Bike Rental Prediction

Nivedha Radhakrishnan

16 July 2020

Contents

Chapter 1
Introduction
1.1 Problem Statement

The objective of building this predictive model is to predict the daily count of rental bike users based on the environment and seasonal factors.

1.2 Data

It is crucial to understand the data before processing it. We conduct Exploratory Data Analysis (EDA) to determine whether a predictive model is a viable analytical tool for the given business problem. Performing EDA reveals trends, patterns, and relationships that are not readily apparent. Understanding the relationships between different key variables eventually leads to the selection of an appropriate predictive model.

Dataset details:

The Table 1.1 shows the details of the various factors that influence the count of bike rentals

Columns	Description
instant	Record index
dteday	*Date*
season	Season (1:springer, 2:summer, 3:fall, 4:winter)
yr	*Year (0: 2011, 1:2012)*
mnth	*Month (1 to 12)*
hr	Hour (0 to 23)
holiday	weather day is holiday or not (extracted fromHoliday Schedule)
weekday	Day of the week
workingday	If day is neither weekend nor holiday is 1, otherwise is 0.

Columns	Description
weathersit	*(extracted fromFreemeteo)* *1: Clear, Few clouds, Partly cloudy, Partly cloudy* *2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist* *3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds* *4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog*
temp	*Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min),* *t_min=-8, t_max=+39 (only in hourly scale)*
atemp	*Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_max-* *t_min), t_min=-16, t_max=+50 (only in hourly scale)*
hum	*Normalized humidity. The values are divided to 100 (max)*
windspeed	*Normalized wind speed. The values are divided to 67 (max)*
casual	*count of casual users*
registered	*count of registered users*
cnt	count of total rental bikes including both casual and registered

Table 1.1 Dataset description table

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp
1	1	40544	1	0	1	0	6	0	2	0.344167
2	2	40545	1	0	1	0	0	0	2	0.363478
3	3	40546	1	0	1	0	1	1	1	0.196364
4	4	40547	1	0	1	0	2	1	1	0.200000
5	5	40548	1	0	1	0	3	1	1	0.226957
6	6	40549	1	0	1	0	4	1	1	0.204348

	atemp	hum	windspeed	casual	registered	cnt
1	0.363625	0.805833	0.1604460	331	654	985
2	0.353739	0.696087	0.2485390	131	670	801
3	0.189405	0.437273	0.2483090	120	1229	1349
4	0.212122	0.590435	0.1602960	108	1454	1562
5	0.229270	0.436957	0.1869000	82	1518	1600
6	0.233209	0.518261	0.0895652	88	1518	1606

Fig 1.1 Sample dataset shows the first 6 rows and all columns

```
'data.frame':   731 obs. of  16 variables:
 $ instant   : int  1 2 3 4 5 6 7 8 9 10 ...
 $ dteday    : int  40544 40545 40546 40547 40548 40549 40550 40551 40552 40553
 $ 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 ...
```

Fig 1.2 Summary of data after changing data type

All the attributes given are numeric, however, we split them into categorical and numeric variables based on logic. **Nominal** variables take qualitative values representing different categories, and there is no intrinsic ordering of these categories. These variables are considered as **categorical variables** and the datatype is modified accordingly. '**cnt**' is the target variable and it is **continuous.**

Chapter 2
Methodology
2.1 Pre Processing

Pre-processing refers to exploring the data, cleaning the data, and visualizing the data through summary statistics and graphical representation. This is done with the help of **Exploratory Data Analysis.**

From the data we obtain that the target/dependent variable is continuous, hence we deal with the model using regression. Regression analysis requires the data to be **normally distributed**. We use histograms, scatter plots, and bar plots to visualize the distribution of the variables. In Fig 2.1, Fig 2.2, and Fig 2.3 we see the distribution plot of all the variables. The numeric variables are plotted using histogram and scatter plot while the categorical data is plotted using barplot. EDA includes multiple stages of processing based on the requirement of the data. Let us explore each stage in detail in further sections.

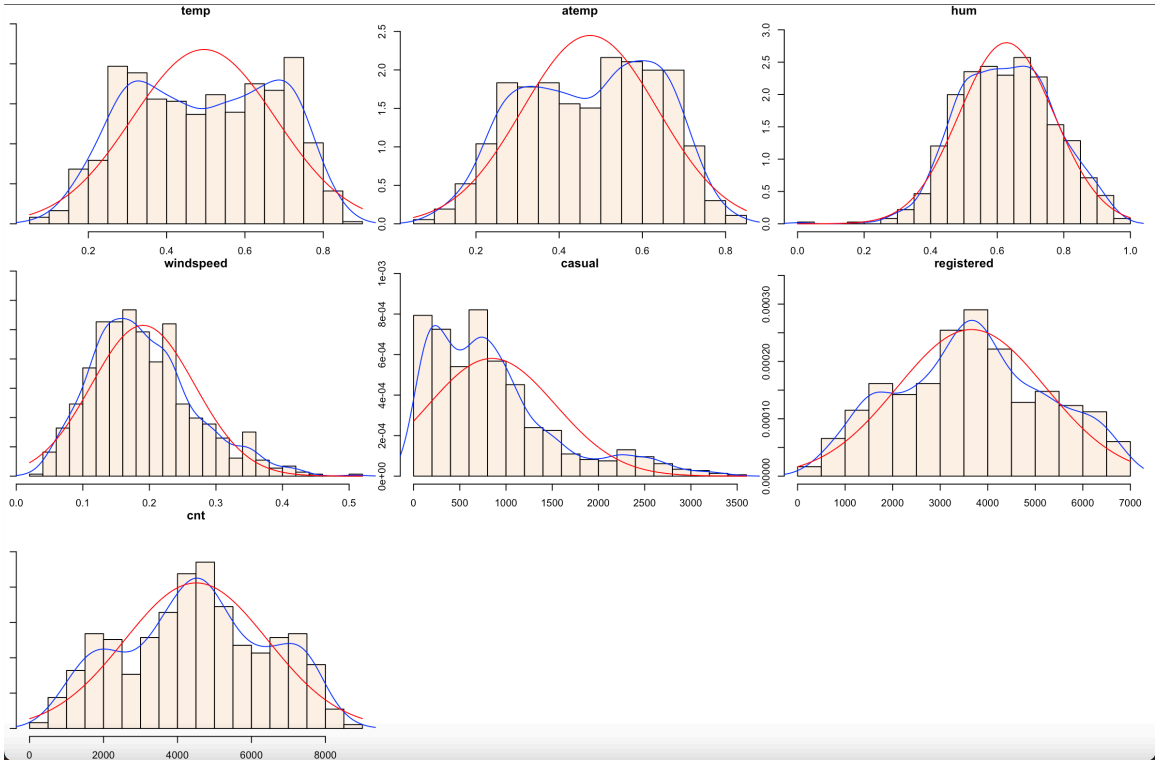

Fig 2.1: Histogram of numerical variables showing the distribution

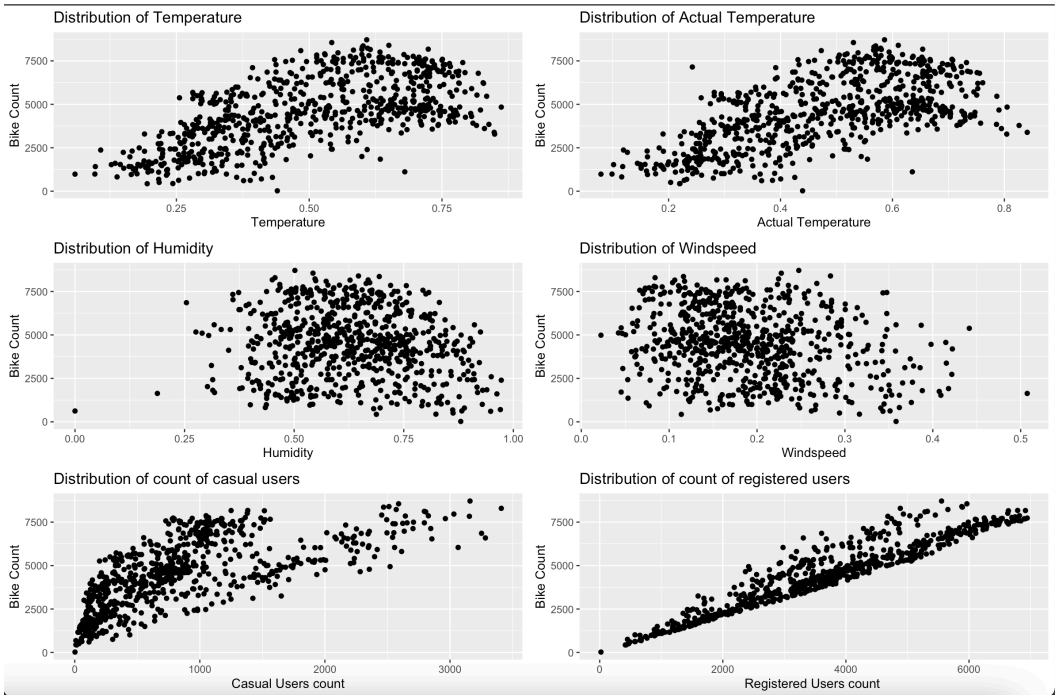

Fig 2.2: Scatter plot of numerical variables showing the distribution with respect to cnt variable

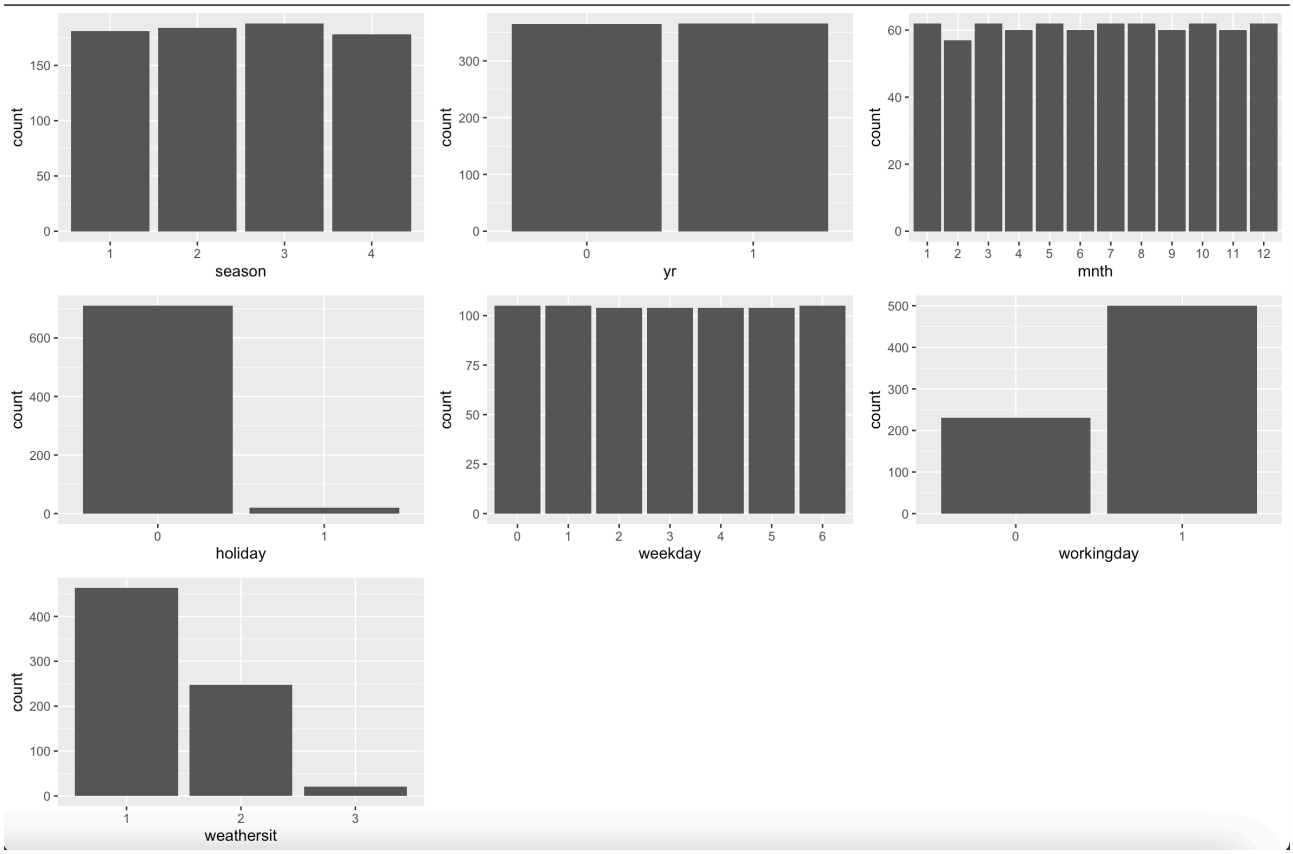

Fig 2.3: Barplot of numerical variables showing the distribution

2.1.1 Missing Value Analysis

No missing values are present in the data and hence this step can be skipped.

	NA_count	NA_Percent
season	0	0
yr	0	0
mnth	0	0
holiday	0	0
weekday	0	0
workingday	0	0
weathersit	0	0
temp	0	0
atemp	0	0
hum	0	0
windspeed	0	0
casual	0	0
registered	0	0
cnt	0	0

Figure 2.4 : Column wise missing value count showing absence of missing data

2.1.2 Outlier Analysis

An outlier is a value that deviates significantly from the rest of the values of that column. They can be just an exception or caused by measurement or execution errors. The analysis of outlier data is referred to as outlier analysis or outlier mining.

On observing the plot Fig 2.1, variable **casual** is found to be **skewed** towards the left. The skewness in the distribution shows the presence of outliers and extreme values in the data.

Boxplot (Fig 2.5) is used to visualize the outliers present in each variable. These outliers are then removed using **Tukey's method.** (By using quartiles). (Fig 2.6) shows the boxplots after removing the outliers.

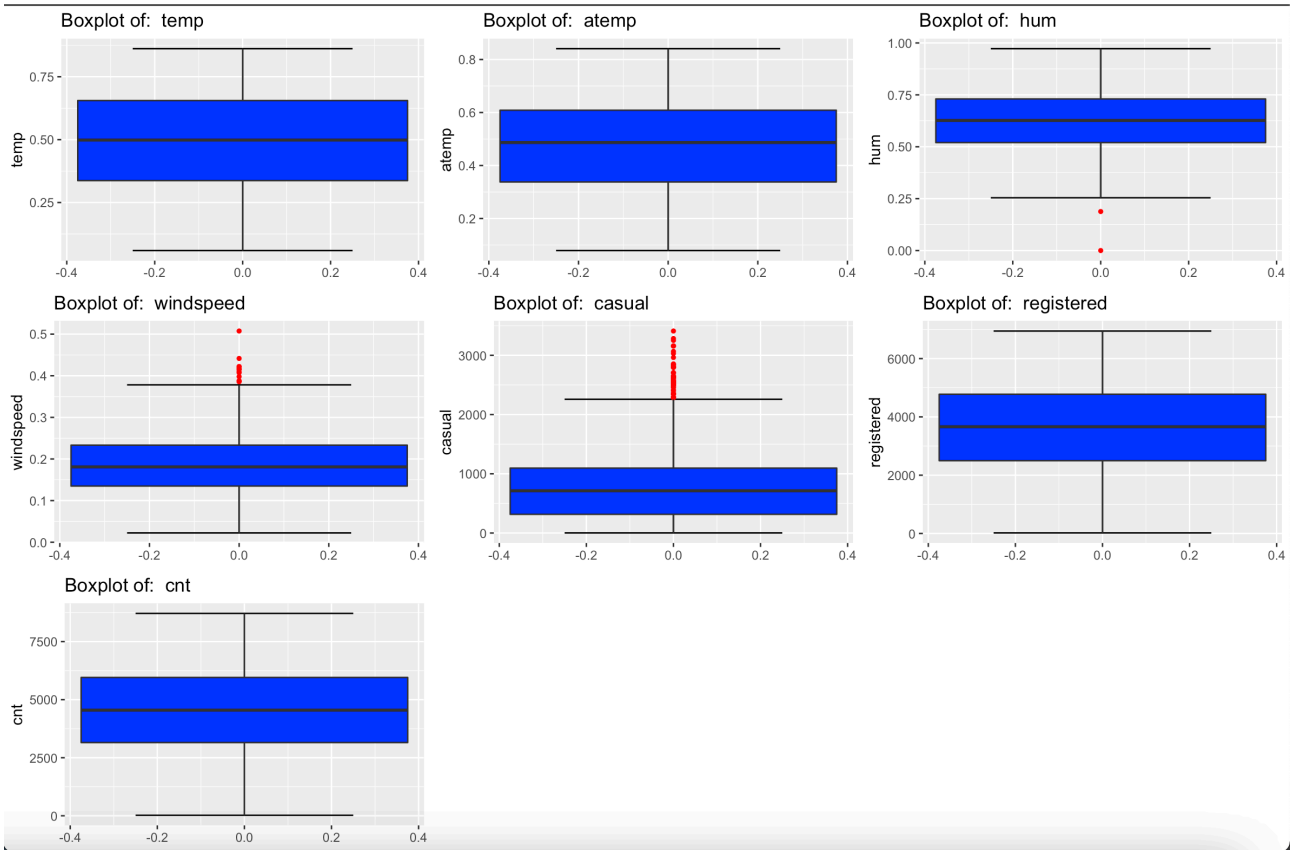

Fig 2.5: Boxplot showing the presence of outliers

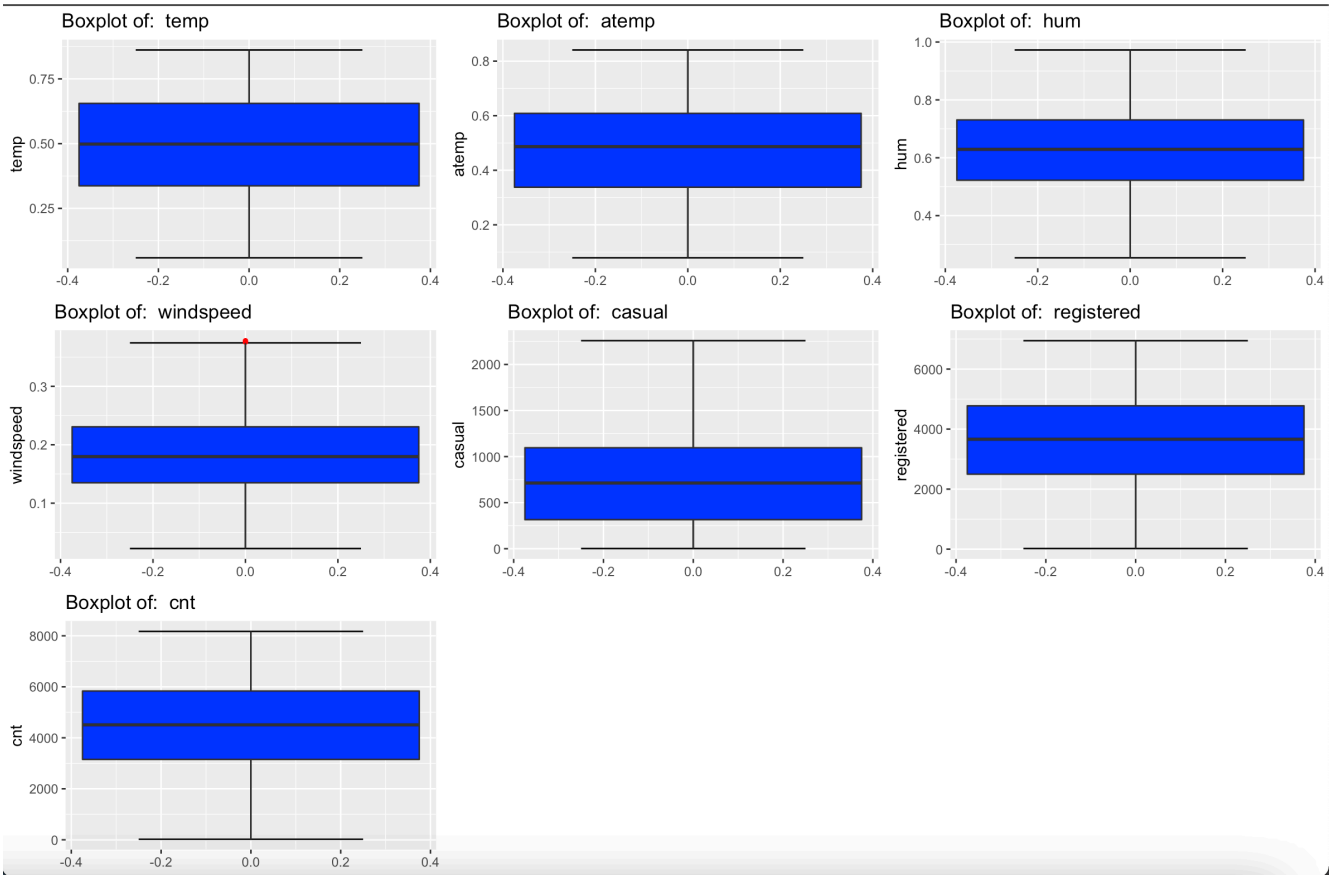

Fig 2.6: Boxplot after outlier analysis

2.1.3 Feature Selection

Before building the model we need to assess the significance of each predictor value and the correlation between them. If multicollinearity is present, it has to be rectified. The methods we are using are as follows:

2.1.3.1 Correlation matrix and correlation plot

We use **a correlation matrix, correlation plot** (Fig 2.7) and **heat map** (Fig 2.8 - can be viewed in appendix) to find the correlation between numerical variables. From Figure 2.7 we derive that **temp** and atemp variables have high correlation and **registered** and cnt variables have a high correlation (>0.8) hence one of the predictors for each pair can be removed.

```
correlation_matrix = cor(bike_rental_data[,numeric_variable_set])
corrplot(correlation_matrix,method = "number",type = 'lower')
```

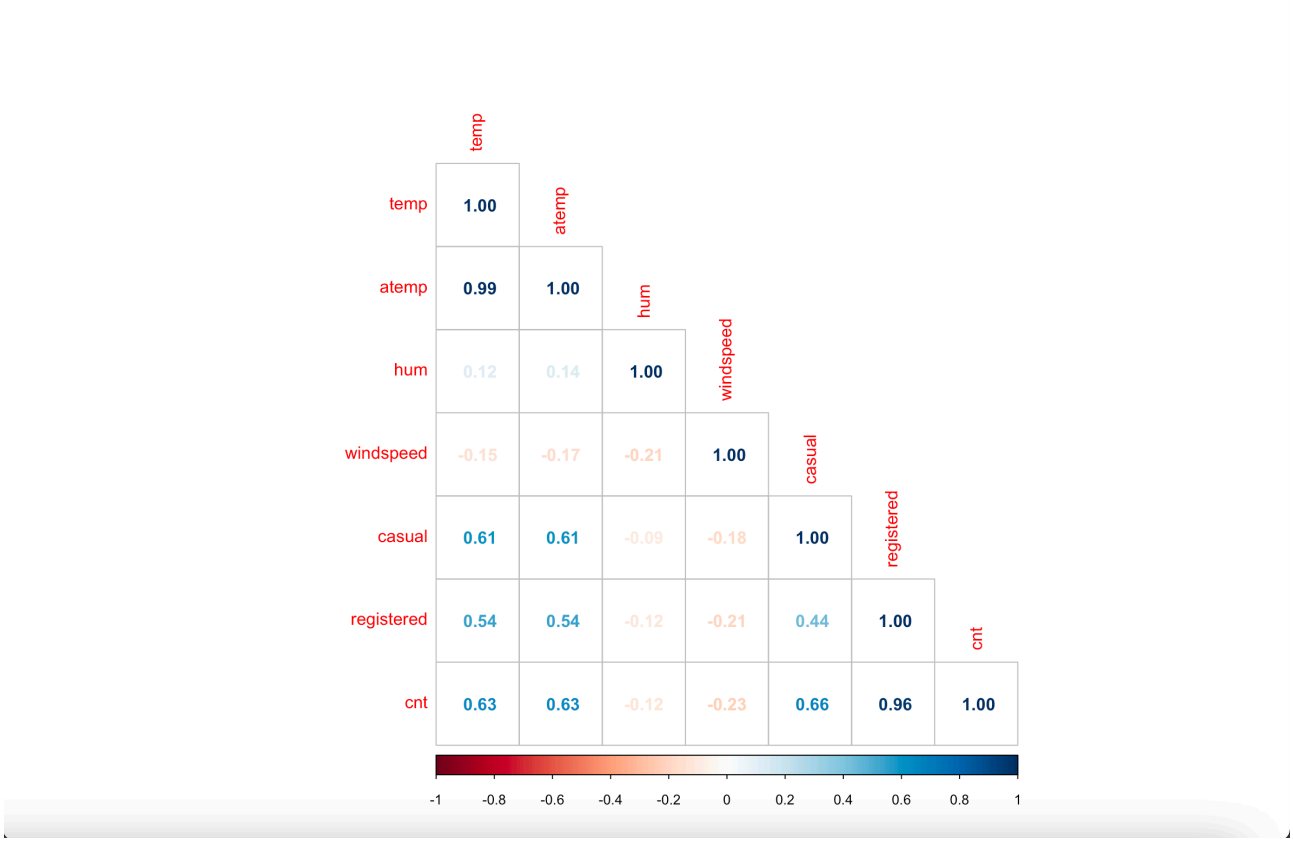

Figure 2.7 Correlation plot for numerical variables

2.1.3.2 VIF

To detect multicollinearity, we use VIF for numeric variables. Variance inflation factor (VIF) is the measure of multicollinearity in a set of multiple regression variables. Mathematically, the VIF for a regression model variable is the ratio of the overall model variance to the variance of a model that includes only that single independent variable. A high VIF indicates that the associated independent variable is highly collinear with the other variables in the model.

Fig 2.9 (can be viewed in the appendix) shows that the VIF of **casual, registered, and cnt** are **infinite**. This depicts the presence of **multicollinearity** in the data. We know that cnt is the sum of casual and registered. So **registered** variable can be removed as it is redundant. On further analysis of VIF, we find that there is no multicollinearity present in the data after removing registered variable. Fig 2.9 (can be viewed in the appendix) shows the new VIF.

```
vif(bike_rental_data[,numeric_variable_set])
```

2.1.3.3 ANOVA

To detect multicollinearity, we use VIF for numeric variables. For categorical independent variables, we use Analysis of variance (ANOVA).ANOVA is a collection of statistical models and their associated estimation procedures (such as the "variation" among and between groups) used to analyze the differences among means. From Fig 2.10 (can be viewed in the appendix) we derive that the variables **weekday** has p-value more than the level of significance (ie.,0.05) and hence can be omitted.

```
for(i in categorical_variable_set){
 print(i)
 aov_summary = summary(aov(cnt~bike_rental_data[,i],data =
       bike_rental_data))
 print(aov_summary)
}
```

2.1.3.4 Dimentionality Reduction

From the observations from correlation matrix, correlation plot, VIF, and ANOVA we conclude that the columns **atemp, registered, weekday** are **not significant.**

```
bike_rental_data = subset(bike_rental_data, select = -
(which(names(bike_rental_data) %in%
c("registered","atemp","weekday"))))
```

2.2 Modeling

2.2.1 Model Selection

The variables can be nominal, ordinal, interval, ratio, or continuous. If the dependent variable, in our case *cnt* is Nominal the only predictive analysis that we can perform is Classification, and if the dependent variable is Interval or Ratio the normal method is to do a Regression analysis or classification after binning. If the dependent variable we are dealing with is *Ordinal*, for which both classification and regression can be done because even though the Quality variable has categories, these categories have an order associated with them, which is ranks. But our dependent variable is continuous, so we use **Regression Analysis**.

2.2.2 Sampling

We split the data into 4:1 ratio ie 80 % data would be used to train and the remaining 25% data will be used to test.

```
set.seed(123)
train_index =
sample(1:nrow(bike_rental_data),0.8*nrow(bike_rental_data))
train = bike_rental_data[train_index,]
test = bike_rental_data[-train_index,]
```

2.2.3 Multiple Linear Regression

```
regressor_mlr = lm(formula = cnt ~ .,data = train)
y_pred_mlr = predict(regressor_mlr, newdata = test)
summary(regressor_mlr)
```

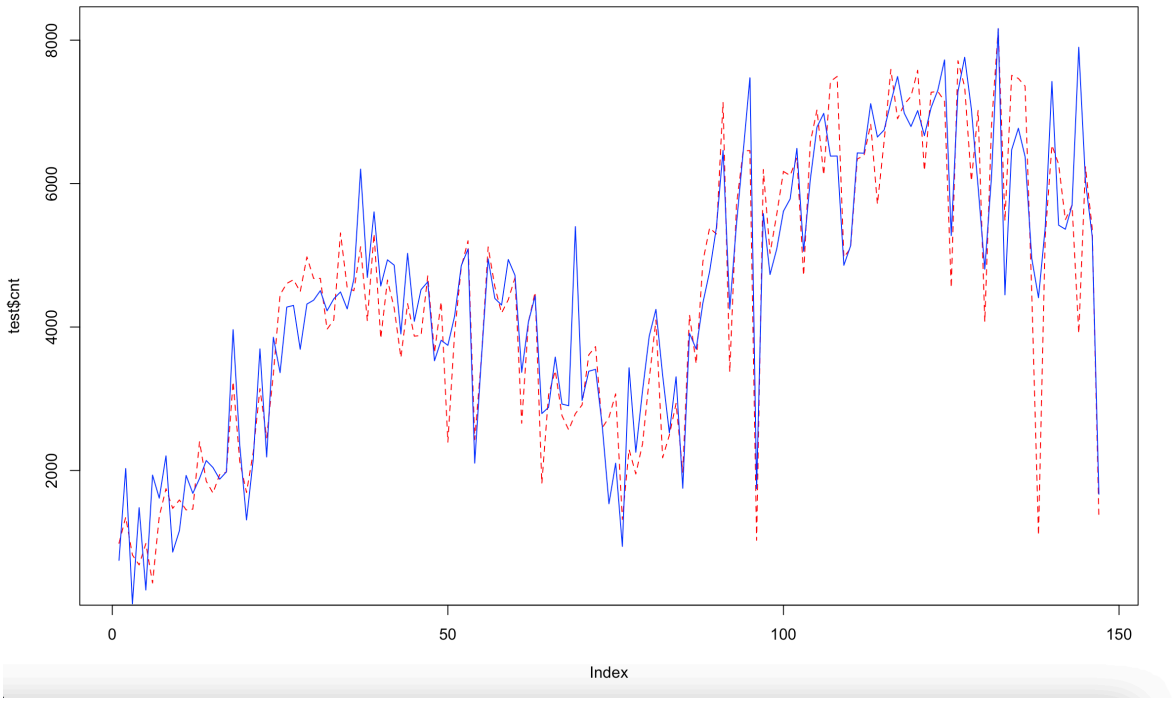

Figure 2.8 Multiple Linear Regression Actual vs Predicted plot

2.2.4 Decision Tree

```
regressor_dt = rpart(cnt ~., data = train, method = "anova")
y_pred_dt = predict(regressor_dt, newdata = test)
summary(regressor_dt)
```

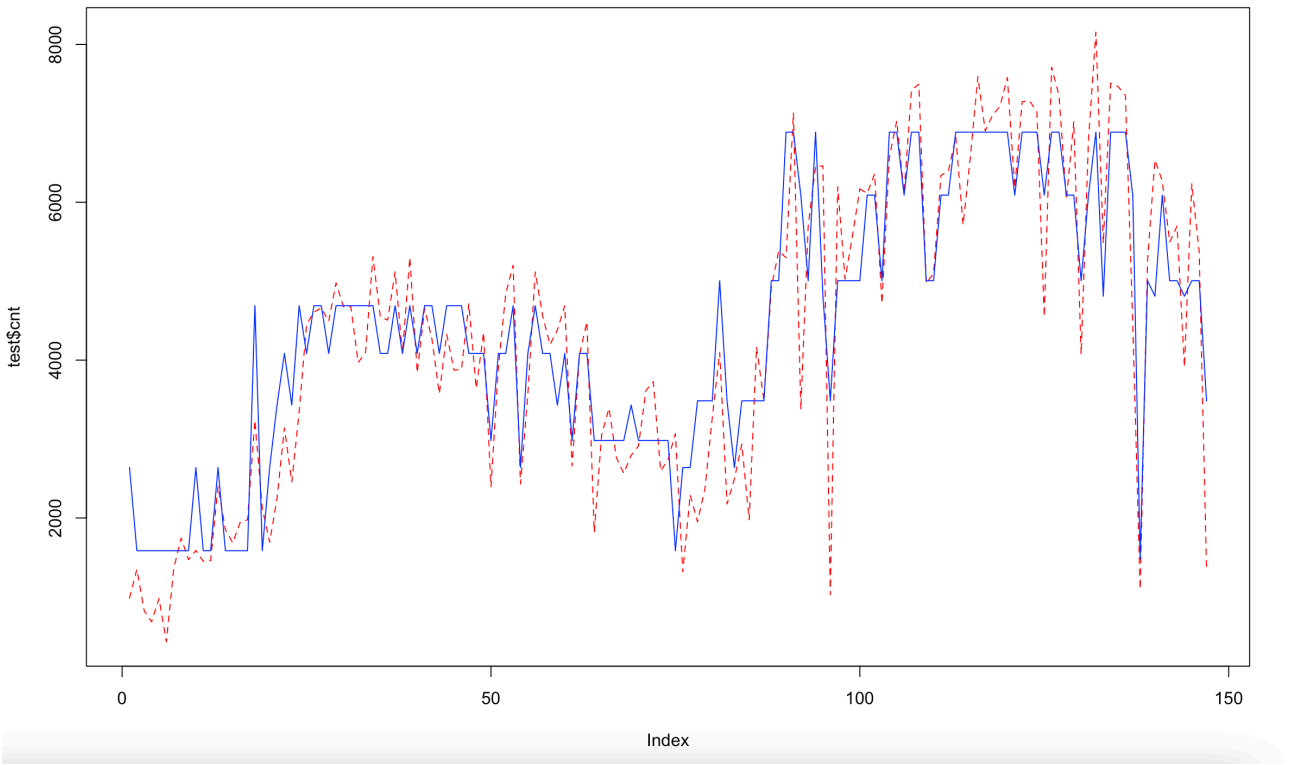

Figure 2.9 Multiple Linear Regression Actual vs Predicted plot

2.2.5 Random Forest

```
regressor_rf = randomForest(cnt~., data = train, ntrees = 500)
y_pred_rf = predict(regressor_rf,test)
summary(regressor_rf)
```

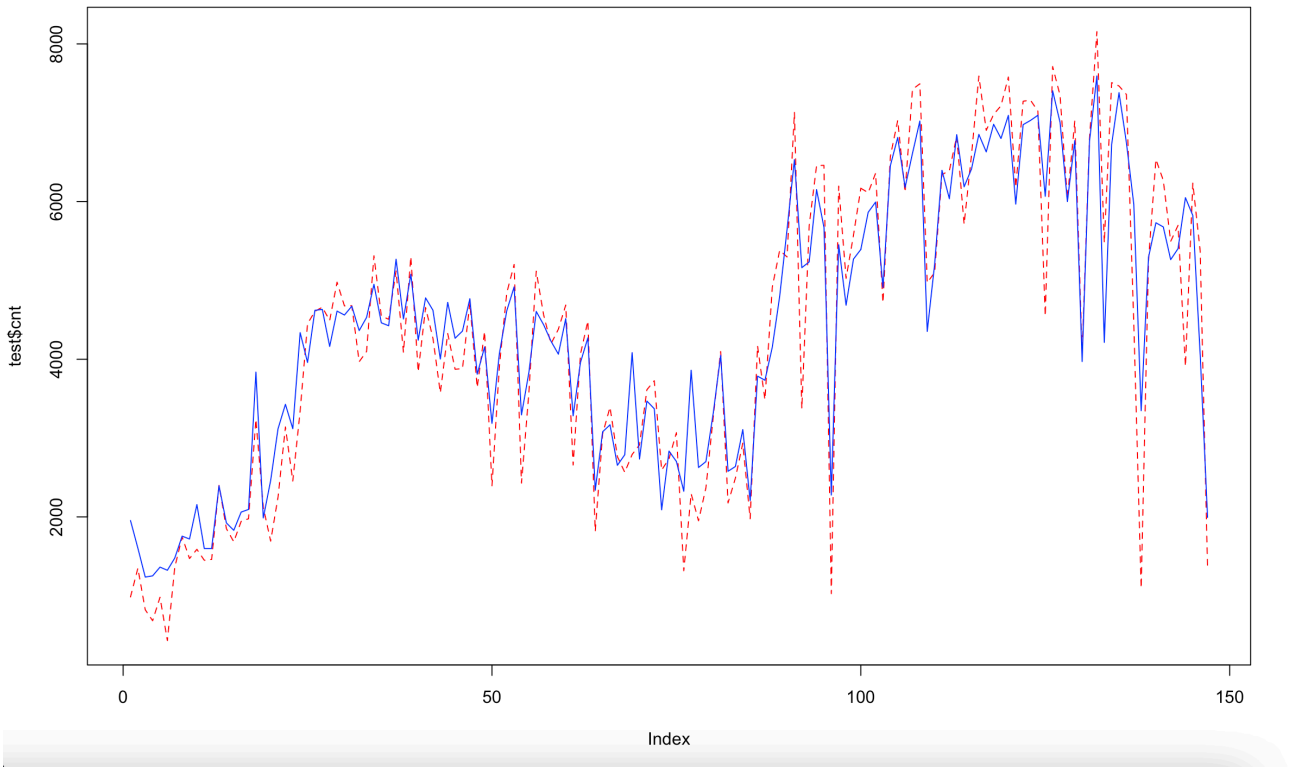

Figure 2.10 Multiple Linear Regression Actual vs Predicted plot

2.3 Model optimization

2.3.1 Hyperparameter Tuning

Hyperparameter optimization or tuning is the technique of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

2.3.1 Multiple Linear Regression

The actual vs predicted plot shows the comparison of tuned predicted values from Multiple Linear Regression and the actual values in the test dataset.

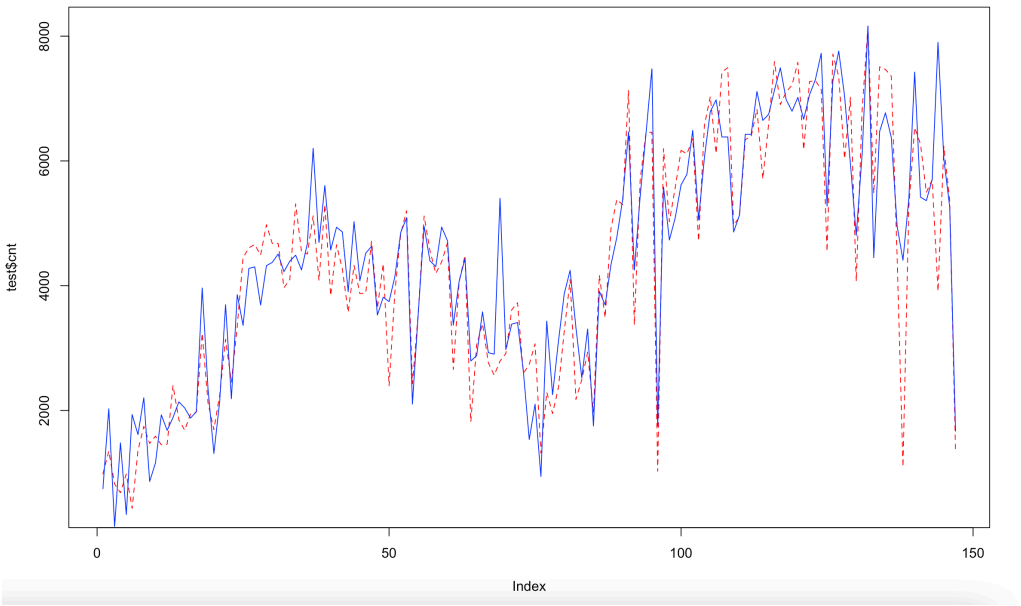

Figure 2.11 Multiple Linear Regression Actual vs tuned Predicted plot

2.3.2 Decision Tree

The actual vs predicted plot shows the comparison of tuned predicted values from Decision Tree and the actual values in the test dataset.

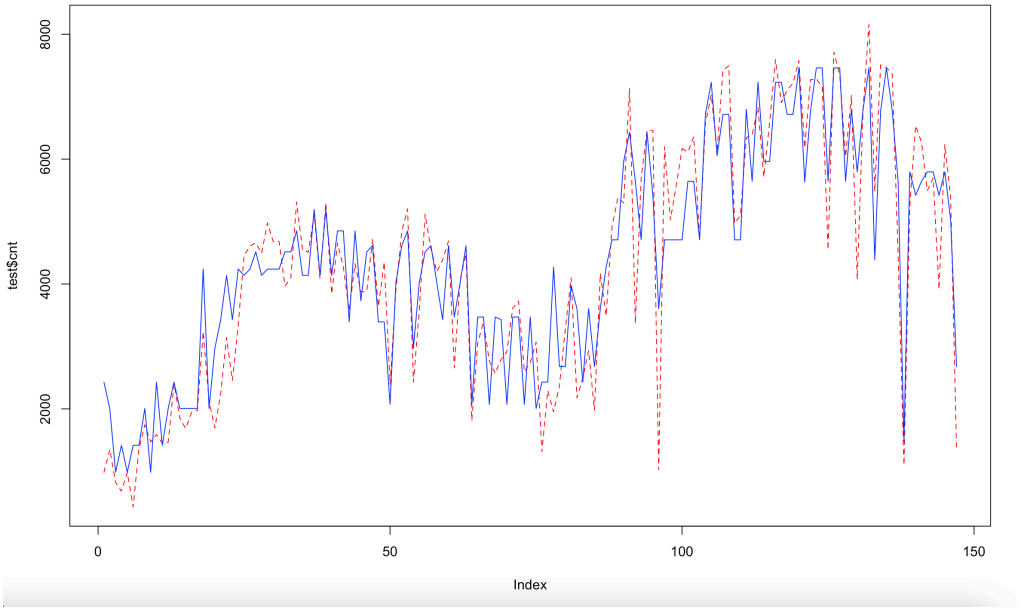

Figure 2.12 Decision tree Actual vs tuned Predicted plot

2.3.3 Random Forest

The actual vs predicted plot shows the comparison of tuned predicted values from Random Forest and the actual values in the test dataset.

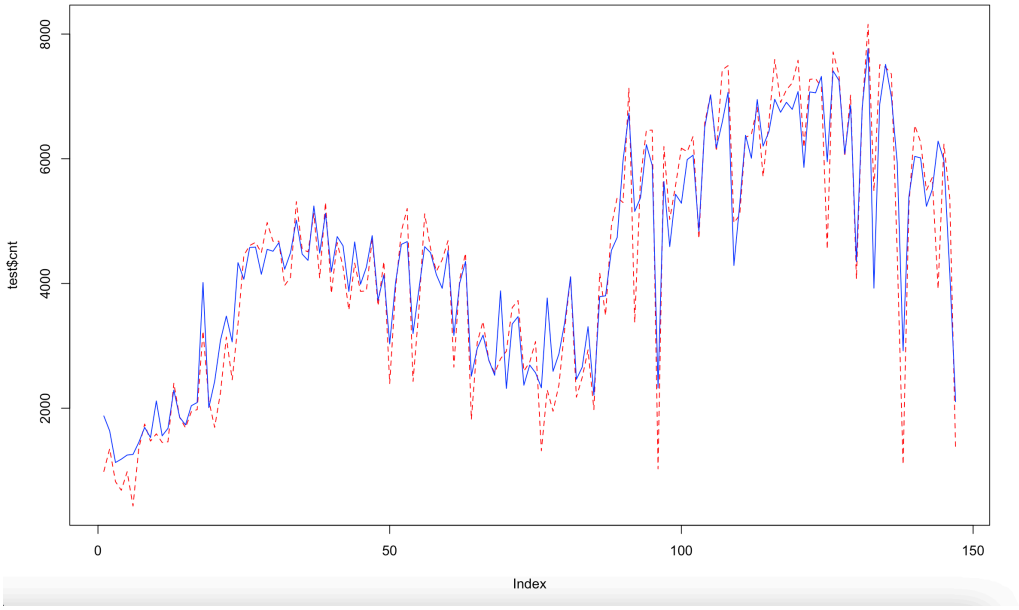

Figure 2.13 Random Forest Actual vs Predicted plot

Chapter 3
Conclusion
3.1 Model Evaluation
3.1.1 Error metrics

In the previous section we obtained the results from the following models:

1. Multiple Linear Regression
2. Decision Tree
3. Random Forest

We evaluate the models based on the MAE, RMSE, and R-squared values.

Root Mean Square Error(RMSE) is the square root of the average of squared errors. **Mean Absolute Error(MAE)** of a model refers to the mean of the absolute values of each prediction error on all instances of the test data-set. **R squared** value is the percentage of the response variable variation that is explained by a linear model.

Table 3.1, Table 3.2, Table 3.3, Table 3.4 shows the comparison of RMSE, MAE, R-Squared values.

Language	Model	Value
Python (MAPE)	Multiple linear regression	0.147304399078773
	Decision Tree	0.248677666969623
	Random Forest	0.150170163353573
	Multiple linear regression (Tuned)	0.147304399078773
	Decision Tree (Tuned)	0.269033096937692
	Random Forest (Tuned)	0.323643745648355
R	Multiple linear regression	510.7905055
	Decision Tree	594.3689043
	Random Forest	423.8855792
	Multiple linear regression (Tuned)	510.7905055
	Decision Tree (Tuned)	544.8011413
	Random Forest (Tuned)	396.1909000

Table 3.1 shows the comparison of MAE

Language	Model	Value
Python	Multiple linear regression	583.026208392622
	Decision Tree	837.69156829119
	Random Forest	562.0580174063
	Multiple linear regression (Tuned)	583.026208392622
	Decision Tree (Tuned)	984.69251606846
	Random Forest (Tuned)	984.549155446321
R	Multiple linear regression	735.8215126
	Decision Tree	782.6174370
	Random Forest	587.0125339
	Multiple linear regression (Tuned)	735.8215126
	Decision Tree (Tuned)	716.9992125
	Random Forest (Tuned)	561.6378007

Table 3.2 shows the comparison of RMSE

Language	Model	Value
Python	Multiple linear regression	0.914564412081745
	Decision Tree	0.823627469872803
	Random Forest	0.920599186665255
	Multiple linear regression (Tuned)	0.914564412081745
	Decision Tree (Tuned)	0.756295285253473
	Random Forest (Tuned)	0.756366241649873
R	Multiple linear regression	0.8598805

Language	Model	Value
	Decision Tree	0.8460872
	Random Forest	0.9204742
	Multiple linear regression (Tuned)	0.8598805
	Decision Tree (Tuned)	0.8685536
	Random Forest (Tuned)	0.9241534

Table 3.3 shows the comparison of R squared value

3.1.1 K-Fold Cross Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation. Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data.

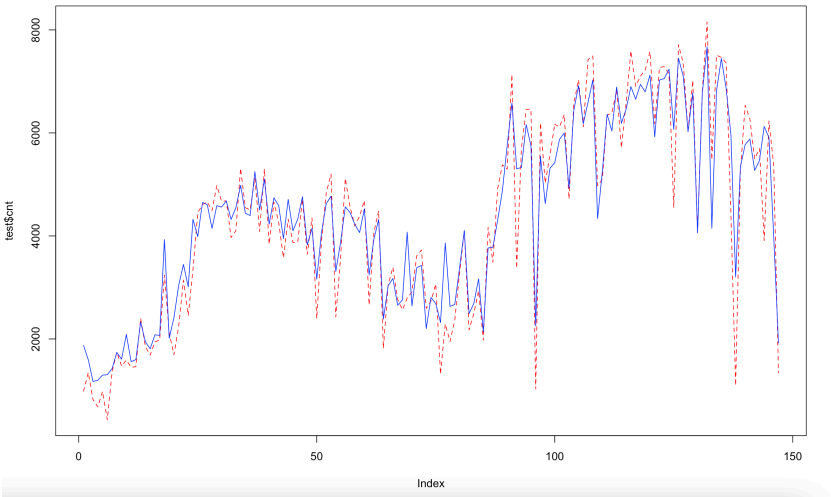

Fig 3.1 K-Fold Cross Validation Actual vs Predicted plot

3.2 Model Selection

After comparing MAE, RMSE and R squared values, Both **the Random forest** model and **Multiple Linear Regression** can be chosen as the best fit as Random Forest has the minimum MAE, RMSE value and maximum R squared value and the K-Fold cross-validation score is higher for Multiple Linear Regression.

3.3 Business Solution

The dendrogram (Fig 3.1) obtained from decision tree training shows the significance of each predictor.

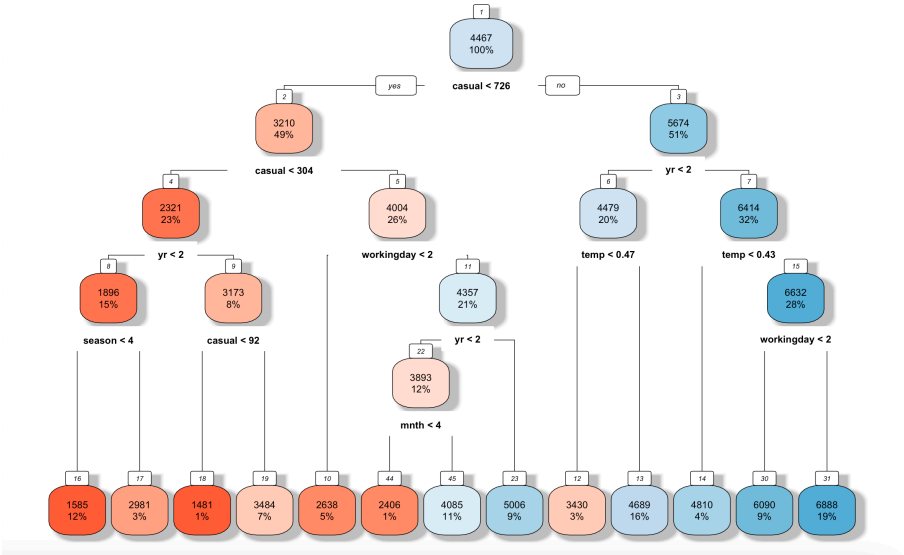

Fig 3.2 Dendogram

From the dendrogram that there is an increase in total user count at a normalised temperature more than 0.47 (ie., >14.09 Celsius)

From Fig 2.3, we infer that the count of users is higher on working days and the sky is clear without clouds.

Appendix A - Extra Figures

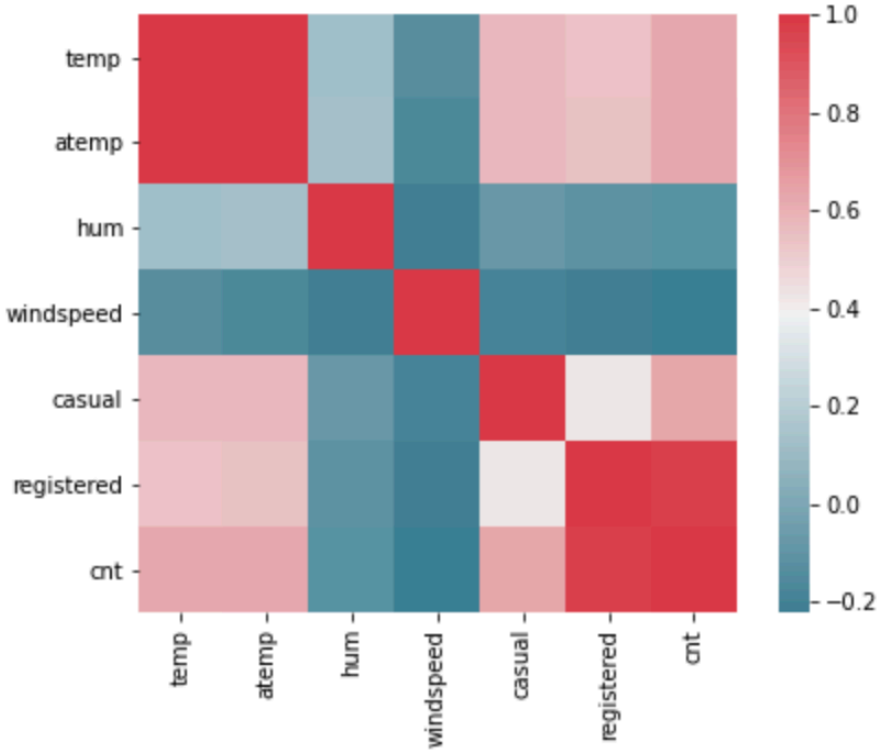

Fig 2.8 Heat Map for correlation matrix

	Variables	VIF
1	temp	63.371903
2	atemp	64.183811
3	hum	1.193014
4	windspeed	1.168540
5	casual	Inf
6	registered	Inf
7	cnt	Inf

	Variables	VIF
1	temp	2.037897
2	hum	1.181754
3	windspeed	1.133578
4	casual	2.010734
5	cnt	2.224667

Fig 2.9 VIF before and after removing registered column

```
[1] "season"
                     Df    Sum Sq   Mean Sq F value Pr(>F)
bike_rental_data[, i]   3 8.955e+08 298493887   132.5 <2e-16 ***
Residuals             727 1.637e+09   2252301
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "yr"
                     Df    Sum Sq   Mean Sq F value Pr(>F)
bike_rental_data[, i]   1 8.002e+08 800192441   336.7 <2e-16 ***
Residuals             729 1.733e+09   2376834
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "mnth"
                     Df    Sum Sq  Mean Sq F value Pr(>F)
bike_rental_data[, i]  11 1.000e+09 90916753   42.65 <2e-16 ***
Residuals             719 1.533e+09  2131878
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "holiday"
                     Df    Sum Sq  Mean Sq F value Pr(>F)
bike_rental_data[, i]   1 1.689e+07 16888694   4.893 0.0273 *
Residuals             729 2.516e+09  3451324
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "weekday"
                     Df    Sum Sq Mean Sq F value Pr(>F)
bike_rental_data[, i]   6 3.071e+07 5119046   1.481  0.182
Residuals             724 2.502e+09 3456063
[1] "workingday"
                     Df    Sum Sq  Mean Sq F value  Pr(>F)
bike_rental_data[, i]   1 3.317e+07 33171539   9.674 0.00194 **
Residuals             729 2.500e+09  3428989
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
[1] "weathersit"
                     Df    Sum Sq   Mean Sq F value Pr(>F)
bike_rental_data[, i]   2 2.578e+08 128894616   41.24 <2e-16 ***
Residuals             728 2.275e+09   3125158
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Fig 2.10 ANOVA summary

Appendix B - R Code

The complete R code is attached below.

```
#########################Bike Rental Count Prediction R
Code#############################
#Cleaning the environment
rm(list = ls())

#Setting the working directory
setwd("/Users/nivedharakigmail.com/Desktop/Edwisor/Project 2")

#Load libraries
libraries =
c("psych","tidyverse","ggpubr","DMwR2","corrplot","usdm","caret","rpart","rpart
.plot","randomForest")
lapply(X = libraries,require, character.only = TRUE)
rm(libraries)

#Importing the csv file
bike_rental_data= read.csv(file = "BikeRentalDataset.csv", header = T)

#########################Exploratory data
analysis#################################
#Observing sample data
head(bike_rental_data)

#Observing structure of the data
str(bike_rental_data)

#Dropping instant and dteday as they are non significant/redundent for the
prediction
bike_rental_data = subset(bike_rental_data, select = -
(which(names(bike_rental_data) %in% c("instant","dteday"))))

##Sorting the variables into numerical and categorical
categorical_variable_set =
c("season","yr","mnth","holiday","weekday","workingday","weathersit")
numeric_variable_set =
```

```
c("temp","atemp","hum","windspeed","casual","registered","cnt")

#Parsing the datatype of categorical variables and assigning levels
for(i in categorical_variable_set){
  bike_rental_data[,i] = as.factor(bike_rental_data[,i])
}

#verifying the structure of the categorical variables
str(bike_rental_data)

#Defining function to store categorical data and numeric data
fetch_data = function(data_set,typename){
  #Get the data with only numeric columns
  if (typename == "numeric"){
    numeric_index = sapply(data_set, is.numeric)
    numeric_data = data_set[,numeric_index]
  }
  #Get the data with only factor columns
  else{
    numeric_index = sapply(data_set, is.numeric)
    factor_data = bike_rental_data[,!numeric_index]
  }
}

#Calling function fetch_data to store numeric and cateogrical data separately
numeric_data = fetch_data(bike_rental_data,"numeric")
factor_data = fetch_data(bike_rental_data,"factor")

############################Missing value
Analysis#################################
###-----------------------Observation-------------------------------------###
#Count of missing values in each variable
missing_values_df = data.frame(apply(bike_rental_data,2,function(x)
{sum(is.na(x))}))

#Creating a new column for missing value percentage
names(missing_values_df)[1] = "NA_count"
missing_values_df$NA_Percent = (missing_values_df$NA_count/
```

```
nrow(bike_rental_data))*100

#Sorting missing values in decreasing order
missing_values_df = missing_values_df[order(-missing_values_df$NA_Percent),]
missing_values_df

##No missing values found in any of the variable, hence imputaion is not required

##########################Analysis of data distribution using
histogram#########

#Multiple histograms for numerical predictors
bike_rental_data_plot_hist = numeric_data[1:7]
multi.hist(bike_rental_data_plot_hist,dcol= c("blue","red"),dlty=c("solid",
"solid"),bcol="linen")
#The predictor "casual" is skewed to the left.
#The rest of the predictors are normally distributed.

#Multiple scatterplot for numerical predictors
scat1 = ggplot(data = bike_rental_data_plot_hist, aes(x =temp, y = cnt)) +
ggtitle("Distribution of Temperature") + geom_point() + xlab("Temperature") +
ylab("Bike Count")
scat2 = ggplot(data = bike_rental_data_plot_hist, aes(x =atemp, y = cnt)) +
ggtitle("Distribution of Actual Temperature") + geom_point() + xlab("Actual
Temperature") + ylab("Bike Count")
scat3 = ggplot(data = bike_rental_data_plot_hist, aes(x =hum, y = cnt)) +
ggtitle("Distribution of Humidity") + geom_point() + xlab("Humidity") +
ylab("Bike Count")
scat4 = ggplot(data = bike_rental_data_plot_hist, aes(x =windspeed, y = cnt)) +
ggtitle("Distribution of Windspeed") + geom_point() + xlab("Windspeed") +
ylab("Bike Count")
scat5 = ggplot(data = bike_rental_data_plot_hist, aes(x =casual, y = cnt)) +
ggtitle("Distribution of count of casual users") + geom_point() + xlab("Casual
Users count") + ylab("Bike Count")
scat6 = ggplot(data = bike_rental_data_plot_hist, aes(x =registered, y = cnt)) +
ggtitle("Distribution of count of registered users") + geom_point() +
xlab("Registered Users count ") + ylab("Bike Count")
gridExtra::grid.arrange(scat1,scat2,scat3,scat4,scat5,scat6,ncol=2)
```

```
#The predictor "casual" is skewed to the left.
#The rest of the predictors are normally distributed.

#Multiple barplot
#Bar plot for categorically predictors
bike_rental_data_plot_bar = factor_data[1:7]
gplot1 = ggplot(bike_rental_data_plot_bar, aes(x = season ) )+ geom_bar()
gplot2 = ggplot(bike_rental_data_plot_bar, aes(x = yr ) )+ geom_bar()
gplot3 = ggplot(bike_rental_data_plot_bar, aes(x = mnth ) )+ geom_bar()
gplot4 = ggplot(bike_rental_data_plot_bar, aes(x = holiday ) )+ geom_bar()
gplot5 = ggplot(bike_rental_data_plot_bar, aes(x = weekday ) )+ geom_bar()
gplot6 = ggplot(bike_rental_data_plot_bar, aes(x = workingday ) )+ geom_bar()
gplot7 = ggplot(bike_rental_data_plot_bar, aes(x = weathersit ) )+ geom_bar()
ggarrange(gplot1,gplot2,gplot3,gplot4,gplot5,gplot6,gplot7)
#The count of bikes hired are evenly distributed across different level of the
predictor except holiday,workingday and weathersit

############################Outlier
Analysis#########################################
#Outlier boxplot
for(i in 1:ncol(numeric_data)) {
  assign(paste0("box_plot",i), ggplot(data = bike_rental_data, aes_string(y =
numeric_data[,i])) +
         stat_boxplot(geom = "errorbar", width = 0.5) +
         geom_boxplot(outlier.colour = "red", fill = "blue", outlier.size = 1) +
         labs(y = colnames(numeric_data[i])) +
         ggtitle(paste("Boxplot of: ",colnames(numeric_data[i]))))
}

#Arrange the plots in grids
gridExtra::grid.arrange(box_plot1,box_plot2,box_plot3,box_plot4,box_plot5,
                        box_plot6,box_plot7,ncol=3)
#Outliers are found in the predictors hum,windspeed and casual

#Replacing all outliers with NA
for(i in numeric_variable_set){
  val1 = bike_rental_data[,i][bike_rental_data[,i] %in%
boxplot.stats(bike_rental_data[,i])$out]
```

```
  print(paste(i,length(val1)))
  bike_rental_data[,i][bike_rental_data[,i] %in% val1] = NA
}

#Checking for missing values
sum(is.na(bike_rental_data))

#Imputing the outliers using KNNimputation
bike_rental_data = knnImputation(bike_rental_data, k = 10)
# bike_rental_data$casual[149]

#Checking for missing values
sum(is.na(bike_rental_data))

#Parsing datatype of casual to int
bike_rental_data["casual"] = as.integer(bike_rental_data["casual"])
#Observing structure of the data
str(bike_rental_data)

#Since cnt is the total of casual and registered replacing the values of cnt with the
sum of the imputed casual registered values.
bike_rental_data$cnt <- bike_rental_data$casual + bike_rental_data$registered

# #Verifying if there are any outliers present after the imputation
# #Calling function fetch_data to store numeric and cateogrical data separately
# numeric_data = fetch_data(bike_rental_data,"numeric")
# factor_data = fetch_data(bike_rental_data,"factor")
#
# #Outlier boxplot
# for(i in 1:ncol(numeric_data)){
#   assign(paste0("box_plot",i), ggplot(data = bike_rental_data, aes_string(y =
numeric_data[,i])) +
#          stat_boxplot(geom = "errorbar", width = 0.5) +
#          geom_boxplot(outlier.colour = "red", fill = "blue", outlier.size = 1) +
#          labs(y = colnames(numeric_data[i])) +
#          ggtitle(paste("Boxplot of: ",colnames(numeric_data[i]))))
# }
#
```

```
# #Arrange the plots in grids
# gridExtra::grid.arrange(box_plot1,box_plot2,box_plot3,box_plot4,box_plot5,
#                         box_plot6,box_plot7,ncol=3)

##########################Feature
Selection#########################################
##Numerical columns
#Creating correlation matrix
correlation_matrix = cor(bike_rental_data[,numeric_variable_set])
# dev.off()
corrplot(correlation_matrix,method = "number",type = 'lower')
#The correlation coefficient of temp and atemp pair and registered and cnt pair
seems to be higher than the level of significance ie., 0.8, hence one predictor in
each pair can be omitted.

#Checking multi-collinearity
#Using VIF technique for numerical data
vif(bike_rental_data[,numeric_variable_set])
#The VIF of casual, registered and cnt is found to be infinite, which shows the
presence of multicollinearity. Hence registered column should be dropped.

#Using ANOVA technique for categorical data
for(i in categorical_variable_set){
  print(i)
  aov_summary = summary(aov(cnt~bike_rental_data[,i],data = bike_rental_data))
  print(aov_summary)
}
#The p-value of weekday is above the level of significance (ie., 0.05). weekday
column can be dropped too.

#Dimentionality reduction
bike_rental_data_all_columns = bike_rental_data
bike_rental_data = subset(bike_rental_data, select = -
(which(names(bike_rental_data) %in% c("registered","atemp","weekday"))))

#Checking multi-collinearity after dimentionality reduction
#Using VIF technique for numerical data
numeric_variable_set = c("temp","hum","windspeed","casual","cnt")
```

```
vif(bike_rental_data[,numeric_variable_set])
#Multicollinearity is not present among the predictors

############################Feature
Sampling#####################################
variable_set = c("season","yr","mnth","holiday","workingday","weathersit")
#Parsing all the columns to numeric value
for(i in variable_set){
  bike_rental_data[,i] = as.numeric(bike_rental_data[,i])
}

#verifying the structure of the variables
str(bike_rental_data)

#Separating dataset into test and train set
set.seed(123)
#Splitting the train and test set in 4:1 ratio (ie., 80% training data and 20% test
data)
train_index = sample(1:nrow(bike_rental_data), 0.8*nrow(bike_rental_data))
train = bike_rental_data[train_index,]
test = bike_rental_data[-train_index,]

############################### Model Development
###############################
###-------------------Multiple linear regression--------------------------###
# RMSE     : 735.8215126
# Rsquared : 0.8598805
# MAE      : 510.7905055
#Train the model using training data
# Fitting Multiple Linear Regression to the Training set
regressor_mlr = lm(formula = cnt ~ .,data = train)
# Predicting the Test set results
y_pred_mlr = predict(regressor_mlr, newdata = test)
#Get the summary of the model
#summary(regressor_mlr)
#Create dataframe for actual and predicted values
model_pred = data.frame("actual"=test, "model_pred"=y_pred_mlr)
#head(model_pred)
```

```
#Calcuate MAE, RMSE, R-squared for testing data
print(postResample(pred = y_pred_mlr, obs =test$cnt))
#Plot a graph for actual vs predicted values
plot(test$cnt,type="l",lty=2,col="red")
lines(y_pred_mlr,col="blue")

#Predict a sample data
predict(regressor_mlr,test[4,])
#18
#1481.066
#####***************MLR Hyperparameter tuning with
caret*********************####
# RMSE     : 735.8215126
# Rsquared : 0.8598805
# MAE      : 510.7905055
# Define the control
trControl <- trainControl(method = "cv", number = 3, search = "grid",
                          verboseIter = TRUE)

set.seed(123)
tuned_model_regression = caret::train(cnt ~ .,
                              data = train,
                              method = "lm",
                              metric = "MAE",
                              trControl = trControl)

predictions_tuned_model_regression = predict(tuned_model_regression,test)

#Calcuate MAE, RMSE, R-squared for testing data
print(postResample(pred = predictions_tuned_model_regression, obs = test$cnt))

#Plot a graph for actual vs predicted values
plot(test$cnt,type="l",lty=2,col="red")
lines(predictions_tuned_model_regression,col="blue")

#Predict a sample data
predict(tuned_model_regression,test[4,])
```

```
#18
#1481.066

###-------------------Decision Tree----------------------------------------###
# RMSE     : 782.6174370
# Rsquared : 0.8460872
# MAE      : 594.3689043
#Build decision tree using rpart
regressor_dt = rpart(cnt ~., data = train, method = "anova")

#Predict the test cases
y_pred_dt = predict(regressor_dt, newdata = test)

#Create data frame for actual and predicted values
model_pred = cbind(model_pred,y_pred_dt)
#head(model_pred)

#Get the summary of the model
#summary(regressor_dt)

#Calcuate MAE, RMSE, R-squared for testing data
print(postResample(pred = y_pred_dt, obs = test$cnt))

#Plot a graph for actual vs predicted values
plot(test$cnt,type="l",lty=2,col="red")
lines(y_pred_dt,col="blue")

# Visualize the decision tree with rpart.plot
rpart.plot(regressor_dt, box.palette="RdBu", shadow.col="gray", nn=TRUE)

#Predict a sample data
predict(regressor_dt,test[4,])
#18
#1585.271

#####**********DT Hyperparameter tuning with
caret**********####
# RMSE     : 716.9992125
```

```
# Rsquared : 0.8685536
# MAE      : 544.8011413
rpart_grid = expand.grid(cp = c(0.01,0.001,0.02,0.3,0.3,0.3,0.001,0.3,0.3,0.001))

# Define the control
trControl <- trainControl(method = "cv", number = 3, search = "grid",
                          verboseIter = TRUE)

set.seed(123)
tuned_model_tree = caret::train(cnt ~ .,
                                data = train,
                                method = "rpart",
                                metric = "MAE",
                                trControl = trControl,
                                tuneGrid = rpart_grid)

predictions_tuned_model_tree = predict(tuned_model_tree,test)

#Calcuate MAE, RMSE, R-squared for testing data
print(postResample(pred = predictions_tuned_model_tree, obs = test$cnt))

#Plot a graph for actual vs predicted values
plot(test$cnt,type="l",lty=2,col="red")
lines(predictions_tuned_model_tree,col="blue")

#Predict a sample data
predict(tuned_model_tree,test[4,])
#18
#1416

###-------------------Random forest---------------------------------------###
# RMSE     : 587.0125339
# Rsquared : 0.9204742
# MAE      : 423.8855792
regressor_rf = randomForest(cnt~., data = train, ntrees = 500)

#Predict the test cases
y_pred_rf = predict(regressor_rf,test)
```

```
#Create dataframe for actual and predicted values
model_pred = cbind(model_pred,y_pred_rf)
#head(model_pred)

#Get the summary of the model
#summary(regressor_rf)

#Calcuate MAE, RMSE, R-squared for testing data
print(postResample(pred = y_pred_rf, obs = test$cnt))

#Plot a graph for actual vs predicted values
plot(test$cnt,type="l",lty=2,col="red")
lines(y_pred_rf,col="blue")

#Predict a sample data
predict(regressor_rf,test[4,])
#18
#1252.337

#####**********RF Hyperparameter tuning with
caret**********####
##----------Commenting the code as computation time is long------- ##########
# # RMSE     : 561.6378007
# # Rsquared : 0.9241534
# # MAE      : 396.1909000
# set.seed(123)
# # default ntrees = 500
# tuned_model_forest = caret::train(cnt~.,
#                           data = train,
#                           method = 'rf',
#                           metric = 'MAE',
#                           trControl = trainControl(method = "repeatedcv",
#                                                    number = 10,
#                                                    repeats = 3,
#                                                    verboseIter = TRUE
#                           ),
#                           tuneGrid = expand.grid(.mtry = c(1:10)),
```

```
#                             ntree = 500
# )
#
#
# predictions_tuned_model_forest = predict(tuned_model_forest,test[,-11])
#
# #Calcuate MAE, RMSE, R-squared for testing data
# print(postResample(pred = predictions_tuned_model_forest, obs = test$cnt))
#
# #Plot a graph for actual vs predicted values
# plot(test$cnt,type="l",lty=2,col="red")
# lines(predictions_tuned_model_forest,col="blue")
#
# #Predict a sample data
# predict(tuned_model_forest,test[4,])
# #18
# #1182.849

##############K-Fold Cross
validation################################################
set.seed(123)
KFData = bike_rental_data
train_KF = KFData[train_index,]
test_KF = KFData[-train_index,]

###--------------------Random forest----------------------------------------###
# RMSE     :
# Rsquared :
# MAE      :
KF_RF = train(cnt~.,
              data = train_KF,
              method = "rf",
              tuneGrid = expand.grid(mtry = c(2,3,4)),
              trControl = trainControl(method = "cv",
                                       number = 5,
                                       verboseIter = FALSE,))
print(KF_RF)
knitr::kable(head(KF_RF$results), digits = 3)
```

```
print(KF_RF$bestTune)
predictions_KF_RF = predict(KF_RF, test_KF[-25])

#Calcuate MAE, RMSE, R-squared for testing data
print(postResample(pred = predictions_KF_RF, obs = test$cnt))

#Plot a graph for actual vs predicted values
plot(test$cnt,type="l",lty=2,col="red")
lines(predictions_KF_RF,col="blue")

##################################################################################
```

References

[1] https://rdrr.io
[2] https://www.rdocumentation.org [3] https://edwisor.com/home
[4] https://www.udemy.com
[5] https://docs.python.org/3.9/