BIKE RENTING

SHRAVYA SURESH

TABLE OF CONTENTS

CHAPTER 1

Introduction

With the Modern age of all new vehicles, with consume fuel and help us travel from place to place, bicycles still remain as a traditional and eco friendly means of travelling. It literally consumes no fuel. A wide range of people buy bicycle, like the ones who cant afford a motorcycle, the ones who can afford and still prefer to go eco friendly, and the ones who who are fitness freaks. All aged people buy it, specially kids consider it as an essential childhood object.

1.1 Problem Statement

The objective of this case study is the prediction of bike rental count on daily based on the environmental and seasonal settings. The dataset contains 731 observations, 15 predictors and 1 target variable. The predictors are describing various environment factors and settings like season, humidity etc. We need to build a prediction model to predict estimated count or demand of bikes on a particular day based on the environmental factors.

1.2 Dataset

The data set consist of 731 observation recorded over a period of 2 years, between 2011 and 2012. It has 15 predictors or variables and 1 target variable.

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 from Holiday Schedule)
weekday	Day of the week
workingDay	If day is neither weekend nor holiday is 1, otherwise is 0.
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, Light Rain + Scattered clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
Temp	Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale)
Atemp	Normalized feeling temperature in Celsius. The values are derived via (t-t_min)/(t_maxt-

	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 is a target variable which is the total count of users.

The data set consist of 7 continuous and 8 categorical variables. Sample data is shown below.

instant	dteday	season	yr	mnth	holiday	weekday	workingday
1	########	1	0	1	0	6	0
2	########	1	0	1	0	0	0
3	########	1	0	1	0	1	1
4	########	1	0	1	0	2	1

weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
2	0.344167	0.363625	0.805833	0.160446	331	654	985
2	0.363478	0.353739	0.696087	0.248539	131	670	801
1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
1	0.2	0.212122	0.590435	0.160296	108	1454	1562

CHAPTER 2

Methodology

The solution of this problem is divided into three parts. First was EDA (Exploratory Data analysis) and pre-processing, followed by modelling and performance tuning and comparison. During first part data pre-processing step like missing value analysis, outlier analysis, univariate and bi-variate analysis etc. were performed. After that data was split into train and test. The target variable is a continuous variable, so it a regression problem. Linear regression, decision tree and Random forest regression were used for modelling and their performance comparison was performed. Both the algorithms were implemented in R and python.

2.1 Exploratory Data Analysis (EDA)

Exploratory data analysis helps us in understanding the data as well as clearing it and hence cleaning it eventually.

Using the following exploratory data analysis, we have understood the data.

2.1.1 The Target Variable - Churn

2.1.2 Missing Value Analysis

No missing values were present in the given dataset.

Variables	Missing_percentage
instant	0
dteday	0
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
atemp	0
hum	0
windspeed	0
casual	0
registered	0
cnt	0

2.1.3 Multicollinearity

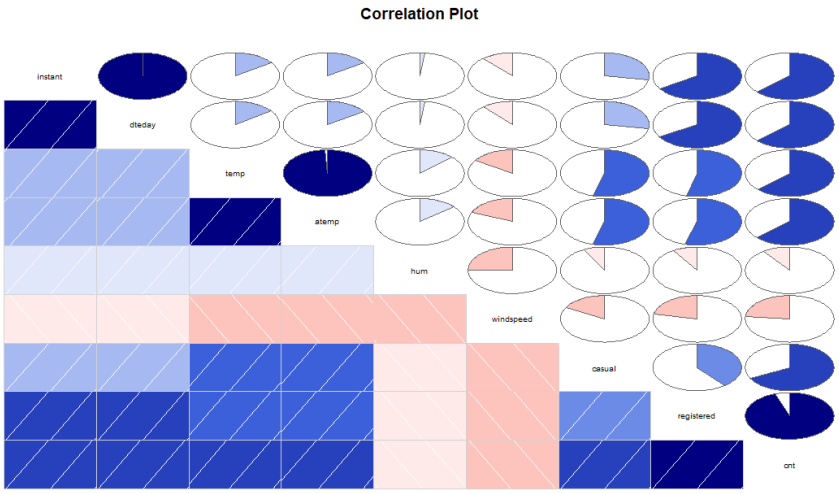
Multicollinearity is basically a feature which defines the close relation between two variables or predictors. We say that two variables are highly collinear when the are very much dependent on each other. This is a problem because, it gives unnecessary variables in our dataset. Hence if two variables are highly collinear, then we can remove one of them, and retain the other one, preferably the one which is more dependent on our target variable. On applying the correlation plot in R and python, we get the following data.

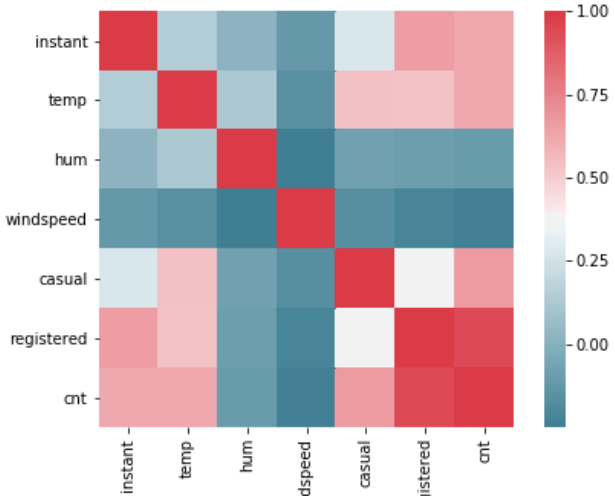
From the correlation plot we can see that –

Temp and Atemp are highly correlated.

Variables are highly collinear are highlighted with red colour with their corresponding score. Apart from correlation matrix, multicollinearity was again checked during logistic regression diagnostics using VIF (Variation inflation factor).

The plot looks as follows,



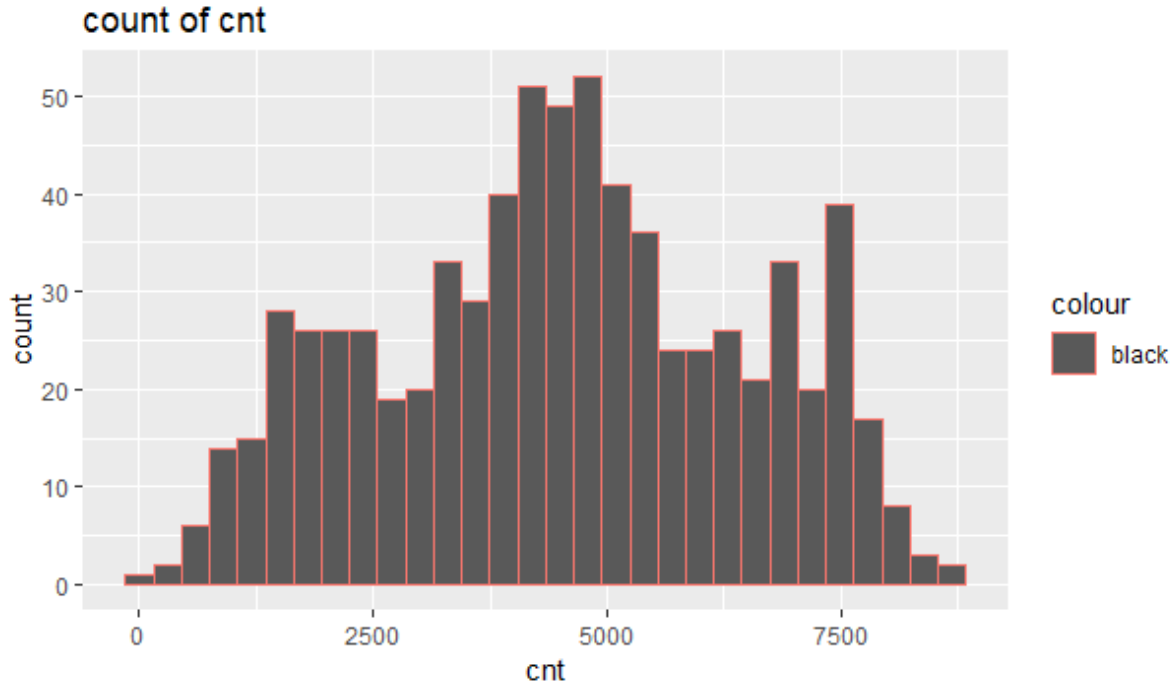


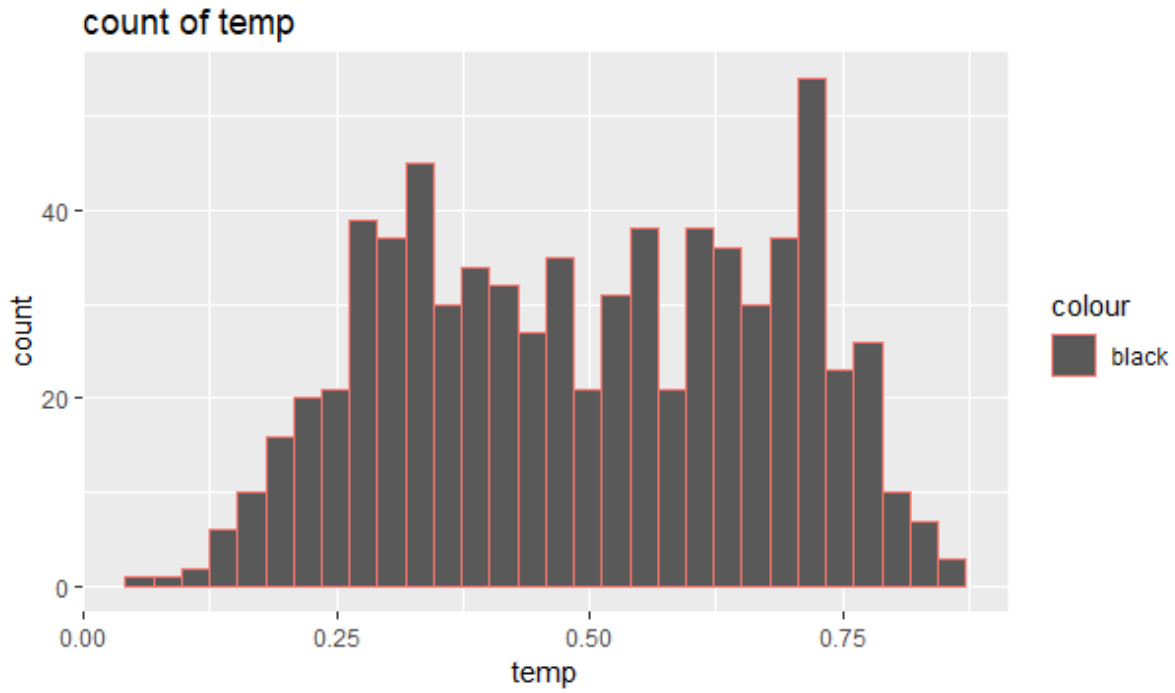
2.1.4 variable reduction

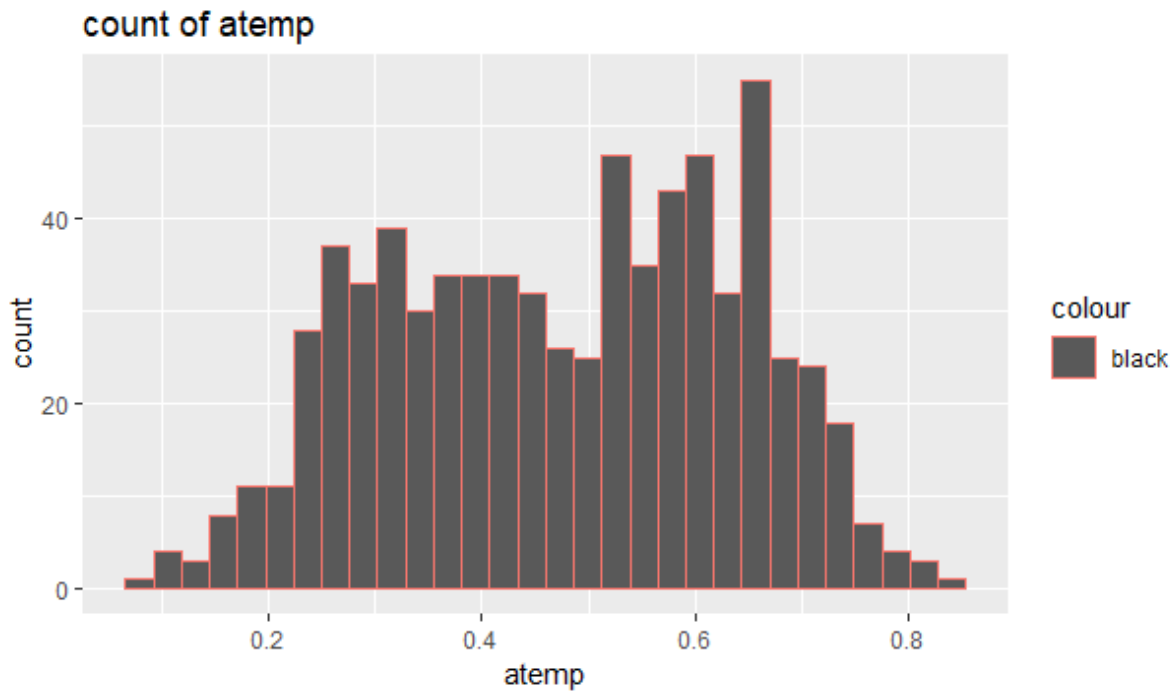
After applying the co relation analysis for continuous variables and also analysing the importance of the variable, we hence reduce the dimention of the dataset by removing the following variables,

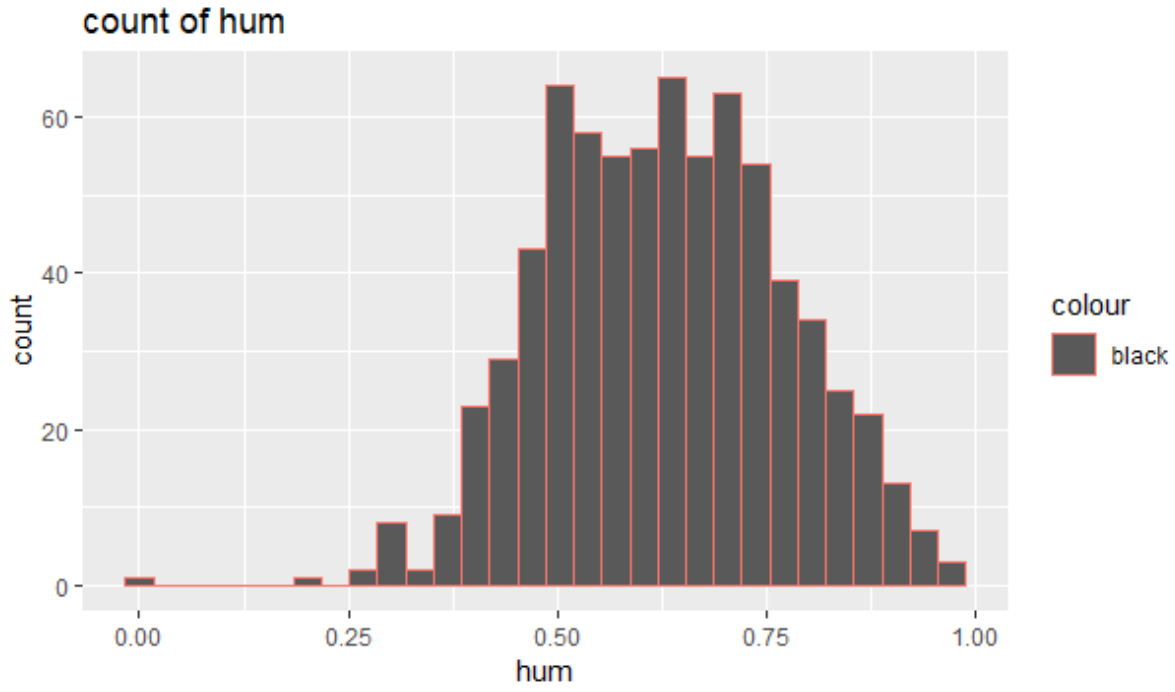
atemp,dteday

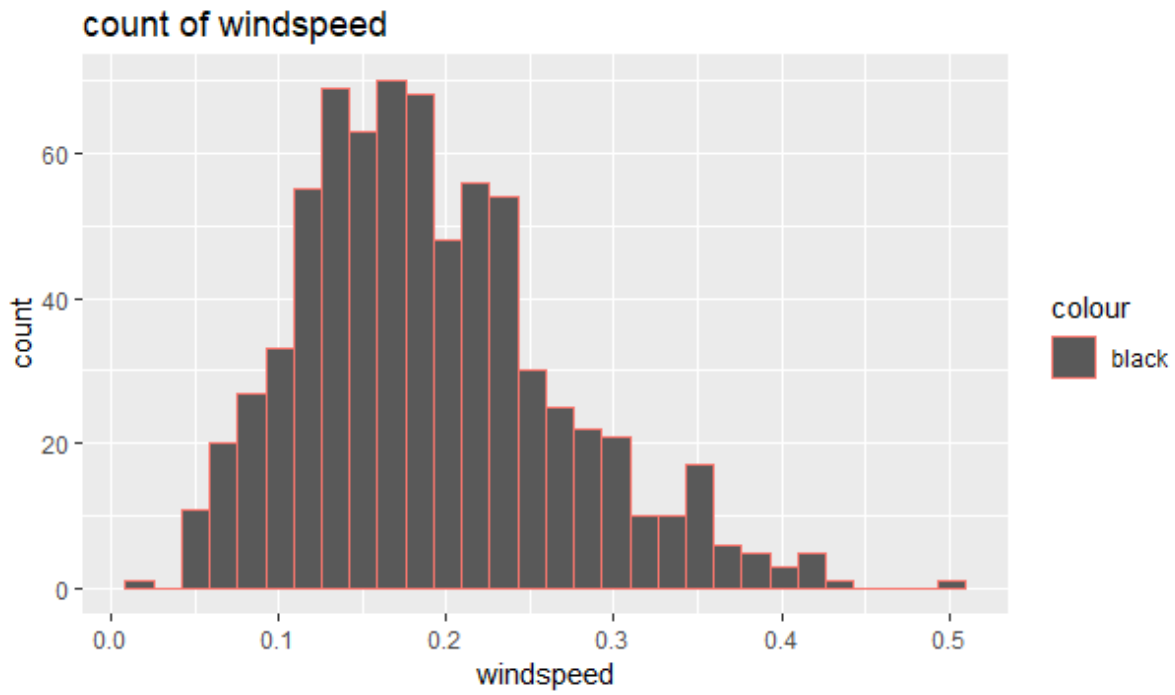
2.1.5 Analysis of cnt ratio with different predictors

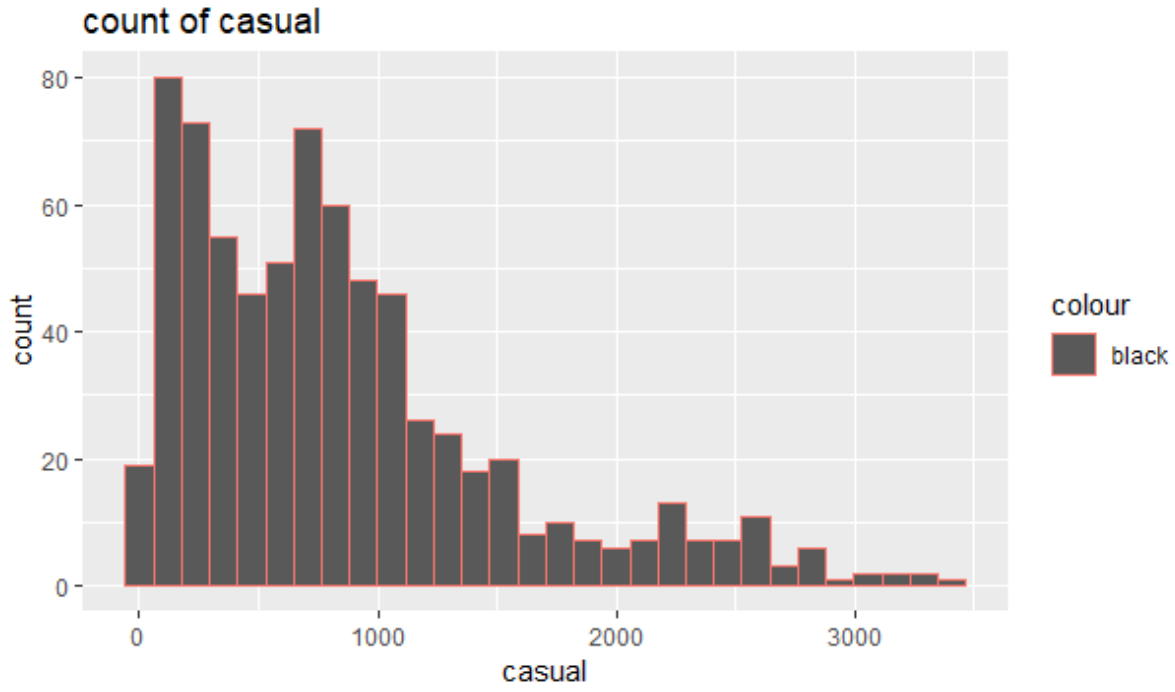


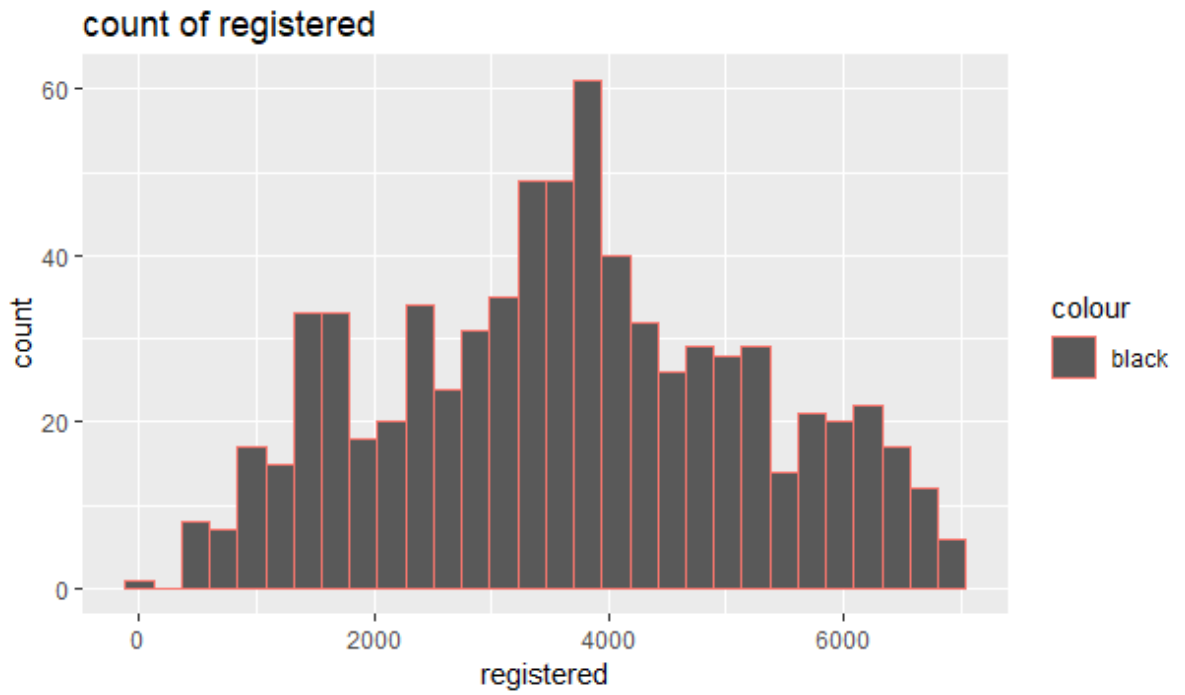


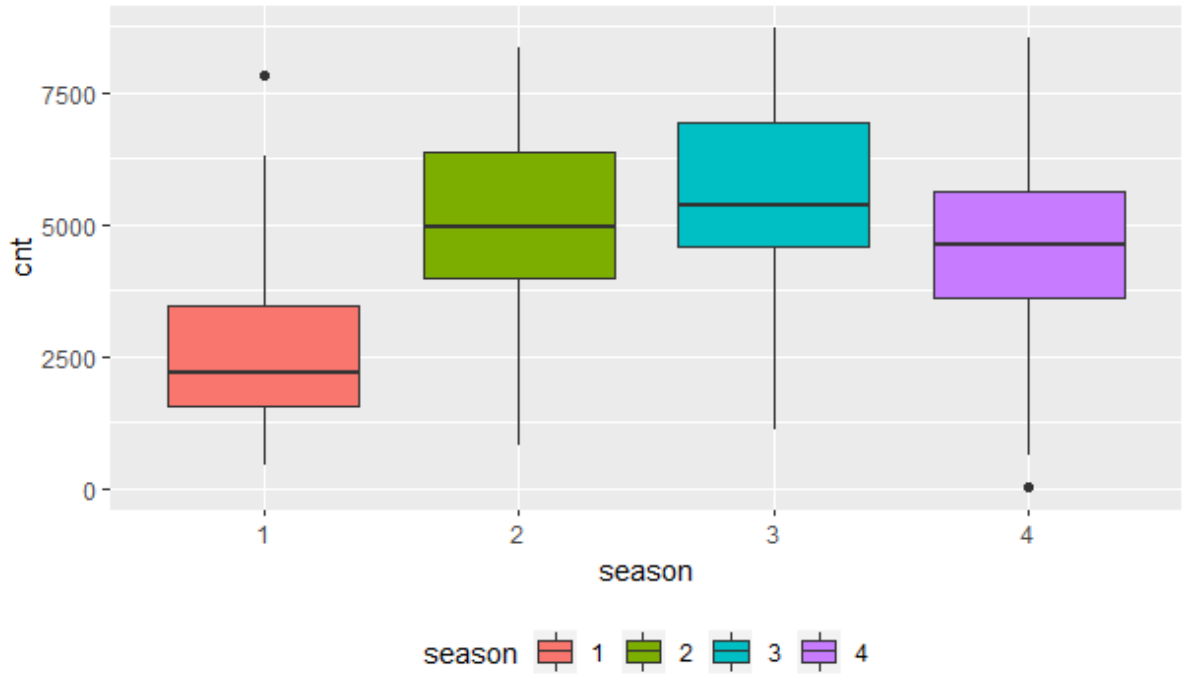

count of atemp
count
40
20
0
0.2
0.4
0.6
0.8
atemp
colour
black

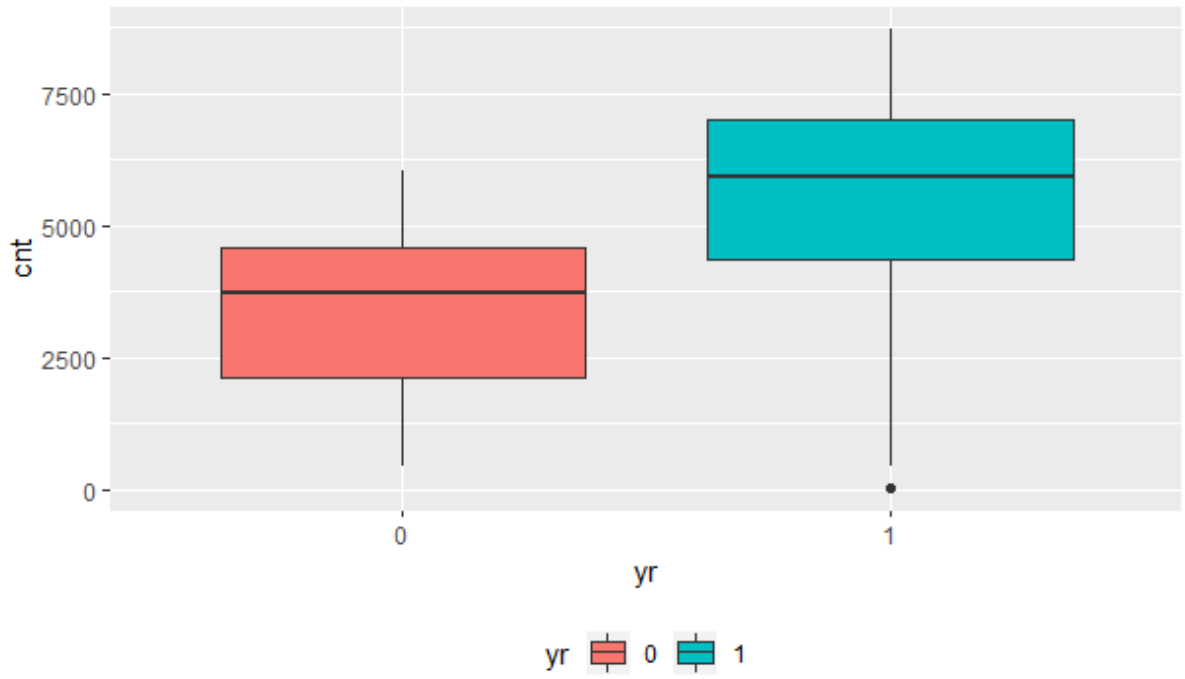

count of hum
count
60
40
20
0
0.00
0.25
0.50
0.75
1.00
hum
colour
black

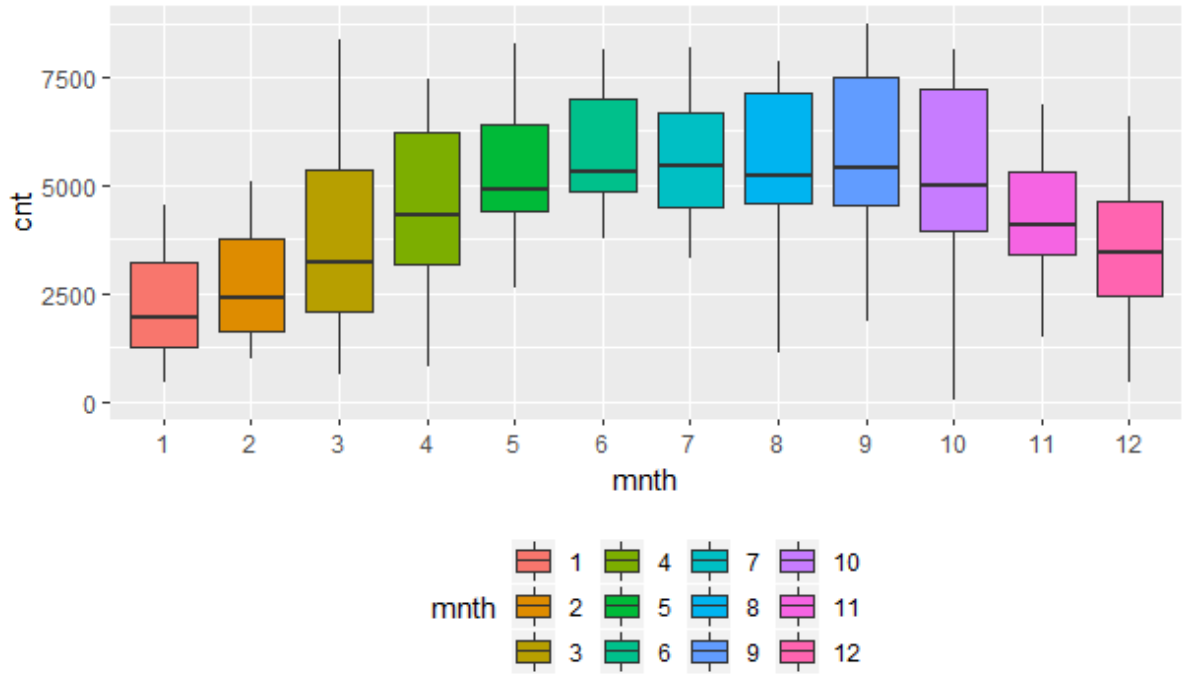

count of windspeed
count
0
20
40
60
0.0
0.1
0.2
0.3
0.4
0.5
windspeed
colour
black

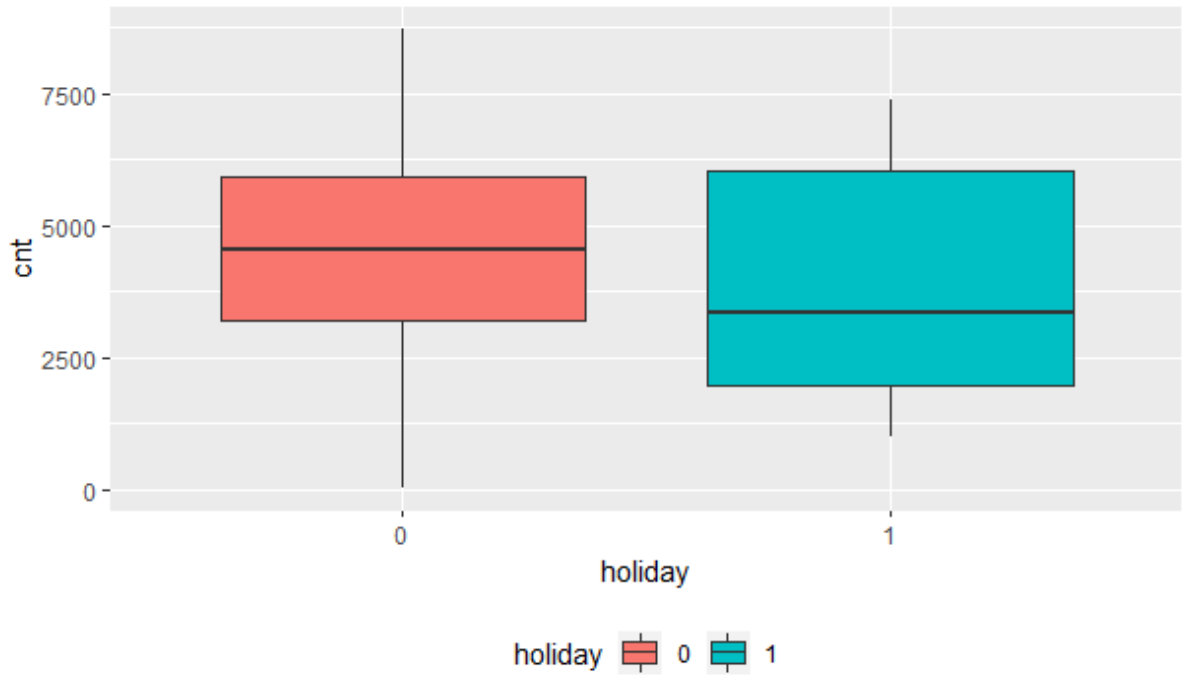

count of casual
count
0
20
40
60
80
0
1000
2000
3000
casual
colour
black

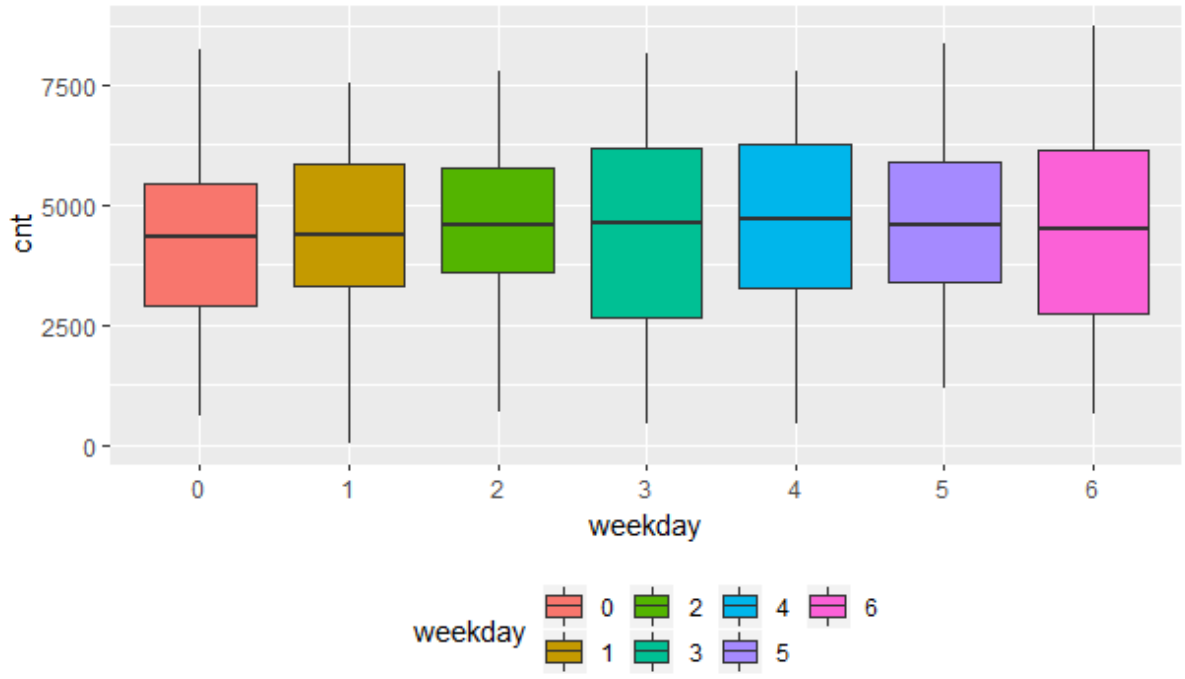

count of registered
count
registered
colour
black
0
20
40
60
0
2000
4000
6000


cnt
season
season 1 2 3 4
0
2500
5000
7500
1
2
3
4


7500
5000
2500
0
cnt
0
1
yr
yr 0 1


7500
5000
2500
0
cnt
1 2 3 4 5 6 7 8 9 10 11 12
mnth
mnth 1 2 3 4 5 6 7 8 9 10 11 12


7500
5000
2500
0
cnt
0
1
holiday
holiday
0
1


7500
5000
2500
0
cnt
0
1
2
3
4
5
6
weekday
weekday
0
1
2
3
4
5
6

2.1.6 Outlier analysis

After missing value analysis, we check for outliers in target variable and predictors. There were no outliers present in the dataset. Some extreme values were present in the predictors but those seems to logical. So no observations were removed and no imputation was performed on the dataset. Boxplot method was used to check for outliers.

2.1.7 Feature Scaling and Normalization

Data normalization is the process of rescaling one or more attributes to the range of [0, 1]. This means largest value of each attribute is 1 and smallest is 0. Normalization is a good technique to use when you know that your data distribution is not Gaussian. Feature scaling was used in the R implementation using MLR package. It was not applied in python for the reason of performance comparison.

2.2 Modelling

Out dataset is a regression model with our target variable as a continuous variable. In bike renting case study, the target variable is continuous in nature. Our task is predicting the bike demand on a single day. This makes it a regression problem. Two machine learning algorithms were used for learning. Both were implemented in R and python.

1. Random forest
2. Linear regression

After applying the various exploratory data analysis and cleaning the data, we apply the following models for evaluation.

2.2.1 Random Forest

Random forest model was first implemented in both R and python. We got satisfactory results using random forest. The code used for Random forest for R and python is given below.

R implementation

In R , the code used was as follows.

The rmse value is, 0.012

Error Rate: 1.34

Accuracy: 99.66

The R code is as follows,

```
###Random Forest
library(randomForest)
RF_model = randomForest(cnt ~ ., train, importance = TRUE, ntree = 1000)
#Extract rules fromn random forest
#transform rf object to an inTrees' format
library(RRF)
library(inTrees)
treeList <- RF2List(RF_model)
#Extract rules
exec = extractRules(treeList, train[,-14])  # R-executable conditions
ruleExec <- extractRules(treeList,train[,-14],digits=4)

##Make rules more readable:
readableRules = presentRules(exec, colnames(train))
readableRules[1:2,]
##Get rule metrics
ruleMetric = getRuleMetric(exec, train[,-14], train$cnt)  # get rule metrics



#Presdict test data using random forest model
```

```
RF_Predictions = predict(RF_model, test[,-14])


MAPE(test[,15], RF_Predictions)


#Error Rate: 1.34
#Accuracy: 99.66
rmse(test$bike_counting, RF_Predictions)
#rmse value is 0.012
```

Python implementation

In Python , the code used was as follows,

The rmse value was as follows, 0.0119

```
#Import Random Forest Model
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier


rf  =  RandomForestRegressor (random_state=12345)
```

In [25]:

```
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(X_train, y_train);
```

In [26]:

```
# Use the forest's predict method on the test data
predictions = rf.predict(X_test)
```

In [27]:

```
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

In [28]:

```
rmse(predictions, y_test)
#rmse value is 0.0119
```

Out[28]:

```
0.011917680784755776
```

2.2.2 Linear regression

Linear regression is a technique in which we try to model a linear relationship with target and predictors.

R implementation

In R , the code used was as follows.

The rmse value is, 10.33

```
Error Rate: 13.41
Accuracy: 88.59
```

```
#linear regression model
#Divide the data into train and test
#set.seed(123)
train_index = sample(1:nrow(bike_counting), 0.8 * nrow(bike_counting))
train = bike_counting[train_index,]
test = bike_counting[-train_index,]
# ##rpart for regression
fit = rpart(cnt ~ ., data = train, method = "anova")
library(rpart)
#Predict for new test cases
predictions_DT = predict(fit, test[,-14])
#MAPE
#calculate MAPE
MAPE = function(y, yhat){
  mean(abs((y - yhat)/y))
}

MAPE(test[,14], predictions_DT)

#Error Rate: 13.41
#Accuracy: 88.59
```

```
rmse(test$bike_counting, predictions_DT)

#rmse value is 10.33
```

Python implementation

In Python , the code used was as follows.

The rmse value is, 3.08

```
# Create linear regression object
regr = linear_model.LinearRegression()


                                                                In [115]:
# Train the model using the training sets
regr.fit(X_train, y_train)
                                                                Out[115]:
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False
)
                                                                In [116]:
# Make predictions using the testing set
y_pred = regr.predict(X_test)
                                                                In [117]:
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(y_test, y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))

Coefficients:
 [-5.52871026e-15 -1.38777878e-16  2.45636844e-15  3.12250226e-16
  4.29703202e-16  8.47303998e-17  3.18813884e-16  6.31005664e-17
 -2.69098979e-16  4.57641737e-16  7.97972799e-17  2.19205824e-01
  4.45486952e-01  4.40922236e-01]
Mean squared error: 0.00
Variance score: 1.00
                                                                In [119]:
rmse(y_pred, y_test)
#rmse value is 3.08

                                                                Out[119]:
3.083111316345951e-16
```

CHAPTER 3

Result and performance measure

The data was pre processed and the result was obtained. This Is the various output obtained.

	R implementation		Python implementation		
Metric	Random forest	Linear regression	Random forest	Linear regression	
Rmse value	0.012	10.33	0.0199	3.08	

3.2 Result

From the error metric we can see that random forest is performing better than linear regression in both implementations. The result for random forest is similar in both R and python. But in case of linear regression, python implementation is performing better than R.

Since the performance of random forest we better, we finally accept the python model of random forest.

Using the various visualizations, we can also conclude the following statements,

1. **The count is highest for fall season and lowest for spring season.**
2. **It can be inferred that count is high in the month of august, September and October. It is lowest count is for January and February.**
3. **People prefer to rent bike on holidays.**
4. **The count is maximum when weather situation is good. It is least when weather conditions are bad.**

Complete R code

```
#remove all the objects stored
rm(list=ls())
#set current working directory
setwd("C:/Users/SHRAVYA/Desktop/edwisor/project 3")
#install packages
install.packages(c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "dummies", "e1071", "Information", "MASS", "rpart", "gbm", "ROSE", "sampling", "DataCombine", "inTrees","dplyr","usdm"))
## Read the data
bike_counting = read.csv("day.csv", header = T, na.strings = c(" ", "", "NA"))
#.....exploratory data analysis.....
#exploratory data analysis
str(bike_counting)
bike_counting$dteday  = as.numeric(bike_counting$dteday )
bike_counting$season  = as.factor(bike_counting$season )
bike_counting$yr  = as.factor(bike_counting$yr )
bike_counting$mnth  = as.factor(bike_counting$mnth )
bike_counting$holiday  = as.factor(bike_counting$holiday )
bike_counting$weekday  = as.factor(bike_counting$weekday )
bike_counting$workingday  = as.factor(bike_counting$workingday )
bike_counting$weathersit  = as.factor(bike_counting$weathersit )
#.....missing value analysis.....


missing_val = data.frame(apply(bike_counting,2,function(x){sum(is.na(x))}))
```

```
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(bike_counting)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
write.csv(missing_val, "Miising_perc.csv", row.names = F)
#no missing values found
#ggplot analysis
ggplot(data = missing_val[1:3,], aes(x=reorder(Columns, -Missing_percentage),y = Missing_percentage))+
  geom_bar(stat = "identity",fill = "grey")+xlab("Parameter")+
  ggtitle("Missing data percentage (Train)") + theme_bw()
library(ggplot2)
numeric_index = sapply(bike_counting,is.numeric) #selecting only numeric


numeric_data = bike_counting[,numeric_index]



cnames = colnames(numeric_data)
# ............................. Outlier analysis .....................................#


outlierKD <- function(dt, var) {
  var_name <- eval(substitute(var), eval(dt))
```

```
na1 <- sum(is.na(var_name))
m1 <- mean(var_name, na.rm = T)
par(mfrow = c(2, 2), oma = c(0, 0, 3, 0))
boxplot(var_name, main = "With outliers")
hist(var_name,
     main = "With outliers",
     xlab = NA,
     ylab = NA)
outlier <- boxplot.stats(var_name)$out
mo <- mean(outlier)
var_name <- ifelse(var_name %in% outlier, NA, var_name)
boxplot(var_name, main = "Without outliers")
hist(var_name,
     main = "Without outliers",
     xlab = NA,
     ylab = NA)
title("Outlier Check", outer = TRUE)
na2 <- sum(is.na(var_name))
cat("Outliers identified:", na2 - na1, "n")
cat("Propotion (%) of outliers:", round((na2 - na1) / sum(!is.na(var_name)) *
                                        100, 1), "n")
cat("Mean of the outliers:", round(mo, 2), "n")
m2 <- mean(var_name, na.rm = T)
cat("Mean without removing outliers:", round(m1, 2), "n")
cat("Mean if we remove outliers:", round(m2, 2), "n")
```

```
}



outlierKD(bike_counting, temp) #no outliers
outlierKD(bike_counting, atemp) #no outliers
outlierKD(bike_counting, hum) # no extreme outlier detected
outlierKD(bike_counting, windspeed) #some extreme values are present but canot be considered as outlier
outlierKD(bike_counting, casual) # no logical outliers
outlierKD(bike_counting, registered)# no ouliers
outlierKD(bike_counting, cnt)# no ouliers
#.....Univariate Analysis .....
# 1. Continous predictors
univariate_continuous <- function(dataset, variable, variableName) {
  var_name = eval(substitute(variable), eval(dataset))
  print(summary(var_name))
  ggplot(data = dataset, aes(var_name)) +
    geom_histogram(aes(binwidth = .5, colour = "black")) +
    labs(x = variableName) +
    ggtitle(paste("count of", variableName))
}

univariate_continuous(bike_counting, cnt, "cnt")
univariate_continuous(bike_counting, temp, "temp")
```

```
univariate_continuous(bike_counting, atemp, "atemp")
univariate_continuous(bike_counting, hum, "hum") # skwed towards left
univariate_continuous(bike_counting, windspeed, "windspeed") #skewed towards right
univariate_continuous(bike_counting, casual, "casual") # skwed towards right
univariate_continuous(bike_counting, registered, "registered")
#2. categorical variables
univariate_categorical  <- function(dataset, variable, variableName) {
 variable <- enquo(variable)

 percentage <- dataset +
  dplyr::select(!!variable) +
  group_by(!!variable) +
  summarise(n = n()) +
  mutate(percantage = (n / sum(n)) * 100)
 print(percentage)

 dataset +
  count(!!variable) +
  ggplot(mapping = aes_(
   x = rlang::quo_expr(variable),
   y = quote(n),
   fill = rlang::quo_expr(variable)
  )) +
  geom_bar(stat = 'identity',
```

```
            colour = 'white') +
    labs(x = variableName, y = "count") +
    ggtitle(paste("count of ", variableName)) +
    theme(legend.position = "bottom") -> p
  plot(p)
}

univariate_categorical(bike_counting, season, "season")
univariate_categorical(bike_counting, yr, "yr")
univariate_categorical(bike_counting, mnth, "mnth")
univariate_categorical(bike_counting, holiday, "holiday")
univariate_categorical(bike_counting, weekday, "weekday")
univariate_categorical(bike_counting, workingday, "workingday")
univariate_categorical(bike_counting, weathersit, "weathersit")
#.....bivariate analysis.....

#.....# bivariate analysis for categorical variables
bivariate_categorical <-
  function(dataset, variable, targetVariable) {
    variable <- enquo(variable)
    targetVariable <- enquo(targetVariable)

    ggplot(
      data = dataset,
      mapping = aes_(
```

```
      x = rlang::quo_expr(variable),
      y = rlang::quo_expr(targetVariable),
      fill = rlang::quo_expr(variable)
    )
  ) +
    geom_boxplot() +
    theme(legend.position = "bottom") -> p
  plot(p)

}

bivariate_continous <-
 function(dataset, variable, targetVariable) {
  variable <- enquo(variable)
  targetVariable <- enquo(targetVariable)
  ggplot(data = dataset,
         mapping = aes_(
           x = rlang::quo_expr(variable),
           y = rlang::quo_expr(targetVariable)
         )) +
    geom_point() +
    geom_smooth() -> q
  plot(q)

}
```

```
bivariate_categorical(bike_counting, season, cnt)
bivariate_categorical(bike_counting, yr, cnt)
bivariate_categorical(bike_counting, mnth, cnt)
bivariate_categorical(bike_counting, holiday, cnt)
bivariate_categorical(bike_counting, weekday, cnt)
bivariate_categorical(bike_counting, workingday, cnt)
bivariate_categorical(bike_counting, weathersit, cnt)

bivariate_continous(bike_counting, temp, cnt)
bivariate_continous(bike_counting, atemp, cnt)
bivariate_continous(bike_counting, hum, cnt)
bivariate_continous(bike_counting, windspeed, cnt)
bivariate_continous(bike_counting, casual, cnt)
bivariate_continous(bike_counting, registered, cnt)
#.....................feature selection.....................


library(corrgram)
## Correlation Plot
corrgram(bike_counting[,numeric_index], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")
#atemp and temp are highly correlated.
#.....................................feature reduction.....................................
```

```
bike_counting = subset(bike_counting, select = -c(atemp,dteday))


#Feature Scaling
#Normality check
qqnorm(bike_counting$cnt )
hist(bike_counting$cnt )
str(bike_counting)
#the variables casual,registered,and count are not much skwed but the other variables are highly skwed, hence we apply normalization.
#Normalisation
cnames = c("temp","hum","windspeed","casual","registered","cnt")


for(i in cnames){
  print(i)
  bike_counting[,i] = (bike_counting[,i] - min(bike_counting[,i]))/
    (max(bike_counting[,i] - min(bike_counting[,i])))
}
#.....................................Sampling.....................................
.........
##Systematic sampling
#Function to generate Kth index
sys.sample = function(N,n)
{
  k = ceiling(N/n)
  r = sample(1:k, 1)
```

```
  sys.samp = seq(r, r + k*(n-1), k)
}
lis = sys.sample(731, 300) #select the repective rows
# #Create index variable in the data
bike_counting$index = 1:731
# #Extract subset from whole data
systematic_data = bike_counting[which(bike_counting$index %in% lis),]
#.....Model
Development.....#

#Clean the environment
library(DataCombine)
rmExcept("bike_counting")
#Divide data into train and test using stratified sampling method
set.seed(1234)
bike_counting$description = NULL
library(caret)
train.index = createDataPartition(bike_counting$cnt, p = .80, list = FALSE)
train = bike_counting[ train.index,]
test  = bike_counting[-train.index,]
#load libraries
#rpart for regression
library(rpart)
fit = rpart(cnt ~ ., data = train, method = "anova")
```

```
#Predict for new test cases
predictions_DT = predict(fit, test[,-14])


#MAPE
#calculate MAPE
MAPE = function(y, yhat){
  mean(abs((y - yhat)/y))
}

MAPE(test[,15], predictions_DT)


#Error Rate: 1.35
#Accuracy: 99.65
#rmse calculation
install.packages("Metrics")
library(Metrics)
rmse(test$bike_counting, predictions_DT)
#rmse value is 0.021
###Random Forest
library(randomForest)
RF_model = randomForest(cnt ~ ., train, importance = TRUE, ntree = 1000)
#Extract rules fromn random forest
#transform rf object to an inTrees' format
library(RRF)
library(inTrees)
```

```
treeList <- RF2List(RF_model)
#Extract rules
exec = extractRules(treeList, train[,-14])  # R-executable conditions
ruleExec <- extractRules(treeList,train[,-14],digits=4)

##Make rules more readable:
readableRules = presentRules(exec, colnames(train))
readableRules[1:2,]
##Get rule metrics
ruleMetric = getRuleMetric(exec, train[,-14], train$cnt)  # get rule metrics



#Presdict test data using random forest model
RF_Predictions = predict(RF_model, test[,-14])

MAPE(test[,15], RF_Predictions)

#Error Rate: 1.34
#Accuracy: 99.66
rmse(test$bike_counting, RF_Predictions)
#rmse value is 0.012
#linear regression model
#Divide the data into train and test
#set.seed(123)
train_index = sample(1:nrow(bike_counting), 0.8 * nrow(bike_counting))
```

```
train = bike_counting[train_index,]
test = bike_counting[-train_index,]
# ##rpart for regression
fit = rpart(cnt ~ ., data = train, method = "anova")
library(rpart)
#Predict for new test cases
predictions_DT = predict(fit, test[,-14])
#MAPE
#calculate MAPE
MAPE = function(y, yhat){
  mean(abs((y - yhat)/y))
}

MAPE(test[,14], predictions_DT)

#Error Rate: 13.41
#Accuracy: 88.59
rmse(test$bike_counting, predictions_DT)
#rmse value is 10.33
```

Complete python code

```
#Load libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
import seaborn as sns
from random import randrange, uniform
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
```

```
C:\Users\SHRAVYA\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41
: DeprecationWarning: This module was deprecated in version 0.18 in favor o
f the model_selection module into which all the refactored classes and func
tions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
#Set working directory
os.chdir("C:/Users/SHRAVYA/Desktop/edwisor/project 3")
```

In [3]:

```
#Load data
bike_counting = pd.read_csv("day.csv")
```

In [4]:

```
#----------------------------------PRE PROCESSING-EXPLORATORY DATA ANALYSIS
--------------------------------------------------#
```

In [5]:

```
#Exploratory Data Analysis

#converting to categorical variable#conver

categorical_variable = ["season","yr","mnth","holiday","weekday","workingda
y","weathersit"]

for i in categorical_variable:
    bike_counting[i] = bike_counting[i].astype("category")
bike_counting.dtypes
```

```
instant          int64
dteday          object
season        category
yr            category
mnth          category
holiday       category
weekday       category
```

```
workingday     category
weathersit     category
temp            float64
atemp           float64
hum             float64
windspeed       float64
casual            int64
registered        int64
cnt               int64
dtype: object
```

In [6]:

```
#--------------------------------------MISSING VALUE ANALYSIS-------------
-------------------------------------------------------#
```

In [7]:

```
#Create dataframe with missing percentage
missing_val = pd.DataFrame(bike_counting.isnull().sum())

#Reset index
missing_val = missing_val.reset_index()

#Rename variable
missing_val = missing_val.rename(columns = {'index': 'Variables', 0: 'Missi
ng_percentage'})

#Calculate percentage
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(
bike_counting))*100
```

731 rows × 16 columns

In [10]:

```
#---------------------------------------FEATURE SELECTION-----------------
-------------------------------------------------#
```

In [11]:

```
#Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(7, 5))

#Generate correlation matrix
corr = df_corr.corr()

#Plot using seaborn library
sns.heatmap(corr, mask=np.zeros_like(corr, dtype=np.bool), cmap=sns.divergi
ng_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
plt.savefig('correlation.png')
```

In [12]:

```
#dimension reduction
#feature reduction
bike_counting = bike_counting.drop(['atemp','dteday'], axis=1)
```

In [13]:

```
#Chisquare test of independence
#Save categorical variables
cat_names = ["season", "yr", "mnth", "holiday", "weekday", "workingday", "w
eathersit"]
```

In [14]:

```
#feature reduction

bike_counting.shape
```

Out[14]:

```
(731, 14)
```

In [15]:

```
#-----------------------------------------------FEATURE SCALING-------------
-------------------------------------------------#
```

In [16]:

```
#Nomalisation
#the variables casual,registered,and count are not much skwed but the other
variables are highly skwed, hence we apply normalization
for i in cnames:
    print(i)
    bike_counting[i] = (bike_counting[i] - min(bike_counting[i]))/(max(bike
_counting[i]) - min(bike_counting[i]))
```

```
instant
temp
hum
windspeed
casual
registered
cnt
```

In [17]:

```
#-----------------------------------------------MODELLING-------------------
-------------------------------------------------#
```

In [18]:

```
#Divide data into train and test
train, test = train_test_split(bike_counting, test_size=0.25, random_state=
42)
bike_counting.shape
```

Out[18]:

```
(731, 14)
```

In [19]:

```
# Decision Tree
```

In [20]:
```
#Decision tree for regression
fit_DT = DecisionTreeRegressor(max_depth=2).fit(train.iloc[:,0:13], train.i
loc[:,13])
```

In [21]:
```
#Apply model on test data
predictions_DT = fit_DT.predict(test.iloc[:,0:13])
```

In [22]:
```
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
rmse(test.iloc[:,13], predictions_DT)
#rmse value=0.0832
```

Out[22]:
```
0.08326961072062442
```

In [23]:
```
#Divide data into train and test
X = bike_counting.values[:, 0:13]
Y = bike_counting.values[:,13]

X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.2)
```

In [24]:
```
#Import Random Forest Model
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier


rf  =  RandomForestRegressor (random_state=12345)
```

In [25]:
```
# Import the model we are using
from sklearn.ensemble import RandomForestRegressor
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(X_train, y_train);
```

In [26]:
```
# Use the forest's predict method on the test data
predictions = rf.predict(X_test)
```

In [27]:
```
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

In [28]:
```
rmse(predictions, y_test)
#rmse value is 0.0119
```

Out[28]:

```
0.011917680784755776
```

In [27]:

```
#linear regression model
```

In [28]:

```
fit = rpart(cnt~ ., data = X_train, method = "anova")
```

```
  File "<ipython-input-28-9c242b60677a>", line 1
    fit = rpart(cnt~ ., data = X_train, method = "anova")
                    ^
SyntaxError: invalid syntax
```

In [29]:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
```

In [113]:

```
# Split the data into training/testing sets
X_train = bike_counting[:-13]
X_test = bike_counting[-13:]
y_train = bike_counting.cnt[:-13]
y_test = bike_counting.cnt[-13:]
```

In [114]:

```
# Create linear regression object
regr = linear_model.LinearRegression()
```

In [115]:

```
# Train the model using the training sets
regr.fit(X_train, y_train)
```

Out[115]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False
)
```

In [116]:

```
# Make predictions using the testing set
y_pred = regr.predict(X_test)
```

In [117]:

```
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(y_test, y_pred))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))
```

```
Coefficients:
 [-5.52871026e-15 -1.38777878e-16  2.45636844e-15  3.12250226e-16
  4.29703202e-16  8.47303998e-17  3.18813884e-16  6.31005664e-17
```

```
 -2.69098979e-16  4.57641737e-16  7.97972799e-17  2.19205824e-01
  4.45486952e-01  4.40922236e-01]
Mean squared error: 0.00
Variance score: 1.00
```

In [119]:

```
rmse(y_pred, y_test)
#rmse value is 3.08
```

Out[119]:

```
3.083111316345951e-16
```