BANK LOAN DEFAULT CASE

Yashwanth S G

07/12/2019

TABLE OF CONTENTS

INTRODUCTION:

Problem Statement:

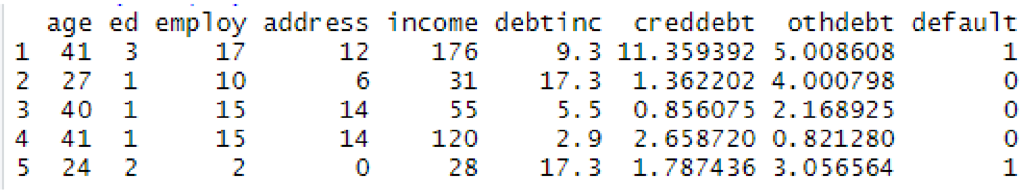
The loan default dataset has 8 variables and 850 records, each record being loan default status for each customer. Each Applicant was rated as "Defaulted" or "Not-Defaulted". New applicants for loan application can also be evaluated on these 8 predictor variables and classified as a default or non-default based on predictor variables. So, it's easy for the loan provider to know whom to provide loan and whom not to.

Data and Proposed Approach:

As our prediction variable is "FACTOR or CLASS". Our task is to build the classification model to predict whether the customer is "Defaulted" or "Not-Defaulted" based on the given dependent variables.

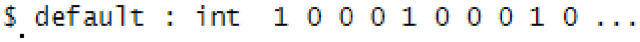
Initially, the assumption is that all the other independent variables are critical in determining the default variable. For classification, we will apply a variety of machine learning classifiers. 'Logistic Regression', 'Support Vector Machines', 'Random forests', and 'Gradient boosting' will be used. The latter two are 'ensemble' methods, and with hyper parameter tuning, are expected to perform better. We will then evaluate our models by calculation various performance metrics and choose the best classifier.

Table 1.1: Bank Loan Default Case Sample data:



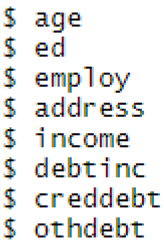
	age	ed	employ	address	income	debtinc	creddebt	othdebt	default
1	41	3	17	12	176	9.3	11.359392	5.008608	1
2	27	1	10	6	31	17.3	1.362202	4.000798	0
3	40	1	15	14	55	5.5	0.856075	2.168925	0
4	41	1	15	14	120	2.9	2.658720	0.821280	0
5	24	2	2	0	28	17.3	1.787436	3.056564	1

- The dataset contains 850 observations and 9 variables of which first 8 variables are independent variable and 9th variable is dependent variable (predicting class - default).
- Below is the variable (dependent), which is to be predicted.



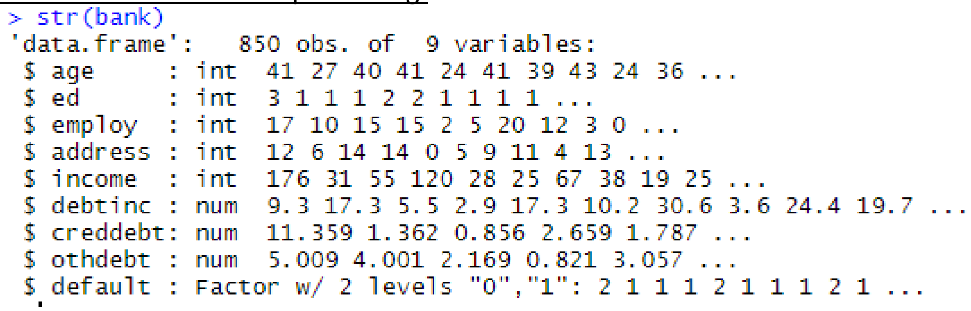
```
$ default : int  1 0 0 0 1 0 0 0 1 0 ...
```

- Below are the variables (independent) we used to predict the class of default.



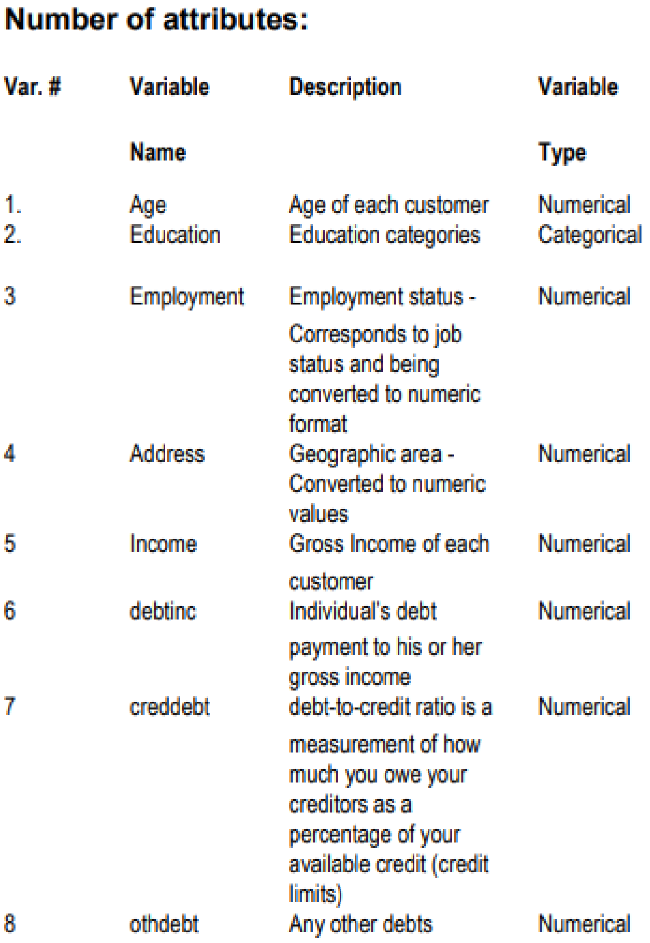
```
$ age
$ ed
$ employ
$ address
$ income
$ debtinc
$ creddebt
$ othdebt
```

The structure of the dataset after processing:



```
> str(bank)
'data.frame':   850 obs. of  9 variables:
 $ age     : int  41 27 40 41 24 41 39 43 24 36 ...
 $ ed      : int  3 1 1 1 2 2 1 1 1 1 ...
 $ employ  : int  17 10 15 15 2 5 20 12 3 0 ...
 $ address : int  12 6 14 14 0 5 9 11 4 13 ...
 $ income  : int  176 31 55 120 28 25 67 38 19 25 ...
 $ debtinc : num  9.3 17.3 5.5 2.9 17.3 10.2 30.6 3.6 24.4 19.7 ...
 $ creddebt: num  11.359 1.362 0.856 2.659 1.787 ...
 $ othdebt : num  5.009 4.001 2.169 0.821 3.057 ...
 $ default : Factor w/ 2 levels "0","1": 2 1 1 1 2 1 1 1 2 1 ...
```

Bank Loan Default Case variables descriptions.



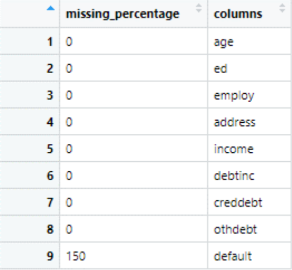
Number of attributes:

Var. #	Variable	Description	Variable
	Name		Type
1.	Age	Age of each customer	Numerical
2.	Education	Education categories	Categorical
3	Employment	Employment status - Corresponds to job status and being converted to numeric format	Numerical
4	Address	Geographic area - Converted to numeric values	Numerical
5	Income	Gross Income of each customer	Numerical
6	debtinc	Individual's debt payment to his or her gross income	Numerical
7	creddebt	debt-to-credit ratio is a measurement of how much you owe your creditors as a percentage of your available credit (credit limits)	Numerical
8	othdebt	Any other debts	Numerical

Missing Value Analysis:

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models.

They are mean, median, KNN impute methods to deal with missing values.
Below table illustrate no missing value present in the data.
In R function(x){sum(is.na(x))} is the function used to check the sum of missing values.
In python bike_train.isnull().sum() is used to detect any missing value

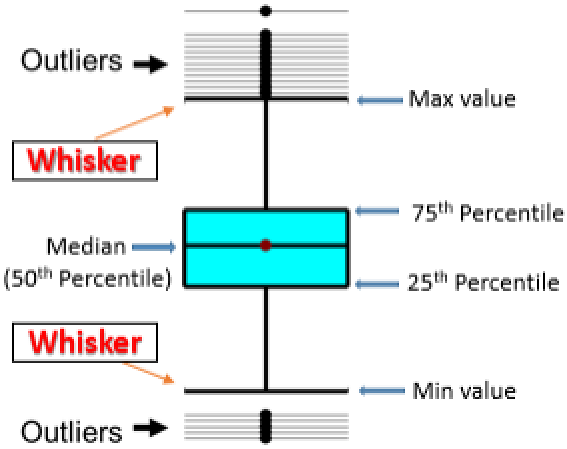


	missing_percentage	columns
1	0	age
2	0	ed
3	0	employ
4	0	address
5	0	income
6	0	debtinc
7	0	creddebt
8	0	othdebt
9	150	default

Note: The the dataset has 150 missing labeled values in the dependent variable column. Those 150 observations can be separated from the dataset and can be named as test data. After choosing the best model, these observations can be evaluated and classified as “default” or “not default”.

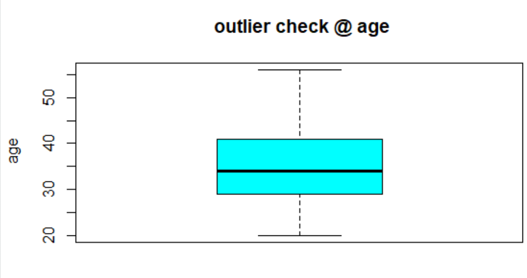
Outlier Analysis:

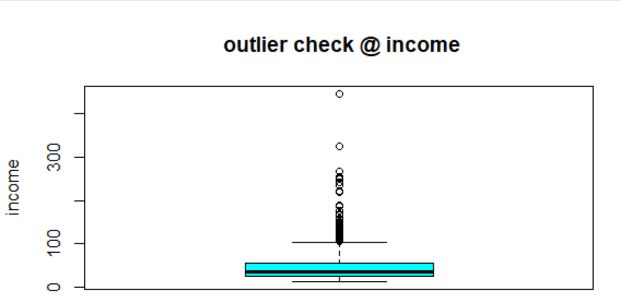
Outliers - An outlier is an observation point that is distant from other observations. These data points could be a result of errors or wrongly recorded observations. These are often excluded from analysis as it could have an adverse effect on the reliability of our model. We will identify outliers with the help of box plots. A typical box plot with outliers is illustrated below:

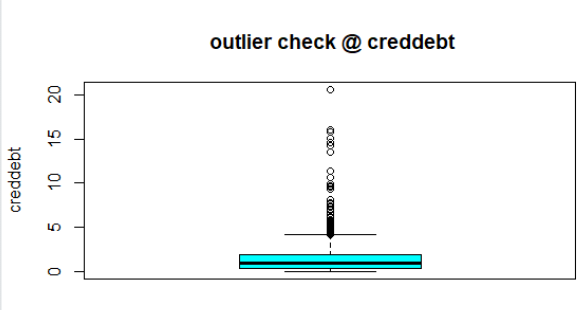


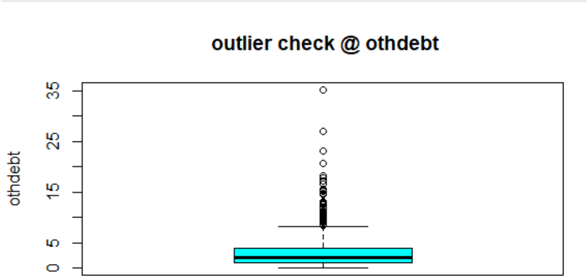
The data points above and below the whiskers are our outliers. The min and max values in this plot have been plotted by excluding the outliers from the calculation.

Below are the figures showing boxplots for outlier check. The age variable has no outliers present in it. All other variables has outliers in it. But, we can make out that the income and debt values can't be outliers. Those values will be either machine generated or based on the work experience. Also, we can understand that income and spending will be based on the individual age and education.









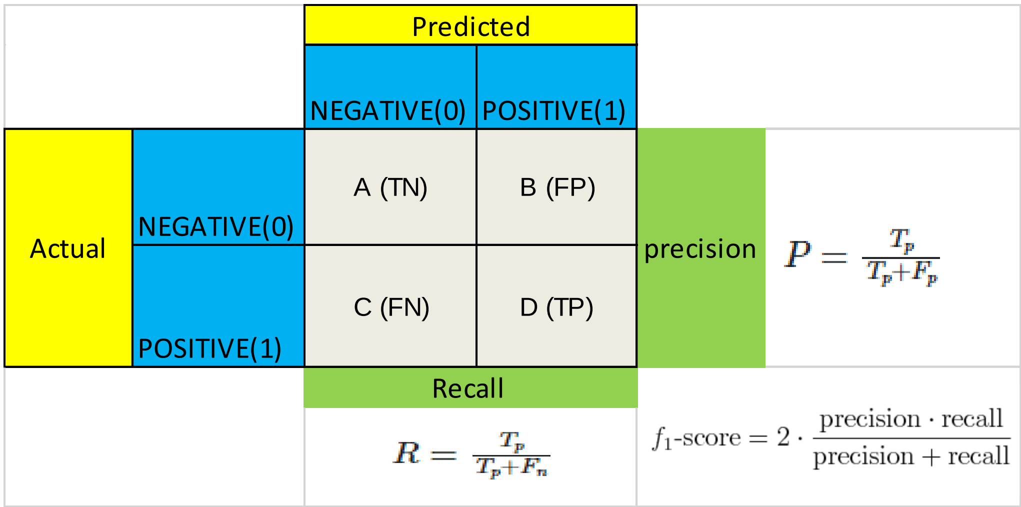
Proposed course of action: All rows will be considered for analysis.

Machine Learning for Classification:

Machine learning is a branch of Artificial Intelligence that provides systems with the capabilities to automate learning and improvise based on information in the training set without having to be explicitly programmed. One of the greatest uses of machine learning today is to automate decision making. Machine Learning can be used in a wide variety of industries like the financial sector, ecommerce industry, social networking and health industry.

Performance metrics explained:
Confusion Matrix:

A confusion matrix contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using the data in the matrix. The following table shows the confusion matrix for a two class classifier.
The entries in the confusion matrix have the following meaning in the context of our study:
● A is the number of correct predictions that an instance is negative,
● B is the number of incorrect predictions that an instance is positive,
● C is the number of incorrect of predictions that an instance negative, and
● D is the number of correct predictions that an instance is positive.



		Predicted			
		NEGATIVE(0)	POSITIVE(1)		
Actual	NEGATIVE(0)	A (TN)	B (FP)	precision	$P = \frac{T_p}{T_p+F_p}$
	POSITIVE(1)	C (FN)	D (TP)		
		Recall		$f_1\text{-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$	
		$R = \frac{T_p}{T_p+F_n}$			

Figure showing Confusion Matrix.

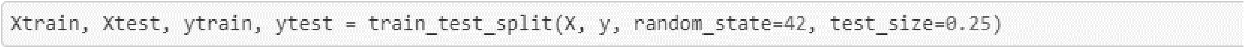
Nomenclature of terms in Confusion Matrix:

- True Negative: Predicted negative and it’s true.
- False Positive (Type 1 Error): Predicted positive and it’s false.
- False Negative (Type 2 Error): Predicted negative and it’s false.
- True Positive: Predicted positive and it’s true.
- Recall: Out of all the positive classes, how much we predicted correctly. It should be high as possible. Recall is also known as “sensitivity” and “true positive rate” (TPR).
- Precision: Out of all the positive classes we have predicted correctly, how many are actually positive.
- Accuracy: Out of all the classes, how much we predicted correctly. It should be high as possible.

- F1-score: this is just the harmonic mean of precision and recall.
- High recall, low precision: This means that most of the positive examples are correctly recognized (low FN) but there are a lot of false positives.
- Low recall, high precision: This shows that we miss a lot of positive examples (high FN) but those We predict as positive are indeed positive (low FP).

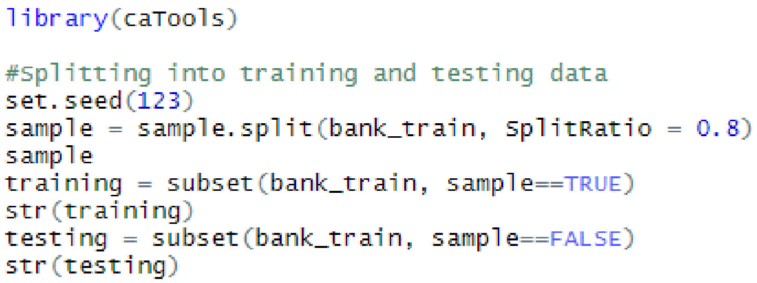
Various classification techniques were tried considering factors like accuracy, area under the ROC curve and area under the Precision vs. Recall curve.

In Python:
Splitting the data the dataset will be split into the training and test sets. The test set size has been chosen to be 25% of the original dataset. That is, the classification algorithm will train on 75% of the data and then, the model will be used to make predictions on the test set. We are using Sklearn package. Using train_test_split method dataset is divided into training and testing observations.



```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=42, test_size=0.25)
```

In R:
Splitting the data the dataset will be split into the training and test sets. The test set size has been chosen to be 20% of the original dataset. That is, the classification algorithm will train on 80% of the data and then, the model will be used to make predictions on the test set. CaTools is used to split the data into training and testing observations.

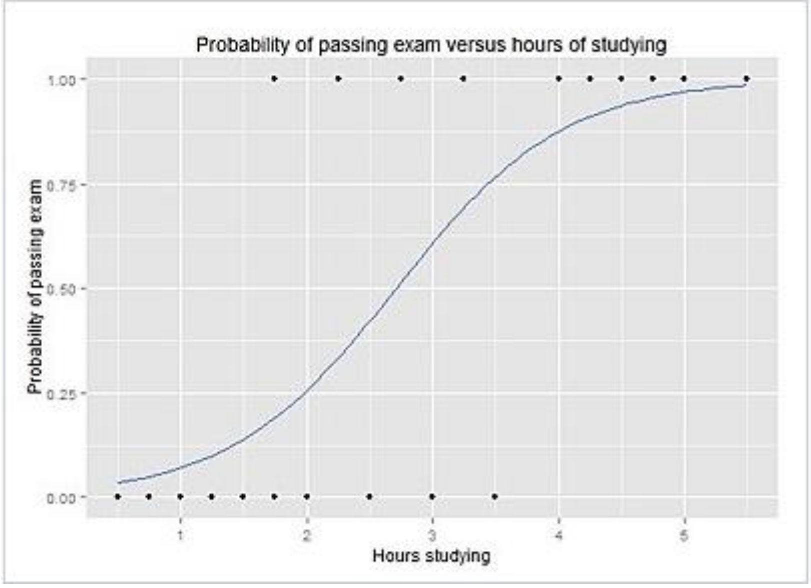


```
library(caTools)

#Splitting into training and testing data
set.seed(123)
sample = sample.split(bank_train, SplitRatio = 0.8)
sample
training = subset(bank_train, sample==TRUE)
str(training)
testing = subset(bank_train, sample==FALSE)
str(testing)
```

Logistic Regression:

This a traditional and basic approach to classification in supervised learning. It derives its name from the logit function.

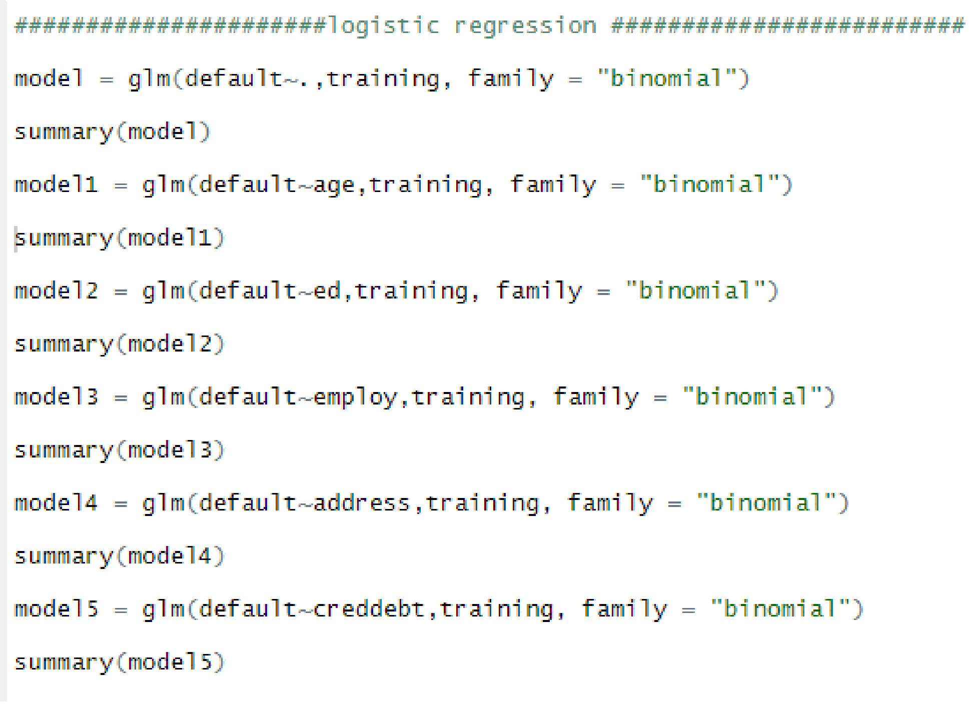
Figure showing sigmoid curve.



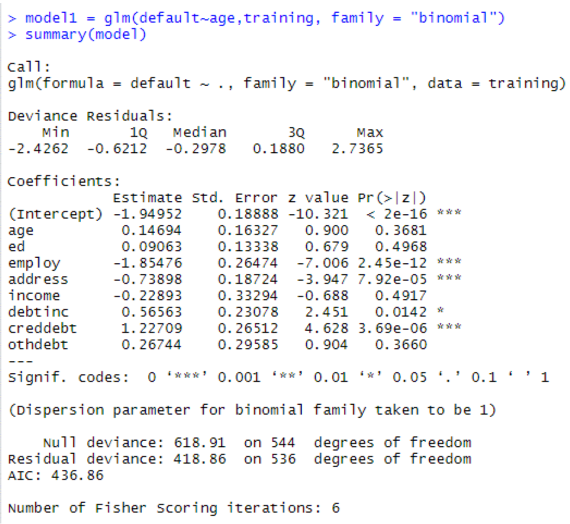
The logistic regression model is performed with some assumptions: that data has no outliers, there are two classes to be predicted, and that no two independent variables are highly correlated to each other.

Building Logistic regression in R:

Using glm (Generalized linear models) to build regression model.

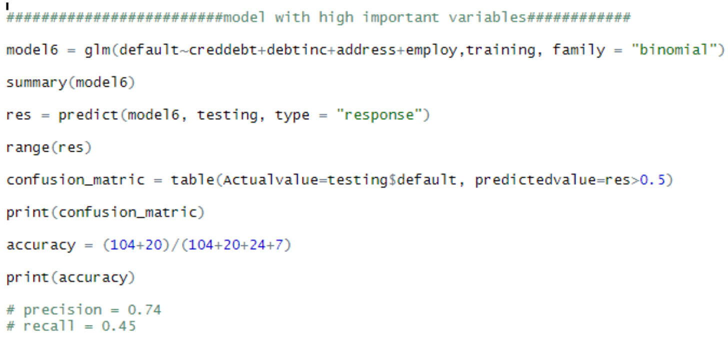


```
######################logistic regression #########################

model = glm(default~.,training, family = "binomial")

summary(model)

model1 = glm(default~age,training, family = "binomial")

summary(model1)

model2 = glm(default~ed,training, family = "binomial")

summary(model2)

model3 = glm(default~employ,training, family = "binomial")

summary(model3)

model4 = glm(default~address,training, family = "binomial")

summary(model4)

model5 = glm(default~creddebt,training, family = "binomial")

summary(model5)
```

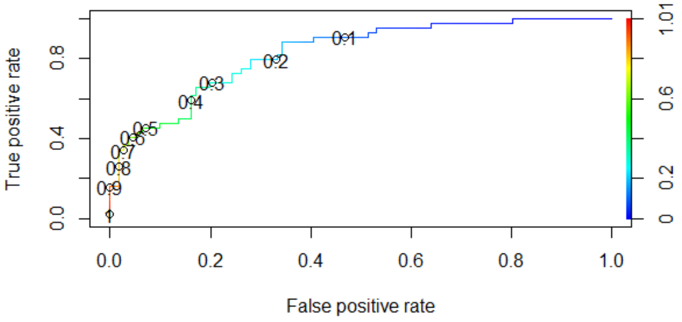


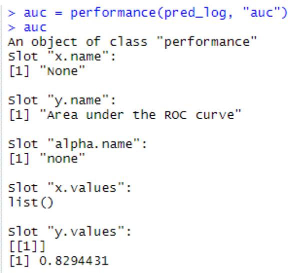
```
> model1 = glm(default~age,training, family = "binomial")
> summary(model)

Call:
glm(formula = default ~ ., family = "binomial", data = training)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4262  -0.6212  -0.2978   0.1880   2.7365

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.94952    0.18888 -10.321  < 2e-16 ***
age          0.14694    0.16327   0.900   0.3681
ed           0.09063    0.13338   0.679   0.4968
employ      -1.85476    0.26474  -7.006 2.45e-12 ***
address     -0.73898    0.18724  -3.947 7.92e-05 ***
income      -0.22893    0.33294  -0.688   0.4917
debtinc      0.56563    0.23078   2.451   0.0142 *
creddebt     1.22709    0.26512   4.628 3.69e-06 ***
othdebt      0.26744    0.29585   0.904   0.3660
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 618.91  on 544  degrees of freedom
Residual deviance: 418.86  on 536  degrees of freedom
AIC: 436.86

Number of Fisher Scoring iterations: 6
```

Null deviance shows how well the response variable is predicted by a model that includes only the intercept (grand mean).
Residual deviance shows how well the response variable is predicted with the inclusion of independent variables.
The Akaike information criterion (AIC) is an estimator of the relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection.

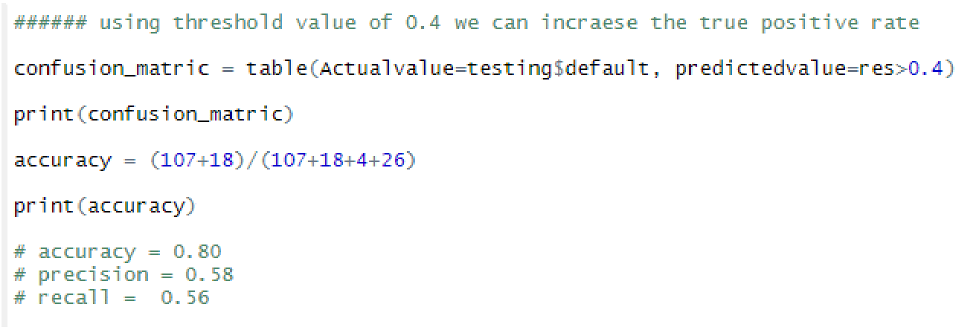


```
########################model with high important variables###########

model6 = glm(default~creddebt+debtinc+address+employ,training, family = "binomial")

summary(model6)

res = predict(model6, testing, type = "response")

range(res)

confusion_matric = table(Actualvalue=testing$default, predictedvalue=res>0.5)

print(confusion_matric)

accuracy = (104+20)/(104+20+24+7)

print(accuracy)

# precision = 0.74
# recall = 0.45
```





```
> auc = performance(pred_log, "auc")
> auc
An object of class "performance"
Slot "x.name":
[1] "None"

Slot "y.name":
[1] "Area under the ROC curve"

Slot "alpha.name":
[1] "none"

Slot "x.values":
list()

Slot "y.values":
[[1]]
[1] 0.8294431
```



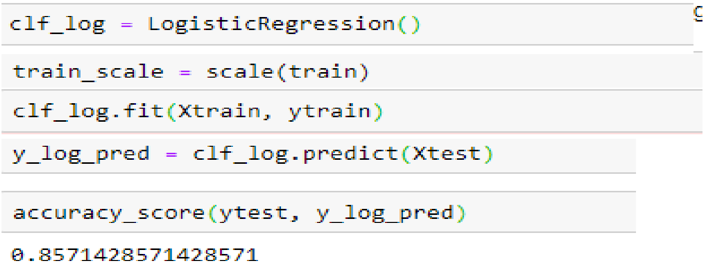
```
###### using threshold value of 0.4 we can incraese the true positive rate

confusion_matric = table(Actualvalue=testing$default, predictedvalue=res>0.4)

print(confusion_matric)

accuracy = (107+18)/(107+18+4+26)

print(accuracy)

# accuracy = 0.80
# precision = 0.58
# recall =  0.56
```

A **receiver operating characteristic curve**, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. X**-axis is False positive rate and Y-axis is True positive rate**. The ideal threshold value should have maximum true positive rate and with minimum false positive rate. The area under the ROC curve is good perfomance evaluator.

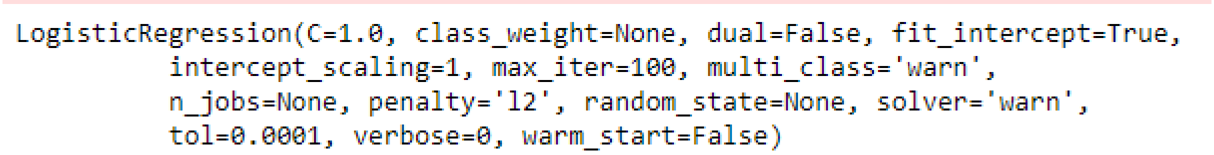
The default threshold value will be set at 0.5 cutoff. To gain good trade-off between the false and true positive rates, choosing the cutoff to be **0.4.** The area under the curve is **0.8294** which is good for the model. The area under the curve ranges from 0 to 1. The accuracy of the model is **80 %** with precision is 0.58 and recall is 0.56.

<u>Building Logistic regression in Python:</u>

The Sklearn logistic linear model is used to build logistic model in python.



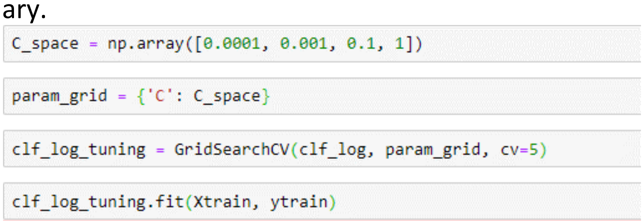
```
clf_log = LogisticRegression()

train_scale = scale(train)

clf_log.fit(Xtrain, ytrain)

y_log_pred = clf_log.predict(Xtest)

accuracy_score(ytest, y_log_pred)

0.8571428571428571
```



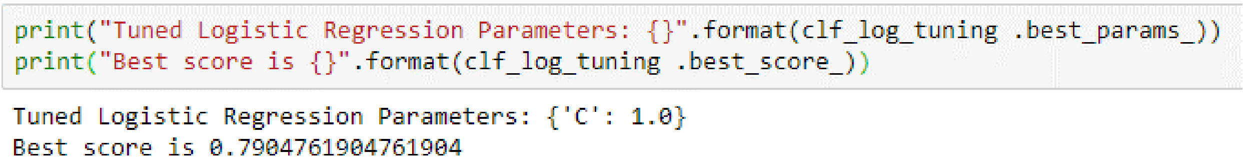
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

Using the default parameters, the test accuracy was around ~85%.

The hyper parameter tuned was ‘C’. This parameter accounts for the “slack”. Lower values of ‘C’ imply more slack and vice versa. Usually, we would prefer to have small values of ‘C’ which results in a smoother decision boundary.

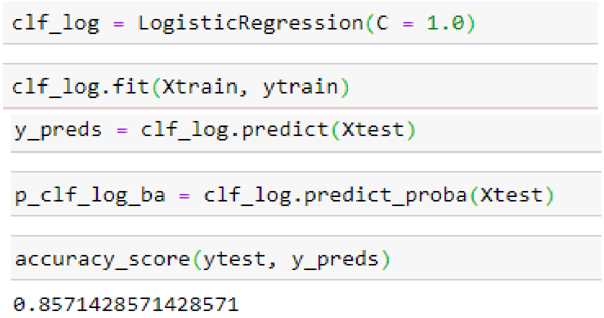


```
C_space = np.array([0.0001, 0.001, 0.1, 1])

param_grid = {'C': C_space}

clf_log_tuning = GridSearchCV(clf_log, param_grid, cv=5)

clf_log_tuning.fit(Xtrain, ytrain)
```



```
print("Tuned Logistic Regression Parameters: {}".format(clf_log_tuning .best_params_))
print("Best score is {}".format(clf_log_tuning .best_score_))
```

```
Tuned Logistic Regression Parameters: {'C': 1.0}
Best score is 0.7904761904761904
```



```
clf_log = LogisticRegression(C = 1.0)
```

```
clf_log.fit(Xtrain, ytrain)
```

```
y_preds = clf_log.predict(Xtest)
```

```
p_clf_log_ba = clf_log.predict_proba(Xtest)
```

```
accuracy_score(ytest, y_preds)
```

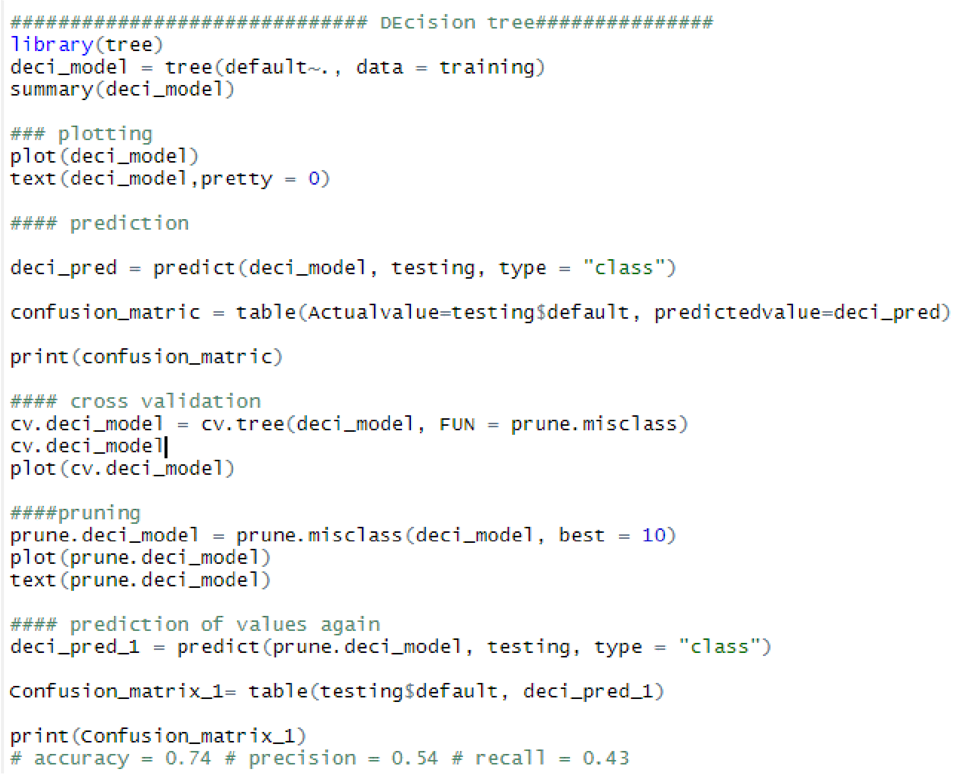
```
0.8571428571428571
```

Using hyper parameter tuning, the best value for c was found to be 1.0. This time, the accuracy on the training set was almost ~85%, which is same as the default because, C value is 1.0 in the default parameters.

Decision Tree:

A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

R:



```
############################## DEcision tree###############
library(tree)
deci_model = tree(default~., data = training)
summary(deci_model)

### plotting
plot(deci_model)
text(deci_model,pretty = 0)

#### prediction

deci_pred = predict(deci_model, testing, type = "class")

confusion_matric = table(Actualvalue=testing$default, predictedvalue=deci_pred)

print(confusion_matric)

#### cross validation
cv.deci_model = cv.tree(deci_model, FUN = prune.misclass)
cv.deci_model
plot(cv.deci_model)

####pruning
prune.deci_model = prune.misclass(deci_model, best = 10)
plot(prune.deci_model)
text(prune.deci_model)

#### prediction of values again
deci_pred_1 = predict(prune.deci_model, testing, type = "class")

Confusion_matrix_1= table(testing$default, deci_pred_1)

print(Confusion_matrix_1)
# accuracy = 0.74 # precision = 0.54 # recall = 0.43
```

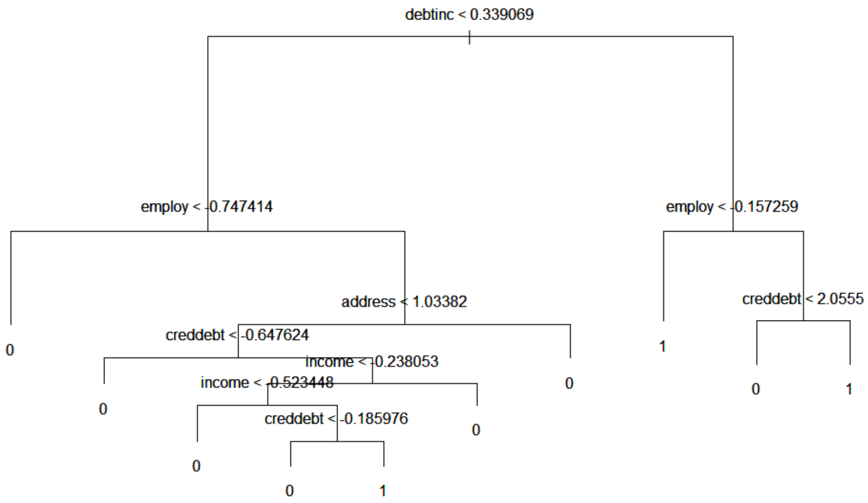
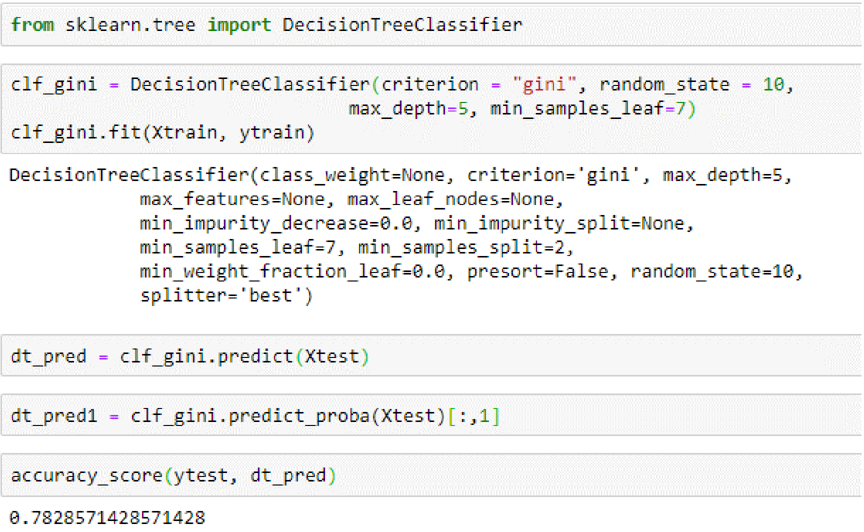


Figure showing decision tree.

Python:



```
from sklearn.tree import DecisionTreeClassifier
```

```
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 10,
                                  max_depth=5, min_samples_leaf=7)
clf_gini.fit(Xtrain, ytrain)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=7, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=10,
            splitter='best')
```

```
dt_pred = clf_gini.predict(Xtest)
```

```
dt_pred1 = clf_gini.predict_proba(Xtest)[:,1]
```

```
accuracy_score(ytest, dt_pred)
```

```
0.7828571428571428
```

Looking at the above figure tree, here decision tree is using only one predictors variables to predict the model , which is not very impressive here the model is over fitted and biased towards only one predictors i.e. “Employ”. The accuracy for the decision tree algorithm is around ~75%.

Random Forest:

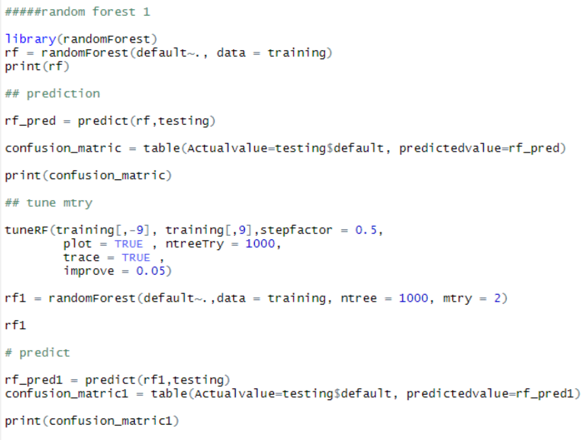
Why Random Forests over decision trees?

Decision tress train over a single training set only. Decision trees take into account each and every variable and every observation in the training set. While decision trees are very fast (in terms of computational speed), they generally overfit the data and perform poorly on test sets.

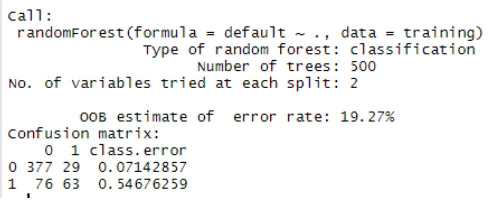
To solve this, we could generate bootstrap samples. That is, we generate samples with replacement from the original sample set. We now have multiple samples instead of one training sample. But these training samples are bound to have duplicate observations since we sampled with replacement. While this model does perform better, it does have access to all the features and most of the observations. To enhance the randomness, each sample is trained with a unique set of features. If we have "n" number of features, the algorithm uses sqrt (n) features for each model. What results is a series of "weak models". These weak models are then "aggregated" to give a complex but strong prediction model. Hence, the randomness comes from the bootstrap samples generated and the features involved for predicting each sample.

The primary assumptions for this algorithm is that the randomly generated bootstrap samples with each sample set using a distinct set of features are independent of each other.

Building Random Forest in R:

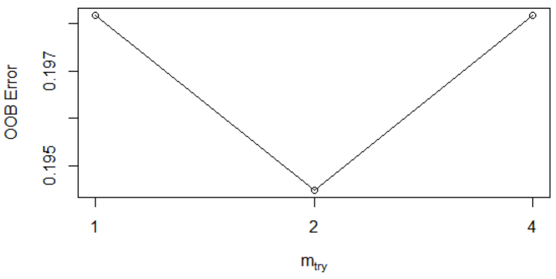


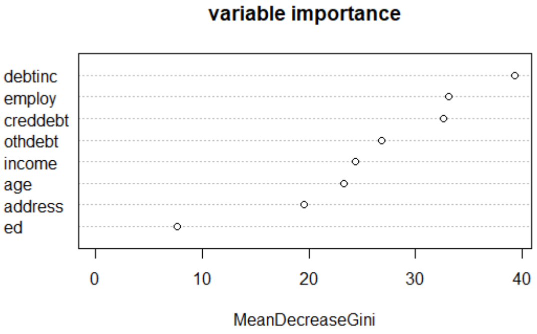
```
#####random forest 1

library(randomForest)
rf = randomForest(default~., data = training)
print(rf)

## prediction

rf_pred = predict(rf,testing)

confusion_matric = table(Actualvalue=testing$default, predictedvalue=rf_pred)

print(confusion_matric)

## tune mtry

tuneRF(training[,-9], training[,9],stepfactor = 0.5,
        plot = TRUE , ntreeTry = 1000,
        trace = TRUE ,
        improve = 0.05)

rf1 = randomForest(default~.,data = training, ntree = 1000, mtry = 2)

rf1

# predict

rf_pred1 = predict(rf1,testing)
confusion_matric1 = table(Actualvalue=testing$default, predictedvalue=rf_pred1)

print(confusion_matric1)
```

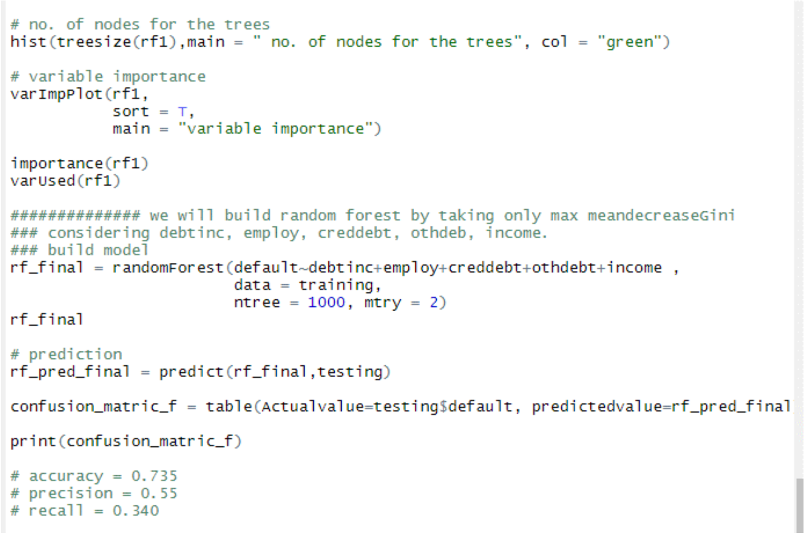


```
Call:
 randomForest(formula = default ~ ., data = training)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 19.27%
Confusion matrix:
    0  1 class.error
0 377 29  0.07142857
1  76 63  0.54676259
```

Using TuneRF function to find the mtry.



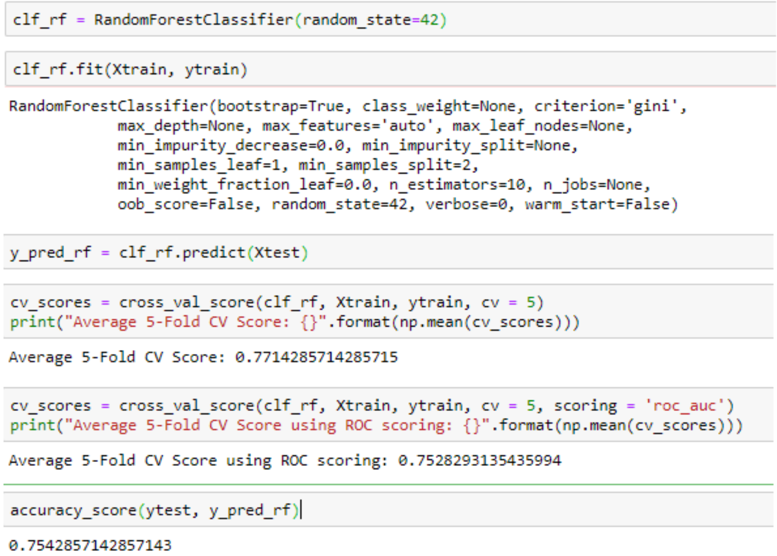




```
# no. of nodes for the trees
hist(treesize(rf1),main = " no. of nodes for the trees", col = "green")

# variable importance
varImpPlot(rf1,
           sort = T,
           main = "variable importance")

importance(rf1)
varUsed(rf1)

############## we will build random forest by taking only max meandecreaseGini
### considering debtinc, employ, creddebt, othdeb, income.
### build model
rf_final = randomForest(default~debtinc+employ+creddebt+othdebt+income ,
                        data = training,
                        ntree = 1000, mtry = 2)
rf_final

# prediction
rf_pred_final = predict(rf_final,testing)

confusion_matric_f = table(Actualvalue=testing$default, predictedvalue=rf_pred_final

print(confusion_matric_f)

# accuracy = 0.735
# precision = 0.55
# recall = 0.340
```

Building Random Forest in Python:

The Random forest classifier with the default parameters achieved a cross validation score of 0.77. The test set accuracy was also 75%. Since the Random Forest Classifier is an ensemble method, hyper parameter tuning will be used to boost performance.



```
clf_rf = RandomForestClassifier(random_state=42)
```

```
clf_rf.fit(Xtrain, ytrain)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=42, verbose=0, warm_start=False)
```

```
y_pred_rf = clf_rf.predict(Xtest)
```

```
cv_scores = cross_val_score(clf_rf, Xtrain, ytrain, cv = 5)
print("Average 5-Fold CV Score: {}".format(np.mean(cv_scores)))
```

```
Average 5-Fold CV Score: 0.7714285714285715
```

```
cv_scores = cross_val_score(clf_rf, Xtrain, ytrain, cv = 5, scoring = 'roc_auc')
print("Average 5-Fold CV Score using ROC scoring: {}".format(np.mean(cv_scores)))
```

```
Average 5-Fold CV Score using ROC scoring: 0.7528293135435994
```

```
accuracy_score(ytest, y_pred_rf)
```

```
0.7542857142857143
```

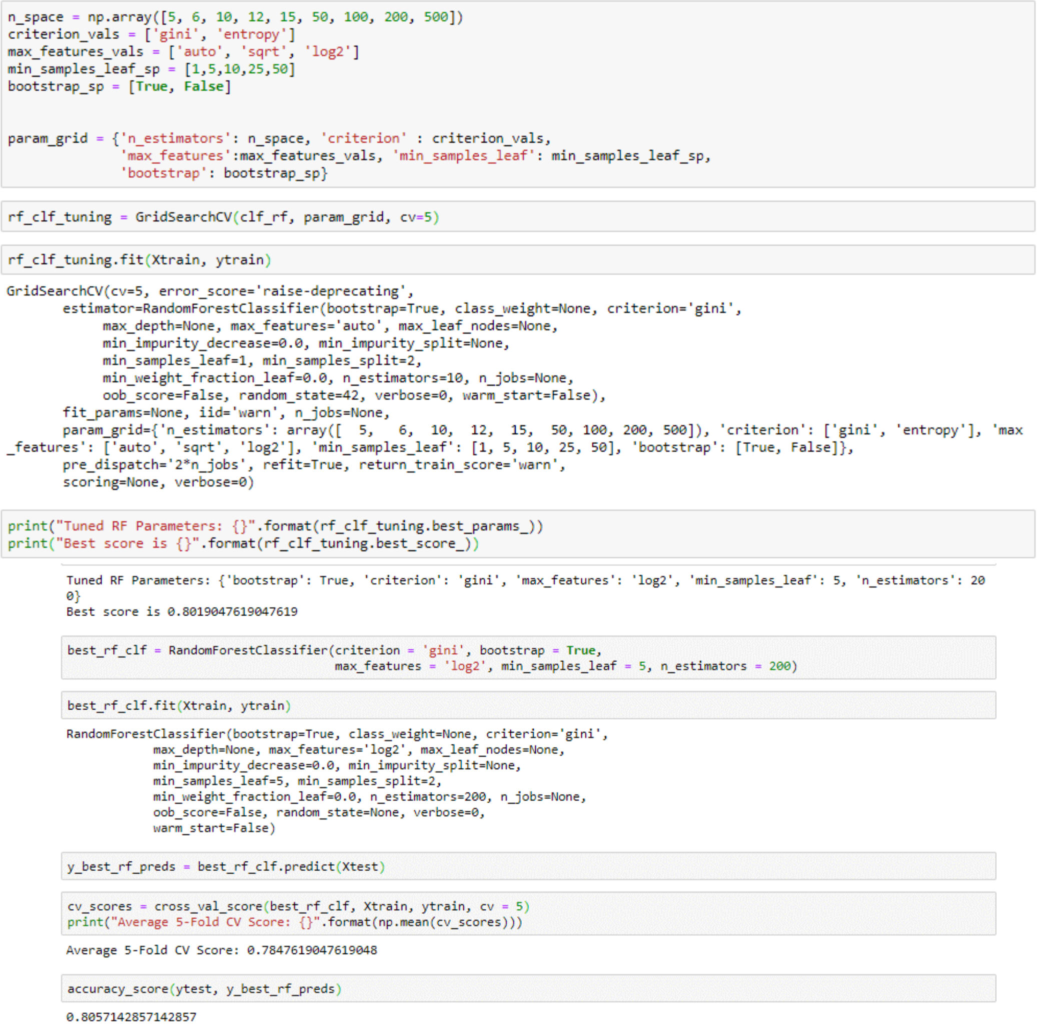
The hyper parameters that were tuned are:
n_estimtors: This represents the maximum number of trees to build before aggregating all of these trees to form a single predictor. Generally, higher number of trees give better results.

max_features: Represents the maximum number of features to consider for a particular tree. Usually, the square root of the total number of features is chosen.

Criterion: We choose between 'gini' and 'entropy.

Bootstrap: The default is true. If set to true (default), the random samples are generated by sampling with replacement.

Min_sample-leaf: A leaf represent the end node of a decision tree. Smaller leaves tend to produce ‘noisier’ models.



```
n_space = np.array([5, 6, 10, 12, 15, 50, 100, 200, 500])
criterion_vals = ['gini', 'entropy']
max_features_vals = ['auto', 'sqrt', 'log2']
min_samples_leaf_sp = [1,5,10,25,50]
bootstrap_sp = [True, False]


param_grid = {'n_estimators': n_space, 'criterion' : criterion_vals,
              'max_features':max_features_vals, 'min_samples_leaf': min_samples_leaf_sp,
              'bootstrap': bootstrap_sp}
```

```
rf_clf_tuning = GridSearchCV(clf_rf, param_grid, cv=5)
```

```
rf_clf_tuning.fit(Xtrain, ytrain)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=42, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': array([  5,   6,  10,  12,  15,  50, 100, 200, 500]), 'criterion': ['gini', 'entropy'], 'max
_features': ['auto', 'sqrt', 'log2'], 'min_samples_leaf': [1, 5, 10, 25, 50], 'bootstrap': [True, False]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
print("Tuned RF Parameters: {}".format(rf_clf_tuning.best_params_))
print("Best score is {}".format(rf_clf_tuning.best_score_))
```

```
Tuned RF Parameters: {'bootstrap': True, 'criterion': 'gini', 'max_features': 'log2', 'min_samples_leaf': 5, 'n_estimators': 20
0}
Best score is 0.8019047619047619
```

```
best_rf_clf = RandomForestClassifier(criterion = 'gini', bootstrap = True,
                                     max_features = 'log2', min_samples_leaf = 5, n_estimators = 200)
```

```
best_rf_clf.fit(Xtrain, ytrain)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='log2', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
y_best_rf_preds = best_rf_clf.predict(Xtest)
```

```
cv_scores = cross_val_score(best_rf_clf, Xtrain, ytrain, cv = 5)
print("Average 5-Fold CV Score: {}".format(np.mean(cv_scores)))
```

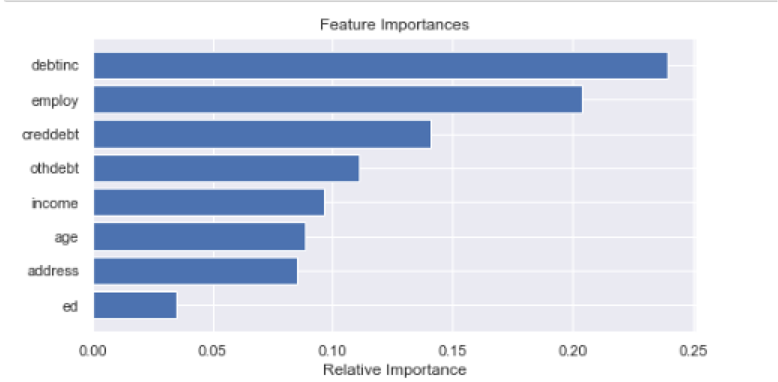
```
Average 5-Fold CV Score: 0.7847619047619048
```

```
accuracy_score(ytest, y_best_rf_preds)
```

```
0.8057142857142857
```

With the tuned hyper parameters, a cross validation score of 0.78 was achieved on the training set. The test set accuracy was found to be 80.5%.

A feature importance plot has been illustrated below:



From the above graph, we can observe that debtinc, employ, creddebt are the variables which explain the dependent variable much i.e., these variables has high variance.

Gradient boosting:

Gradient boosting, like the Random Forest Classifier, is an ensemble algorithm. In this algorithm, detection errors in earlier trials are prioritized and the model then tries to classify these points correctly. These "weak" learners to combines to form a complex but effective predictor.

Using the default parameters, the accuracy on the test set was ~77%. The following parameters were tuned for improving performance:

Loss: ‘Deviance’ and ‘exponential’ were chosen. The logistic loss computes deviance. The exponential loss is calculated using the exponential loss function.

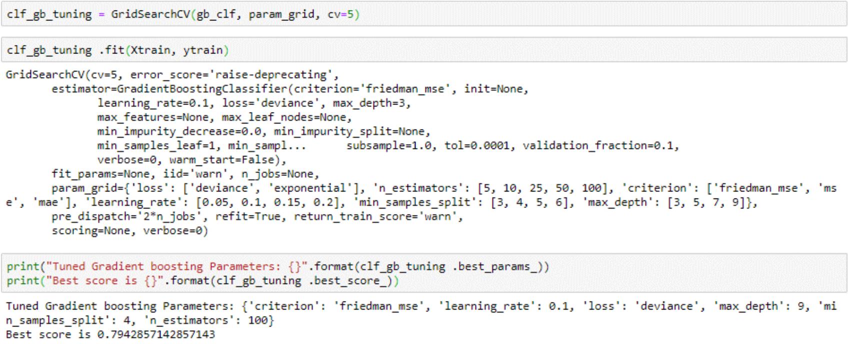
N_estimators: This is the sequential number of trees to be modeled.

Max_depth: It is the maximum depth of the tree. It is used to control overfitting.

Learning_rate: This determines the impact of each tree on the final outcome. Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well.

Min_samples_split: The minimum number of samples in a terminal node or leaf.

Criterion: This parameter measures the quality of split. ‘'friedman_mse', 'mse', 'mae' are the options. ‘MSE’ represents ‘mean squared error’ and ‘mae’ represents the ‘mean absolute error’.



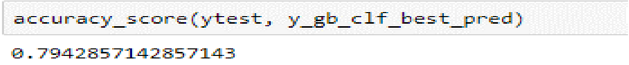
```
clf_gb_tuning = GridSearchCV(gb_clf, param_grid, cv=5)
```

```
clf_gb_tuning .fit(Xtrain, ytrain)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_sampl...      subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'loss': ['deviance', 'exponential'], 'n_estimators': [5, 10, 25, 50, 100], 'criterion': ['friedman_mse', 'ms
e', 'mae'], 'learning_rate': [0.05, 0.1, 0.15, 0.2], 'min_samples_split': [3, 4, 5, 6], 'max_depth': [3, 5, 7, 9]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
print("Tuned Gradient boosting Parameters: {}".format(clf_gb_tuning .best_params_))
print("Best score is {}".format(clf_gb_tuning .best_score_))
```

```
Tuned Gradient boosting Parameters: {'criterion': 'friedman_mse', 'learning_rate': 0.1, 'loss': 'deviance', 'max_depth': 9, 'mi
n_samples_split': 4, 'n_estimators': 100}
Best score is 0.7942857142857143
```



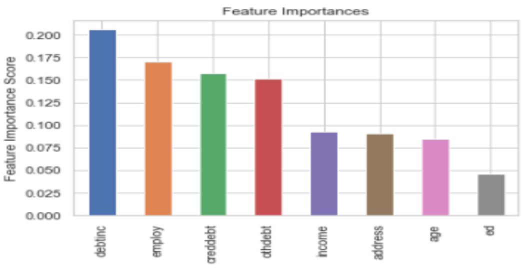
```
accuracy_score(ytest, y_gb_clf_best_pred)
```

```
0.7942857142857143
```

By choosing the best hyper parameters, a test set accuracy of ~80% was obtained. In terms of accuracy, gradient boosting is almost near to random forest.

The feature importance plot for the gradient boosting classifier has been illustrated below.

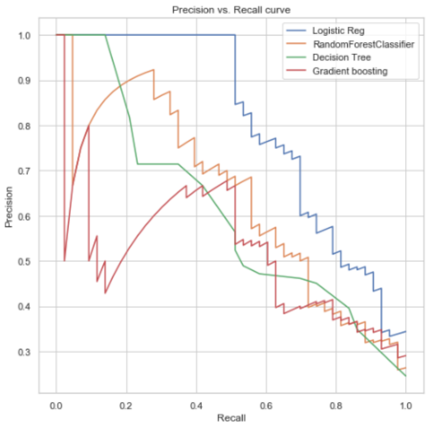
```
Text(0, 0.5, 'Feature Importance Score')
```

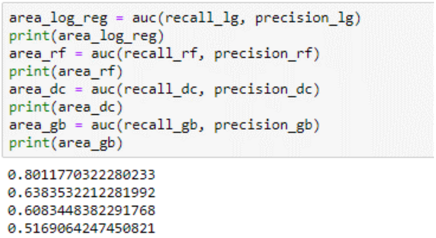


Choosing the best Classifier:

As our dataset is imbalanced, only accuracy cannot judge the performance of the model. A plot of the precision vs. recall curves is illustrated below. Greater the area under the curve, the better the model. Logistic model has higher accuracy both in R and Python.

ROC Curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.
Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.
ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets.

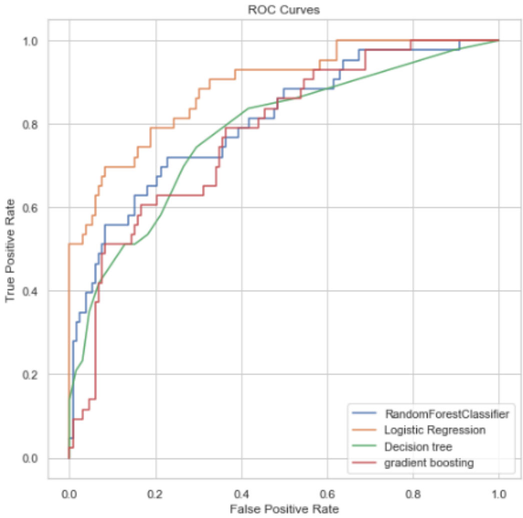


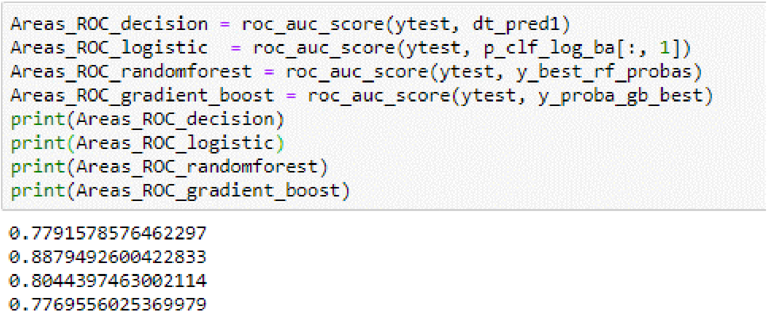


```
area_log_reg = auc(recall_lg, precision_lg)
print(area_log_reg)
area_rf = auc(recall_rf, precision_rf)
print(area_rf)
area_dc = auc(recall_dc, precision_dc)
print(area_dc)
area_gb = auc(recall_gb, precision_gb)
print(area_gb)
```

```
0.8011770322280233
0.6383532212281992
0.6083448382291768
0.5169064247450821
```

Once again, the logistic regression classifier has the max. Area under the curve. Next, we will plot the ROC curves. These curves explain the tradeoff between the true positive rate and the false positive rate. Just as the Precision vs. Recall curves, greater areas under the curve represent better models. This plot has been illustrated in the following page:





```
Areas_ROC_decision = roc_auc_score(ytest, dt_pred1)
Areas_ROC_logistic  = roc_auc_score(ytest, p_clf_log_ba[:, 1])
Areas_ROC_randomforest = roc_auc_score(ytest, y_best_rf_probas)
Areas_ROC_gradient_boost = roc_auc_score(ytest, y_proba_gb_best)
print(Areas_ROC_decision)
print(Areas_ROC_logistic)
print(Areas_ROC_randomforest)
print(Areas_ROC_gradient_boost)
```

```
0.7791578576462297
0.8879492600422833
0.8044397463002114
0.7769556025369979
```

The logistic regression classifier has the maximum area under the Roc curve. The decision tree and gradient boosting has almost same area under the curve.

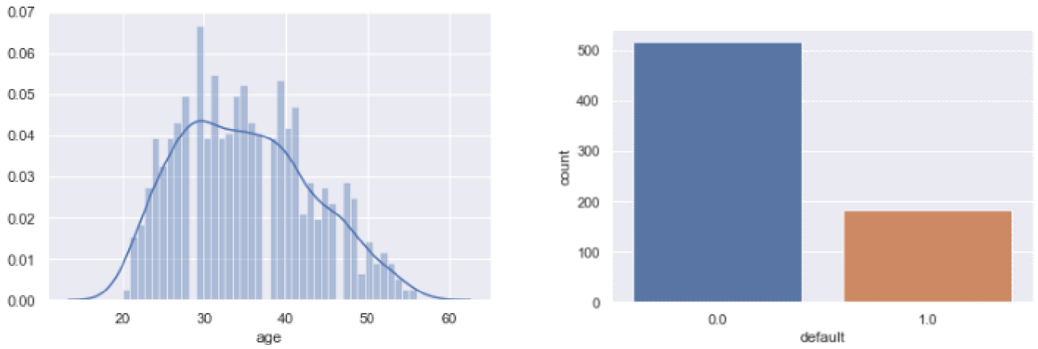
Upon all considerations of evaluation matrix, Logistic regression performs better.

The conclusion is “the logistic regression is performing well for the given bank loan data case”.

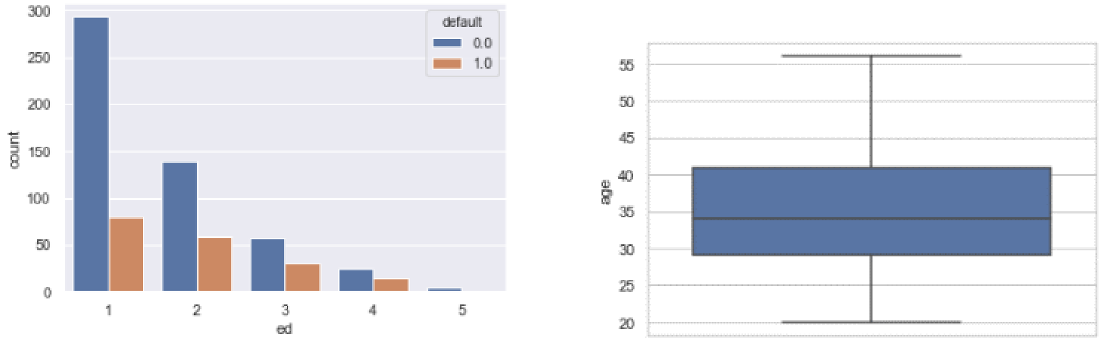
The codes are done in juypter notebook and RStudio.

Appendix – figures and graphs:

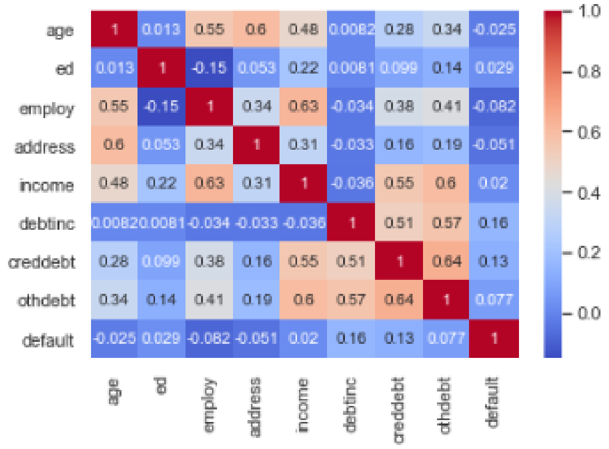
Distribution graph of age and count plot of default variable.



Count plot of education separated by default variable and box plot of age variable.



Correlation heatmap


age 1 0.013 0.55 0.6 0.48 0.0082 0.28 0.34 -0.025
ed 0.013 1 -0.15 0.053 0.22 0.0081 0.099 0.14 0.029
employ 0.55 -0.15 1 0.34 0.63 -0.034 0.38 0.41 -0.082
address 0.6 0.053 0.34 1 0.31 -0.033 0.16 0.19 -0.051
income 0.48 0.22 0.63 0.31 1 -0.036 0.55 0.6 0.02
debtinc 0.0082 0.0081 -0.034 -0.033 -0.036 1 0.51 0.57 0.16
creddebt 0.28 0.099 0.38 0.16 0.55 0.51 1 0.64 0.13
othdebt 0.34 0.14 0.41 0.19 0.6 0.57 0.64 1 0.077
default -0.025 0.029 -0.082 -0.051 0.02 0.16 0.13 0.077 1
age
ed
employ
address
income
debtinc
creddebt
othdebt
default