Problem Statement

Context

In the realm of modern finance, businesses encounter the perpetual challenge of managing debt obligations effectively to maintain a favourable credit standing and foster sustainable growth. Investors keenly scrutinize companies capable of navigating financial complexities while ensuring stability and profitability. A pivotal instrument in this evaluation process is the balance sheet, which provides a comprehensive overview of a company's assets, liabilities, and shareholder equity, offering insights into its financial health and operational efficiency. In this context, leveraging available financial data, particularly from preceding fiscal periods, becomes imperative for informed decision-making and strategic planning.

Objective

A group of venture capitalists want to develop a Financial Health Assessment Tool. With the help of the tool, it endeavors to empower businesses and investors with a robust mechanism for evaluating the financial well-being and creditworthiness of companies. By harnessing machine learning techniques, they aim to analyze historical financial statements and extract pertinent insights to facilitate informed decision-making via the tool. Specifically, they foresee facilitating the following with the help of the tool:

Debt Management Analysis: Identify patterns and trends in debt management practices to assess the ability of businesses to fulfill financial obligations promptly and efficiently, and identify potential cases of default.

Credit Risk Evaluation: Evaluate credit risk exposure by analyzing liquidity ratios, debt-to-equity ratios, and other key financial indicators to ascertain the likelihood of default and inform investment decisions.

They have hired you as a data scientist and provided you with the financial metrics of different companies. The task is to analyze the data provided and develop a predictive model leveraging machine learning techniques to identify whether a given company will be tagged as a defaulter in terms of net worth next year. The predictive model will help the organization anticipate potential challenges with the financial performance of the companies and enable proactive risk mitigation strategies.

Data Dictionary

The data consists of financial metrics from the balance sheets of different companies. The detailed data dictionary is given below.

- Networth Next Year: Net worth of the customer in the next year

- Total assets: Total assets of customer
- Net worth: Net worth of the customer of the present year
- Total income: Total income of the customer
- Change in stock: Difference between the current value of the stock and the value of stock in the last trading day
- Total expenses: Total expenses done by the customer
- Profit after tax: Profit after tax deduction
- PBDITA: Profit before depreciation, income tax, and amortization
- PBT: Profit before tax deduction
- Cash profit: Total Cash profit
- PBDITA as % of total income: PBDITA / Total income
- PBT as % of total income: PBT / Total income
- PAT as % of total income: PAT / Total income
- Cash profit as % of total income: Cash Profit / Total income
- PAT as % of net worth: PAT / Net worth
- Sales: Sales done by the customer
- Income from financial services: Income from financial services
- Other income: Income from other sources
- Total capital: Total capital of the customer
- Reserves and funds: Total reserves and funds of the customer
- Borrowings: Total amount borrowed by the customer
- Current liabilities & provisions: current liabilities of the customer
- Deferred tax liability: Future income tax customer will pay because of the current transaction
- Shareholders funds: Amount of equity in a company which belongs to shareholders
- Cumulative retained profits: Total cumulative profit retained by customer
- Capital employed: Current asset minus current liabilities
- TOL/TNW: Total liabilities of the customer divided by Total net worth
- Total term liabilities / tangible net worth: Short + long term liabilities divided by tangible net worth
- Contingent liabilities / Net worth (%): Contingent liabilities / Net worth
- Contingent liabilities: Liabilities because of uncertain events
- Net fixed assets: The purchase price of all fixed assets
- Investments: Total invested amount
- Current assets: Assets that are expected to be converted to cash within a year
- Net working capital: Difference between the current liabilities and current assets
- Quick ratio (times): Total cash divided by current liabilities
- Current ratio (times): Current assets divided by current liabilities
- Debt to equity ratio (times): Total liabilities divided by its shareholder equity
- Cash to current liabilities (times): Total liquid cash divided by current liabilities
- Cash to average cost of sales per day: Total cash divided by the average cost of the sales
- Creditors turnover: Net credit purchase divided by average trade creditors

- Debtors turnover: Net credit sales divided by average accounts receivable
- Finished goods turnover: Annual sales divided by average inventory
- WIP turnover: The cost of goods sold for a period divided by the average inventory for that period
- Raw material turnover: Cost of goods sold is divided by the average inventory for the same period
- Shares outstanding: Number of issued shares minus the number of shares held in the company
- Equity face value: cost of the equity at the time of issuing
- EPS: Net income divided by the total number of outstanding share
- Adjusted EPS: Adjusted net earnings divided by the weighted average number of common shares outstanding on a diluted basis - during the plan year
- Total liabilities: Sum of all types of liabilities
- PE on BSE: Company's current stock price divided by its earnings per share

In [1]:

```
# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# to scale the data using StandardScaler
from sklearn.preprocessing import StandardScaler

# to impute using KNNImputer
from sklearn.impute import KNNImputer

# to perform Logistic Regression and Random Forest
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn import metrics

# To check model performance
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_curve,
    roc_auc_score
)

# to suppress warnings
import warnings
```

```
warnings.filterwarnings("ignore")
```

Understanding the structure of data

In [76]:
```
Company = pd.read_csv('Comp_Fin_Data.csv') # Importing the data
```

In [77]:
```
Company.head() # Returns first 5 rows
```

Out[77]:

	Num	Networth Next Year	Total assets	Net worth	Total income	Change in stock	Total expenses	Profit after tax	PBDITA	PBT
0	1	395.30	827.60	336.50	534.10	13.50	508.70	38.90	124.40	64.60
1	2	36.20	67.70	24.30	137.90	-3.70	131.00	3.20	5.50	1.00
2	3	84.00	238.40	78.90	331.20	-18.10	309.20	3.90	25.80	10.50
3	4	2041.40	6883.50	1443.30	8448.50	212.20	8482.40	178.30	418.40	185.10
4	5	41.80	90.90	47.00	388.60	3.40	392.70	-0.70	7.20	-0.60

5 rows × 51 columns

Fixing column names (containing spaces or '(' or ')' or '%' or '/') for ease of use

In [78]:
```
Company.columns = Company.columns.str.replace(' ', '_').str.replace('(', '').str.re
```

Checking top 5 rows again

In [79]:
```
Company.head() # Returns first 5 rows
```

Out[79]:

	Num	Networth_Next_Year	Total_assets	Net_worth	Total_income	Change_in_stock	Tot
0	1	395.30	827.60	336.50	534.10	13.50	
1	2	36.20	67.70	24.30	137.90	-3.70	
2	3	84.00	238.40	78.90	331.20	-18.10	
3	4	2041.40	6883.50	1443.30	8448.50	212.20	
4	5	41.80	90.90	47.00	388.60	3.40	

5 rows × 51 columns

Number of rows and columns in the dataset

In [80]:
```
# checking shape of the data
```

```
rows = str(Company.shape[0])
columns = str(Company.shape[1])

print(f"There are \033[1m" + rows + "\033[0m rows and \033[1m" + columns + "\033[0m
```

There are **4256** rows and **51** columns in the dataset.

Datatypes of the different columns in the dataset

In [81]:
```
Company.info() # Concise summary of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4256 entries, 0 to 4255
Data columns (total 51 columns):
 #   Column                                      Non-Null Count  Dtype  
---  ------                                      --------------  -----  
 0   Num                                         4256 non-null   int64  
 1   Networth_Next_Year                          4256 non-null   float64
 2   Total_assets                                4256 non-null   float64
 3   Net_worth                                   4256 non-null   float64
 4   Total_income                                4025 non-null   float64
 5   Change_in_stock                             3706 non-null   float64
 6   Total_expenses                              4091 non-null   float64
 7   Profit_after_tax                            4102 non-null   float64
 8   PBDITA                                      4102 non-null   float64
 9   PBT                                         4102 non-null   float64
 10  Cash_profit                                 4102 non-null   float64
 11  PBDITA_as_perc_of_total_income              4177 non-null   float64
 12  PBT_as_perc_of_total_income                 4177 non-null   float64
 13  PAT_as_perc_of_total_income                 4177 non-null   float64
 14  Cash_profit_as_perc_of_total_income         4177 non-null   float64
 15  PAT_as_perc_of_net_worth                    4256 non-null   float64
 16  Sales                                       3951 non-null   float64
 17  Income_from_fincial_services                3145 non-null   float64
 18  Other_income                                2700 non-null   float64
 19  Total_capital                               4251 non-null   float64
 20  Reserves_and_funds                          4158 non-null   float64
 21  Borrowings                                  3825 non-null   float64
 22  Current_liabilities_&_provisions            4146 non-null   float64
 23  Deferred_tax_liability                      2887 non-null   float64
 24  Shareholders_funds                          4256 non-null   float64
 25  Cumulative_retained_profits                 4211 non-null   float64
 26  Capital_employed                            4256 non-null   float64
 27  TOL_to_TNW                                  4256 non-null   float64
 28  Total_term_liabilities__to__tangible_net_worth  4256 non-null   float64
 29  Contingent_liabilities__to__Net_worth_perc  4256 non-null   float64
 30  Contingent_liabilities                      2854 non-null   float64
 31  Net_fixed_assets                            4124 non-null   float64
 32  Investments                                 2541 non-null   float64
 33  Current_assets                              4176 non-null   float64
 34  Net_working_capital                         4219 non-null   float64
 35  Quick_ratio_times                           4151 non-null   float64
 36  Current_ratio_times                         4151 non-null   float64
 37  Debt_to_equity_ratio_times                  4256 non-null   float64
 38  Cash_to_current_liabilities_times           4151 non-null   float64
 39  Cash_to_average_cost_of_sales_per_day       4156 non-null   float64
 40  Creditors_turnover                          3865 non-null   float64
 41  Debtors_turnover                            3871 non-null   float64
 42  Finished_goods_turnover                     3382 non-null   float64
 43  WIP_turnover                                3492 non-null   float64
 44  Raw_material_turnover                       3828 non-null   float64
 45  Shares_outstanding                          3446 non-null   float64
 46  Equity_face_value                           3446 non-null   float64
 47  EPS                                         4256 non-null   float64
 48  Adjusted_EPS                                4256 non-null   float64
 49  Total_liabilities                           4256 non-null   float64
 50  PE_on_BSE                                   1629 non-null   float64
```

```
dtypes: float64(50), int64(1)
memory usage: 1.7 MB
```

There are 51 columns in the dataset. Out of which 50 have float data type and 1 has integer data type.

Check duplicate records

In [82]:
```
Company.duplicated().sum() # Check duplicate records
```

Out[82]:
```
0
```

There are no duplicate records in the dataset.

Statistical summary of the data

In [83]:
```
pd.options.display.float_format = '{:.2f}'.format # Rounding off float value to 2 d

Company.describe(include='all').T
```

Out[83]:

	count	mean	std	
Num	4256.00	2128.50	1228.75	
Networth_Next_Year	4256.00	1344.74	15936.74	-742
Total_assets	4256.00	3573.62	30074.44	
Net_worth	4256.00	1351.95	12961.31	
Total_income	4025.00	4688.19	53918.95	
Change_in_stock	3706.00	43.70	436.92	-302
Total_expenses	4091.00	4356.30	51398.09	-
Profit_after_tax	4102.00	295.05	3079.90	-390
PBDITA	4102.00	605.94	5646.23	-44
PBT	4102.00	410.26	4217.42	-389
Cash_profit	4102.00	408.27	4143.93	-224
PBDITA_as_perc_of_total_income	4177.00	3.18	172.26	-640
PBT_as_perc_of_total_income	4177.00	-18.20	419.91	-2134
PAT_as_perc_of_total_income	4177.00	-20.03	423.58	-2134
Cash_profit_as_perc_of_total_income	4177.00	-9.02	299.96	-1502
PAT_as_perc_of_net_worth	4256.00	10.17	61.53	-74
Sales	3951.00	4645.68	53080.90	
Income_from_fincial_services	3145.00	81.36	1042.76	
Other_income	2700.00	55.95	1178.42	
Total_capital	4251.00	224.56	1684.95	
Reserves_and_funds	4158.00	1210.56	12816.23	-652
Borrowings	3825.00	1176.25	8581.25	
Current_liabilities_&_provisions	4146.00	960.63	9140.54	
Deferred_tax_liability	2887.00	234.50	2106.25	
Shareholders_funds	4256.00	1376.49	13010.69	
Cumulative_retained_profits	4211.00	937.18	9853.10	-653
Capital_employed	4256.00	2433.62	20496.40	
TOL_to_TNW	4256.00	4.03	20.88	-35
Total_term_liabilities__to__tangible_net_worth	4256.00	1.85	15.88	-32
Contingent_liabilities__to__Net_worth_perc	4256.00	55.71	369.17	

	count	mean	std	
Contingent_liabilities	2854.00	948.55	12056.74	
Net_fixed_assets	4124.00	1209.49	12502.40	
Investments	2541.00	721.87	6793.86	
Current_assets	4176.00	1350.36	10155.57	
Net_working_capital	4219.00	162.87	3182.03	-6383
Quick_ratio_times	4151.00	1.50	9.33	
Current_ratio_times	4151.00	2.26	12.48	
Debt_to_equity_ratio_times	4256.00	2.87	15.60	
Cash_to_current_liabilities_times	4151.00	0.53	4.80	
Cash_to_average_cost_of_sales_per_day	4156.00	145.16	2521.99	
Creditors_turnover	3865.00	16.81	75.67	
Debtors_turnover	3871.00	17.93	90.16	
Finished_goods_turnover	3382.00	84.37	562.64	-
WIP_turnover	3492.00	28.68	169.65	-
Raw_material_turnover	3828.00	17.73	343.13	-
Shares_outstanding	3446.00	23764909.56	170979041.33	-214748364
Equity_face_value	3446.00	-1094.83	34101.36	-9999
EPS	4256.00	-196.22	13061.95	-84318
Adjusted_EPS	4256.00	-197.53	13061.93	-84318
Total_liabilities	4256.00	3573.62	30074.44	
PE_on_BSE	1629.00	55.46	1304.45	-11

Observations and Insights:

- Num variable is similar to serial number which is starting with 1 and ending with 4256 (total number of records in the dataset).
- All variables (except Num) have minimum value either as -ve or 0 and maximum value as +ve.
- Mean value is -ve for some variables.

Removing first column (Num) in the dataset

In [84]:
```
# Removing first column in the dataset as it is a serial number

Company.drop('Num', axis=1, inplace=True)
```

Removed first column from the dataset as it is a serial number.

In [85]:
```
Company.head() # Returns first 5 rows
```

Out[85]:

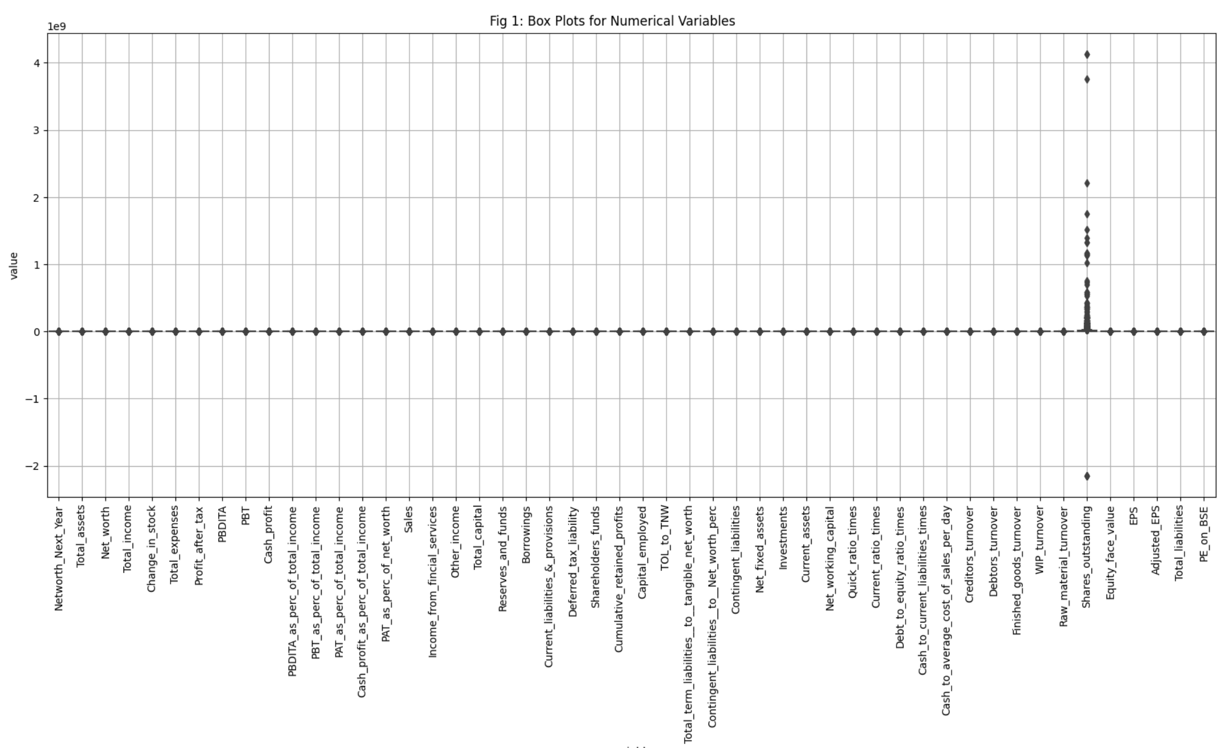
	Networth_Next_Year	Total_assets	Net_worth	Total_income	Change_in_stock	Total_expe
0	395.30	827.60	336.50	534.10	13.50	50
1	36.20	67.70	24.30	137.90	-3.70	13
2	84.00	238.40	78.90	331.20	-18.10	30
3	2041.40	6883.50	1443.30	8448.50	212.20	848
4	41.80	90.90	47.00	388.60	3.40	39

5 rows × 50 columns

Exploratory Data Analysis (EDA)

Univariate Analysis

In [86]:
```
_, ax = plt.subplots(figsize=(20,8))
sns.boxplot(x = 'variable',y = 'value', data = pd.melt(Company),ax=ax)
plt.xticks(rotation=90)
plt.grid()
plt.suptitle('Fig 1: Box Plots for Numerical Variables', y=0.91)
plt.show()
```



Observations and Insights:

- Shares_outstanding distribution has large number of outliers.

Multivariate Analysis

Correlation among variables

In [87]:
```
# Correlation between all numerical variables in the dataset

Company_corr = Company.select_dtypes(include=[np.number])
Company_corr.corr()
```

Out[87]:

	Networth_Next_Year	Total_assets	Net_worth
Networth_Next_Year	1.00	0.88	0.93
Total_assets	0.88	1.00	0.96
Net_worth	0.93	0.96	1.00
Total_income	0.71	0.87	0.78
Change_in_stock	0.35	0.47	0.39
Total_expenses	0.69	0.85	0.76
Profit_after_tax	0.87	0.91	0.95
PBDITA	0.87	0.94	0.96
PBT	0.83	0.90	0.93
Cash_profit	0.91	0.94	0.98
PBDITA_as_perc_of_total_income	0.01	0.01	0.01
PBT_as_perc_of_total_income	0.01	0.01	0.01
PAT_as_perc_of_total_income	0.01	0.01	0.01
Cash_profit_as_perc_of_total_income	0.01	0.01	0.01
PAT_as_perc_of_net_worth	0.02	0.02	0.02
Sales	0.72	0.87	0.79
Income_from_fincial_services	0.52	0.73	0.60
Other_income	0.13	0.32	0.24
Total_capital	0.35	0.43	0.44
Reserves_and_funds	0.94	0.95	0.99
Borrowings	0.78	0.93	0.87
Current_liabilities_&_provisions	0.67	0.88	0.75
Deferred_tax_liability	0.89	0.98	0.96
Shareholders_funds	0.93	0.96	1.00
Cumulative_retained_profits	0.90	0.96	0.98
Capital_employed	0.90	0.98	0.98
TOL_to_TNW	-0.01	-0.01	-0.01
Total_term_liabilities__to__tangible_net_worth	-0.01	-0.01	-0.01
Contingent_liabilities__to__Net_worth_perc	-0.00	0.00	-0.00
Contingent_liabilities	0.92	0.89	0.93

	Networth_Next_Year	Total_assets	Net_worth
Net_fixed_assets	0.93	0.94	0.97
Investments	0.79	0.89	0.87
Current_assets	0.65	0.89	0.77
Net_working_capital	-0.07	0.03	0.06
Quick_ratio_times	-0.01	-0.01	-0.01
Current_ratio_times	-0.01	-0.01	-0.01
Debt_to_equity_ratio_times	-0.01	-0.01	-0.01
Cash_to_current_liabilities_times	-0.00	-0.00	-0.00
Cash_to_average_cost_of_sales_per_day	-0.06	0.04	0.02
Creditors_turnover	-0.01	-0.01	-0.01
Debtors_turnover	0.01	0.01	0.01
Finished_goods_turnover	-0.01	-0.01	-0.01
WIP_turnover	-0.01	-0.01	-0.01
Raw_material_turnover	-0.00	-0.00	-0.00
Shares_outstanding	0.39	0.47	0.49
Equity_face_value	0.00	0.01	0.01
EPS	0.00	0.00	0.00
Adjusted_EPS	0.00	0.00	0.00
Total_liabilities	0.88	1.00	0.96
PE_on_BSE	-0.00	-0.01	-0.00

50 rows × 50 columns

In [200…

```
# Heatmap to plot correlation between all numerical variables in the dataset

plt.figure(figsize=(35, 20))
sns.heatmap(Company_corr.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spec
plt.title('Fig 2: Correlation Between Numerical Variables')
plt.show()
```

Observations and Insights:

- There is moderate to strong correlation between some numerical variables (assets, profit and liabilities).

Data Pre-processing

Target variable creation using 'Networth_Next_Year'

In [90]:
```
Company['default'] = np.where((Company['Networth_Next_Year'] > 0), 0, 1) # Target v
```

Checking top 5 rows

In [91]:
```
Company[['default','Networth_Next_Year']].head() # Returns first 5 rows
```

Out[91]:

	default	Networth_Next_Year
0	0	395.30
1	0	36.20
2	0	84.00
3	0	2041.40
4	0	41.80

Value count of default variable

In [92]:
```
Company['default'].value_counts() # Frequency of each distinct value in the default
```

Out[92]:
```
default
0    3352
1     904
Name: count, dtype: int64
```

Checking proportion of default variable

In [93]:
```
Company['default'].value_counts(normalize = True)  # Proportion of default variable
```

Out[93]:
```
default
0   0.79
1   0.21
Name: proportion, dtype: float64
```

Dropping 'Networth_Next_Year' variable

In [94]:
```
Company.drop(['Networth_Next_Year'], inplace = True, axis = 1) # Dropping variables
```

In [95]:
```
Company.head() # Returns first 5 rows
```

Out[95]:

	Total_assets	Net_worth	Total_income	Change_in_stock	Total_expenses	Profit_after_tax
0	827.60	336.50	534.10	13.50	508.70	38.90
1	67.70	24.30	137.90	-3.70	131.00	3.20
2	238.40	78.90	331.20	-18.10	309.20	3.90
3	6883.50	1443.30	8448.50	212.20	8482.40	178.30
4	90.90	47.00	388.60	3.40	392.70	-0.70

5 rows × 50 columns

Finding missing values in the dataset

In [96]:
```
Company.isna().sum()  # Count NaN values in all columns of dataset
```

Out[96]:
```
Total_assets                                      0
Net_worth                                         0
Total_income                                    231
Change_in_stock                                 550
Total_expenses                                  165
Profit_after_tax                                154
PBDITA                                          154
PBT                                             154
Cash_profit                                     154
PBDITA_as_perc_of_total_income                   79
PBT_as_perc_of_total_income                      79
PAT_as_perc_of_total_income                      79
Cash_profit_as_perc_of_total_income              79
PAT_as_perc_of_net_worth                          0
Sales                                           305
Income_from_fincial_services                   1111
Other_income                                   1556
Total_capital                                     5
Reserves_and_funds                               98
Borrowings                                      431
Current_liabilities_&_provisions                110
Deferred_tax_liability                         1369
Shareholders_funds                                0
Cumulative_retained_profits                      45
Capital_employed                                  0
TOL_to_TNW                                        0
Total_term_liabilities__to__tangible_net_worth    0
Contingent_liabilities__to__Net_worth_perc        0
Contingent_liabilities                         1402
Net_fixed_assets                                132
Investments                                    1715
Current_assets                                   80
Net_working_capital                              37
Quick_ratio_times                               105
Current_ratio_times                             105
Debt_to_equity_ratio_times                        0
Cash_to_current_liabilities_times               105
Cash_to_average_cost_of_sales_per_day           100
Creditors_turnover                              391
Debtors_turnover                                385
Finished_goods_turnover                         874
WIP_turnover                                    764
Raw_material_turnover                           428
Shares_outstanding                              810
Equity_face_value                               810
EPS                                               0
Adjusted_EPS                                      0
Total_liabilities                                 0
PE_on_BSE                                      2627
default                                           0
dtype: int64
```

In [97]:
```
Company.isnull().sum().sum() # Total number of missing values in the dataset
```

Out[97]:
```
17778
```

There are 17778 missing values in the dataset.

Outlier Detection and Treatment

In [98]:

```
# User Defined Function (UDF) to treat outliers
def treat_outlier(x):

    # taking 25,75 percentile of column
    q25=np.percentile(x,25)
    q75=np.percentile(x,75)

    #calculationg IQR range
    IQR=q75-q25
    #Calculating minimum threshold
    lower_bound=q25-(1.5*IQR)
    upper_bound=q75+(1.5*IQR)
    #Capping outliers
    return x.apply(lambda y: upper_bound if y > upper_bound else y).apply(lambda y:
```

In [99]:

```
no_outlier = ['default'] # default column is a target variable
outlier_list = [x for x in Company.columns if x not in no_outlier]

# Using for loop to iterate over numerical columns and calling treat_outlier UDF to
for i in Company[outlier_list]:
    Company[i]=treat_outlier(Company[i])
```

In [100…

```
Company.head() # Returns first 5 rows
```

Out[100…

	Total_assets	Net_worth	Total_income	Change_in_stock	Total_expenses	Profit_after_tax
0	827.60	336.50	534.10	13.50	508.70	38.90
1	67.70	24.30	137.90	-3.70	131.00	3.20
2	238.40	78.90	331.20	-18.10	309.20	3.90
3	2665.05	927.41	8448.50	212.20	8482.40	178.30
4	90.90	47.00	388.60	3.40	392.70	-0.70

5 rows × 50 columns

Data Split

In [101…

```
Company_X = Company.drop('default', axis = 1)
Company_Y = Company['default']
```

Splitting the data into Train and Test sets

In [102…

```
# Splitting the data for training and testing (70:30)

X_train, X_test, y_train, y_test = train_test_split(Company_X, Company_Y, test_size
```

In [103…

```
X_train.head() # Returns first 5 rows
```

Out[103…

	Total_assets	Net_worth	Total_income	Change_in_stock	Total_expenses	Profit_after_
2939	48.10	22.60	94.30	0.40	86.60	8
62	1080.70	563.40	1308.60	8.70	1237.70	79
2690	38.90	36.00	33.20	0.30	32.60	0
635	5.60	3.60	3.00	0.60	4.00	-0
4241	108.40	35.50	295.20	0.00	294.00	1

5 rows × 49 columns

In [104…

```
X_test.head() # Returns first 5 rows
```

Out[104…

	Total_assets	Net_worth	Total_income	Change_in_stock	Total_expenses	Profit_after_
1516	257.20	56.40	1205.60	-6.40	1192.90	6
2178	37.60	22.00	22.00	0.10	21.10	1
2591	2665.05	927.41	2997.70	-4.10	2587.10	406
3158	2364.20	514.70	1362.10	173.00	1479.60	55
2323	77.00	10.90	58.60	0.40	58.90	0

5 rows × 49 columns

Imputing the missing values

In [105…

```
imputer = KNNImputer(n_neighbors=5) # KNNImputer
```

In [106…

```
# Total number of Null values before imputation in Train dataset

print('Total number of Null values before imputation in Train dataset:', X_train.is
```

```
Total number of Null values before imputation in Train dataset: 12851
```

In [107…

```
# Total number of Null values before imputation in Test dataset

print('Total number of Null values before imputation in Test dataset:', X_test.isnu
```

```
Total number of Null values before imputation in Test dataset: 4927
```

In [108…

```
X_train = pd.DataFrame(imputer.fit_transform(X_train), columns = X_train.columns)
X_test = pd.DataFrame(imputer.fit_transform(X_test), columns = X_test.columns)
```

In [109…

```
# Total number of Null values after imputation in Train dataset

print('Total number of Null values after imputation in Train dataset:', X_train.isn
```

```
Total number of Null values after imputation in Train dataset: 0
```

In [110…

```
# Total number of Null values after imputation in Test dataset

print('Total number of Null values after imputation in Test dataset:', X_test.isnul
```

```
Total number of Null values after imputation in Test dataset: 0
```

Scaling

In [111…

```
# scaling the data before model building

scaler = StandardScaler()

X_train_scaled = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.column
X_test_scaled = pd.DataFrame(scaler.fit_transform(X_test), columns=X_test.columns)
```

In [112…

```
X_train_scaled.head() # Returns first 5 rows
```

Out[112…

	Total_assets	Net_worth	Total_income	Change_in_stock	Total_expenses	Profit_after_tax
0	-0.77	-0.74	-0.08	-0.10	-0.08	-0.09
1	0.34	0.92	-0.06	-0.07	-0.06	-0.07
2	-0.78	-0.70	-0.08	-0.10	-0.08	-0.09
3	-0.82	-0.80	-0.08	-0.10	-0.08	-0.09
4	-0.71	-0.71	-0.07	-0.10	-0.07	-0.09

5 rows × 49 columns

In [113…

```
X_test_scaled.head() # Returns first 5 rows
```

Out[113…

	Total_assets	Net_worth	Total_income	Change_in_stock	Total_expenses	Profit_after_tax
0	-0.58	-0.66	-0.08	-0.10	-0.08	-0.10
1	-0.82	-0.77	-0.11	-0.09	-0.11	-0.10
2	2.02	2.06	-0.03	-0.10	-0.04	0.04
3	1.70	0.77	-0.08	0.23	-0.07	-0.08
4	-0.78	-0.80	-0.11	-0.09	-0.11	-0.10

5 rows × 49 columns

Model Evaluation Criterion

Metrics of Choice

F1 Score, Precision and Recall

Rational: Dataset is not balanced (minority class in the dataset constitutes less than 30% of total data).

In [114…

```
# defining a function to compute different metrics to check performance of a classi

def model_performance_classification(model, predictors, target, threshold = 0.5):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    y_pred = model.predict(predictors)

    if len(list(set(y_pred))) != 2:
        y_prob_pred = model.predict(predictors)

        y_pred=[]
        for i in range(0,len(y_prob_pred)):
            if np.array(y_prob_pred)[i] > threshold:
                a=1
            else:
                a=0
            y_pred.append(a)
    else:
        pass

    acc = accuracy_score(target, y_pred)  # to compute Accuracy
    recall = recall_score(target, y_pred)  # to compute Recall
    precision = precision_score(target, y_pred)  # to compute Precision
    f1 = f1_score(target, y_pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1 Score": f1,
        index=[0],
    )

    return df_perf
```

In [115…

```
def model_confusion_matrix(model, predictors, target, threshold = 0.5):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
```

```
    if len(list(set(y_pred))) != 2:
        y_prob_pred = model.predict(predictors)

        y_pred=[]
        for i in range(0,len(y_prob_pred)):
            if np.array(y_prob_pred)[i] > threshold:
                a=1
            else:
                a=0
            y_pred.append(a)
    else:
        pass

    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum()
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Logistic Regression Model

In [116...

```
# Adding constant to data for Logistic Regression
X_train_with_intercept = sm.add_constant(X_train_scaled)
X_test_with_intercept = sm.add_constant(X_test_scaled)
```

In [117...

```
X_train_with_intercept.head() # Returns first 5 rows
```

Out[117...

	const	Total_assets	Net_worth	Total_income	Change_in_stock	Total_expenses	Profit_af
0	1.00	-0.77	-0.74	-0.08	-0.10	-0.08	
1	1.00	0.34	0.92	-0.06	-0.07	-0.06	
2	1.00	-0.78	-0.70	-0.08	-0.10	-0.08	
3	1.00	-0.82	-0.80	-0.08	-0.10	-0.08	
4	1.00	-0.71	-0.71	-0.07	-0.10	-0.07	

5 rows × 50 columns

In [118...

```
y_train.reset_index(inplace = True, drop = True)
```

In [119...

```
LogisticReg = LogisticRegression(random_state=1)

LogisticReg.fit(X_train_with_intercept,y_train)
```

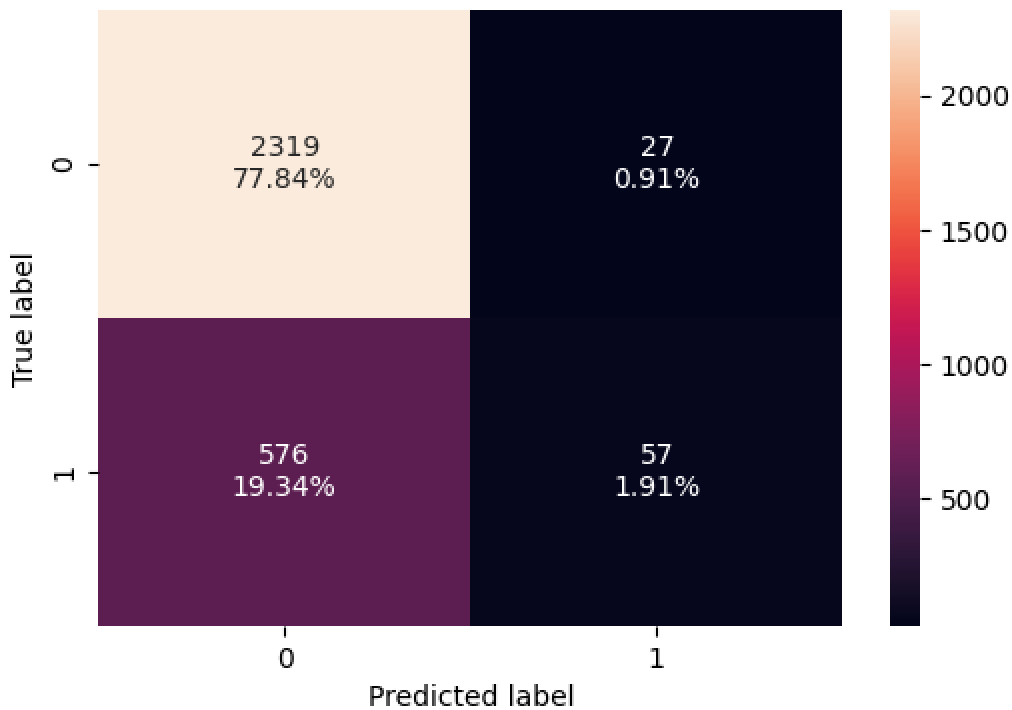
Out[119…

```
▾ LogisticRegression
LogisticRegression(random_state=1)
```

Logistic Regression Model - Training Performance

In [120…

```python
# Create confusion metrix

model_confusion_matrix(LogisticReg, X_train_with_intercept, y_train)
```



In [121…

```python
# Calculate Accuracy, Recall, Precision and F1 score

logistic_regression_perf_train = model_performance_classification(LogisticReg, X_tr
logistic_regression_perf_train
```

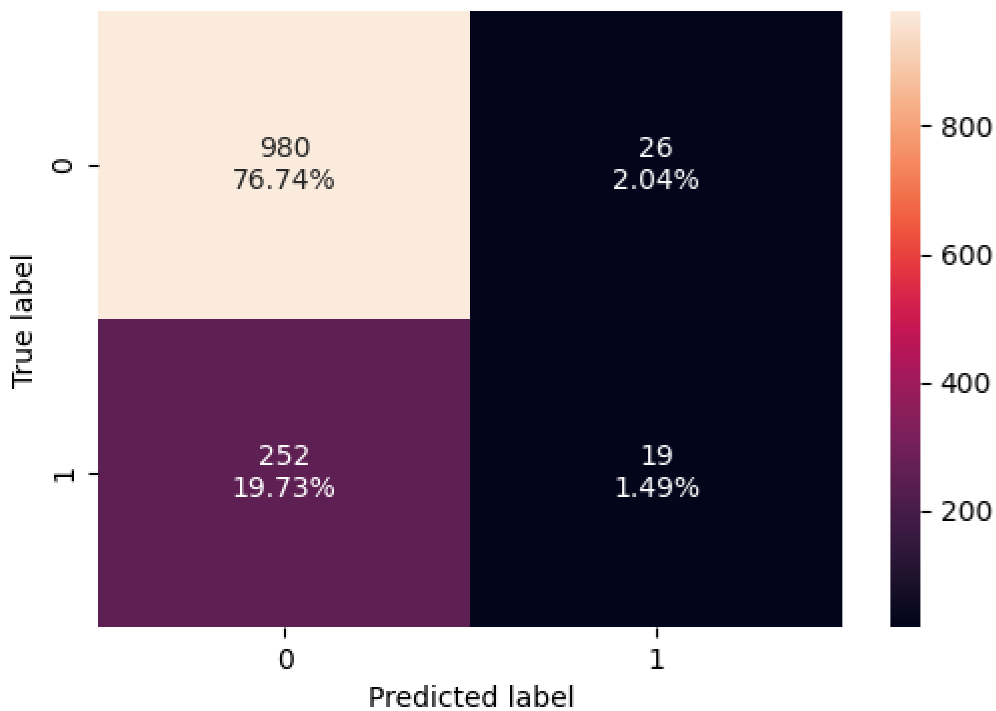
Out[121…

	Accuracy	Recall	Precision	F1 Score
0	0.80	0.09	0.68	0.16

Logistic Regression Model - Test Performance

In [122…

```python
# Create confusion metrix

model_confusion_matrix(LogisticReg, X_test_with_intercept, y_test)
```



In [123…

```
# Calculate Accuracy, Recall, Precision and F1 score

logistic_regression_perf_test = model_performance_classification(LogisticReg, X_tes
logistic_regression_perf_test
```

Out[123…

	Accuracy	Recall	Precision	F1 Score
0	0.78	0.07	0.42	0.12

Random Forest Model

In [124…

```
# Initialise a Random Forest Classifier

rf_classifier = RandomForestClassifier(random_state=1)
rf_classifier.fit(X_train, y_train)
```

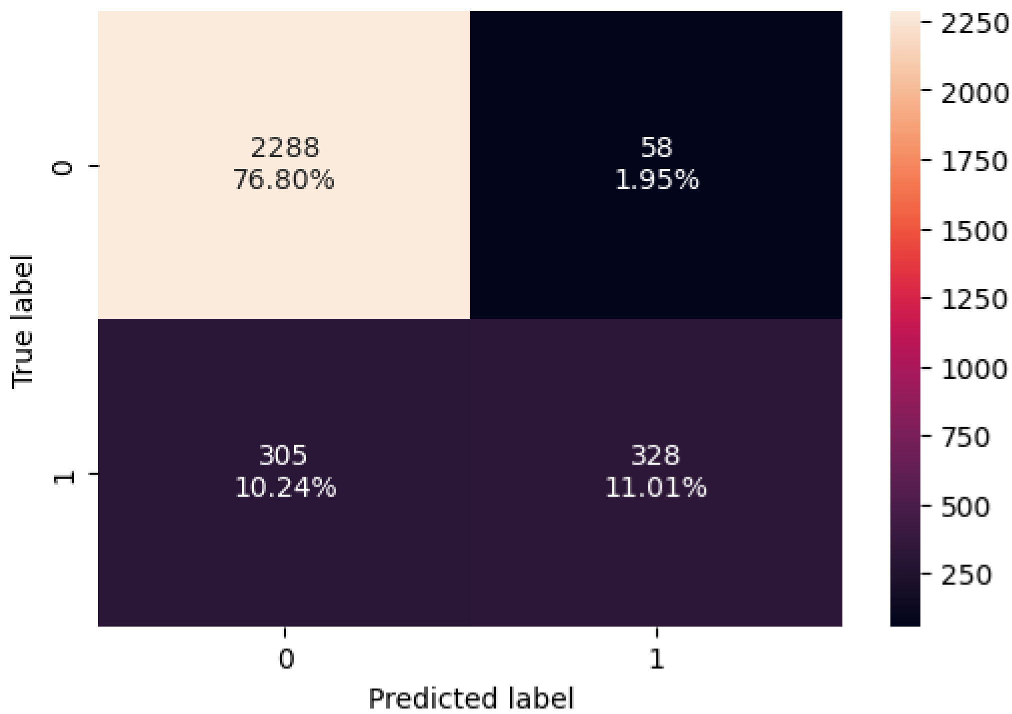
Out[124…

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=1)
```

Random Forest Model - Training Performance

In [125…

```
# Create confusion metrix

model_confusion_matrix(rf_classifier, X_train, y_train)
```



In [126…

```
# Calculate Accuracy, Recall, Precision and F1 score

random_forest_perf_train = model_performance_classification(rf_classifier, X_train,
random_forest_perf_train
```

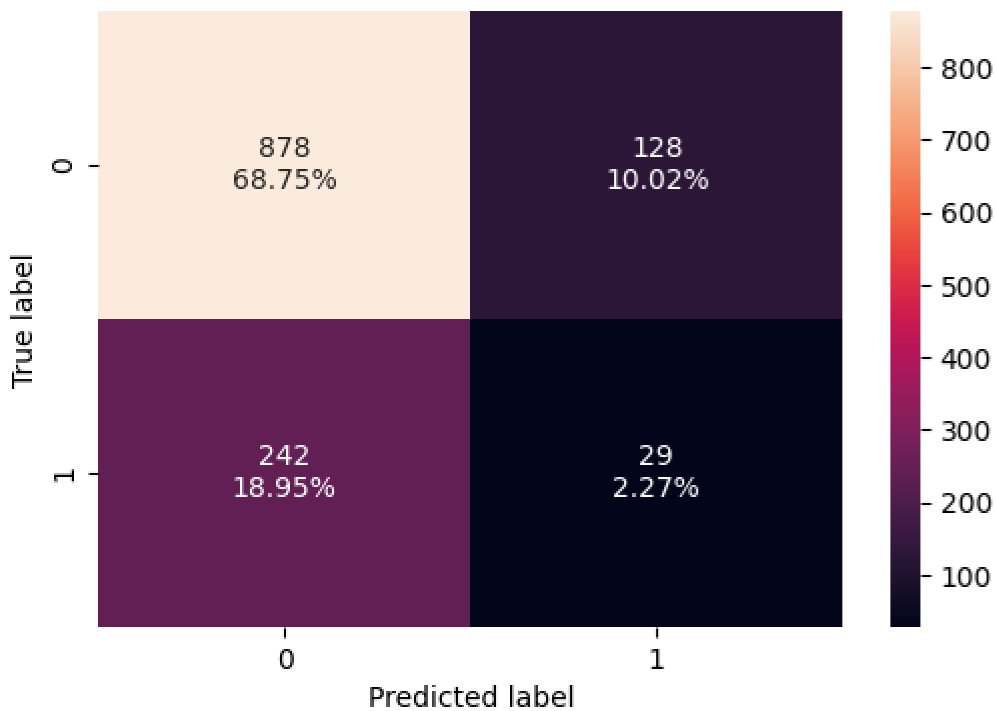
Out[126…

	Accuracy	Recall	Precision	F1 Score
0	0.88	0.52	0.85	0.64

Random Forest Model - Test Performance

In [127…

```
# Create confusion metrix

model_confusion_matrix(rf_classifier, X_test, y_test)
```



In [128…

```
# Calculate Accuracy, Recall, Precision and F1 score

random_forest_perf_test = model_performance_classification(rf_classifier, X_test, y
random_forest_perf_test
```

Out[128…

	Accuracy	Recall	Precision	F1 Score
0	0.71	0.11	0.18	0.14

Model Performance Improvement - Logistic Regression Model

In [129…

```
# defining a function to calculate Variance Inflation Factor (VIF)

def calculate_vif(idf):
    """
    Calculate Variance Inflation Factor (VIF) for each variable in a DataFrame.

    Parameters:
    df (DataFrame): Input DataFrame containing numerical variables.

    Returns:
    vif_df (DataFrame): DataFrame containing variable names and their corresponding
    """
    variables = idf.values
    vif_df = pd.DataFrame()
    vif_df["Variable"] = idf.columns
    vif_df["VIF"] = [variance_inflation_factor(variables, i) for i in range(idf.sha
    return vif_df
```

In [130…

```
# Call the function to calculate VIF
```

```python
vif_result = calculate_vif(X_train_scaled).sort_values(by = 'VIF', ascending=False)

print("Variance Inflation Factors:")
vif_result
```

```
Variance Inflation Factors:
```

Out[130…

	Variable	VIF
0	Total_assets	inf
47	Total_liabilities	inf
2	Total_income	19273782.33
4	Total_expenses	17400995.60
14	Sales	324274.50
5	Profit_after_tax	55680.86
8	Cash_profit	2000.76
7	PBT	1860.68
6	PBDITA	1451.75
3	Change_in_stock	621.28
18	Reserves_and_funds	351.78
29	Net_fixed_assets	266.39
23	Cumulative_retained_profits	222.50
10	PBT_as_perc_of_total_income	170.31
20	Current_liabilities_&_provisions	157.78
31	Current_assets	146.41
11	PAT_as_perc_of_total_income	137.98
22	Shareholders_funds	114.19
1	Net_worth	103.82
15	Income_from_fincial_services	103.46
21	Deferred_tax_liability	96.54
16	Other_income	75.14
28	Contingent_liabilities	70.48
30	Investments	52.16
24	Capital_employed	50.51
33	Quick_ratio_times	46.55
12	Cash_profit_as_perc_of_total_income	39.85
34	Current_ratio_times	38.82
19	Borrowings	34.69
45	EPS	10.94

	Variable	VIF
32	Net_working_capital	10.55
46	Adjusted_EPS	10.09
35	Debt_to_equity_ratio_times	6.25
43	Shares_outstanding	5.30
25	TOL_to_TNW	4.88
26	Total_term_liabilities__to__tangible_net_worth	4.51
17	Total_capital	3.54
44	Equity_face_value	3.08
36	Cash_to_current_liabilities_times	2.76
9	PBDITA_as_perc_of_total_income	2.25
37	Cash_to_average_cost_of_sales_per_day	1.56
13	PAT_as_perc_of_net_worth	1.55
27	Contingent_liabilities__to__Net_worth_perc	1.28
41	WIP_turnover	1.12
40	Finished_goods_turnover	1.11
38	Creditors_turnover	1.02
48	PE_on_BSE	1.02
39	Debtors_turnover	1.02
42	Raw_material_turnover	1.00

Finding and Dropping variables with VIF >= 5

In [131…

```
# Finding variables with VIF >= 5

high_vif_columns = []
for i, row in vif_result.iterrows():
    if row['VIF'] >= 5:
        high_vif_columns.append(row['Variable'])
high_vif_columns
```

Out[131…

```
['Total_assets',
 'Total_liabilities',
 'Total_income',
 'Total_expenses',
 'Sales',
 'Profit_after_tax',
 'Cash_profit',
 'PBT',
 'PBDITA',
 'Change_in_stock',
 'Reserves_and_funds',
 'Net_fixed_assets',
 'Cumulative_retained_profits',
 'PBT_as_perc_of_total_income',
 'Current_liabilities_&_provisions',
 'Current_assets',
 'PAT_as_perc_of_total_income',
 'Shareholders_funds',
 'Net_worth',
 'Income_from_fincial_services',
 'Deferred_tax_liability',
 'Other_income',
 'Contingent_liabilities',
 'Investments',
 'Capital_employed',
 'Quick_ratio_times',
 'Cash_profit_as_perc_of_total_income',
 'Current_ratio_times',
 'Borrowings',
 'EPS',
 'Net_working_capital',
 'Adjusted_EPS',
 'Debt_to_equity_ratio_times',
 'Shares_outstanding']
```

In [132…

```
# Dropping variables with VIF >= 5

X_train_scaled.drop(columns = high_vif_columns, axis=1, inplace=True)
X_test_scaled.drop(columns = high_vif_columns, axis=1, inplace=True)
```

In [135…

```
# Shape of scaled train data

print('Shape of Train dataset:', X_train_scaled.shape)
```

```
Shape of Train dataset: (2979, 15)
```

In [136…

```
# Shape of scaled test data

print('Shape of Test dataset:', X_test_scaled.shape)
```

```
Shape of Test dataset: (1277, 15)
```

In [137…

```
X_train_new_with_intercept = sm.add_constant(X_train_scaled)
X_test_new_with_intercept = sm.add_constant(X_test_scaled)
```

In [138…

```
# Retraining Logistic Regression Model with new data

LogisticReg_improved = LogisticRegression(random_state=1)
LogisticReg_improved.fit(X_train_new_with_intercept,y_train)
```

Out[138…

```
▾ LogisticRegression
LogisticRegression(random_state=1)
```

Finding Optimal Threshold value

In [139…

```
# Finding Optimal Threshold value

logit_y_pred = LogisticReg_improved.predict(X_train_new_with_intercept)
fpr, tpr, thresholds = roc_curve(y_train, logit_y_pred)
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold_logit = round(thresholds[optimal_idx], 3)
optimal_threshold_logit
```

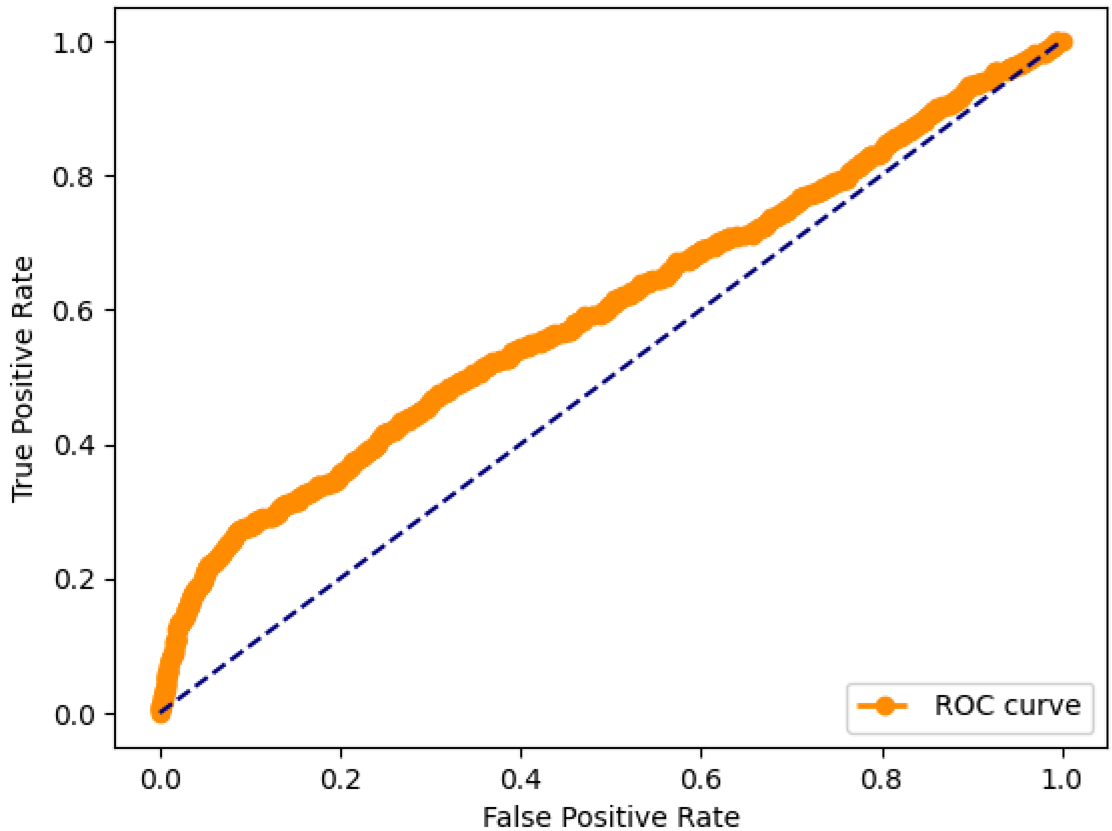
Out[139…

```
1.0
```

In [140…

```
# predict probabilities
probs = LogisticReg_improved.predict_proba(X_train_new_with_intercept)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
auc = roc_auc_score(y_train, probs)
# calculate roc curve
train_fpr, train_tpr, train_thresholds = roc_curve(y_train, probs)
# plot the roc curve for the model
plt.plot(train_fpr, train_tpr, linestyle='--', marker='o', color='darkorange', lw =
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Fig 3: ROC curve, AUC = %.2f - Logistic Regression Model'%auc)
plt.legend(loc="lower right")
plt.show()
```

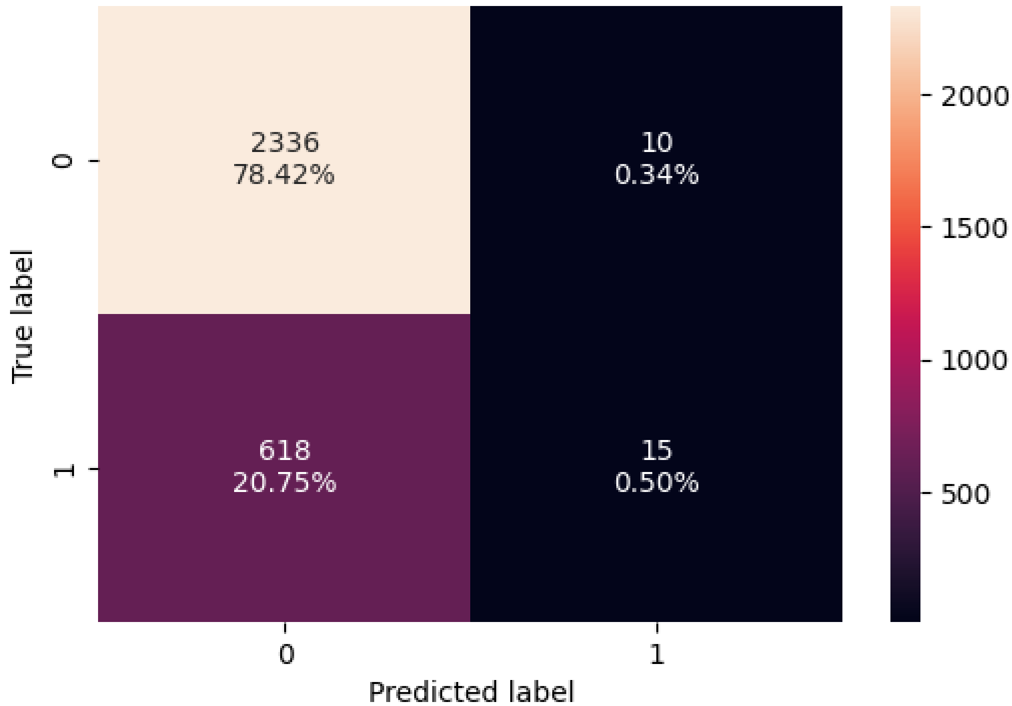
Fig 3: ROC curve, AUC = 0.60 - Logistic Regression Model



Logistic Regression Performance - Training Set

In [141…

```
# Create confusion metrix

model_confusion_matrix(LogisticReg_improved, X_train_new_with_intercept, y_train, o
```



In [142…

```
# Calculate Accuracy, Recall, Precision and F1 score

logistic_regression_tuned_perf_train = model_performance_classification(
    LogisticReg_improved, X_train_new_with_intercept, y_train, optimal_threshold_lo
)
logistic_regression_tuned_perf_train
```

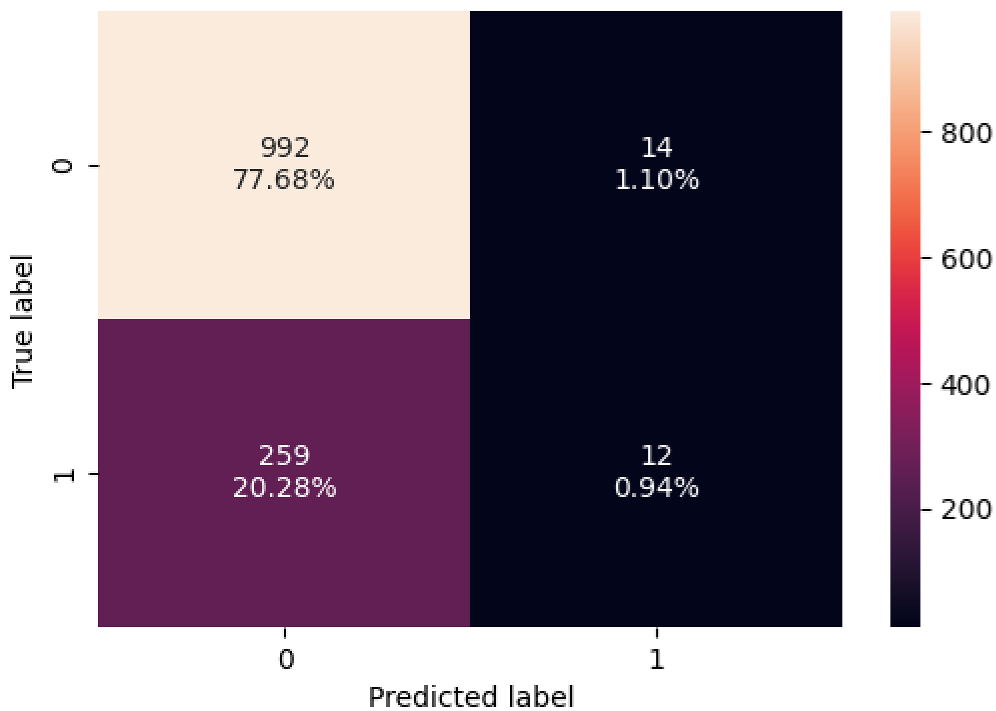
Out[142…

	Accuracy	Recall	Precision	F1 Score
0	0.79	0.02	0.60	0.05

Logistic Regression Performance - Test Set

In [143…

```
# Create confusion metrix

model_confusion_matrix(LogisticReg_improved, X_test_new_with_intercept, y_test, opt
```



In [144…

```
# Calculate Accuracy, Recall, Precision and F1 score

logistic_regression_tuned_perf_test = model_performance_classification(
    LogisticReg_improved, X_test_new_with_intercept, y_test, optimal_threshold_logi
)
logistic_regression_tuned_perf_test
```

Out[144…

	Accuracy	Recall	Precision	F1 Score
0	0.79	0.04	0.46	0.08

Model Performance Improvement - Random Forest Model

In [145…

```
param_grid = {
    'n_estimators': [10, 50, 100],  # Number of trees in the forest
    'max_depth': [5, 7, 9],     # Maximum depth of the trees
    'min_samples_split': [2, 5, 10], # Minimum number of samples required to split
    'min_samples_leaf': [5, 6, 7],  # Minimum number of samples required at each le
}

rf_classifier = RandomForestClassifier(class_weight='balanced', random_state=1)

grid_search = GridSearchCV(
    estimator=rf_classifier,
    param_grid=param_grid,
    cv=5,
    scoring='recall',
    n_jobs=-1
)

grid_search.fit(X_train, y_train)
```

```
print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'max_depth': 5, 'min_samples_leaf': 6, 'min_samples_split': 2, 'n_
estimators': 10}
```

In [146…

```
# Random Forest Model creation - Tuned

best_rf_classifier = grid_search.best_estimator_
```

In [147…

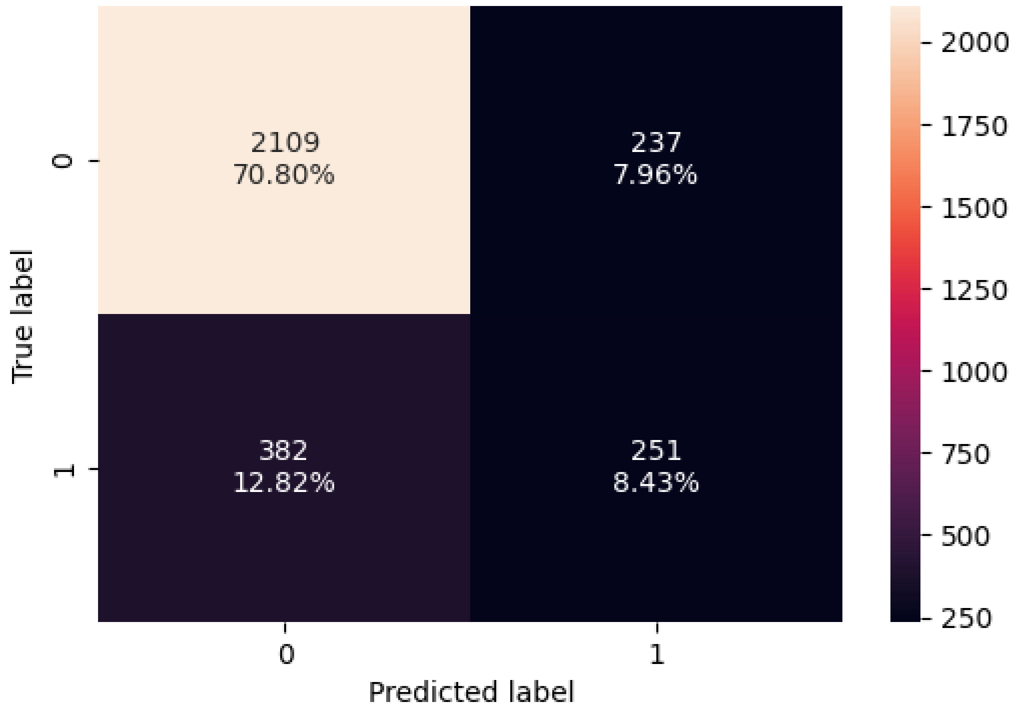
```
params_used = best_rf_classifier.get_params()

# Print the parameters

print("Parameters used in the Random Forest Classifier:\n")
for param_name, param_value in params_used.items():
    print(f"{param_name}: {param_value}")
```

```
Parameters used in the Random Forest Classifier:

bootstrap: True
ccp_alpha: 0.0
class_weight: balanced
criterion: gini
max_depth: 5
max_features: sqrt
max_leaf_nodes: None
max_samples: None
min_impurity_decrease: 0.0
min_samples_leaf: 6
min_samples_split: 2
min_weight_fraction_leaf: 0.0
n_estimators: 10
n_jobs: None
oob_score: False
random_state: 1
verbose: 0
warm_start: False
```

Random Forest Performance - Training Set

In [148…

```
# Create confusion metrix

model_confusion_matrix(best_rf_classifier, X_train, y_train)
```



In [149…

```
# Calculate Accuracy, Recall, Precision and F1 score

random_forest_tuned_perf_train = model_performance_classification(best_rf_classifie
random_forest_tuned_perf_train
```

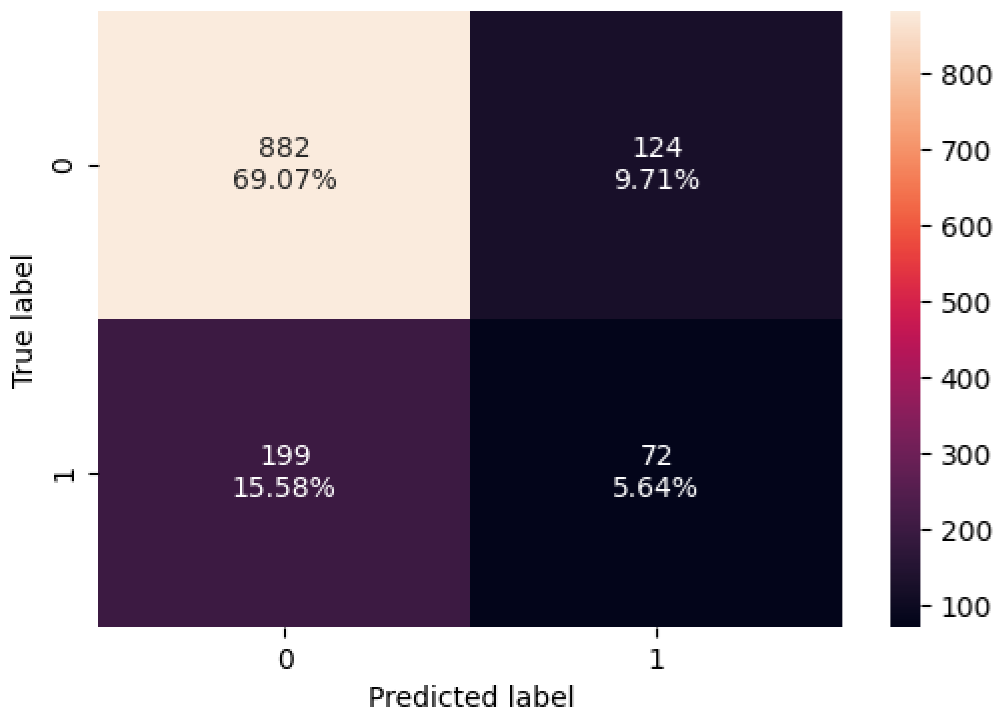
Out[149…

	Accuracy	Recall	Precision	F1 Score
0	0.79	0.40	0.51	0.45

Random Forest Performance - Test Set

In [150…

```
# Create confusion metrix

model_confusion_matrix(best_rf_classifier, X_test, y_test)
```



In [151…

```
# Calculate Accuracy, Recall, Precision and F1 score

random_forest_tuned_perf_test = model_performance_classification(best_rf_classifier
random_forest_tuned_perf_test
```

Out[151…

	Accuracy	Recall	Precision	F1 Score
0	0.75	0.27	0.37	0.31

Model Comparison and Final Model Selection

In [152…

```
# Training performance comparison

models_train_comp_df = pd.concat(
    [
        logistic_regression_perf_train.T,
        logistic_regression_tuned_perf_train.T,
        random_forest_perf_train.T,
        random_forest_tuned_perf_train.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Logistic Regression",
    "Tuned Logistic Regression",
    "Random Forest",
    "Tuned Random Forest",
]
print("Training performance comparison:")
models_train_comp_df
```

Training performance comparison:

Out[152…

	Logistic Regression	Tuned Logistic Regression	Random Forest	Tuned Random Forest
Accuracy	0.80	0.79	0.88	0.79
Recall	0.09	0.02	0.52	0.40
Precision	0.68	0.60	0.85	0.51
F1 Score	0.16	0.05	0.64	0.45

In [153…

```
# Testing performance comparison

models_test_comp_df = pd.concat(
    [
        logistic_regression_perf_test.T,
        logistic_regression_tuned_perf_test.T,
        random_forest_perf_test.T,
        random_forest_tuned_perf_test.T,
    ],
    axis=1,
)
models_test_comp_df.columns = [
    "Logistic Regression",
    "Tuned Logistic Regression",
    "Random Forest",
    "Tuned Random Forest",
]
print("Testing performance comparison:")
models_test_comp_df
```

Testing performance comparison:

Out[153…

	Logistic Regression	Tuned Logistic Regression	Random Forest	Tuned Random Forest
Accuracy	0.78	0.79	0.71	0.75
Recall	0.07	0.04	0.11	0.27
Precision	0.42	0.46	0.18	0.37
F1 Score	0.12	0.08	0.14	0.31

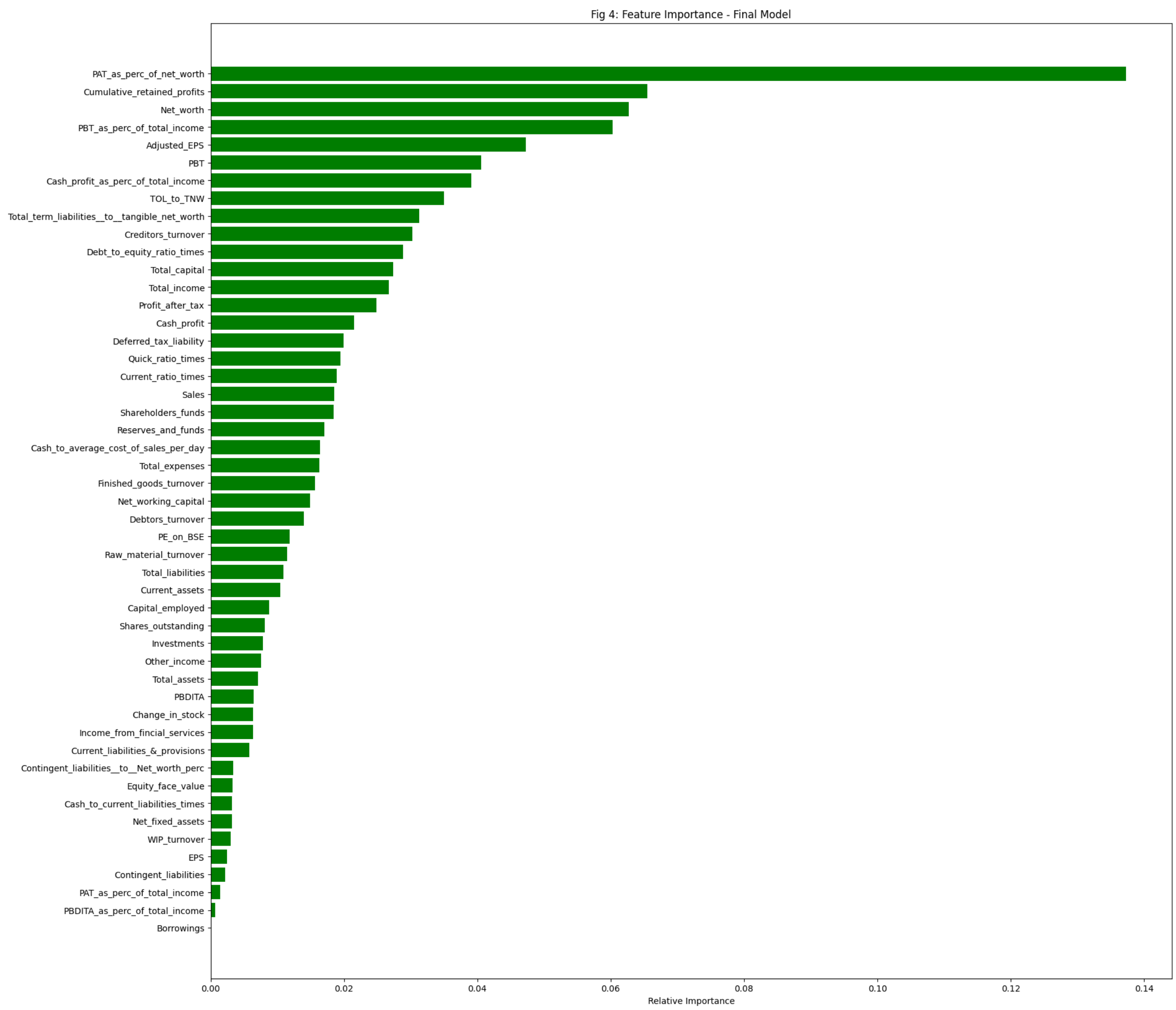
Final Model: Tuned Random Forest (It has highest F1 Score and Recall value). Precision and Recall values for test data is close to training data.

Feature importance based on Final Model

In [154…

```
# Feature importance based on Final Model

feature_names = X_train.columns
importances = best_rf_classifier.feature_importances_
indices = np.argsort(importances)
```

```
plt.figure(figsize=(20, 20))
plt.title("Fig 4: Feature Importance - Final Model")
plt.barh(range(len(indices)), importances[indices], color="green", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Observations and Insights:

- Features linked to assets, profit and liabilities are the most important factors for the companies.

Actionable Insights:

- Features linked to assets (ex: PAT_as_perc_of_net_worth, Net_worth) are the most important factors for the companies.
- Features linked to profit (ex: Cumulative_retained_profits, PBT_as_perc_of_total_income) are the most important factors for the companies.

- Features linked to liabilities (ex: TOL_to_TNW, Total_term_liabilities__to__tangible_net_worth) are the most important factors for the companies.

Business Recommendations:

- Companies can increase their assets so that there is less probability of default in coming years.
- Companies can increase their profit so that there is less probability of default in coming years.
- Companies can decrease their liabilities so that there is less probability of default in coming years.

In [ ]: