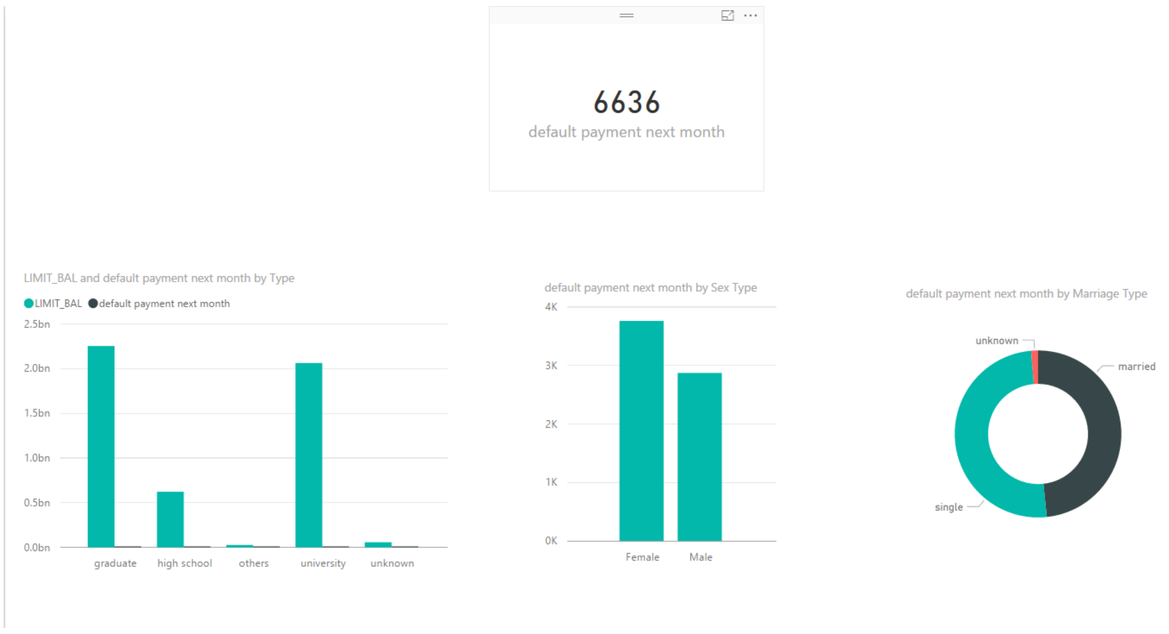
Midterm Report~ Team1

Q1. The data provided is for credit card defaults. This problem was provided with 23 explanatory variables. The 23 variables are as follows:

- X1: amount of given credit
- X2: Gender (1= male, 2=female)
- X3: Education (1= graduate school; 2= university; 3=high school; 4=others)
- Marital status (1 = married; 2 = single; 3 = others)
- Age (year)
- X6 - X11: History of past payment
- X12-X17: Amount of bill statement
- X18 – X23= amount paid

<u>Data Exploration:</u>



The dashboard shown above consists of 3 charts and one card.

- The first vertical bar chart shows the categories of Education present in the data vs. Balance Limit and Default Payment. From this graph we can see that there are unknown values which need to be dealt with during data cleansing.
- The second bar chart shows the Sex Type vs. Default Payment.
- The third bar chart shows the Marriage Type vs. Default Payment. . From this graph we can see that there are unknown values which need to be dealt with during data cleansing.
- The card gives a count of the Default Payment throughout the dashboard.

Data Cleansing:

The data cleansing is done is python and the code is shared.

Steps involved in data cleansing are as follows:

1. The headers (X1, X2, X3…) are replaced with the actual meaningful headings such as (Limit, Gender, Education..)
2. The column "Education " where (1= graduate school; 2= university; 3=high school; 4=others)

   The values that are actually present in the dataset are (0, 1, 2, 3, 4, 5, 6)

   As 0, 5 and 6 are not mentioned in the problem statement, they are transformed into NAN values. The values with 4 too are transformed into NAN as it is unknown. Finally the Education column consists of:

   - 1 has 10585 values
   - 2 has 14030 values

- 3 has 4917 values
- NAN has 468 values

3. The column “Marriage” where (1 = married; 2 = single; 3 = others)

   The values that are actually present in the dataset are (0, 1, 2, 3). As 0 and 3 are not mentioned in the problem statement, they are transformed into NAN values.

   Finally the Marriage column consists of:

   - 0 has 15964
   - 1 has 13659
   - NAN has 377

4. All the values in “Education” and “Marriage” which contain NAN are replaced with mean of their respective columns.

5. Three new features have been added to the dataset called “Total Bill”, “Total Paid”, “Avg_Status”.

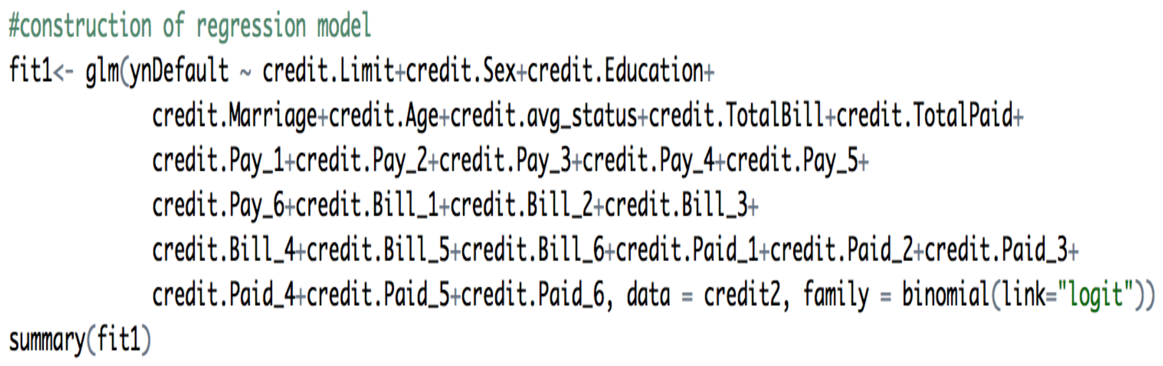
   The columns “Pay”, “Bill_Amt”, “Pay_Amt” has 6 columns each for each month from April to September.

   - Total Bill- has the sum of the bill generated by the customer for the past six months.
   - Total Paid- has the sum of the bill cleared by the customer is the past six months.
   - Average status- counts the number of this the customer has faid to pay their bill. In 6 months if they have cleared their bill more than 3 times then this column stores a “paid” status else ”not paid” status.

- Another column for the purpose of classification problem is introduced called "ynDefault" which stores "Yes" and "No". Where "Yes" is for "will default" and "No" is for "will not default".

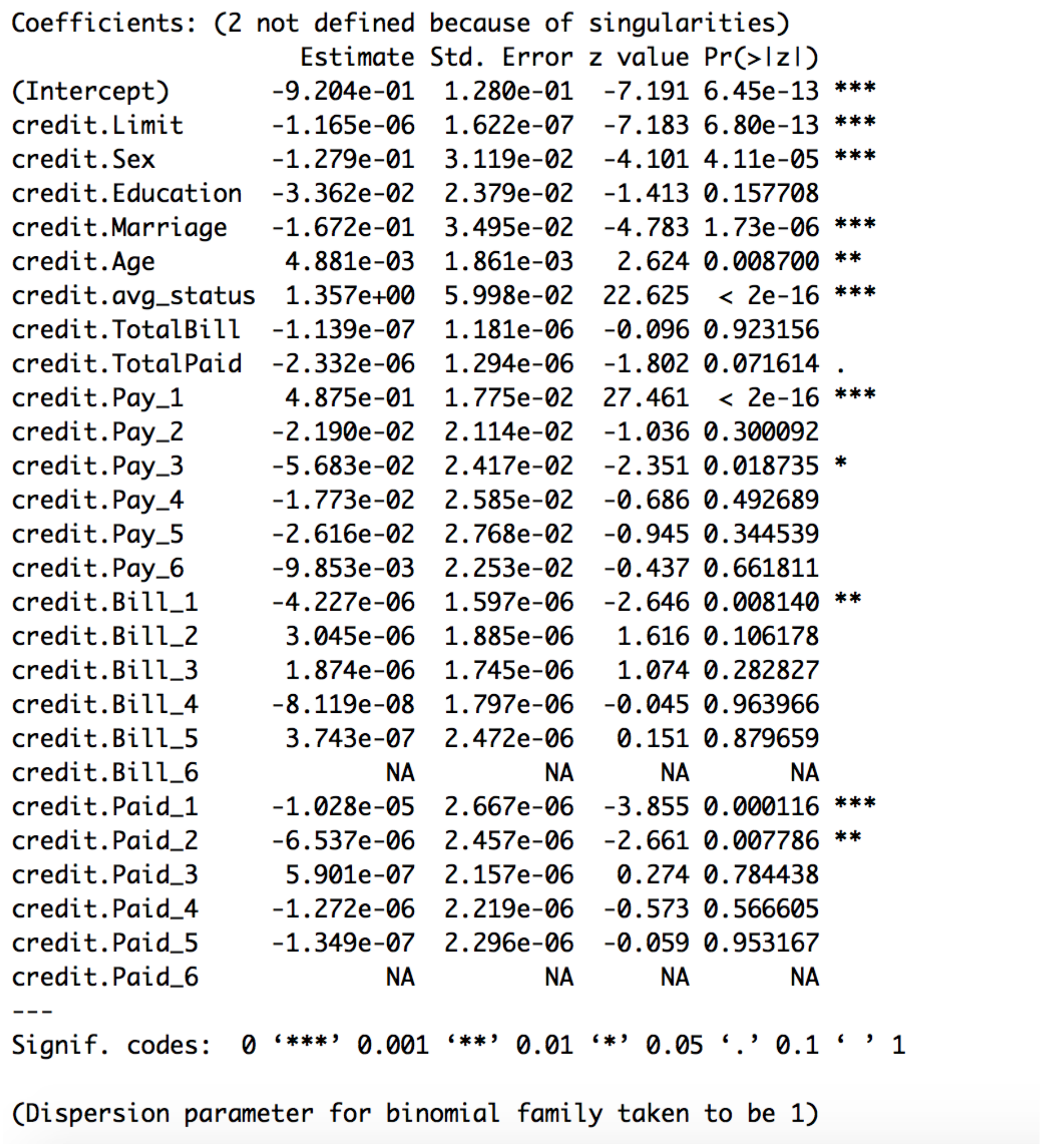
Logistic Regression:

- Constructing Logistic Regression Model



```
#construction of regression model
fit1<- glm(ynDefault ~ credit.Limit+credit.Sex+credit.Education+
             credit.Marriage+credit.Age+credit.avg_status+credit.TotalBill+credit.TotalPaid+
             credit.Pay_1+credit.Pay_2+credit.Pay_3+credit.Pay_4+credit.Pay_5+
             credit.Pay_6+credit.Bill_1+credit.Bill_2+credit.Bill_3+
             credit.Bill_4+credit.Bill_5+credit.Bill_6+credit.Paid_1+credit.Paid_2+credit.Paid_3+
             credit.Paid_4+credit.Paid_5+credit.Paid_6, data = credit2, family = binomial(link="logit"))
summary(fit1)
```

Output:



```
Coefficients: (2 not defined because of singularities)
                    Estimate Std. Error z value Pr(>|z|)    
(Intercept)       -9.204e-01  1.280e-01  -7.191 6.45e-13 ***
credit.Limit      -1.165e-06  1.622e-07  -7.183 6.80e-13 ***
credit.Sex        -1.279e-01  3.119e-02  -4.101 4.11e-05 ***
credit.Education  -3.362e-02  2.379e-02  -1.413 0.157708    
credit.Marriage   -1.672e-01  3.495e-02  -4.783 1.73e-06 ***
credit.Age         4.881e-03  1.861e-03   2.624 0.008700 ** 
credit.avg_status  1.357e+00  5.998e-02  22.625  < 2e-16 ***
credit.TotalBill  -1.139e-07  1.181e-06  -0.096 0.923156    
credit.TotalPaid  -2.332e-06  1.294e-06  -1.802 0.071614 .  
credit.Pay_1       4.875e-01  1.775e-02  27.461  < 2e-16 ***
credit.Pay_2      -2.190e-02  2.114e-02  -1.036 0.300092    
credit.Pay_3      -5.683e-02  2.417e-02  -2.351 0.018735 *  
credit.Pay_4      -1.773e-02  2.585e-02  -0.686 0.492689    
credit.Pay_5      -2.616e-02  2.768e-02  -0.945 0.344539    
credit.Pay_6      -9.853e-03  2.253e-02  -0.437 0.661811    
credit.Bill_1     -4.227e-06  1.597e-06  -2.646 0.008140 ** 
credit.Bill_2      3.045e-06  1.885e-06   1.616 0.106178    
credit.Bill_3      1.874e-06  1.745e-06   1.074 0.282827    
credit.Bill_4     -8.119e-08  1.797e-06  -0.045 0.963966    
credit.Bill_5      3.743e-07  2.472e-06   0.151 0.879659    
credit.Bill_6             NA         NA      NA       NA    
credit.Paid_1     -1.028e-05  2.667e-06  -3.855 0.000116 ***
credit.Paid_2     -6.537e-06  2.457e-06  -2.661 0.007786 ** 
credit.Paid_3      5.901e-07  2.157e-06   0.274 0.784438    
credit.Paid_4     -1.272e-06  2.219e-06  -0.573 0.566605    
credit.Paid_5     -1.349e-07  2.296e-06  -0.059 0.953167    
credit.Paid_6             NA         NA      NA       NA    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)
```

- The dataset is split into training and testing dataset, 75% of dataset for training and 25% for testing.

```r
#partitioning dataframe
smp_size<- floor(0.75* nrow(credit2))
set.seed(123)
train_index<- sample(seq_len(nrow(credit2)),size = smp_size)

#splitting data into test and train
train<- credit2[train_index,]
test<- credit2[- train_index,]
```

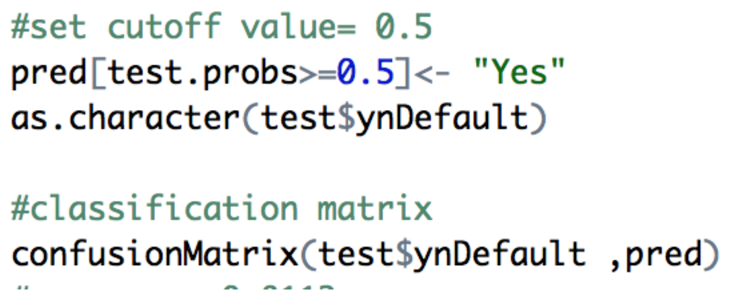
- Selecting the features that will impact the regression model and running logistic regression on them.

```r
#constructing regression model with significant factors
fit2<- glm(ynDefault ~ credit.Limit+credit.Sex+credit.Marriage+credit.Age
           +credit.avg_status+credit.TotalBill+credit.Pay_1+credit.Pay_3
           +credit.Bill_1 +credit.Paid_1+credit.Paid_2
           +credit.TotalPaid, data = train, family = binomial(link="logit"))
summary(fit2)
```

Output:

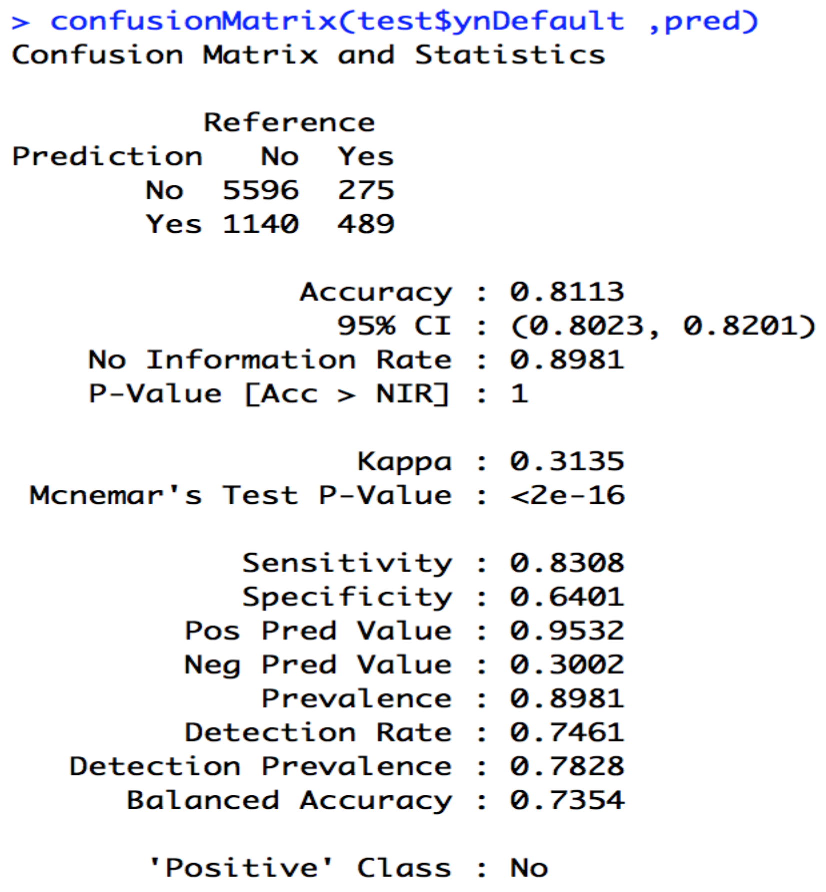
```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6730  -0.6499  -0.5369  -0.3309   3.1461

Coefficients:
                    Estimate Std. Error z value Pr(>|z|)
(Intercept)       -9.596e-01  1.387e-01  -6.919 4.54e-12 ***
credit.Limit      -1.064e-06  1.762e-07  -6.041 1.53e-09 ***
credit.Sex        -1.440e-01  3.591e-02  -4.010 6.08e-05 ***
credit.Marriage   -1.583e-01  3.999e-02  -3.958 7.57e-05 ***
credit.Age         4.962e-03  2.114e-03   2.347 0.018950 *
credit.avg_status  1.288e+00  6.549e-02  19.667  < 2e-16 ***
credit.TotalBill   4.379e-07  1.782e-07   2.457 0.013998 *
credit.Pay_1       4.984e-01  1.966e-02  25.356  < 2e-16 ***
credit.Pay_3      -9.445e-02  2.127e-02  -4.441 8.97e-06 ***
credit.Bill_1     -2.384e-06  9.008e-07  -2.646 0.008142 **
credit.Paid_1     -8.605e-06  2.655e-06  -3.241 0.001191 **
credit.Paid_2     -6.659e-06  2.378e-06  -2.800 0.005111 **
credit.TotalPaid  -2.919e-06  7.581e-07  -3.850 0.000118 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)
```

- The Classification matrix is as follows:



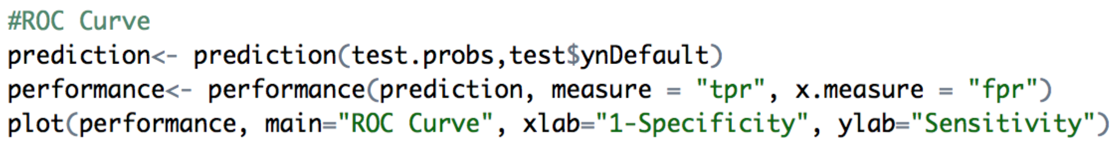
```
#set cutoff value= 0.5
pred[test.probs>=0.5]<- "Yes"
as.character(test$ynDefault)

#classification matrix
confusionMatrix(test$ynDefault ,pred)
```

Output:



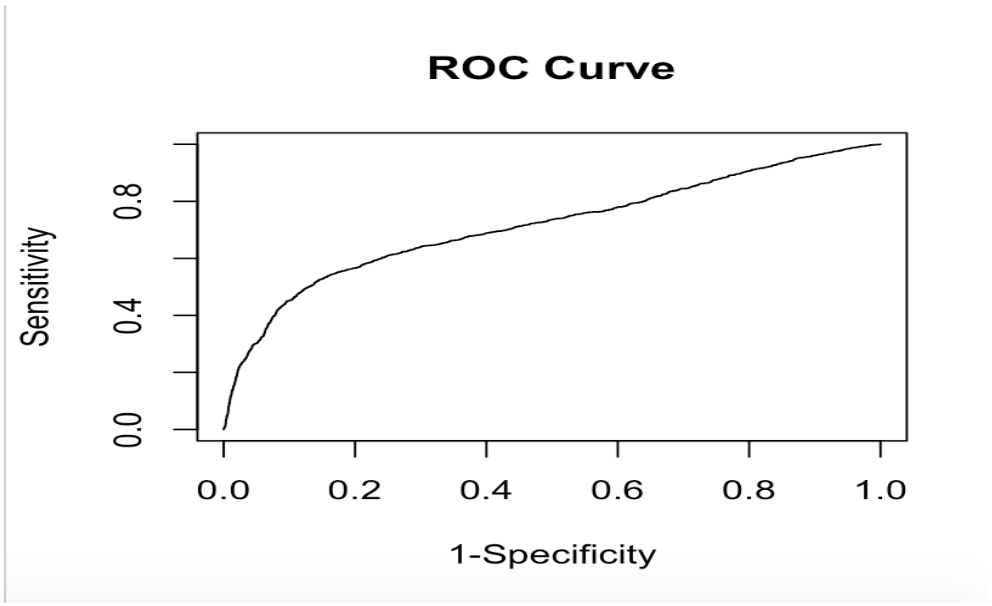
```
> confusionMatrix(test$ynDefault ,pred)
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  5596  275
       Yes 1140  489

               Accuracy : 0.8113
                 95% CI : (0.8023, 0.8201)
    No Information Rate : 0.8981
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3135
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8308
            Specificity : 0.6401
         Pos Pred Value : 0.9532
         Neg Pred Value : 0.3002
             Prevalence : 0.8981
         Detection Rate : 0.7461
   Detection Prevalence : 0.7828
      Balanced Accuracy : 0.7354

       'Positive' Class : No
```

- ROC Curve



```
#ROC Curve
prediction<- prediction(test.probs,test$ynDefault)
performance<- performance(prediction, measure = "tpr", x.measure = "fpr")
plot(performance, main="ROC Curve", xlab="1-Specificity", ylab="Sensitivity")
```

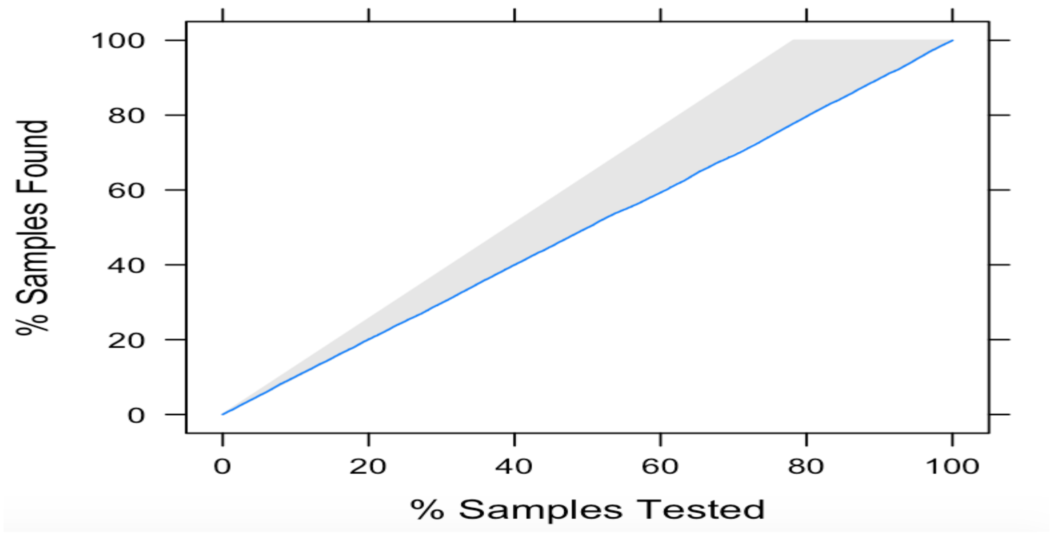
Output:



- Lift Chart

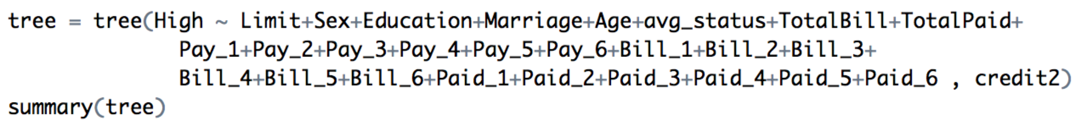
```
#Lift Curve
test$probs<- test.probs
test$prob<- sort(test$probs, decreasing = T)
lift<- lift(ynDefault ~ prob, data = test)
lift
xyplot(lift, plot="gain")
```

Output:



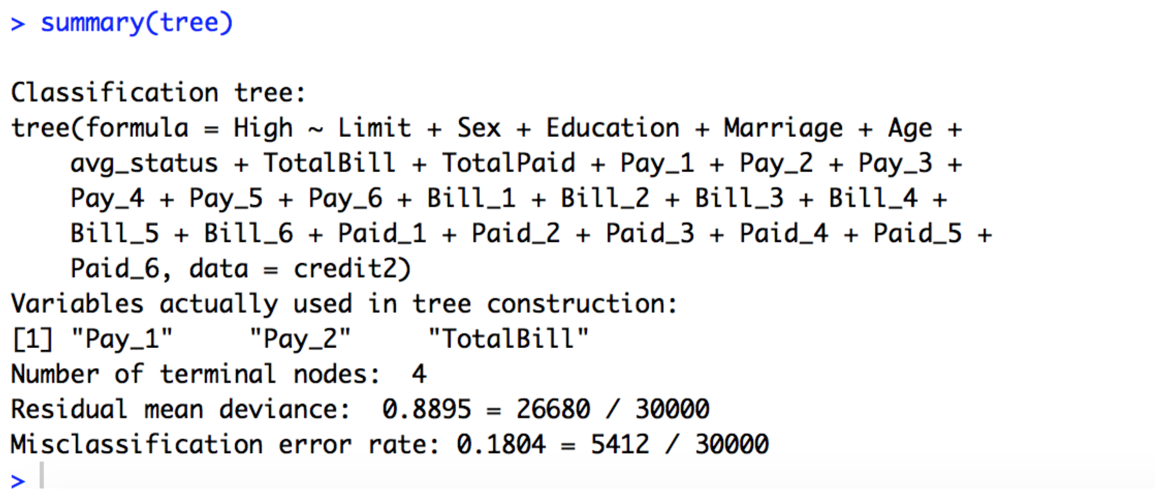
Classification Tree:

- Running classification tree on the dataset



```
tree = tree(High ~ Limit+Sex+Education+Marriage+Age+avg_status+TotalBill+TotalPaid+
              Pay_1+Pay_2+Pay_3+Pay_4+Pay_5+Pay_6+Bill_1+Bill_2+Bill_3+
              Bill_4+Bill_5+Bill_6+Paid_1+Paid_2+Paid_3+Paid_4+Paid_5+Paid_6 , credit2)
summary(tree)
```

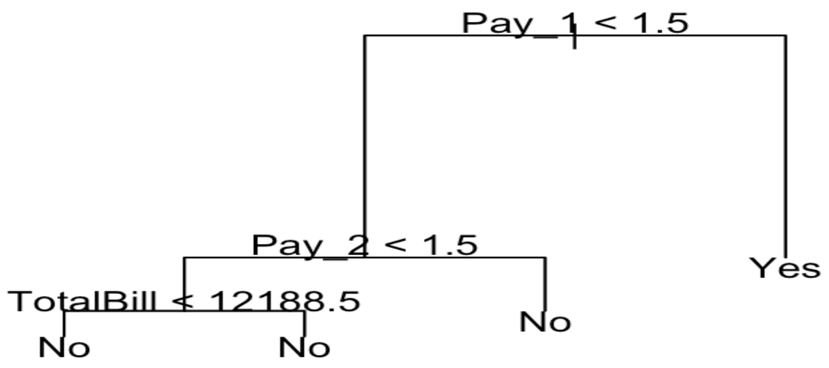
Output:



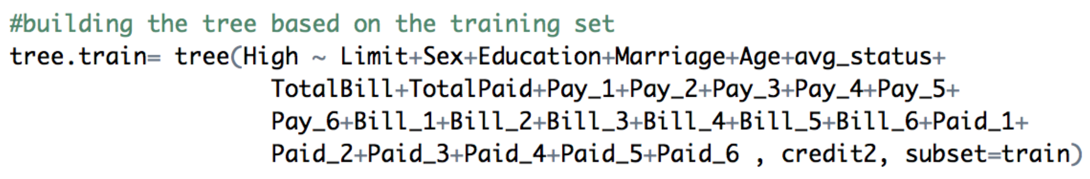
```
> summary(tree)

Classification tree:
tree(formula = High ~ Limit + Sex + Education + Marriage + Age +
    avg_status + TotalBill + TotalPaid + Pay_1 + Pay_2 + Pay_3 +
    Pay_4 + Pay_5 + Pay_6 + Bill_1 + Bill_2 + Bill_3 + Bill_4 +
    Bill_5 + Bill_6 + Paid_1 + Paid_2 + Paid_3 + Paid_4 + Paid_5 +
    Paid_6, data = credit2)
Variables actually used in tree construction:
[1] "Pay_1"     "Pay_2"     "TotalBill"
Number of terminal nodes:  4
Residual mean deviance:  0.8895 = 26680 / 30000
Misclassification error rate: 0.1804 = 5412 / 30000
>
```

error rate-> 5412 / 30000 = 0.1804

RMS-> 0.94

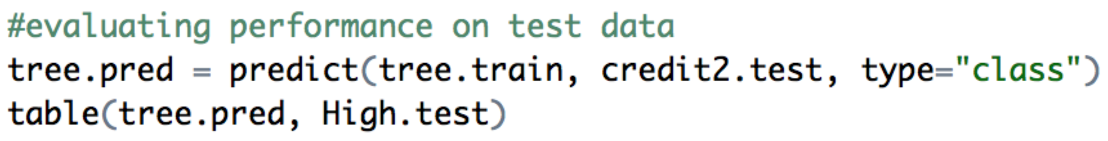


- Splitting the data into test and train and growing the tree on the training set.



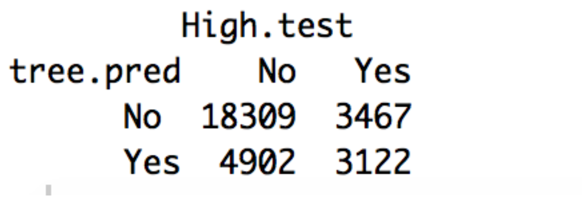
```
#building the tree based on the training set
tree.train= tree(High ~ Limit+Sex+Education+Marriage+Age+avg_status+
                 TotalBill+TotalPaid+Pay_1+Pay_2+Pay_3+Pay_4+Pay_5+
                 Pay_6+Bill_1+Bill_2+Bill_3+Bill_4+Bill_5+Bill_6+Paid_1+
                 Paid_2+Paid_3+Paid_4+Paid_5+Paid_6 , credit2, subset=train)
```

- Performance matrix:



```
#evaluating performance on test data
tree.pred = predict(tree.train, credit2.test, type="class")
table(tree.pred, High.test)
```

Output-



```
         High.test
tree.pred    No   Yes
      No  18309  3467
      Yes  4902  3122
```

We can calculate the following from the classification matrix-

Accuracy: (TP+TN)/total

(18309+3122)/(18309+3467+4902+3122)=0.7191611

error rate: (FP+FN)/total

(3467+4902)/(18309+3467+4902+3122)=0.2808389

precision: TP/predicted yes

3122/(3467+3122)= 0.47382

specificity: TN/actual no

18309/(18309+3467)= 0.840788

- After Pruning the tree and from the classification matrix calculated from the pruned tree we can say that the pruned tree gives better values i.e. the error rate is reduced.

```
#prune tree
prune.credit2 = prune.misclass(tree, best= 3)
plot(prune.credit2 )
text(prune.credit2 , pretty=0)

#confusion matrix
prune.pred= predict(prune.credit2,credit2.test,type = "class")
table(High.test,prune.pred)
```

Output:

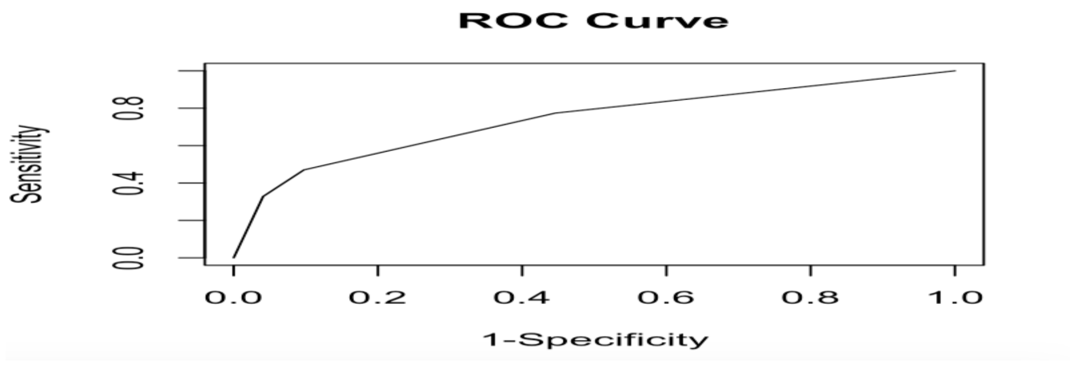
```
         prune.pred
High.test    No   Yes
      No  22263   948
      Yes  4427  2162
```

error rate=(948+4427)/(22263+948+4427+2162)= 0.180

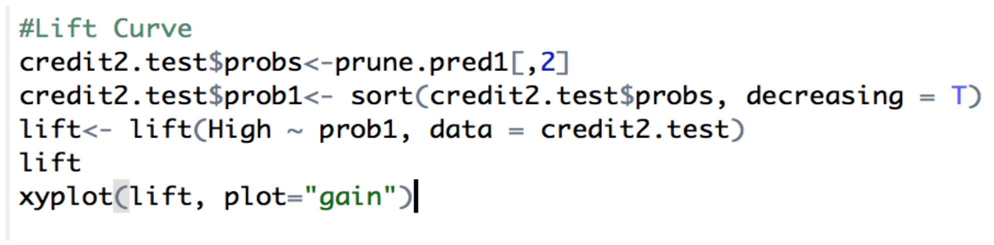
- ROC Curve is created on the pruned tree

```
#ROC Curve

prediction<- prediction(prune.pred1[,2],High.test)
performance<- performance(prediction, measure = "tpr", x.measure = "fpr")
plot(performance, main="ROC Curve", xlab="1-Specificity", ylab="Sensitivity")
```

Output:

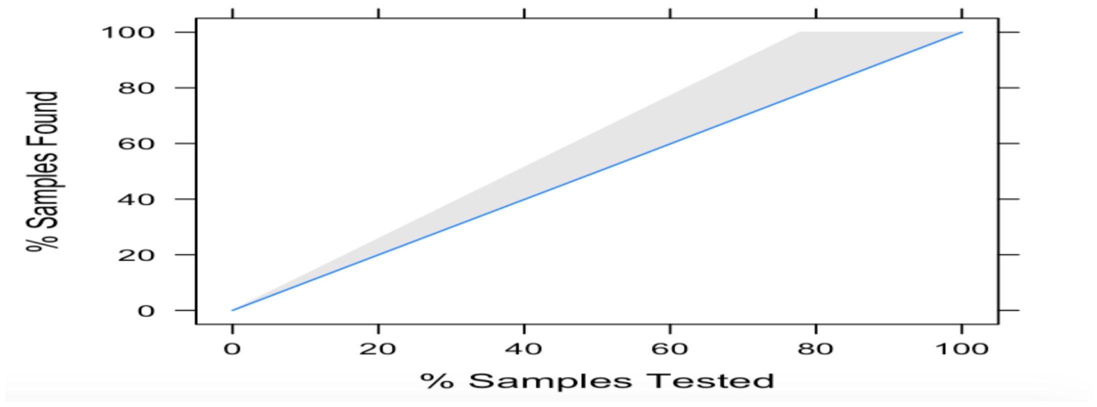


- Lift Chart is created on the pruned tree



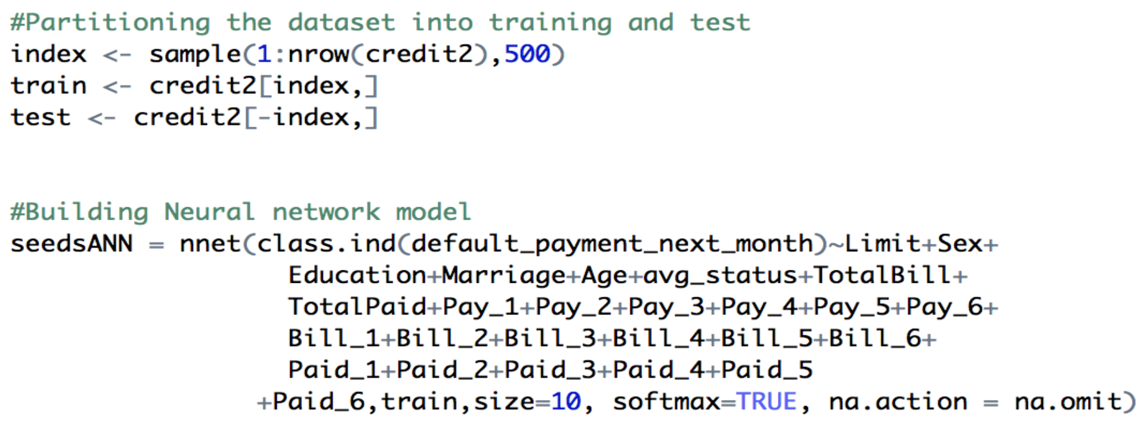
```
#Lift Curve
credit2.test$probs<-prune.pred1[,2]
credit2.test$prob1<- sort(credit2.test$probs, decreasing = T)
lift<- lift(High ~ prob1, data = credit2.test)
lift
xyplot(lift, plot="gain")
```

Output:



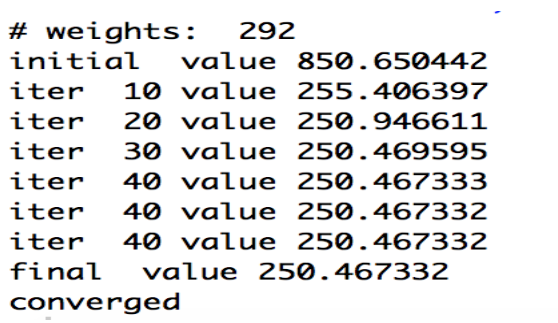
Neural Networks:

- The libraries used for neural networks are: nnet, NeuralNetTools
- Dataset is partitioned into test and train and the neural network model is created by running it on the dataset.



```
#Partitioning the dataset into training and test
index <- sample(1:nrow(credit2),500)
train <- credit2[index,]
test <- credit2[-index,]


#Building Neural network model
seedsANN = nnet(class.ind(default_payment_next_month)~Limit+Sex+
                  Education+Marriage+Age+avg_status+TotalBill+
                  TotalPaid+Pay_1+Pay_2+Pay_3+Pay_4+Pay_5+Pay_6+
                  Bill_1+Bill_2+Bill_3+Bill_4+Bill_5+Bill_6+
                  Paid_1+Paid_2+Paid_3+Paid_4+Paid_5
                +Paid_6,train,size=10, softmax=TRUE, na.action = na.omit)
```

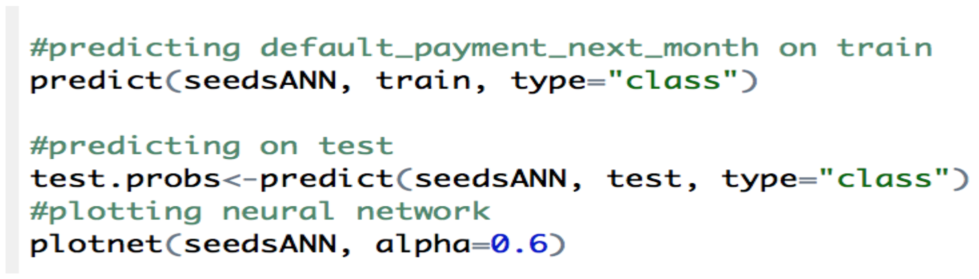
Output:



```
# weights:  292
initial  value 850.650442
iter  10 value 255.406397
iter  20 value 250.946611
iter  30 value 250.469595
iter  40 value 250.467333
iter  40 value 250.467332
iter  40 value 250.467332
final  value 250.467332
converged
```

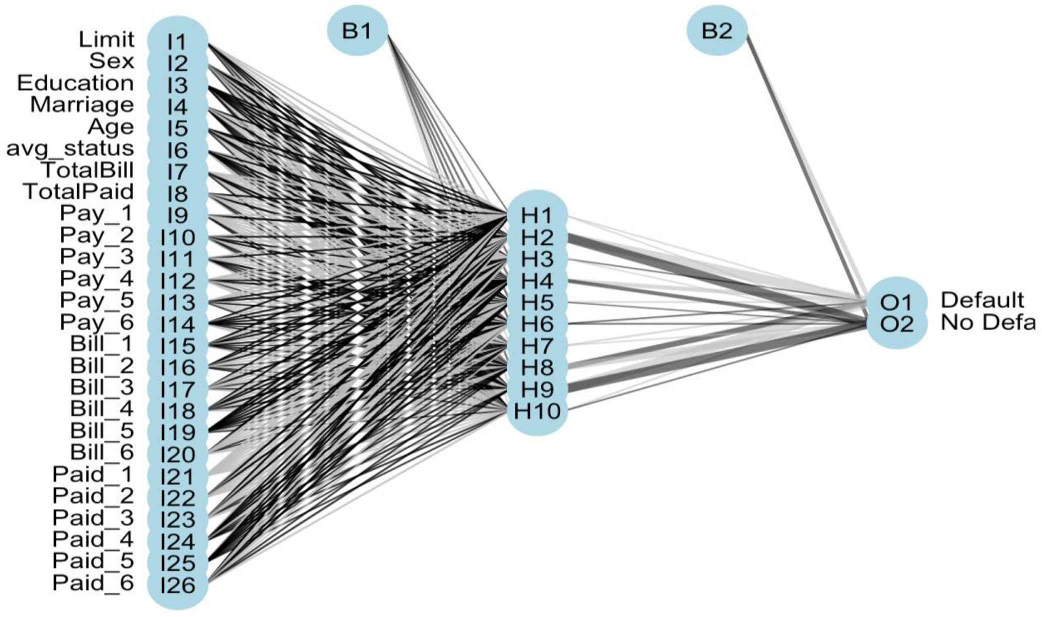
- The predictions are then made on the training dataset and the neural

network is plotted.

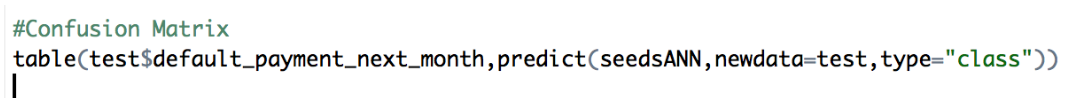


```
#predicting default_payment_next_month on train
predict(seedsANN, train, type="class")

#predicting on test
test.probs<-predict(seedsANN, test, type="class")
#plotting neural network
plotnet(seedsANN, alpha=0.6)
```

The neural Network:

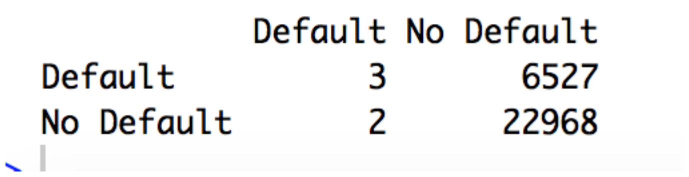


- Confusion Matrix produced is as follows:



```
#Confusion Matrix
table(test$default_payment_next_month,predict(seedsANN,newdata=test,type="class"))
```

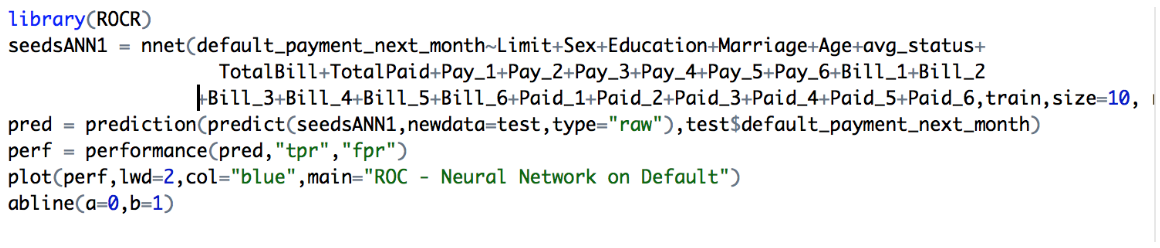
Output:



```
            Default No Default
Default           3       6527
No Default        2      22968
```

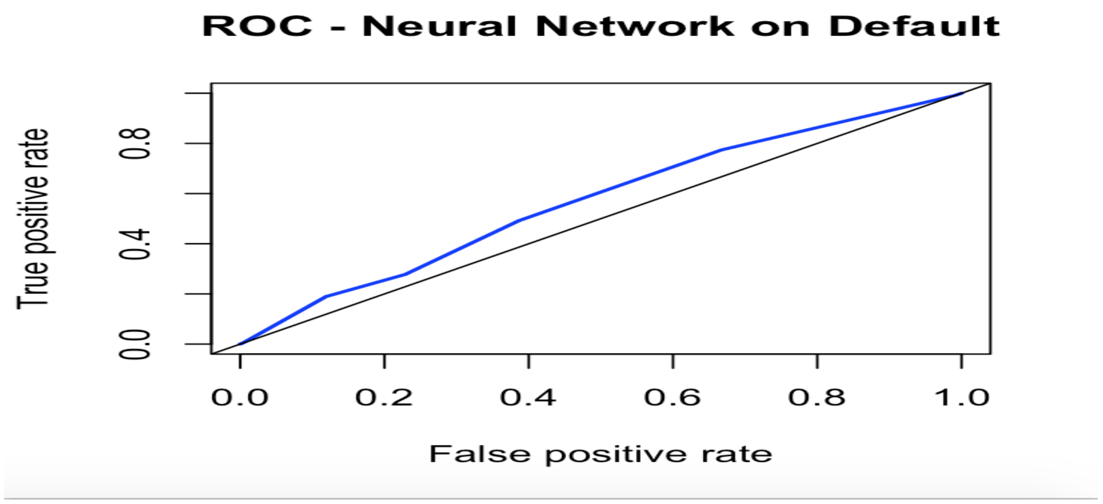
error rate: (6527+2)/(22968+3+6527+2)=0.22
Accuracy: (3+22968)/(22968+3+6527+2)= 0.778678
Precision: (22968)/(22968+6527)= 0.7787083

- On plotting the ROC curve for the neural network model



```
library(ROCR)
seedsANN1 = nnet(default_payment_next_month~Limit+Sex+Education+Marriage+Age+avg_status+
                   TotalBill+TotalPaid+Pay_1+Pay_2+Pay_3+Pay_4+Pay_5+Pay_6+Bill_1+Bill_2
                 +Bill_3+Bill_4+Bill_5+Bill_6+Paid_1+Paid_2+Paid_3+Paid_4+Paid_5+Paid_6,train,size=10,
pred = prediction(predict(seedsANN1,newdata=test,type="raw"),test$default_payment_next_month)
perf = performance(pred,"tpr","fpr")
plot(perf,lwd=2,col="blue",main="ROC - Neural Network on Default")
abline(a=0,b=1)
```

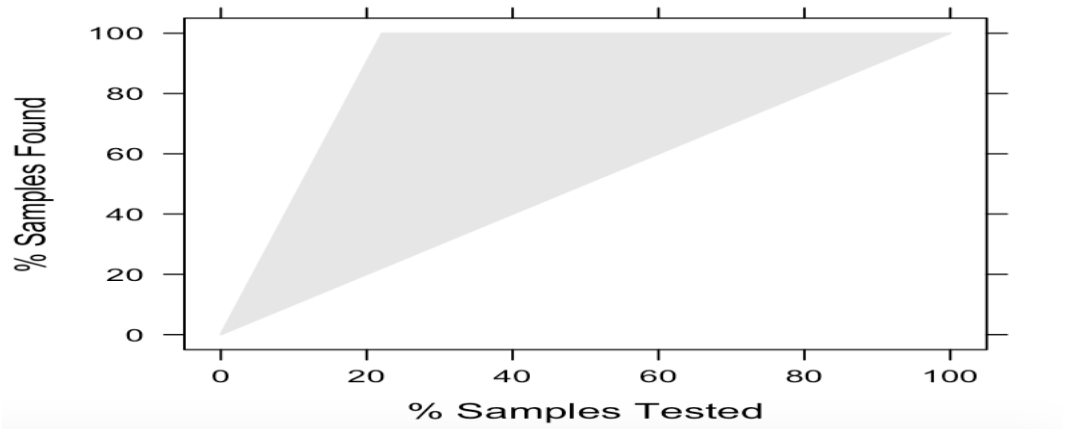
Output:



- On plotting the lift curve:

```
#lift curve
test$probs<-test.probs
test$prob<- sort(test$probs, decreasing = T)
lift<- lift(default_payment_next_month ~ prob, data = test)
lift
xyplot(lift, plot="gain")
```

Output:

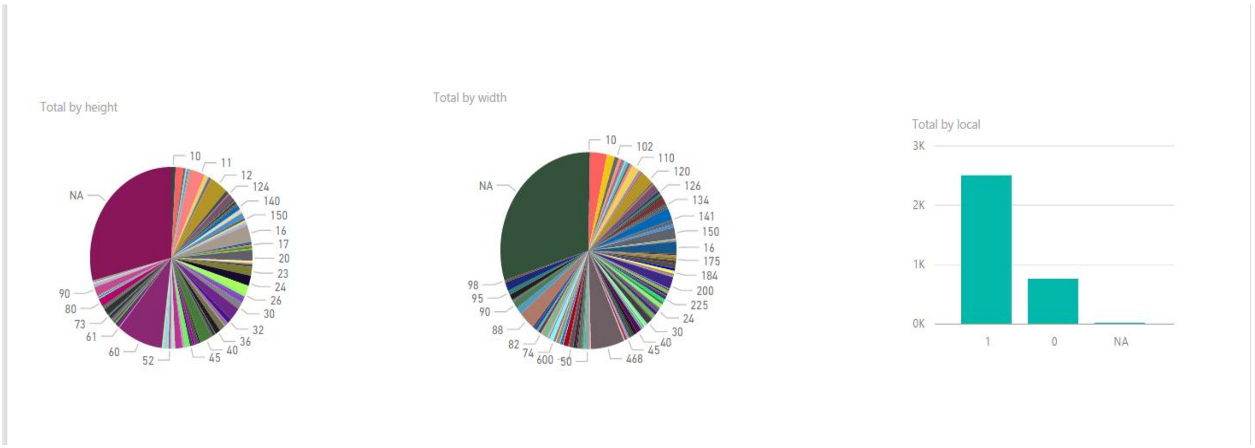


Selection of the appropriate model:

After running classification trees model, logistic regression model and neural network model, the best results are obtained with the classification tree.
The classification tree is pruned to give better results.
If you refer to the classification matrix and the lift, ROC curve above, in comparison to neural network model and logistic model.
Error Rate: How often the predictions are wrong is 18%
Accuracy: How often the predictions are correct is 81%
Precision: If predicted Yes, how often is it right 70%

Q2. The data represents possible advertisements on the Internet. There are 1558 variables of which 3 are continuous in nature. There are three files provided to us of which one has the data, the other has the headers of the columns and the third file has the documentations.

Data Exploration:



This dashboard contains 3 charts:

- The first is a pie chart which tells us that the column height contains 27.54% NAs.
- The first is a pie chart which tells us that the column width contains 27.48% NAs.
- The chart bar chart tells us that the column local contains 0.46% of NAs.

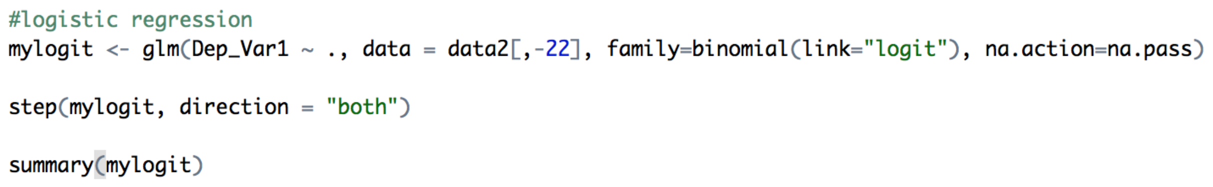
Data Cleansing and Pre-processing:

1. After importing both files containing the header and the other file containing the footer is imported. The file containing header is made into the column headers for the second file containing the data.
2. The 3 continuous columns of “height”, ”weight” and “ratio” contains “?” which are replaced by NA .

3. As 28% percent of the data is stored as NA therefore it import to deal with NAs'.
    - Using a library called "Hmisc" and a function called aregImpute{Hmisc}.
    - This function calculates and predicts values for the NA values from a full Bayesian predictive distribution.
    - The function provides five values for each NA value.
    - Then median is calculated of the five values to substitute the NA value. Median is calculated such that if there are any extreme values the calculated value shouldn't be affected as much ,unlike in mean.
    - This process in applied for height ,width and local columns.
    - To obtain the missing values "aratio", the values of width is divided by height.
4. There are 1559 features that are present. Therefore, using non zero variance to select columns, which have 95%-5% ratio of most commonly used values. This is done to reduce the number of columns in the dataset.
5. On running non- zero variance function on the dataset we are left with 22 columns.
6. A new column is introduced for the purpose of classification which stores 1 if it is an "ad" else it stores 0.

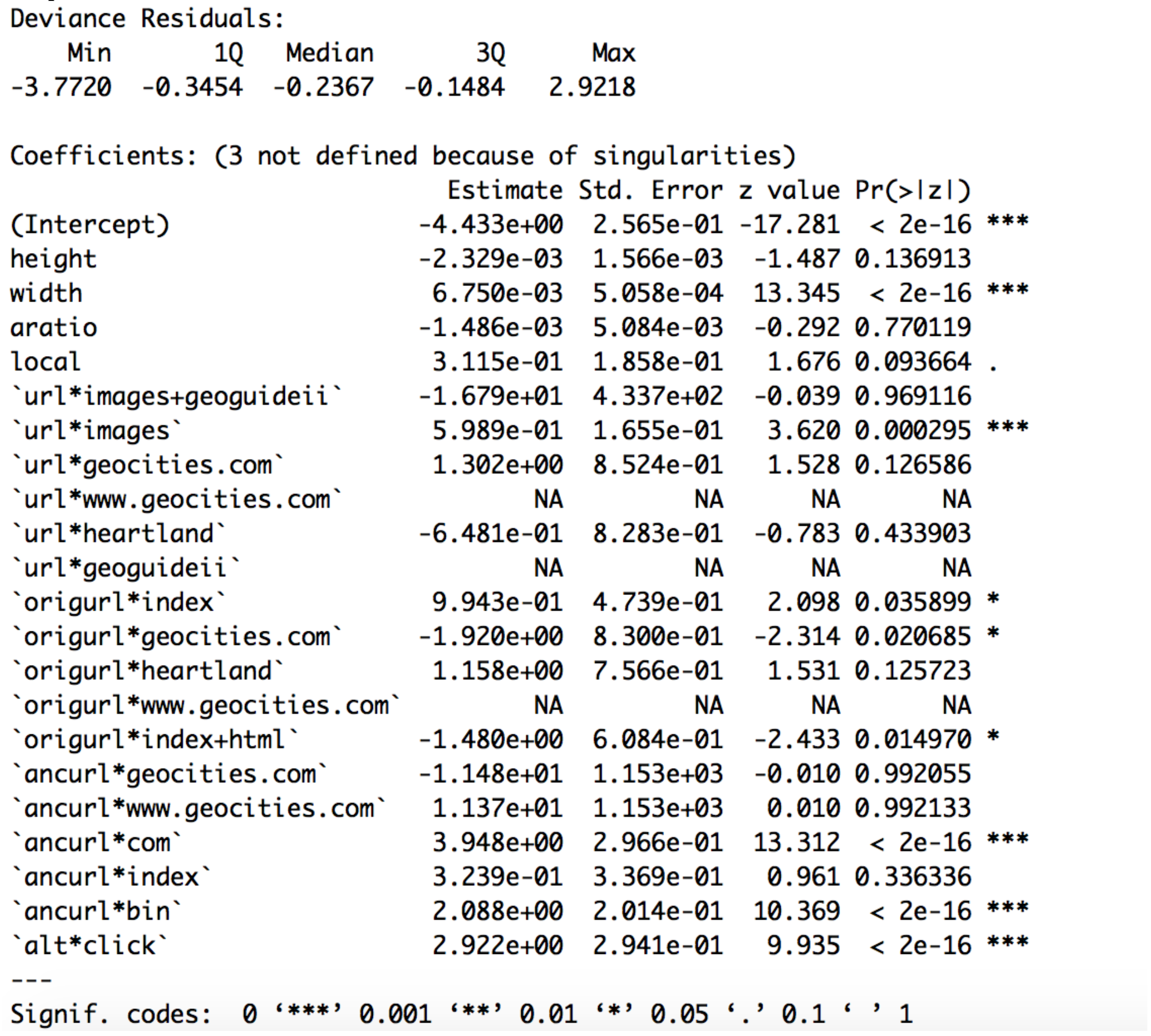
Logistic Regression:

- On running logistic regression on the dataset after performing non-zero variance:



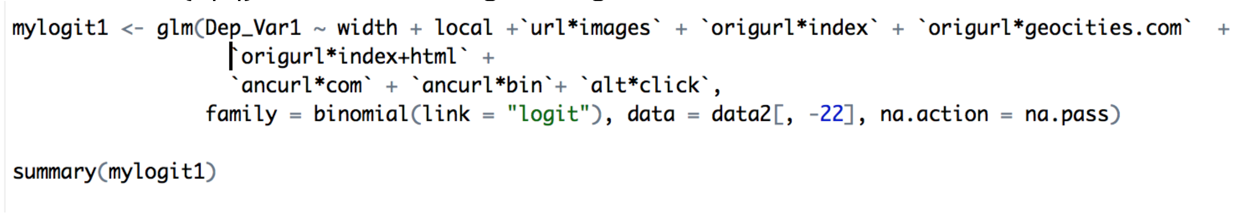
```
#logistic regression
mylogit <- glm(Dep_Var1 ~ ., data = data2[,-22], family=binomial(link="logit"), na.action=na.pass)

step(mylogit, direction = "both")

summary(mylogit)
```

Output:



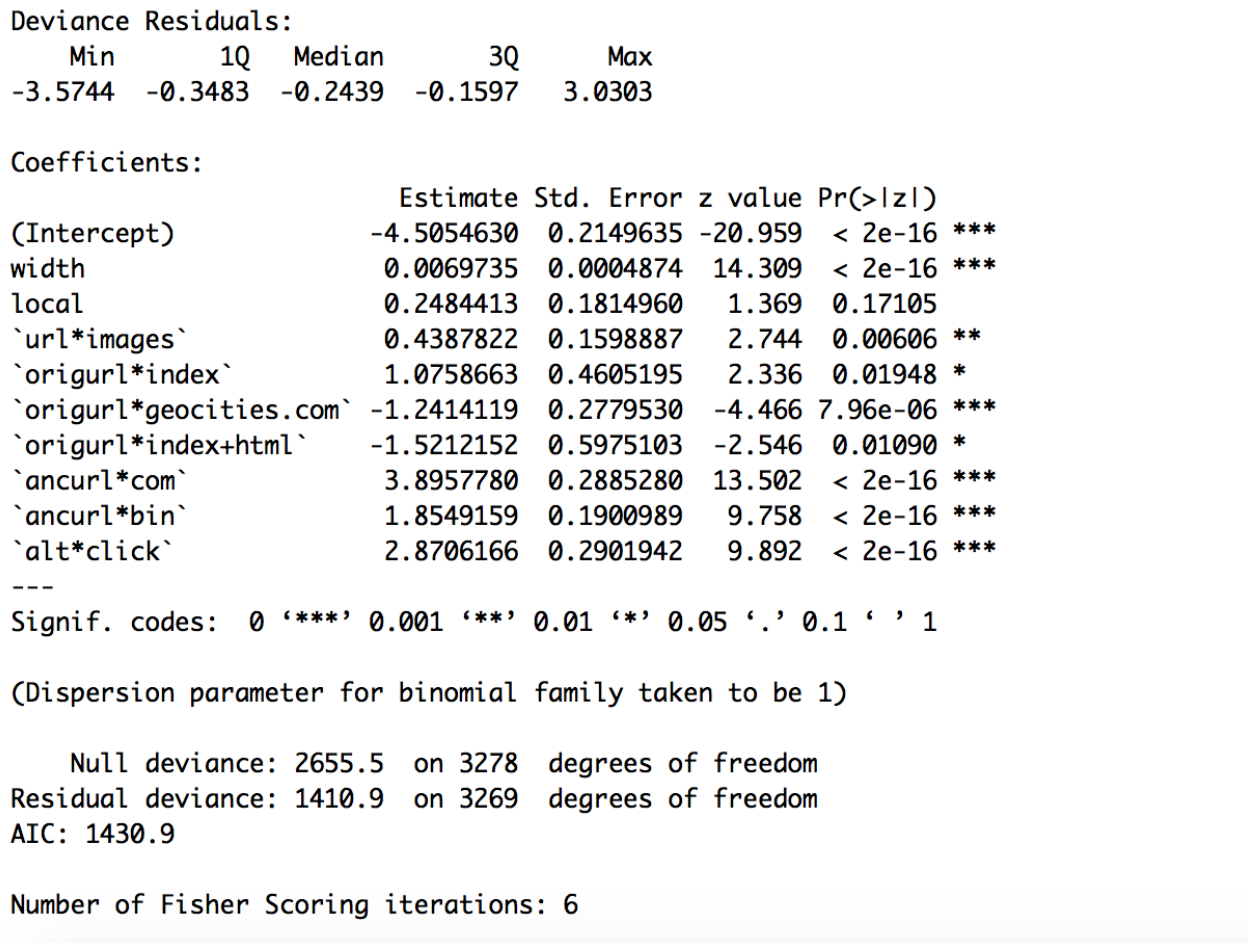
```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.7720  -0.3454  -0.2367  -0.1484   2.9218

Coefficients: (3 not defined because of singularities)
                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                 -4.433e+00  2.565e-01 -17.281  < 2e-16 ***
height                      -2.329e-03  1.566e-03  -1.487 0.136913
width                        6.750e-03  5.058e-04  13.345  < 2e-16 ***
aratio                      -1.486e-03  5.084e-03  -0.292 0.770119
local                        3.115e-01  1.858e-01   1.676 0.093664 .
`url*images+geoguideii`     -1.679e+01  4.337e+02  -0.039 0.969116
`url*images`                 5.989e-01  1.655e-01   3.620 0.000295 ***
`url*geocities.com`          1.302e+00  8.524e-01   1.528 0.126586
`url*www.geocities.com`             NA         NA      NA       NA
`url*heartland`             -6.481e-01  8.283e-01  -0.783 0.433903
`url*geoguideii`                    NA         NA      NA       NA
`origurl*index`              9.943e-01  4.739e-01   2.098 0.035899 *
`origurl*geocities.com`     -1.920e+00  8.300e-01  -2.314 0.020685 *
`origurl*heartland`          1.158e+00  7.566e-01   1.531 0.125723
`origurl*www.geocities.com`         NA         NA      NA       NA
`origurl*index+html`        -1.480e+00  6.084e-01  -2.433 0.014970 *
`ancurl*geocities.com`      -1.148e+01  1.153e+03  -0.010 0.992055
`ancurl*www.geocities.com`   1.137e+01  1.153e+03   0.010 0.992133
`ancurl*com`                 3.948e+00  2.966e-01  13.312  < 2e-16 ***
`ancurl*index`               3.239e-01  3.369e-01   0.961 0.336336
`ancurl*bin`                 2.088e+00  2.014e-01  10.369  < 2e-16 ***
`alt*click`                  2.922e+00  2.941e-01   9.935  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

- Therefore further selecting the features that are important on looking at Pr(>|z|) values. A new logistic regression model is created:



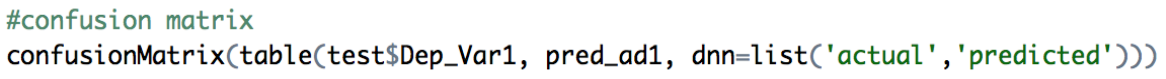
```
mylogit1 <- glm(Dep_Var1 ~ width + local +`url*images` + `origurl*index` + `origurl*geocities.com`  +
               `origurl*index+html` +
               `ancurl*com` + `ancurl*bin`+ `alt*click`,
             family = binomial(link = "logit"), data = data2[, -22], na.action = na.pass)

summary(mylogit1)
```

Output:



```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.5744  -0.3483  -0.2439  -0.1597   3.0303

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)             -4.5054630  0.2149635 -20.959  < 2e-16 ***
width                    0.0069735  0.0004874  14.309  < 2e-16 ***
local                    0.2484413  0.1814960   1.369  0.17105
`url*images`             0.4387822  0.1598887   2.744  0.00606 **
`origurl*index`          1.0758663  0.4605195   2.336  0.01948 *
`origurl*geocities.com` -1.2414119  0.2779530  -4.466 7.96e-06 ***
`origurl*index+html`    -1.5212152  0.5975103  -2.546  0.01090 *
`ancurl*com`             3.8957780  0.2885280  13.502  < 2e-16 ***
`ancurl*bin`             1.8549159  0.1900989   9.758  < 2e-16 ***
`alt*click`              2.8706166  0.2901942   9.892  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2655.5  on 3278  degrees of freedom
Residual deviance: 1410.9  on 3269  degrees of freedom
AIC: 1430.9

Number of Fisher Scoring iterations: 6
```

- Confusion Matrix is generated on the new dataset :



```
#confusion matrix
confusionMatrix(table(test$Dep_Var1, pred_ad1, dnn=list('actual','predicted')))
```

Output:

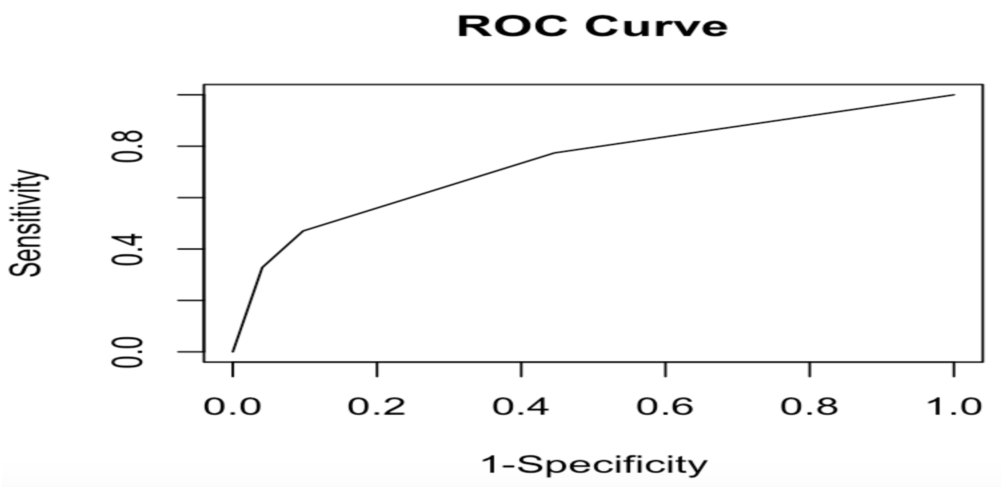
```
        predicted
actual    0    1
     0  686   17
     1   42   75

               Accuracy : 0.928
                 95% CI : (0.9082, 0.9448)
    No Information Rate : 0.8878
    P-Value [Acc > NIR] : 7.201e-05

                  Kappa : 0.6771
 Mcnemar's Test P-Value : 0.001781

            Sensitivity : 0.9423
            Specificity : 0.8152
         Pos Pred Value : 0.9758
         Neg Pred Value : 0.6410
             Prevalence : 0.8878
         Detection Rate : 0.8366
   Detection Prevalence : 0.8573
      Balanced Accuracy : 0.8788

       'Positive' Class : 0
```

Precision= 0.815

- ROC Curve is generated for the logistic regression model:

```
library(ROCR)
pr = prediction(pred_ad1, test$Dep_Var1)

prf = performance(pr,"tpr","fpr")

performance(pr,"auc")

plot(performance, main="ROC Curve", xlab="1-Specificity", ylab="Sensitivity")
```

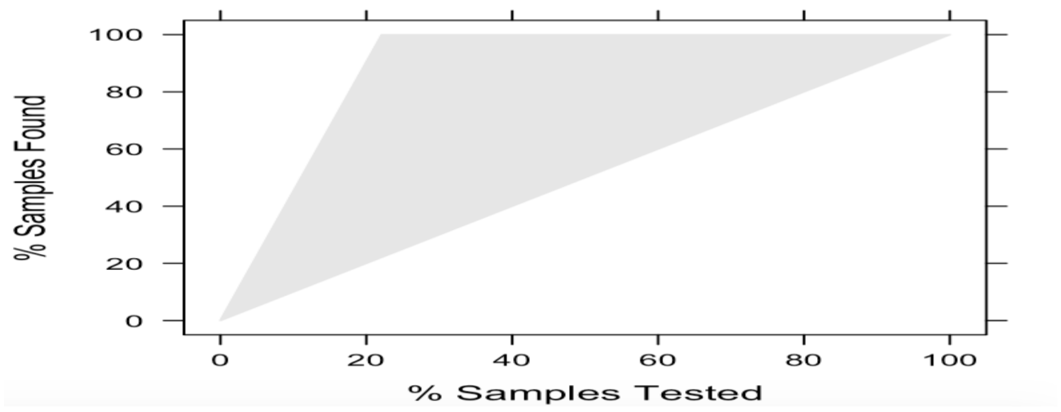
Output:



- The lift chart is generated for the logistic regression model:

```
#Lift Chart
test$probs<- pred_ad
test$prob<- sort(test$probs, decreasing = T)
lift<- lift(Dep_Var1 ~ prob, data = test)
lift
xyplot(lift, plot="gain")
```

Output:



Classification Tree:

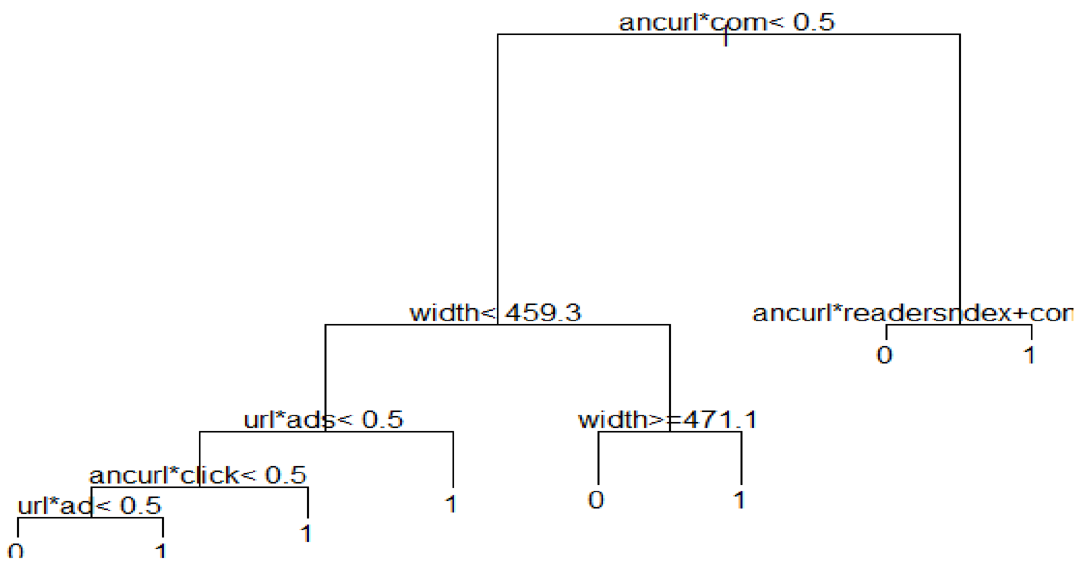
- On splitting the data in test and train and running the classification model on the dataset:

```
#fitting classification tree

set.seed(1)
train <- sample(1:nrow(data1), 0.75 * nrow(data1))

library(rpart)
library(rpart.plot)

adTree <- rpart(Dep_Var1 ~ . - Dep_Var, data = data1[train, ], method = 'class')
plot(adTree)
text(adTree, pretty = 0)
printcp(adTree)
summary(adTree)
```

Output:

```
Classification tree:
rpart(formula = Dep_Var1 ~ . - Dep_Var, data = data1[train, ],
    method = "class")

Variables actually used in tree construction:
[1] ancurl*click           ancurl*com             ancurl*readersndex+com
[4] url*ad                 url*ads                width

Root node error: 351/2459 = 0.14274

n= 2459

        CP nsplit rel error  xerror     xstd
1 0.398860      0   1.00000 1.00000 0.049420
2 0.145299      1   0.60114 0.60114 0.039569
3 0.076923      2   0.45584 0.47293 0.035446
4 0.071225      3   0.37892 0.43305 0.034022
5 0.039886      4   0.30769 0.31339 0.029205
6 0.025641      5   0.26781 0.28490 0.027905
7 0.019943      6   0.24217 0.27066 0.027227
8 0.010000      7   0.22222 0.23362 0.025365
```

- The tree selects the 6 features to run the tree model on:



- After predicting on the Test dataset then the confusion matrix is calculated.

```
#prediction
AdPrediction <- predict(adTree, data1[-train, ], type = 'vector')
AdPrediction

#confusion matrix
table(AdPrediction, data1[-train, ]$Dep_Var1)

> #confusion matrix
> table(AdPrediction, data1[-train, ]$Dep_Var1)

AdPrediction   0   1
           1 706  27
           2   6  81
>
```

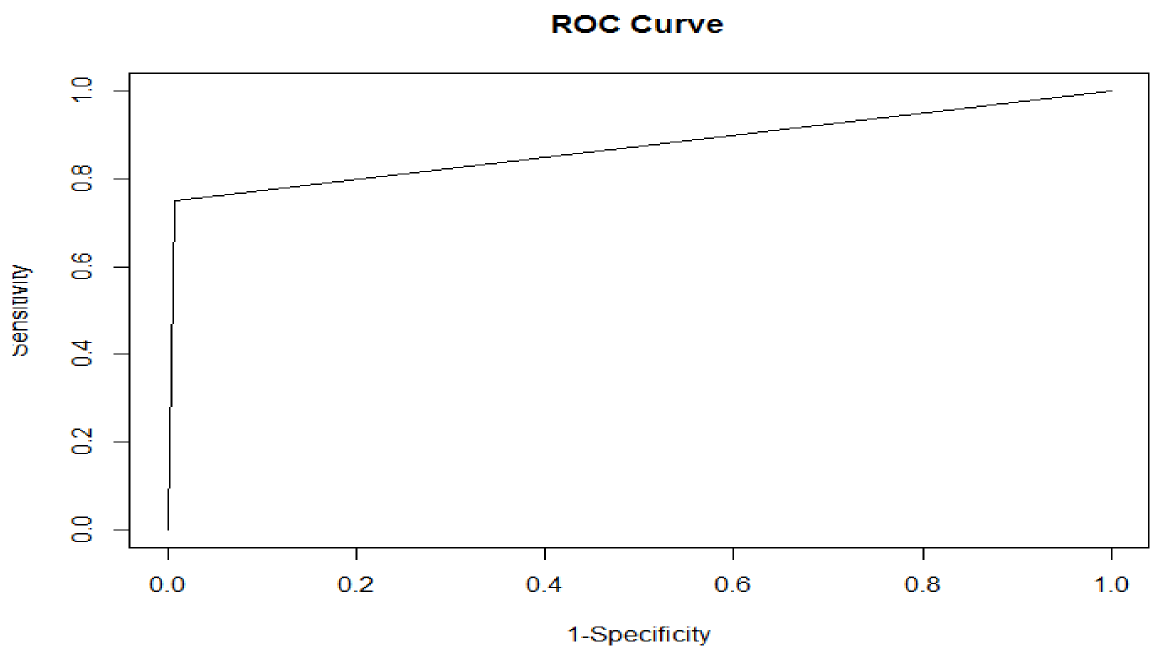
Error rate: (27+6)/(706+81)=0.0402439
Accuracy: (706+81)/(706+81+27+6)= 0.9597561

Precision: 81/(27+81)=0.75

- The ROC Curve is generated on the classification tree model

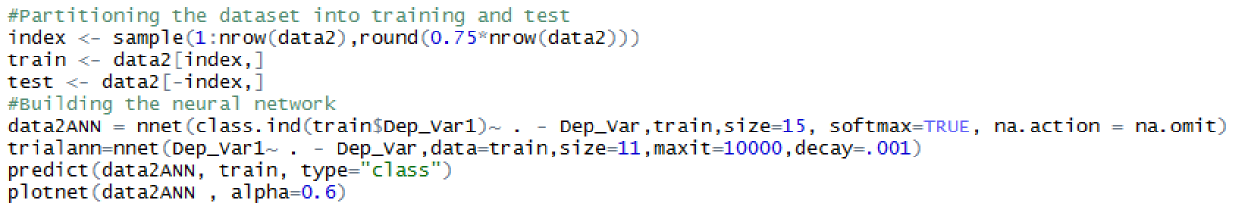
```
library(arulesViz)
#ROC Curve
prediction<- prediction(AdPrediction, data1[-train, ]$Dep_Var1)
performance<- performance(prediction, measure = "tpr", x.measure = "fpr")
plot(performance, main="ROC Curve", xlab="1-Specificity", ylab="Sensitivity")
```

Output:



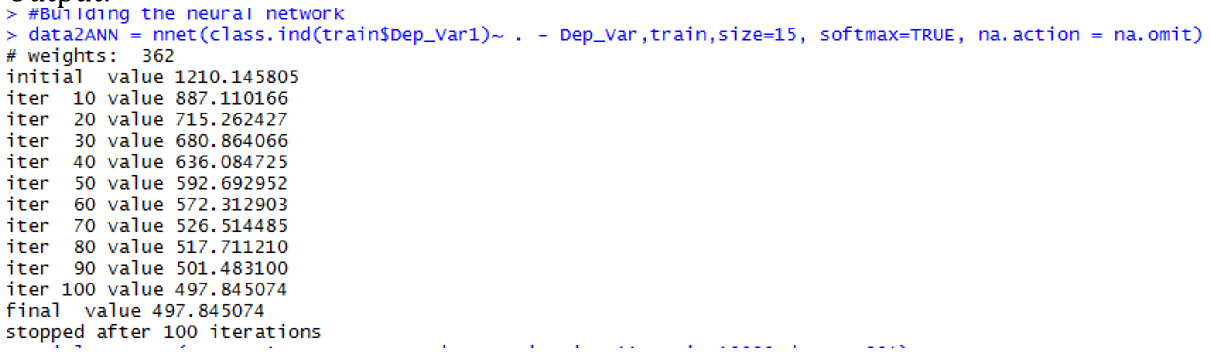
Neural Network:

- On splitting the data in test and train and running the classification model on the dataset:



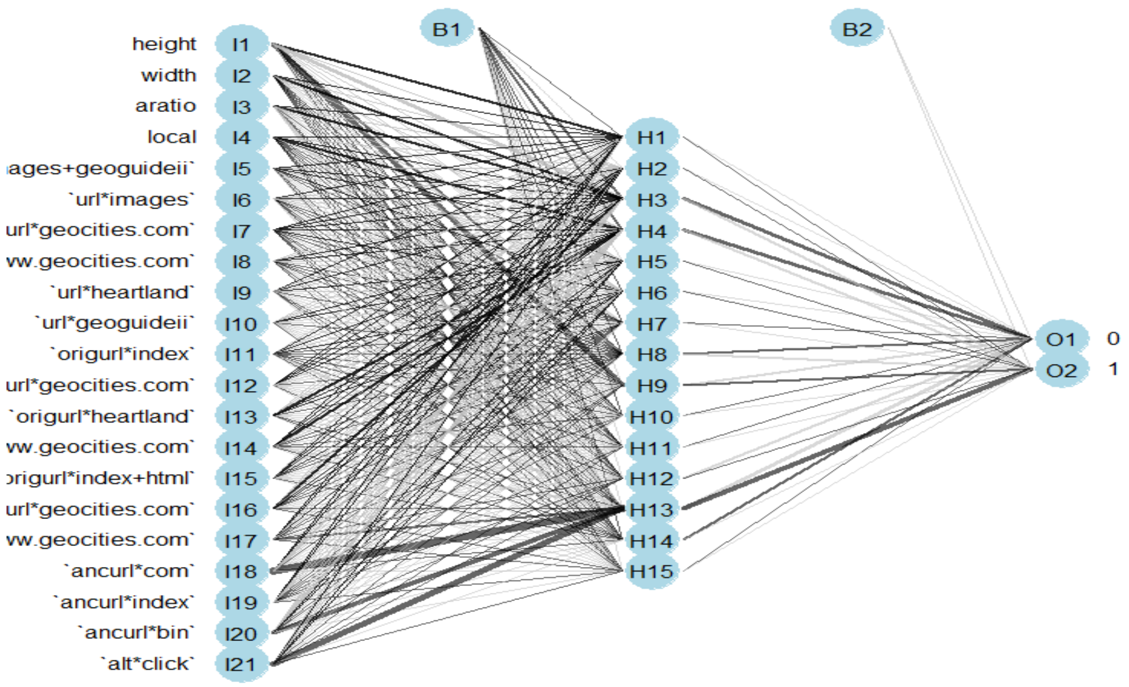
```
#Partitioning the dataset into training and test
index <- sample(1:nrow(data2),round(0.75*nrow(data2)))
train <- data2[index,]
test <- data2[-index,]
#Building the neural network
data2ANN = nnet(class.ind(train$Dep_Var1)~ . - Dep_Var,train,size=15, softmax=TRUE, na.action = na.omit)
trialann=nnet(Dep_Var1~ . - Dep_Var,data=train,size=11,maxit=10000,decay=.001)
predict(data2ANN, train, type="class")
plotnet(data2ANN , alpha=0.6)
```

Output:



```
> #Building the neural network
> data2ANN = nnet(class.ind(train$Dep_Var1)~ . - Dep_Var,train,size=15, softmax=TRUE, na.action = na.omit)
# weights:  362
initial  value 1210.145805
iter  10 value 887.110166
iter  20 value 715.262427
iter  30 value 680.864066
iter  40 value 636.084725
iter  50 value 592.692952
iter  60 value 572.312903
iter  70 value 526.514485
iter  80 value 517.711210
iter  90 value 501.483100
iter 100 value 497.845074
final  value 497.845074
stopped after 100 iterations
```

Neural Network:



- The confusion Matrix generated:

```
#confusion matrix
table(predict(data2ANN,test, type="class"),test$Dep_Var1)
```

Output:

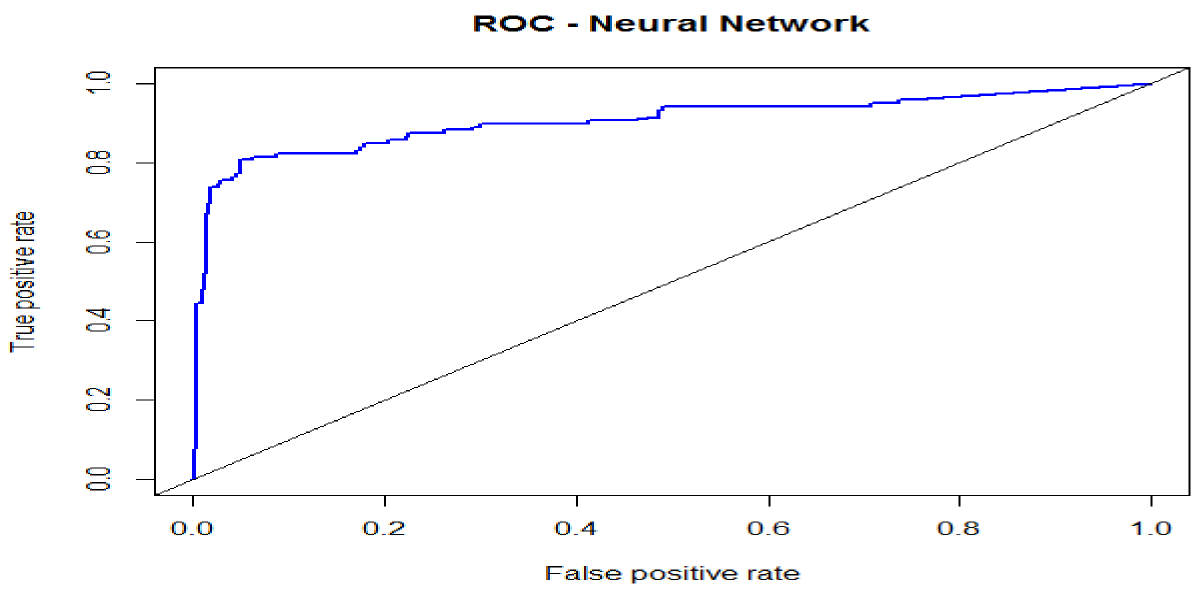
```
> #confusion matrix
> table(predict(data2ANN,test, type="class"),test$Dep_Var1)

      0   1
  0 691  34
  1  10  85
> 
```

```
Error rate: 0.05365854
Accuracy: 0.9463415
Specificity: 0.9531034
Precision: 0.7142857
```

- The ROC curve that is generated:

```
#ROCR
library(ROCR)
pred = prediction(predict(trialann,newdata=test,type="raw"),test$Dep_Var1)
perf = performance(pred,"tpr","fpr")
plot(perf,lwd=2,col="blue",main="ROC - Neural Network")
abline(a=0,b=1)
```

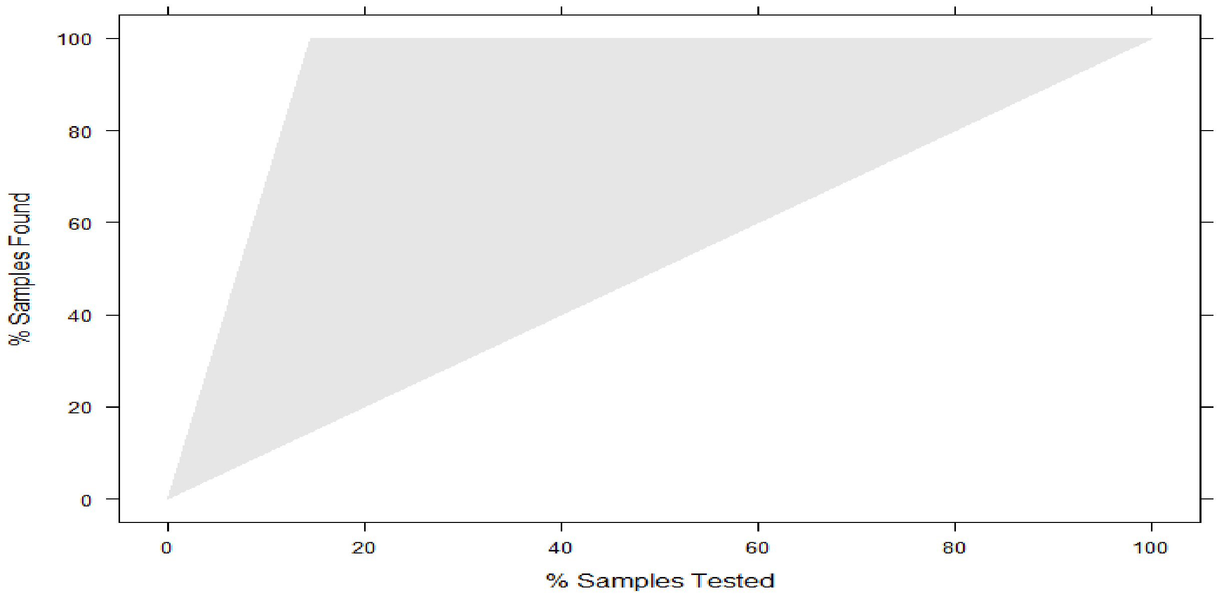
Output:



- Lift Chart on the neural network is generated is as follows

```
#lift curve
test$probs<-predict(data2ANN,test, type="class")
test$prob<- sort(test$probs, decreasing = T)
lift<- lift(Dep_Var ~ prob, data = test)
lift
xyplot(lift, plot="gain")
```

Output:

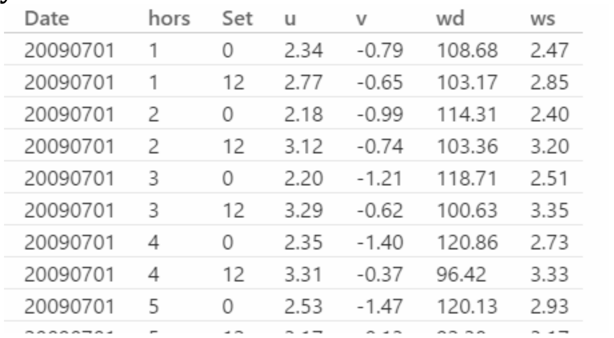


Selection of the appropriate model:

After running classification trees model, logistic regression model and neural network model, the best results are obtained with the classification tree.
Even though the classification tree and the neural network has close values for Accuracy and Error but the Precision value for classification tree is better .
If you refer to the classification matrix and the lift, ROC curves above, in comparison to neural network model and logistic model.
Error Rate: How often the predictions are wrong is 4%
Accuracy: How often the predictions are correct is 95%
Precision: If predicted Yes, how often is it right 75%

Question (3)

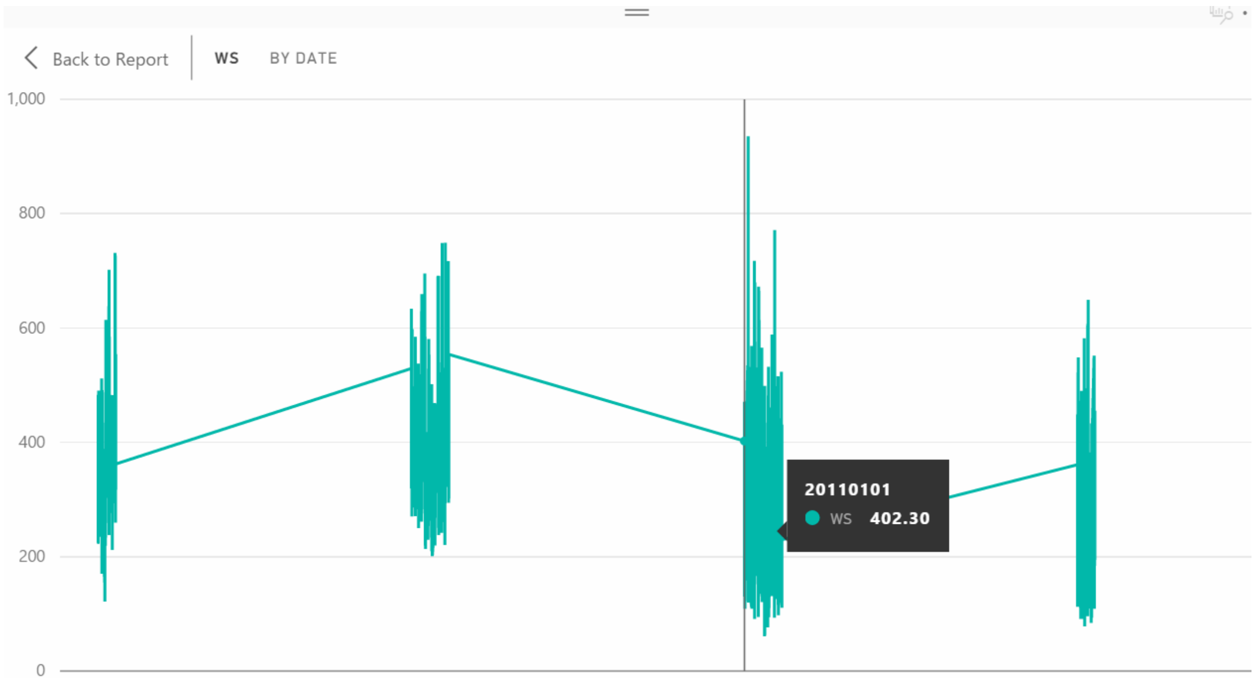
a) Power BI Observations:
Windfarm datasets
After dividing the date column and splitting the data set we observed that forecasts are issued every 12 hours with a forecast horizon of 48 hours, one at 0:00 hour and other at

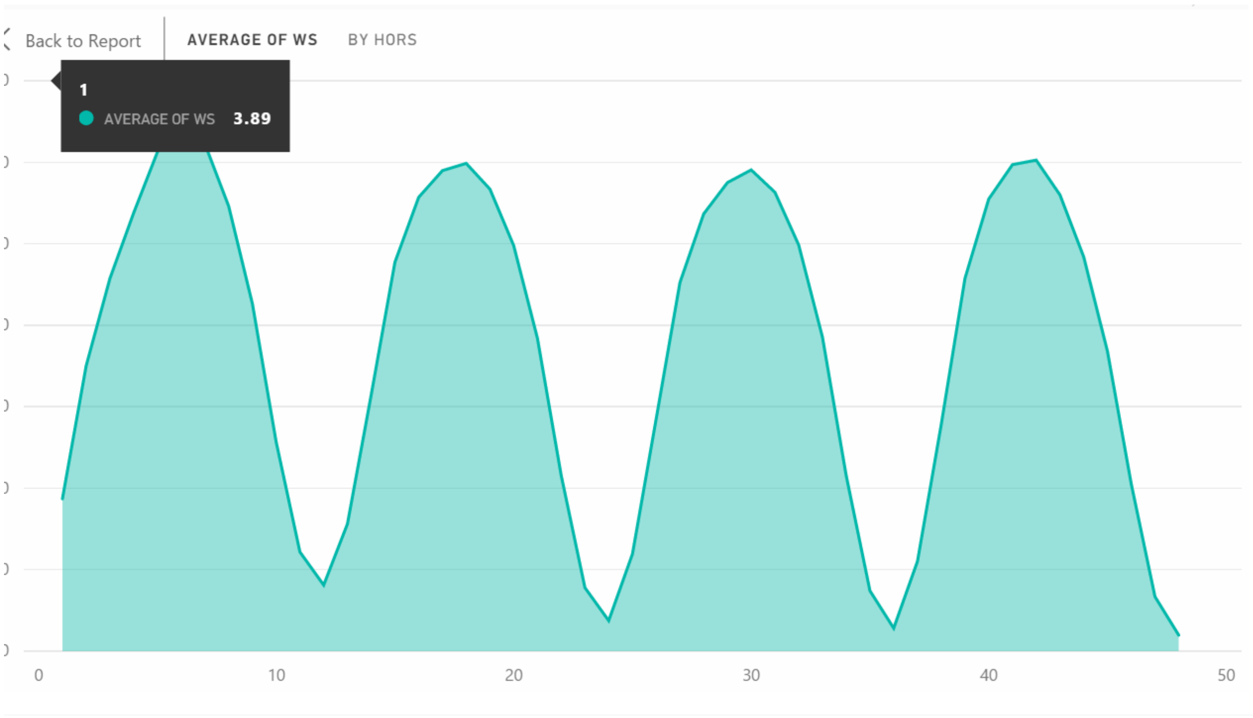


Date	hors	Set	u	v	wd	ws
20090701	1	0	2.34	-0.79	108.68	2.47
20090701	1	12	2.77	-0.65	103.17	2.85
20090701	2	0	2.18	-0.99	114.31	2.40
20090701	2	12	3.12	-0.74	103.36	3.20
20090701	3	0	2.20	-1.21	118.71	2.51
20090701	3	12	3.29	-0.62	100.63	3.35
20090701	4	0	2.35	-1.40	120.86	2.73
20090701	4	12	3.31	-0.37	96.42	3.33
20090701	5	0	2.53	-1.47	120.13	2.93

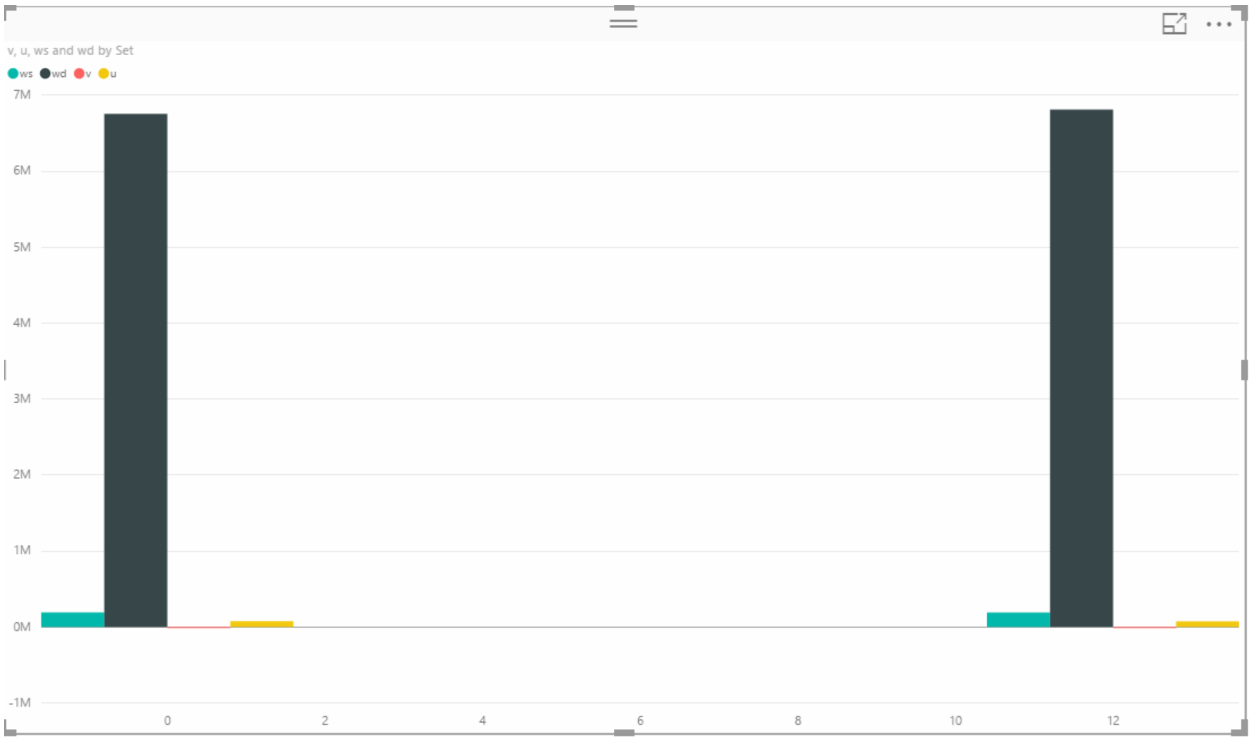
12:00 hour

The pattern of average wind speed is not consistent.

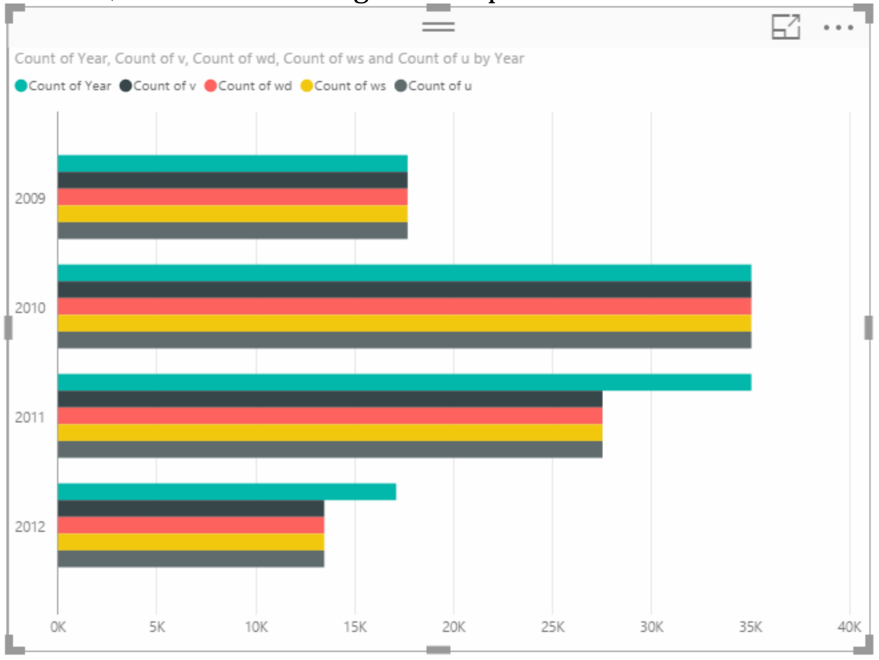




Distribution of the zonal and meridional wind components, wind speed and direction is similar as well.

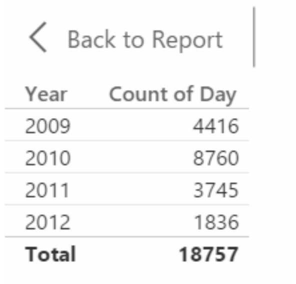


Observation regarding missing values:
The period between 2009/7/1 and 2010/12/31 was for model identification and training period (full data was available), while the remaining dataset i.e. from 2011/1/1 to 2012/6/28 was for evaluation which had many 48 hour missing periods at 36 hours interval , that were the target of the prediction.



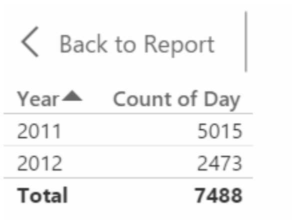
Observations on training data :
There is drop in count of records in 2011 and 2012 compared to benchmark counts.

Train.csv count


Back to Report

Year	Count of Day
2009	4416
2010	8760
2011	3745
2012	1836
Total	**18757**

Benchmark.csv count

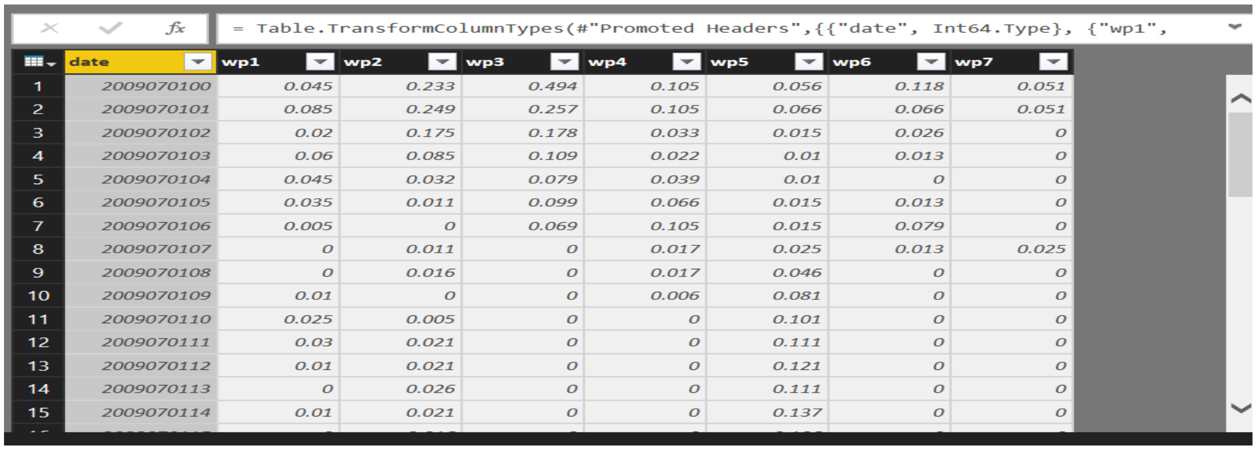


Back to Report

Year	Count of Day
2011	5015
2012	2473
Total	**7488**

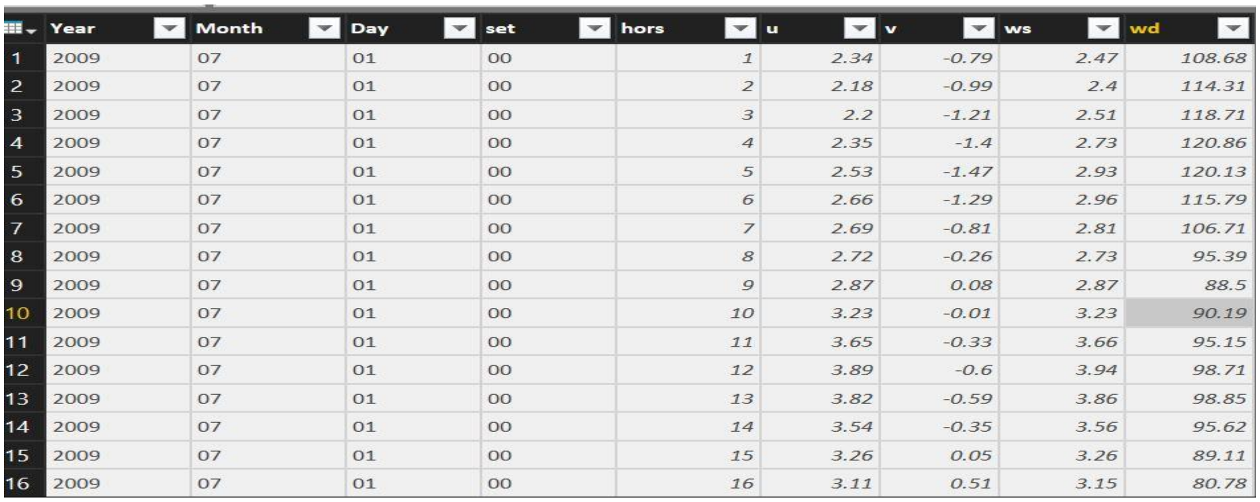
1. Features creation:

   The provided dataset did contain few features that we added explicitly. To do so, the files “train.csv” and “benchmark.csv” were processed creating one entry for each date and each farm. Additionally, the features hour, month, and year were created by splitting the date column:



```
= Table.TransformColumnTypes(#"Promoted Headers",{{"date", Int64.Type}, {"wp1",
```

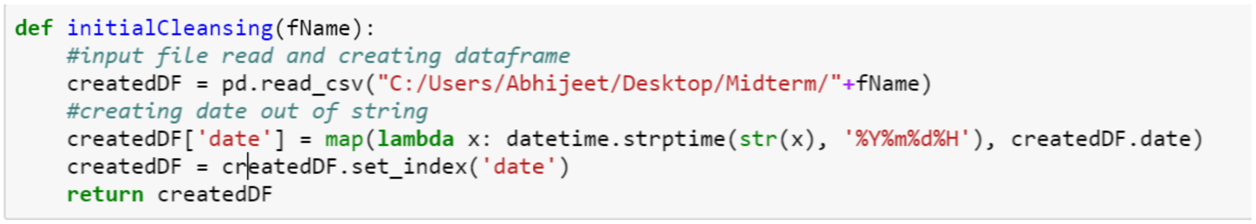
	date	wp1	wp2	wp3	wp4	wp5	wp6	wp7
1	2009070100	0.045	0.233	0.494	0.105	0.056	0.118	0.051
2	2009070101	0.085	0.249	0.257	0.105	0.066	0.066	0.051
3	2009070102	0.02	0.175	0.178	0.033	0.015	0.026	0
4	2009070103	0.06	0.085	0.109	0.022	0.01	0.013	0
5	2009070104	0.045	0.032	0.079	0.039	0.01	0	0
6	2009070105	0.035	0.011	0.099	0.066	0.015	0.013	0
7	2009070106	0.005	0	0.069	0.105	0.015	0.079	0
8	2009070107	0	0.011	0	0.017	0.025	0.013	0.025
9	2009070108	0	0.016	0	0.017	0.046	0	0
10	2009070109	0.01	0	0	0.006	0.081	0	0
11	2009070110	0.025	0.005	0	0	0.101	0	0
12	2009070111	0.03	0.021	0	0	0.111	0	0
13	2009070112	0.01	0.021	0	0	0.121	0	0
14	2009070113	0	0.026	0	0	0.111	0	0
15	2009070114	0.01	0.021	0	0	0.137	0	0



	Year	Month	Day	set	hors	u	v	ws	wd
1	2009	07	01	00	1	2.34	-0.79	2.47	108.68
2	2009	07	01	00	2	2.18	-0.99	2.4	114.31
3	2009	07	01	00	3	2.2	-1.21	2.51	118.71
4	2009	07	01	00	4	2.35	-1.4	2.73	120.86
5	2009	07	01	00	5	2.53	-1.47	2.93	120.13
6	2009	07	01	00	6	2.66	-1.29	2.96	115.79
7	2009	07	01	00	7	2.69	-0.81	2.81	106.71
8	2009	07	01	00	8	2.72	-0.26	2.73	95.39
9	2009	07	01	00	9	2.87	0.08	2.87	88.5
10	2009	07	01	00	10	3.23	-0.01	3.23	90.19
11	2009	07	01	00	11	3.65	-0.33	3.66	95.15
12	2009	07	01	00	12	3.89	-0.6	3.94	98.71
13	2009	07	01	00	13	3.82	-0.59	3.86	98.85
14	2009	07	01	00	14	3.54	-0.35	3.56	95.62
15	2009	07	01	00	15	3.26	0.05	3.26	89.11
16	2009	07	01	00	16	3.11	0.51	3.15	80.78

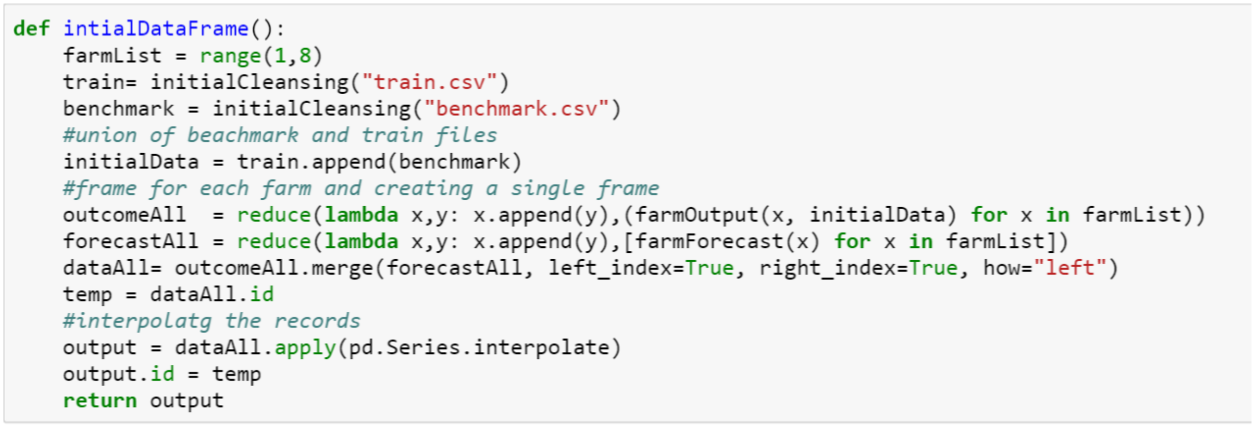
(A)Data Cleansing and pre-processing

Step 1: Converting the date column from string to Date



```
def initialCleansing(fName):
    #input file read and creating dataframe
    createdDF = pd.read_csv("C:/Users/Abhijeet/Desktop/Midterm/"+fName)
    #creating date out of string
    createdDF['date'] = map(lambda x: datetime.strptime(str(x), '%Y%m%d%H'), createdDF.date)
    createdDF = createdDF.set_index('date')
    return createdDF
```

Step 2: Creating basic dataframes of train and benchmark files



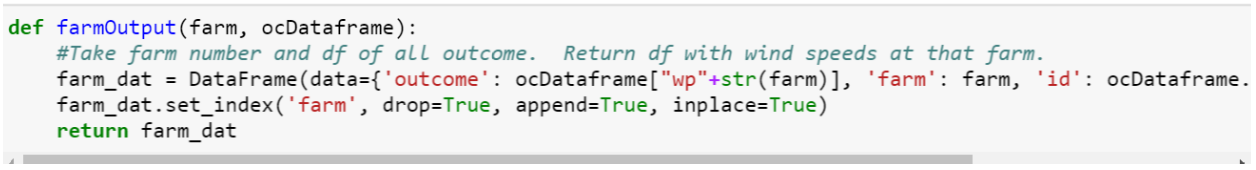
```
def intialDataFrame():
    farmList = range(1,8)
    train= initialCleansing("train.csv")
    benchmark = initialCleansing("benchmark.csv")
    #union of beachmark and train files
    initialData = train.append(benchmark)
    #frame for each farm and creating a single frame
    outcomeAll  = reduce(lambda x,y: x.append(y),(farmOutput(x, initialData) for x in farmList))
    forecastAll = reduce(lambda x,y: x.append(y),[farmForecast(x) for x in farmList])
    dataAll= outcomeAll.merge(forecastAll, left_index=True, right_index=True, how="left")
    temp = dataAll.id
    #interpolatg the records
    output = dataAll.apply(pd.Series.interpolate)
    output.id = temp
    return output
```

To fill missing values in each column, we have used linear interpolation. Missing values from each column are filled except for id(to distinguish between training data set and evaluation data set)

Functions used for making basic dataframe:

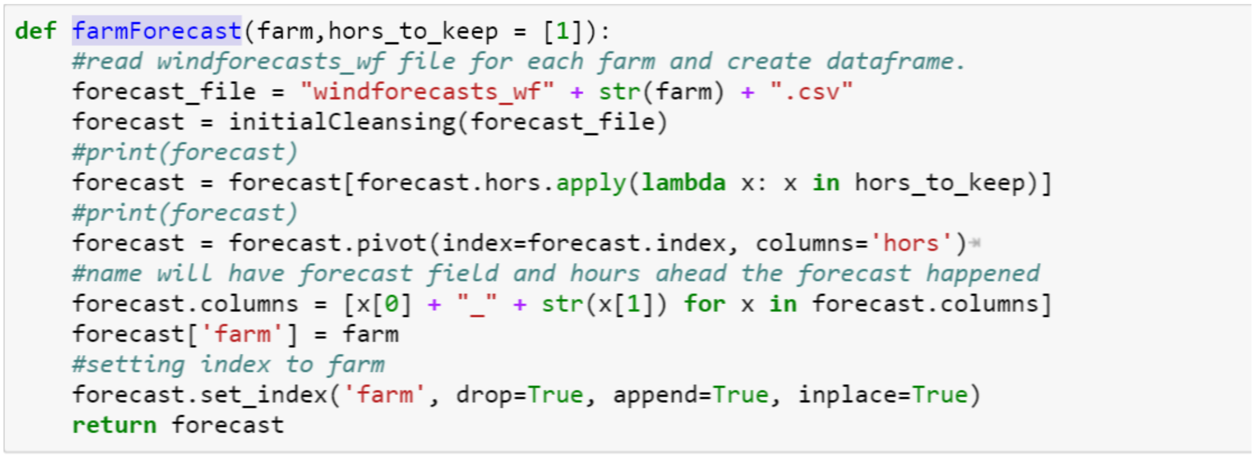
Functions used for making basic dataframe:

(a)farmOutput(farm,ocDataFrame): This function takes site number and creates a dataframe with wind speeds at respective wind farms.



```
def farmOutput(farm, ocDataframe):
    #Take farm number and df of all outcome.  Return df with wind speeds at that farm.
    farm_dat = DataFrame(data={'outcome': ocDataframe["wp"+str(farm)], 'farm': farm, 'id': ocDataframe.
    farm_dat.set_index('farm', drop=True, append=True, inplace=True)
    return farm_dat
```

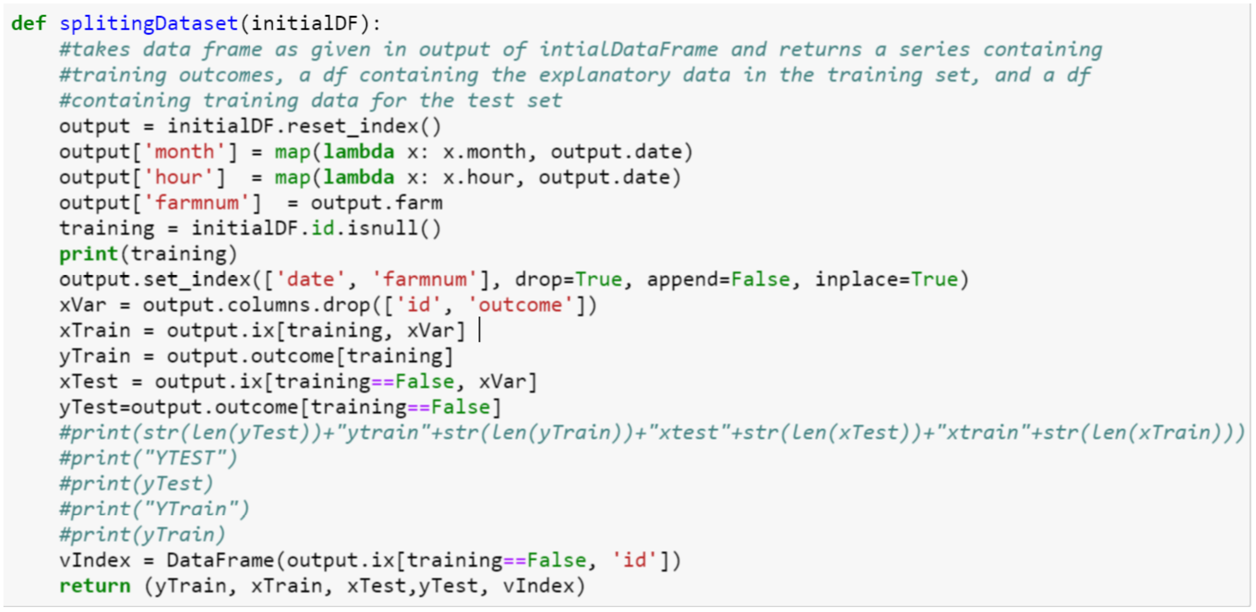
(b) farmForecast(farm,hors_to_keep=[1]): This functions reads forecast file for every farm_num(wind farm) and returns data frame with forecast data. We are using the existing features provided like wind speed etc from windforecast files



```
def farmForecast(farm,hors_to_keep = [1]):
    #read windforecasts_wf file for each farm and create dataframe.
    forecast_file = "windforecasts_wf" + str(farm) + ".csv"
    forecast = initialCleansing(forecast_file)
    #print(forecast)
    forecast = forecast[forecast.hors.apply(lambda x: x in hors_to_keep)]
    #print(forecast)
    forecast = forecast.pivot(index=forecast.index, columns='hors')
    #name will have forecast field and hours ahead the forecast happened
    forecast.columns = [x[0] + "_" + str(x[1]) for x in forecast.columns]
    forecast['farm'] = farm
    #setting index to farm
    forecast.set_index('farm', drop=True, append=True, inplace=True)
    return forecast
```

(2) Regression Models

Partitioning dataset into training and test:

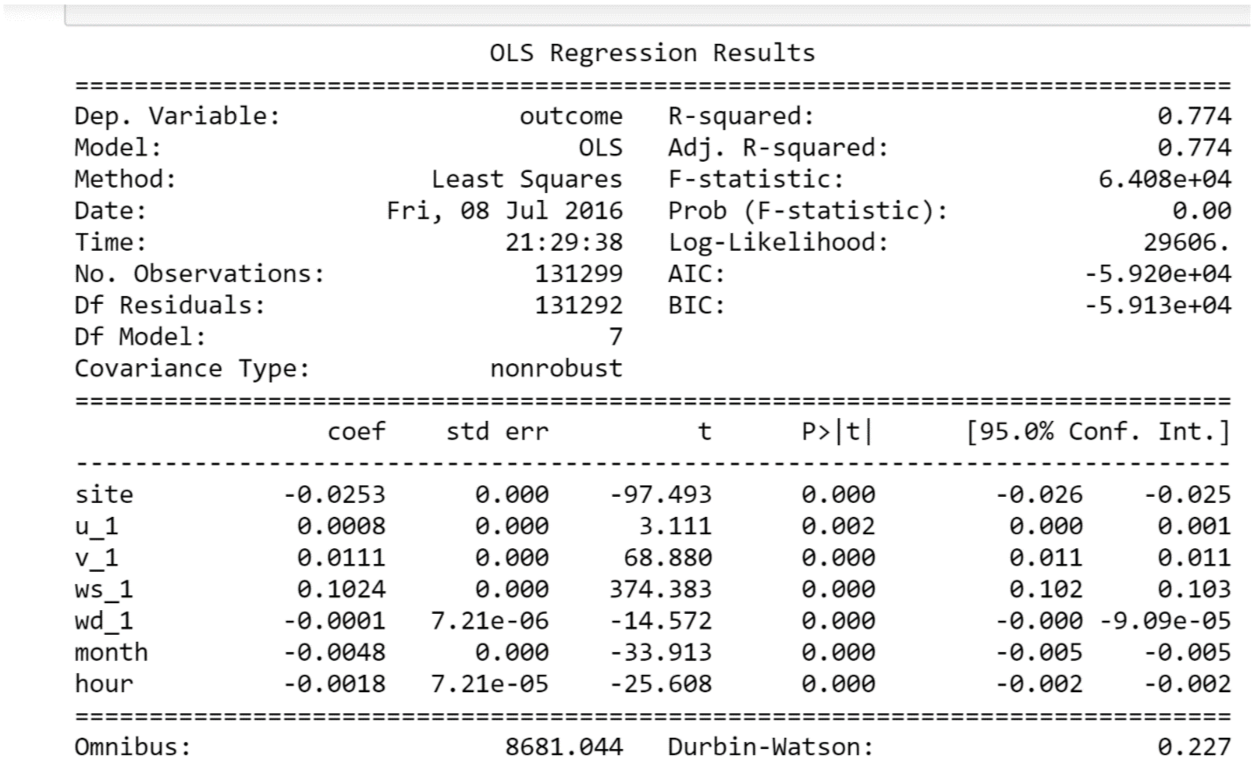


```
def splitingDataset(initialDF):
    #takes data frame as given in output of intialDataFrame and returns a series containing
    #training outcomes, a df containing the explanatory data in the training set, and a df
    #containing training data for the test set
    output = initialDF.reset_index()
    output['month'] = map(lambda x: x.month, output.date)
    output['hour']  = map(lambda x: x.hour, output.date)
    output['farmnum']  = output.farm
    training = initialDF.id.isnull()
    print(training)
    output.set_index(['date', 'farmnum'], drop=True, append=False, inplace=True)
    xVar = output.columns.drop(['id', 'outcome'])
    xTrain = output.ix[training, xVar]
    yTrain = output.outcome[training]
    xTest = output.ix[training==False, xVar]
    yTest=output.outcome[training==False]
    #print(str(len(yTest))+"ytrain"+str(len(yTrain))+"xtest"+str(len(xTest))+"xtrain"+str(len(xTrain)))
    #print("YTEST")
    #print(yTest)
    #print("YTrain")
    #print(yTrain)
    vIndex = DataFrame(output.ix[training==False, 'id'])
    return (yTrain, xTrain, xTest,yTest, vIndex)
```

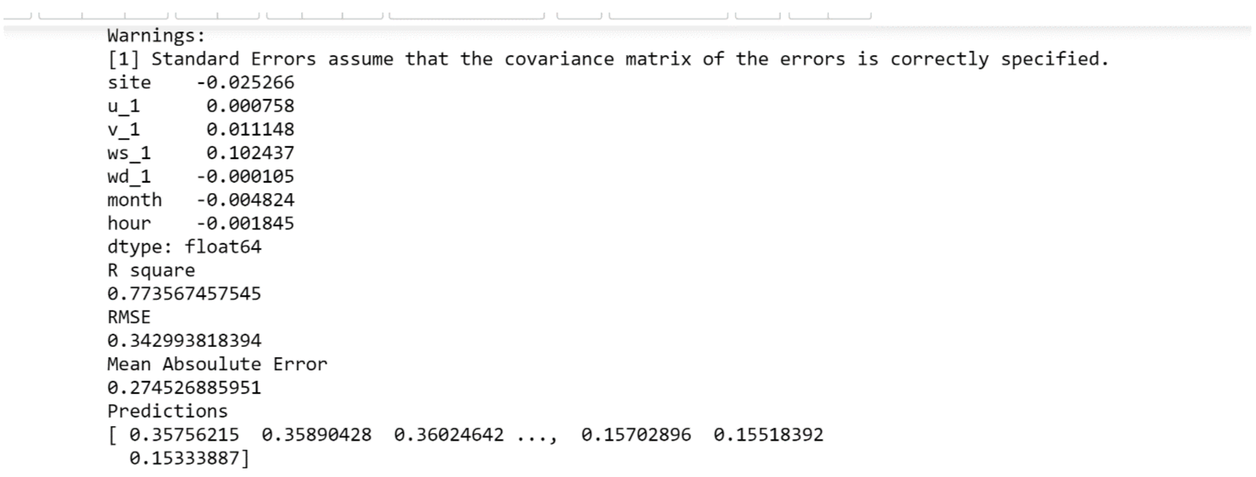
(a)Ordinary Least Squares Regression
Function to predict the values using

```
def olsPrediction(yTrain, xTrain, xTest,yTest):
    olsPr= sm.OLS(yTrain, xTrain).fit()
    predictions = olsPr.predict(xTest)
    print("##########################################################")
    print("#########Performance Metrics for Linear Regression #######")
    print("##########################################################")
    print(olsPr.summary())
    print(olsPr.params)
    print("R square ")
    print(olsPr.rsquared)
    print("RMSE")
    print(mean_squared_error(yTest, predictions)**0.5)
    print("Mean Absoulute Error")
    print(mean_absolute_error(yTest, predictions))
    return predictions
```

Performance metrics

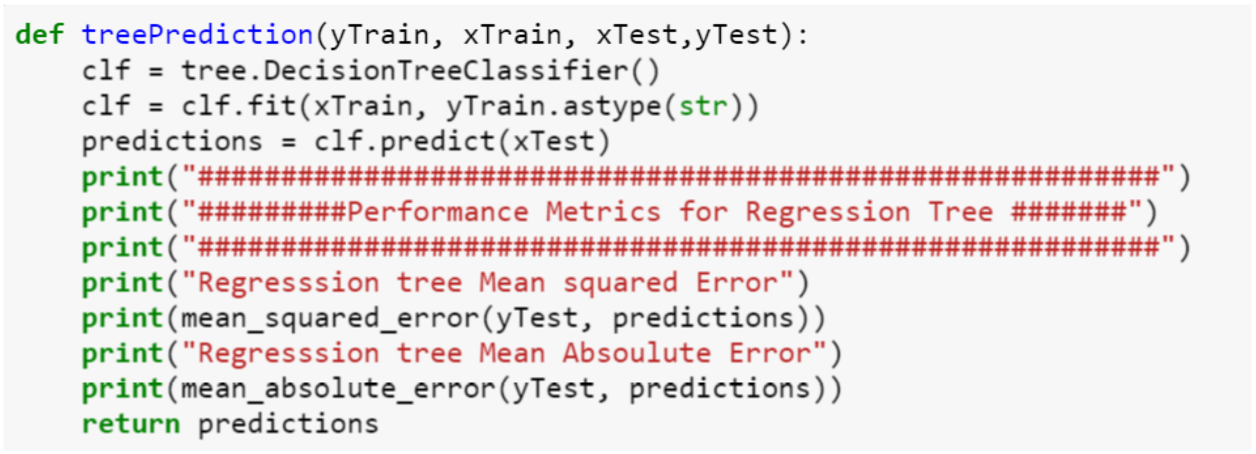


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                outcome   R-squared:                       0.774
Model:                            OLS   Adj. R-squared:                  0.774
Method:                 Least Squares   F-statistic:                 6.408e+04
Date:                Fri, 08 Jul 2016   Prob (F-statistic):               0.00
Time:                        21:29:38   Log-Likelihood:                 29606.
No. Observations:              131299   AIC:                        -5.920e+04
Df Residuals:                  131292   BIC:                        -5.913e+04
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
site          -0.0253      0.000    -97.493      0.000        -0.026    -0.025
u_1            0.0008      0.000      3.111      0.002         0.000     0.001
v_1            0.0111      0.000     68.880      0.000         0.011     0.011
ws_1           0.1024      0.000    374.383      0.000         0.102     0.103
wd_1          -0.0001   7.21e-06    -14.572      0.000        -0.000 -9.09e-05
month         -0.0048      0.000    -33.913      0.000        -0.005    -0.005
hour          -0.0018   7.21e-05    -25.608      0.000        -0.002    -0.002
==============================================================================
Omnibus:                     8681.044   Durbin-Watson:                   0.227
```

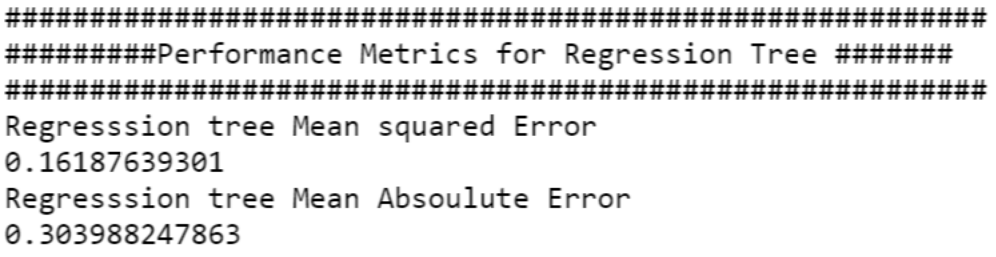


```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
site    -0.025266
u_1      0.000758
v_1      0.011148
ws_1     0.102437
wd_1    -0.000105
month   -0.004824
hour    -0.001845
dtype: float64
R square
0.773567457545
RMSE
0.342993818394
Mean Absoulute Error
0.274526885951
Predictions
[ 0.35756215  0.35890428  0.36024642 ...,  0.15702896  0.15518392
  0.15333887]
```

b) Regression Tree
Function to build the model and predict values

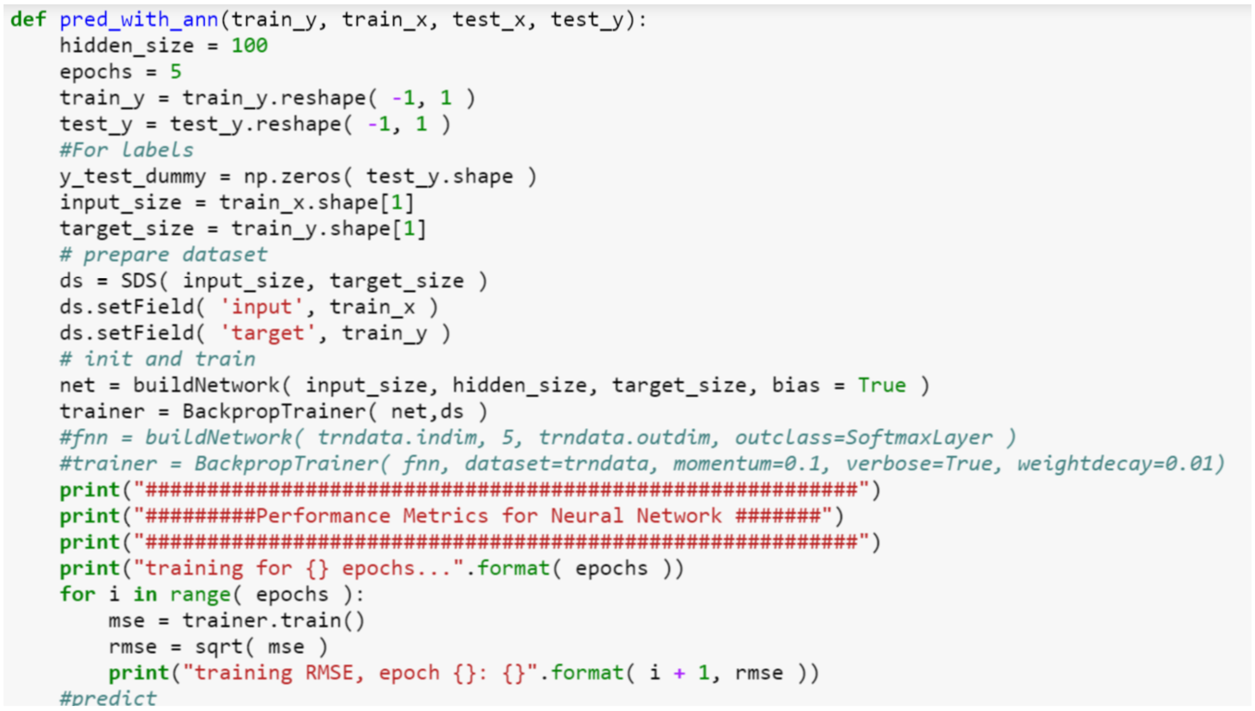


```python
def treePrediction(yTrain, xTrain, xTest,yTest):
    clf = tree.DecisionTreeClassifier()
    clf = clf.fit(xTrain, yTrain.astype(str))
    predictions = clf.predict(xTest)
    print("#########################################################")
    print("#########Performance Metrics for Regression Tree #######")
    print("#########################################################")
    print("Regresssion tree Mean squared Error")
    print(mean_squared_error(yTest, predictions))
    print("Regresssion tree Mean Absoulute Error")
    print(mean_absolute_error(yTest, predictions))
    return predictions
```

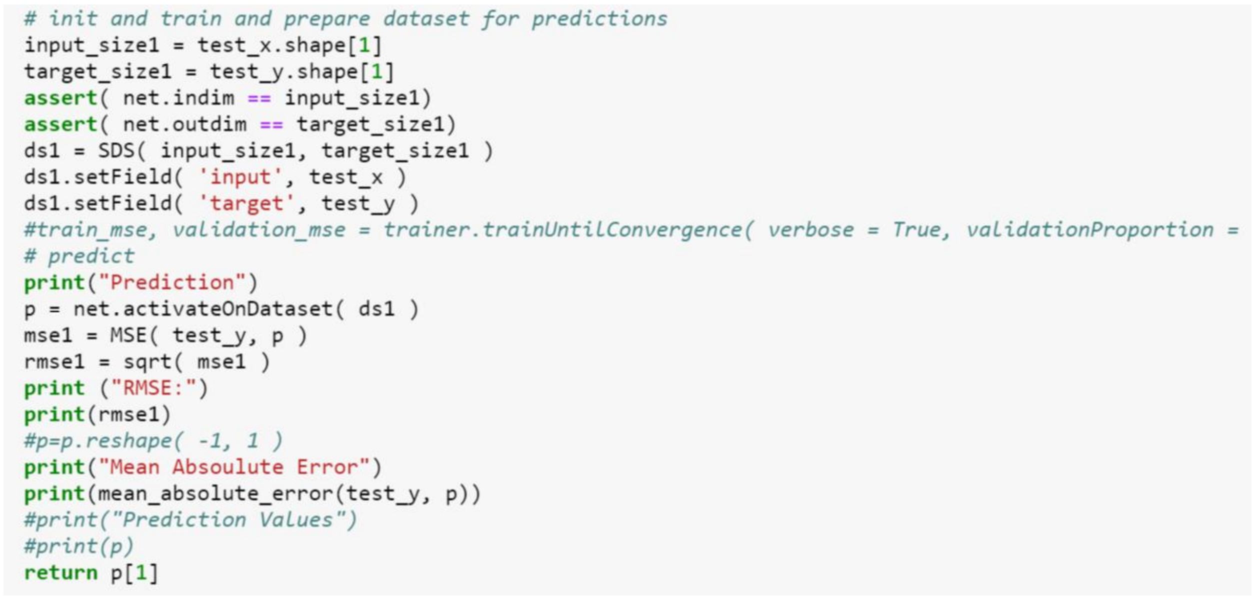


```
#########################################################
#########Performance Metrics for Regression Tree #######
#########################################################
Regresssion tree Mean squared Error
0.16187639301
Regresssion tree Mean Absoulute Error
0.303988247863
```

c)Neural Network
Function to build the model and predict values

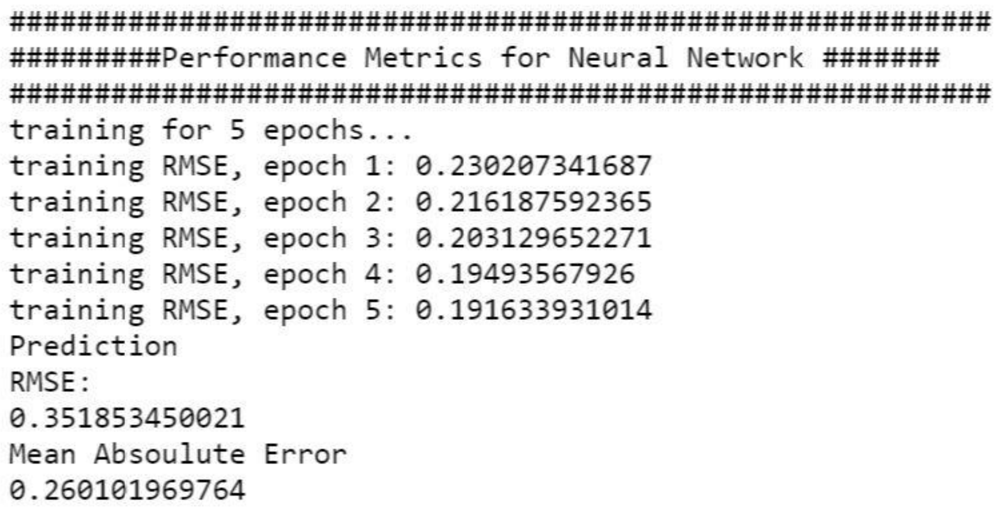


```
def pred_with_ann(train_y, train_x, test_x, test_y):
    hidden_size = 100
    epochs = 5
    train_y = train_y.reshape( -1, 1 )
    test_y = test_y.reshape( -1, 1 )
    #For labels
    y_test_dummy = np.zeros( test_y.shape )
    input_size = train_x.shape[1]
    target_size = train_y.shape[1]
    # prepare dataset
    ds = SDS( input_size, target_size )
    ds.setField( 'input', train_x )
    ds.setField( 'target', train_y )
    # init and train
    net = buildNetwork( input_size, hidden_size, target_size, bias = True )
    trainer = BackpropTrainer( net,ds )
    #fnn = buildNetwork( trndata.indim, 5, trndata.outdim, outclass=SoftmaxLayer )
    #trainer = BackpropTrainer( fnn, dataset=trndata, momentum=0.1, verbose=True, weightdecay=0.01)
    print("########################################################")
    print("#########Performance Metrics for Neural Network #######")
    print("########################################################")
    print("training for {} epochs...".format( epochs ))
    for i in range( epochs ):
        mse = trainer.train()
        rmse = sqrt( mse )
        print("training RMSE, epoch {}: {}".format( i + 1, rmse ))
    #predict
```



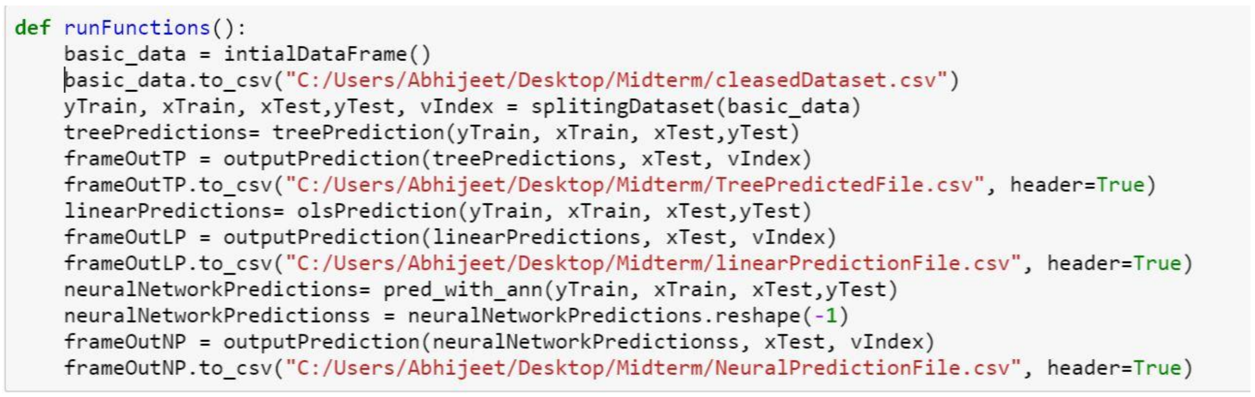
```
# init and train and prepare dataset for predictions
input_size1 = test_x.shape[1]
target_size1 = test_y.shape[1]
assert( net.indim == input_size1)
assert( net.outdim == target_size1)
ds1 = SDS( input_size1, target_size1 )
ds1.setField( 'input', test_x )
ds1.setField( 'target', test_y )
#train_mse, validation_mse = trainer.trainUntilConvergence( verbose = True, validationProportion =
# predict
print("Prediction")
p = net.activateOnDataset( ds1 )
mse1 = MSE( test_y, p )
rmse1 = sqrt( mse1 )
print ("RMSE:")
print(rmse1)
#p=p.reshape( -1, 1 )
print("Mean Absoulute Error")
print(mean_absolute_error(test_y, p))
#print("Prediction Values")
#print(p)
return p[1]
```

Performance Metrics



```
##########################################################
#########Performance Metrics for Neural Network #######
##########################################################
training for 5 epochs...
training RMSE, epoch 1: 0.230207341687
training RMSE, epoch 2: 0.216187592365
training RMSE, epoch 3: 0.203129652271
training RMSE, epoch 4: 0.19493567926
training RMSE, epoch 5: 0.191633931014
Prediction
RMSE:
0.351853450021
Mean Absoulute Error
0.260101969764
```

To run the script:



```
def runFunctions():
    basic_data = intialDataFrame()
    basic_data.to_csv("C:/Users/Abhijeet/Desktop/Midterm/cleasedDataset.csv")
    yTrain, xTrain, xTest,yTest, vIndex = splitingDataset(basic_data)
    treePredictions= treePrediction(yTrain, xTrain, xTest,yTest)
    frameOutTP = outputPrediction(treePredictions, xTest, vIndex)
    frameOutTP.to_csv("C:/Users/Abhijeet/Desktop/Midterm/TreePredictedFile.csv", header=True)
    linearPredictions= olsPrediction(yTrain, xTrain, xTest,yTest)
    frameOutLP = outputPrediction(linearPredictions, xTest, vIndex)
    frameOutLP.to_csv("C:/Users/Abhijeet/Desktop/Midterm/linearPredictionFile.csv", header=True)
    neuralNetworkPredictions= pred_with_ann(yTrain, xTrain, xTest,yTest)
    neuralNetworkPredictionss = neuralNetworkPredictions.reshape(-1)
    frameOutNP = outputPrediction(neuralNetworkPredictionss, xTest, vIndex)
    frameOutNP.to_csv("C:/Users/Abhijeet/Desktop/Midterm/NeuralPredictionFile.csv", header=True)
```

Model selected for prediction:

We have selected Regression Tree model for prediction as compared to OLS and ANN, RMSE and MEA values were less

	OLS Regression	Regression Tree	Artificial Neural Network
MEA	0.2745	0.3039	0.2601
RMSE	0.35299	0.161	0.3518