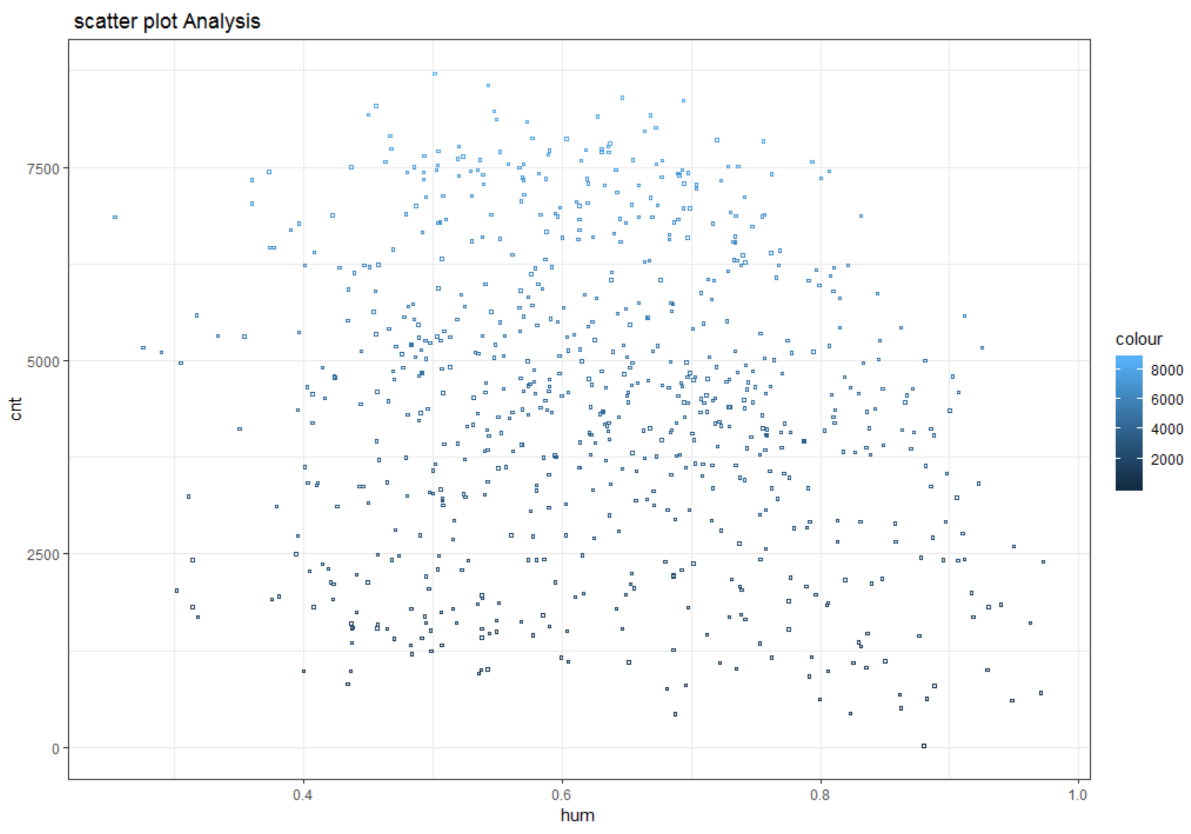
10. Positive correlation between "temp" and "cnt"



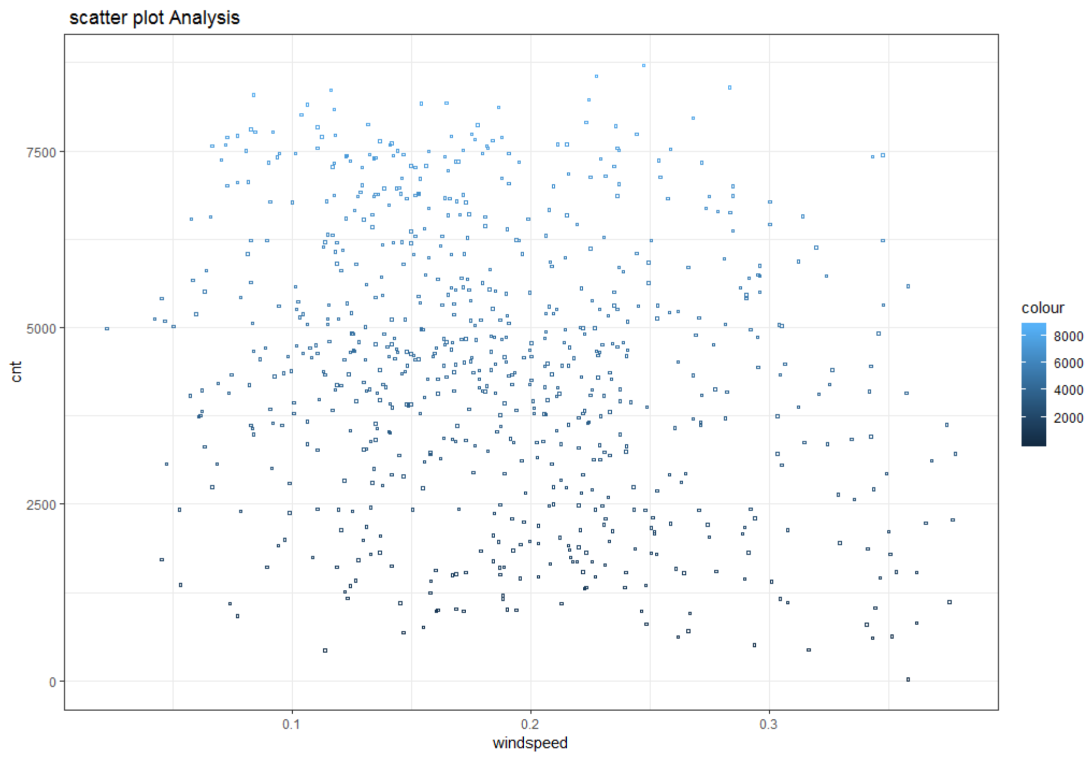
11. Positive correlation between “atemp” and “cnt”



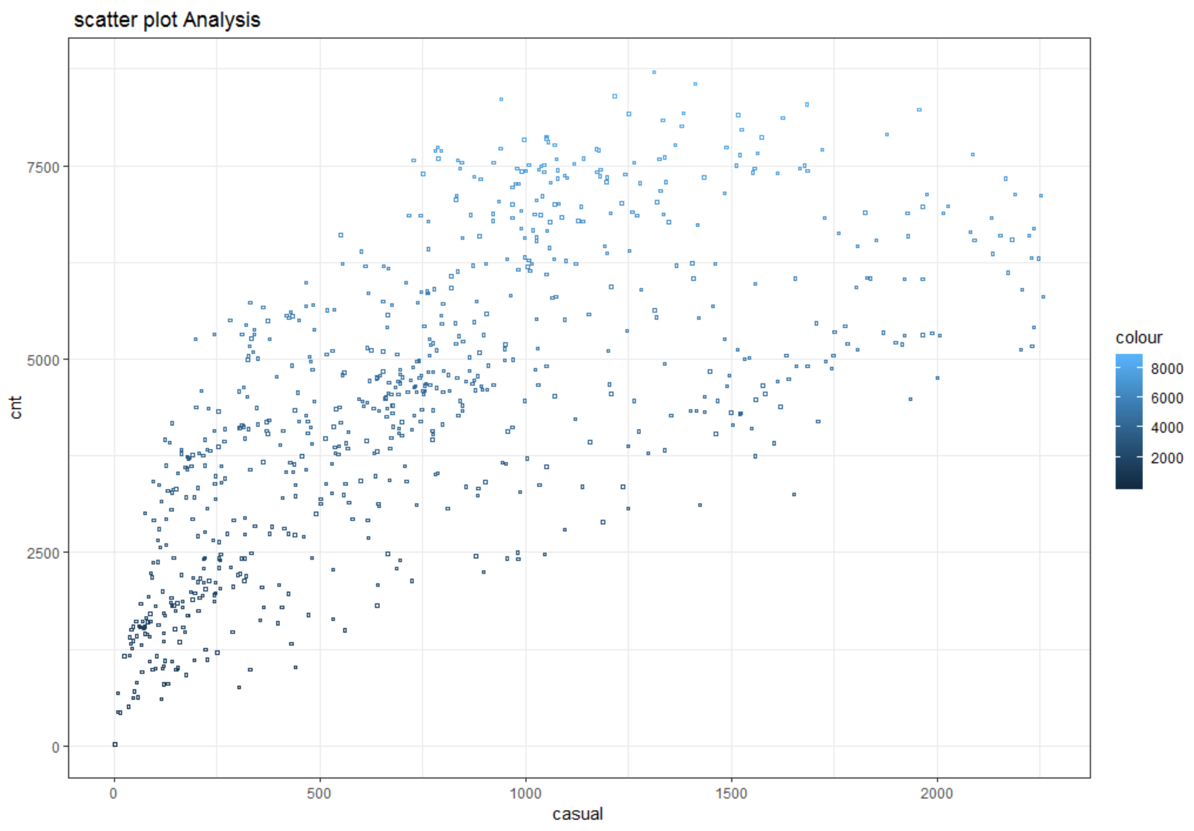
12. No correlation between “hum” and “cnt”



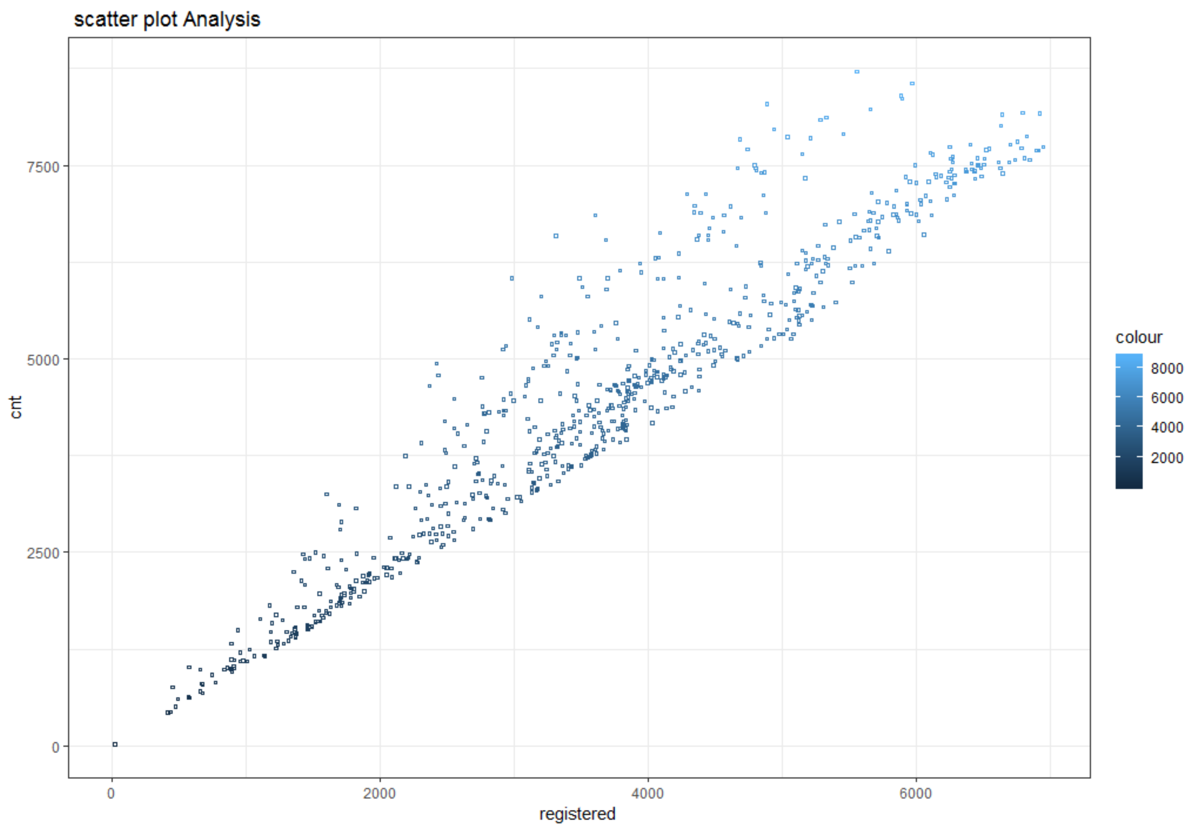
13. No correlation between “windspeed” and “cnt”



14. A slightly positive correlation between “casual’ and “ cnt”.



15. A high positive correlation between “registered” and “cnt”.



PYTHON CODE:

```
# Importing the required libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#Setting the directory for working
os.chdir('C:/Users/Bustec/Desktop/Edwisor project 1')
os.getcwd()

# Importing the required datasets
dataset= pd.read_csv('day.csv')
dataset.head()

# Checking for the Null values
dataset.isnull().sum().sum()

#  Counting of Unique values in each column
dataset.nunique()
```

```
## Import the libraries for visualization
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
##counting the categories of the season
plt.figure(figsize=(14,7))

sns.countplot(x='season',data=dataset)

#bar plot for each of categories of season with cnt
plt.figure(figsize=(14,7))

sns.barplot(x='season', y='cnt',data=dataset,ci=99)

# plot of the 'season'
plt.figure(figsize=(14,7))
# strip
sns.stripplot(x='season', y='cnt',data=dataset,jitter=0.4)

#To visualize the boxplot for concentration of the data in each of the
categories with cnt

plt.figure(figsize=(14,7))
sns.boxplot(x='season', y='cnt',data=dataset)
# ### Plot for the "year"
# counting the each categories of season
plt.figure(figsize=(14,7))
sns.countplot(x='yr',data=dataset)


# Bar plot for the each categories of "year" with "count of total no of
rental bikes (cnt)"
plt.figure(figsize=(14,7))


sns.barplot(x='yr', y='cnt',data=dataset,ci=99)
# plot for Column 'year'
plt.figure(figsize=(14,7))
```

```
# strip
sns.stripplot(x='yr', y='cnt',data=dataset,jitter=0.4)

# Box plot for visualizing the concentration of data with cnt
plt.figure(figsize=(15,8))

sns.boxplot(x='yr', y='cnt',data=dataset)
```

```
dataset.nunique()
#Plot for the Month
# count for the categories of "month"
plt.figure(figsize=(14,7))
sns.countplot(x='mnth',data=dataset)
# Bar plot for each of the categories of "month" with cnt
plt.figure(figsize=(14,7))
sns.barplot(x='mnth', y='cnt',data=dataset,ci=99)

#  Strip plot for the  Column 'month'
plt.figure(figsize=(14,7))
# strip
sns.stripplot(x='mnth', y='cnt',data=dataset,jitter=0.3)

# Box plot for visualize the concentration of  the data in each
categories with ent
plt.figure(figsize=(14,7))
sns.boxplot(x='mnth', y='cnt',data=dataset)

#plot for the  holiday
# counting the categories for "holiday"
plt.figure(figsize=(14,7))


sns.countplot(x='holiday',data=dataset)

# Bar plot for each  of the categories of "holiday" with cnt
plt.figure(figsize=(14,7))

sns.barplot(x='holiday', y='cnt',data=dataset,ci=99)

# plot for the Column 'holiday'
plt.figure(figsize=(14,7))
# strip for the holiday
sns.stripplot(x='holiday', y='cnt',data=dataset,jitter=0.4)

# Box plot for the visualization of the concentration of data in each
categories with cnt
plt.figure(figsize=(14,7))

sns.boxplot(x='holiday', y='cnt',data=dataset)

#Plot for the weekday

# counting the  categories of the "weekday"
plt.figure(figsize=(14,7))
sns.countplot(x='weekday',data=dataset)
```

```
# Bar plot for each of the categories of "weekday" with cnt
plt.figure(figsize=(14,7))

sns.barplot(x='weekday', y='cnt',data=dataset,ci=99)

# plot for  the Column of 'weekday '
plt.figure(figsize=(14,7))
# strip for the weekday
sns.stripplot(x='weekday', y='cnt',data=dataset,jitter=0.4)

# Box plot for visualization of the concentration of data in each
categories with cnt
plt.figure(figsize=(14,7))

sns.boxplot(x='weekday', y='cnt',data=dataset)

dataset.nunique()

# ### Plot for the each workingday

# counting the categories for "workingday"
plt.figure(figsize=(14,7))


sns.countplot(x='workingday',data=dataset)

# Bar plot for each categories of the "workingday" with cnt

plt.figure(figsize=(14,7))

sns.barplot(x='workingday', y='cnt',data=dataset,ci=99)

# plot for the each Column workingday
plt.figure(figsize=(14,7))

# strip for the workingday
sns.stripplot(x='workingday', y='cnt',data=dataset,jitter=0.4)

# Box plot fot the visualization of the concentration of data in each
categories with cnt

plt.figure(figsize=(14,7))

sns.boxplot(x='workingday', y='cnt',data=dataset)

#Plot for weathersit

# counting categories of weathersit
plt.figure(figsize=(14,7))


sns.countplot(x='weathersit',data=dataset)

# Bar plot for each of the categories of "weathersit" with cnt
plt.figure(figsize=(14,7))
```

```
sns.barplot(x='weathersit', y='cnt',data=dataset,ci=99)

# plot for Column 'weathersit'
plt.figure(figsize=(14,7))

# strip for the weathersit
sns.stripplot(x='weathersit', y='cnt',data=dataset,jitter=.4)

# Box plot for the visualization of the concentration of data in each
categories with cnt
plt.figure(figsize=(14,7))

sns.boxplot(x='weathersit', y='cnt',data=dataset)

#Plot for temp

# Bar plot for each categories of "temp" with cnt
plt.figure(figsize=(800,7))

sns.barplot(x='temp', y='cnt',data=dataset,ci=9)

dataset['temp'].nunique()
```

```
# plot for the Column 'temp'
plt.figure(figsize=(14,7))
# strip for temp
sns.stripplot(x='temp', y='cnt',data=dataset,jitter=0.0)
#Plot for  the "atemp"

# plot for the  Column 'atemp'
plt.figure(figsize=(14,7))
# strip for the atemp
sns.stripplot(x='atemp', y='cnt',data=dataset,jitter=0.4)
dataset.head()

#Plot for the "humidity"

# plot for the Column 'humidity'
plt.figure(figsize=(14,7))
# strip for the humidity
sns.stripplot(x='hum', y='cnt',data=dataset,jitter=0.0)

dataset.nunique()

#Plot for the windspeed

# plot for Column 'windspeed'
plt.figure(figsize=(14,7))
# strip for the windspeed
sns.stripplot(x='windspeed', y='cnt',data=dataset,jitter=0.1)

# plot for the Column 'casual'
plt.figure(figsize=(14,7))
# strip for the casual
sns.stripplot(x='casual', y='cnt',data=dataset,jitter=0.4)

# Bar plot for each categories of registered count
```

```
plt.figure(figsize=(300,8))

sns.barplot(x='registered', y='cnt',data=dataset,ci=99)

# plot for Column 'registered '
plt.figure(figsize=(14,7))
# strip for the registered
sns.stripplot(x='registered', y='cnt',data=dataset,jitter=0.0)

# Box plot for visualize the concentration of data in each categories
with cnt
plt.figure(figsize=(14,7))

sns.boxplot(x='registered', y='cnt',data=dataset)

#check correlation between the Independent Numeric variables

# plot for the registered and cnt
plt.figure(figsize=(10,8))
# strip
sns.stripplot(x='registered', y='casual',data=dataset,jitter=0.0)

#Missing Value Analysis

# Checking Null values
dataset.isnull().sum().sum()

#Outlier Analysis

#Checking the outlier values with the Box Plot

# seperating the categorical and numerical columns
numeric_columns=['temp', 'atemp', 'hum', 'windspeed','casual',
'registered']
categorical_columns=['instant','season', 'yr', 'mnth', 'holiday',
'weekday', 'workingday', 'weathersit']

dataset.columns

dataset.head()

#now we can find the outlier with the boxplot
    sns.boxplot(dataset[i])
    plt.title("Checking outlier values for Variable " +str(i))
    plt.ylabel("Density")
    plt.show()

# copy the dataset
dataset_1 = dataset.copy()
#dataset = dataset_1.copy()

# these are the variables with the outliers

numeric_columns_outliers=['hum', 'windspeed','casual']

#Removing the outliers values
```

```
for i in numeric_columns_outliers:
    print(i)
     q75,q25=np.percentile(dataset.loc[:,i],[75,25])
     iqr=q75-q25

     min=q25-(iqr*1.5)
     max=q75+(iqr*1.5)
     print(min)
     print(max)

#Droping the outliers values

dataset=dataset.drop(dataset[dataset.loc[:,i]<min].index)
dataset=dataset.drop(dataset[dataset.loc[:,i]>max].index)

dataset.shape

# Detect the outliers with the value NA

for i in numeric_columns_outliers:
    print(i)
    q75,q25=np.percentile(dataset.loc[:,i],[75,25])
    iqr=q75-q25
    min=q25-(iqr*1.5)
    max=q75+(iqr*1.5)
    print(min)
    print(max)

    dataset.loc[dataset[i]< min,i] = np.nan
    dataset.loc[dataset[i]> max,i] = np.nan
# Checking if there is any missing value in the
dataset.isnull().sum().sum()

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

#checking missing values after  replacing the outliers with NA through
the Heat Map
plt.figure(figsize=(10,5))
sns.heatmap(dataset.isnull(), cmap="viridis")
print("Total NA value in the dataset is ="
+str(dataset.isnull().sum().sum()))

for i in numeric_columns_outliers:
    print(i)
dataset.loc[:,i]=dataset.loc[:,i].fillna(dataset.loc[:,i].mean())

#checking missing values after the Imputation by Mean Method
plt.figure(figsize=(10,5))
sns.heatmap(dataset.isnull(), cmap="viridis")
print("Total NA value in the dataset is ="
+str(dataset.isnull().sum().sum()))

#to check outlier value in dataset
for i in numeric_columns_outliers:
    sns.boxplot(dataset[i])
    plt.title("Checking outlier values for Variable " +str(i))
    plt.ylabel("Density")
    plt.show()

numeric_columns

#Correlation Analysis
dataset_corr=dataset.loc[:,numeric_columns]

# set the width and height of plot
f, ax=plt.subplots(figsize=(10,10))

# Generate the correlation Matrix
corr=dataset_corr.corr()

#plot using the seaborn Library
sns.heatmap(corr,mask=np.zeros_like(corr,dtype=np.bool),
cmap=sns.diverging_palette(250,10,as_cmap=True),
           square=True,ax=ax,annot = True)



# Loading Library for calculating VIF of each numeric columns.
from statsmodels.stats.outliers_influence import
variance_inflation_factor


# For each X, calculate VIF and save in dataframe
vif = pd.DataFrame()
vif["VIF Factor"] =
[variance_inflation_factor(dataset[numeric_columns].values, i)
for i in range(dataset[numeric_columns].shape[1])]
vif["features"] = dataset[numeric_columns].columns
vif

###### Here, VIF for "temp" and "atemp" are high. This means both are
highly correlated.

# ### 1.3.2 ANOVA Analysis

categorical_columns1=['instant','season', 'yr', 'mnth', 'holiday',
'weekday', 'workingday', 'weathersit']


#loop for ANOVA test Since the target variable is continuous
for i in categorical_columns1:
    f, p = stats.f_oneway(dataset[i], dataset["cnt"])
    print("P value for variable "+str(i)+" is "+str(p))

# Here P value for all the variables are zero.
# H0= Independent varaible does not explain our Target variable.

# H1= Independent varaible explain our Target variable.


# Droping the variables which are not Important
dataset_del = dataset.drop(['instant',"atemp", "dteday"], axis = 1)
```

```
dataset_del.columns

# copy dataset
dataset_2=dataset_del.copy()
#dataset_del=dataset_2.copy()

#Feature scaling
numeric_columns=['temp', 'hum', 'windspeed','casual', 'registered']

dataset[numeric_columns].head()

numeric_columns_scale=['casual', 'registered']

# Checking if there is any normally distributed variable in the data
for i in numeric_columns_scale:
    if i == 'cnt':
        continue
    sns.distplot(dataset[i],bins = 'auto')
    plt.title("Checking Distribution for Variable "+str(i))
    plt.ylabel("Density")
    plt.show()

#Since there is no normally distributed curve we will use Normalizationg
for Feature Scalling
# #Normalization

for i in numeric_columns_scale:
    if i == 'Absenteeism time in hours':
        continue
    dataset_del[i] = (dataset_del[i] -
dataset_del[i].min())/(dataset_del[i].max()-dataset_del[i].min())

dataset_del[numeric_columns].head()

# copying dataset
dataset_3=dataset_del.copy()
#dataset_del=dataset_3.copy()

# # 2 Machine Leaning Model
# Important categorical variable to convert into dummy variables.
categorical_columns=['season','mnth', 'weekday', 'weathersit']
dataset_del[categorical_columns].nunique()

# Get the dummy variables for the categorical variables
dataset_del = pd.get_dummies(data = dataset_del, columns =
categorical_columns )

# Copying dataframe
dataset_4 = dataset_del.copy()
dataset_del.head()
dataset_del.nunique()
### AVoiding dummy variable trap
# selecting the columns to Avoid the dummy variable trap
drop_columns=['season_1', 'mnth_1', 'weekday_0', 'weathersit_1']
dataset_del = dataset_del.drop(drop_columns, axis = 1)
dataset_del.shape
dataset_del.head()

# splitting the dataset into X and y
X = dataset_del.drop("cnt", axis = 1)
y = dataset_del.iloc[:,8].values

X=pd.DataFrame(X)

X.head()

# splitting the dataset into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test = train_test_split(X,y,test_size=0.20,
random_state=0)

X_train.shape

X_test.shape

# ##Multiple Linear Regression
# Importing Library for Linear Regression
from sklearn.linear_model import LinearRegression
# Fitting simple linear regression to the training data

regressor=LinearRegression()
LR_model=regressor.fit(X_train,y_train)

# Applying the k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = LR_model, X = X_train, y =
y_train, cv =10 )
accuracy=accuracies.mean()
print(' Accuracy of Multiple Linear Regression ='+str(accuracy))
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std()))

# Importing Library for the Linear regression
mport statsmodels.api as sm


# train the models using training set
LR_model = sm.OLS(y_train, X_train).fit()
LR_model.summary()

# predicting the test results that are setted
y_pred= LR_model.predict(X_test)
y_pred

# Calculate MAPE
def mape(y_test,y_pred):
    mape = np.mean(np.abs((y_test - y_pred)/y_test))*100
    return mape
mape(y_test,y_pred)

# Calculating RMSE for training data to check for over fitting
from sklearn.metrics import mean_squared_error
pred_train = LR_model.predict(X_train)
```

```
rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

# Calculating  the RMSE for test data for checking accuracy
y_pred= LR_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))

# calculate  the R^2 value to check the goodness of fit
from sklearn.metrics import r2_score
print("R^2 Score(coefficient of determination) =
"+str(r2_score(y_test,y_pred)))

dataset.shape

# Importing Library for the Decision Tree

from sklearn.tree import DecisionTreeRegressor

regressor = DecisionTreeRegressor(random_state = 0)
DT_model=regressor.fit(X_train, y_train)

# Predicting the new result
y_pred = DT_model.predict(X_test)
y_pred

y_test

# Applying k-Fold Cross Validation

from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = DT_model, X = X_train, y =
y_train, cv =10 )
accuracy=accuracies.mean()
print(' Accuracy ='+str(accuracy))
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std())

# Calculate the MAPE
def mape(y_test,y_pred):
    mape = np.mean(np.abs((y_test - y_pred)/y_test))*100
    return mape
mape(y_test,y_pred)

# Calculating RMSE for training data to check for the over fitting
from sklearn.metrics import mean_squared_error
pred_train = DT_model.predict(X_train)
rmse_for_train = np.sqrt(mean_squared_error(y_train, pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

# Calculating the RMSE for test data to check accuracy
y_pred= DT_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))

# calculate the tR^2 value to check the goodness of fit
from sklearn.metrics import r2_score
```

```
print("R^2 Score(coefficient of determination) =
"+str(r2_score(y_test,y_pred)))

# ##### Difference of RMSE train and test is High. Therefore, Decision
Tree model is overfitting.

#Random Forest

# Importing Library for Random Forest

from sklearn.ensemble import RandomForestRegressor
# Fitting Random Forest Regression to the dataset
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
RF_model=regressor.fit(X_train, y_train)

# Predicting a new result
y_pred = regressor.predict(X_test)
y_pred

y_test

# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = RF_model, X = X_train, y =
y_train, cv =10 )
accuracy=accuracies.mean()
print(' Accuracy ='+str(accuracy))
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std()))

# Calculate MAPE
def mape(y_test,y_pred):
    mape = np.mean(np.abs((y_test - y_pred)/y_test))*100
    return mape

mape(y_test,y_pred)

# Calculating the RMSE for training data to check for over fitting
from sklearn.metrics import mean_squared_error
pred_train = RF_model.predict(X_train)
rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

# Calculating RMSE for test data to check accuracy
y_pred= RF_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))

# calculate the R^2 value to check the goodness of fit

from sklearn.metrics import r2_score
print("R^2 Score(coefficient of determination) =
"+str(r2_score(y_test,y_pred)))

#Gradiet Boosting

# Importing library for Gradient Boosting
```

```
from sklearn.ensemble import GradientBoostingRegressor
# Building model on top of training dataset

GB_model = GradientBoostingRegressor().fit(X_train, y_train)

# Predicting a new result
y_pred = GB_model.predict(X_test)
y_pred


# Applying k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = GB_model, X = X_train, y =
y_train, cv =10 )
accuracy=accuracies.mean()
print(' Accuracy ='+str(accuracy))
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std()))

# Calculate the MAPE
def mape(y_test,y_pred):
    mape = np.mean(np.abs((y_test - y_pred)/y_test))*100
    return mape

mape(y_test,y_pred)

# Calculating the RMSE for training data to check for over fitting
pred_train = GB_model.predict(X_train)
rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

# Calculating the RMSE for test data to check accuracy
pred_test = GB_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,pred_test))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))

# calculate the R^2 value to check the goodness of fit
print("R^2 Score(coefficient of determination) =
"+str(r2_score(y_test,pred_test)))

#Importing Library for SVR
from sklearn.svm import SVR


# Fitting SVR to the dataset
regressor = SVR(kernel = 'rbf')
SVR_model=regressor.fit(X_train,y_train)

# predicting the test results
y_pred= SVR_model.predict(X_test)
y_pred

# Applying the k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = regressor, X = X_train, y =
y_train, cv =10 )
```

```
accuracy=accuracies.mean()
print(' Accuracy ='+str(accuracy))
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std()))

#Calculating the RMSE for test data to check accuracy

from sklearn.metrics import mean_squared_error
y_pred= SVR_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))

# Calculating RMSE for training data to check for over fitting
from sklearn.metrics import mean_squared_error
pred_train = SVR_model.predict(X_train)
rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

# calculate  the R^2 value to check the goodness of fit
from sklearn.metrics import r2_score
print("R^2 Score(coefficient of determination) =
"+str(r2_score(y_test,y_pred)))

# Copying dataframe
#dataset1 = dataset.copy()
#dataset = dataset1.copy()
dataset_del.shape
X_test.shape

# splitting the dataset into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test = train_test_split(X,y,test_size=0.20,
random_state=0)

# Performing PCA on the Train and test data seperately
from sklearn.decomposition import PCA


#Data has 92 variables so no of components of PCA = 92
pca=PCA(n_components=30)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

# The amount of variance that each PC explains
explained_variance = pca.explained_variance_ratio_

# Cumulative Variance explains
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
plt.plot(var1)
plt.show()

pca = PCA(n_components=26)


# Fitting the selected components to the data
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

```
# Building the optimal model using Backward elimination
import statsmodels.formula.api as sm

X_train=np.append(arr=np.ones((584,1)).astype(int), values=X_train
,axis=1)
X_train=pd.DataFrame(X_train)
# selecting training columns
X_opt = X_train.iloc[:, 0:30]

regressor_OLS=sm.OLS(endog=y_train, exog=X_opt).fit()
regressor_OLS.summary()


X_train.shape
X_train = X_train.drop([16], axis = 1)
X_train.shape

# Building the optimal model using Backward elimination
import statsmodels.formula.api as sm


# selecting training columns
X_opt = X_train.iloc[:, 0:30]


regressor_OLS=sm.OLS(endog=y_train, exog=X_opt).fit()
regressor_OLS.summary()

# Building the optimal model using Backward elimination
import statsmodels.formula.api as sm

X_test=np.append(arr=np.ones((147,1)).astype(int), values=X_test ,axis=1)
X_test=pd.DataFrame(X_test)
# selecting training columns
X_opt = X_test.iloc[:, 0:30]


regressor_OLS=sm.OLS(endog=y_test, exog=X_opt).fit()
regressor_OLS.summary()

# Dropping the same columns in  test dataset
X_test=pd.DataFrame(X_test)
X_test = X_test.drop([16], axis = 1)
X_test.shape

X_train=pd.DataFrame(X_train)
X_train.shape

y.shape
y_test.shape
```

```
#Multiple Linear Regression after Dimensionality Reduction

# Importing Library for Linear Regression
from sklearn.linear_model import LinearRegression

# Fitting simple linear regression to the training data
regressor=LinearRegression()
LR_model=regressor.fit(X_train,y_train)

# predicting the test set results
y_pred= LR_model.predict(X_test)
y_pred

# Applying the k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = LR_model, X = X_train, y =
y_train, cv =10 )
accuracy=accuracies.mean()
print(' Accuracy of Multiple Linear Regression ='+str(accuracy))
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std()))

# Importing the Library for Linear regression
import statsmodels.api as sm

# train the model using training set
LR_model = sm.OLS(y_train, X_train).fit()
LR_model.summary()

# Calculate the  MAPE
def mape(y_test,y_pred):
    mape = np.mean(np.abs((y_test - y_pred)/y_test))*100
    return mape
mape(y_test,y_pred)

# Calculating RMSE for training data to check for over fitting
pred_train = LR_model.predict(X_train)
rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

from sklearn.metrics import mean_squared_error
y_pred= LR_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))

# calculate the R^2 value to check the goodness of fit
from sklearn.metrics import r2_score
print("R^2 Score(coefficient of determination) =
"+str(r2_score(y_test,y_pred)))

#Decision Tree after Dimensionality Reduction


# Importing Library for Decision Tree

from sklearn.tree import DecisionTreeRegressor
```

```
regressor = DecisionTreeRegressor(random_state = 0)
DT_model=regressor.fit(X_train, y_train)
DT_model

#predicting the new result
y_pred = DT_model.predict(X_test)
y_pred

# Apply the k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = DT_model, X = X_train, y =
y_train, cv =5 )
accuracy=accuracies.mean()
print(' Accuracy of Decision Tree='+str(accuracy))
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std()))

# Calculate MAPE
def mape(y_test,y_pred):
    mape = np.mean(np.abs((y_test - y_pred)/y_test))*100
    return mape
mape(y_test,y_pred)

# Calculating the RMSE for training data to check for over fitting.
from sklearn.metrics import mean_squared_error
pred_train = DT_model.predict(X_train)
rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

# Calculating the RMSE for test data to check accuracy
y_pred= DT_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))


from sklearn.metrics import r2_score
print("R^2 Score(coefficient of determination) =
"+str(r2_score(y_test,y_pred)))

#Decision tree model is completely overfitting

# Importing Library for Random Forest
from sklearn.ensemble import RandomForestRegressor
# Fitting Random Forest Regression to the dataset
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
RF_model=regressor.fit(X_train, y_train)

# Predicting the new result
y_pred = regressor.predict(X_test)
y_pred

# Apply the k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = RF_model, X = X_train, y =
y_train, cv =5 )
accuracy=accuracies.mean()
print(' Accuracy of Random forest ='+str(accuracy))
```

```python
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std()))

#Calculate MAPE
def mape(y_test,y_pred):
    mape = np.mean(np.abs((y_test - y_pred)/y_test))*100
    return mape
mape(y_test,y_pred)

# Calculating RMSE for training data to check for over fitting
from sklearn.metrics import mean_squared_error
pred_train = RF_model.predict(X_train)
rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

# Calculating RMSE for test data to check accuracy
y_pred= RF_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))

# calculate the  R^2 value to check the goodness of fit
from sklearn.metrics import r2_score
print("R^2 Score(coefficient of determination) =
"+str(r2_score(y_test,y_pred)))

#Gradiet Boosting after Dimensionality Reduction

# Importing library for Gradient Boosting
from sklearn.ensemble import GradientBoostingRegressor
# Building model on top of training dataset
GB_model = GradientBoostingRegressor().fit(X_train, y_train)

#Predicting a new result
y_pred = GB_model.predict(X_test)
y_pred

# Applying the k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = GB_model, X = X_train, y =
y_train, cv =10 )
accuracy=accuracies.mean()
print(' Accuracy of Gradiet Boosting ='+str(accuracy))
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std()))

# Calculate MAPE
def mape(y_test,y_pred):
    mape = np.mean(np.abs((y_test - y_pred)/y_test))*100
    return mape
mape(y_test,y_pred)

# Calculating the RMSE for training data to check for over fitting
pred_train = GB_model.predict(X_train)
rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

# Calculating RMSE for test data to check accuracy
```

```
pred_test = GB_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,pred_test))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))

# calculate R^2 value to check the goodness of fit
print("R^2 Score(coefficient of determination) =
"+str(r2_score(y_test,pred_test)))

#Support the Vector Regression after Dimensionality Reduction

# Importing Library for SVR
from sklearn.svm import SVR

#Fitting SVR to the dataset
regressor = SVR(kernel = 'rbf')
SVR_model=regressor.fit(X_train,y_train)
SVR_model

# predicting the test results
y_pred= SVR_model.predict(X_test)
y_pred

# Apply the k-Fold Cross Validation
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator =SVR_model, X = X_train, y =
y_train, cv =5 )
accuracy=accuracies.mean()
print(' Accuracy of SVR_Model ='+str(accuracy))
print(' Accuracy of all  the partitions='+str(accuracies))
print('Accuracy standard Deviation = ' + str(accuracies.std()))

# Calculating the RMSE value for training data to check for over fitting
from sklearn.metrics import mean_squared_error
pred_train = SVR_model.predict(X_train)
rmse_for_train = np.sqrt(mean_squared_error(y_train,pred_train))
print("Root Mean Squared Error For Training data = "+str(rmse_for_train))

# Calculating the RMSE value for test the data to check accuracy
from sklearn.metrics import mean_squared_error
y_pred= SVR_model.predict(X_test)
rmse_for_test =np.sqrt(mean_squared_error(y_test,y_pred))
print("Root Mean Squared Error For Test data = "+str(rmse_for_test))
0
```

<u>R CODE:</u>

```
# clean the environment
rm(list=ls(all=T))
install.packages(x)
Loading packeges
x=c("ggplot2","lattice","gridExtra","corrgram","DMwR","usdm","caret","randomForest",
  "unbalanced","caTools","C50","dummies","e1071","MASS","rpart")
install.packages(x)
install.packages(x)
```

```
lapply(x, require,character.only=T)

# set working Directory
setwd("C:/Users/Bustec/Desktop/Edwisor project 1")

# Load dataset
dataset=read.csv("day.csv")

# see the dimension of data
dim(dataset)

#lapply(dataset, unique)

Missing_val=data.frame(apply(dataset, 2, function(x){sum(is.na(x))}))
Missing_val

# Installing important  Packages for visualization
install.packages("lattice")
library(lattice)
library(ggplot2)

#Plotting "cnt" with others variables
# "instant"
ggplot(data = dataset, aes(x=instant,y=cnt))+
  geom_bar(stat = 'identity', fill = "dark violet")

# dteday
ggplot(data = dataset, aes(x=dteday,y=cnt))+
  geom_bar(stat = 'identity', fill = "navy blue")
# season
ggplot(data = dataset, aes(x=season,y=cnt))+
  geom_bar(stat = 'identity', fill = "navy blue")

# "yr"
ggplot(data = dataset, aes(x=yr,y=cnt))+
  geom_bar(stat = 'identity', fill = "navy blue")


# "mnth"
ggplot(data = dataset, aes(x=mnth, y=cnt))+
  geom_bar(stat = 'identity', fill = "blue")


# "holiday"
ggplot(data = dataset, aes(x=holiday, y=cnt))+
  geom_bar(stat = 'identity', fill = "blue")


# "holiday"
ggplot(data = dataset, aes(x=holiday, y=cnt))+
  geom_bar(stat = 'identity', fill = "blue")
```

```
# "weekday"
ggplot(data = dataset, aes(x=weekday, y=cnt))+
  geom_bar(stat = 'identity', fill = "blue")

# "workingday"
ggplot(data = dataset, aes(x=workingday, y=cnt))+
  geom_bar(stat = 'identity', fill = "blue")

# "holiday"
ggplot(data = dataset, aes(x=holiday, y=cnt))+
  geom_bar(stat = 'identity', fill = "blue")

# "weathersit"
ggplot(data = dataset, aes(x=weathersit, y=cnt))+
  geom_bar(stat = 'identity', fill = "blue")

# "temp"
ggplot(data = dataset, aes(x=temp, y=cnt))+
  geom_point(aes_string(colour = dataset$cnt), size = 0.5, shape = dataset$temp)+
  theme_bw() + ggtitle(" scatter plot Analysis")

# "atemp"
ggplot(data = dataset, aes(x=atemp, y=cnt))+
  geom_point(aes_string(colour = dataset$cnt), size = 0.5, shape = dataset$temp)+
  theme_bw() + ggtitle(" scatter plot Analysis")

# "hum"
ggplot(data = dataset, aes(x=hum, y=cnt))+
  geom_point(aes_string(colour = dataset$cnt), size = 0.5, shape = dataset$temp)+
  theme_bw() + ggtitle(" scatter plot Analysis")

# "windspeed"
ggplot(data = dataset, aes(x=windspeed, y=cnt))+
  geom_point(aes_string(colour = dataset$cnt), size = 0.5, shape = dataset$temp)+
  theme_bw() + ggtitle(" scatter plot Analysis")

# "casual"
ggplot(data = dataset, aes(x=casual, y=cnt))+
  geom_point(aes_string(colour = dataset$cnt), size = 0.5, shape = dataset$temp)+
  theme_bw() + ggtitle(" scatter plot Analysis")

# "registered"
ggplot(data = dataset, aes(x=registered, y=cnt))+
  geom_point(aes_string(colour = dataset$cnt), size = 0.5, shape = dataset$temp)+
  theme_bw() + ggtitle(" scatter plot Analysis")

# seperate out numeric columns
numeric_columns = c("temp","atemp","hum","windspeed","casual","registered")

## Normality check before removing outliers
```

```
# Histogram using ggplot
# Install and load packages
#install.packages("gridExtra")
#library(gridExtra)

for(i in 1:length(numeric_columns))
{
  assign(paste0("gn",i),ggplot(dataset,aes_string(x=numeric_columns[i]))+
         geom_histogram(fill="sky blue",colour="black")+
         geom_density()+
         theme_bw()+
         labs(x=numeric_columns[i])+
         ggtitle(paste("Histogram of ",numeric_columns[i])))
}

gridExtra::grid.arrange(gn1,gn2,gn3,ncol=3)
gridExtra::grid.arrange(gn4,gn5,gn6,ncol=3)

# No any variables are  normally distributed.

### outlier Analysis
# Boxplot distribution and outlier check

for(i in 1:length(numeric_columns)){
  assign(paste0("gn",i),ggplot(dataset,aes_string(y=numeric_columns[i],x="cnt",
                                              fill=dataset$Absenteeism.time.in.hours))+
         geom_boxplot(outlier.colour = "red",fill="skyblue",outlier.shape = 18,
                      outlier.size = 1,notch = F)+
         theme_bw()+
         labs(y=numeric_columns[i],x="cnt")+
         ggtitle(paste("Box Plot of cnt for",numeric_columns[i])))
}
gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,gn6,ncol=3)

# our new numeric variable which contains outliers-
numeric_columns_outliers = c("hum","windspeed","casual")

for(i in numeric_columns_outliers){
  print(i)
  val=dataset[,i][dataset[,i] %in% boxplot.stats(dataset[,i])$out]
  dataset[,i][dataset[,i] %in% val]= NA
}
#library(DMwR)
dataset = knnImputation(dataset , k=5)

for(i in 1:length(numeric_columns)){
  assign(paste0("gn",i),ggplot(dataset,aes_string(y=numeric_columns[i],x="cnt",
                                              fill=dataset$Absenteeism.time.in.hours))+
         geom_boxplot(outlier.colour = "red",fill="skyblue",outlier.shape = 18,
                      outlier.size = 1,notch = F)+
         theme_bw()+
        labs(y=numeric_columns[i],x="cnt")+
        ggtitle(paste("Box Plot of cnt for",numeric_columns[i])))
}
gridExtra::grid.arrange(gn1,gn2,gn3,gn4,gn5,gn6,ncol=2)

# correlation Plot
Install library
install.packages("corrgram")
library(corrgram)
corrgram(dataset[,numeric_columns], order=F,
       lower.panel=panel.pie,text.panel=panel.txt,main="correlation plot")

# Here from correlaion plot "temp" and "atemp" are highly correlated,
#  so we need to remove one out of them

dataset_deleted = subset(dataset, select = -c(dteday,atemp))
                                    # "dteday" is irrelevant for prediction.

# check Multicollinearity
install.packages("usdm")
library(usdm)

# checking coliinearity with VIF
vifcor(dataset_deleted[,-14], th=0.95) #--> No variable from the 13 input variables has collinearity
problem.

# # Discard those variables whose p value >0.05
dataset_deleted = subset(dataset_deleted, select = -c(holiday,weekday,workingday))


# Run regression model again to check p value , R-squared and Adjusted R-squared

LR_model = lm(cnt~., data=dataset_deleted)

summary(LR_model) # R-squared:0.9827,           Adjusted R-squared:  0.9824
#
#
### Dimension Reduction - Discard  "holiday" p>0.75014
dataset_deleted = subset(dataset, select = -c(dteday,atemp,holiday))

# Run regression model again to check p value , R-squared and Adjusted R-squared
LR_model = lm(cnt~., data=dataset_deleted)

summary(LR_model) #R-squared: 0.9827,           Adjusted R-squared: 0.9824

## Dimension Reduction - Discard  "season" p>0.71185
dataset_deleted = subset(dataset_deleted, select = -
c(season,holiday,windspeed,hum,weathersit))
dataset_deleted = subset(dataset, select = -c(dteday,atemp,holiday,season))

# Run regression model again to check p value , R-squared and Adjusted R-squared
```

```
LR_model = lm(cnt~., data=dataset_deleted)

summary(LR_model) # R-squared:0.9827,          Adjusted R-squared: 0.9824

#Dimension Reduction - Discard  "windspeed" p> 0.63997
dataset_deleted = subset(dataset, select = -c(dteday,atemp,holiday,season,windspeed))

 # Run regression model again to check p value , R-squared and Adjusted R-squared
LR_model = lm(cnt~., data=dataset_deleted)

summary(LR_model) # R-squared:0.9827,          Adjusted R-squared: 0.9824

# ## Dimension Reduction - Discard  "yr" p>0.64768
dataset_deleted = subset(dataset, select = -c(dteday,atemp,holiday,season,windspeed,yr))

# # Run regression model again to check p value , R-squared and Adjusted R-squared

LR_model = lm(cnt~., data=dataset_deleted)
summary(LR_model)t.test(x, y)
#Multiple R-squared:  0.9827,    Adjusted R-squared:  0.9824

## Normality check before removing outliers
# Histogram using ggplot
# Instlal and load packages
install.packages("gridExtra")
library(gridExtra)

numeric_columns = c("temp","casual","registered")
for(i in 1:length(numeric_columns))
{
  assign(paste0("gn",i),ggplot(dataset_deleted,aes_string(x=numeric_columns[i]))+
         geom_histogram(fill="dark red",colour="black")+
         geom_density()+
         theme_bw()+
         labs(x=numeric_columns[i])+
         ggtitle(paste("Histogram of ",numeric_columns[i])))
}
gridExtra::grid.arrange(gn1,gn2,gn3,ncol=2)
# WE can see no any numeric variables are normally distributed

dataset_deleted[numeric_columns][1,] # Only "casual"  and  "registered" variables need to be
scaled
##  Normalization for Non uniformly distributed features
new_numeric_columns = c( "casual","registered" )

for(i in new_numeric_columns){
  print(i)
  dataset_deleted[,i]=(dataset_deleted[,i]-min(dataset_deleted[,i]))/
    (max(dataset_deleted[,i]-min(dataset_deleted[,i])))
}
```

```
#  # devide dataset into raining and test
library(caTools)
set.seed(123)
split = sample.split(dataset_deleted$cnt, SplitRatio = 0.8)
training_set = subset(dataset_deleted, split == TRUE)
test_set = subset(dataset_deleted, split == FALSE)

library(caret)
library(e1071)

  #principal component analysis
pca = preProcess(x= training_set[-14], method = "pca", pcaComp = 14)
training_set_pca = predict(pca, training_set)
test_set_pca = predict(pca, test_set)

#Plot of explained varience of principal components
#  #principal component analysis
prin_comp = prcomp(training_set_pca)

#compute standard deviation of each principal component
std_dev = prin_comp$sdev
#compute variance
pr_var = std_dev^2
 #proportion of variance explained
prop_varex = pr_var/sum(pr_var)

  #cumulative scree plot
  plot(cumsum(prop_varex), xlab = "Principal Component",
       ylab = "Cumulative Proportion of Variance Explained",
       type = "b")


  # choosing to 10 explained varience of Principal components
  pca = preProcess(x= training_set[-14], method = "pca", pcaComp = 10)
  training_set = predict(pca, training_set)
  training_set_pca = training_set_pca[c(2:21,1)]
  test_set= predict(pca, test_set)
  test_set_pca = test_set_pca[c(2:21,1)]

# devide dataset into raining and test
library(caTools)
set.seed(123)
split = sample.split(dataset_deleted$cnt, SplitRatio = 0.8)
training_set = subset(dataset_deleted, split == TRUE)
test_set = subset(dataset_deleted, split == FALSE)

#Run Multiple Linear Regression
LR_model = lm(cnt~., data=training_set)

print(LR_model)
```

```
# Summary of the model
summary(LR_model)


#Lets predict for training data
pred_LR_train = predict(LR_model, training_set[,names(training_set) != "cnt"])

#Lets predict for testing data
pred_LR_test = predict(LR_model, test_set[,names(test_set) != "cnt"])


###  Error Matrics
install.packages("caret")
library(caret)

#For training data
print(postResample(pred = pred_LR_train, obs = training_set$cnt))


# For testing data
print(postResample(pred = pred_LR_test, obs = test_set$cnt))

install.packages("DMwR")
library(DMwR)

regr.eval(test_set$cnt, pred_LR_test, stats = c('mape',"mse"))

# Load Library
install.packages("rpart")
install.packages("MASS")
library(rpart)
library(MASS)

## rpart for regression
DT_model= rpart(cnt ~ ., data = training_set, method = "anova")

summary(DT_model)

#write rules into disk
write(capture.output(summary(DT_model)), "Rules.txt")

#Lets predict for training data
pred_DT_train = predict(DT_model, training_set[,names(training_set) != "cnt"])

#Lets predict for training data
pred_DT_test = predict(DT_model,test_set[,names(test_set) != "cnt"])

# For training data
print(postResample(pred = pred_DT_train, obs = training_set$cnt))

# For testing data
```

```
print(postResample(pred = pred_DT_test, obs = test_set$cnt))

install.packages("DMwR")

regr.eval(test_set$cnt, pred_DT_test, stats = c('mape',"mse"))

# Fitting Random Forest Regression to the dataset
install.packages('randomForest')
library(randomForest)
set.seed(1234)
RF_model= randomForest(x = training_set[,names(training_set) != "cnt"],
                      y = training_set$cnt,
                      ntree = 500)


#Lets predict for training data
pred_RF_train = predict(RF_model, training_set[,names(training_set) != "cnt"])

#Lets predict for testing data
pred_RF_test = predict(RF_model, test_set[,names(test_set) != "cnt"])

# For training data
print(postResample(pred = pred_RF_train, obs = training_set$cnt))

# For testing data
print(postResample(pred = pred_RF_test, obs = test_set$cnt))

install.packages("DMwR")

regr.eval(test_set$cnt, pred_RF_test, stats = c('mape',"mse"))

# Fitting SVR to the dataset
install.packages('e1071')
library(e1071)
SVR_model = svm(formula = cnt ~ .,
                data = training_set,
                type = 'eps-regression',
                kernel = 'radial')


Lets predict for training data
pred_SVR_train = predict(SVR_model, training_set[,names(training_set) != "cnt"])

Lets predict for testing data
pred_SVR_test = predict(SVR_model, test_set[,names(test_set) != "cnt"])

### Error Matrics
# For training data
print(postResample(pred = pred_SVR_train, obs = training_set$cnt))

install.packages("DMwR")
```

```
regr.eval(test_set$cnt, pred_SVR_test, stats = c('mape',"mse"))

#Model selection

#                          RMSE train     RMSE test       difference
# Multiple Linear Regression: 265.3598716    212.2170503       -53.142
# Decision Tree            : 503.005411     598.3854972        95.38
# Random Forest            : 129.4719764    240.1007197        110.629
# SVR                      : 218.790003     221.1328272        2.342

# R-squqare
# Multiple Linear Regression: 0.9807779
# Decision Tree            : 0.9821495
# Random Forest            : 0.9865681
# SVR                      : 0.9880324

# SVR has less difference of RMSE test and RMSE train value as well as R^2 is also maximum.
# Therefore, SVR is working as a better pridictive model...


# writing csv file
write.csv(dataset_deleted,"dataset_output in R.csv", row.names = F)
```

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