BIKE RENTAL COUNT PREDICTION

DATA SCIENCE PROJECT

Abstract

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

Shivam Baslas

Shivam72baslas@gmail.com

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.

1.2 Data

We have 16 variables in given data set. There are 13 independent variable and 3 dependent variables (casual, Registered, cnt).

Table 1.1: Sample Data

```
> head(df)
  instant     dteday season yr mnth holiday weekday workingday weathersit     temp     atemp      hum
1       1 2011-01-01      1  0    1       0       6          0          2 0.344167 0.363625 0.805833
2       2 2011-01-02      1  0    1       0       0          0          2 0.363478 0.353739 0.696087
3       3 2011-01-03      1  0    1       0       1          1          1 0.196364 0.189405 0.437273
4       4 2011-01-04      1  0    1       0       2          1          1 0.200000 0.212122 0.590435
5       5 2011-01-05      1  0    1       0       3          1          1 0.226957 0.229270 0.436957
6       6 2011-01-06      1  0    1       0       4          1          1 0.204348 0.233209 0.518261
  windspeed casual registered  cnt
1 0.1604460    331        654  985
2 0.2485390    131        670  801
3 0.2483090    120       1229 1349
4 0.1602960    108       1454 1562
5 0.1869000     82       1518 1600
6 0.0895652     88       1518 1606
```

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. **holiday**: weather day is holiday or not (extracted fromHoliday Schedule)
7. **weekday:** Day of the week
8. **workingday:** If day is neither weekend nor holiday is 1, otherwise is 0.
9. **weathersit:** (extracted fromFreemeteo)
   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

10. **temp:** Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min),t_min=-8, t_max=+39 (only in hourly scale)
11. **atemp:** Normalized feeling temperature in Celsius. The values are derived via(t-t_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)
12. **hum:** Normalized humidity. The values are divided to 100 (max)
13. **windspeed:** Normalized wind speed. The values are divided to 67 (max)
14. **casual**: count of casual users
15. **registered**: count of registered users
16. **cnt:** count of total rental bikes including both casual and registered

1.3 Expletory Data Analysis

Exploratory Data Analysis (EDA) is an approach to analysing data sets and summarize their main characteristics.

1.3.1. Data Understanding

Our data consist 731 observation and 16 variables. We have data type factor for dteday and rest of the variables have data type integer or float.

Note:- dteday can't be categorical as it consist date of each day.

As per the data analysis we have to find which variables are the categorical variables, continuous variables and target variable. Data types need to be change accordingly if required.

We have distributed the variables on the basis of continuous and categorical variables. Target variable is continuous.

All the data type for the categorical variables need to be change to factor.

Table 1.2: Variables and its Data types.

```
> dim(df)
[1] 731  16
> str(df)
'data.frame':   731 obs. of  16 variables:
 $ instant   : int  1 2 3 4 5 6 7 8 9 10 ...
 $ dteday    : Factor w/ 731 levels "2011-01-01","2011-01-02",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ yr        : int  0 0 0 0 0 0 0 0 0 0 ...
 $ mnth      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ weekday   : int  6 0 1 2 3 4 5 6 0 1 ...
 $ workingday: int  0 0 1 1 1 1 1 0 0 1 ...
 $ weathersit: int  2 2 1 1 1 1 2 2 1 1 ...
 $ temp      : num  0.344 0.363 0.196 0.2 0.227 ...
 $ atemp     : num  0.364 0.354 0.189 0.212 0.229 ...
 $ hum       : num  0.806 0.696 0.437 0.59 0.437 ...
 $ windspeed : num  0.16 0.249 0.248 0.16 0.187 ...
 $ casual    : int  331 131 120 108 82 88 148 68 54 41 ...
 $ registered: int  654 670 1229 1454 1518 1518 1362 891 768 1280 ...
 $ cnt       : int  985 801 1349 1562 1600 1606 1510 959 822 1321 ...
```

1.3.2. Data Pre Observation

continuous variable

- instant
- dteday
- temp
- atemp
- hum
- windspeed
- casual
- registered:
- cnt

categorical variable

- season
- yr
- mnth
- holiday
- weekday
- workingday:
- weathersit:

target variable

- casual
- registered:
- cnt

- we have 4 independent continuous variables(temp, atemp, hum, windspeed) and 3 target continuous variables (Casual, registered, cnt)
- All independent variables having the values between 0 to 1 it means dataset contain normalized value for all 4 variables.
- Cnt is the sum of casual and registered variables so we can drop these variables for overall bike count prediction.

Table 1.3: summary of continuous Variables

```
      temp             atemp              hum          windspeed          casual
 Min.   :0.05913   Min.   :0.07907   Min.   :0.0000   Min.   :0.02239   Min.   :   2.0
 1st Qu.:0.33708   1st Qu.:0.33784   1st Qu.:0.5200   1st Qu.:0.13495   1st Qu.: 315.5
 Median :0.49833   Median :0.48673   Median :0.6267   Median :0.18097   Median : 713.0
 Mean   :0.49538   Mean   :0.47435   Mean   :0.6279   Mean   :0.19049   Mean   : 848.2
 3rd Qu.:0.65542   3rd Qu.:0.60860   3rd Qu.:0.7302   3rd Qu.:0.23321   3rd Qu.:1096.0
 Max.   :0.86167   Max.   :0.84090   Max.   :0.9725   Max.   :0.50746   Max.   :3410.0
   registered         cnt
 Min.   :  20   Min.   :  22
 1st Qu.:2497   1st Qu.:3152
 Median :3662   Median :4548
 Mean   :3656   Mean   :4504
 3rd Qu.:4776   3rd Qu.:5956
 Max.   :6946   Max.   :8714
```

- We have same count of observation in each unique value of season ,yr, month, weekdays. as these are the day or time related data and we have the 2 years data with everyday observation.
- Holiday have the 710 and 21 observation which is highly imbalance, it may be not help us to prediction or model development.
- Weathersit 3 having only 21 observation. It and low cnt of bikes it may cause imbalance in data.

Table 1.4: summary of categorical Variable

```
season  yr        mnth     holiday  weekday  workingday  weathersit
1:181   0:365   1  :62   0:710    0:105    0:231       1:463
2:184   1:366   2  :57   1: 21    1:105    1:500       2:247
3:188           3  :62            2:104               3: 21
4:178           4  :60            3:104
                5  :62            4:104
                6  :60            5:104
                7  :62            6:105
                8  :62
                9  :60
                10:62
                11:60
                12:62
```

Table 1.5: Bar Graph for categorical variable

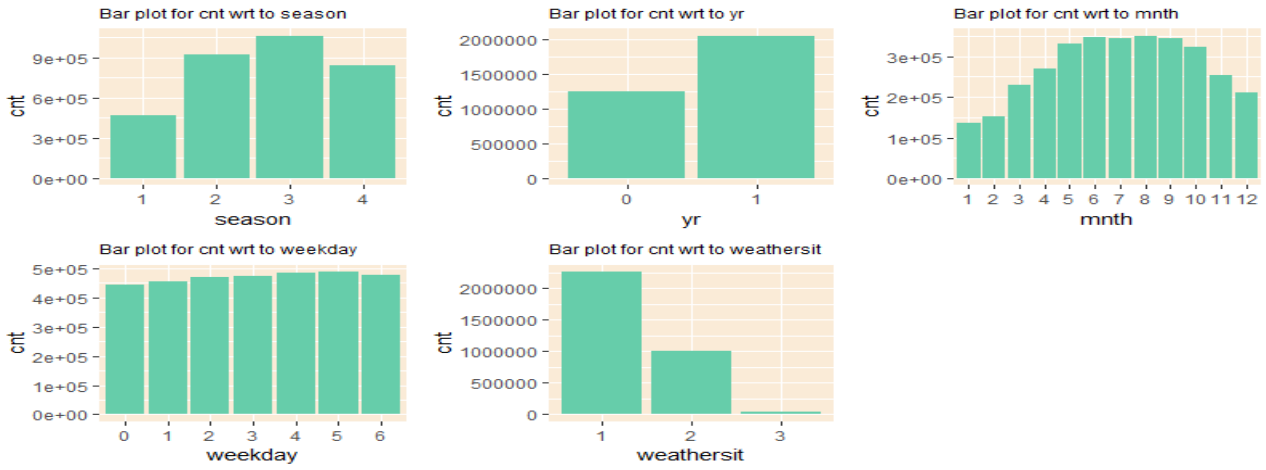
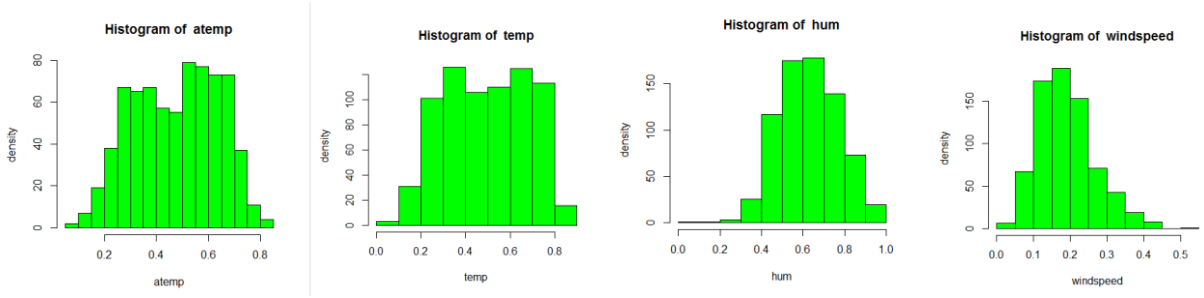


Table 1.7: Histogram for continuous variable



Chapter 2

Methodology

2.1 Data Preprocessing:

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

2.1.1. Missing Value Analysis

In statistics, missing data, or missing values, occur when no data value is stored for the variable in an observation. If a columns has more than 30% of data as missing value either we ignore the entire column or we ignore those observations. In the given data there is no any missing value. So we do not need to impute missing values.

Table 2.1: Missing values in dataset

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum
1	0	0	0	0	0	0	0	0	0	0	0	0

	windspeed	casual	registered	cnt
1	0	0	0	0

2.1.2. Outlier Analysis

Outliers are the abnormal values, which inconsistent with rest of dataset, or observation which lies abnormal distance from other values in dataset.

There are different-different causes of outlier like- Poor Data Quality, Manual Error, Malfunctioning Equipment, Low Quality Measurement and sometimes correct but exceptional data.

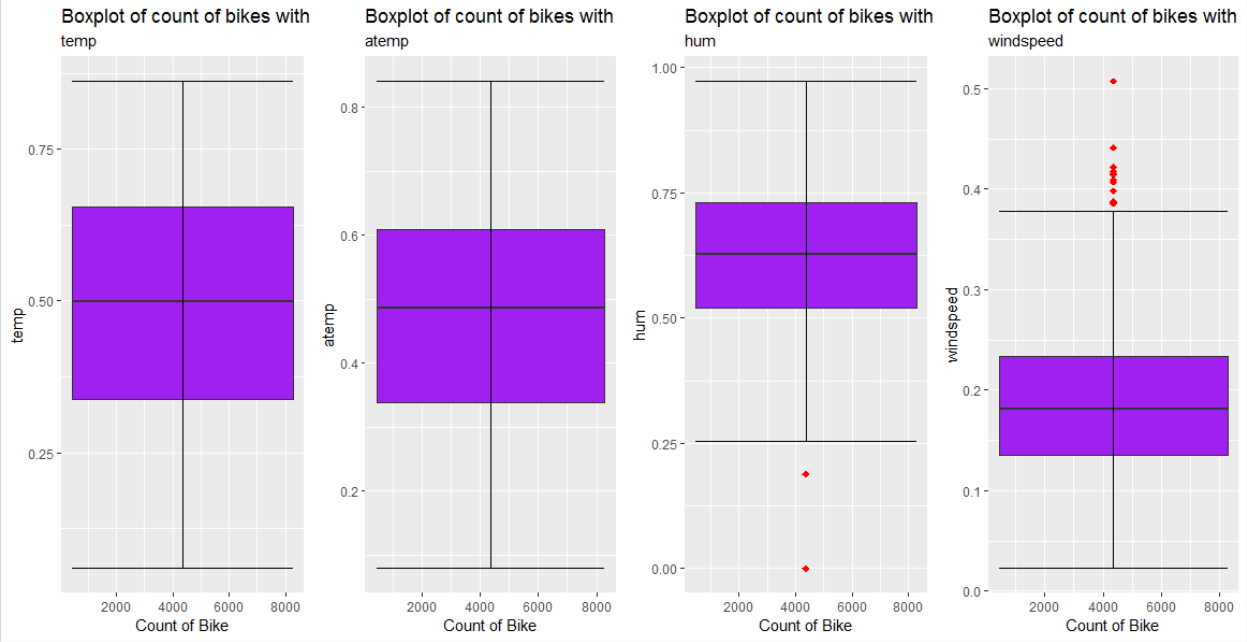
So, to detect the outlier and remove the outlier there are different-different method like Box Plot method, Grubb's test for outlier, R Package Outlier etc.

But from all of them prefer the Box Plot Method to detect and remove the outliers.

Box Plot method is a simplest way for detecting the outliers. A box plot is a graphical display for describing the distribution of data. It shows the data in its graphical representation with the median, 1st quartile, 3rd quartiles, inner fence and outer fence. If the values of a particular variable are outside from inner and outer fence then these values can be considered as outliers.

The box plot method detects outlier if any value is present greater than (**Q3 + (1.5 * IQR)**) or less than (**Q1 – (1.5 * IQR)**)
Q1 > 25% of data are less than or equal to this value
Q2 or Median -> 50% of data are less than or equal to this value
Q3 > 75% of data are less than or equal to this value
IQR(Inter Quartile Range) = Q3 – Q1

Table 2.2: Outlier analysis in dataset with Box Plot



Before treating outlier, we should look at nature of outlier. Is it information or outlier(error)?

Our dataset is based on season and environmental condition. Boxplot finds outliers by calculating distance on a single column only. Suppose for clear weather condition, windspeed is normal maximum time. Now some day windspeed is higher than others day and it has high value. Now, Boxplot may consider it as outlier, as distance from median would be high for this value and due to high windspeed there may be decrease in bike rental counting for that day. So, boxplot method would remove that data point, but that data point could be an important predictor.

So we have impute the outlier data point by the median in 'hum' and 'windspeed'.

Table 2.2: dataset summary after Outlier imputation.

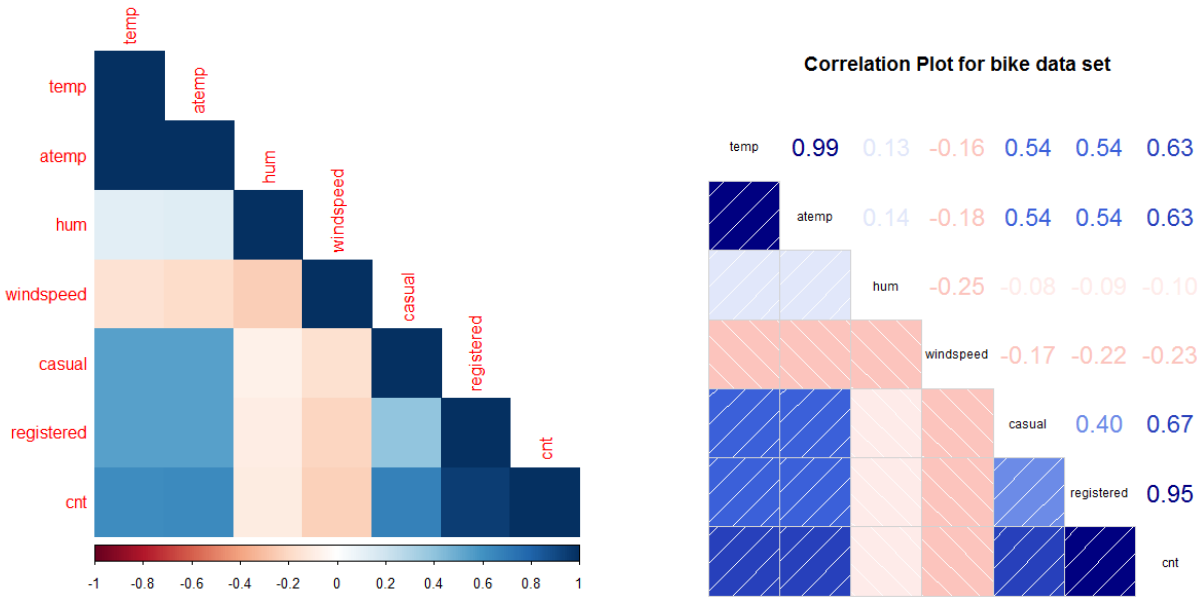
temp	atemp	hum	windspeed
Min. :0.05913	Min. :0.07907	Min. :0.2542	Min. :0.02239
1st Qu.:0.33708	1st Qu.:0.33784	1st Qu.:0.5223	1st Qu.:0.13495
Median :0.49833	Median :0.48673	Median :0.6275	Median :0.17880
Mean :0.49538	Mean :0.47435	Mean :0.6294	Mean :0.18626
3rd Qu.:0.65542	3rd Qu.:0.60860	3rd Qu.:0.7302	3rd Qu.:0.22979
Max. :0.86167	Max. :0.84090	Max. :0.9725	Max. :0.37811

casual	registered	cnt
Min. : 2	Min. : 20	Min. : 22
1st Qu.: 295	1st Qu.:2497	1st Qu.:3152
Median : 674	Median :3662	Median :4548
Mean : 732	Mean :3656	Mean :4504
3rd Qu.:1026	3rd Qu.:4776	3rd Qu.:5956
Max. :2258	Max. :6946	Max. :8714

2.1.3. Feature Selection

Selecting subset of relevant columns for the model construction is known as Feature Selection. We cannot use all the features because some features may be carrying the same information or irrelevant information which can increase overhead. To reduce overhead we adopt feature selection technique to extract meaningful features out of data. This in turn helps us to avoid the problem of multi collinearity.
In this project we have selected Correlation Analysis for numerical variable and ANOVA (Analysis of variance) for categorical variables.

Table 2.3: Correlation analysis for continuous variable with Heatmap and Correlation Plot



From the above analysis temp and atemp is highly correlated (>.9). so we drop the atemp variable as both carry the same information. Registered and cnt also high correlated and Target variables. We take total count (cnt) and drop the casual and registered variable for model development.

Table 2.4: VIF Test for Continuous Variables

```
> vifcor(numeric_data,th=0.7)
No variable from the 3 input variables has collinearity problem.

The linear correlation coefficients ranges between:
min correlation ( hum ~ temp ):  0.1237232
max correlation ( windspeed ~ hum ):  -0.2002375

---------- VIFs of the remained variables --------
  Variables      VIF
1      temp 1.029744
2       hum 1.052048
3 windspeed 1.056335
```

Table 2.5: ANOVA Test for Categorical Variables

```
[1] "season"
             Df    Sum Sq   Mean Sq F value Pr(>F)
df[, i]       3 9.506e+08 316865289   128.8 <2e-16 ***
Residuals   727 1.789e+09   2460715
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "yr"
             Df    Sum Sq   Mean Sq F value Pr(>F)
df[, 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] "mnth"
             Df    Sum Sq  Mean Sq F value Pr(>F)
df[, i]      11 1.070e+09 97290206    41.9 <2e-16 ***
Residuals   719 1.669e+09  2321757
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "holiday"
             Df    Sum Sq  Mean Sq F value Pr(>F)
df[, 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)
df[, i]       6 1.766e+07 2943170   0.783  0.583
Residuals   724 2.722e+09 3759498
[1] "workingday"
             Df    Sum Sq  Mean Sq F value Pr(>F)
df[, 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] "weathersit"
             Df    Sum Sq   Mean Sq F value Pr(>F)
df[, i]       2 2.716e+08 135822286   40.07 <2e-16 ***
Residuals   728 2.468e+09   3389960
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

As per the above result P value is greater then .05 for holiday, weekday and working day variables. So we can drop these variables as our target variable cnt is not much dependent on these variables.

2.1.4. Feature Scaling:

We have numerical columns temp, atemp, hum and windspeed, which are provided in normalized form. As all the values b/w 0 to 1.
All variable are in same range. So we don't require feature scaling for our dataset.

2.1.5. Data after EDA and preprocessing

We remove 'instant', 'dteday', 'atemp', 'casual', 'registered' , 'holiday', 'weekday', 'workingday:' . rest of the variables for further data analysis are-

continuous variable	categorical variable	target variable
• temp • hum • windspeed • cnt	• season • yr • mnth • weathersit:	• cnt

Table 2.6: Data after EDA and Preprocessing

	season	yr	mnth	weathersit	temp	hum	windspeed	cnt
1	1	0	1	2	0.344167	0.805833	0.1604460	985
2	1	0	1	2	0.363478	0.696087	0.2485390	801
3	1	0	1	1	0.196364	0.437273	0.2483090	1349
4	1	0	1	1	0.200000	0.590435	0.1602960	1562
5	1	0	1	1	0.226957	0.436957	0.1869000	1600
6	1	0	1	1	0.204348	0.518261	0.0895652	1606

2.2. Model Development

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

2.2.1. Model Building

2.2.2.1. Decision Tree

Decision Tree is a supervised machine learning algorithm, which is used to predict the data for classification and regression. It accepts both continuous and categorical variables. A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Each branch connects nodes with “and” and multiple branches are connected by “or”. Extremely easy to understand by the business users. It provides its output in the form of rule, which can easily understood by a non-technical person also.

2.2.2.2. Random Forest

Random Forest is an ensemble technique that consists of many decision trees. The idea behind Random Forest is to build n number of trees to have more accuracy in dataset. It is called random forest as we are building n no. of trees randomly. In other words, to build the decision trees it selects randomly n no of variables and n no of observations. It means to build each decision tree on random forest we are not going to use the same data. The higher no of trees in the random forest will give higher no of accuracy, so in random forest we can go for multiple trees. It can handle large no of independent variables without variable deletion and it will give the estimates that what variables are important.

2.2.2.3. Liner Regression

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

2.2.2.4. 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.

2.2.2.5. Results

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

Table 2.6: R result after model development

	Model	MAPE_Train	MAPE_Test	R.Squared_Train	R.Squared_Test
1	Decision Tree for Regression	56.30015	23.70970	0.7939743	0.7521948
2	Random Forest	27.17905	17.22288	0.9660954	0.8685453
3	Linear Regression	47.40023	16.87103	0.8365491	0.8334009
4	Gradient Boosting	35.05797	16.38084	0.9015068	0.8593749

Table 2.7: PYTHON result after model development

	MAPE_Test	MAPE_Train	Model Name	R-squared_Test	R-squared_Train
0	36.948093	62.260133	Decision Tree	0.646470	0.677563
0	20.390551	16.248647	Random Forest	0.886672	0.980072
0	18.800696	44.444512	Linear Regression	0.841110	0.832760
0	11.744815	44.444512	Gradient Boosting	0.865779	0.945385

Now, we will tune our two best models i.e. Random Forest and Gradient Boosting with the help of hyperparameter tuning we would find optimum values for parameter used in function and would increase our accuracy.

2.2.2. Hyperparameter Tuning

In statistics, hyperparameter is a parameter from a prior distribution; it captures the prior belief before data is observed. In any machine learning algorithm, these

parameters need to be initialized before training a model. Choosing appropriate hyperparameters plays a crucial role in the success of good model.
Since it makes a huge impact on the learned model. For example, if the learning rate is too low, the model will miss the important patterns in the data. If it is high, it may have collisions. we used two techniques of Hyperparameter in our model-

- Random Search
- Grid Search

2.2.2.1. Random Search Hyperparameter Tuning

Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model.
In this search pattern, random combinations of parameters are considered in every iteration. The chances of finding the optimal parameter are comparatively higher in random search because of the random search pattern where the model might end up being trained on the optimised parameters without any aliasing.

2.2.2.2. Grid Search Hyperparameter Tuning

Grid search is a technique which tends to find the right set of hyperparameters for the particular model. Hyperparameters are not the model parameters and it is not possible to find the best set from the training data. Model parameters are learned during training when we
optimise a loss function using something like a gradient descent. In this tuning technique, we simply build a model for every combination of various hyperparameters and evaluate each model. The model which gives the highest accuracy wins. The pattern followed here is similar to the grid, where all the values are placed in the form of a matrix. Each set of parameters is taken into consideration and the accuracy is noted. Once all the combinations are evaluated, the model with the set of parameters which give the top accuracy is considered to be the best.

Chapter 3

Conclusion

3.1. Model Evaluation

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

Table 2.6: R Final result after Hyperparameter Tuning

	Model	MAPE_Train	MAPE_Test	R.Squared_Train	R.Squared_Test
1	Decision Tree for Regression	56.30015	23.70970	0.7939743	0.7521948
2	Random Forest	27.17905	17.22288	0.9660954	0.8685453
3	Linear Regression	47.40023	16.87103	0.8365491	0.8334009
4	Gradient Boosting	35.05797	16.38084	0.9015068	0.8593749
5	Random Search CV in Random Forest	25.41512	17.34445	0.9646437	0.8674905
6	Grid Search CV in Random Forest	24.51039	17.49735	0.9700832	0.8672778
7	Random Search CV in Gradient Boosting	26.13775	17.52599	0.9678736	0.8665142
8	Grid Search CV in Gradient Boosting	26.28839	17.67315	0.9672954	0.8665138

Table 2.7: Python Final result after Hyperparameter Tuning

	MAPE_Test	MAPE_Train	Model Name	R-squared_Test	R-squared_Train
0	36.948093	62.260133	Decision Tree	0.646470	0.677563
0	20.390551	16.248647	Random Forest	0.886672	0.980072
0	18.800696	44.444512	Linear Regression	0.841110	0.832760
0	11.744815	44.444512	Gradient Boosting	0.865779	0.945385
0	20.934713	20.522484	Random Search CV in Random Forest	0.880680	0.978894
0	20.556715	21.314452	Grid Search CV in Random Forest	0.875579	0.964836
0	23.511275	47.439860	Random Search CV in Gradient Boosting	0.817430	0.834542
0	25.485646	18.833448	Grid Search CV in Gradient Booster	0.833746	0.922114

3.2. Model Selection

From the observation of all MAPE and R-Squared Value we have concluded that Grid search Random Forest has minimum value of MAPE (17.5%) and it's R-Squared Value is also maximum (0.86).Means, By Random forest algorithm predictor are able to explain 86% to the target variable on the test data. The MAPE value of Test data and Train does not differs a lot this implies that it is not the case of overfitting.

Chapter 4

Codes

4.1 R Code:

```
### clear environment ###
rm(list=ls())

### load the libraries  ###

library(ggplot2)
library(corrgram)
library(corrplot)


###set working directory###

setwd("D:/DATA SCIENCE STUDY METERIAL/Projects/Bike renting")
getwd()


### Load Bike renting Data CSV file ###

df=read.csv("day.csv",header = T)

head(df)
#############################################################################
####################   1.3 Expletory Data Analysis   ########################
#############################################################################


#####################  1.3.1. Data Understanding  ##########################

dim(df)      # checking the dimension of data frame.

str(df)      # checking datatypes of all columns.

####################  1.3.2. Data Pre Observation  ##########################

#store categorical and continuous Variable column names

cat_var=c("season","yr","mnth","holiday","weekday","workingday","weathersit")

numeric_Var=c('temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt')
```

```
#convert the data type of categorical variables to factor from integer

for(i in c(3:9)){

  df[,i]=as.factor(df[,i])
}

str(df)                                  #checking datatypes of all columns

summary(df[,numeric_var])                # checking numerical variables
summary(df[,cat_var],maxsum=13)          # checking categorical variables

# bar graph for categorical variables

plot_bar <- function(cat, y, fun){
  gp = aggregate(x = df[, y], by=list(cat=df[, cat]), FUN=fun)
  ggplot(gp, aes_string(x = 'cat', y = 'x'))+
    geom_bar(stat = 'identity',fill = "aquamarine3")+
    labs(y = y, x = cat)+theme(panel.background = element_rect("antiquewhite"))+
    theme(plot.title = element_text(size = 9))+
    ggtitle(paste("Bar plot for",y,"wrt to",cat))
}

PB1=plot_bar('season', 'cnt', 'sum')
PB2=plot_bar('yr', 'cnt', 'sum')
PB3=plot_bar('mnth', 'cnt', 'sum')
PB4=plot_bar('weekday','cnt','sum')
PB5=plot_bar('weathersit', 'cnt', 'sum')
gridExtra::grid.arrange(PB1,PB2,PB3,PB4,PB5,ncol=3)

# Chacking VIF for skewness

cnames=c('temp', 'atemp', 'hum', 'windspeed')
library(propagate)
for(i in cnames){
  print(i)
  skew= skewness(df[,i])
  print(skew)
}                                        #windspeed variable is right skewed in dataset.

# histogram for continuous variables

hist_plot <- function(column, dataset){
  hist(x=dataset[,column],col="Green",xlab=column,ylab="density",
       main=paste("Histogram of ", column))
}

PH1=hist_plot('temp',dataset=df)
PH2=hist_plot('atemp',dataset=df)
PH3=hist_plot('hum',dataset=df)
PH4=hist_plot('windspeed',dataset=df)


#############################################################################
###################   2.1. Data Preprocessing   #############################
#############################################################################


###################  2.1.1. Missing Value Analysis  ########################

miss_val=data.frame(lapply(df,function(x) sum(is.na(x))))
miss_val


###################  2.1.2. outlier analysis ##############################

numeric_index=sapply(df,is.numeric)
numeric_data=df[,numeric_index]
cnames=colnames(numeric_data)       #selecting numerical variables

#create Box plot for outlier analysis

for(i in 1:length(cnames)){
  assign(paste0("AB",i),ggplot(aes_string(x="cnt",y=(cnames[i])),data=subset(df))+
           geom_boxplot(outlier.color = "Red",outlier.shape = 18,outlier.size = 2,
```

```
for(i in 1:length(cnames)){
  assign(paste0("AB",i),ggplot(aes_string(x="cnt",y=(cnames[i])),data=subset(df))+
           geom_boxplot(outlier.color = "Red",outlier.shape = 18,outlier.size = 2,
                        fill="Purple")+theme_get()+
           stat_boxplot(geom = "errorbar",width=0.5)+
           labs(x="Count of Bike",y=cnames[i])+
           ggtitle("Boxplot of count of bikes with",cnames[i]))
}

gridExtra::grid.arrange(AB2,AB3,AB4,AB5,ncol=4)   # plot all graph

#Replace outliers with NA

for(i in cnames){
  print(i)
  outlier= df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  print(length(outlier))
  df[,i][df[,i] %in% outlier]=NA
}

sum(is.na(df))

#Impute outliers by median method

df$hum[is.na(df$hum)]=median(df$hum,na.rm=TRUE)
df$windspeed[is.na(df$windspeed)]=median(df$windspeed,na.rm=TRUE)
df$casual[is.na(df$casual)]=median(df$casual,na.rm=TRUE)

summary(df[,numeric_Var])                          #summary of dataset for numeric variables


#########################  2.1.3.Feature Selection  ##############################


####  for continuous variables ####

# correlation plot for numerical feature
####  for continuous variables ####

# correlation plot for numerical feature

corrgram(df[,numeric_Var], order = FALSE,
         upper.panel = panel.cor, text.panel = panel.txt,
         main = "Correlation Plot for bike data set")

# heatmap plot for numerical features

corrplot(cor(df[,numeric_Var]), method = 'color', type = 'lower',title = "Heatmap plot for bike data set")


####  for categorical Variable  ####

#Anova analysis for categorical variable with target numeric variable-
for(i in cat_var){
  print(i)
  Anova_result= summary(aov(formula = cnt~df[,i],df))
  print(Anova_result)
}


#####################  2.1.5.  Data after EDA and preprocessing  ################

df= subset(df,select=-c(instant,dteday,atemp,casual,registered,holiday,weekday,workingday))


head(df)

write.csv(df,"bike data for model development.csv", row.names=FALSE )

# change categorical to numeric making bin for regression model

cat_index=sapply(df,is.factor)
cat_data=df[,cat_index]
cat_var=colnames(cat_data)
```

```
library(dummies)
df= dummy.data.frame(df,cat_var)

#############################################################################
####################   2.2. Model Development   ############################
#############################################################################


########################  2.2.1 Model building  ########################

#clear all the data except final data set.

data=df
df=data

library(DataCombine)
rmExcept("data")


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

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


# devide the data in train and test

set.seed(123)
train_index= sample(1:nrow(data),0.8*nrow(data))
train= data[train_index,]
test= data[-train_index,]
### 2.2.1.desision tree for regression  ###

library(rpart)

fit=rpart(cnt~.,data=train,method = "anova") #model development on train data

DT_test=predict(fit,test[,-25])            #predict test data
DT_train= predict(fit,train[,-25])         #predict train data

DT_MAPE_Test = mape(test[,25],DT_test)     # MAPE calculation for test data
DT_MAPE_Train = mape(train[,25],DT_train)  # MAPE calculation for train data

DT_r2_test=rsquare(test[,25],DT_test)      # r2 calculation for test data
DT_r2_train= rsquare(train[,25],DT_train)  # r2 calculation for train data


### 2.2.2. Random forest for regression ###

library(randomForest)

RF_model= randomForest(cnt~.,train,ntree=100,method="anova") #Model development on train data

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

RF_MAPE_Test=mape(test[,25],RF_test)       #MAPE calculation of test data-
RF_MAPE_Train=mape(train[,25],RF_train)    #MAPE calculation of train data

RF_r2_test=rsquare(test[,25],RF_test)      #r2 calculation for test data-
RF_r2_train=rsquare(train[,25],RF_train)   #r2 calculation for train data-
```

```
###  2.2.3. Linear Regression ###


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

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

LR_MAPE_Test=mape(test[,25],LR_test)      #MAPE calculation of test data
LR_MAPE_Train=mape(train[,25],LR_train)   #MAPE calculation of train data

LR_r2_test=rsquare(test[,25],LR_test)     #r2 calculation for test data
LR_r2_train=rsquare(train[,25],LR_train)  #r2 calculation for train data


### 2.2.4. Gradient Boosting ###

library(gbm)

GB_model = gbm(cnt~., data = train, n.trees = 100, interaction.depth = 2) #Model devlopment on train data

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

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

GB_r2_test=rsquare(test[,25],GB_test)                      #r2 calculation for test data-
GB_r2_train=rsquare(train[,25],GB_train)                   #r2 calculation for train data-



Result= data.frame('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),
Result= data.frame('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),
                   'R-Squared_Train'=c(DT_r2_train,RF_r2_train,LR_r2_train,GB_r2_train),
                   'R-Squared_Test'=c(DT_r2_test,RF_r2_test,LR_r2_test,GB_r2_test))

Result           #Random forest and Gradient Bosting have best fit model for the data.


########################  2.2.2. Hyperparameter Tuning  ########################

#Random Search CV in Random Forest

library(caret)

control = trainControl(method="repeatedcv", number=10, repeats=3,search='random')


RRF_model = caret::train(cnt~., data=train, method="rf",trControl=control,tuneLength=10)   #model devlopment on train data
best_parameter = RRF_model$bestTune                              #Best fit parameters
print(best_parameter)
#mtry=7 As per the result of best_parameter

RRF_model = randomForest(cnt ~ .,train, method = "rf", mtry=7,importance=TRUE)        #build model based on best fit

RRF_test= predict(RRF_model,test[-25])                        #Prediction on test data
RRF_train= predict(RRF_model,train[-25])                      #Prediction on train data


RRF_MAPE_Test = mape(test[,25],RRF_test)                      #Mape calculation of test data
RRF_MAPE_Train = mape(train[,25],RRF_train)                   #Mape calculation of train data


RRF_r2_test=rsquare(test[,25],RRF_test)                      #r2 calculation for test data
RRF_r2_train= rsquare(train[,25],RRF_train)                  #r2 calculation for train data
```

```
# Grid Search CV in Random Forest

control = trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
tunegrid = expand.grid(.mtry=c(6:18))


GRF_model= caret::train(cnt~.,train, method="rf", tuneGrid=tunegrid, trControl=control)     #model devlopment on train data
best_parameter = GRF_model$bestTune                          #Best fit parameters
print(best_parameter)
#mtry=9 As per the result of best_parameter

GRF_model = randomForest(cnt ~ .,train, method = "anova", mtry=9)                  #build model based on best fit

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


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


GRF_r2_test=rsquare(test[,25],GRF_test)                   #r2 calculation for test data
GRF_r2_train= rsquare(train[,25],GRF_train)               #r2 calculation for train data



##############Random Search CV in Gradient Boosting###########################################

control = trainControl(method="repeatedcv", number=5, repeats=1)

RGB_model = caret::train(cnt~., data=train, method="gbm",trControl=control,tuneLength=10)     #model devlopment on train data
best_parameter = RGB_model$bestTune                       #Best fit parameters
print(best_parameter)
# n.trees=100,interaction.depth=5,shrinkage=0.1,n.minobsinnode=10

RGB_model = randomForest(cnt ~ .,train, method = "anova", n.trees=100,
                         interaction.depth=5,shrinkage=0.1,n.minobsinnode=10)            #build model based on best fit


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


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


RGB_r2_test=rsquare(test[,25],RGB_test)                   #r2 calculation for test data
RGB_r2_train= rsquare(train[,25],RGB_train)               #r2 calculation for train data



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

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

-
GGB_model= caret::train(cnt~.,train, method="gbm", tuneGrid=tunegrid, trControl=control)      #model devlopment on train data
best_parameter = GGB_model$bestTune                       #Best fit parameters
print(best_parameter)
# n.trees = 2567,interaction.depth = 3,shrinkage = 0.01,n.minobsinnode = 18 as per best fit output

GGB_model = randomForest(cnt ~ .,train, method = "anova", n.trees = 2567,
                         interaction.depth = 3,shrinkage = 0.01,n.minobsinnode = 18)          #build model based on best fit

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


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

```
GGB_r2_test=rsquare(test[,25],GGB_test)                    #r2 calculation for test data
GGB_r2_train= rsquare(train[,25],GGB_train)                #r2 calculation for train data


final_result= data.frame('Model'=c('Decision Tree for Regression','Random Forest',
                                   'Linear Regression','Gradient Boosting',
                                   'Random Search CV in Random Forest','Grid Search CV in Random Forest',
                                   'Random Search CV in Gradient Boosting','Grid Search CV in Gradient Boosting'),
                         'MAPE_Train'=c(DT_MAPE_Train,RF_MAPE_Train,LR_MAPE_Train,GB_MAPE_Train,
                                        RRF_MAPE_Train,GRF_MAPE_Train,RGB_MAPE_Train,GGB_MAPE_Train),
                         'MAPE_Test'=c(DT_MAPE_Test,RF_MAPE_Test,LR_MAPE_Test,GB_MAPE_Test,
                                       RRF_MAPE_Test,GRF_MAPE_Test,RGB_MAPE_Test,GGB_MAPE_Test),
                         'R-Squared_Train'=c(DT_r2_train,RF_r2_train,LR_r2_train,GB_r2_train,
                                             RRF_r2_train,GRF_r2_train,RGB_r2_train,GGB_r2_train),
                         'R-Squared_Test'=c(DT_r2_test,RF_r2_test,LR_r2_test,GB_r2_test,
                                            RRF_r2_test,GRF_r2_test,RGB_r2_test,GGB_r2_test))

print(final_result)
```

4.2 Python Code (Jupyter Notebook) :

```
#Load libraries-
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from ggplot import *
from sklearn.metrics import r2_score
from scipy import stats
```

```
###set working directory###
os.chdir("D:\DATA SCIENCE STUDY METERIAL\Projects\Bike renting")
os.getcwd()
```

```
### Load Bike renting Data CSV file ###
df= pd.read_csv("day.csv")
df.head()
```

1.3 Expletory Data Analysis

1.3.1. Data Understanding

```
print(type(df))
print(df.shape)
print(df.dtypes)
```

```
print(df.columns)
print(df.shape)
```

1.3.2. Data Pre Observation

```
#store categorical and continuous Variable column names
cat_var=["season","yr","mnth","holiday","weekday","workingday","weathersit"]
numeric_var=['temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt']
```

```
#convert the data type of categorical variables to factor from integer
for i in cat_var:
    df.loc[:,i] = df.loc[:i].astype(object)

print(df.dtypes)
```

```
for i in numeric_var:
    print(df.loc[:,i].describe())                          #Checking Numerical variables summary

for i in cat_var:
    print(df.loc[:,i].describe())                          #Checking Categorical variable summary
```

```
#Bar graph for Categorical variables
def plot_bar(x, y, method = 'sum'):
    fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize= (12,4), squeeze=False)
    super_title = fig.suptitle("Bar Plot ",  fontsize='x-large')
    gp = df.groupby(by = x).sum()
    gp = gp.reset_index()
    sns.barplot(x= x, y = y, data = gp, ax=ax[0][0])
    fig.subplots_adjust(top = 0.90)
    plt.show()
```

```
plot_bar('season','cnt')
plot_bar('yr','cnt')
plot_bar('mnth','cnt')
plot_bar('weekday','cnt')
plot_bar('weathersit','cnt')
```

```
#distribution check and Histogram for continuous variables
for i in numeric_var:
    print(i)
    sns.distplot(df[i],bins='auto',color='green')
    plt.title("Distribution for Variable "+i)
    plt.ylabel("Density")
    plt.show()
```

2.1 Data Preprocessing

2.1.1. Missing Value Analysis

```
###################  2.1.1.Missing Value Analysis  ########################
df.isnull().sum()
```

2.1.2. Outlier Analysis

```
cnames=['temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt']

#create Box plot for outlier analysis
for i in cnames:
    print(i)
    sns.boxplot(y=df[i])
    plt.xlabel(i)
    plt.ylabel("values")
    plt.title("Boxplot of "+i)
    plt.show()
```

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

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

```
#impute NA with median-
df['hum']=df['hum'].fillna(df['hum'].median())
df['windspeed']=df['windspeed'].fillna(df['windspeed'].median())
df['casual']=df['casual'].fillna(df['casual'].median())
```

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

2.1.3. Feature selection

```
#correlation analysis for numeric variables-

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

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

```
#correlation plot-
f,ax= plt.subplots(figsize=(8,8))

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

```
#Anova analysis for categorical variable with target numeric variable

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

```
#Anova analysis for categorical variable with target numeric variable

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

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

2.1.5. Data after EDA and preprocessing

```
#Dimension Reduction-
df= df.drop(['instant','dteday','atemp','casual','registered','holiday','weekday','workingday'],axis=1)

print(df.shape)
```

```
df.head()
df.to_csv("bike data for model development.csv", index = False)
```

```
#store categorical and continuous Variable column names
cat_var=["season","yr","mnth","weathersit"]
numeric_var=['temp', 'hum', 'windspeed', 'cnt']
```

```
df=data
data=df
```

```
# change categorical to numeric making bin for regression model
df= pd.get_dummies(df,columns=cat_var)

df.shape
```

```
df.head()
```

2.2 Model Development

2.2.1 Model Building

```
#Import libraries-
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

```
#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
```

```
#split data for predictor and target seperatly-
X= df.drop(['cnt'],axis=1)
y= df['cnt']
```

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

2.2.1.desision tree for regression

```
from sklearn.tree import DecisionTreeRegressor                          #import libraries

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

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


MAPE_test= MAPE(y_test,DT_test)                                        #Model performance on test data
MAPE_train= MAPE(y_train,DT_train)                                     #Model performance on train data


r2_test=r2_score(y_test,DT_test)                                      #r2 value for test data
r2_train= r2_score(y_train,DT_train)                                    #r2 value for train data


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))
```

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

2.2.2. Random forest for regression ¶

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

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


RF_test= RF_model.predict(X_test)                                           #model prediction on test data
RF_train= RF_model.predict(X_train)                                         #model prediction on train data


MAPE_test= MAPE(y_test,RF_test)                                             #Model performance on test data
MAPE_train= MAPE(y_train,RF_train)                                          #Model performance on train data


r2_test=r2_score(y_test,RF_test)                                            #r2 value for test data
r2_train= r2_score(y_train,RF_train)                                        #r2 value for train data

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))
```

```
#result
df2= {'Model Name': ['Random Forest '],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],
      'R-squared_Test':[r2_test]}
result2= pd.DataFrame(df2)
```

```
#appand the results
result= result1.append(result2)
result
```

2.2.3. Linear Regression

```
import statsmodels.api as sm                                        #import libraries

LR_model= sm.OLS(y_train,X_train).fit()                             #Linear Regression model for regression
LR_test= LR_model.predict(X_test)                                     #model prediction on test data
LR_train= LR_model.predict(X_train)                                   #model prediction on train data

MAPE_test= MAPE(y_test,LR_test)                                       #Model performance on test data
MAPE_train= MAPE(y_train,LR_train)                                    #Model performance on train data

r2_test=r2_score(y_test,LR_test)                                      #r2 value for test data
r2_train= r2_score(y_train,LR_train)                                  #r2 value for train data

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))
```

```
df3= {'Model Name': ['Linear Regression '],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],
      'R-squared_Test':[r2_test]}
result3= pd.DataFrame(df3)
```

```
result= result.append(result3)
result
```

2.2.4. Gradient Boosting

```
from sklearn.ensemble import GradientBoostingRegressor                #import libraries

GB_model = GradientBoostingRegressor().fit(X_train, y_train)          #Gradient Boosting for regression

GB_test= GB_model.predict(X_test)                                     #model prediction on test data
GB_train= GB_model.predict(X_train)                                   #model prediction on train data

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

r2_test=r2_score(y_test,GB_test)                                      #r2 value for test data
r2_train= r2_score(y_train,GB_train)                                  #r2 value for train data

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))
```

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

```
result= result.append(result4)
result
```

2.2.2. Hyperparameter Tuning

Random Search CV in Random Forest

```
from sklearn.model_selection import RandomizedSearchCV                              #import libraries

RandomRandomForest = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,100,2))
depth = list(range(1,20,2))
random_search = {'n_estimators':n_estimator, 'max_depth': depth}

#Random Grid Random Forest model-
RRF_model= RandomizedSearchCV(RandomRandomForest,param_distributions= random_search,n_iter=3,cv=10)
RRF_model= RRF_model.fit(X_train,y_train)

best_parameters = RRF_model.best_params_                                            #Best parameters for model

best_model = RRF_model.best_estimator_                                              #Best model

RRF_test = best_model.predict(X_test)                                               #Model prediction on test data
RRF_train = best_model.predict(X_train)                                             #Model prediction on train data

MAPE_test= MAPE(y_test,RRF_test)                                                    #Model performance on test data
MAPE_train= MAPE(y_train,RRF_train)                                                 #Model performance on train data

r2_test=r2_score(y_test,RRF_test)                                                   #r2 value for test data
r2_train= r2_score(y_train,RRF_train)                                               #r2 value for train data

print("Best Parameter="+str(best_parameters))
print("Best Model="+str(best_model))
print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage Error for test data="+str(MAPE_test))
print("R^2_score for train data="+str(r2_train))
print("R^2_score for test data="+str(r2_test))
```

```
df5= {'Model Name': ['Random Search CV in Random Forest '],'MAPE_Train':[MAPE_train],
      'MAPE_Test':[MAPE_test],'R-squared_Train':[r2_train],'R-squared_Test':[r2_test]}
result5= pd.DataFrame(df5)
```

```
result= result.append(result5)
result
```

Grid Search CV in Random Forest

```
from sklearn.model_selection import GridSearchCV                                    #import libraries

GridRandomForest= RandomForestRegressor(random_state=0)
n_estimator = list(range(1,20,2))
depth= list(range(1,20,2))
grid_search= {'n_estimators':n_estimator, 'max_depth': depth}

#Grid Search CV Random Forest model-
GRF_model= GridSearchCV(GridRandomForest,param_grid=grid_search,cv=10)
GRF_model= GRF_model.fit(X_train,y_train)

best_parameters = GRF_model.best_params_                                            #Best parameters for model

best_model = GRF_model.best_estimator_                                              #Best model

GRF_test = best_model.predict(X_test)                                               #Model prediction on test data
GRF_train = best_model.predict(X_train)                                             #Model prediction on train data

MAPE_test= MAPE(y_test,GRF_test)                                                    #Model performance on test data
MAPE_train= MAPE(y_train,GRF_train)                                                 #Model performance on train data
```

```
r2_test=r2_score(y_test,GRF_test)                                    #r2 value for test data
r2_train= r2_score(y_train,GRF_train)                                #r2 value for train data

print("Best Parameter="+str(best_parameters))
print("Best Model="+str(best_model))
print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage Error for test data="+str(MAPE_test))
print("R^2_score for train data="+str(r2_train))
print("R^2_score for test data="+str(r2_test))
```

```
df6= {'Model Name': ['Grid Search CV in Random Forest '],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],
      'R-squared_Train':[r2_train],'R-squared_Test':[r2_test]}
result6= pd.DataFrame(df6)
```

```
result= result.append(result6)
result
```

Random Search CV in Gradient Boosting

```
from sklearn.model_selection import RandomizedSearchCV                    #import libraries

RandomGradientBoosting = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(1,100,2))
depth = list(range(1,20,2))
random_search = {'n_estimators':n_estimator, 'max_depth': depth}

#Random Gradient Boosting model-
RGB_model= RandomizedSearchCV(RandomGradientBoosting,param_distributions= random_search,n_iter=3,cv=10)
RGB_model= RGB_model.fit(X_train,y_train)

best_parameters = RGB_model.best_params_                                  #Best parameters for model
best_model = RGB_model.best_estimator_                                    #Best model
```

```
RGB_test = best_model.predict(X_test)                                #Model prediction on test data
RGB_train = best_model.predict(X_train)                              #Model prediction on train data

MAPE_test= MAPE(y_test,RGB_test)                                     #Model performance on test data
MAPE_train= MAPE(y_train,RGB_train)                                  #Model performance on train data

r2_test=r2_score(y_test,RGB_test)                                    #r2 value for test data
r2_train= r2_score(y_train,RGB_train)                                #r2 value for train data

print("Best Parameter="+str(best_parameters))
print("Best Model="+str(best_model))
print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage Error for test data="+str(MAPE_test))
print("R^2_score for train data="+str(r2_train))
print("R^2_score for test data="+str(r2_test))
```

```
df7= {'Model Name': ['Random Search CV in Gradient Boosting '],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],
      'R-squared_Train':[r2_train],'R-squared_Test':[r2_test]}
result7= pd.DataFrame(df7)
```

```
result= result.append(result7)
result
```

Grid Search CV in Gradient Boosting

```
from sklearn.model_selection import GridSearchCV                          #import libraries

GridGradientBoosting= GradientBoostingRegressor(random_state=0)
n_estimator = list(range(1,20,2))
depth= list(range(1,20,2))
grid_search= {'n_estimators':n_estimator, 'max_depth': depth}
```

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

best_parameters = GGB_model.best_params_                                                #Best parameters for model
best_model = GGB_model.best_estimator_                                                  #Best model

GGB_test = best_model.predict(X_test)                                                   #Model prediction on test data
GGB_train = best_model.predict(X_train)                                                 #Model prediction on train data

MAPE_test= MAPE(y_test,GGB_test)                                                        #Model performance on test data
MAPE_train= MAPE(y_train,GGB_train)                                                     #Model performance on train data

r2_test=r2_score(y_test,GGB_test)                                                       #r2 value for test data
r2_train= r2_score(y_train,GGB_train)                                                   #r2 value for train data

print("Best Parameter="+str(best_parameters))
print("Best Model="+str(best_model))
print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage Error for test data="+str(MAPE_test))
print("R^2_score for train data="+str(r2_train))
print("R^2_score for test data="+str(r2_test))
```

```
df8= {'Model Name': ['Grid Search CV in Gradient Booster '],'MAPE_Train':[MAPE_train],'MAPE_Test':[MAPE_test],
      'R-squared_Train':[r2_train],'R-squared_Test':[r2_test]}
result8= pd.DataFrame(df8)
```

```
result= result.append(result8)
result
```

Note: all the graphs and output in the reports are take from R environment.

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-learningbootcamp/

THANK YOU