Bike Renting

Hitesh Juneja

31 December 2018

Contents

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

The objective of this project is to count the number of bike hired on rent on daily basis. We are provided with the seasonal and environmental condition by using which we have to predict the count of bike on the particular day. We are provided with the data of year 2011 and 2012 with the count of bike rented on each day of the year.

1.2 Data

Our task is to build the regression model which will predict the rental bike count on daily basis based on multiple predictor provided in the problem statement. Given below is the data set which we will be using for predicting the values of target variable ('cnt') of test dataset.

Table 1.1 : Bike Rental Sample Data (Columns: 1-10)

instant	dteday	season	yr	mnth	holiday	weekday	working day	weathersit	temp
1	2011-01-01	1	0	1	0	6	0	2	0.344167
2	2011-01-02	1	0	1	0	0	0	2	0.363478
3	2011-01-03	1	0	1	0	1	1	1	0.196364
4	2011-01-04	1	0	1	0	2	1	1	0.2
5	2011-01-05	1	0	1	0	3	1	1	0.226957

Table 1.2 : Bike Rental Sample Data (Columns: 11-16)

atemp	hum	windspeed	casual	registered	cnt
0.363625	0.805833	0.160446	331	654	985
0.353739	0.696087	0.248539	131	670	801
0.189405	0.437273	0.248309	120	1229	1349
0.212122	0.590435	0.160296	108	1454	1562
0.22927	0.436957	0.1869	82	1518	1600

As we can see above that the table have 13 predictor variables, using which we have to predict the count of bike rented on daily basis. The 'casual' and 'registered' variables simply tell us that how many casual user and how many registered user have rented the bike and cnt variable tell us about count of total rental bikes including both casual and registered, so these variables cannot be considered as the predictor variables.

Table 1.3 : Predictor variables

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

CHAPTER 2

METHODOLOGY

2.1 Pre Processing

We should always look at the data before preparing the model. However, in data mining terms looking at data refers to much more than just looking. Looking at the data means exploring the data, cleaning the data and visualizing the data through graphs and plot. This is often called **Exploratory Data Analysis**.

Looking at the data we can see that the **'instant'** and **'dteday'** feature is not contributing much while calculating the count of rental bike, since both the feature contain unique values, that means we can remove this feature. Also, we can remove the **'casual'** and **'registered'** feature from the data set because **'cnt'** feature is the sum of both the feature, hence this two feature are correlated to the 'cnt' and can be removed.

2.1.1 Missing Value Analysis

Before preceding, we must check our data set for the missing values. If there are missing values available in our data set then we must impute those values for getting better result. According to industry standards, if the data set have less than 30 % missing values than we must impute values. There are different techniques which can be used to impute missing values in data set.
Below you can see the missing values analysis of the provided data set.

```
### --------- Missing Value Analysis
missing_value = data.frame(apply(bike_data_copy, 2, function(x){sum(is.na(x))}))
```

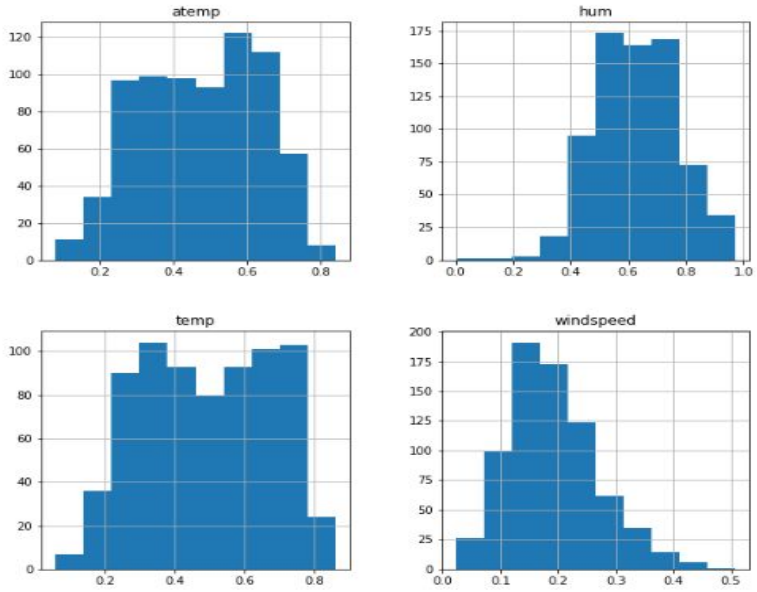
	apply.bike_data_copy..2..function.x...
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
atemp	0
hum	0
windspeed	0
cnt	0

As we can see in above pictures that there are no missing values present in the data set, that means we can proceed further to the next step of Exploratory Data Analysis.

2.1.2 Outlier Analysis

As we can see in the Figure 2.1 the 'hum' and 'windspeed' predictors are skewed. The skew in the distribution is mostly the presence of outliers and extreme values in data.

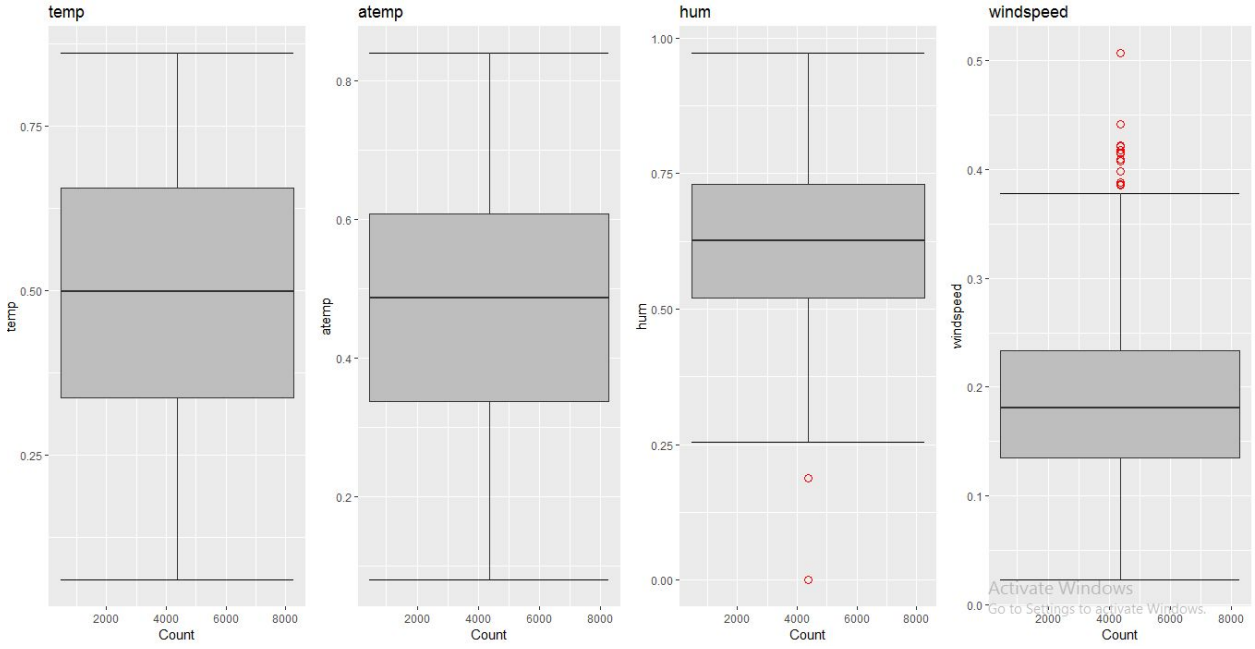
Fig 2.1 Histogram of Numeric Predictors



One of the other step of preprocessing apart from the missing values analysis is presence of outliers. An outlier is an observation that lies an abnormal distance from other values in random sample from the population. In this case we can use one of the classic approach of removing outliers, Tukey's method. We can visualize the outliers using the boxplot. We can only plot the boxplot of numeric variables with the target variable. In our data set we have only four numeric features, hence we can only plot boxplot of these feature.

In Figure 2.2 we have plotted the box plot of the four feature with respect to the count variable. A lot of useful inferences can be made from these plots.

Figure 2.2 Boxplot of Numeric Feature for Outlier Analysis



As we can see in the above figure that there are some outliers present in the numeric data. We have to remove these outliers to bring our data in the proper shape. We can remove the outliers and impute the nearest value in place of outliers, in our case we have impute the values with floor and ceiling of boxplot of feature. Below is the line of code which we have used to impute the values at outlier.

```r
for(i in cnames){
  print(i)
  percentiles = quantile(bike_data_copy[,i],c(0.75,0.25))
  q75= percentiles[[1]]
  q25= percentiles[[2]]
  iqr = q75-q25
  min <- q25 - (1.5*iqr)
  max <- q75 + (1.5*iqr)
  bike_data_copy[,i][bike_data_copy[,i] < min]= min
  bike_data_copy[,i][bike_data_copy[,i] > max]= max
}
```

After performing the outlier analysis and removing outliers using Tukey's Boxplot method. We can plot the histogram and boxplot of the numeric variables again and check how the distribution look. Figure 2.3 and Figure 2.4 show the histogram and boxplot of numeric data after removing the outliers.

Figure 2.3 Boxplot of Numeric Feature after removing outliers.

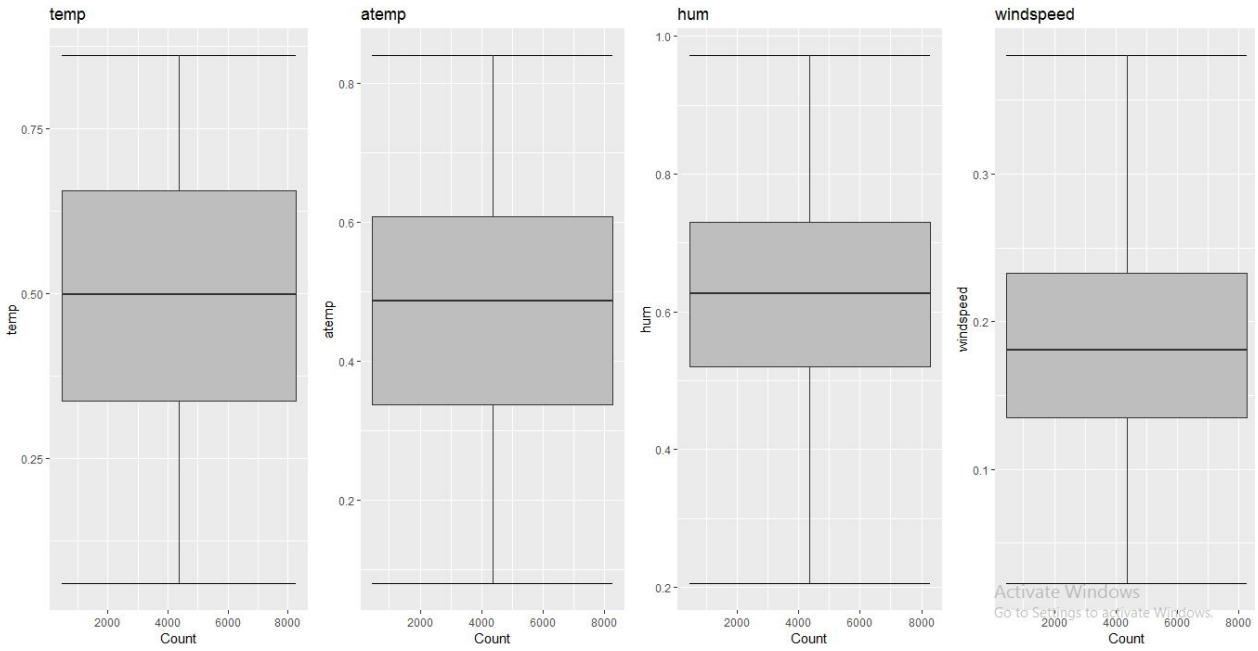
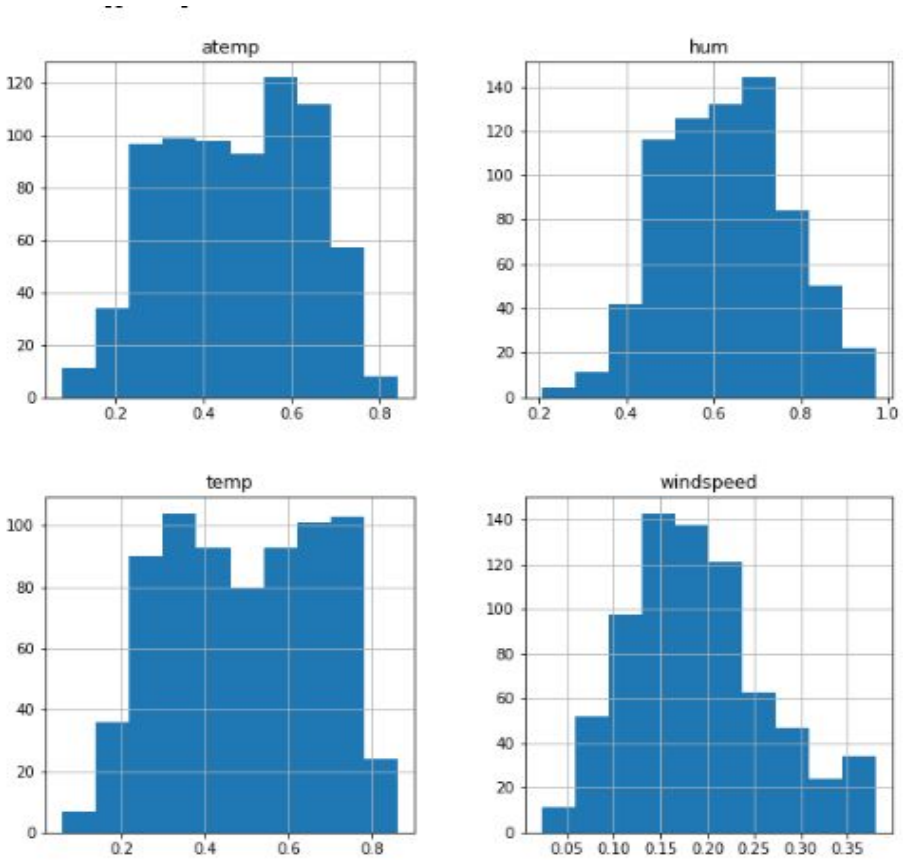


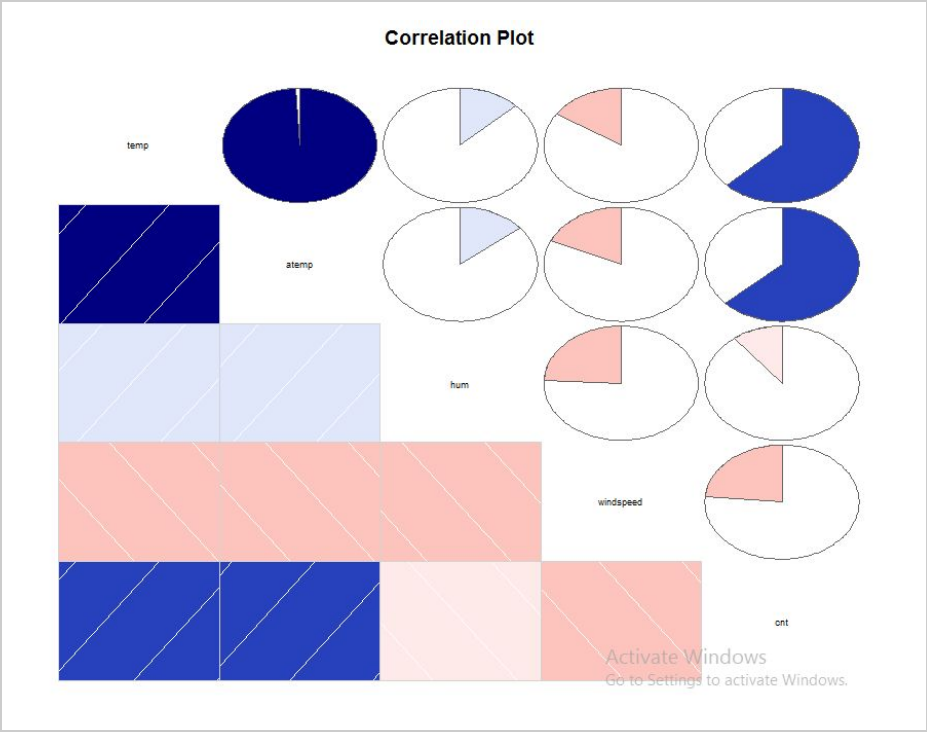
Figure 2.4 Histogram of Numeric Feature after removing outliers



2.1.3 Feature Selection (Variable Reduction)

Before performing any type of modeling we need to check the importance of each predictor variable present in our analysis. There is a possibility that many variables are not at all important to the problem of prediction. The process of selecting only important features from our data set is called Feature Selection or Variable Reduction. There are several method for selecting feature from the data set. Below we have use Correlation method for Numeric Feature Selection and Variance Inflation Factor.

Figure 2.5 Correlation Plot for Feature Selection



As we can see above in correlation plot that **'atemp'** predictor is highly correlated with **'temp'** variable. So we can remove any, either remove 'atemp' or' temp' predictor from our data set.

Variance Inflation Factor for Feature Selection

	Variables	VIF
1	season	3.547569
2	yr	1.021592
3	mnth	3.335362
4	holiday	1.083103
5	weekday	1.024364
6	workingday	1.076185
7	weathersit	1.793275
8	temp	63.413424
9	atemp	64.457410
10	hum	1.962168
11	windspeed	1.198374

> |

After finding the variance inflation factor for all the predictors , we can see that ‘atemp’ and ‘temp’ have **high vif**, that means that these features are highly correlated with each other that means one of them can be removed.

After performing Correlation Analysis and Variance Inflation Factor Analysis, we came to know that ‘atemp’ feature can be removed from the data set because it is dependent on the ‘temp’ variable.

2.1.4 Feature Scaling

Feature Scaling is the method which is used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization. As we can see in the problem statement that all the independent features are already normalized, that is why we don’t have to perform normalization in our dataset.

2.2 Modeling

2.2.1 Model Selection

The independent variables can fall in either of the categories :

1. Nominal
2. Ordinal
3. Interval
4. Ratio

If the dependent variable, in our case ‘cnt’, is Interval or Ratio then the normal method is to do a Regression Analysis, or Classification after binning. If the dependent variable is Ordinal, then both regression and classification can be done. If dependent variable is Nominal the only predictive analysis that we can perform is Classification. In our case the dependent variable is continuous that means we can do Regression Analysis.

There are lot of machine learning algorithm available which can be used to solve the regression problem. Few of them are Decision Tree, Random Forest, Linear Regression, XGBoost etc. We

can use any machine learning algorithm to predict the values of target variable. We will use multiple algorithm to find the values of target variable. And, then after calculating the values of target variable, we will select only that model which have the highest accuracy.

2.2.2 Decision Tree

It is predictive model based on the branching series of boolean test. Decision Tree is basically used to create the predictive model which can be used to find value of target variable by using a decision rules which are collected from the past. Output of the decision tree is simple business rules. There are different algorithm present in the Decision Tree like rpart, C5.0, CART etc.

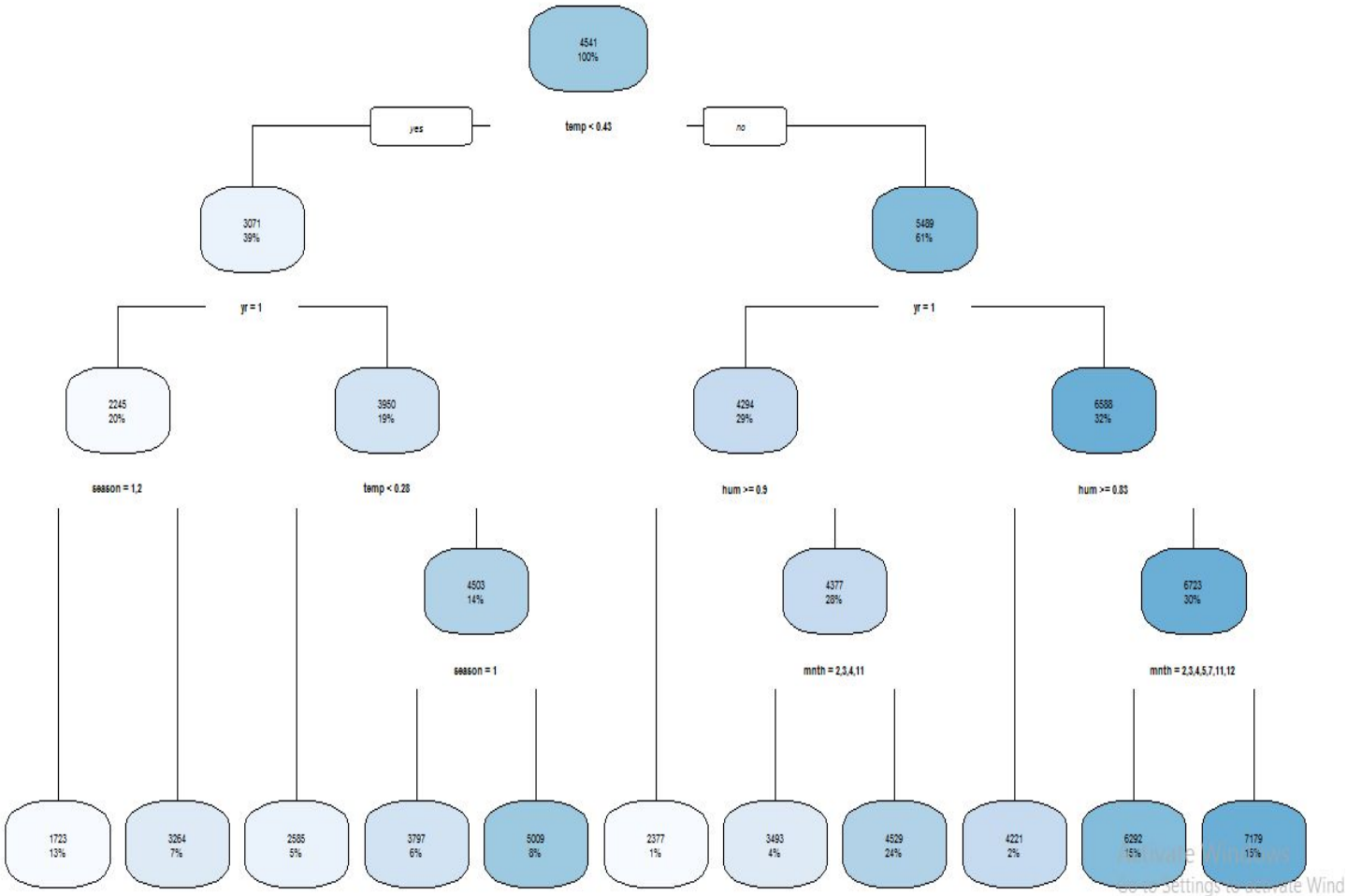
The output of the decision tree is alway in form of rules, these rules are then applied on the test data to predict the values of target variables.

```
> ### -------- Rpart for regreesion
> dt_model = rpart(cnt~.,data=train_data,method = 'anova')
> rpart.plot(dt_model)
> rpart.rules(dt_model)
  cnt
 1723 when temp <  0.43         & yr is 1                                                    & season is 1 or 2
 2377 when temp >=         0.43 & yr is 1 & hum >= 0.90
 2585 when temp <  0.28         & yr is 2
 3264 when temp <  0.43         & yr is 1                                                    & season is      4
 3493 when temp >=         0.43 & yr is 1 & hum <  0.90 & mnth is                2 or 3 or 4 or 11
 3797 when temp is 0.28 to 0.43 & yr is 2                                                    & season is      1
 4221 when temp >=         0.43 & yr is 2 & hum >= 0.83
 4529 when temp >=         0.43 & yr is 1 & hum <  0.90 & mnth is       5 or 6 or 7 or 8 or 9 or 10
 5009 when temp is 0.28 to 0.43 & yr is 2                                                    & season is 2 or 4
 6292 when temp >=         0.43 & yr is 2 & hum <  0.83 & mnth is 2 or 3 or 4 or 5 or 7 or 11 or 12
 7179 when temp >=         0.43 & yr is 2 & hum <  0.83 & mnth is                6 or 8 or 9 or 10
> predict_dt = predict(dt_model,test_data[,-11])
> Acc(test_data[,11],predict_dt)
[1] "Mean Absolute Error : 599.49"
[1] "Mean Absolute Percentage Error : 21.98"
[1] "Root Mean Square Error : 856.19"
[1] "Accuracy : 78.02"
```

The Figure 2.6 is showing the rules which our decision tree have used to predict the values of the target variables.

As we can see above that rpart method of Decision Tree is giving us the accuracy 78.02% , which is not very impressive. We should try some other models to get the better accuracy than the Decision Tree Model.

Figure 2.6 Decision Tree Rules



2.2.3 Linear Regression

The regression is basically used to predict the relationship among two or more independent variables and dependent variable. Lets use linear regression model to predict the values of target variables.

```
> ### --------- Linear Regression
> lm_model = lm(cnt~.,data= train_data)
> summary(lm_model)

Call:
lm(formula = cnt ~ ., data = train_data)

Residuals:
     Min       1Q   Median       3Q      Max
-3341.5  -360.4    64.9   464.4  3044.9

Coefficients: (1 not defined because of singularities)
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1678.68     269.84   6.221 9.72e-10 ***
season2       645.21     212.96   3.030 0.002561 **
season3       699.46     243.61   2.871 0.004245 **
season4      1521.18     201.71   7.542 1.90e-13 ***
yr2          2016.61      65.36  30.855  < 2e-16 ***
mnth2         169.32     165.84   1.021 0.307714
mnth3         665.47     184.55   3.606 0.000339 ***
mnth4         891.80     293.10   3.043 0.002456 **
mnth5        1204.82     310.67   3.878 0.000118 ***
mnth6         983.22     330.04   2.979 0.003017 **
mnth7         442.13     360.31   1.227 0.220307
mnth8         915.78     348.62   2.627 0.008855 **
mnth9        1369.61     304.99   4.491 8.64e-06 ***
mnth10        763.26     272.07   2.805 0.005203 **
mnth11         -1.29     261.47  -0.005 0.996066
mnth12        -80.12     205.91  -0.389 0.697335
holiday2     -656.73     202.34  -3.246 0.001242 **
weekday2      276.10     123.75   2.231 0.026076 *
weekday3      362.25     118.92   3.046 0.002428 **
weekday4      376.34     120.48   3.124 0.001879 **
weekday5      394.04     119.70   3.292 0.001058 **
weekday6      477.85     120.92   3.952 8.75e-05 ***
weekday7      414.07     120.33   3.441 0.000623 ***
```

```
weathersit2  -483.81      85.55  -5.655 2.49e-08 ***
weathersit3 -2205.40     222.18  -9.926  < 2e-16 ***
temp         3831.98     470.00   8.153 2.37e-15 ***
hum         -1522.77     333.54  -4.566 6.14e-06 ***
windspeed   -2883.89     469.23  -6.146 1.52e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 767 on 556 degrees of freedom
Multiple R-squared:  0.8473,    Adjusted R-squared:  0.8399
F-statistic: 114.3 on 27 and 556 DF,  p-value: < 2.2e-16

> predictions_LR = predict(lm_model,test_data[,-11])
Warning message:
In predict.lm(lm_model, test_data[, -11]) :
  prediction from a rank-deficient fit may be misleading
> Acc(test_data[,11], predictions_LR)
[1] "Mean Absolute Error : 577.47"
[1] "Mean Absolute Percentage Error : 20.14"
[1] "Root Mean Square Error : 807.31"
[1] "Accuracy : 79.86"
```

As we can see linear regression model is giving us the accuracy of 79.86% , which is better than the decision tree but not very impressive. Lets implement some other models to find the better accuracy than linear regression.

2.2.4 Random Forest

2.2.4.1 Random Forest with default parameters

Random forest is an ensemble that consists of many decision trees. Random forest is basically combination of weak learner to produce the strong learning model. Lets apply random forest algorithm and try to find the values of target variable.

```
> ### --------- Random Forest Aglorithm
> RF_model = randomForest(cnt ~ ., train_data, importance = TRUE, ntree = 300)
> RF_Predictions = predict(RF_model, test_data[,-11])
> Acc(test_data[,11], RF_Predictions)
[1] "Mean Absolute Error : 519.82"
[1] "Mean Absolute Percentage Error : 19.94"
[1] "Root Mean Square Error : 732.00"
[1] "Accuracy : 80.06"
```

As we can see above that Random Forest is giving us the accuracy of 80.26%, which is better than decision tree and linear regression model. But as per Machine learning, the accuracy is still not good enough, let's try to get better accuracy.

2.2.4.2 Random Forest Hyperparameter Tuning

In machine learning, Hyperparameter tuning is a problem of choosing a set of optimal hyperparameters for learning algorithm. The same kind of model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters and have to be tuned so that the model can optimally solve the machine learning problem. Hyperparameter tuning finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data. Cross-validation is often use to estimate this generalization performance.

Lets tune the parameter of random forest and try to find the parameters which will give better accuracy than the previous random forest model which was having the default parameters.

```
> ### Creating Random Forest Model using MLR librayr
> ## Hypertuning parameters of random forest model
> ## Create task
> traintask = makeRegrTask( data = train_data, target = "cnt")
> testtask = makeRegrTask( data = test_data, target = "cnt")
> rf.lrn <- makeLearner("regr.randomForest")
> rf.lrn$par.vals <- list(ntree = 100L, importance=TRUE)
> #set 5 fold cross validation
> rdesc <- makeResampleDesc("CV",iters=5L)
> ## set parallelbackend
> parallelStartSocket(cpus = detectCores())
Starting parallelization in mode=socket with cpus=4.
> r <- resample(learner = rf.lrn, task = traintask, resampling = rdesc, measures = list(mae,mape,rmse), show.info = T)
Exporting objects to slaves for mode socket: .mlr.slave.options
Resampling: cross-validation
Measures:            mae       mape      rmse
Mapping in parallel: mode = socket; cpus = 4; elements = 5.


Aggregated Result: mae.test.mean=506.8678521,mape.test.mean=0.4354962,rmse.test.rmse=698.6329710


> getParamSet(rf.lrn)
                Type  len   Def                 Constr Req Tunable Trafo
ntree        integer    -   500               1 to Inf   -    TRUE     -
se.ntree     integer    -   100               1 to Inf   Y    TRUE     -
se.method   discrete    -    sd bootstrap,jackknife,sd   Y    TRUE     -
se.boot      integer    -    50               1 to Inf   -    TRUE     -
mtry         integer    -     -               1 to Inf   -    TRUE     -
replace      logical    -  TRUE                      -   -    TRUE     -
strata       untyped    -     -                      -   -   FALSE     -
sampsize integervector <NA>    -               1 to Inf   -    TRUE     -
nodesize     integer    -     5               1 to Inf   -    TRUE     -
maxnodes     integer    -     -               1 to Inf   -    TRUE     -
importance   logical    - FALSE                      -   -    TRUE     -
localImp     logical    - FALSE                      -   -    TRUE     -
nPerm        integer    -     1            -Inf to Inf   -    TRUE     -
proximity    logical    - FALSE                      -   -   FALSE     -
```

```
> #Set parameters space
> params <- makeParamSet(makeIntegerParam("mtry",lower = 2,upper = 10),makeIntegerParam("nodesize",lower = 10,upper = 50)
)
> #set optimization technique
> ctrl <- makeTuneControlRandom(maxit = 10L)
> #start tuning
> tune <- tuneParams(learner = rf.lrn, task = traintask, resampling = rdesc, measures = list(mae,mape,rmse),
+                    par.set = params, control = ctrl, show.info = T)
[Tune] Started tuning learner regr.randomForest for parameter set:
           Type len Def   Constr Req Tunable Trafo
mtry    integer   -   - 2 to 10   -    TRUE     -
nodesize integer   -   - 10 to 50   -    TRUE     -
With control class: TuneControlRandom
Imputation value: InfImputation value: InfImputation value: Inf
Exporting objects to slaves for mode socket: .mlr.slave.options
Mapping in parallel: mode = socket; cpus = 4; elements = 10.
[Tune] Result: mtry=5; nodesize=10 : mae.test.mean=502.4985988,mape.test.mean=0.5105001,rmse.test.rmse=706.6270459
> tune$x
$`mtry`
[1] 5

$nodesize
[1] 10

> #Set hyperparameters
> lrn_tune <- setHyperPars(rf.lrn,par.vals = list(mtry = 5, nodesize=10))
> #train model
> xgmodel_rf <- train(learner = lrn_tune,task = traintask)
> #predict model
> xgpred_rf <- data.frame(predict(xgmodel_rf,testtask))
> Acc(xgpred_rf$response,xgpred_rf$truth)
[1] "Mean Absolute Error : 492.91"
[1] "Mean Absolute Percentage Error : 12.78"
[1] "Root Mean Square Error : 704.11"
[1] "Accuracy : 87.22"
```

As we can see above, after tuning the parameters of random forest we get much better accuracy than the previous random forest model, which was having the default parameters. After tuning

the hyperparameters we get the accuracy of 87.22% which is better than the accuracy of random forest with default parameter, decision tree and linear regression model.

2.2.5 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting is optimized distributed gradient boosting library. It uses the gradient boosting framework at the core. It is used for supervised ML problems. XGBoost is better than other model because it is enabled with parallel processing, enabled cross validation, handle missing values internally and have provision of regularization technique to avoid overfitting.

2.2.5.1 Extreme Gradient Boosting with Predefined parameters.

In XGBoost, only numeric variables are used to predict the target variable. So first thing we do is we convert all the categorical variables to dummy variables.

After converting all the categorical variables to numeric then we define the parameters for XGBoost and by doing cross validation, we find the minimum number of rounds required. And then we apply our XGBoost model on our train data to predict the values of target variable of test data.

```
> ### --------- Gradient Boosting Algorithm
> ## creating Spare Matrix which converts Categorical variables to dummy variables
> trainm = sparse.model.matrix(cnt~.-1,data = train_data)
> train_label <- train_data[,"cnt"]
> train_matrix = xgb.DMatrix(data = as.matrix(trainm),label = train_label)
> testm = sparse.model.matrix(cnt~.-1,data = test_data)
> test_label = test_data[,"cnt"]
> test_matrix = xgb.DMatrix(data = as.matrix(testm), label = test_label)
> ## Defining Parameters for Xgboost
> params <- list(booster = "gbtree", objective = "reg:linear", eta=0.2, gamma=0, max_depth=6,
+                min_child_weight=1,
+                subsample=1, colsample_bytree=1)
> xgbcv <- xgb.cv( params = params, data = train_matrix, nrounds = 100, nfold = 5, showsd = F, stratified = T,
+                  maximize = F)
[1]     train-rmse:3982.737158  test-rmse:3986.443603
[2]     train-rmse:3228.149805  test-rmse:3242.408398
[3]     train-rmse:2625.302783  test-rmse:2655.416943
[4]     train-rmse:2141.368750  test-rmse:2192.581592
[5]     train-rmse:1753.199268  test-rmse:1826.081738
[6]     train-rmse:1442.992603  test-rmse:1540.500024
[7]     train-rmse:1192.249292  test-rmse:1316.457715
[8]     train-rmse:992.210132   test-rmse:1145.002490
[9]     train-rmse:832.344140   test-rmse:1021.298401
[10]    train-rmse:703.886182   test-rmse:928.493555
[11]    train-rmse:602.087891   test-rmse:861.415894
[12]    train-rmse:520.543750   test-rmse:812.788318
[13]    train-rmse:455.045203   test-rmse:778.697559
[14]    train-rmse:403.409155   test-rmse:753.494128
[15]    train-rmse:359.379034   test-rmse:736.711951
[16]    train-rmse:325.443219   test-rmse:725.095752
[17]    train-rmse:296.855273   test-rmse:714.477222
[18]    train-rmse:274.916846   test-rmse:707.493103
[19]    train-rmse:255.943811   test-rmse:703.760193
[20]    train-rmse:239.904553   test-rmse:701.012073
```

```
[86]    train-rmse:40.297851    test-rmse:695.048560
[87]    train-rmse:39.406548    test-rmse:695.183362
[88]    train-rmse:38.636388    test-rmse:695.386646
[89]    train-rmse:37.827238    test-rmse:695.538928
[90]    train-rmse:37.041830    test-rmse:695.551062
[91]    train-rmse:36.174363    test-rmse:695.752576
[92]    train-rmse:35.365350    test-rmse:695.844421
[93]    train-rmse:34.742996    test-rmse:695.705591
[94]    train-rmse:33.807679    test-rmse:695.761157
[95]    train-rmse:33.083634    test-rmse:695.630579
[96]    train-rmse:32.610171    test-rmse:695.637244
[97]    train-rmse:31.810744    test-rmse:695.532947
[98]    train-rmse:31.285145    test-rmse:695.566504
[99]    train-rmse:30.752132    test-rmse:695.569824
[100]   train-rmse:30.224550    test-rmse:695.571008
> ## creating watchlist
> watchlist = list(train = train_matrix, test = test_matrix)
> ## Apply XGBoost Algorithm for train data
> xgb1 <- xgb.train (params = params, data = train_matrix, nrounds = 43, watchlist = watchlist,
+                    print_every_n = 10, early_stop_round = 10, maximize = F , eval_metric = "rmse")
[1]     train-rmse:3980.550049  test-rmse:3887.259521
[11]    train-rmse:598.427612   test-rmse:849.290161
[21]    train-rmse:235.871307   test-rmse:684.881714
[31]    train-rmse:160.590942   test-rmse:678.419312
[41]    train-rmse:127.152191   test-rmse:678.003540
[43]    train-rmse:122.690819   test-rmse:677.579346
> ## Predicting the values of test data using training data
> xgb_predict = predict(xgb1,test_matrix)
> Acc(test_data[,11],xgb_predict)
[1] "Mean Absolute Error : 476.77"
[1] "Mean Absolute Percentage Error : 15.78"
[1] "Root Mean Square Error : 677.58"
[1] "Accuracy : 84.22"
```

As we can see above that XGBoost(with predefined parameters is giving us the accuracy of 84.22% which is better than Decision Tree, Linear Regression and Random Forest (with default parameters) models, but is less than Random Forest tuned hyperparameter. Lets tune the parameter of XGBoost algorithm and then try to find the target variable.

2.2.5.2 XGBoost with Hyperparameter tuning

```
> ## One Hot Encoding
> traintask = createDummyFeatures(obj = traintask)
> testtask = createDummyFeatures(obj = testtask)
> ## Create Learner
> lrn = makeLearner("regr.xgboost",predict.type = "response")
Warning in makeParam(id = id, type = "numeric", learner.param = TRUE, lower = lower,  :
  NA used as a default value for learner parameter missing.
ParamHelpers uses NA as a special value for dependent parameters.
> lrn$par.vals <- list( objective="reg:linear", eval_metric="rmse", nrounds=100L, eta=0.1)
> ## set parameter space
> params <- makeParamSet( makeDiscreteParam("booster",values = c("gbtree","gblinear")),
+                         makeIntegerParam("max_depth",lower = 3L,upper = 10L), makeNumericParam("min_child_weight",lower
 = 1L,upper = 10L),
+                         makeNumericParam("subsample",lower = 0.5,upper = 1), makeNumericParam("colsample_bytree",lower
= 0.5,upper = 1))
> ## set resampling strategy
> rdesc <- makeResampleDesc("CV",stratify = F,iters=5L)
> ## search strategy
> ctrl <- makeTuneControlRandom(maxit = 10L)
> mytune <- tuneParams(learner = lrn, task = traintask, resampling = rdesc,
+                      par.set = params, control = ctrl, show.info = T)
[Tune] Started tuning learner regr.xgboost for parameter set:
                     Type len Def          Constr Req Tunable Trafo
booster          discrete   -   - gbtree,gblinear   -    TRUE     -
max_depth         integer   -   -         3 to 10   -    TRUE     -
min_child_weight  numeric   -   -         1 to 10   -    TRUE     -
subsample         numeric   -   -        0.5 to 1   -    TRUE     -
colsample_bytree  numeric   -   -        0.5 to 1   -    TRUE     -
With control class: TuneControlRandom
```

```
> mytune$y
mse.test.mean
     399627.1
> ## Set hyperparameters
> lrn_tune <- setHyperPars(lrn,par.vals = mytune$x)
> ## train model
> xgmodel <- train(learner = lrn_tune,task = traintask)
[1]     train-rmse:4455.941406
[2]     train-rmse:4034.816162
[3]     train-rmse:3653.119873
[4]     train-rmse:3308.362305
[5]     train-rmse:2995.987549
[6]     train-rmse:2717.268555
[7]     train-rmse:2470.368896
[8]     train-rmse:2246.498291
[9]     train-rmse:2045.831055
[10]    train-rmse:1861.274048
[11]    train-rmse:1694.890625
[12]    train-rmse:1548.165039
[13]    train-rmse:1418.272217
[14]    train-rmse:1299.574463
[15]    train-rmse:1196.216553
[16]    train-rmse:1102.741821
[17]    train-rmse:1017.406677
[18]    train-rmse:942.403198
[19]    train-rmse:875.450745
[20]    train-rmse:814.589844
[21]    train-rmse:754.762573
[22]    train-rmse:701.888184
[23]    train-rmse:659.343384
[24]    train-rmse:618.944885
[25]    train-rmse:583.207458
[26]    train-rmse:552.674561
[27]    train-rmse:527.245605
[28]    train-rmse:500.372894
[29]    train-rmse:473.792328
```

```
> ## predict model
> xgpred <- data.frame(predict(xgmodel,testtask))
> Acc(xgpred$response,xgpred$truth)
[1] "Mean Absolute Error : 440.21"
[1] "Mean Absolute Percentage Error : 11.87"
[1] "Root Mean Square Error : 676.17"
[1] "Accuracy : 88.13"
>
```

By tuning the parameters of XGBoost we try to find the optimal parameters for XGBoost, which will give us the better accuracy. As we see above, after tuning the parameters of XGBoost Algorithm, we get the accuracy of 88.13% which is greater than Decision Tree, Linear Regression, Random Forest (with default parameters) and Random Forest Algorithm with tuned parameter model.

CHAPTER 3

CONCLUSION

3.1 Model Evaluation

Now that we have few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the model using any of the following criteria:

1. Predictive Performance
2. Interpretability
3. Computational Efficiency

We will use Predictive Performance as our criteria to compare and evaluate the models. Predictive Performance can be measured by comparing Predictions of models with real values of target variable and calculating some average error measure.

Table 3.1 : Model Evaluation

Model	Mean Absolute Error	Mean Absolute Percentage Error	Root Mean Square Error	Accuracy
Decision Tree	599.49	21.98	856.19	78.02
Linear Regression	577.47	20.14	807.31	79.86
Random Forest (with default parameters)	519.82	19.94	732	80.06
Random Forest (with Hyperparameter tuning)	492.91	12.78	704.11	87.22
XGBoost (with predefined parameters)	476.77	15.78	677.58	84.22
XGBoost (with Hyperparameter tuning)	440.21	11.87	676.17	88.13

3.1.1 Mean Absolute Error (MAE)

Mean Absolute Error is the average mean of the absolute Errors and is calculated by the below mentioned formula

$$MAE = (1/n)*(\sum abs\ (y_{actual} - y_{predict}))$$

MAE is one of the error measures used to calculate the predictive performance of the model. We will apply this measure to our model and try to analyze our model.

As we can see in Table 3.1, we have calculate the Mean Absolute Error of all the models which we have coded for predicting the target variable of test data set. From the above table we can say that XGBoost (with Hyperparameter tuning) is giving us the least value of MAE.

3.1.2 Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error measures accuracy as a percentage of the error and is calculated by the below formula.

$$MAPE = (1/n)*(\sum abs((y_{actual} - y_{predict})/y_{actual}))$$

MAPE is one of the error measures used to calculate the Mean Absolute Percentage Error of all the models. In Table 3.1 we can say that XGBoost (with Hyperparameter tuning) is giving us the least value of MAPE.

3.1.3 Root Mean Square Error (RMSE)

Root Mean Square Error is frequently used measure to calculate the difference between values predicted by model or an estimator and the values observed. The RMSE of the model is calculated by the below mentioned formula.

$$RMSE = \sqrt{((1/n)*(\sum abs\ (y_{actual} - y_{predict})^2))}$$

In Table 3.1 we can see that XGBoost (with Hyperparameter tuning) model is giving least value of Root Mean Square Error.

3.1.4 Accuracy

Accuracy is one of the measure which we can use to evaluate the performance of the model.
As we can see in Table 3.1 the maximum accuracy is given by the XGBoost model with Hyperparameter tuning.

3.2 Model Selection

We can use XGBoost model with Hyperparameter tuning as we can see above in the table that this model is given us the maximum accuracy. We could have also used Random Forest with parameter tuning but considering other metrics like MAE, RMSE and MAPE, the XGBoost model is the one which is performing better than the other models.

Appendix A - R Code

Churn Reduction BoxPlot

```
for ( i in 1:length(cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "cnt"), data =
subset(bike_data_copy))+
          stat_boxplot(geom = "errorbar", width = 0.5) +
          geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=1,
                       outlier.size=3, notch=FALSE) +
          theme(legend.position="bottom")+
          labs(x="Count")+
          ggtitle(paste(cnames[i])))+scale_x_discrete(breaks = NULL)
}
```

Outlier Analysis

```
for(i in cnames){
  print(i)
  percentiles = quantile(bike_data_copy[,i],c(0.75,0.25))
  q75= percentiles[[1]]
  q25= percentiles[[2]]
  iqr = q75-q25
  min <- q25 - (1.5*iqr)
  max <- q75 + (1.5*iqr)
  bike_data_copy[,i][bike_data_copy[,i] < min]= min
  bike_data_copy[,i][bike_data_copy[,i] > max]= max
}
```

Correlation Plot

```
corrgram(bike_data_copy[,numeric_index], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")
```

Complete R file

```
### --------- Setting the workirg directory

rm(list=ls())
setwd("D:/Data Science/Projects/Project 2")
getwd()

### --------- Installing packages
lib_list = c("ggplot2","gridExtra","corrgram","randomForest","usdm","sampling","caret",
             "xgboost","magrittr","dplyr","rpart",
             "Matrix","mlr","parallel","parallelMap","rpart.plot")

lapply(lib_list, require,character.only=TRUE)

rm(lib_list)


### --------- Reading the data
bike_data = read.csv("day.csv",header = T ,na.strings = c("","  ","NA"))
bike_data_copy=bike_data
str(bike_data_copy)


### --------- Removing Features "instant" , "dteday" ,"registered" and "casual" from the
dataset. These features does have any relevance.
bike_data_copy=subset(bike_data_copy,select=-c(instant,dteday,registered,casual))
dim(bike_data_copy)

### --------- Changing the datatype of target variable from integer to numeric
bike_data_copy$cnt=as.numeric(bike_data_copy$cnt)


### --------- Changing the datatype to factor and assigning the labels
for(i in 1:ncol(bike_data_copy)){
  if(class(bike_data_copy[,i]) == 'integer'){
    bike_data_copy[,i]=as.factor(bike_data_copy[,i])
    bike_data_copy[,i]= factor(bike_data_copy[,i],labels =
1:length(levels(factor(bike_data_copy[,i]))))
  }
}

str(bike_data_copy)
```

```
### --------- Missing Value Analysis
missing_value = data.frame(apply(bike_data_copy, 2, function(x){sum(is.na(x))}))


### --------- Getting all the numeric columns from the data frame
numeric_index = sapply(bike_data_copy,is.numeric)
numeric_data=bike_data_copy[,numeric_index]
cnames = colnames(numeric_data)

cnames = cnames[-5]
cnames

### --------- Creating box plot of the numeric data for outlier analysis
for ( i in 1:length(cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "cnt"), data =
subset(bike_data_copy))+
          stat_boxplot(geom = "errorbar", width = 0.5) +
          geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=1,
                       outlier.size=3, notch=FALSE) +
          theme(legend.position="bottom")+
          labs(x="Count")+
          ggtitle(paste(cnames[i])))+scale_x_discrete(breaks = NULL)
}



### --------- Plotting Box Plot on the Plot window
gridExtra::grid.arrange(gn1,gn2,gn3,gn4,ncol=4)

### --------- Creating the histogram of the features having outliers plot
hist(bike_data_copy$windspeed)
hist(bike_data_copy$hum)


### --------- Finding all the outliers in the data set and replace them with the floor and ceiling
values.
for(i in cnames){
  print(i)
  percentiles = quantile(bike_data_copy[,i],c(0.75,0.25))
  q75= percentiles[[1]]
  q25= percentiles[[2]]
  iqr = q75-q25
```

```
  min <- q25 - (1.5*iqr)
  max <- q75 + (1.5*iqr)
  bike_data_copy[,i][bike_data_copy[,i] < min]= min
  bike_data_copy[,i][bike_data_copy[,i] > max]= max
}



### --------- Histogram of features after removing the outliers
hist(bike_data_copy$windspeed)
hist(bike_data_copy$hum)



### --------- Feature Selection
### --------- Plotting Correlation plot of the numeric data
corrgram(bike_data_copy[,numeric_index], order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")


### --------- Checking the correlation amoung all the features using variance inflation factor.

bike_data_copy_2= bike_data_copy
for(i in 1:ncol(bike_data_copy_2)){
  if(class(bike_data_copy_2[,i]) == 'factor'){
    bike_data_copy_2[,i]=as.numeric(bike_data_copy_2[,i])
  }
}

vif(bike_data_copy_2[,-12])
vifcor(bike_data_copy_2[,-12],th=0.9)


### -------- Feature Selection

bike_data_copy_rel = subset(bike_data_copy,select =-c(atemp))
str(bike_data_copy_rel)

### -------- Modeling

### -------- Spliting the data into train and test data
set.seed(1234)
train.index = sample(1:nrow(bike_data_copy_rel),0.8*nrow(bike_data_copy_rel))
train_data = bike_data_copy_rel[train.index,]
```

```
test_data = bike_data_copy_rel[-train.index,]

str(train_data)


## Defining Function which will we used to find the accuracy of the model
## Mean Absolute Percentage Error
MAPE = function(y, yhat){
  mean(abs((y - yhat)/y))*100
}

## Mean Absolute Error
MAE = function(y,f){
  mean(abs(y-f))
}

## Root Mean Square Error
RMSE = function(y,f){
  sqrt(mean((y-f)^2))
}


## Accuracy
Acc = function(test_data_true, predicted_values){
  mean_abs_error = format(round(MAE(test_data_true,predicted_values),2),nsmall = 2)
  root_mean_sq_er = format(round(RMSE(test_data_true,predicted_values),2),nsmall = 2)
  Error = format(round(MAPE(test_data_true,predicted_values),2),nsmall = 2)
  Accuracy = 100 - as.numeric(Error)
  print(paste0("Mean Absolute Error : ", mean_abs_error))
  print(paste0("Mean Absolute Percentage Error : " , Error))
  print(paste0("Root Mean Square Error : ", root_mean_sq_er))
  print(paste0("Accuracy : ", Accuracy))
}



### -------- Decision Tree

### -------- Rpart for regreesion
dt_model = rpart(cnt~.,data=train_data,method = 'anova')
rpart.plot(dt_model)
rpart.rules(dt_model)
predict_dt = predict(dt_model,test_data[,-11])
Acc(test_data[,11],predict_dt)
```

```
## Error Rate 21.98
## Accuracy 78.02


### --------- Linear Regression
lm_model = lm(cnt~.,data= train_data)
summary(lm_model)
predictions_LR = predict(lm_model,test_data[,-11])
Acc(test_data[,11], predictions_LR)

## Error Rate 20.13
## Accuracy 79.86

### --------- Random Forest Aglorithm
RF_model = randomForest(cnt ~ ., train_data, importance = TRUE, ntree = 300)
RF_Predictions = predict(RF_model, test_data[,-11])

Acc(test_data[,11], RF_Predictions)

## Error 19.31
## Accuracy 80.69

### Creating Random Forest Model using MLR librayr
## Hypertuning parameters of random forest model
## Create task
traintask = makeRegrTask( data = train_data, target = "cnt")
testtask = makeRegrTask( data = test_data, target = "cnt")

rf.lrn <- makeLearner("regr.randomForest")
rf.lrn$par.vals <- list(ntree = 100L, importance=TRUE)

#set 5 fold cross validation
rdesc <- makeResampleDesc("CV",iters=5L)


## set parallelbackend
parallelStartSocket(cpus = detectCores())

r <- resample(learner = rf.lrn, task = traintask, resampling = rdesc, measures =
list(mae,mape,rmse), show.info = T)
```

```
getParamSet(rf.lrn)

#Set parameters space
params <- makeParamSet(makeIntegerParam("mtry",lower = 2,upper =
10),makeIntegerParam("nodesize",lower = 10,upper = 50))


#set optimization technique
ctrl <- makeTuneControlRandom(maxit = 10L)

#start tuning
tune <- tuneParams(learner = rf.lrn, task = traintask, resampling = rdesc, measures =
list(mae,mape,rmse),
                   par.set = params, control = ctrl, show.info = T)
tune$x

#Set hyperparameters
lrn_tune <- setHyperPars(rf.lrn,par.vals = list(mtry = 5, nodesize=10))

#train model
xgmodel_rf <- train(learner = lrn_tune,task = traintask)

#predict model
xgpred_rf <- data.frame(predict(xgmodel_rf,testtask))
Acc(xgpred_rf$response,xgpred_rf$truth)

##Error 12.77
## Accuracy 87.23



### --------- Gradient Boosting Algorithm
## creating Spare Matrix which converts Categorical Variables to dummy variables
trainm = sparse.model.matrix(cnt~.-1,data = train_data)
train_label <- train_data[,"cnt"]
train_matrix = xgb.DMatrix(data = as.matrix(trainm),label = train_label)

testm = sparse.model.matrix(cnt~.-1,data = test_data)
test_label = test_data[,"cnt"]
test_matrix = xgb.DMatrix(data = as.matrix(testm), label = test_label)



## Defining Parameters for Xgboost
```

```
params <- list(booster = "gbtree", objective = "reg:linear", eta=0.2, gamma=0, max_depth=6,
               min_child_weight=1,
               subsample=1, colsample_bytree=1)

## Cross Validation - for finding the minimum number of rounds required to find the best
accuracy

xgbcv <- xgb.cv( params = params, data = train_matrix, nrounds = 100, nfold = 5, showsd = F,
stratified = T,
                 maximize = F)

## Creating watchlist
watchlist = list(train = train_matrix, test = test_matrix)

## Apply XGBoost Algorithm for train data
xgb1 <- xgb.train (params = params, data = train_matrix, nrounds = 43, watchlist = watchlist,
                   print_every_n = 10, early_stop_round = 10, maximize = F , eval_metric = "rmse")

## Predicting the values of test data using training data
xgb_predict = predict(xgb1,test_matrix)
Acc(test_data[,11],xgb_predict)

##Error 15.78
##Acuracy 84.22

### -------- Random / Grid search procedure


## One Hot Encoding
traintask = createDummyFeatures(obj = traintask)
testtask = createDummyFeatures(obj = testtask)


## Create Learner
lrn = makeLearner("regr.xgboost",predict.type = "response")
lrn$par.vals <- list( objective="reg:linear", eval_metric="rmse", nrounds=100L, eta=0.1)


## set parameter space
params <- makeParamSet( makeDiscreteParam("booster",values = c("gbtree","gblinear")),
                        makeIntegerParam("max_depth",lower = 3L,upper = 10L),
makeNumericParam("min_child_weight",lower = 1L,upper = 10L),
                        makeNumericParam("subsample",lower = 0.5,upper = 1),
makeNumericParam("colsample_bytree",lower = 0.5,upper = 1))
```

```
## set resampling strategy
rdesc <- makeResampleDesc("CV",stratify = F,iters=5L)

## search strategy
ctrl <- makeTuneControlRandom(maxit = 10L)


mytune <- tuneParams(learner = lrn, task = traintask, resampling = rdesc,
                     par.set = params, control = ctrl, show.info = T)


mytune$y


## Set hyperparameters
lrn_tune <- setHyperPars(lrn,par.vals = mytune$x)

## train model
xgmodel <- train(learner = lrn_tune,task = traintask)

## predict model
xgpred <- data.frame(predict(xgmodel,testtask))
Acc(xgpred$response,xgpred$truth)

#Error 12.56
#Accuracy 87.44
```

References

- Edwisor.com - Online Learning Material
- https://www.analyticsvidhya.com/
- https://www.hackerearth.com/