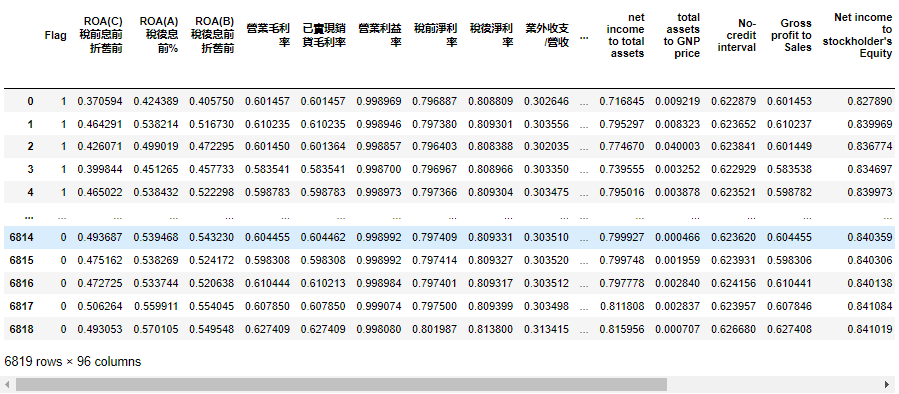
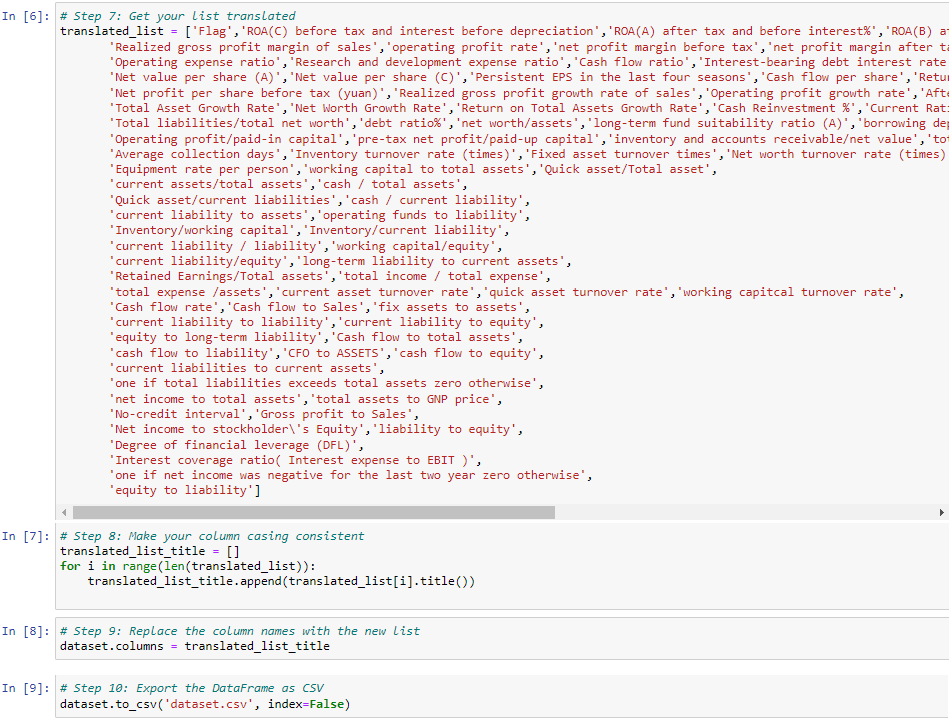
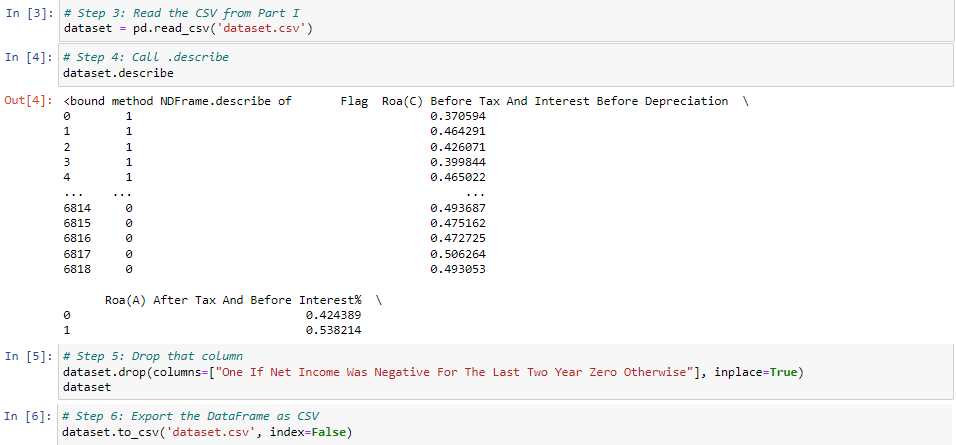
Loan Default/ Bankruptcy Prediction – Final Report  
  
Done by: Yeo Theng Hee, TFIP-Data Analytics 2021, NUS-ISS  
Project Supervisor: Prof Nirmal  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
1. Business Understanding

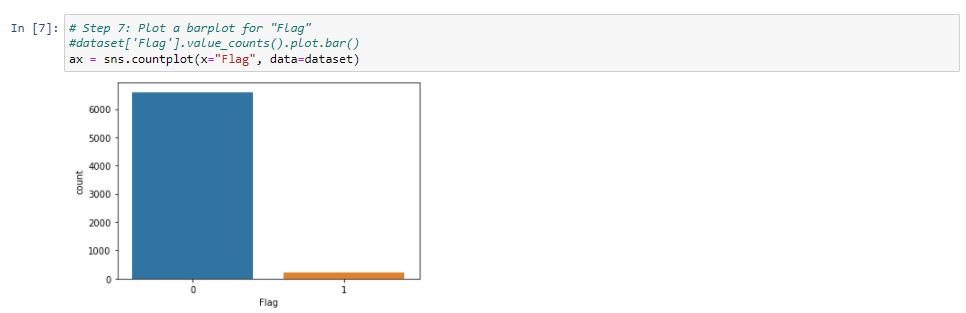
We have been tasked to assess whether a company would bankrupt in the long run. This is useful because it helps the bank save money from reducing loan defaults. Due to the highly confidential nature of the bank customer data, we used some publicly available data (as a substitute for the bank’s customer data), to experiment with predictive analytics techniques. Proxy data used is Taiwanese bankruptcy prediction.

We found a dataset on Taiwanese bankruptcy prediction, based on company financials. The dataset titled ‘TEJ - Normalized\_2013\_0430\_Taiwan\_data.csv’ is found on <https://www.kaggle.com/chihfongtsai/taiwanese-bankruptcy-prediction/tasks>.

Our task is to assess whether a company would bankrupt in the long run. This would tie back to the private bank’s business objective to reduce loan defaults – notably in the collections process (to help identify any problems soonest possible).  
  
  
  
2. Data Understanding  
  
This project focuses on data analytics – data cleaning/ preparation; exploratory data analysis; feature engineering; machine learning; upsampling; hyperparameter tuning.  
  
In terms of hardware/ software requirements, we used a personal computer installed with Python (using Jupyter notebook from Anaconda Navigator). Python is chosen because it is very widely used by the (software) developers in the bank. No software license is required.  
  
The input data file is a little special, because we had to use a python library (chardet) to identify encoding first. We found the encoding to be Big5, and this is how it looks like – some column names are in Chinese characters as seen in this screenshot.  


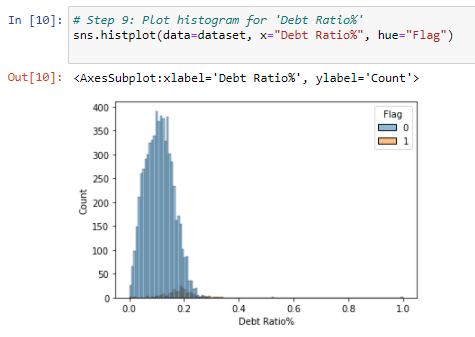
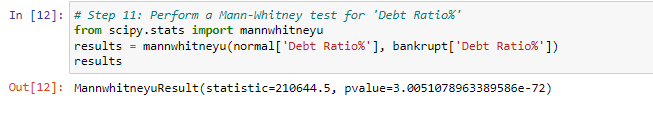
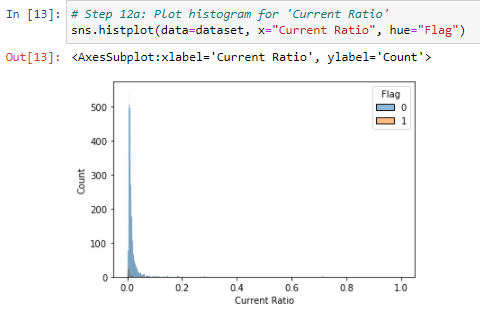
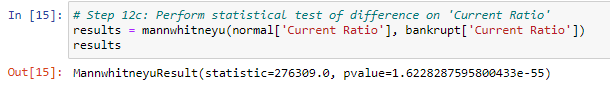
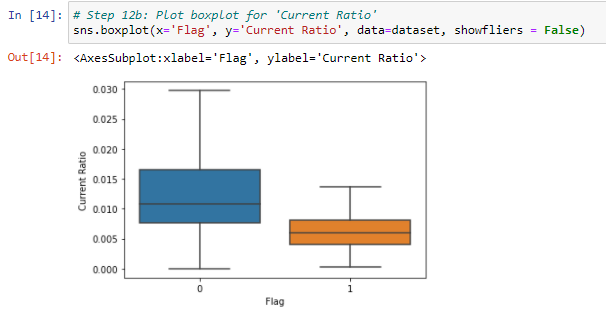
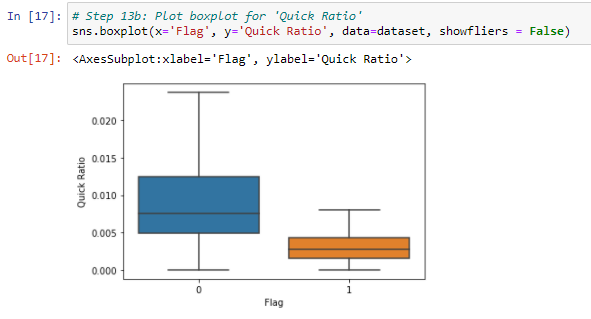
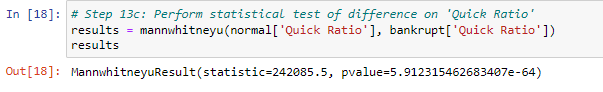
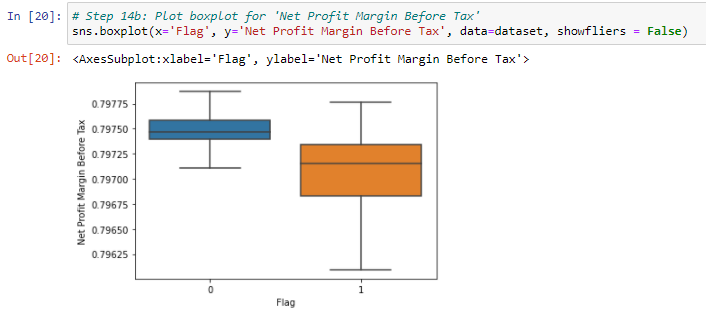
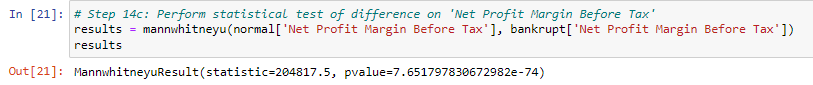
3. Data Preparation  
  
Data preparation consists of the following steps:  
a) we used chardet to identify encoding, and found it to be Big5.

b) we headed to Google Translate, to get all the Chinese column names translated into English.   
c) we fixed the casing, after we received the list of translated column names.   
d) we replaced the column names, then saved the dataset as CSV file for next steps.  
  
  
  
  
We performed some Data Checking and found the columns to be mostly OK, except for one column we found to have the values for All rows to be identical. Therefore, we dropped that column “One If Net Income Was Negative For The Last Two Year Zero Otherwise”.  
  
  
  
4. Exploratory Data Analysis  
  
The column 'Flag' indicates whether a company went bankrupt (1) or remained ok (0).

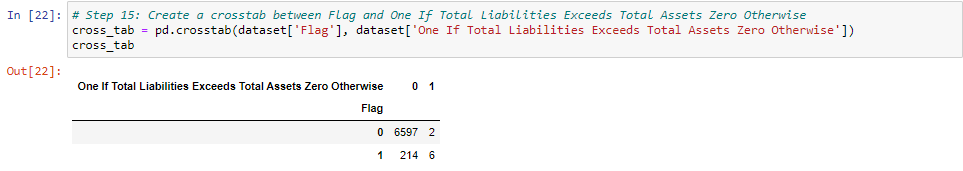
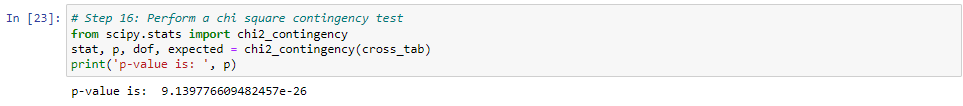
When we plotted a countplot to show the distribution between bankrupt and ok companies, we found the dataset to be (highly) imbalanced, i.e. there is a lot fewer 1 compared to 0. This will affect model training later.  
  
  
We split the Dataframe into normal and bankrupt, then performed data exploration on a few financial ratios first:

* Debt Ratio
* Current Ratio
* Quick Ratio
* Net Profit Margin Before Tax

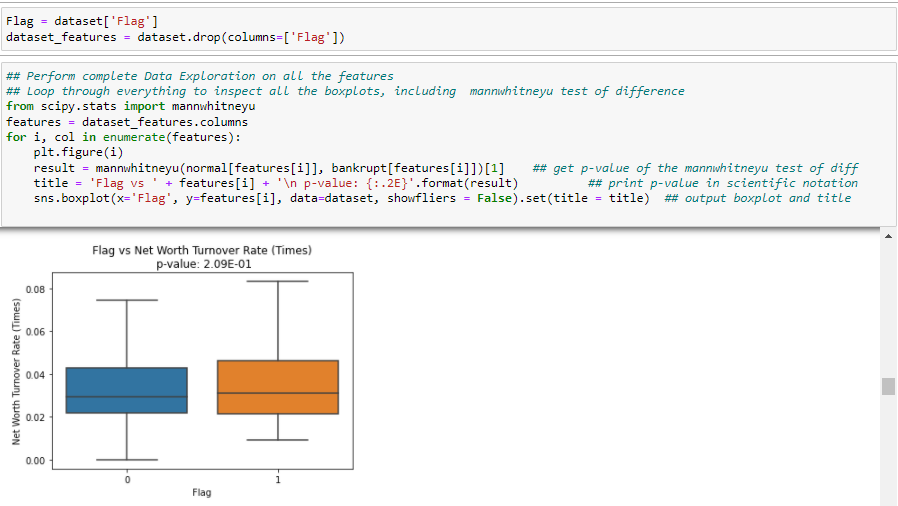
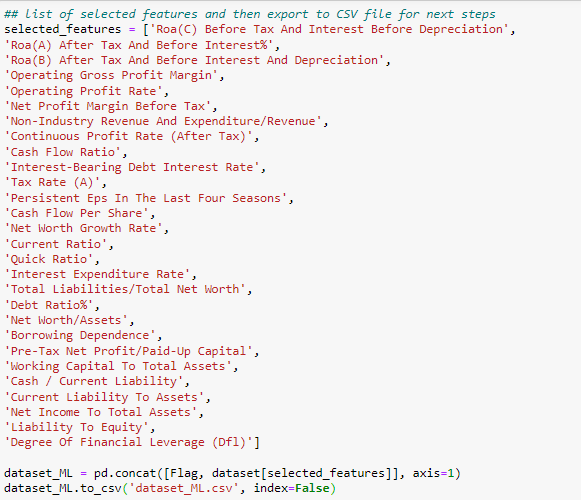


**Data exploration on Debt Ratio%** showed that the debt ratios for normal and bankrupt companies are very different – it is most obvious from the non-overlapping boxplots shown below. We also performed hypothesis testing (of difference) to check whether results are significant. Mann-Whitney U test is used, instead of a t-test, because debt ratio is not normally distributed in the dataset. The results are significant.  
  
  
  
  
  
**Data exploration on Current Ratio** showed that the current ratios for normal and bankrupt companies are somewhat different. The hypothesis test confirms result is significant.  
  
  
  
  
  
  
Boxplots and test of difference confirms the Quick Ratio and Net Profit Margin Before Tax for normal and bankrupt companies are different, as seen in the screenshots below.  
  
  
  
  
  
  
  
  
  
We have one categorical variable 'One if Total Liabilities Exceeds Total Assets Zero Otherwise’.

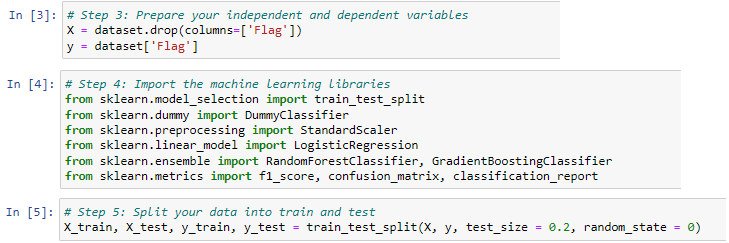
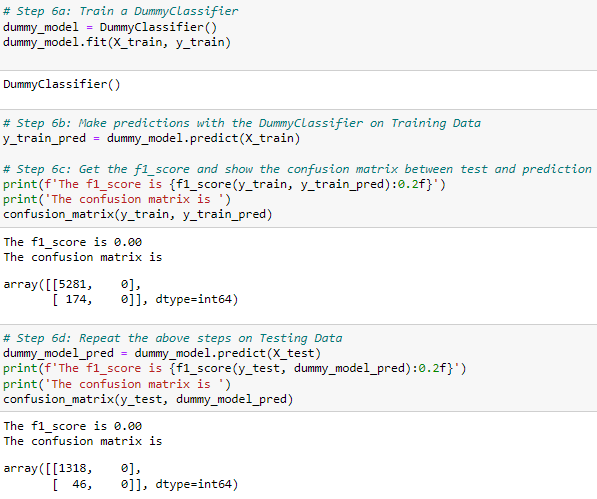
We created a crosstab to visually inspect and then perform a chi square contingency test.

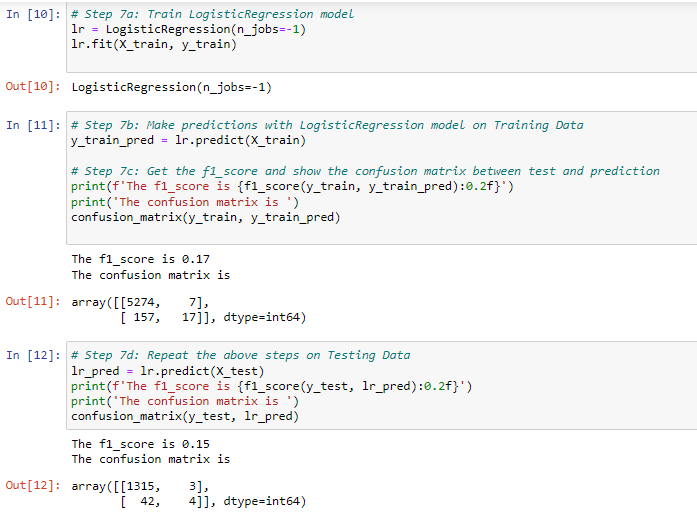
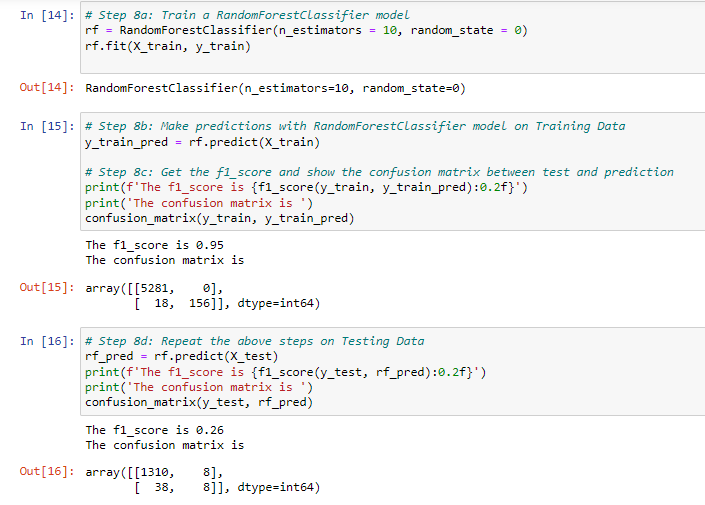
The p-value is very very small (less than 0.05), which means the two features are likely to be not independent from each other. Therefore, we also drop this feature.  
  
  
  
  
 **Data exploration on all other numeric features->**  
For completeness, we performed an extensive data exploration exercise - boxplots + test of difference – on all other (90+) features!

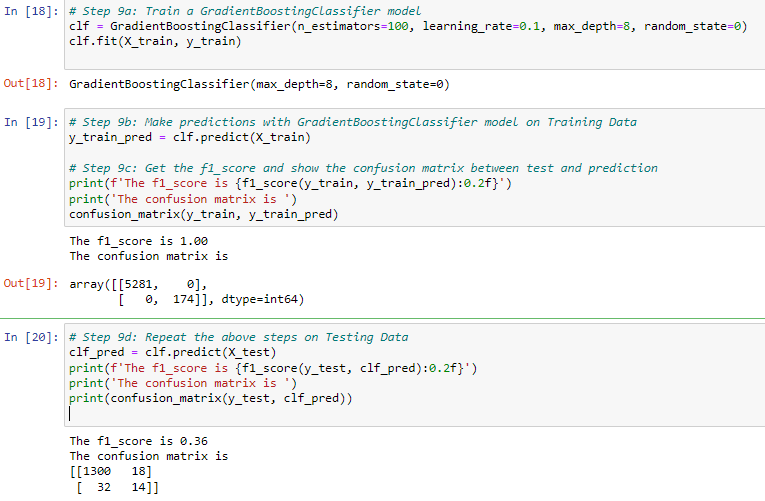
We went through all the plots to weed out those that are not statistically significant; or with noticeable overlap between the two boxplots.

  
Remove also those identical/ similar features, e.g. Current Ratio is similar to these ratios:  
- (Current Liabilities To Current Assets);  
- Current Liability To Equity;  
…  
  
After going through all the boxplots, we finally cherry-picked this list of features and then exported to a CSV file for next step – Modelling and Evaluation.  


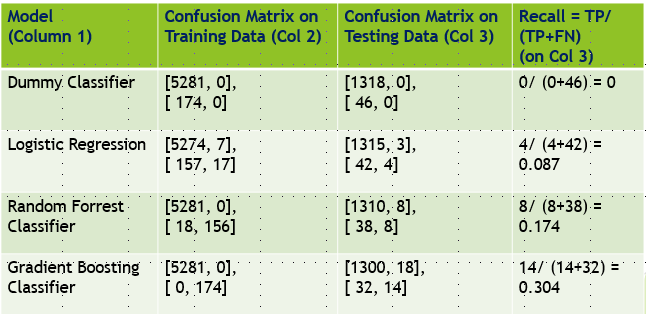
5. Modelling and Evaluation  
  
We used this standard Machine Learning process amongst the data science community.  
1. Split your data into train and test set.

2. Model creation - Import your models from sklearn and instantiate them (assign model object to a variable).  
3. Model fitting - Fit your training data into the model and train.  
4. Model prediction - Make a set of predictions using your test data.  
5. Model assessment - Compare your predictions with ground truth in test data.  
  
  
We tried the following models:  
- Dummy Classifier as the baseline/ benchmark model;  
- Logistic Regression;  
- Radom Forest Classifier; and   
- Gradient Boosting Classifier  
  
  
  
 **Dummy Classifier**  
  


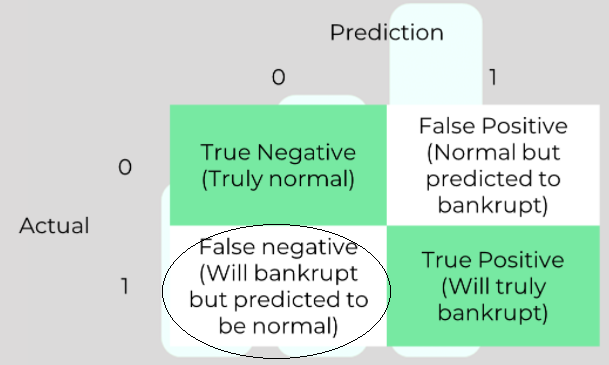
**Logistic Regression**  
  
  
**Random Forest Classifier**

**Gradient Boosting Classifier**  


Tabulating all the confusion matrix from the various models, it is not difficult to see that Gradient Boosting Classifier appears to be most promising in terms of minimizing false negatives (on the testing data).

  
  
  
  
 **Confusion Matrix – an interpretation**  
In our exercise, **false negatives** translate to lost capital (from lending to companies that go bankrupt), while **false positives** represent lost interest income (from not lending to healthy companies). When FN decrease, FP increase; and vice versa. Here, lost capital is much more important than lost interest income!

We aspire to minimize lost capital + lost interest income, hence maximize Net Amount Made.

  
  
  
**Handling Imbalanced Data**  
The modelling has proved challenging because of how imbalanced the dataset was, i.e. the bankrupt companies (minority class) is less than 5%. As a result, the model is unable to learn from minority class well.

We need to oversample minority class or undersample majority class, or mix of both.

* Over-sampling techniques: Synthetic Minority Oversampling Technique (SMOTE), Adaptive Synthetic Sampling Approach (ADASYN), etc, to add some new “information” to the training data
* Under-sampling techniques: Edited Nearest Neighbour (ENN), TomekLinks, etc, to help remove some majority class points

It is also common practice to combine both oversampling and undersampling to further improve effectiveness:

* SMOTE + ENN
* SMOTE + Tomek  
    
    
    
    
  For example, this is how the SMOTE + Tomek process works:  
  **1. (Start of SMOTE)** Choose random data from the minority class.

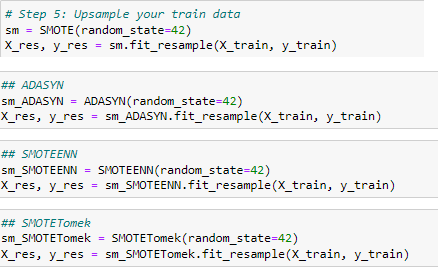
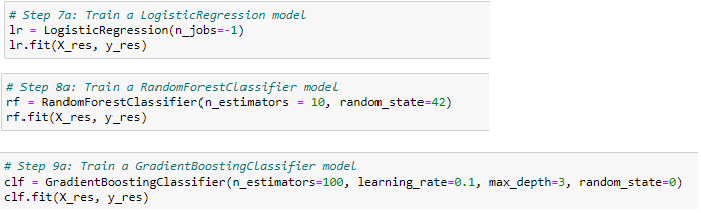
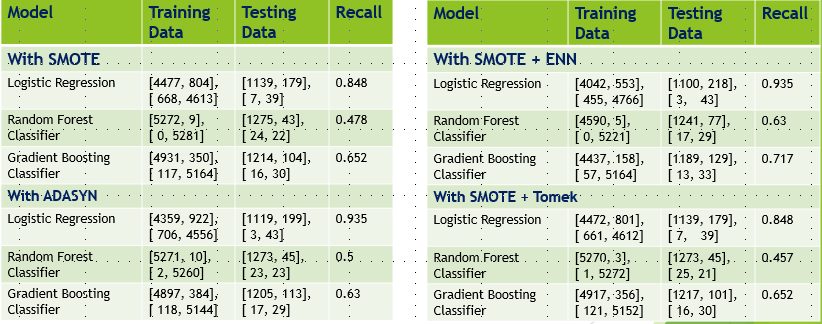
2. Calculate the distance between the random data and its k nearest neighbors.

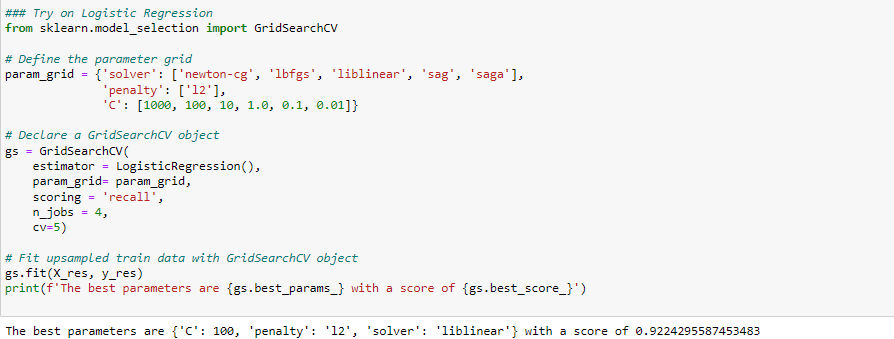
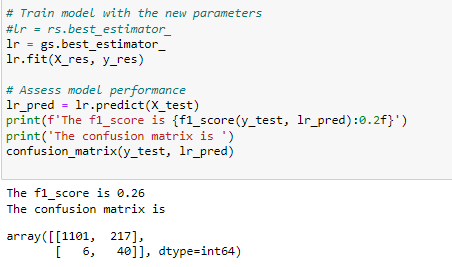
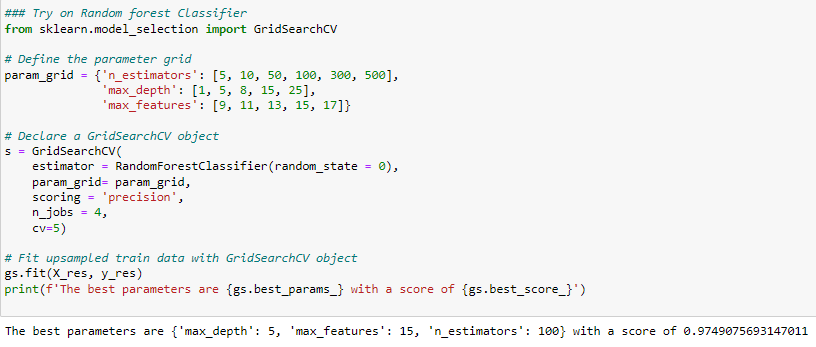
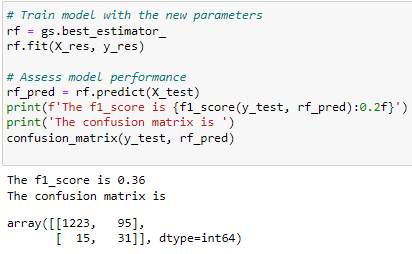
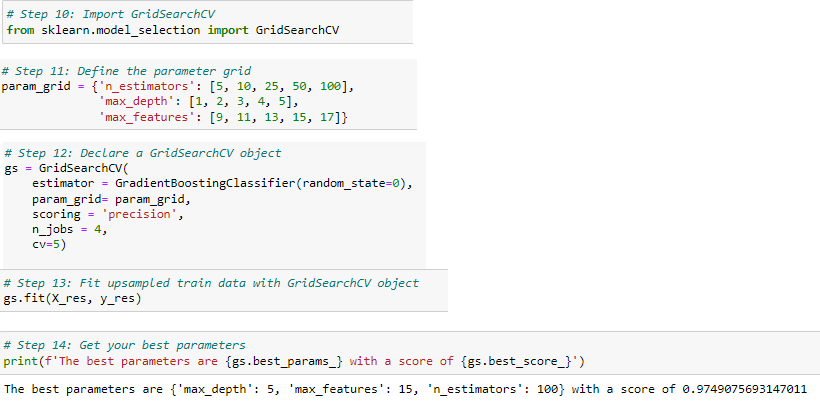
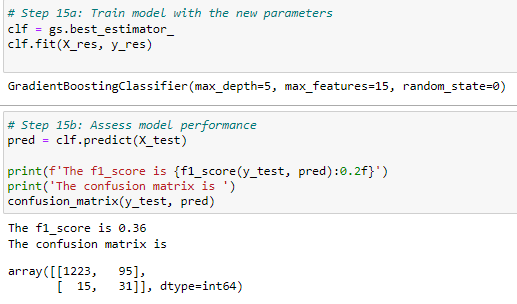
3. Multiply the difference with a random number between 0 and 1, then add the result to the minority class as a synthetic sample.

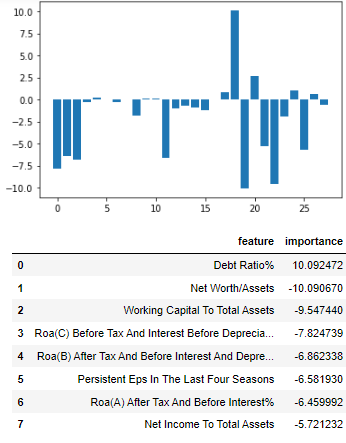
4. Repeat step number 2–3 until the desired proportion of minority class is met. **(End of SMOTE)**

**5. (Start of Tomek Links)** Choose random data from the majority class.

6. If the random data’s nearest neighbor is the data from the minority class (i.e. create the Tomek Link), then remove the Tomek Link.

**Source:** Imbalanced Classification in Python: SMOTE-Tomek Links Method<https://towardsdatascience.com/imbalanced-classification-in-python-smote-tomek-links-method-6e48dfe69bbc>  
  
  
After resampling, we reuse all our earlier code to fit model on the upsampled training data (X\_res, y\_res), then test the resulting model. Please refer to Appendix for program listing.  
  
  
  
From our results, best upsampler turn out to be SMOTE + ENN.. that surprisingly gives best result on Logistic Regression model.  
  
  
  
  
**Hyperparameter Tuning**  
Thereafter, we performed hyperparameter tuning to help us determine the right combination of hyperparameters that maximizes the model performance. We used SMOTE + ENN – the best upsampler on:

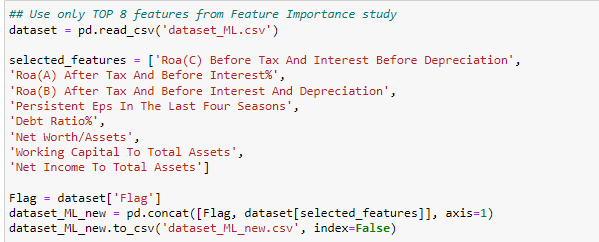
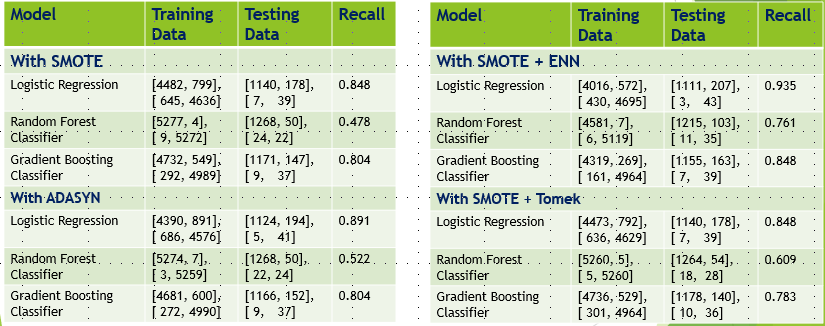
* Logistic Regression (best model)-> results became less favourable than before
* Gradient Boosting Classifier-> results improved a little bit, but insignificant
* Random Forest Classifier-> results improved a little bit, but insignificant  
    
    
    
    
    
    
    
    
    
    
  For now, we decided to settle on SMOTE + ENN upsampler on Logistic Regression model, as seen in this confusion matrix from testing data,  
  [1100, 218],   
  [ 3, 43]

**SMOTE + ENN upsampler on Logistic Regression model** – Which are the KEY features?  
  


We note that positive scores indicate a feature that predicts class 1 (bankrupt), whereas the negative scores indicate a feature that predicts class 0 (normal).  
  
  
Take the top 3 features for example:

* Higher Debt Ratio% denote higher likelihood of class 1 (becoming bankrupt)
* Higher Net Worth (as fraction of Assets) denote greater likelihood of class 0 (normal/ healthy company)
* Higher Working Capital To Total Assets denote greater likelihood of class 0 (normal/ healthy company)

…

These features do make sense intuitively.  
  
  
  
  
Finally, let’s give it our best shot – try using a refined model using TOP 8 features only!  
  
  
  
Limiting the number of features further shows general improvement across the board, especially Random Forest and Gradient Boosting Classifiers.  


6. Business Implications  
  
Having come this far, we finally need to determine which model best meets the bank’s objective -> most profitable.  
  
Let’s assume this hypothetical scenario:  
-> 1364 companies applied for loans   
-> Amount borrowed per company is $1000  
-> Interest charged is 5%  
  
  
Best Case Scenario (no bankruptcy):   
-> Amount made = 1364 x $1000 x 5% = $68,200  
  
Worst Case Scenario (if all 46 bankrupt companies were undetected):  
-> Amount made (from healthy companies) = (1364 – 46) x $1000 x 5% = $65,900  
-> Capital lost from bankrupt companies = $46,000  
-> Net amount made = $65,900 - $46,000 = $19,900  
  
For each resampling + model combination, we tabulate the following:  
-> Lost Capital = FN x $1000 (from lending to companies that go bankrupt);  
-> Lost Income = FP x $1000 x 5% (for failing to lend to healthy companies)  
We aspire to minimize Lost Capital + Lost Income; or maximize Net Amount made.  
  
  
  
With our best model after re-sampling:  
-> 1364 companies applied for loans  
-> 250 (= 207 + 43) companies predicted to be bankrupt  
Total amount lent = 1111 x $1000 x 5% = $55,550  
  
False Negatives = 3; False Positives = 207  
Lost capital = $3000; Lost interest income = 207 x $1000 x 5% = $10,350  
  
-> Net amount made = 1111 x $1000 x 5% - 3 x $1000 = $52,550

7. Conclusion  
  
In this project, we applied predictive analytics on loan default/ bankruptcy prediction use case. We leveraged on resampling techniques (for handling highly imbalanced dataset) to improve our models. We further limited to just TOP 8 features to make our models a little more robust.

Our best model (i.e. most profitable model) successfully maximized the Net Amount Made, taking into account lost capital (from False Negatives) and also lost interest income (from False Positives). Not only did the bank lend less money, the revenue from interest also increased.  
  
From our results, SMOTE + ENN, with Logistic Regression model, was most profitable and achieved highest Net Amount Made. While the results appeared to be decent, we acknowledge that over-fitting exists and can pose a challenge to deployment later, due to the relatively small dataset; over-fitting was most evident in Random Forest Classifier.  
  
  
  
8. References  
  
Handling Imbalanced Datasets with SMOTE in Python, by Juan De Dios Santos

<https://www.kite.com/blog/python/smote-python-imbalanced-learn-for-oversampling/>   
  
Imbalanced Classification in Python: SMOTE-Tomek Links Method, by Raden Aurelius Andhika Viadinugroho

<https://towardsdatascience.com/imbalanced-classification-in-python-smote-tomek-links-method-6e48dfe69bbc>

Stop using SMOTE to handle all your Imbalanced Data, by Satyam Kumar

<https://towardsdatascience.com/stop-using-smote-to-handle-all-your-imbalanced-data-34403399d3be>

Overcoming Class Imbalance using SMOTE Techniques, by SWASTIK SATPATHY

<https://towardsdatascience.com/stop-using-smote-to-handle-all-your-imbalanced-data-34403399d3be>

Hyperparameter Tuning in Python: a Complete Guide, by Shahul ES, Aayush Bajaj

<https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide>

Optimizing Hyperparameters in Random Forest Classification, by Reilly Meinert

<https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6>

Tune Hyperparameters for Classification Machine Learning Algorithms, by Jason Brownlee

<https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/>

How to Calculate Feature Importance With Python, by Jason Brownlee

<https://machinelearningmastery.com/calculate-feature-importance-with-python/>

9. Appendix - Program Listing in code snippets.py  
# -\*- coding: utf-8 -\*-

"""

Spyder Editor

Code snippets for Loan Default/ Bankruptcy Prediction

Done by: Theng Hee Yeo

TFIP-Data Analytics 2021

NUS-ISS

"""

#########################################################

##

## STAGE 1: DATA Import/ Preparation

##

##

#########################################################

# Import libraries and use chardet to identify encoding

import pandas as pd

import chardet

rawdata = open("TEJ - Normalized\_2013\_0430\_Taiwan\_data.csv", 'rb').read()

result = chardet.detect(rawdata)

charenc = result['encoding']

dataset = pd.read\_csv('TEJ - Normalized\_2013\_0430\_Taiwan\_data.csv', encoding='Big5')

dataset.columns

## Half of the columns are in Chinese,

## the data dictionary in the original UCI Machine Learning Repository is not useful because the dictionary is not in the same order.

## Therefore, head over to Google translate ... and then we get the translated list here.

translated\_list = ['Flag','ROA(C) before tax and interest before depreciation','ROA(A) after tax and before interest%','ROA(B) after tax and before interest and depreciation','Operating gross profit margin',

'Realized gross profit margin of sales','operating profit rate','net profit margin before tax','net profit margin after tax','non-industry revenue and expenditure/revenue','continuous profit rate (after tax)',

'Operating expense ratio','Research and development expense ratio','Cash flow ratio','Interest-bearing debt interest rate','Tax rate (A)','Net value per share (B)',

'Net value per share (A)','Net value per share (C)','Persistent EPS in the last four seasons','Cash flow per share','Return per share (yuan)','Operating profit per share ( Yuan)',

'Net profit per share before tax (yuan)','Realized gross profit growth rate of sales','Operating profit growth rate','After-tax net profit growth rate','Regular net profit growth rate','Continuous net profit growth rate' ,

'Total Asset Growth Rate','Net Worth Growth Rate','Return on Total Assets Growth Rate','Cash Reinvestment %','Current Ratio','Quick Ratio','Interest Expenditure Rate',

'Total liabilities/total net worth','debt ratio%','net worth/assets','long-term fund suitability ratio (A)','borrowing dependence','contingent liabilities/net worth',

'Operating profit/paid-in capital','pre-tax net profit/paid-up capital','inventory and accounts receivable/net value','total asset turnover times','accounts receivable turnover times',

'Average collection days','Inventory turnover rate (times)','Fixed asset turnover times','Net worth turnover rate (times)','Revenue per person','Operating profit per person',

'Equipment rate per person','working capital to total assets','Quick asset/Total asset',

'current assets/total assets','cash / total assets',

'Quick asset/current liabilities','cash / current liability',

'current liability to assets','operating funds to liability',

'Inventory/working capital','Inventory/current liability',

'current liability / liability','working capital/equity',

'current liability/equity','long-term liability to current assets',

'Retained Earnings/Total assets','total income / total expense',

'total expense /assets','current asset turnover rate','quick asset turnover rate','working capitcal turnover rate',

'Cash flow rate','Cash flow to Sales','fix assets to assets',

'current liability to liability','current liability to equity',

'equity to long-term liability','Cash flow to total assets',

'cash flow to liability','CFO to ASSETS','cash flow to equity',

'current liabilities to current assets',

'one if total liabilities exceeds total assets zero otherwise',

'net income to total assets','total assets to GNP price',

'No-credit interval','Gross profit to Sales',

'Net income to stockholder\'s Equity','liability to equity',

'Degree of financial leverage (DFL)',

'Interest coverage ratio( Interest expense to EBIT )',

'one if net income was negative for the last two year zero otherwise',

'equity to liability']

# Make your column casing consistent

translated\_list\_title = []

for i in range(len(translated\_list)):

translated\_list\_title.append(translated\_list[i].title())

dataset.columns = translated\_list\_title

dataset.to\_csv('dataset.csv', index=False) # Export the DataFrame as CSV

#########################################################

##

## STAGE 2: Exploratory Data Analysis

##

##

#########################################################

# Import libraries for Data Exploration

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import scipy as sp

pd.options.display.max\_rows = 999 # Set pandas to display all columns

dataset = pd.read\_csv('dataset.csv')

dataset.describe

# We found one row with all rows identical, hence we drop that column

dataset.drop(columns=["One If Net Income Was Negative For The Last Two Year Zero Otherwise"], inplace=True)

# Plot a barplot for "Flag"

ax = sns.countplot(x="Flag", data=dataset)

# Split Dataframe into normal and bankrupt

normal = dataset[dataset.Flag == 0]

bankrupt = dataset[dataset.Flag == 1]

# Data exploration for 'Debt Ratio%'

sns.histplot(data=dataset, x="Debt Ratio%", hue="Flag") # Plot histogram

sns.boxplot(x='Flag', y='Debt Ratio%', data=dataset, showfliers = False) # plot a boxplot

from scipy.stats import mannwhitneyu

results = mannwhitneyu(normal['Debt Ratio%'], bankrupt['Debt Ratio%']) # Perform a Mann-Whitney test

results

# Data exploration for 'Current ratio%'

sns.histplot(data=dataset, x="Current Ratio", hue="Flag")

sns.boxplot(x='Flag', y='Current Ratio', data=dataset, showfliers = False)

results = mannwhitneyu(normal['Current Ratio'], bankrupt['Current Ratio'])

# Data exploration for 'Quick ratio%'

sns.histplot(data=dataset, x="Quick Ratio", hue="Flag")

sns.boxplot(x='Flag', y='Quick Ratio', data=dataset, showfliers = False)

results = mannwhitneyu(normal['Quick Ratio'], bankrupt['Quick Ratio'])

# Data exploration for 'Net Profit Margin Before Tax'

sns.histplot(data=dataset, x="Net Profit Margin Before Tax", hue="Flag")

sns.boxplot(x='Flag', y='Net Profit Margin Before Tax', data=dataset, showfliers = False)

results = mannwhitneyu(normal['Net Profit Margin Before Tax'], bankrupt['Net Profit Margin Before Tax'])

# Create a crosstab between Flag and One If Total Liabilities Exceeds Total Assets Zero Otherwise

cross\_tab = pd.crosstab(dataset['Flag'], dataset['One If Total Liabilities Exceeds Total Assets Zero Otherwise'])

from scipy.stats import chi2\_contingency

stat, p, dof, expected = chi2\_contingency(cross\_tab)

print('p-value is: ', p)

## Drop also that column that is Not independent of flag

dataset.drop(columns=['One If Total Liabilities Exceeds Total Assets Zero Otherwise'], inplace=True)

## Perform complete Data Exploration on all the features

## Loop through everything to inspect all the boxplots, including mannwhitneyu test of difference

from scipy.stats import mannwhitneyu

Flag = dataset['Flag']

dataset\_features = dataset.drop(columns=['Flag'])

features = dataset\_features.columns

for i, col in enumerate(features):

plt.figure(i)

result = mannwhitneyu(normal[features[i]], bankrupt[features[i]])[1] ## p-value of mannwhitneyu test of diff

title = 'Flag vs ' + features[i] + '\n p-value: {:.2E}'.format(result) ## print p-value in scientific notation

sns.boxplot(x='Flag', y=features[i], data=dataset, showfliers = False).set(title = title) ## output boxplot and title

## list of selected features and then export to CSV file for next steps

selected\_features = ['Roa(C) Before Tax And Interest Before Depreciation',

'Roa(A) After Tax And Before Interest%',

'Roa(B) After Tax And Before Interest And Depreciation',

'Operating Gross Profit Margin',

'Operating Profit Rate',

'Net Profit Margin Before Tax',

'Non-Industry Revenue And Expenditure/Revenue',

'Continuous Profit Rate (After Tax)',

'Cash Flow Ratio',

'Interest-Bearing Debt Interest Rate',

'Tax Rate (A)',

'Persistent Eps In The Last Four Seasons',

'Cash Flow Per Share',

'Net Worth Growth Rate',

'Current Ratio',

'Quick Ratio',

'Interest Expenditure Rate',

'Total Liabilities/Total Net Worth',

'Debt Ratio%',

'Net Worth/Assets',

'Borrowing Dependence',

'Pre-Tax Net Profit/Paid-Up Capital',

'Working Capital To Total Assets',

'Cash / Current Liability',

'Current Liability To Assets',

'Net Income To Total Assets',

'Liability To Equity',

'Degree Of Financial Leverage (Dfl)']

dataset\_ML = pd.concat([Flag, dataset[selected\_features]], axis=1)

dataset\_ML.to\_csv('dataset\_ML.csv', index=False)

#########################################################

##

## STAGE 3: Machine Learning

##

##

#########################################################

# Steps 1 and 2: Import libraries and read datafile

import pandas as pd

pd.set\_option("display.precision", 4)

dataset = pd.read\_csv('dataset\_ML.csv')

# Step 3: Prepare your independent and dependent variables

X = dataset.drop(columns=['Flag'])

y = dataset['Flag']

# Step 4: Import the machine learning libraries

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

from sklearn.dummy import DummyClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.metrics import f1\_score, confusion\_matrix, classification\_report

# Step 5: Split your data into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

# Step 6a: Train a DummyClassifier

dummy\_model = DummyClassifier()

dummy\_model.fit(X\_train, y\_train)

# Step 6b: Make predictions with the DummyClassifier on Training Data

y\_train\_pred = dummy\_model.predict(X\_train)

# Step 6c: Get the f1\_score and show the confusion matrix between test and prediction

print(f'The f1\_score is {f1\_score(y\_train, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_train, y\_train\_pred)

# Step 6d: Repeat the above steps on Testing Data

dummy\_model\_pred = dummy\_model.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, dummy\_model\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, dummy\_model\_pred)

# Step 6e: Print the classification report between test and prediction

print(classification\_report(y\_test, dummy\_model\_pred))

# Step 7a: Train LogisticRegression model

lr = LogisticRegression(n\_jobs=-1)

lr.fit(X\_train, y\_train)

# Step 7b: Make predictions with LogisticRegression model on Training Data

y\_train\_pred = lr.predict(X\_train)

# Step 7c: Get the f1\_score and show the confusion matrix between test and prediction

print(f'The f1\_score is {f1\_score(y\_train, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_train, y\_train\_pred)

# Step 7d: Repeat the above steps on Testing Data

lr\_pred = lr.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, lr\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, lr\_pred)

# Step 7e: Print the classification report

print(classification\_report(y\_test, lr\_pred))

# Step 8a: Train a RandomForestClassifier model

rf = RandomForestClassifier(n\_estimators = 10, random\_state = 0)

rf.fit(X\_train, y\_train)

# Step 8b: Make predictions with RandomForestClassifier model on Training Data

y\_train\_pred = rf.predict(X\_train)

# Step 8c: Get the f1\_score and show the confusion matrix between test and prediction

print(f'The f1\_score is {f1\_score(y\_train, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_train, y\_train\_pred)

# Step 8d: Repeat the above steps on Testing Data

rf\_pred = rf.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, rf\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, rf\_pred)

# Step 8e: Print the classification report

print(classification\_report(y\_test, rf\_pred))

# Step 9a: Train a GradientBoostingClassifier model

clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.1, max\_depth=8, random\_state=0)

clf.fit(X\_train, y\_train)

# Step 9b: Make predictions with GradientBoostingClassifier model on Training Data

y\_train\_pred = clf.predict(X\_train)

# Step 9c: Get the f1\_score and show the confusion matrix between test and prediction

print(f'The f1\_score is {f1\_score(y\_train, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_train, y\_train\_pred)

# Step 9d: Repeat the above steps on Testing Data

clf\_pred = clf.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, clf\_pred):0.2f}')

print('The confusion matrix is ')

print(confusion\_matrix(y\_test, clf\_pred))

# Step 10: Get the feature importances of the model

feature\_importance = list(zip(X\_train.columns, clf.feature\_importances\_))

feature\_importance.sort(key=lambda x: x[1], reverse=True)

feature\_importance = pd.DataFrame(feature\_importance, columns=['feature', 'importance'])

#feature\_importance['importance'] = round(feature\_importance['importance'], 5)

feature\_importance.head(5)

#########################################################

##

## STAGE 4: Advanced Modelling

## Re-sampling

##

#########################################################

# Step 1: Import your libraries

import pandas as pd

from matplotlib import pyplot

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

from sklearn.dummy import DummyClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.metrics import f1\_score, confusion\_matrix, classification\_report

from imblearn.over\_sampling import SMOTE, ADASYN

from imblearn.combine import SMOTEENN, SMOTETomek

# Step 2: Read CSV from Part II

dataset = pd.read\_csv('dataset\_ML.csv')

# Step 3: Prepare your independent and dependent variables

X = dataset.drop(columns=['Flag'])

y = dataset['Flag']

# Step 4: Split data into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

# Step 5: Upsample your train data

## SMOTE

sm = SMOTE(random\_state=42)

X\_res, y\_res = sm.fit\_resample(X\_train, y\_train)

## ADASYN

sm\_ADASYN = ADASYN(random\_state=42)

X\_res, y\_res = sm\_ADASYN.fit\_resample(X\_train, y\_train)

## SMOTEENN

sm\_SMOTEENN = SMOTEENN(random\_state=42)

X\_res, y\_res = sm\_SMOTEENN.fit\_resample(X\_train, y\_train)

## SMOTETomek

sm\_SMOTETomek = SMOTETomek(random\_state=42)

X\_res, y\_res = sm\_SMOTETomek.fit\_resample(X\_train, y\_train)

#### The following code is for the various models - to reuse with each upsampler above ####

############### LOGISTIC REGRESSION ###############

# Step 7a: Train a LogisticRegression model

lr = LogisticRegression(n\_jobs=-1)

lr.fit(X\_res, y\_res)

# Step 7b: Assess LogisticRegression model on Training Data

y\_train\_pred = lr.predict(X\_res)

print(f'The f1\_score is {f1\_score(y\_res, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_res, y\_train\_pred)

# Step 7c: Assess LogisticRegression model on Testing Data

lr\_pred = lr.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, lr\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, lr\_pred)

# get feature importance, display and plot

importance = lr.coef\_[0]

for i,v in enumerate(importance):

print('Feature: %0d, Score: %.5f' % (i,v))

pyplot.bar([x for x in range(len(importance))], importance)

pyplot.show()

importance = lr.coef\_[0]

feature\_importance = list(zip(X\_train.columns, importance, abs(importance)))

feature\_importance.sort(key=lambda x: x[2], reverse=True)

feature\_importance = pd.DataFrame(feature\_importance, columns=['feature', 'importance', 'rank'])

feature\_importance = feature\_importance[['feature', 'importance']]

feature\_importance.head(10)

############### RANDOM FOREST CLASSIFIER ###############

# Step 8a: Train a RandomForestClassifier model

rf = RandomForestClassifier(n\_estimators = 10, random\_state=42)

rf.fit(X\_res, y\_res)

# Step 8b: Assess model performance on Training Data

y\_train\_pred = rf.predict(X\_res)

print(f'The f1\_score is {f1\_score(y\_res, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_res, y\_train\_pred)

# Step 8c: Assess model performance on Testing Data

rf\_pred = rf.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, rf\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, rf\_pred)

############### GRADIENT BOOSTING CLASSIFIER ###############

# Step 9a: Train a GradientBoostingClassifier model

clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.1, max\_depth=3, random\_state=0)

clf.fit(X\_res, y\_res)

# Step 9b: Assess model performance on Training Data

y\_train\_pred = clf.predict(X\_res)

print(f'The f1\_score is {f1\_score(y\_res, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_res, y\_train\_pred)

# Step 9c: Assess model performance on Testing Data

clf\_pred = clf.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, clf\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, clf\_pred)

#########################################################

##

## STAGE 4: Advanced Modelling

## Hyperparameter Tuning

##

#########################################################

# Step 10: Import GridSearchCV

from sklearn.model\_selection import GridSearchCV

# Step 11: Define the parameter grid

param\_grid = {'n\_estimators': [5, 10, 25, 50, 100],

'max\_depth': [1, 2, 3, 4, 5],

'max\_features': [9, 11, 13, 15, 17]}

# Step 12: Declare a GridSearchCV object

gs = GridSearchCV(

estimator = GradientBoostingClassifier(random\_state=0),

param\_grid= param\_grid,

scoring = 'recall',

n\_jobs = 4,

cv=5)

# Step 13: Fit upsampled train data with GridSearchCV object

gs.fit(X\_res, y\_res)

# Step 14: Get your best parameters

print(f'The best parameters are {gs.best\_params\_} with a score of {gs.best\_score\_}')

# Step 15a: Train model with the new parameters

clf = gs.best\_estimator\_

clf.fit(X\_res, y\_res)

# Step 15b: Assess model performance

pred = clf.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, pred)

##### Try on Random forest Classifier #####

from sklearn.model\_selection import GridSearchCV

# Define the parameter grid

param\_grid = {'n\_estimators': [5, 10, 50, 100, 300, 500],

'max\_depth': [1, 3, 5, 8, 15],

'max\_features': [9, 11, 13, 15, 17]}

# Declare a GridSearchCV object

gs = GridSearchCV(

estimator = RandomForestClassifier(random\_state = 0),

param\_grid= param\_grid,

scoring = 'recall',

n\_jobs = 4,

cv=5)

# Fit upsampled train data with GridSearchCV object

gs.fit(X\_res, y\_res)

print(f'The best parameters are {gs.best\_params\_} with a score of {gs.best\_score\_}')

# Train model with the new parameters

rf = gs.best\_estimator\_

rf.fit(X\_res, y\_res)

# Assess model performance

rf\_pred = rf.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, rf\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, rf\_pred)

##### Try on Logistic Regression #####

from sklearn.model\_selection import GridSearchCV

# Define the parameter grid

param\_grid = {'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],

'penalty': ['l2'],

'C': [1000, 100, 10, 1.0, 0.1, 0.01]}

# Declare a GridSearchCV object

gs = GridSearchCV(

estimator = LogisticRegression(),

param\_grid= param\_grid,

scoring = 'recall',

n\_jobs = 4,

cv=5)

# Fit upsampled train data with GridSearchCV object

gs.fit(X\_res, y\_res)

print(f'The best parameters are {gs.best\_params\_} with a score of {gs.best\_score\_}')

# Train model with the new parameters

#lr = rs.best\_estimator\_

lr = gs.best\_estimator\_

lr.fit(X\_res, y\_res)

# Assess model performance

lr\_pred = lr.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, lr\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, lr\_pred)

#########################################################

##

## STAGE 4: Advanced Modelling

## Feature Engineering

##

#########################################################

# Step 1: Import your libraries

import pandas as pd

from matplotlib import pyplot

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

from sklearn.dummy import DummyClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.metrics import f1\_score, confusion\_matrix, classification\_report

from imblearn.over\_sampling import SMOTE, ADASYN

from imblearn.combine import SMOTEENN, SMOTETomek

## Step 2: Use only TOP 8 features from Feature Importance study

dataset = pd.read\_csv('dataset\_ML.csv')

selected\_features = ['Roa(C) Before Tax And Interest Before Depreciation',

'Roa(A) After Tax And Before Interest%',

'Roa(B) After Tax And Before Interest And Depreciation',

'Persistent Eps In The Last Four Seasons',

'Debt Ratio%',

'Net Worth/Assets',

'Working Capital To Total Assets',

'Net Income To Total Assets']

Flag = dataset['Flag']

dataset\_ML\_new = pd.concat([Flag, dataset[selected\_features]], axis=1)

dataset\_ML\_new.to\_csv('dataset\_ML\_new.csv', index=False)

# Step 3: Read from subset file, then decide your independent and dependent variables

dataset = pd.read\_csv('dataset\_ML\_new.csv')

X = dataset.drop(columns=['Flag'])

y = dataset['Flag']

# Step 4: Split your data into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

# Step 5: Upsample your train data

sm = SMOTE(random\_state=42)

X\_res, y\_res = sm.fit\_resample(X\_train, y\_train)

## ADASYN

sm\_ADASYN = ADASYN(random\_state=42)

X\_res, y\_res = sm\_ADASYN.fit\_resample(X\_train, y\_train)

## SMOTEENN

sm\_SMOTEENN = SMOTEENN(random\_state=42)

X\_res, y\_res = sm\_SMOTEENN.fit\_resample(X\_train, y\_train)

## SMOTETomek

sm\_SMOTETomek = SMOTETomek(random\_state=42)

X\_res, y\_res = sm\_SMOTETomek.fit\_resample(X\_train, y\_train)

#### The following code is for the various models - to reuse with each upsampler above ####

############### LOGISTIC REGRESSION ###############

# Step 7a: Train a LogisticRegression model

lr = LogisticRegression(n\_jobs=-1)

lr.fit(X\_res, y\_res)

# Step 7b: Assess LogisticRegression model on Training Data

y\_train\_pred = lr.predict(X\_res)

print(f'The f1\_score is {f1\_score(y\_res, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_res, y\_train\_pred)

# Step 7c: Assess LogisticRegression model on Testing Data

lr\_pred = lr.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, lr\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, lr\_pred)

# get feature importance, display and plot

importance = lr.coef\_[0]

for i,v in enumerate(importance):

print('Feature: %0d, Score: %.5f' % (i,v))

pyplot.bar([x for x in range(len(importance))], importance)

pyplot.show()

importance = lr.coef\_[0]

feature\_importance = list(zip(X\_train.columns, importance, abs(importance)))

feature\_importance.sort(key=lambda x: x[2], reverse=True)

feature\_importance = pd.DataFrame(feature\_importance, columns=['feature', 'importance', 'rank'])

feature\_importance = feature\_importance[['feature', 'importance']]

feature\_importance.head(10)

############### RANDOM FOREST CLASSIFIER ###############

# Step 8a: Train a RandomForestClassifier model

rf = RandomForestClassifier(n\_estimators = 10, random\_state=42)

rf.fit(X\_res, y\_res)

# Step 8b: Assess model performance on Training Data

y\_train\_pred = rf.predict(X\_res)

print(f'The f1\_score is {f1\_score(y\_res, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_res, y\_train\_pred)

# Step 8c: Assess model performance on Testing Data

rf\_pred = rf.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, rf\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, rf\_pred)

############### GRADIENT BOOSTING CLASSIFIER ###############

# Step 9a: Train a GradientBoostingClassifier model

clf = GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.1, max\_depth=3, random\_state=0)

clf.fit(X\_res, y\_res)

# Step 9b: Assess model performance on Training Data

y\_train\_pred = clf.predict(X\_res)

print(f'The f1\_score is {f1\_score(y\_res, y\_train\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_res, y\_train\_pred)

# Step 9c: Assess model performance on Testing Data

clf\_pred = clf.predict(X\_test)

print(f'The f1\_score is {f1\_score(y\_test, clf\_pred):0.2f}')

print('The confusion matrix is ')

confusion\_matrix(y\_test, clf\_pred)