Project 2 – Predicting credit risk defaults

Problem background:

Analysing credit risk has become one of the most important and pressing needs today. Loan defaults and bad loans have been on a rise and thus the need of analysing the risk of lending credit before sanctioning a loan is extremely critical. Most of the lending institutions have already adopted a risk analysis process which not only determines a risk but also helps reason the potential outcomes depending on various behavioural, financial and expense factors of the applicant. The behavioural factors include the **age**, **gender**, **education**, **address** etc. which helps us determine an overall living of the applicant. The financial factors include **income**, **credit cards**, **expenses**, **debts**, **mortgages** which shed light on the sources of income and expense. The expense factors include the **credit limits**, the **debt to income** & **debt to credit ratios** and also if there are any other **outstanding debts** which overall contribute to the **credit score**.

Finally, the credit score is a numerical value ranging from **300 to 850** which is assigned to every individual who has executed a financial transaction such as borrowing a loan or issuing a credit card etc. This score is a combination of multiple factors such **borrowing capacity**, i.e. whether the applicant has a steady and well above cut-off income or has collaterals which can be mortgaged against a loan. Also the **repayment history,** i.e. whether or not the applicant has been repaying the credit card dues or the loan installments in a timely manner. These contribute majorly in the overall score determination process, but other factors such length of association or payment modes also contribute to a small extent in credit score determination.

At the end of the day, we're surrounded by Finance and financial transactions and the rapid rise of expansion in this sector at both private and public levels has triggered a sense of urgency for lending institutions to have such robust models in place to avoid any bad loans or defaults on loans, therefore we're analysing a credit risk for a bank issuing loans to their customers. The bank has a list of parameters on basis of which it will determine the loan eligibility, which we'll see below:

Problem statement:

A bank wants to analyse the risk of issuing credit to its customer and classify their applicants as potential borrowers or defaulters on the basis of 8 behavioural & financial attributes.

Train data – **700** observations, **8** variables
Test data – **150** observations, **8** variables

Dependent variable – **default** (Categorical – Binary i.e. 2 levels, **0** and **1**)
0 – no default
1 – default

Independent variables:

Age – Age of the applicant in years. Continuous variable. No missing values.
Education – Education of the applicant. Ordinal levels – 5. No missing values.
Employment – No. of years at the current company. Continuous variable. No missing values.
Address – No. of years at the current address. Continuous variable. No missing values.
Income – Gross annual income of the applicant. Continuous variable. No missing values.
Debt-to-income ratio – The DTI ratio, monthly debts against monthly income. No missing values.
Debt-to-credit ratio – The DC ratio, sum of debts against total available credit. No missing values.
Other debts – The other outstanding debts of the applicant. No missing values.

Default – Historical data of applicants who have & who have not defaulted in the past.

Domain knowledge:

This problem, being a finance domain problem, will need some input related to the financial terms and calculations on the basis of which we'll list our assumptions and build our models. We have been provided with applicant's gross annual income (in thousand dollars) and two ratios indicating the debt to income & debt to credit. Understanding these ratios is critical in predicting credit risk.

Debt to income ratio – It is all monthly debt divided by gross monthly income. We have been provided with annual income, thus we'll assume the debt to also be annually. This debt accounts for credit card bill dues, EMI's, mortgages etc. against your monthly gross income. Assuming my monthly debt is $ 2000 and my monthly income is $ 6000, then my DTI is 2000/6000 → **33.33%.** There is no good or bad ratio in this case, but generally lenders would want to see a medium debt against a high income.

Debt to credit ratio – It is total credit you owe divided by total credit available to you. In simple terms the credit limit set on your credit card is a good example of this ratio. The credit limit is set by your lender based on your income and the amount you spend against your credit limit is the debt to credit ratio. Assuming I only have a single credit card and my credit card limit is $ 10000 per month and on an average my spending on credit card is $ 6000 every month, thus we'll calculate the debt to credit ratio as 6000/10000 → **60%.** If I add another card, with credit limit of $ 5000, my debt to credit ratio will change to 6000/15000 → **40%.** This ratio keeps changing every month, thus lenders prefer this ratio be less than or equal to **30%**.

Any other outstanding debts will be added to this debt to credit ratio and so will be any further credit limit upgrades or downgrades. Both ratios check your individual ability to borrow credit as well as the capacity to repay based on your previous history, hence are very critical.

Data pre-processing:

Missing values:

As listed previously, none of the independent variables have any missing values, although the dependent variable '**default**' has 150 missing observations. However we have been only provided with a train dataset, thus we'll use the missing observations as our test data to predict the outcomes.

Outliers:

As listed in our domain knowledge section, the ratios in our case are extremely critical. Thus capping the outliers from our dataset for inadvertent values could prove fatal, because an abnormal ratio could mean the applicant has been defaulting on previous credits, thus we'll retain all outlier values from our dataset and continue with our analysis.

Assumptions:

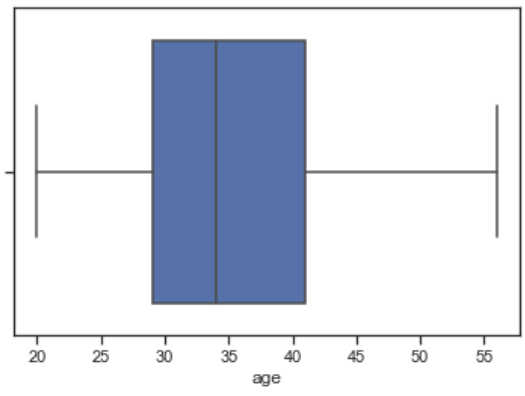
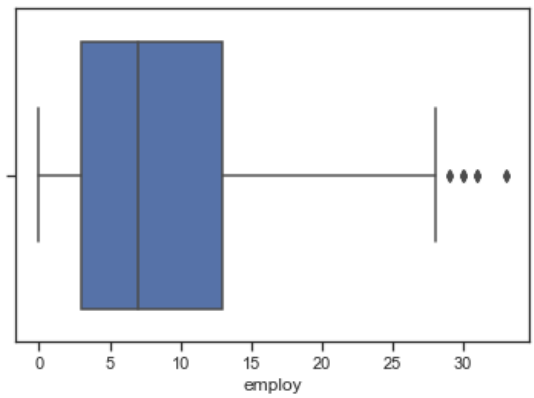
Age & experience variables are directly proportional & linearly correlated.
Education & income variables are directly proportional.
Income is assumed to be value times thousand in dollars.
Debt to income ratio is assumed to be annual and not monthly.
Debt to credit ratio is assumed to be annual and on all products of the bank.
Debt to income & default are correlated, higher the ratio, higher the chances of default.
Debt to credit & default are correlated, higher the ratio, higher the chances of default.
More the no. of years at one address, less the chances of default.
Higher the educational degree, lower the chances of defaulting on a loan.

Data exploration:

In data exploration, we'll be exploring relationships between our independent variables,

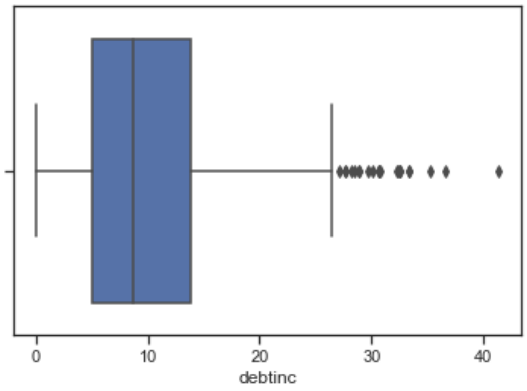
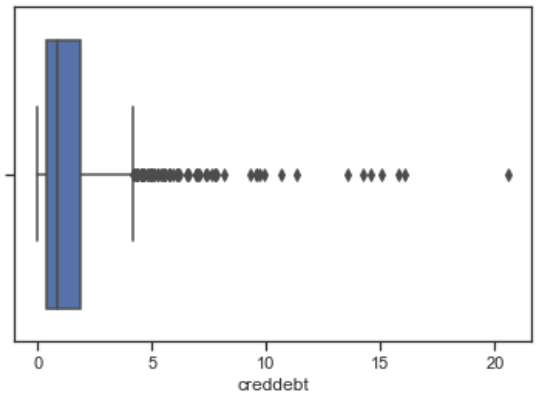
Boxplots:

Age	Employ

`sns.boxplot(x=train_loan['age'])`
`<AxesSubplot:xlabel='age'>`
 
	`sns.boxplot(x=train_loan['employ'])`
	`<AxesSubplot:xlabel='employ'>`
	 

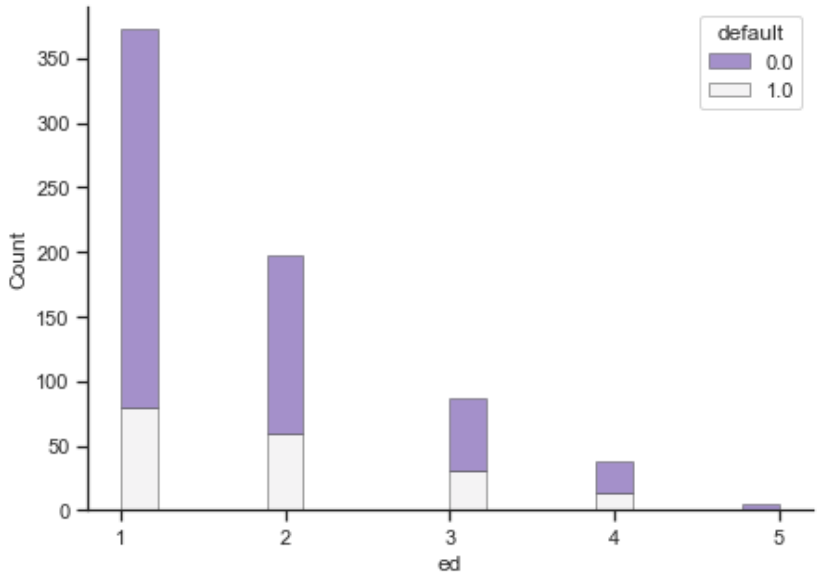
As data is sensitive to prediction, we'll retain the outlier values and not treat them.

Debt to income ratio	Debt to credit ratio

`sns.boxplot(x=train_loan['debtinc'])`
`<AxesSubplot:xlabel='debtinc'>`
 
	`sns.boxplot(x=train_loan['creddebt'])`
	`<AxesSubplot:xlabel='creddebt'>`
	 

Both of these ratios are extremely important for our analysis. If we treat these outliers by capping off the values, the model would **not understand** the extreme values which will in turn help model to categorize it as a **default** or **no default**.

Let us now check the count of defaults or no defaults depending on the education:



The above plot shows the count of defaults or no defaults based on the education level each applicant has attained. We initially assumed that the higher the level of education, lower the chances of the applicant defaulting. Let us find out if it is actually true.

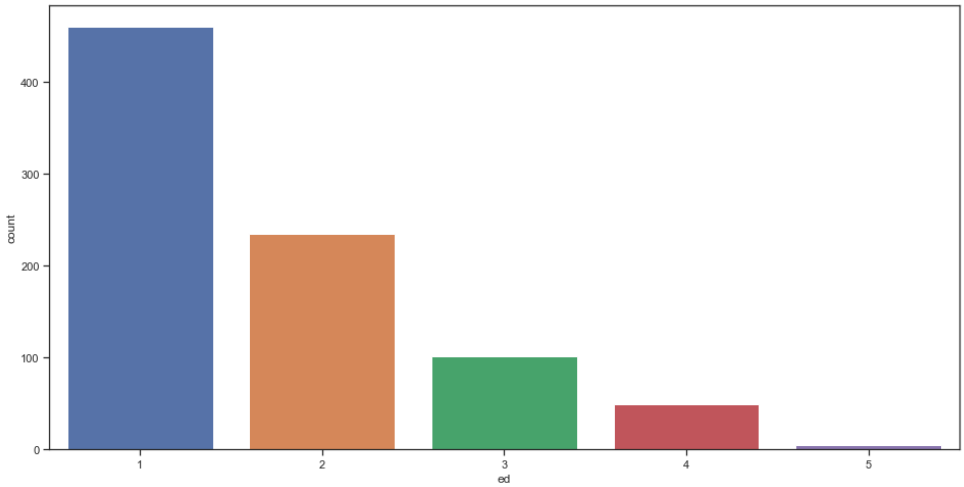
We do see a decline in no. of defaults as the education levels starts to rise. So it confirms our assumption that more educated the applicant is, less likely the applicant would default.

Education level 1 sees the highest no. of defaults, which is expected given the highest level of education those applicants have attained is just **high school**. These applicants are probably skilled workers and work low-grade jobs and just make enough to make ends meet. Such applicants also do not show a very strong employment history, thus are most likely to default on money borrowed.

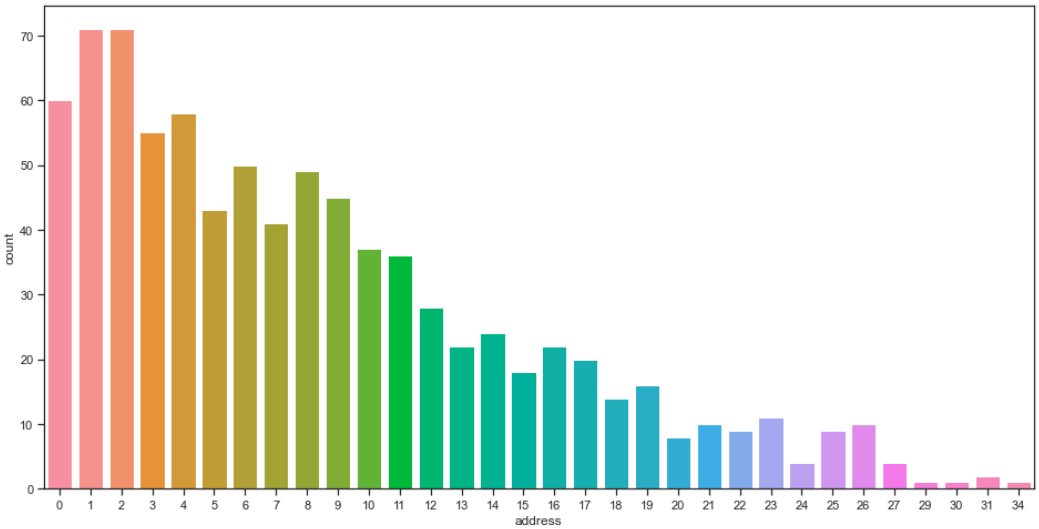
Education level 2 & level 3 see drastic drops in count of no. of defaults. Considering the Education level 2 as college dropouts, the count is reasonable. Education level 3 represents bachelor's degree, which is a well-respected degree in any profession. Thus the count of defaults with level 3 comes as a surprise, it could mean that at a particular time in life, the applicants came across a financial crunch, maybe due to limited education.

Education level 4, too sees a few no. of defaults. The education level is masters, a post-graduation degree, post which the applicant would definitely be eligible for a well-paying job, so ideally shouldn't be of any concern, yet we see a spike. Possible reasons could be loss of a job or a sudden loss in a business venture which yielded a financial crunch.

Education level 5 sees no defaults. The highest possible doctorate degree applicants are definitely dedicated and highly respected and would ideally never default on a borrowed loan.



The above plot shows the count of applicants grouped by education level. We see most of the applicants are not well educated. The count drops as the education level rises.

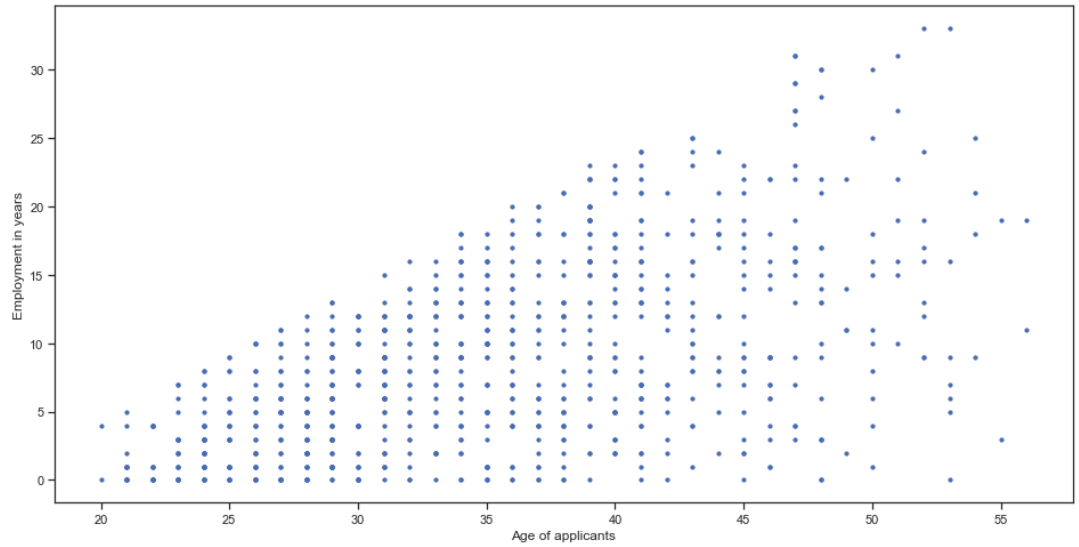


The above plot shows the count of applicants grouped by address or locality. The distribution of the applicants is over a wide area, with no more than 70 applicants from one locality.

Localities **1** & **2** each has 70 applicants, which could mean that the bank would be near to these localities or the bank is taking special efforts to target applicants from this locality.

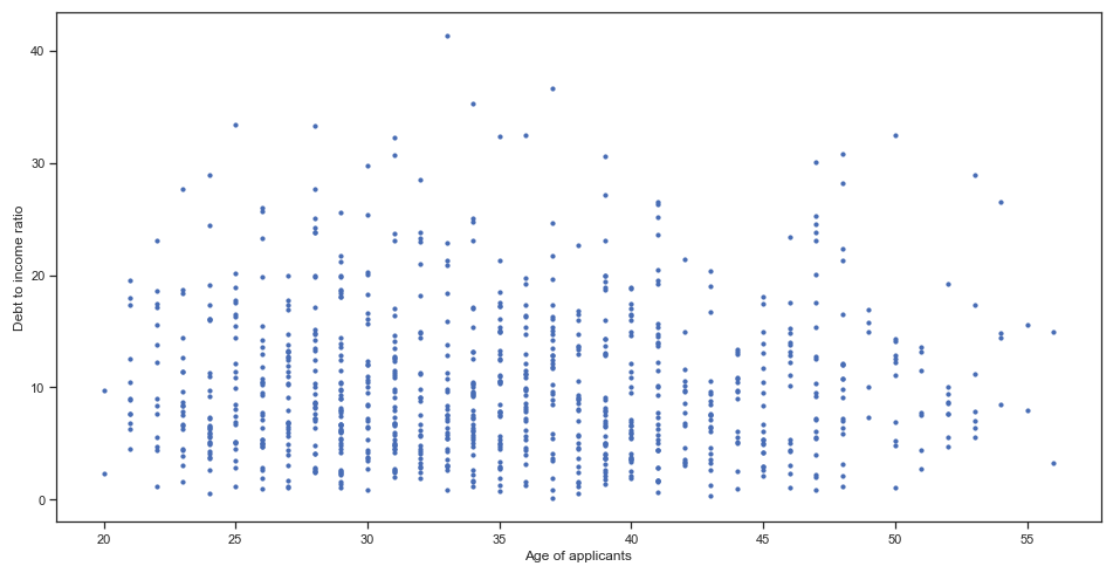
Scatter plots:

Age vs employment:



We previously assumed that the relationship between age and no. of years of employment would be linear and directly proportional, which is very evident here in this plot. As the age increases, the experience at the current company also increases.

Age vs debt to income ratio

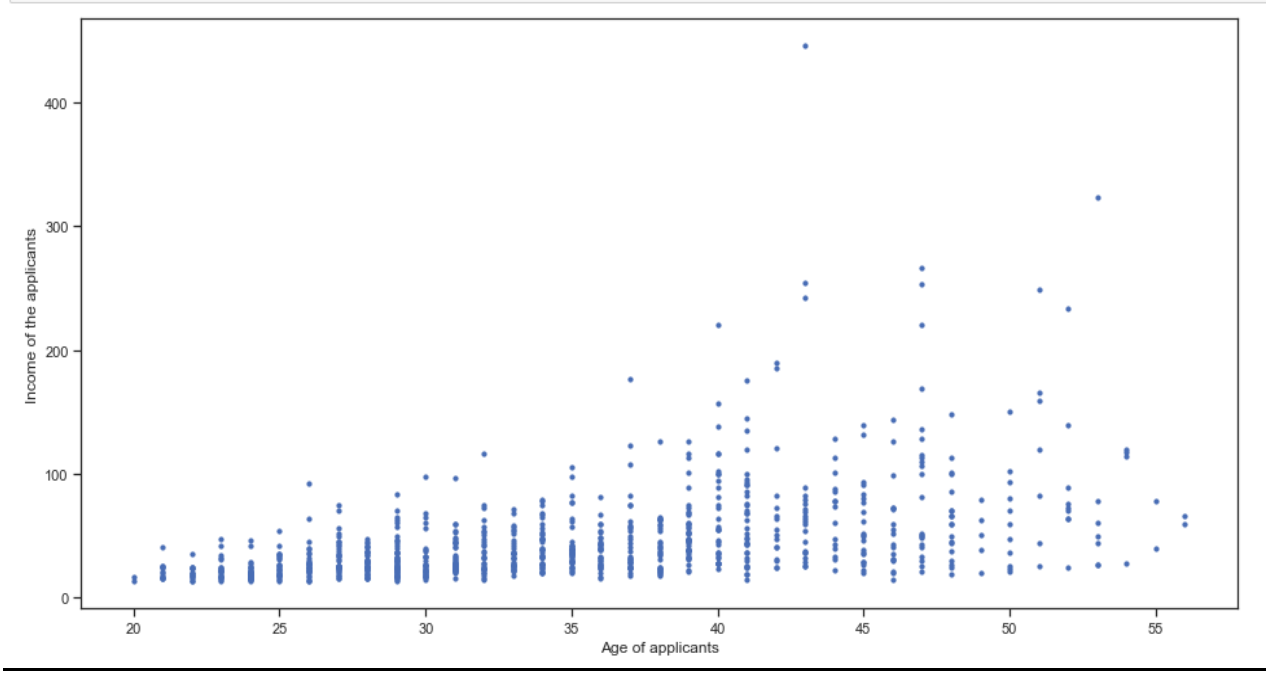


The above plot shows the relationship between the age of the applicants against their annual debt to income ratio. The total debt they owe against the income they earn is what is evident here.

A higher debt to income ratio means that an applicant has more debt to repay on a consistent basis. This could vary with age, because at different ages, the need for money is different. An unmarried person would have less dependencies than a married person thus would need less money compared to a married person, in turn would owe lesser debt than the married person.

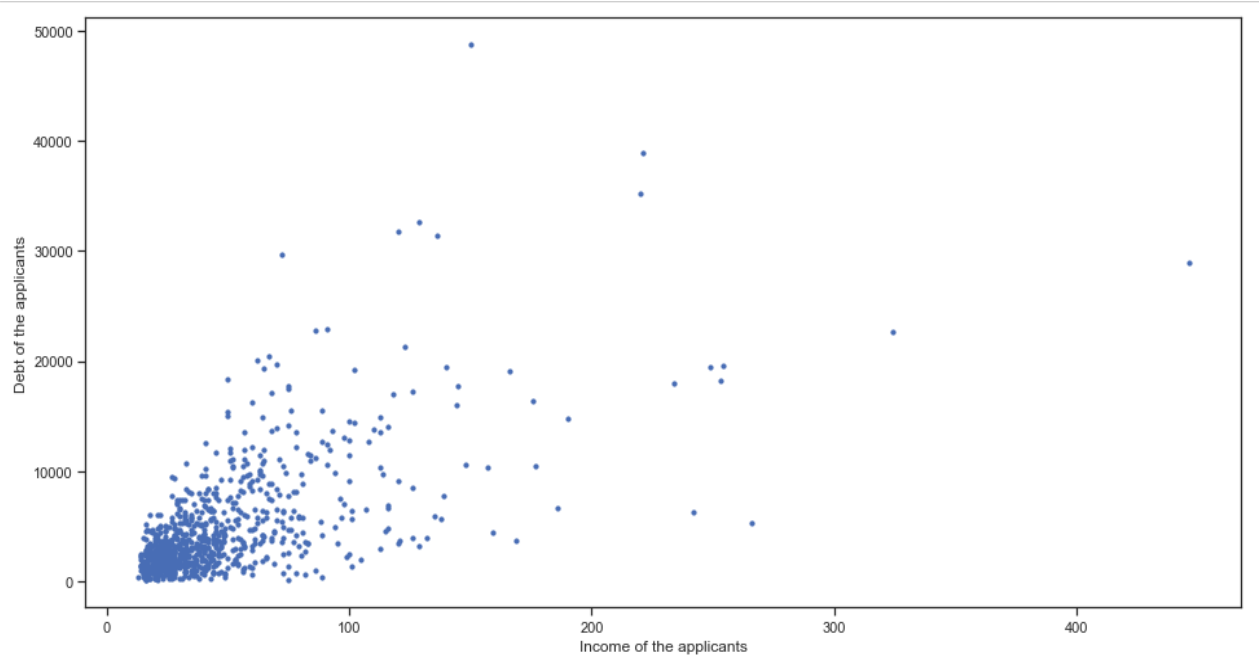
We observe that most applicants between 30 & 40 years old have the highest debt to income ratios, which is pretty standard given the fact that they have recently added new responsibilities.

Age vs income



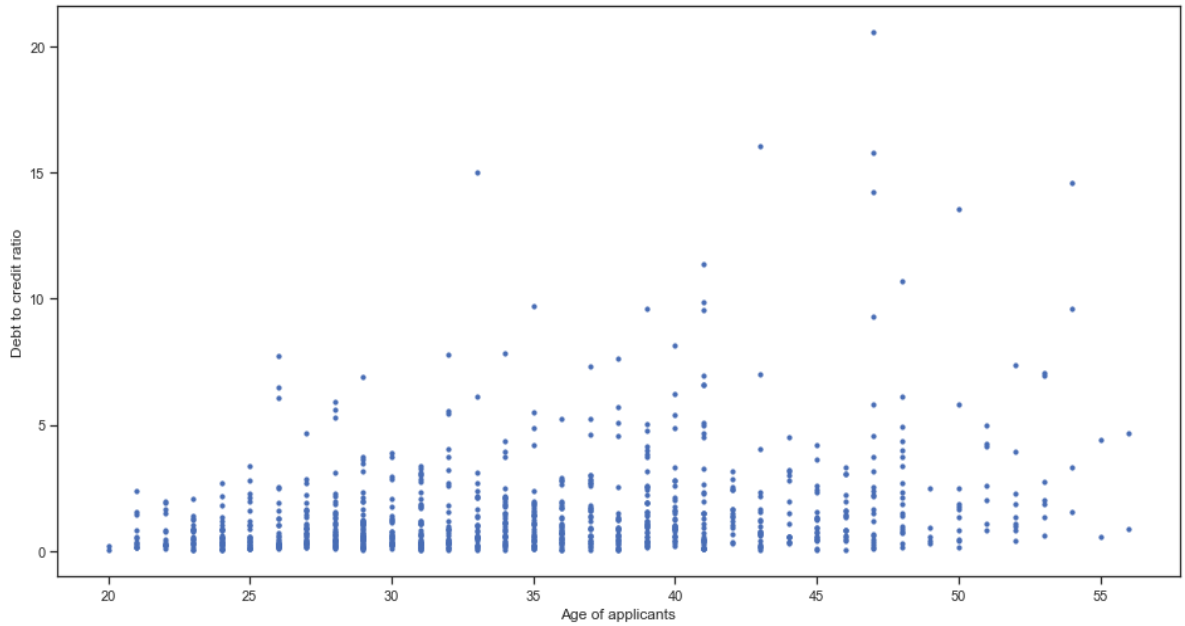
Again one of our assumptions earlier that the income will increase with age as the experience increases with age and income increases with experience.

The applicants with income above 200 thousand dollars are all aged **above 40**. The experience is one of the key parameter which adds depth to this statistic.



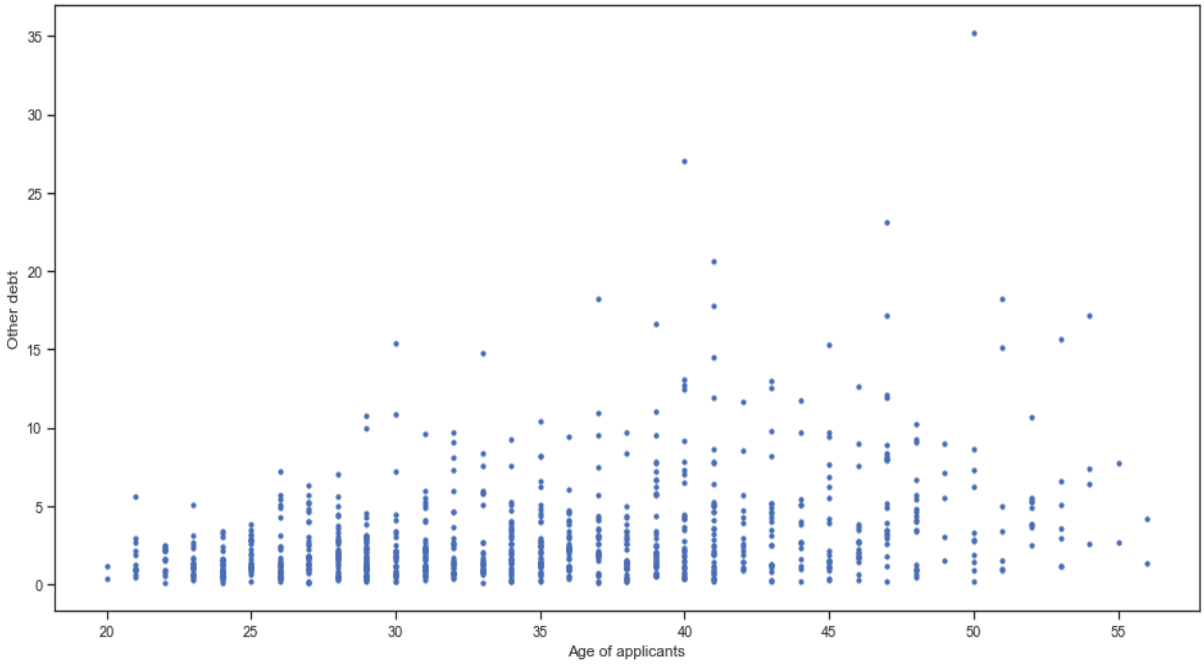
This is a comparison of income of the applicant against the debt owed by the applicant. We observe that the points are very closely related, which means the debt owed is directly proportional to the income. If the income is high, we spend more, thus have higher debts on our cards.

Age vs debt to credit ratio



The above plot shows the correlation between the age of applicants and debt to credit ratio. Here again the debt to credit ratio talks about your owed debts against the total available credit. The total available credit depends on the income, thus applicants aged 40 or more have high credit limit, and which is why the debt to credit scores are high.

Age vs other debt



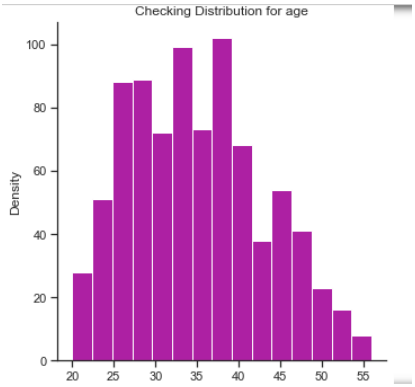
We observe from the plot above that applicants aged 35 or less owe lesser debts compared to applicants aged above 40. This is primarily due to the additional responsibilities of a family. The concentration of high number of debts falls between **ages 35-50.**

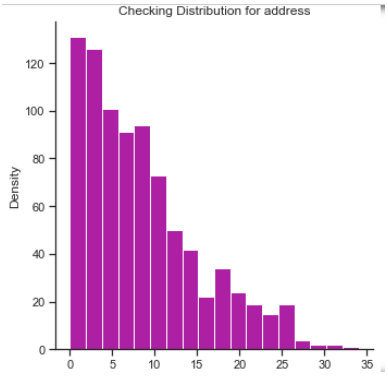
Feature transformation & scaling:

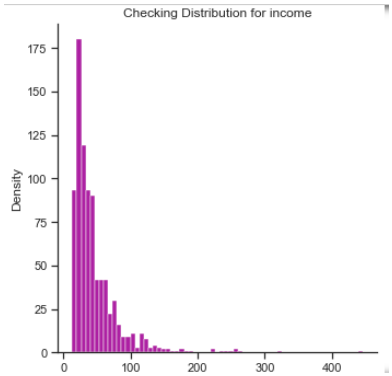
Transforming the independent variables or features of our dataset is critical as the range of values for our observations is varied.

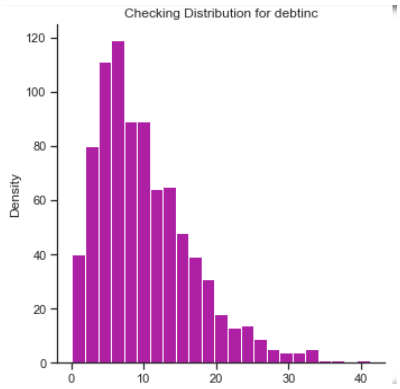
- Age – is a continuous variable with a defined range in **tens** throughout all observations
- Education – is a categorical variable with **5 levels**. Thus no transformation is necessary
- No. of years at a company – is a continuous variable with range in **tens** for all observations
- No. of years at Address – is a continuous variable with range in **tens** for all observations
- Income – is a continuous variable with range in **thousands** for all observations
- DTI ratio – is a continuous variable with range in **tens** for all observations
- DTC ratio – is a continuous variable with range in **tens** for all observations
- Other debt – is a continuous variable with range in **tens** for all observations
- Default – is a categorical variable with **2 levels**. Thus no transformation is necessary

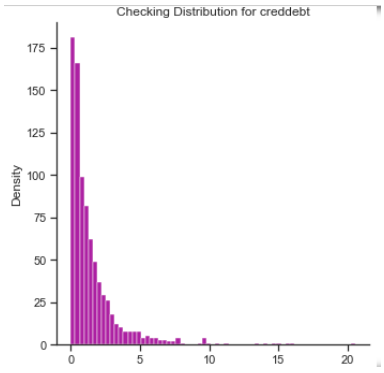
Therefore, we'll need to scale these variables to bring them to a same scale for easy interpretation. The distribution of these variables prior and post scaling is seen below:

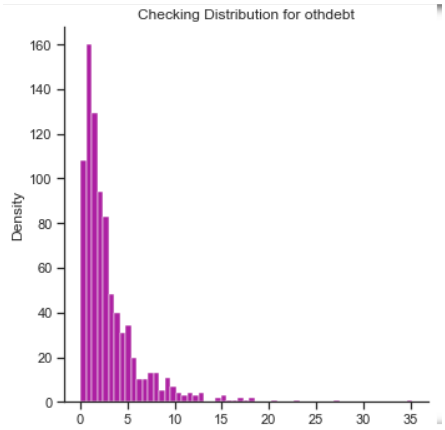




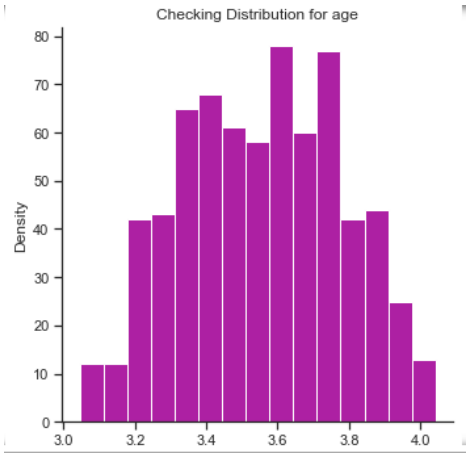


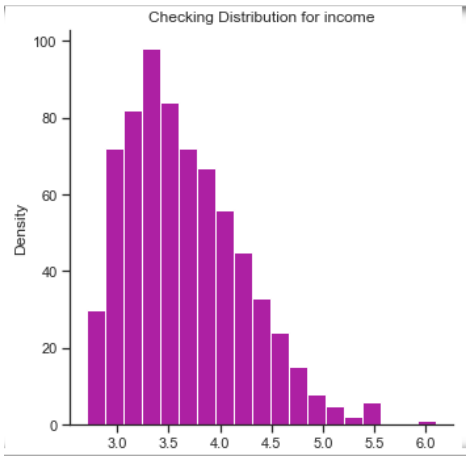


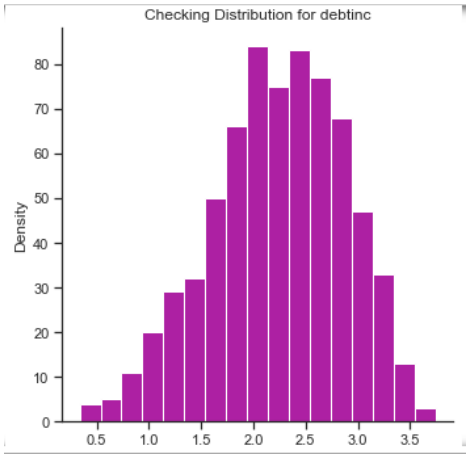


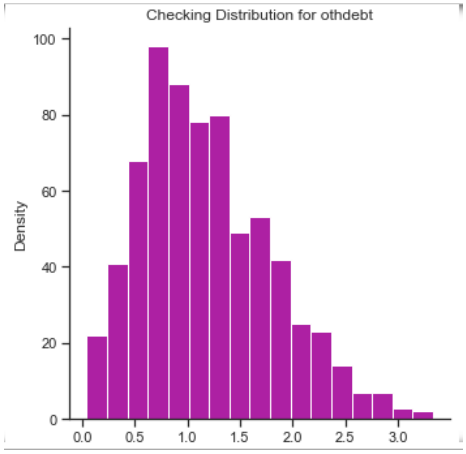


As we can see, the distribution for each of the above variables is skewed, to be precise, a positive right skew indicating that as the value on the x-axis increases, the count of observations decreases. Now, we'll apply the log scaling to our features and see if we observe a normal distribution.









As we clearly observe, all of the above variables are now in **units range.** They have been scaled down using log transformation and we also observe that the distribution is uniform as opposed to previously seen. Now that these variables are scaled uniformly, we'll split our data into train and test and also ensure that our predictor variable default has a right balance of both classes.

Balancing data:

The pre-processing of the data is now complete, the dataset is ready to be fed to our models. However this problem is a classification problem. The dependent variable is a categorical variable with 2 levels (binary classification). As it is with most real-life use cases, the distribution of positive & negative classes is not uniform, rather heavily skewed towards a particular class. To better explain with our example, **the no. of people who default are very few** compared to the **most who repay their borrowed loan.** This uneven distribution of classes makes the **model learn from a biased class** (majority class), this will certainly improve your accuracy of predicting the majority class correctly, but **fail to predict the minority class** correctly, because of insufficient training data fed to our model. We must always try to feed our model unbiased data, which will give our model sufficient training data for all the classes and thus ensure a more robust model. In short, we must balance out the classes by either evening out the **minority** or **majority** class. This can be achieved as follows:

Random oversampling – As the name suggests, oversampling refers to adding observations of minority class to match the no. of majority class observations. The word random means, the decision of which observations to sample is completely random. This only duplicates the existing observations and samples the data, causing a lot more noise than the imbalanced dataset.

Random under sampling – As the name suggests, under sampling refers to removing observations of majority class to match the no. of minority class observations. Again random means, the decisions of which observations to remove is completely random. Under sampling also causes loss of information. Meaning, the majority class observations which are randomly removed could hold important information for our model to learn from. Thus, is not really widely adopted.

Both these sampling techniques deal with data randomly, which causes our data to lose its essence either by addition of excess noise or loss of critical information. Therefore, a preferred technique is **SMOTE –** abbreviated for **S**ynthetic **M**inority **O**versampling **Te**chnique, which again as the name suggests adds observations to minority class, just that it uses ML techniques such as **K-nearest neighbour or equivalent** algorithms to synthetically predict new observations as opposed to randomly creating them.

We have used **SMOTE,** to fill in our minority class (1) with observations to match with our majority class (0). After running SMOTE, the dataset was perfectly balanced at **ratio 1:1,** indicating that our model was fed information with exact entries of class (0) and class (1), making it perfectly unbiased.

As previously mentioned, I have used **75%** of my data to train the model and **25%** test our evaluation metrics on. This concludes the pre-processing stage and we now move to **model building**.

Model building:

As of now, our data has been imported, cleaned, normalised, explored and pre-processed for it to be fed to our models, which will run and be evaluated on few classification metrics. This being a classification problem we've modelled our dataset using the following models:

- Logistic Regression
- K-Nearest Neighbour – Classifier
- Naïve Bayes – Classifier
- Decision Tree – Classifier (Naïve as well as Hyperparameter tuned)
- Random Forest – Classifier (Naïve as well as Hyperparameter tuned)

We've then each evaluated these models on the basis of their respective confusion matrices. Let's look into the details of the confusion matrix parameters.

Confusion Matrix		***Actuals***	
		1 (default)	**0** (no default)
Predicted	**1** (default)	True Positive (**TP**)	False Positive (**FP**)
	0 (no default)	False Negative (**FN**)	True Negative (**TN**)

Table headers:

Actuals – The actual class of an observation before a prediction is made. (historical data value)

Predicted – The predicted class of an observation after a prediction is made. (predicted value)

Classes:

1 – Is the positive class in both actual as well as predicted.

0 – Is the negative class in both actual as well as predicted.

Table values:

True positive – When, both the actual & predicted values of the positive class (**1**) are correctly predicted by the model. In our context, an applicant was labelled as a '**defaulter**' and our model correctly predicted prospective applicant with **similar** predictors also as a '**potential defaulter**'. This is important for our bank to get it right.

False positive – When, the actual value of negative class (**0**) is incorrectly predicted as positive class (**1**), we get a false positive. In our context, an applicant was labelled as a '**non-defaulter**' and our model incorrectly predicted a prospective applicant with **different** predictors as a '**potential defaulter**'. This contributes to an **error** by our model.

False negative – When, the actual value of positive class (**1**) is incorrectly predicted as negative class (**0**), we get a false negative. In our context, an applicant was labelled as a '**defaulter**' and our model incorrectly predicted a prospective applicant with **similar** predictors as a '**potential non-defaulter**'. This also is attributed as an **error** by our model.

True negative – When, both the actual & predicted values of the negative class (**0**) are correctly predicted by the model. In our context, an applicant labelled as a '**non defaulter**' and our model correctly predicted prospective applicant with similar predictors also a '**potential non defaulter**'. This is perhaps the most important metric to evaluate.

Summary:

True positives (TP's) → Correct rejections
False positives (FP's) → **Type I error**
False negatives (FN's) → **Type II error**
True negatives (TN's) → Correct acceptance

Objective – predict **TP's** and **TN's** i.e. correctly predicting the defaulters & non defaulters. To have minimal error on **FP's** and **FN's** i.e. incorrectly accepting or rejecting a defaulter or non-defaulter.
Interpreting the confusion matrix:

Now that we've understood the confusion matrix and its structure, it is even more important to understand how to interpret these table values and convert them to evaluation metrics.

Accuracy → TP + TN / TP + TN + FP + FN

Sensitivity or **Recall** or **True Positive Rate** → TP / TP + FN

Specificity or **True Negative Rate** → TN / TN + FP

Precision → TP / TP + FP

False Negative Rate → FN / FN + TP

False Positive Rate → FP / FP + TN

F1 Score → 2 * (Precision * TPR) / (Precision + TPR)

ROC → TPR vs FPR

Accuracy – No. of correct predictions (defaulters & non-defaulters) against total observations.
Sensitivity – No. of correctly predicted positives (defaulters) against sum of TP & Type II error.
Specificity – No. of correctly predicted negatives (non-defaulters) against sum of TN & Type I error.
Precision – No. of correctly predicted positives (defaulters) against total positives (TP + FP).
FNR – No. of incorrectly predicted negatives (non-defaulters) against predicted positives (TP + FN).
FPR – No. of incorrectly predicted positives (defaulters) against predicted negatives (FP + TN).

Now the final check before we move to modelling is, the evaluation cut-offs.

Accuracy → **High** | Sensitivity → **High** | Specificity → **Low** | Precision → **High** | FNR & FPR → **Low**

Logistic Regression:

Starting with our very first model, we've built a naïve logistic regression model as follows:

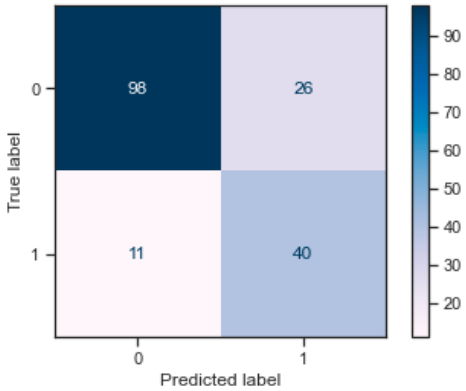
Logistic Regression

```
logreg = LogisticRegression() # Initiate an instance of LogisticRegression class and assign it to an object
logreg.fit(X_train.sm, y_train.sm) # Call the function fit to fit our model using the object we created
```

```
LogisticRegression()
```

```
pred_LR = logreg.predict(X_test) # Predicting the model on test data and storing it in an object
```

The confusion matrix for the prediction on the test data is as follows:



As we observe, the **TN** and **TP** have been identified well. We have a total of **175** observations in our test dataset, and out of those many, **138** have been correctly identified.

The accuracy of the model is **78.85**%, but accuracy at times can be misleading, thus we'll also take a look at the other parameters we mentioned above.

```
print("Accuracy of LR model is -",metrics.accuracy_score(y_test, pred_LR))
print("Precision of LR model is -",metrics.precision_score(y_test, pred_LR))
print("Recall of LR model is -",metrics.recall_score(y_test, pred_LR))
print("F1 score of LR model is -",metrics.f1_score(y_test, pred_LR))
```

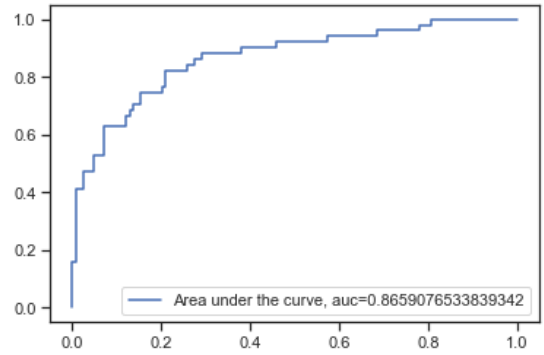
```
Accuracy of LR model is - 0.7885714285714286
Precision of LR model is - 0.6060606060606061
Recall of LR model is - 0.7843137254901961
F1 score of LR model is - 0.6837606837606838
```

Precision of the model is **60**%, i.e. model is correctly predicting the potential loan rejections at 60% accuracy from all the potential rejections pile.

Recall of the model is **78**%, i.e. model is correctly predicting the potential loan rejections at 78% accuracy including the ones which should have been identified as defaulters.

F1 score of the model is **68**%, i.e. the harmonic mean of precision & recall is measured at 68%. With relatively low number of false positives (26) and false negatives (11), our model has achieved an above average precision & recall score, thus a **good f1 score**.

```
Pred_LR_prob = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  Pred_LR_prob)
auc = metrics.roc_auc_score(y_test, Pred_LR_prob)
plt.plot(fpr,tpr,label="Area under the curve, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



As we can see from the above plot, the **AUC – Area under the curve** represents the measure of ability to distinguish the classes in a classifier. In our context, the AUC for this model is ~**86**%, which indicates that our model identifies the two classes, i.e. the defaulters and non-defaulters with a 86% accuracy, which is an extremely good functioning model.

K-Nearest Neighbour:

The next algorithm is KNN classifier. We've built a KNN classifier model as follows:

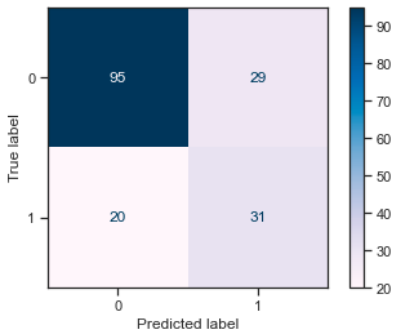
K-Nearest Neighbor

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train.sm, y_train.sm)
```

```
KNeighborsClassifier(n_neighbors=3)
```

```
pred_KNN = knn.predict(X_test)
```

We've fed the no. of neighbours as **3,** we'll first evaluate the performance with 3 neighbours and try changing the values for optimising the results obtained from this model.



According to the above confusion matrix, our model is performing poorer compared to the logistic regression model, however as mentioned, we could optimise the performance by altering the no. of neighbours we provide in the subsequent runs.

It is still predicting descent numbers of **TN** & **TP,** only with slightly higher number of errors. If we check the evaluation metrics for this model we have:

```
print("Accuracy of KNN model is -",metrics.accuracy_score(y_test, pred_KNN))
print("Precision of KNN model is -",metrics.precision_score(y_test, pred_KNN))
print("Recall of KNN model is -",metrics.recall_score(y_test, pred_KNN))
print("F1 score of KNN model is -",metrics.f1_score(y_test, pred_KNN))
```

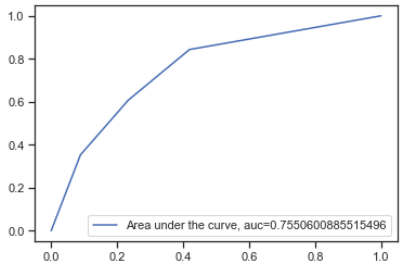
```
Accuracy of KNN model is - 0.72
Precision of KNN model is - 0.5166666666666667
Recall of KNN model is - 0.6078431372549019
F1 score of KNN model is - 0.5585585585585585
```

Accuracy of the model is **72**%.

Precision of the model is **51**%, i.e. model is correctly predicting the potential loan rejections at 51% accuracy from all the potential rejections pile.

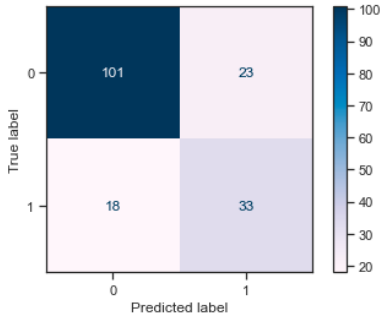
Recall of the model is **60**%, i.e. model is correctly predicting the potential loan rejections at 60% accuracy including the ones which should have been identified as defaulters.
F1 score of the model is **55**%, i.e. the harmonic mean of precision & recall is measured at 55%. With relatively high number of false positives (29) and false negatives (20), our model has achieved an slightly below average precision & recall scores, thus an average **f1 score**.

```
Pred_KNN_prob = knn.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  Pred_KNN_prob)
auc = metrics.roc_auc_score(y_test, Pred_KNN_prob)
plt.plot(fpr,tpr,label="Area under the curve, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



The **AUC is 75%**, which is still pretty descent. However let us try to update the no. of neighbours and check if it has any positive optimisation impact on our model.

We changed the no. of neighbours to **5,** instead of 3. The confusion matrix is as follows:



We observe that the accuracy of prediction of both **TN** & **TP** has increased, which in turn means that the classification error rate has reduced.

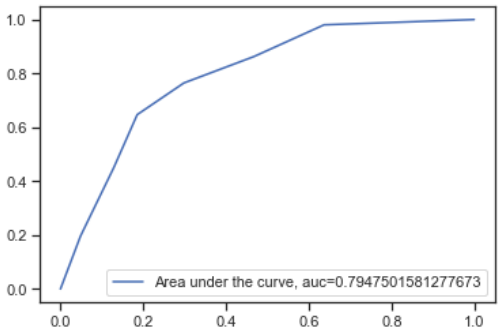
```
print("Accuracy of hypertuned KNN model is -",metrics.accuracy_score(y_test, pred_KNN1))
print("Precision of hypertuned KNN model is -",metrics.precision_score(y_test, pred_KNN1))
print("Recall of hypertuned KNN model is -",metrics.recall_score(y_test, pred_KNN1))
print("F1 score of hypertuned KNN model is -",metrics.f1_score(y_test, pred_KNN1))
```

```
Accuracy of hypertuned KNN model is - 0.7657142857142857
Precision of hypertuned KNN model is - 0.5892857142857143
Recall of hypertuned KNN model is - 0.6470588235294118
F1 score of hypertuned KNN model is - 0.6168224299065421
```

The accuracy has improved from **72**% to **76.5**%.
Precision has improved to **59**%.
Recall has also improved to **64**%, thus so is our f1 score to ~**62**%.

The model with no. of neighbours as **5** is performing better compared to **3** neighbours. We've also tried iterating the values of neighbours, but further increasing the value beyond 5 or decreasing the value below 3 is not yielding any better results than we've achieved with **5.**

Let's take a look at the ROC curve with neighbours as 5.



The AUC value has also increased from **75**% to ~**80**%, which means our model got better at classifying the defaulters and non-defaulters as we increased the radius of the neighbour search.

<u>Naïve Bayes Classifier:</u>

Moving on, we've built a naïve bayes classifier for probability calculations on our classes.

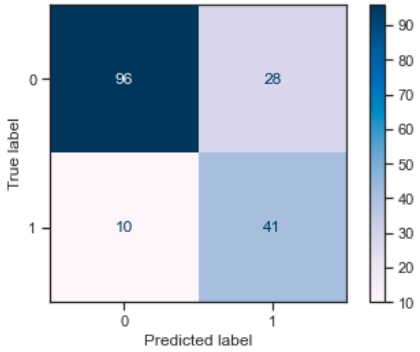
Naive Bayes

```
nb = GaussianNB()
nb.fit(X_train.sm, y_train.sm)
```

```
GaussianNB()
```

```
pred_NB = nb.predict(X_test)
```

Let us check the confusion matrix to evaluate the predictions.



The model is performing well in predicting **TP's** than both of our previous models. Also it performs well on predicting the **TN's** as well. Let's check the metrics:

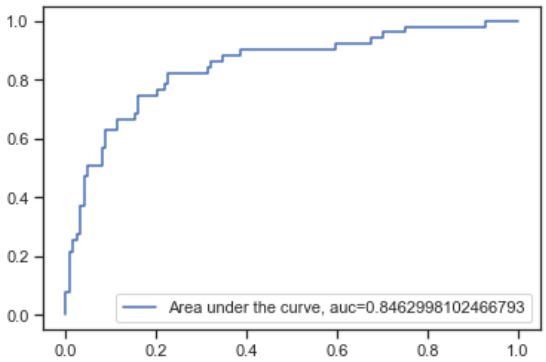
```
print("Accuracy of NB model is -",metrics.accuracy_score(y_test, pred_NB))
print("Precision of NB model is -",metrics.precision_score(y_test, pred_NB))
print("Recall of NB model is -",metrics.recall_score(y_test, pred_NB))
print("F1 score of NB model is -",metrics.f1_score(y_test, pred_NB))
```

```
Accuracy of NB model is - 0.7828571428571428
Precision of NB model is - 0.5942028985507246
Recall of NB model is - 0.803921568627451
F1 score of NB model is - 0.6833333333333333
```

Accuracy is **78**% | Precision is ~**60**% | Recall is at **80**% | F1 score is **68**%

As mentioned, the recall is high. It means, this model identifies the defaulters at **80**% accuracy, which is one of the important decisions for our bank. Rejecting the potential defaulters.

The ROC curve for the Naïve Bayes model is below:



The **AUC,** is at **84.6**%, which is extremely good. It means that this model is able to classify and distinguish between 2 classes at almost **85**% accuracy. This model has correctly identified the potential defaulters and also correctly identified the potential non-defaulters. Thus it performs well from the previous two models.

<u>Decision Tree Classifier:</u>

Moving on the ensemble models, first of which is our classification trees.

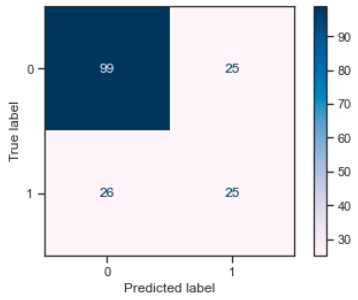
CART - Classification Tree

```
ct = DecisionTreeClassifier()
ct.fit(X_train.sm, y_train.sm)
```

```
DecisionTreeClassifier()
```

```
pred_CT = ct.predict(X_test)
```

Let us check the confusion matrix.



The model is predicting the TN's well. However the naïve version of the tree is susceptible to a high classification error, as no parameters are tuned. We'll still check the metrics:

```
print("Accuracy of CT model is -",metrics.accuracy_score(y_test, pred_CT))
print("Precision of CT model is -",metrics.precision_score(y_test, pred_CT))
print("Recall of CT model is -",metrics.recall_score(y_test, pred_CT))
print("F1 score of CT model is -",metrics.f1_score(y_test, pred_CT))
```

```
Accuracy of CT model is - 0.7085714285714285
Precision of CT model is - 0.5
Recall of CT model is - 0.49019607843137253
F1 score of CT model is - 0.495049504950495
```

The accuracy is **70**% | Precision is **50**% | Recall is **49**% | F1 score is **49.5**%

This confirms that the model with no parameters is performing poorly. We'll try to add a few hyperparameters which we can tune and also build our hyperparameter tuned models with both '**Gini index**' and '**entropy'** as criteria for node split and calculating the **information gain**.

Both **Gini index** and **Entropy** are used to calculate the information gain and in turn measure the quality of split at each node in a decision tree.

Gini measures the frequency of an element being mislabelled. The Gini index has a minimum value of **0** and maximum value of **0.5,** the decision to split or not is chosen with **lower** Gini value.

Entropy is a measure of information gain that indicates disorder of an element with the target. Unlike Gini, the range of entropy is between **0** and **1.** However the split is still with **lower** entropy.

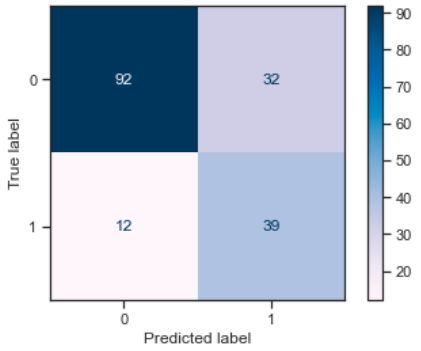
Hyperparameters tuning - Decision Tree (Entropy)

```
ct1 = DecisionTreeClassifier(criterion="entropy", max_depth=3)
ct1.fit(X_train.sm, y_train.sm)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

```
pred_CT1 = ct1.predict(X_test)
```

Here, we've only added the **max_depth** parameter, which will only restrict the growth of the tree.



If we observe carefully, the accuracy to predict the defaulters (TP) has increased. In turn reducing the classification error for incorrectly classifying non-defaulters as defaulters. It is an important step in optimisation. Let us explore further.

```
print("Accuracy of hypertuned CT model is -",metrics.accuracy_score(y_test, pred_CT1))
print("Precision of hypertuned CT model is -",metrics.precision_score(y_test, pred_CT1))
print("Recall of hypertuned CT model is -",metrics.recall_score(y_test, pred_CT1))
print("F1 score of hypertuned CT model is -",metrics.f1_score(y_test, pred_CT1))
```

```
Accuracy of hypertuned CT model is - 0.7485714285714286
Precision of hypertuned CT model is - 0.5492957746478874
Recall of hypertuned CT model is - 0.7647058823529411
F1 score of hypertuned CT model is - 0.6393442622950819
```

All the metrics have improved with just 1 addition of hyperparameter. The recall, f1 score and the accuracy all have improved drastically. Let us try to provide other hyperparameters.

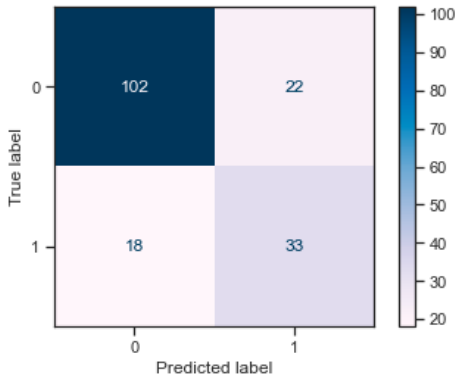
Hyperparameters tuning - Decision Tree (Entropy)

```
ct2 = DecisionTreeClassifier(criterion="entropy", max_depth=4, min_samples_leaf=10, min_samples_split=10)
ct2.fit(X_train.sm, y_train.sm)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4, min_samples_leaf=10,
                       min_samples_split=10)
```

```
pred_CT2 = ct2.predict(X_test)
```

In addition to max_depth, we've provided **min_samples_leaf** and **min_samples_split,** where min_samples_leaf guarantees the minimum no. of samples in every leaf irrespective of the min_samples_split.

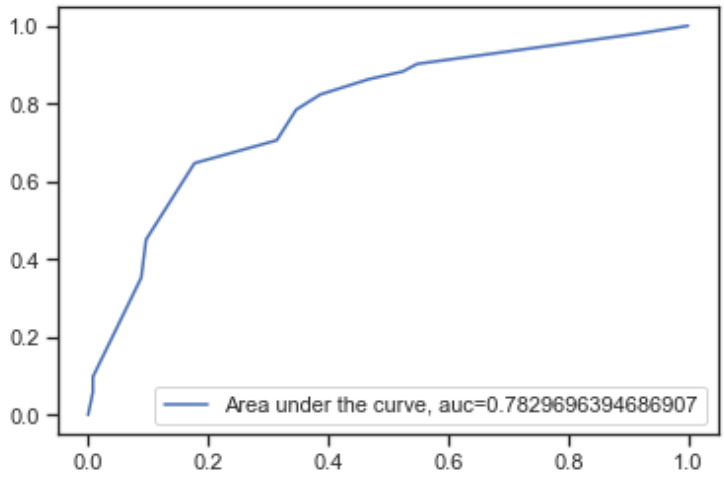


We see a drastic improvement in accurate predictions on **TN's** & **TP's.** Thus the classification error count has also reduced.

```
print("Accuracy of hypertuned CT model is -",metrics.accuracy_score(y_test, pred_CT2))
print("Precision of hypertuned CT model is -",metrics.precision_score(y_test, pred_CT2))
print("Recall of hypertuned CT model is -",metrics.recall_score(y_test, pred_CT2))
print("F1 score of hypertuned CT model is -",metrics.f1_score(y_test, pred_CT2))
```

```
Accuracy of hypertuned CT model is - 0.7714285714285715
Precision of hypertuned CT model is - 0.6
Recall of hypertuned CT model is - 0.6470588235294118
F1 score of hypertuned CT model is - 0.6226415094339622
```

The accuracy of the model has increased to **77**%. Precision & Recall to **60**% and **64.5**% respectively. Thus the f1 score is also at **62.2**%.



The **AUC** is **78.2%.** The model is able to classify between the 2 classes with **78.2**% accuracy. This model is also correctly predicting the **TP's** and **TN's** i.e. the potential defaulters and non-defaulters are accurately predicted.

Next, we'll retain the hyperparameters but will change the criteria of checking split quality.

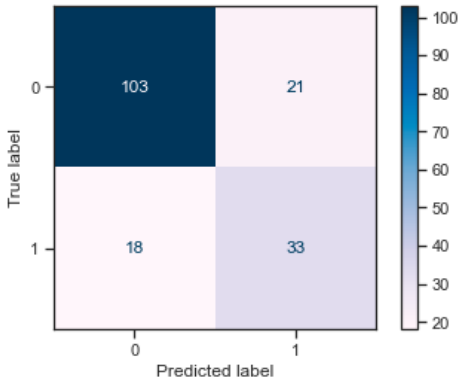
Hyperparameters tuning - Decision Tree (Gini)

```
ct3 = DecisionTreeClassifier(criterion="gini", max_depth=4, min_samples_leaf=2, min_samples_split=3)
ct3.fit(X_train.sm, y_train.sm)
```

```
DecisionTreeClassifier(max_depth=4, min_samples_leaf=2, min_samples_split=3)
```

```
pred_CT3 = ct3.predict(X_test)
```

Here, we've specified the criteria to be as '**Gini**', which is also default by the way and retained the other hyperparameters from the model above.

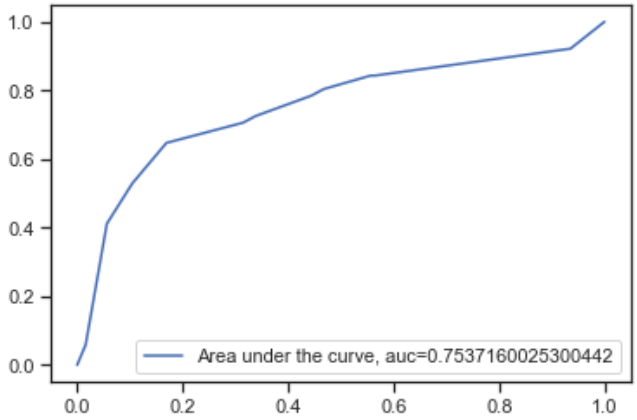


The matrix looks quite identical to what we observed with '**entropy**' as the criteria. This also means that the information gain is identical with both criterion and the classification is also accurate.

```
print("Accuracy of hypertuned CT model is -",metrics.accuracy_score(y_test, pred_CT3))
print("Precision of hypertuned CT model is -",metrics.precision_score(y_test, pred_CT3))
print("Recall of hypertuned CT model is -",metrics.recall_score(y_test, pred_CT3))
print("F1 score of hypertuned CT model is -",metrics.f1_score(y_test, pred_CT3))
```

```
Accuracy of hypertuned CT model is - 0.7771428571428571
Precision of hypertuned CT model is - 0.6111111111111112
Recall of hypertuned CT model is - 0.6470588235294118
F1 score of hypertuned CT model is - 0.6285714285714287
```

We see an identical accuracy, a slight increase in **Precision** and **Recall** and thus in **f1 score**.



The **AUC is also 75.3%,** which is predicting accurately the positive and negative classes.

<u>Random Forest:</u>

Finally the last model is also an ensemble model – Random Forest.

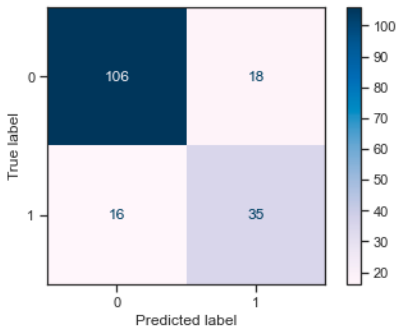
Random Forest

```
rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train.sm, y_train.sm)
```

```
RandomForestClassifier()
```

```
pred_RF = rf.predict(X_test)
```

Let's take a look at the confusion matrix



The accurate prediction of **TN's** & **TP's** is very high. The classification error is also the lowest. Even though it is still a naïve model, the initial parameters seem promising.

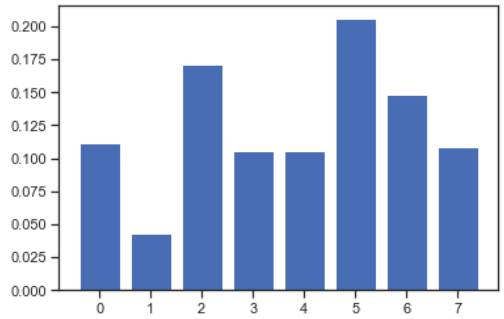
```
print("Accuracy of RF model is -",metrics.accuracy_score(y_test, pred_RF))
print("Precision of RF model is -",metrics.precision_score(y_test, pred_RF))
print("Recall of RF model is -",metrics.recall_score(y_test, pred_RF))
print("F1 score of RF model is -",metrics.f1_score(y_test, pred_RF))
```

```
Accuracy of RF model is - 0.8057142857142857
Precision of RF model is - 0.660377358490566
Recall of RF model is - 0.6862745098039216
F1 score of RF model is - 0.6730769230769231
```

The accuracy of the model is **80.5**% | Precision of the model – **66**% | Recall is **68**% | f1 score is **67**%

The metrics seem promising, but we'll explore if there is a scope for improvement when we tune the hyperparameters. First let us check the variable importance.

```
plt.bar([x for x in range(len(importance_RF))], importance_RF)
plt.show()
```



The X-axis lists the 8 predictor variables, the label with index **5 is 'debtinc'** i.e. debt to income ratio and **6** is '**creddebt**' i.e. debt to credit ratio & **2** is '**employ**' i.e. no. of years of employment are listed the most important variables contributing to our prediction.

Now, we'll only model with 3 most important predictors.

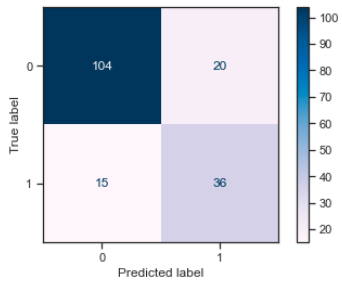
Hyperparameter tuning - Random Forest

```
rf1 = RandomForestClassifier(n_estimators=100, max_depth = 6, max_features = 3)
rf1.fit(X_train.sm, y_train.sm)
```

```
RandomForestClassifier(max_depth=6, max_features=3)
```

```
pred_RF1 = rf1.predict(X_test)
```

Let us check the confusion matrix



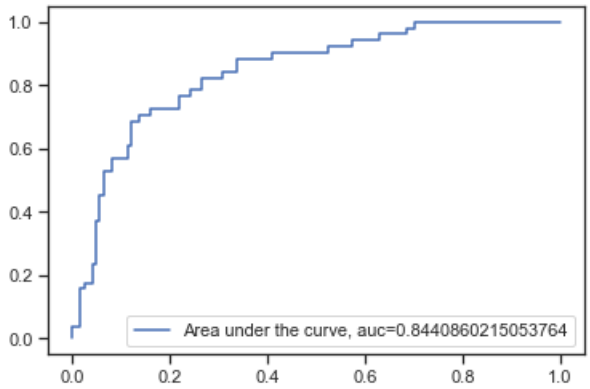
The accurate prediction of **TN's** & **TP's** is still impressive. The classification error for **FP's** has reduced but overcompensated a bit for **FN's**. Let's verify our assumptions:

```
print("Accuracy of hyperparameter tuned RF model is -",metrics.accuracy_score(y_test, pred_RF1))
print("Precision of hyperparameter tuned RF model is -",metrics.precision_score(y_test, pred_RF1))
print("Recall of RF hyperparameter tuned model is -",metrics.recall_score(y_test, pred_RF1))
print("F1 score of hyperparameter tuned RF model is -",metrics.f1_score(y_test, pred_RF1))
```

```
Accuracy of hyperparameter tuned RF model is - 0.8
Precision of hyperparameter tuned RF model is - 0.6428571428571429
Recall of RF hyperparameter tuned model is - 0.7058823529411765
F1 score of hyperparameter tuned RF model is - 0.6728971962616823
```

Accuracy is **80**% | Precision is **64**% | Recall is **70.5**% | F1 score is **67**%

The recall has improved. Which means that the **model is now picking out the defaulters even more accurately without compromising the non-defaulters**. Which is the main objective of the project.



Finally, the **AUC is 84.4**%, which is mighty impressive. The model classifies extremely well. After successfully running & testing models, evaluating them on given metrics, it is time to summarize the models with their optimal model performance.

```
summary
```

	Accuracy	Error	Precision	Recall	F1 Score
Logistic Regression	0.789	0.211	0.606	0.784	0.684
K-Nearest Neighbor	0.766	0.234	0.589	0.647	0.617
Naive Bayes	0.783	0.217	0.594	0.804	0.683
Classification Tree	0.777	0.223	0.611	0.647	0.629
Random Forest	0.800	0.200	0.643	0.706	0.673

After observing the summary table, it is easy to distinguish based on categories.

- If we only were to consider accuracy – **Random Forest** is the best model.
- If we only were to consider precision – **Random Forest** is the best model.
- If we only were to consider recall – **Naïve Bayes** is the best model.
- If we only were to consider f1 score – **Naïve Bayes** is the best model.

However the domain of this problem is **Finance,** and the problem we're trying to solve requires:

- **Correctly** identifying the potential defaulters from the applicants.
- **Correctly** identifying the potential no-defaulters from the applicants.
- **Avoiding incorrect** predictions of non-defaulters as defaulters.

- **Avoiding incorrect** predictions of defaulters as non-defaulters.

Therefore, the metrics we should tilt our focus on true positives & true negatives.

The True Negative Rates (**TNR**) for all models are below:

LR – **79.07**% | KNN - **81.45**% | NB - **77.41**% | CT - **83.06**% | RF - **83.87**%

If a balance is to be considered, **Naïve Bayes** model outperforms all the others.

----------End----------

Instructions: Run the *.rmd & *.ipynb files.

References:

Wikipedia – Confusion Matrix
Stack overflow – Programming doubts