|  |
| --- |
| Photo displaying partial image of two pie charts on a canvas-textured page |
| SUPERVISED LEARNING  Bank Marketing Dataset |
| |  |  |  | | --- | --- | --- | |  |  |  | |

## Data and Goals

The data under study here is called Bank Marketing Dataset obtained from Kaggle.com. In this dataset, each record represents a phone call made to a potential customer. Each column represents an attribute that was observed and recorded from the phone call. In the Bank Marketing Dataset, we have **11162** observations, with **seventeen** **features**. The seventeen features are briefly described in Table 1, were in the left column we have the original feature name in the dataset, and in the right column its description, mentioning also if the feature is numeric, categorial, and with how many levels (if categorical, of course). The first one called of y is the response, the desired target. The other features are presented in the same order that they appear in the dataset. We have explored different classification models in pursuit of championing the best model for our business case.

The business question we are trying to answer is:

**Potential customer exhibiting what kind of behaviors are more likely to subscribe to a term deposit?**

Our business problem is to devise marketing strategy for the bank based on the behavioural data collected. As logistic model will have high interpretability as compared to the random forest and boosting, we may consider using logistic regression for our business case.

Exploratory Data Analysis

**Dataset**

|  |  |  |
| --- | --- | --- |
| Sr. No | Feature | Description |
|  | Age | numeric |
|  | Job | type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown') |
|  | Marital | marital status (categorical: 'divorced', 'married', 'single', 'unknown'; note: 'divorced' means divorced or widowed) |
|  | Education | categorical: basic.4y','basic.6y','basic.9y','high. school','illiterate','professional. course’, ‘university. degree’, ‘unknown' |
|  | Default | has credit in default? (categorical: 'no','yes','unknown') |
|  | Housing | has housing loan? (categorical: 'no','yes','unknown') |
|  | Loan | has personal loan? (categorical: 'no','yes','unknown') |
|  | Contact | contact communication type (categorical: ,'telephone') |
|  | Month | last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec') |
|  | Day\_of\_week | last contact day of the week (categorical: 'mon','tue','wed','thu','fri') |
|  | Duration | last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. |
|  | Campaign | number of contacts performed during this campaign and for this client (numeric, includes last contact) |
|  | PDays | number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means clients was not previously contacted) |
|  | Previous | number of contacts performed before this campaign and for this client (numeric) |
|  | Balance | the amount balance of the user |
|  | Poutcome | outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success') |
|  | Deposit | has the client subscribed a term deposit? (binary: 'yes','no') |

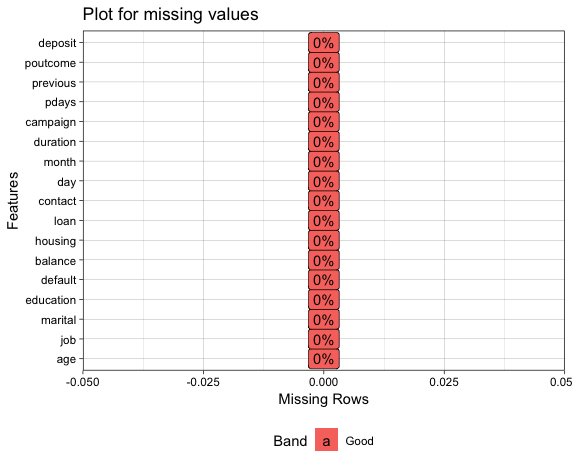
Table 1

**Data Cleaning**

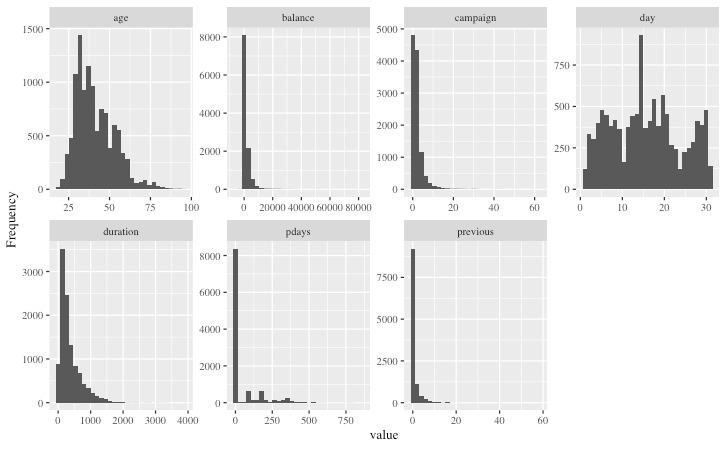
Below, we are checking if our data has any missing values. From the graph we can see that there are no missing values in our dataset, hence no need to clean the data further

plot\_missing(ny, title = 'Plot for missing values', ggtheme = theme\_linedraw(),

theme\_config = list(legend.position = c("bottom")))



aggr\_plot <- aggr(b, col=c('navyblue','red'), numbers=TRUE, sortVars=TRUE, labels=names(df), cex.axis=.7, gap=3, ylab=c("Histogram of missing data","Pattern"))



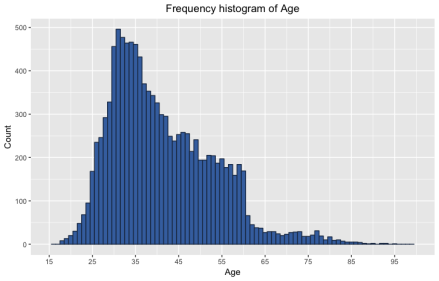
plot\_histogram(b[,-17],ggtheme = theme\_gray(base\_size = 10, base\_family = "serif"))

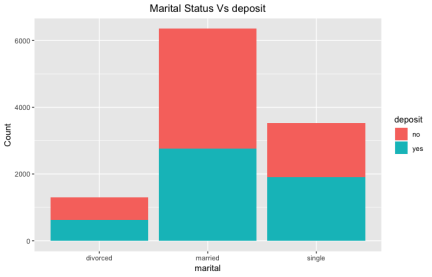
**Bivariate Analysis**

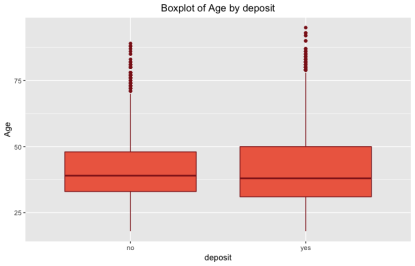
Similarly, bivariate analysis of each of the variable against our dependent variable deposit is shown in the below plots.

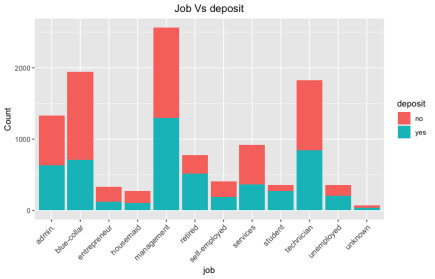
Plotting using the below function for each variable:

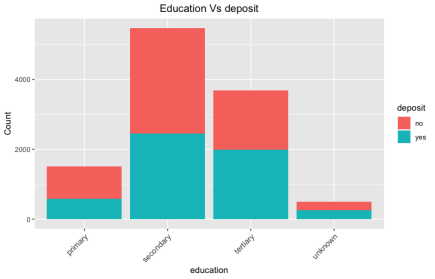
ggplot(b, aes(x = age)) + geom\_histogram(aes(fill = ..count..), binwidth = 1,colour = barlines, fill = barfill)+ scale\_x\_continuous(name = "Age”,breaks = seq(15,100,10),limits = c(15,100)) + xlab("Age")+ylab("Count")+ ggtitle("Frequency histogram of Age") + theme(plot.title = element\_text(hjust = 0.5))

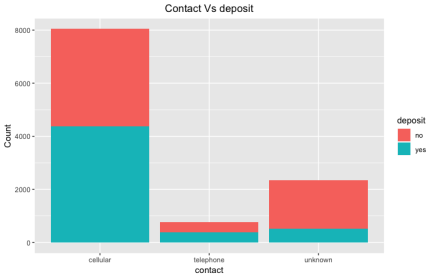


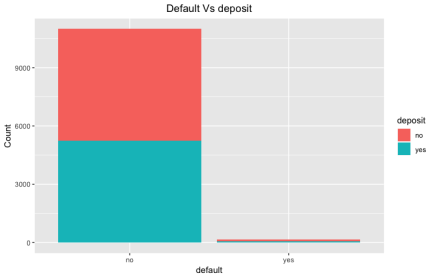


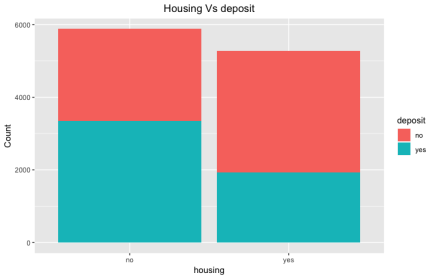


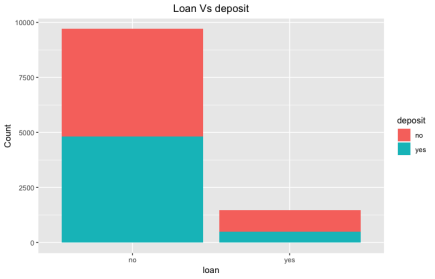


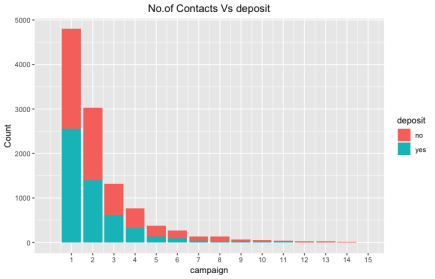


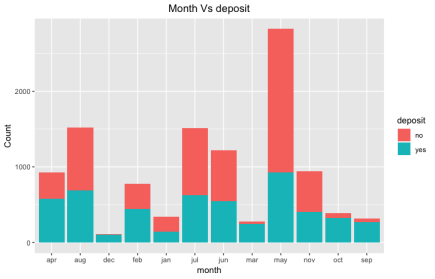


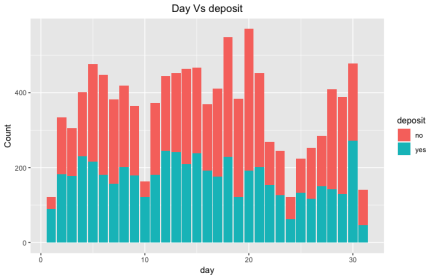


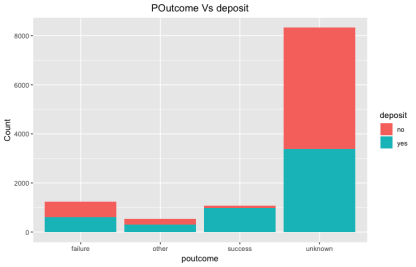


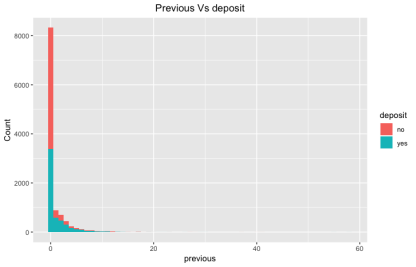


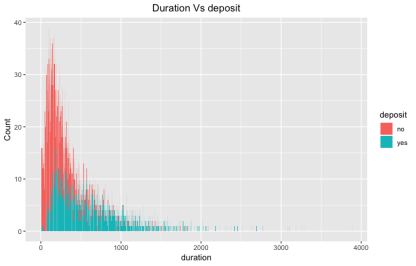


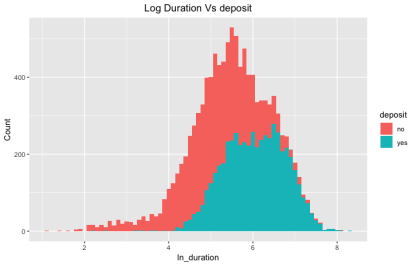


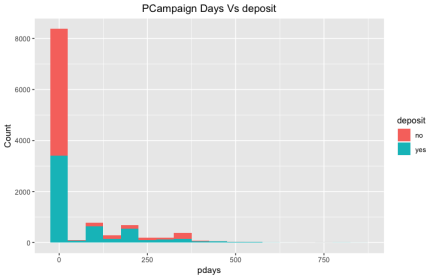












**Inferences from EDA**

Following are some of the inferences obtained through the above plots and EDA:

* 75% of the clients are **younger** than 47 years old.
* Looking at the **marital** variable, it looks like **married** people are more inclined to make a deposit.
* The **average age** for customers making a deposit and not making a deposit is quite similar, between **30 and 50** years old.
* Variable **housing** does not provide sufficient evidence to support a conclusion as there are almost an equal number or deposit and non-deposit makers.
* The **Job** variable has a few categories with higher number of yes than no for deposit, namely, retired student and unemployed
* Variable **Month** shows that Customers are less likely to make a deposit in May, and highly likely for months Apr, Oct, Sept, Mar, Dec, and Feb

## Implementation and Analysis

### **3.1 Logistic Regression Model**

The logistic regression model is a method of predictive analysis. It is usually conducted when the response variable is dichotomous or binary. We can use it to describe the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

#### Creating the Logistic Regression Model

First, the csv file was read into a data-frame in R using the read.csv() command.

Code:

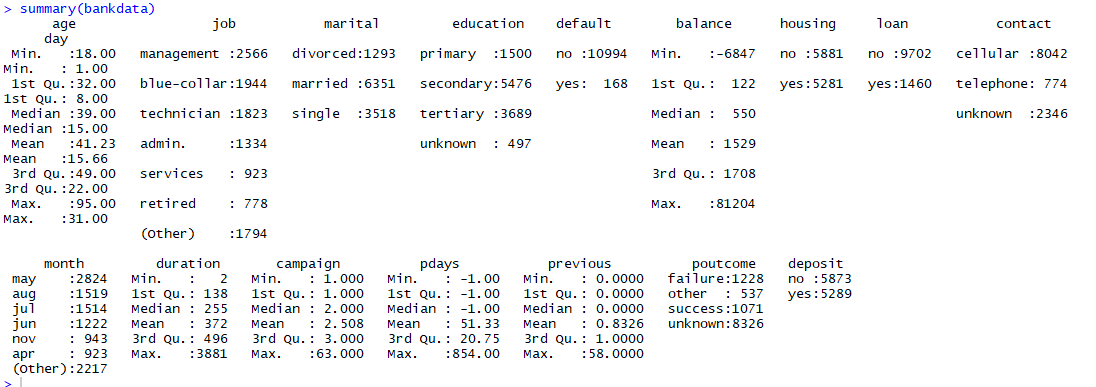
bankdata <- read.csv("bank.csv")

Next, the summary of this data was found so that information about the dataset is handy.

**R Code:**

summary(bankdata)

**Output:**



From the output shown above, we can see the different column names as well as possible factor values for each column. As already discussed in the exploratory data analysis, the binary variable ‘deposit’ is going to be out response variable while all the other variables – namely, age, job, marital, education, default, balance, housing, loan, contact, day, month, duration, campaign, pdays, previous and poutcome are our covariates.

We are hoping to find out which of these covariates actually affect the outcome of deposit and by what measure.

#### Splitting Data into Train and Test

Since the aim here is to perform a predictive analysis, the dataset needs to be split into train and test data, so the model can be tested once it is created. Here, the data was split in an 80-20 ratio into trainData and testData.

**Rcode:**

index <- sample(nrow(bankdata), nrow(bankdata)\*0.80)

trainData = bankdata[index,]

testData = bankdata[-index,]

The code above first indexes 80% of the bankdata dataset. Then it takes all the indexed rows and creates a new dataset with it. This is our training data and it is stored in a data-frame called trainData.

The next time in the code creates testData which is our test dataset. This set will contain all the rows that were not indexed in bankdata.

The dimensions of each of these subsets was then checked.

**R Code:**

> dim(trainData)

[1] 8929 17

> dim(testData)

[1] 2233 17

The code and output above show that the trainData dataset has 8929 rows and 17 columns and the testData dataset gas 2233 rows and 17 columns. The number of columns remains the same because the dataset was split vertically.

#### Creating the Model

Since the response variable is binary, the best course of action is to create a logistic regression model. A logistic regression model can be created using the glm() function as follows -

**R Code:**

> bank\_model1 <- glm(deposit ~ poutcome+previous+pdays+campaign+duration+month+day+contact+loan+housing+balance+education+marital+job+age, data = trainData, family = "binomial")

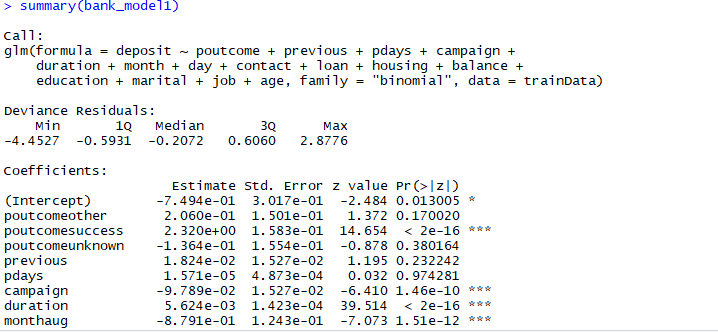
Summary for bank\_model1

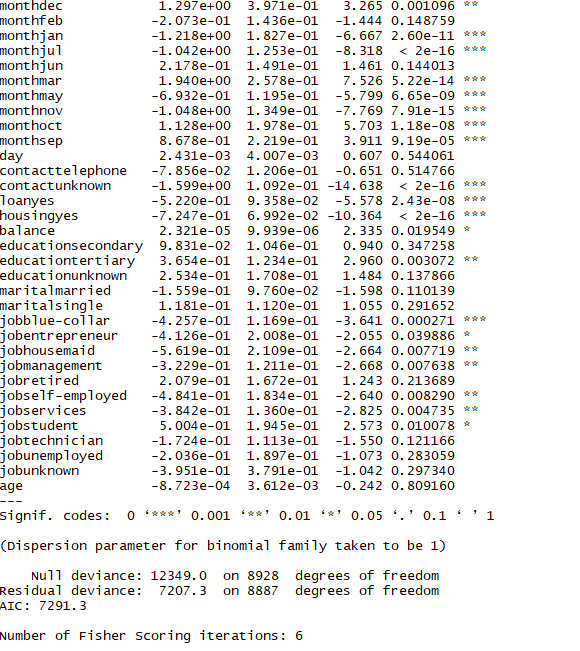
The summary of a model tells us which factors significantly impact our response variable and which ones have little or no impact. The summary() function was used to get the summary for bank\_model1 – containing all original covariates.

**R code:**

> summary(bank\_model1)

Output:





#### Process of Backward Elimination

The process of backward elimination is used to remove covariates which do not significantly impact the outcome of the response variable. These are identified by looking at the p-value in the summary. The threshold for an acceptable p-value is 0.05. From the summary of bank\_model1, it can be seen that the covariates ‘previous’, ‘pdays’, ‘day’, ‘marital’ and ‘age’ have bad p-values. Hence, these covariates can be removed while creating the second model.

#### Model Creation after eliminating covariates

A second model – bank\_model2 was created, containing only covariates with significant p-values.

**R code:**

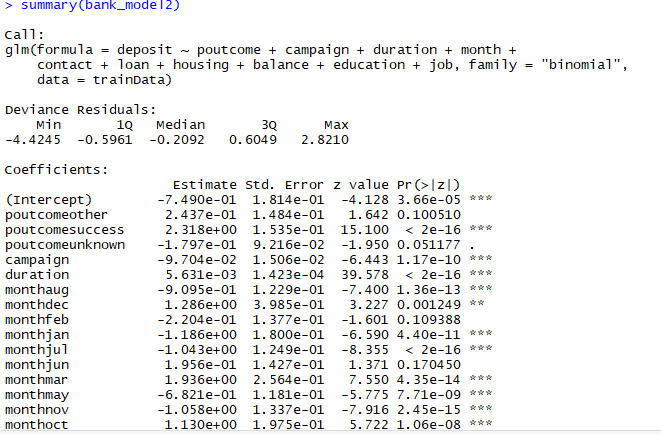
> bank\_model2 <- glm(deposit ~ poutcome+campaign+duration+month+contact+loan+housing+balance+education+job, data = trainData, family = "binomial")

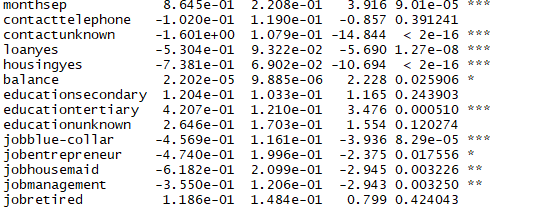
Checking the summary of bank\_model2

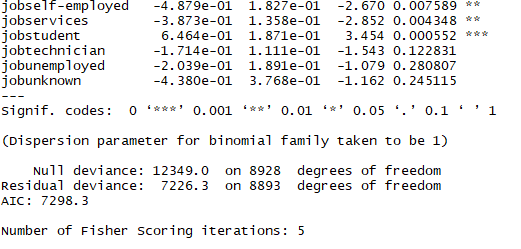
**R code:**

> summary(bank\_model2)

Output:







As seen in the output above, all covariates now seem to have significant values. This model will be used for further analysis.

#### Calculating prediction for all points in-sample

#Calculating prediction for all points in trainData range (in sample)

> pred\_prob = predict(bank\_model2, trainData, type = "response")

> pred\_value = 1\*(pred\_prob>0.5)

> cbind(trainData, pred\_prob, pred\_value)

The predict() function can be used to find values for all the data points for which we do not have actual observations using our model. Since the probabilities are given in the form of a percentage and we require a binary response, we assume that any probability greater than 50% i.e greater than 0.5 is a positive value – ‘yes’ in this case and any probability less than 0.5 is a negative value or a ‘no’.

The pred\_prob and pred\_value are then bound to the trainData dataset using cbind().

#### Model Adequacy Checking

##### Creating the Confusion Matrix for trainData

A confusion matrix is a table with 4 different combination of predicted and actual values. We can determine how many of our predictions fall under the following categories

1. True-positives: Values that were correctly predicted to be true.

2. False-positives: Values that were false but predicted to be true.

3. True-negatives: Values that were correctly predicted to be negative.

4. False-negatives: Values were true but predicted to be negative.

A confusion matrix was created to check how adequate our logistic regression model was within the trainData.

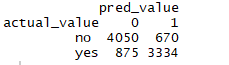
**R Code:**

> actual\_value = trainData$deposit

> confusion\_matrix = table(actual\_value, pred\_value)

> confusion\_matrix

Output:



##### Calculating the Misclassification Error Rate

**R Code:**

> misclassification\_error\_rate=1-sum(diag(confusion\_matrix))/sum(confusion\_matrix)

> misclassification\_error\_rate

[1] 0.1730317

We can see above that the misclassification error rate for our confusion matrix is 0.17 which translates to a 17% error rate. This is an acceptable error rate and means that the model works for the given dataset.

##### Calculating the Precision and Recall

**R Code:**

install.packages("PRROC")

> library(PRROC)

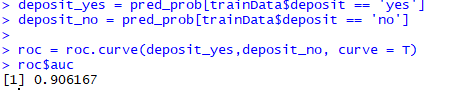
> deposit\_yes = pred\_prob[trainData$deposit == 'yes']

> deposit\_no = pred\_prob[trainData$deposit == 'no']

> roc = roc.curve(deposit\_yes,deposit\_no, curve = T)

> roc$auc

**Output:**



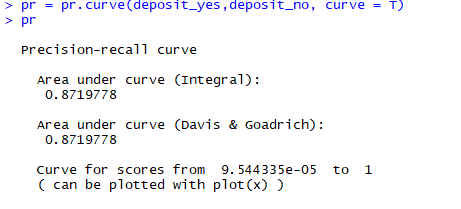
##### Calculating the Precision Curve for train data

**R Code:**

> pr = pr.curve(deposit\_yes,deposit\_no, curve = T)

> pr

**Output:**

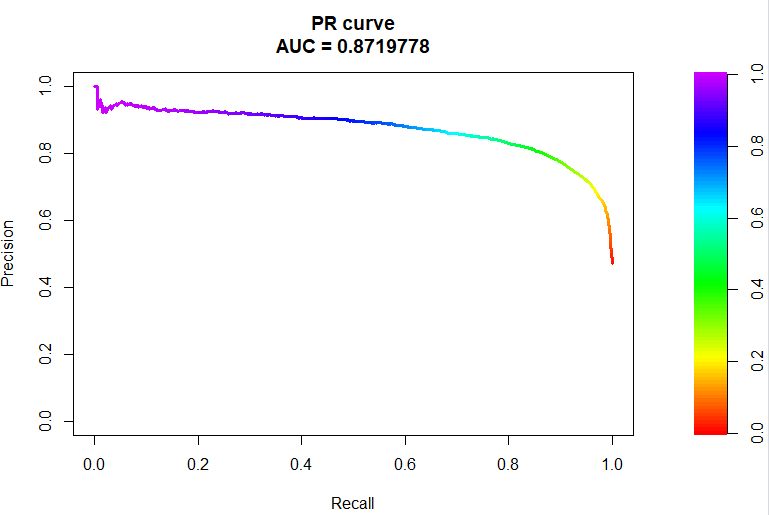


##### Plotting the Precision Curve for train data

**R Code:**

plot(pr)

**Output:**



##### Validating Model with Test Data

The model created using the train data can now be validated using the test data.

**R Code:**

> pred\_prob\_test = predict(bank\_model2, testData, type = "response")

> pred\_value\_test = 1\*(pred\_prob\_test>0.5)

> cbind(testData, pred\_prob\_test, pred\_value\_test)

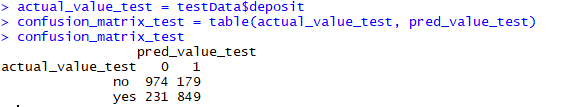
##### Creating the confusion matrix for the testData dataset

> actual\_value\_test = testData$deposit

> confusion\_matrix\_test = table(actual\_value\_test, pred\_value\_test)

> confusion\_matrix\_test

**Output:**



##### Calculating the Misclassification Error Rate for test data

**R Code:**

> misclassification\_error\_rate\_test=1-sum(diag(confusion\_matrix\_test))/sum(confusion\_matrix\_test)

> misclassification\_error\_rate\_test

**Output:**



##### Calculating the precision and recall for test data

**R Code:**

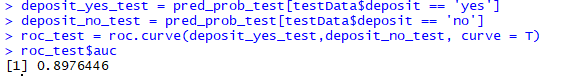
> deposit\_yes\_test = pred\_prob\_test[testData$deposit == 'yes']

> deposit\_no\_test = pred\_prob\_test[testData$deposit == 'no']

> roc\_test = roc.curve(deposit\_yes\_test,deposit\_no\_test, curve = T)

> roc\_test$auc

**Output:**



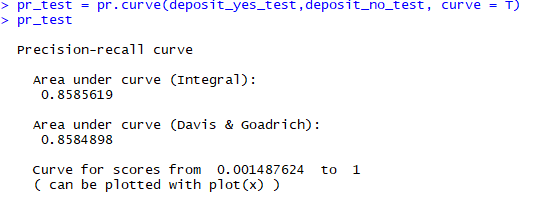
##### Calculating the Precision Curve for test data

**R Code:**

> pr\_test = pr.curve(deposit\_yes\_test,deposit\_no\_test, curve = T)

> pr\_test

**Output:**

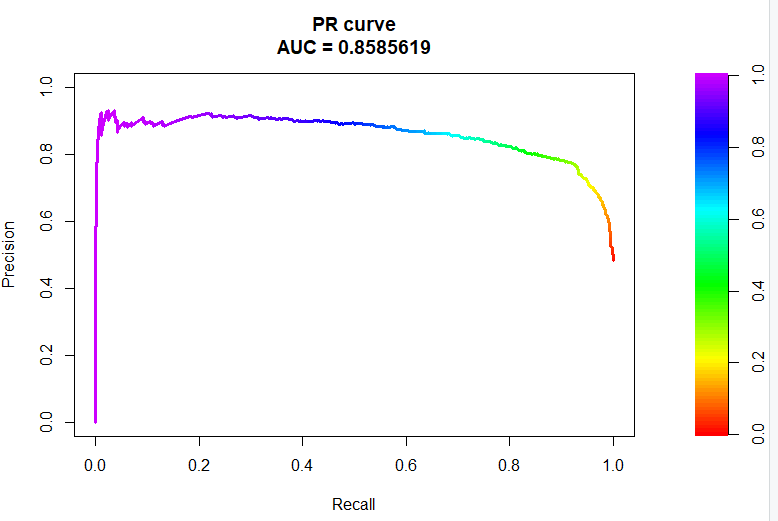


##### Plotting the precision curve

**R Code:**

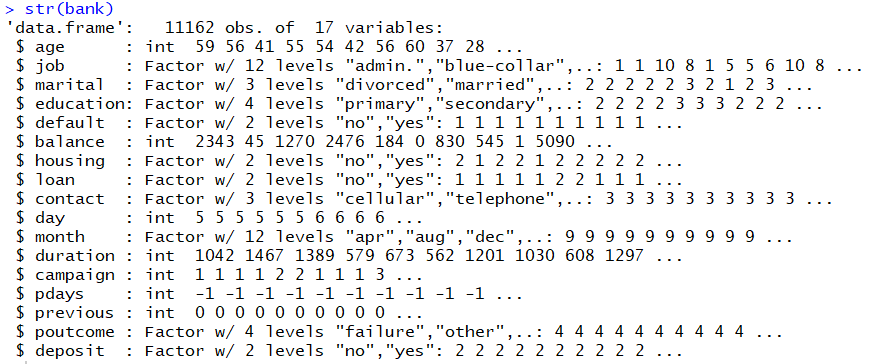
> plot(pr\_test)

**Output:**

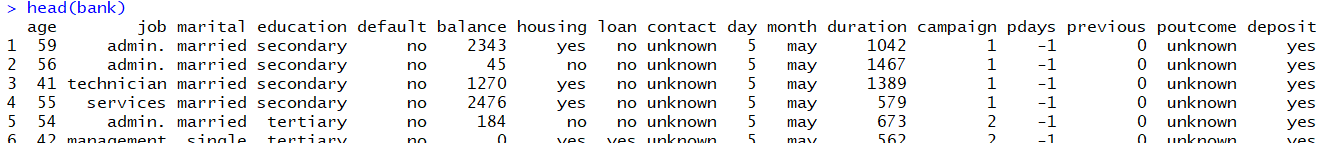


### **3.2 Random Forest Model**

Bank Marketing data was imported using read.csv() function in R. The class for each variable was studied using str() function as below:



A sample data can be seen below:

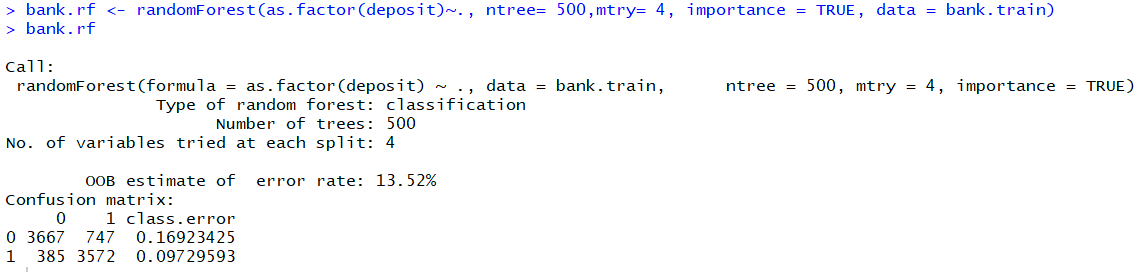


As we can see that our response variable has two levels ‘yes’ and ‘no’. The levels were transformed to 1 and 0 respectively.

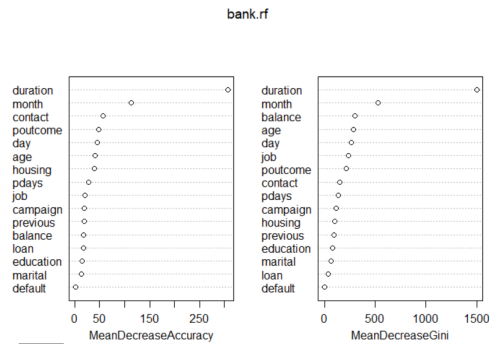
#### Model Building:

First the data set was divided into training and testing data with 75%-25% split respectively. A seed value was set using set.seed() function to make sure that the randomly split data could be regenerated. A random forest model was built using training data using randomforest package. We use 4 predictors for each split and grow 500 trees fully without pruning. A subset of predictors is randomly chosen without replacement at each split which helps in reducing the variance of the model overall. This is a prime advantage of random forest as compared to traditional decision trees.

In the below summary we can see that this model has an Out-Of-Bag error rate of 13.52%. The model also outputs a confusion matrix. We can see that random forest is doing a fairly good job in predicting the response variable i.e. deposit(Yes/No) field.

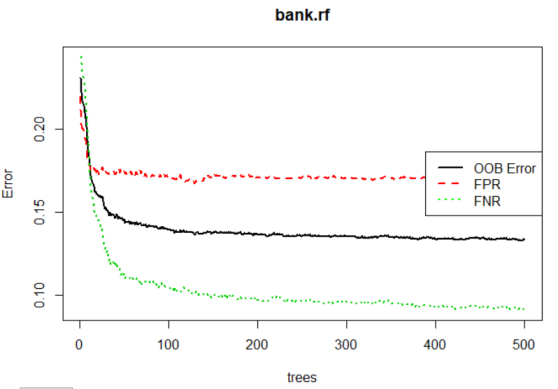


In the above confusion matrix, we can see that the model is doing a good job predicting the ‘no’ case of bank deposit as compared to the ‘yes’ case.



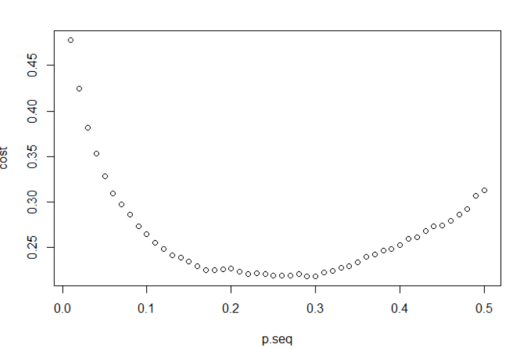
By setting the importance argument on, we obtained the variable importance plot as above using varImpPlot() function and we can see that duration is highly significant in our data set.

We plot a graph for the error rate (False Positive Rate, False Negative Rate and Out-Of-Bag Error) with the increasing number of trees.



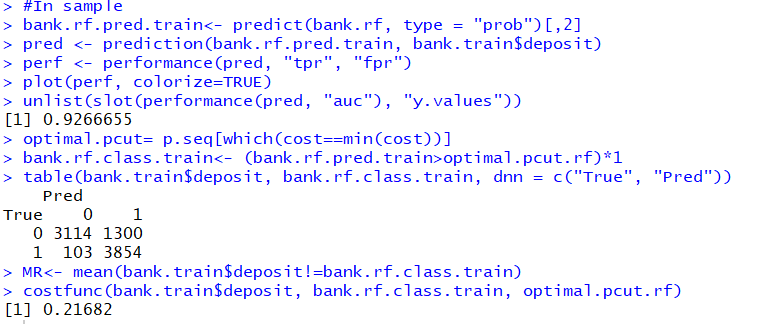
In the above plot, we can see the change of error with increasing number of trees. The False Positive Rate is higher compared to other error rate and False Negative Rate is lowest. The error rate starts dropping for at ntree~ 30. This says that our model is predicting ‘No’ cases more accurately than ‘Yes’ cases which can also be confirmed by the confusion matrix above.

Our goal here is to predict the customers who have signed up for the bank deposit, so we want to develop our marketing strategies by studying the behavior of the customers who have historically taken the subscription which can be used to customize the marketing strategy. The cost of predicting ‘yes’ is comparatively higher than the cost of predicting ‘no’. So, we define an asymmetric cost function with the weight ratio 5:1 to find the optimal cut off probability. We plot the graph of the cost function as below and we can see that the minimum cost of prediction is at 0.29. Hence, we use this cut off probability to do the in-sample and out-of-sample prediction.



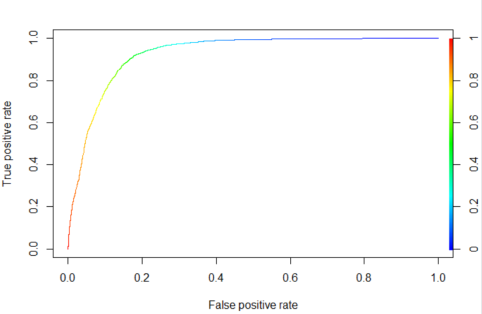
#### In sample prediction:

In sample prediction was done using the training data and a confusion matrix was obtained using the above cut off probability with a misclassification rate of 16.76%. Area Under the curve was found to be 92.66% and asymmetric cost was found to be 21.68% as shown in the below code.



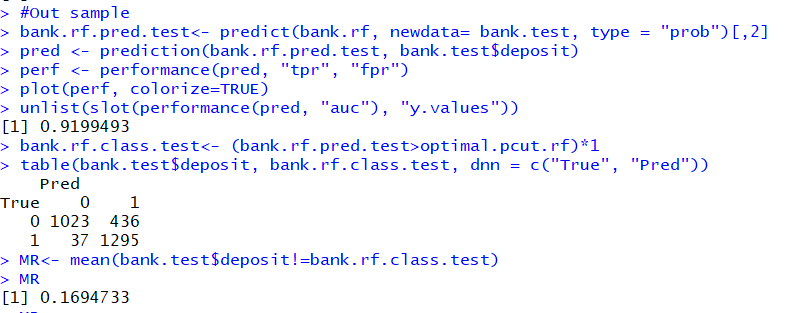


ROC curve was plotted for the training data as below:



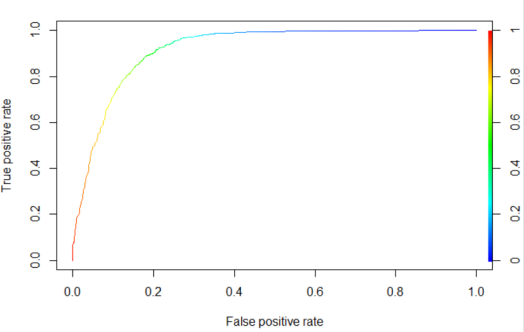
#### Out-of-sample prediction:

Out-of-sample prediction was performed using test data and a confusion matrix was obtained using the above cut off probability with a misclassification rate of 16.94%. Area Under the curve was found to be 92.66% and asymmetric cost was found to be 22.25% as shown in the below code.





ROC curve was plotted with the test sample as below and AUC was found to be 91.99.



Using the above cut off probability, a confusion matrix was obtained as shown above, and we can see we have low the false positives which says that we are doing a fairly good job in predicting the true values.

Below are the results from random forest model:

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Misclassification Rate | AUC | Asymmetric Cost |
| In-sample | 16.76 | 92.66 | 21.68 |
| Out-of-sample | 16.94 | 91.99 | 22.25 |

From the above table, we can see that we could predict the response variable with 83.06% accuracy. ‘Duration’ was the most significant variable in predicting the response variable.

### **XGBOOST**

**XGBoost** provides a parallel tree boosting that solve many data science problems in a fast and accurate way. Boosting is a sequential technique which works on the principle of ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t, the model outcomes are weighed based on the outcomes of previous instant t-1. The outcomes predicted correctly are given a lower weight and the ones miss-classified are weighted higher.

**Performing XGBoost for predicting whether the customer will subscribe for a term deposit or not:**

#### Importing the necessary packages

To start, we first import the necessary packages in our Python workspace:

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **scipy.optimize** **as** **opt**

**from** **sklearn** **import** preprocessing

**from** **sklearn.model\_selection** **import** train\_test\_split, GridSearchCV, StratifiedKFold

**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, precision\_recall\_curve, roc\_auc\_score, roc\_curve

**import** **xgboost**

**from** **matplotlib** **import** pyplot

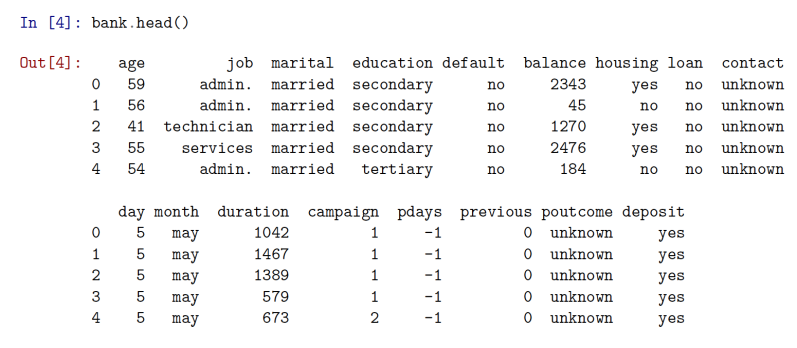
**from** **sklearn.preprocessing** **import** LabelEncoder

#### Reading the dataset as a Pandas Data Frame

Since we’ll be reading in our dataset which is in a csv format, we import the dataset csv in python as a pandas data frame:

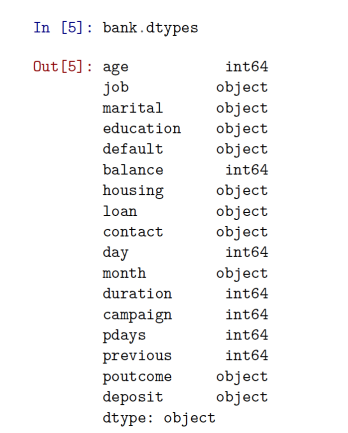
bank = pd.read\_csv("/Users/roshanaraman/OneDrive - University of Cincinnati/FLEX 4/Data Mining for BI/bank.csv")

To ensure the data is imported correctly, we use the **head()**  function to take a glance at the dataset:

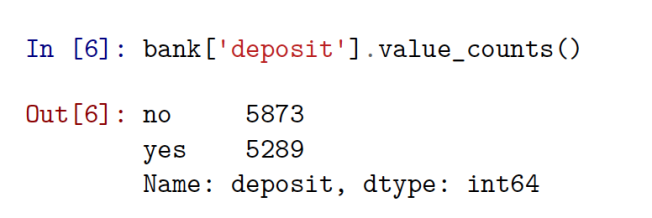


#### Analyzing data types for the variables

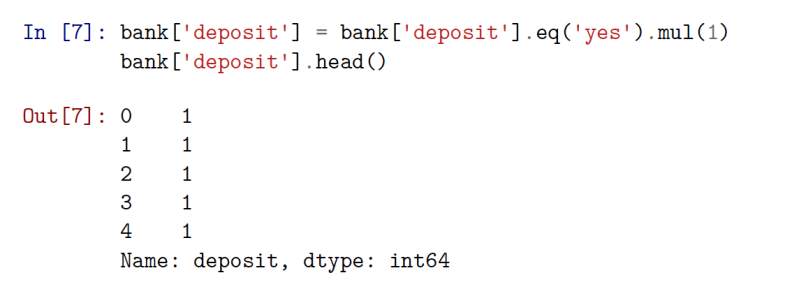
Now, the datatypes of the features imported are checked to analyze the manipulation required to be done to bring it in a format that is compatible for performing model building:



Now, we take a look at what values does our response variable (deposit) have currently:

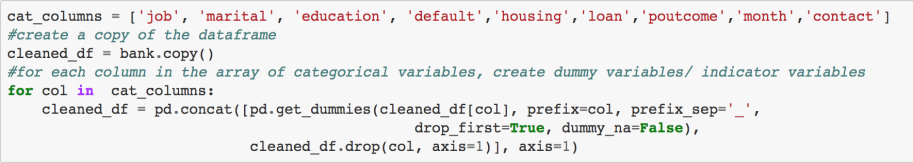


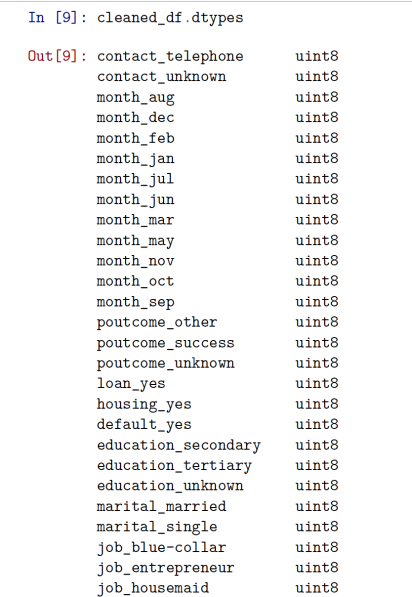
We convert the target variable to a binary variable to ease the computation process substituting “Yes” and “No” with 1 and 0 respectively:



#### XGBoost manipulations

Now since XGBoost doesn’t support or take categorical variables as input, it is essential that we first convert the categorical variables present in our data to dummy or indicator variables. To achieve this, we wrote a for each loop and utilized the **pandas** library to create the dummy variables and then verifies the same by printing out their data types:

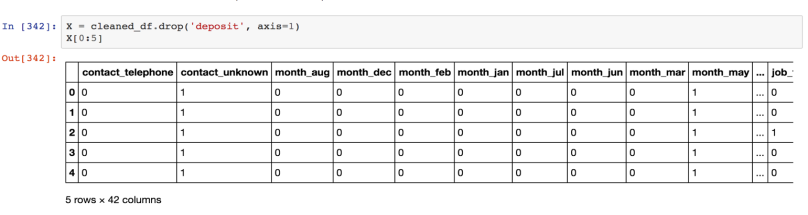


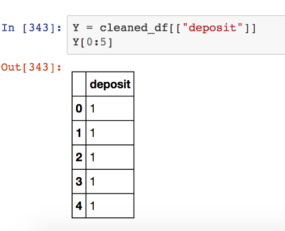


We can see that all the categorical variables now have been converted into dummy variables with a data type of uint8.

#### Creating predictor and feature variables

Now in the next step, we first create a data frame that contains all our feature variables or covariates and name it as X followed by creating the target or predictor variable named Y:



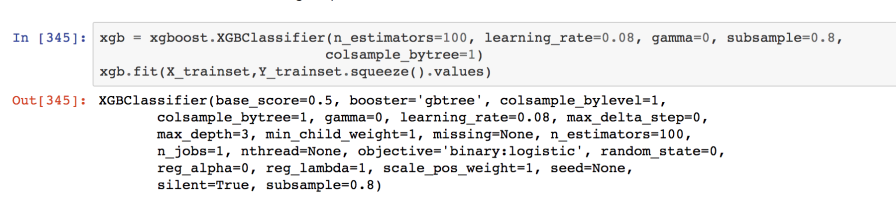


#### Train and Test Split

Now, finally when we have successfully converted the data into a form that can be used for prediction, we perform the train and test split of the data set using the predefined function in **Sci-kit learn's** preprocessing library called **train\_test\_split:**



#### Creating the XGBoost model using the predefined function XGBClassifier

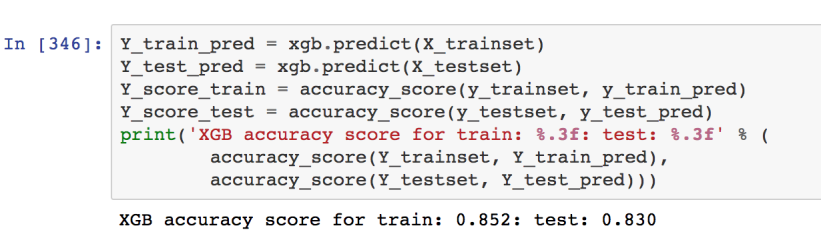


**Parameter explanations:**

1. **n\_estimators**: number of boosted trees to fit
2. **gamma**: A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.
3. **subsample**: Denotes the fraction of observations to be randomly samples for each tree. Lower values make the algorithm more conservative and prevents overfitting, but too small values might lead to under-fitting. Typical values are between 0.5 to 1.
4. **max\_depth**: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.

#### Running the model with the train and test data

Now, we use the above built model on train and test data to calculate the accuracy score and evaluate the model:



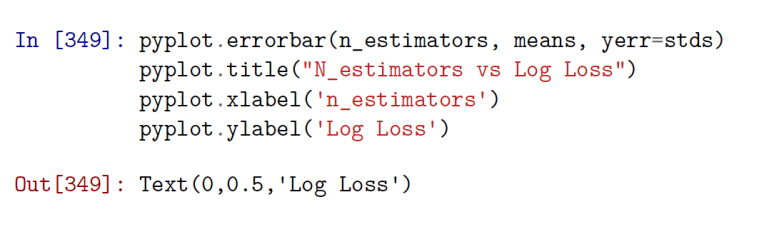
Currently the model gives us an of **85%** accuracy rate for the train set and an **83%** accuracy for the test set. Now we will try to improve the model by performing some assessments for parameter tuning.

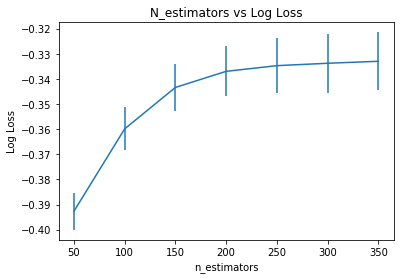
#### Tuning the number of decision trees in XGBoost

Using **scikit-learn** we can perform a grid search of the **n\_estimators** model parameter, evaluating a series of values from 50 to 350 with a step size of 50 (50, 150, 200, 250, 300, 350). We also make use of **LabelEncoder**. LabelEncoder can be used to normalize labels. It can also be used to transform non-numerical labels (as long as they are hash-able and comparable) to numerical labels. It encodes labels with value between 0 and n\_classes-1.

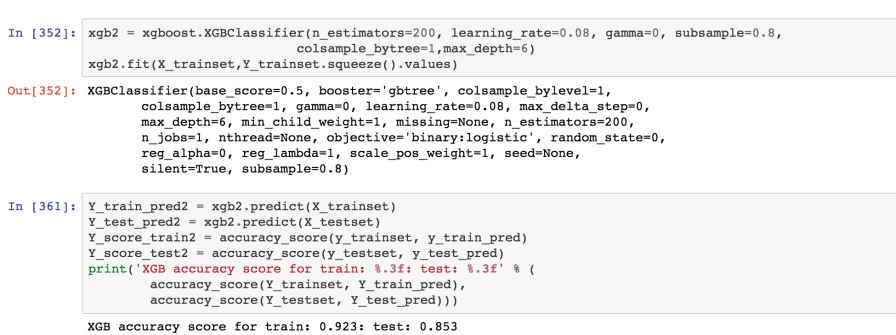


The Best n\_estimators is printed as -0.332967 using {'n\_estimators': 350}. We plot the number of estimators Vs Log loss as just calculated using the grid to further confirm the optimal number of estimators.





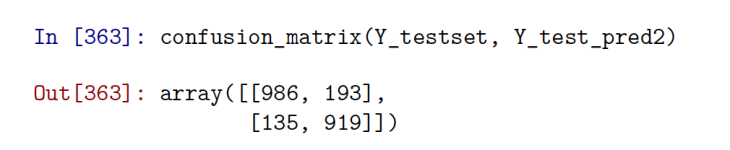
Here we can clearly observe that there is not much difference between the log loss of estimators ranging from 200 to 350. Hence, we can use the **optimal n\_estimator = 200** and then rerun the model to see the results.



We can observe that the accuracy of the model increases to a **92.3 %** for the train set and a **85.3 %** for the test set.

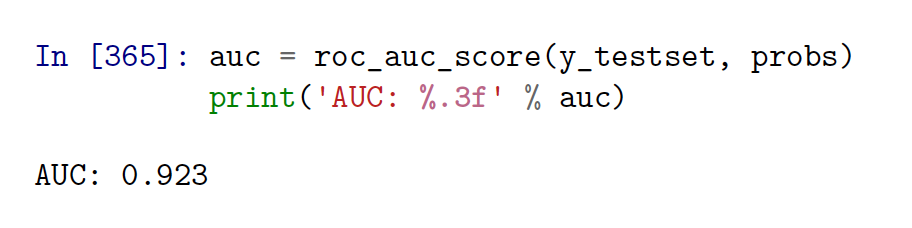
#### Confusion Matrix for XGBoost

Finally, we create the confusion matrix to assess the Accuracy rates, precision, recall and so on**:**

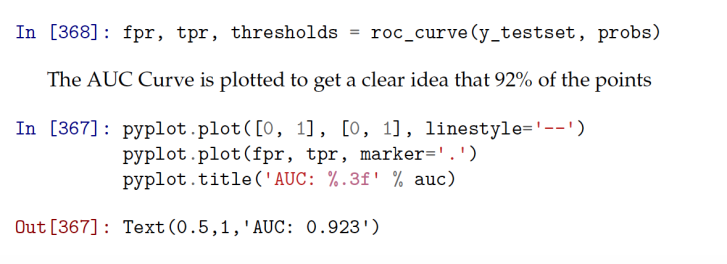


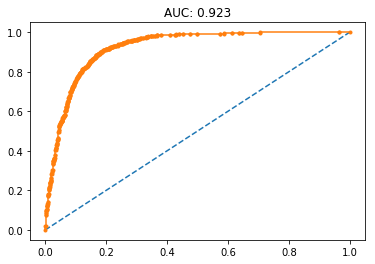
#### Receiver Operating Characteristic Curve

The ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. AUC stands for Area Under the Curve. Example: When AUC is 0.7, it means there is 70% chance that model will be able to distinguish between positive class and negative class.

****

Here we have an **AUC** of 0.92 meaning that there is a **92%** chance that the model will perform classification accurately.   
**fpr**: false positive rates   
**tpr**: true positive rates





The “steepness” of ROC curves is also important, since it is ideal to maximize the true positive rate while minimizing the false positive rate. Since we have a steep curve, we can say that our model is quite accurate in out of sample evaluation.

## Conclusion

Based on the analysis we did above, we have displayed a table which gives the AUC and Misclassification rate for all the three models.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Logistic Regression** | **Random Forest** | **XGBOOST** |
| AUC | 85.85 | 92.7 | 92.3 |
| Misclassification error rate | 18.4% | 16.9% | 14.7% |
| Accuracy | 82.6% | 83.1% | 85.3% |