RAHUL _ JHA

Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.style
import seaborn as sns

%config InlineBackend.figure_format ='retina'
%matplotlib inline
sns.set(color_codes=True)
sns.set_palette('Accent_r')

from scipy.stats import zscore
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.linear_model import LogisticRegression
from sklearn import metrics, model_selection
from sklearn.metrics import confusion_matrix,classification_report,roc_auc_score,accuracy_s

from sklearn.model_selection import train_test_split, GridSearchCV

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
pd.set_option('display.expand_frame_repr', False)
```

In [ ]:

```

```

Load Dataset

In [2]:

```
df = pd.read_excel('Company_Data2015-1.xlsx')
df.head()
```

Out[2]:

	Co_Code	Co_Name	Networth Next Year	Equity Paid Up	Networth	Capital Employed	Total Debt	Gross Block	Net Working Capital	(
0	16974	Hind.Cables	-8021.60	419.36	-7027.48	-1007.24	5936.03	474.30	-1076.34	
1	21214	Tata Tele. Mah.	-3986.19	1954.93	-2968.08	4458.20	7410.18	9070.86	-1098.88	
2	14852	ABG Shipyard	-3192.58	53.84	506.86	7714.68	6944.54	1281.54	4496.25	9
3	2439	GTL	-3054.51	157.30	-623.49	2353.88	2326.05	1033.69	-2612.42	1
4	23505	Bharati Defence	-2967.36	50.30	-1070.83	4675.33	5740.90	1084.20	1836.23	4

Renaming Column Names

In [3]:

```
df.columns = df.columns.str.replace(' ', '_').str.replace('(', '').str.replace(')', '').str
```

In [4]:

```
df['Current_Liabilities_and_Provisions'] = df['Current_Liabilities_and_Provisions_']
df['Total_Assets_to_Liabilities'] = df['Total_Assets_to_Liabilities_']
df['Gross_Block'] = df['Gross_Block_']
df['Net_Working_Capital'] = df['Net_Working_Capital_']
df['Current_Assets'] = df['Current_Assets_']
df['Book_Value_Adj_Unit_Curr'] = df['Book_Value_Adj._Unit_Curr']
```

In [5]:

```
df.drop(['Current_Liabilities_and_Provisions_', 'Total_Assets_to_Liabilities_', 'Gross_Bloc
```

Basic Exploration

In [6]:

```
print('Number of Rows(Companies) =', df.shape[0])
print('Number of Columns =', df.shape[1])
```

```
Number of Rows(Companies) = 3586
Number of Columns = 67
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3586 entries, 0 to 3585
Data columns (total 67 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Co_Code                               3586 non-null   int64  
 1   Co_Name                               3586 non-null   object 
 2   Networth_Next_Year                    3586 non-null   float64
 3   Equity_Paid_Up                        3586 non-null   float64
 4   Networth                              3586 non-null   float64
 5   Capital_Employed                      3586 non-null   float64
 6   Total_Debt                            3586 non-null   float64
 7   Gross_Sales                           3586 non-null   float64
 8   Net_Sales                             3586 non-null   float64
 9   Other_Income                          3586 non-null   float64
 10  Value_Of_Output                       3586 non-null   float64
 11  Cost_of_Production                    3586 non-null   float64
 12  Selling_Cost                          3586 non-null   float64
 13  PBIDT                                 3586 non-null   float64
 14  PBDT                                  3586 non-null   float64
 15  PBIT                                  3586 non-null   float64
 16  PBT                                   3586 non-null   float64
 17  PAT                                   3586 non-null   float64
 18  Adjusted_PAT                          3586 non-null   float64
 19  CP                                    3586 non-null   float64
 20  Revenue_earnings_in_forex             3586 non-null   float64
 21  Revenue_expenses_in_forex             3586 non-null   float64
 22  Capital_expenses_in_forex             3586 non-null   float64
 23  Book_Value_Unit_Curr                  3586 non-null   float64
 24  Market_Capitalisation                 3586 non-null   float64
 25  CEPS_annualised_Unit_Curr             3586 non-null   float64
 26  Cash_Flow_From_Operating_Activities   3586 non-null   float64
 27  Cash_Flow_From_Investing_Activities   3586 non-null   float64
 28  Cash_Flow_From_Financing_Activities   3586 non-null   float64
 29  ROG_Net_Worth_perc                    3586 non-null   float64
 30  ROG_Capital_Employed_perc             3586 non-null   float64
 31  ROG_Gross_Block_perc                  3586 non-null   float64
 32  ROG_Gross_Sales_perc                  3586 non-null   float64
 33  ROG_Net_Sales_perc                    3586 non-null   float64
 34  ROG_Cost_of_Production_perc           3586 non-null   float64
 35  ROG_Total_Assets_perc                 3586 non-null   float64
 36  ROG_PBIDT_perc                        3586 non-null   float64
 37  ROG_PBDT_perc                         3586 non-null   float64
 38  ROG_PBIT_perc                         3586 non-null   float64
 39  ROG_PBT_perc                          3586 non-null   float64
 40  ROG_PAT_perc                          3586 non-null   float64
 41  ROG_CP_perc                           3586 non-null   float64
 42  ROG_Revenue_earnings_in_forex_perc    3586 non-null   float64
 43  ROG_Revenue_expenses_in_forex_perc    3586 non-null   float64
 44  ROG_Market_Capitalisation_perc        3586 non-null   float64
 45  Current_Ratio_Latest                  3585 non-null   float64
 46  Fixed_Assets_Ratio_Latest             3585 non-null   float64
 47  Inventory_Ratio_Latest                3585 non-null   float64
 48  Debtors_Ratio_Latest                  3585 non-null   float64
 49  Total_Asset_Turnover_Ratio_Latest     3585 non-null   float64
 50  Interest_Cover_Ratio_Latest           3585 non-null   float64
```

```
 51  PBIDTM_perc_Latest                  3585 non-null   float64
 52  PBITM_perc_Latest                   3585 non-null   float64
 53  PBDTM_perc_Latest                   3585 non-null   float64
 54  CPM_perc_Latest                     3585 non-null   float64
 55  APATM_perc_Latest                   3585 non-null   float64
 56  Debtors_Velocity_Days               3586 non-null   int64
 57  Creditors_Velocity_Days             3586 non-null   int64
 58  Inventory_Velocity_Days             3483 non-null   float64
 59  Value_of_Output_to_Total_Assets     3586 non-null   float64
 60  Value_of_Output_to_Gross_Block      3586 non-null   float64
 61  Current_Liabilities_and_Provisions  3586 non-null   float64
 62  Total_Assets_to_Liabilities         3586 non-null   float64
 63  Gross_Block                         3586 non-null   float64
 64  Net_Working_Capital                 3586 non-null   float64
 65  Current_Assets                      3586 non-null   float64
 66  Book_Value_Adj_Unit_Curr            3582 non-null   float64
dtypes: float64(63), int64(3), object(1)
memory usage: 1.8+ MB
```

In [8]:

```
df.head()
```

Out[8]:

	Co_Code	Co_Name	Networth_Next_Year	Equity_Paid_Up	Networth	Capital_Employed	To
0	16974	Hind.Cables	-8021.60	419.36	-7027.48	-1007.24	
1	21214	Tata Tele. Mah.	-3986.19	1954.93	-2968.08	4458.20	
2	14852	ABG Shipyard	-3192.58	53.84	506.86	7714.68	
3	2439	GTL	-3054.51	157.30	-623.49	2353.88	
4	23505	Bharati Defence	-2967.36	50.30	-1070.83	4675.33	

In [9]:

```
df.describe().T
```

Out[9]:

	count	mean	std	min	25
Co_Code	3586.0	16065.388734	19776.817379	4.00	3029.25
Networth_Next_Year	3586.0	725.045251	4769.681004	-8021.60	3.98
Equity_Paid_Up	3586.0	62.966584	778.761744	0.00	3.75
Networth	3586.0	649.746299	4091.988792	-7027.48	3.89
Capital_Employed	3586.0	2799.611054	26975.135385	-1824.75	7.60
Total_Debt	3586.0	1994.823779	23652.842746	-0.72	0.03
Gross_Sales	3586.0	1123.738985	10603.703837	-62.59	1.44
Net_Sales	3586.0	1079.702579	9996.574173	-62.59	1.44
Other_Income	3586.0	48.729824	426.040665	-448.72	0.02
Value_Of_Output	3586.0	1077.187292	9843.880293	-119.10	1.41
Cost_of_Production	3586.0	798.544621	9076.702982	-22.65	0.94
Selling_Cost	3586.0	25.554997	194.244466	0.00	0.00
PBIDT	3586.0	248.175282	1949.593350	-4655.14	0.04
PBDT	3586.0	116.268795	956.199566	-5874.53	0.00
PBIT	3586.0	217.659395	1850.972782	-4812.95	0.00
PBT	3586.0	85.752909	799.925768	-6032.34	-0.06
PAT	3586.0	61.218313	620.298432	-6032.34	-0.06
Adjusted_PAT	3586.0	60.058963	580.432912	-4418.72	-0.09
CP	3586.0	91.734200	780.790561	-5874.53	0.00
Revenue_earnings_in_forex	3586.0	131.165270	1150.730209	0.00	0.00
Revenue_expenses_in_forex	3586.0	256.327002	4132.339619	0.00	0.00
Capital_expenses_in_forex	3586.0	7.655689	111.432070	0.00	0.00
Book_Value_Unit_Curr	3586.0	157.237836	1622.664105	-3371.57	7.96
Market_Capitalisation	3586.0	1664.092387	12805.173084	0.00	0.00
CEPS_annualised_Unit_Curr	3586.0	36.018709	828.420796	-1808.00	0.00
Cash_Flow_From_Operating_Activities	3586.0	65.770750	1455.048376	-25469.23	-0.30
Cash_Flow_From_Investing_Activities	3586.0	-60.870365	701.974713	-23843.45	-5.11
Cash_Flow_From_Financing_Activities	3586.0	11.436453	1272.257361	-38374.04	-5.84
ROG_Net_Worth_perc	3586.0	1237.624576	41041.930017	-14485.71	-1.48
ROG_Capital_Employed_perc	3586.0	2988.884612	126472.870285	-8614.63	-3.83
ROG_Gross_Block_perc	3586.0	37.554306	893.619402	-116.12	0.00
ROG_Gross_Sales_perc	3586.0	242.672962	6103.527897	-5503.70	-8.07
ROG_Net_Sales_perc	3586.0	242.588530	6103.487655	-5503.70	-8.11
ROG_Cost_of_Production_perc	3586.0	310.488405	5573.215095	-2130.23	-7.24

	count	mean	std	min	2
ROG_Total_Assets_perc	3586.0	2793.282621	125941.653747	-136.13	-3.97
ROG_PBIDT_perc	3586.0	375.852181	23278.396117	-52200.00	-23.36
ROG_PBDT_perc	3586.0	336.379947	20353.396660	-52200.00	-30.59
ROG_PBIT_perc	3586.0	374.699958	22462.789381	-58500.00	-31.35
ROG_PBT_perc	3586.0	224.070248	19659.232661	-78900.00	-41.23
ROG_PAT_perc	3586.0	112.231654	13480.515287	-114500.00	-43.73
ROG_CP_perc	3586.0	221.091523	13980.202791	-52200.00	-29.50
ROG_Revenue_earnings_in_forex_perc	3586.0	37.227844	658.666041	-100.00	0.00
ROG_Revenue_expenses_in_forex_perc	3586.0	364.863221	15233.643027	-100.00	0.00
ROG_Market_Capitalisation_perc	3586.0	63.682220	1047.928144	-98.05	0.00
Current_Ratio_Latest	3585.0	12.056603	108.410131	0.00	0.88
Fixed_Assets_Ratio_Latest	3585.0	51.538840	681.150910	0.00	0.27
Inventory_Ratio_Latest	3585.0	37.798946	458.189394	0.00	0.00
Debtors_Ratio_Latest	3585.0	33.026996	489.563498	0.00	0.42
Total_Asset_Turnover_Ratio_Latest	3585.0	1.237236	2.673228	0.00	0.07
Interest_Cover_Ratio_Latest	3585.0	16.387894	351.737840	-5450.00	0.00
PBIDTM_perc_Latest	3585.0	-51.162890	1795.131025	-78870.45	0.00
PBITM_perc_Latest	3585.0	-109.213414	3057.635870	-141600.00	0.00
PBDTM_perc_Latest	3585.0	-311.570357	10921.592639	-590500.00	0.00
CPM_perc_Latest	3585.0	-307.005632	10676.149629	-572000.00	0.00
APATM_perc_Latest	3585.0	-365.056187	12500.051387	-688600.00	0.00
Debtors_Velocity_Days	3586.0	603.894032	10636.759580	0.00	8.00
Creditors_Velocity_Days	3586.0	2057.854992	54169.479197	0.00	8.00
Inventory_Velocity_Days	3483.0	79.644559	137.847792	-199.00	0.00
Value_of_Output_to_Total_Assets	3586.0	0.819757	1.201400	-0.33	0.07
Value_of_Output_to_Gross_Block	3586.0	61.884548	976.824352	-61.00	0.27
Current_Liabilities_and_Provisions	3586.0	391.992078	2675.001631	-0.23	0.73
Total_Assets_to_Liabilities	3586.0	1778.453751	11437.574690	-4.51	10.55
Gross_Block	3586.0	594.178829	4871.547802	-41.19	0.57
Net_Working_Capital	3586.0	410.809665	6301.218546	-13162.42	0.94
Current_Assets	3586.0	1960.349172	22577.570829	-0.91	4.00
Book_Value_Adj_Unit_Curr	3582.0	2243.152917	128283.728186	-33715.70	7.06

Checking Duplicates

In [10]:

```
print('Number of Duplicates =', df.duplicated().sum().sum())
```

```
Number of Duplicates = 0
```

Dropping Unique Identifiers

Unique Identifiers do not add any value to model building, hence dropping them

In [11]:

```
df.drop(['Co_Code', 'Co_Name'], axis=1, inplace=True)
df.head()
```

Out[11]:

	Networth_Next_Year	Equity_Paid_Up	Networth	Capital_Employed	Total_Debt	Gross_Sales
0	-8021.60	419.36	-7027.48	-1007.24	5936.03	0.00
1	-3986.19	1954.93	-2968.08	4458.20	7410.18	2892.73
2	-3192.58	53.84	506.86	7714.68	6944.54	392.13
3	-3054.51	157.30	-623.49	2353.88	2326.05	1354.39
4	-2967.36	50.30	-1070.83	4675.33	5740.90	38.72

Creating Target/Response Variable

- We consider 'Networth Next Year' as our Default Variable
- SO, we call negative values as Default = 1
- And, positive values as Default = 0

In [12]:

```
df_orig = df.copy()
df['default'] = np.where(df['Networth_Next_Year']>=0, 0, 1)
df[['default', 'Networth_Next_Year']].head()
df[['default', 'Networth_Next_Year']].tail()
```

Out[12]:

	default	Networth_Next_Year
0	1	-8021.60
1	1	-3986.19
2	1	-3192.58
3	1	-3054.51
4	1	-2967.36

Out[12]:

	default	Networth_Next_Year
3581	0	72677.77
3582	0	79162.19
3583	0	88134.31
3584	0	91293.70
3585	0	111729.10

Dropping variable 'Networth Next Year'

In [13]:

```
df.drop('Networth_Next_Year', axis=1, inplace=True)
df.head()
```

Out[13]:

	Equity_Paid_Up	Networth	Capital_Employed	Total_Debt	Gross_Sales	Net_Sales	Other_Inc
0	419.36	-7027.48	-1007.24	5936.03	0.00	0.00	
1	1954.93	-2968.08	4458.20	7410.18	2892.73	2892.73	4
2	53.84	506.86	7714.68	6944.54	392.13	392.13	
3	157.30	-623.49	2353.88	2326.05	1354.39	1354.39	22
4	50.30	-1070.83	4675.33	5740.90	38.72	38.72	

Missing Value Treatment

In [14]:

```
print('MISSING VALUES PER VARIABLE -->')
print('')
df.isnull().sum()
```

```
MISSING VALUES PER VARIABLE -->
```

Out[14]:

```
Equity_Paid_Up                          0
Networth                                0
Capital_Employed                        0
Total_Debt                              0
Gross_Sales                             0
Net_Sales                               0
Other_Income                            0
Value_Of_Output                         0
Cost_of_Production                      0
Selling_Cost                            0
PBIDT                                   0
PBDT                                    0
PBIT                                    0
PBT                                     0
PAT                                     0
Adjusted_PAT                            0
CP                                      0
Revenue_earnings_in_forex               0
Revenue_expenses_in_forex               0
Capital_expenses_in_forex               0
Book_Value_Unit_Curr                    0
Market_Capitalisation                   0
CEPS_annualised_Unit_Curr               0
Cash_Flow_From_Operating_Activities     0
Cash_Flow_From_Investing_Activities     0
Cash_Flow_From_Financing_Activities     0
ROG_Net_Worth_perc                      0
ROG_Capital_Employed_perc               0
ROG_Gross_Block_perc                    0
ROG_Gross_Sales_perc                    0
ROG_Net_Sales_perc                      0
ROG_Cost_of_Production_perc             0
ROG_Total_Assets_perc                   0
ROG_PBIDT_perc                          0
ROG_PBDT_perc                           0
ROG_PBIT_perc                           0
ROG_PBT_perc                            0
ROG_PAT_perc                            0
ROG_CP_perc                             0
ROG_Revenue_earnings_in_forex_perc      0
ROG_Revenue_expenses_in_forex_perc      0
ROG_Market_Capitalisation_perc          0
Current_Ratio_Latest                    1
Fixed_Assets_Ratio_Latest               1
Inventory_Ratio_Latest                  1
Debtors_Ratio_Latest                    1
Total_Asset_Turnover_Ratio_Latest       1
Interest_Cover_Ratio_Latest             1
PBIDTM_perc_Latest                      1
```

```
PBITM_perc_Latest                    1
PBDTM_perc_Latest                    1
CPM_perc_Latest                      1
APATM_perc_Latest                    1
Debtors_Velocity_Days                0
Creditors_Velocity_Days              0
Inventory_Velocity_Days            103
Value_of_Output_to_Total_Assets      0
Value_of_Output_to_Gross_Block       0
Current_Liabilities_and_Provisions   0
Total_Assets_to_Liabilities          0
Gross_Block                          0
Net_Working_Capital                  0
Current_Assets                       0
Book_Value_Adj_Unit_Curr             4
default                              0
dtype: int64
```

In [15]:

```
print('PERCENT OF MISSING VALUES TO THE WHOLE DATA --->')
print('')
round(df.isna().sum()*100/df.shape[0], 2)
```

```
PERCENT OF MISSING VALUES TO THE WHOLE DATA --->
```

Out[15]:

```
Equity_Paid_Up                          0.00
Networth                                0.00
Capital_Employed                        0.00
Total_Debt                              0.00
Gross_Sales                             0.00
Net_Sales                               0.00
Other_Income                            0.00
Value_Of_Output                         0.00
Cost_of_Production                      0.00
Selling_Cost                            0.00
PBIDT                                   0.00
PBDT                                    0.00
PBIT                                    0.00
PBT                                     0.00
PAT                                     0.00
Adjusted_PAT                            0.00
CP                                      0.00
Revenue_earnings_in_forex               0.00
Revenue_expenses_in_forex               0.00
Capital_expenses_in_forex               0.00
Book_Value_Unit_Curr                    0.00
Market_Capitalisation                   0.00
CEPS_annualised_Unit_Curr               0.00
Cash_Flow_From_Operating_Activities     0.00
Cash_Flow_From_Investing_Activities     0.00
Cash_Flow_From_Financing_Activities     0.00
ROG_Net_Worth_perc                      0.00
ROG_Capital_Employed_perc               0.00
ROG_Gross_Block_perc                    0.00
ROG_Gross_Sales_perc                    0.00
ROG_Net_Sales_perc                      0.00
ROG_Cost_of_Production_perc             0.00
ROG_Total_Assets_perc                   0.00
ROG_PBIDT_perc                          0.00
ROG_PBDT_perc                           0.00
ROG_PBIT_perc                           0.00
ROG_PBT_perc                            0.00
ROG_PAT_perc                            0.00
ROG_CP_perc                             0.00
ROG_Revenue_earnings_in_forex_perc      0.00
ROG_Revenue_expenses_in_forex_perc      0.00
ROG_Market_Capitalisation_perc          0.00
Current_Ratio_Latest                    0.03
Fixed_Assets_Ratio_Latest               0.03
Inventory_Ratio_Latest                  0.03
Debtors_Ratio_Latest                    0.03
Total_Asset_Turnover_Ratio_Latest       0.03
Interest_Cover_Ratio_Latest             0.03
PBIDTM_perc_Latest                      0.03
PBITM_perc_Latest                       0.03
```

```
PBDTM_perc_Latest                    0.03
CPM_perc_Latest                      0.03
APATM_perc_Latest                    0.03
Debtors_Velocity_Days                0.00
Creditors_Velocity_Days              0.00
Inventory_Velocity_Days              2.87
Value_of_Output_to_Total_Assets      0.00
Value_of_Output_to_Gross_Block       0.00
Current_Liabilities_and_Provisions   0.00
Total_Assets_to_Liabilities          0.00
Gross_Block                          0.00
Net_Working_Capital                  0.00
Current_Assets                       0.00
Book_Value_Adj_Unit_Curr             0.11
default                              0.00
dtype: float64
```

- We note that there are not many missing values
- We'll come back to this

Checking Zero Values

In [16]:

```
(df.drop('default', axis=1)==0).sum().sort_values(ascending=False)
```

Out[16]:

```
Capital_expenses_in_forex              2892
ROG_Revenue_earnings_in_forex_perc     2269
Revenue_earnings_in_forex              2230
ROG_Revenue_expenses_in_forex_perc     1971
Revenue_expenses_in_forex              1909
ROG_Market_Capitalisation_perc         1422
Market_Capitalisation                  1249
Inventory_Velocity_Days                1123
Inventory_Ratio_Latest                 1100
Interest_Cover_Ratio_Latest            1001
ROG_Gross_Block_perc                    940
Selling_Cost                            938
Total_Debt                              844
Debtors_Ratio_Latest                    745
Debtors_Velocity_Days                   729
Creditors_Velocity_Days                 704
Other_Income                            690
Cash_Flow_From_Financing_Activities     656
Value_of_Output_to_Gross_Block          611
Fixed_Assets_Ratio_Latest               607
Cash_Flow_From_Investing_Activities     510
Total_Asset_Turnover_Ratio_Latest       493
Value_of_Output_to_Total_Assets         485
ROG_Gross_Sales_perc                    455
ROG_Net_Sales_perc                      455
APATM_perc_Latest                       417
CPM_perc_Latest                         401
PBITM_perc_Latest                       393
PBDTM_perc_Latest                       392
Gross_Block                             390
PBIDTM_perc_Latest                      390
ROG_Cost_of_Production_perc             368
ROG_PAT_perc                            347
ROG_PBT_perc                            326
ROG_PBIT_perc                           315
ROG_PBDT_perc                           307
ROG_CP_perc                             307
Net_Sales                               306
Gross_Sales                             305
Value_Of_Output                         290
ROG_PBIDT_perc                          290
ROG_Net_Worth_perc                      201
ROG_Capital_Employed_perc               190
Cash_Flow_From_Operating_Activities     180
ROG_Total_Assets_perc                   173
Cost_of_Production                      159
CEPS_annualised_Unit_Curr               139
Adjusted_PAT                            124
PAT                                     121
PBT                                     119
CP                                      111
PBIT                                    109
PBDT                                    108
PBIDT                                   102
Current_Liabilities_and_Provisions       98
```

```
Current_Ratio_Latest                87
Net_Working_Capital                 48
Current_Assets                      27
Capital_Employed                     5
Total_Assets_to_Liabilities          3
Networth                             3
Book_Value_Adj_Unit_Curr             1
Book_Value_Unit_Curr                 1
Equity_Paid_Up                       1
dtype: int64
```

In [17]:

```
round(((df.drop('default', axis=1)==0).sum())*100/df.shape[0], 2).sort_values(ascending=Fal
```

Out[17]:

Capital_expenses_in_forex	80.65
ROG_Revenue_earnings_in_forex_perc	63.27
Revenue_earnings_in_forex	62.19
ROG_Revenue_expenses_in_forex_perc	54.96
Revenue_expenses_in_forex	53.23
ROG_Market_Capitalisation_perc	39.65
Market_Capitalisation	34.83
Inventory_Velocity_Days	31.32
Inventory_Ratio_Latest	30.67
Interest_Cover_Ratio_Latest	27.91
ROG_Gross_Block_perc	26.21
Selling_Cost	26.16
Total_Debt	23.54
Debtors_Ratio_Latest	20.78
Debtors_Velocity_Days	20.33
Creditors_Velocity_Days	19.63
Other_Income	19.24
Cash_Flow_From_Financing_Activities	18.29
Value_of_Output_to_Gross_Block	17.04
Fixed_Assets_Ratio_Latest	16.93
Cash_Flow_From_Investing_Activities	14.22
Total_Asset_Turnover_Ratio_Latest	13.75
Value_of_Output_to_Total_Assets	13.52
ROG_Gross_Sales_perc	12.69
ROG_Net_Sales_perc	12.69
APATM_perc_Latest	11.63
CPM_perc_Latest	11.18
PBITM_perc_Latest	10.96
PBDTM_perc_Latest	10.93
Gross_Block	10.88
PBIDTM_perc_Latest	10.88
ROG_Cost_of_Production_perc	10.26
ROG_PAT_perc	9.68
ROG_PBT_perc	9.09
ROG_PBIT_perc	8.78
ROG_PBDT_perc	8.56
ROG_CP_perc	8.56
Net_Sales	8.53
Gross_Sales	8.51
Value_Of_Output	8.09
ROG_PBIDT_perc	8.09
ROG_Net_Worth_perc	5.61
ROG_Capital_Employed_perc	5.30
Cash_Flow_From_Operating_Activities	5.02
ROG_Total_Assets_perc	4.82
Cost_of_Production	4.43
CEPS_annualised_Unit_Curr	3.88
Adjusted_PAT	3.46
PAT	3.37
PBT	3.32
CP	3.10
PBIT	3.04
PBDT	3.01
PBIDT	2.84
Current_Liabilities_and_Provisions	2.73

```
Current_Ratio_Latest                2.43
Net_Working_Capital                 1.34
Current_Assets                      0.75
Capital_Employed                    0.14
Total_Assets_to_Liabilities         0.08
Networth                            0.08
Book_Value_Adj_Unit_Curr            0.03
Book_Value_Unit_Curr                0.03
Equity_Paid_Up                      0.03
dtype: float64
```

- There are many variables with a lot of zero values
- They may be correct values at times, but also seems like a missing data replaced with zero
- Let us for this analysis, decide to ignore and drop all variables with number of zeros greater than 30%
- We drop the top 9 variables from above

Dropping Variables with zeros greater than 30%

In [18]:

```
df.drop(['Capital_expenses_in_forex', 'ROG_Revenue_earnings_in_forex_perc', 'Revenue_earnin
df.shape
```

Out[18]:

```
(3586, 56)
```

In [19]:

```
df.columns
```

Out[19]:

```
Index(['Equity_Paid_Up', 'Networth', 'Capital_Employed', 'Total_Debt', 'Gros
s_Sales', 'Net_Sales', 'Other_Income', 'Value_Of_Output', 'Cost_of_Productio
n', 'Selling_Cost', 'PBIDT', 'PBDT', 'PBIT', 'PBT', 'PAT', 'Adjusted_PAT',
'CP', 'Book_Value_Unit_Curr', 'CEPS_annualised_Unit_Curr', 'Cash_Flow_From_O
perating_Activities', 'Cash_Flow_From_Investing_Activities', 'Cash_Flow_From
_Financing_Activities', 'ROG_Net_Worth_perc', 'ROG_Capital_Employed_perc',
'ROG_Gross_Block_perc', 'ROG_Gross_Sales_perc', 'ROG_Net_Sales_perc', 'ROG_C
ost_of_Production_perc', 'ROG_Total_Assets_perc', 'ROG_PBIDT_perc', 'ROG_PBD
T_perc', 'ROG_PBIT_perc', 'ROG_PBT_perc', 'ROG_PAT_perc', 'ROG_CP_perc', 'Cu
rrent_Ratio_Latest', 'Fixed_Assets_Ratio_Latest', 'Debtors_Ratio_Latest', 'T
otal_Asset_Turnover_Ratio_Latest', 'Interest_Cover_Ratio_Latest', 'PBIDTM_pe
rc_Latest', 'PBITM_perc_Latest', 'PBDTM_perc_Latest', 'CPM_perc_Latest', 'AP
ATM_perc_Latest', 'Debtors_Velocity_Days', 'Creditors_Velocity_Days',
       'Value_of_Output_to_Total_Assets', 'Value_of_Output_to_Gross_Block',
'Current_Liabilities_and_Provisions', 'Total_Assets_to_Liabilities', 'Gross_
Block', 'Net_Working_Capital', 'Current_Assets', 'Book_Value_Adj_Unit_Curr',
'default'],
      dtype='object')
```

Converting other Zeros to Missing Values for Treatment

In [20]:

```
df1 = df.drop('default', axis=1)
df1[df1==0] = np.nan
(df1==0).sum()
(df1==0).shape
df1.shape
```

Out[20]:

```
Equity_Paid_Up                          0
Networth                                0
Capital_Employed                        0
Total_Debt                              0
Gross_Sales                             0
Net_Sales                               0
Other_Income                            0
Value_Of_Output                         0
Cost_of_Production                      0
Selling_Cost                            0
PBIDT                                   0
PBDT                                    0
PBIT                                    0
PBT                                     0
PAT                                     0
Adjusted_PAT                            0
CP                                      0
Book_Value_Unit_Curr                    0
CEPS_annualised_Unit_Curr               0
Cash_Flow_From_Operating_Activities     0
Cash_Flow_From_Investing_Activities     0
Cash_Flow_From_Financing_Activities     0
ROG_Net_Worth_perc                      0
ROG_Capital_Employed_perc               0
ROG_Gross_Block_perc                    0
ROG_Gross_Sales_perc                    0
ROG_Net_Sales_perc                      0
ROG_Cost_of_Production_perc             0
ROG_Total_Assets_perc                   0
ROG_PBIDT_perc                          0
ROG_PBDT_perc                           0
ROG_PBIT_perc                           0
ROG_PBT_perc                            0
ROG_PAT_perc                            0
ROG_CP_perc                             0
Current_Ratio_Latest                    0
Fixed_Assets_Ratio_Latest               0
Debtors_Ratio_Latest                    0
Total_Asset_Turnover_Ratio_Latest       0
Interest_Cover_Ratio_Latest             0
PBIDTM_perc_Latest                      0
PBITM_perc_Latest                       0
PBDTM_perc_Latest                       0
CPM_perc_Latest                         0
APATM_perc_Latest                       0
Debtors_Velocity_Days                   0
Creditors_Velocity_Days                 0
Value_of_Output_to_Total_Assets         0
Value_of_Output_to_Gross_Block          0
Current_Liabilities_and_Provisions      0
Total_Assets_to_Liabilities             0
```

```
Gross_Block                    0
Net_Working_Capital            0
Current_Assets                 0
Book_Value_Adj_Unit_Curr       0
dtype: int64
```

Out[20]:

```
(3586, 55)
```

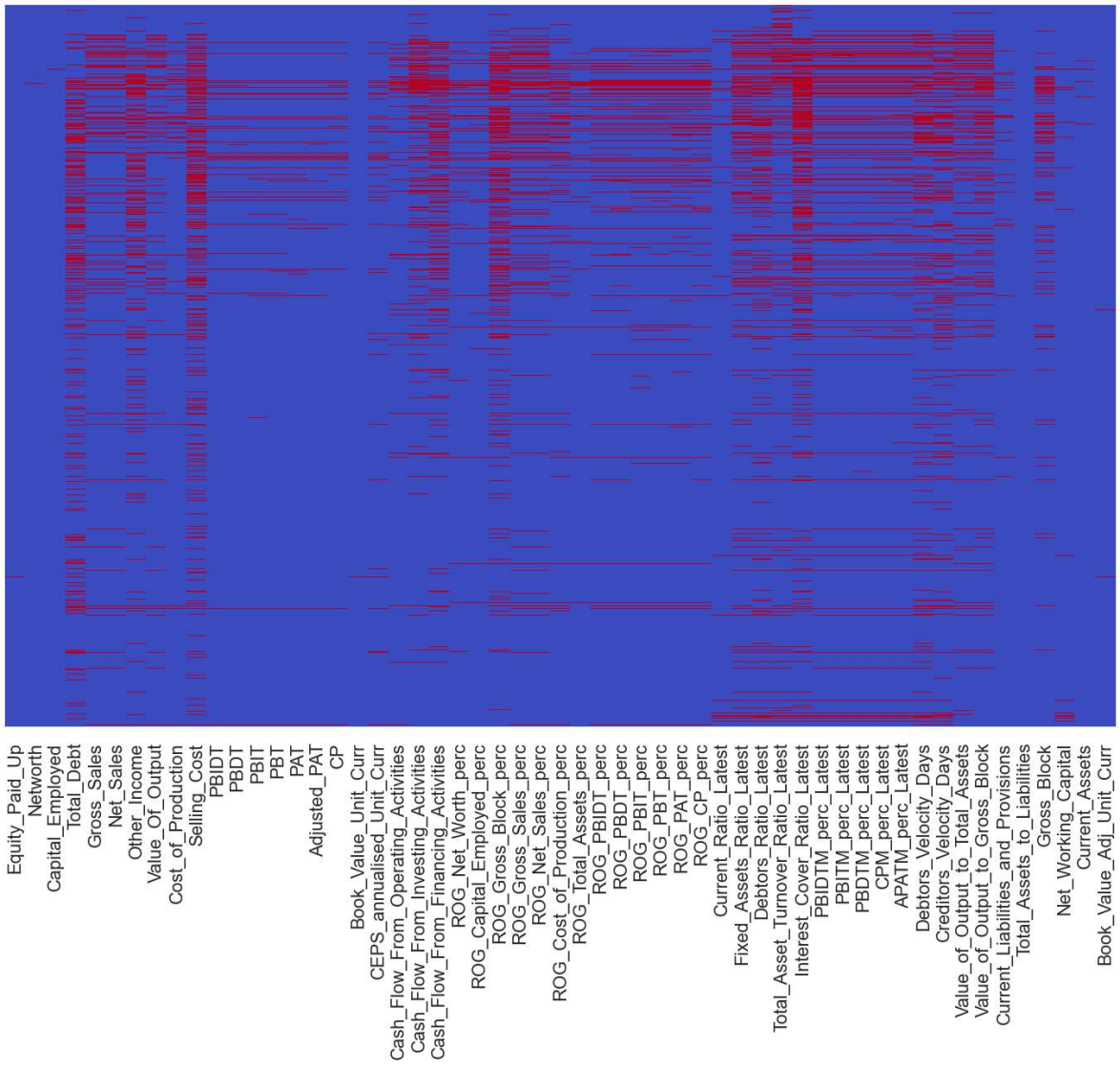
Out[20]:

```
(3586, 55)
```

Visualising Missing Values

In [21]:

```
plt.figure(figsize = (12,8))
sns.heatmap(df1.isnull(), cbar = False, cmap = 'coolwarm', yticklabels = False)
plt.show();
```



Checking Missing Values by Rows

- We check for rows with more than 5 missing values
- Out of these rows, we note that 164 out of 387 defaults exist
- Hence, we ll not drop these rows
- But, also conclude that more the missing or zero values, higher is the probability of default

In [22]:

```
df1['default'] = df['default']
df_rows = df1[df1.isna().sum(axis=1)>5]
df_rows.shape
```

Out[22]:

```
(1004, 56)
```

In [23]:

```
df1['default'].value_counts()
df_rows['default'].value_counts()
```

Out[23]:

```
0    3199
1     387
Name: default, dtype: int64
```

Out[23]:

```
0    840
1    164
Name: default, dtype: int64
```

Split Target and Predictor Variables

In [24]:

```
df_x = df1.drop('default', axis=1)
df_y = df1['default']
```

Impute Null Values using KNN Imputer (n_neighbors=10)

In [25]:

```
df1.isna().sum()
```

Out[25]:

```
Equity_Paid_Up                          1
Networth                                3
Capital_Employed                        5
Total_Debt                            844
Gross_Sales                           305
Net_Sales                             306
Other_Income                          690
Value_Of_Output                       290
Cost_of_Production                    159
Selling_Cost                          938
PBIDT                                 102
PBDT                                  108
PBIT                                  109
PBT                                   119
PAT                                   121
Adjusted_PAT                          124
CP                                    111
Book_Value_Unit_Curr                    1
CEPS_annualised_Unit_Curr             139
Cash_Flow_From_Operating_Activities   180
Cash_Flow_From_Investing_Activities   510
Cash_Flow_From_Financing_Activities   656
ROG_Net_Worth_perc                    201
ROG_Capital_Employed_perc             190
ROG_Gross_Block_perc                  940
ROG_Gross_Sales_perc                  455
ROG_Net_Sales_perc                    455
ROG_Cost_of_Production_perc           368
ROG_Total_Assets_perc                 173
ROG_PBIDT_perc                        290
ROG_PBDT_perc                         307
ROG_PBIT_perc                         315
ROG_PBT_perc                          326
ROG_PAT_perc                          347
ROG_CP_perc                           307
Current_Ratio_Latest                   88
Fixed_Assets_Ratio_Latest             608
Debtors_Ratio_Latest                  746
Total_Asset_Turnover_Ratio_Latest     494
Interest_Cover_Ratio_Latest          1002
PBIDTM_perc_Latest                    391
PBITM_perc_Latest                     394
PBDTM_perc_Latest                     393
CPM_perc_Latest                       402
APATM_perc_Latest                     418
Debtors_Velocity_Days                 729
Creditors_Velocity_Days               704
Value_of_Output_to_Total_Assets       485
Value_of_Output_to_Gross_Block        611
Current_Liabilities_and_Provisions     98
Total_Assets_to_Liabilities             3
Gross_Block                           390
Net_Working_Capital                    48
Current_Assets                         27
Book_Value_Adj_Unit_Curr                5
```

```
default                         0
dtype: int64
```

In [26]:

```
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=10)

df_imputed = pd.DataFrame(imputer.fit_transform(df1), columns = df1.columns)

df_imputed.isnull().sum()
```

Out[26]:

```
Equity_Paid_Up                          0
Networth                                0
Capital_Employed                        0
Total_Debt                              0
Gross_Sales                             0
Net_Sales                               0
Other_Income                            0
Value_Of_Output                         0
Cost_of_Production                      0
Selling_Cost                            0
PBIDT                                   0
PBDT                                    0
PBIT                                    0
PBT                                     0
PAT                                     0
Adjusted_PAT                            0
CP                                      0
Book_Value_Unit_Curr                    0
CEPS_annualised_Unit_Curr               0
Cash_Flow_From_Operating_Activities     0
Cash_Flow_From_Investing_Activities     0
Cash_Flow_From_Financing_Activities     0
ROG_Net_Worth_perc                      0
ROG_Capital_Employed_perc               0
ROG_Gross_Block_perc                    0
ROG_Gross_Sales_perc                    0
ROG_Net_Sales_perc                      0
ROG_Cost_of_Production_perc             0
ROG_Total_Assets_perc                   0
ROG_PBIDT_perc                          0
ROG_PBDT_perc                           0
ROG_PBIT_perc                           0
ROG_PBT_perc                            0
ROG_PAT_perc                            0
ROG_CP_perc                             0
Current_Ratio_Latest                    0
Fixed_Assets_Ratio_Latest               0
Debtors_Ratio_Latest                    0
Total_Asset_Turnover_Ratio_Latest       0
Interest_Cover_Ratio_Latest             0
PBIDTM_perc_Latest                      0
PBITM_perc_Latest                       0
PBDTM_perc_Latest                       0
CPM_perc_Latest                         0
APATM_perc_Latest                       0
Debtors_Velocity_Days                   0
Creditors_Velocity_Days                 0
Value_of_Output_to_Total_Assets         0
Value_of_Output_to_Gross_Block          0
```

```
Current_Liabilities_and_Provisions     0
Total_Assets_to_Liabilities            0
Gross_Block                            0
Net_Working_Capital                    0
Current_Assets                         0
Book_Value_Adj_Unit_Curr               0
default                                0
dtype: int64
```

Outlier Treatment using Z-Score method

$z - score = \frac{x - mean}{std.dev}$

We'll cap all values to 3 and -3 which lie beyond them

In [27]:

```
df_x = df_imputed.drop('default', axis=1)
df_y = df_imputed['default']
```

In [28]:

```
sc = StandardScaler()
scaled_x = pd.DataFrame(sc.fit_transform(df_x), columns=df_x.columns)
scaled_x.shape
scaled_x.head()
```

Out[28]:

```
(3586, 55)
```

Out[28]:

	Equity_Paid_Up	Networth	Capital_Employed	Total_Debt	Gross_Sales	Net_Sales	Other_Inc
0	0.457700	-1.876422	-0.141144	0.164945	-0.104687	-0.106625	-0.09
1	2.429785	-0.884247	0.061494	0.227253	0.166239	0.180775	-0.00
2	-0.011726	-0.034923	0.182233	0.207572	-0.069530	-0.069321	-0.09
3	0.121144	-0.311197	-0.016526	0.012361	0.021196	0.026919	0.40
4	-0.016273	-0.420533	0.069545	0.156697	-0.102851	-0.104667	-0.09

In [29]:

```
for column in df_x.columns:
    lr = -3
    ur = 3
    df_x[column]=np.where(df_x[column]>ur,ur,df_x[column])
    df_x[column]=np.where(df_x[column]<lr,lr,df_x[column])
```

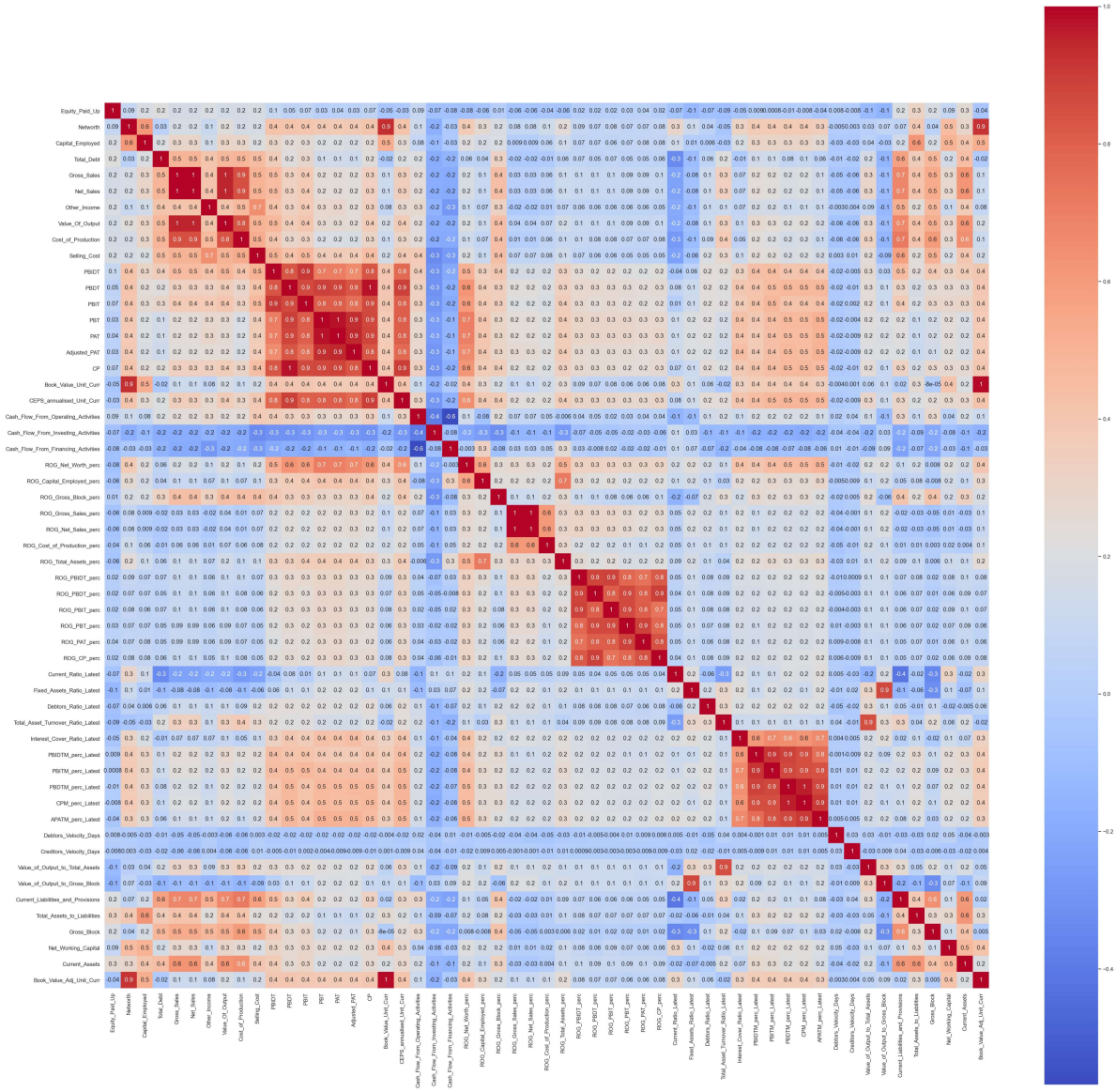
correlation Heatmap

In [30]:

```
#calculate column correlations and make a seaborn heatmap - Before standardisation

plt.figure(figsize=(40,40))
p=sns.heatmap(df_x.corr(), annot=True,cmap='coolwarm',square=True, fmt='.1g')
```

Out[30]:

```
<Figure size 2880x2880 with 0 Axes>
```



- We see a lot of red patches showing high correlation between variables
- This gives rise to issues of Multi-Collinearity
- Lets check for Multi-Collinearity using Variance Inflation Factor

Check Multi-Colinearity using Variance Inflation Factor

In [31]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)
```

In [32]:

```
X = df_x.copy()
f = calc_vif(X).sort_values(by = 'VIF', ascending = False)
f
```

Out[32]:

	variables	VIF
5	Net_Sales	91096.145862
4	Gross_Sales	90763.926316
17	Book_Value_Unit_Curr	201.523094
54	Book_Value_Adj_Unit_Curr	189.628506
45	Debtors_Velocity_Days	140.189690
46	Creditors_Velocity_Days	136.567955
25	ROG_Gross_Sales_perc	106.347196
26	ROG_Net_Sales_perc	106.298466
7	Value_Of_Output	73.148662
50	Total_Assets_to_Liabilities	65.545310
11	PBDT	56.628646
16	CP	55.954958
14	PAT	49.686883
13	PBT	47.601656
53	Current_Assets	28.482099
1	Networth	26.439996
8	Cost_of_Production	23.285694
43	CPM_perc_Latest	23.194378
42	PBDTM_perc_Latest	23.038187
36	Fixed_Assets_Ratio_Latest	22.532632
48	Value_of_Output_to_Gross_Block	20.932909
2	Capital_Employed	20.439226
0	Equity_Paid_Up	17.391032
49	Current_Liabilities_and_Provisions	15.362727
40	PBIDTM_perc_Latest	14.851391
37	Debtors_Ratio_Latest	14.557305
10	PBIDT	13.444311
12	PBIT	13.311206
41	PBITM_perc_Latest	13.231833
51	Gross_Block	12.183639
30	ROG_PBDT_perc	11.930882
3	Total_Debt	11.871350
15	Adjusted_PAT	11.389917

	variables	VIF
32	ROG_PBT_perc	10.644976
38	Total_Asset_Turnover_Ratio_Latest	10.376982
29	ROG_PBIDT_perc	9.735606
47	Value_of_Output_to_Total_Assets	9.402801
31	ROG_PBIT_perc	9.185175
44	APATM_perc_Latest	9.135073
18	CEPS_annualised_Unit_Curr	8.645571
34	ROG_CP_perc	8.072176
33	ROG_PAT_perc	7.704496
35	Current_Ratio_Latest	7.312974
9	Selling_Cost	4.590620
52	Net_Working_Capital	4.140082
6	Other_Income	4.085117
39	Interest_Cover_Ratio_Latest	3.215533
22	ROG_Net_Worth_perc	3.177454
23	ROG_Capital_Employed_perc	2.797857
19	Cash_Flow_From_Operating_Activities	2.482344
28	ROG_Total_Assets_perc	2.466330
21	Cash_Flow_From_Financing_Activities	2.171347
20	Cash_Flow_From_Investing_Activities	1.845301
27	ROG_Cost_of_Production_perc	1.832412
24	ROG_Gross_Block_perc	1.614971

In [33]:

```
X.shape
X.columns
```

Out[33]:

```
(3586, 55)
```

Out[33]:

```
Index(['Equity_Paid_Up', 'Networth', 'Capital_Employed', 'Total_Debt', 'Gros
s_Sales', 'Net_Sales', 'Other_Income', 'Value_Of_Output', 'Cost_of_Productio
n', 'Selling_Cost', 'PBIDT', 'PBDT', 'PBIT', 'PBT', 'PAT', 'Adjusted_PAT',
'CP', 'Book_Value_Unit_Curr', 'CEPS_annualised_Unit_Curr', 'Cash_Flow_From_O
perating_Activities', 'Cash_Flow_From_Investing_Activities', 'Cash_Flow_From
_Financing_Activities', 'ROG_Net_Worth_perc', 'ROG_Capital_Employed_perc',
'ROG_Gross_Block_perc', 'ROG_Gross_Sales_perc', 'ROG_Net_Sales_perc', 'ROG_C
ost_of_Production_perc', 'ROG_Total_Assets_perc', 'ROG_PBIDT_perc', 'ROG_PBD
T_perc', 'ROG_PBIT_perc', 'ROG_PBT_perc', 'ROG_PAT_perc', 'ROG_CP_perc', 'Cu
rrent_Ratio_Latest', 'Fixed_Assets_Ratio_Latest', 'Debtors_Ratio_Latest', 'T
otal_Asset_Turnover_Ratio_Latest', 'Interest_Cover_Ratio_Latest', 'PBIDTM_pe
rc_Latest', 'PBITM_perc_Latest', 'PBDTM_perc_Latest', 'CPM_perc_Latest', 'AP
ATM_perc_Latest', 'Debtors_Velocity_Days', 'Creditors_Velocity_Days',
       'Value_of_Output_to_Total_Assets', 'Value_of_Output_to_Gross_Block',
'Current_Liabilities_and_Provisions', 'Total_Assets_to_Liabilities', 'Gross_
Block', 'Net_Working_Capital', 'Current_Assets', 'Book_Value_Adj_Unit_Cur
r'],
      dtype='object')
```

Dropping Variables recursively with VIF greater than 5

- We check VIF of all predictor variables
- We arrange these variables in decreasing order of VIF
- We check if the top variable has VIF > 5
- If yes, then we drop ONLY this Top Variable and check for VIF again
- We run this in a loop and drop variables one by one till we get max VIF < 5

In [34]:

```
import time
start = time.time()

for i in range(55):
    v = calc_vif(X).sort_values(by = 'VIF', ascending = False)
    if (v.iloc[0,1] > 5):
        X = X.drop(columns=v.iloc[0,0])

end = time.time()
print('Time taken =', end-start)
```

```
Time taken = 14.505756139755249
```

Check VIF again

In [35]:

```
calc_vif(X).sort_values(by = 'VIF', ascending = False)
```

Out[35]:

	variables	VIF
20	Value_of_Output_to_Gross_Block	4.841672
16	Current_Ratio_Latest	4.496532
1	Selling_Cost	4.155857
22	Book_Value_Adj_Unit_Curr	4.081417
2	PBIDT	3.996385
13	ROG_PBIDT_perc	3.898672
15	ROG_CP_perc	3.649697
0	Other_Income	3.464304
14	ROG_PBT_perc	3.158321
21	Net_Working_Capital	2.973412
17	Interest_Cover_Ratio_Latest	2.920517
19	Value_of_Output_to_Total_Assets	2.862014
7	ROG_Net_Worth_perc	2.803484
3	Adjusted_PAT	2.785549
8	ROG_Capital_Employed_perc	2.754500
4	Cash_Flow_From_Operating_Activities	2.427376
12	ROG_Total_Assets_perc	2.402298
18	APATM_perc_Latest	2.274563
6	Cash_Flow_From_Financing_Activities	2.126515
10	ROG_Net_Sales_perc	1.959210
5	Cash_Flow_From_Investing_Activities	1.813096
11	ROG_Cost_of_Production_perc	1.806302
9	ROG_Gross_Block_perc	1.455157

In [36]:

```
X.shape
X.head()
```

Out[36]:

(3586, 23)

Out[36]:

	Other_Income	Selling_Cost	PBIDT	Adjusted_PAT	Cash_Flow_From_Operating_Activities	Ca
0	3.0	0.492	-3.0	-3.0	-3.0	
1	3.0	3.000	3.0	-3.0	3.0	
2	3.0	3.000	-3.0	-3.0	-3.0	
3	3.0	3.000	-3.0	-3.0	3.0	
4	3.0	1.970	-3.0	-3.0	-3.0	

- After dropping variables one by one with high VIF (>5)
- We are left with final 23 predictor variables for modelling

Split the whole data into Train and Test 67-33

In [37]:

```
X_train, X_test, y_train, y_test = train_test_split(X, df_y,
                                                    test_size = 0.33, random_state=42)
```

In [38]:

```
X.shape
df_y.shape
```

Out[38]:

(3586, 23)

Out[38]:

(3586,)

MODEL 1

In [39]:

```
import statsmodels.api as SM

# f1 = 'default ~ Equity_Paid_Up + Total_Debt + Other_Income + Selling_Cost + Adjusted_PAT
model_1 = SM.Logit(y_train, X_train).fit()
print(model_1.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.065853
         Iterations 10
                           Logit Regression Results
==============================================================================
=====
Dep. Variable:                 default   No. Observations:
2402
Model:                           Logit   Df Residuals:
2379
Method:                            MLE   Df Model:
22
Date:                Sat, 08 Jan 2022   Pseudo R-squ.:
0.8001
Time:                        20:55:26   Log-Likelihood:                -1
58.18
converged:                       True   LL-Null:                       -7
91.34
Covariance Type:            nonrobust   LLR p-value:                3.040
e-254
==============================================================================
==============================
                                   coef    std err          z
P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
------------------------------
Other_Income                      -0.0754      0.156     -0.484
0.629      -0.381       0.230
Selling_Cost                       0.4824      0.192      2.512
0.012       0.106       0.859
PBIDT                             -0.2433      0.098     -2.476
0.013      -0.436      -0.051
Adjusted_PAT                       0.0465      0.122      0.380
0.704      -0.194       0.287
Cash_Flow_From_Operating_Activities    -0.0405      0.113     -0.359
0.720      -0.262       0.181
Cash_Flow_From_Investing_Activities     0.0049      0.106      0.046
0.963      -0.203       0.213
Cash_Flow_From_Financing_Activities     0.0085      0.114      0.074
0.941      -0.216       0.233
ROG_Net_Worth_perc                -0.2886      0.091     -3.184
0.001      -0.466      -0.111
ROG_Capital_Employed_perc          0.3880      0.090      4.288
0.000       0.211       0.565
ROG_Gross_Block_perc               0.0902      0.075      1.203
0.229      -0.057       0.237
ROG_Net_Sales_perc                -0.0289      0.067     -0.430
0.667      -0.161       0.103
ROG_Cost_of_Production_perc       -0.0440      0.067     -0.657
0.511      -0.175       0.087
ROG_Total_Assets_perc             -0.1804      0.087     -2.064
```

```
0.039      -0.352      -0.009
ROG_PBIDT_perc                          0.1435      0.103      1.395
0.163      -0.058       0.345
ROG_PBT_perc                           -0.0218      0.095     -0.228
0.820      -0.209       0.165
ROG_CP_perc                            -0.0974      0.105     -0.929
0.353      -0.303       0.108
Current_Ratio_Latest                   -0.5365      0.169     -3.166
0.002      -0.869      -0.204
Interest_Cover_Ratio_Latest            -0.2420      0.088     -2.749
0.006      -0.415      -0.069
APATM_perc_Latest                      -0.2091      0.085     -2.462
0.014      -0.376      -0.043
Value_of_Output_to_Total_Assets        -0.4137      0.266     -1.554
0.120      -0.936       0.108
Value_of_Output_to_Gross_Block         -0.0142      0.140     -0.101
0.919      -0.289       0.261
Net_Working_Capital                     0.0355      0.089      0.397
0.692      -0.140       0.211
Book_Value_Adj_Unit_Curr               -1.4219      0.104    -13.615
0.000      -1.627      -1.217
==============================================================================
==============================
```

Metrics of Model 1

In [40]:

```
# MODEL 1 TRAIN

y_prob_pred_train = model_1.predict(X_train)

y_class_pred=[]
for i in range(0,len(y_prob_pred_train)):
    if np.array(y_prob_pred_train)[i]>0.5:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_train,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TRAIN MODEL 1')

print(metrics.classification_report(y_train,y_class_pred))
```

Out[40]:

```
<AxesSubplot:>
```

Out[40]:

```
Text(0.5, 12.5, 'Predicted')
```

Out[40]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[40]:

```
Text(0.5, 1.0, 'TRAIN MODEL 1')
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99      2157
         1.0       0.92      0.85      0.89       245

    accuracy                           0.98      2402
   macro avg       0.95      0.92      0.94      2402
weighted avg       0.98      0.98      0.98      2402
```

MODEL 2

In [41]:

```
import time
start = time.time()

for i in range(25):
    model_2 = SM.Logit(y_train, X_train).fit()
    v = pd.DataFrame(model_2.pvalues.sort_values(ascending=False), columns=['pvalue']).rese
    if (v.iloc[0,1] > 0.05):
        X_train = X_train.drop(columns=v.iloc[0,0])

end = time.time()
print('Time taken =', end-start)
```

```
Optimization terminated successfully.
         Current function value: 0.065853
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065854
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065854
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065857
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065867
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065899
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065927
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065967
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066029
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066105
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066357
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066616
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066698
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.067105
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
```

```
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Time taken = 0.19743704795837402
```

We check p-values of all predictor variables

We arrange these variables in decreasing order of p-values

We check if the top predictor variable has p-value > 0.05

If yes, then it means that this variable is not significant in predicting default at 95% Confidence

Hence, we drop ONLY this Top Variable and check for p-values again

We run this in a loop and drop variables one by one till we get all p-values < 0.05

In [42]:

```
print(model_2.summary())
```

```
                           Logit Regression Results
==============================================================================
==
Dep. Variable:                 default   No. Observations:                24
02
Model:                           Logit   Df Residuals:                    23
93
Method:                            MLE   Df Model:
8
Date:                Sat, 08 Jan 2022   Pseudo R-squ.:                  0.79
38
Time:                        20:55:27   Log-Likelihood:                -163.
20
converged:                       True   LL-Null:                       -791.
34
Covariance Type:            nonrobust   LLR p-value:                6.562e-2
66
==============================================================================
====================
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
--------------------
Selling_Cost                     0.3771      0.133      2.845      0.004
0.117       0.637
PBIDT                           -0.2783      0.082     -3.389      0.001
-0.439      -0.117
ROG_Net_Worth_perc              -0.2545      0.074     -3.424      0.001
-0.400      -0.109
ROG_Capital_Employed_perc        0.3865      0.084      4.595      0.000
0.222       0.551
ROG_Total_Assets_perc           -0.1607      0.081     -1.991      0.047
-0.319      -0.002
Current_Ratio_Latest            -0.6789      0.123     -5.537      0.000
-0.919      -0.439
Interest_Cover_Ratio_Latest     -0.2750      0.084     -3.278      0.001
-0.439      -0.111
APATM_perc_Latest               -0.1655      0.078     -2.110      0.035
-0.319      -0.012
Book_Value_Adj_Unit_Curr        -1.3490      0.092    -14.718      0.000
-1.529      -1.169
==============================================================================
====================
```

In [43]:

```
X_train.shape
```

Out[43]:

```
(2402, 9)
```

In [44]:

```
y_train.shape
```

Out[44]:

```
(2402,)
```

In [45]:

```
X_test = X_test[X_train.columns]
```

In [46]:

```
X_test.shape
```

Out[46]:

```
(1184, 9)
```

In [47]:

```
X_test.columns
```

Out[47]:

```
Index(['Selling_Cost', 'PBIDT', 'ROG_Net_Worth_perc', 'ROG_Capital_Employed_
perc', 'ROG_Total_Assets_perc', 'Current_Ratio_Latest', 'Interest_Cover_Rati
o_Latest', 'APATM_perc_Latest', 'Book_Value_Adj_Unit_Curr'], dtype='object')
```

In [48]:

```
y_prob_pred_test = model_2.predict(X_test)

y_class_pred=[]
for i in range(0,len(y_prob_pred_test)):
    if np.array(y_prob_pred_test)[i]>0.5:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_test,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TEST MODEL 2')

print(metrics.classification_report(y_test,y_class_pred))
```

Out[48]:

```
<AxesSubplot:>
```

Out[48]:

```
Text(0.5, 12.5, 'Predicted')
```

Out[48]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[48]:

```
Text(0.5, 1.0, 'TEST MODEL 2')
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98      1042
         1.0       0.89      0.88      0.88       142

    accuracy                           0.97      1184
   macro avg       0.94      0.93      0.93      1184
weighted avg       0.97      0.97      0.97      1184
```

Logistic Regression using StatsModel

MODEL 3

In [49]:

```
import statsmodels.api as SM

# f1 = 'default ~ Equity_Paid_Up + Total_Debt + Other_Income + Selling_Cost + Adjusted_PAT
model_3 = SM.Logit(y_train, X_train).fit()
print(model_3.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
                           Logit Regression Results
==============================================================================
==
Dep. Variable:                 default   No. Observations:                  24
02
Model:                           Logit   Df Residuals:                      23
93
Method:                            MLE   Df Model:
8
Date:                Sat, 08 Jan 2022   Pseudo R-squ.:                   0.79
38
Time:                        20:55:27   Log-Likelihood:                -163.
20
converged:                       True   LL-Null:                       -791.
34
Covariance Type:            nonrobust   LLR p-value:                6.562e-2
66
==============================================================================
==================
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
------------------
Selling_Cost                    0.3771      0.133      2.845      0.004
0.117       0.637
PBIDT                          -0.2783      0.082     -3.389      0.001
-0.439      -0.117
ROG_Net_Worth_perc             -0.2545      0.074     -3.424      0.001
-0.400      -0.109
ROG_Capital_Employed_perc       0.3865      0.084      4.595      0.000
0.222       0.551
ROG_Total_Assets_perc          -0.1607      0.081     -1.991      0.047
-0.319      -0.002
Current_Ratio_Latest           -0.6789      0.123     -5.537      0.000
-0.919      -0.439
Interest_Cover_Ratio_Latest    -0.2750      0.084     -3.278      0.001
-0.439      -0.111
APATM_perc_Latest              -0.1655      0.078     -2.110      0.035
-0.319      -0.012
Book_Value_Adj_Unit_Curr       -1.3490      0.092    -14.718      0.000
-1.529      -1.169
==============================================================================
==================
```

Metrics of Model 3

In [50]:

```
## MODEL 3 TRAIN

y_prob_pred_train = model_3.predict(X_train)

y_class_pred=[]
for i in range(0,len(y_prob_pred_train)):
    if np.array(y_prob_pred_train)[i]>0.5:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_train,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TRAIN MODEL 3')

print(metrics.classification_report(y_train,y_class_pred))
```

Out[50]:

```
<AxesSubplot:>
```

Out[50]:

```
Text(0.5, 12.5, 'Predicted')
```

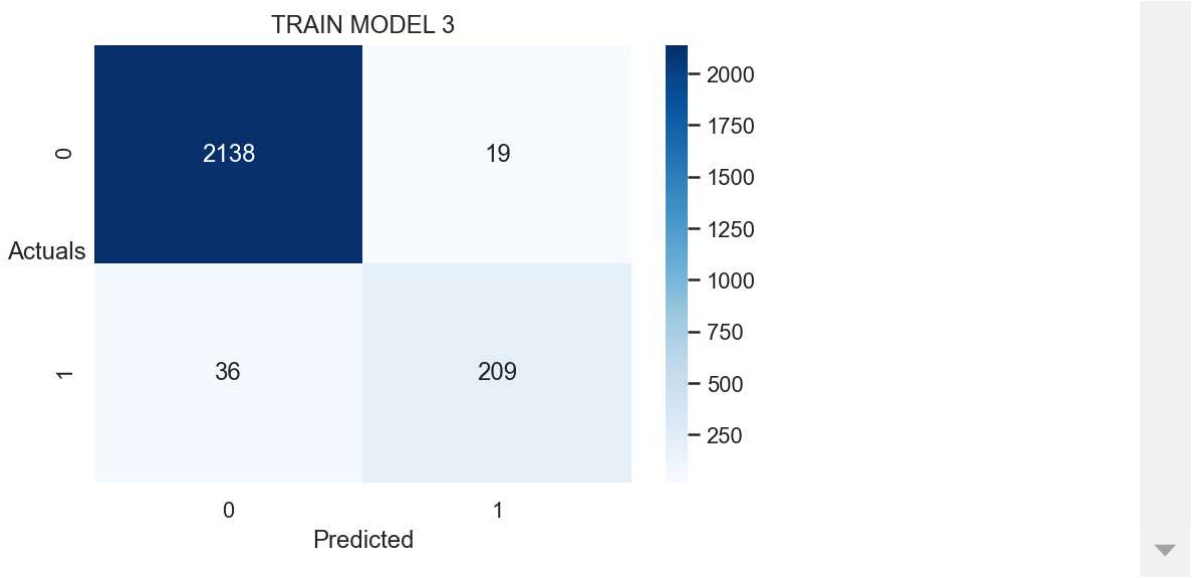
Out[50]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[50]:

```
Text(0.5, 1.0, 'TRAIN MODEL 3')
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99      2157
         1.0       0.92      0.85      0.88       245

    accuracy                           0.98      2402
   macro avg       0.95      0.92      0.94      2402
weighted avg       0.98      0.98      0.98      2402
```


TRAIN MODEL 3
2138
19
36
209
0
1
Actuals
0
1
Predicted
2000
1750
1500
1250
1000
750
500
250

In [51]:

```
## MODEL 3 TEST

y_prob_pred_test = model_3.predict(X_test)

y_class_pred=[]
for i in range(0,len(y_prob_pred_test)):
    if np.array(y_prob_pred_test)[i]>0.5:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_test,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TEST MODEL 1')

print(metrics.classification_report(y_test,y_class_pred))
```

Out[51]:

```
<AxesSubplot:>
```

Out[51]:

```
Text(0.5, 12.5, 'Predicted')
```

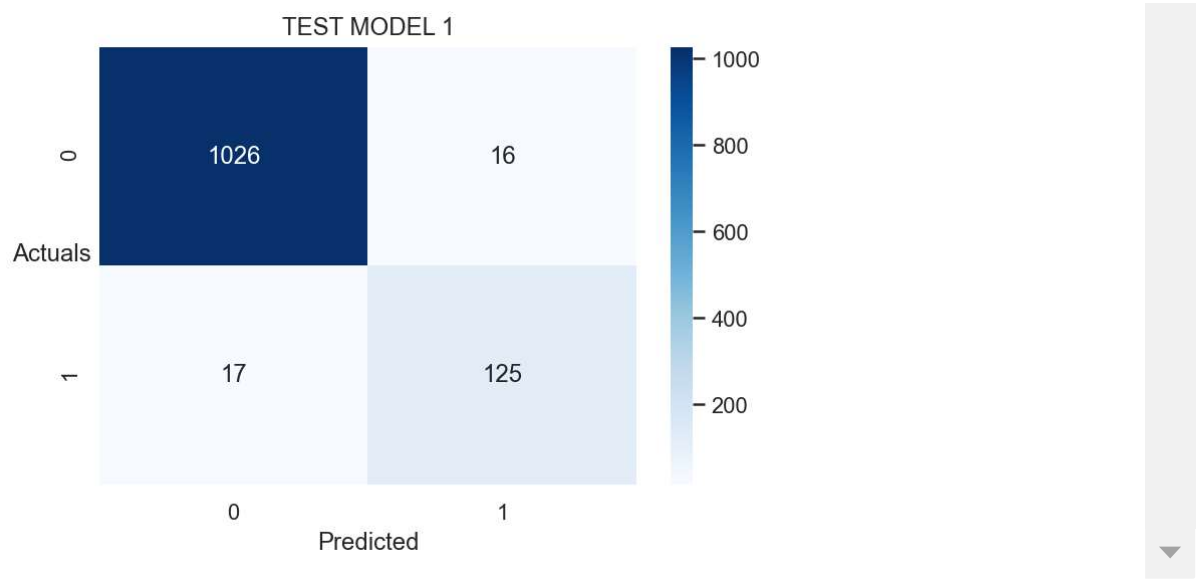
Out[51]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[51]:

```
Text(0.5, 1.0, 'TEST MODEL 1')
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98      1042
         1.0       0.89      0.88      0.88       142

    accuracy                           0.97      1184
   macro avg       0.94      0.93      0.93      1184
weighted avg       0.97      0.97      0.97      1184
```



In [52]:

```
X_test.shape
```

Out[52]:

```
(1184, 9)
```

MODEL 4

In [53]:

```
X_train, X_test, y_train, y_test = train_test_split(X, df_y,
                                                    test_size = 0.33, random_state=42)
```

In [54]:

```
v = pd.DataFrame(model_3.pvalues.sort_values(ascending=False), columns=['pvalue']).reset_in
v
```

Out[54]:

	index	pvalue
0	ROG_Total_Assets_perc	4.651067e-02
1	APATM_perc_Latest	3.481715e-02
2	Selling_Cost	4.446268e-03
3	Interest_Cover_Ratio_Latest	1.044771e-03
4	PBIDT	7.023879e-04
5	ROG_Net_Worth_perc	6.159490e-04
6	ROG_Capital_Employed_perc	4.331451e-06
7	Current_Ratio_Latest	3.068876e-08
8	Book_Value_Adj_Unit_Curr	4.974105e-49

Dropping Variables recursively with p-values greater than 0.05

- We check p-values of all predictor variables
- We arrange these variables in decreasing order of p-values
- We check if the top predictor variable has p-value > 0.05
- If yes, then it means that this variable is not significant in predicting default at 95% Confidence
- Hence, we drop ONLY this Top Variable and check for p-values again
- We run this in a loop and drop variables one by one till we get all p-values < 0.05

In [55]:

```
import time
start = time.time()

for i in range(23):
    model_4 = SM.Logit(y_train, X_train).fit()
    v = pd.DataFrame(model_4.pvalues.sort_values(ascending=False), columns=['pvalue']).rese
    if (v.iloc[0,1] > 0.05):
        X_train = X_train.drop(columns=v.iloc[0,0])

end = time.time()
print('Time taken =', end-start)
```

```
Optimization terminated successfully.
         Current function value: 0.065853
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065854
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065854
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065857
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065867
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065899
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065927
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.065967
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066029
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066105
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066357
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066616
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.066698
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.067105
         Iterations 10
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
```

```
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.067941
         Iterations 9
Time taken = 0.2717146873474121
```

Check p-values again

In [56]:

```
print(model_4.summary())
```

```
                           Logit Regression Results
==============================================================================
==
Dep. Variable:                default   No. Observations:                24
02
Model:                          Logit   Df Residuals:                    23
93
Method:                           MLE   Df Model:
8
Date:                Sat, 08 Jan 2022   Pseudo R-squ.:                  0.79
38
Time:                        20:55:28   Log-Likelihood:                -163.
20
converged:                       True   LL-Null:                       -791.
34
Covariance Type:            nonrobust   LLR p-value:                6.562e-2
66
==============================================================================
==================
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
------------------
Selling_Cost                     0.3771      0.133      2.845      0.004
0.117       0.637
PBIDT                           -0.2783      0.082     -3.389      0.001
-0.439      -0.117
ROG_Net_Worth_perc              -0.2545      0.074     -3.424      0.001
-0.400      -0.109
ROG_Capital_Employed_perc        0.3865      0.084      4.595      0.000
0.222       0.551
ROG_Total_Assets_perc           -0.1607      0.081     -1.991      0.047
-0.319      -0.002
Current_Ratio_Latest            -0.6789      0.123     -5.537      0.000
-0.919      -0.439
Interest_Cover_Ratio_Latest     -0.2750      0.084     -3.278      0.001
-0.439      -0.111
APATM_perc_Latest               -0.1655      0.078     -2.110      0.035
-0.319      -0.012
Book_Value_Adj_Unit_Curr        -1.3490      0.092    -14.718      0.000
-1.529      -1.169
==============================================================================
==================
```

- We note that only those variables are included in the model which p-values less than 0.05
- All other variables with p-values > 0.05 are recursively dropped one by one

Re-setting the Test set with only the required 9 variables chosen

In [57]:

```
X_train.shape
```

Out[57]:

```
(2402, 9)
```

In [58]:

```
y_train.shape
```

Out[58]:

```
(2402,)
```

In [59]:

```
X_test = X_test[X_train.columns]
```

In [60]:

```
X_test.shape
```

Out[60]:

```
(1184, 9)
```

In [61]:

```
X_test.columns
```

Out[61]:

```
Index(['Selling_Cost', 'PBIDT', 'ROG_Net_Worth_perc', 'ROG_Capital_Employed_
perc', 'ROG_Total_Assets_perc', 'Current_Ratio_Latest', 'Interest_Cover_Rati
o_Latest', 'APATM_perc_Latest', 'Book_Value_Adj_Unit_Curr'], dtype='object')
```

Metrics of Model 4

In [62]:

```
## MODEL 4 TRAIN

y_prob_pred_train = model_4.predict(X_train)

y_class_pred=[]
for i in range(0,len(y_prob_pred_train)):
    if np.array(y_prob_pred_train)[i]>0.5:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_train,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TRAIN MODEL 4')

print(metrics.classification_report(y_train,y_class_pred))
```

Out[62]:

```
<AxesSubplot:>
```

Out[62]:

```
Text(0.5, 12.5, 'Predicted')
```

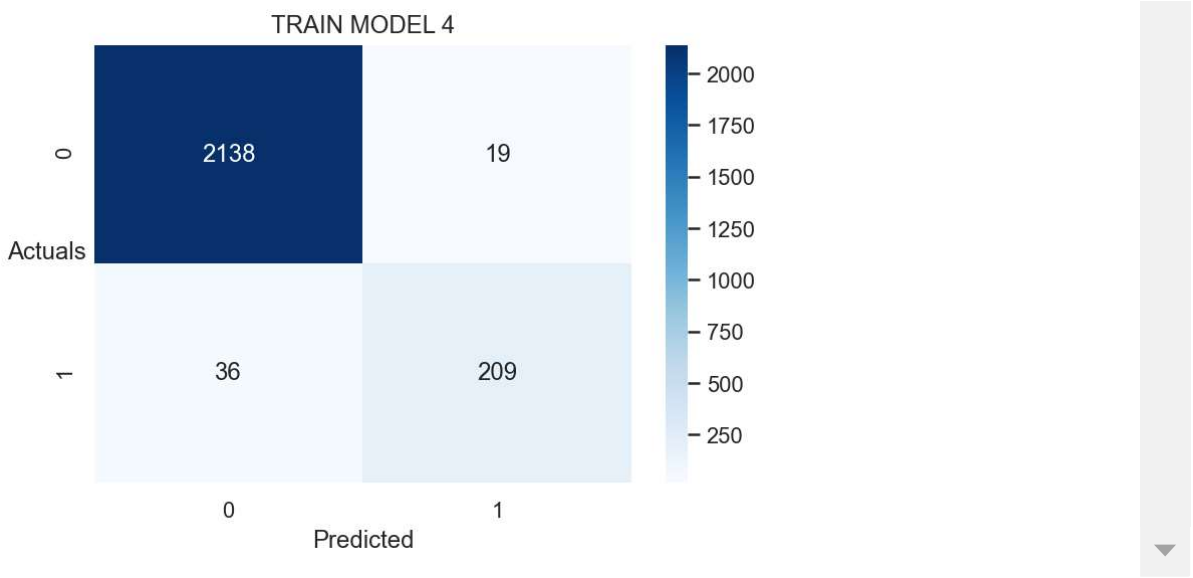
Out[62]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[62]:

```
Text(0.5, 1.0, 'TRAIN MODEL 4')
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99      2157
         1.0       0.92      0.85      0.88       245

    accuracy                           0.98      2402
   macro avg       0.95      0.92      0.94      2402
weighted avg       0.98      0.98      0.98      2402
```


TRAIN MODEL 4
2138
19
36
209
0
1
Actuals
0
1
Predicted
2000
1750
1500
1250
1000
750
500
250

In [63]:

```
## MODEL 4 TEST

y_prob_pred_test = model_4.predict(X_test)

y_class_pred=[]
for i in range(0,len(y_prob_pred_test)):
    if np.array(y_prob_pred_test)[i]>0.5:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_test,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TEST MODEL 4')

print(metrics.classification_report(y_test,y_class_pred))
```

Out[63]:

```
<AxesSubplot:>
```

Out[63]:

```
Text(0.5, 12.5, 'Predicted')
```

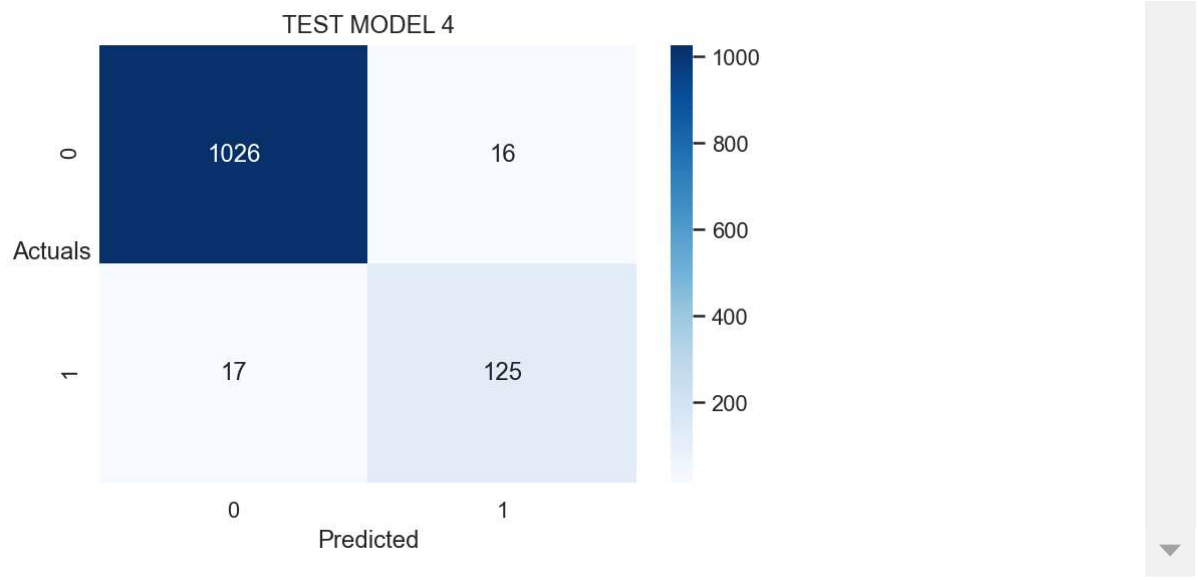
Out[63]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[63]:

```
Text(0.5, 1.0, 'TEST MODEL 4')
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98      1042
         1.0       0.89      0.88      0.88       142

    accuracy                           0.97      1184
   macro avg       0.94      0.93      0.93      1184
weighted avg       0.97      0.97      0.97      1184
```



- Model 3 has the best metrics
- But, Model 4 is also just slightly behind
- But majorly, Model 4 has only 9 predictors versus 23 of Model 3
- Hence, lets decide to continue with more analysis on Model 4

In [64]:

```
X_train.shape
```

Out[64]:

```
(2402, 9)
```

In [65]:

```
final_cols = X_train.columns
final_cols
```

Out[65]:

```
Index(['Selling_Cost', 'PBIDT', 'ROG_Net_Worth_perc', 'ROG_Capital_Employed_
perc', 'ROG_Total_Assets_perc', 'Current_Ratio_Latest', 'Interest_Cover_Rati
o_Latest', 'APATM_perc_Latest', 'Book_Value_Adj_Unit_Curr'], dtype='object')
```

In [66]:

```
data = pd.concat([X_train, y_train], axis=1)
data1 = pd.concat([X_test, y_test], axis=1)
data.shape
data1.shape
```

Out[66]:

```
(2402, 10)
```

Out[66]:

```
(1184, 10)
```

In [67]:

```
final_data = pd.concat([data, data1], axis=0)
final_data.shape
```

Out[67]:

```
(3586, 10)
```

In [68]:

```
final_data.to_csv('final_data.csv')
final_data.to_excel('final_data.xlsx')
```

Model 5

Logistic Regression - Fine Tuning by using GridSearchCV

In [69]:

```
X_train.shape
```

Out[69]:

```
(2402, 9)
```

In [70]:

```
grid={'penalty':['l2','none', 'l1'],
      'solver':['lbfgs', 'liblinear', 'sag', 'saga', 'newton-cg'],
      'tol':[0.0001,0.00001]}
```

In [71]:

```
model = LogisticRegression(max_iter=10000,n_jobs=2)
```

In [72]:

```
grid_search = GridSearchCV(estimator = model, param_grid = grid, cv = 10,n_jobs=-1,scoring=
```

In [73]:

```
grid_search.fit(X_train, y_train)
```

```
Fitting 10 folds for each of 30 candidates, totalling 300 fits

C:\Users\rahul\anaconda3\lib\site-packages\sklearn\model_selection\_validati
on.py:372: FitFailedWarning:
80 fits failed out of a total of 300.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting err
or_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------
----
20 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\model_selection\_
validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\linear_model\_log
istic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\linear_model\_log
istic.py", line 464, in _check_solver
    raise ValueError("penalty='none' is not supported for the liblinear solv
er")
ValueError: penalty='none' is not supported for the liblinear solver

--------------------------------------------------------------------------
----
20 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\model_selection\_
validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\linear_model\_log
istic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\linear_model\_log
istic.py", line 447, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 pena
lty.

--------------------------------------------------------------------------
----
20 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\model_selection\_
validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\linear_model\_log
istic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\linear_model\_log
istic.py", line 447, in _check_solver
    raise ValueError(
```

```
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1 penalt
y.

--------------------------------------------------------------------------
----
20 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\model_selection\_
validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\linear_model\_log
istic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\rahul\anaconda3\lib\site-packages\sklearn\linear_model\_log
istic.py", line 447, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got l1
penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\rahul\anaconda3\lib\site-packages\sklearn\model_selection\_search.p
y:969: UserWarning: One or more of the test scores are non-finite: [0.880675
5  0.8806755  0.87895001 0.87895001 0.8806755  0.8806755
 0.8806755  0.8806755  0.8806755  0.8806755  0.88159272 0.88159272
        nan        nan 0.88159272 0.88159272 0.88159272 0.88159272
 0.88159272 0.88159272        nan        nan 0.87895001 0.87895001
        nan        nan 0.87895001 0.87895001        nan        nan]
  warnings.warn(
```

Out[73]:

```
GridSearchCV(cv=10, estimator=LogisticRegression(max_iter=10000, n_jobs=2),
             n_jobs=-1,
             param_grid={'penalty': ['l2', 'none', 'l1'],
                         'solver': ['lbfgs', 'liblinear', 'sag', 'saga',
                                    'newton-cg'],
                         'tol': [0.0001, 1e-05]},
             scoring='f1', verbose=True)
```

In [74]:

```
print(grid_search.best_params_,'\n')
print(grid_search.best_estimator_)
```

```
{'penalty': 'none', 'solver': 'lbfgs', 'tol': 0.0001}

LogisticRegression(max_iter=10000, n_jobs=2, penalty='none')
```

In [75]:

```
best_model = grid_search.best_estimator_
```

In [76]:

```
logit4_train_score = best_model.score(X_train, y_train)
logit4_test_score = best_model.score(X_test, y_test)
logit4_test_f1 = metrics.f1_score(y_test, best_model.predict(X_test))
logit4_test_f0 = metrics.f1_score(y_test, best_model.predict(X_test), pos_label=0)

print('Regularised Logistic Regression Train Score =', round(logit4_train_score, 4))
print('Regularised Logistic Regression Test Score =', round(logit4_test_score, 4))
print('Regularised Logistic Regression f1 Test Score for "1" =', round(logit4_test_f1, 4))
print('Regularised Logistic Regression f1 Test Score for "0" =', round(logit4_test_f0, 4))
```

```
Regularised Logistic Regression Train Score = 0.9771
Regularised Logistic Regression Test Score = 0.9721
Regularised Logistic Regression f1 Test Score for "1" = 0.8834
Regularised Logistic Regression f1 Test Score for "0" = 0.9842
```

In [77]:

```
print('TEST REGULARISED MODEL')
print('')
print(metrics.classification_report(y_test, best_model.predict(X_test)))
```

```
TEST REGULARISED MODEL

              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98      1042
         1.0       0.89      0.88      0.88       142

    accuracy                           0.97      1184
   macro avg       0.94      0.93      0.93      1184
weighted avg       0.97      0.97      0.97      1184
```

Model 6

Choosing Optimal Threshold for Model 4

In [78]:

```
X_train.shape
```

Out[78]:

```
(2402, 9)
```

In [79]:

```
from sklearn.metrics import roc_curve

y_prob_pred_train = model_4.predict(X_train)
fpr, tpr, thresholds = roc_curve(y_train,y_prob_pred_train)
```

In [80]:

```
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
optimal_threshold
```

Out[80]:

```
0.08409205000274865
```

In [81]:

```
## MODEL 6 TRAIN

y_prob_pred_train = model_4.predict(X_train)

y_class_pred=[]
for i in range(0,len(y_prob_pred_train)):
    if np.array(y_prob_pred_train)[i]>optimal_threshold:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_train,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TRAIN MODEL 6')

print(metrics.classification_report(y_train,y_class_pred))
```

Out[81]:

```
<AxesSubplot:>
```

Out[81]:

```
Text(0.5, 12.5, 'Predicted')
```

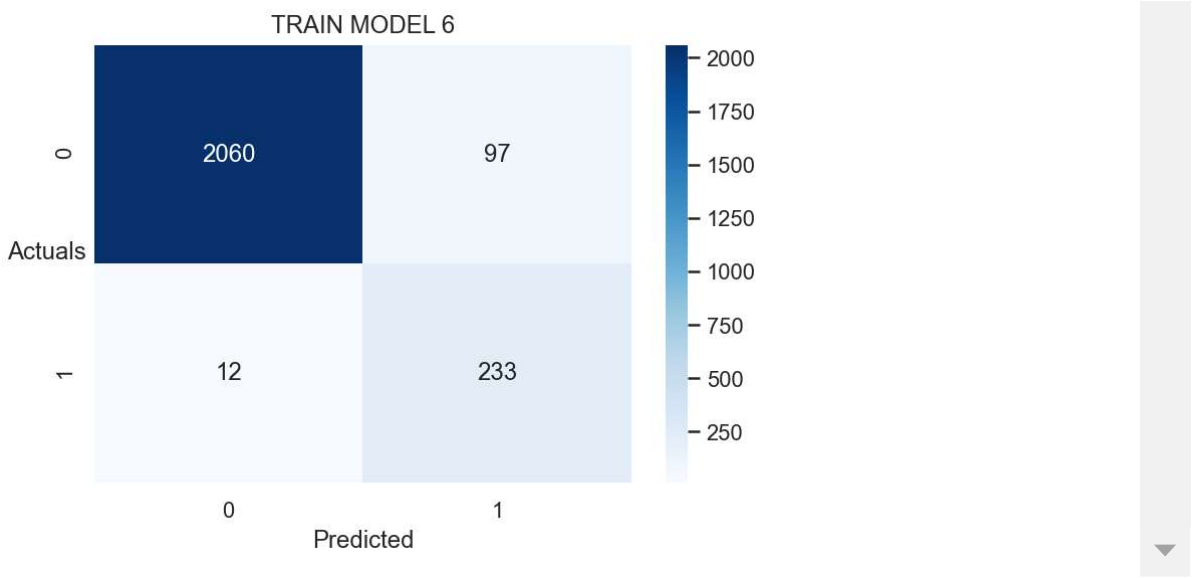
Out[81]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[81]:

```
Text(0.5, 1.0, 'TRAIN MODEL 6')
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.96      0.97      2157
         1.0       0.71      0.95      0.81       245

    accuracy                           0.95      2402
   macro avg       0.85      0.95      0.89      2402
weighted avg       0.96      0.95      0.96      2402
```


TRAIN MODEL 6
2060
97
12
233
0
1
Actuals
0
1
Predicted
2000
1750
1500
1250
1000
750
500
250

In [82]:

```
## MODEL 6 TEST

y_prob_pred_test = model_4.predict(X_test)

y_class_pred=[]
for i in range(0,len(y_prob_pred_test)):
    if np.array(y_prob_pred_test)[i]>optimal_threshold:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_test,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TEST MODEL 6')

print(metrics.classification_report(y_test,y_class_pred))
```

Out[82]:

```
<AxesSubplot:>
```

Out[82]:

```
Text(0.5, 12.5, 'Predicted')
```

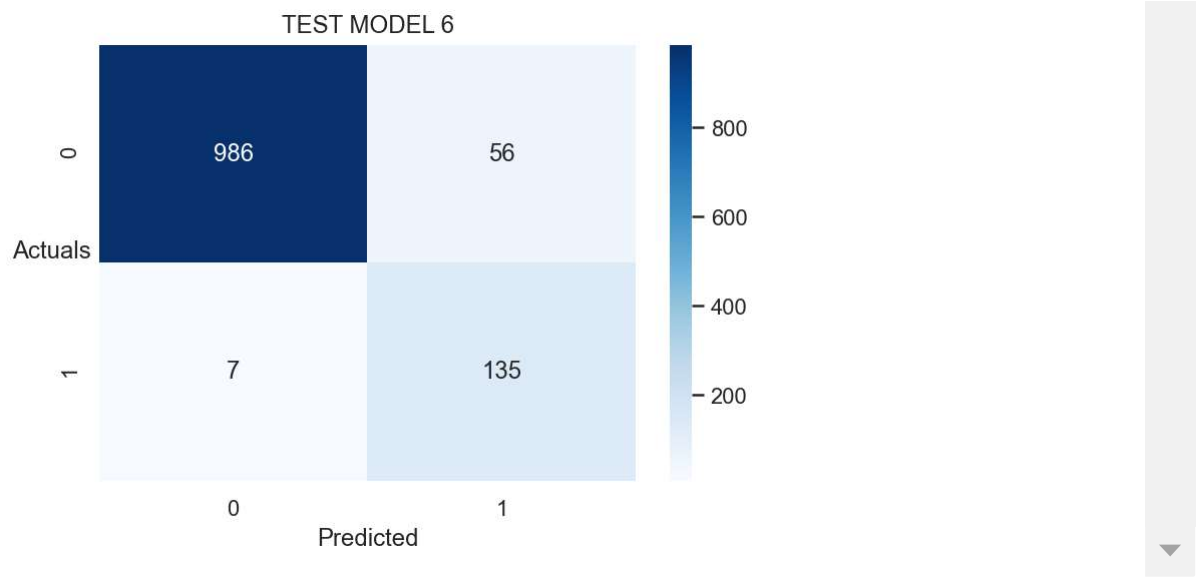
Out[82]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[82]:

```
Text(0.5, 1.0, 'TEST MODEL 6')
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.95      0.97      1042
         1.0       0.71      0.95      0.81       142

    accuracy                           0.95      1184
   macro avg       0.85      0.95      0.89      1184
weighted avg       0.96      0.95      0.95      1184
```



Model 7

Using Recursive Feature Elimination Method to choose features

In [83]:

```
X_train, X_test, y_train, y_test = train_test_split(X, df_y,
                                                    test_size = 0.33, random_state=42)


#### For modeling we will use Logistic Regression with recursive feature elimination
from sklearn.feature_selection import RFE

LogR = LogisticRegression()

selector = RFE(estimator = LogR, n_features_to_select=15, step=1)

selector = selector.fit(X_train, y_train)

df = pd.DataFrame({'Feature': X.columns, 'Rank': selector.ranking_})
```

In [84]:

```
df.sort_values(by='Rank')
```

Out[84]:

	Feature	Rank
0	Other_Income	1
20	Value_of_Output_to_Gross_Block	1
19	Value_of_Output_to_Total_Assets	1
18	APATM_perc_Latest	1
17	Interest_Cover_Ratio_Latest	1
16	Current_Ratio_Latest	1
15	ROG_CP_perc	1
13	ROG_PBIDT_perc	1
12	ROG_Total_Assets_perc	1
22	Book_Value_Adj_Unit_Curr	1
8	ROG_Capital_Employed_perc	1
7	ROG_Net_Worth_perc	1
2	PBIDT	1
1	Selling_Cost	1
9	ROG_Gross_Block_perc	1
11	ROG_Cost_of_Production_perc	2
3	Adjusted_PAT	3
4	Cash_Flow_From_Operating_Activities	4
21	Net_Working_Capital	5
10	ROG_Net_Sales_perc	6
14	ROG_PBT_perc	7
6	Cash_Flow_From_Financing_Activities	8
5	Cash_Flow_From_Investing_Activities	9

In [85]:

```
pred_train = selector.predict(X_train)
pred_test = selector.predict(X_test)

print(confusion_matrix(y_train, pred_train))

print(classification_report(y_train, pred_train))
```

```
[[2140   17]
 [  33  212]]
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99      2157
         1.0       0.93      0.87      0.89       245

    accuracy                           0.98      2402
   macro avg       0.96      0.93      0.94      2402
weighted avg       0.98      0.98      0.98      2402
```

In [86]:

```
print(confusion_matrix(y_test, pred_test))

print(classification_report(y_test, pred_test))
```

```
[[1026   16]
 [  18  124]]
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98      1042
         1.0       0.89      0.87      0.88       142

    accuracy                           0.97      1184
   macro avg       0.93      0.93      0.93      1184
weighted avg       0.97      0.97      0.97      1184
```

Model 8

Using SMOTE to Balance the target Variable 'default'

We use SMOTE on Model 7 - RFE

In [87]:

```
df = df[df['Rank'] == 1]

l = df['Feature'].values
list(l)

X = X[l]
X.columns
X.shape
```

Out[87]:

```
['Other_Income',
 'Selling_Cost',
 'PBIDT',
 'ROG_Net_Worth_perc',
 'ROG_Capital_Employed_perc',
 'ROG_Gross_Block_perc',
 'ROG_Total_Assets_perc',
 'ROG_PBIDT_perc',
 'ROG_CP_perc',
 'Current_Ratio_Latest',
 'Interest_Cover_Ratio_Latest',
 'APATM_perc_Latest',
 'Value_of_Output_to_Total_Assets',
 'Value_of_Output_to_Gross_Block',
 'Book_Value_Adj_Unit_Curr']
```

Out[87]:

```
Index(['Other_Income', 'Selling_Cost', 'PBIDT', 'ROG_Net_Worth_perc', 'ROG_C
apital_Employed_perc', 'ROG_Gross_Block_perc', 'ROG_Total_Assets_perc', 'ROG
_PBIDT_perc', 'ROG_CP_perc', 'Current_Ratio_Latest', 'Interest_Cover_Ratio_L
atest', 'APATM_perc_Latest', 'Value_of_Output_to_Total_Assets', 'Value_of_Ou
tput_to_Gross_Block', 'Book_Value_Adj_Unit_Curr'], dtype='object')
```

Out[87]:

```
(3586, 15)
```

In [88]:

```
X_train, X_test, y_train, y_test = train_test_split(X, df_y,
                                                    test_size = 0.33, random_state=42)
```

In [89]:

```
X_train.shape
```

Out[89]:

```
(2402, 15)
```

In [90]:

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=33)
X_res, y_res = sm.fit_resample(X_train, y_train)
```

In [91]:

```
selector_smote = selector.fit(X_res, y_res)
```

In [92]:

```
selector_smote.n_features_
```

Out[92]:

15

In [93]:

```
smote_model8 = pd.DataFrame({'Feature': X_train.columns, 'Rank': selector_smote.ranking_})
smote_model8 = smote_model8[smote_model8['Rank'] == 1]
smote_model8
```

Out[93]:

	Feature	Rank
0	Other_Income	1
1	Selling_Cost	1
2	PBIDT	1
3	ROG_Net_Worth_perc	1
4	ROG_Capital_Employed_perc	1
5	ROG_Gross_Block_perc	1
6	ROG_Total_Assets_perc	1
7	ROG_PBIDT_perc	1
8	ROG_CP_perc	1
9	Current_Ratio_Latest	1
10	Interest_Cover_Ratio_Latest	1
11	APATM_perc_Latest	1
12	Value_of_Output_to_Total_Assets	1
13	Value_of_Output_to_Gross_Block	1
14	Book_Value_Adj_Unit_Curr	1

In [94]:

```
smote_model8.to_csv('smote_model8.csv')
```

In [95]:

```
l = smote_model8['Feature'].values
list(l)
l
```

Out[95]:

```
['Other_Income',
 'Selling_Cost',
 'PBIDT',
 'ROG_Net_Worth_perc',
 'ROG_Capital_Employed_perc',
 'ROG_Gross_Block_perc',
 'ROG_Total_Assets_perc',
 'ROG_PBIDT_perc',
 'ROG_CP_perc',
 'Current_Ratio_Latest',
 'Interest_Cover_Ratio_Latest',
 'APATM_perc_Latest',
 'Value_of_Output_to_Total_Assets',
 'Value_of_Output_to_Gross_Block',
 'Book_Value_Adj_Unit_Curr']
```

Out[95]:

```
array(['Other_Income', 'Selling_Cost', 'PBIDT', 'ROG_Net_Worth_perc',
       'ROG_Capital_Employed_perc', 'ROG_Gross_Block_perc',
       'ROG_Total_Assets_perc', 'ROG_PBIDT_perc', 'ROG_CP_perc',
       'Current_Ratio_Latest', 'Interest_Cover_Ratio_Latest',
       'APATM_perc_Latest', 'Value_of_Output_to_Total_Assets',
       'Value_of_Output_to_Gross_Block', 'Book_Value_Adj_Unit_Curr'],
      dtype=object)
```

In [96]:

```
data15 = X[l]
data15['default'] = df_y
data15_orig = df_imputed[l]
data15_orig['default'] = df_y

data15.to_csv('data15.csv')
data15_orig.to_csv('data15_orig.csv')
```

```
<ipython-input-96-0930bfd0ebeb>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  data15_orig['default'] = df_y
```

In [97]:

```
df_orig_new = df_orig.copy()
df_orig_new = df_orig_new[l]
df_orig_new['default'] = df_y
df_orig_new.to_csv('df_orig15.csv')
```

In [98]:

```
pred_train_smote = selector_smote.predict(X_res)
pred_test_smote = selector_smote.predict(X_test)
```

In [99]:

```
print(classification_report(y_res, pred_train_smote))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.97      0.97      2157
         1.0       0.97      0.96      0.97      2157

    accuracy                           0.97      4314
   macro avg       0.97      0.97      0.97      4314
weighted avg       0.97      0.97      0.97      4314
```

In [100]:

```
print(classification_report(y_test, pred_test_smote))
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.96      0.98      1042
         1.0       0.78      0.95      0.86       142

    accuracy                           0.96      1184
   macro avg       0.89      0.96      0.92      1184
weighted avg       0.97      0.96      0.96      1184
```

In [101]:

```
X_res.shape
y_res.shape
```

Out[101]:

```
(4314, 15)
```

Out[101]:

```
(4314,)
```

In [102]:

```
X_test.shape
y_test.shape
```

Out[102]:

```
(1184, 15)
```

Out[102]:

```
(1184,)
```

In [103]:

```
import statsmodels.api as SM

# f1 = 'default ~ Equity_Paid_Up + Total_Debt + Other_Income + Selling_Cost + Adjusted_PAT
model_8 = SM.Logit(y_res, X_res).fit()
print(model_8.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.098486
         Iterations 9
                           Logit Regression Results
==============================================================================
=====
Dep. Variable:                 default   No. Observations:
4314
Model:                           Logit   Df Residuals:
4299
Method:                            MLE   Df Model:
14
Date:                Sat, 08 Jan 2022   Pseudo R-squ.:
0.8579
Time:                        20:55:38   Log-Likelihood:                -4
24.87
converged:                       True   LL-Null:                       -2
990.2
Covariance Type:            nonrobust   LLR p-value:
0.000
==============================================================================
==========================
                                   coef    std err          z      P>|
z|      [0.025      0.975]
------------------------------------------------------------------------------
-------------------------
Other_Income                     0.3180      0.101      3.145      0.0
02       0.120       0.516
Selling_Cost                     0.6835      0.119      5.754      0.0
00       0.451       0.916
PBIDT                           -0.3132      0.056     -5.553      0.0
00      -0.424      -0.203
ROG_Net_Worth_perc              -0.4148      0.049     -8.388      0.0
00      -0.512      -0.318
ROG_Capital_Employed_perc        0.4119      0.054      7.656      0.0
00       0.306       0.517
ROG_Gross_Block_perc             0.0368      0.047      0.790      0.4
29      -0.055       0.128
ROG_Total_Assets_perc           -0.2154      0.055     -3.939      0.0
00      -0.323      -0.108
ROG_PBIDT_perc                   0.2171      0.057      3.835      0.0
00       0.106       0.328
ROG_CP_perc                     -0.1513      0.057     -2.677      0.0
07      -0.262      -0.041
Current_Ratio_Latest            -0.3419      0.107     -3.199      0.0
01      -0.551      -0.132
Interest_Cover_Ratio_Latest     -0.3591      0.059     -6.077      0.0
00      -0.475      -0.243
APATM_perc_Latest               -0.2973      0.055     -5.371      0.0
```

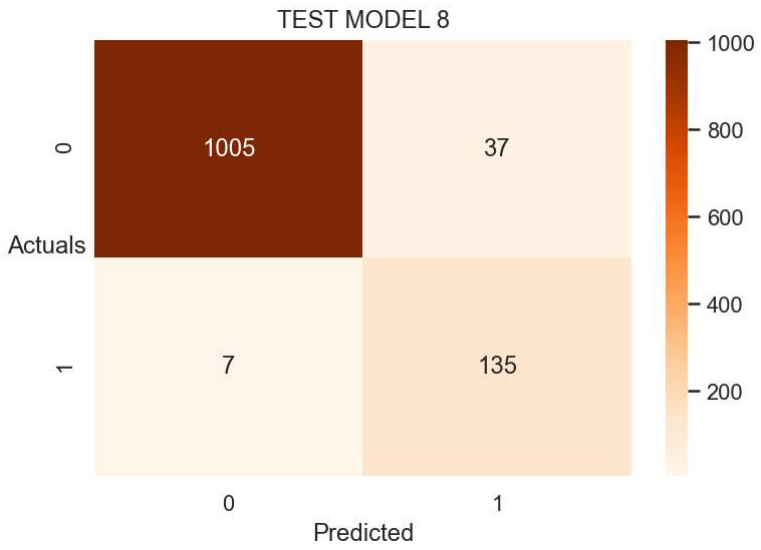
```
00       -0.406       -0.189
Value_of_Output_to_Total_Assets     -0.0817       0.177      -0.461       0.6
45       -0.429        0.266
Value_of_Output_to_Gross_Block       0.2140       0.092       2.329       0.0
20        0.034        0.394
Book_Value_Adj_Unit_Curr            -1.6181       0.071     -22.933       0.0
00       -1.756       -1.480
==============================================================================
==========================
```

In [104]:

```
LRresult1 = (model_8.summary().tables[1])
LRresult1 = pd.DataFrame(LRresult1)
LRresult1.to_csv('model_8_summary.csv')
```

In [105]:

```
## MODEL 8 TEST

sns.heatmap((metrics.confusion_matrix(y_test,pred_test_smote)),annot=True,fmt='.5g'
            ,cmap='Oranges');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TEST MODEL 8')
plt.savefig('model8_cm.jpg', bbox_inches='tight');
```



In [106]:

```
print(metrics.classification_report(y_test,pred_test_smote))

cr = classification_report(y_test, pred_test_smote, output_dict=True)
cr = pd.DataFrame(cr).T
cr.to_csv('cr.csv')
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.96      0.98      1042
         1.0       0.78      0.95      0.86       142

    accuracy                           0.96      1184
   macro avg       0.89      0.96      0.92      1184
weighted avg       0.97      0.96      0.96      1184
```

Model 9

Using SMOTE to Balance the target Variable 'default' and Choosing Optimal Threshold

In [107]:

```
selector_smote.predict_proba(X_res)
```

Out[107]:

```
array([[9.89099942e-01, 1.09000581e-02],
       [9.99459901e-01, 5.40099233e-04],
       [9.25872325e-01, 7.41276754e-02],
       ...,
       [3.25320052e-02, 9.67467995e-01],
       [8.21928397e-05, 9.99917807e-01],
       [1.22675352e-03, 9.98773246e-01]])
```

In [108]:

```
from sklearn.metrics import roc_curve

y_prob_pred_train = selector_smote.predict_proba(X_res)[:,1]
y_prob_pred_test = selector_smote.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_res,y_prob_pred_train)
```

In [109]:

```
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
optimal_threshold
```

Out[109]:

```
0.4246321187560022
```

In [110]:

```
y_prob_pred_test
```

Out[110]:

```
array([0.01127219, 0.00159853, 0.20162513, ..., 0.01332857, 0.00097907,
       0.00378365])
```

In [111]:

```
y_class_pred=[]
for i in range(0,len(y_prob_pred_test)):
    if np.array(y_prob_pred_test)[i]>optimal_threshold:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_test,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TEST MODEL 9')

print(metrics.classification_report(y_test,y_class_pred))
```

Out[111]:

```
<AxesSubplot:>
```

Out[111]:

```
Text(0.5, 12.5, 'Predicted')
```

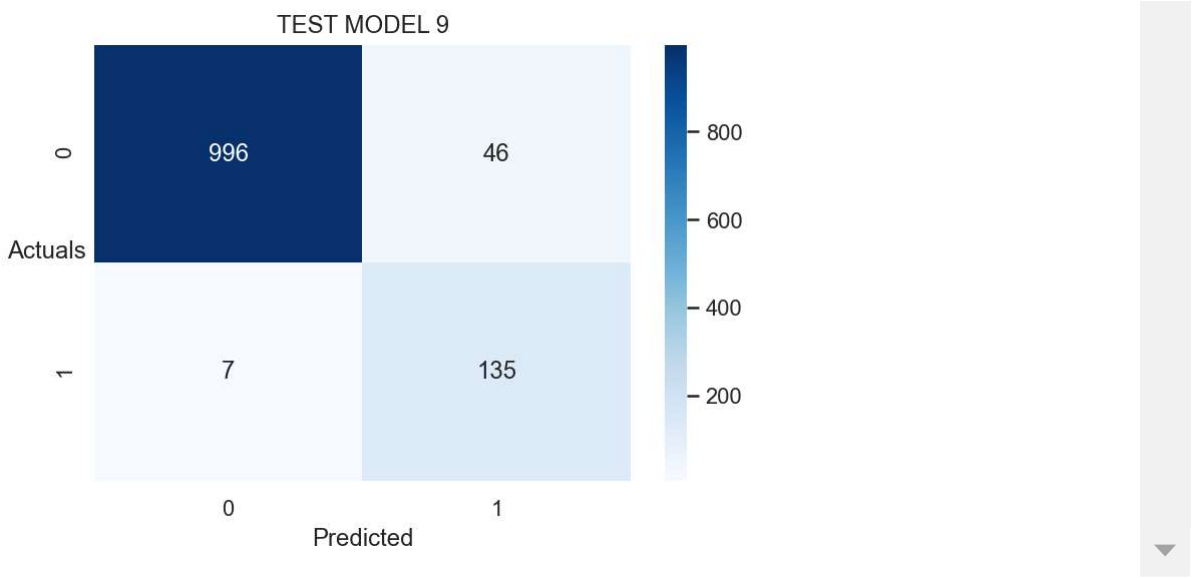
Out[111]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[111]:

```
Text(0.5, 1.0, 'TEST MODEL 9')
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.96      0.97      1042
         1.0       0.75      0.95      0.84       142

    accuracy                           0.96      1184
   macro avg       0.87      0.95      0.90      1184
weighted avg       0.96      0.96      0.96      1184
```


TEST MODEL 9
996
46
7
135
0
1
Actuals
0
1
Predicted
800
600
400
200

In [112]:

```
import statsmodels.api as SM

# f1 = 'default ~ Equity_Paid_Up + Total_Debt + Other_Income + Selling_Cost + Adjusted_PAT
model_9 = SM.Logit(y_res, X_res).fit()
print(model_9.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.098486
         Iterations 9
                           Logit Regression Results
==============================================================================
=====
Dep. Variable:                 default   No. Observations:
4314
Model:                           Logit   Df Residuals:
4299
Method:                            MLE   Df Model:
14
Date:                Sat, 08 Jan 2022   Pseudo R-squ.:
0.8579
Time:                        20:55:39   Log-Likelihood:                -4
24.87
converged:                       True   LL-Null:                       -2
990.2
Covariance Type:            nonrobust   LLR p-value:
0.000
==============================================================================
==========================
                                 coef    std err          z      P>|
z|      [0.025      0.975]
------------------------------------------------------------------------------
--------------------------
Other_Income                     0.3180      0.101      3.145      0.0
02       0.120       0.516
Selling_Cost                     0.6835      0.119      5.754      0.0
00       0.451       0.916
PBIDT                           -0.3132      0.056     -5.553      0.0
00      -0.424      -0.203
ROG_Net_Worth_perc              -0.4148      0.049     -8.388      0.0
00      -0.512      -0.318
ROG_Capital_Employed_perc        0.4119      0.054      7.656      0.0
00       0.306       0.517
ROG_Gross_Block_perc             0.0368      0.047      0.790      0.4
29      -0.055       0.128
ROG_Total_Assets_perc           -0.2154      0.055     -3.939      0.0
00      -0.323      -0.108
ROG_PBIDT_perc                   0.2171      0.057      3.835      0.0
00       0.106       0.328
ROG_CP_perc                     -0.1513      0.057     -2.677      0.0
07      -0.262      -0.041
Current_Ratio_Latest            -0.3419      0.107     -3.199      0.0
01      -0.551      -0.132
Interest_Cover_Ratio_Latest     -0.3591      0.059     -6.077      0.0
00      -0.475      -0.243
APATM_perc_Latest               -0.2973      0.055     -5.371      0.0
00      -0.406      -0.189
```

```
Value_of_Output_to_Total_Assets     -0.0817      0.177      -0.461      0.6
45       -0.429       0.266
Value_of_Output_to_Gross_Block       0.2140      0.092       2.329      0.0
20        0.034       0.394
Book_Value_Adj_Unit_Curr            -1.6181      0.071     -22.933      0.0
00       -1.756      -1.480
==============================================================================
==========================
```

In [113]:

```
LRresult = (model_9.summary().tables[1])
LRresult = pd.DataFrame(LRresult)
LRresult.to_csv('model_9_summary.csv')
LRresult
```

Out[113]:

	0	1	2	3	4	5	6
0		coef	std err	z	P>|z|	[0.025	0.975]
1	Other_Income	0.3180	0.101	3.145	0.002	0.120	0.516
2	Selling_Cost	0.6835	0.119	5.754	0.000	0.451	0.916
3	PBIDT	-0.3132	0.056	-5.553	0.000	-0.424	-0.203
4	ROG_Net_Worth_perc	-0.4148	0.049	-8.388	0.000	-0.512	-0.318
5	ROG_Capital_Employed_perc	0.4119	0.054	7.656	0.000	0.306	0.517
6	ROG_Gross_Block_perc	0.0368	0.047	0.790	0.429	-0.055	0.128
7	ROG_Total_Assets_perc	-0.2154	0.055	-3.939	0.000	-0.323	-0.108
8	ROG_PBIDT_perc	0.2171	0.057	3.835	0.000	0.106	0.328
9	ROG_CP_perc	-0.1513	0.057	-2.677	0.007	-0.262	-0.041
10	Current_Ratio_Latest	-0.3419	0.107	-3.199	0.001	-0.551	-0.132
11	Interest_Cover_Ratio_Latest	-0.3591	0.059	-6.077	0.000	-0.475	-0.243
12	APATM_perc_Latest	-0.2973	0.055	-5.371	0.000	-0.406	-0.189
13	Value_of_Output_to_Total_Assets	-0.0817	0.177	-0.461	0.645	-0.429	0.266
14	Value_of_Output_to_Gross_Block	0.2140	0.092	2.329	0.020	0.034	0.394
15	Book_Value_Adj_Unit_Curr	-1.6181	0.071	-22.933	0.000	-1.756	-1.480

In [114]:

```
import time
start = time.time()

for i in range(23):
    model_10 = SM.Logit(y_res, X_res).fit()
    v = pd.DataFrame(model_10.pvalues.sort_values(ascending=False), columns=['pvalue']).res
    if (v.iloc[0,1] > 0.05):
        X_res = X_res.drop(columns=v.iloc[0,0])

end = time.time()
print('Time taken =', end-start)
```

```
Optimization terminated successfully.
         Current function value: 0.098486
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098510
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
```

```
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Optimization terminated successfully.
         Current function value: 0.098570
         Iterations 9
Time taken = 0.2236802577972412
```

In [115]:

```
print(model_10.summary())
```

```
                           Logit Regression Results
==============================================================================
==
Dep. Variable:                default   No. Observations:                 43
14
Model:                          Logit   Df Residuals:                     43
01
Method:                           MLE   Df Model:
12
Date:                Sat, 08 Jan 2022   Pseudo R-squ.:                  0.85
78
Time:                        20:55:39   Log-Likelihood:                -425.
23
converged:                       True   LL-Null:                       -299
0.2
Covariance Type:            nonrobust   LLR p-value:                     0.0
00
==============================================================================
======================
                                   coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
----------------------
Other_Income                     0.3276      0.100      3.278      0.001
0.132       0.523
Selling_Cost                     0.6835      0.115      5.937      0.000
0.458       0.909
PBIDT                           -0.3166      0.056     -5.640      0.000
-0.427      -0.207
ROG_Net_Worth_perc              -0.4091      0.048     -8.462      0.000
-0.504      -0.314
ROG_Capital_Employed_perc        0.4083      0.053      7.642      0.000
0.304       0.513
ROG_Total_Assets_perc           -0.2070      0.054     -3.862      0.000
-0.312      -0.102
ROG_PBIDT_perc                   0.2241      0.056      4.015      0.000
0.115       0.334
ROG_CP_perc                     -0.1541      0.056     -2.745      0.006
-0.264      -0.044
Current_Ratio_Latest            -0.3555      0.106     -3.367      0.001
-0.562      -0.149
Interest_Cover_Ratio_Latest     -0.3603      0.059     -6.121      0.000
-0.476      -0.245
APATM_perc_Latest               -0.2972      0.055     -5.431      0.000
-0.404      -0.190
Value_of_Output_to_Gross_Block   0.1846      0.078      2.365      0.018
0.032       0.338
Book_Value_Adj_Unit_Curr        -1.6073      0.069    -23.381      0.000
-1.742      -1.473
==============================================================================
======================
```

In [116]:

```
LRresult1 = (model_10.summary().tables[1])
LRresult1 = pd.DataFrame(LRresult1)
# LRresult1.to_csv('model_9_summary.csv')
```

In [117]:

```
X_test.drop(['Value_of_Output_to_Total_Assets', 'ROG_Gross_Block_perc'], axis=1, inplace=Tr
X_test.shape
```

Out[117]:

```
(1184, 13)
```

In [118]:

```
## MODEL 10 TEST

y_prob_pred_test = model_10.predict(X_test)

y_class_pred=[]
for i in range(0,len(y_prob_pred_test)):
    if np.array(y_prob_pred_test)[i]>0.5:
        a=1
    else:
        a=0
    y_class_pred.append(a)

from sklearn import metrics

sns.heatmap((metrics.confusion_matrix(y_test,y_class_pred)),annot=True,fmt='.5g'
            ,cmap='Blues');
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0);
plt.title('TEST MODEL 4')

print(metrics.classification_report(y_test,y_class_pred))
```

Out[118]:

```
<AxesSubplot:>
```

Out[118]:

```
Text(0.5, 12.5, 'Predicted')
```

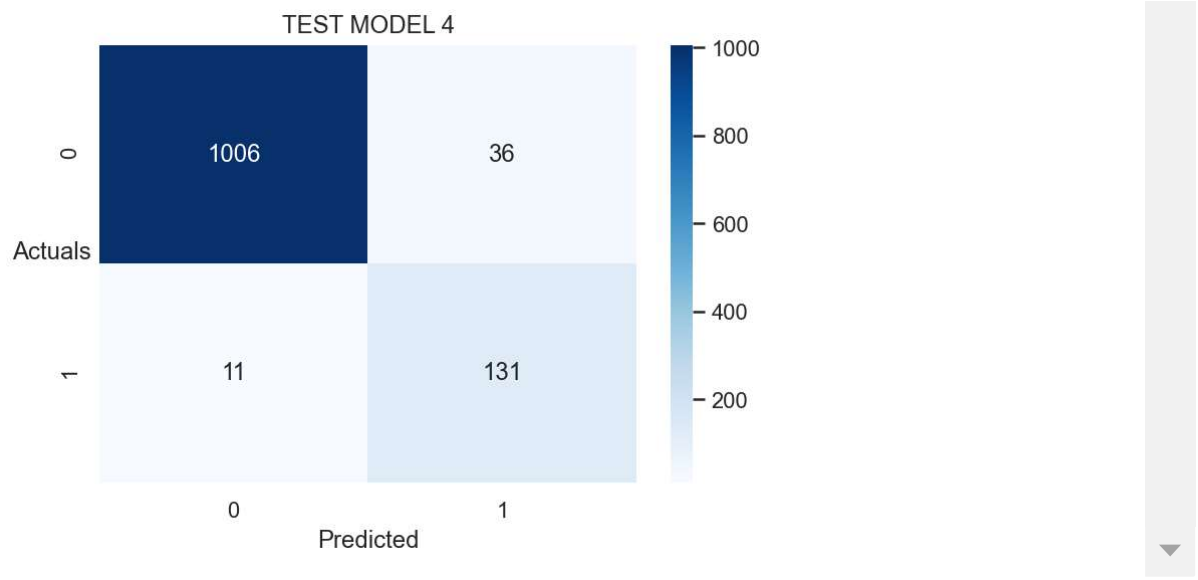
Out[118]:

```
Text(30.5, 0.5, 'Actuals')
```

Out[118]:

```
Text(0.5, 1.0, 'TEST MODEL 4')
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.97      0.98      1042
         1.0       0.78      0.92      0.85       142

    accuracy                           0.96      1184
   macro avg       0.89      0.94      0.91      1184
weighted avg       0.96      0.96      0.96      1184
```



Data visualisation _ company data scalled

In [119]:

```
comp = pd.read_csv('df_orig15.csv')
scaled_comp = pd.read_csv('data15.csv')
```

In [120]:

```
comp.drop('Unnamed: 0', axis=1, inplace=True)
scaled_comp.drop('Unnamed: 0', axis=1, inplace=True)
comp.head()
```

Out[120]:

	Other_Income	Selling_Cost	PBIDT	ROG_Net_Worth_perc	ROG_Capital_Employed_perc	RO
0	7.60	0.00	-179.06	-15.31	-20.76	
1	46.27	40.51	646.46	-26.15	6.30	
2	9.55	54.83	-281.92	-61.86	15.66	
3	223.85	3.34	-213.01	-450.67	-40.84	
4	9.82	1.97	-647.86	-559.83	-11.76	

In [121]:

```
scaled_comp.head()
```

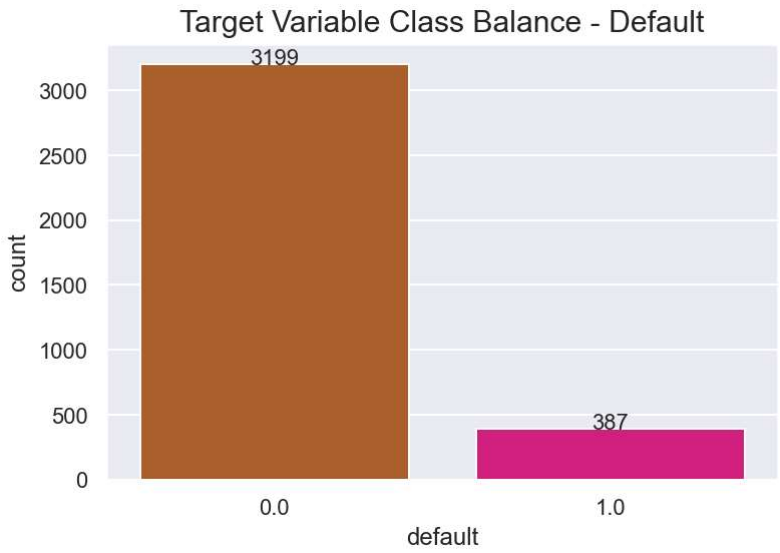
Out[121]:

	Other_Income	Selling_Cost	PBIDT	ROG_Net_Worth_perc	ROG_Capital_Employed_perc	ROG
0	3.0	0.492	-3.0	-3.0	-3.0	
1	3.0	3.000	3.0	-3.0	3.0	
2	3.0	3.000	-3.0	-3.0	3.0	
3	3.0	3.000	-3.0	-3.0	-3.0	
4	3.0	1.970	-3.0	-3.0	-3.0	

In [122]:

```
countplt, ax = plt.subplots()
ax =sns.countplot(comp['default'] );


# show count (+ annotate)
for rect in ax.patches:
    ax.text (rect.get_x() + rect.get_width()  / 2,rect.get_height()+ 0.75,rect.get_height())
countplt

plt.title('Target Variable Class Balance - Default', fontsize=15);

plt.savefig('default_count.jpg', bbox_inches='tight');
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Future
Warning: Pass the following variable as a keyword arg: x. From version 0.12,
the only valid positional argument will be `data`, and passing other argumen
ts without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

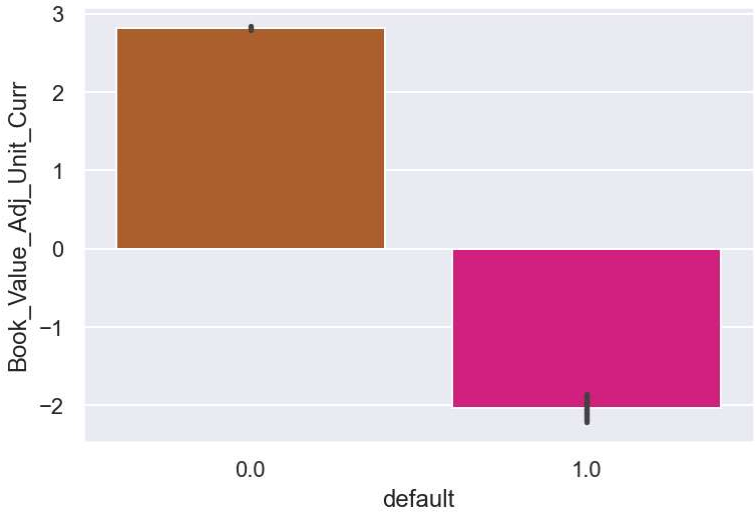


In [123]:

```
temp=scaled_comp.drop('default', axis=1)
sns.barplot(x=scaled_comp['default'], y=scaled_comp['Book_Value_Adj_Unit_Curr'])
```

Out[123]:

```
<AxesSubplot:xlabel='default', ylabel='Book_Value_Adj_Unit_Curr'>
```

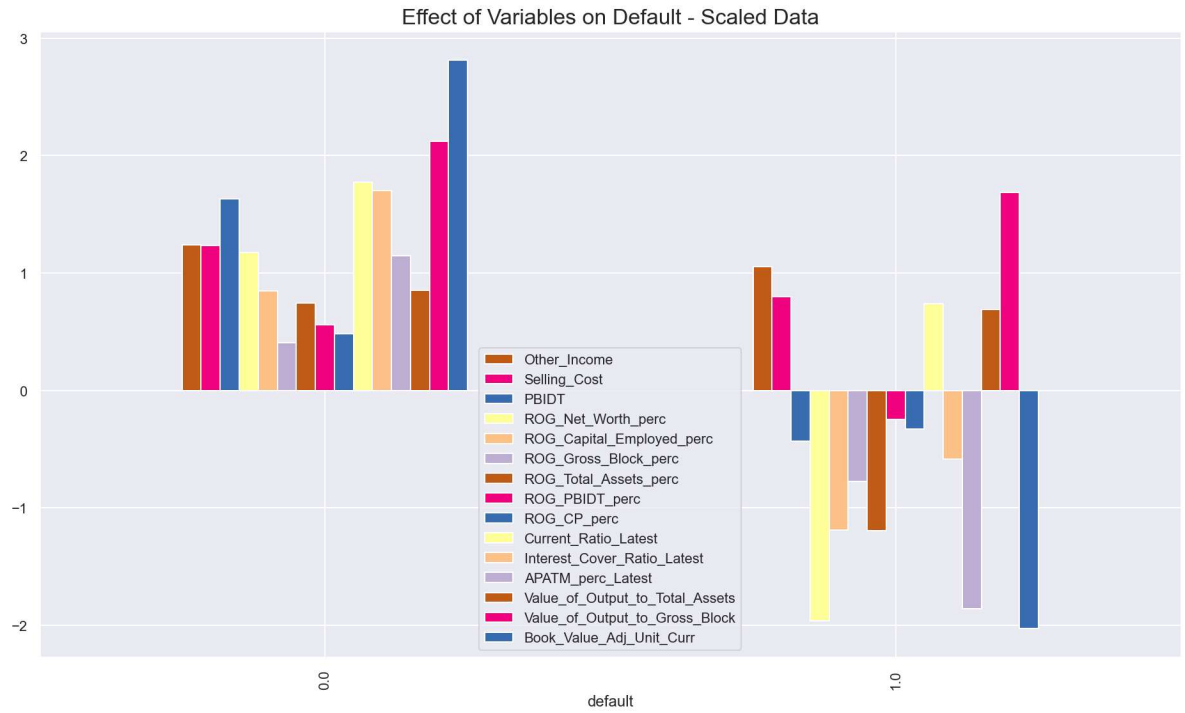


In [124]:

```
grp_sc = scaled_comp.groupby('default').mean()
```

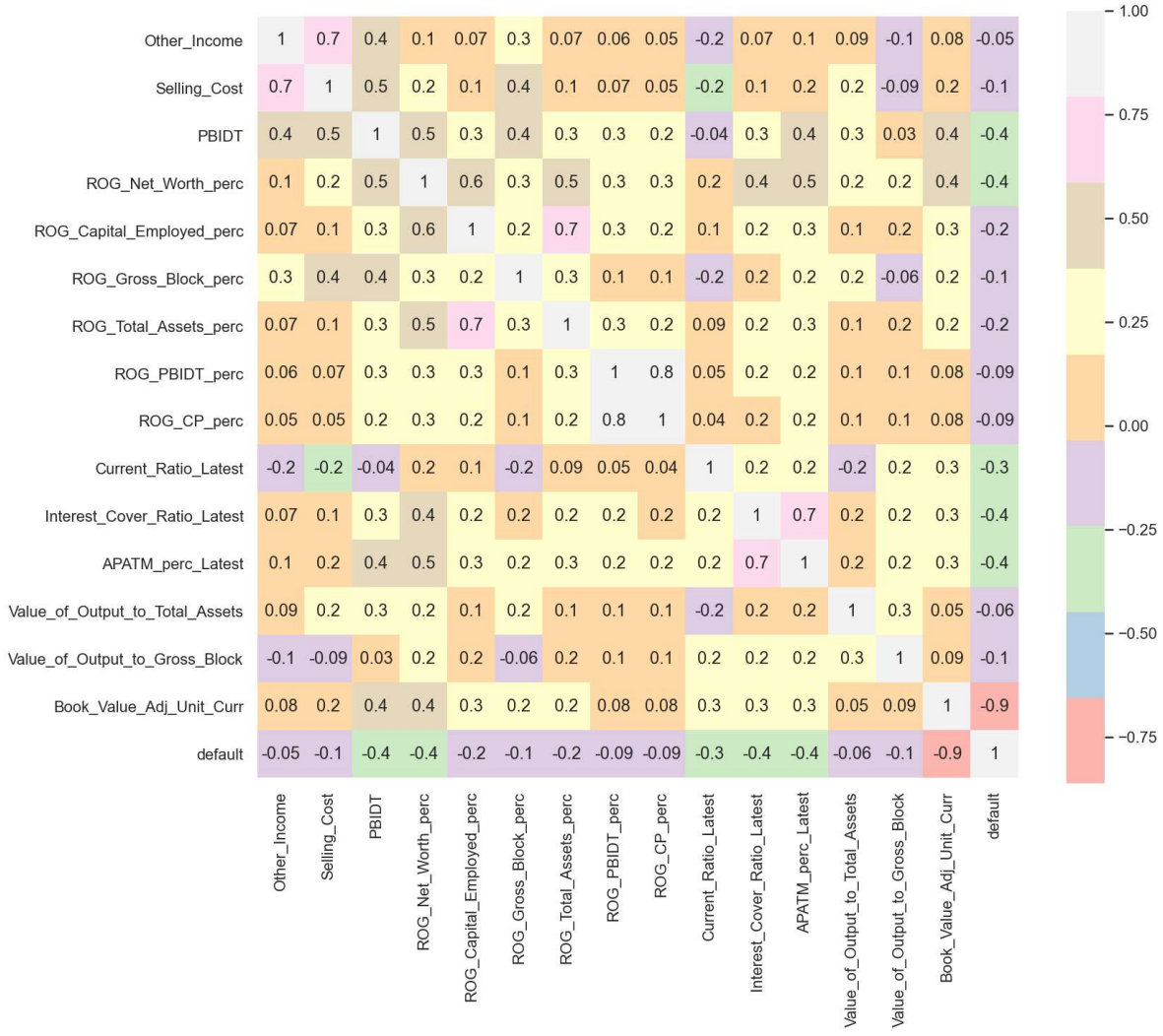
In [125]:

```
grp_sc.plot(kind='bar', figsize=(16,9))
plt.title('Effect of Variables on Default - Scaled Data', fontsize=17);
plt.savefig('vars_deault.jpg', bbox_inches='tight');
```



In [126]:

```
plt.figure(figsize=[12,10])
sns.heatmap(scaled_comp.corr(), annot=True, square=True, cmap='Pastel1', fmt='.1g')
plt.savefig('heatmap.jpg', bbox_inches='tight');
```



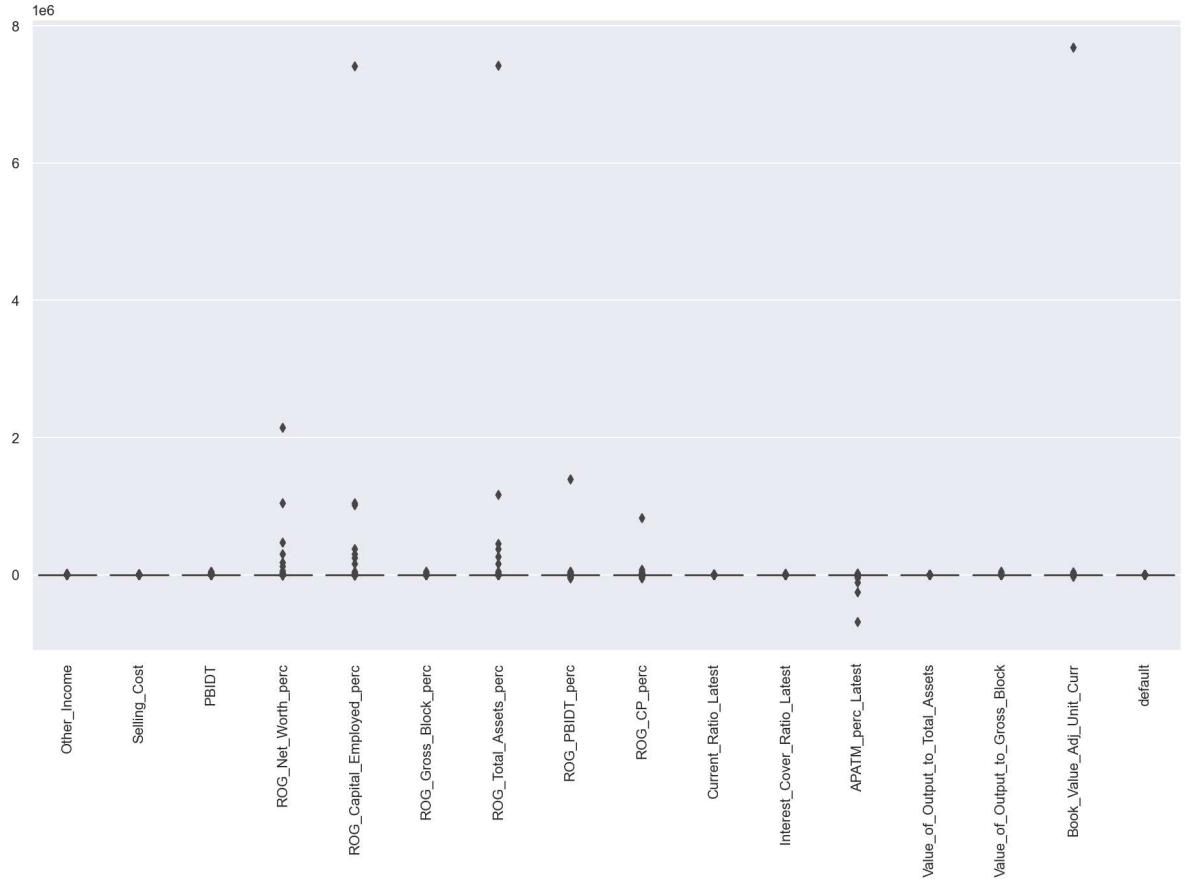
In [127]:

```
df = comp.drop('default', axis=1).copy()
cont=df.columns
fig, axes = plt.subplots(nrows=15,ncols=2);

fig.set_size_inches(15, 60);

for i in range(0, len(cont)):
    a = sns.distplot(df[cont[i]], ax=axes[i][0], color='b');
    a.set_title('{} Distribution'.format(cont[i]), fontsize=15);

    a.axvline(df[cont[i]].mean(), color='r', linewidth=1);
    a.axvline(df[cont[i]].median(), color='black', linestyle='dashed', linewidth=1);
    a.axvline(df[cont[i]].mode()[0], color='g', linestyle='dashed', linewidth=1);

    a = sns.boxplot(df[cont[i]], orient = "v" , ax=axes[i][1], palette='Pastel1');
    a.set_title('{} Distribution'.format(cont[i]), fontsize=15);
    plt.tight_layout();

plt.savefig('df_Distribution.jpg', bbox_inches='tight');
```

```
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing othe
r arguments without an explicit keyword will result in an error or misint
erpretation.
  warnings.warn(
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\_core.py:1303: UserWar
ning: Vertical orientation ignored with only `x` specified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
C:\Users\rahul\anaconda3\lib\site-packages\seaborn\distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in
a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level f
```

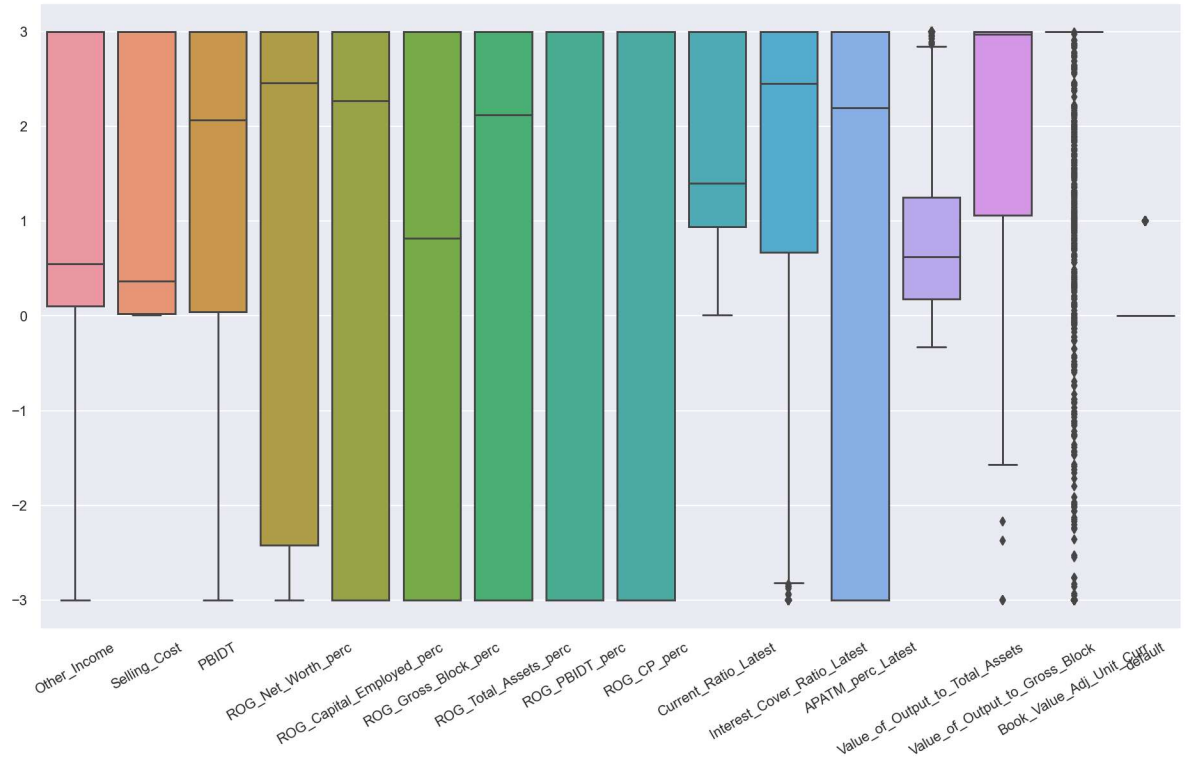
In [128]:

```
plt.figure(figsize=[16,9]);
sns.boxplot(data=comp);
plt.xticks(rotation=90);
plt.savefig('out_comp.jpg', bbox_inches='tight');
```



In [129]:

```
plt.figure(figsize=[16,9]);
sns.boxplot(data=scaled_comp);
plt.xticks(rotation=30);
plt.savefig('out_scaled_comp.jpg', bbox_inches='tight');
```



In [130]:

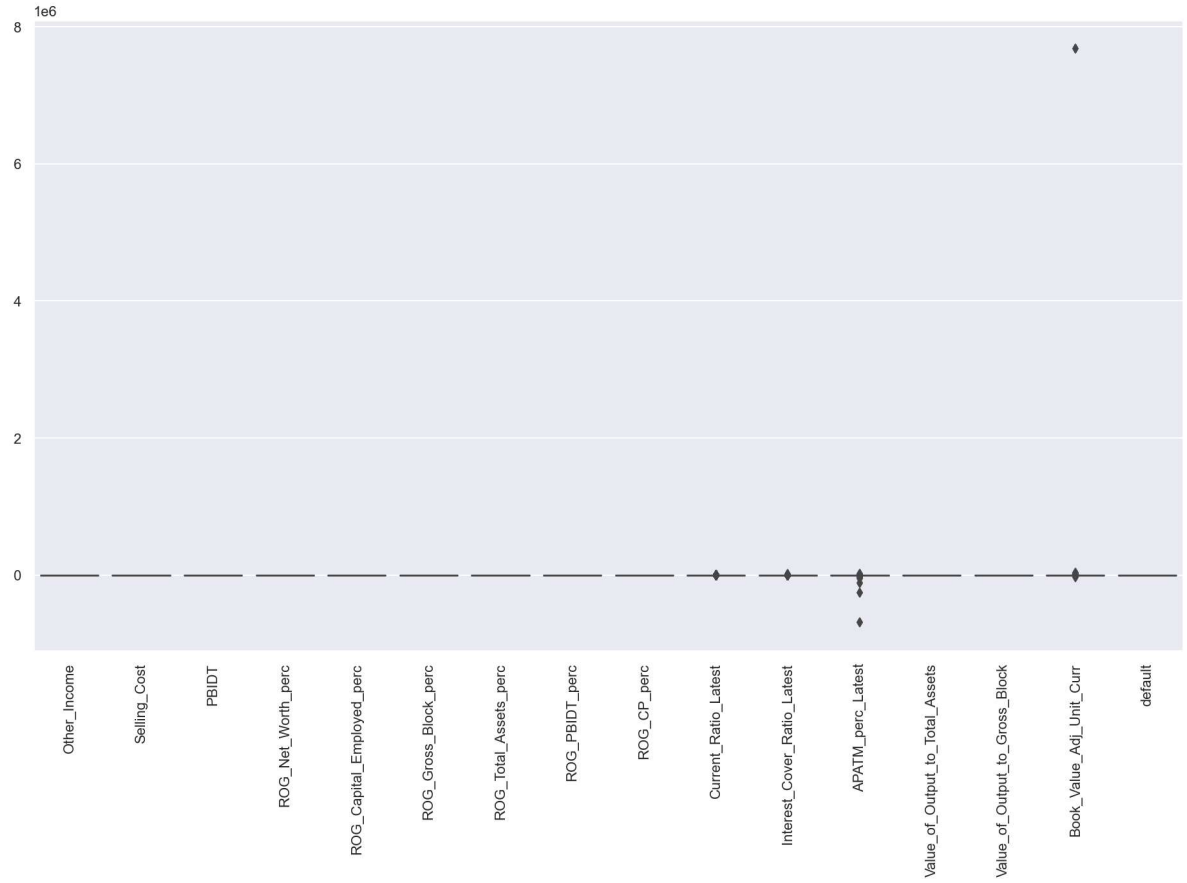
```
### <span style="font-family: Arial; font-weight:bold;font-size:1.2em;color:#9B59B6"> Outli

def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

In [131]:

```
comp_out = comp.copy()

for column in comp_out.columns:
    lr,ur=remove_outlier(comp_out[column])
    comp_out[column]=np.where(comp_out[column]>ur,ur,comp_out[column])
    comp_out[column]=np.where(comp_out[column]<lr,lr,comp_out[column])
```

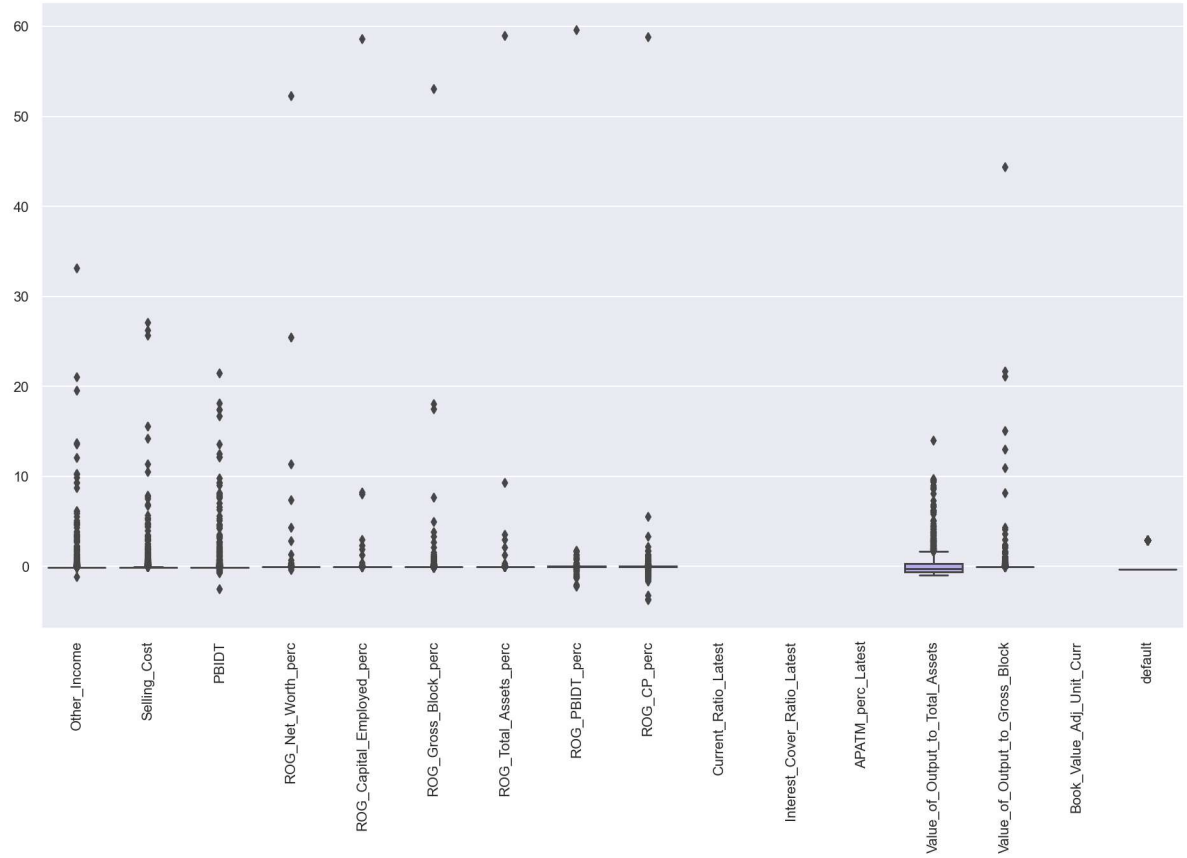
In [132]:

```
plt.figure(figsize=[16,9]);
sns.boxplot(data=comp_out);
plt.xticks(rotation=90);
plt.savefig('out_iqr_comp.jpg', bbox_inches='tight');
```



In [133]:

```
comp_scaled = comp.apply(zscore)

plt.figure(figsize=[16,9]);
sns.boxplot(data=comp_scaled);
plt.xticks(rotation=90);
plt.savefig('out_comp_scaled_before.jpg', bbox_inches='tight');
```

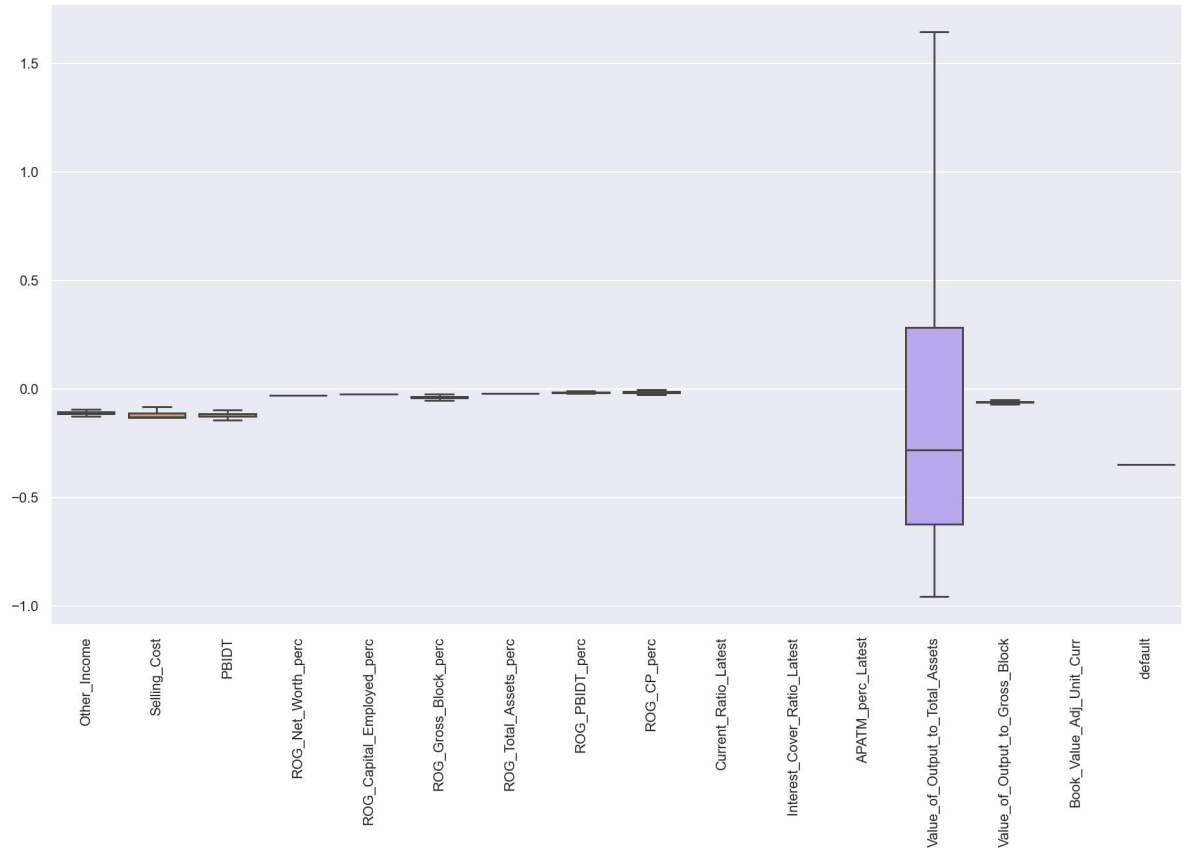


In [134]:

```
comp_out = comp_scaled.copy()

for column in comp_out.columns:
    lr,ur=remove_outlier(comp_out[column])
    comp_out[column]=np.where(comp_out[column]>ur,ur,comp_out[column])
    comp_out[column]=np.where(comp_out[column]<lr,lr,comp_out[column])
```

In [135]:

```
plt.figure(figsize=[16,9]);
sns.boxplot(data=comp_out);
plt.xticks(rotation=90);
plt.savefig('out_comp_scaled_after.jpg', bbox_inches='tight');
```



THANKS