What is Regression

Regression analysis is a powerful statistical analysis technique. A **dependent** variable of our interest is used to predict the values of other **independent variables** in a data-set.

It uses many techniques to analyse and predict the outcome, but the emphasis is mainly on **relationship between dependent variable and one or more independent variable.**

Logistic Regression In Python

It is a technique to analyse a data-set which has a dependent variable and one or more independent variables to predict the outcome in a binary variable, meaning it will have only two outcomes.

The dependent variable is **categorical** in nature. Dependent variable is also referred as **target variable** and the independent variables are called the **predictors**.

Logistic regression is a special case of linear regression where we only predict the outcome in a categorical variable. It predicts the probability of the event using the log function.

We use the **Sigmoid function/curve** to predict the categorical value. The threshold value decides the outcome(win/lose).

Linear regression equation: $\mathbf{y = \beta 0 + \beta 1X1 + \beta 2X2 \ldots. + \beta nXn}$

- Y stands for the dependent variable that needs to be predicted.
- β0 is the Y-intercept, which is basically the point on the line which touches the y-axis.
- β1 is the slope of the line (the slope can be negative or positive depending on the relationship between the dependent variable and the independent variable.)
- X here represents the independent variable that is used to predict our resultant dependent value.

Sigmoid function: $\boldsymbol{p = 1 / 1 + e^{-y}}$

Apply sigmoid function on the linear regression equation.

Logistic Regression equation: $\boldsymbol{p = 1 / 1 + e^{-(\beta 0 + \beta 1X1 + \beta 2X2 \ldots. + \beta nXn)}}$

Lets take a look at different types of logistic regression.

Types Of Logistic Regression

- Binary logistic regression – It has only two possible outcomes. Example- yes or no
- Multinomial logistic regression – It has three or more nominal categories.Example- cat, dog, elephant.
- Ordinal logistic regression- It has three or more ordinal categories, ordinal meaning that the categories will be in a order. Example- user ratings (1-5).

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.

Demo

We are going to build a prediction model using logical regression in Python with the help of a dataset, in this we are going to cover the following steps to achieve logical regression.

1. Collecting Data
2. Analyzing Data
3. Data Wrangling
4. Split the data into Train and Test
5. Accuracy Report

1. **Collecting Data**: The first step is to load the data (Bank_loan) csv file into the programs using the pandas and some other library

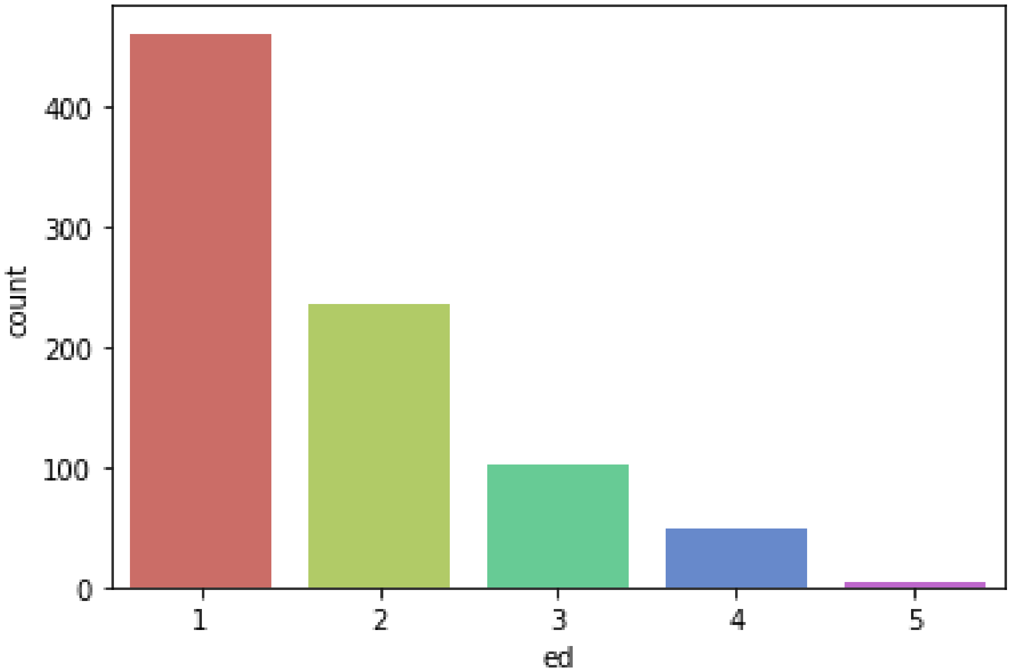
```
import pandas as pd      ##Data manipulation and data analysis

import numpy as np      ##Support for large multi-dimensional arrays and matrix

import seaborn as sb     ## Statistical plotting of data like styles,color

import matplotlib.pyplot as plt      ## For plotting


##sklearn-all data-mining concepts which are interoperate with python

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split   ##train and test split

from sklearn import metrics   ## accuracy calculation

from sklearn.metrics import classification_report   ##for classification of precision and recall matrix

from sklearn.metrics import accuracy_score  ##For the calculation of accuracy




##Loading the data

Bank_loan = pd.read_csv("D:\\EDWISOR\\Project Loan default\\Bank_loan.csv")

Print(Bank_loan.head(5))
```

	age	ed	employ	address	income	debtinc	creddebt	othdebt	default
0	41	3	17	12	176	9.3	11.359392	5.008608	1.0
1	27	1	10	6	31	17.3	1.362202	4.000798	0.0
2	40	1	15	14	55	5.5	0.856075	2.168925	0.0
3	41	1	15	14	120	2.9	2.658720	0.821280	0.0
4	24	2	2	0	28	17.3	1.787436	3.056564	1.0

##Analyzing Data

##Getting the barplot for the categorical columns

sb.countplot(x="ed",data=Bank_loan,palette="hls")



Bank_loan.ed.value_counts() ##For numerical count

Out[20]:

1 460

2 235

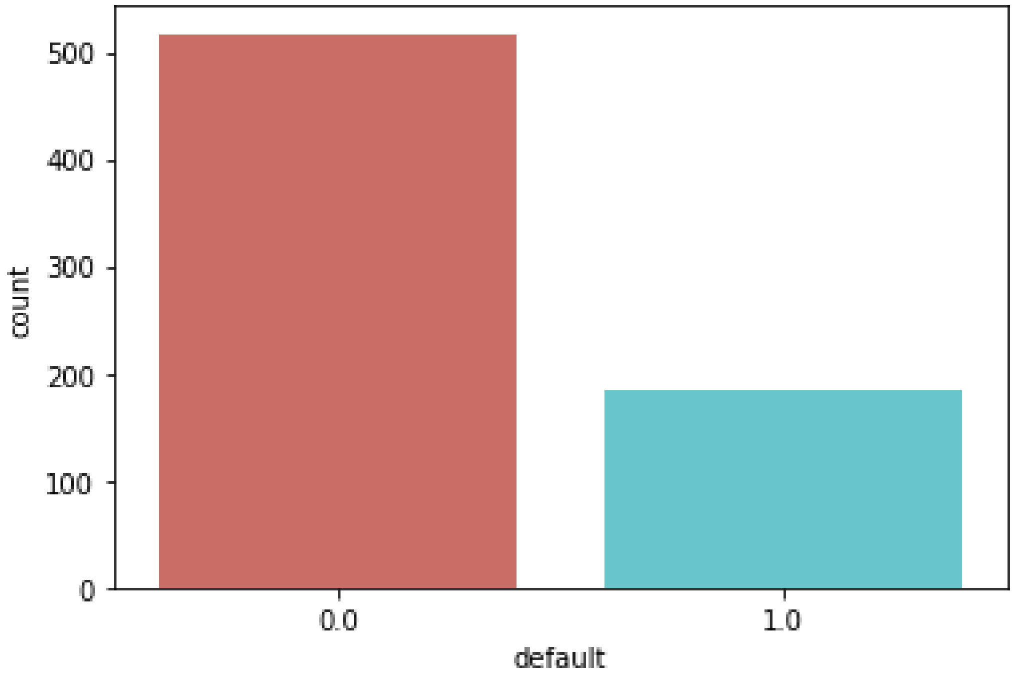
3 101

4 49

5 5

Name: ed, dtype: int64

sb.countplot(x="default",data=Bank_loan,palette="hls")



Bank_loan.default.value_counts()   ##For numerical count

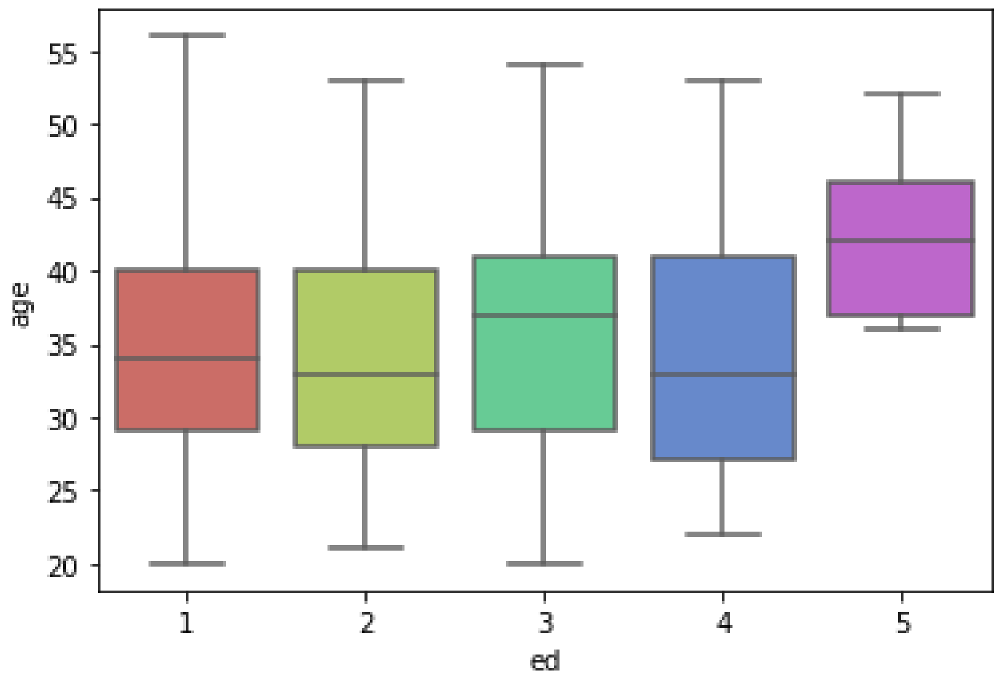
0.0   517

1.0   183

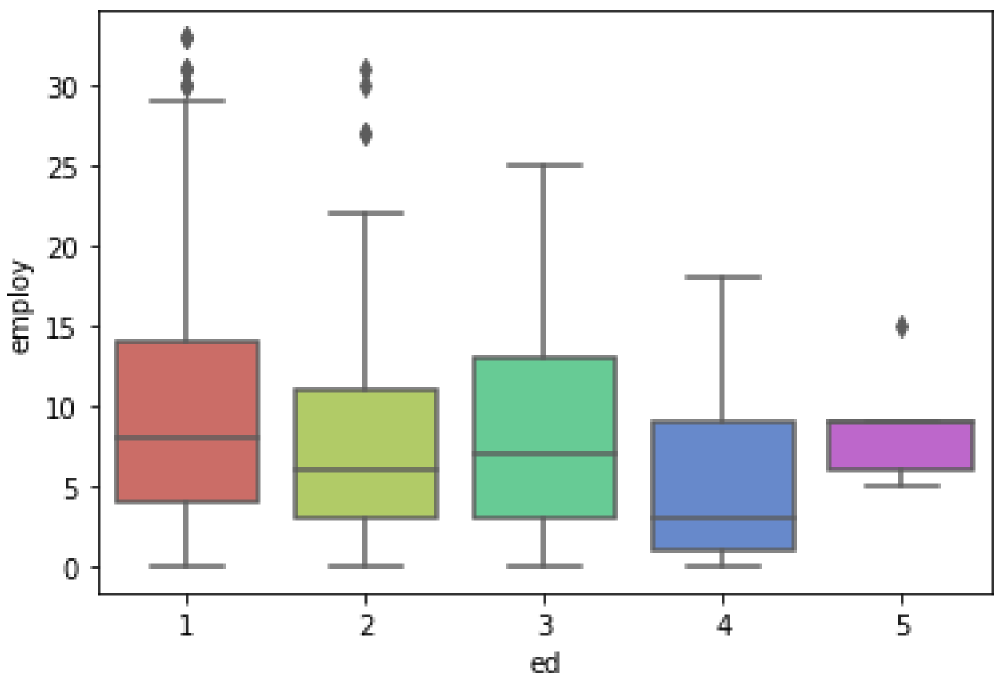
Name: default, dtype: int64

Data Distribution - Boxplot of continuous variables wrt to each category of categorical columns

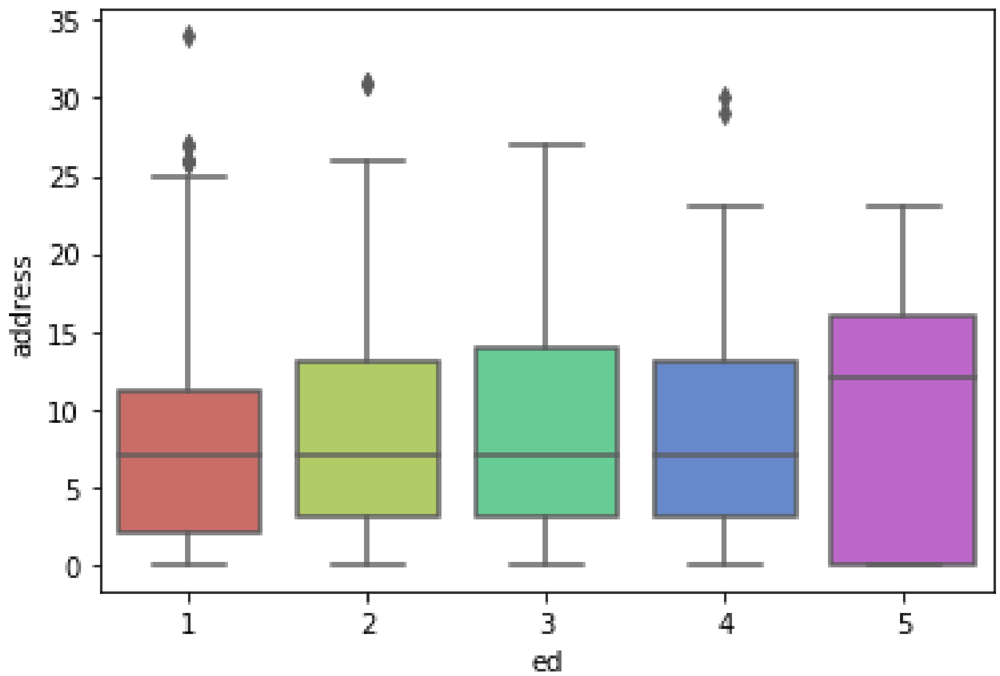
x= ed



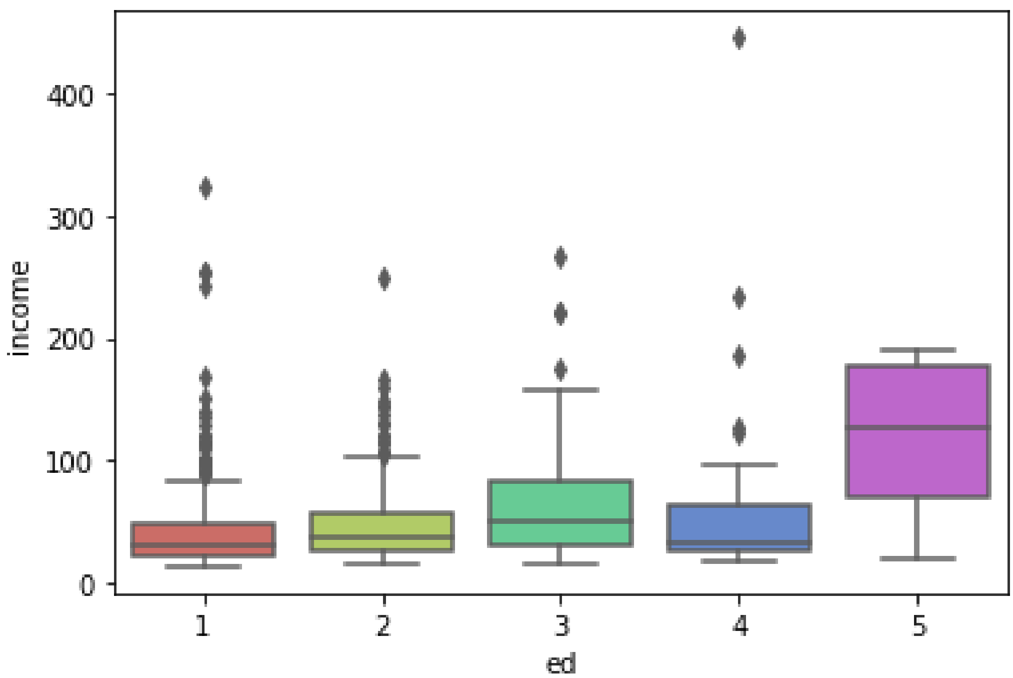
sb.boxplot(x="ed",y="employ",data=Bank_loan,palette="hls")



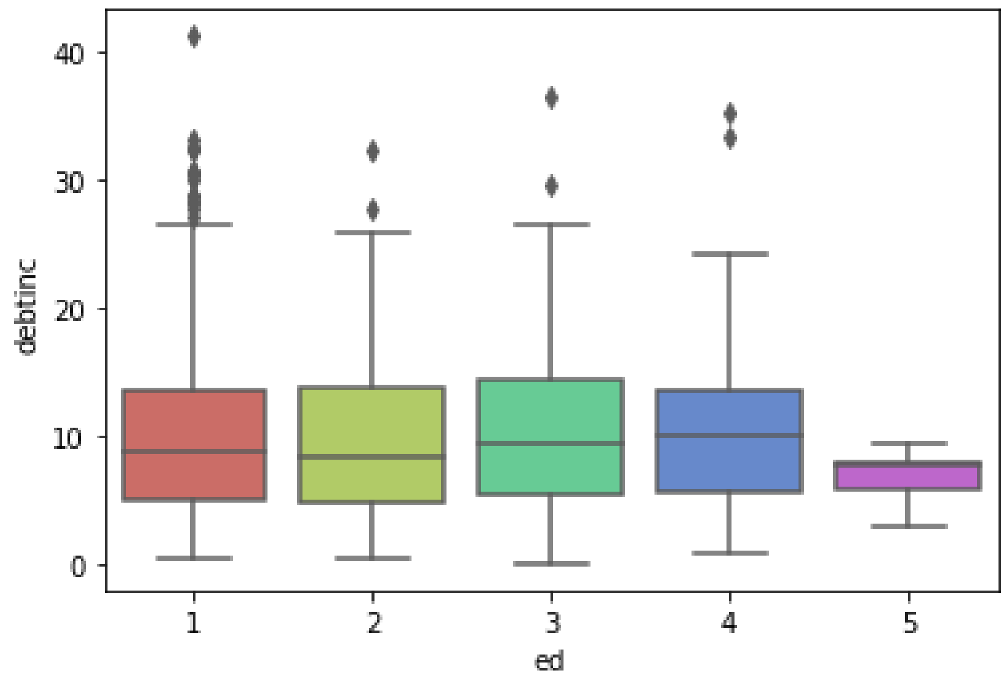
sb.boxplot(x="ed",y="address",data=Bank_loan,palette="hls")



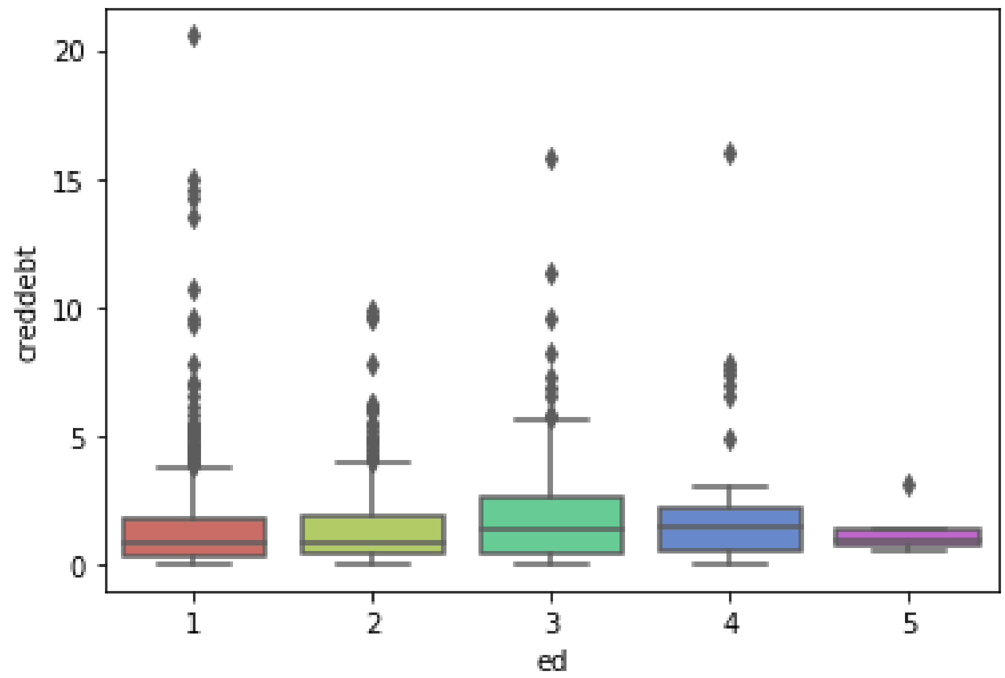

```
sb.boxplot(x="ed",y="income",data=Bank_loan,palette="hls")
```



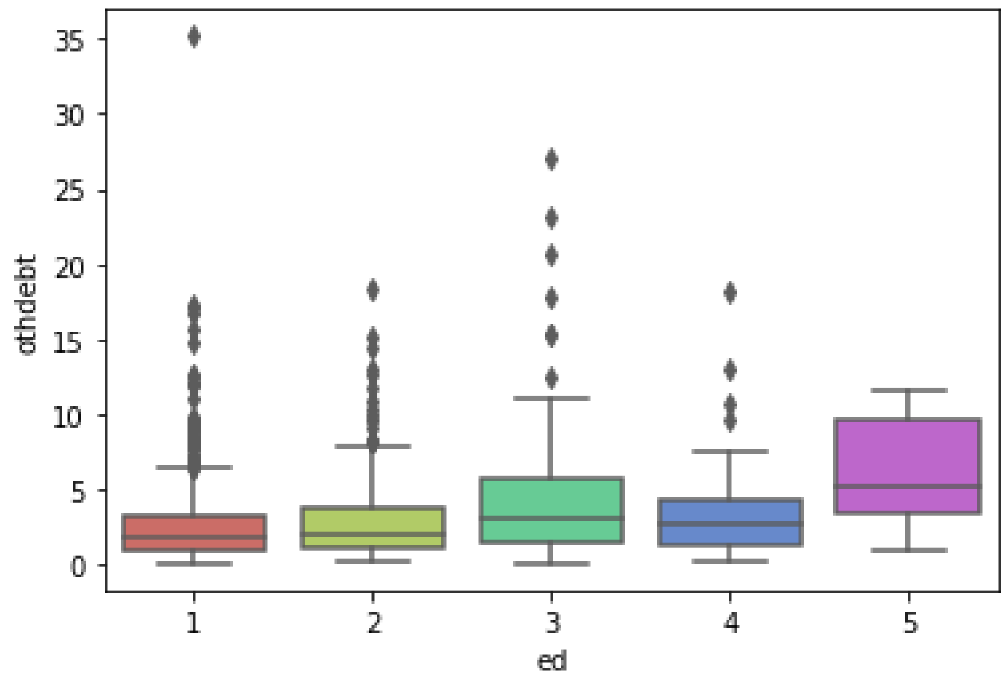

```
sb.boxplot(x="ed",y="debtinc",data=Bank_loan,palette="hls")
```



sb.boxplot(x="ed",y="creddebt",data=Bank_loan,palette="hls")

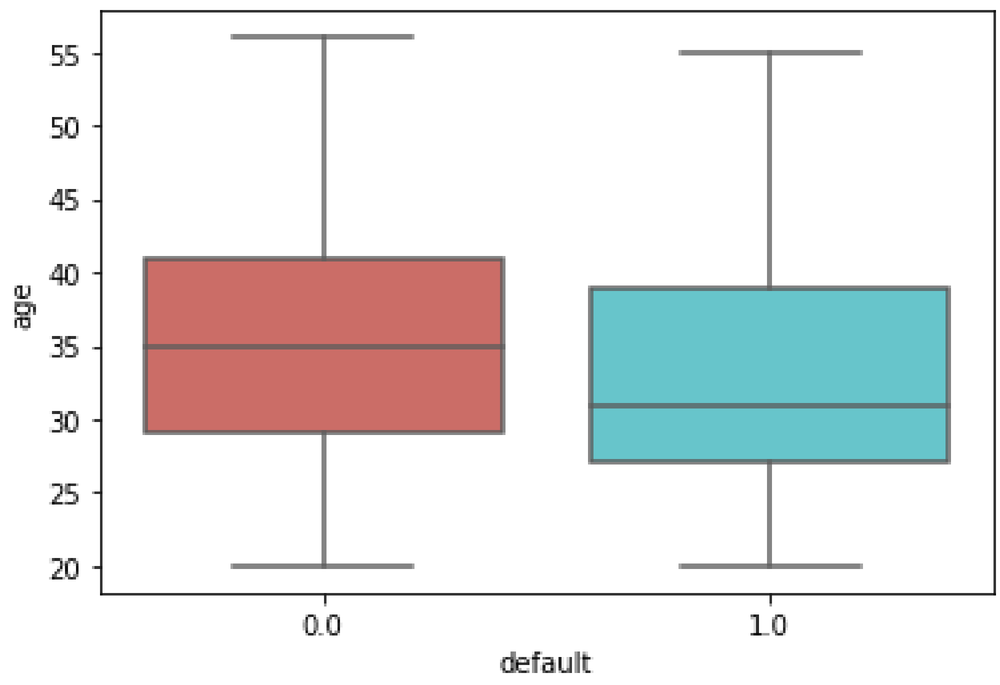


sb.boxplot(x="ed",y="othdebt",data=Bank_loan,palette="hls")

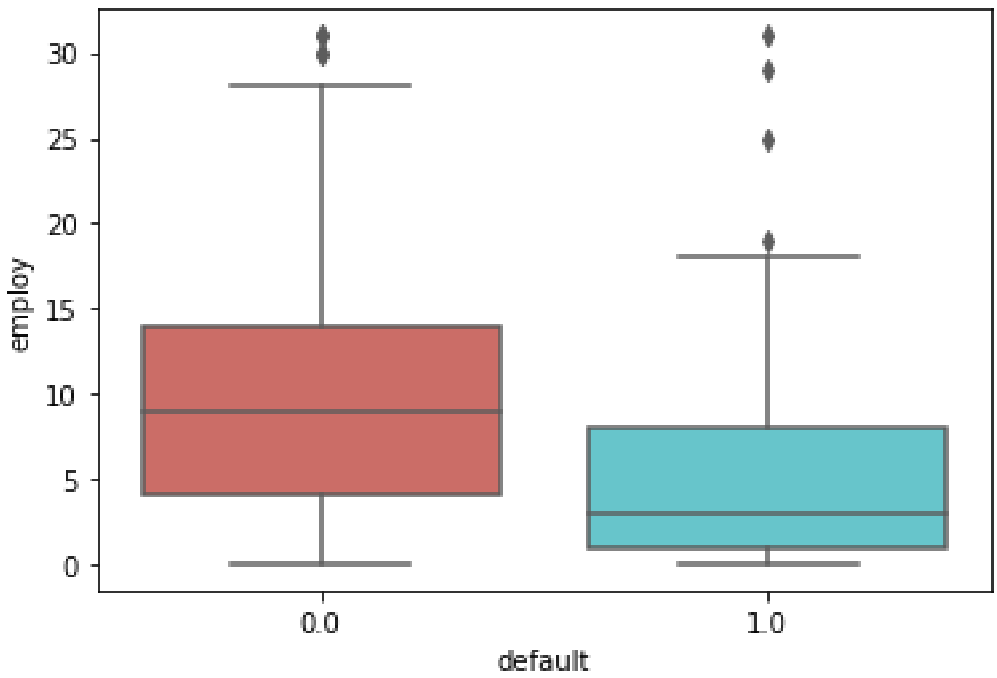


x= default

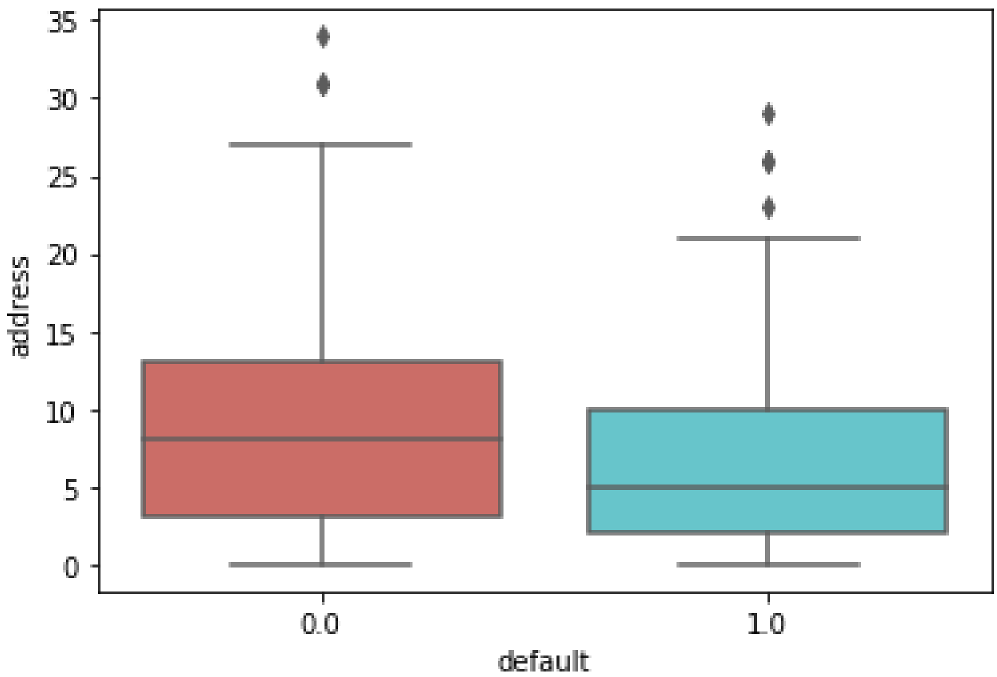
sb.boxplot(x="default",y="age",data=Bank_loan,palette="hls")



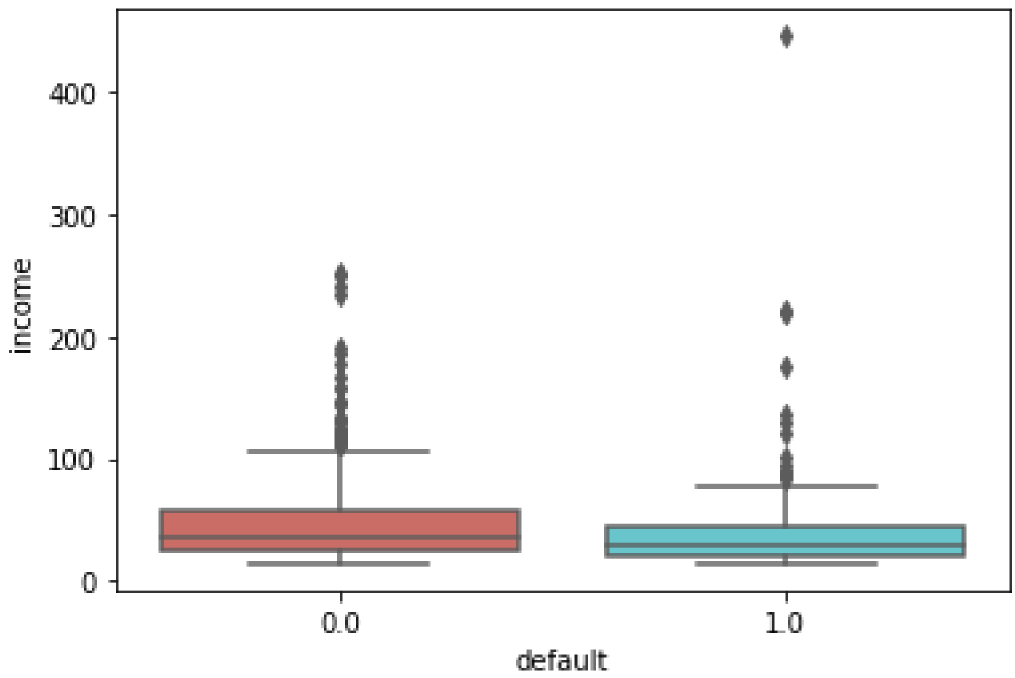
sb.boxplot(x="default",y="employ",data=Bank_loan,palette="hls")



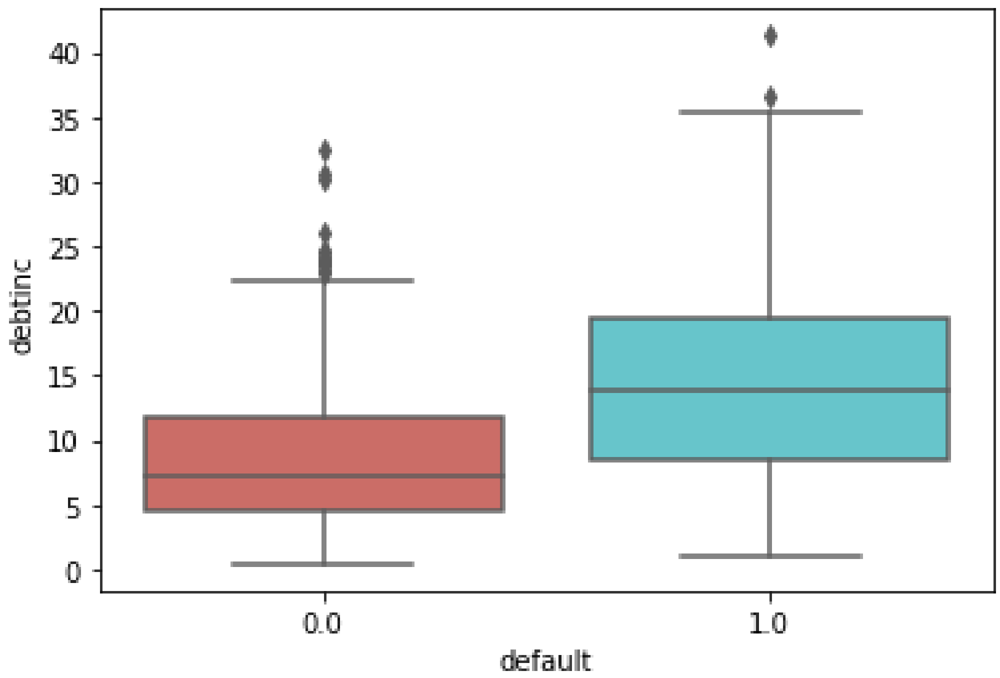
sb.boxplot(x="default",y="address",data=Bank_loan,palette="hls")



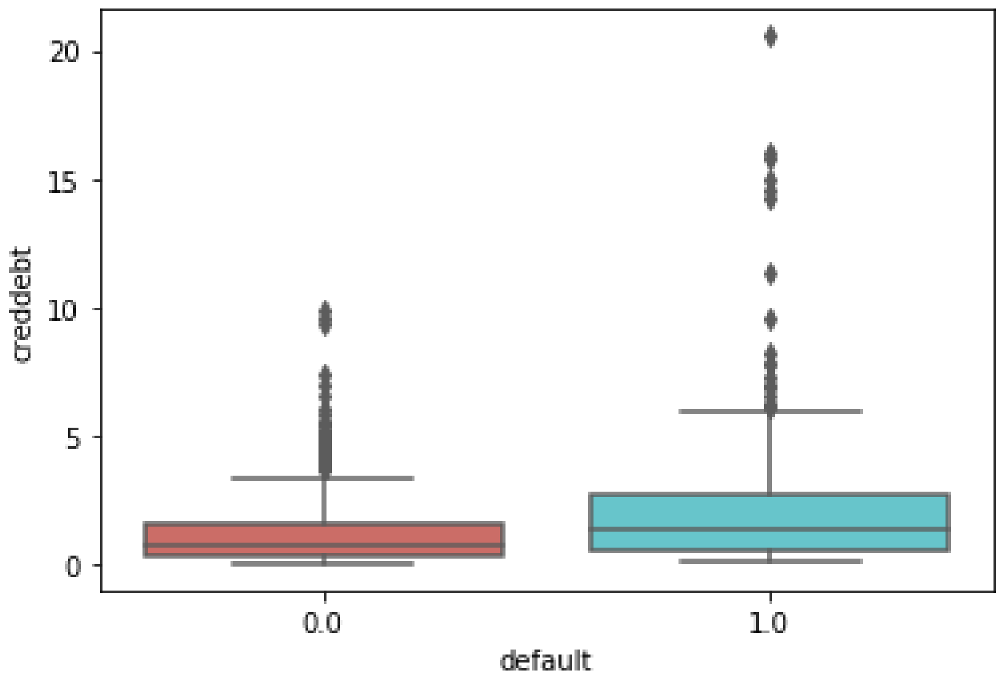
sb.boxplot(x="default",y="income",data=Bank_loan,palette="hls")



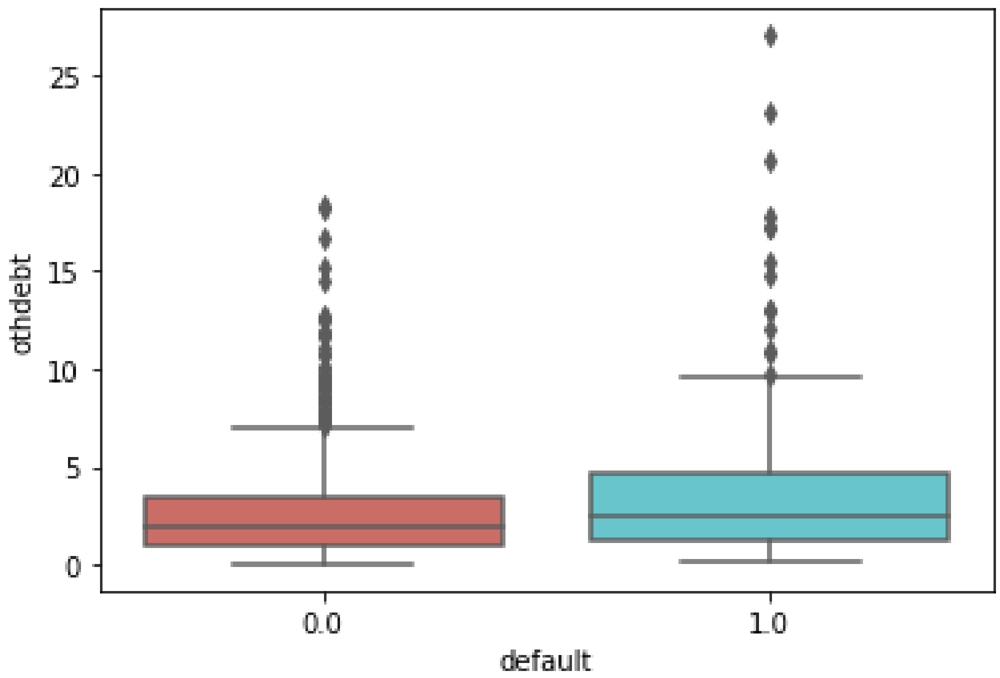

```
sb.boxplot(x="default",y="debtinc",data=Bank_loan,palette="hls")
```




```
sb.boxplot(x="default",y="creddebt",data=Bank_loan,palette="hls")
```



sb.boxplot(x="default",y="othdebt",data=Bank_loan,palette="hls")



##convert the types of attributes

Bank_loan['default'] = pd.Categorical(Bank_loan.default)

print (Bank_loan.dtypes)

Bank_loan["ed"] = pd.Categorical(Bank_loan.ed)

print(Bank_loan.dtypes)

Data Wrangling

Bank_loan.isnull().sum()

Out[41]:

age 0

ed 0

employ 0

address 0

income 0

debtinc 0

creddebt 0

othdebt 0

default 150

dtype: int64

##Fill nan values with mode of categorical coloumn

##Mode value imputation

Bank_loan.default.mode()

Out[42]:

0 0.0

dtype: float64

Bank_loan["default"].fillna(0,inplace=True) #mode of default variable is 0

##Check again the na value

Bank_loan.isnull().sum()

Out[44]:

age 0

ed 0

employ 0

address 0

income 0

debtinc 0

creddebt 0

othdebt   0

default   0

dtype: int64

##Model Building (Define X and Y) & Spliting the data

X = Bank_loan.iloc[:,[0,1,2,3,4,5,6,7]] ##Here we are defining the input variable to X

Y = Bank_loan.iloc[:,8]   ## Here we are defining output variable to Y

##Split the data

X_train,X_test,Y_train,Y_test = train_test_split(X, Y, test_size = 0.3, random_state=1)

(Here we are splitting the data into train and test, X_train contain 70% of the whole data and Y_test contain 30% of the data)

We are building the logistic regression model and storing the model named as classifier

classifier = LogisticRegression()

we are training our model

classifier.fit(X,Y)

Out[49]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)  ##Now our model is ready
```

print (classifier.intercept_,classifier.coef_) # coeficient of features

print (classifier.intercept_,classifier.coef_) # coeficient of features

```
[-1.52300796] [[ 0.01263364  0.05077209 -0.2020838  -0.0589679  -0.0021791   0.08040288
   0.38016981  0.00826118]]
```

prob = classifier.predict_proba (X_test) ##Probality values

##Accuracy on train data

predict_train = classifier.predict(X_train)

print('Target on train data',predict_train) ## We get output in the form of 0 and 1

accuracy_train = accuracy_score(Y_train,predict_train) print('accuracy_score on train dataset : ', accuracy_train)

Out: print('accuracy_score on train dataset : ', accuracy_train)

accuracy_score on train dataset : 0.8084033613445378

##Accuracy on train data by classifier function

##Accuracy by classifier on train data

predictions = classifier.predict(X_train)

classification_report(Y_train,predictions)

n [182]: predictions = classifier.predict(X_train)

classification_report(Y_train,predictions)

Out[183]: '   precision   recall f1-score   support\n\n   0.0   0.83   0.95   0.89   459\n   1.0   0.66   0.36   0.47   136\n\n   accuracy   0.81   595\n   macro avg   0.75   0.65   0.68   595\nweighted avg   0.79   0.81   0.79   595\n'

Precision =1

##Accuracy through confusion matrix

from sklearn.metrics import confusion_matrix

confusion_matrix = confusion_matrix(Y_train,predict_train)

print(confusion_matrix)

[[435 24]

 [90 46]]

##Accuracy on test data

predict_test = classifier.predict(X_test)

print('Target on test data',predict_test)

accuracy_test =accuracy_score(Y_test,predict_test)

print('accuracy_score on test dataset : ', accuracy_test)

Out: print('accuracy_score on test dataset : ', accuracy_test)

accuracy_score on test dataset : 0.8470588235294118

##Accuracy on test data through classifier function

predictions1 = classifier.predict(X_test) classification_report(Y_test,predictions1)

Out[198]: '   precision   recall f1-score   support\n\n   0.0   0.87   0.96   0.91   208\n   1.0   0.67   0.34   0.45   47\n\n   accuracy   0.85   255\n   macro avg   0.77   0.65   0.68   255\nweighted avg   0.83   0.85   0.83   255\n'

Precision1=1

##Accuracy through confusion matrix

from sklearn.metrics import confusion_matrix

confusion_matrix = confusion_matrix(Y_test,predict_test)

print(confusion_matrix)

[[200   8]

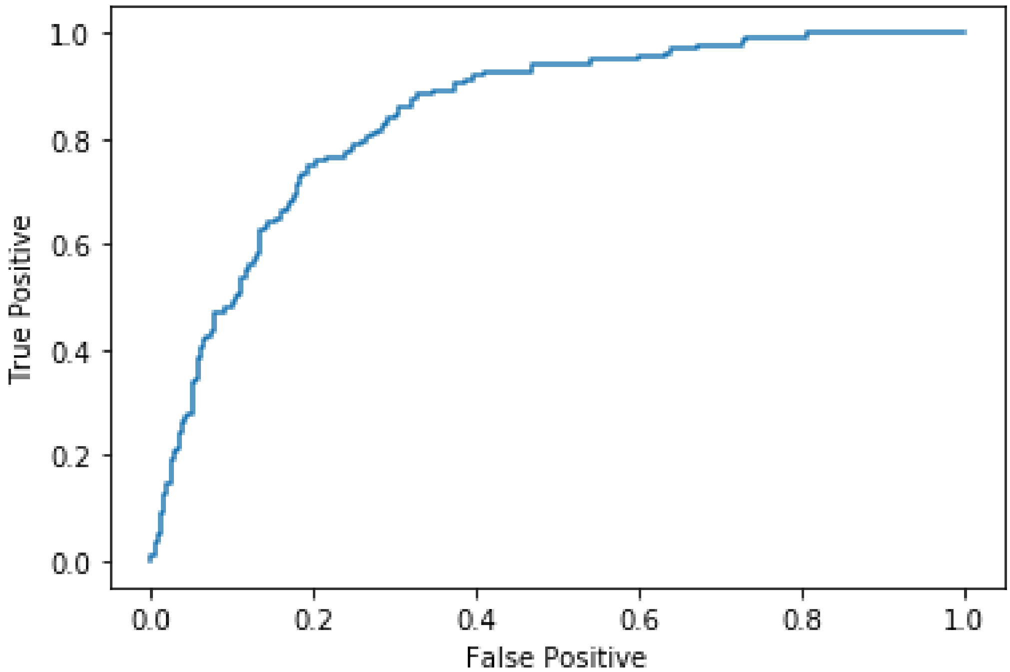
 [31 16]]

##Let check the Roc which is used for threshold value

y_prob = pd.DataFrame(classifier.predict_proba(X_train.iloc[:,:]))

fig,ax = plt.subplots()

plt.plot(fpr,tpr);plt.xlabel("False Positive");plt.ylabel("True Positive");



##ROC curve

fpr,tpr,thresholds = metrics.roc_curve(Y_train,y_prob.iloc[:,1:])

roc_auc = metrics.auc(fpr, tpr)

roc_auc (Area Under curve)

0.8429770601050878

We should keep the more value of area under the curve as much as possible.

So our train model accuracy is 80.84 but test model accuracy is 84.70. This means our model is underfit.

Let"s try another algorithm

Decision Tree(So I done all the EDA and data pre processing so I directly build the model)

DECISION TREES

import pandas as pd

import matplotlib.pyplot as plt

Bank_loan = pd.read_csv("D:\\EDWISOR\\Project Loan default\\Bank_loan.csv")

Bank_loan.head() #It shows the top 5 observation

Out[30]:

	age	ed	employ	address	income	debtinc	creddebt	othdebt	default
0	41	3	17	12	176	9.3	11.359392	5.008608	1.0
1	27	1	10	6	31	17.3	1.362202	4.000798	0.0
2	40	1	15	14	55	5.5	0.856075	2.168925	0.0
3	41	1	15	14	120	2.9	2.658720	0.821280	0.0
4	24	2	2	0	28	17.3	1.787436	3.056564	1.0

Bank_loan["default"].unique()   ##It show the number of cateogry in default variable

Out[31]: array([1., 0., nan])

Bank_loan.default.value_counts() ## (It shows the nuber of zero and number of 1 present in default variable)

Out[32]:

0.0   517

1.0   183

Name: default, dtype: int64

##count the na value

```
Bank_loan.isnull().sum()
##Fill nan values with mode of categorical coloumn
##Mode value imputation
Bank_loan.default.mode()
Bank_loan["default"].fillna(0,inplace=True) #mode of default variable is 0

##Check again the na value
Bank_loan.isnull().sum()
colnames = list(Bank_loan.columns)##It make list of all the variables in Bank_loan
predictors = colnames[:8] ##It makes the of all the attributes or input variables
target = colnames[8] ##It separate the target variable

##Splitting the data into train and test dataset
import numpy as np
from sklearn.model_selection import train_test_split
train,test = train_test_split(Bank_loan,test_size = 0.3,random_state = 1)

train.default.value_counts()
Out[43]:
0.0    459
1.0    136
Name: default, dtype: int64

test.default.value_counts()
0.0    208
1.0     47
Name: default, dtype: int64
```

##Model Building

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion = "entropy")
model.fit(train[predictors],train[target])
```

Out[47]:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, 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, presort=False,
            random_state=None, splitter='best')
```

##Accuracy for train

np.mean(pd.Series(train.default).reset_index(drop=True)== pd.Series(model.predict(train[predictors])))

Out[110]: 1.0

Accuracy is 100%

##Accuracy for test

np.mean(pd.Series(test.default).reset_index(drop=True)== pd.Series(model.predict(test[predictors])))

Out[111]: 0.7254901960784313

Accuracy is 72.54

Conclusion: Here accuracy of train data is 100% but accuracy of test data is 72.54% that mean our model is overfit.

So we go for another algorithm

RANDOM FOREST

import pandas as pd ##Data manipulation and data analysis

import numpy as np ##Support for large multi-dimensional arrays and matrix

import seaborn as sb ## Statistical plotting of data like styles,color

import matplotlib.pyplot as plt ## For plotting

##sklearn-all data-mining concepts which are interoperate with python from sklearn.model_selection import train_test_split ##train and test split

from sklearn import metrics ## accuracy calculation

from sklearn.metrics import classification_report

##Loading the data

Bank_loan = pd.read_csv("D:\\EDWISOR\\Project Loan default\\Bank_loan.csv")

##Fill nan values with mode of categorical coloumn

##Mode value imputation

Bank_loan.default.mode()

Bank_loan["default"].fillna(0,inplace=True) #mode of default variable is 0

##Check again the na value

Bank_loan.isnull().sum()

Bank_loan.head()

Out[165]:

	age	ed	employ	address	income	debtinc	creddebt	othdebt	default
0	41	3	17	12	176	9.3	11.359392	5.008608	1.0
1	27	1	10	6	31	17.3	1.362202	4.000798	0.0
2	40	1	15	14	55	5.5	0.856075	2.168925	0.0
3	41	1	15	14	120	2.9	2.658720	0.821280	0.0
4	24	2	2	0	28	17.3	1.787436	3.056564	1.0

Bank_loan["default"].unique()

Out[166]: array([1., 0.])

Bank_loan.default.value_counts()

Out[167]:

0.0 667

1.0 183

```
colnames = list(Bank_loan.columns)

##Splitting the data into train and test dataset

train,test = train_test_split(Bank_loan,test_size = 0.3,random_state = 1)



train.default.value_counts()

0.0    459

1.0    136

test.default.value_counts()

0.0    208

1.0     47


colnames = train.columns

len(colnames[0:8])

trainX = train[colnames[0:8]]

trainY = train[colnames[8]]


from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_jobs=2,oob_score=True,n_estimators=15,criterion="entropy")

rf.fit(trainX,trainY)

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            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=15, n_jobs=2,
            oob_score=True, random_state=None, verbose=0,
            warm_start=False)
```

##Accuracy of training data by classifier method

predictions = rf.predict(trainX)

classification_report(trainY,predictions)

Out[180]: '   precision   recall f1-score   support\n\n   0.0   0.99   1.00   1.00   459\n   1.0   1.00   0.97   0.99   136\n\n   accuracy   0.99   595\n   macro avg   1.00   0.99   0.99   595\nweighted avg   0.99   0.99   0.99   595\n'

##Precision =100%

 # Check the accuracy by confusion matrix

trainX["rf_pred"] = rf.predict(trainX)

from sklearn.metrics import confusion_matrix

confusion_matrix(trainY,trainX["rf_pred"]) # Confusion matrix

array([[459,   0],

       [4, 132]]

print ("Accuracy",(459+132)/(459+133+0+3)) ## 99.32

 # Accuracy on testing data

testX = test[colnames[0:8]]

testY = test[colnames[8]]

rf.fit(testX,testY)

RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',

            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=15, n_jobs=2,

            oob_score=True, random_state=None, verbose=0,

            warm_start=False)

##Accuracy of test data by classifier method

predictions1= rf.predict(testX)

classification_report(testY,predictions1)

Out[189]: '   precision   recall f1-score   support\n\n   0.0   1.00   1.00   1.00   208\n   1.0   1.00   0.98   0.99   47\n\n   accuracy   1.00   255\n   macro avg   1.00   0.99   0.99   255\nweighted avg   1.00   1.00   1.00   255\n'

Here also precision is 100%

##Check the accuracy of test data by confusion matrix

testX["rf_pred"] = rf.predict(testX)

confusion_matrix(testY,testX["rf_pred"])

array([[208,   0],

       [1, 46]])

print ("Accuracy",(208+45)/(208+45+2+0)) # 99.21

Conclusion: I finalize the above algorithm and model (Random Forest) for this data because through this model I get maximum accuracy and maximum precision

And our accuracy & Precision of Train data is also matching with Accuracy & Precision of Test data

Now we will perform the same activity in R

Collecting Data:

Bank_loan <- read.csv(file.choose())

View(Bank_loan)

str(Bank_loan)

```
str(Bank_loan)
'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 : int  1 0 0 0 1 0 0 0 1 0 ...
```

In our business problem ed and default variable are factor variable

##Conversion the variables into required type

Bank_loan$age = as.numeric(Bank_loan$age)

Bank_loan$ed = as.factor(Bank_loan$ed)

Bank_loan$employ = as.numeric(Bank_loan$employ)

Bank_loan$address = as.numeric(Bank_loan$address)

Bank_loan$income = as.numeric(Bank_loan$income)

Bank_loan$default = as.factor(Bank_loan$default)

##Again check the variable types

##Check the conversion

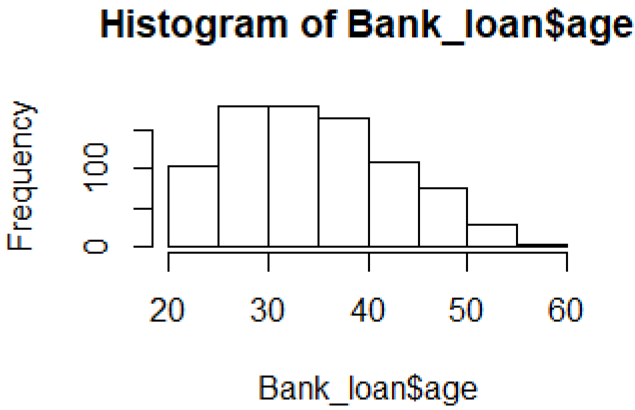
str(Bank_loan)

```
str(Bank_loan)
'data.frame':	850 obs. of  9 variables:
 $ age     : num  41 27 40 41 24 41 39 43 24 36 ...
 $ ed      : Factor w/ 5 levels "1","2","3","4",..: 3 1 1 1 2 2 1 1 1 1
...
 $ employ  : num  17 10 15 15 2 5 20 12 3 0 ...
 $ address : num  12 6 14 14 0 5 9 11 4 13 ...
 $ income  : num  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 ...
```

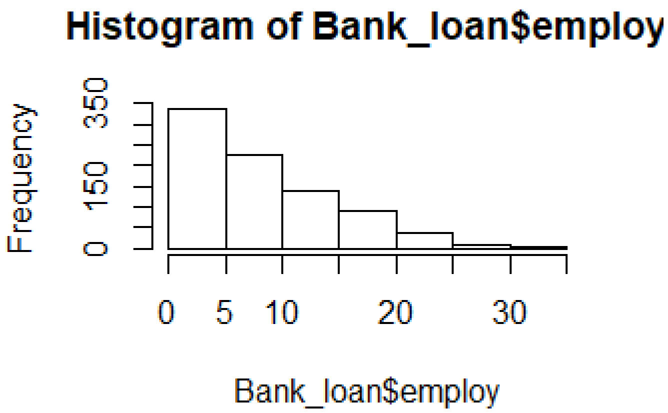
- **Analysing Data**

##Distribution of data

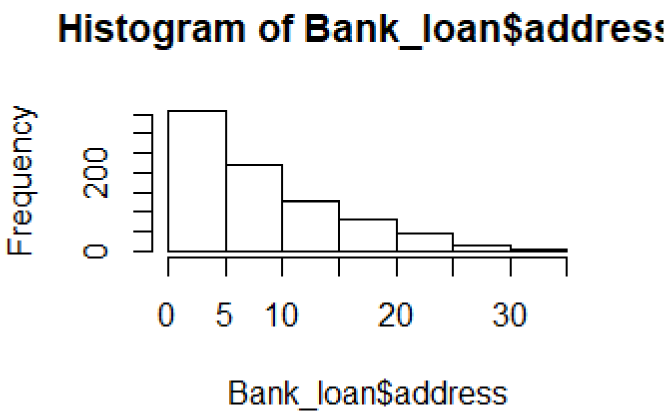
hist(Bank_loan$age)



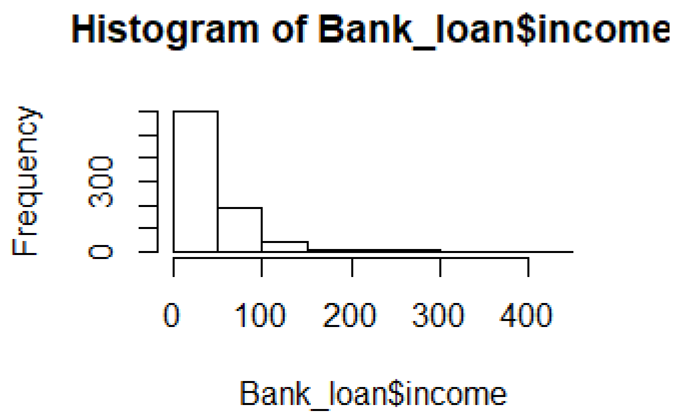
hist(Bank_loan$employ)



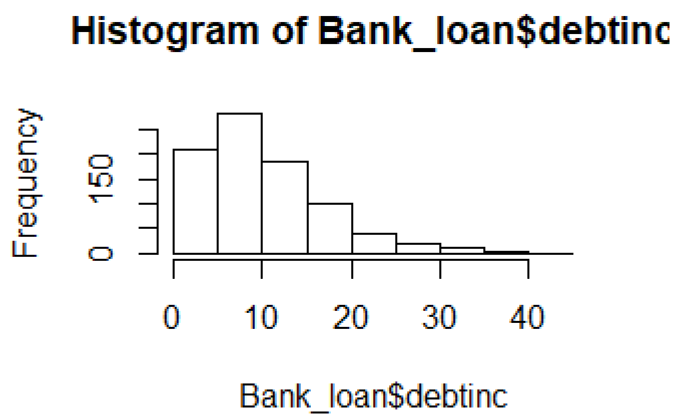
hist(Bank_loan$address)



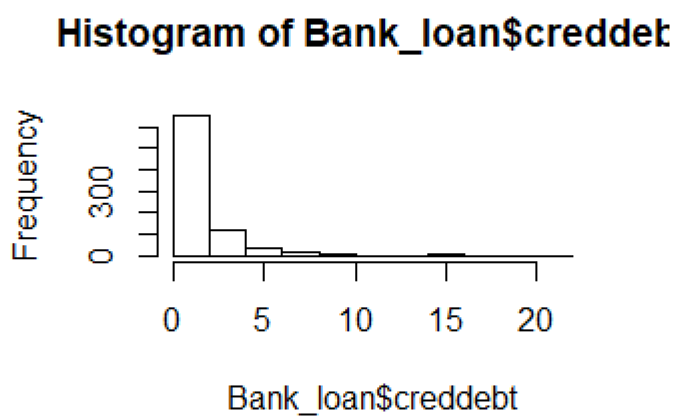
hist(Bank_loan$income)



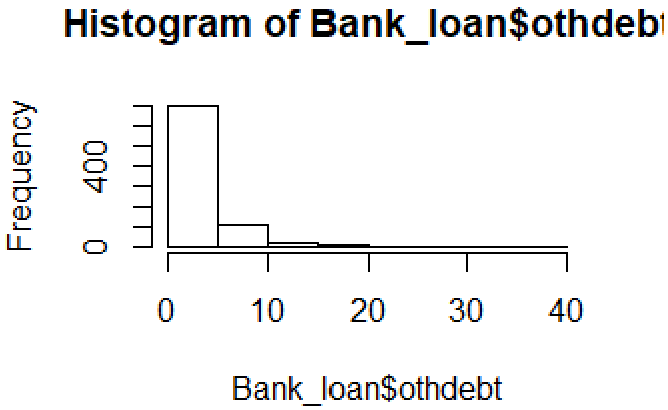
hist(Bank_loan$debtinc)



hist(Bank_loan$creddebt)



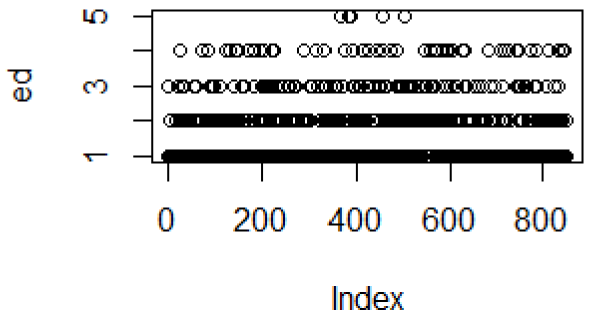
hist(Bank_loan$othdebt)



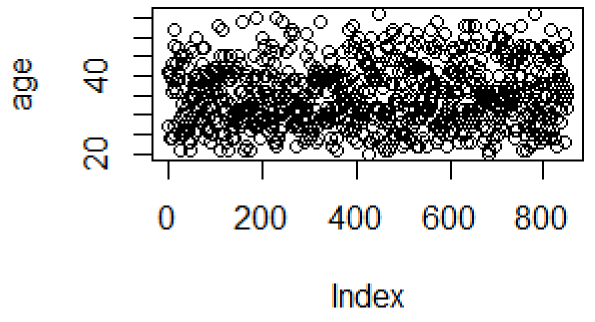
#plot relation with each X and Y

attach(Bank_loan)

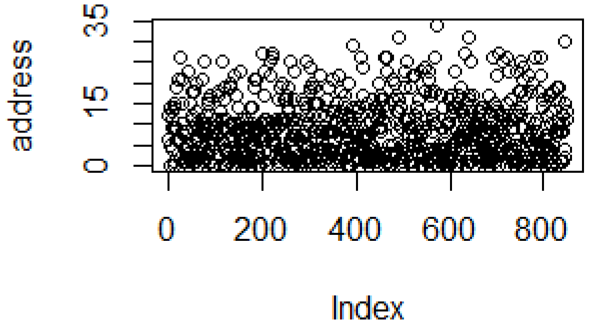
plot.default(ed)



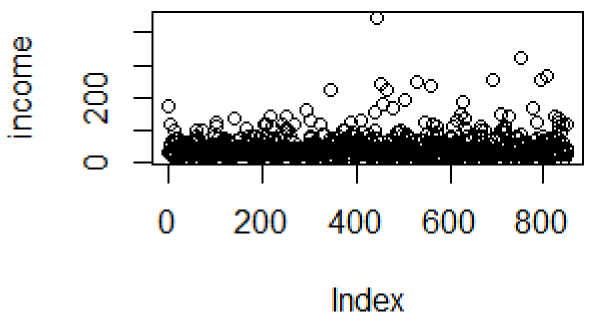
plot.default(age)



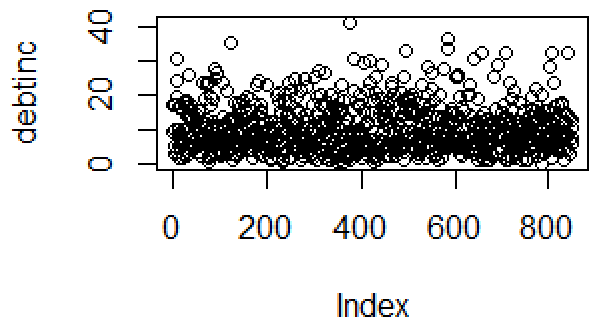
plot.default(address)



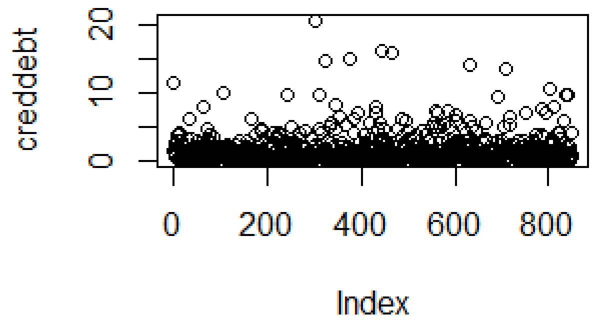
plot.default(income)



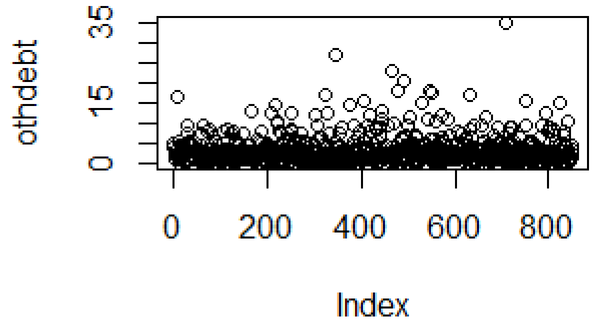
plot.default(debtinc)



plot.default(creddebt)

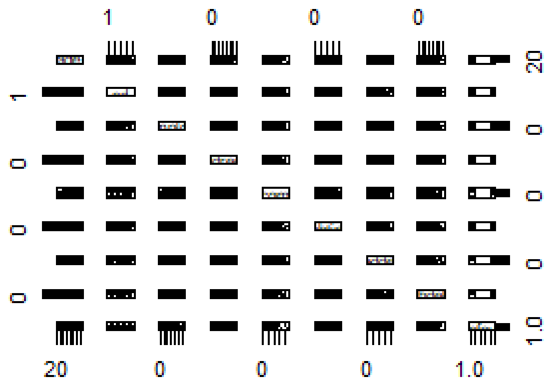


plot.default(othdebt)



Scatter plot of all plots with all variables

pairs(Bank_loan)



#we can also see correlation coefficient and scatter plot together

#install.packages("GGally")

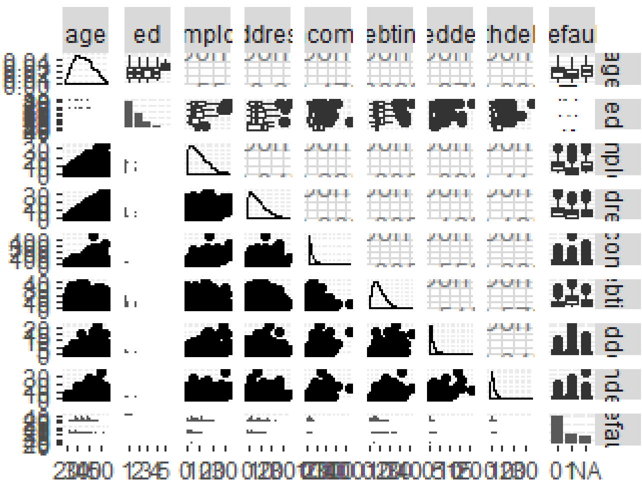
#install.packages("stringi")

library(GGally)

library(stringi)

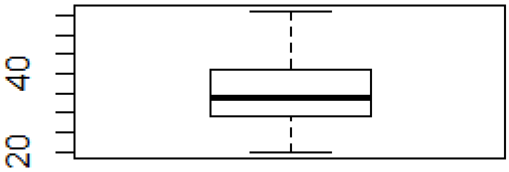
windows()

ggpairs(Bank_loan)

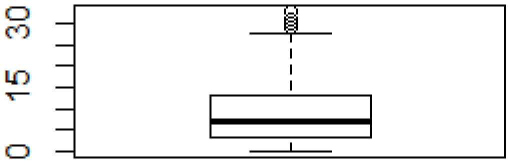


##Check for the outliers

boxplot(Bank_loan$age)



boxplot(Bank_loan$employ)

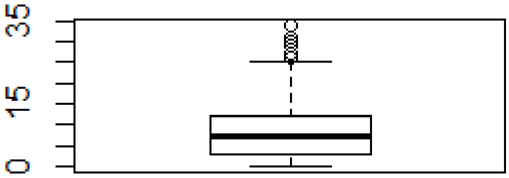


boxplot(Bank_loan$employ)$out

```
boxplot(Bank_loan$employ)$out
 [1] 29 31 30 31 30 31 30 33 33 29
```

This imply that at this point outliers are lieing.

boxplot(Bank_loan$address)

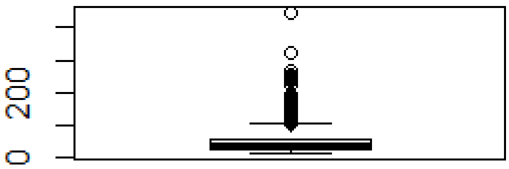


boxplot(address)$out

```
boxplot(address)$out
 [1] 26 27 27 26 29 26 26 26 31 26 34 27 31 26 27 26 26 26 30
```

These are the points where outliers are present

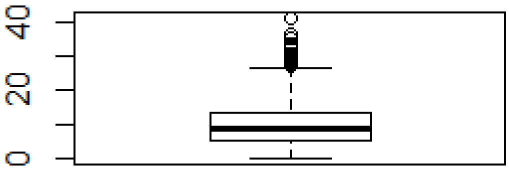
boxplot(Bank_loan$income)



boxplot(Bank_loan$income)$out

```
boxplot(Bank_loan$income)$out
 [1] 176 120 113 121 135 116 116 145 113 118 144 105 120 159 129 120 220
126 132 157 446 242 177 221 166 190 249 123
[29] 234 115 114 113 129 148 186 136 113 253 150 107 108 139 324 169 126
254 266 140 126 138 110 116 116
```

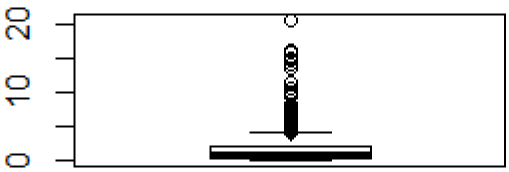
boxplot(Bank_loan$debtinc)



boxplot(Bank_loan$debtinc)$out

```
boxplot(Bank_loan$debtinc)$out
 [1] 30.6 27.7 35.3 27.1 41.3 30.8 29.7 30.1 28.9 33.3 28.5 27.7 36.6 33.4
30.7 32.5 28.9 32.5 28.2 32.3 32.4
```

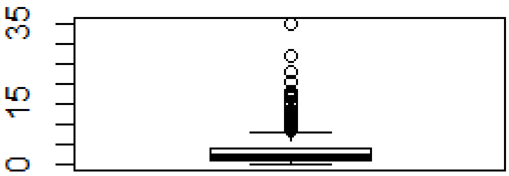
boxplot(Bank_loan$creddebt)



boxplot(Bank_loan$creddebt)$out

```
> boxplot(Bank_loan$creddebt)$out
 [1] 11.359392  6.048900  7.758900  4.582400  9.876600  6.226794  4.637360
4.404816  9.600480  4.874716  4.373382
[12] 20.561310  9.593400  4.593402 14.596200  5.001711  5.574294  8.166400
4.521696  6.565583  5.245296 15.016680
[23]  6.113800  6.935916  5.439966  6.948680  7.817144  4.764760  5.402000
16.031470  4.672800 15.791776  4.584030
[34]  6.111369  5.715360  4.513860  4.272840  4.991010  5.549544  7.387380
5.090526  6.911520  4.935645  5.283498
[45]  7.320000  4.960032  5.896743  6.588540  4.637556  5.781564 14.231448
5.060000  4.334400  5.501188  9.308376
[56]  4.880700 13.552500  5.250528  6.506240  7.053480  7.612542  7.001764
10.679340  7.754240  4.212968  4.183900
[67]  5.821200  9.542358  9.702504
```

boxplot(Bank_loan$othdebt)



boxplot(Bank_loan$othdebt)$out

```
boxplot(Bank_loan$othdebt)$out
 [1] 16.668126  9.736768  9.716100  8.399496  8.502006 13.051206 12.421860
14.452730 10.183560 10.753960 12.659328
[12]  8.362380 12.075690  9.498822 17.203800 12.714006 27.033600  9.043830
14.719320  9.286200 15.405390  9.390654
[23] 11.874450  8.631320  9.250856  9.198000 11.042325 12.958530  9.555345
23.104224  9.974640 18.269130 20.615868
[34]  9.704240 11.663340 15.149160 10.811388 18.257382 17.798990 10.630620
11.893518 10.980000  8.600436 17.184552
[45]  9.591294  9.459450 11.723976  8.907624 35.197500  9.008766  9.018324
15.626520  9.727536  9.649458 12.556236
[56]  9.060660  8.386560 15.276100 10.385496
```

Conclusion: From the boxplot we conlude that outliers are present in all the attributes except age but we cannot treat outliers because we don't know that it was misprint or real

And if we delete the outliers then our observation become so small whereas in logistic regression we need large number of dataset. I mean the sample will large then it will good for logistic regression.

Data Wrangling

Check for the missing value

sum(is.na(Bank_loan))

[1] 150

##Check the missing value in which variable

sum(is.na(Bank_loan$age)) ##0

sum(is.na(Bank_loan$ed)) ##0

sum(is.na(Bank_loan$employ)) ##0

sum(is.na(Bank_loan$address)) ## 0

sum(is.na(Bank_loan$income)) ## 0

sum(is.na(Bank_loan$debtinc)) ##0

sum(is.na(Bank_loan$creddebt)) ## 0

sum(is.na(Bank_loan$othdebt)) ## 0

`sum(is.na(Bank_loan$default))` 150

sum(is.na(Bank_loan)) ##150

##We can remove the na value or we can also impute the na value(But we can not simply remove the na value because in logistic regression we need more data so imputing is best option)

##All the missing value lies only in default variable and default variable is factor so we have to impute with mode.

##So for finding the mode of default variable we have to use the table function

table(Bank_loan$default)

```
  0   1
667 183
```

So our mode of default variable is 0

Bank_loan$default[is.na(Bank_loan$default)] = 0

sum(is.na(Bank_loan$default)) ## 0

Before building the model we sholu also know that which feature is important

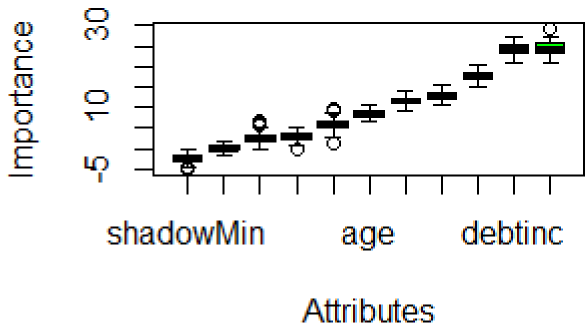
##For feature selection

library(Boruta)

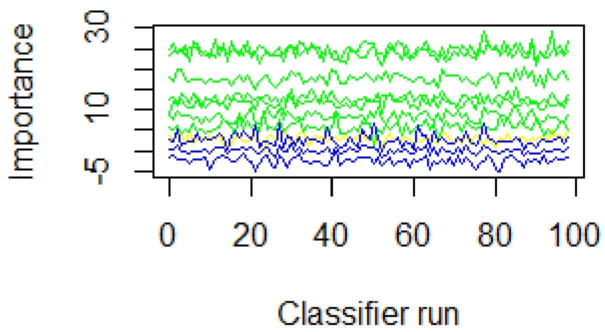
set.seed(111)

boruta = Boruta(default ~., data = Bank_loan, doTrace = 2)

plot(boruta)



plotImpHistory(boruta)



attStats(boruta)

```
> attStats(boruta)
            meanImp medianImp    minImp    maxImp  normHits  decision
age        8.414749  8.424106  6.490880 10.782893 1.0000000 Confirmed
ed         2.934860  2.992830 -0.193490  5.019787 0.5656566 Tentative
employ    24.492803 24.424113 20.755406 29.242785 1.0000000 Confirmed
address    5.784894  5.631352  1.335332  9.411704 0.9494949 Confirmed
income    12.811831 12.802082 10.577694 15.299174 1.0000000 Confirmed
debtinc   24.302863 24.390046 21.032744 27.425612 1.0000000 Confirmed
creddebt  17.798908 17.770780 15.028223 20.328303 1.0000000 Confirmed
othdebt   11.671913 11.801044  8.939753 13.928274 1.0000000 Confirmed
```

Here I can easily see in decision column all the attributes are important exceptd ed because it come up with decision “Confirmed”

And for ed it come up with Tentative(Not sure whether important or not important)

##Split the data into train and test

library(caTools)

split = sample.split(Bank_loan, SplitRatio = 0.70)

split

train_data = subset(Bank_loan, split==TRUE)

test_data = subset(Bank_loan, split==FALSE)

##General logistic model

model1 = glm(default~., family = "binomial", data = train_data)

In this model1 I consider all the attributes and consider default as Y &pass the train data

summary(model1)

```
> summary(model1)

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

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.5980  -0.6580  -0.3288  -0.0688   2.7161  

Coefficients:
              Estimate Std. Error z value Pr(>|z|)    
(Intercept) -1.7209851  0.6652461  -2.587  0.00968 ** 
age          0.0220874  0.0195737   1.128  0.25914    
ed2          0.0487721  0.2779240   0.175  0.86070    
ed3          0.5403713  0.3902153   1.385  0.16611    
ed4         -0.1890054  0.5231547  -0.361  0.71789    
ed5          0.8892333  1.2625193   0.704  0.48123    
employ      -0.2303519  0.0345628  -6.665 2.65e-11 ***
address     -0.0550324  0.0237937  -2.313  0.02073 *  
income      -0.0007742  0.0078242  -0.099  0.92117    
debtinc      0.0839966  0.0308908   2.719  0.00655 ** 
creddebt     0.4649372  0.1082740   4.294 1.75e-05 ***
othdebt     -0.0343237  0.0850336  -0.404  0.68647    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 619.55  on 565  degrees of freedom
Residual deviance: 451.22  on 554  degrees of freedom
AIC: 475.22

Number of Fisher Scoring iterations: 6
```

From the summary I conclude that age, ed, income, othdebt are not significant that mean these are the variable are not contributing in the prediction.

Our Null deviance always greater than Residual Deviance

And AIC value is for the comparison, so the least the AIC the better the model

So again I made the model one by one by removing the attributes which are not significant

And check the summary and compare the model through AIC and other important factor

model2 = glm(default~.-ed, family = "binomial", data = train_data)

summary(model2)

```
summary(model2)

Call:
glm(formula = default ~ . - ed, family = "binomial", data = train_data)

Deviance Residuals: 
     Min        1Q    Median        3Q       Max  
-2.60413  -0.66830  -0.33399  -0.07194   2.68388  

Coefficients:
             Estimate Std. Error z value Pr(>|z|)    
(Intercept) -1.626907   0.627548  -2.592  0.00953 ** 
age          0.021528   0.019249   1.118  0.26340    
employ      -0.231659   0.032973  -7.026 2.13e-12 ***
address     -0.054198   0.023754  -2.282  0.02251 *  
income      -0.000689   0.007930  -0.087  0.93076    
debtinc      0.078724   0.030726   2.562  0.01040 *  
creddebt     0.452947   0.107506   4.213 2.52e-05 ***
othdebt     -0.006768   0.082920  -0.082  0.93495    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 619.55  on 565  degrees of freedom
Residual deviance: 453.94  on 558  degrees of freedom
AIC: 469.94

Number of Fisher Scoring iterations: 6
```

In this model we remove ed variable,so by removing ed variable our AIC value is decreased

model3 = glm(default~.-age,family = "binomial", data = train_data)

summary(model3)

```
> summary(model3)

Call:
glm(formula = default ~ . - age, family = "binomial", data = train_data)

Deviance Residuals: 
     Min        1Q    Median        3Q       Max  
-2.56925  -0.64680  -0.32678  -0.07179   2.67908  

Coefficients:
```

```
                 Estimate Std. Error z value Pr(>|z|)    
(Intercept) -1.1345332  0.4107440  -2.762  0.00574 ** 
ed2          0.0047053  0.2748526   0.017  0.98634    
ed3          0.5122450  0.3894390   1.315  0.18840    
ed4         -0.2128998  0.5209971  -0.409  0.68280    
ed5          0.9412489  1.2627118   0.745  0.45602    
employ      -0.2219627  0.0336932  -6.588 4.47e-11 ***
address     -0.0411584  0.0203867  -2.019  0.04350 *  
income      -0.0003797  0.0079663  -0.048  0.96199    
debtinc      0.0830834  0.0309355   2.686  0.00724 ** 
creddebt     0.4613032  0.1082128   4.263 2.02e-05 ***
othdebt     -0.0258657  0.0848200  -0.305  0.76041    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 619.55  on 565  degrees of freedom
Residual deviance: 452.48  on 555  degrees of freedom
AIC: 474.48

Number of Fisher Scoring iterations: 6
```

model4 = glm(default~.-income,family = "binomial", data = train_data)

summary(model4)

```
summary(model4)

Call:
glm(formula = default ~ . - income, family = "binomial", data = 
train_data)

Deviance Residuals: 
     Min        1Q    Median        3Q       Max  
-2.58795  -0.65742  -0.32892  -0.06924   2.71040  

Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) -1.74329    0.62564  -2.786 0.005329 ** 
age          0.02200    0.01955   1.125 0.260541    
ed2          0.04718    0.27749   0.170 0.864982    
ed3          0.53835    0.38966   1.382 0.167089    
ed4         -0.20026    0.51061  -0.392 0.694909    
ed5          0.86566    1.23946   0.698 0.484916    
employ      -0.23061    0.03445  -6.695 2.16e-11 ***
address     -0.05505    0.02379  -2.314 0.020657 *  
debtinc      0.08605    0.02283   3.769 0.000164 ***
creddebt     0.45913    0.09037   5.080 3.77e-07 ***
othdebt     -0.03965    0.06572  -0.603 0.546317    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 619.55  on 565  degrees of freedom
Residual deviance: 451.23  on 555  degrees of freedom
AIC: 473.23

Number of Fisher Scoring iterations: 6
```

model5 = glm(default~.-othdebt,family = "binomial", data = train_data)

summary(model5)

```
summary(model5)

Call:
glm(formula = default ~ . - othdebt, family = "binomial", data =
train_data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6231  -0.6582  -0.3273  -0.0662   2.7335

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.614607   0.609953  -2.647 0.008118 **
age          0.021360   0.019471   1.097 0.272638
ed2          0.043539   0.277263   0.157 0.875219
ed3          0.508308   0.382073   1.330 0.183388
ed4         -0.185551   0.522414  -0.355 0.722455
ed5          0.869680   1.260921   0.690 0.490372
employ      -0.232481   0.034311  -6.776 1.24e-11 ***
address     -0.054748   0.023788  -2.301 0.021363 *
income      -0.002734   0.005962  -0.459 0.646548
debtinc      0.075127   0.021703   3.462 0.000537 ***
creddebt     0.471903   0.108272   4.358 1.31e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 619.55  on 565  degrees of freedom
Residual deviance: 451.38  on 555  degrees of freedom
AIC: 473.38

Number of Fisher Scoring iterations: 6
```

From above all model, the model2 have least AIC value and in feature selection technique (Boruta function) we find that ed is least contributing but I finalize the model1 because in logistic regression we need big sample size and in model2 debtinc become least significant as compare to model1.

##Check the **accuracy**

prob = predict(model1,type = c("response"),train_data)

prob

confusion = table(prob>0.50,train_data$default)

confusion

```
confusion

         0   1
  FALSE 405  79
  TRUE   27  55
```

##Model accuracy

Accuracy = sum(diag(confusion)/sum(confusion))

Accuracy

##0.8127208

##Now check the accuracy for test data

prob1 = predict(model1,type = c("response"),test_data)

prob1

confusion_1 = table(prob1>0.50,test_data$default)

confusion_1

```
confusion_1

         0   1
  FALSE 222  31
  TRUE   13  18
```

Accuracy_1 = sum(diag(confusion_1)/sum(confusion_1))

Accuracy_1

##0.8450704

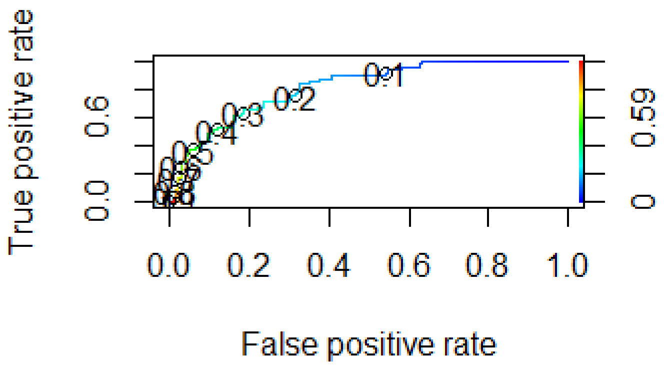
##Check the threshold value to decrease the false positive rate

library(ROCR)

ROCRPred = prediction(prob1,test_data$default)

ROCRPref = performance(ROCRPred,"tpr","fpr")

plot(ROCRPref, colorize=TRUE,print.cutoffs.at=seq(0.1,by=0.1))



prob = predict(model1,type = c("response"),train_data)

prob

##From ROC curve I take the threshold value 0.44

confusion = table(prob>0.44,train_data$default)

confusion

```
         0   1
  FALSE 392  66
  TRUE   40  68
```

Accuracy = sum(diag(confusion)/sum(confusion))

Accuracy

```
0.8127208
```

Here we conclude that our accuracy is same of train data but precision is increased.

##Calculate the AUC

library(pROC)

auc = performance(ROCRPred,measure = "auc")

auc = auc@y.values[[1]]

```
auc ## 0.8197134
```

Here accuracy of train data is less than accuracy of test data

So our model is underfit

DECISION TREE

##We use Decision tree algorithm for enhancing accuracy

For applying the decision tree first we have to check the distribution of 0 & 1 in train and test data.Distribution of 0 & 1 must be or approx equal in train and test data

prop.table(table(train_data$default))

```
        0         1
0.7632509 0.2367491
```

prop.table(table(test_data$default))

```
        0         1
```

```
0.8274648 0.1725352
```

We can proceed because the value of distribution are acceptable. The distribution is not biased

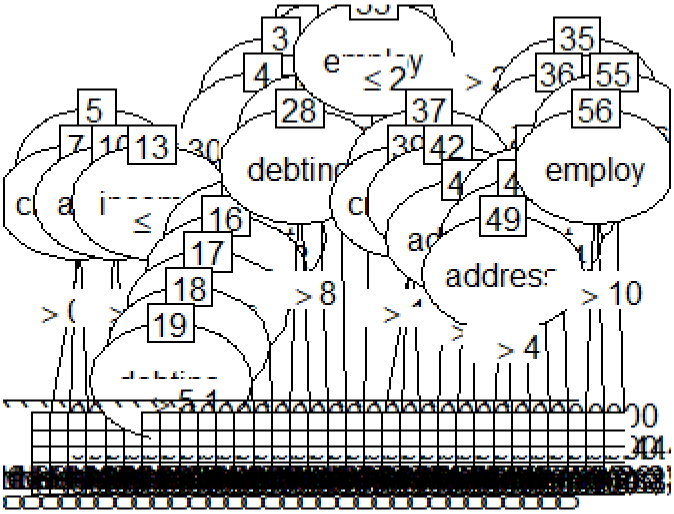
##Library for Decision Tree

library(C50)

##Building model on Training Data

Train_model = C5.0(train_data[,-9],train_data$default)

plot(Train_model)



##Training Accuracy

pred_train = predict(Train_model, train_data[,-9])

table(pred_train, train_data$default)

```
pred_train   0   1
         0 415  39
         1  17  95
```

mean(train_data$default==predict(Train_model,train_data))

```
## 0.9010601
```

##Testing Accuracy

pred_test = predict(Train_model,newdata = test_data[,-9])

table(pred_test, test_data$default)

```
pred_test   0   1
        0 207  30
        1  28  19
mean(pred_test==test_data$default)
0.7957746
```

I think that decision tree is not suitable for this dataset because the accuracy of train data is too much higher than test data. I think our model is overfit .

RANDOM FOREST

##By using Random Forest
library(randomForest)

#Building the Random Forest model on training model

fit.forest=randomForest(default~.,data = train_data, na.action = na.roughfix, importance=TRUE)

##Fit.forest(Prediction)

pred_t1 = fit.forest$predicted

table(pred_t1,train_data$default)

```
pred_t1   0   1
      0 399  89
      1  33  45
```

mean(train_data$default==pred_t1)

```
## 0.7844523
```

##Predicting accuracy on test data

pred_t2 = predict(fit.forest,newdata = test_data[,-9])

table(pred_t2,test_data$default)

```
pred_t2   0   1
      0 217  33
      1  18  16
```

mean(test_data$default==pred_t2)

```
## 0.8204225
```

##This model should be exceotable because the accuracy of both the tran and test data almost equal

##confusion matrix(using caret)

library(caret)

confusionMatrix(train_data$default,fit.forest$predicted)

```
          Accuracy : 0.7845
                 95% CI : (0.7483, 0.8177)
    No Information Rate : 0.8622
    P-Value [Acc > NIR] : 1

                  Kappa : 0.3031

 Mcnemar's Test P-Value : 6.376e-07

            Sensitivity : 0.8176
            Specificity : 0.5769
         Pos Pred Value : 0.9236
         Neg Pred Value : 0.3358
             Prevalence : 0.8622
         Detection Rate : 0.7049
   Detection Prevalence : 0.7633
      Balanced Accuracy : 0.6973

       'Positive' Class : 0
```

confusionMatrix(test_data$default,pred_t2)

```
               Accuracy : 0.8204
                 95% CI : (0.7707, 0.8633)
    No Information Rate : 0.8803
    P-Value [Acc > NIR] : 0.99876

                  Kappa : 0.2844

 Mcnemar's Test P-Value : 0.04995

            Sensitivity : 0.8680
            Specificity : 0.4706
         Pos Pred Value : 0.9234
         Neg Pred Value : 0.3265
             Prevalence : 0.8803
         Detection Rate : 0.7641
   Detection Prevalence : 0.8275
      Balanced Accuracy : 0.6693

       'Positive' Class : 0
```

##Crosstable

library(gmodels)

rf_perf<-CrossTable(train_data$default,fit.forest$predicted,prop.chisq=FALSE,prop.c= FALSE,prop.r = FALSE,dnn = c("actual default","predicted default"))

```
  Cell Contents
|-------------------------|
|                       N |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  566
```

```
               | predicted default
actual default |         0 |         1 | Row Total |
---------------|-----------|-----------|-----------|
             0 |       399 |        33 |       432 |
               |     0.705 |     0.058 |           |
---------------|-----------|-----------|-----------|
             1 |        89 |        45 |       134 |
               |     0.157 |     0.080 |           |
---------------|-----------|-----------|-----------|
  Column Total |       488 |        78 |       566 |
---------------|-----------|-----------|-----------|
```

I am not able to get the same accuracy which I get in python and my precision is also very high when I use Random Forest algorithm in Python.