Machine Learning and Big Data
DATA 622

CUNY
School of Professional Studies

Professor Joseph Sabelja

Date: Nov 13, 2022

(HW3)

Exploratory Analysis and Essay

PPREPARED BY

ANJAL HUSSAN

Table of Contents

Introduction

Data Analysis is the statistics and probability to figure out trends in the data set. It helps in drilling down the information, to transform metrics, facts, and figures into initiatives for improvement. In this essay, we will explore the data using python and the supporting libraries.

Data

The data set we are going to use is the Taiwanese Bankruptcy Prediction Data Set publicly available in the UCI Machine earning Repository (https://archive.ics.uci.edu/ml/datasets/Taiwanese+Bankruptcy+Prediction)

The data were collected from the Taiwan Economic Journal for the years 1999 to 2009. Company bankruptcy was defined based on the business regulations of the Taiwan Stock Exchange

Exploratory Data Analysis

We will explore the data sets and perform the exploratory data analysis in python.

First, we start by importing libraries in the Python notebook:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from scipy import stats
from sklearn import linear_model
from sklearn import tree
from sklearn import ensemble
from sklearn import neural_network
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
#import scikitplot as skplt
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
```

Once we import the libraries and load the csv files, we explore the shape of the data frame:

```
[3]: data= pd.read_csv("data.csv")
     pd.set_option('display.max_columns', None)
```

```
[4]: data.shape
```

```
[4]: (6819, 96)
```

Let's take a look to the head of the dataset. For our screenshot, we are going to display first few columns of the data set.

```
[12]: data.head()
```

[12]:

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After-tax net Interest Rate	Non-industry income and expenditure/revenue	Continuous interest rate (after tax)	Ex
0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.808809	0.302646	0.780985	1.2
1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.809301	0.303556	0.781506	2.8
2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.808388	0.302035	0.780284	2.3
3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.808966	0.303350	0.781241	1.0
4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.809304	0.303475	0.781550	7.8

Is there any null value in the data frame?

```
[13]: data.isnull().any().any()
```

```
[13]: False
```

Let's find out the name of the columns in this data frame:

```
#all columns
for name in data:
    print(name)
```

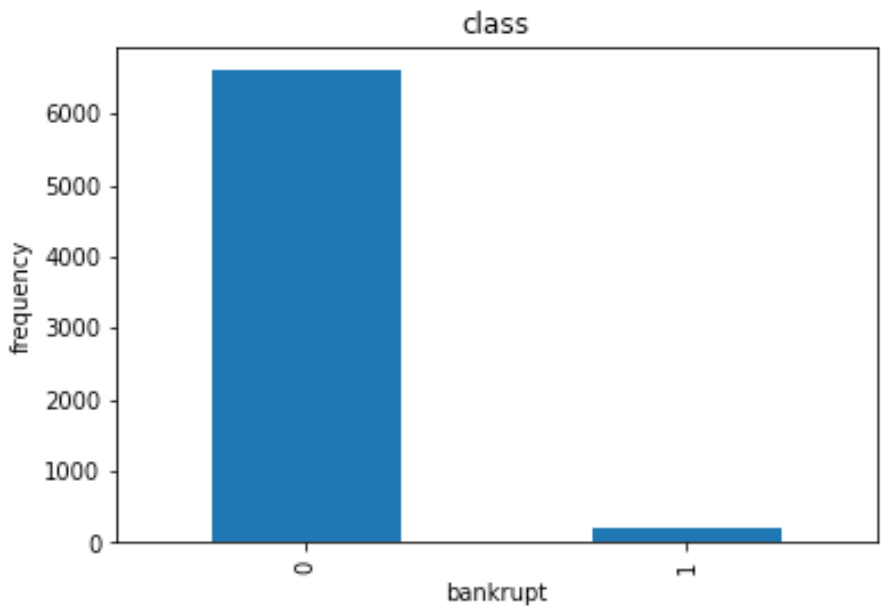
Bankrupt?	Operating Gross Margin	After-tax net Interest Rate
ROA(C) before interest and depreciation before interest	Realized Sales Gross Margin	Non-industry income and expenditure/revenue
ROA(A) before interest and % after tax	Operating Profit Rate	Continuous interest rate (after tax)
ROA(B) before interest and depreciation after tax	Pre-tax net Interest Rate	Operating Expense Rate
Research and development expense rate	Net Value Per Share (A)	Operating Profit Per Share (Yuan ¥)
Cash flow rate	Net Value Per Share (C)	Per Share Net profit before tax (Yuan ¥)
Interest-bearing debt interest rate	Persistent EPS in the Last Four Seasons	Realized Sales Gross Profit Growth Rate
Tax rate (A)	Cash Flow Per Share	Operating Profit Growth Rate
Net Value Per Share (B)	Revenue Per Share (Yuan ¥)	After-tax Net Profit Growth Rate
Regular Net Profit Growth Rate	Cash Reinvestment %	Debt ratio %
Continuous Net Profit Growth Rate	Current Ratio	Net worth/Assets
Total Asset Growth Rate	Quick Ratio	Long-term fund suitability ratio (A)
Net Value Growth Rate	Interest Expense Ratio	Borrowing dependency
Total Asset Return Growth Rate Ratio	Total debt/Total net worth	Contingent liabilities/Net worth

Operating profit/Paid-in capital	Average Collection Days	Operating profit per person
Net profit before tax/Paid-in capital	Inventory Turnover Rate (times)	Allocation rate per person
Inventory and accounts receivable/Net value	Fixed Assets Turnover Frequency	Working Capital to Total Assets
Total Asset Turnover	Net Worth Turnover Rate (times)	Quick Assets/Total Assets
Cash/Total Assets	Revenue per person	Current Assets/Total Assets

For the sake of simplicity, some of the variable names have been dropped from the above table.

Let’s look at the column Bankrupt? Because this column is one of the columns that we are going predict based on the other variables.

```
[38]: # Plot class balance
data["Bankrupt?"].value_counts().plot(
    kind="bar",
    xlabel="bankrupt",
    ylabel="frequency",
    title="class"

);
```



Since the dataset contains 96 variables and a lot of the columns we are not going to user in our training, let's make a new data frame using the variables that will be in our interest to predict the target variable.

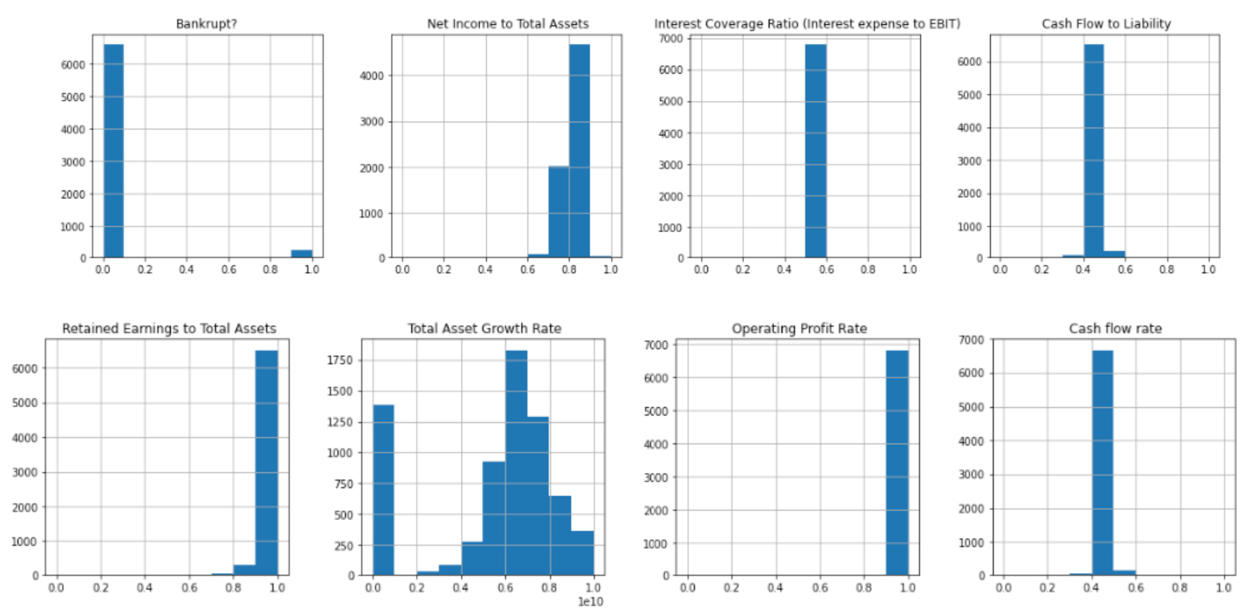
[8]:
```
df = data[["Bankrupt?", " Net Income to Total Assets", " Interest Coverage Ratio (Interest expense to EBIT)",
        " Cash Flow to Liability", " Retained Earnings to Total Assets", " Total Asset Growth Rate",
        " Operating Profit Rate", " Cash flow rate", " After-tax net Interest Rate", " Operating Profit Per Share (Yuan ¥)",
        " Operating Expense Rate", " Cash Flow to Sales", " Cash Reinvestment %", " Cash Flow Per Share",
        " CFO to Assets", " Cash/Current Liability"]]
```

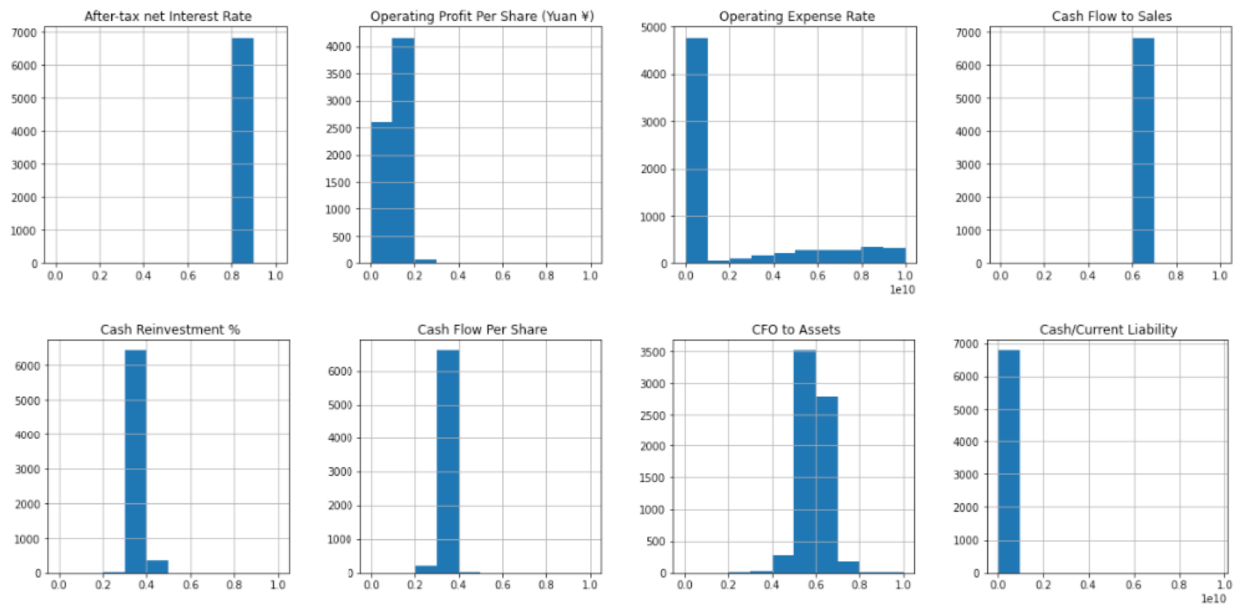
[17]:
```
df.dtypes
```

[17]:
```
Bankrupt?                                              int64
 Net Income to Total Assets                          float64
 Interest Coverage Ratio (Interest expense to EBIT)  float64
 Cash Flow to Liability                              float64
 Retained Earnings to Total Assets                   float64
 Total Asset Growth Rate                             float64
 Operating Profit Rate                               float64
 Cash flow rate                                      float64
 After-tax net Interest Rate                         float64
 Operating Profit Per Share (Yuan ¥)                 float64
 Operating Expense Rate                              float64
 Cash Flow to Sales                                  float64
 Cash Reinvestment %                                 float64
 Cash Flow Per Share                                 float64
 CFO to Assets                                       float64
 Cash/Current Liability                              float64
dtype: object
```

Now let's look at the distribution of the data from our data frame

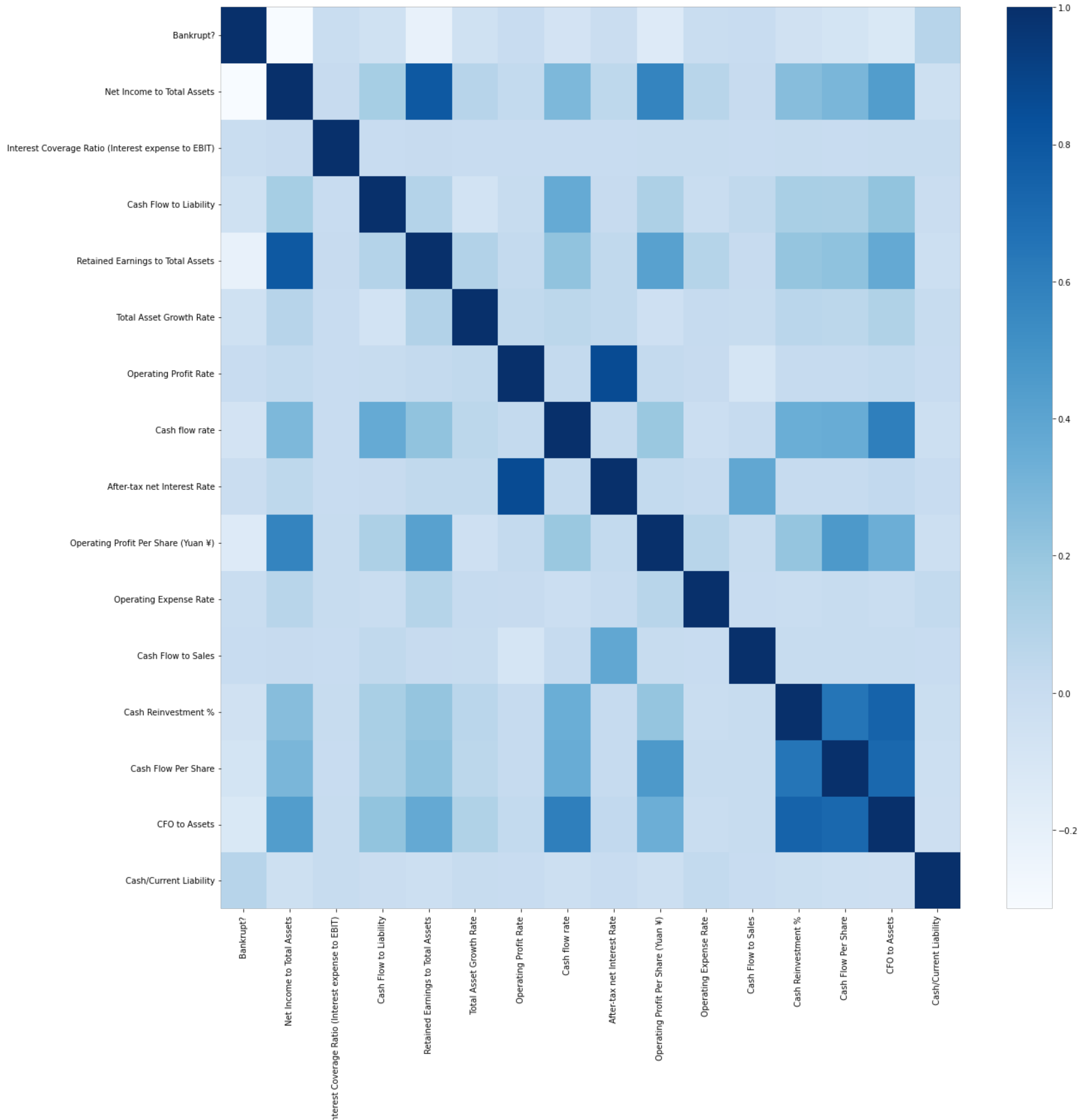
0]:
```
#distribution of all the features
df.hist(figsize=(20,20))
```





Correlations

The correlations between variable are very important to draw any conclusion regarding the relationship between data points. Looking at the row data, it's hard to find any relationship however, looking at the visual, it's obvious to establish a correlation. Let's construct correlation matrix to observe the strength of relationships of each variable with bankruptcy.



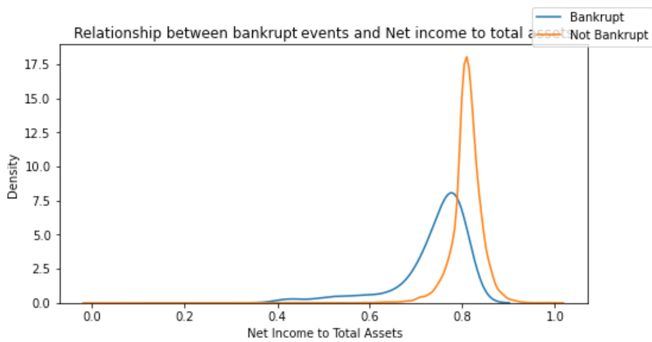
From the distribution and the heatmap, there are a lot of multicollinearity issues, skewed features and the data is imbalanced.

Let's look at the relationship between some variables:

Does Net Income to Total Assets has any relationship with being bankrupt?

```
[24]: fig = plt.figure(figsize=(8,4))
      sns.kdeplot(df[df['Bankrupt?']==1][' Net Income to Total Assets'])
      sns.kdeplot(df[df['Bankrupt?']==0][' Net Income to Total Assets'])
      fig.legend(labels=['Bankrupt', 'Not Bankrupt'])
      plt.title('Relationship between bankrupt events and Net income to total assets')
```

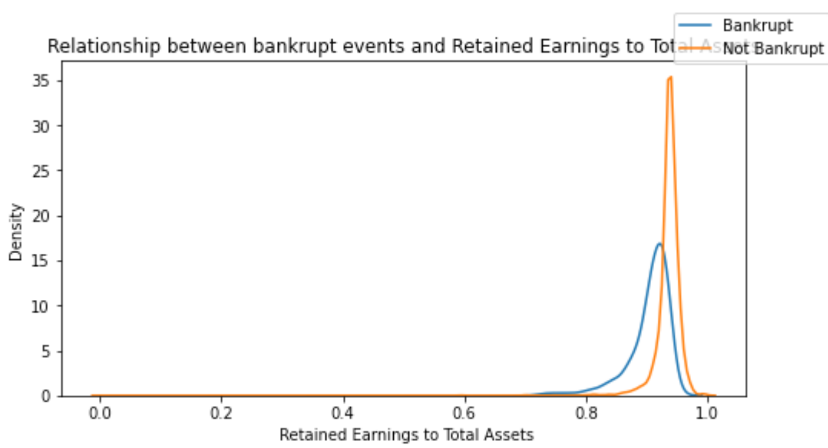
```
[24]: Text(0.5, 1.0, 'Relationship between bankrupt events and Net income to total assets')
```



Does Retained Earnings to Total Assets has any relationship with being bankrupt?

```
[25]: fig = plt.figure(figsize=(8,4))
      sns.kdeplot(df[df['Bankrupt?']==1][' Retained Earnings to Total Assets'])
      sns.kdeplot(df[df['Bankrupt?']==0][' Retained Earnings to Total Assets'])
      fig.legend(labels=['Bankrupt', 'Not Bankrupt'])
      plt.title('Relationship between bankrupt events and Retained Earnings to Total Assets')
```

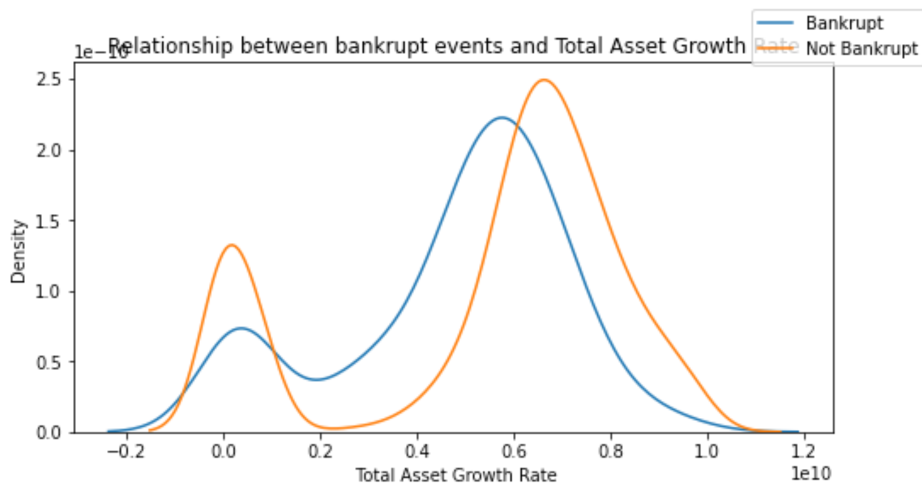
```
[25]: Text(0.5, 1.0, 'Relationship between bankrupt events and Retained Earnings to Total Assets')
```



Does Total Asset Growth Rate have any relationship with being bankrupt?

```
[26]: fig = plt.figure(figsize=(8,4))
      sns.kdeplot(df[df['Bankrupt?']==1][' Total Asset Growth Rate'])
      sns.kdeplot(df[df['Bankrupt?']==0][' Total Asset Growth Rate'])
      fig.legend(labels=['Bankrupt', 'Not Bankrupt'])
      plt.title('Relationship between bankrupt events and Total Asset Growth Rate')
```

```
[26]: Text(0.5, 1.0, 'Relationship between bankrupt events and Total Asset Growth Rate')
```



Split Data into Train and Test Data set

Now that we have explored the data, we want to come up with a model which should predict the target variable based on the data provided. in our case the target variable is “Bankrupt?”

Using the sklearn library, we now divide the data set into training and test data set

```
[58]: X = df.drop(columns = "Bankrupt?")
      Y = df["Bankrupt?"]
```

```
[59]: X_train,X_test,Y_train,Y_test = train_test_split(X,Y,random_state=1)
      print("X_train shape:", X_train.shape)
      print("y_train shape:", Y_train.shape)
      print("X_test shape:", X_test.shape)
      print("y_test shape:", Y_test.shape)
```

```
X_train shape: (5114, 15)
y_train shape: (5114,)
X_test shape: (1705, 15)
y_test shape: (1705,)
```

Random Oversampling and Undersampling for Imbalanced Classification

As we already seen that the data is imbalanced and skewness exists in the dataset, we need to address this issue. The skewness in training dataset can influence the modes leading some to ignore the minority class entirely. To address this issue, we need to randomly resample the training dataset. The two approaches are as follows:

- Undersampling
- Oversampling

We will take advantage of the library imbalanced-learn available.

```
: under_sampler = RandomUnderSampler(random_state=42)
  X_train_under, y_train_under = under_sampler.fit_resample(X_train,Y_train)
  print(X_train_under.shape)
```

```
(318, 15)
```

```
: over_sampler = RandomOverSampler(random_state=42)
  X_train_over, y_train_over = over_sampler.fit_resample(X_train,Y_train)
  print(X_train_over.shape)
```

```
(9910, 15)
```

Now that we have two additional training set, let's draw a baseline accuracy which will help us with our model evaluation

```
[63]: acc_baseline = Y_train.value_counts(normalize=True).max()
      print("Baseline Accuracy:", round(acc_baseline, 4))
```

```
Baseline Accuracy: 0.9689
```

Decision Tree Classifier

Using our training datasets, let's make Decision tree classifier:

```
[74]:
model_reg = tree.DecisionTreeClassifier(random_state=42)

model_reg.fit(X_train, Y_train)

model_under = tree.DecisionTreeClassifier(random_state=42)

model_under.fit(X_train_under, y_train_under)

model_over =  tree.DecisionTreeClassifier(random_state=42)

model_over.fit(X_train_over, y_train_over)
```

```
[74]:
▾        DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In the above code, we essentially created three models. First one is the regular Decision tree classifier model which took our original training datasets. The second and third took the undersampling and oversampling training dataset respectively.

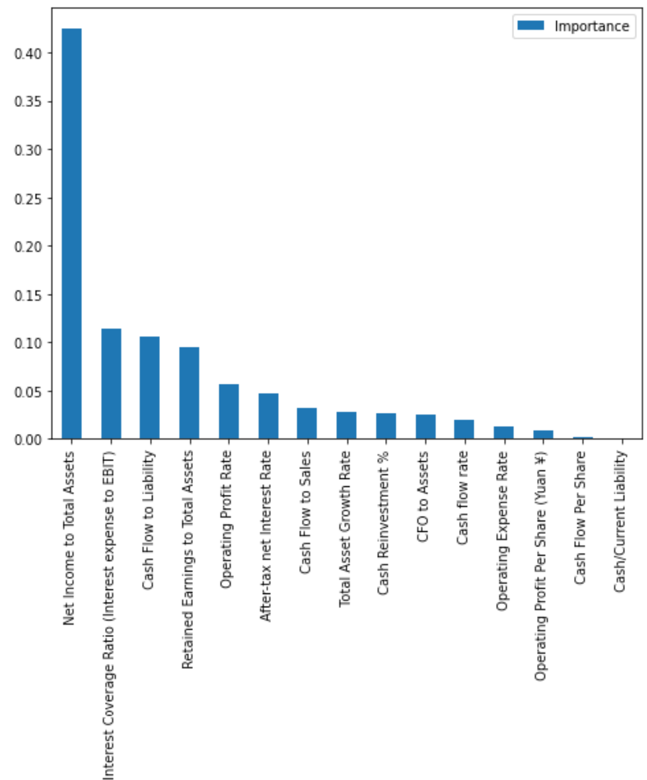
Now that we have the models, let's evaluate with our test dataset:

```
[66]: for m in [model_reg, model_under, model_over]:
          acc_train = m.score(X_train,Y_train)
          acc_test = m.score(X_test,Y_test)

          print("Training Accuracy:", round(acc_train, 4))
          print("Test Accuracy:", round(acc_test, 4))

      Training Accuracy: 1.0
      Test Accuracy: 0.9267
      Training Accuracy: 0.7953
      Test Accuracy: 0.8387
      Training Accuracy: 1.0
      Test Accuracy: 0.9584
```

Looks like the undersampling model performance is not near the baseline. Oversampling model is close to the baseline buy none of the models was able to beat the base line in terms of their accuracy.

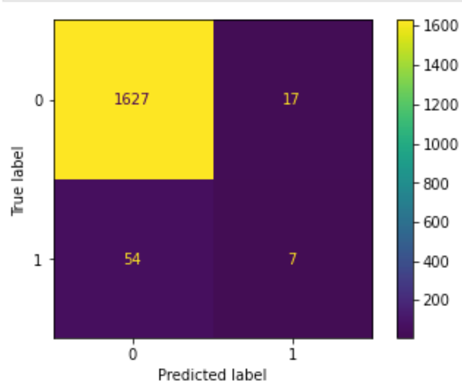
Let's take a look which variable made most impact in terms of its importance.

```
[75]: feat_importances = pd.DataFrame(model_over.feature_importances_, index=X_train.columns, columns=["Importance"])
      feat_importances.sort_values(by='Importance', ascending=False, inplace=True)
      feat_importances.plot(kind='bar', figsize=(8,6))

[75]: <AxesSubplot:>
```



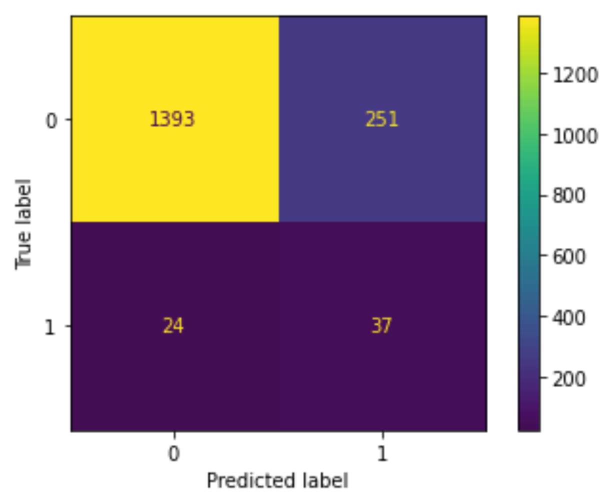
Confusion matrix of the oversampling decision tree model:

```
]: ConfusionMatrixDisplay.from_estimator(model_over, X_test,Y_test);
```



In the confusion matrix, the count of wrong label is 71.

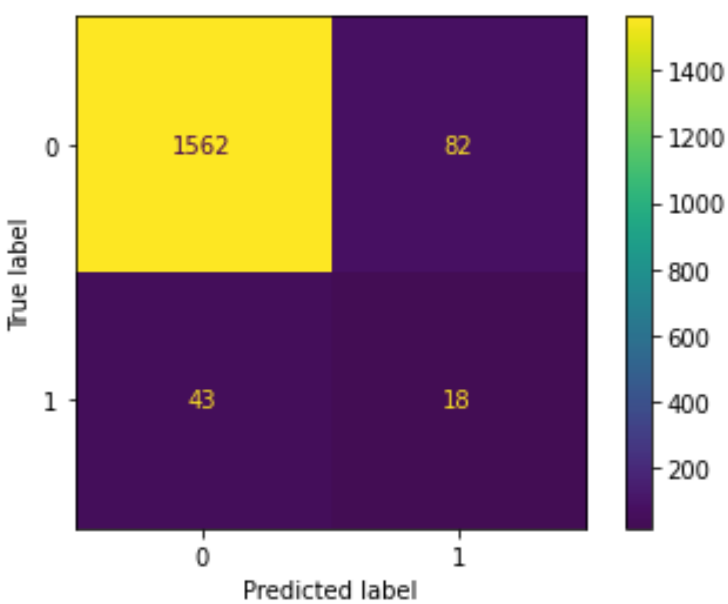
Confusion matrix of the undersampling decision tree model:



In the confusion matrix, the count of wrong label is 275.

Confusion matrix of the regular decision tree model:

```
ConfusionMatrixDisplay.from_estimator(model_reg, X_test,Y_test);
```



In the confusion matrix, the count of wrong label is 125.

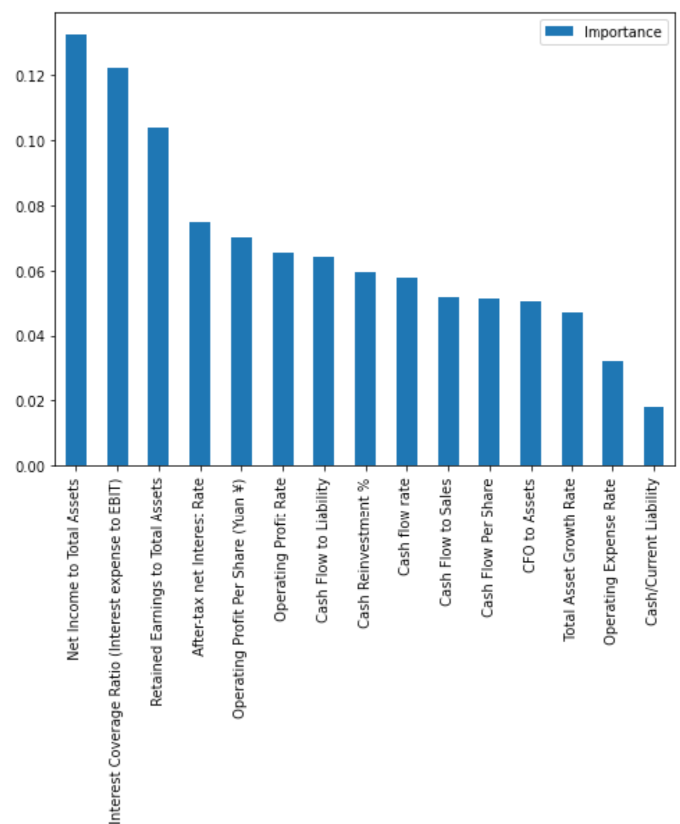
Random Forest

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression. Let's apply the Random Forest classifier to predict our target variable.

```
[69]: model = ensemble.RandomForestClassifier(random_state=1)
      model.fit(X_train,Y_train)
      pred = model.predict(X_test)
      cm = confusion_matrix(Y_test,pred)
      print("Train set Accuracy: ", metrics.accuracy_score(Y_train, model.predict(X_train)))
      print("Test set Accuracy: ", metrics.accuracy_score(Y_test, pred))

      Train set Accuracy:  1.0
      Test set Accuracy:  0.9642228739002933
```

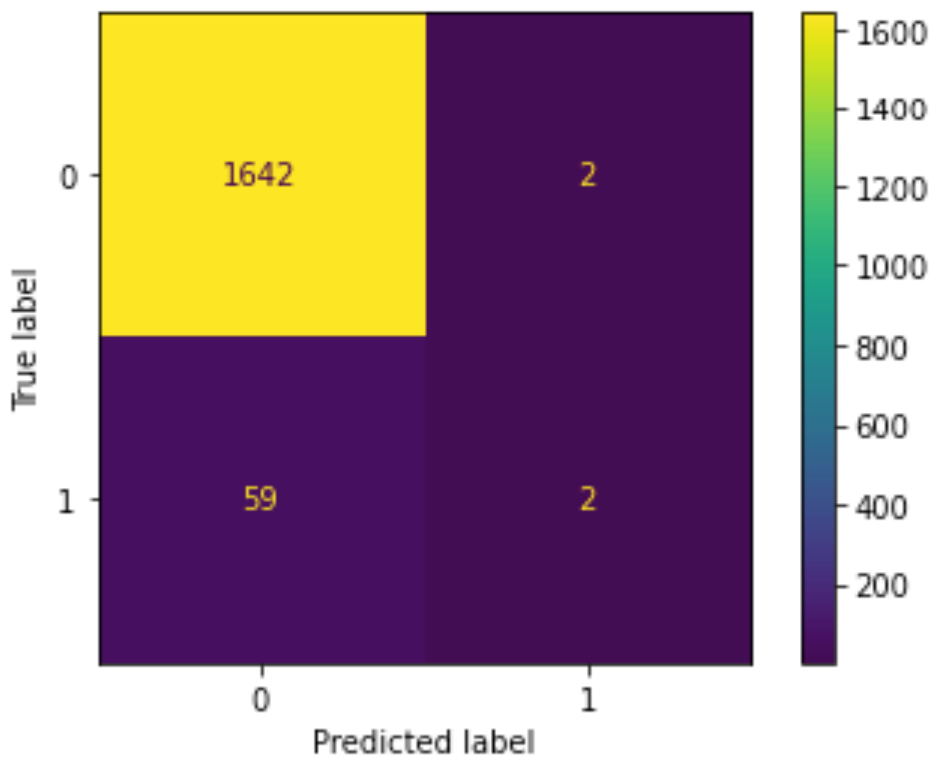
Let's take a look which variable made most impact in terms of its importance in the Random Forest model.

```
[70]: feat_importances = pd.DataFrame(model.feature_importances_, index=X_train.columns, columns=["Importance"])
      feat_importances.sort_values(by='Importance', ascending=False, inplace=True)
      feat_importances.plot(kind='bar', figsize=(8,6))
```



Confusion Matrix for the Random Forest Model:

```
[71]: cm_display = ConfusionMatrixDisplay(cm).plot()
```



In the confusion matrix from Random Forest Model, the count of wrong label is 61.

KNN

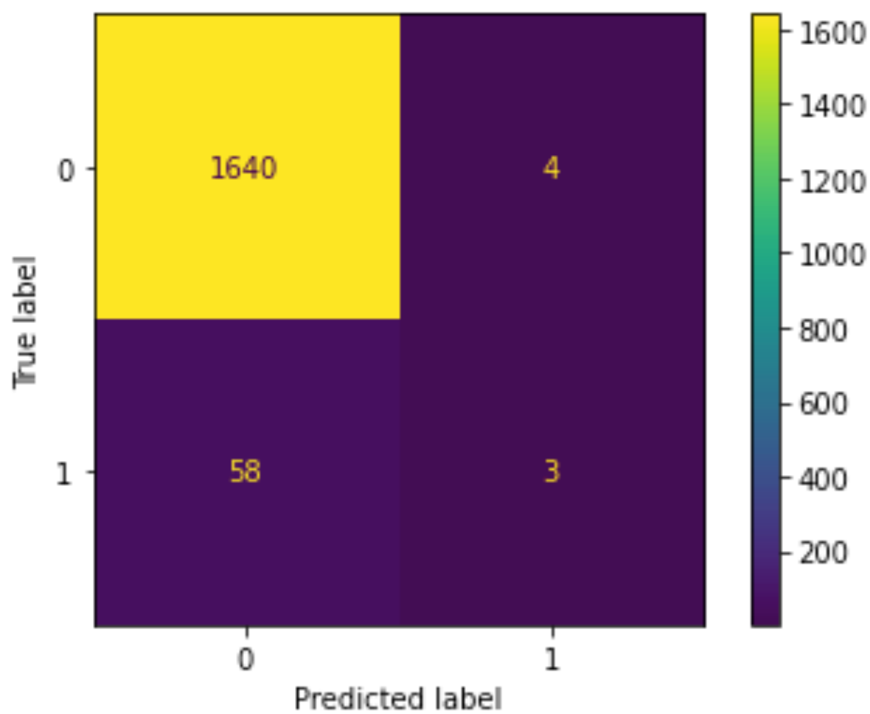
KNN is one of the simplest forms of machine learning algorithms mostly used for classification. It classifies the data point on how its neighbor is classified. KNN classifies the new data points based on the similarity measure of the earlier stored data points. Let's apply the KNN algorithms to predict our target variable.

```
model = KNeighborsClassifier(n_neighbors=2)
model.fit(X_train,Y_train)
pred = model.predict(X_test)
cm = confusion_matrix(Y_test,pred)
print("Train set Accuracy: ", metrics.accuracy_score(Y_train, model.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(Y_test, pred))
```

```
Train set Accuracy:  0.9755572937035588
Test set Accuracy:  0.9636363636363636
```

Confusion Matrix for the KNN:

```
cm_display = ConfusionMatrixDisplay(cm).plot()
```



In the confusion matrix from KNN Model, the count of wrong label is 62.

SVM

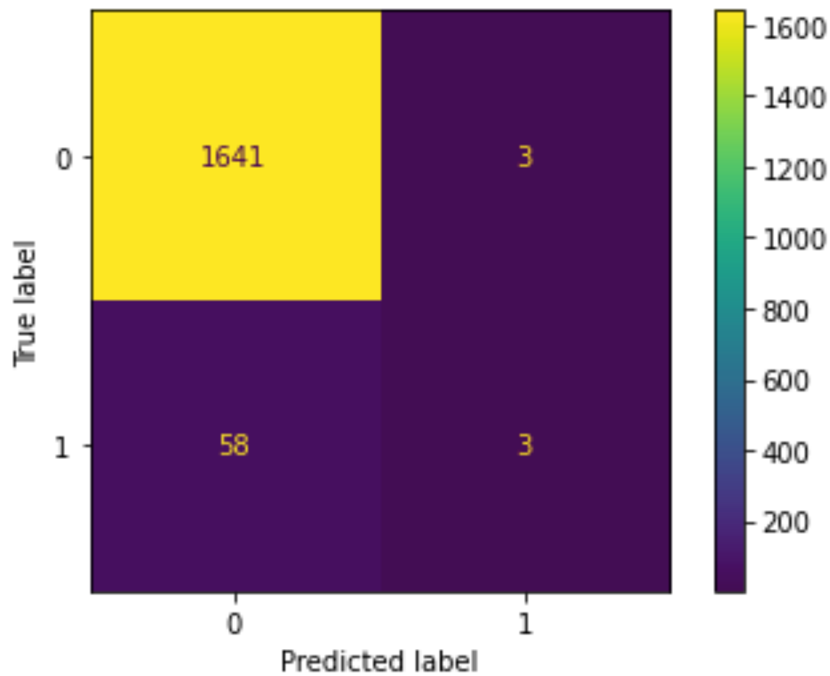
A support Vector machine is a supervised machine learning model that is used in both classification and regression problems. But SVM mostly gets used in classification problems. The SVM algorithm takes each data point and put it in an n-dimensional space. n is the number of features available in the data set. The value of each feature is the coordinate of the data point.

Using this algorithm SVM solves the classification problem by finding the hyper-plane that differentiates the classes. Let's apply the SVM algorithms to predict our target variable.

```
[110]: model =svm.LinearSVC(dual=False)
model.fit(X_train,Y_train)
pred = model.predict(X_test)
cm = confusion_matrix(Y_test,pred)
print("Train set Accuracy: ", metrics.accuracy_score(Y_train, model.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(Y_test, pred))
cm_display = ConfusionMatrixDisplay(cm).plot()
```

```
Train set Accuracy:  0.9712553773953853
Test set Accuracy:  0.9642228739002933
```

Confusion Matrix for the SVM:



In the confusion matrix from svm.LinerSVC Model, the count of wrong label is 61. The accuracy is 96.42% also close to the baseline accuracy.

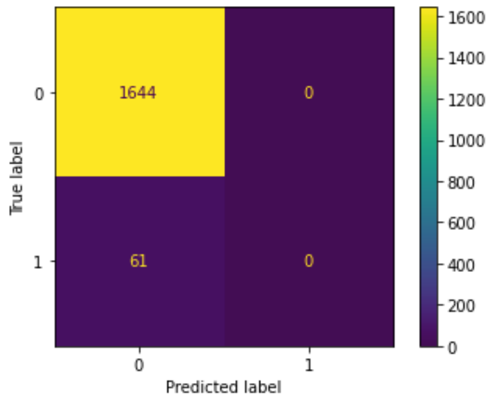
We also created a SVM model with Linear kernel and code looks like this:

```
model =svm.SVC(kernel='linear', C=1, gamma='auto')
model.fit(X_train,Y_train)
pred = model.predict(X_test)
cm = confusion_matrix(Y_test,pred)
print("Train set Accuracy: ", metrics.accuracy_score(Y_train, model.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(Y_test, pred))
```

```
Train set Accuracy:  0.9689088775909269
Test set Accuracy:  0.9642228739002933
```

The confusion matrix for the svm.svc is below:

```
cm_display = ConfusionMatrixDisplay(cm).plot()
```



In the confusion matrix from svm.svc Model, the count of wrong label is 61. The accuracy is 96.42% also close to the baseline accuracy.

Results and Discussion

In this study, seven alterative models were build using the relevant training data set. To evaluate these models, we used the remaining data set in the test data set for each model. the accuracy rates for each model are given below:

Classifier Model	Description	Accuracy Rate	Wrong Label
Decision Tree Classifier	Regular Decision Tree Model	0.9267	125
	Undersampler	0.8387	275
	Oversampler	0.9584	71
Random Forest	RandomForestClassifier, random_state=1	0.9642228739002933	61
KNN	KNeighborsClassifier, n_neighbors=2	0.9636363636363636	62
Support Vector Machine	LinearSVC	0.9642228739002933	61
	SVC	0.9642228739002933	61

Accuracy rates were used to describe the usefulness of the models. Accuracy is probably the most used metric to measure the performance of targeting models in classification applications.

The confusion Matrix for decision tree model doesn't look very good because of the higher count of wrong labels. Rest of the models produces similar kind of confusion matrix with the count of wrong label is around 61.

Performance Evaluation Metrics:

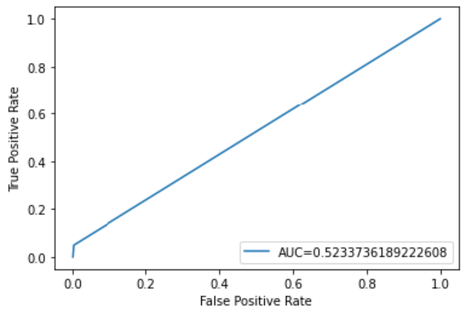
Let's look to the ROC curve (AUC) of the models. Since the Decision Tree models are poor models, we are not going to consider the performance of the Decision Tree models. To evaluate the performance metrics, I used the area under the ROC Curve (AUC) which was computed using continuous outputs of the classifiers (distances from separating hyperplane for SVMs and outcome probabilities for RFs).

Let's look at the AUC for each of the models to evaluate their performances:

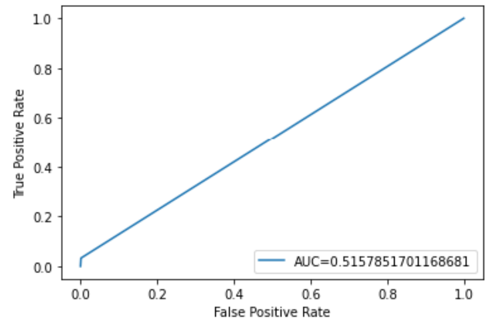
Classifier Model	Description	AUC
Random Forest	RandomForestClassifier, random_state=1	0.5157
KNN	KNeighborsClassifier, n_neighbors=2	0.5233
Support Vector Machine	LinearSVC	0.5236
	SVC	0.5

Visualize the AUCs

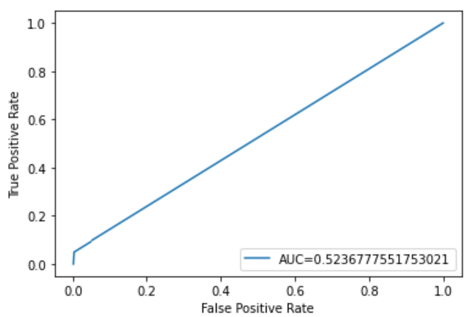
KNN



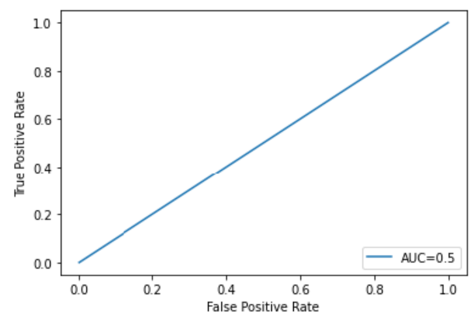
Random Forest



LinearSVC



SVC



The AUC of all the models is around 0.5. The value for AUC ranges from 0 to 1. A model that has an AUC of 1 can perfectly classify observations into classes. The higher the AUC score, the better the model can classify the observations into classes. However, there is no magic number that determines the good or bad AUC score.

Hosmer and Lemeshow described the rule of thumb in Applied Logistic Regression book and created a chart to understand AUC Score. The chart is as follows:

- 0.5 = No discrimination
- 0.5-0.7 = Poor discrimination
- 0.7-0.8 = Acceptable discrimination
- 0.8-0.9= Excellent discrimination
- >0.9 = Outstanding discrimination

The chart indicates that the performance of our models is not excellent but it’s not poor either.

In general, both Random Forest and SVM works well for binary classification. In the classification problem, Random Forest gives us the probability of belonging to class, whereas SVM gives us the distance to the boundary. Where SVM applies, it generally performs better than Random Forest but, in our case, we don't see any significant difference between SVM and RF models. The choice of algorithm depends upon the desired outcome. Although both models are good at their place, but it very much depends upon the quality of data when it comes to algorithm's performance.

Conclusion

Among the models, it's seeming to me that Random Forest model was closest to the baseline in terms of the test data set accuracy which is 0.9642. the count of incorrect predicted label is also lowest. Hence, we can conclude that our best model is the Random Forest Model to predict the target variable for the Taiwanese Bankruptcy Prediction Data Set.

Reference

Applied Logistic Regression, 3rd Edition (3rd ed.). (2013). Wiley.

Mandrekar, J. N. (2010). Receiver operating characteristic curve in diagnostic test assessment. Journal of Thoracic Oncology, 5(9), 1315–1316. https://doi.org/10.1097/jto.0b013e3181ec173d

Heumann, B. W. (2011). An Object-Based Classification of Mangroves Using a Hybrid Decision Tree—Support Vector Machine Approach. Remote Sensing, 3(11), 2440–2460. https://doi.org/10.3390/rs3112440

Ahmad, A., Safi, O., Malebary, S., Alesawi, S., & Alkayal, E. (2021). Decision Tree Ensembles to Predict Coronavirus Disease 2019 Infection: A Comparative Study. *Complexity*, *2021*, 1–8. https://doi.org/10.1155/2021/5550344

Guhathakurata, S., Kundu, S., Chakraborty, A., & Banerjee, J. S. (2021). A novel approach to predict COVID-19 using support vector machine. *Data Science for COVID-19*, 351–364. https://doi.org/10.1016/b978-0-12-824536-1.00014-9

Alexander Statnikov, & Constantin F. Aliferis. (2007). Are random forests better than support vector machines for microarray-based cancer classification? *American Medical Informatics Association Annual Symposium*. https://pubmed.ncbi.nlm.nih.gov/18693924/