In [1]:

```
import warnings
warnings.filterwarnings('ignore')

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

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


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    # Train test Split an


%matplotlib inline
sns.set(color_codes=True)
sns.set_palette('Accent_r')

%config InlineBackend.figure_format ='retina'

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)
```

---------- _ PROBLEM-1_

In [2]:

```
df = pd.read_excel('Company_Data2015.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.shape
```

Out[8]:

```
(3586, 67)
```

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	25
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

In [12]:

```
df.shape
```

Out[12]:

```
(3586, 65)
```

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 [13]:

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

Dropping variable 'Networth Next Year'

In [14]:

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

Out[14]:

	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	

In [15]:

```
df.shape
```

Out[15]:

```
(3586, 65)
```

Missing Value Treatment

In [16]:

```
df.isnull().sum()
```

Out[16]:

```
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 [17]:

```
print('Percent of Total Missing Values in the data =' ,(df.isna().sum().sum()*100/df.size).
```

```
Percent of Total Missing Values in the data = 0.05 %
```

* We note that there are not many missing values

* We'll come back to this

In [18]:

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

Out[18]:

```
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 [19]:

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

Out[19]:

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
```

In [20]:

```
((df.drop('default', axis=1)==0).sum().sum())*100/df.drop('default', axis=1).size
```

Out[20]:

```
15.111719185722253
```

. There are many variables with a lot of zero values (15.1%)

. 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 [21]:

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

Out[21]:

```
(3586, 56)
```

In [22]:

```
df.columns
```

Out[22]:

```
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 [23]:

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

Out[23]:

```
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[23]:

```
(3586, 55)
```

Out[23]:

```
(3586, 55)
```

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 [24]:

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

Out[24]:

```
(1004, 56)
```

In [25]:

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

Out[25]:

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

Out[25]:

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

Impute Null Values using KNN Imputer

```
    n_neighbors=10
```

In [26]:

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

Out[26]:

```
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 [27]:

```
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[27]:

```
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 IQR method

In [28]:

```
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 [29]:

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

In [30]:

```
for column in df_x.columns:
    lr,ur=remove_outlier(df_x[column])
    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])
```

In [31]:

```
df = pd.concat([df_x, df_y], axis = 1)
```

In [32]:

```
df.shape
```

Out[32]:

```
(3586, 56)
```

Check Multi-Colinearity using Variance Inflation Factor

In [33]:

```
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 [34]:

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

Out[34]:

	variables	VIF
5	Net_Sales	1155.479289
25	ROG_Gross_Sales_perc	990.784386
26	ROG_Net_Sales_perc	989.280569
4	Gross_Sales	692.791960
7	Value_Of_Output	475.547298
11	PBDT	118.320353
50	Total_Assets_to_Liabilities	106.812670
16	CP	104.422022
14	PAT	76.554676
13	PBT	75.937631
2	Capital_Employed	75.392095
8	Cost_of_Production	46.735330
40	PBIDTM_perc_Latest	42.402482
43	CPM_perc_Latest	37.862411
41	PBITM_perc_Latest	37.818037
53	Current_Assets	34.547182
10	PBIDT	32.400976
12	PBIT	29.315844
44	APATM_perc_Latest	26.146282
42	PBDTM_perc_Latest	25.555390
30	ROG_PBDT_perc	24.736311
49	Current_Liabilities_and_Provisions	22.790781
32	ROG_PBT_perc	19.608808
34	ROG_CP_perc	17.891621
17	Book_Value_Unit_Curr	17.545250
15	Adjusted_PAT	17.175283
33	ROG_PAT_perc	15.333147
29	ROG_PBIDT_perc	13.777720
54	Book_Value_Adj_Unit_Curr	13.765402
1	Networth	12.707238
51	Gross_Block	11.379502
31	ROG_PBIT_perc	11.323107
3	Total_Debt	10.488985

	variables	VIF
47	Value_of_Output_to_Total_Assets	9.711381
38	Total_Asset_Turnover_Ratio_Latest	9.141450
18	CEPS_annualised_Unit_Curr	6.573202
36	Fixed_Assets_Ratio_Latest	6.303578
48	Value_of_Output_to_Gross_Block	6.163473
52	Net_Working_Capital	5.970186
9	Selling_Cost	5.091993
0	Equity_Paid_Up	4.689543
6	Other_Income	4.488961
23	ROG_Capital_Employed_perc	4.004754
19	Cash_Flow_From_Operating_Activities	3.762776
28	ROG_Total_Assets_perc	3.492193
45	Debtors_Velocity_Days	3.222514
46	Creditors_Velocity_Days	3.155168
35	Current_Ratio_Latest	2.975223
22	ROG_Net_Worth_perc	2.846607
21	Cash_Flow_From_Financing_Activities	2.712140
20	Cash_Flow_From_Investing_Activities	2.409733
37	Debtors_Ratio_Latest	2.307813
39	Interest_Cover_Ratio_Latest	2.290381
27	ROG_Cost_of_Production_perc	2.027809
24	ROG_Gross_Block_perc	1.538953

In [35]:

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

Out[35]:

```
(3586, 55)
```

Out[35]:

```
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 [36]:

```
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 = 17.276147603988647
```

Check VIF again

In [37]:

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

Out[37]:

	variables	VIF
1	Total_Debt	4.894654
5	CEPS_annualised_Unit_Curr	4.441096
3	Selling_Cost	4.129128
10	ROG_Capital_Employed_perc	3.859918
0	Equity_Paid_Up	3.751747
2	Other_Income	3.644490
6	Cash_Flow_From_Operating_Activities	3.576169
26	Book_Value_Adj_Unit_Curr	3.363715
14	ROG_Total_Assets_perc	3.326385
4	Adjusted_PAT	3.279813
22	Debtors_Velocity_Days	3.068992
23	Creditors_Velocity_Days	3.054788
16	ROG_PBT_perc	3.027388
15	ROG_PBIDT_perc	2.935575
25	Net_Working_Capital	2.882657
9	ROG_Net_Worth_perc	2.770007
17	Current_Ratio_Latest	2.678811
8	Cash_Flow_From_Financing_Activities	2.662383
24	Value_of_Output_to_Gross_Block	2.358438
7	Cash_Flow_From_Investing_Activities	2.328282
18	Debtors_Ratio_Latest	2.190916
20	Interest_Cover_Ratio_Latest	2.171664
19	Total_Asset_Turnover_Ratio_Latest	2.122612
12	ROG_Net_Sales_perc	2.118504
13	ROG_Cost_of_Production_perc	1.992129
21	APATM_perc_Latest	1.525579
11	ROG_Gross_Block_perc	1.483308

In [38]:

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

Out[38]:

(3586, 27)

Out[38]:

	Equity_Paid_Up	Total_Debt	Other_Income	Selling_Cost	Adjusted_PAT	CEPS_annualised_Un
0	43.16875	220.71875	7.600	0.49200	-8.36	
1	43.16875	220.71875	9.602	11.90625	-8.36	
2	43.16875	220.71875	9.550	11.90625	-8.36	
3	43.16875	220.71875	9.602	3.34000	-8.36	
4	43.16875	220.71875	9.602	1.97000	-8.36	

- After dropping variables one by one with high VIF (>5)
- We are left with final 27 predictor variables for modelling

Scale the data using Standard Scaler

In [39]:

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

Out[39]:

(3586, 27)

Out[39]:

	Equity_Paid_Up	Total_Debt	Other_Income	Selling_Cost	Adjusted_PAT	CEPS_annualised_Un
0	2.083934	1.983979	1.386708	-0.576680	-1.677338	-1.
1	2.083934	1.983979	1.942253	1.951643	-1.677338	-0.
2	2.083934	1.983979	1.927823	1.951643	-1.677338	-1.
3	2.083934	1.983979	1.942253	0.054169	-1.677338	-1.
4	2.083934	1.983979	1.942253	-0.249294	-1.677338	-1.

Split the whole data into Train and Test 67-33

In [40]:

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

In [41]:

```
df1_train = pd.concat([X_train, y_train], axis=1)
```

1.8 Build a Random Forest Model on Train Dataset. Also showcase your model building approach

In [42]:

```
##Random Forest Model
from sklearn.ensemble import RandomForestClassifier
rfcl = RandomForestClassifier(n_estimators = 501)
rfcl = rfcl.fit(X_train, y_train)
rfcl
```

Out[42]:

```
RandomForestClassifier(n_estimators=501)
```

In [43]:

```
param_grid = {
    'max_depth': [7, 10],
    'max_features': [4, 6],
    'min_samples_leaf': [50, 100],
    'min_samples_split': [150, 300],
    'n_estimators': [301, 501]
}

rfcl = RandomForestClassifier()

grid_search = GridSearchCV(estimator = rfcl, param_grid = param_grid, cv = 5)
```

In [44]:

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

Out[44]:

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [7, 10], 'max_features': [4, 6],
                         'min_samples_leaf': [50, 100],
                         'min_samples_split': [150, 300],
                         'n_estimators': [301, 501]})
```

In [45]:

```
grid_search.best_params_
```

Out[45]:

```
{'max_depth': 10,
 'max_features': 6,
 'min_samples_leaf': 50,
 'min_samples_split': 150,
 'n_estimators': 501}
```

In [46]:

```
best_grid2 = grid_search.best_estimator_
```

In [47]:

```
ytrain_predict2 = best_grid2.predict(X_train)

ytest_predict2 = best_grid2.predict(X_test)
```

In [48]:

```
# Variable Importance
print (pd.DataFrame(best_grid2.feature_importances_, columns = ["Imp"], index = X_train.col
```

	Imp
Book_Value_Adj_Unit_Curr	0.483272
Current_Ratio_Latest	0.132213
ROG_Net_Worth_perc	0.083394
CEPS_annualised_Unit_Curr	0.064441
APATM_perc_Latest	0.053192
Adjusted_PAT	0.039868
Net_Working_Capital	0.038775
Interest_Cover_Ratio_Latest	0.036824
ROG_Capital_Employed_perc	0.025922
ROG_Total_Assets_perc	0.010483
Total_Asset_Turnover_Ratio_Latest	0.010478
Cash_Flow_From_Investing_Activities	0.002942
Total_Debt	0.002584
ROG_PBT_perc	0.002423
Creditors_Velocity_Days	0.002372
ROG_PBIDT_perc	0.002101
ROG_Cost_of_Production_perc	0.002046
Value_of_Output_to_Gross_Block	0.001980
Cash_Flow_From_Operating_Activities	0.000945
ROG_Net_Sales_perc	0.000804
Selling_Cost	0.000528
Other_Income	0.000503
Debtors_Velocity_Days	0.000457
Cash_Flow_From_Financing_Activities	0.000426
ROG_Gross_Block_perc	0.000410
Debtors_Ratio_Latest	0.000396
Equity_Paid_Up	0.000223

In [49]:

```
ytrain_predict2 = best_grid2.predict(X_train)

ytest_predict2 = best_grid2.predict(X_test)
```

In [50]:

```
print(classification_report(y_train, ytrain_predict2))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.98      2157
         1.0       0.93      0.79      0.85       245

    accuracy                           0.97      2402
   macro avg       0.95      0.89      0.92      2402
weighted avg       0.97      0.97      0.97      2402
```

In [51]:

```
confusion_matrix(y_train, ytrain_predict2)
```

Out[51]:

```
array([[2143,   14],
       [  52,  193]], dtype=int64)
```

In [52]:

```
# AUC and ROC for the training data

# predict probabilities
probs = best_grid2.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
rf_train_auc = roc_auc_score(y_train, probs)
print('AUC: %.3f' % rf_train_auc)
# calculate roc curve
rf_train_fpr,rf_train_tpr, rf_train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(rf_train_fpr,rf_train_tpr, marker='.')
# show the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.show()
```

```
AUC: 0.992
```

Out[52]:

```
[<matplotlib.lines.Line2D at 0x16fdd680970>]
```

Out[52]:

```
[<matplotlib.lines.Line2D at 0x16fdd680dc0>]
```

Out[52]:

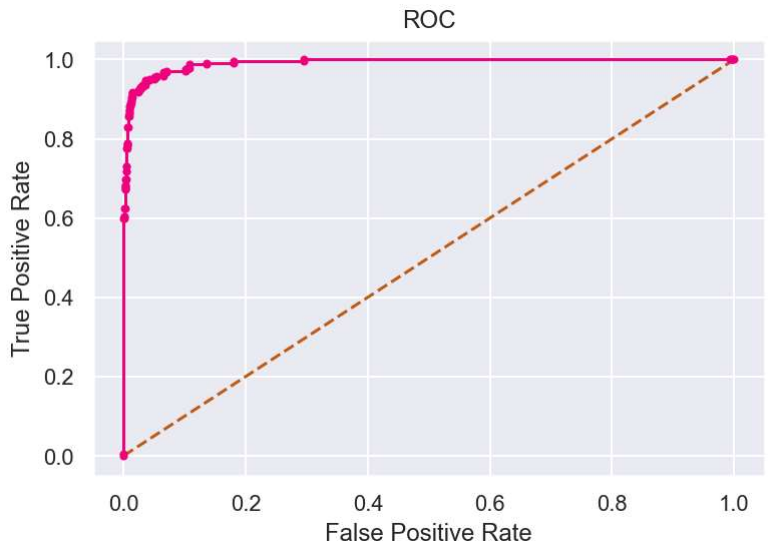
```
Text(0.5, 0, 'False Positive Rate')
```

Out[52]:

```
Text(0, 0.5, 'True Positive Rate')
```

Out[52]:

```
Text(0.5, 1.0, 'ROC')
```



1.9 Validate the Random Forest Model on test Dataset and state the performance matrices. Also state interpretation from the model

In [53]:

```
ytest_predict2 = best_grid2.predict(X_test)
```

In [54]:

```
print(classification_report(y_test, ytest_predict2))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99      1042
         1.0       0.93      0.88      0.91       142

    accuracy                           0.98      1184
   macro avg       0.96      0.94      0.95      1184
weighted avg       0.98      0.98      0.98      1184
```

In [55]:

```
confusion_matrix(y_test, ytest_predict2)
```

Out[55]:

```
array([[1033,    9],
       [  17,  125]], dtype=int64)
```

In [56]:

```
# AUC and ROC for the test data


# predict probabilities
probs = best_grid2.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]

# calculate AUC
rf_test_auc = roc_auc_score(y_test, probs)
print('AUC: %.3f' % rf_test_auc)

# calculate roc curve
rf_test_fpr, rf_test_tpr, thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')

# plot the roc curve for the model
plt.plot(rf_test_fpr, rf_test_tpr, marker='.')
# show the plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.show()
```

```
AUC: 0.992
```

Out[56]:

```
[<matplotlib.lines.Line2D at 0x16fded7b3d0>]
```

Out[56]:

```
[<matplotlib.lines.Line2D at 0x16fded7b7f0>]
```

Out[56]:

```
Text(0.5, 0, 'False Positive Rate')
```

Out[56]:

```
Text(0, 0.5, 'True Positive Rate')
```

Out[56]:

```
Text(0.5, 1.0, 'ROC')
```

ROC

1.10 Build a LDA Model on Train Dataset. Also showcase your model building approach

In [57]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [58]:

```
#Build LDA Model
model2 = LinearDiscriminantAnalysis()
model2.fit(X_train,y_train)
```

Out[58]:

```
LinearDiscriminantAnalysis()
```

In [59]:

```
# Training Data Class Prediction with a cut-off value of 0.5
ytrain_predict2 = model2.predict(X_train)

# Test Data Class Prediction with a cut-off value of 0.5
ytest_predict2 = model2.predict(X_test)
```

In [60]:

```
### Getting the Predicted Classes and Probs
ytest_predict2_prob=model2.predict_proba(X_test)
pd.DataFrame(ytest_predict2_prob).head()
```

Out[60]:

	0	1
0	0.860517	0.139483
1	0.917397	0.082603
2	0.990656	0.009344
3	0.509220	0.490780
4	0.975081	0.024919

In [61]:

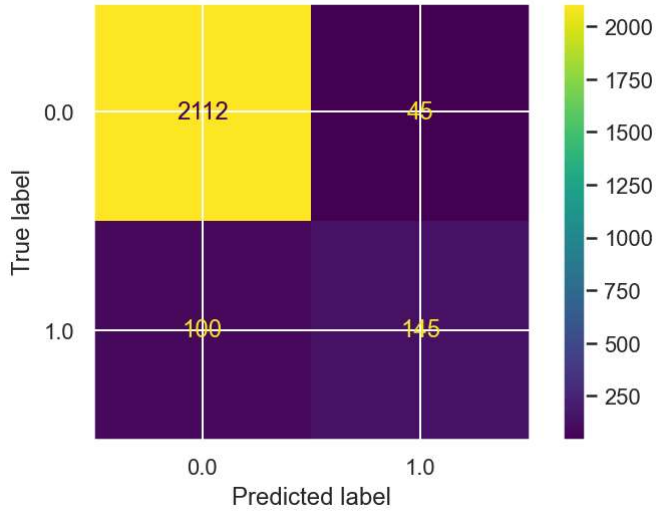
```
from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_matrix,
confusion_matrix(y_train, ytrain_predict2)
```

Out[61]:

```
array([[2112,   45],
       [ 100,  145]], dtype=int64)
```

In [62]:

```
##Confusion matrix on train data
plot_confusion_matrix(model2,X_train,y_train);
```



In [63]:

```
# Accuracy - Training Data
LDA_train_acc= model2.score(X_train, y_train)
print('LDA_train_acc', LDA_train_acc )
```

```
LDA_train_acc 0.9396336386344712
```

In [64]:

```
print(classification_report(y_train, ytrain_predict2))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.98      0.97      2157
         1.0       0.76      0.59      0.67       245

    accuracy                           0.94      2402
   macro avg       0.86      0.79      0.82      2402
weighted avg       0.94      0.94      0.94      2402
```

In [65]:

```
# AUC and ROC for the training data
# predict probabilities
probs = model2.predict_proba(X_train)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
LDA_train_auc = roc_auc_score(y_train, probs)
print('LDA_train_AUC: %.3f' % LDA_train_auc)
# calculate roc curve
LDA_train_fpr, LDA_train_tpr, LDA_train_thresholds = roc_curve(y_train, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(LDA_train_fpr, LDA_train_tpr);
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LDA_ROC_train_data')
plt.show()
```

```
LDA_train_AUC: 0.953
```

Out[65]:

```
[<matplotlib.lines.Line2D at 0x16fdd45de50>]
```

Out[65]:

```
[<matplotlib.lines.Line2D at 0x16fdd50c250>]
```

Out[65]:

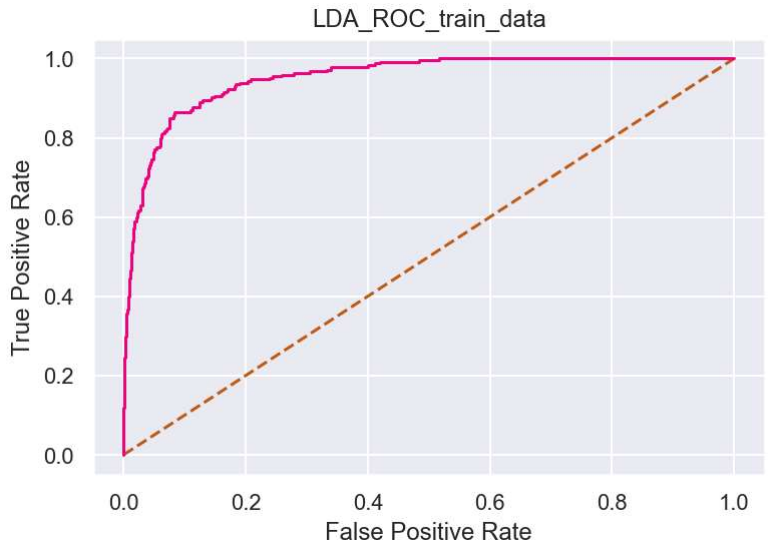
```
Text(0.5, 0, 'False Positive Rate')
```

Out[65]:

```
Text(0, 0.5, 'True Positive Rate')
```

Out[65]:

```
Text(0.5, 1.0, 'LDA_ROC_train_data')
```



1.11 Validate the LDA Model on test Dataset and state the performance matrices. Also state interpretation from the model

In [66]:

```
# Test Data Class Prediction with a cut-off value of 0.5
ytest_predict2 = model2.predict(X_test)
```

In [67]:

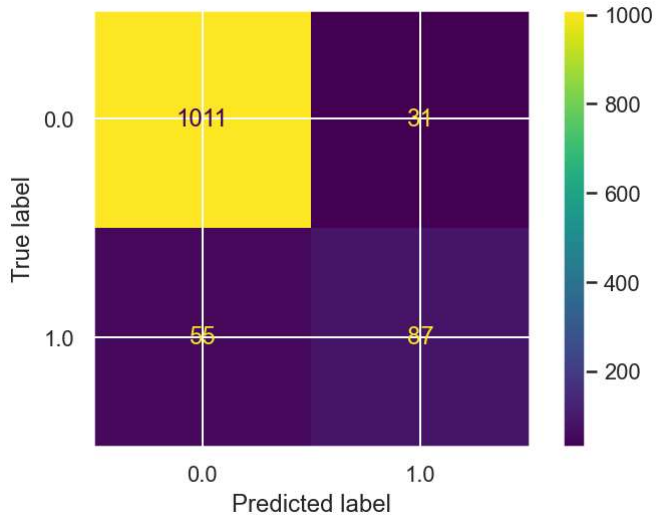
```
##Confusion matrix on test data
confusion_matrix(y_test, ytest_predict2)
```

Out[67]:

```
array([[1011,   31],
       [  55,   87]], dtype=int64)
```

In [68]:

```
plot_confusion_matrix(model2,X_test,y_test);
```



In [69]:

```
# Accuracy - Test Data
LDA_test_acc =model2.score(X_test, y_test)
print('LDA_test_acc', LDA_test_acc )
```

```
LDA_test_acc 0.9273648648648649
```

In [70]:

```
print(classification_report(y_test, ytest_predict2))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.97      0.96      1042
         1.0       0.74      0.61      0.67       142

    accuracy                           0.93      1184
   macro avg       0.84      0.79      0.81      1184
weighted avg       0.92      0.93      0.92      1184
```

In [71]:

```
# predict probabilities
probs = model2.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
LDA_test_auc = roc_auc_score(y_test, probs)
print('LDA_test_AUC: %.3f' % LDA_test_auc)
# calculate roc curve
LDA_test_fpr, LDA_test_tpr, LDA_test_thresholds = roc_curve(y_test, probs)
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(LDA_test_fpr, LDA_test_tpr);
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LDA_ROC_test_data')
plt.show()
```

```
LDA_test_AUC: 0.948
```

Out[71]:

```
[<matplotlib.lines.Line2D at 0x16fdd5783d0>]
```

Out[71]:

```
[<matplotlib.lines.Line2D at 0x16fdd578790>]
```

Out[71]:

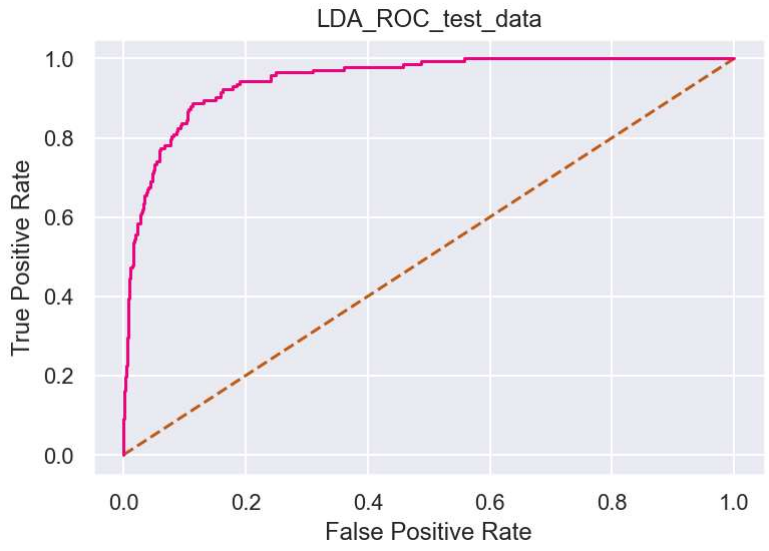
```
Text(0.5, 0, 'False Positive Rate')
```

Out[71]:

```
Text(0, 0.5, 'True Positive Rate')
```

Out[71]:

```
Text(0.5, 1.0, 'LDA_ROC_test_data')
```



1.12 Compare the performances of Logistics, Radom Forest and LDA models (include ROC Curve)

ROC Curve for the 2 models on the Train data

In [72]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(LDA_train_fpr, LDA_train_tpr,color='red',label="LDA")
plt.plot(rf_train_fpr,rf_train_tpr,color='green',label="RF")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower right')
```

Out[72]:

```
[<matplotlib.lines.Line2D at 0x16fdd607fd0>]
```

Out[72]:

```
[<matplotlib.lines.Line2D at 0x16fdd61eb20>]
```

Out[72]:

```
[<matplotlib.lines.Line2D at 0x16fded69c40>]
```

Out[72]:

```
Text(0.5, 0, 'False Positive Rate')
```

Out[72]:

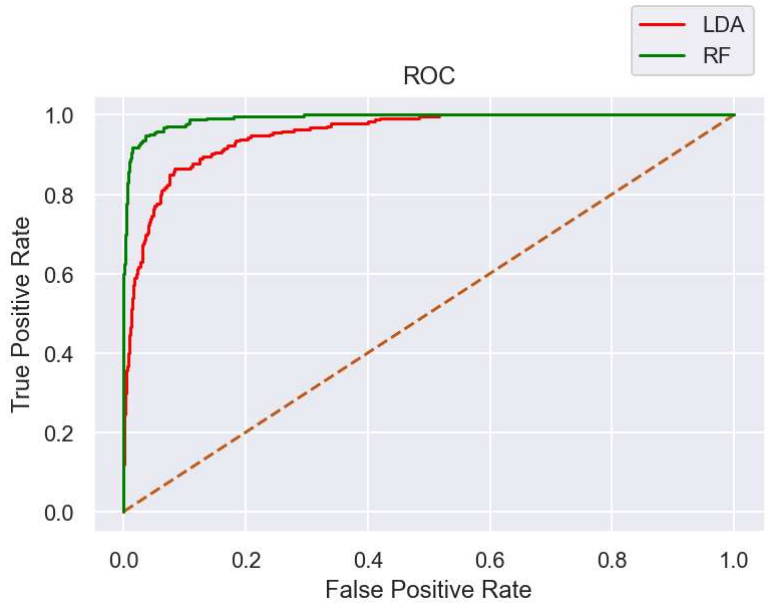
```
Text(0, 0.5, 'True Positive Rate')
```

Out[72]:

```
Text(0.5, 1.0, 'ROC')
```

Out[72]:

```
<matplotlib.legend.Legend at 0x16fdd61eb80>
```



ROC Curve for the 2 models on the Test data

In [73]:

```
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(LDA_test_fpr, LDA_test_tpr,color='red',label="LDA")
plt.plot(rf_test_fpr,rf_test_tpr,color='green',label="RF")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc='lower right')
```

Out[73]:

```
[<matplotlib.lines.Line2D at 0x16fdd2d60d0>]
```

Out[73]:

```
[<matplotlib.lines.Line2D at 0x16fdcee4f40>]
```

Out[73]:

```
[<matplotlib.lines.Line2D at 0x16fdd3a2580>]
```

Out[73]:

```
Text(0.5, 0, 'False Positive Rate')
```

Out[73]:

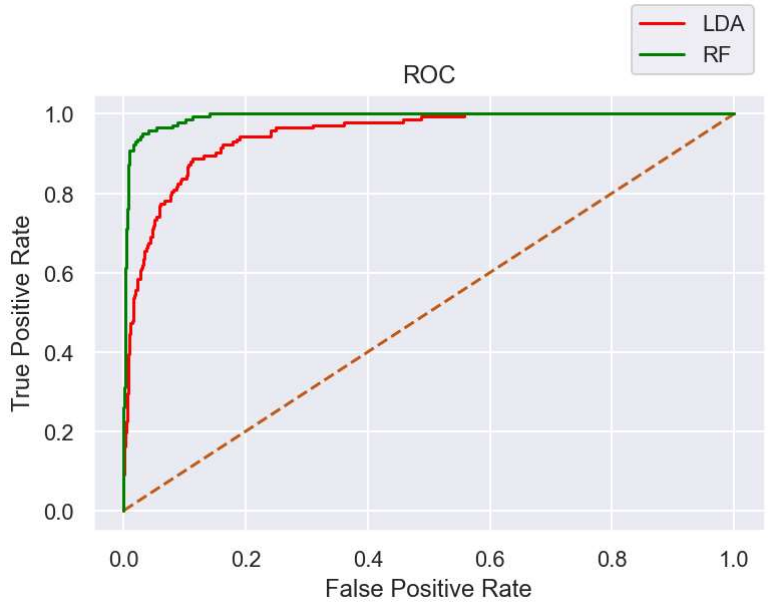
```
Text(0, 0.5, 'True Positive Rate')
```

Out[73]:

```
Text(0.5, 1.0, 'ROC')
```

Out[73]:

```
<matplotlib.legend.Legend at 0x16fdcee4e80>
```



1.13 State Recommendations from the above models

In [ ]:

```
```

In [ ]:

```
```

---------- _ PROBLEM-2

MARKET RISK ANALYSIS-PROBLEM STATEMENT

Q 2.1 Draw Stock Price Graph(Stock Price vs Time) for any 2 given stocks with inference

In [74]:

```
ddf=pd.read_csv("Market+Risk+Dataset (3).csv")
```

In [75]:

```
ddf.head()
```

Out[75]:

	Date	Infosys	Indian Hotel	Mahindra & Mahindra	Axis Bank	SAIL	Shree Cement	Sun Pharma	Jindal Steel	Idea Vodafone	Jet Airways
0	31-03-2014	264	69	455	263	68	5543	555	298	83	278
1	07-04-2014	257	68	458	276	70	5728	610	279	84	303
2	14-04-2014	254	68	454	270	68	5649	607	279	83	280
3	21-04-2014	253	68	488	283	68	5692	604	274	83	282
4	28-04-2014	256	65	482	282	63	5582	611	238	79	243

In [76]:

```
ddf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 314 entries, 0 to 313
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Date                 314 non-null    object
 1   Infosys              314 non-null    int64 
 2   Indian Hotel         314 non-null    int64 
 3   Mahindra & Mahindra  314 non-null    int64 
 4   Axis Bank            314 non-null    int64 
 5   SAIL                 314 non-null    int64 
 6   Shree Cement         314 non-null    int64 
 7   Sun Pharma           314 non-null    int64 
 8   Jindal Steel         314 non-null    int64 
 9   Idea Vodafone        314 non-null    int64 
 10  Jet Airways          314 non-null    int64 
dtypes: int64(10), object(1)
memory usage: 27.1+ KB
```

In [77]:

```
ddf.isnull().sum()
```

Out[77]:

```
Date                 0
Infosys              0
Indian Hotel         0
Mahindra & Mahindra  0
Axis Bank            0
SAIL                 0
Shree Cement         0
Sun Pharma           0
Jindal Steel         0
Idea Vodafone        0
Jet Airways          0
dtype: int64
```

In [78]:

```
ddf.columns
```

Out[78]:

```
Index(['Date', 'Infosys', 'Indian Hotel', 'Mahindra & Mahindra', 'Axis Ban
k', 'SAIL', 'Shree Cement', 'Sun Pharma', 'Jindal Steel', 'Idea Vodafone',
'Jet Airways'], dtype='object')
```

In [79]:

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

In [80]:

```
ddf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 314 entries, 0 to 313
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Date                 314 non-null    object
 1   Infosys              314 non-null    int64 
 2   Indian_Hotel         314 non-null    int64 
 3   Mahindra_&_Mahindra  314 non-null    int64 
 4   Axis_Bank            314 non-null    int64 
 5   SAIL                 314 non-null    int64 
 6   Shree_Cement         314 non-null    int64 
 7   Sun_Pharma           314 non-null    int64 
 8   Jindal_Steel         314 non-null    int64 
 9   Idea_Vodafone        314 non-null    int64 
 10  Jet_Airways          314 non-null    int64 
dtypes: int64(10), object(1)
memory usage: 27.1+ KB
```

In [81]:

```
ddf.shape
```

Out[81]:

```
(314, 11)
```

In [82]:

```
ddf.size
```

Out[82]:

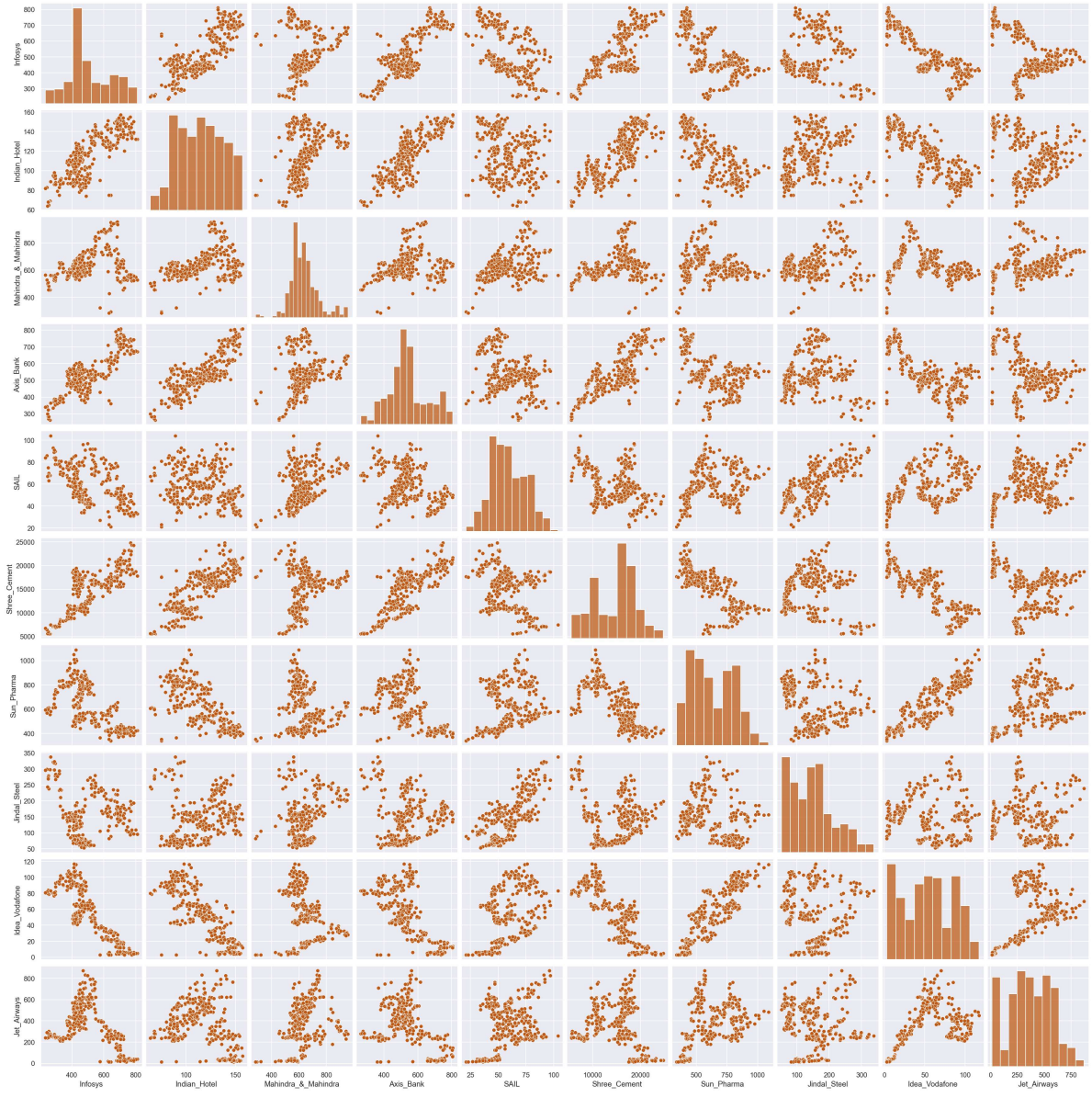
```
3454
```

In [83]:

```
sns.pairplot(ddf)
plt.show()
```

Out[83]:

```
<seaborn.axisgrid.PairGrid at 0x16fded20970>
```



In [84]:

```
ddf['Date'] = pd.to_datetime(ddf['Date'])
```

In [85]:

```
ddf['year'] = ddf['Date'].dt.year
ddf['month'] = ddf['Date'].dt.month
```

In [86]:

```
plt.scatter(ddf['year'],ddf['Infosys'], c ="Red")
plt.ylabel('Infosys')
plt.xlabel('year')
plt.title('Scatter plot')
```

Out[86]:

```
<matplotlib.collections.PathCollection at 0x16fe93db7f0>
```

Out[86]:

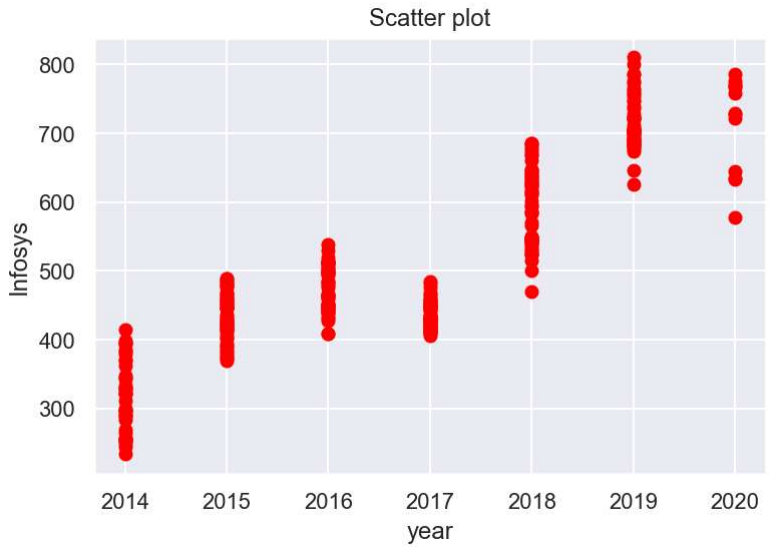
```
Text(0, 0.5, 'Infosys')
```

Out[86]:

```
Text(0.5, 0, 'year')
```

Out[86]:

```
Text(0.5, 1.0, 'Scatter plot')
```

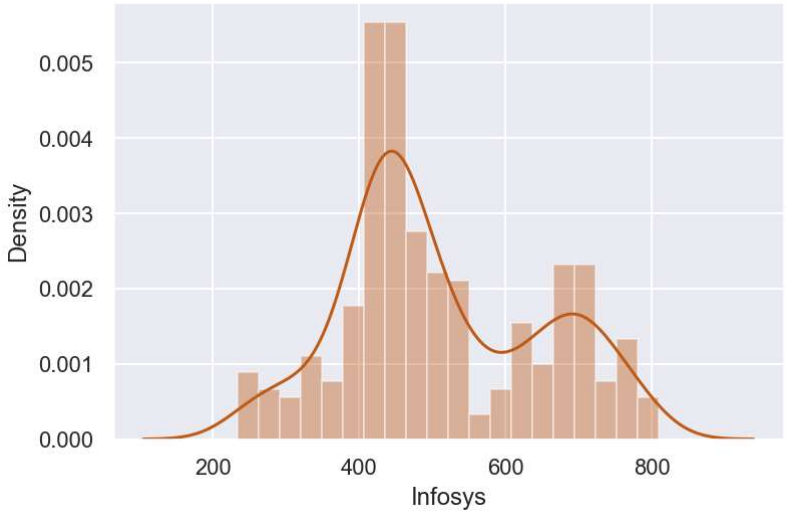


In [87]:

```
sns.distplot(ddf.Infosys,bins=20)
```

Out[87]:

```
<AxesSubplot:xlabel='Infosys', ylabel='Density'>
```



In [88]:

```
plt.scatter(ddf['year'],ddf['Sun_Pharma'], c ="blue")
plt.ylabel('Sun_Pharma')
plt.xlabel('year')
plt.title('Scatter plot')
```

Out[88]:

```
<matplotlib.collections.PathCollection at 0x16fe61bfb20>
```

Out[88]:

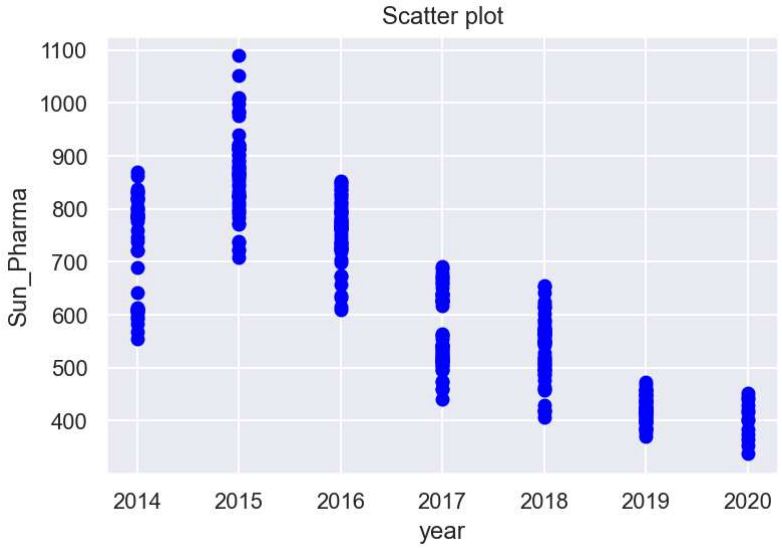
```
Text(0, 0.5, 'Sun_Pharma')
```

Out[88]:

```
Text(0.5, 0, 'year')
```

Out[88]:

```
Text(0.5, 1.0, 'Scatter plot')
```

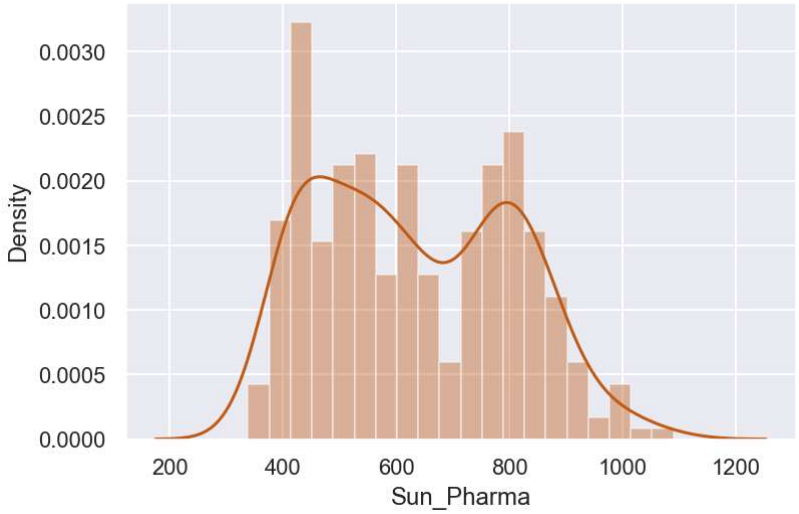


In [89]:

```
sns.distplot(ddf.Sun_Pharma,bins=20)
```

Out[89]:

```
<AxesSubplot:xlabel='Sun_Pharma', ylabel='Density'>
```



Q 2.2 Calculate Returns for all stocks with inference

In [90]:

```
ddf.corr()
```

Out[90]:

	Infosys	Indian_Hotel	Mahindra_&_Mahindra	Axis_Bank	SAIL	Sh
Infosys	1.000000	0.795461	0.195053	0.837789	-0.479648	
Indian_Hotel	0.795461	1.000000	0.410030	0.829801	-0.156767	
Mahindra_&_Mahindra	0.195053	0.410030	1.000000	0.193252	0.400131	
Axis_Bank	0.837789	0.829801	0.193252	1.000000	-0.318087	
SAIL	-0.479648	-0.156767	0.400131	-0.318087	1.000000	
Shree_Cement	0.757489	0.790911	0.239813	0.761522	-0.358722	
Sun_Pharma	-0.645646	-0.680355	-0.127840	-0.523302	0.200852	
Jindal_Steel	-0.095364	0.145155	0.263296	-0.058390	0.736362	
Idea_Vodafone	-0.850325	-0.775013	-0.160497	-0.676477	0.451571	
Jet_Airways	-0.483300	-0.224433	0.315606	-0.437216	0.311574	
year	0.888543	0.820601	0.218069	0.759239	-0.375533	
month	-0.040841	-0.030857	0.071205	-0.012705	-0.015868	

In [91]:

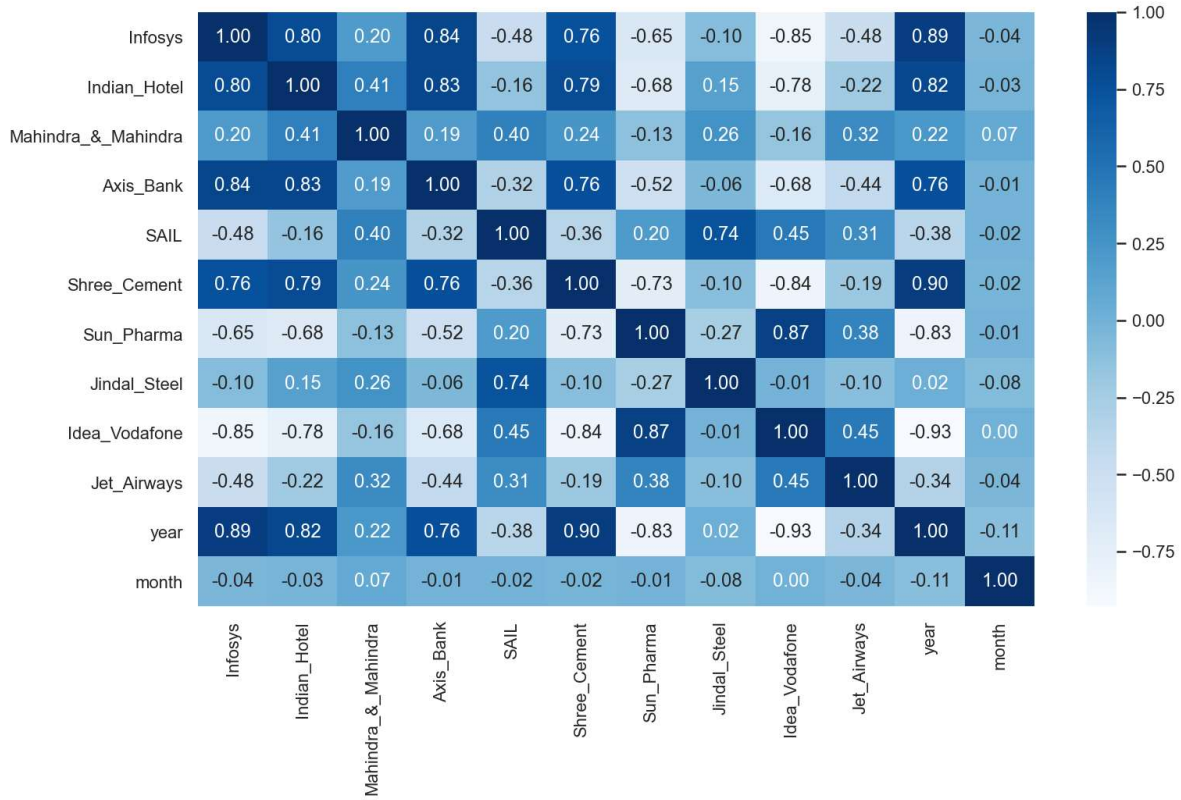
```
plt.figure(figsize=(12,7))
sns.heatmap(ddf.corr(), annot=True, fmt='.2f', cmap='Blues')
plt.show()
```

Out[91]:

```
<Figure size 864x504 with 0 Axes>
```

Out[91]:

```
<AxesSubplot:>
```



In [92]:

```
Ret_Info= ddf['Infosys'].pct_change()
Ret_Info.head(5)
```

Out[92]:

```
0         NaN
1   -0.026515
2   -0.011673
3   -0.003937
4    0.011858
Name: Infosys, dtype: float64
```

In [93]:

```
Ret_IndHotel= ddf['Indian_Hotel'].pct_change()
Ret_IndHotel.head()
```

Out[93]:

```
0         NaN
1   -0.014493
2    0.000000
3    0.000000
4   -0.044118
Name: Indian_Hotel, dtype: float64
```

In [94]:

```
Ret_MM= ddf['Mahindra_&_Mahindra'].pct_change()
Ret_MM.head()
```

Out[94]:

```
0         NaN
1    0.006593
2   -0.008734
3    0.074890
4   -0.012295
Name: Mahindra_&_Mahindra, dtype: float64
```

In [95]:

```
Ret_Axis= ddf['Axis_Bank'].pct_change()
Ret_Axis.head()
```

Out[95]:

```
0         NaN
1    0.049430
2   -0.021739
3    0.048148
4   -0.003534
Name: Axis_Bank, dtype: float64
```

In [96]:

```
Ret_SAIL= ddf['SAIL'].pct_change()
Ret_SAIL.head()
```

Out[96]:

```
0         NaN
1    0.029412
2   -0.028571
3    0.000000
4   -0.073529
Name: SAIL, dtype: float64
```

In [97]:

```
Ret_ShreeC= ddf['Shree_Cement'].pct_change()
Ret_ShreeC.head()
```

Out[97]:

```
0         NaN
1    0.033375
2   -0.013792
3    0.007612
4   -0.019325
Name: Shree_Cement, dtype: float64
```

In [98]:

```
Ret_SunP= ddf['Sun_Pharma'].pct_change()
Ret_SunP.head()
```

Out[98]:

```
0         NaN
1    0.099099
2   -0.004918
3   -0.004942
4    0.011589
Name: Sun_Pharma, dtype: float64
```

In [99]:

```
Ret_JSteel= ddf['Jindal_Steel'].pct_change()
Ret_JSteel.head()
```

Out[99]:

```
0         NaN
1   -0.063758
2    0.000000
3   -0.017921
4   -0.131387
Name: Jindal_Steel, dtype: float64
```

In [100]:

```
Ret_VI= ddf['Idea_Vodafone'].pct_change()
Ret_VI.head()
```

Out[100]:

```
0         NaN
1    0.012048
2   -0.011905
3    0.000000
4   -0.048193
Name: Idea_Vodafone, dtype: float64
```

In [101]:

```
Ret_JetAir= ddf['Jet_Airways'].pct_change()
Ret_JetAir.head()
```

Out[101]:

```
0         NaN
1    0.089928
2   -0.075908
3    0.007143
4   -0.138298
Name: Jet_Airways, dtype: float64
```

In [102]:

```
ddf.describe().T
```

Out[102]:

	count	mean	std	min	25%	50%	75%
Infosys	314.0	511.340764	135.952051	234.0	424.00	466.5	630.75
Indian_Hotel	314.0	114.560510	22.509732	64.0	96.00	115.0	134.00
Mahindra_&_Mahindra	314.0	636.678344	102.879975	284.0	572.00	625.0	678.00
Axis_Bank	314.0	540.742038	115.835569	263.0	470.50	528.0	605.25
SAIL	314.0	59.095541	15.810493	21.0	47.00	57.0	71.75
Shree_Cement	314.0	14806.410828	4288.275085	5543.0	10952.25	16018.5	17773.25
Sun_Pharma	314.0	633.468153	171.855893	338.0	478.50	614.0	785.00
Jindal_Steel	314.0	147.627389	65.879195	53.0	88.25	142.5	182.75
Idea_Vodafone	314.0	53.713376	31.248985	3.0	25.25	53.0	82.00
Jet_Airways	314.0	372.659236	202.262668	14.0	243.25	376.0	534.00
year	314.0	2016.745223	1.767309	2014.0	2015.00	2017.0	2018.00
month	314.0	6.503185	3.452321	1.0	4.00	7.0	9.00

Q 2.3 Calculate Stock Means and Standard Deviation for

all stocks with inference

Calculating Stock Mean

In [103]:

```
print('Infosys : ',Ret_Info.mean())
print('Indian_Hotel: ', Ret_IndHotel.mean())
print('Mahindra_&_Mahindra: ',Ret_MM.mean())
print('Axis_Bank: ', Ret_Axis.mean())
print('SAIL: ',Ret_SAIL.mean())
print('Shree_Cement: ', Ret_ShreeC.mean())
print('Sun_Pharma : ',Ret_SunP.mean())
print('Jindal_Steel: ', Ret_JSteel.mean())
print('Idea_Vodafone: ',Ret_VI.mean())
print('Jet_Airways: ', Ret_JetAir.mean())
```

```
Infosys :  0.003409204468116012
Indian_Hotel:  0.0013687082763244721
Mahindra_&_Mahindra:  -0.0007171835188143648
Axis_Bank:  0.002201418399343187
SAIL:  -0.0015214399817054948
Shree_Cement:  0.004486554973806194
Sun_Pharma :  -0.0004505358584624121
Jindal_Steel:  -0.0013127367898399368
Idea_Vodafone:  -0.005079851359115386
Jet_Airways:  -0.00480476811531217
```

Calculating Stock Standard Deviation

In [104]:

```
print('Infosys : ',Ret_Info.std())
print('Indian_Hotel: ', Ret_IndHotel.std())
print('Mahindra_&_Mahindra: ',Ret_MM.std())
print('Axis_Bank: ', Ret_Axis.std())
print('SAIL: ',Ret_SAIL.std())
print('Shree_Cement: ', Ret_ShreeC.std())
print('Sun_Pharma : ',Ret_SunP.std())
print('Jindal_Steel: ', Ret_JSteel.std())
print('Idea_Vodafone: ',Ret_VI.std())
print('Jet_Airways: ', Ret_JetAir.std())
```

```
Infosys :  0.03491001488350448
Indian_Hotel:  0.046913781970161064
Mahindra_&_Mahindra:  0.03902878858620297
Axis_Bank:  0.0450178572644669
SAIL:  0.0628987033729592
Shree_Cement:  0.04019204469672036
Sun_Pharma :  0.04461570690952949
Jindal_Steel:  0.07507866324366545
Idea_Vodafone:  0.11007500422768263
Jet_Airways:  0.09647289932418758
```

Q 2.4 Draw a plot of Stock Means vs Standard Deviation and state your inference

In [105]:

```
df2=pd.read_csv("compnydata.csv")
```

In [106]:

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Name com    10 non-null     object 
 1   Average     10 non-null     float64
 2   Volatility  10 non-null     float64
dtypes: float64(2), object(1)
memory usage: 368.0+ bytes
```

In [107]:

```
plt.scatter(df2['Average'],df2['Volatility'], c ="Blue")
plt.ylabel('Average')
plt.xlabel('Volatility')
plt.title('Scatter plot')
```

Out[107]:

```
<matplotlib.collections.PathCollection at 0x16fe64f98b0>
```

Out[107]:

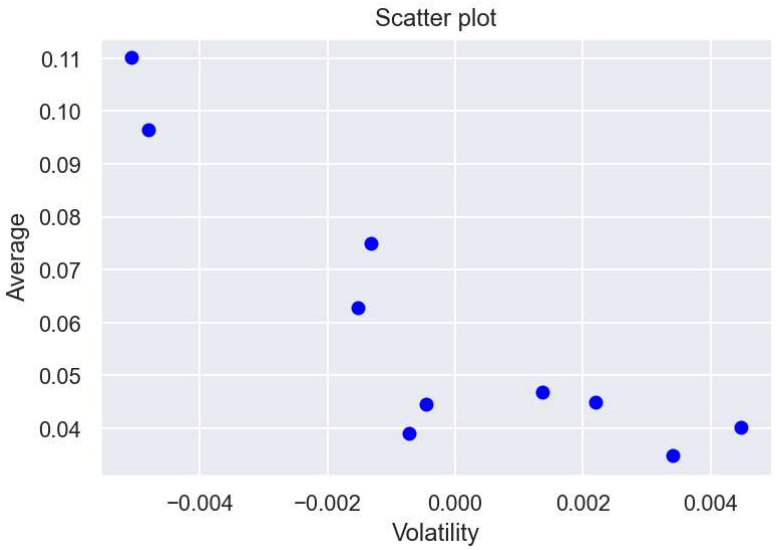
```
Text(0, 0.5, 'Average')
```

Out[107]:

```
Text(0.5, 0, 'Volatility')
```

Out[107]:

```
Text(0.5, 1.0, 'Scatter plot')
```



_Thanks

In [ ]: