Global Stock Market Analytics

Jerry Kiely

Introduction

In this project we will look at predicting the open direction of the Nifty 50 index by looking at other indices and indicators. We will break the project up into three parts:

1. preparing the master data from the global indices
2. preliminary analysis of data
3. predictive modelling of open direction of Nifty 50
4. comparing different models at prediction
5. sentiment analysis of X / Twitter data relating to Nifty 50

Phase 1 - Prepare the Master Data

The indexes of interest are:

- NSEI: Nifty 50
- DJI: Dow Jones Index
- IXIC: Nasdaq
- HSI: Hang Seng
- N225: Nikkei 225
- GDAXI: Dax
- VIX: Volatility Index

Phase 1 - Prepare the Master Data

Download and merge the required data, using LOCF to impute missing data, and adding variables for MONTH, QUARTER, and YEAR.

```
# merge data with outer join
merged = pd.concat(data, axis = 1)

# impute missing data using LOCF (forward fill)
merged.ffill(inplace = True)

# add indicators for MONTH, QUARTER, and YEAR
merged['MONTH']   = merged.index.month
merged['QUARTER'] = merged.index.quarter
merged['YEAR']    = merged.index.year
```

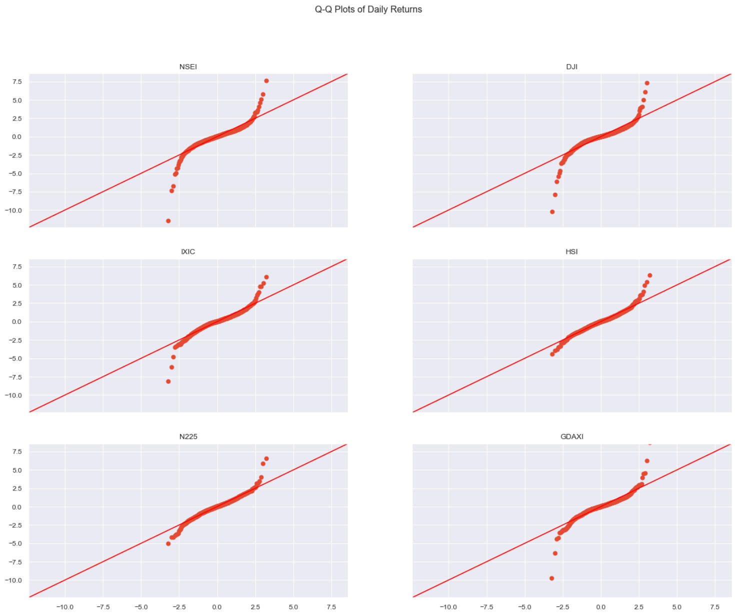
Phase 2 - Preliminary Analysis

We perform some preliminary analysis on the master data hoping to answer the following questions:

1. Which index has given consistently good returns?
2. Which index was highly volatile?
3. How are global markets correlated during 6 years period and is the correlation structure similar in the recent year-2023?
4. Assuming primary target variable as "Nifty Opening Price Direction", what are preliminary insights?

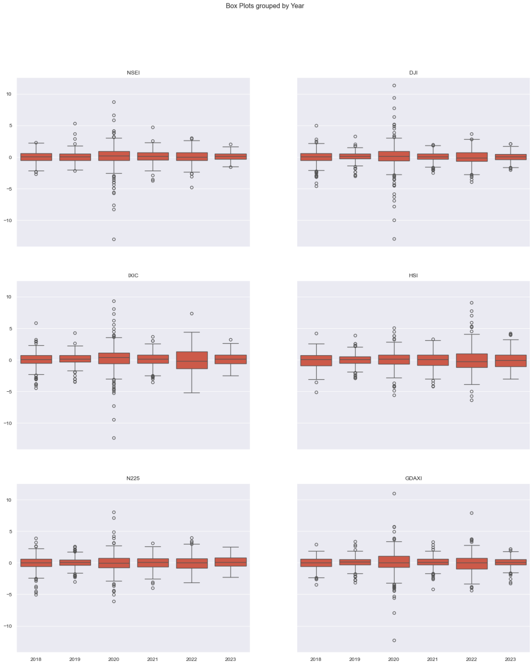
Phase 2 - Preliminary Analysis

Looking at the Q-Q Plots, the daily returns do not appear to follow - or be drawn from - a Normal Distribution - specifically at the tails. But that is OK because we do not intend to model the daily returns.



Phase 2 - Which index has given consistently good returns?

All indexes seem pretty consistent - all years have similar spreads, and consistent medians, with one or two exceptions. All indexes for 2020 have more outliers than normal. But HSI seems to have more outliers in 2022 than in 2020.



Phase 2 - Which index has given consistently good returns?

Looking at the summary statistics, the clear winner here is NSEI - not one year in the range has a negative mean return. And with the exception of 2020 and 2022, NSEI has low volatility (< 1) throughout all years.

```
NSEI

      count   mean    std    var
YEAR
2018    260  0.012  0.804  0.647
2019    260  0.062  0.862  0.744
2020    262  0.059  2.004  4.015
2021    261  0.094  0.980  0.960
2022    260  0.055  1.096  1.202
2023    260  0.079  0.620  0.384
```

Phase 2 - Which index was highly volatile?

As for the most volatile indexes, it's a toss between IXIC and HSI, both of whom have high volatility (> 1) when compared to the other indexes.

IXIC

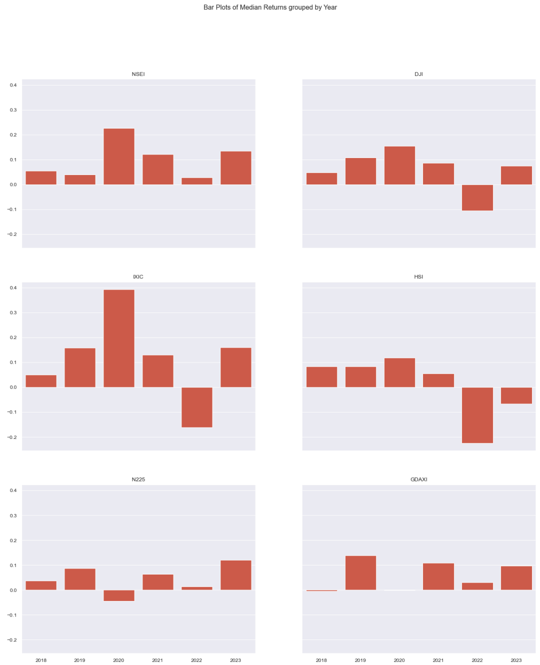
YEAR	count	mean	std	var
2018	260	-0.020	1.330	1.768
2019	260	0.133	0.975	0.950
2020	262	0.170	2.200	4.838
2021	261	0.096	1.124	1.262
2022	260	-0.124	2.000	4.001
2023	260	0.157	1.085	1.177

HSI

YEAR	count	mean	std	var
2018	260	-0.035	1.244	1.547
2019	260	0.033	0.981	0.962
2020	262	0.026	1.445	2.087
2021	261	-0.028	1.262	1.593
2022	260	-0.021	2.054	4.221
2023	260	-0.053	1.409	1.984

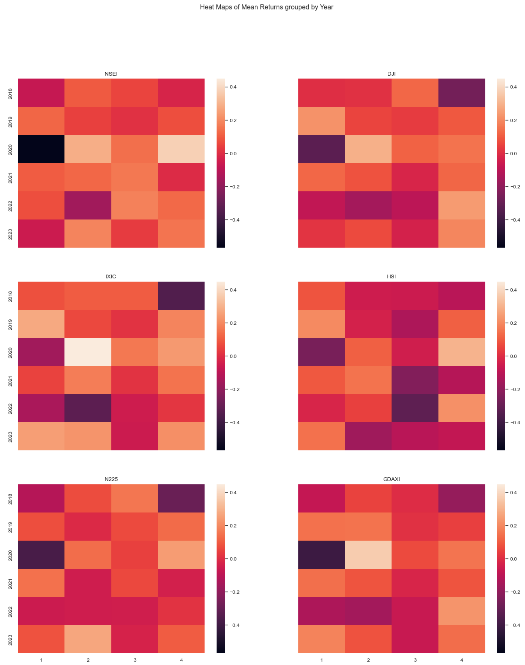
Phase 2 - Which index has given consistently good returns?

Looking at bar plots for median returns by year, again the clear winner here is NSEI - at no time is the median daily returns for any of the years below 0. IXIC has an unusually high 2020, but a bad 2022. HSI also has an unusually bad 2022.



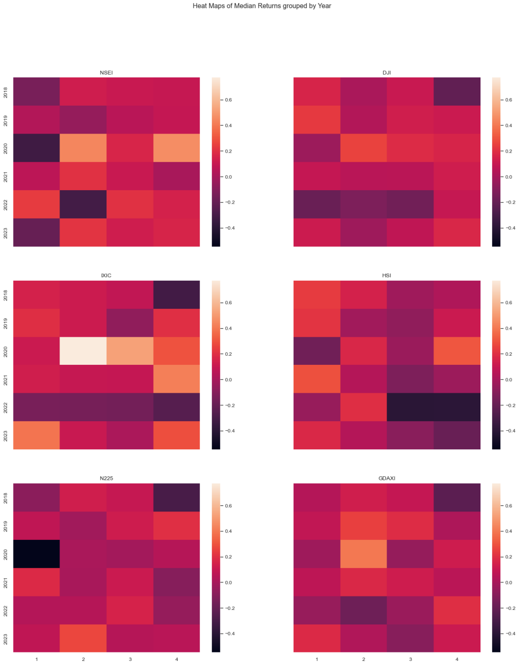
Phase 2 - Which index has given consistently good returns?

Looking at heat maps of mean returns, with the exception of the 1st quarter in 2020, NSEI has pretty consistent daily returns - where most cells sre pretty bright, denoting above 0. Most of the other indexes have a blend of light and dark, which would indicate more volatile behaviour over the quarters.



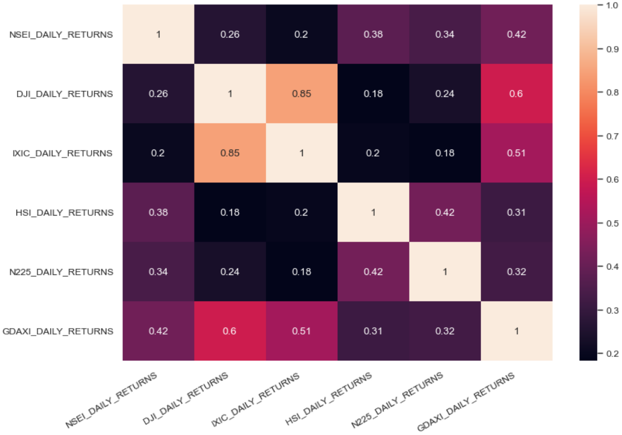
Phase 2 - Which index has given consistently good returns?

On the other hand, when looking at median returns across quarters NSEI seems pretty average - there does not seem to be a clear winner here.



Phase 2 - How are global markets correlated during the 6 years period?

It looks like strong correlation between daily returns of IXIC and DJI, and some correlation between GDAXI and DJI. These indexes are likely to result in multicolinearity at the regression stage.

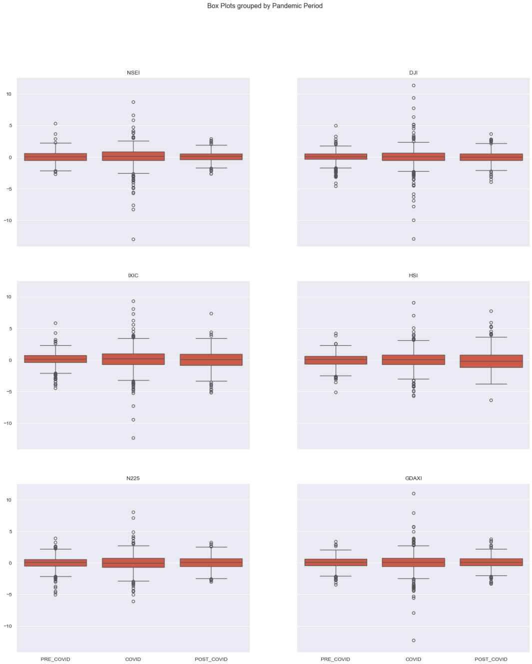


Phase 2 - Is the correlation structure similar in the recent year - 2023?

We can see similar - but slightly weaker - correlations exist between the same indexes for 2023.

Phase 2 - Covid Period

We can see that the spreads of each index over the Pandemic are consistent, with the Covid period itself having more outliers - which of course you might expect.



Phase 2 - Covid Period

All indexes had higher volatility over the Covid period.

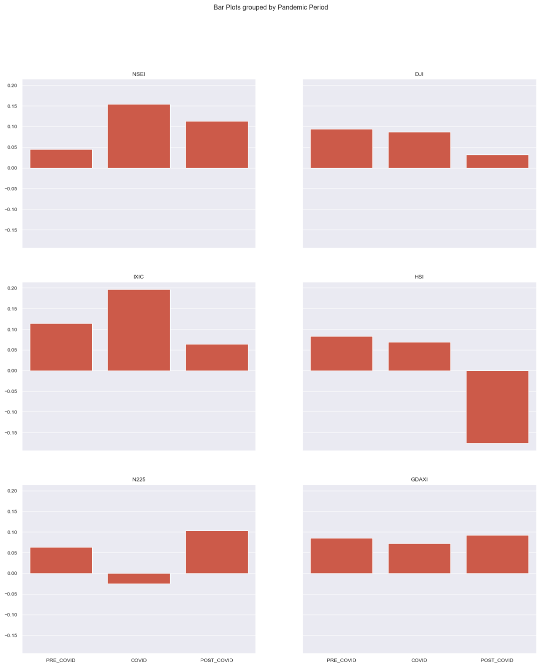
- NSEI performed reasonably well over the Covid period, with an increase in volatility in the Covid period, and with a significant bump in the Post Covid period.
- DJI seemed consisten over the three periods, with an increase in volatility in the Covid period.
- IXIC looked pretty good over the three period, but maybe slightly more volatile overall, and in particular in the Covid period.
- HSI has performed poorely in general, with negative returns in the pre and post Covid periods, and with consistently greater volatility than most.
- N225 appears to perform not so well, and with relatively high volatility.
- GDAXI also appears to perform not so well in general, and with relatively high volatility.

Phase 2 - Covid Period

Global Stock Market Analytics

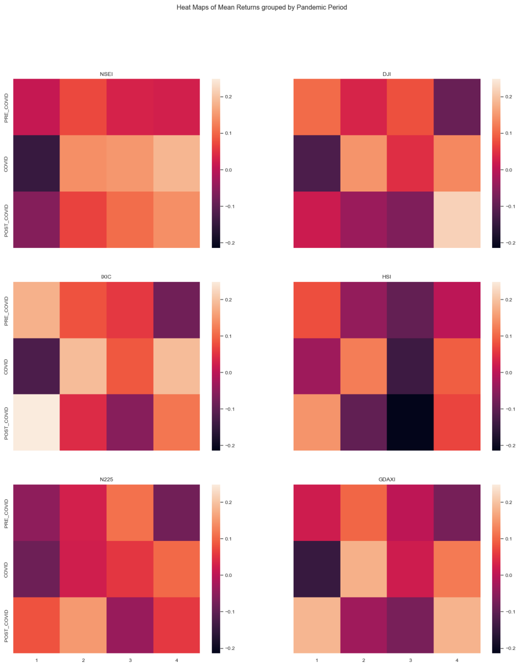
Jerry Kiely

With respect to returns, we can see that IXIC looks like the clear winner, with NSEI in second place, and DJI and GSAXI in a fight for third place. HSI appears to have had a terrible Post Covid period, and N225 appears to have had a pretty bad Covid period.



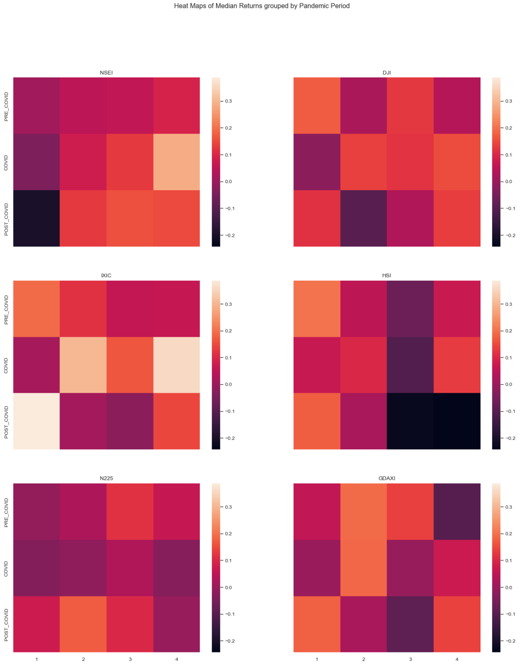
Phase 2 - Covid Period

Again, NSEI appears to be the most consistent of all indexes. All indexes have bad first quarters during the Pandemic, but improve post Covid.



Phase 2 - Covid Period

Median returns tells a similar story over the Pandemic period - HSI in particular appears to have had the worst recovery.



Phase 2 - Covid Period

We try to estimate the time taken for each of the indexes to return to the Pre Covid levels - the approach is to find how many days it takes for each index to reach a value greater than or equal to the Pre Covid mean returns value.

```
 NSEI returned to pre-covid levels on 2022-05-16 after 7 trading day(s)
  DJI returned to pre-covid levels on 2022-05-13 after 6 trading day(s)
 IXIC returned to pre-covid levels on 2022-05-10 after 3 trading day(s)
  HSI returned to pre-covid levels on 2022-05-11 after 4 trading day(s)
 N225 returned to pre-covid levels on 2022-05-06 after 1 trading day(s)
GDAXI returned to pre-covid levels on 2022-05-10 after 3 trading day(s)
```

Interestingly, N225 returned to it's Pre Covid level after just 1 day.

Phase 2 - Nifty Opening Price Direction: Preliminary Insights?

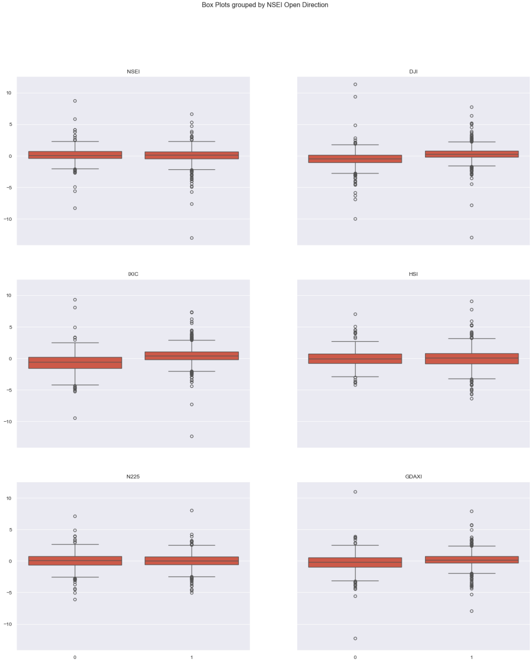
We define NSEI_OPEN_DIR as 1 if NSEI Open at time t $>$ NSEI Close at time t - 1, and 0 otherwise. Lets look at the percentages of NSEI_OPEN_DIR $= 1$ by year:

```
Nifty Fifty Daily Movement

YEAR
2018     70.38%
2019     69.23%
2020     70.61%
2021     71.65%
2022     59.23%
2023     67.31%
```

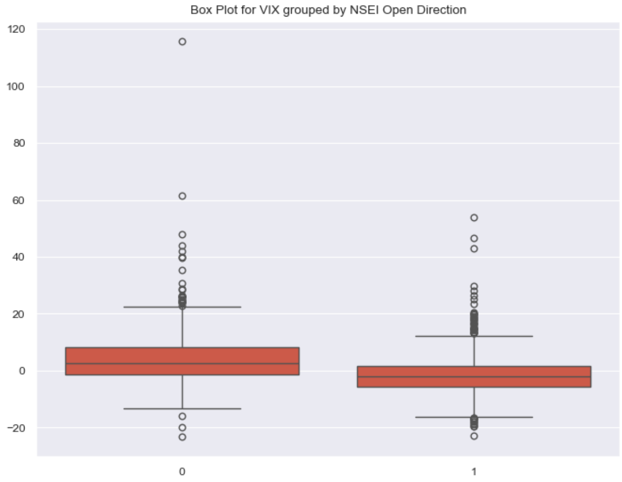
With the exception of 2022, every year has around 70% where NSEI_OPEN_DIR $= 1$.

Phase 2 - Nifty Opening Price Direction

All of the box plots look consistent across each category of NSEI_OPEN_DIR, with the exceptions of IXIC and VIX.



All of the box plots look consistent across each category of NSEI_OPEN_DIR, with the exceptions of IXIC and VIX.



Phase 3 - Logistic Model

Before proceeding with modelling NSEI_OPEN_DIR, lets define, and add, some indicators and ratios:

```
master["NSEI_HL_RATIO"] = master["NSEI_HIGH"] / master["NSEI_LOW"]
master["DJI_HL_RATIO"]  = master["DJI_HIGH"]  / master["DJI_LOW"]

master["NSEI_RSI"]      = ta.momentum.rsi(master["NSEI_CLOSE"])
master["DJI_RSI"]       = ta.momentum.rsi(master["DJI_CLOSE"])

master["NSEI_TSI"]      = ta.momentum.tsi(master["NSEI_CLOSE"])
master["DJI_TSI"]       = ta.momentum.tsi(master["DJI_CLOSE"])
```

Phase 3 - Logistic Model

We define a function that will prune any features that are either found to be insignificant, or that are found to be collinear:

```
dropping DJI_DAILY_RETURNS with p-value 0.7234766099770011
dropping GDAXI_DAILY_RETURNS with p-value 0.6162105670376612
dropping NSEI_HL_RATIO with p-value 0.4277618505298021
dropping DJI_HL_RATIO with p-value 0.1563055988923202
dropping NSEI_DAILY_RETURNS with p-value 0.13281329048460666
dropping NSEI_TSI with vif 5.865700460659149
dropping NSEI_RSI with p-value 0.7783762272653001
```

The function outputs a list of pruned features, together with the associated p-value or vif value. The function returns the pruned model, together with a list of pruned feature names.

Phase 3 - Logistic Model

The Logistic Model Summary

Dep. Variable:	NSEI_OPEN_DIR	**No. Observations:**	1220
Model:	Logit	**Df Residuals:**	1213
Method:	MLE	**Df Model:**	6
Date:	Wed, 10 Jul 2024	**Pseudo R-squ.:**	0.1375
Time:	10:31:02	**Log-Likelihood:**	-660.02
converged:	True	**LL-Null:**	-765.23
Covariance Type:	nonrobust	**LLR p-value:**	1.141e-42

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
Intercept	-1.4041	0.656	-2.139	0.032	-2.690	-0.118
IXIC_DAILY_RETURNS	0.4552	0.075	6.093	0.000	0.309	0.602
HSI_DAILY_RETURNS	-0.1395	0.053	-2.632	0.008	-0.243	-0.036
N225_DAILY_RETURNS	-0.1960	0.068	-2.897	0.004	-0.329	-0.063
VIX_DAILY_RETURNS	-0.0397	0.013	-3.054	0.002	-0.065	-0.014
DJI_RSI	0.0447	0.013	3.415	0.001	0.019	0.070
DJI_TSI	-0.0205	0.008	-2.660	0.008	-0.036	-0.005

Phase 3 - Logistic Model

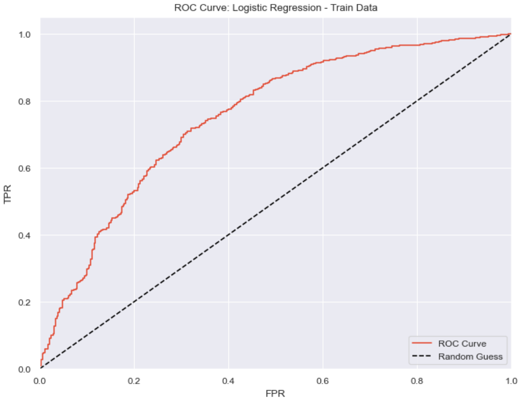
The final model is:

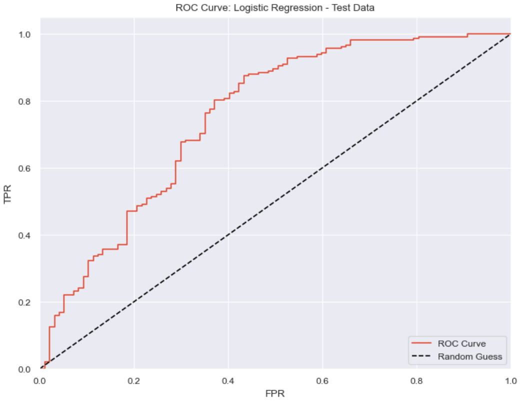
$$ln\left(\frac{p}{1-p}\right) = -1.4041 + 0.4552x_1 - 0.1395x_2 - 0.1960x_3 - 0.0397x_4 + 0.0447x_5 - 0.0205x_6$$

where:

variable	value
x_1	IXIC_DAILY_RETURNS
x_2	HSI_DAILY_RETURNS
x_3	N225_DAILY_RETURNS
x_4	VIX_DAILY_RETURNS
x_5	DJI_RSI
x_6	DJI_TSI

We plot the ROC curve for the train and test data:





Phase 3 - Logistic Model

The confusion matrix and classification report for the train data:

```
Train Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.53      0.68      0.60       391
         1.0       0.83      0.72      0.77       829

    accuracy                           0.70      1220
   macro avg       0.68      0.70      0.68      1220
weighted avg       0.73      0.70      0.71      1220

Train Data - Confusion Matrix:

NSEI_OPEN_DIR  0.0  1.0
row_0
0              265  234
1              126  595
```

Phase 3 - Logistic Model

The classification report and confusion matrix for the test data:

```
Test Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.53      0.65      0.58        97
         1.0       0.82      0.73      0.77       208

    accuracy                           0.70       305
   macro avg       0.67      0.69      0.67       305
weighted avg       0.72      0.70      0.71       305

Test Data - Confusion Matrix:

NSEI_OPEN_DIR  0.0  1.0
row_0
0               63   57
1               34  151
```

Phase 3 - Logistic Model

We compare some statistics across train and test data:

```
Train Data - Sensitivity: 71.77%
 Test Data - Sensitivity: 72.6%

Train Data - Specificity: 67.77%
 Test Data - Specificity: 64.95%

Train Data - AUC ROC: 0.7529115595469844
 Test Data - AUC ROC: 0.7520816812053925
```

Phase 3 - Logistic Model

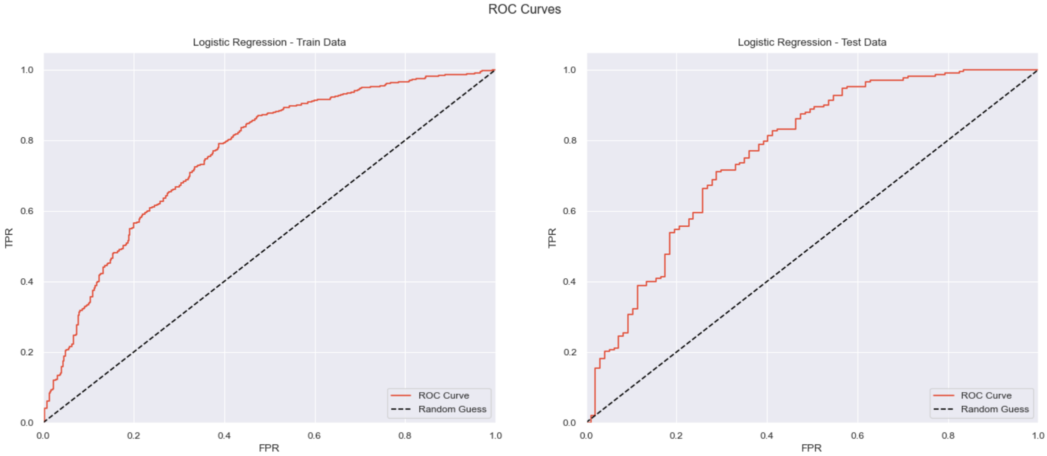
While AUC is consistent across train and test data, sensitivity and specificity values are very inconsistent. Moreover, the accuracy of the model is not great. We could potentially obtain better results by selecting a different classification model.

Phase 4 - Compare Models

We compare the performance of a number of different models to see if we can improve on the accuracy of our original model:

- Logistic Regression
- Naive Bayes
- KNN
- Decision Tree
- Random Forest
- SVM
- MLP
- Deep Learning (PyTorch)

Logistic Regression



Phase 4 - Compare Models

Logistic Regression

```
Train Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.58      0.61      0.59       391
         1.0       0.81      0.79      0.80       829

    accuracy                           0.73      1220
   macro avg       0.70      0.70      0.70      1220
weighted avg       0.74      0.73      0.73      1220


Train Data - Confusion Matrix:

col_0  0.0  1.0
row_0
0      239  174
1      152  655
```

Phase 4 - Compare Models

Logistic Regression

```
 Test Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.56      0.63      0.59        97
         1.0       0.82      0.77      0.79       208

    accuracy                           0.72       305
   macro avg       0.69      0.70      0.69       305
weighted avg       0.73      0.72      0.73       305


 Test Data - Confusion Matrix:

col_0  0.0  1.0
row_0
0       61   48
1       36  160
```

Phase 4 - Compare Models

Logistic Regression

```
Train Data - Sensitivity for cut-off 0.654: 79.01%
 Test Data - Sensitivity for cut-off 0.654: 76.92%

Train Data - Specificity for cut-off 0.654: 61.13%
 Test Data - Specificity for cut-off 0.654: 62.89%

Train Data - AUC ROC: 0.7584554774340638
 Test Data - AUC ROC: 0.767248215701824
```

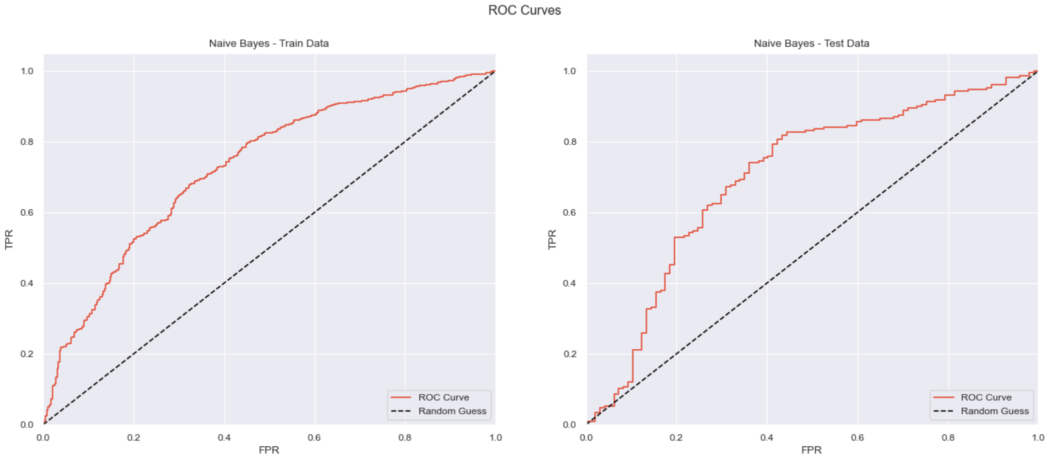
Phase 4 - Compare Models

Logistic Regression

We notice that the scikit-learn Logistic Regression model slightly outperforms the Statsmodels Logit model.

We use the Statsmodels Logit model when we need to perform analysis of features, moving to scikit-learn's LogisticRegression model after the model has been finalised is acceptable.

Naive Bayes


ROC Curves
Naive Bayes - Train Data
Naive Bayes - Test Data
TPR
FPR
1.0
0.8
0.6
0.4
0.2
0.0
ROC Curve
Random Guess

Naive Bayes

```
Train Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.50      0.68      0.57       391
         1.0       0.82      0.68      0.74       829

    accuracy                           0.68      1220
   macro avg       0.66      0.68      0.66      1220
weighted avg       0.71      0.68      0.69      1220


Train Data - Confusion Matrix:

col_0  0.0  1.0
row_0
0      264  266
1      127  563
```

Phase 4 - Compare Models

Naive Bayes

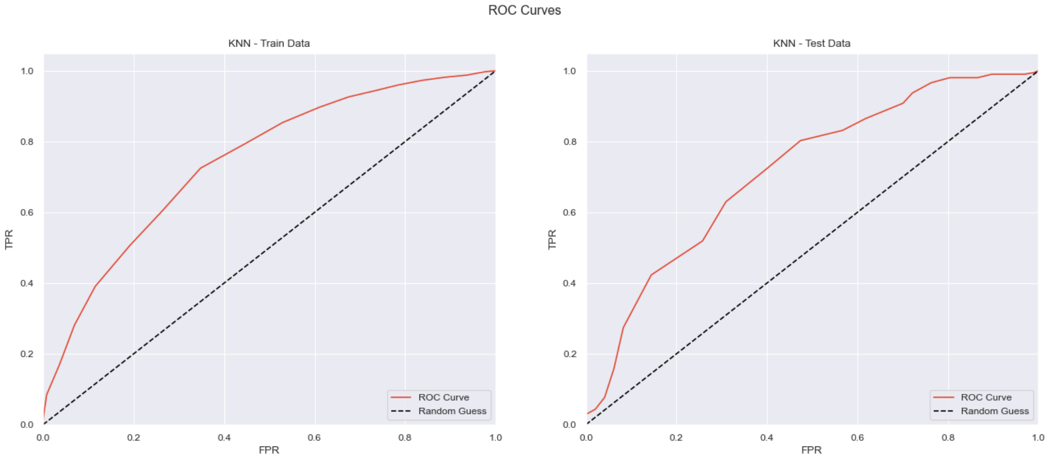
```
 Test Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.47      0.70      0.56        97
         1.0       0.82      0.62      0.71       208

    accuracy                           0.65       305
   macro avg       0.64      0.66      0.63       305
weighted avg       0.71      0.65      0.66       305


 Test Data - Confusion Matrix:

col_0  0.0  1.0
row_0
0       68   78
1       29  130
```

Phase 4 - Compare Models

Naive Bayes

```
Train Data - Sensitivity for cut-off 0.916: 67.91%
 Test Data - Sensitivity for cut-off 0.916: 62.5%

Train Data - Specificity for cut-off 0.916: 67.52%
 Test Data - Specificity for cut-off 0.916: 70.1%

Train Data - AUC ROC: 0.7282400451658085
 Test Data - AUC ROC: 0.7022204599524188
```

KNN


ROC Curves
KNN - Train Data
KNN - Test Data
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve
Random Guess

KNN

Train Data - Classification Report:

	precision	recall	f1-score	support
0.0	0.47	0.74	0.57	391
1.0	0.83	0.60	0.70	829
accuracy			0.65	1220
macro avg	0.65	0.67	0.64	1220
weighted avg	0.71	0.65	0.66	1220

Train Data - Confusion Matrix:

NSEI_OPEN_DIR	0.0	1.0
row_0		
0	288	328
1	103	501

Phase 4 - Compare Models

KNN

```
 Test Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.47      0.69      0.56        97
         1.0       0.81      0.63      0.71       208

    accuracy                           0.65       305
   macro avg       0.64      0.66      0.63       305
weighted avg       0.70      0.65      0.66       305


 Test Data - Confusion Matrix:

NSEI_OPEN_DIR  0.0  1.0
row_0
0               67   77
1               30  131
```

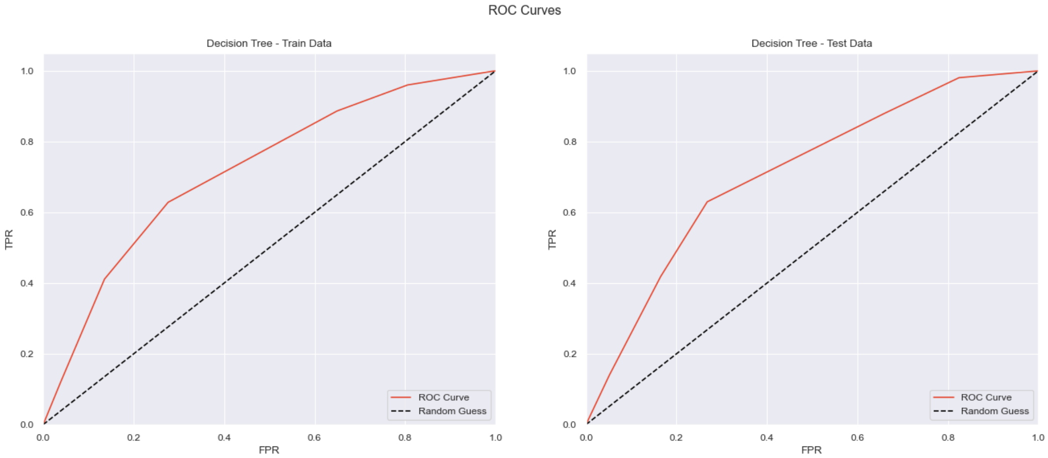
Phase 4 - Compare Models

KNN

```
Train Data - Sensitivity for cut-off 0.7: 60.43%
 Test Data - Sensitivity for cut-off 0.7: 62.98%

Train Data - Specificity for cut-off 0.7: 73.66%
 Test Data - Specificity for cut-off 0.7: 69.07%

Train Data - AUC ROC: 0.7450939257540746
 Test Data - AUC ROC: 0.7160735527359239
```

Phase 4 - Compare Models

Decision Tree



Decision Tree

Train Data - Classification Report:

	precision	recall	f1-score	support
0.0	0.41	0.86	0.56	391
1.0	0.87	0.41	0.56	829
accuracy			0.56	1220
macro avg	0.64	0.64	0.56	1220
weighted avg	0.72	0.56	0.56	1220

Train Data - Confusion Matrix:

NSEI_OPEN_DIR	0.0	1.0
row_0		
0	338	488
1	53	341

Phase 4 - Compare Models

Decision Tree

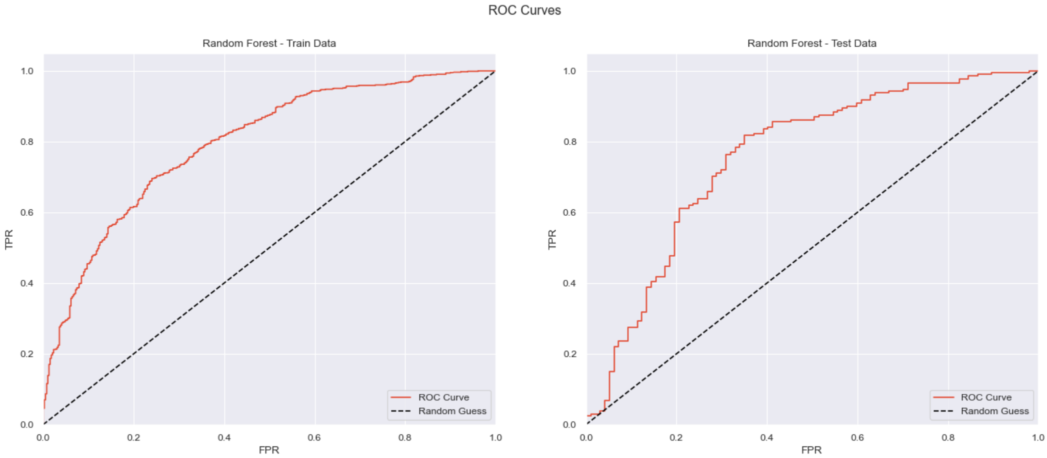
```
 Test Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.40      0.84      0.54        97
         1.0       0.84      0.42      0.56       208

    accuracy                           0.55       305
   macro avg       0.62      0.63      0.55       305
weighted avg       0.70      0.55      0.55       305


 Test Data - Confusion Matrix:

NSEI_OPEN_DIR  0.0  1.0
row_0
0               81  121
1               16   87
```

Phase 4 - Compare Models

Decision Tree

```
Train Data - Sensitivity for cut-off 0.766: 41.13%
 Test Data - Sensitivity for cut-off 0.766: 41.83%

Train Data - Specificity for cut-off 0.766: 86.45%
 Test Data - Specificity for cut-off 0.766: 83.51%

Train Data - AUC ROC: 0.7186808745630733
 Test Data - AUC ROC: 0.7119597541633624
```

Random Forest



Phase 4 - Compare Models

Random Forest

```
Train Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.54      0.76      0.63       391
         1.0       0.86      0.69      0.77       829

    accuracy                           0.71      1220
   macro avg       0.70      0.73      0.70      1220
weighted avg       0.76      0.71      0.72      1220


Train Data - Confusion Matrix:

NSEI_OPEN_DIR  0.0  1.0
row_0
0              297  254
1               94  575
```

Phase 4 - Compare Models

Random Forest

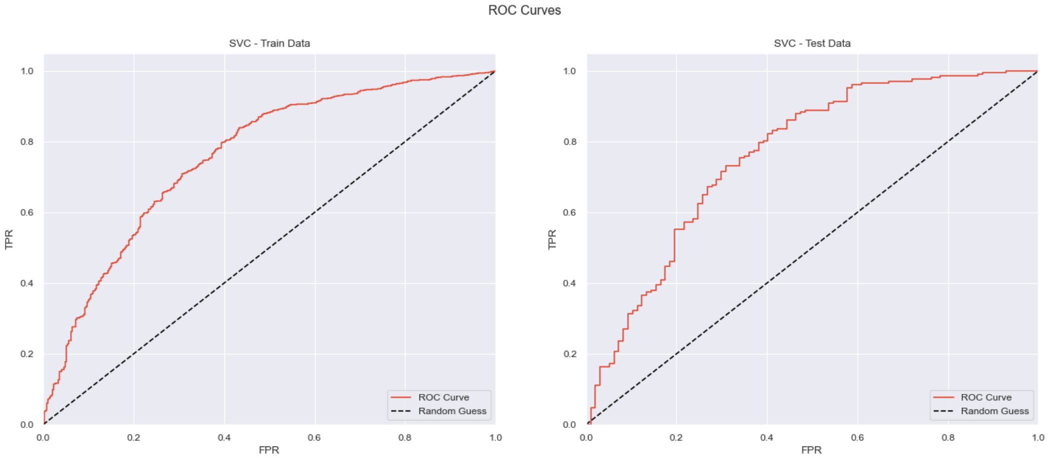
```
 Test Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.51      0.72      0.60        97
         1.0       0.84      0.68      0.75       208

    accuracy                           0.69       305
   macro avg       0.68      0.70      0.67       305
weighted avg       0.73      0.69      0.70       305


 Test Data - Confusion Matrix:

NSEI_OPEN_DIR  0.0  1.0
row_0
0               70   67
1               27  141
```

Phase 4 - Compare Models

Random Forest

```
Train Data - Sensitivity for cut-off 0.71: 69.36%
 Test Data - Sensitivity for cut-off 0.71: 67.79%

Train Data - Specificity for cut-off 0.71: 75.96%
 Test Data - Specificity for cut-off 0.71: 72.16%

Train Data - AUC ROC: 0.7957851415596395
 Test Data - AUC ROC: 0.7562450436161777
```

SVM


ROC Curves
SVC - Train Data
SVC - Test Data
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve
Random Guess

Phase 4 - Compare Models

SVM

Train Data - Classification Report:

	precision	recall	f1-score	support
0.0	0.62	0.57	0.59	391
1.0	0.80	0.84	0.82	829
accuracy			0.75	1220
macro avg	0.71	0.70	0.71	1220
weighted avg	0.75	0.75	0.75	1220

Train Data - Confusion Matrix:

NSEI_OPEN_DIR	0.0	1.0
row_0		
0	222	136
1	169	693

Phase 4 - Compare Models

SVM

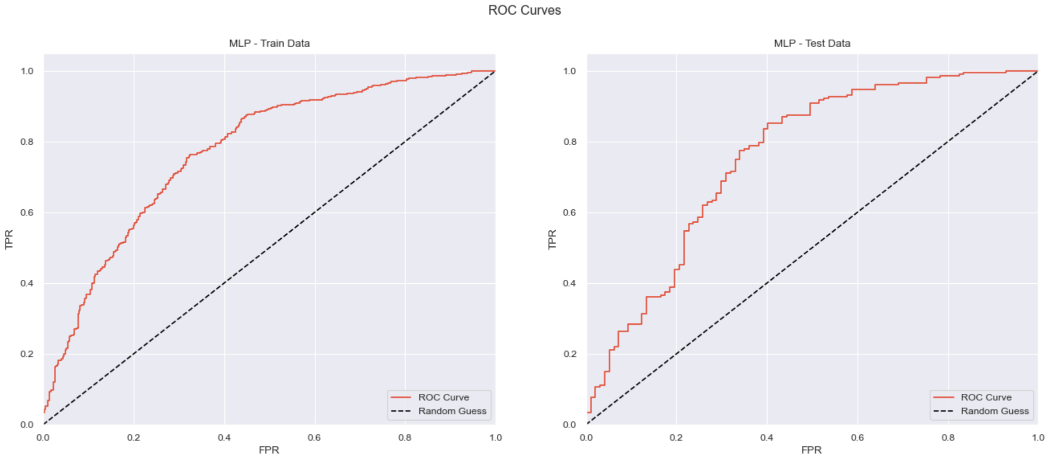
```
 Test Data - Classification Report:

             precision    recall  f1-score   support

        0.0       0.61      0.59      0.60        97
        1.0       0.81      0.82      0.82       208

   accuracy                           0.75       305
  macro avg       0.71      0.70      0.71       305
weighted avg       0.75      0.75      0.75       305


 Test Data - Confusion Matrix:

NSEI_OPEN_DIR  0.0  1.0
row_0
0               57   37
1               40  171
```

Phase 4 - Compare Models

SVM

```
Train Data - Sensitivity for cut-off 0.643: 83.59%
 Test Data - Sensitivity for cut-off 0.643: 82.21%

Train Data - Specificity for cut-off 0.643: 56.78%
 Test Data - Specificity for cut-off 0.643: 58.76%

Train Data - AUC ROC: 0.7597172817834325
 Test Data - AUC ROC: 0.763332672482157
```

MLP


ROC Curves
MLP - Train Data
MLP - Test Data
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve
Random Guess

MLP

Train Data - Classification Report:

	precision	recall	f1-score	support
0.0	0.57	0.68	0.62	391
1.0	0.83	0.76	0.80	829
accuracy			0.73	1220
macro avg	0.70	0.72	0.71	1220
weighted avg	0.75	0.73	0.74	1220

Train Data - Confusion Matrix:

NSEI_OPEN_DIR	0.0	1.0
row_0		
0	264	198
1	127	631

Phase 4 - Compare Models

MLP

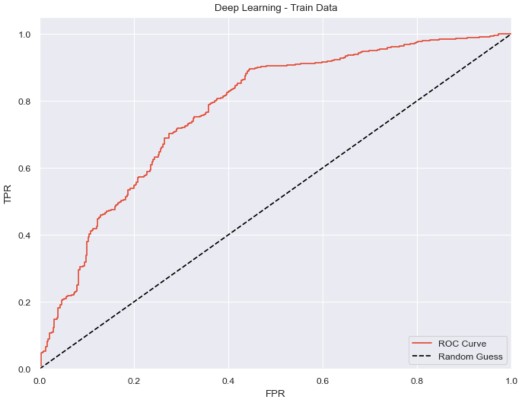
```
 Test Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.55      0.67      0.60        97
         1.0       0.83      0.75      0.78       208

    accuracy                           0.72       305
   macro avg       0.69      0.71      0.69       305
weighted avg       0.74      0.72      0.73       305


 Test Data - Confusion Matrix:

NSEI_OPEN_DIR  0.0  1.0
row_0
0               65   53
1               32  155
```

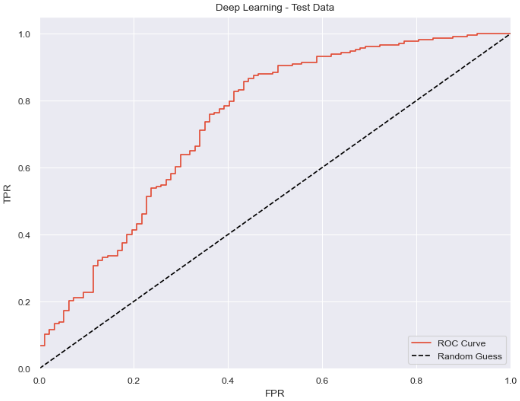
Phase 4 - Compare Models

MLP

```
Train Data - Sensitivity for cut-off 0.71: 76.12%
 Test Data - Sensitivity for cut-off 0.71: 74.52%

Train Data - Specificity for cut-off 0.71: 67.52%
 Test Data - Specificity for cut-off 0.71: 67.01%

Train Data - AUC ROC: 0.773263939235945
 Test Data - AUC ROC: 0.7568893735130848
```

Deep Learning (PyTorch)





Phase 4 - Compare Models

Deep Learning (PyTorch)

```
Train Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.71      0.56      0.62       391
         1.0       0.81      0.89      0.85       829

    accuracy                           0.79      1220
   macro avg       0.76      0.73      0.74      1220
weighted avg       0.78      0.79      0.78      1220


Train Data - Confusion Matrix:

col_0  0.0  1.0
row_0
0      218   89
1      173  740
```

Phase 4 - Compare Models

Deep Learning (PyTorch)

```
 Test Data - Classification Report:

              precision    recall  f1-score   support

         0.0       0.66      0.55      0.60        97
         1.0       0.80      0.87      0.84       208

    accuracy                           0.77       305
   macro avg       0.73      0.71      0.72       305
weighted avg       0.76      0.77      0.76       305


 Test Data - Confusion Matrix:

col_0  0.0  1.0
row_0
0       53   27
1       44  181
```

Phase 4 - Compare Models

Deep Learning (PyTorch)

```
Train Data - Deep Learning - Sensitivity for cut-off 0.36000001430511475 is : 87.02%
 Test Data - Deep Learning - Sensitivity for cut-off 0.36000001430511475 is : 89.26%

Train Data - Deep Learning - Specificity for cut-off 0.36000001430511475 is : 54.64%
 Test Data - Deep Learning - Specificity for cut-off 0.36000001430511475 is : 55.75%

Train Data - Deep Learning - AUC ROC: 0.7720391560410812
 Test Data - Deep Learning - AUC ROC: 0.7385507533703409
```

Phase 4 - Compare Models

```
         Naive Bayes - Test Data - AUC ROC: 0.7022204599524188
       Decision Tree - Test Data - AUC ROC: 0.7119597541633624
                 KNN - Test Data - AUC ROC: 0.7160735527359239
       Deep Learning - Test Data - AUC ROC: 0.7385507533703409
                 MLP - Test Data - AUC ROC: 0.7568893735130848
       Random Forest - Test Data - AUC ROC: 0.7562450436161777
                 SVM - Test Data - AUC ROC: 0.763332672482157
 Logistic Regression - Test Data - AUC ROC: 0.767248215701824
```

Phase 5 - Sentiment Analysis

We now turn to Twitter / X data relating to the Nifty 50 index to see if we can mine some sentiment.

We load the tweets, create a data frame, and then do some basic pre-processing of the data to:

1. transform all words to lowercase
2. remove all punctuation
3. remove all digits
4. remove stopwords

We load the tweets:

```
with open(os.path.join(os.getcwd(), "Tweets.txt")) as file:
    tweets = [line.rstrip() for line in file]

data = pd.DataFrame(
     [line for line in tweets if len(line) > 0],
     columns= ["Tweets"]
)
data.head()
```

```
                                              Tweets
0  #bankNifty 50100 ce looks good at 70+-2 for a ...
1  "#market #banknifty #OptionsTrading #optionbuy...
2  PENNY STOCK MADHUCON PROJECTS LTD cmp-11 FOLLO...
3  #Nifty50 has been in a healthy uptrend since t...
4  #Gravita #livetrading #stockstowatch #stocksin...
```

Phase 5 - Sentiment Analysis

We perform the basic transformation:

```
stop_words    = set(stopwords.words('english'))
remove_punc   = str.maketrans('', '', punctuation)
remove_digits = str.maketrans('', '', digits)

def preprocess_tweet(tweet):
    tokens = word_tokenize(
     tweet.lower().translate(remove_punc).translate(remove_digits)
    )
    return " ".join([word for word in tokens if word not in stop_words])

cleaned = data["Tweets"].apply(preprocess_tweet)
cleaned.head()
```

```
0           banknifty ce looks good target nifty nifty
1    market banknifty optionstrading optionbuying t...
2    penny stock madhucon projects ltd cmp followht...
3    nifty healthy uptrend since beginning year did...
4    gravita livetrading stockstowatch stocksinfocu...
```

Phase 5 - Sentiment Analysis

We look at the top 10 words by frequency:

```
                 Word  Freq
0               nifty   399
1           banknifty   104
2         stockmarket    71
3           niftybank    45
4    stockmarketindia    44
5              sensex    43
6              stocks    38
7      optionstrading    36
8                 bse    34
9      breakoutstocks    31
...
```

Phase 5 - Sentiment Analysis

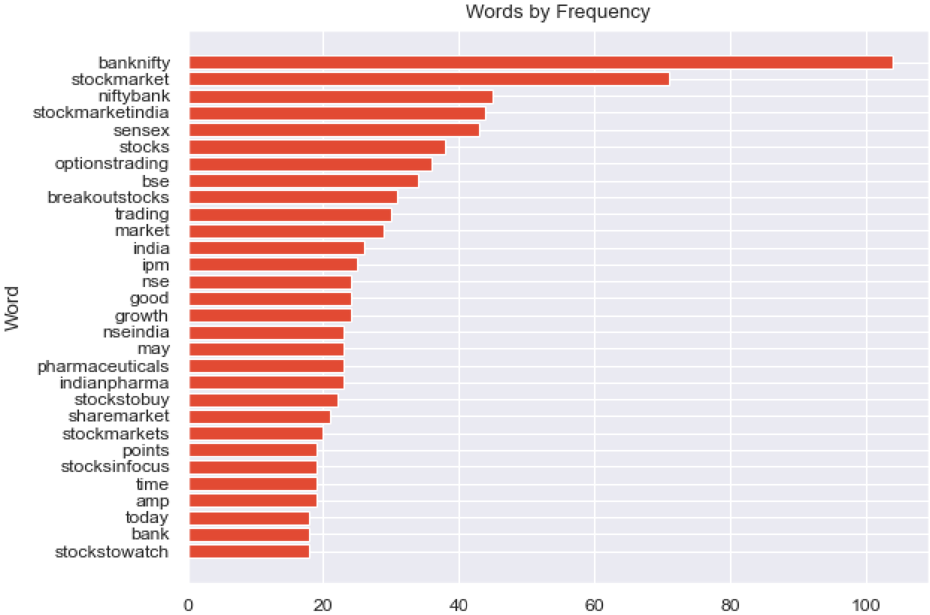
We remove the word “nifty” from the tweets:

```
stop_words = set(stopwords.words('english')) | set(["nifty"])

data["Cleaned_Tweets"] = data["Tweets"].apply(preprocess_tweet)
data.head()
```

```
                                              Tweets  \
0  #bankNifty 50100 ce looks good at 70+-2 for a ...
1  "#market #banknifty #OptionsTrading #optionbuy...
2  PENNY STOCK MADHUCON PROJECTS LTD cmp-11 FOLLO...
3  #Nifty50 has been in a healthy uptrend since t...
4  #Gravita #livetrading #stockstowatch #stocksin...

                                      Cleaned_Tweets
0                        banknifty ce looks good target
1  market banknifty optionstrading optionbuying t...
2  penny stock madhucon projects ltd cmp followht...
3  healthy uptrend since beginning year didnt bre...
4  gravita livetrading stockstowatch stocksinfocu...
```

Phase 5 - Sentiment Analysis

We visualise these top 20 words by frequency:



Phase 5 - Sentiment Analysis

We create a word cloud:

Phase 5 - Sentiment Analysis

We extract sentiment scores for the tweets:

```
sia = SentimentIntensityAnalyzer()

scores = data["Cleaned_Tweets"].apply(lambda x: sia.polarity_scores(x))

data["Positive_Score"] = scores.apply(lambda x: x["pos"])
data["Negative_Score"] = scores.apply(lambda x: x["neg"])
data["Neutral_Score"]  = scores.apply(lambda x: x["neu"])
data["Compound_Score"] = scores.apply(lambda x: x["compound"])
```

Phase 5 - Sentiment Analysis

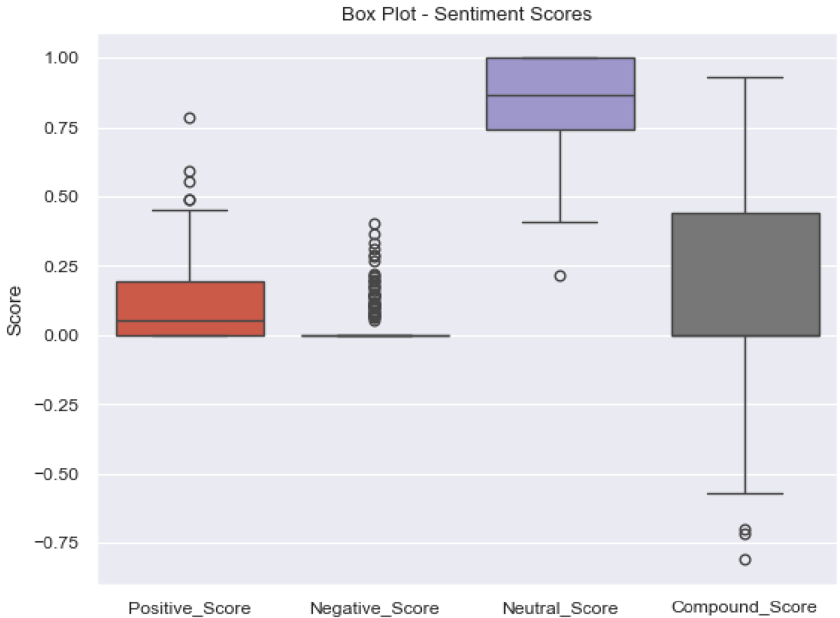
We look at the summary statistics for the sentiment scores:

	Positive_Score	Negative_Score	Neutral_Score	Compound_Score
count	245.000000	245.000000	245.000000	245.000000
mean	0.116216	0.028490	0.855314	0.172913
std	0.146029	0.071052	0.156892	0.343955
min	0.000000	0.000000	0.213000	-0.807400
25%	0.000000	0.000000	0.742000	0.000000
50%	0.053000	0.000000	0.868000	0.000000
75%	0.194000	0.000000	1.000000	0.440400
max	0.787000	0.405000	1.000000	0.928700

The compound score has a reasonable spread with a median of 0.

Phase 5 - Sentiment Analysis

We plot the sentiment scores below:



Phase 5 - Sentiment Analysis

It looks like the compound sentiment scores could be useful as an extra feature - but without access to historical tweets, it would be impossible to tell conclusively, Nevertheless it should be investigated further.

Conclusion

Predicting NSEI open direction is a very interesting problem. We have shown the ability to train models with accuracy of around 75% in the case of the SVM model, and 77% - 79% in the case of the Deep Learning (PyTorch) model - I am confident that with enhanced model tuning, and by adding extra features such as compound sentiment score , we can improve the accuracy significantly.