greatlearning
Power Ahead

Financial Risk Analysis Project

Milestone 2

BUSINESS REPORT

Business Problem

Problem Statement

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.

1.8 Build a Random Forest Model on the Train Set

Approach

The same train and test datasets as used in Milestone 1 were used.

A Base model using the default values of the hyper-parameters was first built to be used as a reference. Grid search was then used to fine-tune the model hyper-parameter

Base Model

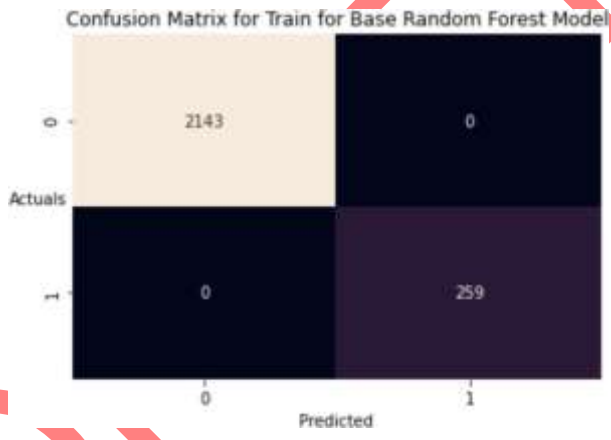
- **Train Dataset**

The results of the Base Model on the Train dataset are shown below:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2143
1	1.00	1.00	1.00	259
accuracy			1.00	2402
macro avg	1.00	1.00	1.00	2402
weighted avg	1.00	1.00	1.00	2402

Clearly, there could be a problem of over-fitting, which will be tested on the test dataset.

The confusion matrix is given below:

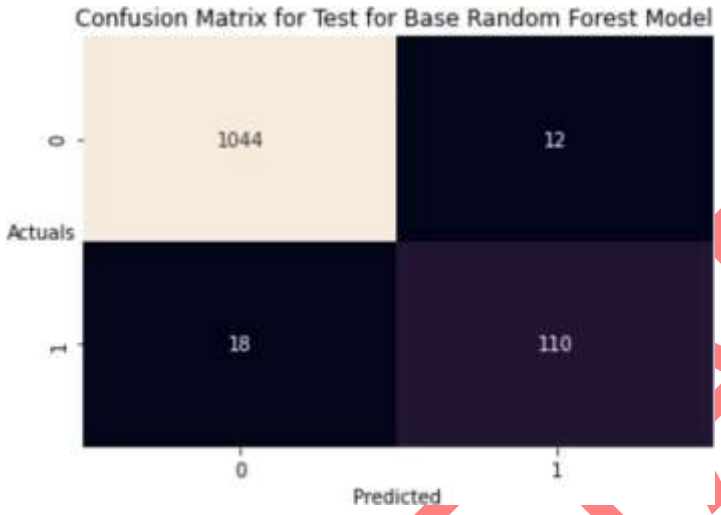


Test Dataset Performance of the Base Model

The classification report of the Base Model on the Test Dataset is given below:

	precision	recall	f1-score	support
0	0.98	0.99	0.99	1056
1	0.90	0.86	0.88	128
accuracy			0.97	1184
macro avg	0.94	0.92	0.93	1184
weighted avg	0.97	0.97	0.97	1184

The confusion matrix of the Test data is given below:



Observations

- The model is overfitting, given that the Recall on Train dataset is 1 on Train dataset and only 0.86 on the Test dataset.
- The comparison of the other performance measures on train and test datasets are given below:

	RFC_Base_Train	RFC_Base_Test
Recall	1.0	0.859
F1 Score	1.0	0.880
Precision	1.0	0.902
Accuracy	1.0	0.975
auc	1.0	0.991

GridSearch Model

In the several iterations, the following hyperparameters emerged as the best:

```
{'criterion': 'entropy',
 'max_depth': 3,
 'max_features': 10,
 'min_samples_leaf': 8,
 'min_samples_split': 15,
 'n_estimators': 200,
 'oob_score': True,
 'random_state': 1}
```

Training Data Performance

The results of the model using these best hyperparameters are as follows (on the training dataset)

	precision	recall	f1-score	support
0	0.99	0.99	0.99	2143
1	0.93	0.89	0.91	259
accuracy			0.98	2402
macro avg	0.96	0.94	0.95	2402
weighted avg	0.98	0.98	0.98	2402

The confusion matrix on the train dataset is seen below:

The importance of the features (top-10) is given below in descending order (top variables):

	Imp
Book_Value_Unit_Curr	0.224802
Capital_exp_in_forex	0.213173
Networth	0.171232
PBDT	0.047699
ROG_Market_Capitalisation_perc	0.044950
Adjusted_PAT	0.033388
PBT	0.028692
PAT	0.026291
Capital_Employed	0.025657
Retained_Earning	0.024905

Impact of SMOTE on Random Forest Model performance

It is seen that though there is high Accuracy (0.98), the Recall is still around 0.89 for the GridSearch model. Since there is an imbalance in the data for the dependent variable 'default', SMOTE technique could be useful to improve model performance

Using SMOTE, the record count increased as follows:

```
The original dataset has  3586  records and  65  variables
The resampled dataset using SMOTE has  6398  records and  65  variables
```

The class imbalance for values of default = 1 and default = 0 on the treated dataset is as follows:

```
1    0.5
0    0.5
Name: default, dtype: float64
```

The resampled dataset was then split into train and test using the same criterion as given earlier (ratio of 67:33 and random_state = 42). The results are as follows:

```
The resampled train dataset has  4286  records and the resampled test dataset has  2112  records
```

GridSearch was used to identify the best-fit hyper-parameters as follows:

```
RandomForestClassifier(criterion='entropy', max_depth=3, max_features=10,
                       min_samples_leaf=5, min_samples_split=15,
                       n_estimators=150, oob_score=True, random_state=1)
```

Performance on Train Dataset

The classification report on the Resampled Train dataset is as follows:

	precision	recall	f1-score	support
0	0.97	0.98	0.97	2143
1	0.98	0.97	0.97	2143
accuracy			0.97	4286
macro avg	0.97	0.97	0.97	4286
weighted avg	0.97	0.97	0.97	4286

The confusion matrix on the Resampled Train dataset is as follows:

The top 10 most-important features of the model on the resampled data are given below:

	Imp
Capital_exp_in_forex	0.232560
Book_Value_Unit_Curr	0.203648
Networth	0.125357
PBDT	0.051759
PBT	0.043254
PAT	0.040495
Market_Capitalisation	0.038161
Adjusted_PAT	0.034758
Retained_Earning	0.031004
Cash_Flow_From_Fin	0.029527

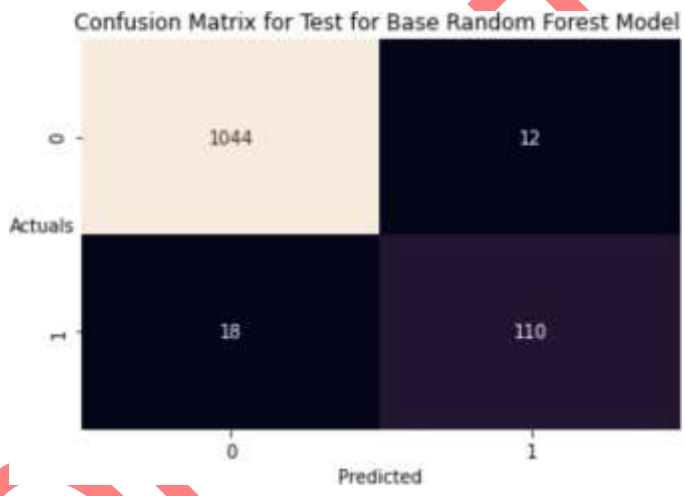
1.9 Validate the Random Forest Model on test Dataset and state the performance matrices

Base Model

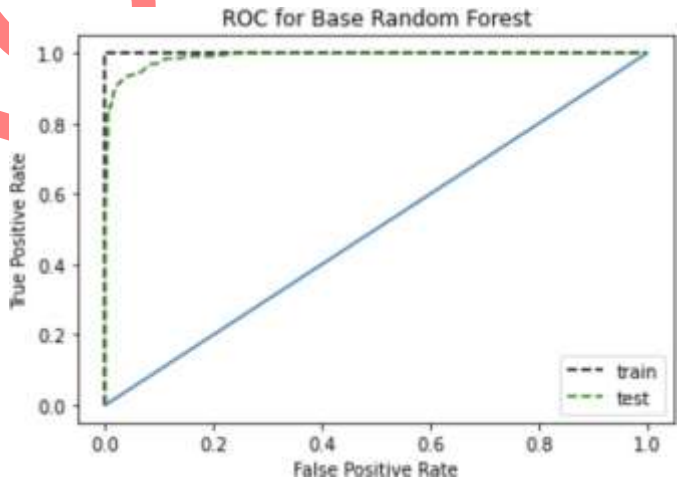
The performance of the Base model on the test datasets is given below:

	precision	recall	f1-score	support
0	0.98	0.99	0.99	1056
1	0.90	0.86	0.88	128
accuracy			0.97	1184
macro avg	0.94	0.92	0.93	1184
weighted avg	0.97	0.97	0.97	1184

The confusion matrix is given below:



The ROC curve is as follows:



The comparison of the performance on the Train and Test data is summarized below:

	RFC_Base_Train	RFC_Base_Test
Recall	1.0	0.859
F1 Score	1.0	0.880
Precision	1.0	0.902
Accuracy	1.0	0.975
auc	1.0	0.991

Observations

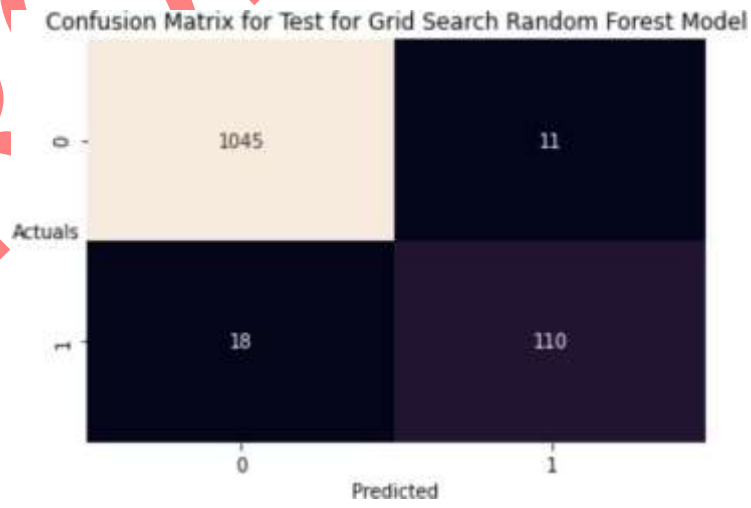
- As expected, the Base Model is overfitting with a significant difference in Recall and Precision scores on test data as compared to train dataset

Grid Search

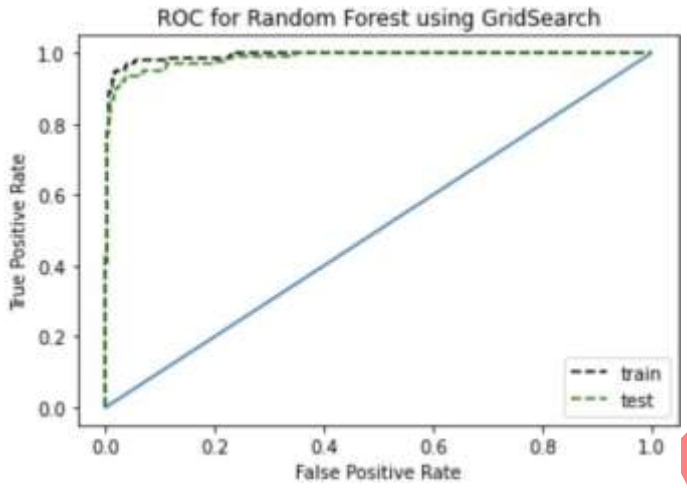
The performance of the Grid Search model on Test Data is given below:

	precision	recall	f1-score	support
0	0.98	0.99	0.99	1056
1	0.91	0.86	0.88	128
accuracy			0.98	1184
macro avg	0.95	0.92	0.93	1184
weighted avg	0.98	0.98	0.98	1184

The confusion matrix is given below:



The ROC curve is as follows:



The comparison with the Train dataset is given below:

	RFC_Grid_Train	RFC_Grid_Test
Recall	0.892	0.859
F1 Score	0.911	0.884
Precision	0.931	0.909
Accuracy	0.981	0.976
auc	0.993	0.986

Observations

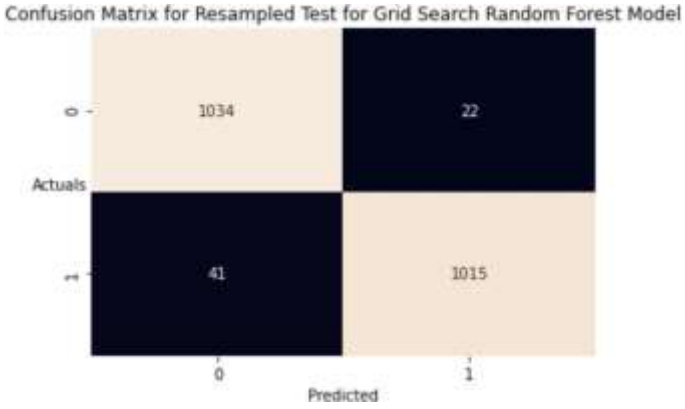
It is seen that there is a slight problem of over-fitting. However, the performance on the test dataset is still fairly good.

SMOTE-based Model

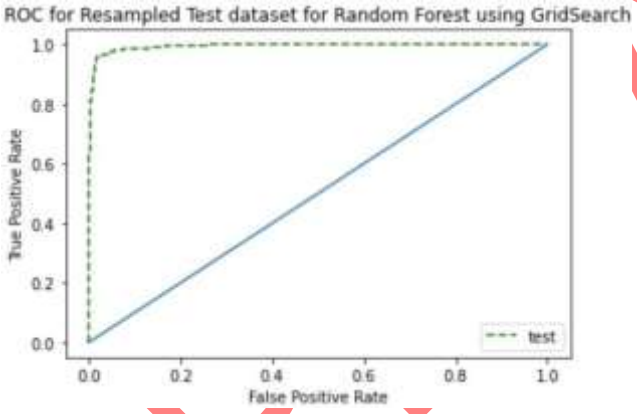
The classification report of the RF model built on the resampled test data is given below:

	precision	recall	f1-score	support
0	0.96	0.98	0.97	1056
1	0.98	0.96	0.97	1056
accuracy			0.97	2112
macro avg	0.97	0.97	0.97	2112
weighted avg	0.97	0.97	0.97	2112

The confusion matrix on the Resampled Test Dataset is given below:



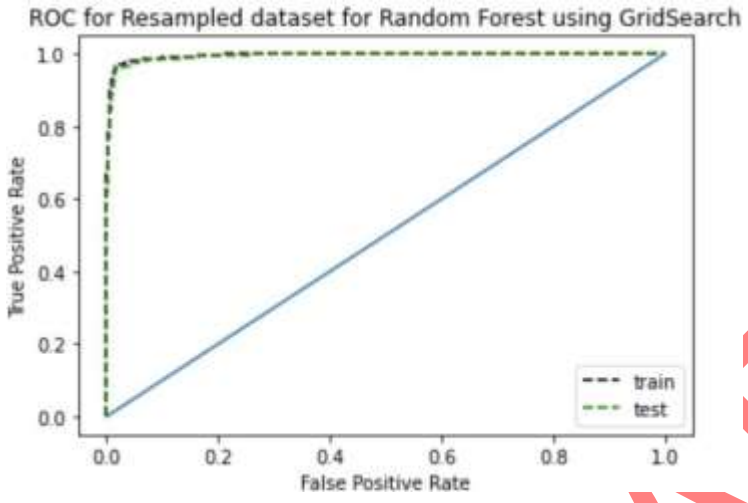
The ROC curve on the Resampled Test Dataset is given below:



The comparison of the RF Model on the resampled train and test datasets is given below:

	RFC_Grid_Resampled_Train	RFC_Grid_Resampled_Test
Recall	0.967	0.961
F1 Score	0.974	0.970
Precision	0.981	0.979
Accuracy	0.974	0.970
auc	0.996	0.994

The ROC curves of both the Resampled Train and test datasets are shown below:



Observations

It is seen that the model is pretty stable in performance on both Train and Test datasets

The table below compares the performance of all the Random Forest models explored in this study:

	RFC_Base_Train	RFC_Base_Test	RFC_Grid_Train	RFC_Grid_Test	RFC_Grid_Resampled_Train	RFC_Grid_Resampled_Test
Recall	1.0	0.859	0.892	0.859	0.967	0.961
F1 Score	1.0	0.880	0.911	0.884	0.974	0.970
Precision	1.0	0.902	0.931	0.909	0.981	0.979
Accuracy	1.0	0.975	0.981	0.976	0.974	0.970
auc	1.0	0.991	0.993	0.986	0.996	0.994

Observations

- The table clearly shows that the model performance on the Resampled data using SMOTE is vastly superior to the other RFC models explored in this study.
- It is seen that the Grid-based model is only very marginally better than the Base RFC model on F1 Score and Precision metrices.

1.10 Build a LDA Model on Train Dataset

Base Model:

This model was built using the default values for the hyper-parameters. The results are as follows:

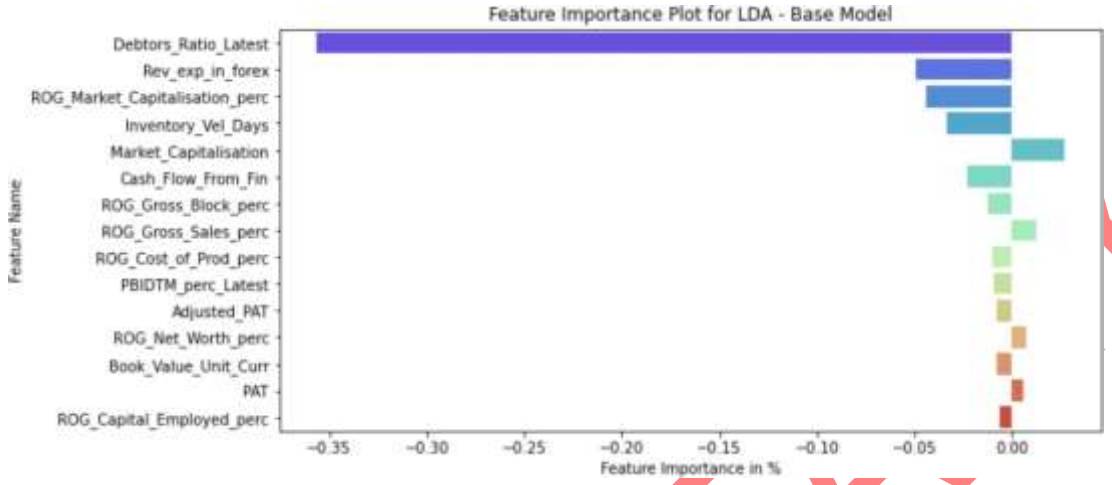
	precision	recall	f1-score	support
0	0.93	0.98	0.95	2143
1	0.70	0.37	0.48	259
accuracy			0.91	2402
macro avg	0.81	0.68	0.72	2402
weighted avg	0.90	0.91	0.90	2402

The confusion matrix on the Train dataset is as follows:

The features in the descending order of importance are as follows (top 10):

	Coeff	Abs_coeff
Debtors_Ratio_Latest	-0.356345	0.356345
Rev_exp_in_forex	-0.049358	0.049358
ROG_Market_Capitalisation_perc	-0.044368	0.044368
Inventory_Vel_Days	-0.033291	0.033291
Market_Capitalisation	0.026563	0.026563
Cash_Flow_From_Fin	-0.022711	0.022711
ROG_Gross_Block_perc	-0.012592	0.012592
ROG_Gross_Sales_perc	0.012352	0.012352
ROG_Cost_of_Prod_perc	-0.010405	0.010405
PBIDTM_perc_Latest	-0.009298	0.009298

Please note that they are sorted by absolute values of the coefficients. This is also seen visually below:



Observations

- Debtors_Ratio_Latest has the strongest influence (negative) on default – this means that companies with low Debtors_Ratio_Latest will have the probability to default closer to 1, and are therefore, more likely to default (given that default = 1 means default). Alternately, a high Debtors_Ratio_Latest value would mean a value closer to zero, and therefore, less likely to default.
- However, the model is not performing well on Recall, and is therefore, not a suitable model.

Impact of SMOTE on LDA Model performance

There is a class imbalance in the original dataset, with only about 10.8% of the records having default = 1. So, it was explored whether SMOTE could improve model performance.

LDA Model on Resampled Train Dataset:

The classification reports for Resampled Train dataset is given below:

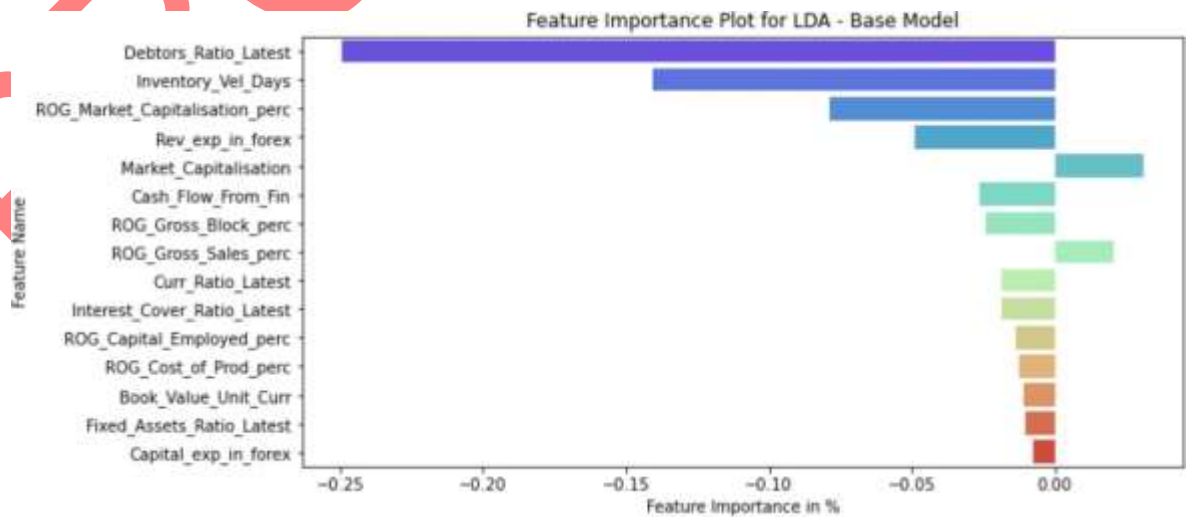
	precision	recall	f1-score	support
0	0.92	0.82	0.87	2143
1	0.84	0.93	0.88	2143
accuracy			0.87	4286
macro avg	0.88	0.87	0.87	4286
weighted avg	0.88	0.87	0.87	4286

The confusion matrix for the Resampled Test dataset is given below:

The features in the descending order of importance are given below (top-10):

	Coeff	Abs_coeff
Debtors_Ratio_Latest	-0.249161	0.249161
Inventory_Vel_Days	-0.140323	0.140323
ROG_Market_Capitalisation_perc	-0.078832	0.078832
Rev_exp_in_forex	-0.049034	0.049034
Market_Capitalisation	0.030884	0.030884
Cash_Flow_From_Fin	-0.026436	0.026436
ROG_Gross_Block_perc	-0.024002	0.024002
ROG_Gross_Sales_perc	0.020820	0.020820
Curr_Ratio_Latest	-0.018678	0.018678
Interest_Cover_Ratio_Latest	-0.018500	0.018500

It is seen in a graphical output below:



Observations

- The performance of the LDA model on the Resampled train dataset is vastly improved over the original dataset (Recall value improved from 0.37 to 0.93)
- While Debtors_Ratio_Latest continued to be the most influential, the order of the other coefficients are different in the two models. However, in both cases, the top 4 features / variable remain the same.

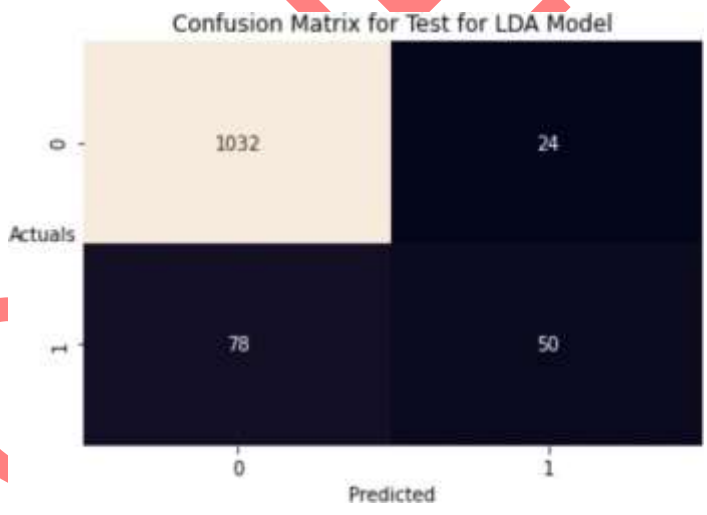
1.11 Validate the LDA Model on test Dataset and state the performance matrices

Base LDA Model

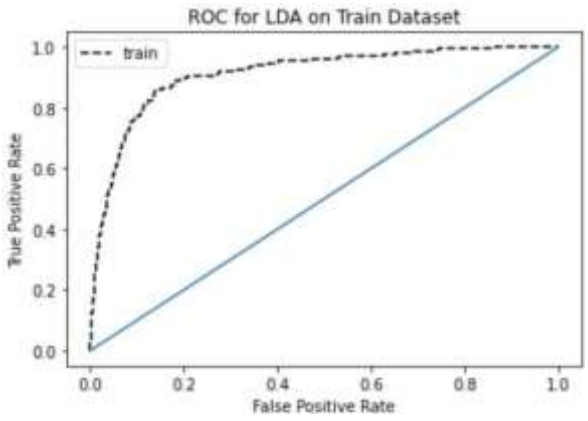
The classification report of the Base LDA model on test dataset is given below:

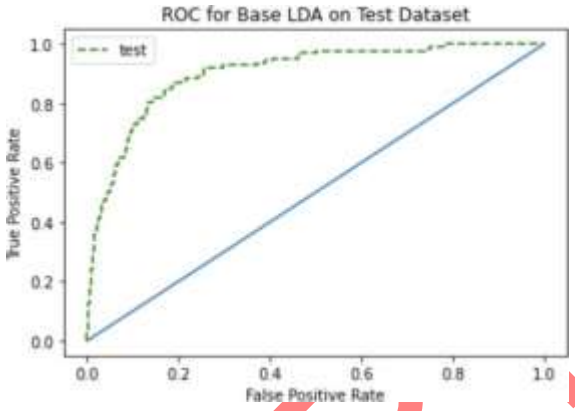
	precision	recall	f1-score	support
0	0.93	0.98	0.95	1056
1	0.68	0.39	0.50	128
accuracy			0.91	1184
macro avg	0.80	0.68	0.72	1184
weighted avg	0.90	0.91	0.90	1184

The confusion matrix on the test dataset is given below:



The ROC curve is given below:





The comparison of the LDA model on the Train and Test datasets is given below:

	LDA_Train	LDA_Test
Recall	0.371	0.391
F1 Score	0.484	0.495
Precision	0.696	0.676
Accuracy	0.915	0.914
auc	0.912	0.903

Observations

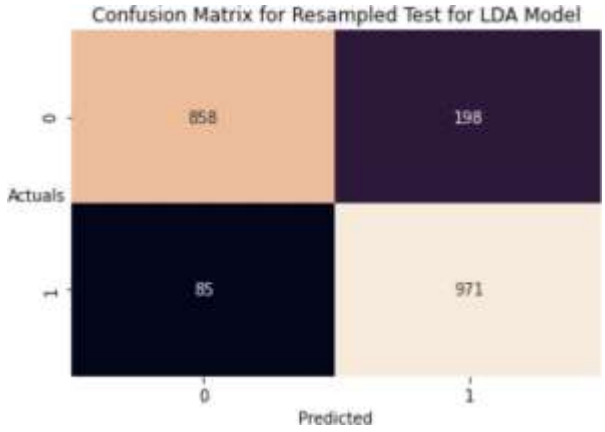
- The Base Model is stable across the Train and Test datasets, with no overfitting
- Recall performance is very poor, though overall Accuracy is good.

LDA Model on Resampled Train Dataset:

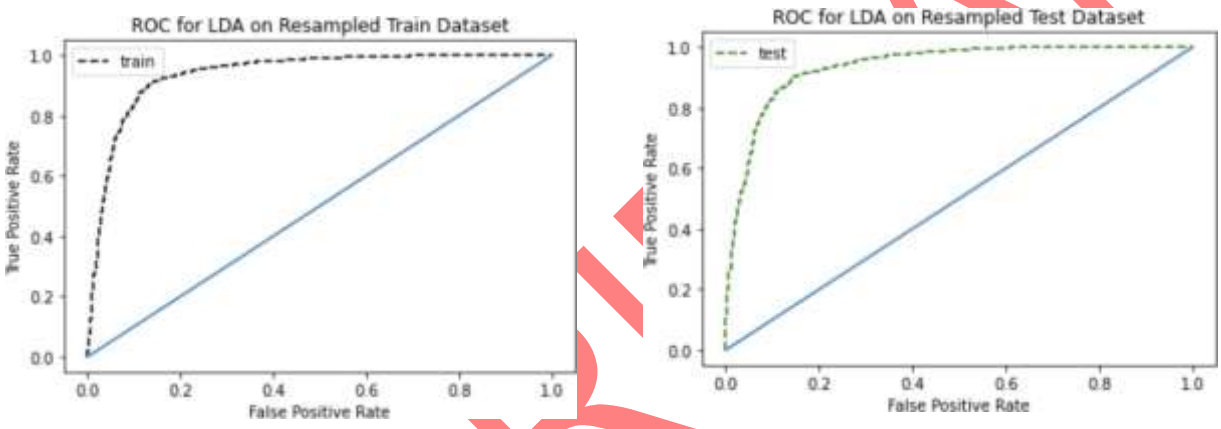
The classification report of the LDA model on resampled test dataset is given below:

	precision	recall	f1-score	support
0	0.91	0.81	0.86	1056
1	0.83	0.92	0.87	1056
accuracy			0.87	2112
macro avg	0.87	0.87	0.87	2112
weighted avg	0.87	0.87	0.87	2112

The confusion matrix on the resampled test dataset is given below:



The ROC curve on the resampled test/train dataset is given below:



The comparison of the performance of the resampled Train and Test datasets is given in the table below:

	LDA_Train	LDA_Test	LDA_Train_Resampled	LDA_Test_Resampled
Recall	0.371	0.391	0.931	0.920
F1 Score	0.484	0.495	0.881	0.873
Precision	0.696	0.676	0.837	0.831
Accuracy	0.915	0.914	0.875	0.866
auc	0.912	0.903	0.939	0.938

Observations

- It is seen that Recall significantly improved on the Resampled Train dataset.
- The model is also stable with comparable performances on train and test datasets

1.12 Compare the performances Logistics, Random Forest and LDA models

In Milestone 1, the Logistic Regression Models were built using 2 approaches. The first one involved excluding the highly correlated independent variables using the VIF criterion (exclude if VIF > 5) and then building the models. Only those features whose p value < 0.05 were considered for the subsequent iterations. This was called Model A.

The second approach was to use all the independent variables and build the first model. Exclude all variables with p > 0.05 for the next iteration, and continue till all variables used in the model had p < 0.05. This was Model B.

The most important measure of model performance was Recall, followed by Precision and Accuracy. Accordingly, both the Models (A and B) were evaluated. Model B was found to be a simpler and better model. A cutoff value of 0.15 was arrived at as the best cutoff to maximize Recall with an acceptable Precision and Accuracy scores.

Comparison of Models

The table below gives a comparison of the different models explored in this study on both the train and test datasets:

	Recall	F1 Score	Precision	Accuracy	auc
RFC_Base_Train	1.000	1.000	1.000	1.000	1.000
RFC_Base_Test	0.859	0.880	0.902	0.975	0.991
RFC_Grid_Train	0.892	0.911	0.931	0.981	0.993
RFC_Grid_Test	0.859	0.884	0.909	0.976	0.986
RFC_Grid_Resampled_Train	0.967	0.974	0.981	0.974	0.996
RFC_Grid_Resampled_Test	0.961	0.970	0.979	0.970	0.994
LDA_Train	0.371	0.484	0.696	0.915	0.912
LDA_Test	0.391	0.495	0.676	0.914	0.903
LDA_Train_Scaled	0.371	0.484	0.696	0.915	0.912
LDA_Test_Scaled	0.375	0.482	0.676	0.913	0.904
LDA_Train_Resampled	0.931	0.881	0.837	0.875	0.939
LDA_Test_Resampled	0.920	0.873	0.831	0.866	0.938
Log_Cutoff_0.15	0.914	0.665	0.522	0.900	1.000
Log_Cutoff_0.28	0.859	0.794	0.738	0.950	0.960

1.13 State Recommendations from the above models

- It is interesting to see that Random Forest Classifier emerged as the best fitting model even without correcting for class imbalance in the dataset. With more datapoints, the model can become even stronger without the need for SMOTE
- In general, higher the value of the feature importance, more is its influence on the dependent variable. However, when the independent variables are highly

correlated with each other, interpreting this becomes difficult.

- On the other hand, Logistic Regression and LDA models are relatively simpler to interpret.

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

The dataset for this exercise is loaded as a time series dataset as seen below:

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

There are no missing values

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 314 entries, 2014-03-31 to 2020-03-30
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Infosys              314 non-null    int64
 1   Indian Hotel         314 non-null    int64
 2   Mahindra & Mahindra  314 non-null    int64
 3   Axis Bank            314 non-null    int64
 4   SAIL                 314 non-null    int64
 5   Shree Cement         314 non-null    int64
 6   Sun Pharma           314 non-null    int64
 7   Jindal Steel         314 non-null    int64
 8   Idea Vodafone        314 non-null    int64
 9   Jet Airways          314 non-null    int64
 10  Year                 314 non-null    int64
 11  Month                314 non-null    int64
 12  Month_abbr           314 non-null    object
 13  Week                 314 non-null    int64
dtypes: int64(13), object(1)
memory usage: 36.8+ KB
```

The description of the dataset is given below

	Infosys	Indian Hotel	Mahindra & Mahindra	Axis Bank	SAIL	Shree Cement	Sun Pharma	Jindal Steel	Idea Vodafone	Jet Airways	Year	Month	Week
count	314.00	314.00	314.00	314.00	314.00	314.00	314.00	314.00	314.00	314.00	314.00	314.00	314.00
mean	511.34	114.56	636.68	540.74	59.10	14808.41	633.47	147.63	53.71	372.66	2016.75	6.52	26.54
std	135.95	22.51	102.88	115.84	15.81	4288.28	171.86	65.88	31.25	202.26	1.77	3.45	15.08
min	234.00	64.00	284.00	263.00	21.00	5543.00	338.00	53.00	3.00	14.00	2014.00	1.00	1.00
25%	424.00	96.00	572.00	470.50	47.00	10952.25	478.50	88.25	25.25	243.25	2015.00	4.00	14.00
50%	466.50	115.00	625.00	528.00	57.00	16018.50	614.00	142.50	53.00	376.00	2017.00	7.00	26.50
75%	630.75	134.00	678.00	605.25	71.75	17773.25	785.00	182.75	82.00	534.00	2018.00	9.75	39.75
max	810.00	157.00	956.00	808.00	104.00	24806.00	1089.00	338.00	117.00	871.00	2020.00	12.00	53.00

Observations

- There are 314 records from 31-03-2014 to 30-03-2020
- The data is at a weekly frequency
- Based on absolute stock prices, Shree Cement seems to be having the maximum price of 24806 and the minimum stock price was for Idea Vodafone which was 3

2.2 Stock Price Graph for Infosys

The stock price graph for Infosys is given below:

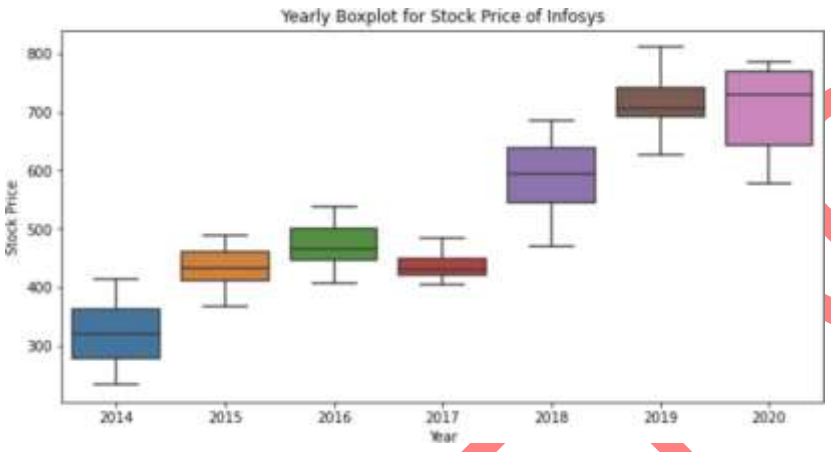
A scatter plot is seen below:

Observations

It is seen that overall, the stock price is increasing over the years. Its value is appreciating, though the most recent prices do show a large variation (or high volatility in 2020)

In 2017 also, the stock prices had decreased, but then, started increasing in 2018.

The boxplot of the year for Stock prices of Infosys is shown below:



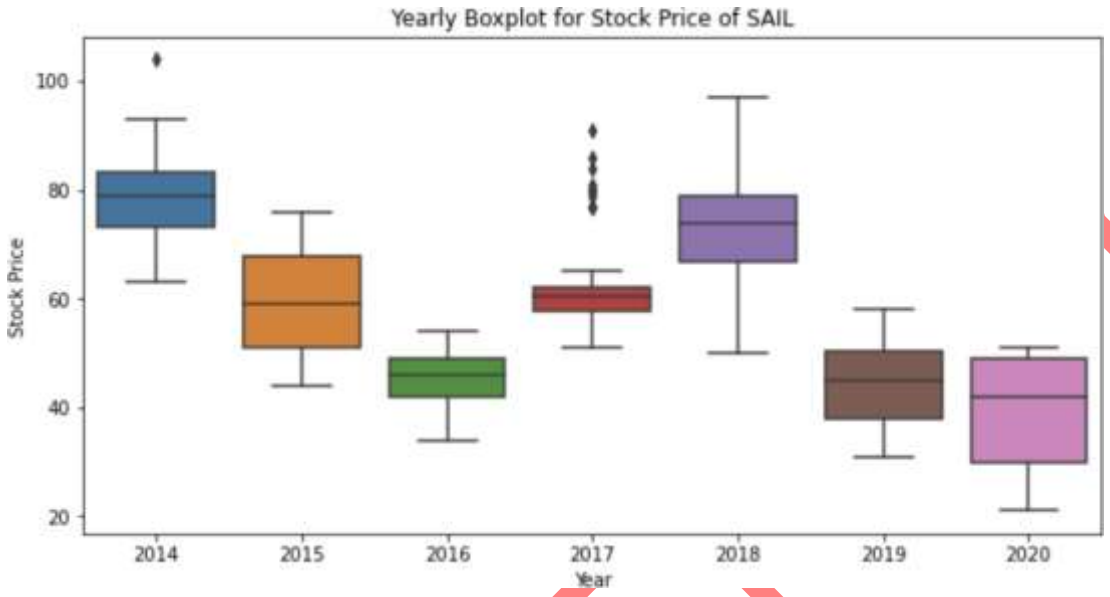
The boxplot clearly shows the increased volatility of the stock in 2020.

Stock Price Graph for SAIL

The stock price movement for SAIL is given below:

The stock prices seem to be on a downward trend after a high in 2018, with a very sharp erosion in the stock price in 2020.

The Yearly Boxplot of SAIL stocks is seen below:



The yearly boxplot also shows the erosion in stock price of SAIL. It also shows the relatively high volatility in 2020, with a left-skewed distribution.

Some outliers are seen in 2017, which means the stock prices had reached very high prices compared to its median.

2.3 Calculate Returns for all stocks

Approach:

- A subset of the original dataset containing only the stock prices of all the companies is created
- Then, the log of the stock prices is taken on this subset
- The difference is then calculated.

Results

Subset of the original dataset containing only the stock prices:

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

The top 5 records of the returns are shown below:

Date	Infosys	Indian Hotel	Mahindra & Mahindra	Axis Bank	SAIL	Shree Cement	Sun Pharma	Jindal Steel	Idea Vodafone	Jet Airways
2014-03-31	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2014-04-07	-0.026873	-0.014599	0.006572	0.048247	0.028988	0.032831	0.094491	-0.065682	0.011976	0.086112
2014-04-14	-0.011742	0.000000	-0.008772	-0.021979	-0.028988	-0.013888	-0.004930	0.000000	-0.011976	-0.078943
2014-04-21	-0.003945	0.000000	0.072218	0.047025	0.000000	0.007583	-0.004955	-0.018084	0.000000	0.007117
2014-04-28	0.011788	-0.045120	-0.012371	-0.003540	-0.076373	-0.019515	0.011523	-0.140857	-0.049393	-0.148846

As expected, the first record is NaN (due to the definition of difference). This has not been dropped to illustrate this effect.

2.4 Calculate Stock Means and Standard Deviation for all stocks

Stock Means

This is the average returns that the stock is making on a week-to-week basis. Since the dataset is already on a weekly basis, the mean can be derived directly from the column means. Higher the Stock mean, means Higher Return. The results in descending order of means are as follows:

	Mean Return
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

Observations

- Shree Cement gives the highest Average Returns, followed by Infosys and Axis Bank. All three have positive Average returns, meaning that these stocks are appreciating over time.
- Idea Vodafone has the least Average Returns, followed by Jet Airways and Jindal Steel. All three have negative returns, meaning that their stock prices are decreasing with time. These stocks are depreciating.

Standard Deviation

Standard Deviation is a measure of volatility of the stock prices. Higher the value of the standard deviation, more is the variation of the stock's returns from its average returns. Higher Standard Deviation means Higher Risk.

The results are sorted in ascending order, i.e. least volatile stocks are on the top, and the volatility increases as one goes down the list. This indicates that the returns on the stocks at the top of the list are more stable than the ones below.

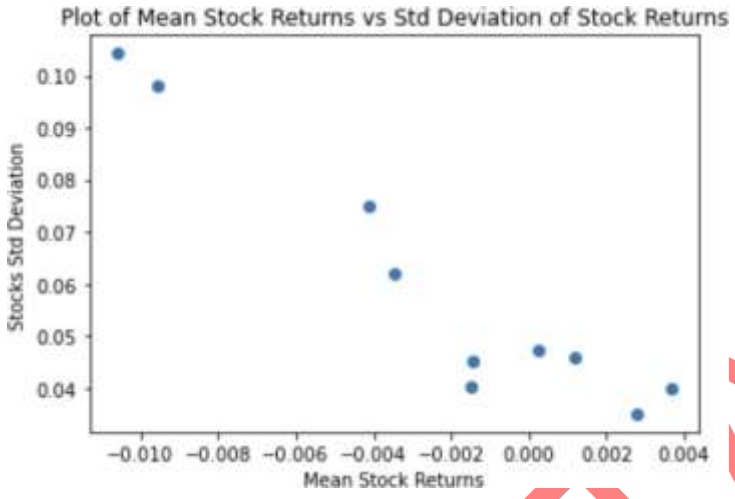
	Std Deviation
Infosys	0.035070
Shree Cement	0.039917
Mahindra & Mahindra	0.040169
Sun Pharma	0.045033
Axis Bank	0.045828
Indian Hotel	0.047131
SAIL	0.062188
Jindal Steel	0.075108
Jet Airways	0.097972
Idea Vodafone	0.104315

Observations

- Infosys has the least standard deviation, and therefore, has the least volatility in stock returns. It gives the most stable returns amongst all the stocks explored in this dataset. This has the lowest risk.
- Shree Cement and Mahindra & Mahindra provide the next most-stable returns
- Idea Vodafone has the maximum Std Deviation, and therefore, the maximum volatility in stock returns. It is followed by Jet Airways and Jindal Steel.

2.5 Draw a plot of Stock Means vs Standard Deviation

The plot is seen below:



This is also summarized in the table below (sorted in descending order by Mean Return)

	Mean Return	std
Shree Cement	0.003681	0.039917
Infosys	0.002794	0.035070
Axis Bank	0.001167	0.045828
Indian Hotel	0.000266	0.047131
Sun Pharma	-0.001455	0.045033
Mahindra & Mahindra	-0.001506	0.040169
SAIL	-0.003463	0.062188
Jindal Steel	-0.004123	0.075108
Jet Airways	-0.009548	0.097972
Idea Vodafone	-0.010608	0.104315

2.6 Conclusions and Recommendations

High Risk Stocks

Stocks with low Mean Return and high Standard deviation fall in this category. These are the stocks on the top half of the graph (high std deviation – y axis and represent High Risk Stocks.

In the present case, Idea Vodafone, Jet Airways, Jindal Steel and SAIL all have relatively high Standard Deviation but negative Mean Return. These are especially High Risk with

average low mean returns, and could be dropped from the portfolio.

On the other hand, Sun Pharma and Mahindra & Mahindra have a negative mean return, but a relatively lower standard deviation. From a pure Risk perspective, Sun Pharma is good, followed by Mahindra & Mahindra.

It is recommended that these stocks be kept in the portfolio if there is an appetite of High Risk.

High Returns Stocks

Stocks with high Mean return and low Standard Deviation are preferred in this strategy. They not only appreciate in value over time, but are also very stable and have a relatively higher predictable return. These are stocks which are on the bottom right of the graph.

In the current portfolio, Shree Cement is the best performing stock with the highest Mean Return and second-lowest Standard Deviation (behind Infosys). Infosys is the second best, followed by Axis Bank and Indian Hotel.

It is recommended that more funds be allotted to these stocks, and the stock portfolio be enhanced by stocks from other similar performing stocks.

Appendix:

FRA: Milestone-1

```
df.duplicated().sum()
```

```
0
```

```
# Drop Co_Name, Co_Code since it is not required
df.drop(['Co_Name', 'Co_Code'], axis = 1, inplace = True)
df.info()
```

Outlier Treatment

```
# Identify Outliers

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

d = {'lower': (df < Q1 - 1.5 * IQR).sum(), 'upper': ((df > Q3 + 1.5 * IQR).sum())}
outlier = pd.DataFrame(data = d)
outlier['total'] = outlier['upper'] + outlier['lower']
outlier
```

```
print("Outlier = 1.5 * (75th - 25th)")
print()
print('No of Lower Outliers = ', outlier['lower'].sum())
print('No of Upper Outliers = ', outlier['upper'].sum())
```

```
# Identify Outliers

Q1 = df.quantile(0.10)
Q3 = df.quantile(0.90)
IQR = Q3 - Q1

d = {'lower': (df < Q1 - 1.5 * IQR).sum(), 'upper': ((df > Q3 + 1.5 * IQR).sum())}
outlier2 = pd.DataFrame(data = d)
outlier2['total'] = outlier2['upper'] + outlier2['lower']
outlier2
```

```
print("Outlier = 1.5 * (90th - 10th)")
print()
print('No of Lower Outliers = ', outlier2['lower'].sum())
print('No of Upper Outliers = ', outlier2['upper'].sum())
```

```
# Identify Outliers

Q1 = df.quantile(0.05)
Q3 = df.quantile(0.95)
IQR = Q3 - Q1

d = {'lower': (df < Q1 - 1.5 * IQR).sum(), 'upper': ((df > Q3 + 1.5 * IQR).sum())}
outlier3 = pd.DataFrame(data = d)
outlier3['total'] = outlier3['upper'] + outlier3['lower']
outlier3
```

```
print("Outlier = 1.5 * (95th - 5th)")
print()
print('No of Lower Outliers = ', outlier3['lower'].sum())
print('No of Upper Outliers = ', outlier3['upper'].sum())
```

```
# Check how many records remain if we drop outliers of all variables without any transformation

df1 = df.copy()

Q1 = df1.quantile(0.25)
Q3 = df1.quantile(0.75)
IQR = Q3 - Q1


for col in df1.columns:
    low = Q1[col] - 1.5 * IQR[col]
    upper = Q3[col] + 1.5 * IQR[col]
    df1.drop(df1[df1[col] < low].index, inplace = True)
    df1.drop(df1[df1[col] > upper].index, inplace = True)

len(df1)
```

```
# Cap the outliers at 5th and 95th percentile

df1 = df.copy()

Q05 = df1.quantile(0.05)
Q95 = df1.quantile(0.95)

Q25 = df1.quantile(0.25)
Q75 = df1.quantile(0.75)
IQR = Q75 - Q25

for col in df1.columns:
    low = Q25[col] - 1.5 * IQR[col]
    upper = Q75[col] + 1.5 * IQR[col]
    df1[col] = np.where(df1[col] < low, Q05[col], df1[col])
    df1[col] = np.where(df1[col] > upper, Q95[col], df1[col])

# Check if all values are within the limits

len(df1)
```

```
# Check that median does not change

Q50_outliers = df1.quantile(0.50)
Q50_originals = df.quantile(0.50)
(Q50_outliers != Q50_originals).sum().sum()
```

```
0
```

```
# Confirm presence of missing values

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

```
118
```

```
missing = pd.DataFrame(df1.isna().sum(), columns = ['No of Missing Values'])
miss_data = list(missing[missing > 0].index)
```

```
missing.dropna(inplace = True)
missing[missing['No of Missing Values'] > 0].sort_values(by = 'No of Missing Values', ascending = False)
```

```
# Use median to impute the values
cols = list(df2)

imputer = SimpleImputer(missing_values=np.nan, strategy='median')

df2 = pd.DataFrame(imputer.fit_transform(df2))
df2.columns = cols
df2.isna().sum().sum()
```

```
0
```

```
df2.isna().sum().sum()
```

```
0
```

```
df2['default'] = np.where(df2['Networth_Next_Year'] < 0,1,0)
print('The distribution of the default values is ')
print(df2['default'].value_counts())
```

```
sep_median = df2.groupby( by = 'default').median().T
#sep_median['Separation'] = abs((sep_median['1'] - sep_median['0']*100) / sep_median['1'])
sep_median.rename(index = {'0': 'No_default', '1': 'default'}, inplace = True)
sep_median
```

```
cols = list(df2)
j = 0
fig, ax = plt.subplots(len(cols), figsize=(8,250))

for i, col_name in enumerate(cols):
    sns.boxplot(y = col_name, x = 'default', data = df2, ax = ax[i])
    ax[i].set_title('Box plot - {}'.format(col_name), fontsize=10)
    ax[i].set_xlabel('default', fontsize=8)
    plt.tight_layout()
```

```
cols = list(df2)
j = 0
fig, ax = plt.subplots(len(cols), figsize=(8,150))

for i, col_name in enumerate(cols):
    sns.barplot(y = col_name, x = 'default', data = df2, ax = ax[i])
    ax[i].set_title('Bar plot - {}'.format(col_name), fontsize=10)
    ax[i].set_xlabel('default', fontsize=8)
    #plt.tight_layout()
```

```
# Bivariate Plots

plt.figure(figsize = (15,15))
#sns.pairplot(df2)
```

```
<Figure size 1080x1080 with 0 Axes>

<Figure size 1080x1080 with 0 Axes>
```

```
plt.figure(figsize = (10,10))
sns.heatmap(df2.corr())
```

```
corr = df2.corr()
cornw = pd.DataFrame(corr['Networth_Next_Year'])
cornw['Abs_Corr'] = abs(cornw['Networth_Next_Year'])
cornw.sort_values(by = 'Abs_Corr', ascending = False, inplace = True)
cornw1 = cornw.drop('Networth_Next_Year', axis = 1)
plt.figure(figsize = (10,8))
plt.xlabel('Abs Corr Value')
sns.heatmap(cornw1)
```

Train_Test Split

```
x = df2.drop(['Networth_Next_Year', 'default'], axis = 1)
y = df2['default']

x_train, x_test, y_train, y_test = tts(x,y,test_size=0.33,random_state=42,stratify = y)

train = pd.concat([x_train,y_train], axis=1)
test = pd.concat([x_test,y_test], axis=1)

train.shape, test.shape
```

```
def vif_calc(X):

    vif_table = pd.DataFrame()
    vif_table["Variables"] = X.columns
    vif_table["VIF"] = [vif(X.values, i) for i in range(X.shape[1])]

    return(vif_table)
```

```
vif_table = vif_calc(x_train).sort_values( by = 'VIF')
var_drop = vif_table[vif_table['VIF'] > 5]
var_retain = vif_table[vif_table['VIF'] <= 5]
# calc_vif(X_train).sort_values(by='VIF', ascending = True)
```

```
print('No of Variables with VIF > 5 and therefore not used for model-building is ', len(var_drop))
print('No of Variables with VIF <= 5 and therefore used for model-building is ', len(var_retain))
```

```
var_1 = list(var_retain['Variables'])
var_1[1]
```

```
'ROG_Rev_earn_in_forex_perc'
```

```
range(len(var_1)); var_1[0]
```

```
'ROG_CP_perc'
```

```
f1 = 'default ~ '

for i in range(len(var_1)):
    f1 = f1 + ' + ' + var_1[i]

print('The independent variables considered for the first iteration are as follows ', '\n')
print(f1)
```

```
The independent variables considered for the first iteration are as follows

default ~  + ROG_CP_perc + ROG_Rev_earn_in_forex_perc + ROG_Capital_Employed_perc + ROG_Market_Capitalisation_perc + ROG_Rev_ex
p_in_forex_perc + Debtors_Vel_Days + Fixed_Assets_Ratio_Latest + Creditors_Vel_Days + APATM_perc_Latest + Inventory_Ratio_Lates
t + Total_Asset_Turnover_Ratio_Latest + ROG_Net_Sales_perc + Cash_Flow_From_Fin + Cash_Flow_From_Inv + CP + Rev_exp_in_forex +
Equity_Paid_Up + Selling_Cost + Other_Income + Rev_earn_in_forex + Cash_Flow_From_Opr + Book_Value_Adj_Unit_Curr + ROG_Cost_of_
Prod_perc + ROG_Net_Worth_perc + Market_Capitalisation + Total_Debt + Net_Working_Capital + CEPS_annualised_Unit_Curr
```

```
model1 = sm.logit(formula = f1, data=train).fit()
model1.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.226718
         Iterations 10
```

```
p_values1 = pd.DataFrame(model1.pvalues, columns = ['p_value'])
p_values1[p_values1['p_value'] > 0.05].sort_values(by = 'p_value', ascending = False)
```

```
var_2 = p_values1[p_values1['p_value'] < 0.05]
iter_2 = list(var_2.index[1:])
var_2[1:]
```

```
f2 = 'default ~ '
for i in range(len(iter_2)):
    f2 = f2 + ' + ' + iter_2[i]

f2
```

```
'default ~  + ROG_Capital_Employed_perc + ROG_Market_Capitalisation_perc + Debtors_Vel_Days + APATM_perc_Latest + Inventory_Rat
io_Latest + Total_Asset_Turnover_Ratio_Latest + ROG_Net_Sales_perc + Cash_Flow_From_Fin + Book_Value_Adj_Unit_Curr + Market_Cap
italisation + Total_Debt + Net_Working_Capital'
```

```
model2 = sm.logit(formula = f2, data=train).fit()
model2.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.233598
         Iterations 9
```

```
vif_table_2 = vif_calc(train[iter_2]).sort_values( by = 'VIF')
```

```
print('VIF values for Variables used in Iteration 2')
vif_table_2
```

```
VIF values for Variables used in Iteration 2
```

```
print('The adjusted pseudo R-square value of Iteration 2 is', round((1 - ((model2.llf - model2.df_model)/model2.llnull)),4))
```

```
The adjusted pseudo R-square value of Iteration 2 is 0.3023
```

```
# Plot of fitted values

sns.distplot(model2.fittedvalues)
plt.title('Plot of Fitted Values');
```

```
# Cutoff of 0.11

y_class_predict = np.where(y_train_predict > 0.11,1,0)
```

```
y_train_predict[:5]
```

```
3221    0.000020
2699    0.005311
881     0.198663
100     0.181214
867     0.137535
dtype: float64
```

```
tn1, fp1, fn1, tp1 = confusion_matrix(y_train,y_class_predict).ravel()
sns.heatmap((confusion_matrix(y_train,y_class_predict)),annot=True,fmt='.5g', cbar = False);
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0)
plt.title('Confusion Matrix for Train for cutoff = 0.11');
```

```
round((tn1 + tp1)/(tn1 + tp1 + fn1 + fp1),2)
```

```
print(classification_report(y_train, y_class_predict))
```

```
# Cutoff of 0.13

y_class_predict_2 = np.where(y_train_predict > 0.13,1,0)
tn2, fp2, fn2, tp2 = confusion_matrix(y_train,y_class_predict_2).ravel()
sns.heatmap((confusion_matrix(y_train,y_class_predict_2)),annot=True,fmt='.5g', cbar = False);
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0)
plt.title('Confusion matrix on train for cutoff = 0.13');
```

```
# Cutoff of 0.15

y_class_predict_3 = np.where(y_train_predict > 0.15,1,0)
tn3, fp3, fn3, tp3 = confusion_matrix(y_train,y_class_predict_3).ravel()
sns.heatmap((confusion_matrix(y_train,y_class_predict_3)),annot=True,fmt='.5g', cbar = False);
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0)
plt.title('Confusion matrix on Train for cutoff = 0.15');
```

```
# Cutoff of 0.17

y_class_predict_4 = np.where(y_train_predict > 0.17,1,0)
tn4, fp4, fn4, tp4 = confusion_matrix(y_train,y_class_predict_4).ravel()
sns.heatmap((confusion_matrix(y_train,y_class_predict_4)),annot=True,fmt='.5g', cbar = False);
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0)
plt.title('Confusion matrix on train for cutoff = 0.17');
```

```
# Cutoff of 0.19

y_class_predict_5 = np.where(y_test_predict > 0.19,1,0)
tn5, fp5, fn5, tp5 = confusion_matrix(y_test,y_class_predict_5).ravel()
sns.heatmap((confusion_matrix(y_test,y_class_predict_5)),annot=True,fmt='.5g', cbar = False);
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0)
plt.title('Confusion Matrix for Test for threshold = 0.19');
```

FRA: Milestone-2

```
# con will contain all the models on the training dataset

index=['Recall', 'F1 Score', 'Precision','Accuracy', 'auc']
#columns = ['Cutoff_1', 'Cutoff_2', 'Cutoff_3']
con = pd.DataFrame(index = index)
con
```

Recall
F1 Score
Precision
Accuracy
auc

```
# test will contain all the models on the test dataset

index=['Recall', 'F1 Score', 'Precision','Accuracy', 'auc']
#columns = ['Cutoff_1', 'Cutoff_2', 'Cutoff_3']
test = pd.DataFrame(index = index)
test
```

```
# models will contain the test performances of all models

models = test.copy()
models
```

```
# Base Model

rfc1 = rfc(random_state = 1)

rfc_model_base = rfc1.fit(x_train, y_train)
rfc_model_base
```

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

```
y_train_predict_base = rfc_model_base.predict(x_train)
y_test_predict_base = rfc_model_base.predict(x_test)

y_train_prob_base = rfc_model_base.predict_proba(x_train)[:,1]
y_test_prob_base = rfc_model_base.predict_proba(x_test)[:,1]

print(classification_report(y_train, y_train_predict_base))
```

```
# Train dataset metrics

rfc_metrics_base = classification_report(y_train, y_train_predict_base, output_dict = True)
recall_base = round(rfc_metrics_base['1']['recall'],3)
f1score_base = round(rfc_metrics_base['1']['f1-score'],3)
precision_base = round(rfc_metrics_base['1']['precision'],3)
accuracy_base = rfc_model_base.score(x_train, y_train).round(3)
roc_base = round(roc_auc_score(y_train, y_train_prob_base),3)

con['RFC_Base_Train'] = [recall_base, f1score_base, precision_base, accuracy_base, roc_base]
```

```
sns.heatmap((confusion_matrix(y_train,y_train_predict_base)),annot=True,fmt='.5g', cbar = False);
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0)
plt.title('Confusion Matrix for Train for Base Random Forest Model');
```

```
rfc_train_fpr, rfc_train_tpr, rfc_train_thresholds = roc_curve(y_train,y_train_prob_base)
#rfc_test_fpr, rfc_test_tpr, rfc_test_thresholds = roc_curve(y_test,y_test_prob)

plt.plot(rfc_train_fpr, rfc_train_tpr,color='black', linestyle = '--', label = 'train')
#plt.plot(rfc_test_fpr, rfc_test_tpr,color='green', linestyle = '--', label = 'test')

plt.plot([0, 1], [0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for Base Random Forest')
plt.legend(loc = 'best');
```

```
# Test dataset metrics

rfc_metrics_base_test = classification_report(y_test, y_test_predict_base, output_dict = True)
recall_base_test = round(rfc_metrics_base_test['1']['recall'],3)
f1score_base_test = round(rfc_metrics_base_test['1']['f1-score'],3)
precision_base_test = round(rfc_metrics_base_test['1']['precision'],3)
accuracy_base_test = rfc_model_base.score(x_test, y_test).round(3)
roc_base_test = round(roc_auc_score(y_test, y_test_prob_base),3)

con['RFC_Base_Test'] = [recall_base_test, f1score_base_test, precision_base_test, accuracy_base_test, roc_base_test]
```

```
rfc_train_fpr, rfc_train_tpr, rfc_train_thresholds = roc_curve(y_train,y_train_prob_base)
rfc_test_fpr, rfc_test_tpr, rfc_test_thresholds = roc_curve(y_test,y_test_prob_base)

plt.plot(rfc_train_fpr, rfc_train_tpr,color='black', linestyle = '--', label = 'train')
plt.plot(rfc_test_fpr, rfc_test_tpr,color='green', linestyle = '--', label = 'test')

plt.plot([0, 1], [0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for Base Random Forest')
plt.legend(loc = 'best');
```

```
features = pd.DataFrame(rfc_model_base.feature_importances_, columns = ["Imp"],
                        index = x_train.columns).sort_values('Imp',ascending=False)
features.head(20)
```

```
rfc1 = rfc()

params = { 'n_estimators' : [100, 200, 250],
           'criterion' : ['gini', 'entropy'],
           'max_depth' : [3, 5],
           'min_samples_leaf' : [8, 15, 20],
           'min_samples_split' : [20, 35, 50],
           'max_features' : [10, 20, 30],
           'oob_score' : [True],
           'random_state' : [1]

         }

rfcgrid = GS(estimator = rfc1, param_grid = params, cv = 3)

rfc_model1 = rfcgrid.fit(x_train, y_train)
```

```
rfc_model1.best_params_
```

```
rfc_model = rfc_model2.best_estimator_
```

```
rfc_model
```

```
RandomForestClassifier(criterion='entropy', max_depth=3, max_features=10,
                       min_samples_leaf=8, min_samples_split=15,
                       n_estimators=200, oob_score=True, random_state=1)
```

```
y_train_predict_rfc = rfc_model.predict(x_train)
y_test_predict_rfc = rfc_model.predict(x_test)

y_train_prob_rfc = rfc_model.predict_proba(x_train)[:,1]
y_test_prob_rfc = rfc_model.predict_proba(x_test)[:,1]

print(classification_report(y_train, y_train_predict_rfc))
```

```
sns.heatmap((confusion_matrix(y_train,y_train_predict_rfc)),annot=True,fmt='.5g', cbar = False);
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0)
plt.title('Confusion Matrix for Train for Grid Search Random Forest Model');
```

```
rfc_train_fpr, rfc_train_tpr, rfc_train_thresholds = roc_curve(y_train,y_train_prob_rfc)
#rfc_test_fpr, rfc_test_tpr, rfc_test_thresholds = roc_curve(y_test,y_test_prob)

plt.plot(rfc_train_fpr, rfc_train_tpr,color='black', linestyle = '--', label = 'train')
#plt.plot(rfc_test_fpr, rfc_test_tpr,color='green', linestyle = '--', label = 'test')

plt.plot([0, 1], [0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for Random Forest using GridSearch on Train dataset')
plt.legend(loc = 'best');
```

```
# Test Data

print(classification_report(y_test, y_test_predict_rfc))
```

```
sns.heatmap((confusion_matrix(y_test,y_test_predict_rfc)),annot=True,fmt='.5g', cbar = False);
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0)
plt.title('Confusion Matrix for Test for Grid Search Random Forest Model');
```

```
rfc_train_fpr, rfc_train_tpr, rfc_train_thresholds = roc_curve(y_train,y_train_prob_rfc)
rfc_test_fpr, rfc_test_tpr, rfc_test_thresholds = roc_curve(y_test,y_test_prob_rfc)

plt.plot(rfc_train_fpr, rfc_train_tpr,color='black', linestyle = '--', label = 'train')
plt.plot(rfc_test_fpr, rfc_test_tpr,color='green', linestyle = '--', label = 'test')

plt.plot([0, 1], [0, 1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC for Random Forest using GridSearch')
plt.legend(loc = 'best');
```

```
features = pd.DataFrame(rfc_model.feature_importances_,
                        columns = ["Imp"], index = x_train.columns).sort_values('Imp',ascending=False)
features.head(20)
```

```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((2402, 64), (1184, 64), (2402,), (1184,))
```

```
y_train.value_counts(normalize = True)
```

```
0    0.892173
1    0.107827
Name: default, dtype: float64
```

```
y_test.value_counts(normalize = True)
```

```
0    0.891892
1    0.108108
Name: default, dtype: float64
```

```
x.shape, y.shape
```

```
((3586, 64), (3586,))
```

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
x_res, y_res = sm.fit_sample(x, y.ravel())
```

```
x_res=pd.DataFrame(x_res)

#Renaming column name of Target variable
y_res=pd.DataFrame(y_res)
y_res.columns = ['default']

df_res = pd.concat([x_res,y_res], axis=1)
```

```
df_res['default'].value_counts(normalize = True)
```

```
1    0.5
0    0.5
Name: default, dtype: float64
```

```
x_res = df_res.drop(['default'], axis = 1)
y_res = df_res['default']

x_train_res, x_test_res, y_train_res, y_test_res = tts(x_res,y_res,test_size=0.33,random_state=42,stratify = y_res)

train_res = pd.concat([x_train_res,y_train_res], axis=1)
test_res = pd.concat([x_test_res,y_test_res], axis=1)

print('The resampled train dataset has ',train_res.shape[0], ' records and the resampled test dataset has ',
      test_res.shape[0], ' records')
```

RFC Model on Resampled Dataset

```
rfc1 = rfc()

params = { 'n_estimators' : [100,150, 200],
           'criterion' : ['entropy'],
           'max_depth' : [1,2,3],
           'min_samples_leaf' : [5, 8, 10],
           'min_samples_split' : [15, 20, 25],
           'max_features' : [5, 7, 10],
           'oob_score' : [True],
           'random_state' : [1]

         }

rfcgrid = GS(estimator = rfc1, param_grid = params, cv = 3)

rfc_model_res = rfcgrid.fit(x_train_res, y_train_res)
```

```
rfc_model_res = rfc_model_res.best_estimator_
rfc_model_res
```

```
RandomForestClassifier(criterion='entropy', max_depth=3, max_features=10,
                       min_samples_leaf=5, min_samples_split=15,
                       n_estimators=150, oob_score=True, random_state=1)
```

```
y_train_predict_rfc = rfc_model_res.predict(x_train_res)
y_test_predict_rfc = rfc_model_res.predict(x_test_res)

y_train_prob_rfc = rfc_model_res.predict_proba(x_train_res)[:,1]
y_test_prob_rfc = rfc_model_res.predict_proba(x_test_res)[:,1]

print(classification_report(y_train_res, y_train_predict_rfc))
```

```
sns.heatmap((confusion_matrix(y_train_res,y_train_predict_rfc)),annot=True,fmt='.5g', cbar = False);
plt.xlabel('Predicted');
plt.ylabel('Actuals',rotation=0)
plt.title('Confusion Matrix for Resampled Train for Grid Search Random Forest Model');
```

Build a LDA Model on Train Dataset

```
lda_model1 = LinearDiscriminantAnalysis()
lda_model1 = lda_model1.fit(x_train, y_train)

y_train_predict_lda = lda_model1.predict(x_train)
y_test_predict_lda = lda_model1.predict(x_test)

y_train_prob_lda = lda_model1.predict_proba(x_train)[:,1]
y_test_prob_lda = lda_model1.predict_proba(x_test)[:,1]

lda_metrics = classification_report(y_train, y_train_predict_lda, output_dict = True)

recall = round(lda_metrics['1']['recall'],3)
f1score = round(lda_metrics['1']['f1-score'],3)
precision = round(lda_metrics['1']['precision'],3)
accuracy = lda_model1.score(x_train, y_train).round(3)
roc = round(roc_auc_score(y_train, y_train_prob_lda),3)

con['LDA_Train'] = [recall, f1score, precision, accuracy, roc]
```

```
abs_coeff = pd.DataFrame(data = abs(lda_model1.coef_[0]), index = x_train.columns, columns = ['Abs_coeff'])
abs_coeff
```

```
coef = pd.DataFrame(lda_model1.coef_[0], columns = ['Coeff'], index = x_train.columns)
coef['Abs_coeff'] = abs_coeff
coef.sort_values(by = 'Abs_coeff', ascending = False, inplace = True)

top_15 = coef.head(15)
top_15
```

```
plt.figure(figsize=(10,5))
sns.barplot(top_15['Coeff'],top_15.index,palette='rainbow')
plt.ylabel('Feature Name')
plt.xlabel('Feature Importance in %')
plt.title('Feature Importance Plot for LDA - Base Model')
plt.show()
```

Market Risk Dataset

Stock Prices vs Time

```
df['Infosys'].plot(figsize = (12,6), color = 'c')
plt.title('Stock Price Movement for Infosys')
plt.xlabel('Year')
plt.ylabel('Stock Price')
plt.grid()
```

```
plt.figure(figsize = (12,6))
plt.scatter(df.index, df['Infosys'])
plt.xlabel('Year')
plt.ylabel('Infosys')
plt.title('Scatter Plot of Infosys Stock Price');
```

```
plt.figure(figsize = (10,5))
sns.barplot(x = 'Year', y = 'Infosys', data = df );
plt.title('Yearly Avg Stock Price of Infosys')
plt.xlabel('Year')
plt.ylabel('Stock Price');
```

```
plt.figure(figsize = (10,5))
sns.boxplot(x = 'Year', y = 'Infosys', data = df)
plt.title('Yearly Boxplot for Stock Price of Infosys')
plt.xlabel('Year')
plt.ylabel('Stock Price');
```

```
df['SAIL'].plot(figsize = (12,6), color = 'c')
plt.title('Stock Price Movement for SAIL')
plt.xlabel('Year')
plt.ylabel('Stock Price')
plt.grid();
```

Returns for all Stocks

```
df1 = df.drop(['Year', 'Month', 'Month_abbr', 'Week'], axis = 1)
df_log = np.log(df1)

df_returns = df_log.diff(axis = 0, periods = 1)
df_returns.shape
```

Stock Means and Std Dev

```
df_returns_mean = pd.DataFrame(df_returns.mean(), columns = ['Mean Return'])
df_returns_mean.sort_values(by = 'Mean Return', ascending = False)
```

```
df_returns_std = pd.DataFrame(df_returns.std(), columns = ['Std Deviation'])
df_returns_std.sort_values(by = 'Std Deviation')
```

Draw a plot of Stock Means vs Standard Deviation

```
plt.scatter(x = df_returns_mean, y = df_returns_std)
plt.xlabel('Mean Stock Returns')
plt.ylabel('Stocks Std Deviation')
plt.title('Plot of Mean Stock Returns vs Std Deviation of Stock Returns');
```

```
df_returns_mean['std'] = df_returns_std['Std Deviation']
df_returns_mean.sort_values(by = 'Mean Return', ascending = False)
```

```
plt.scatter(df_returns_mean['Mean Return'], df_returns_mean['std'], label = df_returns_mean.index);
#plt.legend(loc = 'best')
```