BUSINESS REPORT: FINANCE AND RISK ANALYTICS, PROJECT

VIKRAM RADHAKRISHNAN

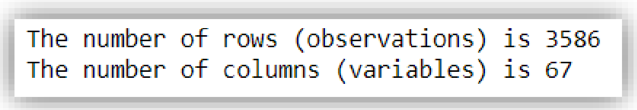
29th November 2020

PART 1 – CREDIT RISK

EDA

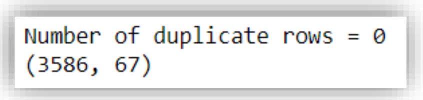
The data is read into a dataframe. The names of some of the features/columns are modified to make them standardized and easy to use by replacing spaces with underscore (_) e.t.c.

The shape of the dataframe is checked to understand the **number of rows and columns** in the data.



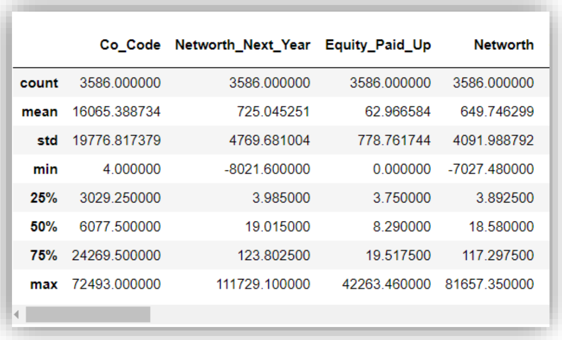
```
The number of rows (observations) is 3586
The number of columns (variables) is 67
```

The records are checked for **duplicity** and none are found.



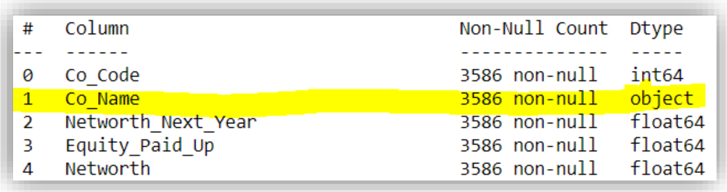
```
Number of duplicate rows = 0
(3586, 67)
```

Using the describe command, the **basic statistics** of features can be calculated and briefly studied.



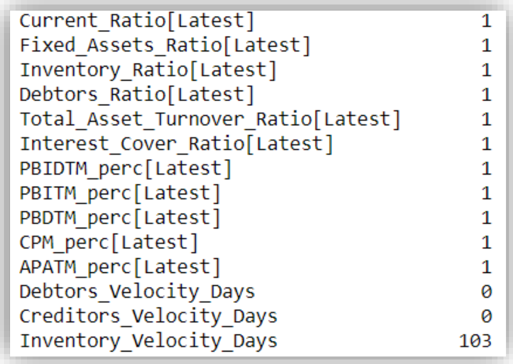
	Co_Code	Networth_Next_Year	Equity_Paid_Up	Networth
count	3586.000000	3586.000000	3586.000000	3586.000000
mean	16065.388734	725.045251	62.966584	649.746299
std	19776.817379	4769.681004	778.761744	4091.988792
min	4.000000	-8021.600000	0.000000	-7027.480000
25%	3029.250000	3.985000	3.750000	3.892500
50%	6077.500000	19.015000	8.290000	18.580000
75%	24269.500000	123.802500	19.517500	117.297500
max	72493.000000	111729.100000	42263.460000	81657.350000

By using the info() command on the dataframe, we are able to get the **data type** used for each of the features. Most of the features have a numerical datatype except 'Co_Name'. This feature is not necessary and can be dropped, since we already have a 'Co_Code' as an identifier.



```
#   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
```

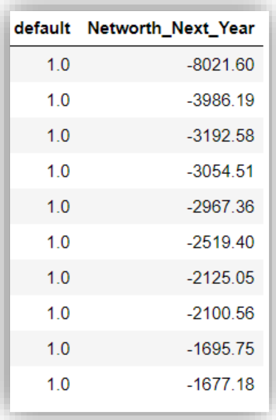
MISSING OR NULL VALUES are checked for. These null values are later replaced by the median value of the feature using the 'Simple Imputer'.



```
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
```

Transform target variable into 0 and 1 :

Using Networth_Next_year as the target variable as indicated, a new column called 'default' is created. Non-Defaulters are indicated by 0 and defaulters are indicated by 1, depending on whether the former feature is positive or negative.

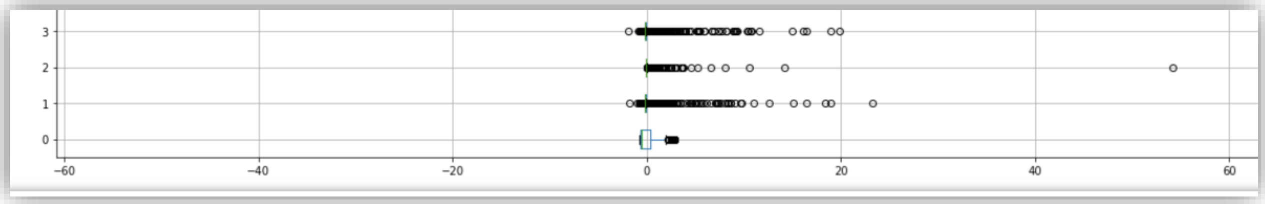


default	Networth_Next_Year
1.0	-8021.60
1.0	-3986.19
1.0	-3192.58
1.0	-3054.51
1.0	-2967.36
1.0	-2519.40
1.0	-2125.05
1.0	-2100.56
1.0	-1695.75
1.0	-1677.18

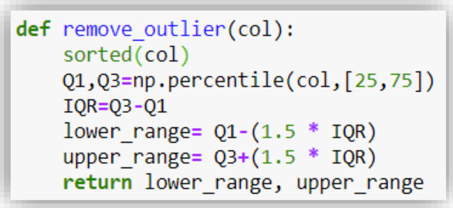
OUTLIER TREATMENT:

Outliers are checked for in the data by plotting BOXPLOTS for numerical features. Since the features are not of the same scale, the visualization of outliers is difficult. In order to make this simple, a STANDARD SCALER is used to scale all the numerical features and plot the corresponding boxplots for better understanding.

It is to be noted that scaling is not necessary for the models used in this project, but its done only for visualization purposes and not carried forward for modeling purposes.

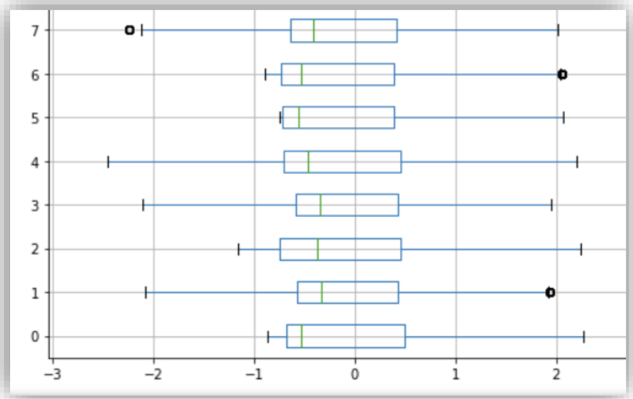


Functions are created to check and treat/remove outliers for numerical values. Upper and Lower ranges are calculated. Outliers on either side of the upper and lower ranges are adjusted and moved to either the upper or lower range value, depending on proximity.



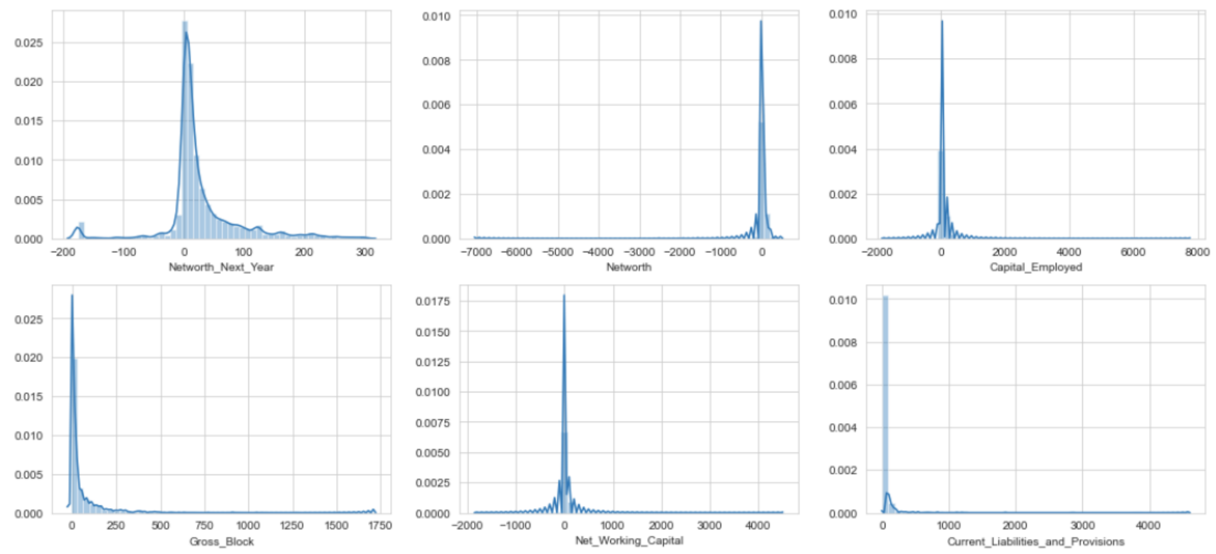
```
def remove_outlier(col):
    sorted(col)
    Q1,Q3=np.percentile(col,[25,75])
    IQR=Q3-Q1
    lower_range= Q1-(1.5 * IQR)
    upper_range= Q3+(1.5 * IQR)
    return lower_range, upper_range
```

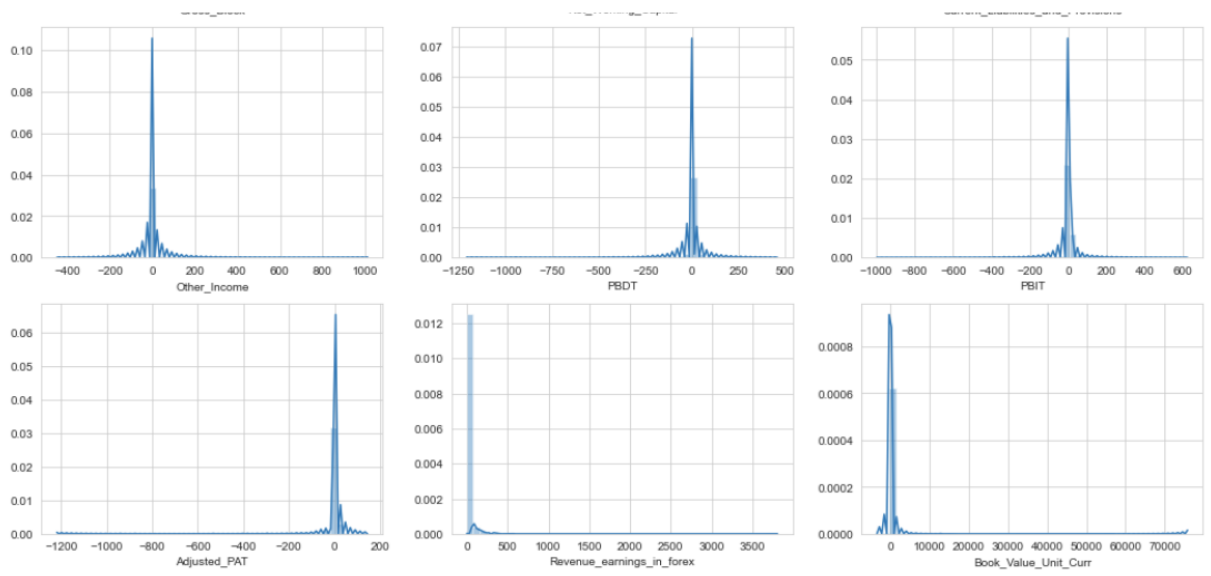
One may observe that values that were outliers are now clustered around the outer ends or extremes of the whiskers.



UNIVARIATE AND BIVARIATE ANALYIS:

UNIVARIATE ANALYSIS is also done for selected important features such as *Networth Value* and other variables through distribution plots. For illustration, we show a few variables in the following graph:





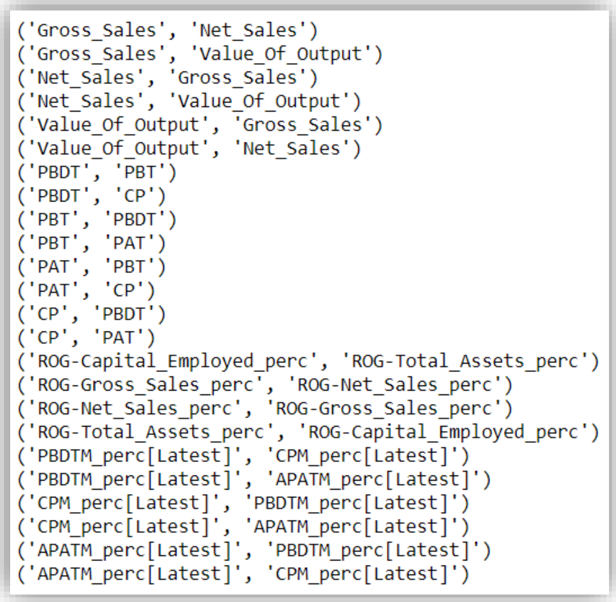
It is to be noted that these values are distributed so, after removing outliers. A normal distribution is not seen for most of the variables.

There are many ways of doing a bivariate analysis. A heatmap for all the variables is plotted but its difficult to represent in one picture in this report. An alternate way of extracting highly correlated values is discussed below as an alternative way to understand correlation without looking at the heat-map.

```
Out[1355]: <matplotlib.axes._subplots.AxesSubplot at 0x1dca5f665e0>
```

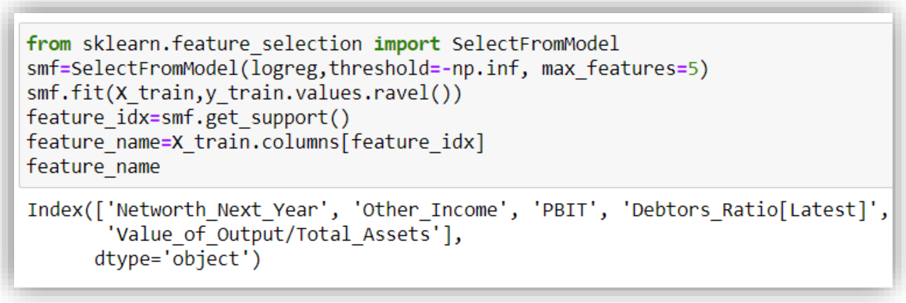
FEATURE RELATIONSHIPS THAT HAVE A CORRELATION OF MORE THAN 0.99

Through a few lines of code, one can extract highly correlated features (with correlation more than 0.99) . For example, Gross_Sales, Net_Sales and Value_Of_Output are highly correlated and only one feature needs to be used going forward (rest can be dropped as an option). Similarly, PBDT, PBT,PAT and CP are highly correlated and only of the variables need to be retained for modeling.



```
('Gross_Sales', 'Net_Sales')
('Gross_Sales', 'Value_Of_Output')
('Net_Sales', 'Gross_Sales')
('Net_Sales', 'Value_Of_Output')
('Value_Of_Output', 'Gross_Sales')
('Value_Of_Output', 'Net_Sales')
('PBDT', 'PBT')
('PBDT', 'CP')
('PBT', 'PBDT')
('PBT', 'PAT')
('PAT', 'PBT')
('PAT', 'CP')
('CP', 'PBDT')
('CP', 'PAT')
('ROG-Capital_Employed_perc', 'ROG-Total_Assets_perc')
('ROG-Gross_Sales_perc', 'ROG-Net_Sales_perc')
('ROG-Net_Sales_perc', 'ROG-Gross_Sales_perc')
('ROG-Total_Assets_perc', 'ROG-Capital_Employed_perc')
('PBDTM_perc[Latest]', 'CPM_perc[Latest]')
('PBDTM_perc[Latest]', 'APATM_perc[Latest]')
('CPM_perc[Latest]', 'PBDTM_perc[Latest]')
('CPM_perc[Latest]', 'APATM_perc[Latest]')
('APATM_perc[Latest]', 'PBDTM_perc[Latest]')
('APATM_perc[Latest]', 'CPM_perc[Latest]')
```

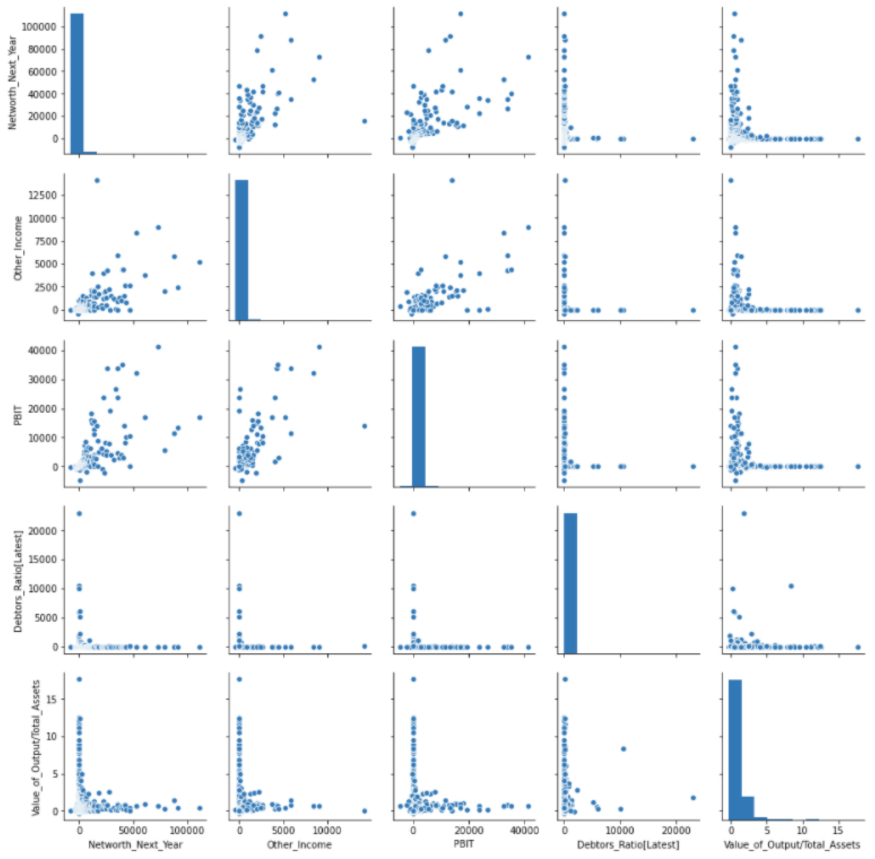
Please note that further to this, feature selection is also done at the modeling stage. Since feature numbers are very large, we can choose to study fewer important features. For example, to perform EDA, one can choose five important features using ‘FEATURE SELECTION’.



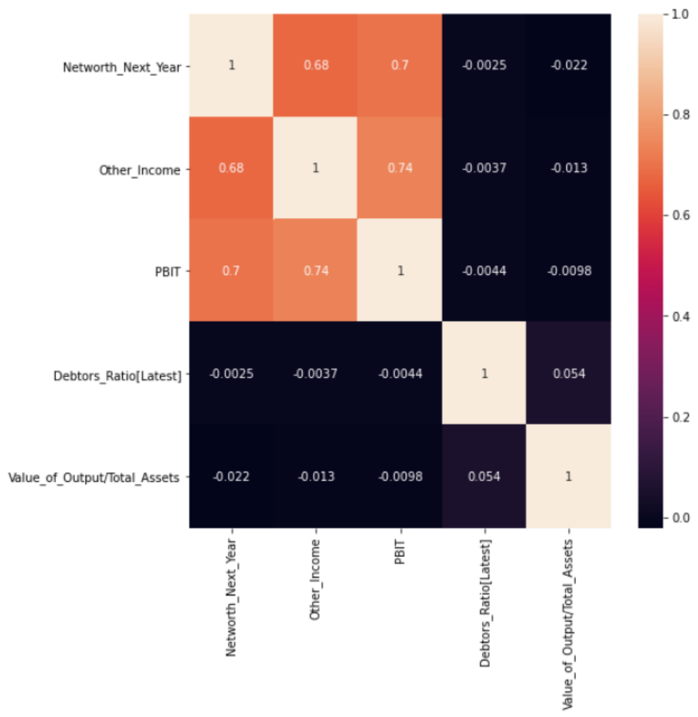
```
from sklearn.feature_selection import SelectFromModel
smf=SelectFromModel(logreg,threshold=-np.inf, max_features=5)
smf.fit(X_train,y_train.values.ravel())
feature_idx=smf.get_support()
feature_name=X_train.columns[feature_idx]
feature_name
```

```
Index(['Networth_Next_Year', 'Other_Income', 'PBIT', 'Debtors_Ratio[Latest]',
       'Value_of_Output/Total_Assets'],
      dtype='object')
```

PAIR-PLOTS can be plotted to observe the relationship between variables.



For the selected featues, one can also draw a heatmap to understand correlation. For example, Networth is positively correlated with PBIT and Other_Income but negatively correlated with Debtors_Ratio.

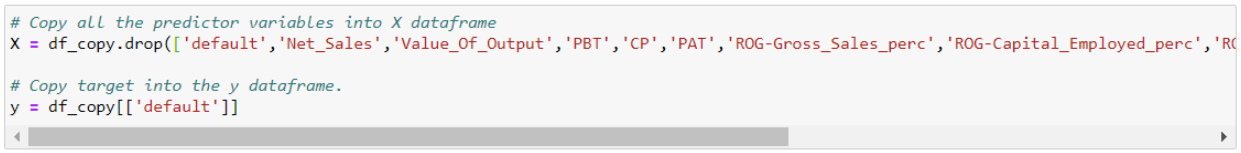


TEST-TRAIN SPLIT:

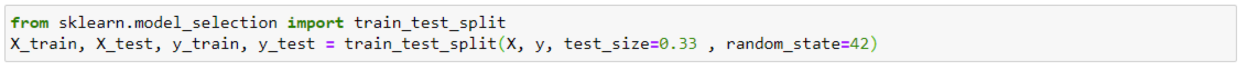
The dependent variables are stored in X and the independent variable are stored in y.

train and test variables are created with required features in each.

We can choose to keep either all features in the X data or selected features based on feature selection.



```
# Copy all the predictor variables into X dataframe
X = df_copy.drop(['default','Net_Sales','Value_Of_Output','PBT','CP','PAT','ROG-Gross_Sales_perc','ROG-Capital_Employed_perc','R

# Copy target into the y dataframe.
y = df_copy[['default']]
```



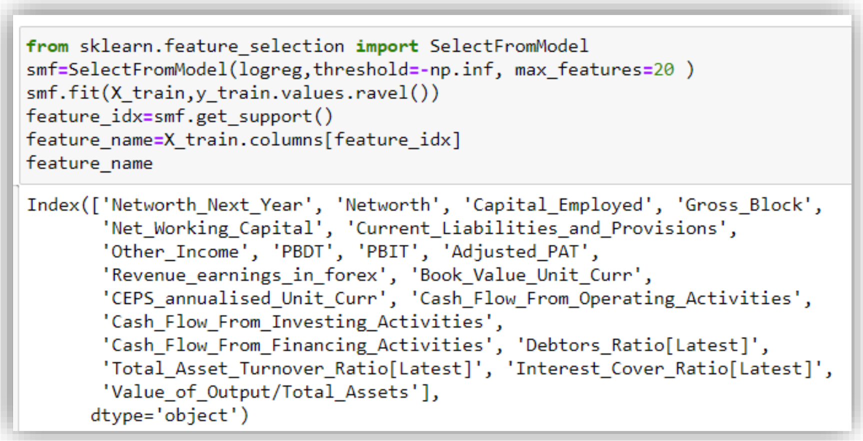
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33 , random_state=42)
```

MODELING :

Using the selected features performed earlier using feature selection, the Logistic regression model can be built using the test and train data. For the data that we use, we get a model with good precision, recall and F1-Score.

FEATURE SELECTION:

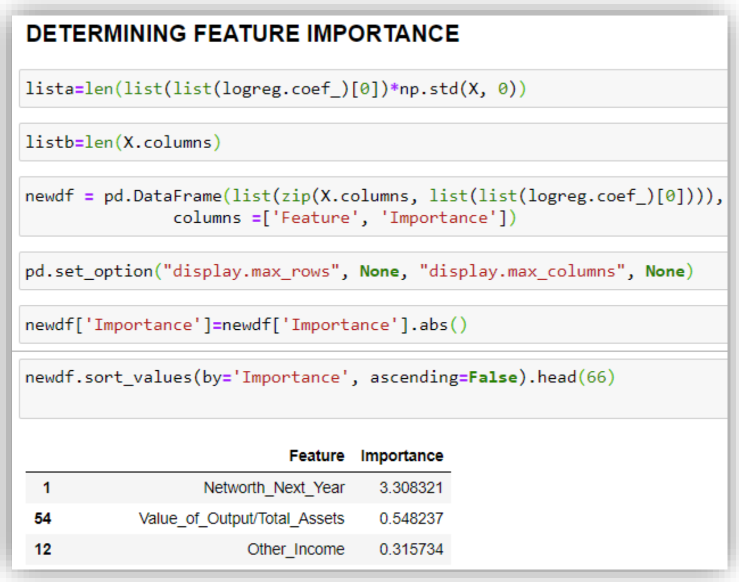
20 of the most important features for the model are selected using feature selection available from SKLEARN. A smaller number than 20 can also be chosen if one wishes, and this number can be decreased till accuracy remains agreeable.



```
from sklearn.feature_selection import SelectFromModel
smf=SelectFromModel(logreg,threshold=-np.inf, max_features=20 )
smf.fit(X_train,y_train.values.ravel())
feature_idx=smf.get_support()
feature_name=X_train.columns[feature_idx]
feature_name
```

```
Index(['Networth_Next_Year', 'Networth', 'Capital_Employed', 'Gross_Block',
       'Net_Working_Capital', 'Current_Liabilities_and_Provisions',
       'Other_Income', 'PBDT', 'PBIT', 'Adjusted_PAT',
       'Revenue_earnings_in_forex', 'Book_Value_Unit_Curr',
       'CEPS_annualised_Unit_Curr', 'Cash_Flow_From_Operating_Activities',
       'Cash_Flow_From_Investing_Activities',
       'Cash_Flow_From_Financing_Activities', 'Debtors_Ratio[Latest]',
       'Total_Asset_Turnover_Ratio[Latest]', 'Interest_Cover_Ratio[Latest]',
       'Value_of_Output/Total_Assets'],
      dtype='object')
```

Alternatively, one can also used logistic region model coefficients and sort them to rank them from largest to smallest. The few largest values can then be chosen as the important features.



DETERMINING FEATURE IMPORTANCE

```
lista=len(list(list(logreg.coef_)[0])*np.std(X, 0))
```

```
listb=len(X.columns)
```

```
newdf = pd.DataFrame(list(zip(X.columns, list(list(logreg.coef_)[0]))),
                     columns =['Feature', 'Importance'])
```

```
pd.set_option("display.max_rows", None, "display.max_columns", None)
```

```
newdf['Importance']=newdf['Importance'].abs()
```

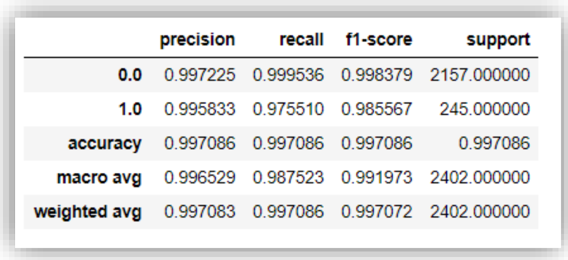
```
newdf.sort_values(by='Importance', ascending=False).head(66)
```

	Feature	Importance
1	Networth_Next_Year	3.308321
54	Value_of_Output/Total_Assets	0.548237
12	Other_Income	0.315734

LOGISTIC REGRESSION :

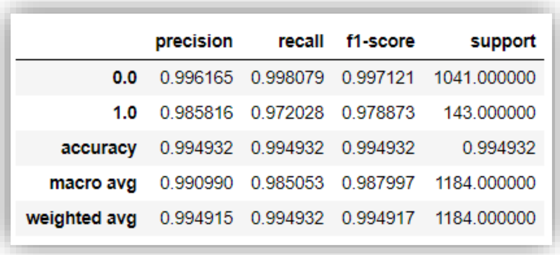
Logistic regression is run on the data and the train and test results are obtained as shown below

TRAIN



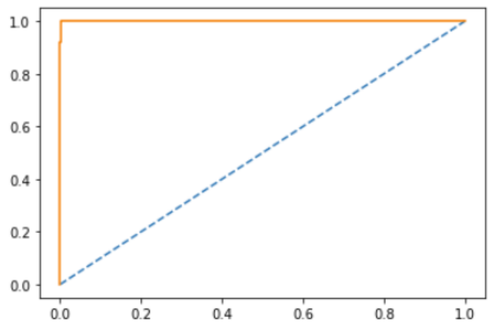
	precision	recall	f1-score	support
0.0	0.997225	0.999536	0.998379	2157.000000
1.0	0.995833	0.975510	0.985567	245.000000
accuracy	0.997086	0.997086	0.997086	0.997086
macro avg	0.996529	0.987523	0.991973	2402.000000
weighted avg	0.997083	0.997086	0.997072	2402.000000

TEST

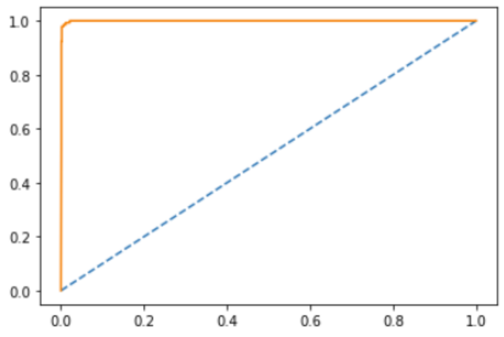


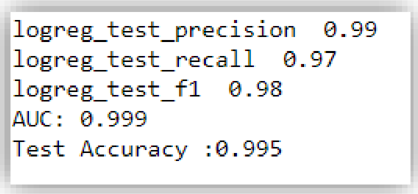
	precision	recall	f1-score	support
0.0	0.996165	0.998079	0.997121	1041.000000
1.0	0.985816	0.972028	0.978873	143.000000
accuracy	0.994932	0.994932	0.994932	0.994932
macro avg	0.990990	0.985053	0.987997	1184.000000
weighted avg	0.994915	0.994932	0.994917	1184.000000

TRAIN (ROC Curve)



TEST (ROC Curve)

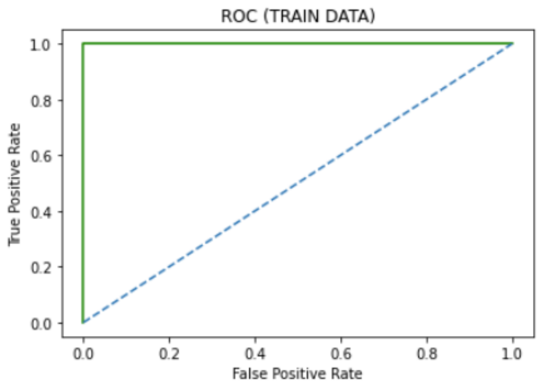


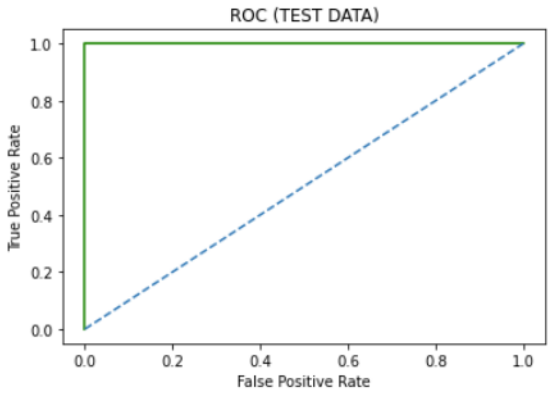


```
logreg_test_precision  0.99
logreg_test_recall  0.97
logreg_test_f1  0.98
AUC: 0.999
Test Accuracy :0.995
```

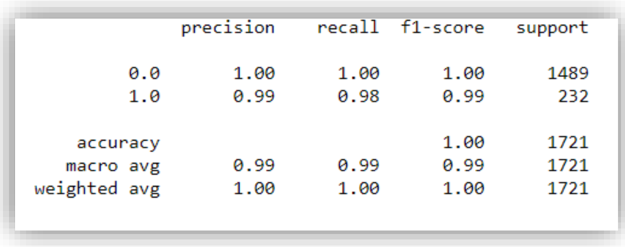
RANDOM FOREST :

The random forest algorithm is used on the X and y variables earlier defined. The GRID SEARCH method is used to arrive at the most optimum parameter for "hypertuning" the algorithm. A good precision, recall and f1 score is obtained.



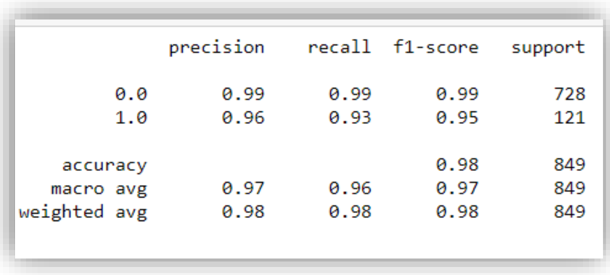


TRAIN DATA

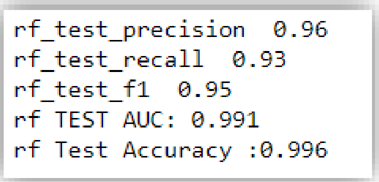


```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      1489
         1.0       0.99      0.98      0.99       232

    accuracy                           1.00      1721
   macro avg       0.99      0.99      0.99      1721
weighted avg       1.00      1.00      1.00      1721
```

TEST DATA

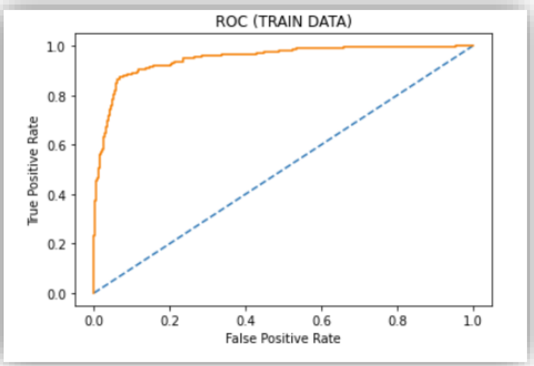


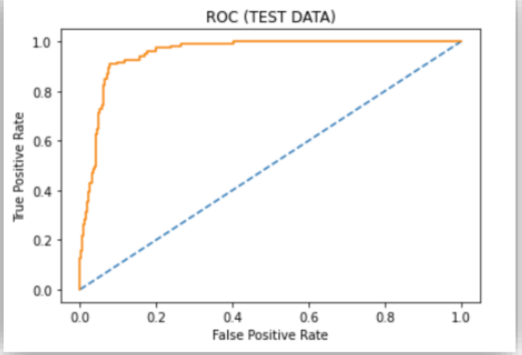
```
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99       728
         1.0       0.96      0.93      0.95       121

    accuracy                           0.98       849
   macro avg       0.97      0.96      0.97       849
weighted avg       0.98      0.98      0.98       849
```

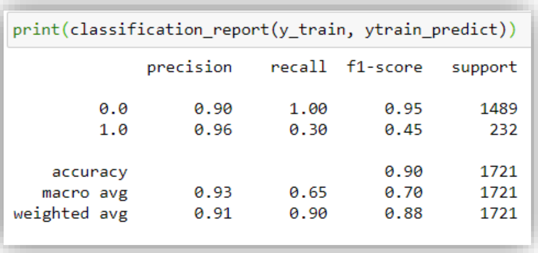

```
rf_test_precision  0.96
rf_test_recall  0.93
rf_test_f1  0.95
rf TEST AUC: 0.991
rf Test Accuracy :0.996
```

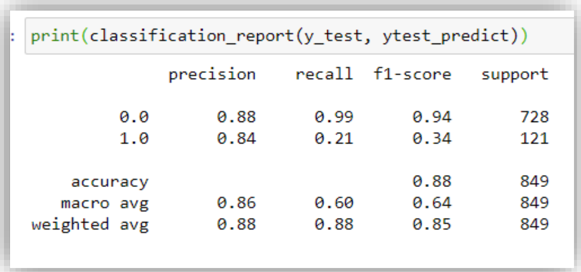
LINEAR DISCRIMINANT ANALYSIS:

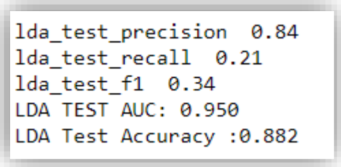
Linear discriminant analysis is used on the same X and y data. This is a linear model and decent results are obtained, though not as good as the earlier models – the LOGIT model and the Random Forest model which deal with nonlinearity.





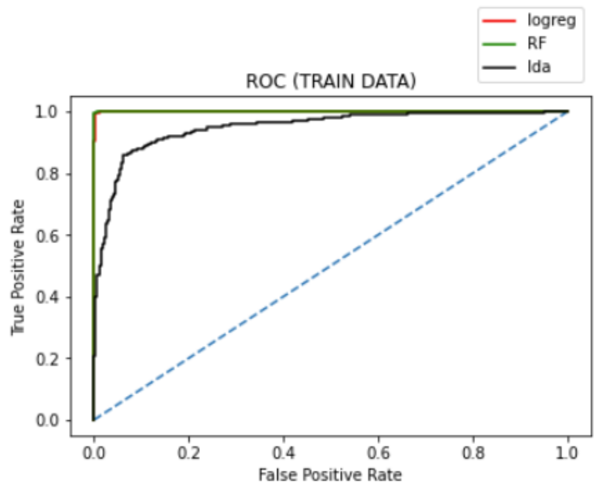

```
print(classification_report(y_train, ytrain_predict))

              precision    recall  f1-score   support

         0.0       0.90      1.00      0.95      1489
         1.0       0.96      0.30      0.45       232

    accuracy                           0.90      1721
   macro avg       0.93      0.65      0.70      1721
weighted avg       0.91      0.90      0.88      1721
```

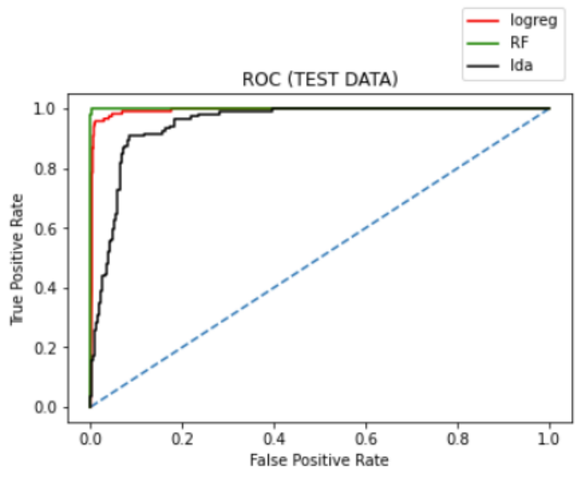

```
: print(classification_report(y_test, ytest_predict))

              precision    recall  f1-score   support

         0.0       0.88      0.99      0.94       728
         1.0       0.84      0.21      0.34       121

    accuracy                           0.88       849
   macro avg       0.86      0.60      0.64       849
weighted avg       0.88      0.88      0.85       849
```


```
lda_test_precision  0.84
lda_test_recall  0.21
lda_test_f1  0.34
LDA TEST AUC: 0.950
LDA Test Accuracy :0.882
```

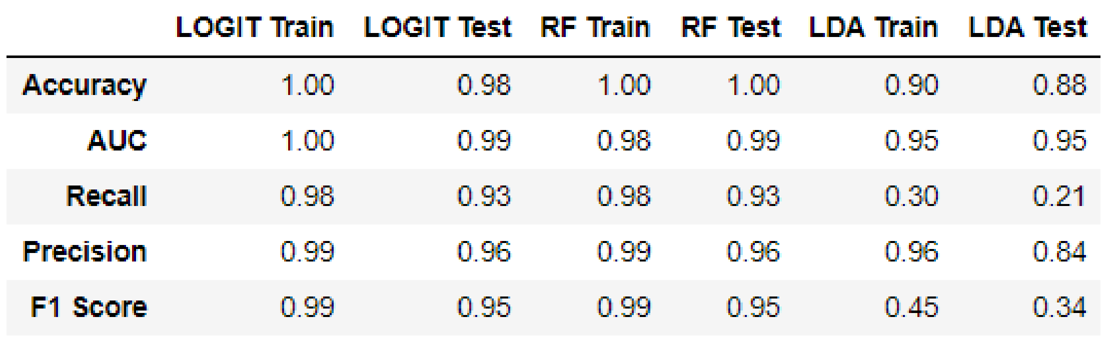
COMPARISON OF MODELS

ROC CURVES OF ALL THREE MODELS (SUPERIMPOSED)





MODEL PERFORMANCE COMPARISON :



	LOGIT Train	LOGIT Test	RF Train	RF Test	LDA Train	LDA Test
Accuracy	1.00	0.98	1.00	1.00	0.90	0.88
AUC	1.00	0.99	0.98	0.99	0.95	0.95
Recall	0.98	0.93	0.98	0.93	0.30	0.21
Precision	0.99	0.96	0.99	0.96	0.96	0.84
F1 Score	0.99	0.95	0.99	0.95	0.45	0.34

RECOMMENDATION

The LOGIT and the RF Train perform nearly the same with excellent train and test accuracy. In terms of other charateristics like AUC, Recall, Precision and F1 Score, the values are nearly the same. Therefore, both models can be used for this type of an analysis. However, the LDA model while showing an agreeable train and test score, shows poor performance on other counts such as recall and F1 score.

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class. If this value is poor, the prediction confidence is not good.

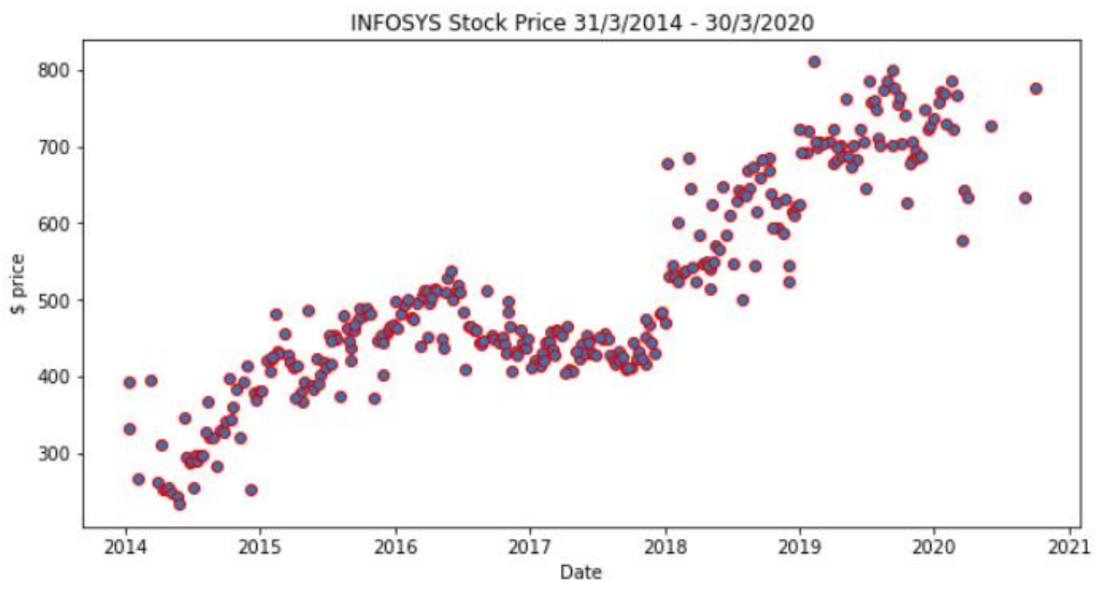
F1 score - F1 Score is the weighted average of Precision and Recall.

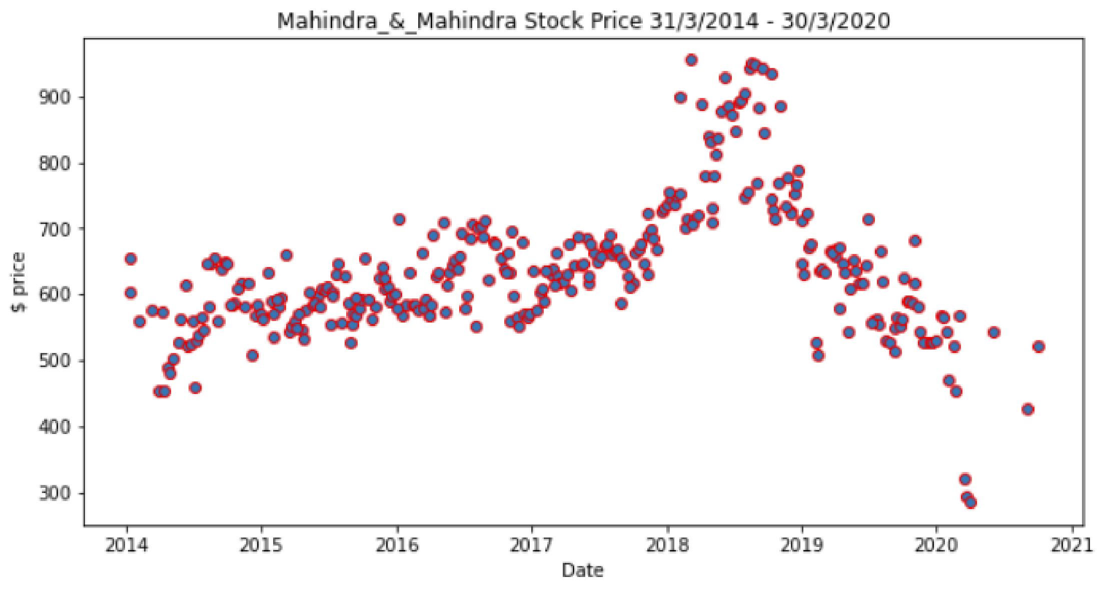
Since we get poor recall and F1 scores for LDA, we can skip this model for using on this data. Its possible that poor performance of LDA is due to the skewed nature of the data in the features. If the data provided was completely normally distributed, its possible that LDA may have performed better than this.

Part 2 -Market Risk

a) Draw Stock Price Chart for any 2 variables

Here, “Infosys” and “ Mahindra & Mahindra” are chosen, their stock charts are plotted below:

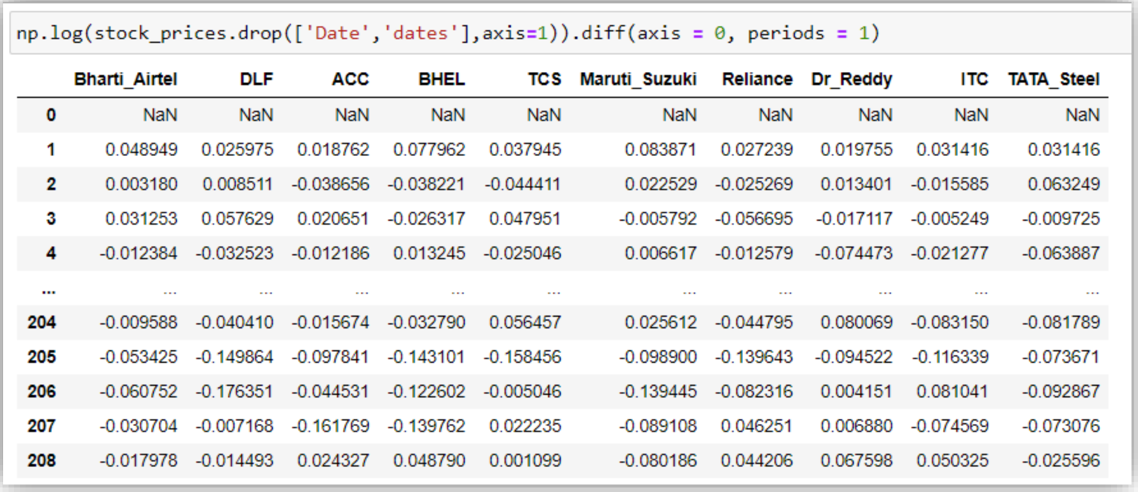




b) Calculate Returns

Returns are calculated from prices by :

a) Taking logarithms
b) Taking differences

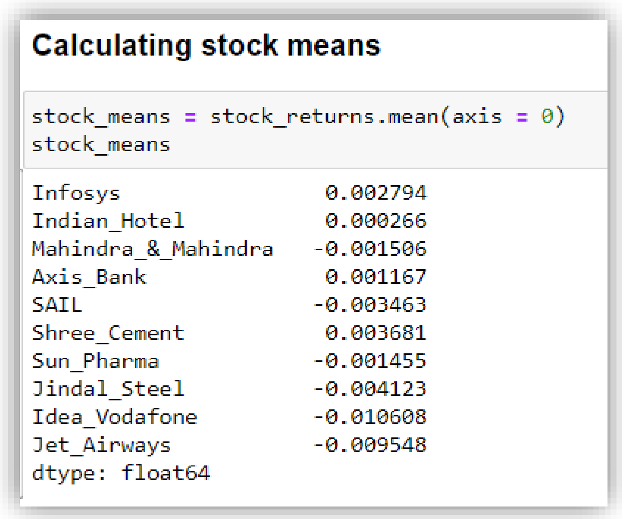


```
np.log(stock_prices.drop(['Date','dates'],axis=1)).diff(axis = 0, periods = 1)
```

	Bharti_Airtel	DLF	ACC	BHEL	TCS	Maruti_Suzuki	Reliance	Dr_Reddy	ITC	TATA_Steel
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	0.048949	0.025975	0.018762	0.077962	0.037945	0.083871	0.027239	0.019755	0.031416	0.031416
2	0.003180	0.008511	-0.038656	-0.038221	-0.044411	0.022529	-0.025269	0.013401	-0.015585	0.063249
3	0.031253	0.057629	0.020651	-0.026317	0.047951	-0.005792	-0.056695	-0.017117	-0.005249	-0.009725
4	-0.012384	-0.032523	-0.012186	0.013245	-0.025046	0.006617	-0.012579	-0.074473	-0.021277	-0.063887
...	...	...	...	...	...	...	...	...	...	...
204	-0.009588	-0.040410	-0.015674	-0.032790	0.056457	0.025612	-0.044795	0.080069	-0.083150	-0.081789
205	-0.053425	-0.149864	-0.097841	-0.143101	-0.158456	-0.098900	-0.139643	-0.094522	-0.116339	-0.073671
206	-0.060752	-0.176351	-0.044531	-0.122602	-0.005046	-0.139445	-0.082316	0.004151	0.081041	-0.092867
207	-0.030704	-0.007168	-0.161769	-0.139762	0.022235	-0.089108	0.046251	0.006880	-0.074569	-0.073076
208	-0.017978	-0.014493	0.024327	0.048790	0.001099	-0.080186	0.044206	0.067598	0.050325	-0.025596

c) Calculate Stock Means and Standard Deviation:

- **Stock Means**: Average returns that the stock is making on a week to week basis

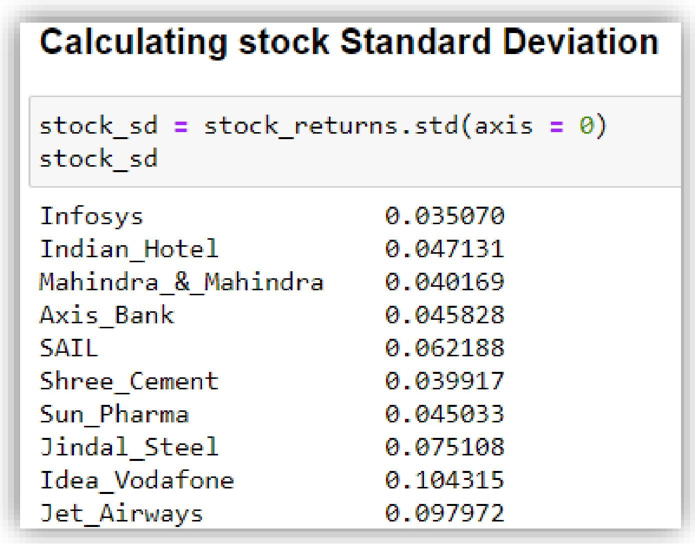


Calculating stock means

```
stock_means = stock_returns.mean(axis = 0)
stock_means
```

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

- **Stock Standard Deviation** : It is a measure of volatility meaning the more a stock's returns vary from the stock's average return, the more volatile the stock

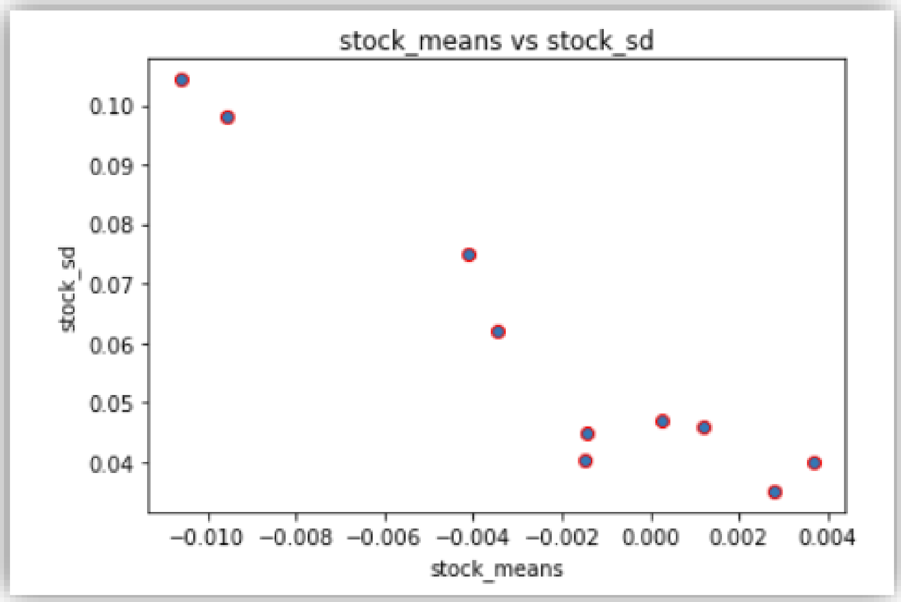


Calculating stock Standard Deviation

```
stock_sd = stock_returns.std(axis = 0)
stock_sd
```

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

d) Draw a plot of Stock Means vs Standard Deviation and share insights



A portfolio of competent stocks needs a good mean value of return with minimal (as possible) standard deviation.

When we look at the data and plot, we see that only four stocks have a positive mean. They are:

Infosys, Indian hotels, Axis Bank and Shree Cements.

These stocks co-incidentally also have a low standard deviation.

If we were to pick only two best stocks, we can pick **Shree Cements** and **Infosys** as they have positive returns with less standard deviation. Indian hotels and Axis bank come close the these in preference. All the other stocks have negative returns. It would be advisable to look at other better performing stocks to replace the ones with negative returns as they are prone to volatility besides having a low mean value of return.