HarvardX: Data Science
Bike Sharing Count Prediction Project

Sabiny Chungath Abdul Jaleel

9/11/2020

Introduction

This is the final project part of Professional Certificate for Data Science course from Harvard University using R programming language. The objective of this project is to analyse the 'BikeSharing' dataset and predict the bike sharing count based on the dataset. This dataset contains datetime, season, holiday, workingday, weather, temp, atemp, humidity, windspeed, casual, registererd, count etc.

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.

Attribute Information: Both train.csv and test.csv have the following fields:

- datetime : date + timestamp in "mm/dd/yyyy hh:mm" format
- season : seasons (1:spring, 2:summer, 3:fall, 4:winter)
- holiday : whether the day is considered a holiday
- workingday : whether the day is neither a weekend nor holiday
- weather : Type of weather (1:Good, 2:Normal, 3:Bad, 4:Very Bad)
- temp : Normalized temperature in Celsius.
- atemp : "feels like" temperature in Celsius
- humidity : relative humidity
- windspeed : Normalized wind speed.
- casual : count of casual users
- registered : count of registered users
- count : count of total rental bikes including both casual and registered

Methods & Analysis

The bike sharing dataset shows 11 columns. We are ignoring the casual and registered fields as this sum is equal to count field. Here we also analysing the average count of bikes rent by season, weather and day.

The goal is to train a machine learning algorithm using the inputs of a provided training subset to predict bike sharing counts in a validation set. Here, 4 different algorithms are used to predict the accuracy. We applied K Nearest Neighbor(KNN) Model, Support Vector Machine (SVM) Model, Linear Regression Model and Random Forest Model. We found that random forest is performing the best.

The focus is on the predictive accuracy of the algorithm. During analysis we will review RMSE and rmsle(log RMSE). We will finally report both RMSE and rmsle.

Loading the Dataset

We will utilize and load several packages from CRAN to assist with our analysis. These will be automatically downloaded and installed during code execution.

```r
#############################################################
# Load the data sets from GitHub link
#############################################################
if(!require(tidyverse)) install.packages("tidyverse",repos="http://cran.us.r-project.org")
if(!require(tidyr)) install.packages("tidyr",repos="http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse",repos="http://cran.us.r-project.org")
if(!require(stringr)) install.packages("stringr",repos="http://cran.us.r-project.org")
if(!require(ggplot2)) install.packages("ggplot2",repos="http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr",repos="http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret",repos="http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071",repos="http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("randomForest",repos="http://cran.us.r-project.org")
if(!require(sqldf)) install.packages("sqldf",repos="http://cran.us.r-project.org")
if(!require(corrplot)) install.packages("corrplot",repos="http://cran.us.r-project.org")
if(!require(funModeling)) install.packages("funModeling",repos="http://cran.us.r-project.org")
if(!require(caTools)) install.packages("caTools",repos="http://cran.us.r-project.org")
if(!require(Metrics)) install.packages("Metrics",repos="http://cran.us.r-project.org")
# Loading all needed libraries

library(dplyr)
library(tidyverse)
library(tidyr)
library(ggplot2)
library(caret)
library(corrplot)


bike_share_train <- read.csv(
  "https://raw.githubusercontent.com/sabinychungath/Bike-Sharing/master/train.csv",
  header = TRUE)
bike_share_test <- read.csv(
  "https://raw.githubusercontent.com/sabinychungath/Bike-Sharing/master/test.csv",
  header = TRUE)
```

Data Exploration

```r
#List first few lines of bike sharing dataset
head(bike_share_train)
```

```
##              datetime season holiday workingday weather temp  atemp humidity
## 1 2011-01-01 00:00:00      1       0          0       1 9.84 14.395       81
## 2 2011-01-01 01:00:00      1       0          0       1 9.02 13.635       80
## 3 2011-01-01 02:00:00      1       0          0       1 9.02 13.635       80
## 4 2011-01-01 03:00:00      1       0          0       1 9.84 14.395       75
## 5 2011-01-01 04:00:00      1       0          0       1 9.84 14.395       75
## 6 2011-01-01 05:00:00      1       0          0       2 9.84 12.880       75
##   windspeed casual registered count
## 1    0.0000      3         13    16
## 2    0.0000      8         32    40
## 3    0.0000      5         27    32
## 4    0.0000      3         10    13
## 5    0.0000      0          1     1
## 6    6.0032      0          1     1
```

```r
#Type of bike sharing dataset
class(bike_share_train)
```

```
## [1] "data.frame"
```

```r
names(bike_share_train)
```

```
##  [1] "datetime"   "season"     "holiday"    "workingday" "weather"
##  [6] "temp"       "atemp"      "humidity"   "windspeed"  "casual"
## [11] "registered" "count"
```

```r
#Variables data types
str(bike_share_train)
```

```
## 'data.frame':    10886 obs. of  12 variables:
##  $ datetime  : chr  "2011-01-01 00:00:00" "2011-01-01 01:00:00" "2011-01-01 02:00:00" "2011-01-01 03
##  $ season    : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ workingday: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ weather   : int  1 1 1 1 1 2 1 1 1 1 ...
##  $ temp      : num  9.84 9.02 9.02 9.84 9.84 ...
##  $ atemp     : num  14.4 13.6 13.6 14.4 14.4 ...
##  $ humidity  : int  81 80 80 75 75 75 80 86 75 76 ...
##  $ windspeed : num  0 0 0 0 0 ...
##  $ casual    : int  3 8 5 3 0 0 2 1 1 8 ...
##  $ registered: int  13 32 27 10 1 1 0 2 7 6 ...
##  $ count     : int  16 40 32 13 1 1 2 3 8 14 ...
```

```r
colnames(bike_share_train)
```

```
##  [1] "datetime"   "season"     "holiday"    "workingday" "weather"
##  [6] "temp"       "atemp"      "humidity"   "windspeed"  "casual"
## [11] "registered" "count"
```

```r
# summary statistics
summary(bike_share_train)
```

```
##    datetime             season          holiday          workingday
##  Length:10886       Min.   :1.000   Min.   :0.00000   Min.   :0.0000
##  Class :character   1st Qu.:2.000   1st Qu.:0.00000   1st Qu.:0.0000
##  Mode  :character   Median :3.000   Median :0.00000   Median :1.0000
##                     Mean   :2.507   Mean   :0.02857   Mean   :0.6809
##                     3rd Qu.:4.000   3rd Qu.:0.00000   3rd Qu.:1.0000
##                     Max.   :4.000   Max.   :1.00000   Max.   :1.0000
##     weather           temp           atemp          humidity
##  Min.   :1.000   Min.   : 0.82   Min.   : 0.76   Min.   :  0.00
##  1st Qu.:1.000   1st Qu.:13.94   1st Qu.:16.66   1st Qu.: 47.00
##  Median :1.000   Median :20.50   Median :24.24   Median : 62.00
##  Mean   :1.418   Mean   :20.23   Mean   :23.66   Mean   : 61.89
##  3rd Qu.:2.000   3rd Qu.:26.24   3rd Qu.:31.06   3rd Qu.: 77.00
##  Max.   :4.000   Max.   :41.00   Max.   :45.45   Max.   :100.00
##    windspeed          casual         registered        count
##  Min.   : 0.000   Min.   :  0.00   Min.   :  0.0   Min.   :  1.0
##  1st Qu.: 7.002   1st Qu.:  4.00   1st Qu.: 36.0   1st Qu.: 42.0
##  Median :12.998   Median : 17.00   Median :118.0   Median :145.0
##  Mean   :12.799   Mean   : 36.02   Mean   :155.6   Mean   :191.6
##  3rd Qu.:16.998   3rd Qu.: 49.00   3rd Qu.:222.0   3rd Qu.:284.0
##  Max.   :56.997   Max.   :367.00   Max.   :886.0   Max.   :977.0
```

Data Cleaning

In this section we will take the first look at the loaded data frames. We will also perform necessary cleaning and some transformations so that the data better suits our needs. Here we are ignore the casual and registered fields as this sum is equal to count field.

```
# Ignore the casual, registered fields as this sum is equal to count field
bike_share_train <- bike_share_train[,-c(10,11)]
```

Now we need to look if there are any missing values in our dataframe.

```
# Check for NA values
#Missing values in dataset
missing_value <- data.frame(apply(bike_share_train,2,
                                  function(x)
                                    {
                                    sum(is.na(x))
                                    }
                                  ))
names(missing_value)[1]='missing_value'
missing_value
```

```
##            missing_value
## datetime               0
## season                 0
## holiday                0
## workingday             0
## weather                0
## temp                   0
## atemp                  0
## humidity               0
## windspeed              0
## count                  0
```

Above you can see that Our dataset has no missing values in the data frame. Let's transform our variables to class **factor** for using them in our analysis.

```
# Converting integer to factor on training set
bike_share_train$season <- as.factor(bike_share_train$season)
bike_share_train$holiday <- as.factor(bike_share_train$holiday)
bike_share_train$workingday <- as.factor(bike_share_train$workingday)
bike_share_train$weather <- as.factor(bike_share_train$weather)
bike_share_train$datetime <-as.POSIXct(bike_share_train$datetime,
                                       format="%Y-%m-%d %H:%M:%S")
```

```
# Converting int to factor on test set
bike_share_test$season <- as.factor(bike_share_test$season)
bike_share_test$holiday <- as.factor(bike_share_test$holiday)
bike_share_test$workingday <- as.factor(bike_share_test$workingday)
bike_share_test$weather <- as.factor(bike_share_test$weather)
bike_share_test$datetime <-as.POSIXct(bike_share_test$datetime,
                                      format="%Y-%m-%d %H:%M:%S")
```

Till now, we have got a fair understanding of the data set. Now, let's test the hypothesis which we had generated earlier. Here I have added some additional hypothesis from the dataset. Let's test them one by one:

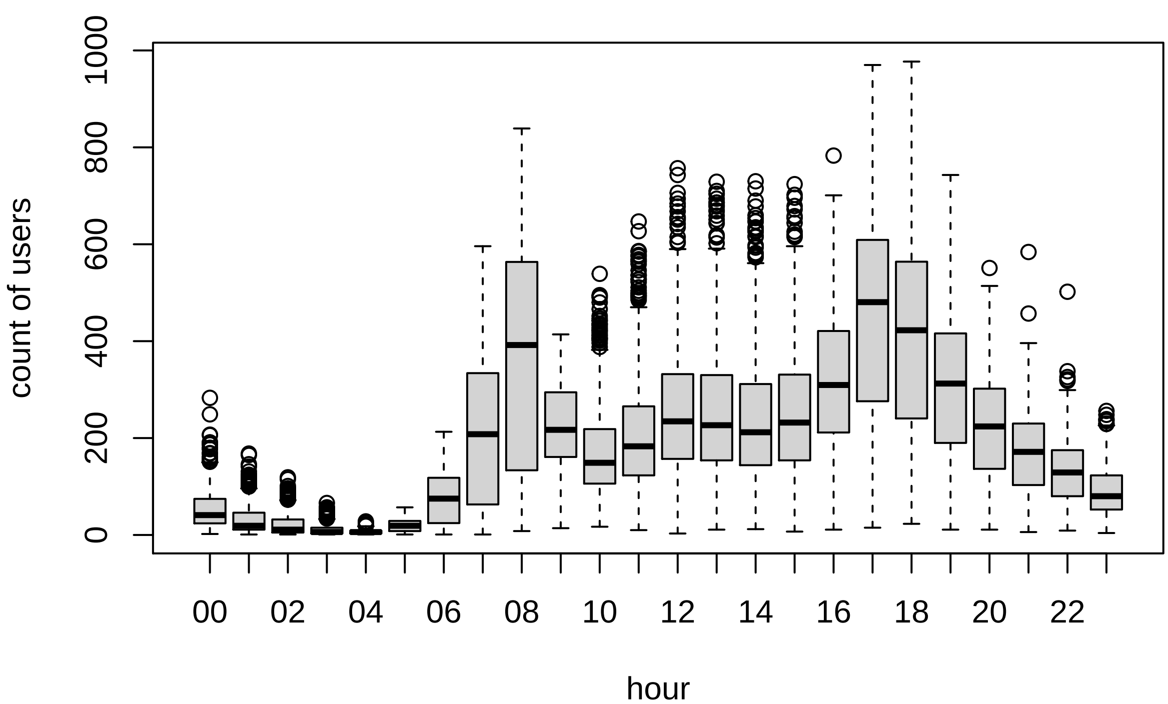
```
# Extract day from datetime value
bike_share_train$day <-  strftime(bike_share_train$datetime, '%u')
bike_share_train$day <- as.factor(bike_share_train$day)
bike_share_test$day <-  strftime(bike_share_test$datetime, '%u')
bike_share_test$day <- as.factor(bike_share_test$day)
```

Let's extract hour from the datetime filed of both train and test datasets to find the bikes are rented out is more on which hour. We don't have the variable 'hour' with us right now. But we can extract it using the datetime column to find the bikes are rented out is more on weekdays or weekends.

```
# Extract hour from datetime value
bike_share_train$hour <- substring(bike_share_train$datetime, 12,13)
bike_share_train$hour <- as.factor(bike_share_train$hour)
bike_share_test$hour <- substring(bike_share_test$datetime, 12,13)
bike_share_test$hour <- as.factor(bike_share_test$hour)
```

Let's plot the hourly trend of count over hours and check if our hypothesis is correct or not. We will separate train and test data set from combined one.

```
boxplot(bike_share_train$count ~ bike_share_train$hour,xlab="hour", ylab="count of users")
```



Above, you can see the trend of bike count over hours. Quickly, I'll segregate the bike demand in three categories:

- High : 7-9 and 17-19 hours
- Average : 10-16 hours
- Low : 0-6 and 20-24 hours

Here I have analyzed the distribution of total bike demand.

We are going to remove the datetime field after extract hour and day from that.

```
# Removing datetime field
bike_share_train <- bike_share_train[,-1]
```

```
#list first few lines of train & test dataset after cleaning
head(bike_share_train)
```

```
##   season holiday workingday weather temp  atemp humidity windspeed count day
## 1      1       0          0       1 9.84 14.395       81    0.0000    16   6
## 2      1       0          0       1 9.02 13.635       80    0.0000    40   6
## 3      1       0          0       1 9.02 13.635       80    0.0000    32   6
## 4      1       0          0       1 9.84 14.395       75    0.0000    13   6
## 5      1       0          0       1 9.84 14.395       75    0.0000     1   6
## 6      1       0          0       2 9.84 12.880       75    6.0032     1   6
##   hour
## 1   00
## 2   01
## 3   02
## 4   03
## 5   04
## 6   05
```

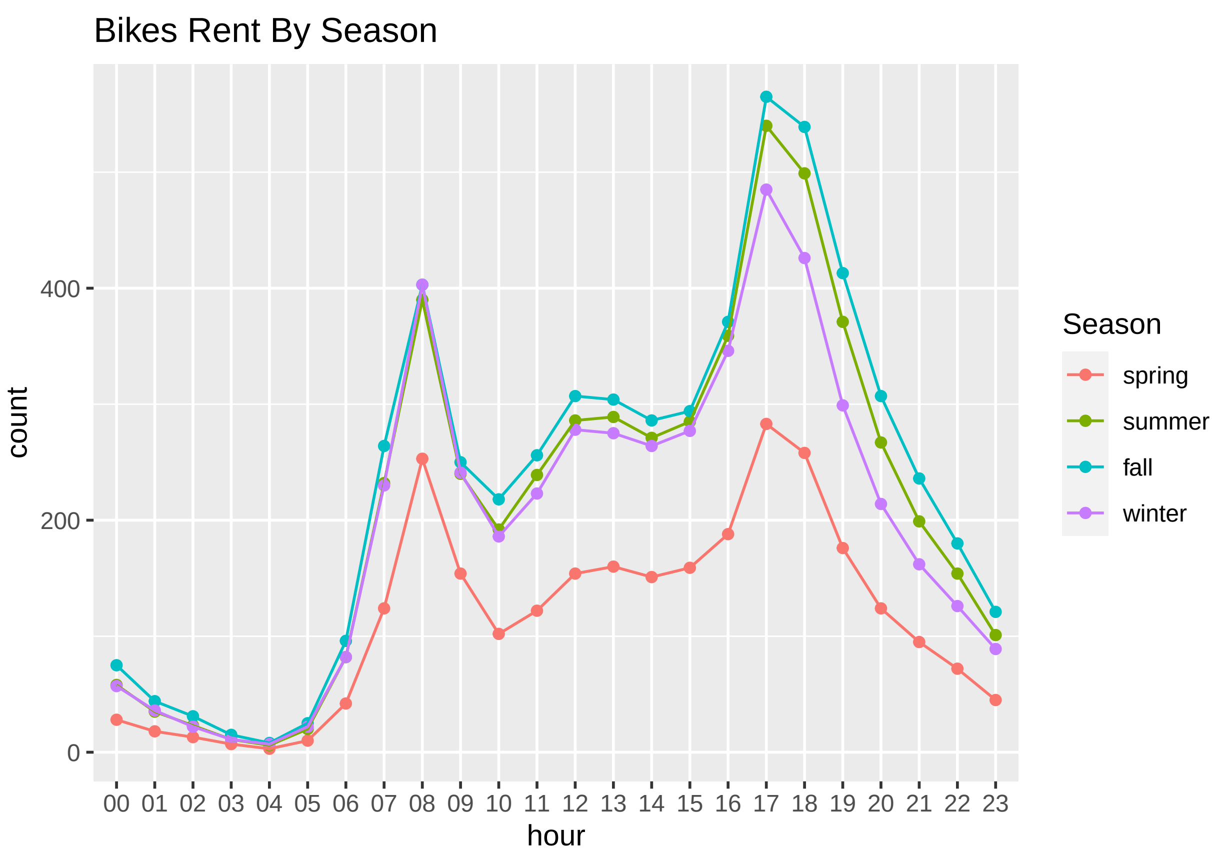
```
head(bike_share_test)
```

```
##              datetime season holiday workingday weather  temp  atemp humidity
## 1 2011-01-20 00:00:00      1       0          1       1 10.66 11.365       56
## 2 2011-01-20 01:00:00      1       0          1       1 10.66 13.635       56
## 3 2011-01-20 02:00:00      1       0          1       1 10.66 13.635       56
## 4 2011-01-20 03:00:00      1       0          1       1 10.66 12.880       56
## 5 2011-01-20 04:00:00      1       0          1       1 10.66 12.880       56
## 6 2011-01-20 05:00:00      1       0          1       1  9.84 11.365       60
##   windspeed day hour
## 1   26.0027   4   00
## 2    0.0000   4   01
## 3    0.0000   4   02
## 4   11.0014   4   03
## 5   11.0014   4   04
## 6   15.0013   4   05
```

Data Visualization

```
library(sqldf)
library(ggplot2)
# Get the average count of bikes rent by season, hour
season_hour <- sqldf(
  'select season, hour, avg(count) as count from bike_share_train group by season, hour')

#Plot average count of bikes rent by season & hour
bike_share_train %>%
  ggplot(aes(x=hour, y=count, color=season)) +
  geom_point(data = season_hour, aes(group = season)) +
  geom_line(data = season_hour, aes(group = season)) +
  ggtitle("Bikes Rent By Season") +
  scale_colour_hue('Season',breaks = levels(bike_share_train$season),
                   labels=c('spring', 'summer', 'fall', 'winter'))
```

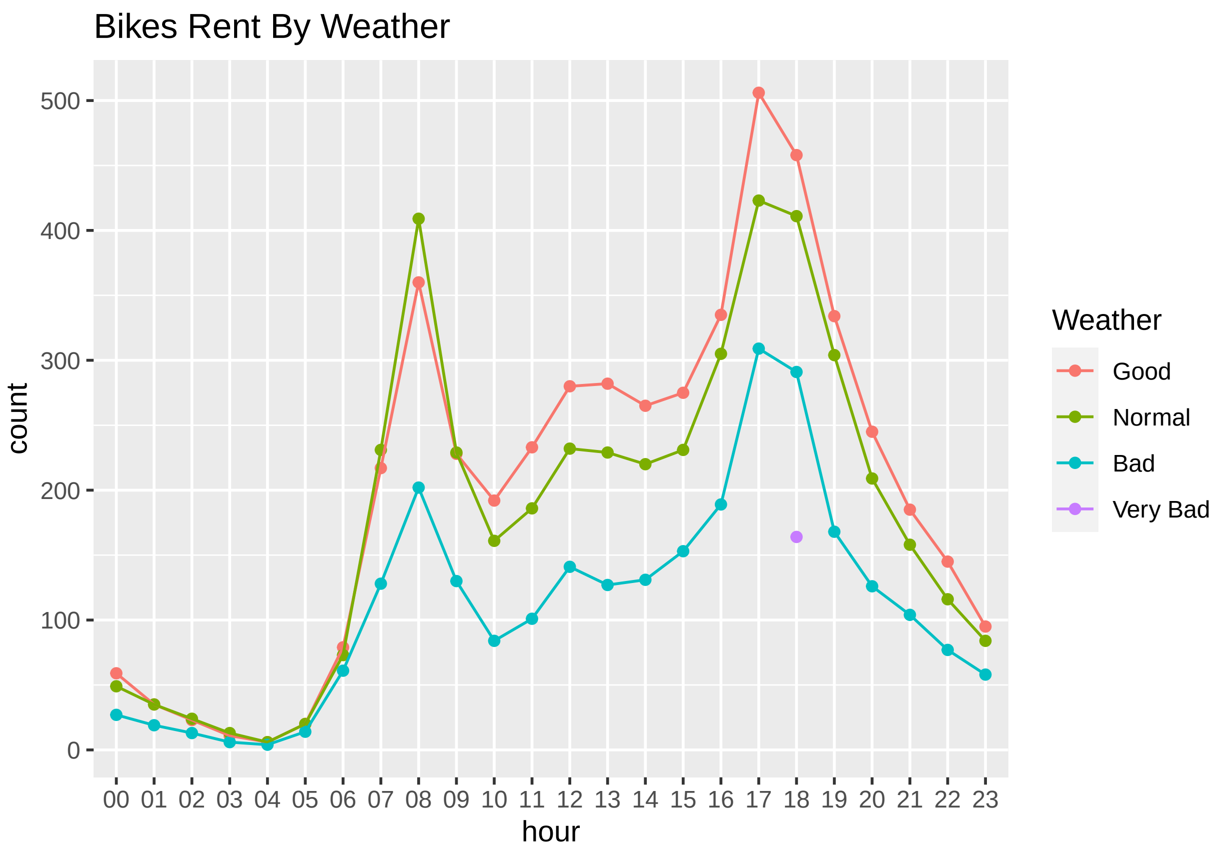


From this plot it shows:

- There are more rental in morning(from 7-9th hour) and evening(16-19th hour)
- People rent bikes more in Fall, and much less in Spring

```r
# Get the average count of bikes rent by weather, hour
weather_hour <- sqldf(
  'select weather, hour, avg(count) as count from bike_share_train group by weather, hour')

#Plot average count of bikes rent by weather & hour
bike_share_train %>%
  ggplot(aes(x=hour, y=count, color=weather)) +
  geom_point(data = weather_hour, aes(group = weather)) +
  geom_line(data = weather_hour, aes(group = weather)) +
  ggtitle("Bikes Rent By Weather") +
  scale_colour_hue('Weather',breaks = levels(bike_share_train$weather),
                   labels=c('Good', 'Normal', 'Bad', 'Very Bad'))
```



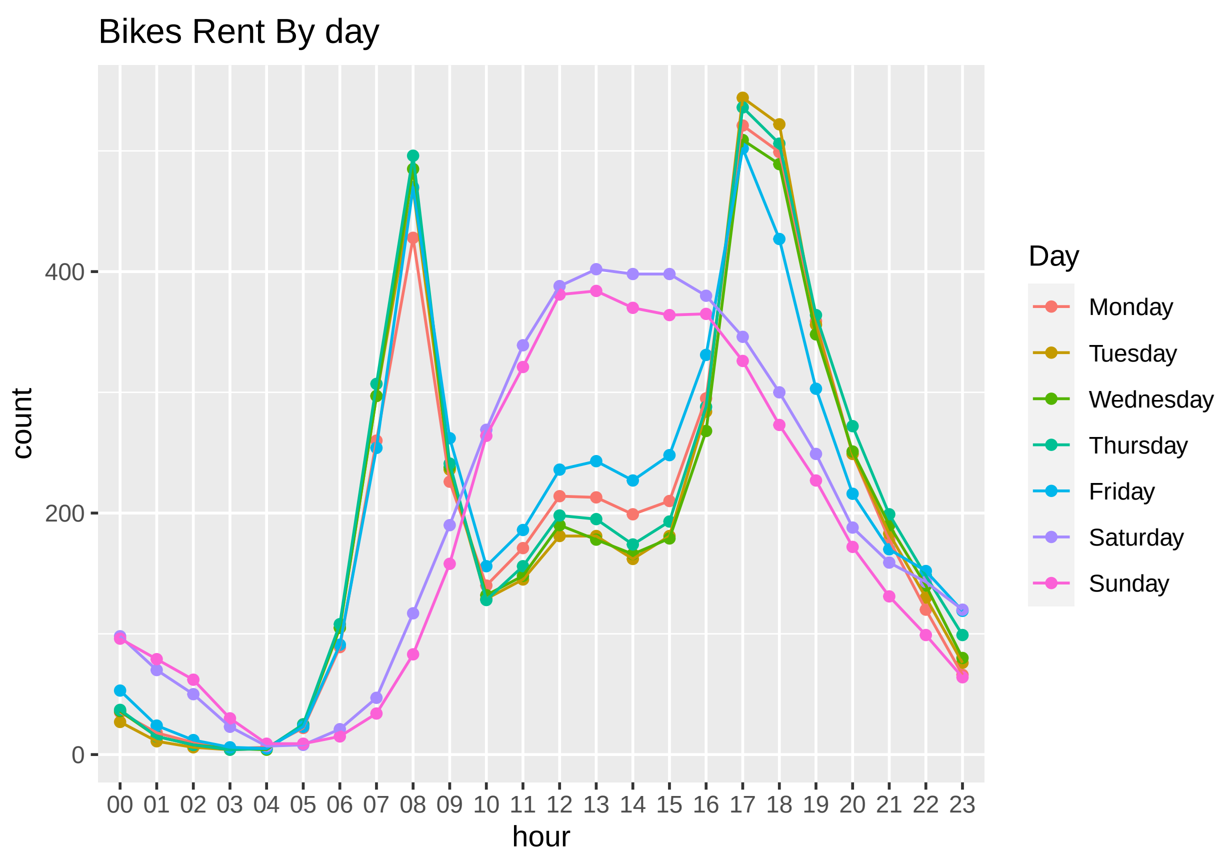
From this plot it shows,

- People rent bikes more when weather is good

-We see bike rent only at 18th hour when weather is very bad

```r
#average count of bikes rent by day & hour
day_hour <- sqldf(
  'select day, hour, avg(count) as count from bike_share_train group by day, hour')

#Plot average count of bikes rent by day & hour
bike_share_train %>%
  ggplot(aes(x=hour, y=count, color=day)) +
  geom_point(data = day_hour, aes(group = day)) +
  geom_line(data = day_hour, aes(group = day)) +
  ggtitle("Bikes Rent By day") +
  scale_colour_hue('Day',breaks = levels(bike_share_train$day),
    labels=c('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'))
```

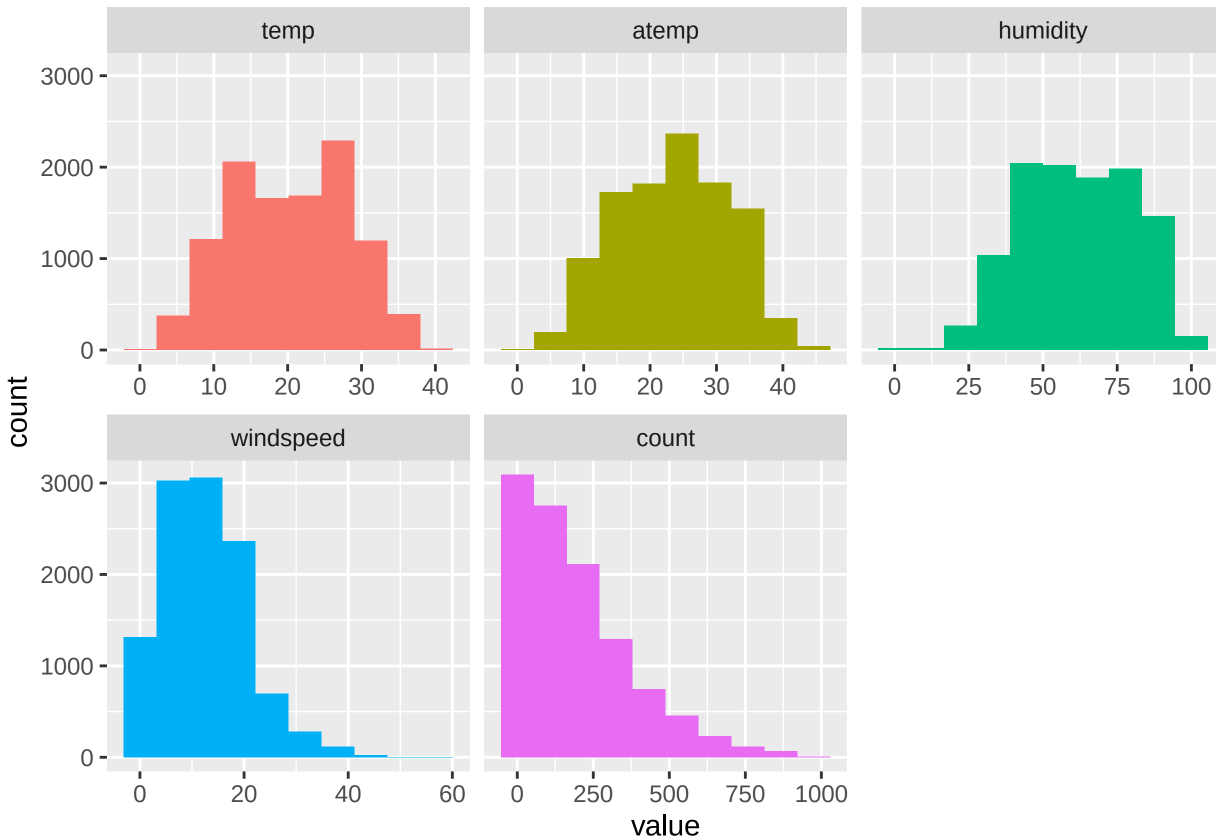


From this plot it shows,

- There are more bikes rent on weekdays during morining and evening
- There are more bikes rent on weekends during daytime

Now, I'll plot a histogram for each numerical variables and analyze the distribution.

```
library(funModeling)
# Plotting Numerical bike_sharing
plot_num(bike_share_train)
```

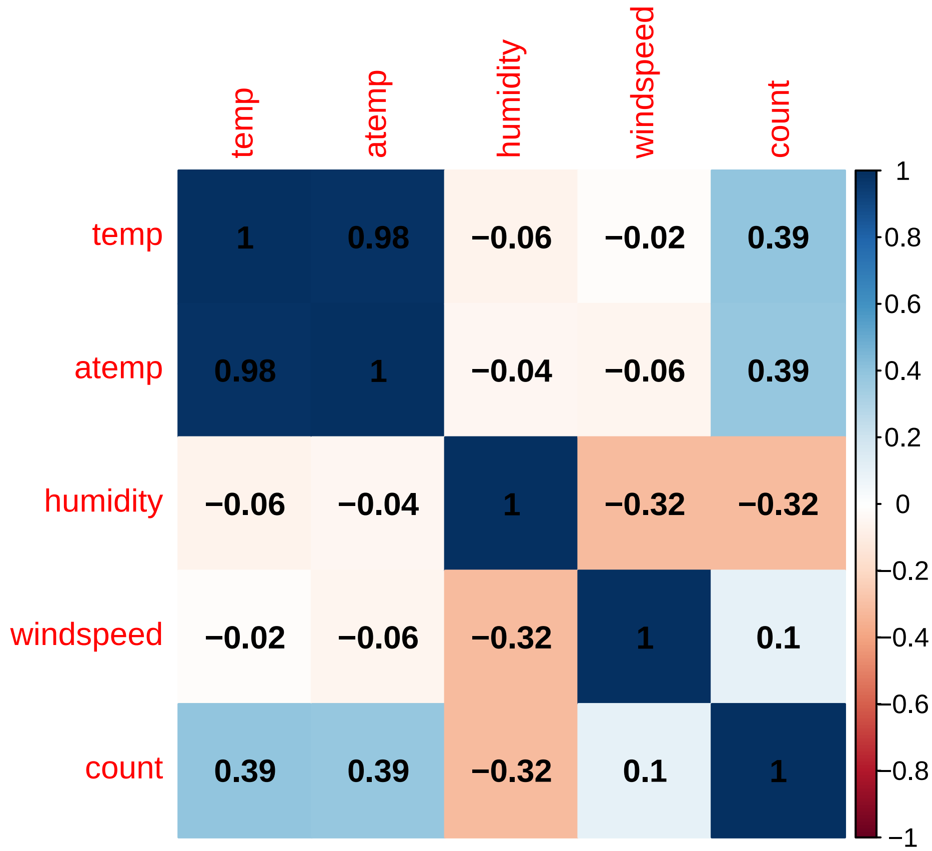


Here, variables temp, atemp, humidity and windspeed looks naturally distributed.

Correlation Analysis

```
# Correlation plot between fields
bike_share_train_subset <- bike_share_train[,5:9]
bike_share_train_subset$humidity <- as.numeric(bike_share_train_subset$humidity)
bike_share_train_subset$count <- as.numeric(bike_share_train_subset$count)

train_cor <- cor(bike_share_train_subset)
corrplot(train_cor, method = 'color', addCoef.col="black")
```



This correlationplot shows that temp, atemp has much correlation.

Splitting the Train dataset

```
library(caTools)
set.seed(123)
split <- sample.split(bike_share_train$count, SplitRatio = 0.75)
training_set <- subset(bike_share_train, split == TRUE)
validation_set <- subset(bike_share_train, split == FALSE)
```

Model Development

The goal is to train a machine learning algorithm that predicts bike sharing counts using the inputs of a above subset to predict movie ratings in a provided validation set.

The loss-function computes the RMSE, defined as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

with N being the number of user/movie combinations and the sum occurring over all these combinations. The RMSE is our measure of model accuracy. We can interpret the RMSE similarly to a standard deviation: it is the typical error we make when predicting a movie rating. If its result is larger than 1, it means that our typical error is larger than one star, which is not a good result. The written function to compute the RMSE for vectors of ratings and their corresponding predictions is:

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

K Nearest Neighbor(KNN) Model

The next method to consider is the "Nearest Neighbours", here the K-nearest neighbours. This will find the k closest matching points from the training data. These will have their results averaged to find the predicted outcome. The value of K is a tuning parameter, and the following code will attempt to find the most appropriate value.

```
#Training the model
knn_model <- train(count ~ ., method = "knn",data = training_set)

# Apply prediction on validation set
knn_predict <- predict(knn_model, newdata = validation_set)
print("summary of KNN prediction")
```

```
## [1] "summary of KNN prediction"
```

```
summary(knn_predict)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   7.222 100.576 167.389 188.733 256.444 662.222
```

```
library(Metrics)
#Root-mean-square error value between actual and predicted
knn_RMSE <- RMSE(validation_set$count,knn_predict)
print("RMSE value between actual and predicted")
```

```
## [1] "RMSE value between actual and predicted"
```

```r
knn_RMSE
```

```
## [1] 139.5918
```

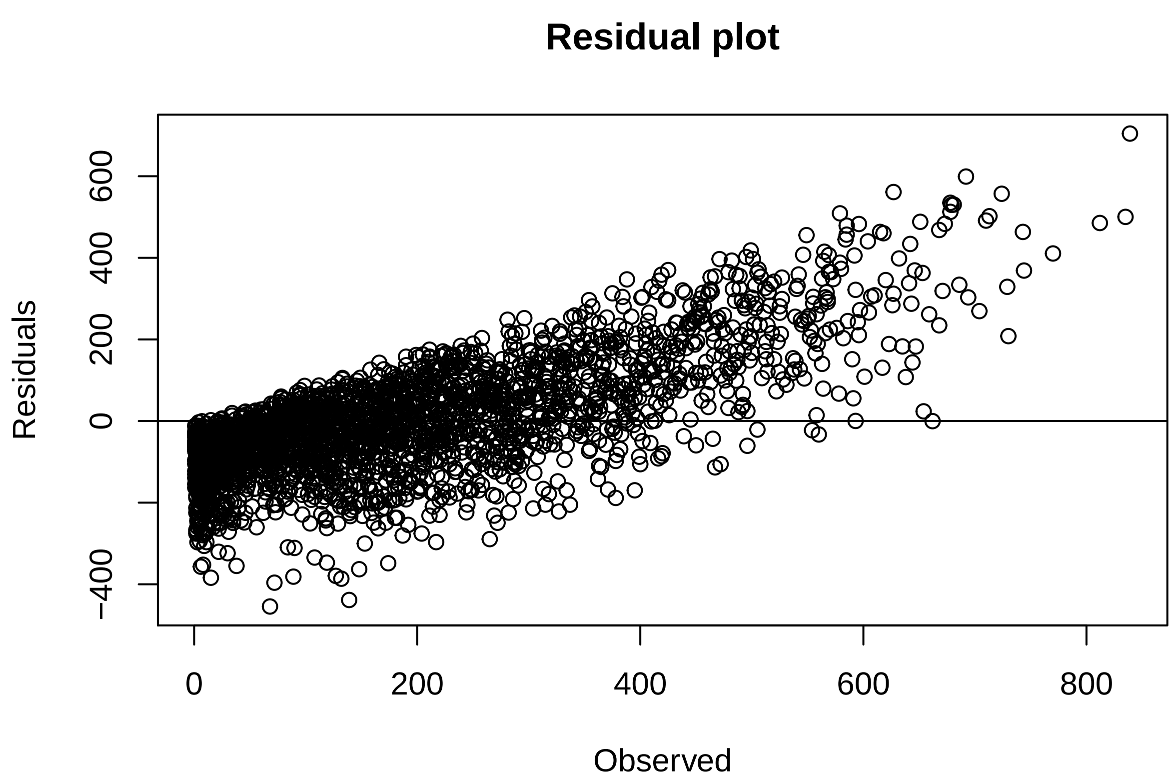
```r
#If we want to penalize under-prediction of count, rmsle might be a better metric
knn_log_RMSE<-rmsle(validation_set$count,knn_predict)
print("Log RMSE value")
```

```
## [1] "Log RMSE value"
```

```r
knn_log_RMSE
```

```
## [1] 1.304029
```

```r
#Residual plot vs Fitted plot
y_test <- validation_set$count
residuals <- y_test - knn_predict
plot(y_test,residuals,
     xlab='Observed',
     ylab='Residuals',
     main='Residual plot')
abline(0,0)
```



Our RMSE score for KNN was 139.5918 and rmsle score 1.304029. Not an improvement; We can try another model for better rmsle.

SVM - Support Vector Machine Model

```r
library(e1071)
#Training the model
svm_model <- svm(count ~ ., data = training_set, kernel='sigmoid')

# Apply prediction on validation set
svm_predict <- predict(svm_model, newdata = validation_set)
print("summary of SVM prediction")
```

```
## [1] "summary of SVM prediction"
```

```r
summary(svm_predict)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -322.55   84.81  186.53  180.17  272.57  748.07
```

```r
#Root-mean-square error value between actual and predicted
svm_RMSE<-RMSE(validation_set$count,svm_predict)
print("RMSE value between actual and predicted")
```

```
## [1] "RMSE value between actual and predicted"
```

```r
svm_RMSE
```

```
## [1] 128.4377
```

From above summary we saw negative values of predicted count. We don't want negative values as forecast for bike count. Replace all negative numbers with 1

```r
#Replace all negative numbers with 1
Output_svmMod <- svm_predict
Output_svmMod[svm_predict<=0] <- 1
```

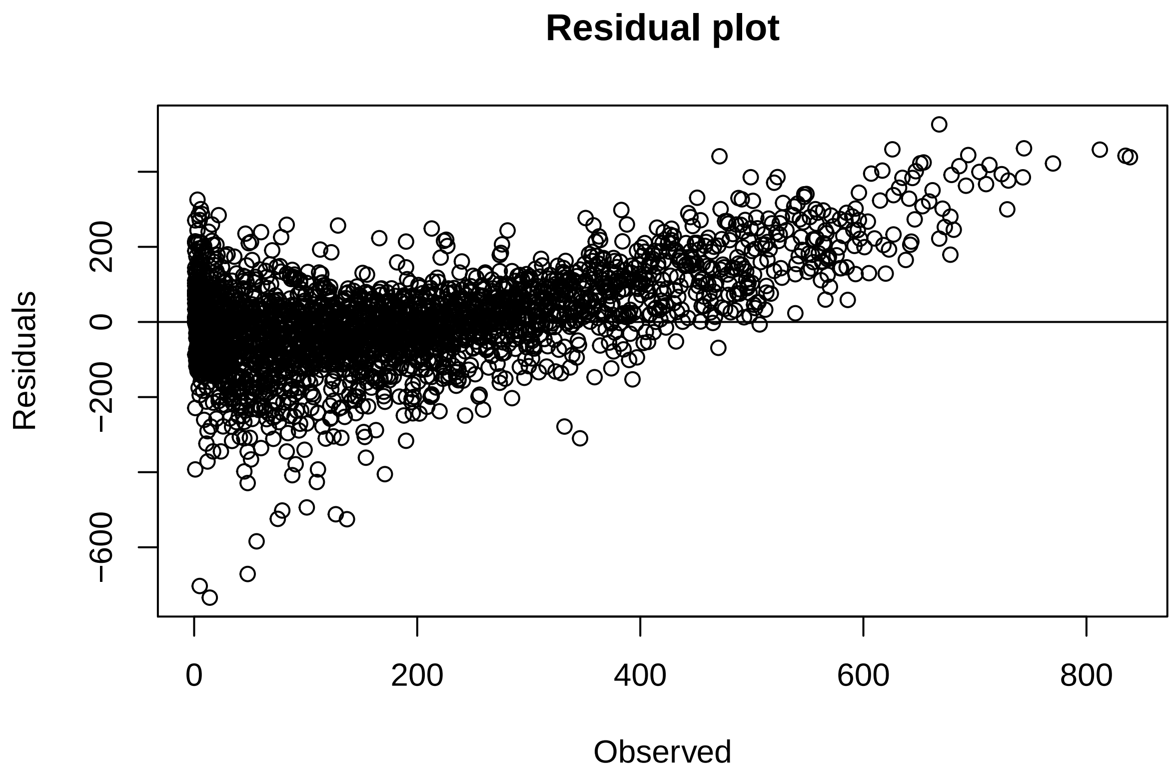
```r
#If we want to penalize under-prediction of demand, rmsle might be a better metric
svm_log_RMSE<-rmsle(validation_set$count,Output_svmMod)
print("RMSE value after replaced the negative values")
```

```
## [1] "RMSE value after replaced the negative values"
```

```r
svm_log_RMSE
```

```
## [1] 1.179682
```

```r
#Residual plot
y_test <- validation_set$count
residuals <- y_test - svm_predict
plot(y_test,residuals,
     xlab='Observed',
     ylab='Residuals',
     main='Residual plot')
abline(0,0)
```



Our RMSE score for SVM was 128.4377 and rmsle score 1.179682. Not an improvement; We can try another model for better rmsle.

Linear Regression model

Linear regression is a linear approach to modeling the relationship between a scalar response and one or more explanatory variables.

```
#Training the model
lm_model <- lm(count~., data = training_set)

#Stepwise Model Selection
# Now performs stepwise model selection by AIC with both directions(Forward, Backward)
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
lm_model_AIC<-stepAIC(lm_model, direction="both")
```

```
## Start:  AIC=77514.76
## count ~ season + holiday + workingday + weather + temp + atemp +
##     humidity + windspeed + day + hour
##
##
## Step:  AIC=77514.76
## count ~ season + holiday + weather + temp + atemp + humidity +
##     windspeed + day + hour
##
##             Df Sum of Sq       RSS   AIC
## - holiday    1      2134 103059123 77513
## <none>                   103056989 77515
## - atemp      1     87643 103144632 77520
## - windspeed  1    176557 103233545 77527
## - day        6    336228 103393217 77529
## - temp       1    235337 103292326 77531
## - humidity   1   1447691 104504680 77627
## - weather    3   1937244 104994233 77662
## - season     3   4849413 107906402 77886
## - hour      23 102698139 205755127 83141
##
## Step:  AIC=77512.93
## count ~ season + weather + temp + atemp + humidity + windspeed +
##     day + hour
##
##              Df Sum of Sq       RSS   AIC
## <none>                    103059123 77513
## + holiday     1      2134 103056989 77515
## + workingday  1      2134 103056989 77515
## - atemp       1     88823 103147946 77518
## - windspeed   1    176591 103235714 77525
## - day         6    346243 103405366 77528
## - temp        1    234141 103293264 77530
## - humidity    1   1447773 104506896 77625
## - weather     3   1936592 104995715 77660
## - season      3   4848986 107908109 77884
## - hour       23 102696072 205755195 83139
```

```r
# Apply prediction on validation set
lm_predict <- predict(lm_model_AIC, newdata = validation_set)
print("summary of lm prediction")
```

```
## [1] "summary of lm prediction"
```

```r
summary(lm_predict)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -162.54   74.64  189.95  191.86  294.82  634.53
```

```r
#RMSE value between actual and predicted
lm_RMSE<-RMSE(validation_set$count,lm_predict)
print("RMSE value between actual and predicted")
```

```
## [1] "RMSE value between actual and predicted"
```

```
lm_RMSE
```

```
## [1] 102.8466
```

From above summary we saw negative values of predicted count. We don't want negative values as forecast for bike count. Replace all negative numbers with 1

```
#Replace all negative numbers with 1
Output_lmMod <- lm_predict
Output_lmMod[lm_predict<=0] <-1
```

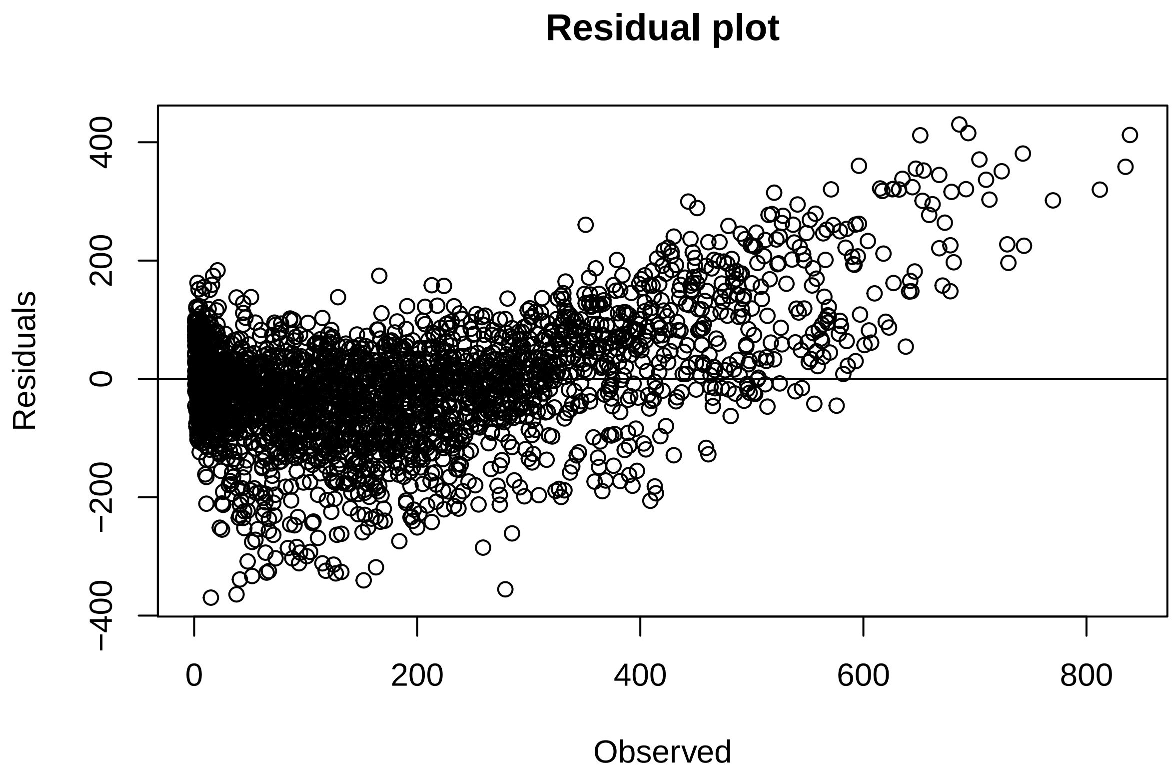
```
#If we want to penalize under-prediction of demand, rmsle might be a better metric
lm_log_RMSE<-rmsle(validation_set$count,Output_lmMod)
print("RMSE value after replaced the negative values")
```

```
## [1] "RMSE value after replaced the negative values"
```

```
lm_log_RMSE
```

```
## [1] 0.9549201
```

```
#Residual plot vs Fitted plot
y_test <- validation_set$count
residuals <- y_test - lm_predict
plot(y_test,residuals,
     xlab='Observed',
     ylab='Residuals',
     main='Residual plot')
abline(0,0)
```



Our RMSE score for Linear Regression Model was 102.8466 and rmsle score 0.9549201. Not an improvement; We can try another model for better rmsle.

Random Forest Model

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression).

```
library(randomForest)
#training the model
rf_model<-randomForest(count~. ,data = training_set,importance=TRUE,ntree=200)
```

```
# Apply prediction on validation set
rf_predict <- predict(rf_model, newdata = validation_set)
print("summary of random forest prediction")
```

```
## [1] "summary of random forest prediction"
```

```
summary(rf_predict)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   5.147  65.552 168.543 188.096 265.306 764.427
```

```r
#RMSE value between actual and predicted
rf_RMSE <- RMSE(validation_set$count,rf_predict)
print("RMSE value between actual and predicted")
```

```
## [1] "RMSE value between actual and predicted"
```

```r
rf_RMSE
```

```
## [1] 66.31489
```

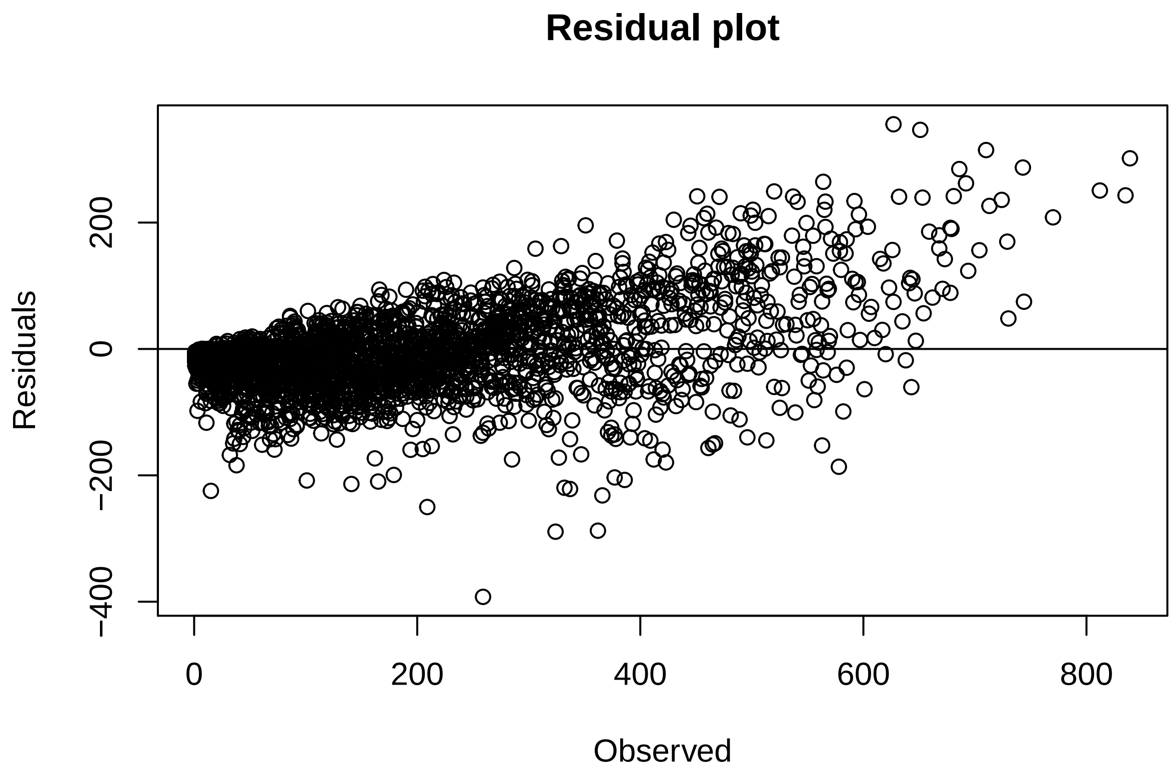
```r
#If we want to penalize under-prediction of demand, rmsle might be a better metric
rf_log_RMSE<-rmsle(validation_set$count,rf_predict)
print("log RMSE value")
```

```
## [1] "log RMSE value"
```

```r
rf_log_RMSE
```

```
## [1] 0.6376547
```

```r
#Residual plot
y_test <- validation_set$count
residuals <- y_test - rf_predict
plot(y_test,residuals,
     xlab='Observed',
     ylab='Residuals',
     main='Residual plot')
abline(0,0)
```



We see a better fitting model here. We observe that Random forest model gets down the RMSE's to 66.31489 and rmsle value to 0.6376547.

Results

The results of 4 models are shown in the table below. It is clearly shown that the best model in terms of RMSE and rmsle is the Random Forest Model. The RMSE and log RMSE values of all the represented models are the following:

Method	RMSE	rmsle
K Nearest Neighbor(KNN) Model	139.59180	1.3040288
Support Vector Machine (SVM) Model	128.43767	1.1796819
Linear Regression Model	102.84655	0.9549201
Random Forest Model	66.31489	0.6376547

We therefore found the lowest value of RMSE that is 66.31489 and lowest value of rmsle is 0.6376547.

Conclusion

The goal of this project was to develop a best bike sharing count of casual and registered users to predict bike count using bike sharing dataset, therefore machine learning algorithm has been built to predict bike count with this dataset. From the above output, we see that Random Forest works best for our dataset prediction.

The optimal model characterised by the lowest RMSE value (66.31489) and lowest value of rmsle value (0.6376547). The resultant RMSE_results table shows an improvement of the model over different assumptions. The goal is to train a machine learning algorithm using the inputs of a provided training subset to predict bike sharing counts in a validation set. Here, 4 different algorithms are used to predict the accuracy. We applied K Nearest Neighbor(KNN) Model, Support Vector Machine (SVM) Model, Linear Regression Model and Random Forest Model. We found that random forest is performing the best. A deeper insight into the data revealed some data point in the features have large effect on errors. The final RMSE is 66.31489 and rmsle is 0.6376547. This implies we can trust our prediction for bike sharing system.