

PROJECT REPORT

Bank Loan Default Prediction

Using Machine Learning Techniques

Submitted By:

Gaurav D Patil,

As on 13 Jan 2020

TABLE OF CONTENTS

Introduction.....	3
Data and proposed approach.....	3
Missing Value Analysis.....	5
Outlier Analysis.....	5
Machine Learning for Classification and Metrics.....	7
logistic model.....	9
Decision tree.....	12
Random Forest.....	13
Gradient Boosting Classifier.....	17
Choosing the best Classifier.....	18
Appendix – figures and graphs.....	21

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
```

```
> 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 ...
```

- The structure of the dataset after processing:

Number of attributes:

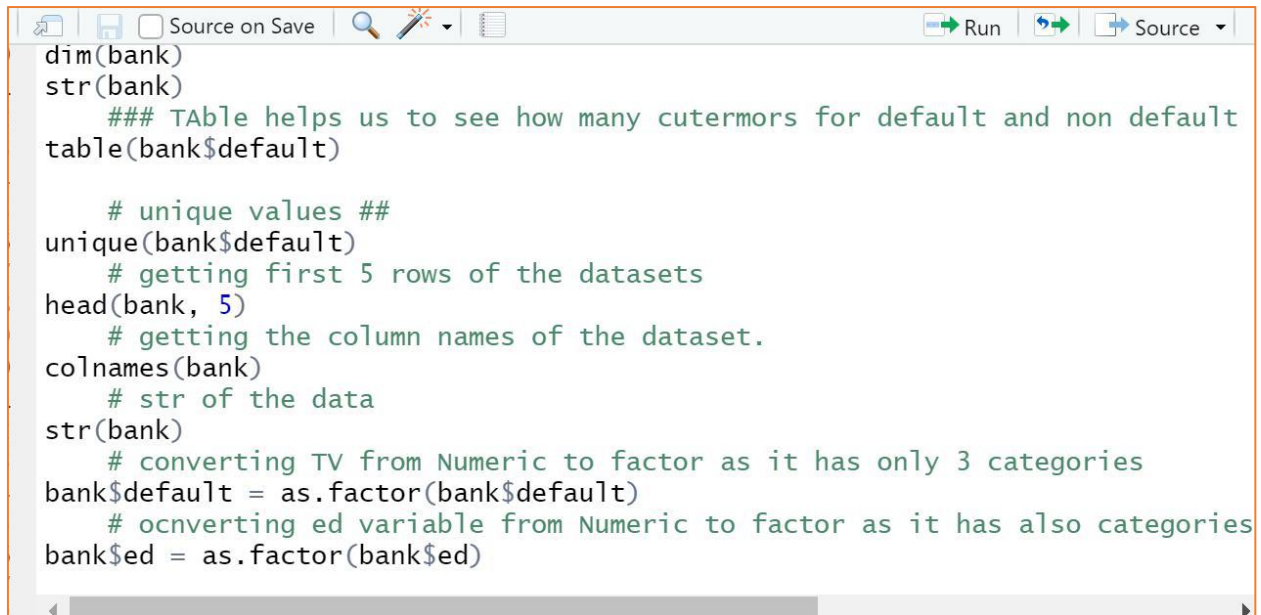
Var. #	Variable Name	Description	Variable Type
1.	Age	Age of each customer	Numerical
2.	Education	Education categories	Categorical
3	Employment	Employment status - Corresponds to job status and being converted to numeric format	Numerical
4	Address	Geographic area - Converted to numeric values	Numerical
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

Bank Loan Default Case variables descriptions.

STEP_1 :- EDA = Exploratory Data Analysis

Here we use to convert original data into proper format,
It means we check distribution of data either it is left skewed or right skewed or uniformly distributed.

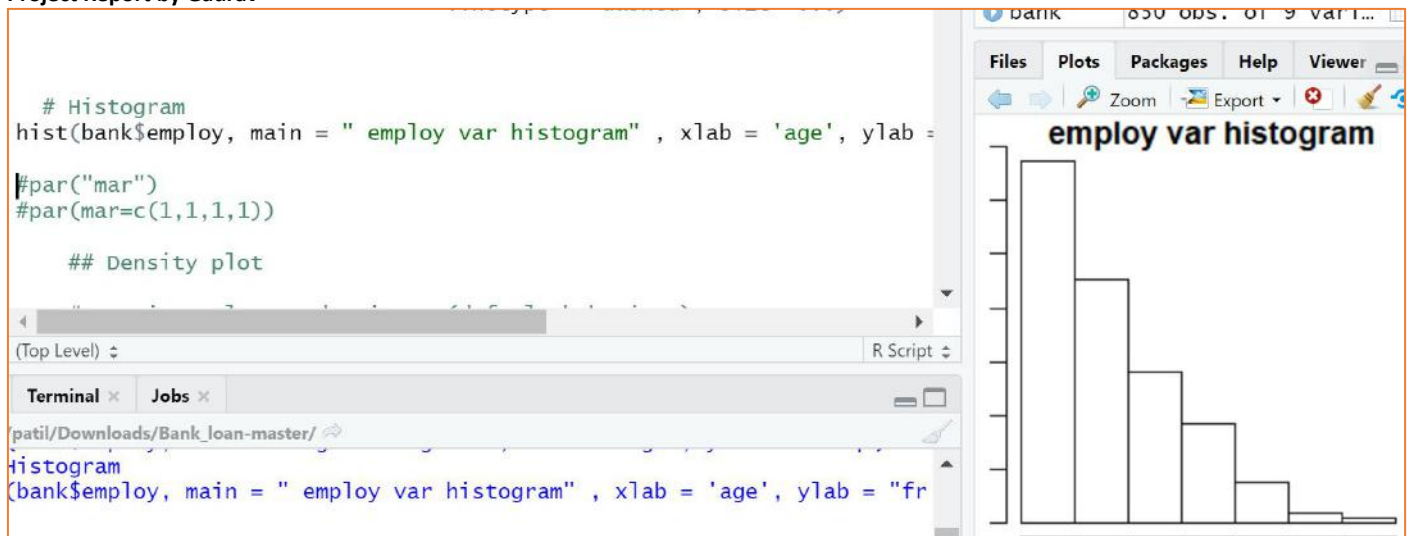
We convert required data variables into proper data types means ,

A screenshot of an RStudio script editor window. The window has a toolbar at the top with icons for file operations, search, and execution. The main area contains R code for exploring a dataset named 'bank'. The code includes commands to check dimensions, structure, unique values, and convert variables to factors. Comments are provided for several steps, such as checking the distribution of 'default' and 'ed' variables.

```
dim(bank)
str(bank)
### Table helps us to see how many cutermors for default and non default
table(bank$default)

# unique values ##
unique(bank$default)
# getting first 5 rows of the datasets
head(bank, 5)
# getting the column names of the dataset.
colnames(bank)
# str of the data
str(bank)
# converting TV from Numeric to factor as it has only 3 categories
bank$default = as.factor(bank$default)
# ocnverting ed variable from Numeric to factor as it has also categories
bank$ed = as.factor(bank$ed)
```

CHECKING DISTRIBUTION OF A VARIABLE



STEP 2 :- Missing Value Analysis:

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

```
8          0.00000  othdebt
> missing_val = missing_val[,c(2,1)]
> missing_val
  columns missing_percentage
9  default          17.64706
1    age              0.00000
2    ed              0.00000
3  employ              0.00000
4  address              0.00000
5  income              0.00000
```

Note: The the dataset has 150 missing labeled values (17.64%) in the dependent variable column.

Impute those Missing Values :-

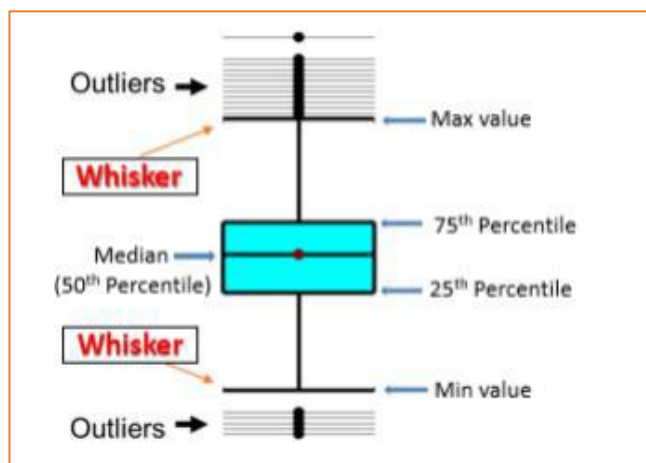
As our data having 17.64% missing values in Default variable which is a categorical var, so we will impute those missing values by Mode imputation method, as illustrated below,

```
# Mode is the score that occurs most frequently in data var,  
# So frequent occurred no. in variable is '0' ( Occured 517 times  
# MODE iputation Method  
  
bank$default[is.na(bank$default)] = 0  
table(bank$default)  
unique(bank$default)
```

```
> bank$default[is.na(bank  
> table(bank$default)  
  
  0    1  
667 183  
> unique(bank$default)  
[1] 1 0  
Levels: 0 1  
> |
```

STEP 3 :- Outlier Analysis:

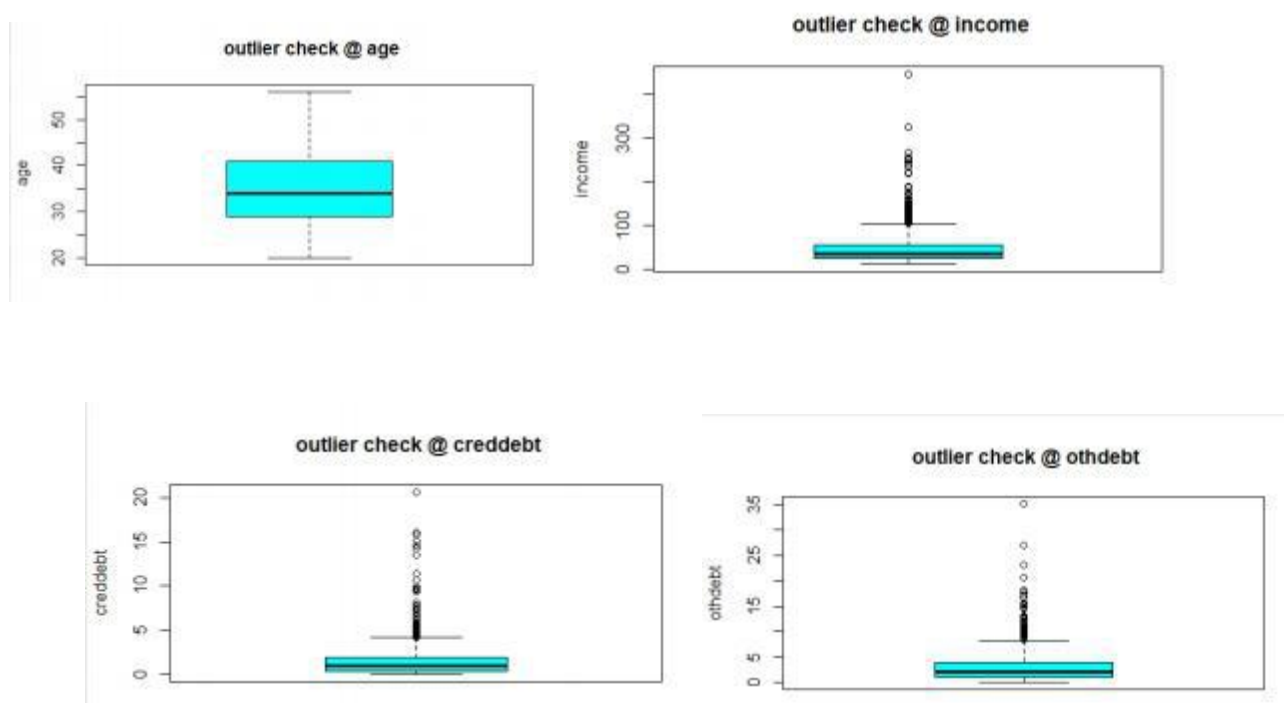
Outliers - An outlier is an observation point that is distant from other observations. These data points could be a result of errors or wrongly recorded observations. These are often excluded from analysis as it could have an adverse effect on the reliability of our model. We will identify outliers with 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. As BoxPlot can only be applied Numeric Vars, collect Num Var,



Variable "age" has zero outliers,

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

As we got outliers now we have two options either to remove them or to impute them,
As we have very less data (850 Obs) removing it will be much loss of information

So we will impute outliers instead of removing it,

Detect & Replace outliers by KNN Imputation method

```
For (i in cnames) {  
  val = bank[,i][bank[,i] %in% boxplot.stats(bank[,i])$out]    # val= extracted outliers  
  #print(length(val))  
  bank[,i][bank[,i] %in% val] = NA                          # %in% is func which checks RHS data  
  present in LHS  
}
```

```
# # #loop to detect n replace outliers from multiple variables instead... to consume less ram  
for(i in cnames){  
  val = bank[,i][bank[,i] %in% boxplot.stats(bank[,i])$out] # val= extracted outliers  
  #print(length(val))  
  bank[,i][bank[,i] %in% val] = NA # %in% is func which checks RHS data present in LHS  
}  
  
sum(is.na(bank))  
View(bank)      ## outliers replaced by NA, now impute those missing values  
# Fill in NA values with the values of the nearest neighbours,  
# by default meth='weighAvg' the function will use a weighted average of the values of the neighbors  
# uses Euclidean distance method, distance = dist between the case n NA values,  
  
bank = knnImputation(bank, k = 3)  
sum(is.na(bank))  
View(bank)  
str(bank) #
```

```
bank = knnImputation(bank, k = 3)  
sum(is.na(bank))  
View(bank)  
str(bank)
```

Now check how much outliers still left in data.

```
# Detect outliers in 'employ' Var
```

```
> ggplot(data = bank, aes(x = "default", y = employ)) +  
  geom_boxplot()
```



As you can see we have successfully replaced outliers with KNN imputation in order to reduce loss of information,

STEP 4 :- FEATURE SELECTION / VARIABLE IMPORTANCE / Dimension Reduction :

FEATURE SELECTION

by Correlation Analysis

by Chi-Sqr test

1) Correlation Plot :- Can only be applied on Numerical Var, to Check dependencies between Continuous variables

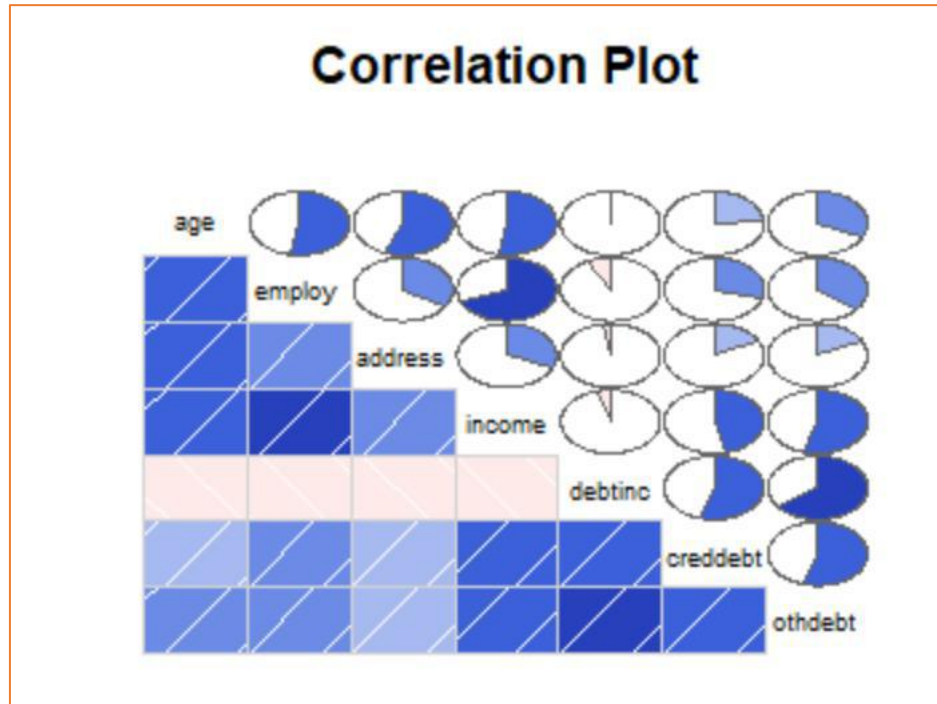
Here we cannot take data = numeric_data , cause it has values before removing outliers,

2) Chi-squared Test of Independence :- Can only applied on categorical vars, to check dependably between two categorical variables

Here we cannot take data = factor_data , cause it has values before removing outliers

Correlation Analysis :-

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



As u can see above in plot, no any variable is identical to other var,
 # it means these vars are no highly correlated variables, so we have to carry all variables
 # and we will put all of them in model developement as all vars are imp right now.

2) Chi-squared Test of Independence :-

Collecting Factorial data

```
factor_index = sapply(bank,is.factor) # checking Fact Vars index (= true)
factor_index
factor_data = bank[,factor_index] # Assigning data to those indexes
head(factor_data)
```

```
head(factor_data)
  ed default
3      1
1      0
1      0
1      0
```

Checking dependencies of default' w.r.t all fact_data variables as,

```
for (i in 1) # as 'default' is TV we will exclude it
{
  print(names(factor_data)[i])
  print(chisq.test(table(factor_data$default,factor_data[,i])))
}
```

OUTPUT =

```
+ print(chisq.test(table(factor_data$default,facto
+ }
[1] "ed"

      Pearson's Chi-squared test

data:  table(factor_data$default, factor_data[, i])
X-squared = 12.384, df = 4, p-value = 0.01471
```

STEP 5 :- FEATURE SCALING :-

To scale the data in same measurable range (0 to 1) or in Unit Standard Deviation ,
To bring high limiting data to certain limit data, so as on variable can be compared with another variable easily.
Can only be performed on continuous variables only,

FEATURE SCALING :-

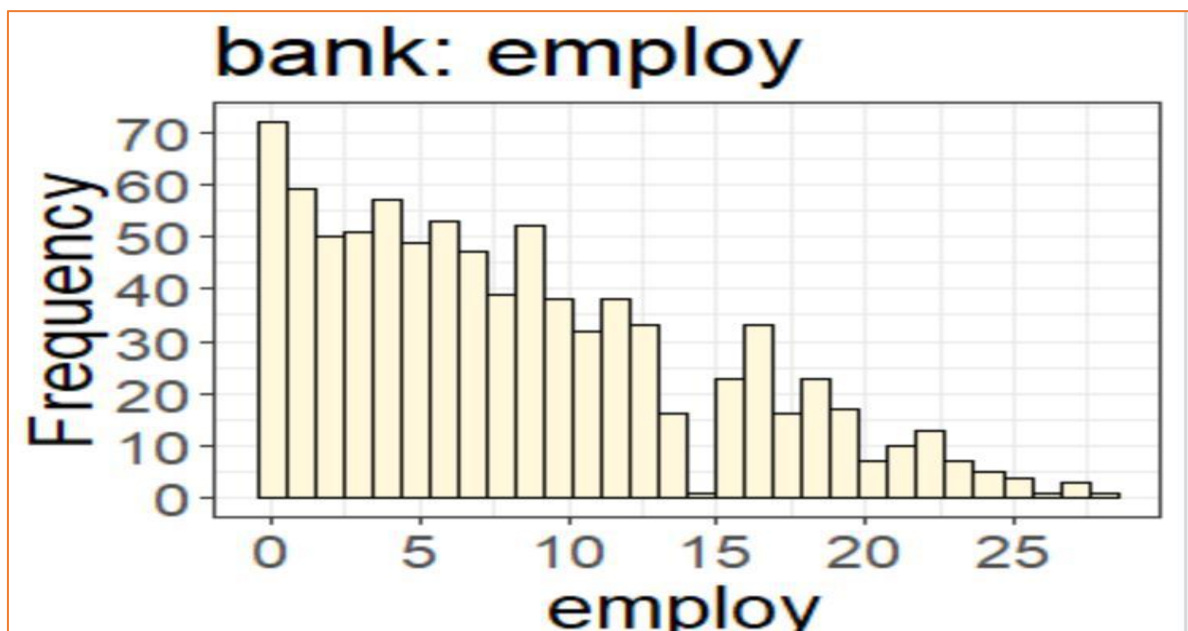
- 1) By Normalization = When data is Normally distributed
- 2) By Standardization (Z-square) = When data is not Normally distributed

So check distribution of data first,

#Histogram

```
> library(scales)
> ggplot(bank, aes_string(x = bank$employ)) +
  geom_histogram(fill="cornsilk", colour = "black") + geom_density() +
  scale_y_continuous(breaks=pretty_breaks(n=10)) +
  scale_x_continuous(breaks=pretty_breaks(n=10))+
  theme_bw() + xlab("employ") + ylab("Frequency") + ggtitle("bank: employ")
  theme(text=element_text(size=20))
```

OUTPUT =



As you can see that our data is not Normally distributed , Thats why we cannot go for Standerdization, Hence we will use Normalization here,

Normalization :-

```
# Saving Continuous vars cause feature scaling can ony be performed on cont.vars
# we cannot take data = numeric_data , cause it has values before removing outliers
# so taking latest upadated 'bank' data
```

View(bank)

	age	ed	employ	address	income	debtinc	creddebt	othde
1	41	3	17	12.00000	63.93101	9.30000	0.6873403	5.0086
2	27	1	10	6.00000	31.00000	17.30000	1.3622020	4.0007
3	40	1	15	14.00000	55.00000	5.50000	0.8560750	2.1689

now lets Normalization of each variable by loop

```
for (i in cnames) {
  print(i)
  bank[,i] = ( bank[,i] - min(bank[,i])) / ( max(bank[,i]) -
min(bank[,i]) )
}
```

View(bank)

Output =

	age	ed	employ	address	income	debtinc	creddebt
1	0.58333333	3	0.60714286	0.4800000	0.57225853	0.34848485	0.16287242
2	0.19444444	1	0.35714286	0.2400000	0.20224719	0.65151515	0.32555618
3	0.55555556	1	0.53571429	0.5600000	0.47191011	0.20454545	0.20354800

Every variable Ranges in between 0 to 1 only

STEP 6 :- MODEL DEVELOPMENT :

So as of now we got data with,

- 1) Zero missing values in it
- 2) Free from outliers
- 3) Feature scaling
- 4) Proper data format

Now lets Start Model Development, as our target variable is binary clarification type so use classification models

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:

		Predicted	
		NEGATIVE(0)	POSITIVE(1)
Actual	NEGATIVE(0)	A (TN)	B (FP)
	POSITIVE(1)	C (FN)	D (TP)
		Recall	

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.

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).

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).

Precision: Out of all the positive classes we have predicted correctly, how many are actually positive.

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.

```
from sklearn.model_selection import train_test_split
```

```
[ ]: #Now devide the data into train and test

X= bank.values[:,0:7]      #saving all var's in X
Y= bank.values[:,7]        #saving 1 dep var in Y

[ ]: # X.head(2)

[ ]: pd.DataFrame(X).head(2)

[ ]:
   0      1      2      3      4      5      6
0  0.343228  0.241205  0.083919  0.862939  0.223301  0.552210  0.141188
1  0.194444  0.333333  0.193548  0.202247  0.417476  0.065719  0.112518

[ ]: #Now split the data into train and test
      #devided 80% and 20% of ALL var's obs (except 'default' var) in X_train and into X_test Respectively
      #devided 80% and 20% of Dep.Var's obs ( default var's) into y_train and into y_test Respectively

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

[ ]: pd.DataFrame(X_train).head(5)

[ ]:
   0      1      2      3      4      5      6
0  0.083333  0.033333  0.000000  0.044944  0.036408  0.000000  0.006108
1  0.444444  0.233333  0.064516  0.337079  0.148058  0.050027  0.044967
```


In R:

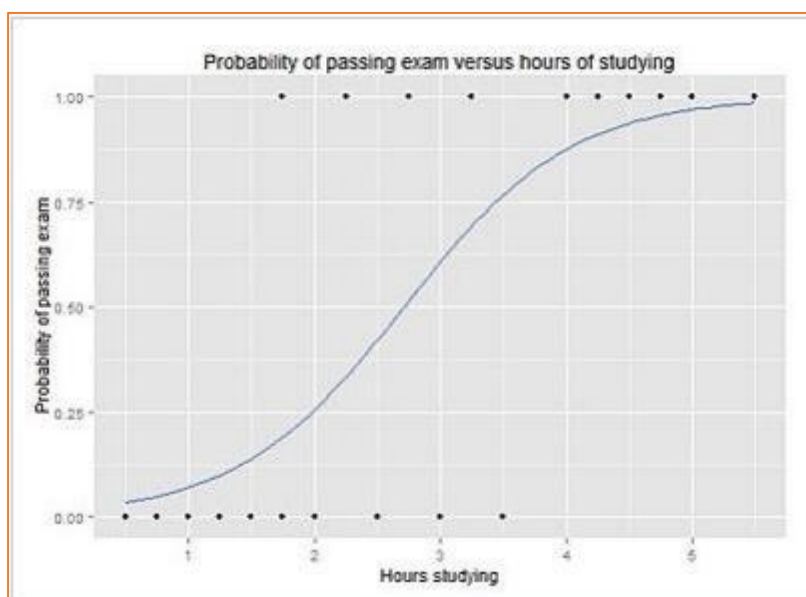
Splitting the data the dataset will be split into the training and test sets. The test set size has been chosen to be 20% of the original dataset. That is, the classification algorithm will train on 80% of the data and then, the model will be used to make predictions on the test set. CaTools is used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

```
#Divide data into train and test using stratified sampling method
set.seed(1234)
train.index = createDataPartition(bank$default, p = .80, list = FALSE)
bank_train = bank[ train.index,]
head(bank_train)
dim(bank_train)
str(bank_train)
sum(is.na(bank_train$default))
```

1) **Logistic Regression:**

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

Figure showing sigmoid curve.



Assumptions :-

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

We satisfy in all of the above assumptions

Building Logistic regression in R:

Using glm (Generalized linear models) to build regression model and also checking variable importance,

```

387 ##### logistic regression #####
388
389 library(caTools)|
390 model = glm(default~.,bank_train, family = "binomial")
391 varImp(model)
392 order(-varImp(model)) # descending order , Most imp = employ=5.6
393
394 ##### Var selection for building model
395 # var imp wrt each independent var
396
389:17 # logistic regression

```

Console Terminal x Jobs x

C:/Users/patil/Downloads/Bank_loan-master/

```

> varImp(model)

```

	Overall
age	1.64576810
ed2	1.14022975
ed3	2.00152683
ed4	0.99046765
ed5	0.02414039
employ	5.11691290
address	3.41650493
income	0.03967375

Most imp = employ=5.6

```

othdebt 0.03626753
> model = glm(default~.,bank_train, family = "binomial")
> model

```

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

Coefficients:

(Intercept)	age	ed2	ed3	ed4	ed5
-1.71698	1.02324	0.29434	0.67912	-0.51763	-12.00578
employ	address	income	debtinc	creddebt	othdebt
-4.43680	-1.98569	-0.04525	2.78146	1.32572	-0.04063

Degrees of Freedom: 680 Total (i.e. Null); 669 Residual

Null Deviance: 710.4

Residual Deviance: 564.7 AIC: 588.7

We built model for dependent variable with respect to all independent variables.

Now build model for each Independent variable,

Build main Logistic Regression model with Important Variables

```
##### build main LogReg model with important variables #####  
varImp(model) # See imp vars  
model6 = glm(default~creddebt+debtinc+address+employ,bank_train, family = "binomial")  
  
summary(model6)  
model6  
res = predict(model6, bank_test, type = "response")  
res  
str(res)  
dim(res)  
  
confusion_matric = table(Actualvalue=bank_test$default, predictedvalue=res>0.5)  
  
print(confusion_matric)  
precision(confusion_matric) # 129/(129+4) #96.99 %  
recall(confusion_matric) # 129/(129+24) #  
accuracy = (129+12)/(129+12+24+4)*100  
print(accuracy)
```

OUTPUT:

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.6667	-0.6760	-0.4249	-0.1609	2.8544

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)	
(Intercept)	-1.4033	0.2669	-5.258	1.45e-07	***
creddebt	1.5374	0.6040	2.545	0.01091	*
debtinc	2.6370	0.5720	4.610	4.03e-06	***
address	-1.5190	0.5009	-3.033	0.00242	**
employ	-4.1075	0.6712	-6.120	9.36e-10	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 710.44 on 680 degrees of freedom
 Residual deviance: 575.40 on 676 degrees of freedom
 AIC: 585.4

Number of Fisher Scoring iterations: 5

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.

Confusion Matrix :-

```

> confusion_matrix = table(Actualvalue, predictedvalue)
> print(confusion_matrix)

```

	predictedvalue	
Actualvalue	FALSE	TRUE
0	129	4
1	24	12

```
<environment: namespace:caret>
> precision = 129/(129+4) #96.99 %
> precision
[1] 0.9699248
> recall = 129/(129+24) #
> recall
[1] 0.8431373
> accuracy = (129+12)/(129+12+24+4)*
> print(accuracy)
[1] 83.43195
> # FNR = FN/FN+TP
> FNR = 24/(24+12) # = 66.66 %
> FNR
[1] 0.6666667
> (12)/(12+24)
```

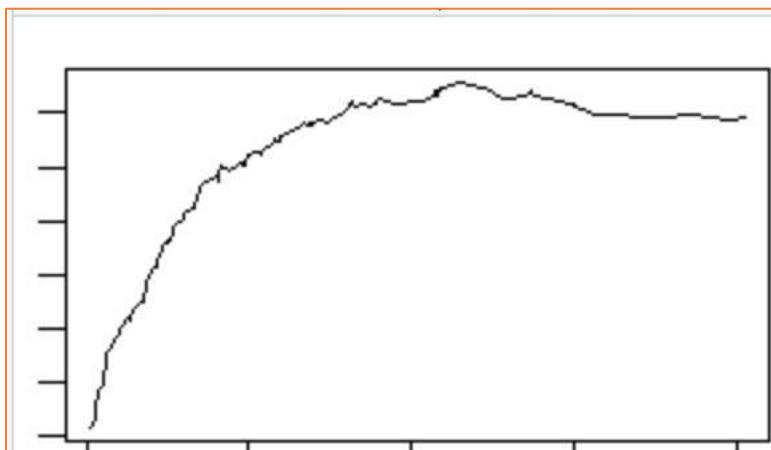
False Negative rate = No. of wrongly -ve predicted, No. of -ve values predicted, which later found wrong

False Positive rate = No. of wrongly +ve predicted, No. of +ve values predicted, which later found wrong

Threshold Evaluation :-

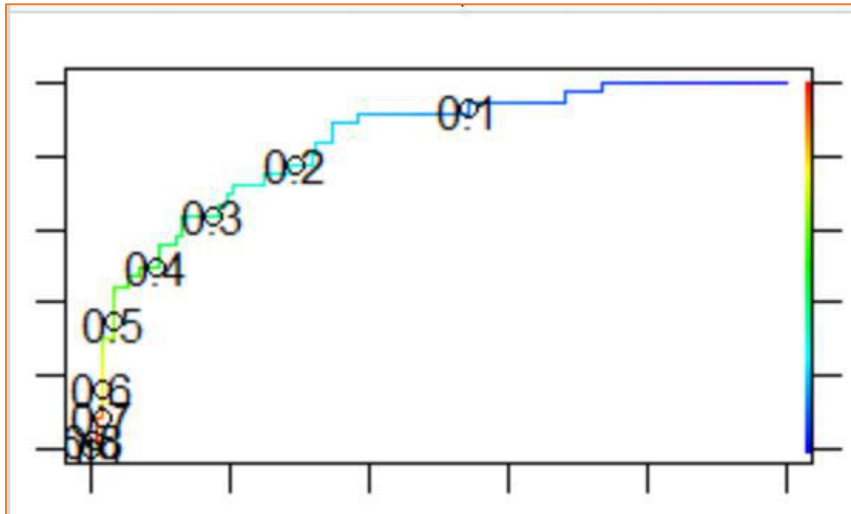
AUC CURVE :-

```
> library(ROCR)
> pred_log = prediction(res,bank_test$default)
> acc = performance(pred_log,"acc")
> plot(acc)
#par("mar") # To neglect plotting error
#par(mar=c(1,1,1,1))
```



ROC Curve :-

```
roc_curve = performance(pred_log, "tpr" , "fpr")
plot(roc_curve,colorize = T)
```



So by using threshold value of 0.4 we can increase the true positive rate

```
confusion_matrix2 = table(Actualvalue= bank_test$default, predictedvalue =res>0.5)
```

OUTPUT :-

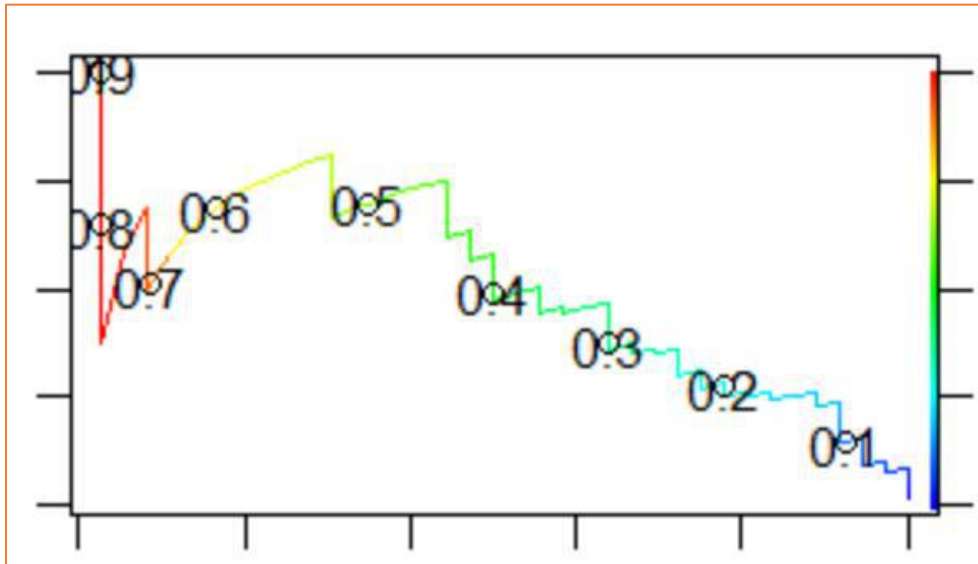
```
> #str(res)
> print(confusion_matrix2)
      predictedvalue
Actualvalue FALSE TRUE
      0      129    4
      1      24   12
> accuracy2 = (129+12)/(129+12+24+4)
> print(accuracy2)
[1] 0.8343195
> |
```

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 performance 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.

Precision Recall curve

```
>library(PRROC)
>PRC_curve = performance(pred_log, "prec", "rec")
>plot(PRC_curve, colorize = T, print.cutoffs.at=seq(0.1,by=0.1)
```



Finally Logistic Regression Accuracy = 83.43 %, FNR = 66.66 %

Building Logistic Regression By Python

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

```
>clf_log = LogisticRegression()
```

In most data sets, we will have features with highly varying range, units and magnitudes. Features with greater magnitudes will carry a greater value than those with low magnitudes. To nullify this effect, feature scaling is performed.

```
> train_scale = scale(X_train)
```

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



```
[578]: train_scale = scale(X_train)
       # Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=42, test_si

In [ ]:

[579]: clf_log.fit(X_train, y_train)

Out[579]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                             intercept_scaling=1, l1_ratio=None, max_iter=100,
                             multi_class='auto', n_jobs=None, penalty='l2',
                             random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                             warm_start=False)

[580]: y_log_pred = clf_log.predict(X_test)

[581]: accuracy_score(y_test, y_log_pred)

Out[581]: 0.8058823529411765

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

[583]: param_grid = {'C': C_space}

[584]: clf_log_tuning = GridSearchCV(clf_log, param_grid, cv=5)
```

Using Logistic Regression, we achieved a training set accuracy of 80.5 % (in terms of training set accuracy alone.)

Next, the model will be fit with the best parameter of C and then it will be used to predict the test set

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

```
[585]: clf_log = LogisticRegression(C = 1.0)

[586]: clf_log.fit(X_train, y_train)

In[586]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
    intercept_scaling=1, l1_ratio=None, max_iter=100,
    multi_class='auto', n_jobs=None, penalty='l2',
    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
    warm_start=False)

[587]: y_preds = clf_log.predict(X_test)

[588]: p_clf_log_ba = clf_log.predict_proba(X_test)

[589]: accuracy_score(y_test, y_preds)

In[589]: 0.8058823529411765
```

Print classification report.

```

print(classification_report(y_test, y_preds))

```

	precision	recall	f1-score	support
0.0	0.81	0.99	0.89	134
1.0	0.71	0.14	0.23	36
accuracy			0.81	170
macro avg	0.76	0.56	0.56	170
weighted avg	0.79	0.81	0.75	170

```

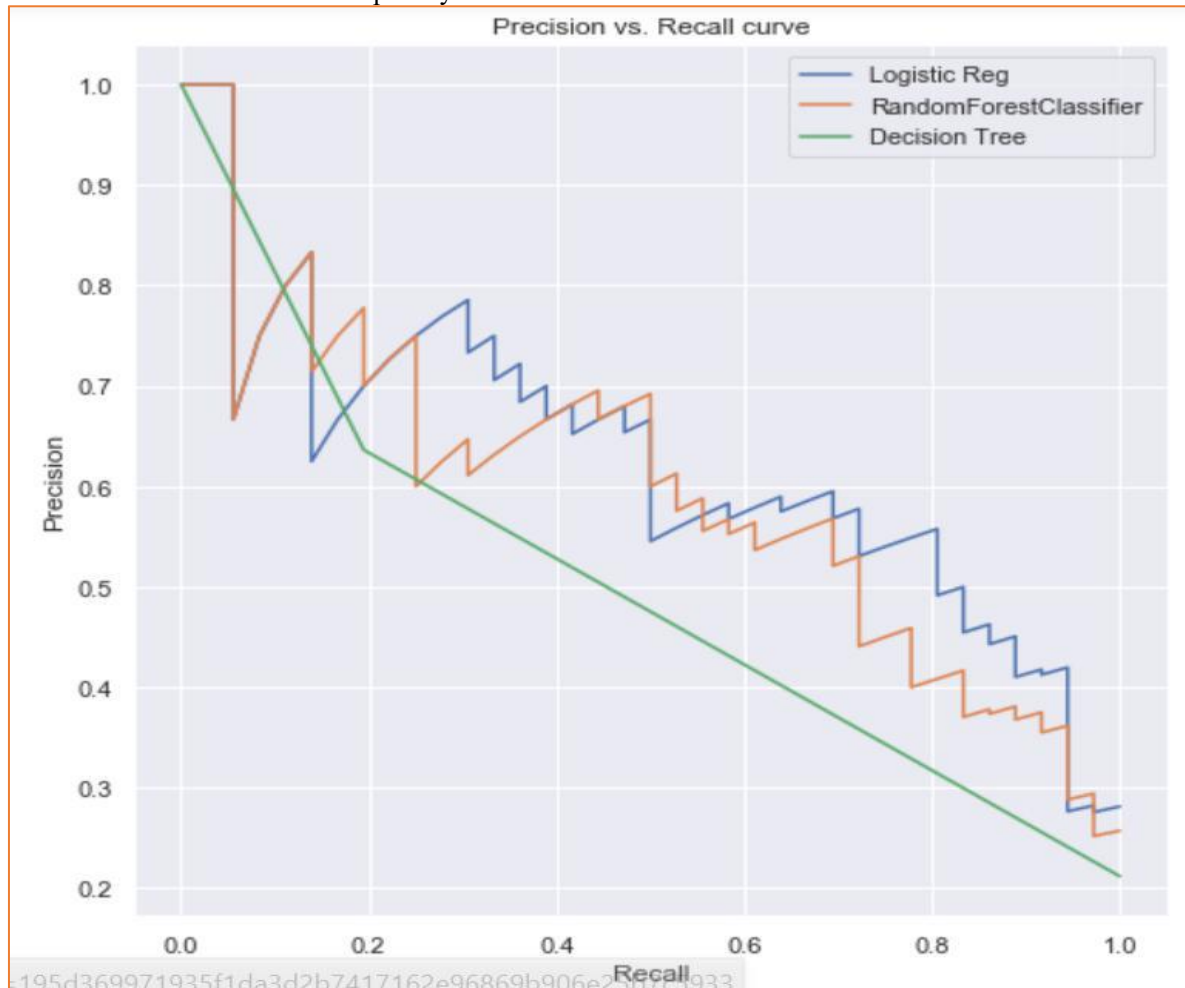
precision_lg, recall_lg, thresholds_lg = precision_recall_curve(y_test, p_clf_log_ba[:, 1])

fpr_lg, tpr_lg, thresholds_lg = roc_curve(y_test, p_clf_log_ba[:, 1])

fig, ax = plt.subplots(figsize=(8,8))
plt.plot(recall_lg, precision_lg)
plt.plot(recall_rf, precision_rf)
plt.plot(recall_dc, precision_dc)
plt.legend(('Logistic Reg', 'RandomForestClassifier', 'Decision Tree'))
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision vs. Recall curve')

Text(0.5, 1.0, 'Precision vs. Recall curve')

```



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 Algorithm

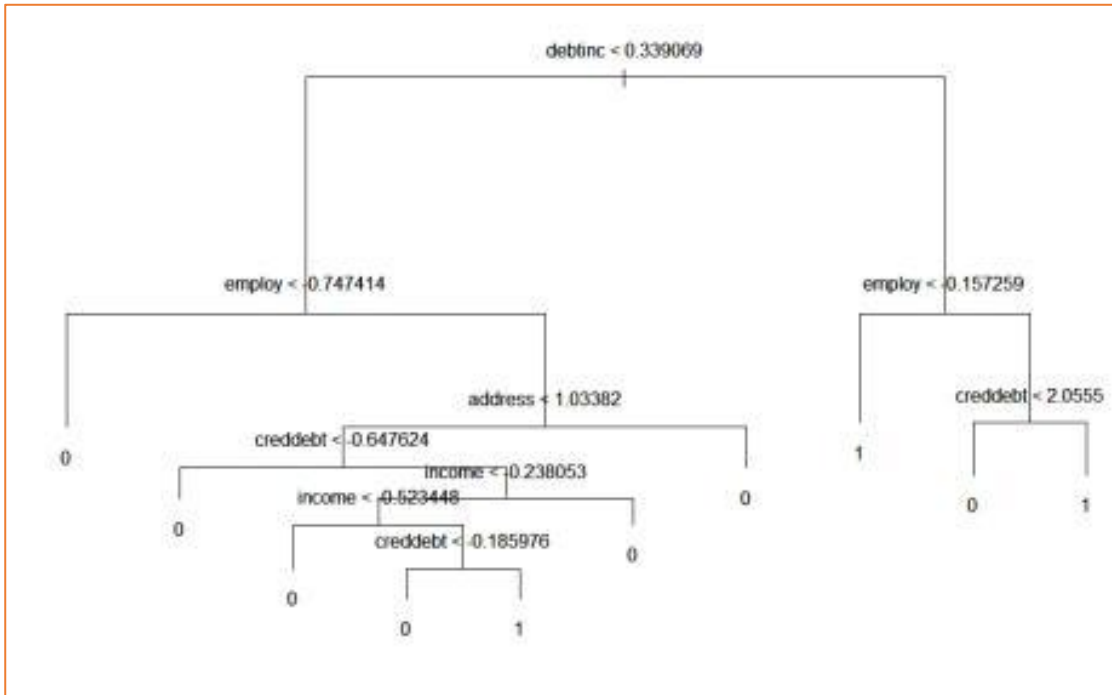
A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it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

Decision Tree in R :-

#Develop Model on training data

```
>C50_model = C5.0(default ~., bank_train, trials = 100, rules = TRUE)
>bank_train$default = as.factor(bank_train$default)
```

```
>str(bank_train$default)
### plotting
>plot(C50_model)
#Summary of DT model
>summary(C50_model)
# 631/(631+50)
```



```
##Evaluate the performance of classification model
ConfMatrix_C50 = table(bank_test$default, C50_Predictions)
confusionMatrix(ConfMatrix_C50)

accuracy2=(127+8) / (124+8+28+9)*100    # 79.88 %
accuracy2

#False Negative rate
# FNR = FN/(FN+TP)
FNR2 = 28/(28+8)*100    # 77.77 %
FNR2

#Recall
#RC = TP/(TP+FN)
RC2 = 8/(8+28)*100    # 22.22 %
RC2

# As we can say now, tht DT is performing poor here,
# although Accuracy is good = 79.88 %
```

Decision Tree in Python

```

age    employ    address    income    debtinc    creddebt    othdebt    default
0    0.343228    0.241205    0.083919    0.862939    0.223301    0.552210    0.141188    1
1    0.194444    0.333333    0.193548    0.202247    0.417476    0.065719    0.112518    0

: #Now devide the data into train and test

X= bank.values[:,0:7]      #saving all  var's in X
Y= bank.values[:,7]        #saving 1 dep var in Y

: # X.head(2)

: pd.DataFrame(X).head(2)

:
      0      1      2      3      4      5      6
0    0.343228    0.241205    0.083919    0.862939    0.223301    0.552210    0.141188
1    0.194444    0.333333    0.193548    0.202247    0.417476    0.065719    0.112518

: #Now split the data into train and test
    #devided 80% and 20% of ALL var's (except 'default' var) in X_train and into X_test Respectively
    #devided 80% and 20% of Dep.Var's obs ( default var's) into y_train and into y_test Respectively

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

```


Bank Loan Default Prediction report by Gaurav

```
] from random import randrange , uniform
from sklearn import tree #for Decision Tree

]:
#Now as data is devided , Build your DT_Model on train (X_train =80% excluding dep var) data and after building test your model

clf = tree.DecisionTreeClassifier(criterion = "gini", random_state = 10, max_depth=5, min_samples_leaf=7 )
clf.fit(X_train,y_train) # criterion='entropy' for C5.0 and gini for CART statistical aproch
# clf = tree.DecisionTreeClassifier(criterion='entropy').fit(X_train,y_train)

#Now as data is devided , Build your DT_Model on train (X_train =80%ofALL) data and after building test your model on test data
# clf = tree.DecisionTreeClassifier(criterion='entropy').fit(X_train,y_train) #entropy for C5.0 and gini for CART statistica
<
]: DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                        random_state=10, splitter='best')

]: # Now predict new TV on this clf model

y_pred = clf.predict(X_test)
y_pred.shape

]: (170,)
```

Error metrics – Classification type

Evaluate Model Performance

```
36]: from sklearn.metrics import confusion_matrix

37]: CM = confusion_matrix(y_test, y_pred)
CM
37]: array([[130,  4],
          [ 29,  7]], dtype=int64)

38]: # OR
CM = pd.crosstab(y_test,y_pred)

#Assign values to parameters
TN= CM.iloc[0,0]
FN= CM.iloc[1,0]
TP= CM.iloc[1,1]
FP= CM.iloc[0,1]

CM

38]:
```

	col_0	0.0	1.0
row_0			
0.0	130	4	
1.0	29	7	

```

accuracy_score (y_test,y_pred)*100
[540]: 80.58823529411765

[541]: # OR
      ((TP+TN)*100 / (TP+TN+FP+FN))
[541]: 80.58823529411765

[542]: # FNR
      (FN*100) / (FN+TP)
[542]: 80.55555555555556

[545]: print(classification_report( y_test , y_pred))

```

	precision	recall	f1-score	support
0.0	0.82	0.97	0.89	134
1.0	0.64	0.19	0.30	36
accuracy			0.81	170
macro avg	0.73	0.58	0.59	170
weighted avg	0.78	0.81	0.76	170

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 ~80%.

Random Forest Algorithm

Why Random Forests over decision trees?

Decision trees 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.

Random Forest by R

Now as we have already partitioned data now we just have to build the RF model on top of it,

```
>library(randomForest)
>rf = randomForest(default~., data = bank_train)
>print(rf)
```

Now predict the new Target Variable on this model

```
rf_pred = predict(rf,bank_test)

confusion_matrix3 = table(Actualvalue=bank_test$default, predictedvalue=rf_pred)

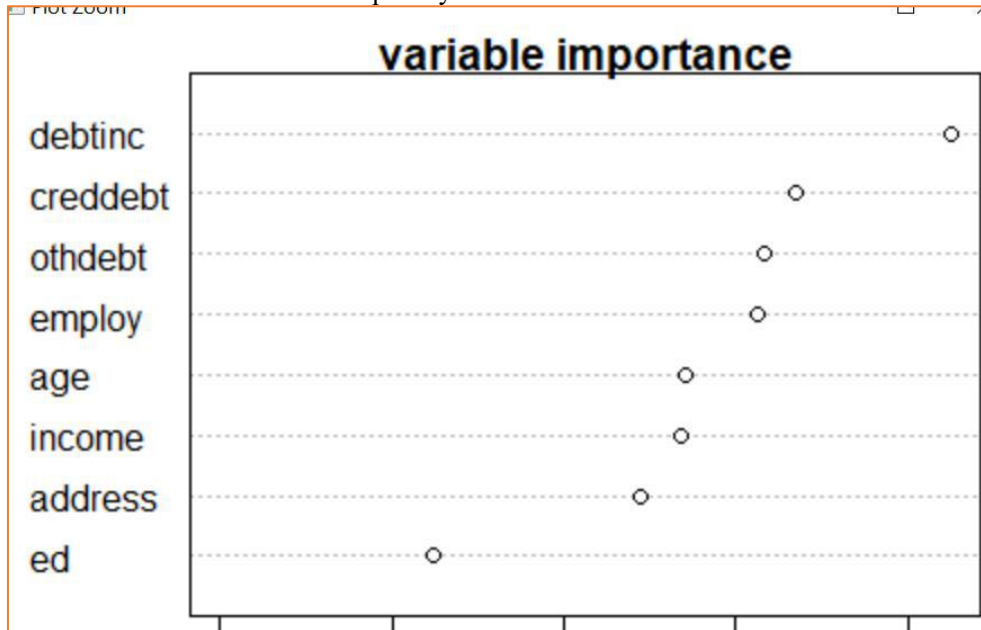
print(confusion_matrix3)
accuracy3 = (129+10) / (129+10+26+4)*100 # = 82.24 %
accuracy3
```

As we can see efficiency of Random Forest (82.24 %) is greater than Decision Tree (79.88, FNR= 71%) but less than Logistic Regression accuracy (83.43 % , FNR = 66.66 %)

Efficiency of DT < RF < LR

Sorting important variables as

```
[1] 0 7 0 5 1 5 4 2
> importance(rf1)
      MeanDecreaseGini
age                27.02962
ed                 12.43948
employ            31.29328
address           24.47187
income            26.80243
debtinc           42.57898
creddebt          33.53675
othdebt           31.61828
> |
```



We will build random forest by taking only max meandecreaseGini, Considering debtinc, employ, creddebt, othdeb, income.

Build final model on these imp vars

```
rf_final = randomForest(default~debtinc+employ+creddebt+othdebt+income ,
                        data = bank_train,
                        ntree = 1000, mtry = 2)

rf_final
|
# prediction
rf_pred_final = predict(rf_final,bank_test)

confusion_matric_f = table(Actualvalue=bank_test$default, predictedvalue=rf_pred

print(confusion_matric_f) # (112+12)/(112+12+16+30)*100
precision(confusion_matric_f)
recall(confusion_matric_f)
accuracyRF_F = (125+13)/(125+13+23+8)*100
accuracyRF_F
# Final Accuracy of RF = 81.65%

# precision = 0.55 # = TP/ (TP+)
```

Final Accuracy of RF = 81.65%

we can not decide the performance of model only based on the accuracy

we need to have a good trade off between precision and recall.

logistic model has 83.43 % accuracy with good trade-off b/t prec and recall.

we can see Efficiency of RF (82.24 %, FNR = 72%) is greater than

Decision Tree (79.88, FNR= 71%) but less than LogReg (83.43 %,FNR = 66.66 %)

```
547]: clf_rf = RandomForestClassifier(random_state=42).fit(X_train, y_train)
      clf_rf

547]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                             criterion='gini', max_depth=None, max_features='auto',
                             max_leaf_nodes=None, max_samples=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=100,
                             n_jobs=None, oob_score=False, random_state=42, verbose=0,
                             warm_start=False)

548]: # Now Lets apply this Model on Test data

      y_pred_rf = clf_rf.predict(X_test)
      y_pred_rf

548]: array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
            0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
            0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0.,
            0., 0., 0., 0., 1., 0., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
            0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
            0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
```

```
550]: #Assign this Conf Matrix values to parameters

TN= CM.iloc[0,0]
FN= CM.iloc[1,0]
TP= CM.iloc[1,1]
FP= CM.iloc[0,1]

551]: # % Accuracy
# or      ((TP+TN)*100 / (TP+TN+FP+FN))

accuracy_score (y_test, RF_predictions)*100      # DT accuracy_score was = 75 %

551]: 74.11764705882354

552]: # FNR      from Dec Tree FNR = 61 %
(FN*100) / (FN+TP)

552]: 83.33333333333333

TRY ONCE BY INCREASING NO OF TREES =500

553]:
clf_rf = RandomForestClassifier(n_estimators = 500, random_state=42).fit(X_train, y_
clf_rf
```

Page 35 of 39

n_estimators: 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

SO NOW AUTO-TUNE, N CHECK HOW MANY TREES WILL BE BEST FOR BEST ACCURACY

```
SO NOW AUTO-TUNE, N CHECK HOW MANY TREES WILL BE BEST FOR BEST ACCURACY

n [ ]:

[563]: # Cross validation with Auto tuned Parameters

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}

[564]: rf_clf_tuning = GridSearchCV(clf_rf, param_grid, cv=5)

[191]: # Now out of above provided all data , now u (model) tell me, By which of above parametrs I will get Maxim
# therefore Auto Tune yourself n generate those Best fitted parameters for RF model

# rf_clf_tuning.fit(X_train, y_train)
```

Calculate Accuracy by predicted TV got by Auto-Tuned parameters

```

9]: # Calculate Accuracy by predicted TV got by Auto-Tuned parameters
accuracy_score(y_test, y_best_rf_preds)*100

9]: 81.76470588235294

0]: y_best_rf_probas = best_rf_clf.predict_proba(X_test)[:,-1]

1]: print(classification_report(y_test, (y_best_rf_probas > 0.5).astype(int)))

```

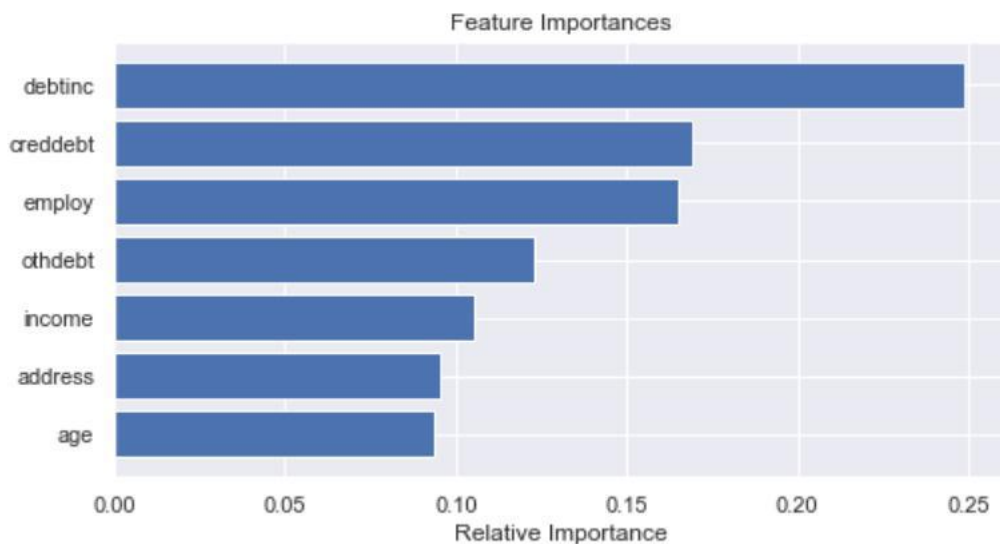
	precision	recall	f1-score	support
0.0	0.82	0.98	0.89	134
1.0	0.73	0.22	0.34	36
accuracy			0.82	170
macro avg	0.78	0.60	0.62	170
weighted avg	0.80	0.82	0.78	170

```

2]: fig, ax = plt.subplots(figsize=(8,4))
features = bank.columns
importances = best_rf_clf.feature_importances_
indices = np.argsort(importances)

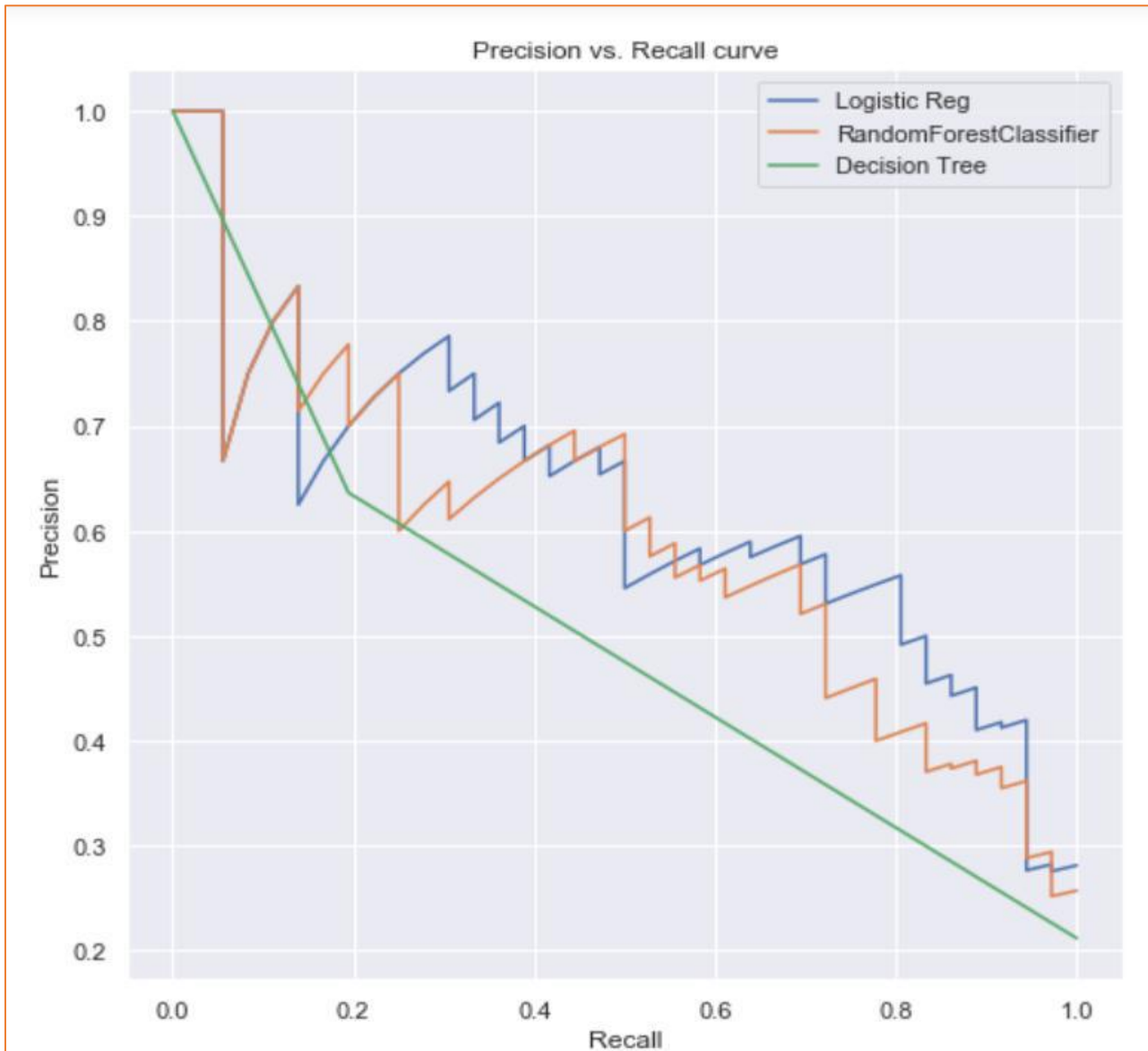
```

Feature Importance in Python

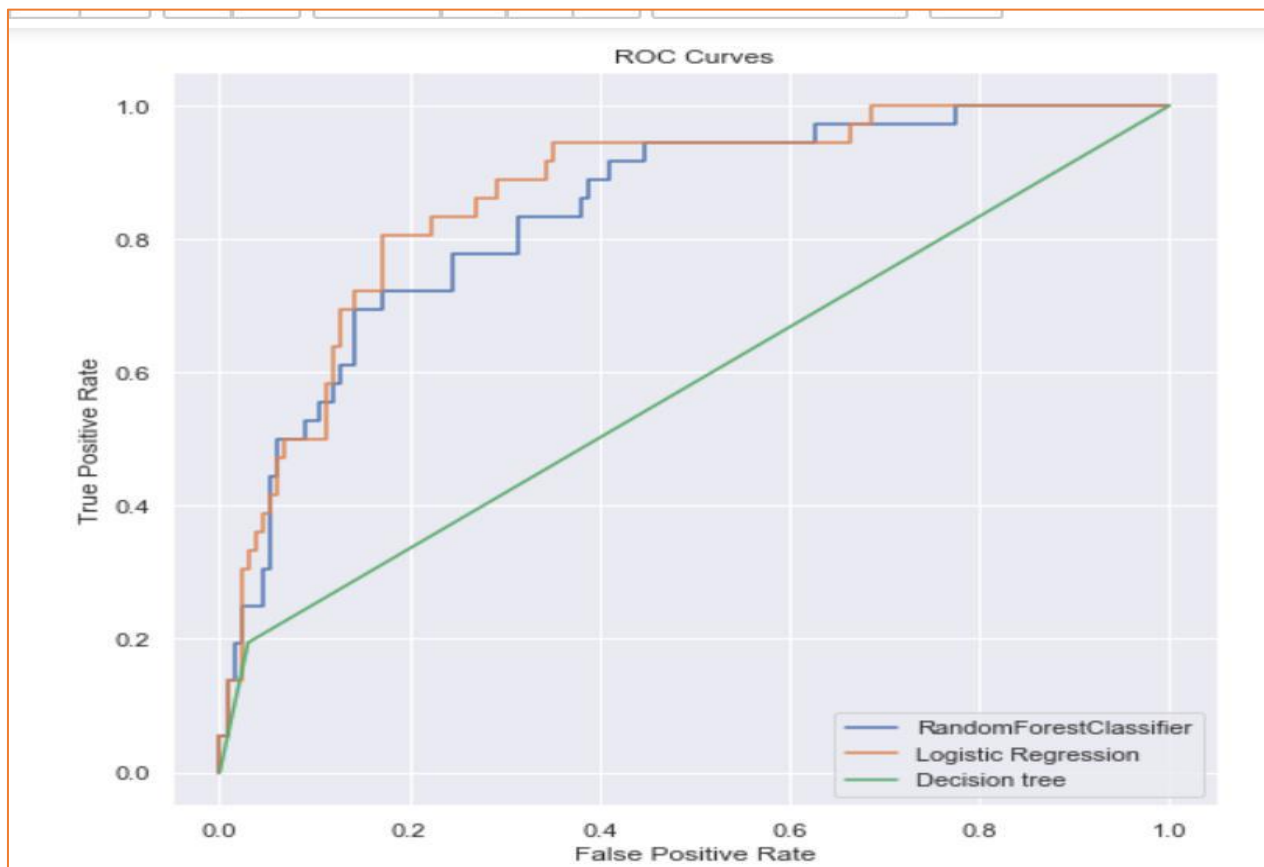


From the above graph, we can observe that debtinc, employ, creddebt are the variables which explain the dependent variable much i.e., these variables has high variance.

Precision Vs Recall Curve :-



ROC Curve :-



```
] Areas_ROC_decision = roc_auc_score(y_test, dt_pred1)
Areas_ROC_logistic = roc_auc_score(y_test, p_clf_log_ba[:, 1])
Areas_ROC_randomforest = roc_auc_score(y_test, y_best_rf_proba)
print(Areas_ROC_decision)
print(Areas_ROC_logistic)
print(Areas_ROC_randomforest)

0.5466417910447762
0.8660862354892205
0.8401741293532339
```

Conclusion :-

As you can see most of the curve is covered by Logistic Regression, Also Logistic Regression gives highest Accuracy than the all others ,

Hence we will select Logistic Regression model