Predicting Bike Rental Count

Project Report

By

Ishan Sharma

Contents

.

Chapter 1

Introduction

1.1 Problem Statement

On a given day, number of bikes that would be rented might depend on various factors. The prediction of the rental count on a daily basis might help rental companies in allocating the resources efficiently.
The aim of the project is to accurately estimate the number of bikes that would be rented based on the environmental and seasonal settings.
Eg. Number of bikes rented might be higher if the weather outside is pleasant. Through the techniques of data analysis and machine learning, we would be modelling such relationships between the Response & the Predictor variables.

1.2 Data

Our task is to build regression models which will predict the number of bikes rented depending on multiple environmental and seasonal factors. Given below is a sample of the data set that we are using to predict the number of bikes rented on a given day.

Table 1.1: Bike Rental Sample Data (Columns: 1-6)

Instant	dteday	season	yr	mnth	holiday
1	2011-01-01	1	0	1	0
2	2011-01-02	1	0	1	0
3	2011 -01- 03	1	0	1	0
4	2011 -01 -04	1	0	1	0
5	2011 -01 -05	1	0	1	0

Table 1.2: Bike Rental Sample Data (Columns: 7-12)

weekday	workingday	weathersit	temp	atemp	hum
6	0	2	0.344167	0.363625	0.805833
0	0	2	0.363478	0.353739	0.696087
1	1	1	0.196364	0.189405	0.437273
2	1	1	0.2	0.212122	0.590435
3	1	1	0.226957	0.22927	0.436957

Table 1.3: Bike Rental Sample Data (Columns: 13-16)

windspeed	casual	registered	cnt
0.160446	331	654	985
0.248539	131	670	801
0.248309	120	1229	1349
0.160296	108	1454	1562
0.1869	82	1518	1600

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**:
   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

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_max), where, 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 & registered

From the table, we have the following 14 variables, using which we have to accurately estimate the number of bike rentals on a given day:

Table 1.5: Predictor Variables

S.No.	Predictor
1	dteday
2	season
3	yr
4	mnth
5	holiday
6	weekday
7	workingday
8	weathersit
9	temp
10	atemp
11	hum
12	windspeed
13	casual
14	registered

Note: The “instant” column is not included in the Predictor variables as it is just an added index in the data file.

Chapter 2

Methodology

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 often called as **Exploratory Data Analysis**.
To start the process, we would first evaluate whether the given dataset has any kind of missing information.

2.1.1 Missing Value Analysis

If the records (observations) in a dataset contain any missing information then it is important to treat those missing values in order to build an optimised prediction model. The missing values, if present, can be imputed using several methods such as KNN imputation, mean imputation or mode imputation.
As a rule of thumb, if any variable in the dataset has more than 30% values missing, then that variable is removed altogether from the model as imputing such a variable would cause highly engineered results.

Missing values in a variable can be checked using **is.null()** function in Python and using **is.na()** in R.
The given dataset does not contain any missing value. So, the dataset is complete and does not require any analysis of such kind.

2.1.2 Datatype Analysis

For proper EDA, it is important that the variables present in the dataset have correct datatype associated with them.
Variables are of 2 types:
1) Numerical
2) Categorical
Categorical variables are further divided into 2 types: Ordinal & Nominal
Numerical variables are those which take numerical values and it makes sense to do arithmetic operations upon them.
Categorical variables are those which take distinct categories as their value and it doesn't make sense to do arithmetic operations on these.
In the given dataset,
Variables: **season**, **yr**, **mnth**, **holiday**, **weekday**, **working day**, **weathersit** are categorical variables having integer encoded values.
Variables: **temp**, **atemp**, **hum**, **windspeed** are numerical variables.

Variables, **casual** and **registered** are numerical variables, which need to be removed for the analysis because their sum directly gives the value of **cnt**, the Target(Response) variable.
Variable, **dteday** needs to be removed because all the information from the date variable is already included in the data such as: Year, Weekday.
Also, the **instant** variable needs to be removed as it is just an added index column in the data file.

2.1.3 Outlier Analysis

Outliers are the data points which are significantly lesser or greater as compared to the other data points. Statistical parameters such as mean, standard deviation are highly sensitive to the outliers because they tend to shift the values towards themselves.
So, in our model, we use the classic approach for removing the outliers, the Tukey's method where the outliers are defined as the data points which are $\pm 1.5SD$, and should be removed from the data.
We visualize the outliers using *boxplots*.
In figure 2.1 we have plotted the boxplots of the 4 predictor variables. As shown in the figure 2.1, there are outliers present in the variables, **hum** and **windspeed**.

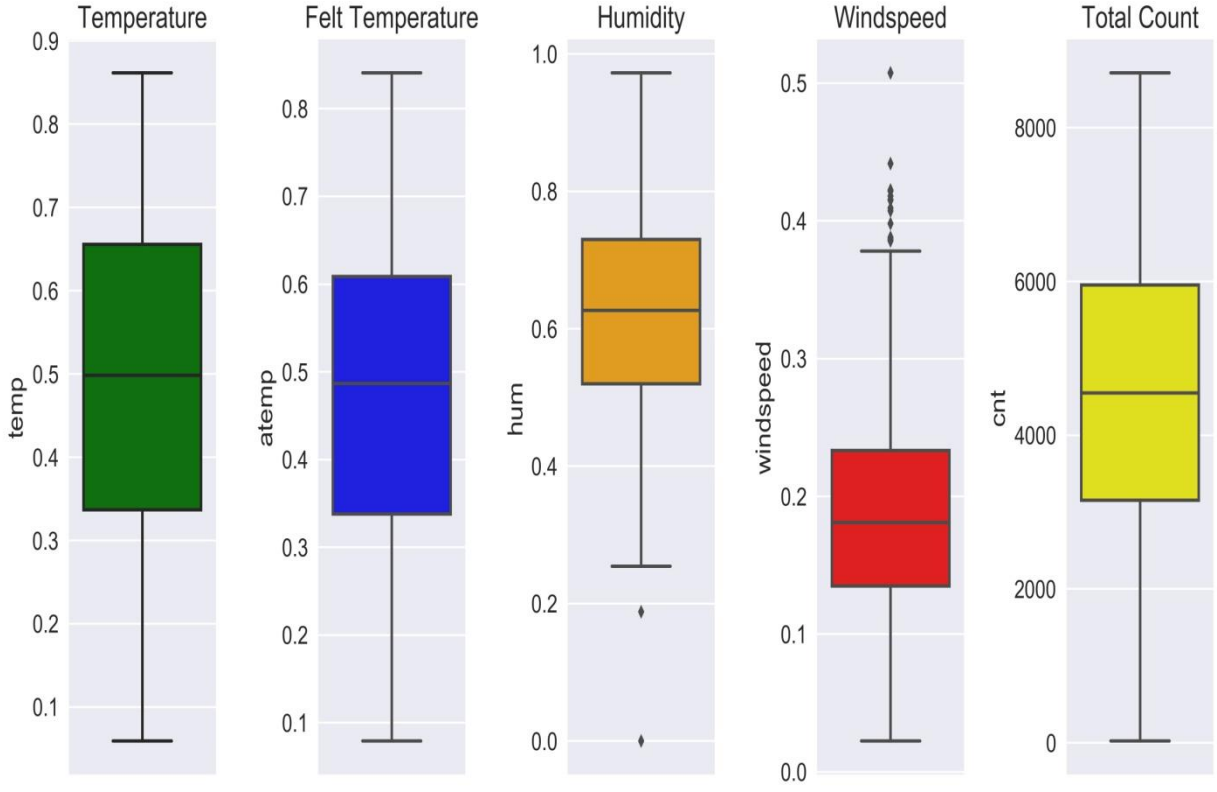


Figure 2.1: Boxplots of Numerical Variables

In R, using the **boxplot.stats()$out**, vector of outliers can be generated and the corresponding observations be removed.

2.1.4 Feature Selection

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

Numerical Feature Selection:

To asses, whether the predictor variables suffer from the problem of multicollinearity, we use correlation heatmap as show in Figure 2.2.

Note that correlation analysis can be done only for the numerical variables.

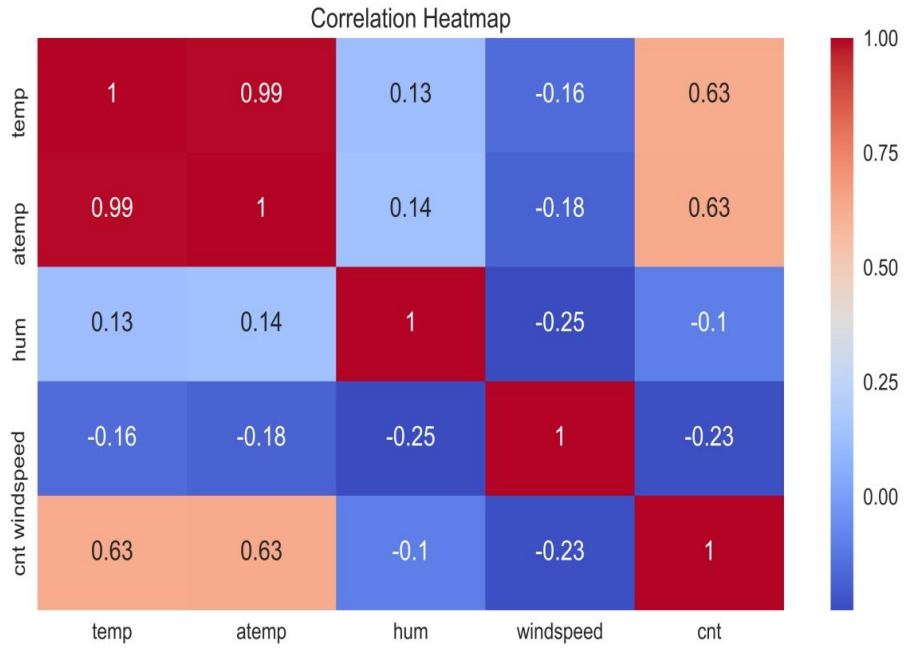


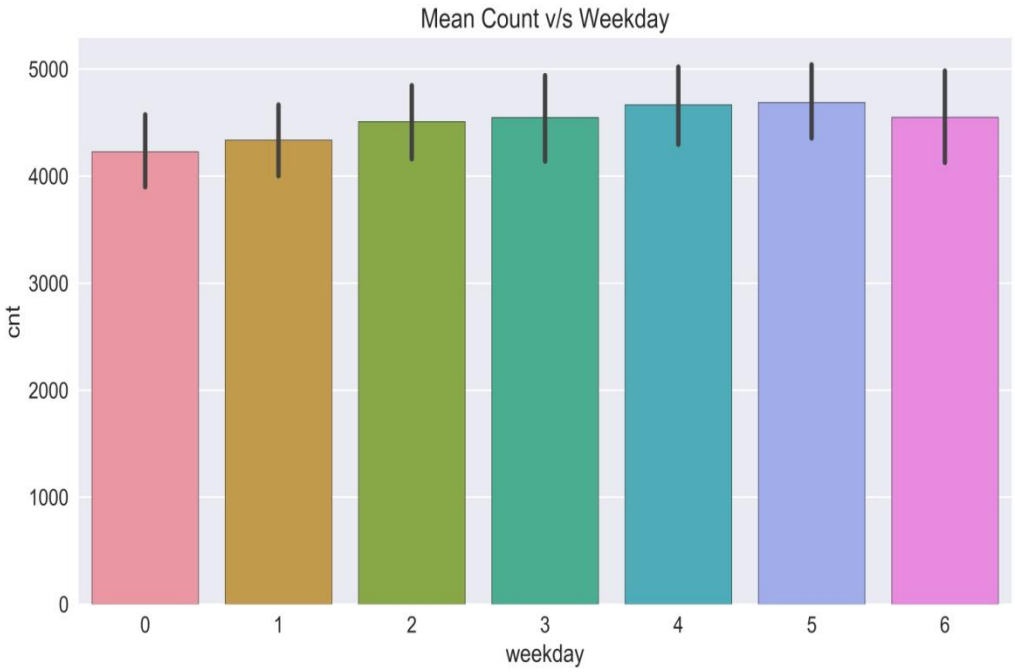
Figure 2.2: Correlation Heatmap

Since, the correlation coefficient of **temp** and **atemp** variables is 0.99, which is unusually very high, one of these predictor variables should be dropped from the model. So, we drop **atemp** predictor variable.

Categorical Feature Selection:

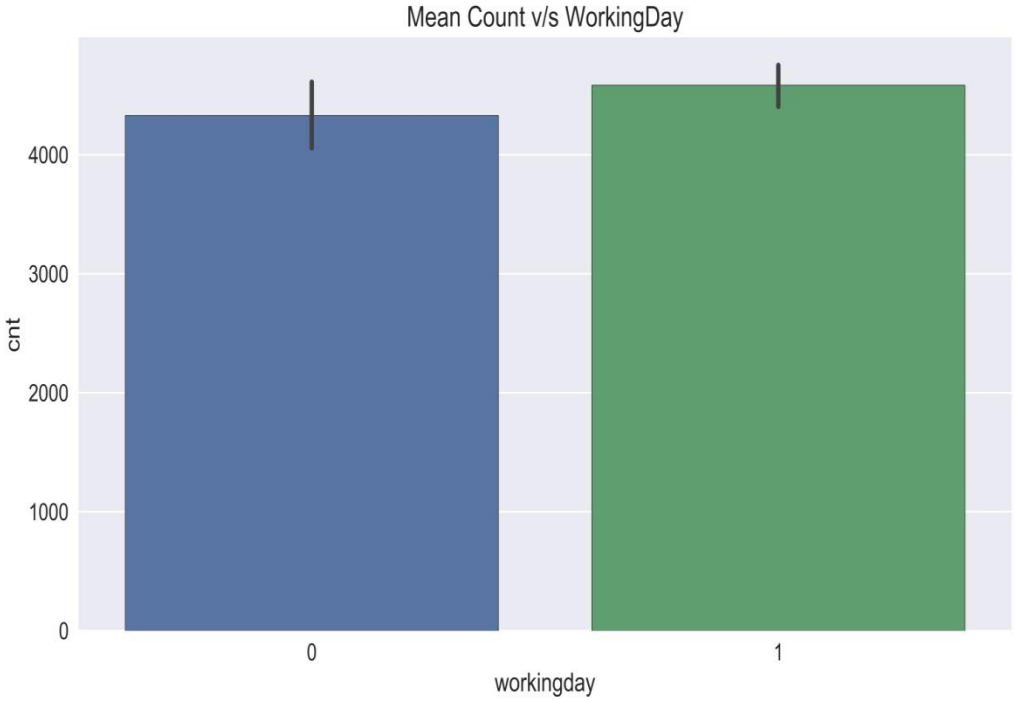
ANOVA test is for testing whether a categorical variable has any significant effect on the Response variable which is numerical.

Out of all the barplot visualisations, (plots can be seen in the appendix section), there was considerable difference in the means of groups in categorical variables, but the difference in group means of day of the week variable, **weekday** and **workingday** variable were very less. So let's do an ANOVA test on these variable because these might be the candidates for removal if p-value > 0.05



ANOVA Test on **weekday** variable (R code):

```
anov_week = aov(cnt ~ weekday, data = df)
summary(anov_week)
##              Df     Sum Sq Mean Sq F value Pr(>F)
## weekday       6   17571224 2928537   0.782  0.584
## Residuals   710 2659256584 3745432
```



ANOVA Test on **workingday** variable (R Code):

```
anov_work = aov(cnt ~ workingday, data = df)
summary(anov_work)
##              Df     Sum Sq Mean Sq F value Pr(>F)
## workingday    1    8494340 8494340   2.276  0.132
## Residuals   715 2668333468 3731935
```

Since, the p-values for these two variables are very high and low respective F-statistic, dropping these variables for the prediction model of bike rental count.

2.2 Model Development

2.2.1 Model Selection

The dependent variable can fall in any of the four categories:
1. Numerical
2. Categorical

If the dependent variable,(in our case **cnt)**, is Numerical, the only predictive analysis that we can perform is **Regression** and if the dependent variable is Categorical, the predictive analysis performed is **Classification.**
The dependent variable in our project is numerical, so regression models would be used.

Model development should always start from simpler models and then proceed to more complex ones.
In this project, we would be analysing three Regression models:

1) Linear Regression
2) Decision Tree Regression
3) Random Forest Regression

Based on the performance in accurately predicting the results, the best model out of the three will be selected.

2.2.2 Linear Regression

Linear regression is a simple approach to supervised learning. It assumes that the dependence of dependent variable, Y on predictor variables X1, X2, . . . Xp is linear.

```
# Fitting the model to the training set:
lin_reg = lm(formula= cnt ~., data= df_train)

# Predciting the values for the test set:
pred_lm = predict(lin_reg, newdata = df_test)


# Mean Absolute Error:
cat("Linear Regression\n",paste("Mean Absolute Error:", mae(actual =
df_test$cnt, predicted = pred_lm)))

## Linear Regression
##  Mean Absolute Error: 617.19156962504
```

2.2.3 Decision Tree Regression

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., Hours Played) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data.

```
library(rpart)

# Fitting the model to the training set:
d_tree = rpart(formula= cnt ~., data= df_train)

# Predicting the values for the test set:
pred_dt = predict(d_tree, df_test)

# Mean Absolute Error:
cat("Decision Tree Regression\n",paste("Mean Absolute Error:", mae(actual
= df_test$cnt, predicted = pred_dt)))
```

```
## Decision Tree Regression
##  Mean Absolute Error: 785.704457857585
```

2.2.4 Random Forest Regression

Random forest regression requires passing the value of **ntree** parameter which is the no. of tress employed in estimating the dependent variable.
Using the elbow method, which tests the error rate for various values of **ntree**, the optimum value of **ntree** can be used. (See full code in Appendix)

```
library(randomForest)
# Fitting the model to the training set:

rf = randomForest(formula= cnt~., data= df_train, ntree = opt)

# Predicting the values for the test set:
pred_rf = predict(rf, df_test)

# Mean Absolute Error:
cat("Random Forest Regression\n",paste("Mean Absolute Error:", mae(actual
= df_test$cnt, predicted = pred_rf)))
```

```
## Random Forest Regression
##  Mean Absolute Error: 608.156896068162
```

Chapter 3

Conclusion

3.1 Model Evaluation

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using any of the following criteria:
1. Predictive Performance
2. Interpretability
3. Computational Efficiency
In our case of Bike Rental Data, the latter two, *Interpretability* and *Computation Efficiency*, do not hold much significance. Therefore we will use *Predictive performance* as the criteria to compare and evaluate models.
Predictive performance can be measured by comparing Predictions of the models with real values of the target variables, and calculating some average error measure.

3.1.1 Mean Absolute Error [MAE]

MAE is one of the error measures used to calculate the predictive performance of the model. We will apply this measure to our models that we have generated in the previous section.

```
# Mean Absolute Error:

cat("Linear Regression\n",paste("Mean Absolute Error:", mae(actual =
df_test$cnt, predicted = pred_lm)))
```

```
## Linear Regression
##  Mean Absolute Error: 617.19156962504
```

```
cat("Decision Tree Regression\n",paste("Mean Absolute Error:", mae(actual
= df_test$cnt, predicted = pred_dt)))
```

```
## Decision Tree Regression
##  Mean Absolute Error: 785.704457857585
```

```
cat("Random Forest Regression\n",paste("Mean Absolute Error:", mae(actual
= df_test$cnt, predicted = pred_rf)))
```

```
## Random Forest Regression
##  Mean Absolute Error: 608.156896068162
```

3.1.2 Mean Squared Error [MSE]

MSE is just the square of MAE. Hence, it can be obtained by squaring the Mean Absolute Error.

3.2 Model Selection

Error Metrics of Random Forest Regression are the best out of all the tested models, so Random Forest Regression model is selected model for the Prediction of Bike Rental Count.

Appendix A - Extra Figures

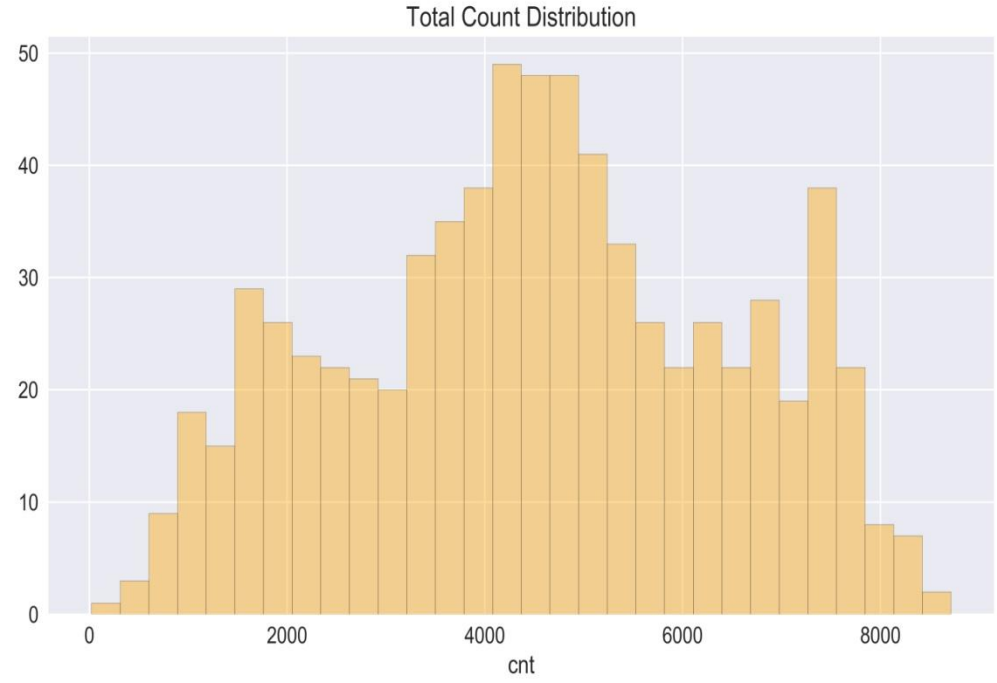


Figure 3.1: Distribution of Target Variable

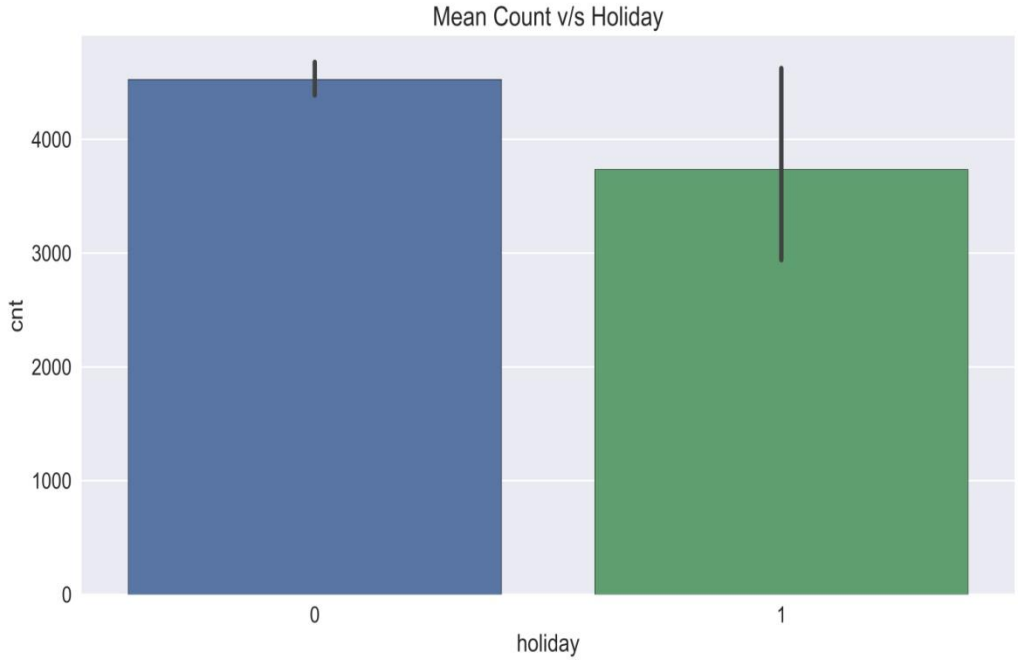


Figure 3.2: Barplot of Holdiay

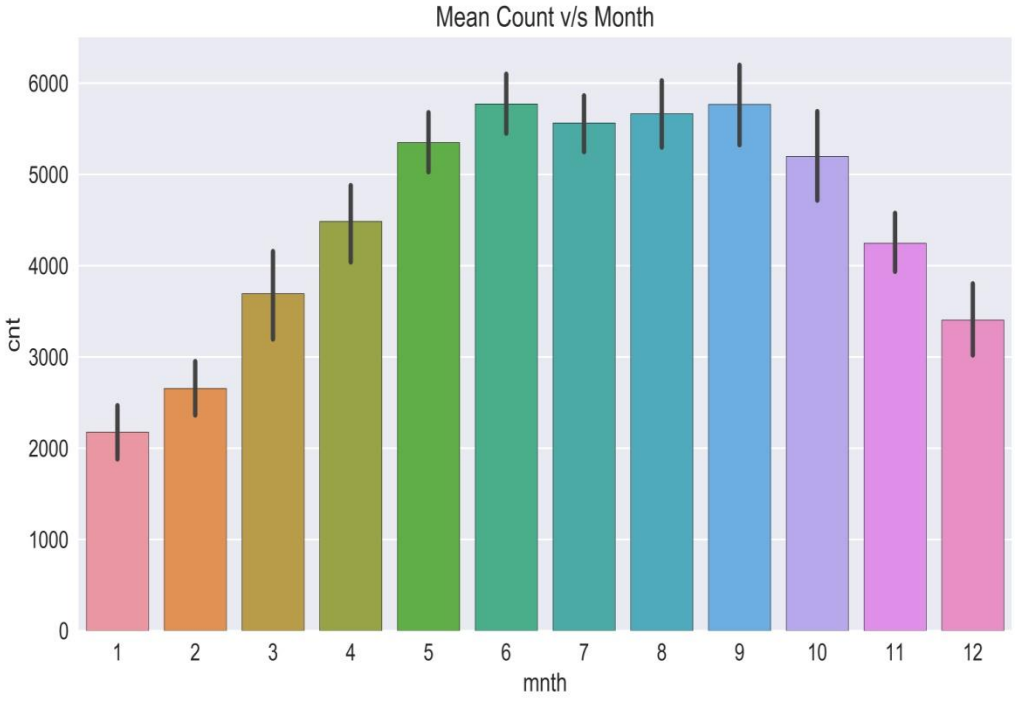


Figure 3.3: Barplot of Month

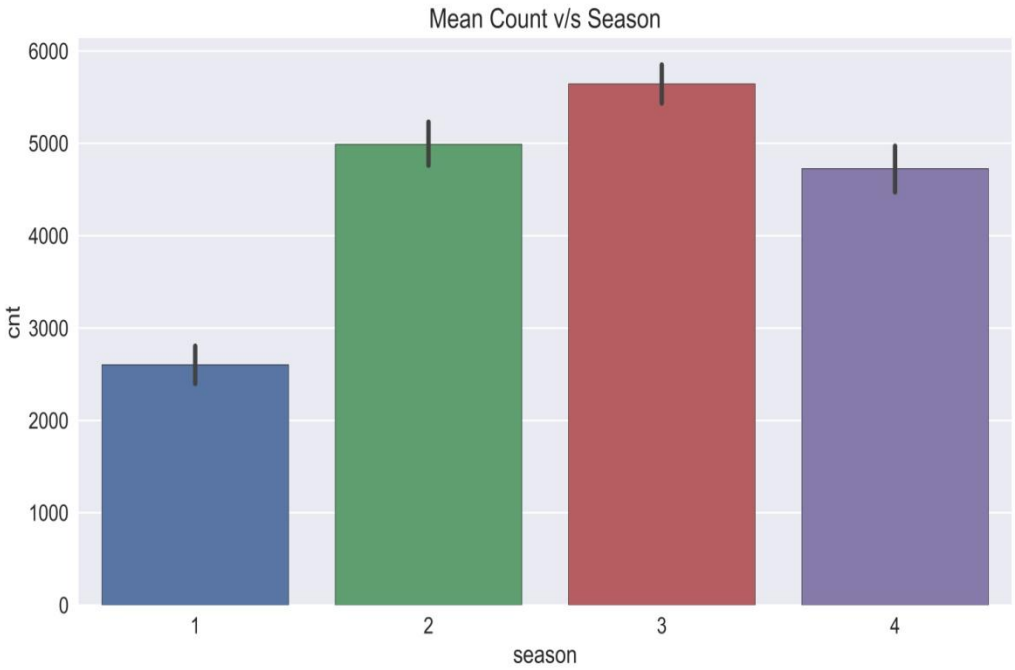


Figure 3.4: Barplot of Season

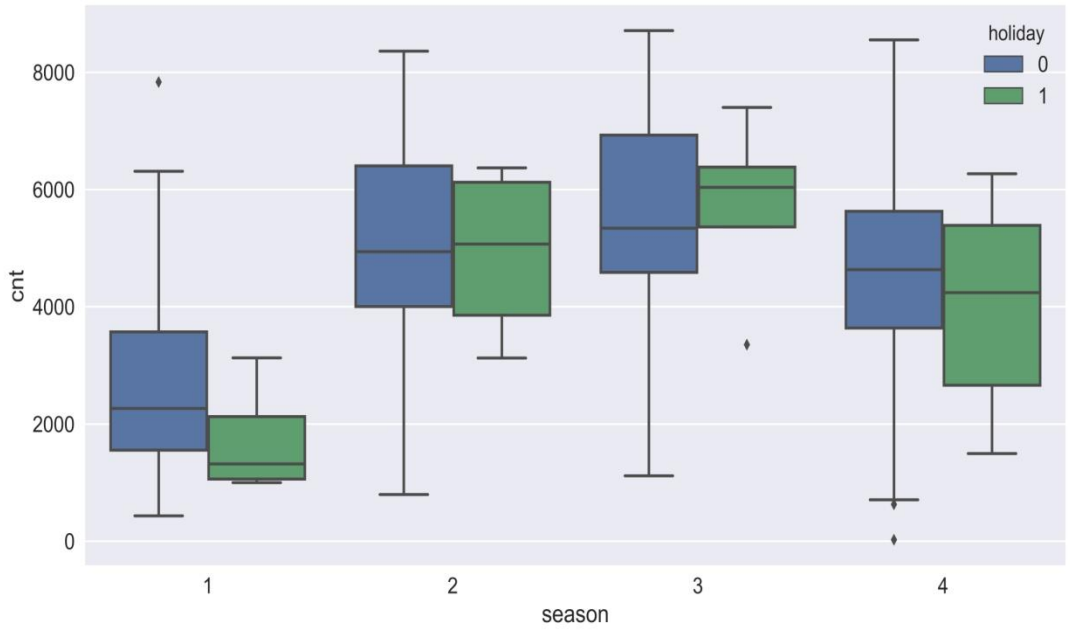


Figure 3.5: Count v/s Season Boxplot (hue= holiday)

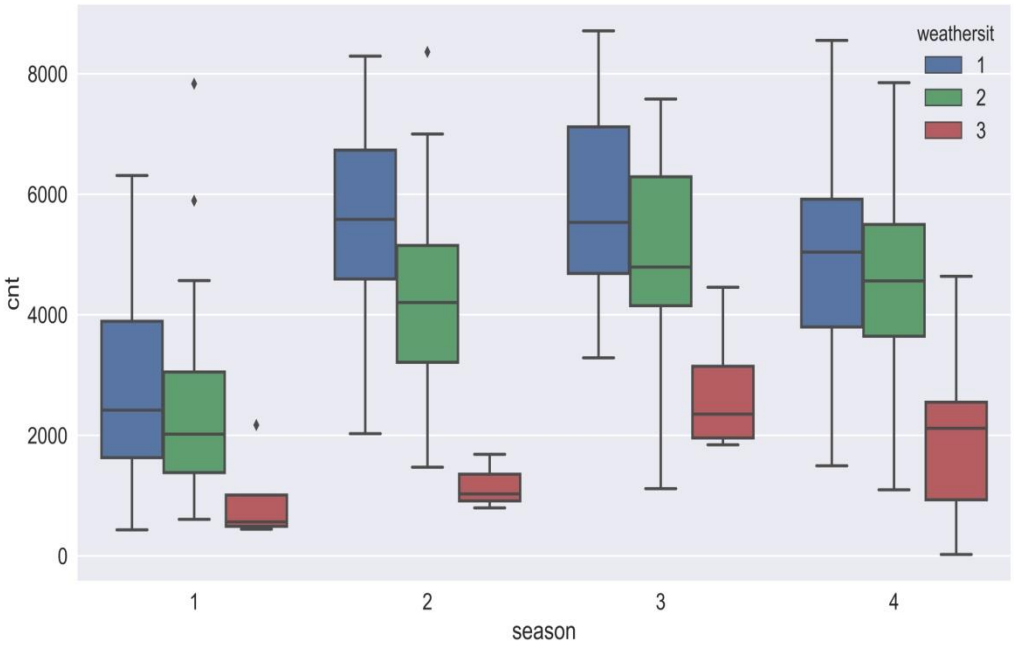


Figure 3.6: Count v/s Season Boxplot (hue= Weather)

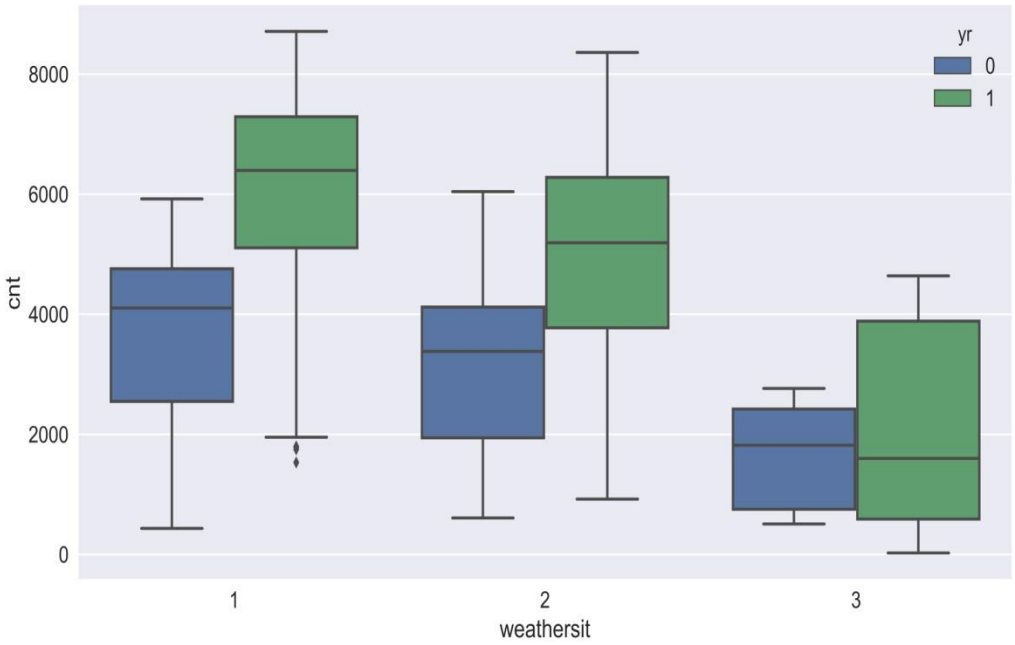


Figure 3.7: Count v/s Weather Boxplot (hue= Year)

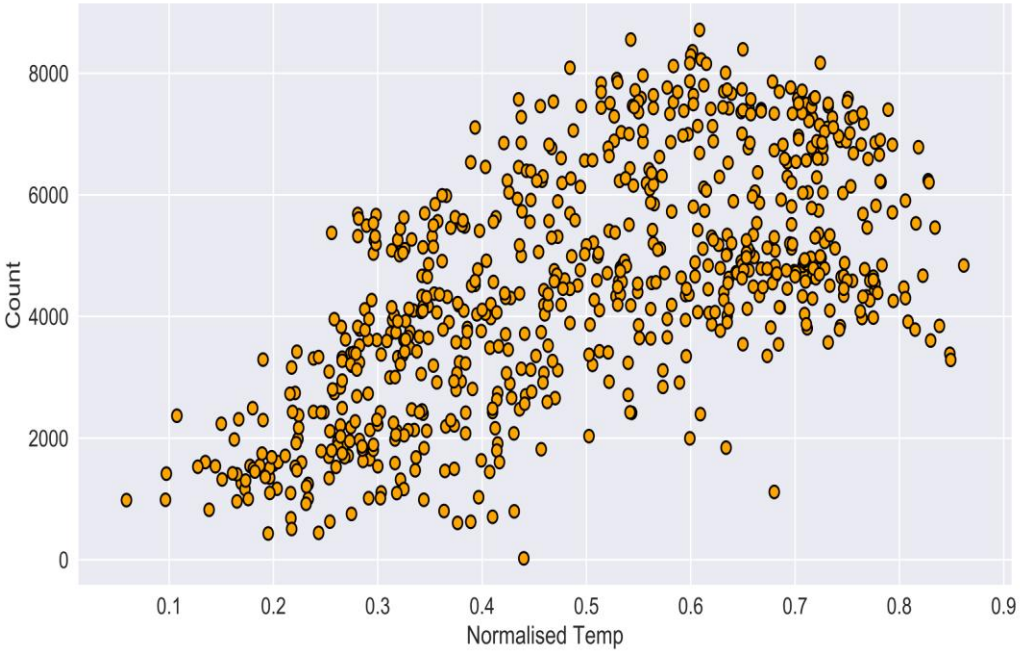


Figure 3.8: Count v/s Normalised Temperature

Appendix B - R Code

```
# Removing all the prior stored objects:

rm(list=ls())

# Setting the working directory:

setwd("W:/DataScience/edWisor/Project")

# Importing Libraries:
library(dplyr)
library(ggplot2)
library(gridExtra)
library(ggcorrplot)


# Loading the data:

df = read.csv("day.csv", sep=",", header = TRUE)

# Checking the dataframe head:

head(df)


# **** DATA PRE PROCESSING **** #


# Checking the strucutre & datatype information of data:

str(df)


# *Missing Value Analysis*


data.frame(apply(df,2,function(x){sum(is.na(x))}))


# No missing value in the data. All records are complete.


# Dropping the unnecessary columns:
```

```
df = select(df,-c(instant,dteday,casual,registered))



# List of categorical & numerical columns:

cat_var =
c("season","yr","mnth","holiday","weekday","workingday","weathersit")
num_var = c("temp","atemp","hum","windspeed","cnt")



# Converting variables in the proper datatype:

df =  df %>% mutate_at(.vars = cat_var,.funs = as.factor)
str(df)



#*Visualisations for exploring relationships b/w the variables*

options(scipen=999)

# Barplot of Mean Count v/s Season:



ggplot(data=df, aes(x= season, y=cnt,fill=season))+ stat_summary(fun.y =
mean, geom = "bar") + ggtitle("Mean Count v/s Season") +
  scale_fill_hue('Season', breaks = levels(df$season),labels=c('spring',
'summer', 'fall', 'winter')) +  labs(y="Mean Count")



# Barplot of Mean Count v/s Weather Situation:



ggplot(data=df, aes(x= weathersit, y=cnt, fill= weathersit)) +
stat_summary(fun.y = mean, geom = "bar") +  ggtitle("Mean Count v/s
Weather Situation") +  scale_fill_hue('weathersit', breaks =
levels(df$weathersit),labels=c("Clear","Cloudy","Rain"))  +  labs(y="Mean
Count")







# Barplot of Mean Count v/s Holiday:



ggplot(data=df, aes(x= holiday, y=cnt, fill=holiday)) + stat_summary(fun.y
= mean, geom = "bar") + ggtitle("Mean Count v/s Holiday") +
scale_fill_hue('Holiday Status', breaks = levels(df$holiday),labels=c("No
Holiday", "Holiday"))  +   labs(y="Mean Count")
```

```
# Barplot of Mean Count v/s Weekday:


ggplot(data=df, aes(x= weekday, y=cnt, fill=weekday)) + stat_summary(fun.y
= mean, geom = "bar") + ggtitle("Mean Count v/s Day of the Week") +
scale_fill_hue("Day of the Week", breaks =
levels(df$weekday),labels=c("Sun","Mon","Tue","Wed","Thu","Fri","Sat")) +
labs(y="Mean Count")


# Barplot of Mean Count v/s Working Day:


ggplot(data=df, aes(x= workingday, y=cnt, fill=workingday)) +
stat_summary(fun.y = mean, geom = "bar") + ggtitle("Mean Count v/s Working
Day") + scale_fill_hue('Day Type', breaks =
levels(df$workingday),labels=c("Not Working","Working")) +  labs(y="Mean
Count")




# Barplot of Mean Count v/s Month:


ggplot(data=df, aes(x= mnth, y=cnt,fill=mnth)) + stat_summary(fun.y =
mean, geom = "bar") +  ggtitle("Mean Count v/s Month") +
scale_fill_hue('Months', breaks =
levels(df$mnth),labels=c("Jan","Feb","March","April","May","June","July","
Aug","Sep","Oct","Nov","Dec"))+ labs(y="Mean Count")


# Barplot of Mean Count v/s Year:


ggplot(data=df, aes(x= yr, y=cnt, fill=yr)) + stat_summary(fun.y = mean,
geom = "bar") +  ggtitle("Mean Count v/s Year")+scale_fill_hue('Year',
breaks = levels(df$yr),labels=c("2011","2012"))  +  labs(x="Year",y="Mean
Count")


# Distribution of Target variable, "cnt":


ggplot(data= df, aes(x= cnt)) + geom_histogram(bins = 30,colour="orange")
+ ggtitle("Total Rental Count Distribution")+ labs(y="")
```

```
# Scatterplots of Numerical Variables:

  s1 = ggplot(data=df, aes(x= temp, y=cnt)) + geom_point(shape = 21,
colour = "black", fill = "white", size= 2, stroke= 0.5) + ggtitle("Rental
Count v/s Temperature") + labs(x="Normalised Temperature",y="Count")


s2 = ggplot(data=df, aes(x= windspeed, y=cnt)) + geom_point(shape = 21,
colour = "black", fill = "white", size= 2, stroke= 0.5)+ ggtitle("Rental
Count v/s Windspeed") + labs(x="Normalised Windspeed",y="Count")


s3 = ggplot(data=df, aes(x= hum, y=cnt)) + geom_point(shape = 21, colour =
"black", fill = "white", size= 2, stroke= 0.5)+ ggtitle("Rental Count v/s
Humidity") + labs(x="Normalised Humidity",y="Count")


s4 = ggplot(data=df, aes(x= atemp, y=cnt)) + geom_point(shape = 21, colour
= "black", fill = "white", size= 2, stroke= 0.5) + ggtitle("Rental Count
v/s Felt Temperature") + labs(x="Normalised Felt Temperature",y="Count")


grid.arrange(s1, s2, s3, s4, nrow = 2,ncol=2)




#*Outlier Analysis*


#Boxplot of numerical variables:




box1 = ggplot(data = df, aes(y = temp)) + geom_boxplot() + theme_classic()
+ ggtitle("Temp.")

box2 = ggplot(data = df, aes(y = atemp)) + geom_boxplot() +
theme_classic() + ggtitle("Felt Temp.")

box3 = ggplot(data = df, aes(y = hum)) + geom_boxplot() + theme_classic()
+ ggtitle("Humidity")

box4 = ggplot(data = df, aes(y = windspeed)) + geom_boxplot() +
theme_classic() + ggtitle("Windspeed")

box5 = ggplot(data = df, aes(y = cnt)) + geom_boxplot() +theme_classic() +
ggtitle("Count")
```

```
grid.arrange(box1,box2,box3,box4,box5, nrow=1)



# Removing Outliers using boxplot method:


for(i in num_var)
{
  print(i)
  val = df[,i][df[,i] %in% boxplot.stats(df[,i])$out]
  print(length(val))
  df = df[which(!df[,i] %in% val),]
}


# *Feature Selection*


# Generating  a correlation matrix:


corr_mat = cor(select(df, num_var))
ggcorrplot(corr_mat,title = "Correlation Heatmap",type = "upper",ggtheme =
theme_classic(),lab=TRUE)






# Dropping the column "atemp" from the dataset:


df = select(df,-c("atemp"))


 # ANOVA Test on "weekday" variable:
anov_week = aov(cnt ~ weekday, data = df)
summary(anov_week)
```

```
# ANOVA Test on "workingday" variable:

anov_work = aov(cnt ~ workingday, data = df)
summary(anov_work)



  # Dropping the "weekday" & "workingday" variables because, the p-value
is very high, F-statistic is low:

df = select(df, -c("weekday","workingday"))



  # *** MODEL DEVELOPMENT ***



  # Splitting the dataset into training & testing set:



library(caTools)
set.seed(123)
split = sample.split(df$cnt, SplitRatio = 0.25)
df_train = subset(df, split== TRUE)
df_test = subset(df, split== FALSE)




# Loading Error Metrics package:

library(Metrics)

 # Linear Regression Model:

 # Fitting the model to the training set:


lin_reg = lm(formula= cnt ~., data= df_train)

 # Predciting the values for the test set:

pred_lm = predict(lin_reg, newdata = df_test)

 # Mean Absolute Error:

 cat("Linear Regression\n",paste("Mean Absolute Error:", mae(actual =
df_test$cnt, predicted = pred_lm)))




# Decision Tree Regression Model:
```

```
library(rpart)


# Fitting the model to the training set:

d_tree = rpart(formula= cnt ~., data= df_train)

 # Predicting the values for the test set:

pred_dt = predict(d_tree, df_test)


# Mean Absolute Error:

cat("Decision Tree Regression\n",paste("Mean Absolute Error:", mae(actual
= df_test$cnt, predicted = pred_dt)))




# Random Forest Regression Model:


library(randomForest)

# Elbow method to find the optimal value of "ntree":


error= vector()

k= seq(10,300,by=10)

for(i in k)

  {

    rf_elbow = randomForest(formula= cnt ~., data= df_train, ntree= i)

    pred_elbow = predict(rf_elbow, newdata =  df_test)

    err = mae(actual= df_test$cnt, predicted = pred_elbow)

    error = c(error,err)

  }

  plot(x= k,y= error,"b")


  # Getting the Optimum value for ntree parameter:

  for (i in 1:length(k))

  {
```

```
    if(error[i]==min(error))
    {
      opt = k[i]
    }
  }

# Fitting the model to the training set:

rf = randomForest(formula= cnt~., data= df_train, ntree = opt)
  # Predicting the values for the test set:

pred_rf = predict(rf, df_test)
  # Mean Absolute Error:
cat("Random Forest Regression\n",paste("Mean Absolute Error:", mae(actual
= df_test$cnt, predicted = pred_rf)))
```

Appendix C - Python Code

```
# Importing libraries for Data Cleaning and EDA:



import os
import numpy as np
import pandas as pd
import matplotlib as mlt
import matplotlib.pyplot as plt
import seaborn as sbn
get_ipython().run_line_magic('matplotlib', 'inline')
mlt.rcParams["patch.force_edgecolor"] = True
sbn.set_style("whitegrid")
sbn.set(font_scale=1.3)



# Loading the data into pandas dataframe:

df = pd.read_csv("day.csv",sep=",")

# Checking the head of data frame:

df.head()




# # Data Pre Processing:

# Checking the dimensions & datatype information of data:



df.info()



# ***Missing Value Analysis:***

# Checking for missing values in the data:



df.isnull().sum()




# Dropping the unnecessary columns:
```

```
df = df.drop(columns=["instant", "dteday","casual","registered"],axis=1)

# List of numerical variables:

num_var = ["temp","atemp","hum","windspeed","cnt"]


#List of categorical variables:

cat_var =
["season","yr","mnth","holiday","weekday","workingday","weathersit"]

# Converting the variables in correct datatype:


for i in cat_var:

    df[i] = df[i].astype("category")

df.info()

# ***Visualisations for exploring relationships b/w the variables:***

# Barplot of Mean Count v/s Season


plt.figure(figsize=(12,6))
sbn.barplot(x="season", y="cnt", data=df)
plt.title(" Mean Count v/s Season ")


# Barplot of Mean Count v/s Weather Situation:


plt.figure(figsize=(12,6))
sbn.barplot(x="weathersit",y='cnt',data=data,palette="magma")
plt.title(" Mean Count v/s Weather Situation")


# Barplot of Mean Count v/s Holiday


plt.figure(figsize=(12,6))
sbn.barplot(x="holiday",y="cnt",data=df)
plt.title("Mean Count v/s Holiday")
```

```
# Barplot of Mean Count v/s Weekday:


plt.figure(figsize=(12,6))
sbn.barplot(x="weekday",y="cnt",data=df)
plt.title("Mean Count v/s Weekday")


# Barplot of Mean Count depending if the day is working or not:


plt.figure(figsize=(12,6))
sbn.barplot(x="workingday", y="cnt", data=df)
plt.title("Mean Count v/s WorkingDay")


# Barplot of Mean Count v/s Month:


plt.figure(figsize=(12,6))
sbn.barplot(x="mnth", y="cnt", data=df,)
plt.title("Mean Count v/s Month")


# Distribution of Target variable:


plt.figure(figsize=(12,6))
sbn.distplot(df["cnt"],bins=30,color="orange",kde=False)
plt.title("Total Count Distribution")


# All the possible scatterplots of numerical variables in the dataset:


plt.figure(figsize=(12,6))
sbn.pairplot(df[num_var])

# ***Outlier Analysis:***

# Checking for outliers in the numerical variables:
# If there are some outliers found in the independent variables, for
making an optimised model, it's best to remove these outliers.

# Making the figure and subplots for the boxplot visualisation:

fig, axes = plt.subplots(nrows=1, ncols=5,figsize=(12,6))
```

```
# Using Seaborn to create boxplots:



sbn.boxplot("temp", data=df, ax=axes[0], orient="v",
color="green").set_title("Temperature")

sbn.boxplot("atemp", data=df, ax=axes[1], orient="v",
color="blue").set_title("Felt Temperature")

sbn.boxplot("hum", data=df, ax=axes[2], orient="v",
color="orange").set_title("Humidity")

sbn.boxplot("windspeed", data=df, ax=axes[3], orient="v",
color="red").set_title("Windspeed")

sbn.boxplot("cnt", data=df, ax=axes[4], orient="v",
color="yellow").set_title("Total Count")



# For better visibility of the labels:
plt.tight_layout()

# Detect and delete outliers from data:

# List of numerical variables:

num_var = ["temp","atemp","hum","windspeed","cnt"]

for i in num_var:

    print("Variable: ",i)
    q75, q25 = np.percentile(df.loc[:,i], [75 ,25])
    iqr = q75 - q25
    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)
    print("Min Value: ",min)
    print("Max Value: ",max)
    print()
    df = df.drop(df[df.loc[:,i] < min].index)
    df = df.drop(df[df.loc[:,i] > max].index)

# ***Feature Selection:***

# Correlation heatmap for numerical variables in the dataset:

plt.figure(figsize=(12,6))
sbn.heatmap(df[num_var].corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")



# Dropping the column "atemp" from the dataset:

df = df.drop("atemp",axis=1)
```

```
# Importing library for ANOVA test:

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

# ANOVA test for "weekday" categorical variable:



anova_week = ols("cnt ~ C(weekday)", data=df).fit()
print(anova_week.summary())

# Making an ANOVA table:

aov_table_week = sm.stats.anova_lm(anova, typ=2)
print(aov_table_week)

# ANOVA test for "workingday" categorical variable:

anova_work = ols("cnt ~ C(workingday)", data=df).fit()
print(anova_work.summary())

# Making an ANOVA table:

aov_table_work= sm.stats.anova_lm(anova_work, typ=2)
print(aov_table_work)



# Dropping the "weekday" & "workingday" variables because, the p-value is
very high, F-statistic is low:

df = df.drop(["weekday","workingday"],axis=1)



## Model Development:

#Importing the required libraries:

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import model_selection, preprocessing
from sklearn import metrics

# Making separate dataframes of dependent & independent variables:



X = df.drop("cnt",axis=1)
y = df["cnt"]

# Dummy variables for the categorical columns:
```

```
X = pd.get_dummies(data=X,
columns=["season","yr","mnth","weathersit","holiday"],drop_first=True)

X.head()

# Splitting the data into training & testing set:

X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.30,
random_state= 1)



# ***Model Selection:***

# Creating Linear Model object:

lm = LinearRegression()

# Fitting the training data:

lm.fit(X_train,y_train)

# Predicting the values for test data:

pred_lm = lm.predict(X_test)

# Plot of Predicted values v/s Observed Values:

plt.figure(figsize=(12,6))
plt.scatter(x=y_test, y=pred_lm, color="orange", s=50, edgecolor="black" )
plt.plot(y_test, y_test, color="green")
plt.xlabel("Observed Values")
plt.ylabel("Predicted Values")
plt.title(" Linear Regression Results")

#Error Metrics:

print("Linear Regression: ")

print("The Accuracy Score is:", lm.score(X_test, y_test))

print("The Mean Absoulte Error[MAE] is:",
metrics.mean_absolute_error(y_test, pred_lm))

print("The Mean Squared Error[MSE] is:",
metrics.mean_squared_error(y_test, pred_lm))



# Creating Decision Tree Regression Model object:

dt = DecisionTreeRegressor()

# Fitting the training data:

dt.fit(X_train,y_train)
```

```
# Predicitng the values of test dataset:

pred_dt = dt.predict(X_test)

# Plot of Predicted values v/s Observed Values:

plt.figure(figsize=(12,6))
plt.scatter(x=y_test, y=pred_dt, color="orange", s=50, edgecolor="black")
plt.plot(y_test, y_test, color="green")
plt.xlabel("Observed Values")
plt.ylabel("Predicted Values")
plt.title(" Decision Tree Regression Results")

# Error Metrics:

print("Decision Tree: ")
print("The Accuracy Score is:\t\t", dt.score(X_test, y_test))
print("The Mean Absoulte Error[MAE] is:",
metrics.mean_absolute_error(y_test, pred_dt))
print("The Mean Squared Error[MSE] is:\t",
metrics.mean_squared_error(y_test, pred_dt))

# Random Forest Regression

# Using elbow method to determine the best value for "n_estimators"



error = []
k = np.linspace(10,300,30)
for i in k:

    rf_elbow = RandomForestRegressor(n_estimators= int(i))

    rf_elbow.fit(X_train,y_train)

    pred_elbow = rf_elbow.predict(X_test)

    mse = metrics.mean_squared_error(y_test, pred_elbow)

    error.append(mse)

plt.plot(k, error)

# Getting the Optimum value for ntree parameter:

for i in range(0,len(k)):

    if(error[i]== np.array(error).min()):

        opt = k[i]



# Creating Random Forest Regression Model object:

rf = RandomForestRegressor(n_estimators= int(opt))
```

```
# Fitting the training data:

rf.fit(X_train,y_train)

# Predicitng the values of test dataset:

pred_rf = rf.predict(X_test)

# Plot of Predicted values v/s Observed Values:

plt.figure(figsize=(12,6))
plt.scatter(x=y_test, y=pred_rf, color="orange", s=50, edgecolor="black" )
plt.plot(y_test, y_test, color="green")
plt.xlabel("Observed Values")
plt.ylabel("Predicted Values")
plt.title("Random Forest Regression Results")

# Error Metrics:

print("Random Forest Regression: ")
print("The Accuracy Score is:\t\t", rf.score(X_test, y_test))
print("The Mean Absoulte Error[MAE] is:",
metrics.mean_absolute_error(y_test, pred_rf))
print("The Mean Squared Error[MSE] is:\t",
metrics.mean_squared_error(y_test, pred_rf))



## Results:

# Accuracy score and MSE(Mean Squared Error) of Random Forest Regression
is the best out of all the tested models, so Random Forest Regression
model is selected model for Prediction of Bike Rental Count.
```

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical*
Learning. Vol. 6. Springer.
Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer Science & Business Media.