FINANCIAL RISK ANALYTICS BUSINESS REPORT

THAKUR ARUN SINGH

DECEMBER 2021

This Business Report shall provide detailed explanation of how we approached each problem given in the assignment. It shall also provide relative resolution and explanation with regards to the problems

CONTENTS

Problem 1:

Businesses or companies can fall prey to default if they are not able to keep up their debt obligations. Defaults will lead to a lower credit rating for the company which in turn reduces its chances of getting credit in the future and may have to pay higher interests on existing debts as well as any new obligations. From an investor's point of view, he would want to invest in a company if it is capable of handling its financial obligations, can grow quickly, and is able to manage the growth scale.

A balance sheet is a financial statement of a company that provides a snapshot of what a company owns, owes, and the amount invested by the shareholders. Thus, it is an important tool that helps evaluate the performance of a business.

Data that is available includes information from the financial statement of the companies for the previous year (2015). Also, information about the Networth of the company in the following year (2016) is provided which can be used to drive the labeled field.

Explanation of data fields available in Data Dictionary, 'Credit Default Data Dictionary.xlsx'

Exploratory data analysis

Dataset has 67 variables of which 63 are of float data type, 3 are integer type and 1 is object type.

The head of the dataset is as below:

	Co_Code	Co_Name	Networth Next Year	Equity Paid Up	Networth	Capital Employed	Total Debt	Gross Block	Net Working Capital	Current Assets	...	PBIDTM (%) [Latest]	PBITM (%) [Latest]	PBDTM (%) [Latest]	CPM (%) [Latest]	APATM (%) [Latest]	Debt Velo (Da
0	16974	Hind.Cables	-8021.60	419.36	-7027.48	-1007.24	5936.03	474.30	-1076.34	40.50	...	0.00	0.00	0.00	0.00	0.00	
1	21214	Tata Tele. Mah.	-3986.19	1954.93	-2968.08	4458.20	7410.18	9070.86	-1098.88	486.86	...	-10.30	-39.74	-57.74	-57.74	-87.18	
2	14852	ABG Shipyard	-3192.58	53.84	506.86	7714.68	6944.54	1281.54	4496.25	9097.64	...	-5279.14	-5516.98	-7780.25	-7723.67	-7961.51	
3	2439	GTL	-3054.51	157.30	-623.49	2353.88	2326.05	1033.69	-2612.42	1034.12	...	-3.33	-7.21	-48.13	-47.70	-51.58	
4	23505	Bharati Defence	-2967.36	50.30	-1070.83	4675.33	5740.90	1084.20	1836.23	4685.81	...	-295.55	-400.55	-845.88	379.79	274.79	3

5 rows × 67 columns

The data has 3586 Rows and 67 Columns. No duplicate data is present in the data set.

```
df.isna().sum().sum()
```

```
0
```

The data has 3586 Rows and 67 Columns. No duplicate data is present in the data set.

We dropped unrequired columns like Co_Code and Co_Name since they do not add value to the analysis.

Descriptive statistics / 5 point summary is shown below.

	Co_Code	Networth Next Year	Equity Paid Up	Networth	Capital Employed	Total Debt	Gross Block	Net Working Capital	Current Assets	Current Liabilities and Provisions	...
count	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	3586.000000	...
mean	16065.388734	725.045251	62.966584	649.746299	2799.611054	1994.823779	594.178829	410.809665	1960.349172	391.992078	...
std	19776.817379	4769.681004	778.761744	4091.988792	26975.135385	23652.842746	4871.547802	6301.218546	22577.570829	2675.001631	...
min	4.000000	-8021.600000	0.000000	-7027.480000	-1824.750000	-0.720000	-41.190000	-13162.420000	-0.910000	-0.230000	...
25%	3029.250000	3.985000	3.750000	3.892500	7.602500	0.030000	0.570000	0.942500	4.000000	0.732500	...
50%	6077.500000	19.015000	8.290000	18.580000	39.090000	7.490000	15.870000	10.145000	24.540000	9.225000	...
75%	24269.500000	123.802500	19.517500	117.297500	226.605000	72.350000	131.895000	61.175000	135.277500	65.650000	...
max	72493.000000	111729.100000	42263.460000	81657.350000	714001.250000	652823.810000	128477.590000	223257.560000	721166.000000	83232.980000	...

8 rows × 66 columns

Current Liabilities and Provisions	...	PBIDTM (%) [Latest]	PBITM (%) [Latest]	PBDTM (%) [Latest]	CPM (%) [Latest]	APATM (%) [Latest]	Debtors Velocity (Days)	Creditors Velocity (Days)	Inventory Velocity (Days)	Value of Output/Total Assets	Value o Output/Gros Bloc
6.000000	...	3585.000000	3585.000000	3585.000000	3585.000000	3585.000000	3586.000000	3.586000e+03	3483.000000	3586.000000	3586.00000
1.992078	...	-51.162890	-109.213414	-311.570357	-307.005632	-365.056187	603.894032	2.057855e+03	79.644559	0.819757	61.88454
5.001631	...	1795.131025	3057.635870	10921.592639	10676.149629	12500.051387	10636.759580	5.416948e+04	137.847792	1.201400	976.82435
0.230000	...	-78870.450000	-141600.000000	-590500.000000	-572000.000000	-688600.000000	0.000000	0.000000e+00	-199.000000	-0.330000	-61.00000
0.732500	...	0.000000	0.000000	0.000000	0.000000	0.000000	8.000000	8.000000e+00	0.000000	0.070000	0.27000
9.225000	...	8.070000	5.230000	4.690000	3.890000	1.590000	49.000000	3.900000e+01	35.000000	0.480000	1.53000
5.650000	...	18.990000	14.290000	14.110000	11.390000	7.410000	106.000000	8.900000e+01	96.000000	1.160000	4.91000
2.980000	...	19233.330000	19195.700000	15640.000000	15640.000000	15266.670000	514721.000000	2.034145e+06	996.000000	17.630000	43404.00000

The values of mean, standard deviation, minimum and maximum, 25th, 50th and 75th percentile are mentioned in the above tables.

Next we checked for null values.

```
Inventory Velocity (Days)              103
Book Value (Adj.) (Unit Curr)            4
Inventory Ratio[Latest]                  1
Interest Cover Ratio[Latest]             1
Current Ratio[Latest]                    1
                                      ...
Value of Output/Total Assets             0
Cash Flow From Operating Activities      0
CEPS (annualised) (Unit Curr)            0
Market Capitalisation                    0
Co_Code                                  0
Length: 67, dtype: int64
```

Further details on missing values is covered under 1.2

PROBLEM 1.8

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

Resolution:

We have created two models using Random Forest one without SMOTE data and one with SMOTE data. We did not scale the data for Random Forest as tree based models are not distance based models and can handle varying ranges of features.

We also used GridSearchCV for hyper parameter tuning. PFB grid which was used, same grid was used for both the models i.e. with and without smote to maintain uniformity.

```
param_grid_rf = {
    'max_depth': [10,20,30],
    'max_features': [2,3,4,5,6,7,8],
    'min_samples_leaf': [25,50,75,100],
    'min_samples_split': [25,50,75,100],
    'n_estimators': [50,100,150]
}
```

A custom function named apply_evl which takes below arguments was created. This function is able to handle modeling for different type of models and returns the performance metrics as the output. The same function has been used to build and evaluate the different models here.

```
def apply_evl(name,model,param_grid,X_train,X_test,y_train,y_test):
```

name --> Name of the model

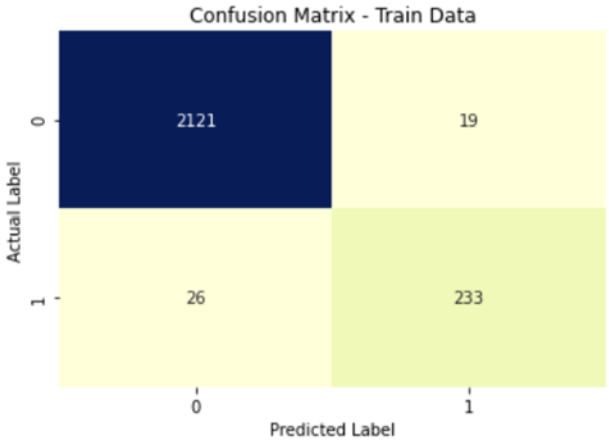
model --> Model object which is created and passed to the function

pram_grid --> this takes the grid as a dictionary object. This can also be passed as *None* in case a grid search is not required.

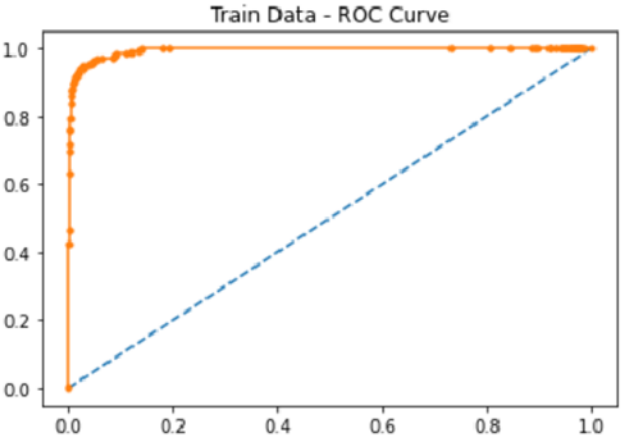
X_Train, X_test,y_train,y_test --> split up data set, in 67:33 ratio with seed as 42 as per project notes.

Below are the performance metrics on **train data** for random forest model **without smote data.**

```
RandomForest
--------------------Best Parameters--------------------
{'max_depth': 10, 'max_features': 6, 'min_samples_leaf': 50, 'min_samples_split': 50, 'n_estimators': 100}


--------------------Best Model Params--------------------
RandomForestClassifier(max_depth=10, max_features=6, min_samples_leaf=50,
                       min_samples_split=50)


Train Accuracy Score for model RandomForestClassifier() is 0.9812421842434348

--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      2140
           1       0.92      0.90      0.91       259

    accuracy                           0.98      2399
   macro avg       0.96      0.95      0.95      2399
weighted avg       0.98      0.98      0.98      2399

------------------------------------------------------------
```

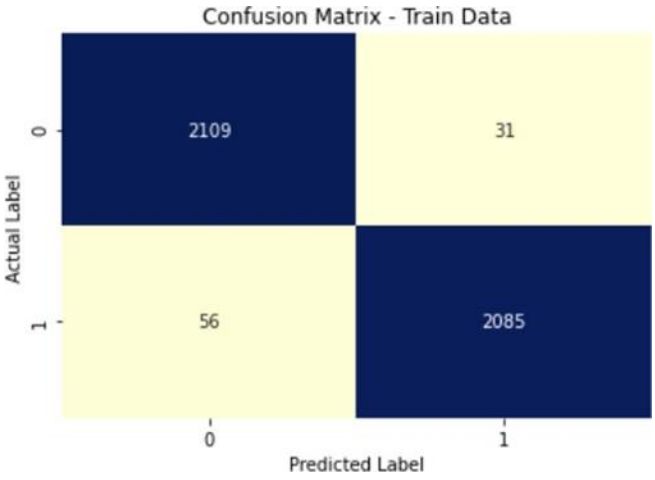


AUC: 0.994

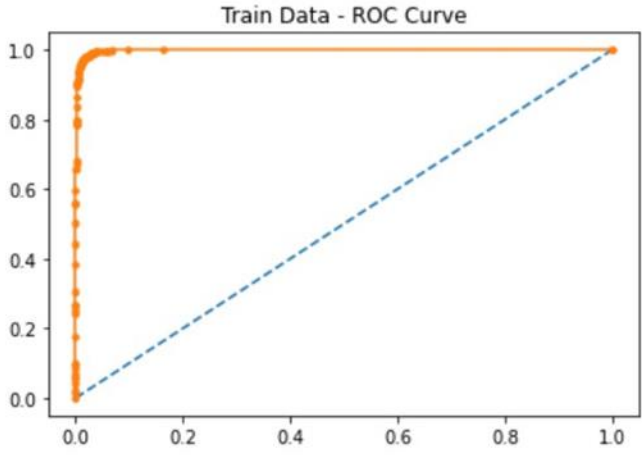


Below are the performance metrics on train data for random forest model with smote data.

```
RF_With_Smote
--------------------Best Parameters--------------------
{'max_depth': 30, 'max_features': 3, 'min_samples_leaf': 25, 'min_samples_split': 25, 'n_estimators': 150}


--------------------Best Model Params--------------------
RandomForestClassifier(max_depth=30, max_features=3, min_samples_leaf=25,
                       min_samples_split=25, n_estimators=150)


Train Accuracy Score for model RandomForestClassifier() is 0.979677645409951

----------------------Classification Report - Train Data----------------------
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      2140
           1       0.99      0.97      0.98      2141

    accuracy                           0.98      4281
   macro avg       0.98      0.98      0.98      4281
weighted avg       0.98      0.98      0.98      4281

------------------------------------------------------------
```



AUC: 0.998



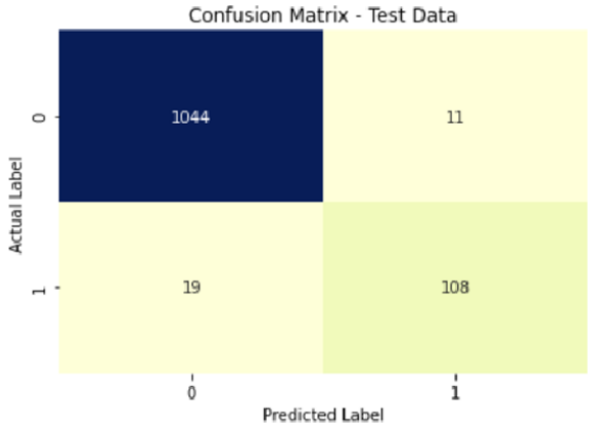
PROBLEM 1.9

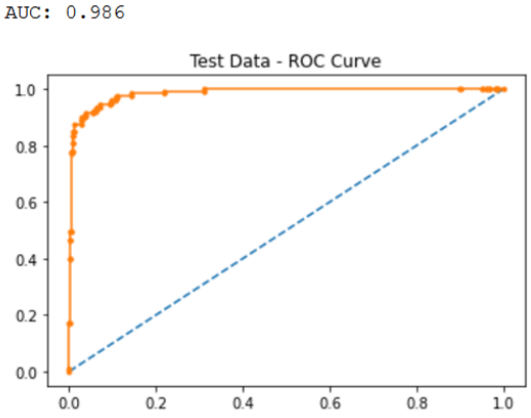
Validate the Random Forest Model on test Dataset and state the performance matrices. Also state interpretation from the model

Resolution:

Both the random forest models were then evaluated on the test data. Below are the results for **test data** for random forest model **without smote data.**

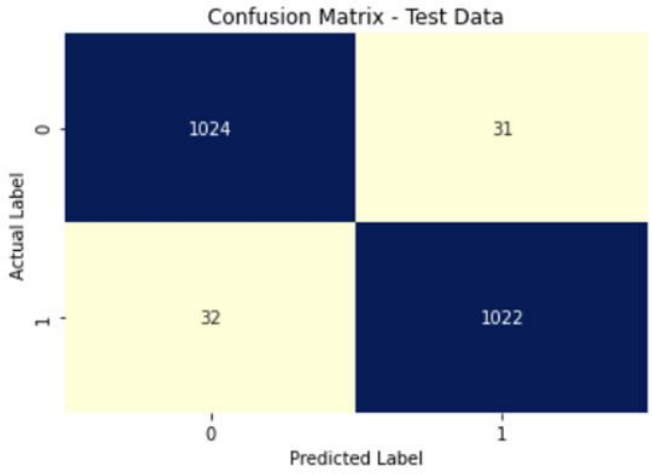
```
Test Accuracy Score for model RandomForestClassifier() is 0.9746192893401016


--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1055
           1       0.91      0.85      0.88       127

    accuracy                           0.97      1182
   macro avg       0.94      0.92      0.93      1182
weighted avg       0.97      0.97      0.97      1182

------------------------------------------------------------
```



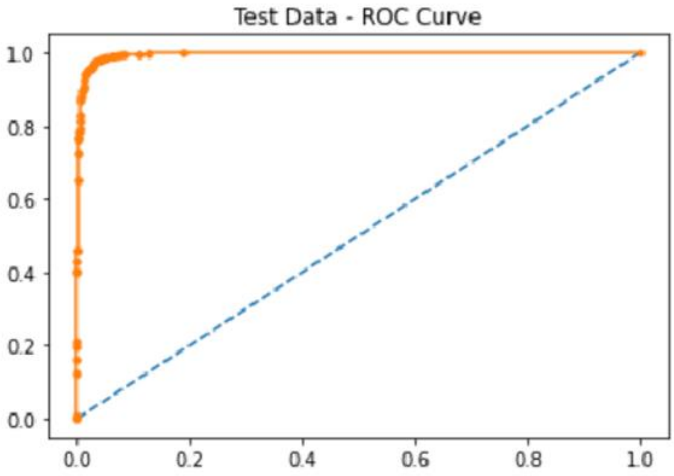


Below are the performance metrics on **test data** for random forest model **with smote data.**

```
Test Accuracy Score for model RandomForestClassifier() is 0.9701280227596017


--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      1055
           1       0.97      0.97      0.97      1054

    accuracy                           0.97      2109
   macro avg       0.97      0.97      0.97      2109
weighted avg       0.97      0.97      0.97      2109

------------------------------------------------------------
```



AUC: 0.996



Below are the comparison of various evaluation metrics in tabular format for both the models i.e. with and without smote.

	Accuracy	Precision	Recall	F1	AUC
RandomForest_Train	0.981242	0.924603	0.899614	0.911937	0.99399
RandomForest_Test	0.974619	0.907563	0.850394	0.878049	0.985857

	Accuracy	Precision	Recall	F1	AUC
RF_With_Smote_Train	0.979678	0.98535	0.973844	0.979563	0.998354
RF_With_Smote_Test	0.970128	0.97056	0.969639	0.9701	0.996038

We can see in the above comparison that even though Random Forest without SMOTE has higher accuracy, but Random Forest with SMOTE surpasses the other model significantly in terms of Precision, Recall, F1 score as well as AUC score.

Also there is not a significant difference in the accuracies of both the models.

Hence we can say Random Forest with SMOTE data is the better of the two models with 97.01% accuracy, 97.05% precision, 96.96% recall and a recall and F1 score of 0.9701 and 0.996038 respectively.

PROBLEM 1.10

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

Resolution:

we have created two models using LDA, one without SMOTE data and one with SMOTE data. We did not scale the data for LDA as it finds it's coefficients using the variation between the classes, hence scaling doesn't matter.

We also used GridSearchCV for hyper parameter tuning. PFB the grid which was used. Same grid was used for both the models i.e. with and without smote to maintain uniformity.

```
param_grid_lda = {'solver': ['svd', 'lsqr', 'eigen'],
                  'shrinkage': ['auto', 'none', 'default'],
                  'tol': [0.000001, 0.0000001, 0.000000001]}
```

A custom function named **apply_evl** which takes below arguments was created. This function is able to handle modeling for different type of models and returns the performance metrics as the output. The same function has been used to build and evaluate the different models here.

```
def apply_evl(name,model,param_grid,X_train,X_test,y_train,y_test):
```

name --> Name of the model

model --> Model object which is created and passed to the function

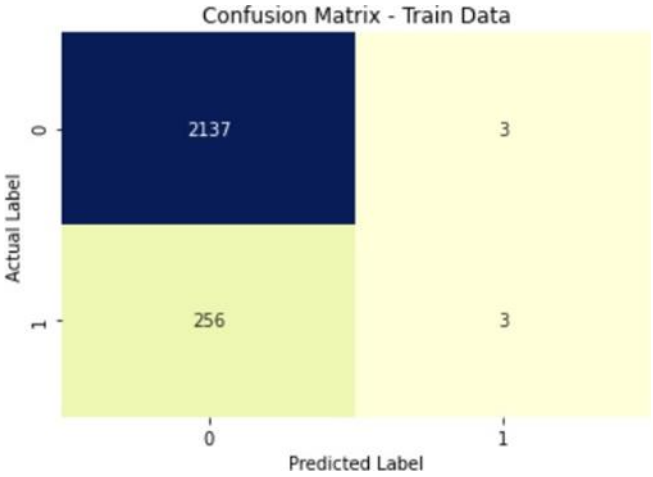
pram_grid --> This takes the grid as a dictionary object. This can also be passed as *None* in case a grid search is not required.

X_Train, X_test,y_train,y_test --> split up data set, in 67:33 ratio with seed as 42 as per project notes.

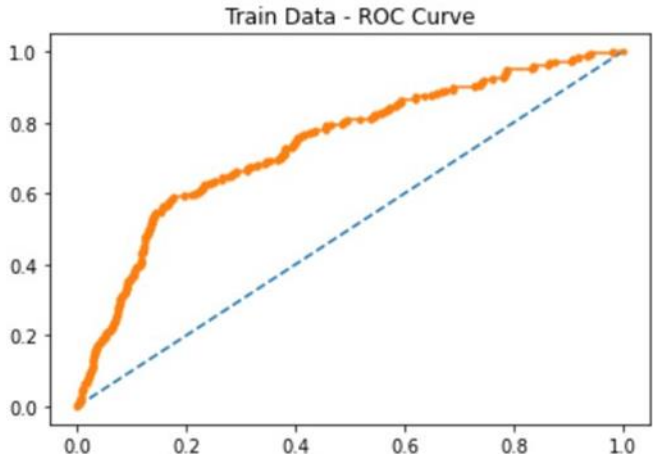
Below are the performance metrics on **train data** for LDA model **without smote data.**

```
LDA
--------------------Best Parameters--------------------
{'shrinkage': 'auto', 'solver': 'lsqr', 'tol': 1e-06}


--------------------Best Model Params--------------------
LinearDiscriminantAnalysis(shrinkage='auto', solver='lsqr', tol=1e-06)


Train Accuracy Score for model LinearDiscriminantAnalysis() is 0.8920383493122134
```

```
--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      2140
           1       0.50      0.01      0.02       259

    accuracy                           0.89      2399
   macro avg       0.70      0.51      0.48      2399
weighted avg       0.85      0.89      0.84      2399

------------------------------------------------------------
```



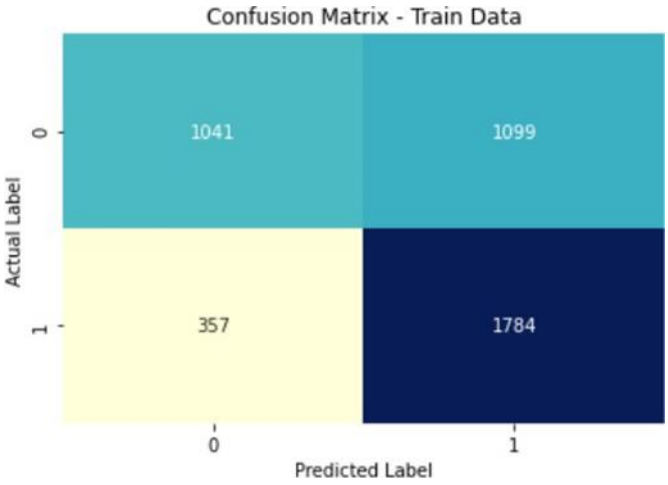

```
AUC: 0.740
```



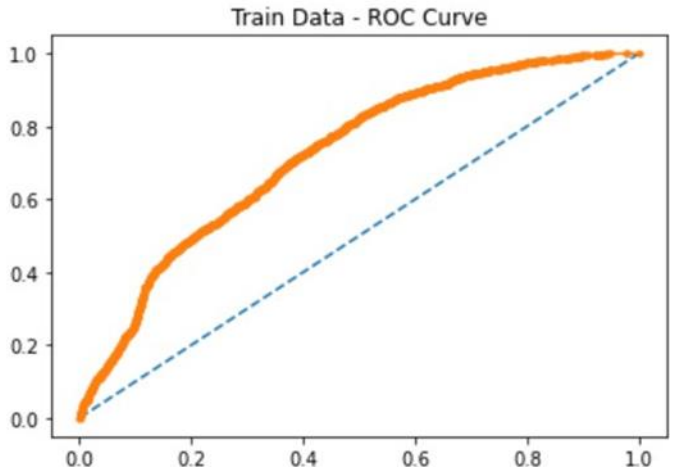
Below are the performance metrics on **train data** for LDA model **with smote data.**

```
LDA_With_Smote
--------------------Best Parameters--------------------
{'shrinkage': 'auto', 'solver': 'lsqr', 'tol': 1e-06}


--------------------Best Model Params--------------------
LinearDiscriminantAnalysis(shrinkage='auto', solver='lsqr', tol=1e-06)


Train Accuracy Score for model LinearDiscriminantAnalysis() is 0.6598925484699837
```

```
--------------------Classification Report - Train Data--------------------
              precision    recall  f1-score   support

           0       0.74      0.49      0.59      2140
           1       0.62      0.83      0.71      2141

    accuracy                           0.66      4281
   macro avg       0.68      0.66      0.65      4281
weighted avg       0.68      0.66      0.65      4281

------------------------------------------------------------
```



AUC: 0.725



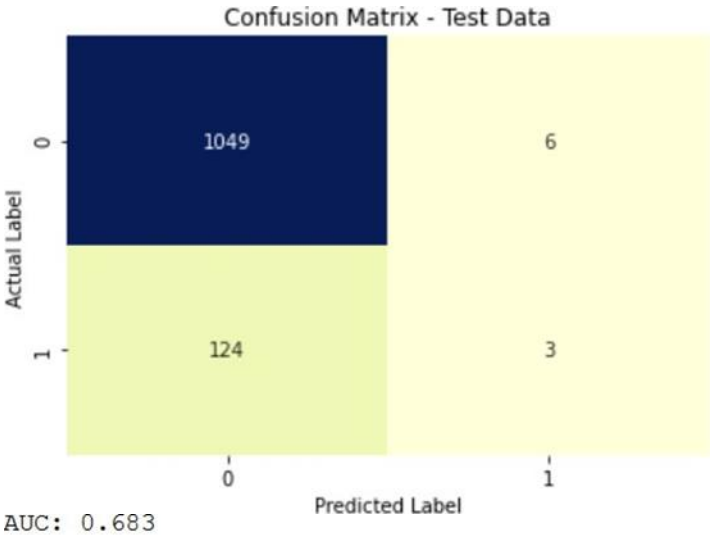
PROBLEM 1.11

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

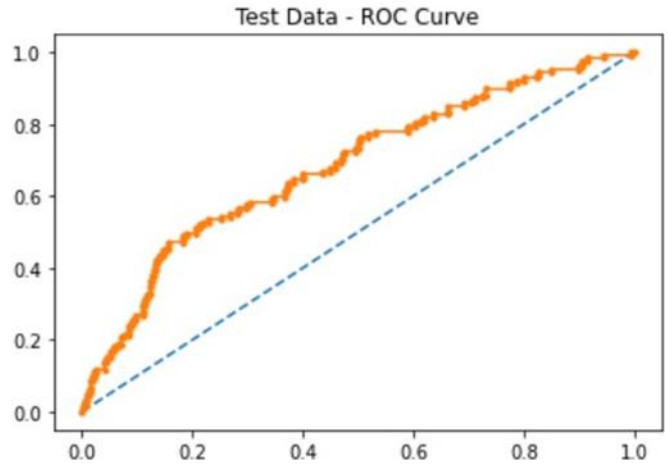
Resolution:

Both the LDA models were then evaluated on the test data. Below are the results for **test data** for LDA model **without smote data.**

```
Test Accuracy Score for model LinearDiscriminantAnalysis() is 0.8900169204737732

--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      1055
           1       0.33      0.02      0.04       127

    accuracy                           0.89      1182
   macro avg       0.61      0.51      0.49      1182
weighted avg       0.83      0.89      0.85      1182

------------------------------------------------------------
```

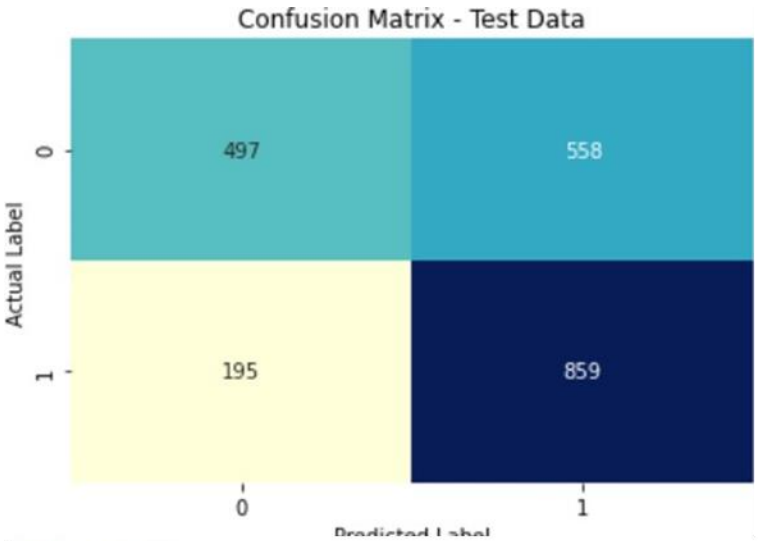


AUC: 0.683

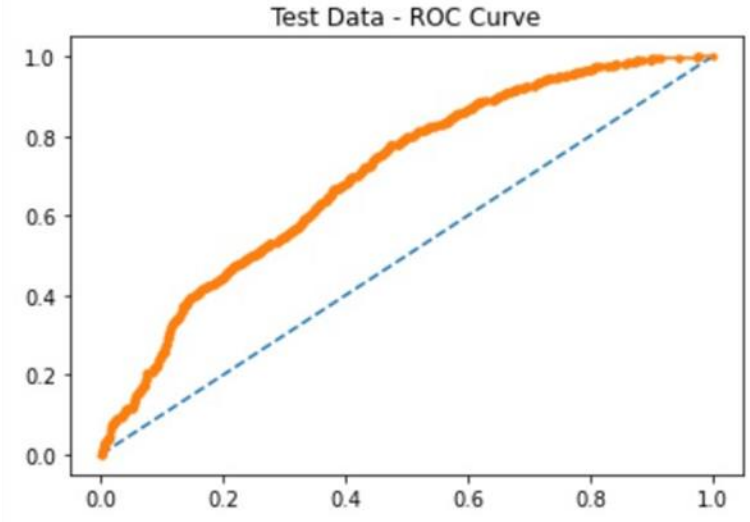


Below are the performance metrics on **test data** for LDA model with **smote data.**

```
Test Accuracy Score for model LinearDiscriminantAnalysis() is 0.6429587482219061


--------------------Classification Report - Test Data--------------------
              precision    recall  f1-score   support

           0       0.72      0.47      0.57      1055
           1       0.61      0.81      0.70      1054

    accuracy                           0.64      2109
   macro avg       0.66      0.64      0.63      2109
weighted avg       0.66      0.64      0.63      2109

------------------------------------------------------------
```



AUC: 0.703



Below are the comparison of various evaluation matrics in tabular format for both the models i.e. with and without smote.

	Accuracy	Precision	Recall	F1	AUC
LDA_Train	0.892038	0.5	0.011583	0.0226415	0.740082
LDA_Test	0.890017	0.333333	0.023622	0.0441176	0.683203

	Accuracy	Precision	Recall	F1	AUC
LDA_With_Smote_Train	0.659893	0.6188	0.833255	0.710191	0.724862
LDA_With_Smote_Test	0.642959	0.60621	0.814991	0.695265	0.702944

Comparing the two models, we can see LDA model without SMOTE data has higher accuracy on test data. However it has really bad precision, recall and F1 scores compared to the other model.

On purely accuracy perspective model without smote data performs better. But in real world scenario model with smote will perform better given better recall and precision values.

PROBLEM 1.12

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

Resolution:

We made following models as part of this exercise. For Random Forest and LDA scaling wasnot taken into account as these models are not impacted by varying magnitudes of the variables.

- Logistic regression using statsModel without SMOTE
- Logistic regression using statsModel with SMOTE
- Logistic Regression using sklearn With Unscaled Variables Without SMOTE
- Logistic Regression using sklearn With Scaled Variables Without SMOTE
- Logistic Regression using sklearn With Unscaled Variables With SMOTE
- Logistic Regression using sklearn With Scaled Variables With SMOTE
- LDA using sklearn without SMOTE
- LDA using sklearn with SMOTE
- Random Forest using sklearn without SMOTE
- Random Forest using sklearn with SMO

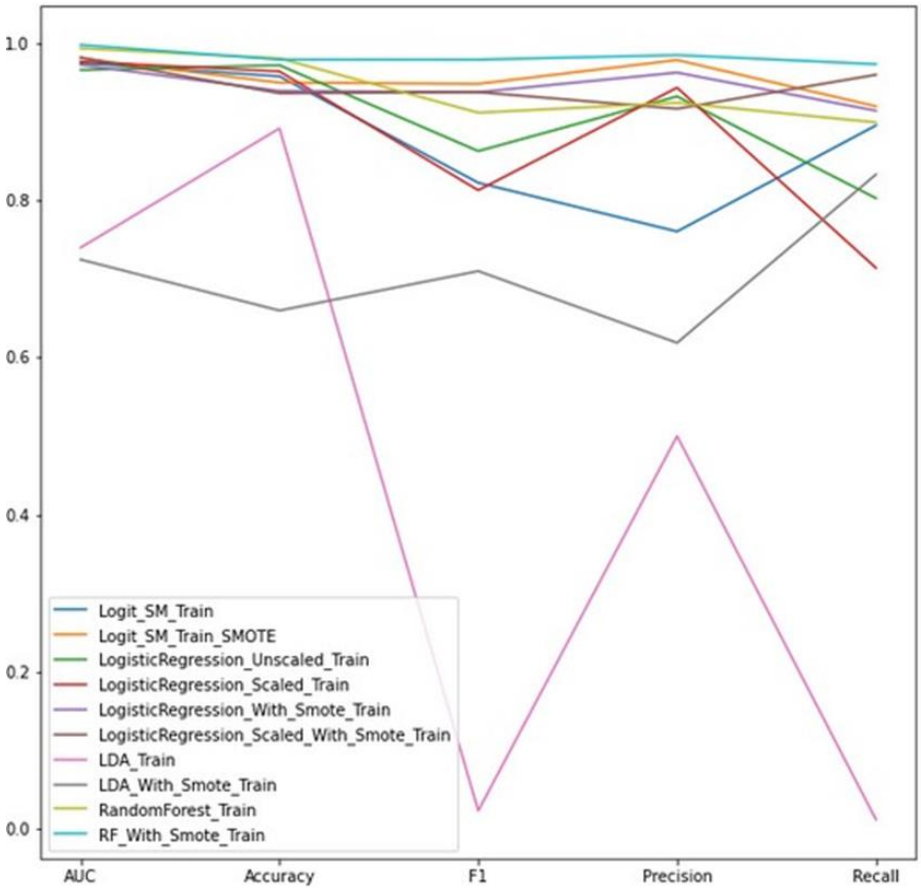
Below is the comparison chart for all the models.

	Accuracy	Precision	Recall	F1	AUC
RandomForest_Train	0.981242	0.924603	0.899614	0.911937	0.99399
RF_With_Smote_Train	0.979678	0.98535	0.973844	0.979563	0.998354
RandomForest_Test	0.974619	0.907563	0.850394	0.878049	0.985857
LogisticRegression_Unscaled_Train	0.972489	0.932735	0.803089	0.863071	0.966442
RF_With_Smote_Test	0.970128	0.97056	0.969639	0.9701	0.996038
LogisticRegression_Scaled_Train	0.964569	0.943878	0.714286	0.813187	0.97694
LogisticRegression_Unscaled_Test	0.959391	0.869159	0.732283	0.794872	0.945725
Logit_SM_Train	0.958316	0.760656	0.895753	0.822695	0.975275
Logit_SM_Test	0.952623	0.726115	0.897638	0.802817	0.956256
LogisticRegression_Scaled_Test	0.951777	0.830189	0.692913	0.755365	0.945098
Logit_SM_Train_SMOTE	0.950245	0.979125	0.920131	0.948712	0.981177
Logit_SM_Test_SMOTE	0.945472	0.962562	0.926945	0.944418	0.974799
LogisticRegression_With_Smote_Train	0.939734	0.963109	0.914526	0.938189	0.972807
LogisticRegression_Scaled_With_Smote_Train	0.936697	0.917038	0.960299	0.93817	0.982349
LogisticRegression_With_Smote_Test	0.935514	0.94305	0.926945	0.934928	0.968668
LogisticRegression_Scaled_With_Smote_Test	0.928876	0.902135	0.962049	0.931129	0.97516
LDA_Train	0.892038	0.5	0.011583	0.0226415	0.740082
LDA_Test	0.890017	0.333333	0.023622	0.0441176	0.683203
LDA_With_Smote_Train	0.659893	0.6188	0.833255	0.710191	0.724862
LDA_With_Smote_Test	0.642959	0.60621	0.814991	0.695265	0.702944

Looking at the above comparison chart we can see Random Forest model has surpassed all other models in terms of accuracy. Both Random Forest with and without smote feature in the top 2 models.

While Random Forest was the best performing model, LDA models were the worst performers of all. LDA with or without were both really bad in terms of accuracies, LDA with smote being the worst of the lot.

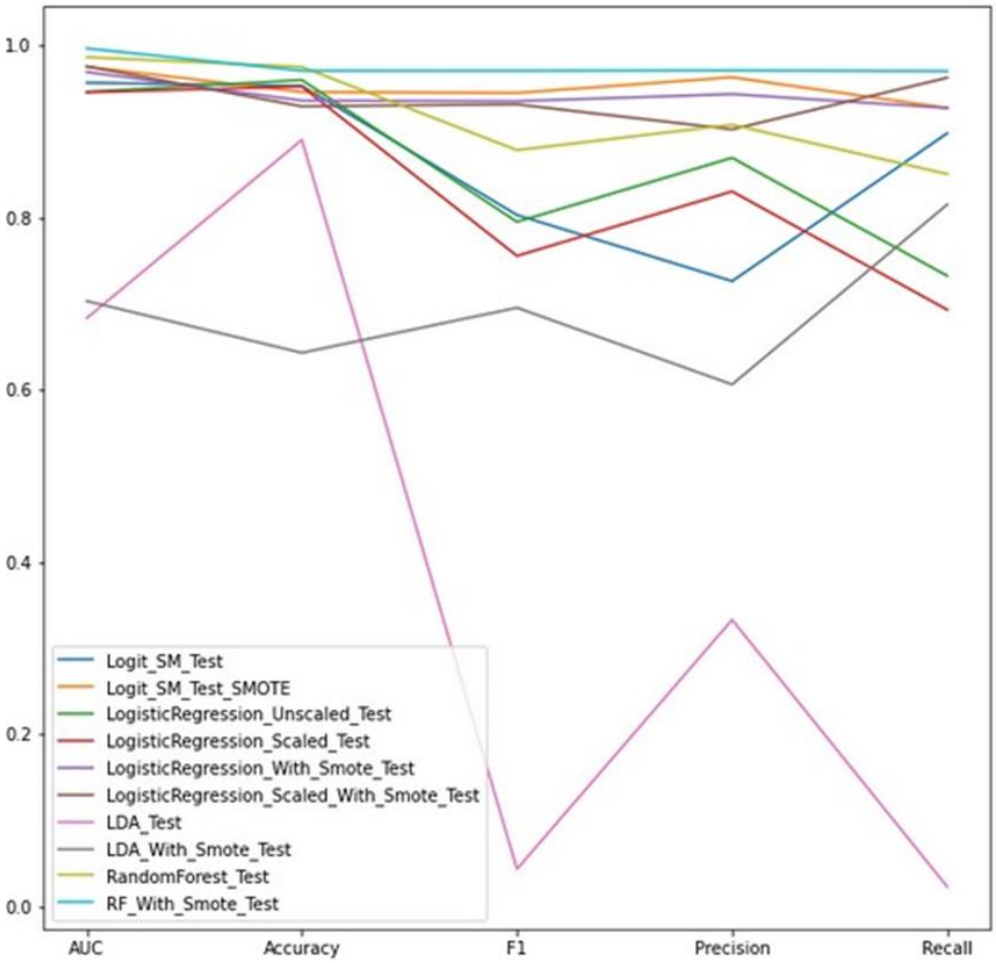
Below is a comparison of different evaluation metrics for all the models on the training data set.



Looking at the above chart we clearly see the blue line indicating "Random Forest With SMIOTE" being on top of all other models, consistently for all the evaluation parameters.

While the pink line for LDA without smote is slightly higher than LDA with smote in terms of accuracy and auc score, it slips down significantly on all other parameters, making it the worst performing model of the lot.

Below is a comparison of different evaluation metrics for all the models on the test data set.



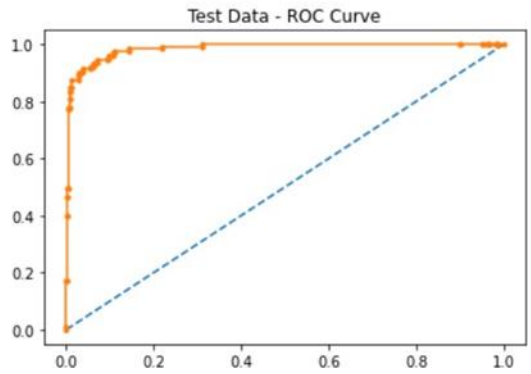
On the test data set, which is the deciding factor, here too we see Random Forest with SMOTE indicated by blue line performing the best on almost all the fronts. We can see the blue line dip only very slightly on accuracy compared to Random Forest without smote, on all other fronts the blue line stays on the top and is the clear winner amongst all the models.

Random Forest model with smote dataset is the best model amongst all the models with 97.01% accuracy, 97.05% precision, 96.96% recall and a recall and F1 score of 0.9701 and 0.996038 respectively.

Also the ROC curves for Logit, LDA and random forest are plotted below. ROC curves for the best performing models on test data set have been plotted for comparison.

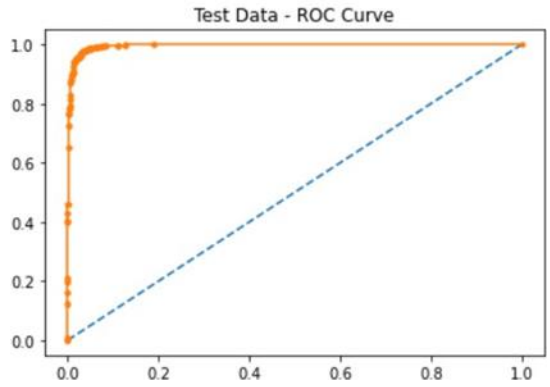
Comparing AUC scores too we see Random forest with smote data clearly wins compared to the others.

AUC: 0.986



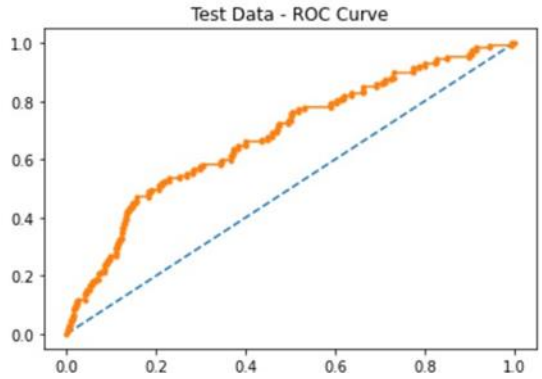
RF without SMOTE

AUC: 0.996



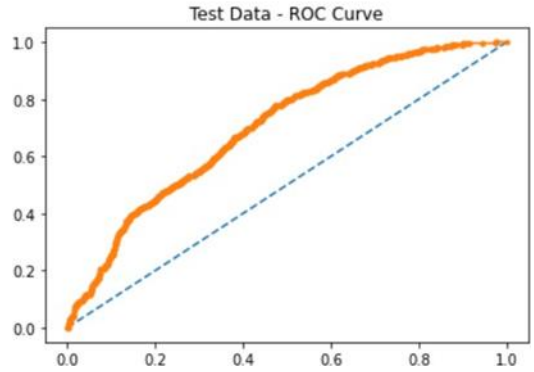
RF with SMOTE

AUC: 0.683



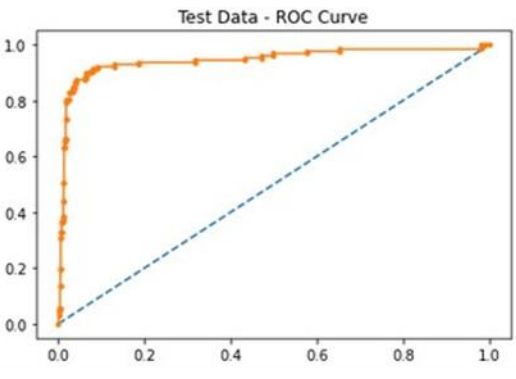
LDA Without SMOTE

AUC: 0.703



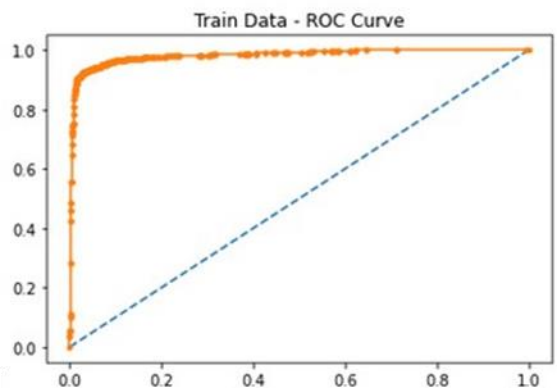
LDA with SMOTE

AUC: 0.945



Logit without SMOTE (Scaled)

AUC: 0.982



Logit with SMOTE (Scaled)

PROBLEM 1.13

State Recommendations from the above models

Resolution:

Using the information gained from above exercise, we can say Random Forest with smote data is the best model. We also looked at the coefficients derived from the best Logit model built using Stats model to derive some more insights.

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.3898	0.148	-2.641	0.008	-0.679	-0.100
Book_Value_Unit_Curr	-0.1514	0.012	-12.174	0.000	-0.176	-0.127
CEPS_annualised_Unit_Curr	-0.0972	0.013	-7.649	0.000	-0.122	-0.072
Curr_Ratio_Latest	-0.4580	0.097	-4.733	0.000	-0.648	-0.268
Interest_Cover_Ratio_Latest	-0.0024	0.001	-2.999	0.003	-0.004	-0.001

From the above analysis, we can infer below business insights. Following things should be kept in mind while investing in these companies.

- Lower the Book_value_unit_curr i.e. Net assets, higher is the chance of a default, which would mean the net worth next year for this company is expected to be negative.
- Lower the CEPS_annualised_Unit_Curr i.e. Cash earning per share, higher is the change of a default.
- Higher the Curr_Ratio_Latest, i.e. the companies ability to pay short term dues, lower are its chances of defaulting or having a negative net worth in next year.
- Higher the Interest_Cover_Ratio_Latest lower the chances of default. Which means easier the company is able to pay the interest on its outstanding debt, lower are its chances to default.
- Curr_Ratio_Latest is most important criteria amongst the above parameters, while Interest_Cover_Ratio_Latest is the least important when considering only these 4 parameters. However all these 4 parameters remain important compared to the other variables in the data set.

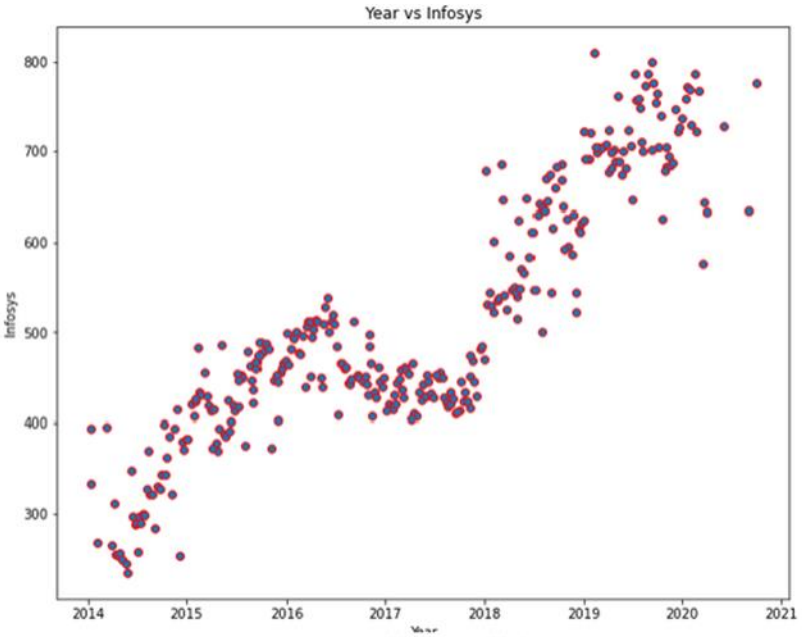
PROBLEM 2.1

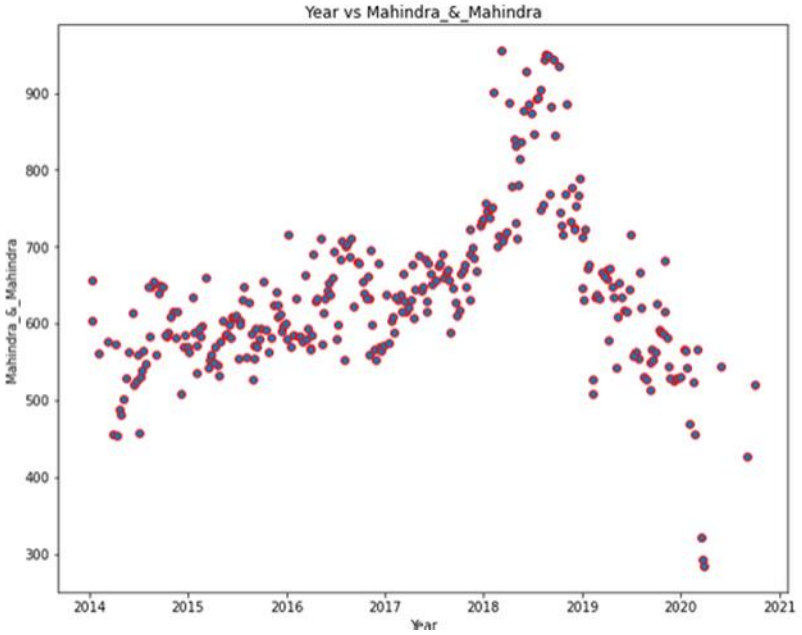
Draw Stock Price Graph (Stock Price vs Time) for any 2 given stocks with inference

Resolution:

The dataset contains 6 years of information (weekly stock information) on the stock prices of 10 different Indian Stocks. Calculate the mean and standard deviation on the stock returns and share insights. Please find attached the files to be referred.

Stock price graphs of Infosys and Mahindra & Mahindra are plotted below vs Time.





PROBLEM 2.2

Calculate Returns for all stocks with inference

Resolution:

Returns for all the stocks i.e. difference of log of price at t and the log of price at t-1 are shown below.

```
stock_returns = np.log(stock_prices.drop(['Date','dates'],axis=1)).diff(axis = 0, periods = 1)
```

Checking top 5 rows

```
stock_returns.head()
```

	Infosys	Indian_Hotel	Mahindra_&_Mahindra	Axis_Bank	SAIL	Shree_Cement	Sun_Pharma	Jindal_Steel	Idea_Vodafone	Jet_Airways
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	-0.026873	-0.014599	0.006572	0.048247	0.028988	0.032831	0.094491	-0.065882	0.011976	0.086112
2	-0.011742	0.000000	-0.008772	-0.021979	-0.028988	-0.013888	-0.004930	0.000000	-0.011976	-0.078943
3	-0.003945	0.000000	0.072218	0.047025	0.000000	0.007583	-0.004955	-0.018084	0.000000	0.007117
4	0.011788	-0.045120	-0.012371	-0.003540	-0.076373	-0.019515	0.011523	-0.140857	-0.049393	-0.148846

Week over week returns for all the stocks has been given in the above figure. We have given only the top 5 rows due to space constraints here. For complete data refer Jupyter notebook.

PROBLEM 2.3

Calculate Stock Means and Standard Deviation for all stocks with inference.

Resolution:

Stock means and standard deviations have been calculated below. The values have been sorted in descending order.

Calculating stock means

```
stock_means = stock_returns.mean(axis = 0)
stock_means.sort_values(ascending=False)
```

```
Shree_Cement           0.003681
Infosys                0.002794
Axis_Bank              0.001167
Indian_Hotel           0.000266
Sun_Pharma            -0.001455
Mahindra_&_Mahindra   -0.001506
SAIL                  -0.003463
Jindal_Steel          -0.004123
Jet_Airways           -0.009548
Idea_Vodafone         -0.010608
dtype: float64
```

Idea_Vodafone has the lowest returns, while shree cements have the highest returns.

Calculating stock standard deviation

```
stock_sd = stock_returns.std(axis = 0)
stock_sd.sort_values(ascending=False)
```

```
Idea_Vodafone           0.104315
Jet_Airways             0.097972
Jindal_Steel            0.075108
SAIL                    0.062188
Indian_Hotel            0.047131
Axis_Bank               0.045828
Sun_Pharma              0.045033
Mahindra_&_Mahindra     0.040169
Shree_Cement            0.039917
Infosys                 0.035070
dtype: float64
```

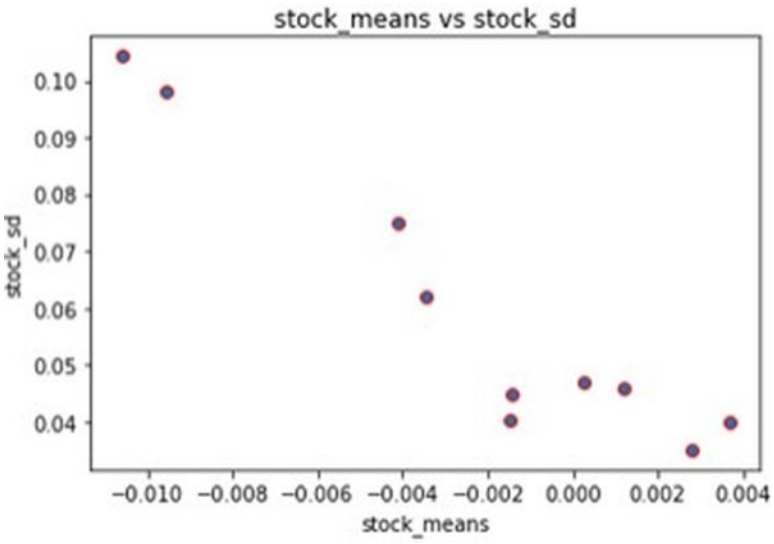
Idea_Vodafone has the highest risk factor while Infosys is the least risky investment option.

PROBLEM 2.4

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

Resolution:

Below is the plot of stock means vs standard deviation.



Stoks higher up but on the far left indicate high volatility and low returns, while the stocks on the bottom right indicate low volatility and high returns.

This is a useful graph to find a balance between risk and reward when it comes to investing in different companies.

PROBLEM 2.5

Conclusion and Recommendations

Resolution:

We can conclude by saying the below.

Stock with a lower mean & higher standard deviation do not play a role in a portfolio that has competing stock with more returns & less risk.

Thus for the data we have here, we are only left few stocks:

- One with highest return and lowest risk &
- One with lowest risk and highest return

Therefore from pure Returns perspective, Shree_Cement followed by Infosys & Axis_Bank looks good in this dataset.

From pure Risk perspective (as measured by standard deviation), Infosys followed by Shree_Cement & Mahindra_&_Mahindra looks good in this dataset.

We would recommend using the stock means vs standard deviation plot to asses the risk to reward ratio. More volatile stock might give short term gains but might not be a good investment in long term. Whereas a low volatile stock might not be a good investment in short term, but might give a good return in long term.

Hence based on the type of investment that one is looking for, a inference should be made from the above mentioned plot.

The End

Thakur Arun Singh

*****************************^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^**************************