**FM9528A Banking Analytics**

**Credit Risk Analytics**

**Student ID: 251139213**

**Word Count: 2200**

**Data Preprocessing**

We analyzed the data and performed the below steps to clean our data for better performance-

1. Some variables are not useful in creating our Credit Risk analysis model as they have the following issues-
   1. The data in all the records are common and don’t have any significance, so removing them from our data
      1. CLERK\_TYPE - All values are C
      2. QUANT\_ADDITIONAL\_CARDS,EDUCATION\_LEVEL,FLAG\_HOME\_ADDRESS\_DOCUMENT,FLAG\_RG,FLAG\_CPF,FLAG\_INCOME\_PROOF - All values are 0
      3. FLAG\_MOBILE\_PHONE,FLAG\_ACSP\_RECORD - All are N
   2. Variables are unethical and doesn’t impact the Credit Risk Analysis
      1. SEX - Loan can’t be provided on the basis of gender or identity
      2. ID\_CLIENT - It's a randomly assigned unique key for a record
      3. PAYMENT\_DAY - For every individual the bill statement is issued at different dates, so the variable doesn’t capture any useful data, had it been a data mentioning that the individual pays the bill on time or not, we would have considered it
      4. APPLICATION\_SUBMISSION\_TYPE - Loan approval can’t be affected on the basis of mode of application
      5. STATE\_OF\_BIRTH and CITY\_OF\_BIRTH - Birth location doesn’t impact the loan application but only the current residence or situation of a person might do
   3. Variables are correlated
      1. QUANT\_CARS itself doesn’t impact the model but its value as an asset does, which is covered in the variable - PERSONAL\_ASSETS\_VALUE
      2. RESIDENCIAL\_STATE, RESIDENCIAL\_CITY,RESIDENCIAL\_BOROUGH,RESIDENCIAL\_ PHONE\_AREA\_CODE denotes the location type –upscale or not, the better financial status of the person will make him a good candidate for loan approval, we can get that value from the variable - RESIDENCIAL\_ZIP\_3
      3. Similarly, PROFESSIONAL\_STATE,PROFESSIONAL\_CITY,PROFESSIONAL\_BOROUGH,PROFESSIONAL\_PHONE\_AREA\_CODE are covered by - PROFESSIONAL\_ZIP\_3
      4. OCCUPATION\_TYPE - will be covered under PROFESSION\_CODE
2. Replaced all empty values with Null and made data categorical where required for data cleaning process
3. Deleted duplicate records
4. Data Visualization and analysis - (data treatment for Nulls and Outliers)
   1. Null Values treated for -
      1. Replaced by Median / Mode of train dataset to capture all possible scenarios-RESIDENCE\_TYPE,MONTHS\_IN\_RESIDENCE, PROFESSION\_CODE
      2. If the applicant has no spouse; or spouse has no job or no education; or the applicant didn't share the data, we don't know the encoding, so we can't split the above scenarios on the basis of any assumption. So, replacing Null values by creating another category, say 20 covering the above scenarios for-MATE\_PROFESSION\_CODE,EDUCATION\_LEVEL
   2. For other variables can’t remove the data on any inference even though some category values are < 1% or data is not clean for -
      1. It could be the case that some applicants might be working part time or on hourly basis so we can't clearly define a cut off point for truncating the income data
   3. Removing and replacing extreme/ wrong values with Median/Mode to capture information from all records -
      1. QUANT\_DEPENDENTS - There can’t be 53 dependents in real life
      2. RESIDENCIAL\_ZIP\_3, PROFESSIONAL\_ZIP\_3 has #DIV/0! - wrong value
   4. Dropping the irrelevant values -
      1. AGE - Given the standard norms for applying the loan one should be between 18- 65 years old but for many banks the limits may vary. Average life expectancy in Brazil in 2007 was 73[[1]](#footnote-1) and around maximum 80[[2]](#footnote-2) in the world, so taking the 18-80 as limits for our model to capture the worst possible scenarios.

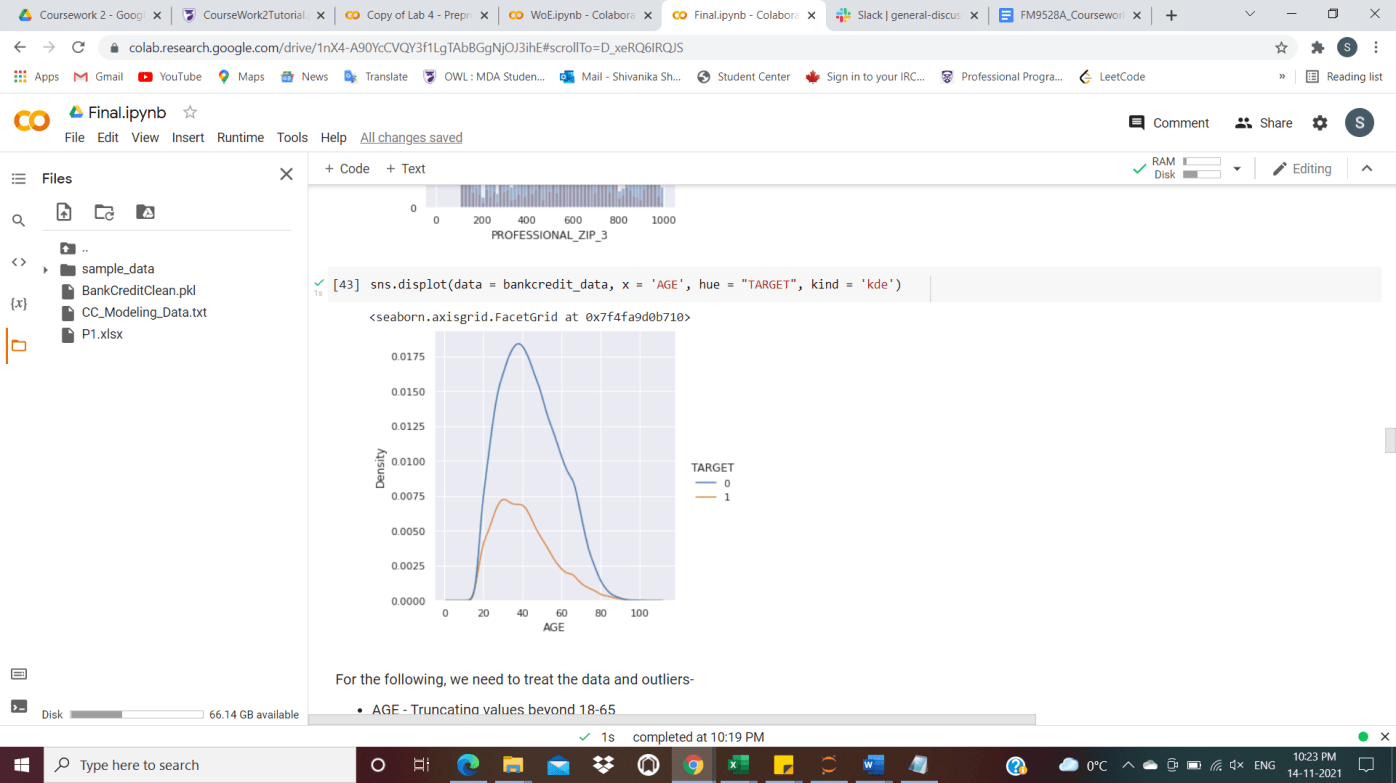
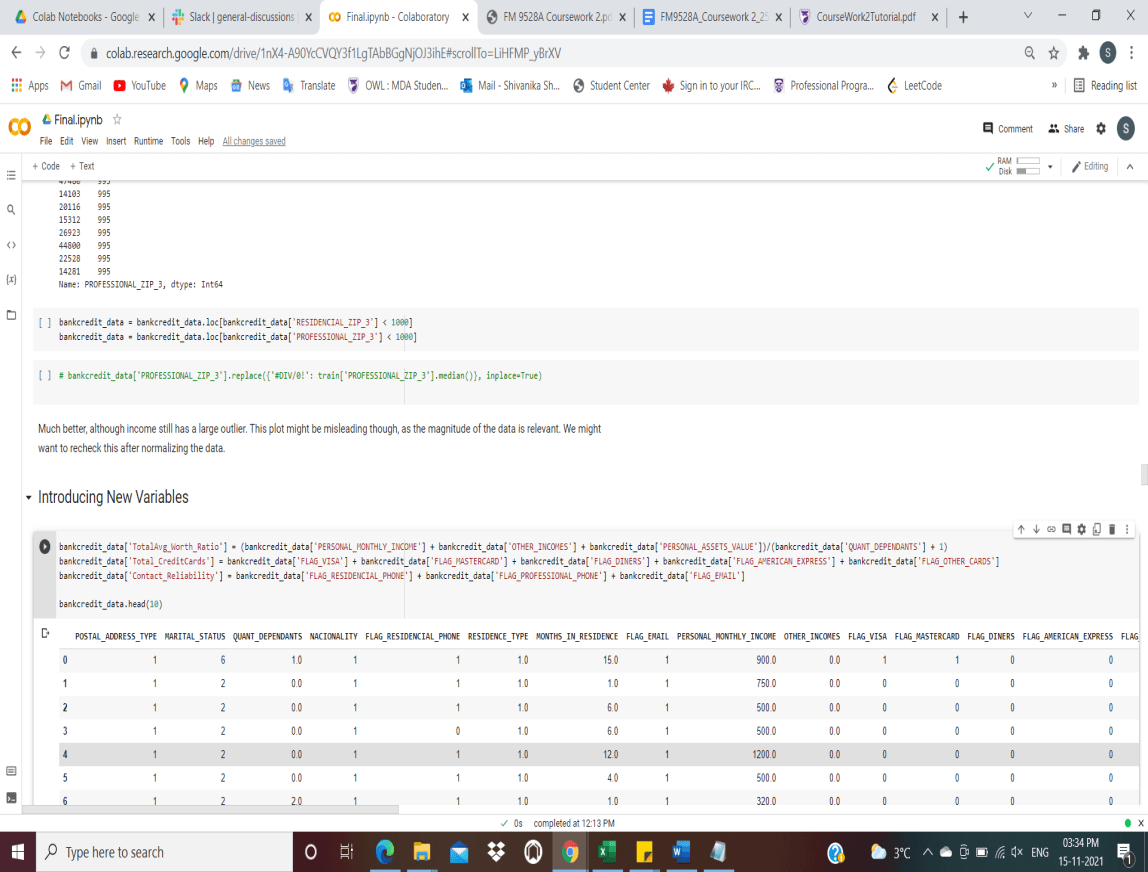


Figure 1: Visualization of Age data

1. Introducing New Variables -
   1. TotalAvg\_Worth\_Ratio - It gives the net financial worth of an applicant considering the fact if he/she has to take care of dependents, this helps us to analyze the standard worth of an applicant - higher the value, better the candidate for loan
   2. Total\_CreditCards - This gives the total credit cards one might have. Having too many credit cards can be seen as a potential risk
   3. Contact\_Reliability - By this we try to capture the reliability of a person i.e if we have enough data to contact/ trace the candidate or not. This covers all the contact variables in the dataset and can help in analyzing the effects in credit risk model



**Weight of Evidence - WoE**

The weight of evidence tells the predictive capacity of an independent variable in relation to the dependent variable.

Parameters selected-

1. The fewer bins, the more smoothing and it captures important patterns in the data, while leaving out noise. Checked with #bins as 5, 8, 10 and 12 and #bins = 8 gave optimal results
2. Bins with less than 5% cases might not be a true picture of the data distribution and lead to model instability; any value greater than that may lead to leaving out any minute information
3. For such a smaller dataset, 1% of initial cuts makes sense and gives better results
4. We use trees sequentially given the constraints we decide for better performance
5. Stop limit is set to 0.001 to capture the least possible information value in our dataset as the data is challenging

We further manually adjust the bins to capture the real trend between the variables and target.

For categorical variables, bins aren’t adjusted and the trend can be seen by the graphs.

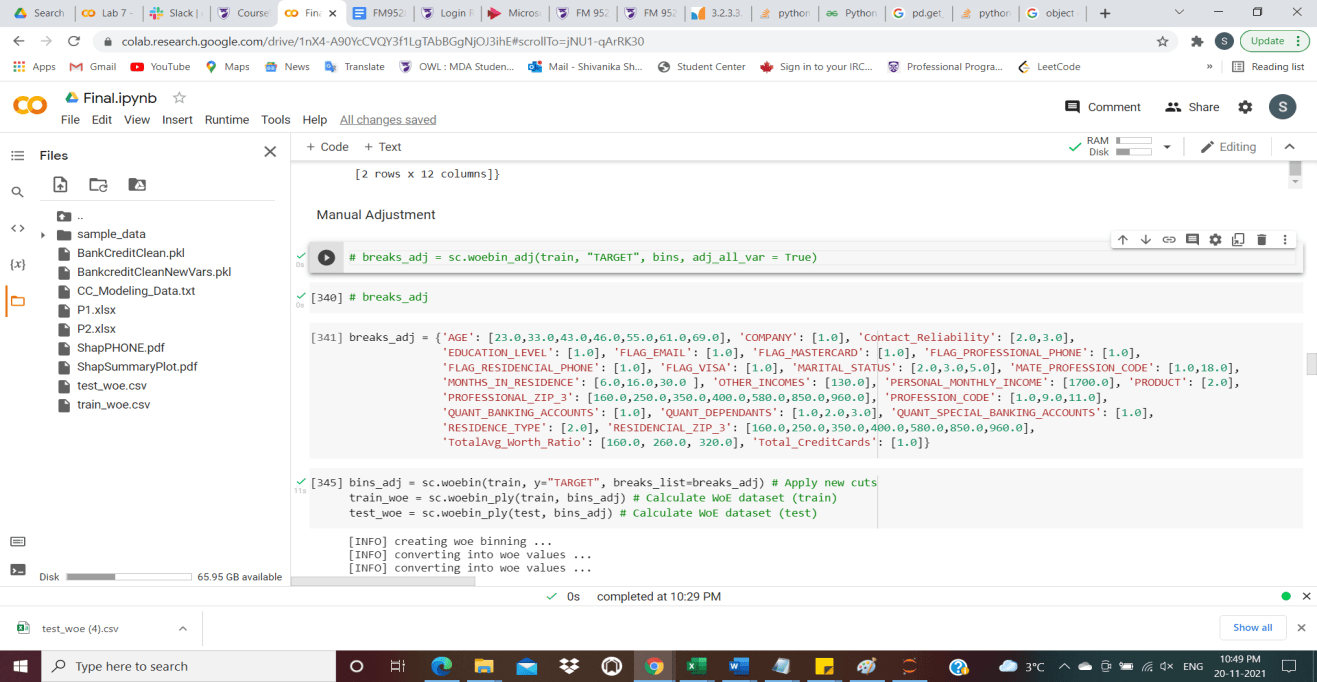
For some variables, the probability of default is low or high depending on the trend being decreasing or increasing. The bins are adjusted to make the trend reasonable. For eg: Higher the total average worth ratio, higher the financial stability of an individual, thus lower the chances of default which is explained by a decreasing trend.



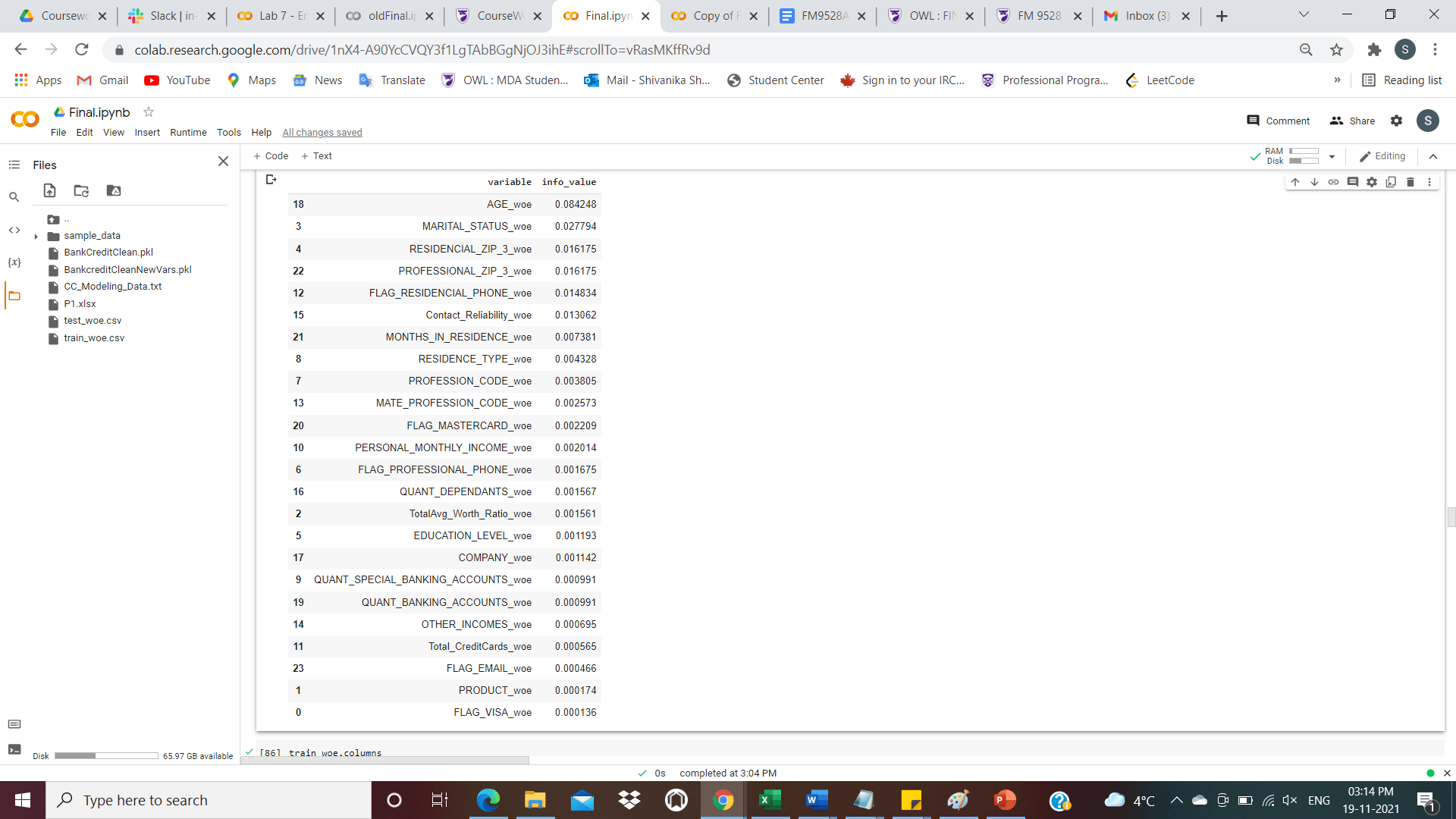
For Age -The graph is explainable as higher the age, better financial stability of a person and lower chances of default but an increasing trend towards the end can be seen maybe because as in a developing country[[3]](#footnote-3) the retirement pensions are low thus lower stability and higher chance of default.



Figure 2: Bins Adjustment

****

Information value is one of the most useful techniques to select important variables in a predictive model. It helps to rank variables on the basis of their importance as below -

****

1. For now, we continue with all the above variables as they all are relatively less predictive and we can’t choose a standard cut off
2. We remove below variables as they have Information\_value=0 and only a single bin which is not useful for our model -

* NACIONALITY,MONTHS\_IN\_THE\_JOB,FLAG\_DINERS,FLAG\_AMERICAN\_EXPRESS,FLAG\_OTHER\_CARDS,PERSONAL\_ASSETS\_VALUE

1. We later apply lasso/ ridge regularization on our model to select the appropriate variables

Further, we check for correlations among our variables and remove them to find our optimal model-

1. QUANT\_SPECIAL\_BANKING\_ACCOUNTS and QUANT\_BANKING\_ACCOUNTS
2. PROFESSIONAL\_ZIP\_3 and RESIDENCIAL\_ZIP\_3
3. The other correlations aren’t that significant so not removing them

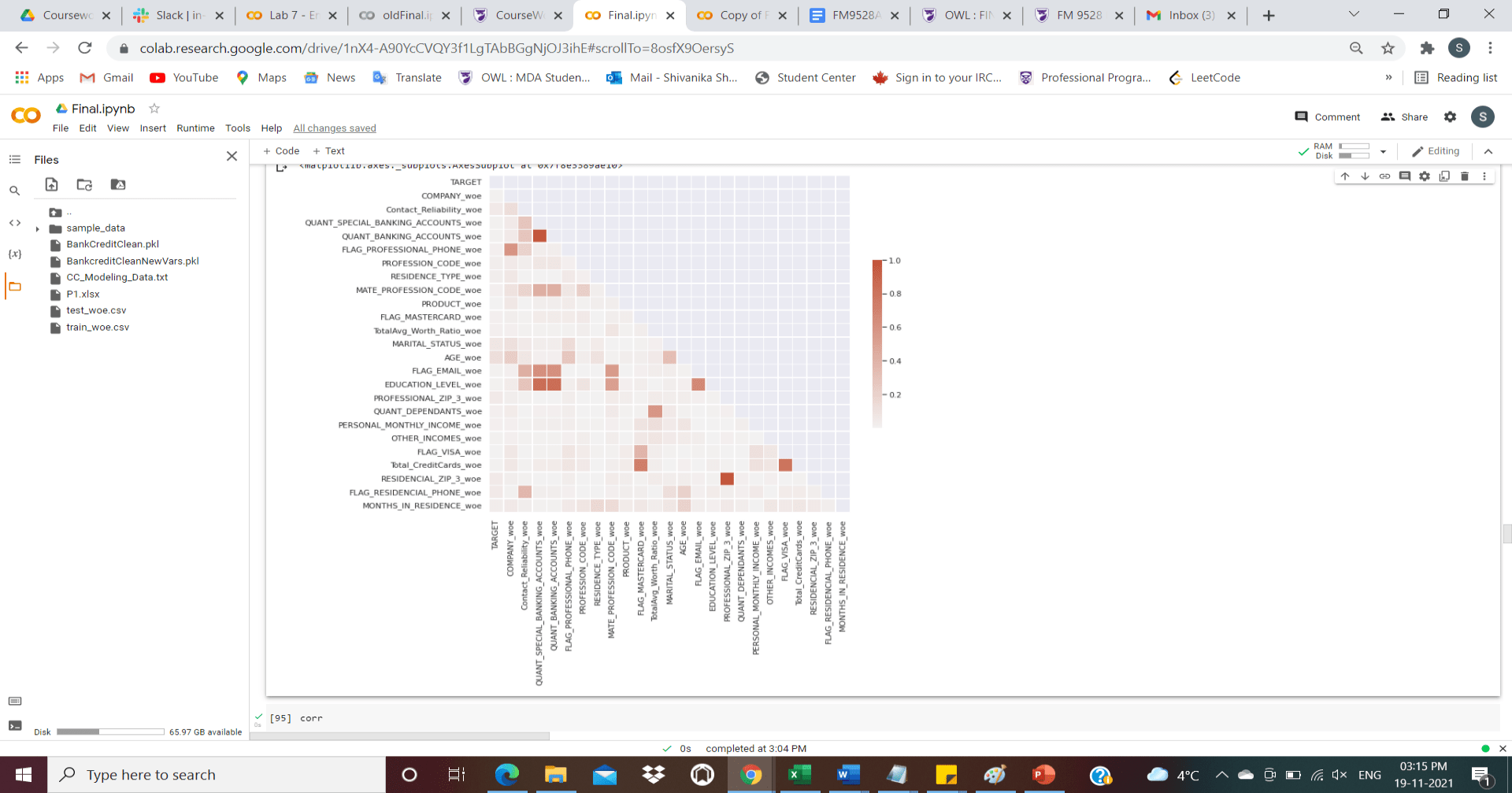
****

Figure 3: Correlation Graph

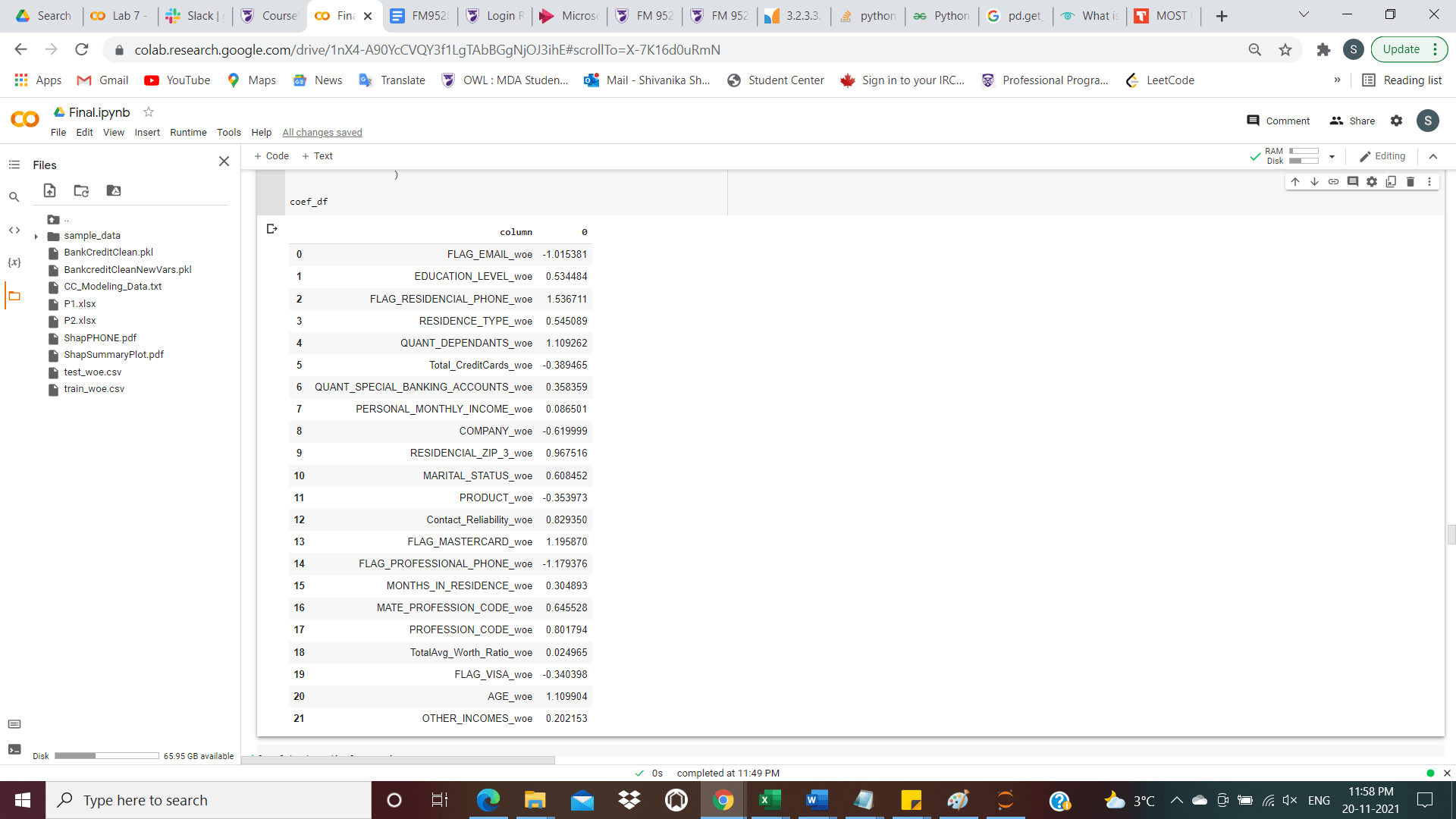
**Logistic Regression and Scorecard Construction**

The logistic model is used to predict the likelihood of a specific class based on prior dataset observations.

Parameters selected -

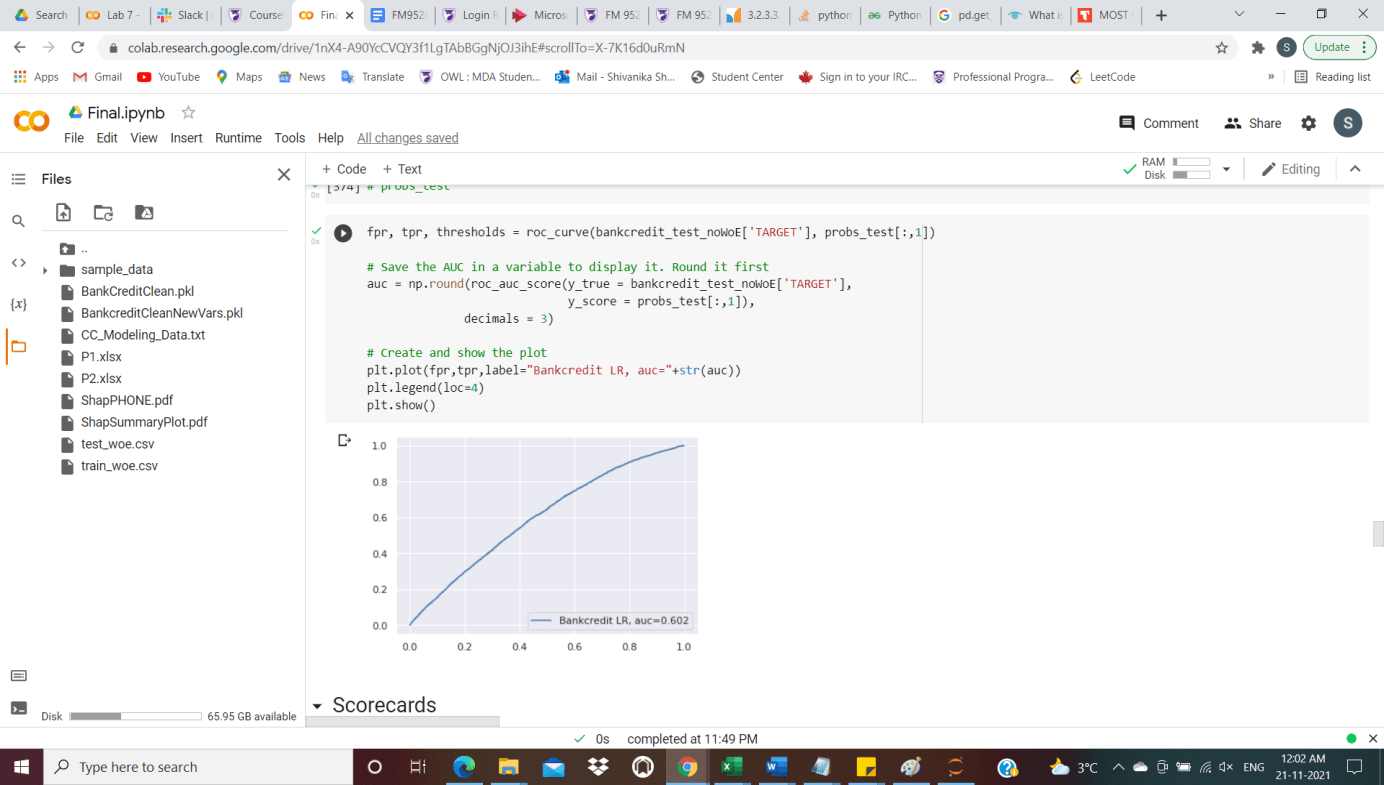
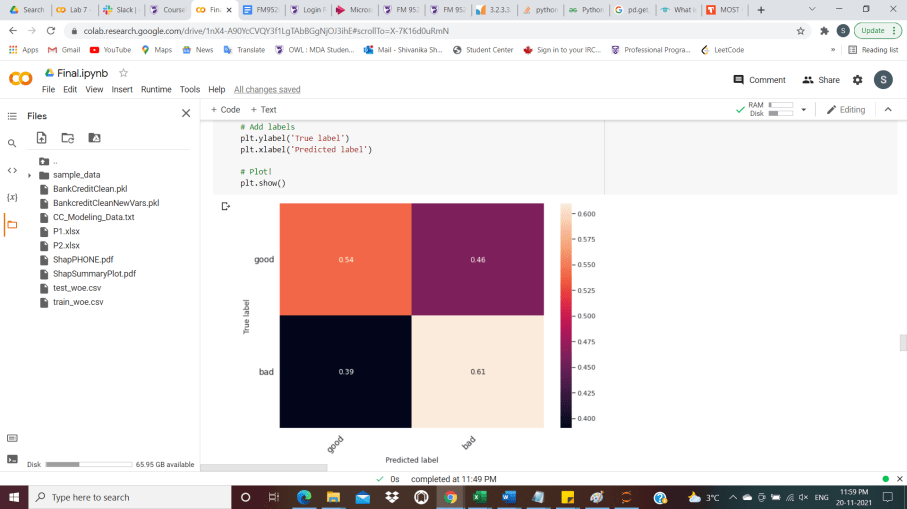
1. For such a small dataset, cv folds = 3 gives optimal results
2. As our data is challenging, after applying elastic-net the model eliminates all the variables so we analyze that ridge gives better results here as it minimizes the variables coefficients and doesn’t eliminate them completely
3. After trying with various max\_iterations parameters, 100 gives favorable results
4. To keep our model balanced we set class\_weights = balanced
5. Other parameters, we keep as generally used to enhance performance of our model
   1. Cs = 10 : tried with parameters= 5, 7,8 and 10, all gave the same results
   2. tol=0.000001, as the parameters are less significant
   3. fit\_intercept=True, for training our model
   4. solver = 'saga', to optimize the model
   5. n\_jobs = 2, to run parallelly
   6. refit = True, to retrain with the best parameter and all data after finishing for better results.

Coefficients value after R2 penalty-



The negative coefficients represent those variables whose significance is very less.

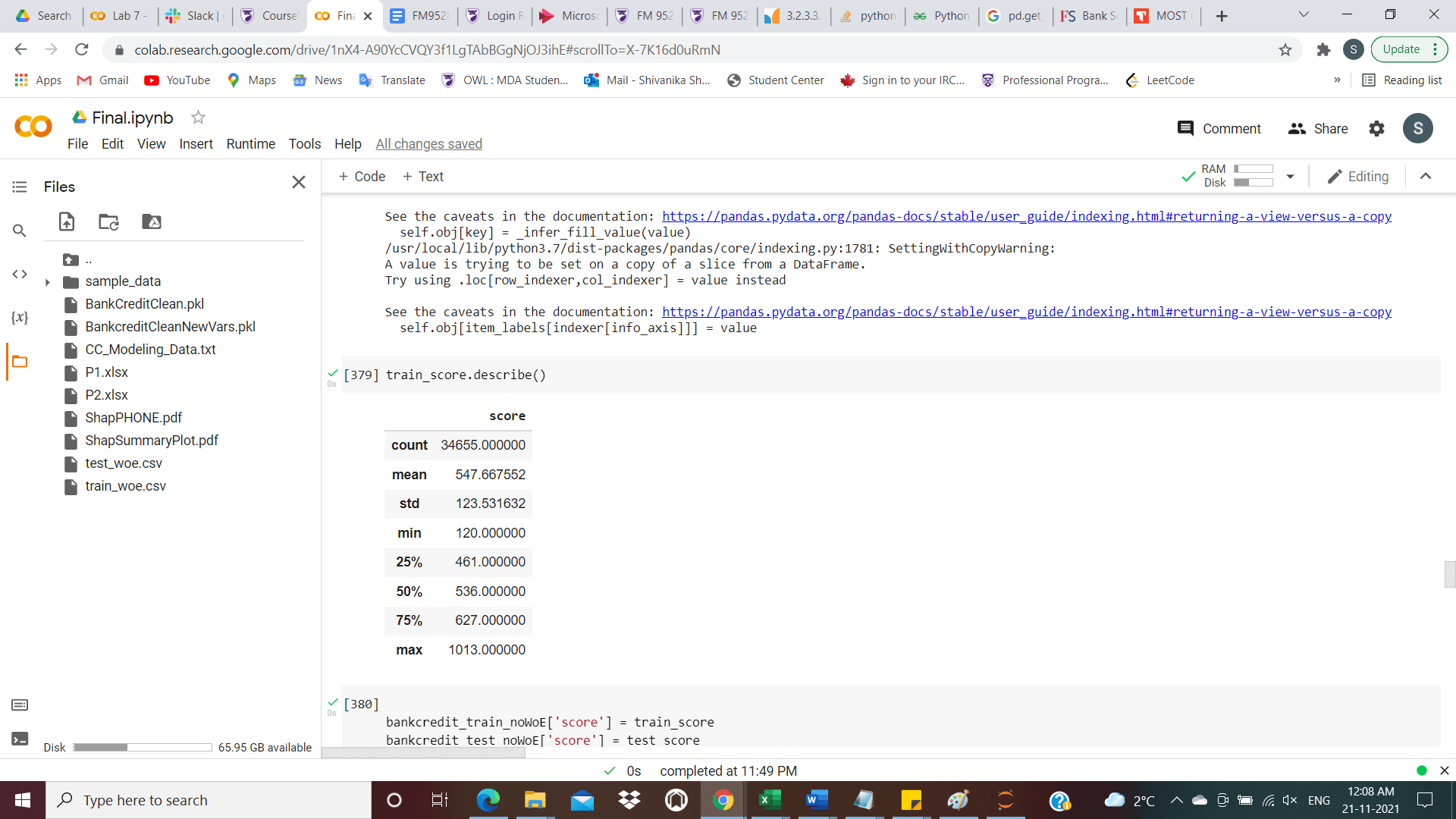
Confusion Matrix and Accuracy for describing performance of our classification model -



Our model gives a 60.2% accuracy along with correctly predicting 54% of customers as good and 61% as defaulters.

Credit risk analysts use the Bank Scorecard to provide consistent standalone credit scores that reflect the underlying creditworthiness of all banks in their portfolio, whether rated or unrated.

After trying various other parameters, we take base points as 1200, odds as 0:1 and points to double the odds = 200 as they represent our data and scorecard range well.



For our model we can further remove the variables whose predictive power is less than 0.001, so that we avoid the chances of overfitting.

**Random Forest**

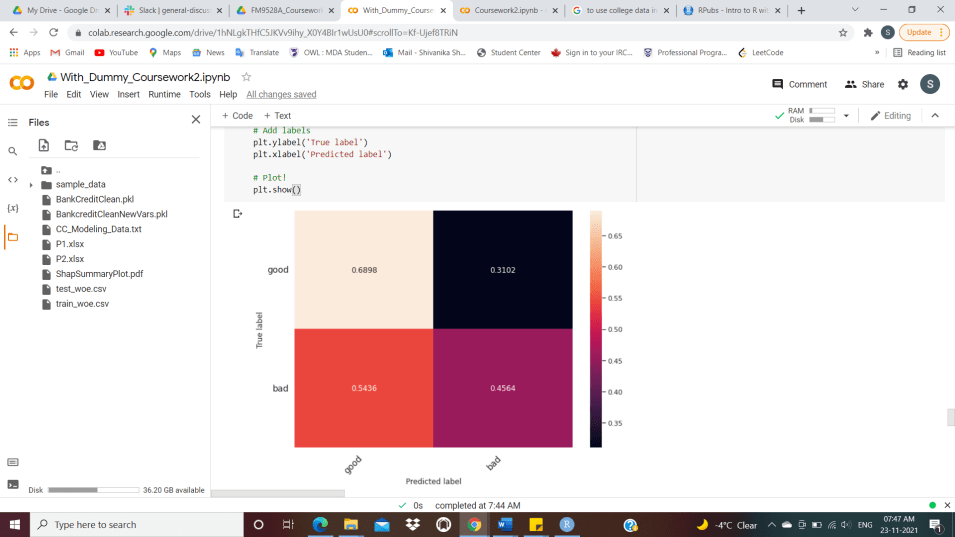
We tried with and without splitting data as one hot encoding but it gives similar results.

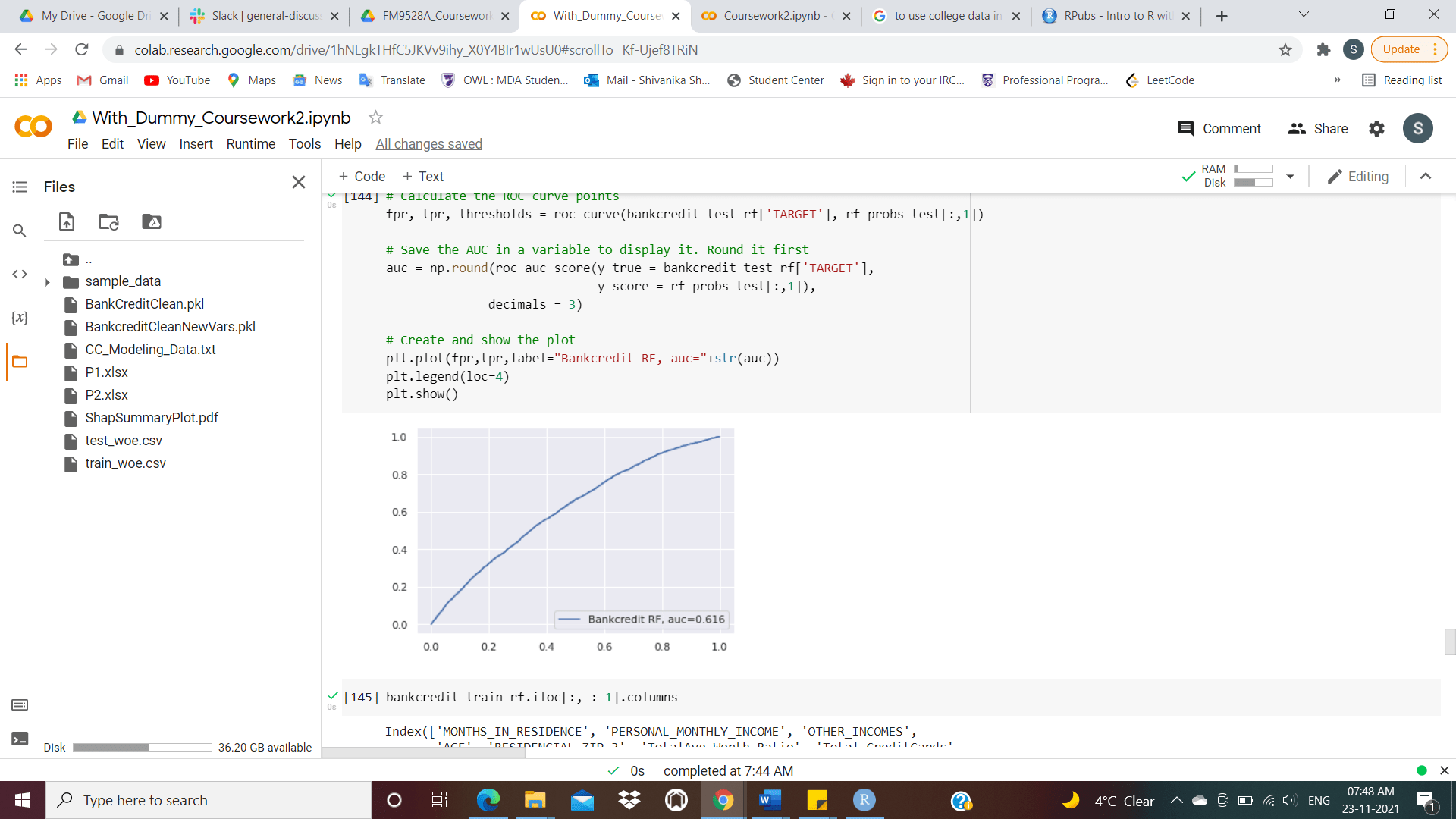
Random forests are a classification and regression ensemble learning method that works by creating numerous decision trees at training time to avoid overfitting to the training set.

Parameters selected-

1. After trying with various #estimators, 1000 gives optimal accuracy
2. Tried v/3 as max\_features but sqrt(v) i.e max\_features=None gives better results
3. We select other parameters as to make use of advantages of random forest features to increase performance of our model and keep it balanced
   1. criterion='entropy', to train the trees and not gini as we don’t need complex computations
   2. letting the classifier find optimal results-max\_depth=None,min\_samples\_split=2,max\_leaf\_nodes=None
   3. For the given size of dataset, the below values make sense -
      1. min\_samples\_leaf=0.0001, 0.1% of sample
      2. min\_weight\_fraction\_leaf=0.0
      3. min\_impurity\_decrease=0.00001, to capture least possible value
      4. bootstrap=True, to sample with repetition
      5. warm\_start=False, to train over previously trained tree
   4. oob\_score=True, to report accuracy with non-selected cases.
   5. n\_jobs=2 for parallel processing

Confusion Matrix and Accuracy





Our model gives a 61.6% accuracy along with correctly predicting 68.98% of customers as good and 45.64% as defaulters.

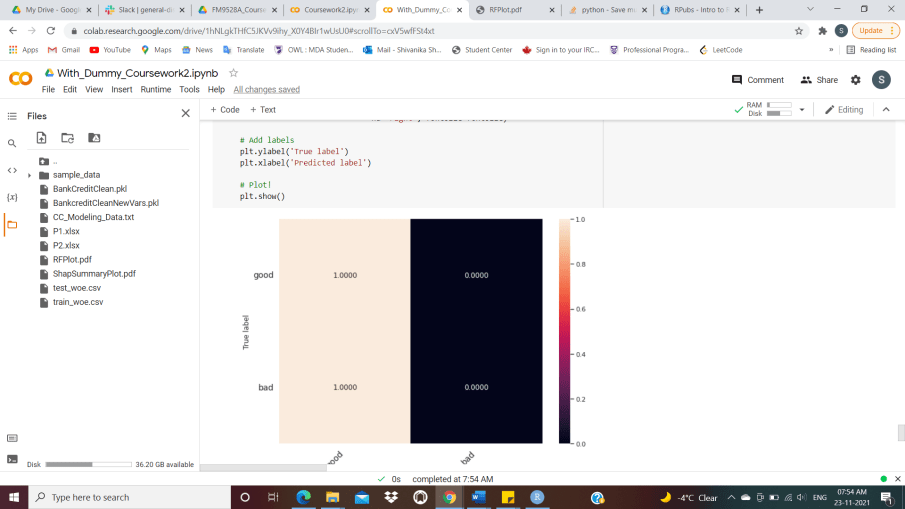
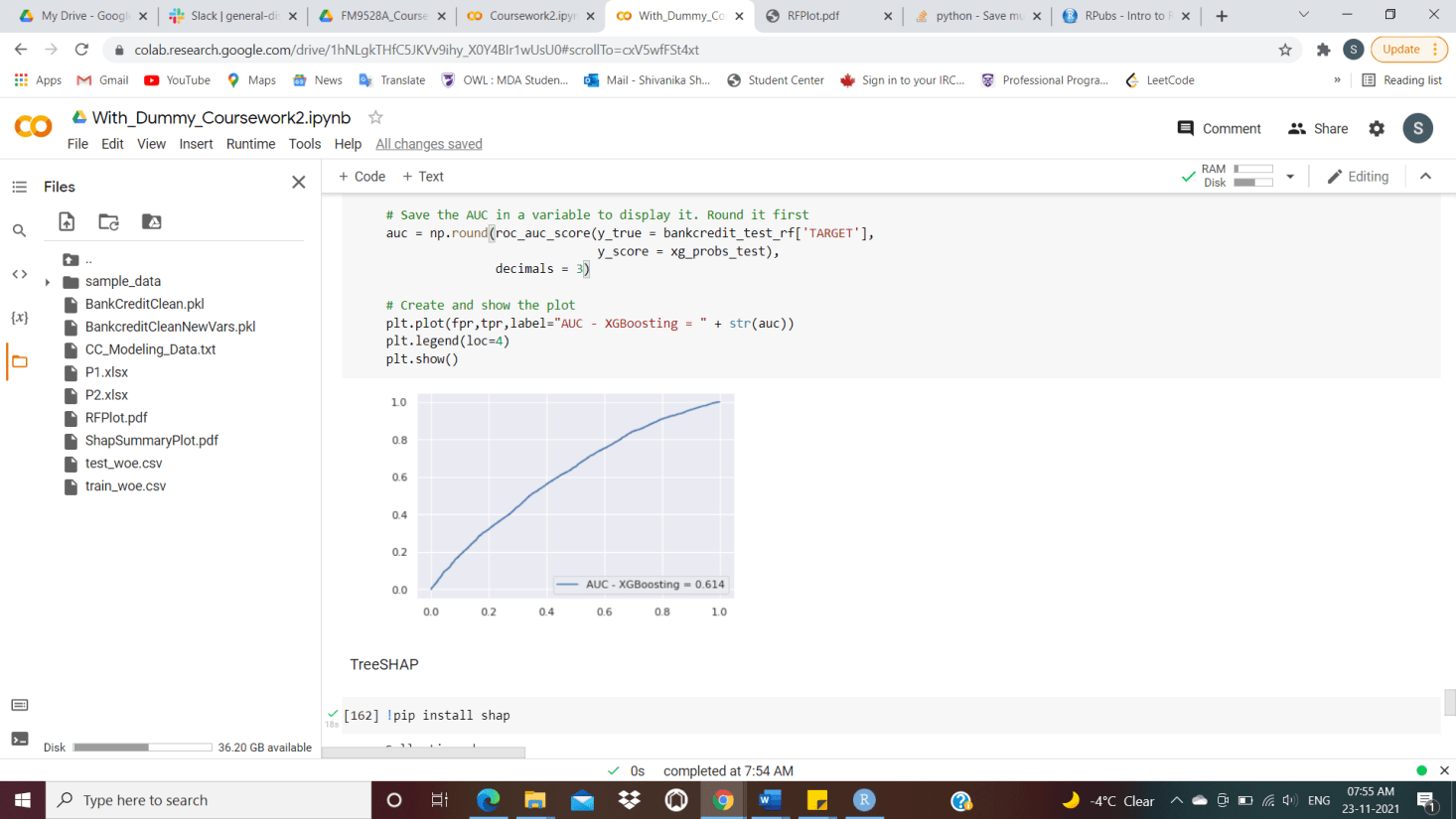
**XGBoosting**

eXtreme Gradient Boosting is a regression and classification machine learning technique that employs prediction models in the form of an ensemble of weak decision trees. These are constructed in a stage-by-stage manner, learning from previous stage failures.

Parameters selected-

1. We train only on 20% of data considering the small size of our dataset
2. max\_depth=3, learning\_rate=0.05 and n\_estimators=50 gave best auc of 0.606
   1. minimum max\_depth of 3 is used control over-fitting as higher depth will allow model to learn relations very specific to a particular sample
   2. n\_estimators need to be small because of the trade-off with the learning rate which should be high as we want to shrink error in each subsequent training
   3. We tried with different sets of parameters, but these gave the optimal results.
3. Considering general parameters to enhance the XGBoosting
   1. gamma=0.001, minimum loss reduction to control the growth of the tree
   2. subsample=0.632 which is the probability that customer does appear in the bootstrapping sample
   3. Lasso for first fit and then the ridge to avoid eliminating the variables entirely on later stage
   4. Balancing of positive and negative weights as Goods/Bads for balanced tree
   5. Tried using base\_score as 0.5, 0.6, 0.3, 0.1- all give same results
   6. missing=None, we have already treated the data
   7. tree\_method = 'hist' as its faster histogram optimized approximate greedy algorithm, tried using exact method but hist gives higher performance
   8. Used logistic regression as loss function for binary classification as it returns predicted probability, not class
   9. booster='gbtree', boosting trees in this case.
   10. n\_jobs=2 for parallel processing

Confusion Matrix and Accuracy

Our model gives a 61.4% accuracy along with correctly predicting 100% of customers as good and 0% as defaulters. This is mainly because of the poor quality of input data.

**Model Comparison: Logistic Regression, Random Forest and XGBoosting**

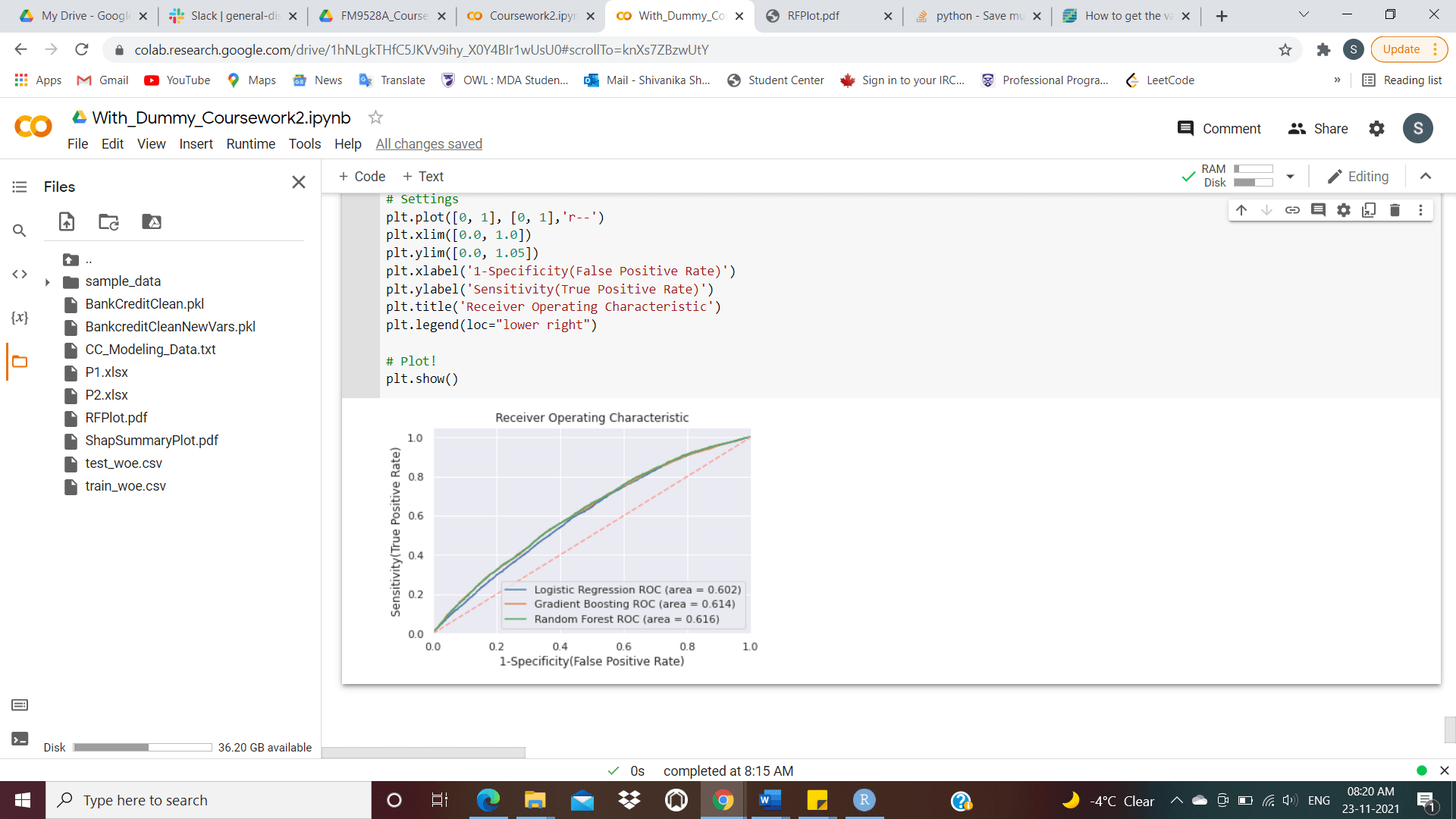


Figure 4: Different Model AUCs

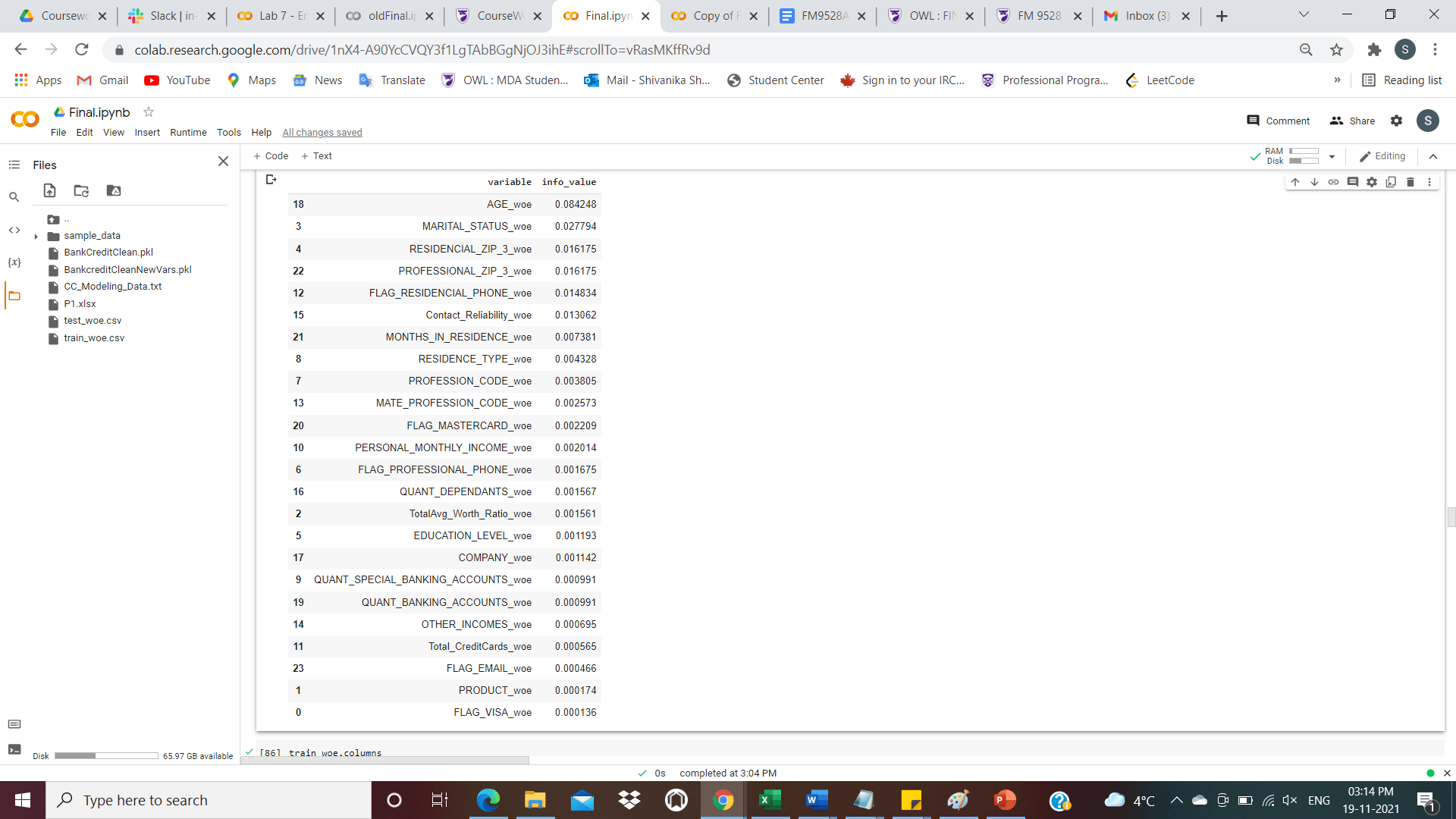
Random forest outperforms logistic regression and XGboosting with a maximum AUC of 61.6% for the supplied data and parameters.

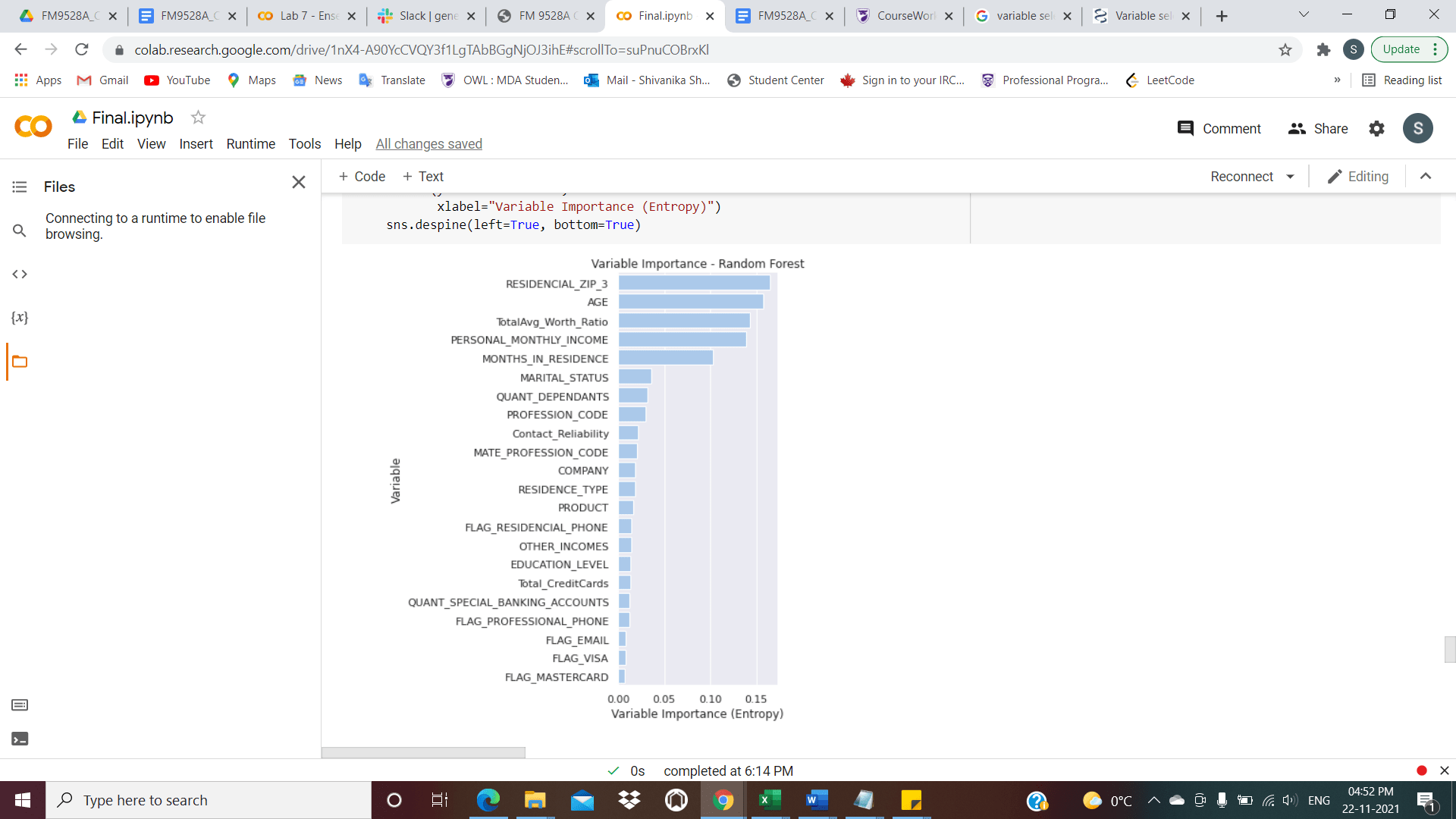
In general, XGboosting produces superior results, but in this case, the model predicts all defaulters incorrectly and performs poorly maybe because it’s not ideal for relatively small training sets. Moreover, there can be overfitting issue because of which the auc is high.

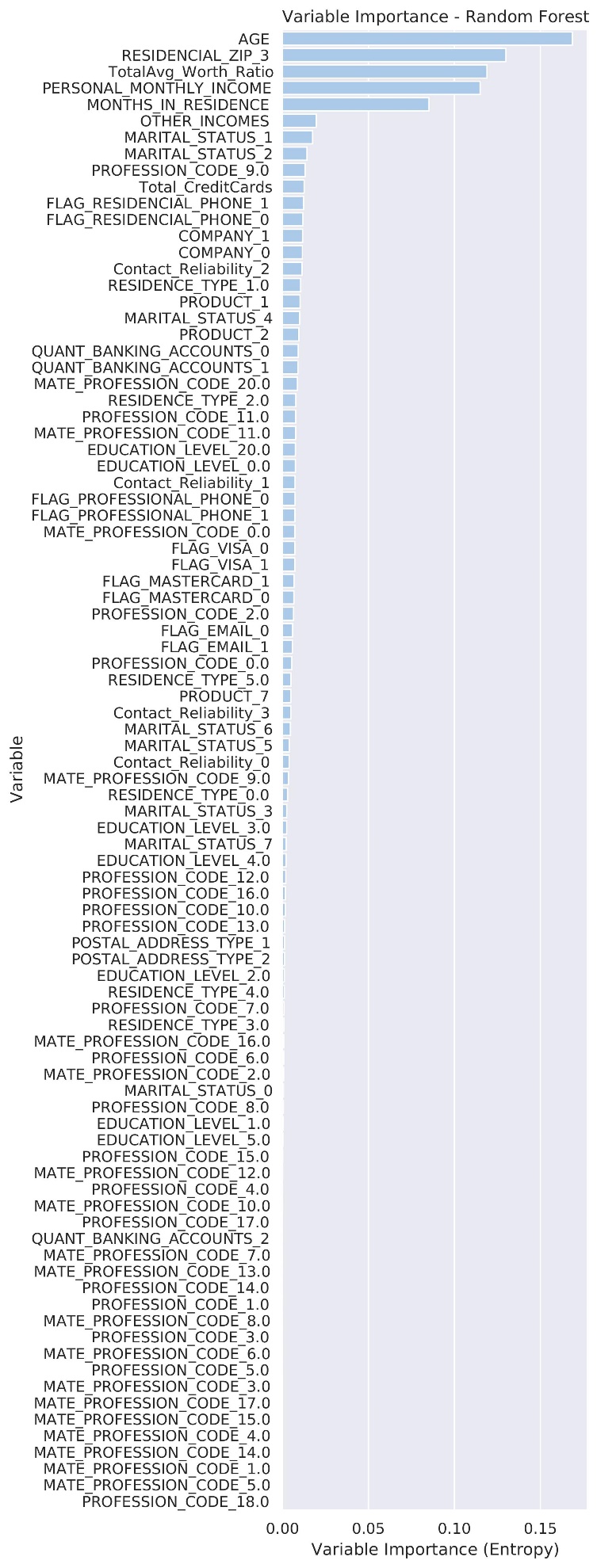
Confusion-matrices and Area-Under-Curve(AUC) are a good metric to compare models as they are aggregate measure of performance across all possible classification thresholds and important predictive analytics like recall, specificity, accuracy, and precision.

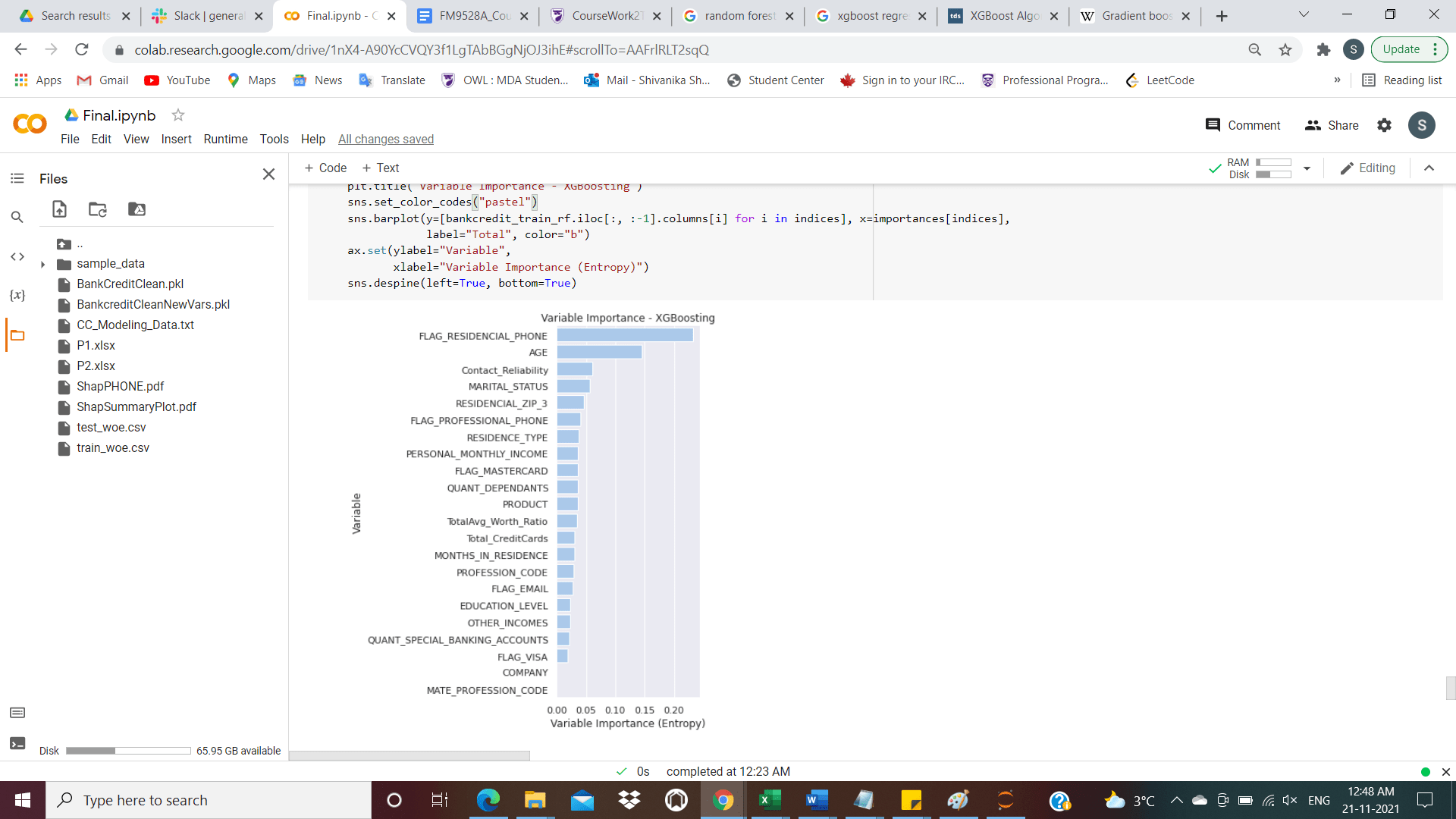
**Variable Selection**

Variable selection is the process of identifying the important features in a dataset which helps in predicting the results accurately for our model.

****

****



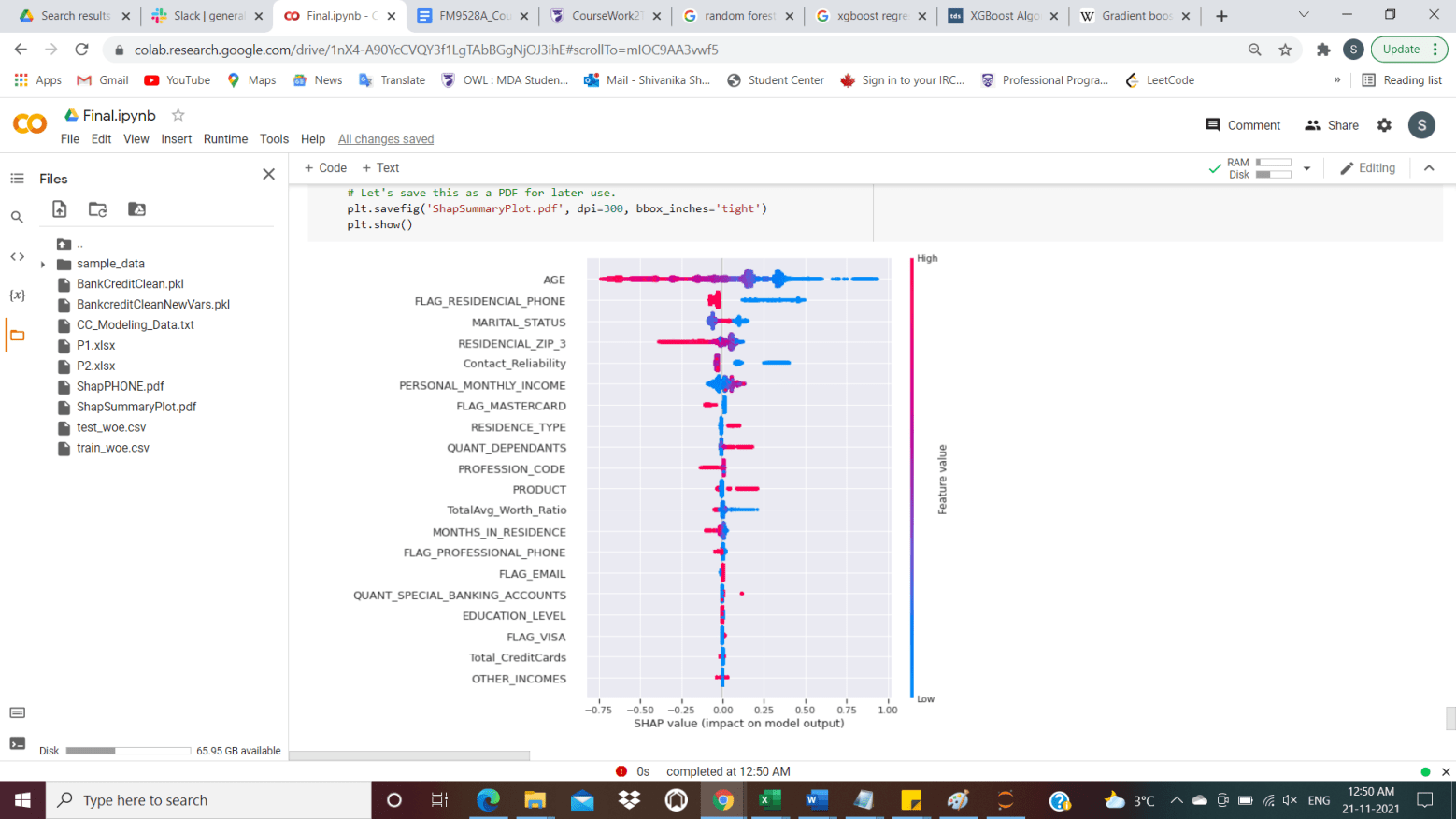
****

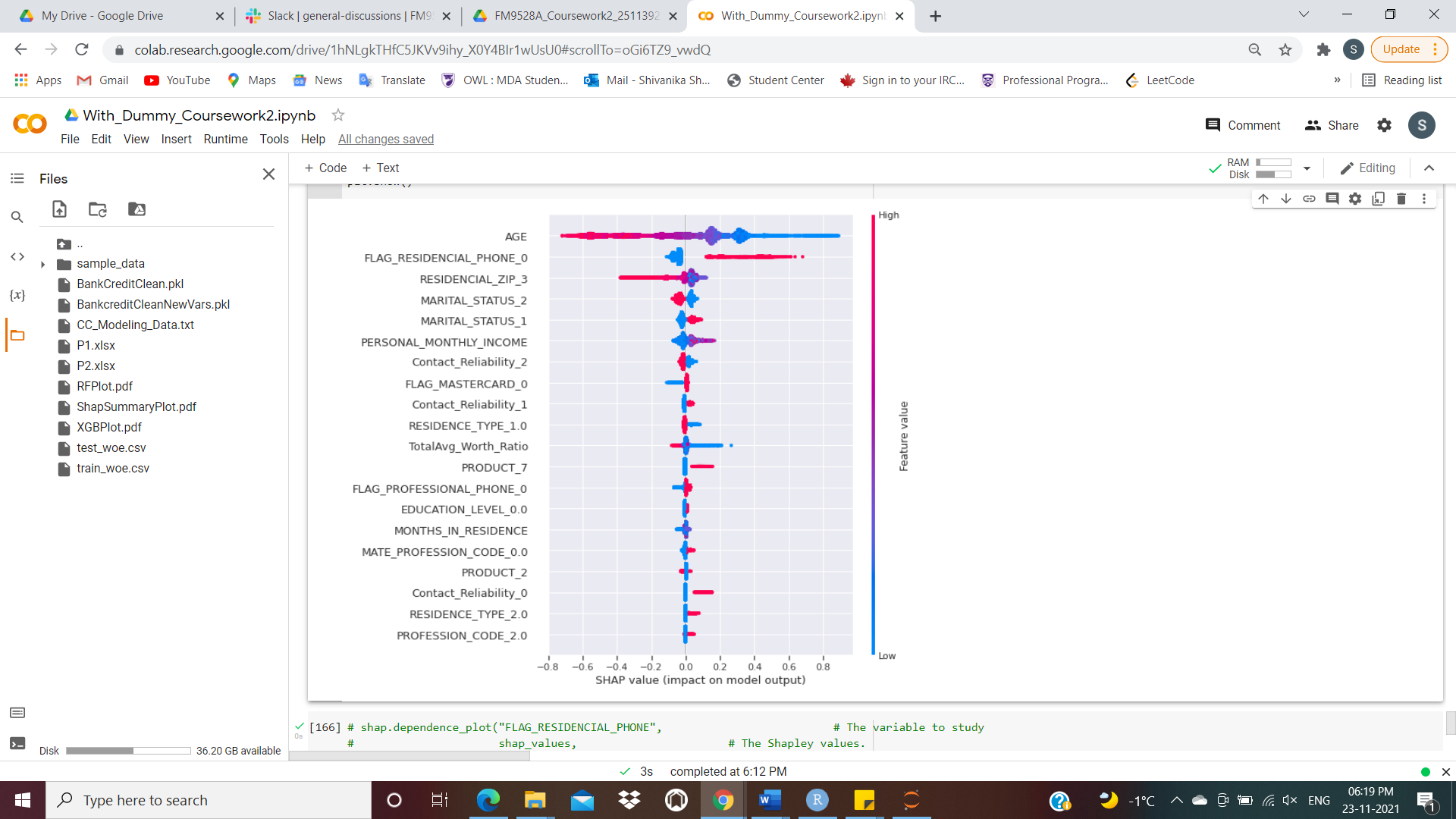


By above, we can clearly say, Age is the most significant variable.

As the models utilize different techniques, the variables chosen, differ. For logistic regression information\_value is calculated on the entire WoE dataset, Random Forest chooses variables at random, whereas Xgboosting using entropy method, chooses variables selectively.

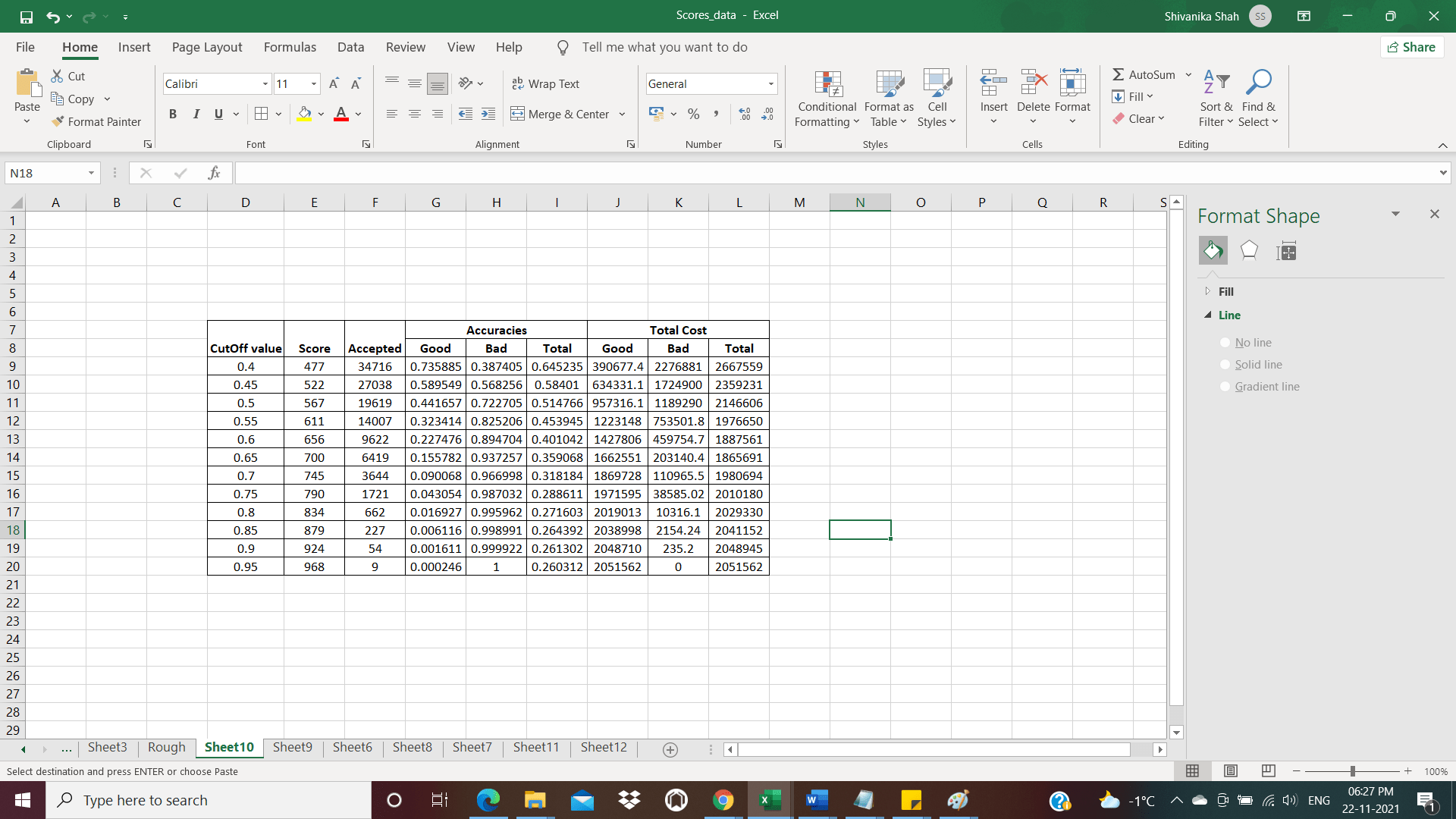
SHAP values for the variables representing their importance

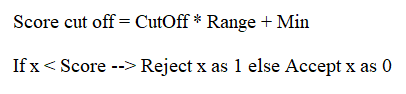


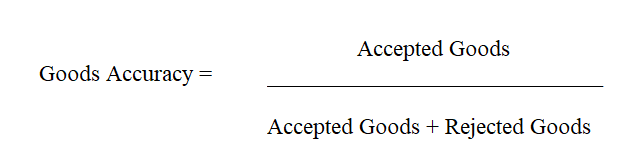


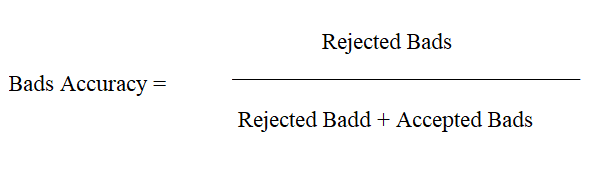
**Two-cut-off Point Strategy**

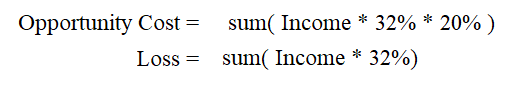
For the scorecard designed, we choose the below cut offs to capture the trend and find the accuracies of good and bad along with the total cost lost - goods as opportunity cost (which the bank could have gained) and bads as the cost lost











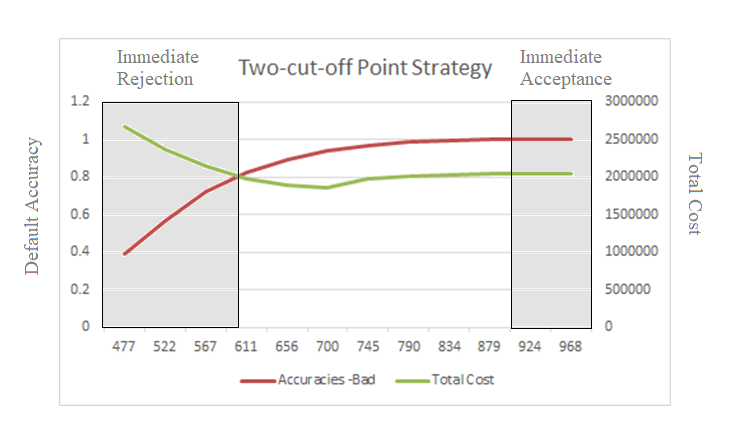


Figure 5: Two-cut-off Point Strategy plot

For scores below 576, we immediately reject the applicant, for score above 879 we immediately accept the applicant and for applicants with score between 576 and 879, we setup a committee to decide to give the loan or not, which is based on the capital in line with risks taken by the bank.

**Word Count: 2200**

**Appendix**

Jupyter Notebook Collab link –

<https://colab.research.google.com/drive/1hNLgkTHfC5JKVv9ihy_X0Y4BIr1wUsU0?usp=sharing>

Google Drive link containing all related files –

<https://drive.google.com/drive/folders/1D4EjWskXyHvoUrrtHIWryv12CH0HvItS?usp=sharing>



**!**pip install scorecardpy

In [2]:

**from** string **import** ascii\_letters

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

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

**import** math

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns

**import** scorecardpy **as** sc

**from** sklearn.model\_selection **import** train\_test\_split

**%matplotlib** inline

In [3]:

bankcredit\_data **=** pd**.**read\_csv("CC\_Modeling\_Data.txt", sep**=**'\t', engine**=**'python', header**=** **None**)

In [ ]:

bankcredit\_data**.**columns **=** [

'ID\_CLIENT',

'CLERK\_TYPE',

'PAYMENT\_DAY',

'APPLICATION\_SUBMISSION\_TYPE',

'QUANT\_ADDITIONAL\_CARDS',

'POSTAL\_ADDRESS\_TYPE',

'SEX',

'MARITAL\_STATUS',

'QUANT\_DEPENDANTS',

'EDUCATION\_LEVEL\_S',

'STATE\_OF\_BIRTH',

'CITY\_OF\_BIRTH',

'NACIONALITY',

'RESIDENCIAL\_STATE',

'RESIDENCIAL\_CITY',

'RESIDENCIAL\_BOROUGH',

'FLAG\_RESIDENCIAL\_PHONE',

'RESIDENCIAL\_PHONE\_AREA\_CODE',

'RESIDENCE\_TYPE',

'MONTHS\_IN\_RESIDENCE',

'FLAG\_MOBILE\_PHONE',

'FLAG\_EMAIL',

'PERSONAL\_MONTHLY\_INCOME',

'OTHER\_INCOMES',

'FLAG\_VISA',

'FLAG\_MASTERCARD',

'FLAG\_DINERS',

'FLAG\_AMERICAN\_EXPRESS',

'FLAG\_OTHER\_CARDS',

'QUANT\_BANKING\_ACCOUNTS',

'QUANT\_SPECIAL\_BANKING\_ACCOUNTS',

'PERSONAL\_ASSETS\_VALUE',

'QUANT\_CARS',

'COMPANY',

'PROFESSIONAL\_STATE',

'PROFESSIONAL\_CITY',

'PROFESSIONAL\_BOROUGH',

'FLAG\_PROFESSIONAL\_PHONE',

'PROFESSIONAL\_PHONE\_AREA\_CODE',

'MONTHS\_IN\_THE\_JOB',

'PROFESSION\_CODE',

'OCCUPATION\_TYPE',

'MATE\_PROFESSION\_CODE',

'EDUCATION\_LEVEL',

'FLAG\_HOME\_ADDRESS\_DOCUMENT',

'FLAG\_RG',

'FLAG\_CPF',

'FLAG\_INCOME\_PROOF',

'PRODUCT',

'FLAG\_ACSP\_RECORD',

'AGE',

'RESIDENCIAL\_ZIP\_3',

'PROFESSIONAL\_ZIP\_3',

'TARGET']

In [ ]:

bankcredit\_data**.**head()

In [ ]:

bankcredit\_data**.**dtypes

Here are the summary statistics of the variables.

In [ ]:

bankcredit\_data**.**describe()

# Data Preprocessing

The goal of this step is to leave the date ready to apply models to it. In general we want to:

1. Eliminate redundant and unethical variables
2. Treat empty and duplicate entries
3. Treat null values
4. Make data categorical wherever necessary
5. Treat outliers
6. Remove correlated features

We can see few variables with same values for all records, thus removing them from our dataframe.

Also, eliminating redundant variables.

In [ ]:

bankcredit\_data **=** bankcredit\_data**.**drop(['ID\_CLIENT',

'CLERK\_TYPE',

'PAYMENT\_DAY',

'APPLICATION\_SUBMISSION\_TYPE',

'QUANT\_ADDITIONAL\_CARDS',

'EDUCATION\_LEVEL\_S',

'SEX',

'STATE\_OF\_BIRTH',

'CITY\_OF\_BIRTH',

'RESIDENCIAL\_STATE',

'RESIDENCIAL\_CITY',

'RESIDENCIAL\_BOROUGH',

'RESIDENCIAL\_PHONE\_AREA\_CODE',

'FLAG\_MOBILE\_PHONE',

'QUANT\_CARS',

'PROFESSIONAL\_STATE',

'PROFESSIONAL\_CITY',

'PROFESSIONAL\_BOROUGH',

'PROFESSIONAL\_PHONE\_AREA\_CODE',

'OCCUPATION\_TYPE',

'FLAG\_ACSP\_RECORD',

'FLAG\_HOME\_ADDRESS\_DOCUMENT',

'FLAG\_RG',

'FLAG\_CPF',

'FLAG\_INCOME\_PROOF',], axis **=** 1)

In [ ]:

bankcredit\_data**.**head()

Treating data and replacing all empty values to NaN

In [ ]:

bankcredit\_data **=** bankcredit\_data**.**dropna(subset**=**['TARGET'])

In [ ]:

bankcredit\_data["RESIDENCIAL\_ZIP\_3"] **=** bankcredit\_data["RESIDENCIAL\_ZIP\_3"]**.**str**.**replace('#DIV/0!' , '0')

bankcredit\_data["RESIDENCIAL\_ZIP\_3"] **=** bankcredit\_data["RESIDENCIAL\_ZIP\_3"]**.**str**.**replace(' ' , '0')

In [ ]:

c **=** (bankcredit\_data["RESIDENCIAL\_ZIP\_3"]**.**values **==** '')

r **=** any(c)

r

In [ ]:

c **=** (bankcredit\_data["PROFESSIONAL\_ZIP\_3"]**.**values **==** '')

r **=** any(c)

r

In [ ]:

bankcredit\_data["PROFESSIONAL\_ZIP\_3"] **=** bankcredit\_data["PROFESSIONAL\_ZIP\_3"]**.**str**.**replace('#DIV/0!' , '0')

bankcredit\_data["PROFESSIONAL\_ZIP\_3"] **=** bankcredit\_data["PROFESSIONAL\_ZIP\_3"]**.**str**.**replace(' ' , '0')

In [ ]:

bankcredit\_data**.**replace(" ", np**.**NaN)

Converting variables to categorical

In [ ]:

bankcredit\_data **=** bankcredit\_data**.**replace('Y', 1)

bankcredit\_data **=** bankcredit\_data**.**replace('N', 0)

## Data Cleaning

Checking any duplicate records

In [ ]:

a **=** bankcredit\_data

duplicate **=** a[a**.**duplicated()]

duplicate

In [ ]:

bankcredit\_data**.**drop\_duplicates(subset**=None**, keep**=**'first', inplace**=True**, ignore\_index**=False**)

bankcredit\_data

## Splitting the data into training and testing dataset

In [ ]:

*# X = bankcredit\_data.drop("TARGET", axis = 1)*

*# y = bankcredit\_data['TARGET']*

In [ ]:

train, test **=** sc**.**split\_df(bankcredit\_data**.**iloc[:,1:],

y **=** 'TARGET',

ratio **=** 0.7,

seed **=** 251139213)**.**values()

In [ ]:

*# train, test = sc.split\_df(X, y ,*

*# ratio = 0.7,*

*# seed = 251139213).values()*

In [ ]:

bankcredit\_data['PROFESSIONAL\_ZIP\_3'] **=** bankcredit\_data['PROFESSIONAL\_ZIP\_3']**.**convert\_dtypes()

bankcredit\_data['PROFESSIONAL\_ZIP\_3'] **=** pd**.**to\_numeric(bankcredit\_data['PROFESSIONAL\_ZIP\_3'], errors**=**'coerce')**.**convert\_dtypes()

In [ ]:

bankcredit\_data['PROFESSIONAL\_ZIP\_3']**.**dtypes

In [ ]:

bankcredit\_data['RESIDENCIAL\_ZIP\_3'] **=** bankcredit\_data['RESIDENCIAL\_ZIP\_3']**.**convert\_dtypes()

bankcredit\_data['RESIDENCIAL\_ZIP\_3'] **=** pd**.**to\_numeric(bankcredit\_data['RESIDENCIAL\_ZIP\_3'], errors**=**'coerce')**.**convert\_dtypes()

Checking for Null values

In [ ]:

train**.**isnull()**.**any()

In [ ]:

null\_columns **=** train**.**columns[train**.**isnull()**.**any()]

train[null\_columns]**.**isnull()**.**sum()

### Null values

The core function here will be Panda's [fillna]. This allows to replace all null values (represented by None or NaN in Python) by a certain value. This also allows to set what the replacement will be with the value argument.

Remember the strategies to deal :

1. Keep: If the null values are a category by themselves. In this case, replace by something meaningful.
2. Delete: If the null values are too many **either by row or by column** then it is better to just drop the case or the variable.
3. Replace: If there are only a few missings for the variable or the row (<1% total), replace by the replace the null values by the **median** for continous variables, and the **mode** for categorical values.

Let's study our dataset's null values. The [isnull()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.isnull.html) function returns which elements in the dataframe are null. The [any()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.any.html) function returns a list with whatever columns (default) or rows (passing axis = 1 to the function) have any element with a boolean of true.

In [ ]:

bankcredit\_data['RESIDENCE\_TYPE']**.**fillna(train['RESIDENCE\_TYPE']**.**median(), inplace**=True**) *# Median = mode*

bankcredit\_data['MONTHS\_IN\_RESIDENCE']**.**fillna(train['MONTHS\_IN\_RESIDENCE']**.**median(), inplace**=True**)

bankcredit\_data['PROFESSION\_CODE']**.**fillna(train['PROFESSION\_CODE']**.**median(), inplace**=True**) *#Median = Mode*

*# test['RESIDENCE\_TYPE'].fillna(train['RESIDENCE\_TYPE'].median(), inplace=True) # Median = mode*

*# test['MONTHS\_IN\_RESIDENCE'].fillna(train['MONTHS\_IN\_RESIDENCE'].median(), inplace=True)*

*# test['PROFESSION\_CODE'].fillna(train['PROFESSION\_CODE'].median(), inplace=True) #Median = Mode*

We can see 2 columns have higher null values. Let's study them in further detail.

In [ ]:

train['MATE\_PROFESSION\_CODE']**.**describe()

In [ ]:

train['EDUCATION\_LEVEL']**.**describe()

In [ ]:

*# There can be scenario where the applicant has no spouse and thus the Mate Profession code and Education Level is blank.*

*# Also, it could be possible that the spouse has no job or has no education or the applicant didn't share the data.*

*# As we don't know the encoding for category of Marital Status, Mate Profession code and Education Level, so we can't split the above scenarios on the basis of any assumption.*

*# So, replacing Null values by creating another category, say 20 covering the above scenarios.*

bankcredit\_data['MATE\_PROFESSION\_CODE']**.**fillna(value **=** 20, inplace**=True**)

bankcredit\_data['EDUCATION\_LEVEL']**.**fillna(value **=** 20, inplace**=True**)

In [ ]:

null\_columns **=** bankcredit\_data**.**columns[bankcredit\_data**.**isnull()**.**any()]

bankcredit\_data[null\_columns]**.**isnull()**.**sum()

In [ ]:

train**.**head()

In [ ]:

*# bankcredit\_data.to\_excel("P1.xlsx")*

Visualizing data

In [ ]:

sns**.**set(color\_codes**=True**)

sns**.**displot(data **=** bankcredit\_data, x **=** 'POSTAL\_ADDRESS\_TYPE', hue **=** "TARGET", kind **=** 'kde')

In [ ]:

*# bankcredit\_data = bankcredit\_data.astype({"POSTAL\_ADDRESS\_TYPE":'category'})*

In [ ]:

sns**.**set(color\_codes**=True**)

**for** col\_id **in** bankcredit\_data**.**columns[np**.**r\_[1,2:6]]:

sns**.**displot(data **=** bankcredit\_data, x **=** col\_id, hue **=** "TARGET", kind **=** 'kde')

In [ ]:

**for** col\_id **in** bankcredit\_data**.**columns[np**.**r\_[1,6:20]]:

sns**.**displot(data **=** bankcredit\_data, x **=** col\_id, hue **=** "TARGET", kind **=** 'kde')

In [ ]:

sns**.**displot(data **=** bankcredit\_data, x **=** 'MONTHS\_IN\_THE\_JOB', hue **=** "TARGET", kind **=** 'kde')

In [ ]:

sns**.**displot(data **=** bankcredit\_data, x **=** 'PROFESSION\_CODE', hue **=** "TARGET", kind **=** 'kde')

In [ ]:

**for** col\_id **in** bankcredit\_data**.**columns[np**.**r\_[1,20:24]]:

sns**.**displot(data **=** bankcredit\_data, x **=** col\_id, hue **=** "TARGET", kind **=** 'kde')

In [ ]:

sns**.**displot(data **=** bankcredit\_data, x **=** 'RESIDENCIAL\_ZIP\_3', hue **=** "TARGET", kind **=** 'hist')

In [ ]:

sns**.**displot(data **=** bankcredit\_data, x **=** 'PROFESSIONAL\_ZIP\_3', hue **=** "TARGET", kind **=**'hist')

In [ ]:

sns**.**displot(data **=** bankcredit\_data, x **=** 'AGE', hue **=** "TARGET", kind **=** 'kde')

For the following, we need to treat the data and outliers-

* AGE - Truncating values beyond 18-65
* QUANT\_DEPENDENTS - Truncating values beyond 20
* RESIDENCIAL\_ZIP\_3 and PROFESSIONAL\_ZIP\_3 - Removing blanks

For other variables we will check variable distribution using graphs again and then treat outliers if needed after standardization as these plots may be misleading though, as the magnitude of the data is relevant.

In [ ]:

bankcredit\_data["AGE"]**.**describe()

In [ ]:

bankcredit\_data **=** bankcredit\_data**.**loc[bankcredit\_data['AGE'] **>** 17]

bankcredit\_data **=** bankcredit\_data**.**loc[bankcredit\_data['AGE'] **<** 81]

In [ ]:

bankcredit\_data['QUANT\_DEPENDANTS']**.**sort\_values(ascending**=False**)**.**head(n**=**20)

In [ ]:

bankcredit\_data["QUANT\_DEPENDANTS"]**.**replace({53: train['QUANT\_DEPENDANTS']**.**median()}, inplace**=True**)

In [ ]:

bankcredit\_data['RESIDENCIAL\_ZIP\_3']**.**sort\_values(ascending**=False**)**.**head(n**=**50)

In [ ]:

bankcredit\_data['PROFESSIONAL\_ZIP\_3']**.**sort\_values(ascending**=False**)**.**head(n**=**50)

In [ ]:

bankcredit\_data **=** bankcredit\_data**.**loc[bankcredit\_data['RESIDENCIAL\_ZIP\_3'] **<** 1000]

bankcredit\_data **=** bankcredit\_data**.**loc[bankcredit\_data['PROFESSIONAL\_ZIP\_3'] **<** 1000]

In [ ]:

*# bankcredit\_data['PROFESSIONAL\_ZIP\_3'].replace({'#DIV/0!': train['PROFESSIONAL\_ZIP\_3'].median()}, inplace=True)*

Much better, although income still has a large outlier. This plot might be misleading though, as the magnitude of the data is relevant. We might want to recheck this after normalizing the data.

## Introducing New Variables

In [ ]:

bankcredit\_data['TotalAvg\_Worth\_Ratio'] **=** (bankcredit\_data['PERSONAL\_MONTHLY\_INCOME'] **+** bankcredit\_data['OTHER\_INCOMES'] **+** bankcredit\_data['PERSONAL\_ASSETS\_VALUE'])**/**(bankcredit\_data['QUANT\_DEPENDANTS'] **+** 1)

bankcredit\_data['Total\_CreditCards'] **=** bankcredit\_data['FLAG\_VISA'] **+** bankcredit\_data['FLAG\_MASTERCARD'] **+** bankcredit\_data['FLAG\_DINERS'] **+** bankcredit\_data['FLAG\_AMERICAN\_EXPRESS'] **+** bankcredit\_data['FLAG\_OTHER\_CARDS']

bankcredit\_data['Contact\_Reliability'] **=** bankcredit\_data['FLAG\_RESIDENCIAL\_PHONE'] **+** bankcredit\_data['FLAG\_PROFESSIONAL\_PHONE'] **+** bankcredit\_data['FLAG\_EMAIL']

bankcredit\_data**.**head(10)

In [ ]:

*# bankcredit\_data.to\_excel("P1.xlsx")*

We can check now how the data looks like.

In [ ]:

*# df\_cat = bankcredit\_data.astype('category')*

In [ ]:

fig, ax **=** plt**.**subplots(figsize**=**(10,5))

a **=** sns**.**violinplot(x**=**'variable', y**=**'value', data**=**pd**.**melt(bankcredit\_data**.**iloc[:, np**.**r\_[1,31:32]]), ax**=**ax)

a**.**set\_xticklabels(a**.**get\_xticklabels(), rotation**=**90);

In [ ]:

fig, ax **=** plt**.**subplots(figsize**=**(10,5))

a **=** sns**.**violinplot(x**=**'variable', y**=**'value', data**=**pd**.**melt(bankcredit\_data**.**iloc[:, np**.**r\_[1,1:5]]), ax**=**ax)

a**.**set\_xticklabels(a**.**get\_xticklabels(), rotation**=**90);

In [ ]:

*# sns.catplot(x="RESIDENCIAL\_ZIP\_3", y="TARGET",*

*# kind="violin", data=bankcredit\_data)*

In [ ]:

ax **=** sns**.**violinplot(x**=**"RESIDENCIAL\_ZIP\_3", y**=**"TARGET", data**=**bankcredit\_data)

a**.**set\_xticklabels(a**.**get\_xticklabels(), rotation**=**90);

In [ ]:

ax **=** sns**.**violinplot(x**=**"PROFESSIONAL\_ZIP\_3", y**=**"TARGET", data**=**bankcredit\_data)

In [ ]:

*# sns.set\_theme(style="whitegrid")*

*# # ax = sns.violinplot(x=tips["total\_bill"])*

*# ax = sns.violinplot(x=bankcredit\_data["PROFESSIONAL\_ZIP\_3"])*

In [ ]:

fig, ax **=** plt**.**subplots(figsize**=**(10,5))

a **=** sns**.**violinplot(x**=**'variable', y**=**'value', data**=**pd**.**melt(bankcredit\_data**.**iloc[:, np**.**r\_[1,1:26]]), ax**=**ax)

a**.**set\_xticklabels(a**.**get\_xticklabels(), rotation**=**90);

In [ ]:

bankcredit\_data**.**describe()

Analysing Correlation

In [ ]:

plt**.**matshow(bankcredit\_data**.**corr())

plt**.**show()

We will now save the output to a compressed format which is very efficient to start whole data structures, [pickle](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_pickle.html).

In [ ]:

bankcredit\_data**.**to\_pickle('BankCreditClean.pkl')

## -----------------------------------------------------------------------------------------------------

In [ ]:

*# df = bankcredit\_data.drop(['RESIDENCIAL\_ZIP\_3', 'PROFESSIONAL\_ZIP\_3'], axis = 1)*

In [ ]:

*# df = bankcredit\_data.astype("category")*

*# df.dtypes*

## WoE

In [ ]:

*# bankcredit\_data.to\_excel("P1.xlsx")*

# Weight of Evidence Transformation

In this lab we will apply a Weight of Evidence transformation to our data. The idea is to:

* Split the data into a train/test set.
* Generate a relevant set of cuts to our data.
* Calculate the WoE for each variable.
* Save the data we just created.

We are assuming we have already cleaned the date of outliers and null values.

In order to do this we will use the fantastic [scorecardpy](https://github.com/ShichenXie/scorecardpy) Python package. First we need to install it, as it is not a standard package.

We use the OS python software pip for this.

In [ ]:

bankcredit\_data **=** pd**.**read\_pickle('BankCreditClean.pkl')

*# bankcredit\_data.describe()*

## Binning

The first step is to properly bin the data. Usually, we will run a tree and manually adjust those cases that do not follow a logical pattern.

However, as calculating WoE means we need to use the objective variable, we need to first create a train and test split. The scorecard package comes with a function to do so easily, split\_df, which takes as an argument the ratio and the seed.

In [ ]:

*# X = bankcredit\_data.drop("TARGET", axis = 1)*

*# y = bankcredit\_data['TARGET']*

In [ ]:

bankcredit\_data**.**head()

In [ ]:

bankcredit\_data **=** bankcredit\_data**.**drop(['NACIONALITY'], axis **=** 1)

In [ ]:

*#Removing as IV = 0*

bankcredit\_data **=** bankcredit\_data**.**drop(['MONTHS\_IN\_THE\_JOB','FLAG\_DINERS','FLAG\_AMERICAN\_EXPRESS','FLAG\_OTHER\_CARDS','PERSONAL\_ASSETS\_VALUE'], axis **=** 1)

In [ ]:

bankcredit\_data**.**head()

In [ ]:

train, test **=** sc**.**split\_df(bankcredit\_data**.**iloc[:,1:],

y **=** 'TARGET',

ratio **=** 0.7,

seed **=** 251139213)**.**values()

In [ ]:

test**.**describe()

In [ ]:

train**.**describe()

In [ ]:

bins **=** sc**.**woebin(train, y **=** 'TARGET',

min\_perc\_fine\_bin**=**0.01, *# How many bins to cut initially into*

min\_perc\_coarse\_bin**=**0.05, *# Minimum percentage per final bin*

stop\_limit**=**0.001, *# Minimum information value*

max\_num\_bin**=**8, *# Maximum number of bins*

method**=**'tree'

)

In [ ]:

*# 1- with 5 bins Age 0.082*

*# 2 - 8 bins Age 0.0842*

*# 3 - 12 bins - better IV value but trend wasn't as expected*

*# 4 - 10 bins - same as 12*

Visualizing trends and bins adjustment

In [ ]:

sc**.**woebin\_plot(bins)

In [ ]:

bins

Manual Adjustment

In [ ]:

*# breaks\_adj = sc.woebin\_adj(train, "TARGET", bins, adj\_all\_var = True)*

In [ ]:

*# breaks\_adj*

In [ ]:

breaks\_adj **=** {'AGE': [23.0,33.0,43.0,46.0,55.0,61.0,69.0], 'COMPANY': [1.0], 'Contact\_Reliability': [2.0,3.0],

'EDUCATION\_LEVEL': [1.0], 'FLAG\_EMAIL': [1.0], 'FLAG\_MASTERCARD': [1.0], 'FLAG\_PROFESSIONAL\_PHONE': [1.0],

'FLAG\_RESIDENCIAL\_PHONE': [1.0], 'FLAG\_VISA': [1.0], 'MARITAL\_STATUS': [2.0,3.0,5.0], 'MATE\_PROFESSION\_CODE': [1.0,18.0],

'MONTHS\_IN\_RESIDENCE': [6.0,16.0,30.0 ], 'OTHER\_INCOMES': [130.0], 'PERSONAL\_MONTHLY\_INCOME': [1700.0], 'PRODUCT': [2.0],

'PROFESSIONAL\_ZIP\_3': [160.0,250.0,350.0,400.0,580.0,850.0,960.0], 'PROFESSION\_CODE': [1.0,9.0,11.0],

'QUANT\_BANKING\_ACCOUNTS': [1.0], 'QUANT\_DEPENDANTS': [1.0,2.0,3.0], 'QUANT\_SPECIAL\_BANKING\_ACCOUNTS': [1.0],

'RESIDENCE\_TYPE': [2.0], 'RESIDENCIAL\_ZIP\_3': [160.0,250.0,350.0,400.0,580.0,850.0,960.0],

'TotalAvg\_Worth\_Ratio': [160.0, 260.0, 320.0], 'Total\_CreditCards': [1.0]}

In [ ]:

bins\_adj **=** sc**.**woebin(train, y**=**"TARGET", breaks\_list**=**breaks\_adj) *# Apply new cuts*

train\_woe **=** sc**.**woebin\_ply(train, bins\_adj) *# Calculate WoE dataset (train)*

test\_woe **=** sc**.**woebin\_ply(test, bins\_adj) *# Calculate WoE dataset (test)*

In [ ]:

train\_woe**.**head()

## IV Filtering value

Now we can check the information value of our variables and remove those who are not predictive. We use the function iv. In general:

* IV<0.02IV<0.02: No predictive ability, remove.
* 0.02≤IV<0.10.02≤IV<0.1: Small predictive ability, suggest to remove.
* 0.1≤IV<0.30.1≤IV<0.3: Medium predictive ability, leave.
* 0.3≤IV<10.3≤IV<1: Good predictive ability, leave.
* 1≤IV1≤IV: Strong predictive ability. Suspicious variable. Study if error in calculation (i.e. WoE leaves a category with 100% goods or bads) or if variable is capturing future information.

In [ ]:

sc**.**iv(train\_woe, 'TARGET')

In [ ]:

train\_woe**.**columns

In [ ]:

*# accepted\_range = np.r\_[0:3,4:10] # Note the last in each subrange is not used*

*# train\_woe = train\_woe.iloc[:, accepted\_range]*

*# test\_woe = test\_woe.iloc[:, accepted\_range]*

*# train\_woe.head()*

In [ ]:

test\_woe**.**head()

Now the data should look much better.

In [ ]:

fig, ax **=** plt**.**subplots(figsize**=**(10,5))

a **=** sns**.**violinplot(x**=**'variable', y**=**'value', data**=**pd**.**melt(bankcredit\_data**.**iloc[:, np**.**r\_[1,1:5]]), ax**=**ax)

a**.**set\_xticklabels(a**.**get\_xticklabels(), rotation**=**90);

Saving the results

In [ ]:

train\_woe**.**to\_csv("train\_woe.csv", index **=** **False**)

test\_woe**.**to\_csv("test\_woe.csv", index **=** **False**)

bankcredit\_data**.**to\_pickle('BankcreditCleanNewVars.pkl')

In [ ]:

**from** google.colab **import** files

files**.**download("train\_woe.csv")

files**.**download("test\_woe.csv")

# Logistic Regression and Scorecards

In [ ]:

bankcredit\_train\_WoE **=** pd**.**read\_csv('train\_woe.csv')

bankcredit\_test\_WoE **=** pd**.**read\_csv('test\_woe.csv')

bankcredit\_data **=** pd**.**read\_pickle('BankcreditCleanNewVars.pkl')

In [ ]:

bankcredit\_train\_noWoE, bankcredit\_test\_noWoE **=** sc**.**split\_df(bankcredit\_data**.**iloc[:,1:],

y **=** 'TARGET',

ratio **=** 0.7,

seed **=** 251139213)**.**values()

In [ ]:

*# Give breaks for WoE*

breaks\_adj **=** {'AGE': [23.0,33.0,43.0,46.0,55.0,61.0,69.0], 'COMPANY': [1.0], 'Contact\_Reliability': [2.0,3.0], 'EDUCATION\_LEVEL': [1.0], 'FLAG\_EMAIL': [1.0], 'FLAG\_MASTERCARD': [1.0], 'FLAG\_PROFESSIONAL\_PHONE': [1.0], 'FLAG\_RESIDENCIAL\_PHONE': [1.0], 'FLAG\_VISA': [1.0], 'MARITAL\_STATUS': [2.0,3.0,5.0], 'MATE\_PROFESSION\_CODE': [1.0,18.0], 'MONTHS\_IN\_RESIDENCE': [6.0,16.0,30.0 ], 'OTHER\_INCOMES': [130.0], 'PERSONAL\_MONTHLY\_INCOME': [1700.0], 'PRODUCT': [2.0], 'PROFESSIONAL\_ZIP\_3': [160.0,250.0,350.0,400.0,580.0,850.0,960.0], 'PROFESSION\_CODE': [1.0,9.0,11.0], 'QUANT\_BANKING\_ACCOUNTS': [1.0], 'QUANT\_DEPENDANTS': [1.0,2.0,3.0], 'QUANT\_SPECIAL\_BANKING\_ACCOUNTS': [1.0], 'RESIDENCE\_TYPE': [2.0], 'RESIDENCIAL\_ZIP\_3': [160.0,250.0,350.0,400.0,580.0,850.0,960.0], 'TotalAvg\_Worth\_Ratio': [160.0, 260.0, 320.0], 'Total\_CreditCards': [1.0]}

*# Apply breaks.*

bins\_adj **=** sc**.**woebin(bankcredit\_train\_noWoE, y**=**"TARGET",

breaks\_list**=**breaks\_adj)

Correlation Analysis

In [ ]:

*# Compute the correlation matrix*

corr **=** bankcredit\_train\_WoE**.**corr()

corr **=** np**.**abs(corr)

*# Generate a mask for the upper triangle*

mask **=** np**.**triu(np**.**ones\_like(corr, dtype**=**bool))

*# Set up the matplotlib figure*

f, ax **=** plt**.**subplots(figsize**=**(11, 9))

*# Generate a custom diverging colormap*

cmap **=** sns**.**diverging\_palette(230, 20, as\_cmap**=True**)

*# Draw the heatmap with the mask and correct aspect ratio*

sns**.**heatmap(corr, mask**=**mask, cmap**=**cmap, vmax**=**1, center**=**0,

square**=True**, linewidths**=.**5, cbar\_kws**=**{"shrink": **.**5})

In [ ]:

corr

In [ ]:

*# Compute the correlation matrix*

corr **=** bankcredit\_train\_WoE**.**corr()

corr **=** np**.**abs(corr)

mask **=** np**.**triu(np**.**ones\_like(corr, dtype**=**bool))

mask

In [ ]:

bankcredit\_train\_WoE **=** bankcredit\_train\_WoE**.**drop(['QUANT\_BANKING\_ACCOUNTS\_woe','PROFESSIONAL\_ZIP\_3\_woe'], axis **=** 1)

In [ ]:

bankcredit\_test\_WoE **=** bankcredit\_test\_WoE**.**drop(['QUANT\_BANKING\_ACCOUNTS\_woe','PROFESSIONAL\_ZIP\_3\_woe'], axis **=** 1)

In [ ]:

*# bankcredit\_train\_WoE = bankcredit\_train\_WoE.drop(['Total\_CreditCards\_woe'], axis = 1)*

*# bankcredit\_test\_WoE = bankcredit\_test\_WoE.drop(['Total\_CreditCards\_woe'], axis = 1)*

In [ ]:

*# bankcredit\_train\_WoE = bankcredit\_train\_WoE.drop(['FLAG\_EMAIL\_woe','FLAG\_VISA\_woe', 'PRODUCT\_woe', 'FLAG\_PROFESSIONAL\_PHONE\_woe', 'COMPANY\_woe'], axis = 1)*

*# bankcredit\_test\_WoE = bankcredit\_test\_WoE.drop(['FLAG\_EMAIL\_woe','FLAG\_VISA\_woe', 'PRODUCT\_woe', 'FLAG\_PROFESSIONAL\_PHONE\_woe', 'COMPANY\_woe'], axis = 1)*

## Generating a logistic regression object

To train a logistic regression, we first need to create an object that stores how we want the model to be trained. In general:

* We create the model we want to train, with all required parameters. This model is **not trained yet**, it just keeps the logic we will use.
* We apply the fit function to the object we just created. This takes as input the training set and the targets (if the model is supervised), and will update our model with trained parameters.
* We then used our trained model to apply it to a test set, and calculate outputs.

In a nutshell, LASSO and Ridge are going to penalize including variables by adding either a linear (LASSO) or quadratic (Ridge) term to the minimization algorithm, or a combination of the two if using Elastic Net.

These methods have hypermparameters that need to be optimized. For this we will use a cross-validation procedure.

In [ ]:

**from** sklearn.linear\_model **import** LogisticRegressionCV

bankcredit\_logreg **=** LogisticRegressionCV(penalty**=**'l2', *# Type of penalization l1 = lasso, l2 = ridge, elasticnet*

Cs **=** 10, *# How many parameters to try. Can also be a vector with parameters to try.*

tol**=**0.000001, *# Tolerance for parameters*

cv **=** 3, *# How many CV folds to try. 3 or 5 should be enough.*

fit\_intercept**=True**, *# Use constant?*

class\_weight**=**'balanced', *# Weights, see below*

random\_state**=**251139213, *# Random seed*

max\_iter**=**100, *# Maximum iterations*

verbose**=**2, *# Show process. 1 is yes.*

solver **=** 'saga', *# How to optimize.*

n\_jobs **=** 2, *# Processes to use. Set to number of physical cores.*

refit **=** **True**, *# If to retrain with the best parameter and all data after finishing.*

*# l1\_ratios = np.arange(0, 0.2, 0.01), # The LASSO / Ridge ratios.*

)

In [ ]:

bankcredit\_train\_WoE**.**head()

In [ ]:

bankcredit\_logreg**.**fit(X **=** bankcredit\_train\_WoE**.**iloc[:, 1:], *# All rows and from the second var to end*

y **=** bankcredit\_train\_WoE['TARGET'] *# The target*

)

In [ ]:

coef\_df **=** pd**.**concat([pd**.**DataFrame({'column': bankcredit\_train\_WoE**.**columns[1:]}),

pd**.**DataFrame(np**.**transpose(bankcredit\_logreg**.**coef\_))],

axis **=** 1

)

coef\_df

In [ ]:

bankcredit\_logreg**.**intercept\_

Applying to test set

In [ ]:

pred\_class\_test **=** bankcredit\_logreg**.**predict(bankcredit\_test\_WoE**.**iloc[:, 1:])

probs\_test **=** bankcredit\_logreg**.**predict\_proba(bankcredit\_test\_WoE**.**iloc[:, 1:])

print(probs\_test[0:5], pred\_class\_test[0:5])

In [ ]:

**from** sklearn.metrics **import** confusion\_matrix

In [ ]:

confusion\_matrix(y\_true **=** bankcredit\_test\_WoE['TARGET'], y\_pred **=** pred\_class\_test)

In [ ]:

**from** sklearn.linear\_model **import** LogisticRegression

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

In [ ]:

*# Calculate confusion matrix*

confusion\_matrix\_lr **=** confusion\_matrix(y\_true **=** bankcredit\_test\_noWoE['TARGET'],

y\_pred **=** pred\_class\_test)

*# Turn matrix to percentages*

confusion\_matrix\_lr **=** confusion\_matrix\_lr**.**astype('float') **/** confusion\_matrix\_lr**.**sum(axis**=**1)[:, np**.**newaxis]

*# Turn to dataframe*

df\_cm **=** pd**.**DataFrame(

confusion\_matrix\_lr, index**=**['good', 'bad'], columns**=**['good', 'bad'],

)

*# Parameters of the image*

figsize **=** (10,7)

fontsize**=**14

*# Create image*

fig **=** plt**.**figure(figsize**=**figsize)

heatmap **=** sns**.**heatmap(df\_cm, annot**=True**, fmt**=**'.2f')

*# Make it nicer*

heatmap**.**yaxis**.**set\_ticklabels(heatmap**.**yaxis**.**get\_ticklabels(), rotation**=**0,

ha**=**'right', fontsize**=**fontsize)

heatmap**.**xaxis**.**set\_ticklabels(heatmap**.**xaxis**.**get\_ticklabels(), rotation**=**45,

ha**=**'right', fontsize**=**fontsize)

*# Add labels*

plt**.**ylabel('True label')

plt**.**xlabel('Predicted label')

*# Plot!*

plt**.**show()

In [ ]:

*# probs\_test*

In [ ]:

fpr, tpr, thresholds **=** roc\_curve(bankcredit\_test\_noWoE['TARGET'], probs\_test[:,1])

*# Save the AUC in a variable to display it. Round it first*

auc **=** np**.**round(roc\_auc\_score(y\_true **=** bankcredit\_test\_noWoE['TARGET'],

y\_score **=** probs\_test[:,1]),

decimals **=** 3)

*# Create and show the plot*

plt**.**plot(fpr,tpr,label**=**"Bankcredit LR, auc="**+**str(auc))

plt**.**legend(loc**=**4)

plt**.**show()

## Scorecards

In [ ]:

bankcredit\_sc **=** sc**.**scorecard(bins\_adj, *# bins from the WoE*

bankcredit\_logreg, *# Trained logistic regression*

bankcredit\_train\_WoE**.**columns[1:], *# The column names in the trained LR*

points0**=**1200, *# Base points*

odds0**=**0.1, *# Base odds bads:goods*

pdo**=**200

) *# PDO*

In [ ]:

bankcredit\_sc

In [ ]:

*# Applying the credit score. Applies over the original data!*

train\_score **=** sc**.**scorecard\_ply(bankcredit\_train\_noWoE, bankcredit\_sc,

print\_step**=**0)

test\_score **=** sc**.**scorecard\_ply(bankcredit\_test\_noWoE, bankcredit\_sc,

print\_step**=**0)

In [ ]:

train\_score**.**describe()

In [ ]:

bankcredit\_train\_noWoE['score'] **=** train\_score

bankcredit\_test\_noWoE['score'] **=** test\_score

In [ ]:

bankcredit\_train\_noWoE**.**head()

In [ ]:

bankcredit\_test\_noWoE**.**to\_excel("P1.xlsx")

bankcredit\_train\_noWoE**.**to\_excel("P2.xlsx")

## Random Forest

In [ ]:

bankcredit\_data **=** pd**.**read\_pickle('BankcreditCleanNewVars.pkl')

In [ ]:

bankcredit\_data**.**head()

In [ ]:

df1 **=** bankcredit\_data**.**pop('TARGET')

bankcredit\_data['TARGET']**=**df1

In [ ]:

bankcredit\_data**.**head()

In [ ]:

df1 **=** bankcredit\_data

In [ ]:

df1**.**dtypes

In [ ]:

df1 **=** df1**.**drop(['QUANT\_SPECIAL\_BANKING\_ACCOUNTS', 'PROFESSIONAL\_ZIP\_3'], axis **=** 1)

In [ ]:

df1**.**head()

In [ ]:

df1["POSTAL\_ADDRESS\_TYPE"] **=** df1['POSTAL\_ADDRESS\_TYPE']**.**astype("category")

df1["MARITAL\_STATUS"] **=** df1['MARITAL\_STATUS']**.**astype("category")

df1["FLAG\_RESIDENCIAL\_PHONE"] **=** df1['FLAG\_RESIDENCIAL\_PHONE']**.**astype("category")

df1["RESIDENCE\_TYPE"] **=** df1['RESIDENCE\_TYPE']**.**astype("category")

df1["FLAG\_EMAIL"] **=** df1['FLAG\_EMAIL']**.**astype("category")

df1["FLAG\_VISA"] **=** df1['FLAG\_VISA']**.**astype("category")

df1["FLAG\_MASTERCARD"] **=** df1['FLAG\_MASTERCARD']**.**astype("category")

df1["QUANT\_BANKING\_ACCOUNTS"] **=** df1['QUANT\_BANKING\_ACCOUNTS']**.**astype("category")

df1["COMPANY"] **=** df1['COMPANY']**.**astype("category")

df1["FLAG\_PROFESSIONAL\_PHONE"] **=** df1['FLAG\_PROFESSIONAL\_PHONE']**.**astype("category")

df1["PROFESSION\_CODE"] **=** df1['PROFESSION\_CODE']**.**astype("category")

df1["MATE\_PROFESSION\_CODE"] **=** df1['MATE\_PROFESSION\_CODE']**.**astype("category")

df1["EDUCATION\_LEVEL"] **=** df1['EDUCATION\_LEVEL']**.**astype("category")

df1["PRODUCT"] **=** df1['PRODUCT']**.**astype("category")

df1["Contact\_Reliability"] **=** df1['Contact\_Reliability']**.**astype("category")

In [ ]:

df2 **=** pd**.**get\_dummies(df1)

In [ ]:

df2**.**head()

In [ ]:

df3 **=** df2**.**pop('TARGET')

df2['TARGET']**=**df3

In [ ]:

df2**.**head()

In [ ]:

bankcredit\_train\_rf, bankcredit\_test\_rf **=** sc**.**split\_df(df2**.**iloc[:,1:],

y **=** 'TARGET',

ratio **=** 0.7,

seed **=** 251139213)**.**values()

In [ ]:

*# #removing because the below variables are perfectly correlated*

*# bankcredit\_train\_rf = bankcredit\_train\_rf.drop(['QUANT\_BANKING\_ACCOUNTS','PROFESSIONAL\_ZIP\_3'], axis = 1)*

*# bankcredit\_test\_rf = bankcredit\_test\_rf.drop(['QUANT\_BANKING\_ACCOUNTS','PROFESSIONAL\_ZIP\_3'], axis = 1)*

In [ ]:

bankcredit\_train\_rf**.**head()

In [ ]:

bankcredit\_train\_rf**.**describe()

In [ ]:

**from** sklearn.ensemble **import** RandomForestClassifier

*#Define the classifier*

bankcredit\_rf **=** RandomForestClassifier(n\_estimators**=**1000, *# Number of trees to train # starting with V\*10*

criterion**=**'entropy', *# How to train the trees. Also supports gini.*

max\_depth**=None**, *# Max depth of the trees. Not necessary to change.*

min\_samples\_split**=**2, *# Minimum samples to create a split.*

min\_samples\_leaf**=**0.0001, *# Minimum samples in a leaf. Accepts fractions for %. This is 0.1% of sample.*

min\_weight\_fraction\_leaf**=**0.0, *# Same as above, but uses the class weights.*

max\_features**=**'auto', *# Maximum number of features per split (not tree!) by default is sqrt(vars)*

*# max\_features=5,# OR V/3*

max\_leaf\_nodes**=None**, *# Maximum number of nodes.*

min\_impurity\_decrease**=**0.00001, *# Minimum impurity decrease. This is 10^-4.*

bootstrap**=True**, *# If sample with repetition. For large samples (>100.000) set to false.*

oob\_score**=True**, *# If report accuracy with non-selected cases.*

n\_jobs**=**2, *# Parallel processing. Set to the number of cores you have. Watch your RAM!!*

random\_state**=**251139213, *# Seed*

verbose**=**1, *# If to give info during training. Set to 0 for silent training.*

warm\_start**=False**, *# If train over previously trained tree.*

class\_weight**=**'balanced' *# Balance the classes.*

)

In [ ]:

bankcredit\_train\_rf**.**head()

In [ ]:

*# Train the RF.*

bankcredit\_rf**.**fit(bankcredit\_train\_rf**.**iloc[:,:**-**1]**.**values, *# X*

bankcredit\_train\_rf['TARGET']**.**values *# y*

)

In [ ]:

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

*# Apply the model to the test set.*

rf\_pred\_class\_test **=** bankcredit\_rf**.**predict(bankcredit\_test\_rf**.**iloc[:, :**-**1]**.**values)

rf\_probs\_test **=** bankcredit\_rf**.**predict\_proba(bankcredit\_test\_rf**.**iloc[:, :**-**1]**.**values)

In [ ]:

*# Calculate confusion matrix*

confusion\_matrix\_rf **=** confusion\_matrix(y\_true **=** bankcredit\_test\_rf['TARGET'],

y\_pred **=** rf\_pred\_class\_test)

*# Turn matrix to percentages*

confusion\_matrix\_rf **=** confusion\_matrix\_rf**.**astype('float') **/** confusion\_matrix\_rf**.**sum(axis**=**1)[:, np**.**newaxis]

*# Turn to dataframe*

df\_cm **=** pd**.**DataFrame(

confusion\_matrix\_rf, index**=**['good', 'bad'], columns**=**['good', 'bad'],

)

*# Parameters of the image*

figsize **=** (10,7)

fontsize**=**14

*# Create image*

fig **=** plt**.**figure(figsize**=**figsize)

heatmap **=** sns**.**heatmap(df\_cm, annot**=True**, fmt**=**'.4f')

*# Make it nicer*

heatmap**.**yaxis**.**set\_ticklabels(heatmap**.**yaxis**.**get\_ticklabels(), rotation**=**0,

ha**=**'right', fontsize**=**fontsize)

heatmap**.**xaxis**.**set\_ticklabels(heatmap**.**xaxis**.**get\_ticklabels(), rotation**=**45,

ha**=**'right', fontsize**=**fontsize)

*# Add labels*

plt**.**ylabel('True label')

plt**.**xlabel('Predicted label')

*# Plot!*

plt**.**show()

In [ ]:

*# Calculate the ROC curve points*

fpr, tpr, thresholds **=** roc\_curve(bankcredit\_test\_rf['TARGET'], rf\_probs\_test[:,1])

*# Save the AUC in a variable to display it. Round it first*

auc **=** np**.**round(roc\_auc\_score(y\_true **=** bankcredit\_test\_rf['TARGET'],

y\_score **=** rf\_probs\_test[:,1]),

decimals **=** 3)

*# Create and show the plot*

plt**.**plot(fpr,tpr,label**=**"Bankcredit RF, auc="**+**str(auc))

plt**.**legend(loc**=**4)

plt**.**show()

In [ ]:

bankcredit\_train\_rf**.**iloc[:, :**-**1]**.**columns

In [ ]:

*# Plot variable importance*

importances **=** bankcredit\_rf**.**feature\_importances\_

indices **=** np**.**argsort(importances)[::**-**1]

f, ax **=** plt**.**subplots(figsize**=**(4, 20))

plt**.**title("Variable Importance - Random Forest")

sns**.**set\_color\_codes("pastel")

sns**.**barplot(y**=**[bankcredit\_train\_rf**.**iloc[:, :**-**1]**.**columns[i] **for** i **in** indices],

x**=**importances[indices],

label**=**"Total", color**=**"b")

ax**.**set(ylabel**=**"Variable",

xlabel**=**"Variable Importance (Entropy)")

sns**.**despine(left**=True**, bottom**=True**)

plt**.**savefig('RFPlot.pdf', dpi**=**300, bbox\_inches**=**'tight')

## XGBoosting

In [ ]:

**from** xgboost **import** XGBClassifier

*#Define the classifier.*

XGB\_Bankcredit **=** XGBClassifier(max\_depth**=**2, *# Depth of each tree*

learning\_rate**=**0.01, *# How much to shrink error in each subsequent training. Trade-off with no. estimators.*

n\_estimators**=**100, *# How many trees to use, the more the better, but decrease learning rate if many used.*

verbosity**=**1, *# If to show more errors or not.*

objective**=**'binary:logistic', *# Type of target variable.*

booster**=**'gbtree', *# What to boost. Trees in this case.*

n\_jobs**=**2, *# Parallel jobs to run. Set your processor number.*

gamma**=**0.001, *# Minimum loss reduction required to make a further partition on a leaf node of the tree. (Controls growth!)*

subsample**=**0.632, *# Subsample ratio. Can set lower*

colsample\_bytree**=**1, *# Subsample ratio of columns when constructing each tree.*

colsample\_bylevel**=**1, *# Subsample ratio of columns when constructing each level. 0.33 is similar to random forest.*

colsample\_bynode**=**1, *# Subsample ratio of columns when constructing each split.*

reg\_alpha**=**1, *# Regularizer for first fit. alpha = 1, lambda = 0 is LASSO.*

reg\_lambda**=**0, *# Regularizer for first fit.*

scale\_pos\_weight**=**1, *# Balancing of positive and negative weights. G / B*

base\_score**=**0.1, *# Global bias. Set to average of the target rate.*

random\_state**=**251139213, *# Seed*

missing**=None**, *# How are nulls encoded?*

tree\_method **=** 'hist'

*# tree\_method='gpu\_hist', # How to train the trees?*

*# gpu\_id=0 # With which GPU?*

)

In [ ]:

*# Define the parameters. Play with this grid!*

param\_grid **=** dict({'n\_estimators': [50, 150, 200],

'max\_depth': [2, 3, 4],

'learning\_rate' : [0.01, 0.05, 0.1, 0.15]

})

In [ ]:

bankcredit\_train\_rf['RESIDENCIAL\_ZIP\_3']**.**astype('category')

bankcredit\_test\_rf['RESIDENCIAL\_ZIP\_3']**.**astype('category')

In [ ]:

bankcredit\_train\_rf['RESIDENCIAL\_ZIP\_3'] **=** pd**.**to\_numeric(bankcredit\_train\_rf['RESIDENCIAL\_ZIP\_3'], downcast**=**'float' )

bankcredit\_test\_rf['RESIDENCIAL\_ZIP\_3'] **=** pd**.**to\_numeric(bankcredit\_test\_rf['RESIDENCIAL\_ZIP\_3'], downcast**=**'float' )

In [ ]:

bankcredit\_train\_rf**.**dtypes

In [ ]:

*# Always a good idea to tune on a reduce sample of the train set, as we will call many functions.*

val\_train **=** bankcredit\_train\_rf**.**sample(frac **=** 0.2, *# The fraction to extract*

random\_state **=** 251139213, *# The seed.*

)

In [ ]:

**from** sklearn.model\_selection **import** GridSearchCV

*# Define grid search object.*

GridXGB **=** GridSearchCV(XGB\_Bankcredit, *# Original XGB.*

param\_grid, *# Parameter grid*

cv **=** 3, *# Number of cross-validation folds.*

scoring **=** 'roc\_auc', *# How to rank outputs.*

n\_jobs **=** 2, *# Parallel jobs. -1 is "all you have"*

refit **=** **False**, *# If refit at the end with the best. We'll do it manually.*

verbose **=** 1 *# If to show what it is doing.*

)

In [ ]:

*# Train grid search.*

GridXGB**.**fit(val\_train**.**iloc[:, :**-**1], val\_train['TARGET'])

In [ ]:

*# Show best params*

print('The best AUC is %.3f' **%** GridXGB**.**best\_score\_)

GridXGB**.**best\_params\_

In [ ]:

*# Create XGB with best parameters.*

XGB\_Bankcredit **=** XGBClassifier(max\_depth**=**GridXGB**.**best\_params\_**.**get('max\_depth'), *# Depth of each tree*

learning\_rate**=**GridXGB**.**best\_params\_**.**get('learning\_rate'), *# How much to shrink error in each subsequent training. Trade-off with no. estimators.*

n\_estimators**=**GridXGB**.**best\_params\_**.**get('n\_estimators'), *# How many trees to use, the more the better, but decrease learning rate if many used.*

verbosity**=**1, *# If to show more errors or not.*

objective**=**'binary:logistic', *# Type of target variable.*

booster**=**'gbtree', *# What to boost. Trees in this case.*

*#n\_jobs=4, # Parallel jobs to run. Set your processor number.*

gamma**=**0.001, *# Minimum loss reduction required to make a further partition on a leaf node of the tree. (Controls growth!)*

subsample**=**0.632, *# Subsample ratio. Can set lower*

colsample\_bytree**=**1, *# Subsample ratio of columns when constructing each tree.*

colsample\_bylevel**=**1, *# Subsample ratio of columns when constructing each level. 0.33 is similar to random forest.*

colsample\_bynode**=**1, *# Subsample ratio of columns when constructing each split.*

reg\_alpha**=**1, *# Regularizer for first fit. alpha = 1, lambda = 0 is LASSO.*

reg\_lambda**=**0, *# Regularizer for first fit.*

scale\_pos\_weight**=**1, *# Balancing of positive and negative weights.*

base\_score**=**0.1, *# Global bias. Set to average of the target rate.*

random\_state**=**251139213, *# Seed*

missing**=None**, *# How are nulls encoded?*

tree\_method **=** 'hist',

*# tree\_method='gpu\_exact', # How to train the trees?*

*# tree\_method = 'gpu\_hist',*

gpu\_id**=**0 *# With which GPU?*

)

In [ ]:

bankcredit\_train\_rf**.**head()

In [ ]:

*# Train over all training data.*

XGB\_Bankcredit**.**fit(bankcredit\_train\_rf**.**iloc[:, :**-**1], bankcredit\_train\_rf['TARGET'])

In [ ]:

*# Plot variable importance*

importances **=** XGB\_Bankcredit**.**feature\_importances\_

indices **=** np**.**argsort(importances)[::**-**1]

f, ax **=** plt**.**subplots(figsize**=**(4, 20))

plt**.**title("Variable Importance - XGBoosting")

sns**.**set\_color\_codes("pastel")

sns**.**barplot(y**=**[bankcredit\_train\_rf**.**iloc[:, :**-**1]**.**columns[i] **for** i **in** indices], x**=**importances[indices],

label**=**"Total", color**=**"b")

ax**.**set(ylabel**=**"Variable",

xlabel**=**"Variable Importance (Entropy)")

sns**.**despine(left**=True**, bottom**=True**)

plt**.**savefig('XGBPlot.pdf', dpi**=**300, bbox\_inches**=**'tight')

In [ ]:

*# Calculate probability*

XGBClassTest **=** XGB\_Bankcredit**.**predict(bankcredit\_test\_rf**.**iloc[:, :**-**1])

xg\_probs\_test **=** XGB\_Bankcredit**.**predict\_proba(bankcredit\_test\_rf**.**iloc[:, :**-**1])

xg\_probs\_test **=** xg\_probs\_test[:, 1]

*# Calculate confusion matrix*

confusion\_matrix\_xgb **=** confusion\_matrix(y\_true **=** bankcredit\_test\_rf['TARGET'],

y\_pred **=** XGBClassTest)

*# Turn matrix to percentages*

confusion\_matrix\_xgb **=** confusion\_matrix\_xgb**.**astype('float') **/** confusion\_matrix\_xgb**.**sum(axis**=**1)[:, np**.**newaxis]

*# Turn to dataframe*

df\_cm **=** pd**.**DataFrame(

confusion\_matrix\_xgb, index**=**['good', 'bad'], columns**=**['good', 'bad'],

)

*# Parameters of the image*

figsize **=** (10,7)

fontsize**=**14

*# Create image*

fig **=** plt**.**figure(figsize**=**figsize)

heatmap **=** sns**.**heatmap(df\_cm, annot**=True**, fmt**=**'.4f')

*# Make it nicer*

heatmap**.**yaxis**.**set\_ticklabels(heatmap**.**yaxis**.**get\_ticklabels(), rotation**=**0,

ha**=**'right', fontsize**=**fontsize)

heatmap**.**xaxis**.**set\_ticklabels(heatmap**.**xaxis**.**get\_ticklabels(), rotation**=**45,

ha**=**'right', fontsize**=**fontsize)

*# Add labels*

plt**.**ylabel('True label')

plt**.**xlabel('Predicted label')

*# Plot!*

plt**.**show()

In [ ]:

*# Calculate the ROC curve points*

fpr, tpr, thresholds **=** roc\_curve(bankcredit\_test\_rf['TARGET'],

xg\_probs\_test)

*# Save the AUC in a variable to display it. Round it first*

auc **=** np**.**round(roc\_auc\_score(y\_true **=** bankcredit\_test\_rf['TARGET'],

y\_score **=** xg\_probs\_test),

decimals **=** 3)

*# Create and show the plot*

plt**.**plot(fpr,tpr,label**=**"AUC - XGBoosting = " **+** str(auc))

plt**.**legend(loc**=**4)

plt**.**show()

TreeSHAP

In [ ]:

**!**pip install shap

In [ ]:

**import** shap

shap**.**initjs() *# Import Java engine.*

In [ ]:

*# Trains the game-theoretic model. Really complex so requires sampling.*

explainer **=** shap**.**TreeExplainer(XGB\_Bankcredit, *# The model*

data **=** shap**.**sample(bankcredit\_train\_rf**.**iloc[:, :**-**1],

100) *# Create a sample of 100 cases*

)

*# Applies model ot the full dataset.*

shap\_values **=** explainer**.**shap\_values(bankcredit\_train\_rf**.**iloc[:, :**-**1],

check\_additivity**=False**)

In [ ]:

shap**.**summary\_plot(shap\_values, *# The Shapley values.*

bankcredit\_train\_rf**.**iloc[:, :**-**1], *# The training sample*

show**=False**) *# Whether to print the model or not*

*# Let's save this as a PDF for later use.*

plt**.**savefig('ShapSummaryPlot.pdf', dpi**=**300, bbox\_inches**=**'tight')

plt**.**show()

In [ ]:

*# shap.dependence\_plot("FLAG\_RESIDENCIAL\_PHONE", # The variable to study*

*# shap\_values, # The Shapley values.*

*# bankcredit\_train\_rf.iloc[:, :-1], # The training sample*

*# show=False) # Whether to print the model or not*

*# plt.savefig('ShapPHONE.pdf', dpi=300, bbox\_inches='tight')*

*# plt.show()*

ROC Curves

In [ ]:

*# Predict probabilities of scorecard.*

*# logreg\_probs\_test = bankcredit\_logreg.predict\_proba(bankcredit\_test\_rf.iloc[:, 1:])*

In [ ]:

*# Set models and probabilities. This structure is called a dictionary.*

models **=** [

{

'label': 'Logistic Regression',

'data' : bankcredit\_test\_noWoE['TARGET'],

'probs': probs\_test[:,1]

},

{

'label': 'Gradient Boosting',

'data' : bankcredit\_test\_rf['TARGET'],

'probs': xg\_probs\_test

},

{

'label': 'Random Forest',

'data' : bankcredit\_test\_rf['TARGET'],

'probs': rf\_probs\_test[:,1]

}

]

*# Loop that creates the plot. I will pass each ROC curve one by one.*

**for** m **in** models:

auc **=** roc\_auc\_score(y\_true **=** m['data'],

y\_score **=** m['probs'])

fpr, tpr, thresholds **=** roc\_curve(m['data'],

m['probs'])

plt**.**plot(fpr, tpr, label**=**'%s ROC (area = %0.3f)' **%** (m['label'], auc))

*# Settings*

plt**.**plot([0, 1], [0, 1],'r--')

plt**.**xlim([0.0, 1.0])

plt**.**ylim([0.0, 1.05])

plt**.**xlabel('1-Specificity(False Positive Rate)')

plt**.**ylabel('Sensitivity(True Positive Rate)')

plt**.**title('Receiver Operating Characteristic')

plt**.**legend(loc**=**"lower right")

*# Plot!*

plt**.**show()

1. <https://countryeconomy.com/demography/life-expectancy/brazil?year=2007> [↑](#footnote-ref-1)
2. <https://en.wikipedia.org/wiki/Life_expectancy#/media/File:Life_Expectancy_At_Birth_By_Region.png> [↑](#footnote-ref-2)
3. <https://destinationscanner.com/is-brazil-a-third-world-country/> [↑](#footnote-ref-3)