Deep Learning For Financial Time Series

Xiaodong YANG
yangxiaodong993993@gmail.com

Final Report
Certificate in Quantitative Finance
June 2022 Program

CQF | INSTITUTE

January 2023

Abstract

The direction of the financial market is always stochastic and volatile but is important for financial market anticipates. Analysts now are trying to apply the modelling techniques from Natural Language Processing in the field of Finance to the similarity of having the sequential property in the data. In this research, I have constructed and applied the state-of-art deep learning sequential model, namely Long Short Term Memory Model (LSTM), Stacked-LSTM and LSTM with optimised hyperparameter into the prediction of stock prices on the next day. Moreover, using the prediction, I built up two trading strategies and compared them with the benchmark. My input data contains all technical indicators, which are carefully selected by feature engineering and applied to LSTM models. The result has shown that LSTM with an optimised hyperparameter beats all other models in terms of prediction error and shows much higher returns in my trading strategy over other models. Furthermore, I discovered that the stacked-LSTM model can improve the predictive power over the single LSTM since it has more complex model structures.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

In the field of quantitative trading, predicting the future security returns lies in the center of the industry, as the future trading strategy is always deployed and created based on our view of the financial market in the future. The trading area has two main different methods, namely fundamental analysis and quantitative trading. Fundamental method makes the trading decision based on the subjective view on the industry or company's future direction, it mainly relies on the public information such as market news, corporate statistics as well as a series of financial statement releases. On the other hand, the quantitative trading strategy uses mathematical models to make the decision hence avoids the interruption of human subjectivity and emotion. Traditionally, the methodology of quantitative strategy involves using linear regressions, ARIMA model as well as GARCH model to capture the features of time series and the stochasticity of the volatility. These methods were proved to be effective for a certain period of time in the old regimes. As the regime shift happens in the financial industry, these models became less effective. The quantitative trading industry has therefore shifted into the 'deep learning era', which has been more frequently used nowadays.In this research, we predicted stocks return using the deep learning model, more specifically LSTM and Attention-LSTM models. Conventional financial time series prediction only uses price and volume to predict the future direction. In this work, the input includes abundant technical indicators calculated by open, high, low, adjusted close price, volumes, etc. Usually these technical indicators have significant impacts on future price movements. The output of our model is the return directions on the next day. Once the future return direction is predicted, we will build up a quantitative trading strategy based on the prediction. The return of our strategy is compared with the market Zou & Qu (2020).

SPDR S&P 500 ETF Trust is chosen in this project, which is one of the earliest investment vehicles created. The SPY aims to track the S&P500 index as closely as possible, holding stocks in direct proportion to their weights within the index. Although the SPY expense ratio is slightly higher than other ETF's that also track the S&P500 index, the SPY's history and popularity make it an interesting and challenging ETF to forecast. Given that the trading volume of the S&P500 is so high amongst both retail and institutional investors, any edge that exists is unlikely to remain for long, being rapidly traded out as soon as it appears. This sides slightly with the efficient market hypothesis to some degree, but there is significantly more achievement and interest embedded in beating one of the most popular and rapidly traded instruments available.

2 Data Processing

2.1 Data Collection

The data I have for this research was fully available and I obtained it from yahoo finance by finance API. These data were compiled in the period 31/12/2010 to 31/12/2022, which was divided into two subsets: the training set and the test set, respectively with ratios of 70% and 30%. The following will describe the data. Figure 1 and Figure 2 show the SPY performance to date.

Figure 1: SPY Adjusted Close Prices

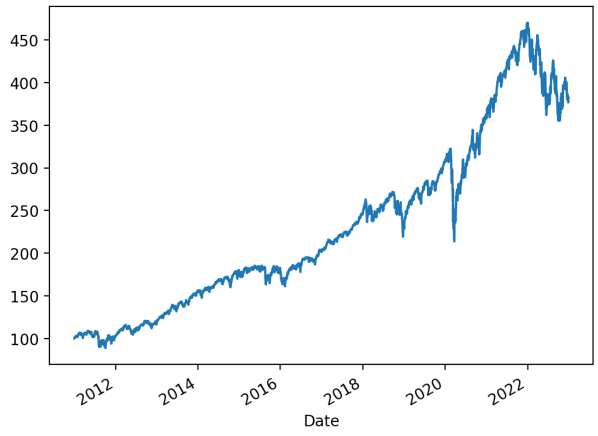
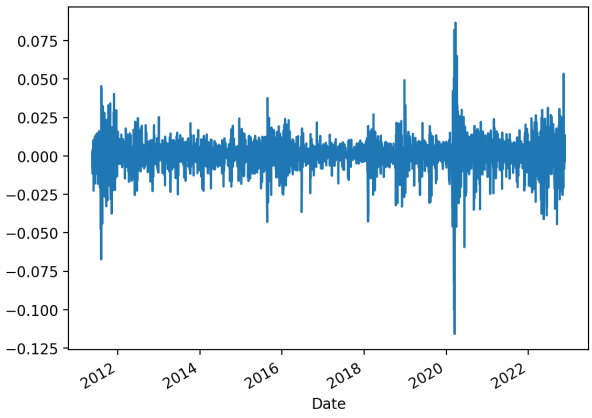


Figure 2: SPY Daily Returns



2.2 Technical Indicators

Technical indicators are available from Technical Analysis Library, including Momentum Indicators, Volume Indicators, Volatility Indicators, Trend Indicators, etc. The features are constructed by these technical indicators. Before feature engineering, several irrelevant variables and NA data need to be removed from the features shown by the below code

```
stock_data.drop(['HILOl_13_21', 'HILOs_13_21', 'PSARl_0.02_0.2',
                 'PSARs_0.02_0.2', 'PSARaf_0.02_0.2', 'QQEl_14_5_4.236',
                 'QQEs_14_5_4.236', 'SUPERTl_7_3.0', 'SUPERTs_7_3.0',
                 'Open', 'High', 'Low', 'Close', 'Adj Close'],
                  axis=1, inplace=True)
stock_data = stock_data.dropna(axis=0)
```

2.3 Labels

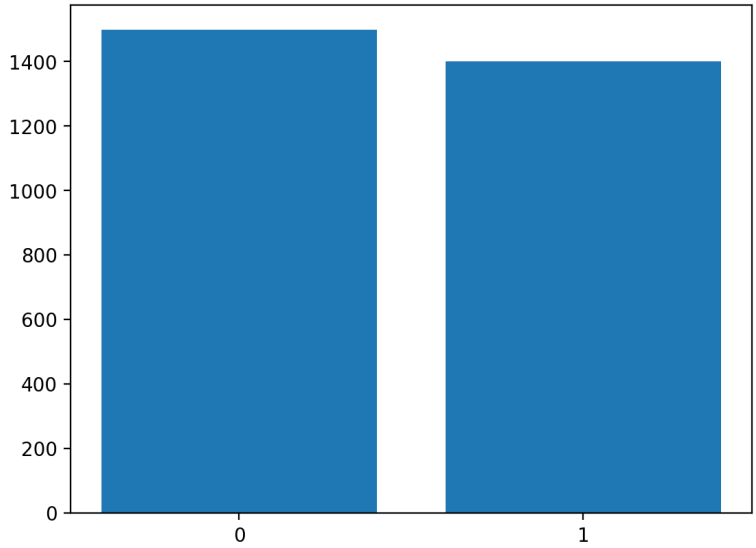
Then "0" and "1" are used to encode the return. To avoid our predictions' accuracy being influenced by the noise that is extremely small near-zero returns, a threshold was set as 0.09%, which is the expense ratio of SPDR S&P 500 ETF Trust referring to Yahoo Finance. In other

words, the returns exceeding its expense ratio were labelled as 1, otherwise, 0, meaning the long indicator will be made only when its return can cover its expense ratio. The labelling criterion is shown by

$$y = \begin{cases} 1, & \text{Returns} > \text{Threshold} \\ 0, & \text{Otherwise} \end{cases}$$

2.4 Imbalanced Class

Figure 3: Class Distribution



Source: My Calculations

Another issue impacting the classification negatively is an unbalanced class of the labels, which might lead to situations in which the [learning algorithm] neural network simply predicts the class with the higher frequency since this already can lead to low loss and high accuracy values Hilpisch (2020). Through observing Figure 3, the frequency of '0' is higher than '1', meaning there is an imbalanced class issue. Hence, it can be fixed by applying appropriate weights to make sure that both classes gain equal importance during the DNN training step Kannan Singaravelu (2022*a*). From the below code, labels '0' and '1' should be applied weights of 0.9670 and 1.0354 respectively.

```
# class frequency
c = stock_data['Returns'].value_counts()
# class weight function
def cwts(stock_data):
    c0, c1 = np.bincount(stock_data['Labels'])
    w0=(1/c0)*(len(stock_data['Returns']))/2
    w1=(1/c1)*(len(stock_data['Returns']))/2
    return {0: w0, 1: w1}
```

```
# check class weights
class_weight = cwts(stock_data)
[Out] {0: 0.9670, 1: 1.0354}
```

3 Feature Engineering

The dimension of data should be reduced. Because high-dimension data might cause two issues. It would cause overfitting when all features are used to train the model. Besides, high-dimension might influence computing speed. Therefore, three methods are employed to reduce the dimension of data, including Boruta, K-Means Clustering, and Self-Organising Maps.

3.1 Boruta

Boruta algorithm is a wrapper built around the random forest classification algorithm, which is relatively quick, can usually be run without tuning of parameters and gives a numerical estimate of the feature importance. Boruta is based on stochasticity which forms the foundation of the random forest classifier. And it adds randomness to the system and collects results from the ensemble of randomized samples one can reduce the misleading impact of random fluctuations and correlations where this extra randomness shall provide us with a clearer view of which attributes are really important Kursa & Rudnicki (2010).

In my study, the features and target were used to train the classifier and got classification. All features are ranked by specific order based on their relevance. Rank 1 means the most relevance. Hence, all features ranked in '1' were selected for the next step. After that, the feature number is reduced from 272 to 64.

3.2 K-Means Clustering

K-Means Clustering is a branch of unsupervised machine learning models that seeks to learn from the properties of the data by identifying groups or clusters in the dataset. The difference between clustering and classification is that the former seeks to identify a set of similar data points and calls the resulting set a cluster Xu & Tian (2015).

The K - Means Clustering algorithm searches for a predetermined number of clusters within an unlabeled dataset. There are two assumptions that the optimal cluster Kannan Singaravelu (2022*b*)

- cluster center which is the mean of all the points belonging to the cluster
- each point is closer to its own cluster center than to other cluster centers

The K-Means Algorithm seeks to find k clusters within a data set. The clusters are chosen to

reduce the inertia, the objective function

$$\min_{C_k} \sum_{k} \sum_{X_j \in C_k} ||X_j - \mu_k||^2$$

and the centroid of a cluster μ_k is equal to

$$\mu_k = \frac{1}{C_k} \sum_{X_j \in C_k} X_j$$

Where C_k is the number of points in cluster k.

The cluster assignments are done based on squared Euclidean distance and the algorithm tries to minimize the objective function. This process is repeated, iteratively, until the sum of the distance is minimized Kannan Singaravelu (2022*b*).

Algorithm 1 K-Means Clustering Algorithm

1: Randomly selecting k -centroids or cluster center
2: **while** Not Converge . . . **do**
3: a. E-Step: assign points to the nearest cluster center
4: b. M-Step: set the cluster center to the mean
5: **end while**

Algorithm 1 was utilised in this part to analyse the clustering. In this algorithm, the number of clusters is a hyperparameter. Hence, it is crucial for the model to select the optimal number of clusters. Too many or few clusters are not appropriate. Instead, The Elbow plot was employed to identify the infection point where increasing the number of clusters no longer results in a significant drop in inertia and select the optimal number of clusters Kannan Singaravelu (2022*b*).

Figure 4: Elbow Plot

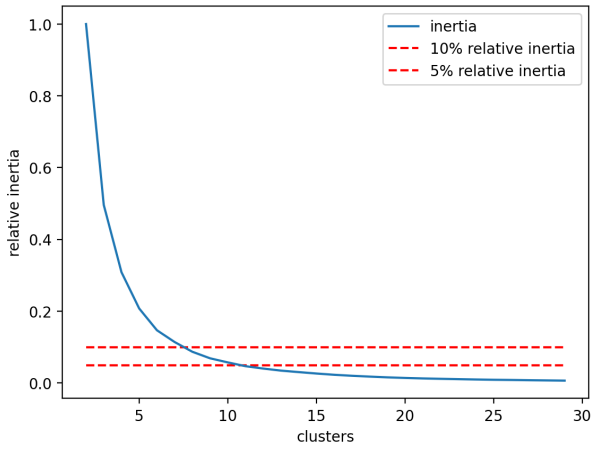
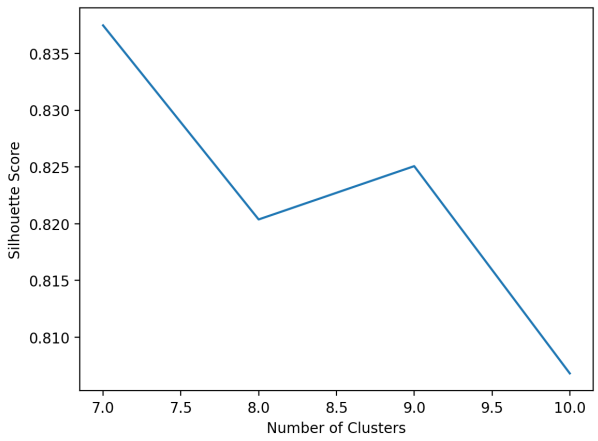


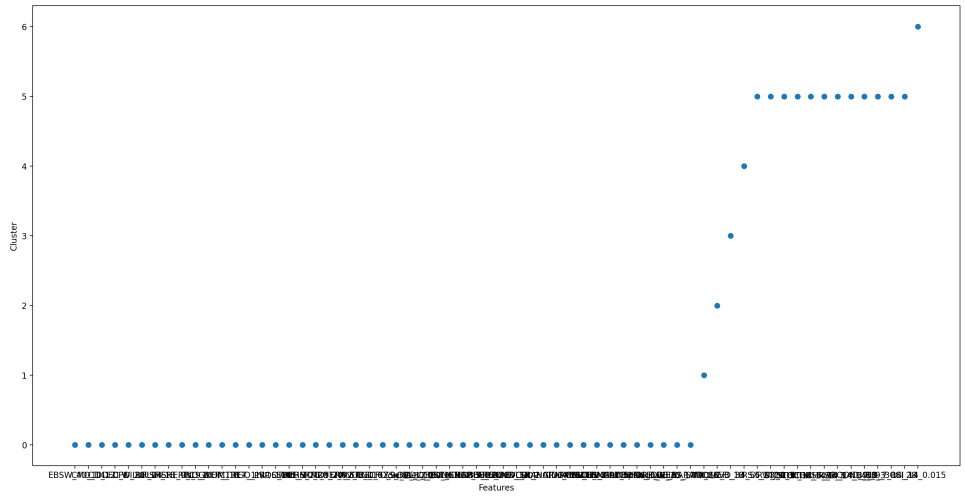
Figure 5: Silhouette Score



The K-Means implementation seeks to find the minimum number of clusters required to achieve low, stable inertia. From Figure 4, it seems the range of the optimal value is from 7 to 10 in

relative inertia from 5% to 10%. Besides, Silhouette Coefficient is another method to measure how dense and separated are the clusters. And its range is from -1 to 1. The samples with larger values are closer to their clusters than others. No doubt that the Silhouette coefficient implementation is more reliable than the Elbow plot since it provides the precise score, which is straightforward Zhou & Gao (2014). Figure 5 shows the Silhouette coefficients with the range of cluster numbers from 7 to 10. It includes that a cluster number equal to 7 has the highest Silhouette Score, meaning 7 is the optimal cluster number in this algorithm. From Figure 6, most of the features are at cluster "5".

Figure 6: SPY Features Clustering



Source: My Calculations

3.3 Self Organising Maps

The self-organising map (SOM) is one of the widely implemented neural network models, which is an unsupervised learning method and computationally simple. It produces a topology-preserving mapping between high-dimensional input space and low-dimensional map space. Hence, it is usually used for feature detection Lau et al. (2006). SOM applies competitive learning when compared to error-correlated learning, which involves backpropagation and gradient descent. In SOM, the training data has no labels and the map learns to differentiate and distinguish features based on similarities Kannan Singaravelu (2022*d*). The SOM algorithm is shown by Algorithm 2.

Algorithm 2 Self Organising Maps Algorithm

1: Initialise weights w_{ij}
2: Calculate distance $d_{ij} = \sum(w_{ij} - x_i)^2$
3: Calculate $w_{ij}(t+1) = w_{ij}(t) + \theta(t) \cdot \alpha(t) \cdot (x_i(t) - w_{ij}(t))$

Where

$x_i(t)$ is the characteristics of the samples

$\alpha(t)$ is a learning rate

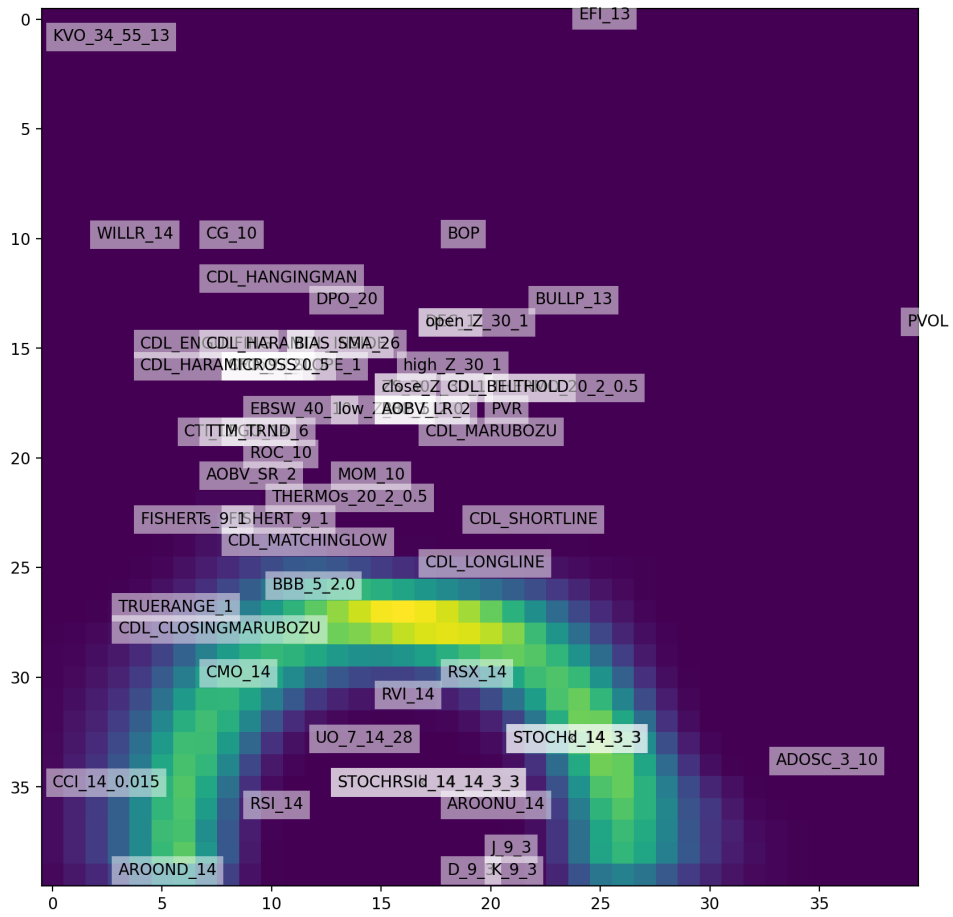
$\theta(t)$ is the neighbourhood function

First of all, the initialised dimension of SOM should be determined. The recommendation dimension about the rule of thumb is based on Kannan Singaravelu (2022*d*)

$$Q^* \approx 5 \times \sqrt{I}$$

Where I is the number of features. The data includes 64 features in this part. Therefore, Q^* is equal to 40. The initialised dimension of SOM is 40×40. The visualisation of SOM is shown by Figure 7. The big purple area contains the majority of the features. Following the SOM selection criteria, the selected features are shown by Table 1.

Figure 7: Self Organising Maps



4 Exploratory Data Analysis

Exploratory Data Analysis (EDA) refers to the critical process of performing initial investigations on data so as to discover patterns, spot anomalies, test hypotheses, check assumptions, etc Kannan Singaravelu (2022*a*). Several important techniques can be used to describe the significant statistical analysis and graphical representations, including correlation matrix, pairwise plot, etc.

Table 1: Features List

	Features
1	KVO_34_55_13
2	EBSW_40_10
3	DPO_20
4	WILLR_14
5	BULLP_13
6	TRUERANGE_1
7	EFI_13
8	ADOSC_3_10
9	BOP
10	PVR

4.1 Features Describe

Figure 8 shows the descriptions of all selected features before applying the scaler. The population of the data is 2899. Several important statistics are described by it, including mean, standard deviation, maximum, etc.

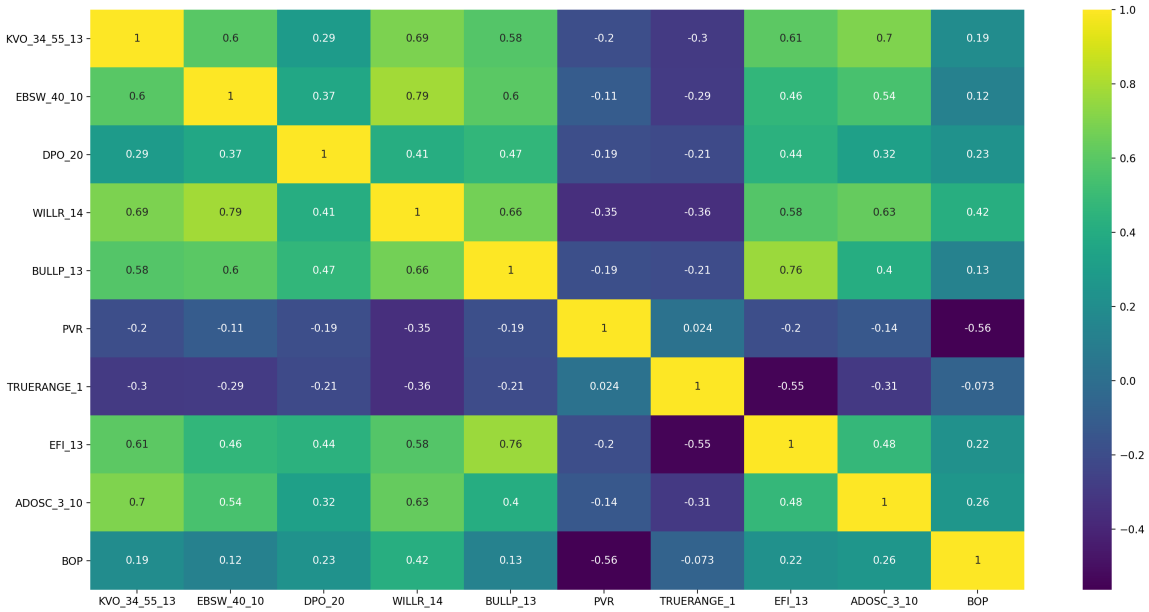
Figure 8: Feature Descriptions

	count	mean	std	min	25%	50%	75%	max
KVO_34_55_13	2899.0	-3.152470e+04	7.442190e+06	-5.814442e+07	-4.040862e+06	4.774819e+05	4.347242e+06	2.995325e+07
EBSW_40_10	2899.0	2.145788e-01	8.124232e-01	-9.999737e-01	-7.566179e-01	6.575463e-01	9.683719e-01	9.999898e-01
DPO_20	2899.0	-1.420945e-01	3.917509e+00	-2.789100e+01	-1.595507e+00	3.150635e-02	1.299499e+00	2.668348e+01
WILLR_14	2899.0	-3.583892e+01	3.061310e+01	-1.000000e+02	-5.989283e+01	-2.692308e+01	-8.134081e+00	-0.000000e+00
BULLP_13	2899.0	1.895417e+00	3.997989e+00	-2.681032e+01	3.935218e-01	1.819845e+00	3.505296e+00	2.179554e+01
PVR	2899.0	2.410486e+00	1.016686e+00	1.000000e+00	2.000000e+00	2.000000e+00	3.000000e+00	4.000000e+00
TRUERANGE_1	2899.0	3.254581e+00	3.039621e+00	4.400024e-01	1.389999e+00	2.190002e+00	3.914993e+00	3.196001e+01
EFI_13	2899.0	-2.309790e+07	1.559325e+08	-2.469719e+09	-4.078148e+07	7.807426e+06	3.476777e+07	4.330140e+08
ADOSC_3_10	2899.0	2.980900e+07	7.917299e+07	-4.047211e+08	-1.544632e+07	3.527280e+07	7.614915e+07	3.343292e+08
BOP	2899.0	5.717779e-02	5.265266e-01	-9.838320e-01	-3.874843e-01	7.758349e-02	5.138909e-01	1.000000e+00

4.2 Correlation Matrix

To check the correlation among features and avoid multicollinearity, the correlation matrix was employed shown by Figure 9. Besides, the below code was implemented to check if there are pair variables with absolute correlation exceeding ± 0.8, which proves that there are no variables with higher correlations.

Figure 9: Feature Correlation Matrix

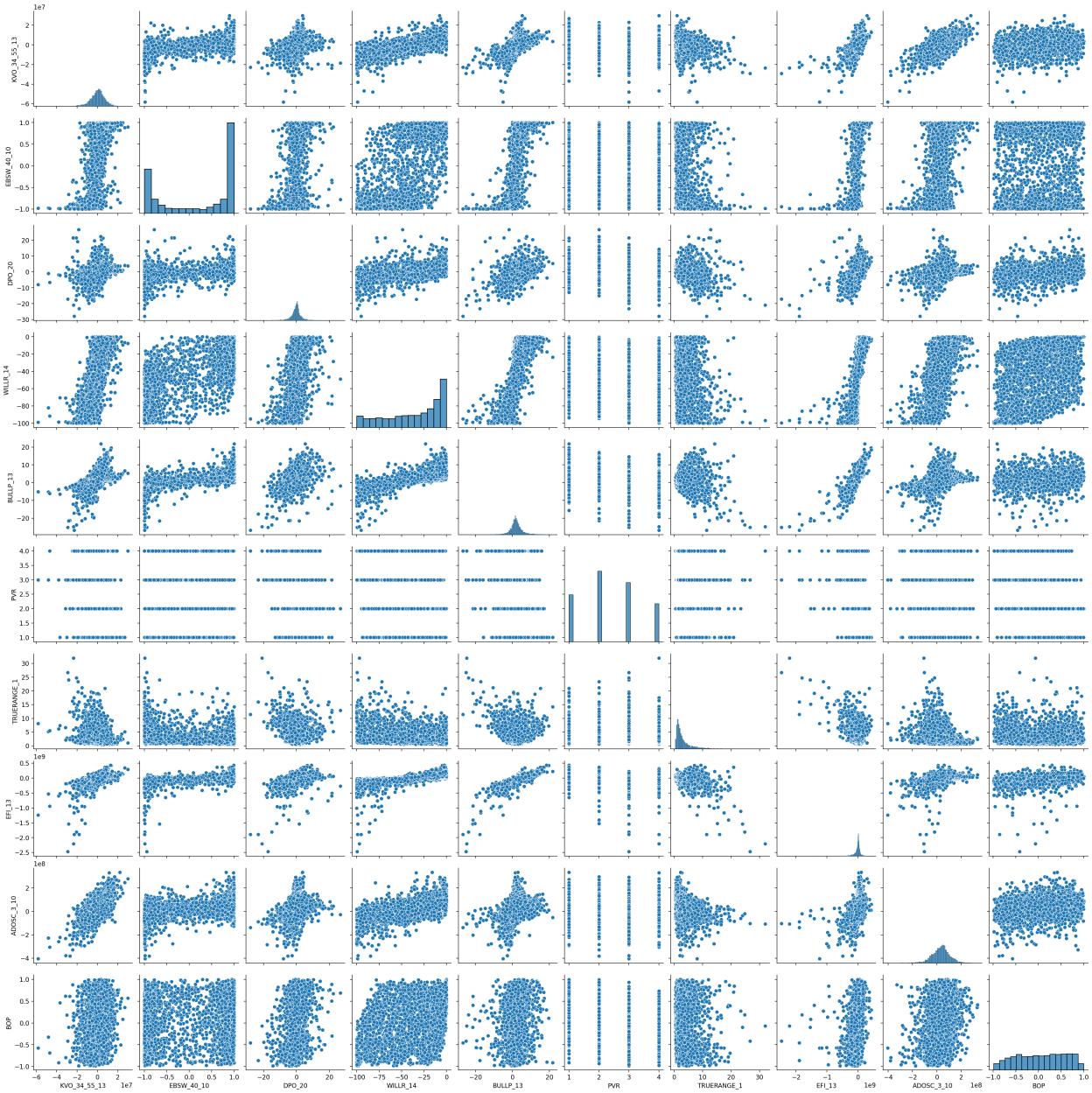



```
# setting correlation maximum threshold
rho_max = 0.80
# detecting pairwise features with abs(corr) > rho_max
idxPairwise = np.where(abs(corrFeatMatrix) >= rho_max)
lstPairwise = [
    [corrFeatMatrix.index[x], corrFeatMatrix.columns[y],
    round(corrFeatMatrix.iloc[x, y], 3)]
    for x, y in zip(*idxPairwise) if x != y and x < y]
print("::>> Pairwise features with correlation = 0.80 :")
print("::::>>>> List/list represents [Feat1, Feat2, Corr]:")
```

4.3 Pairwise Plot

The pairwise plot is another method to describe the relationship among the features, which calculates different subplots to see if there are differentiable relationships between each pair with respect to the labels. For instance, there is a positive relationship between 'ADOSC_3_10' and 'KVO_34_55_13'. Besides, the relationship between 'EBSW_40_10' and 'BULLP_13' is close to logistic regression.

Figure 10: Feature Pairwise Plot



5 Experiments And Results

There are some factors to impact the performance of the Deep Neutral Network, including the number of hidden layers, the type of selected activation, and relevant parameters. Therefore, three models are employed in this project, Simple LSTM, Stacked-LSTM, and LSTM with Hyperparameter Optimization to evaluate the performance, develop an optimal LSTM model for stock trend projections, and construct the trading strategies. In the first two models, MSE and MAE are the loss functions to optimise the model shown by Equation 1 and Equation 2.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{1}$$

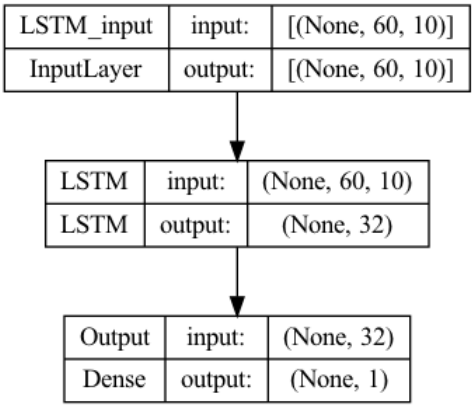
$$\text{MAE} = \frac{\sum_{i=1}^{N}(y_i - y_i)}{N} \tag{2}$$

5.1 Simple LSTM

The simple LSTM model structure is shown in Figure 11. The lookback period is 60 days. It only has one hidden layer with the activation function of "relu", which is defined as Equation 3.

$$f(x) = x^+ = \max(0, x) \begin{cases} x, & x > 0 \\ 0, & \text{Otherwise} \end{cases} \tag{3}$$

Figure 11: Simple LSTM Model



Python code for the single LSTM model is as follows

```
# Create a model
def create_model(hu=256, lookback=60, features=numfeat):
    tensorflow.keras.backend.clear_session()
    # instantiate the model
    model = Sequential()
    model.add(LSTM(units=hu, input_shape=(lookback, features),
    activation = 'relu', return_sequences=False, name='LSTM'))
    model.add(Dense(units=1, name='Output'))
    # specify optimizer separately (preferred method))
    opt = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-08)
    # model compilation
    model.compile(optimizer=opt, loss='mse', metrics=['mse'])
    return model
```

The confusion matrix of the simple LSTM model is shown by Figure 12, which displays the detailed classification. And Figure 13 shows the Area under the ROC curve (AUC) is 0.62,

representing a binary classification model's ability to separate positive classes from negative classes. Moving on to the classification report, the precision indicates the proportion of positive identifications (model predicted class 1) which were actually correct. A model which produces no false positives has a precision of 1.0. Recall indicates the proportion of actual positives which were correctly classified. A model which produces no false negatives has a recall of 1.0. The F1 score is the combination of precision and recall. A perfect model achieves an F1 score of 1.0. Support is the number of samples for each class. From this classification report, the accuracy is 0.64, which is just much better than random. The values of the macro avg are the same as the weighted avg, meaning that our results are not damaged by the imbalanced class of the sample.

Figure 12: Simple LSTM Confusion Matrix

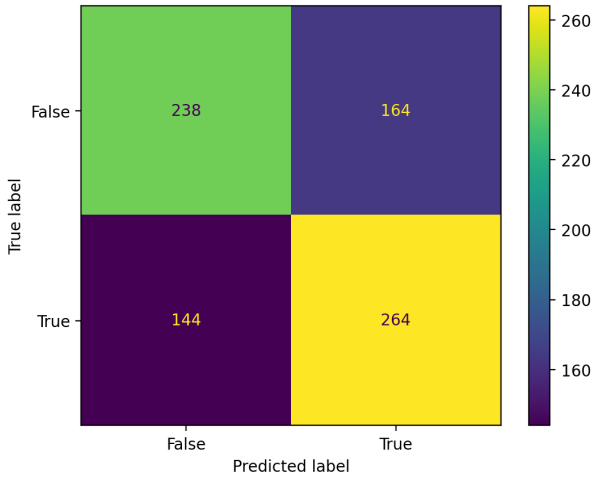
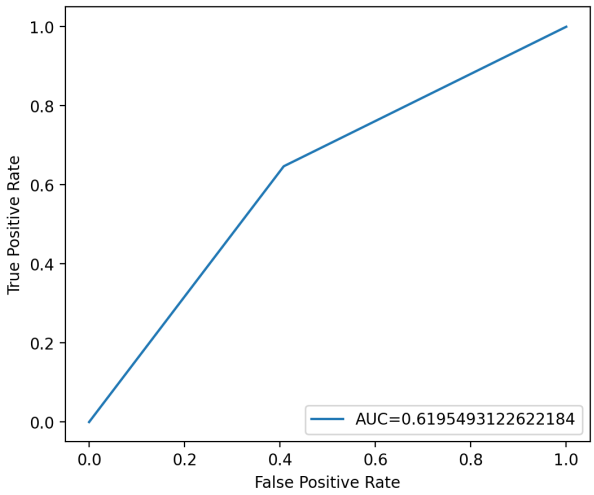


Figure 13: Simple LSTM ROC

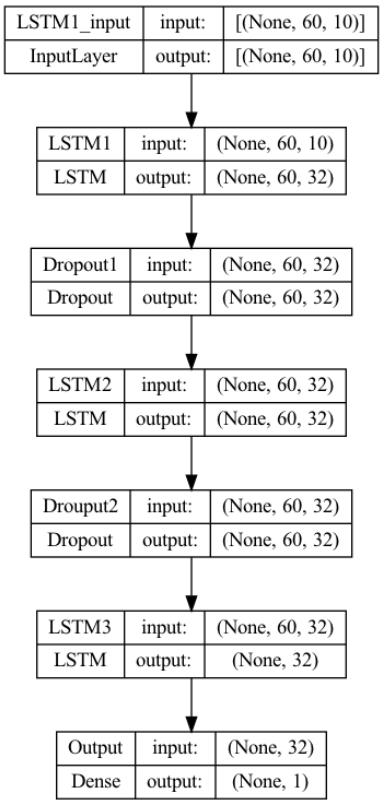



```
# The classification report:
              precision    recall  f1-score   support

           0       0.62      0.59      0.61       402
           1       0.62      0.65      0.63       408

    accuracy                           0.62       810
   macro avg       0.62      0.62      0.62       810
weighted avg       0.62      0.62      0.62       810
```

5.2 Stacked-LSTM Model

The Stacked-LSTM Model was created by adding the number of hidden layers with 3 hidden layers, relu, gelu, and selu respectively shown by Figure 14.

Figure 14: Stacked-LSTM Model



Python code for the Single LSTM model is as follows

```
# Create a model
def create_model(hu=256, lookback=60, features=numfeat):
    tensorflow.keras.backend.clear_session()
    # instantiate the model
    model = Sequential()
    model.add(LSTM(units=hu, input_shape=(lookback, features),
    activation = 'relu', return_sequences=True, name='LSTM1'))
    model.add(Dropout(0.4, name='Dropout1'))
    model.add(LSTM(units=hu, activation = 'gelu',
    return_sequences=True, name='LSTM2'))
    model.add(Dropout(0.4, name='Drouput2'))
    model.add(LSTM(units=hu, activation = 'selu',
    return_sequences=False, name='LSTM3'))
    model.add(Dense(units=1, activation='sigmoid', name='Output'))
    # specify optimizer separately (preferred method))
    opt = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-08)
    # model compilation
    model.compile(optimizer=opt, loss='mse', metrics=['mae'])
    return modelizer=opt, loss='mse', metrics=['mse'])
    return model
```

Figure 15 and Figure 16 show the accuracy from the Stacked-LSTM Model has a little improvement since its AUC of 0.63 is more than 0.62 of the Simple LSTM Model by adding the number of hidden layers. Besides, executed time is another key point. Computing speed is another key point. We need to trade off the model accuracy and computing speed.

Figure 15: Stacked LSTM Confusion Matrix

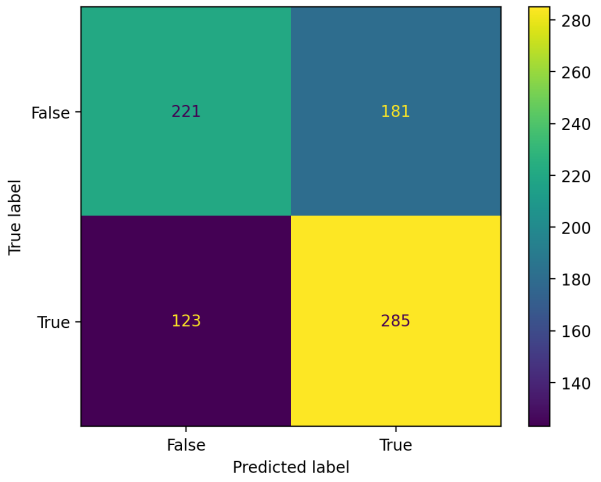
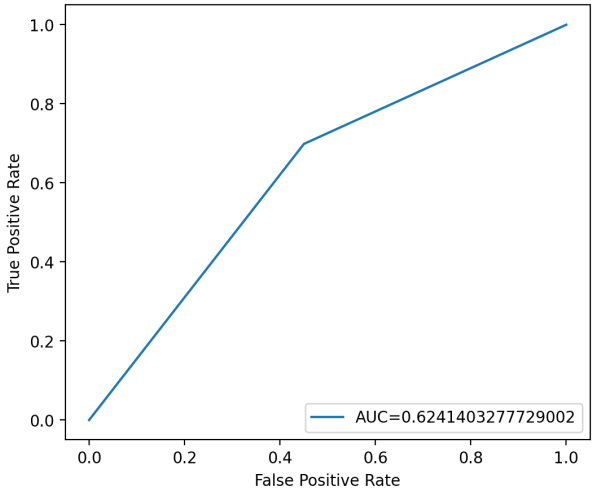


Figure 16: Stacked LSTM ROC




```
# The classification report:
              precision    recall  f1-score   support

           0       0.64      0.55      0.59       402
           1       0.61      0.70      0.65       408

    accuracy                           0.62       810
   macro avg       0.63      0.62      0.62       810
weighted avg       0.63      0.62      0.62       810
```

5.3 Hyperparameter Optimization LSTM

Hyperparameter is a training parameter whose value is used to control the learning process as compared to the values of other parameters (typically weights) that are learned; set before training the model and are not learned by the model during the training process while the model parameters are learned by the model, from the data, during the training process. Hyperparameter Optimization is another the most common method to optimise for better results by choosing hidden layers, number of neurons, choice of activation function, number of epochs, learning rate, batch size and regularisation parameters Kannan Singaravelu (2022*c*). There are three methods for Hyperparameter Optimization, Random Search, HyperBand, and Bayesian Optimization. In this project, Bayesian Optimization is employed to optimise the Hyperparameter. The best parameters are as follows

```
Trial 15 Complete [00h 00m 56s]
val_accuracy: 0.644444465637207
Best val_accuracy So Far: 0.6567901372909546
Total elapsed time: 00h 09m 31s
INFO:tensorflow:Oracle triggered exit
{'units1': 32, 'units2': 32, 'units3': 32, 'Dropout_rate': 0.2,
'learning_rate': 0.001, 'activation': 'elu'}
```

Figure 17: Hyperparameter Optimization LSTM

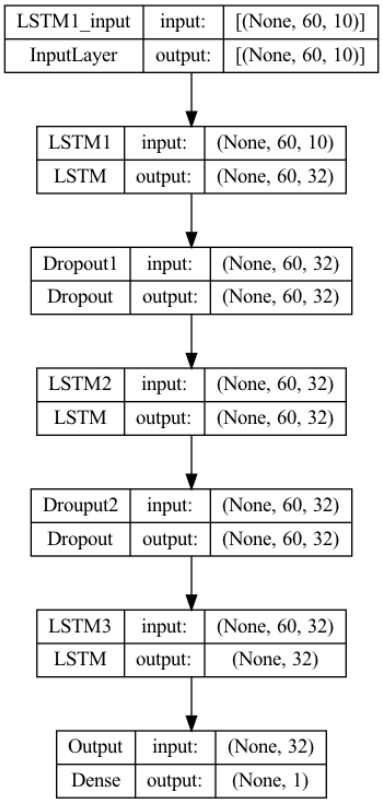


Figure 18 and Figure 19 show the results from the Stacked-LSTM Model with three hidden layers. From Figure 18, the true classification number is larger than the Simple Model. Meanwhile, its AUC is also better equal to 0.64. It proves that Hyperparameter Optimization can improve the accuracy of the LSTM model.

Figure 18: Optimised LSTM Confusion Matrix

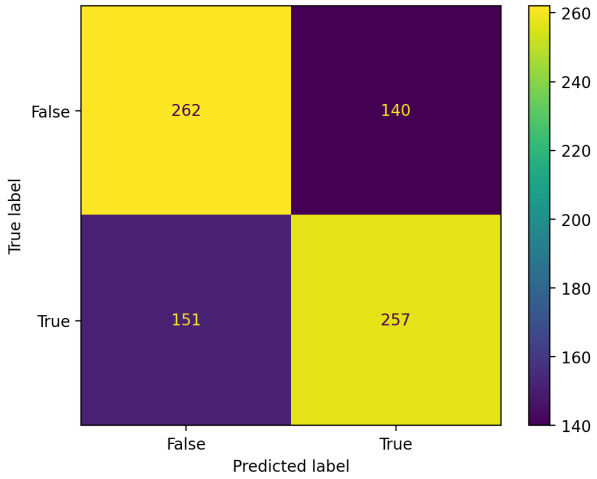
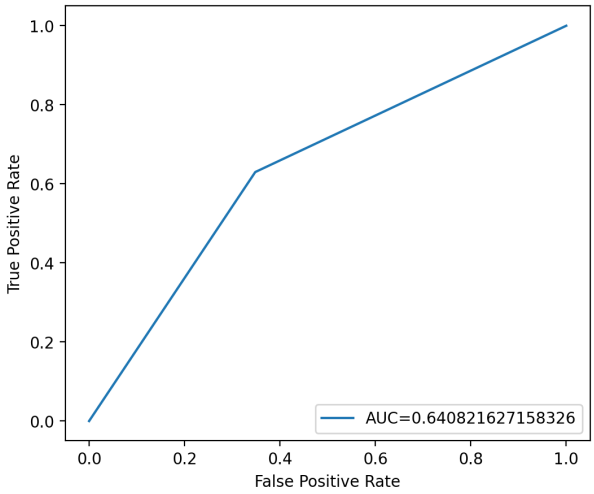


Figure 19: Optimised LSTM ROC




```
# The classification report:
              precision    recall  f1-score   support

           0       0.63      0.65      0.64       402
           1       0.65      0.63      0.64       408

    accuracy                           0.64       810
   macro avg       0.64      0.64      0.64       810
weighted avg       0.64      0.64      0.64       810
```

6 Trading Strategies

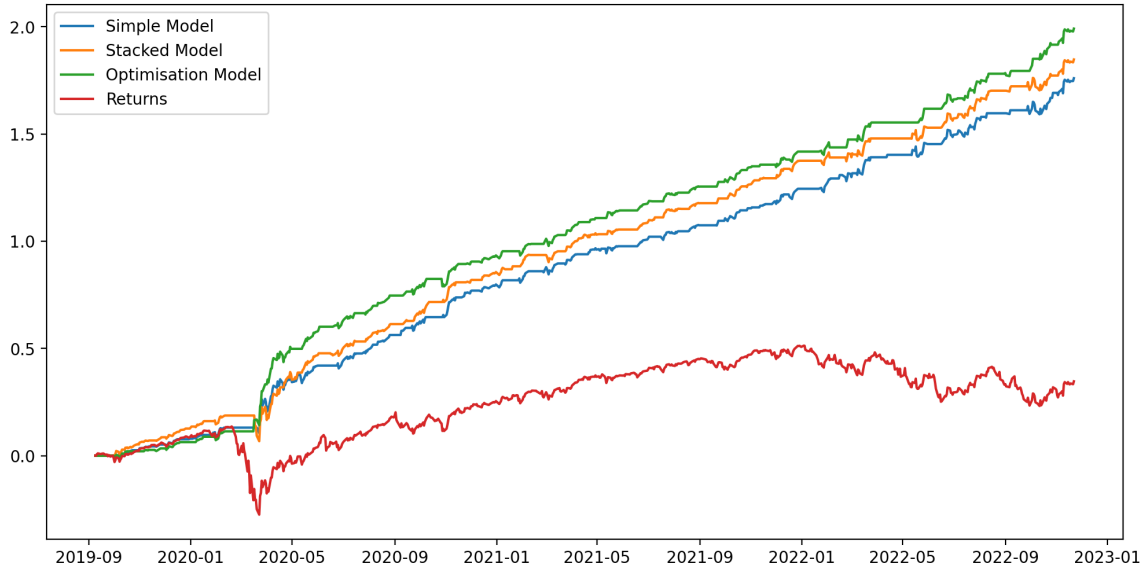
6.1 Long-Only Strategy

For the prediction for each stock on the next day, if the prediction is positive, we buy the stock at the open price and sell the stock at a close price on the same day. If the prediction is negative, no action is taken. Figure 20 shows the performance of applying the model projections to the strategies during the test periods. The cumulative returns are shown by Table 2. Moreover, the LSTM with Hyperparameter Optimisation has the highest cumulative returns. Meanwhile, its accuracy is highest in these three models.

Table 2: Long-Only Returns

Models	Cumulative Returns
Single LSTM	176.01%
Stacked LSTM	184.74%
Optimisation LSTM	199.13%
Holding	34.67%

Figure 20: Long-Only Strategy Return



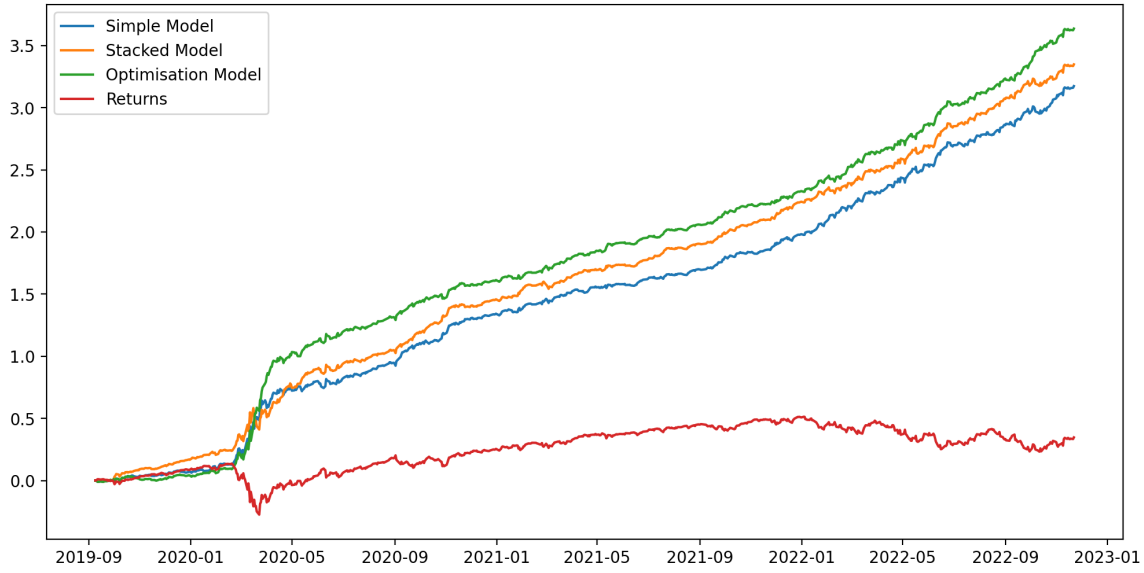
6.2 Long-Short Strategy

For the prediction for each stock on the next day, if the prediction is positive, we buy the stock at the open price and sell the stock at a close price on the same day. If the prediction is negative, we short-sell the stock at the open price and close out the short-sell at the close price. Figure 21 shows the cumulative returns of three strategies from three LSTM models. The results are the same as the Long-Only strategy. However, the cumulative returns of the Long-Short strategy outstrip the Long-Only strategy. Table 3 shows the detailed results.

Table 3: Long-Short Returns

Models	Cumulative Returns
Single LSTM	317.34%
Stacked LSTM	334.80%
Optimisation LSTM	363.58%
Holding	34.67%

Figure 21: Long-Short Strategy Return



7 Conclusion

This project establishes a forecasting framework to predict the direction (Up/Down) of stocks. I leveraged the combinations of price, volumes and corporate statistics as input data by Technical Analysis Library. Three models are proposed, developed, trained, and tested: Single LSTM, Stacked-LSTM, and LSTM with optimised hyperparameter. And two trading strategies are created according to the model predictions. The Optimised hyperparameter LSTM shows more superior results over other models due its its ability to search the best parameters of the LSTM models. Hence the Optimised hyperparameter LSTM is more able to capture the information in the time series and more suitable in predicting financial time series. Our superior trading return from the Optimised hyperparameter LSTM further validates our experimental result. Moreover, we have shown that despite the more complicated model structure of stacked-LSTM over single LSTM model, the stacked-LSTM has better model performance over the single LSTM model. One drawback of the Stacked-LSTM might cause overfitting, which can be avoid by Ridge or LASSO regression.

Table of Contents

Deep Learning For Financial Time Series

1 Import Libraries

In [1]:

```
# Base Libraries
import numpy as np
import pandas as pd
import datetime
from datetime import datetime
import copy

import yfinance as yf
import cufflinks as cf
cf.set_config_file(offline=True, dimensions=((1000,600)))

# Visualization
import matplotlib
import matplotlib.pyplot as plt
# # Plot settings
# plt.style.use('seaborn-white')
# matplotlib.rcParams['figure.figsize'] = [12.0, 6.0]
# matplotlib.rcParams['font.size'] = 10
# matplotlib.rcParams['lines.linewidth'] = 2.0
# matplotlib.rcParams['grid.color'] = 'white'
import seaborn as sns
# plt.style.use('seaborn')
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

import ta
from ta import add_all_ta_features
from ta.utils import dropna

# Preprocessing
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler, RobustScaler, StandardScaler, Normalizer
from sklearn.model_selection import train_test_split, TimeSeriesSplit, GridSearchCV
from sklearn.model_selection import RandomizedSearchCV, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import roc_curve, auc
from sklearn import metrics

# Import sklearn modules
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.pipeline import Pipeline

# SOM & Sklearn library
from minisom import MiniSom

# XGBoost Classifier
from xgboost import XGBClassifier, plot_importance, to_graphviz

# import shap

import datetime
from pathlib import Path

results_path = Path('results', 'lstm_time_series')
if not results_path.exists():
    results_path.mkdir(parents=True)

# imports
from src.config import *
pd.set_option('display.max_columns', 30)
pd.set_option('display.max_rows', 1000)

# import tensorflow and keras
import tensorflow
# tensorflow modules
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, LSTM
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.utils import plot_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard

# Load the TensorBoard notebook extension
%load_ext tensorboard

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

executed in 8.70s, finished 20:07:44 2023-01-23

In [2]:
```
# def set_seeds(seed=2023):
#     random.seed(seed)
#     np.random.seed(seed)
#     tensorflow.random.set_seed(seed)

# Plot a confusion matrix.
# cm is the confusion matrix, names are the names of the classes.
def plot_confusion_matrix(cm, names, title='Confusion matrix', cmap=plt.cm.Blues):
    pit.imshow (cm, interpolation='nearest', cmap=cmap)
    pit.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(names))
    plt.xticks(tick_marks, names, rotation=45)
    plt.yticks(tick_marks, names)
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Plot an ROC. pred - the predictions, y - the expected output.
def plot_roc(pred,y):
    fpr, tpr, _ = roc_curve(y, pred)
    roc_auc = auc(fpr, tpr)

    plt.figure()
    pIt.plot (fpr, tpr, label='ROC curve (area = %0.2f) ' % roc_auc)
    pit.plot ([0, 1], [0, 1], 'k--')
    pit.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    pIt.ylabel('True Positive Rate')
    pit.title('Receiver Operating Characteristic (ROC)')
    plt.legend (loc="lower right")
    pit.show()
```
executed in 5ms, finished 20:07:44 2023-01-23

2 Loading the Historial Data

In [3]:
```
set_seeds(2023)
```
executed in 3ms, finished 20:07:44 2023-01-23

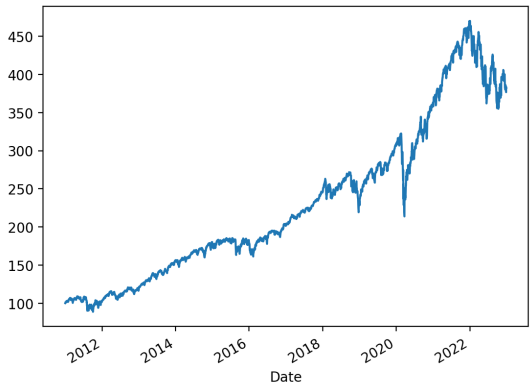
In [4]:
```
ticker = 'SPDR S&P 500 ETF Trust'
```
executed in 2ms, finished 20:07:44 2023-01-23

In [5]:
```
row_data = yf.download('SPY', start='2010-12-31',end='2022-12-31')
```
executed in 575ms, finished 20:07:45 2023-01-23

```
[**********************100%**********************]  1 of 1 completed
```

In [105]:
```
# row_data['Adj Close'].plot(title= ticker + '  Adjusted Close')
row_data['Adj Close'].plot()
```
executed in 249ms, finished 20:30:20 2023-01-23

Out[105]:
```
<AxesSubplot:xlabel='Date'>
```



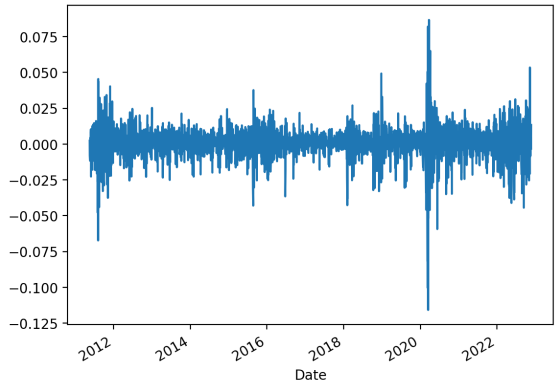
In [7]:
```
threshold = 0.09/100   # Expense Ratio
```
executed in 3ms, finished 20:07:45 2023-01-23

In [8]:
```
row_data.ta.strategy('All')
```
executed in 3.19s, finished 20:07:48 2023-01-23

```
131it [00:03, 43.51it/s]
```

In [9]:
```
stock_data = row_data.copy()
```
executed in 6ms, finished 20:07:48 2023-01-23

In [106]:
```
stock_data['Returns'] = np.log(row_data['Adj Close']).diff().fillna(0)
# stock_data['Returns'].plot(title= ticker + '  Daily Returns')
stock_data['Returns'].plot()
stock_data['Labels'] = stock_data['Returns'].apply(lambda x: 0 if x<=threshold else 1)
```
executed in 279ms, finished 20:30:39 2023-01-23

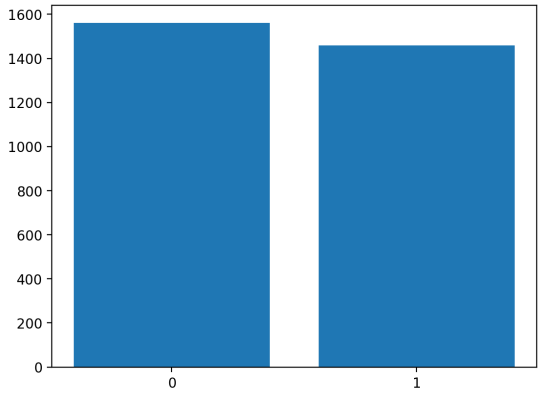


In [11]:
```
stock_data.query('Labels == 1').shape[0]
```
executed in 21ms, finished 20:07:49 2023-01-23

Out[11]: 1459

In [12]:
```
label_list = ['0', '1']
count0 = stock_data.query('Labels == 0').shape[0]
count1 = stock_data.query('Labels == 1').shape[0]
height = [count0, count1]
plt.bar(label_list, height=height)
```
executed in 155ms, finished 20:07:49 2023-01-23

Out[12]: <BarContainer object of 2 artists>



In [13]:
```
stock_data.drop(['HILOl_13_21', 'HILOs_13_21', 'PSARl_0.02_0.2',
                 'PSARs_0.02_0.2', 'PSARaf_0.02_0.2', 'QQEl_14_5_4.236',
                 'QQEs_14_5_4.236', 'SUPERTl_7_3.0', 'SUPERTs_7_3.0',
                 'Open', 'High', 'Low', 'Close', 'Adj Close'],
                axis=1, inplace=True)
```
executed in 6ms, finished 20:07:49 2023-01-23

In [14]:
```
stock_data.describe().T
```
executed in 496ms, finished 20:07:50 2023-01-23

Out[14]:

	count	mean	std	min	25%	50%	75%	max
Volume	3021.0	1.106428e+08	6.434431e+07	2.027000e+07	6.805420e+07	9.365700e+07	1.344574e+08	7.178287e+08
ABER_ZG_5_15	3017.0	2.503349e+02	9.518406e+01	1.130060e+02	1.807540e+02	2.263233e+02	2.998340e+02	4.767867e+02
ABER_SG_5_15	3006.0	2.539932e+02	9.661298e+01	1.163644e+02	1.838875e+02	2.280344e+02	3.033147e+02	4.821783e+02
ABER_XG_5_15	3006.0	2.475736e+02	9.355612e+01	1.096476e+02	1.809428e+02	2.248068e+02	2.968555e+02	4.713950e+02
ABER_ATR_5_15	3006.0	3.209786e+00	2.331560e+00	9.770435e-01	1.592276e+00	2.189555e+00	3.996600e+00	1.498682e+01
ACCBL_20	3002.0	2.431010e+02	9.147485e+01	1.077167e+02	1.773469e+02	2.220095e+02	2.913095e+02	4.572807e+02
ACCBM_20	3002.0	2.503653e+02	9.489745e+01	1.163625e+02	1.810781e+02	2.259125e+02	2.980766e+02	4.699965e+02
ACCBU_20	3002.0	2.575436e+02	9.849327e+01	1.235737e+02	1.848288e+02	2.292262e+02	3.050152e+02	4.832534e+02
AD	3021.0	1.639830e+10	7.824943e+09	5.067631e+07	1.064866e+10	1.780189e+10	2.331822e+10	2.700806e+10
ADOSC_3_10	3012.0	3.083129e+07	7.932924e+07	-4.047211e+08	-1.470026e+07	3.563843e+07	7.706414e+07	3.343292e+08
ADX_14	2994.0	2.180187e+01	7.506001e+00	8.593231e+00	1.595191e+01	2.066805e+01	2.651445e+01	5.300331e+01

In [15]:
```
stock_data.shape
```
executed in 3ms, finished 20:07:50 2023-01-23

Out[15]: (3021, 272)

In [16]:
```
stock_data.isna().sum()
```
executed in 10ms, finished 20:07:50 2023-01-23

Out[16]:
```
Volume                0
ABER_ZG_5_15          4
ABER_SG_5_15         15
ABER_XG_5_15         15
ABER_ATR_5_15        15
ACCBL_20             19
ACCBM_20             19
ACCBU_20             19
AD                    0
ADOSC_3_10            9
ADX_14               27
DMP_14               13
DMN_14               13
ALMA_10_6.0_0.85     10
AMATe_LR_8_21_2       0
AMATe_SR_8_21_2       0
AO_5_34              33
OBV                   0
OBV_min_2             1
```

In [17]:
```
# drop nan values
stock_data = stock_data.dropna(axis=0)
```
executed in 8ms, finished 20:07:50 2023-01-23

In [18]:
```
stock_data.isna().sum()
```

executed in 9ms, finished 20:07:50 2023-01-23

Out[18]:
```
Volume                    0
ABER_ZG_5_15              0
ABER_SG_5_15              0
ABER_XG_5_15              0
ABER_ATR_5_15             0
ACCBL_20                  0
ACCBM_20                  0
ACCBU_20                  0
AD                        0
ADOSC_3_10                0
ADX_14                    0
DMP_14                    0
DMN_14                    0
ALMA_10_6.0_0.85          0
AMATe_LR_8_21_2           0
AMATe_SR_8_21_2           0
AO_5_34                   0
OBV                       0
OBV_min_2                 0
```

3 Features Selection

In [19]:
```
y = stock_data.iloc[:,-1:]
X = stock_data.iloc[:,:-2]
```

executed in 3ms, finished 20:07:50 2023-01-23

In [20]:
```
# class frequency
c = stock_data['Returns'].value_counts()

# class weight function
def cwts(stock_data):
    c0, c1 = np.bincount(stock_data['Labels'])
    w0=(1/c0)*(len(stock_data['Returns']))/2
    w1=(1/c1)*(len(stock_data['Returns']))/2
    return {0: w0, 1: w1}

# check class weights
class_weight = cwts(stock_data)
class_weight
```

executed in 7ms, finished 20:07:50 2023-01-23

Out[20]:
```
{0: 0.966977985323549, 1: 1.0353571428571429}
```

3.1 Borutapy

In [21]:
```
# define random forest classifier
forest = RandomForestClassifier(n_jobs=-1,
                                class_weight=cwts(stock_data),
                                random_state=42,
                                max_depth=5)

# train the model
forest.fit(X, y)
```

executed in 277ms, finished 20:07:50 2023-01-23

Out[21]:
```
RandomForestClassifier(class_weight={0: 0.966977985323549,
                                     1: 1.0353571428571429},
                       max_depth=5, n_jobs=-1, random_state=42)
```

In [22]:
```
# define Boruta feature selection method
feat_selector = BorutaPy(forest, n_estimators='auto', verbose=2, random_state=0)

# find all relevant features
# takes input in array format not as dataframe
X_ary = np.array(X)
y_ary = np.array(y)
feat_selector.fit(X_ary, y_ary)

# check selected features
print(feat_selector.support_)

# check ranking of features
print(feat_selector.ranking_)

# call transform() on X to filter it down to selected features
X_filtered = feat_selector.transform(X_ary)
```

executed in 1m 5.21s, finished 20:08:56 2023-01-23

```
Iteration:      1 / 100
Confirmed:      0
Tentative:      270
Rejected:       0
Iteration:      2 / 100
Confirmed:      0
Tentative:      270
Rejected:       0
Iteration:      3 / 100
Confirmed:      0
Tentative:      270
Rejected:       0
Iteration:      4 / 100
Confirmed:      0
Tentative:      270
Rejected:       0
Iteration:      5 / 100
Confirmed:      0
Tentative:      270
```

In [23]:
```
# zip my names, ranks, and decisions in a single iterable
feature_names = X.columns
feature_ranks = list(zip(feature_names,
                         feat_selector.ranking_,
                         feat_selector.support_))

# iterate through and print out the results
for feat in feature_ranks:
    print(f'Feature: {feat[0]:<30} Rank: {feat[1]:<5} Keep: {feat[2]}')
```

executed in 12ms, finished 20:08:56 2023-01-23

```
Feature: Volume                        Rank: 3     Keep: False
Feature: ABER_ZG_5_15                  Rank: 108   Keep: False
Feature: ABER_SG_5_15                  Rank: 163   Keep: False
Feature: ABER_XG_5_15                  Rank: 97    Keep: False
Feature: ABER_ATR_5_15                 Rank: 49    Keep: False
Feature: ACCBL_20                      Rank: 114   Keep: False
Feature: ACCBM_20                      Rank: 104   Keep: False
Feature: ACCBU_20                      Rank: 129   Keep: False
Feature: AD                            Rank: 89    Keep: False
Feature: ADOSC_3_10                    Rank: 1     Keep: True
Feature: ADX_14                        Rank: 65    Keep: False
Feature: DMP_14                        Rank: 26    Keep: False
Feature: DMN_14                        Rank: 48    Keep: False
Feature: ALMA_10_6.0_0.85              Rank: 134   Keep: False
Feature: AMATe_LR_8_21_2               Rank: 12    Keep: False
Feature: AMATe_SR_8_21_2               Rank: 7     Keep: False
Feature: AO_5_34                       Rank: 17    Keep: False
Feature: OBV                           Rank: 82    Keep: False
Feature: OBV_min_2                     Rank: 86    Keep: False
```

In [24]:
```
selected_rf_features = pd.DataFrame({'Feature':feature_names,
                                     'Ranking':feat_selector.ranking_})

# selected_rf_features#.sort_values(by='Ranking')

selected_rf_features[selected_rf_features['Ranking']==1]
```

executed in 16ms, finished 20:08:56 2023-01-23

Out[24]:

	Feature	Ranking
9	ADOSC_3_10	1
22	AOBV_LR_2	1
23	AOBV_SR_2	1
25	AROOND_14	1
26	AROONU_14	1
32	BBB_5_2.0	1
33	BBP_5_2.0	1
34	BIAS_SMA_26	1
35	BOP	1
38	CCI_14_0.015	1
48	CDL_BELTHOLD	1

In [25]:
```
selected = list(selected_rf_features[selected_rf_features['Ranking']==1]['Feature'])
```

executed in 3ms, finished 20:08:56 2023-01-23

In [26]:
```
X_selected = stock_data[selected]
```

executed in 4ms, finished 20:08:56 2023-01-23

In [27]:
```
X_selected.shape # The number of feature is 64
```

executed in 6ms, finished 20:08:56 2023-01-23

Out[27]:
```
(2899, 64)
```

3.2 K-Means clustering

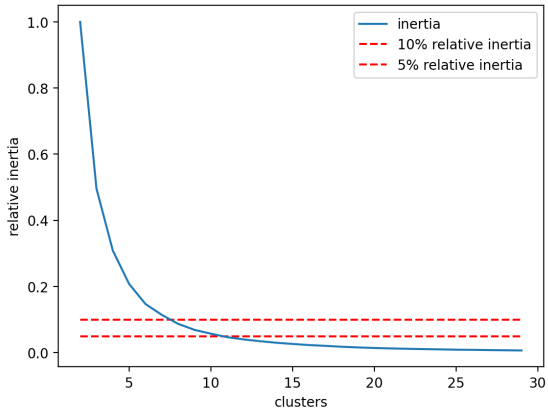
In [28]:
```
X = X_selected
```

executed in 3ms, finished 20:08:56 2023-01-23

3.2.1 Elbow Plot

In [29]:
```
n_clusters = range(2, 30)
inertia = []

for n in n_clusters:
    kmeans = KMeans(n_clusters=n)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

plt.plot(n_clusters, np.divide(inertia,inertia[0]))
plt.hlines(0.1, n_clusters[0], n_clusters[-1], 'r', linestyles='dashed')
plt.hlines(0.05, n_clusters[0], n_clusters[-1], 'r', linestyles='dashed')
plt.xlabel('clusters')
plt.ylabel('relative inertia')
plt.legend(['inertia', '10% relative inertia', '5% relative inertia']);
```

executed in 8.31s, finished 20:09:05 2023-01-23



3.2.2 Silhouette Coefficient

In [30]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```
executed in 2ms, finished 20:09:05 2023-01-23

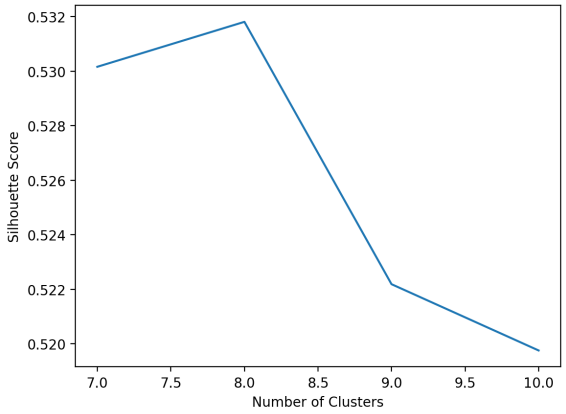
In [31]:
```
set_seeds(2023)
Silhouette_list = []
for i in [7, 8, 9, 10]:
    KMean_test= KMeans(n_clusters=i)
    KMean_test.fit(X)
    label_test=KMean_test.predict(X)
    score = silhouette_score(X, label_test)
    Silhouette_list.append(score)
#     print(f'Silhouette Score: {silhouette_score(Xlabel_test_test)}')
```
executed in 5.12s, finished 20:09:10 2023-01-23

In [32]:
```
Silhouette_list
```
executed in 4ms, finished 20:09:10 2023-01-23

Out[32]:
```
[0.5301639574706409, 0.5318122369209507, 0.522190364323992, 0.5197598976215557]
```

In [33]:
```
Silhouette_label = [7, 8, 9, 10]
plt.plot(Silhouette_label , Silhouette_list)
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
```
executed in 222ms, finished 20:09:10 2023-01-23

Out[33]:
```
Text(0, 0.5, 'Silhouette Score')
```



In [34]:
```
KMean_test= KMeans(n_clusters=7)
KMean_test.fit(X)
label_test=KMean_test.predict(X)
```
executed in 210ms, finished 20:09:11 2023-01-23

In [35]:
```
print(f'Silhouette Score(n=7): {silhouette_score(X, label_test)}')
```
executed in 138ms, finished 20:09:11 2023-01-23

```
Silhouette Score(n=7): 0.5310282741634225
```

In [36]:
```
# Normalize and fit the model
pipe = Pipeline([("normalization", MinMaxScaler()), ("cluster", KMeans(n_clusters=7))])
```
executed in 3ms, finished 20:09:11 2023-01-23

In [37]:
```
# Fit Model
pipe.fit(X.T)

# Assign Label
labels = pipe.predict(X.T)
```
executed in 111ms, finished 20:09:11 2023-01-23

In [38]:
```
Features = X.columns
# Create dataframe to hold cluster details
df = pd.DataFrame({'Cluster': labels,
                   'Features': Features,
                   'X_mean': X.mean(axis=0)}
                  ).sort_values(by=['Cluster'], axis = 0)

df = df.reset_index(drop=True)
df

# 'ATR': weekly_atr.mean(axis=1)
```
executed in 17ms, finished 20:09:11 2023-01-23

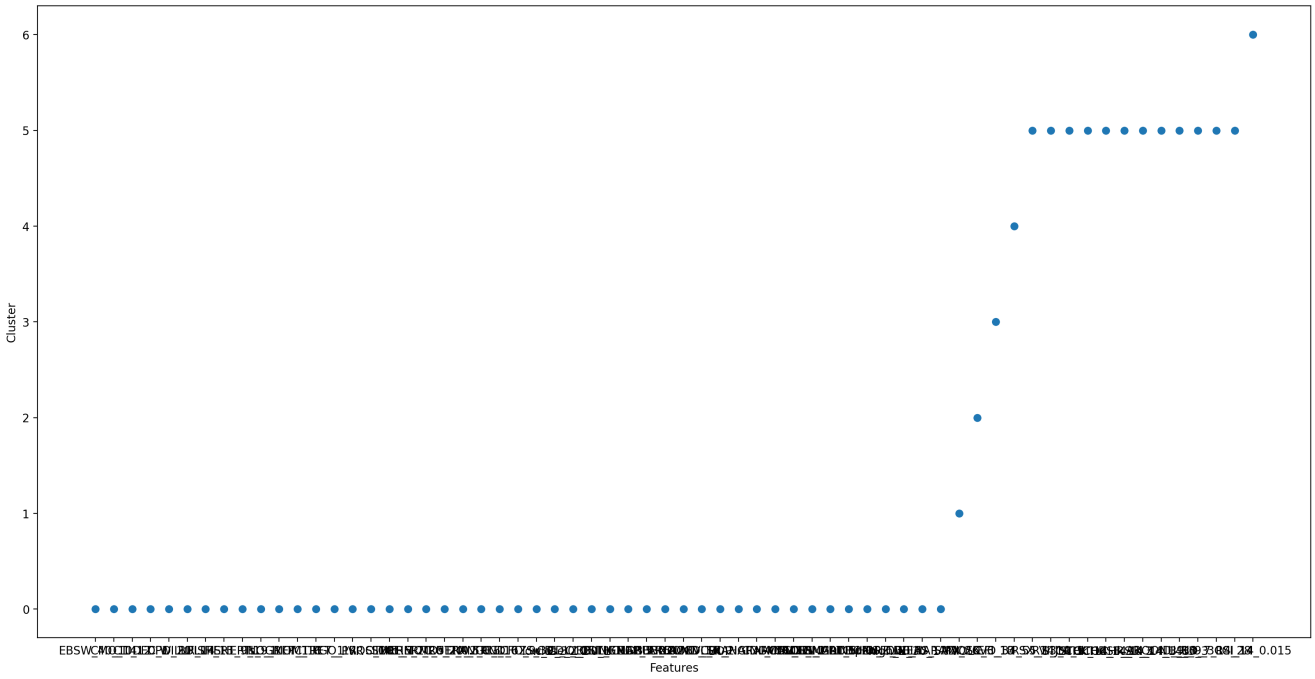
Out[38]:

	Cluster	Features	X_mean
0	0	EBSW_40_10	2.145788e-01
1	0	CMO_14	1.037855e+01
2	0	CTI_12	2.068876e-01
3	0	DEC_1	4.522249e-01
4	0	DPO_20	-1.420945e-01
5	0	WILLR_14	-3.583892e+01
6	0	BULLP_13	1.895417e+00
7	0	FISHERT_9_1	7.843556e-01
8	0	FISHERTs_9_1	7.832866e-01
9	0	INC_1	5.443256e-01
10	0	LOGRET_1	3.762303e-04

In [39]:
```
plt.figure(figsize=(20,10))
plt.scatter(df.Features, df.Cluster)
plt.xlabel('Features')
plt.ylabel('Cluster')
# plt.title('SPY Features Clustering');
```

executed in 1.37s, finished 20:09:13 2023-01-23

Out[39]:
```
Text(0, 0.5, 'Cluster')
```



In [40]:
```
pipe.get_params()
```

executed in 7ms, finished 20:09:13 2023-01-23

Out[40]:
```
{'memory': None,
 'steps': [('normalization', MinMaxScaler()),
  ('cluster', KMeans(n_clusters=7))],
 'verbose': False,
 'normalization': MinMaxScaler(),
 'cluster': KMeans(n_clusters=7),
 'normalization__clip': False,
 'normalization__copy': True,
 'normalization__feature_range': (0, 1),
 'cluster__algorithm': 'auto',
 'cluster__copy_x': True,
 'cluster__init': 'k-means++',
 'cluster__max_iter': 300,
 'cluster__n_clusters': 7,
 'cluster__n_init': 10,
 'cluster__n_jobs': 'deprecated',
 'cluster__precompute_distances': 'deprecated',
 'cluster__random_state': None,
 'cluster__tol': 0.0001,
 'cluster__verbose': 0}
```

In [41]:
```
pipe['cluster'].cluster_centers_
```

executed in 5ms, finished 20:09:13 2023-01-23

Out[41]:
```
array([[3.78908751e-04, 1.22604226e-03, 2.82096284e-03, ...,
        2.70220940e-09, 1.18494592e-09, 5.50018479e-10],
       [1.00000000e+00, 1.00000000e+00, 1.00000000e+00, ...,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
       [5.22050832e-03, 3.12714886e-03, 3.72962350e-03, ...,
        8.14431544e-04, 1.70813081e-03, 2.11729830e-03],
       ...,
       [0.00000000e+00, 8.32559799e-04, 2.26207208e-03, ...,
        1.53280712e-04, 2.01304017e-04, 2.08037790e-04],
       [3.78911025e-04, 1.22604386e-03, 2.82096424e-03, ...,
        4.68488946e-09, 4.55246113e-09, 3.26746973e-09],
       [3.78908452e-04, 1.22603983e-03, 2.82095370e-03, ...,
        4.54573273e-09, 3.64138086e-09, 3.27374184e-09]])
```

In [42]:
```
pipe['cluster'].cluster_centers_.shape
```

executed in 4ms, finished 20:09:13 2023-01-23

Out[42]:
```
(7, 2899)
```

3.3 SOM

In [43]:
```
X = df[['X_mean']]
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

executed in 6ms, finished 20:09:13 2023-01-23

$$Q = 5 \times \sqrt{I}$$

In [44]:

```
# Initialize a 15x15 SOM - Thumb rule is 5 * sqrt(n) > 5x5 SOM
som = MiniSom(40, 40, 1, learning_rate=0.5, sigma=5, random_seed=2023)

# Initialize with randon weights
som.random_weights_init(X)

# Trains SOM with 10000 iterations
som.train_batch(X,10000, verbose=True)
```

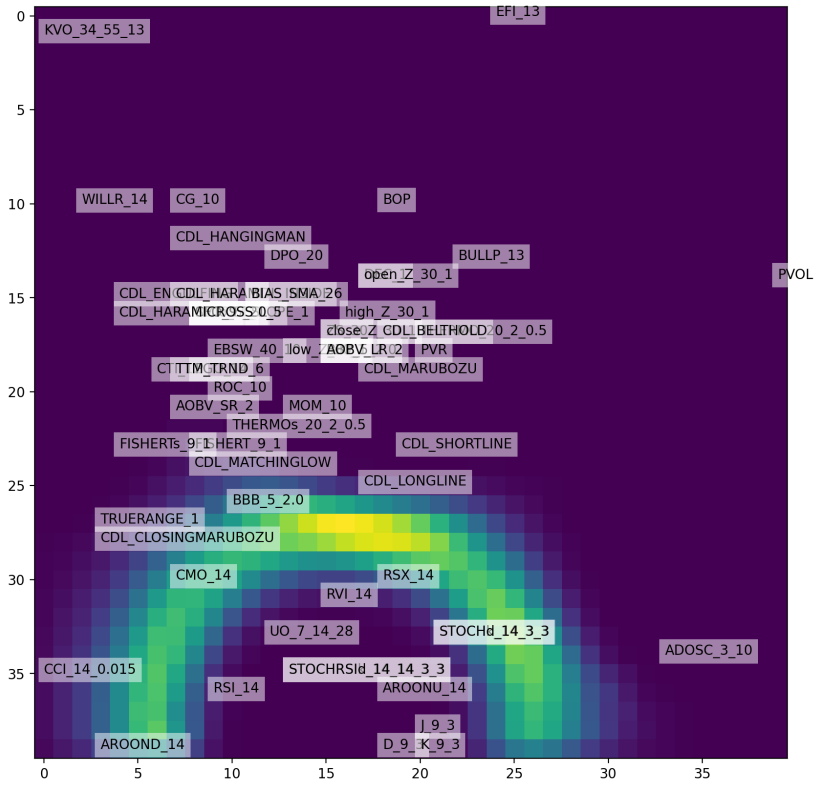
executed in 2.69s, finished 20:09:16 2023-01-23

```
[    346 / 10000 ]    3% - 0:00:02 left

[    665 / 10000 ]    7% - 0:00:02 left

[ 10000 / 10000 ] 100% - 0:00:00 left
quantization error: 7.598641169807632e-12
```

In [109]:

```
# Plot SOM
plt.figure(figsize=(20, 10))
for ix in range(len(X)):
    winner = som.winner(X[ix])
    plt.text(winner[0], winner[1], df.Features[ix], bbox=dict(facecolor='white', alpha=0.5, lw=0))
plt.imshow(som.distance_map())
plt.grid(False)
# plt.title('Self Organizing Maps');
```

executed in 1.32s, finished 21:17:37 2023-01-23



In [46]:
```
corresponding_listnding_list = []
# List the corresponding companies
for ix in range(len(X)):
    winner = som.winner(X[ix])
#     corresponding_list.append(winner)
    print(winner[0], winner[1], df['Features'][ix])
```

executed in 18ms, finished 20:09:18 2023-01-23

```
9 18 EBSW_40_10
7 30 CMO_14
6 19 CTI_12
17 14 DEC_1
12 13 DPO_20
2 10 WILLR_14
22 13 BULLP_13
8 23 FISHERT_9_1
4 23 FISHERTs_9_1
15 18 INC_1
8 16 LOGRET_1
13 21 MOM_10
8 16 PCTRET_1
8 19 PGO_14
20 18 PVR
9 20 ROC_10
11 16 SLOPE_1
8 16 SMIo_5_20_5
20 17 THERMO_20_2_0.5
10 22 THERMOs_20_2_0.5
3 27 TRUERANGE_1
7 19 TTM_TRND_6
7 10 CG_10
8 16 CFO_9
15 17 ZS_30
13 18 low_Z_30_1
15 17 close_Z_30_1
18 17 CDL_BELTHOLD
3 28 CDL_CLOSINGMARUBOZU
4 15 CDL_ENGULFING
15 18 BBP_5_2.0
10 26 BBB_5_2.0
3 39 AROOND_14
7 21 AOBV_SR_2
18 10 BOP
7 12 CDL_HANGINGMAN
4 16 CDL_HARAMICROSS
11 15 CDL_INSIDE
15 18 AOBV_LR_2
17 25 CDL_LONGLINE
17 19 CDL_MARUBOZU
8 24 CDL_MATCHINGLOW
19 23 CDL_SHORTLINE
17 14 open_Z_30_1
16 16 high_Z_30_1
7 15 CDL_HARAMI
11 15 BIAS_SMA_26
39 14 PVOL
33 34 ADOSC_3_10
24 0 EFI_13
0 1 KVO_34_55_13
18 30 RSX_14
15 31 RVI_14
20 38 J_9_3
21 33 STOCHk_14_3_3
21 33 STOCHd_14_3_3
13 35 STOCHRSIk_14_14_3_3
13 35 STOCHRSId_14_14_3_3
18 36 AROONU_14
18 39 D_9_3
20 39 K_9_3
12 33 UO_7_14_28
9 36 RSI_14
0 35 CCI_14_0.015
```

```
['BEARP_13', 'EBSW_40_10', 'BULLP_13', 'WILLR_14', 'PVR', 'TRUERANGE_1', 'CMO_14', 'AROOND_14', 'BBB_5_2.0', 'PVOL', 'EFI_13', 'Volume', 'KVO_34_55_13',
'AROONU_14', 'CCI_14_0.015']
```

```
['EBSW_40_10', 'CMO_14', 'DPO_20', 'WILLR_14', 'BULLP_13', 'PVR', 'TRUERANGE_1', 'ADOSC_3_10', 'EFI_13', 'KVO_34_55_13']
```

```
['KVO_34_55_13', 'CCI_14_0.015', 'PVOL', 'EFI_13', 'ADOSC_3_10', 'D_9_3', 'BOP']
```

In [47]:
```
# selected2 = ['KVO_34_55_13', 'CCI_14_0.015', 'PVOL', 'EFI_13', 'ADOSC_3_10', 'D_9_3', 'BOP']
```

executed in 2ms, finished 20:09:18 2023-01-23

In [48]:
```
selected2 = ['KVO_34_55_13', 'EBSW_40_10', 'DPO_20',
             'WILLR_14', 'BULLP_13', 'PVR',
             'TRUERANGE_1', 'EFI_13', 'ADOSC_3_10', 'BOP']
```

executed in 3ms, finished 20:09:18 2023-01-23

In [49]:
```
# selected2 = ['EBSW_40_10', 'CMO_14', 'DPO_20',
#              'WILLR_14', 'BULLP_13', 'PVR',
#              'TRUERANGE_1', 'ADOSC_3_10',
#              'EFI_13', 'KVO_34_55_13']
```

executed in 2ms, finished 20:09:18 2023-01-23

https://github.com/JustGlowing/minisom/blob/master/examples/FeatureSelection.ipynb (https://github.com/JustGlowing/minisom/blob/master/examples/FeatureSelection.ipynb)

4 Exploratory Data Analysis

In [50]:
```
X_selected = X_selected[selected2]
```

executed in 3ms, finished 20:09:18 2023-01-23

In [51]:
```
X_selected.shape
```

executed in 5ms, finished 20:09:18 2023-01-23

Out[51]:
```
(2899, 10)
```

4.1 Features Describe

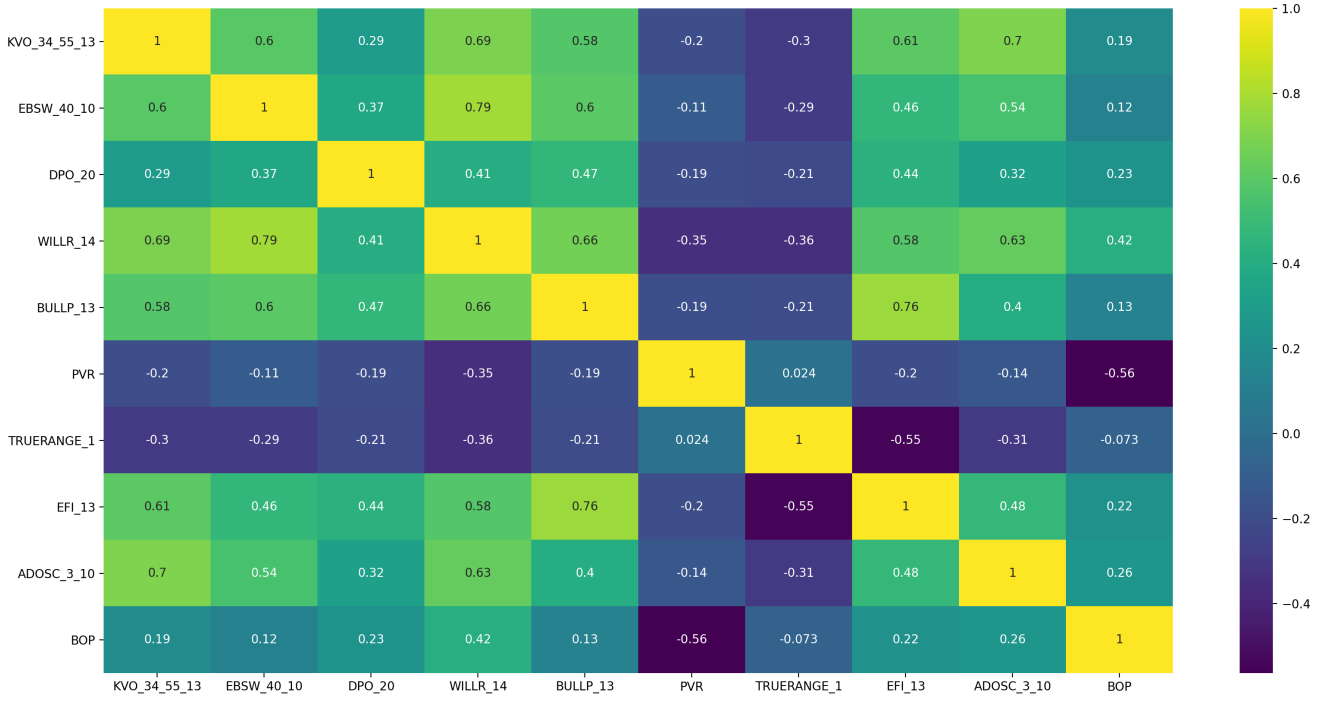
In [52]:
```
X_selected.describe().T
```
executed in 40ms, finished 20:09:18 2023-01-23

Out[52]:

	count	mean	std	min	25%	50%	75%	max
KVO_34_55_13	2899.0	-3.152470e+04	7.442190e+06	-5.814442e+07	-4.040862e+06	4.774819e+05	4.347242e+06	2.995325e+07
EBSW_40_10	2899.0	2.145788e-01	8.124232e-01	-9.999737e-01	-7.566179e-01	6.575463e-01	9.683719e-01	9.999898e-01
DPO_20	2899.0	-1.420945e-01	3.917509e+00	-2.789100e+01	-1.595507e+00	3.150635e-02	1.299499e+00	2.668348e+01
WILLR_14	2899.0	-3.583892e+01	3.061310e+01	-1.000000e+02	-5.989283e+01	-2.692308e+01	-8.134081e+00	-0.000000e+00
BULLP_13	2899.0	1.895417e+00	3.997989e+00	-2.681032e+01	3.935218e-01	1.819845e+00	3.505296e+00	2.179554e+01
PVR	2899.0	2.410486e+00	1.016686e+00	1.000000e+00	2.000000e+00	2.000000e+00	3.000000e+00	4.000000e+00
TRUERANGE_1	2899.0	3.254581e+00	3.039621e+00	4.400024e-01	1.389999e+00	2.190002e+00	3.914993e+00	3.196001e+01
EFI_13	2899.0	-2.309790e+07	1.559325e+08	-2.469719e+09	-4.078148e+07	7.807426e+06	3.476777e+07	4.330140e+08
ADOSC_3_10	2899.0	2.980900e+07	7.917299e+07	-4.047211e+08	-1.544632e+07	3.527280e+07	7.614915e+07	3.343292e+08
BOP	2899.0	5.717779e-02	5.265266e-01	-9.838320e-01	-3.874843e-01	7.758349e-02	5.138909e-01	1.000000e+00

4.2 Correlation Matrix

In [53]:
```
# first, let's compute the corr. matrix among features
corrFeatMatrix = X_selected.corr()

#Plotting of features correlation matrix
plt.figure(figsize=(20,10))
sns.heatmap(corrFeatMatrix, cmap="viridis",annot=True)
# plt.title(
#     'Correlation Matrix among features',
#     fontsize=16, y=1.05, weight='bold'
# )
plt.show()
```
executed in 888ms, finished 20:09:19 2023-01-23



In [54]:
```
# setting correlation maximum threshold
rho_max = 0.80

# detecting pairwise features with abs(corr) > rho_max
idxPairwise = np.where(abs(corrFeatMatrix) >= rho_max)
lstPairwise = [
    [corrFeatMatrix.index[x], corrFeatMatrix.columns[y], round(corrFeatMatrix.iloc[x, y], 3)]
    for x, y in zip(*idxPairwise) if x != y and x < y
]
print("::>> Pairwise features with ρ_max= +- 0.80 :")
print("::::>>>> List/list represents [Feat1, Feat2, Corr]:")
lstPairwise
```
executed in 7ms, finished 20:09:19 2023-01-23

```
::>> Pairwise features with ρ_max= +- 0.80 :
::::>>>> List/list represents [Feat1, Feat2, Corr]:
```

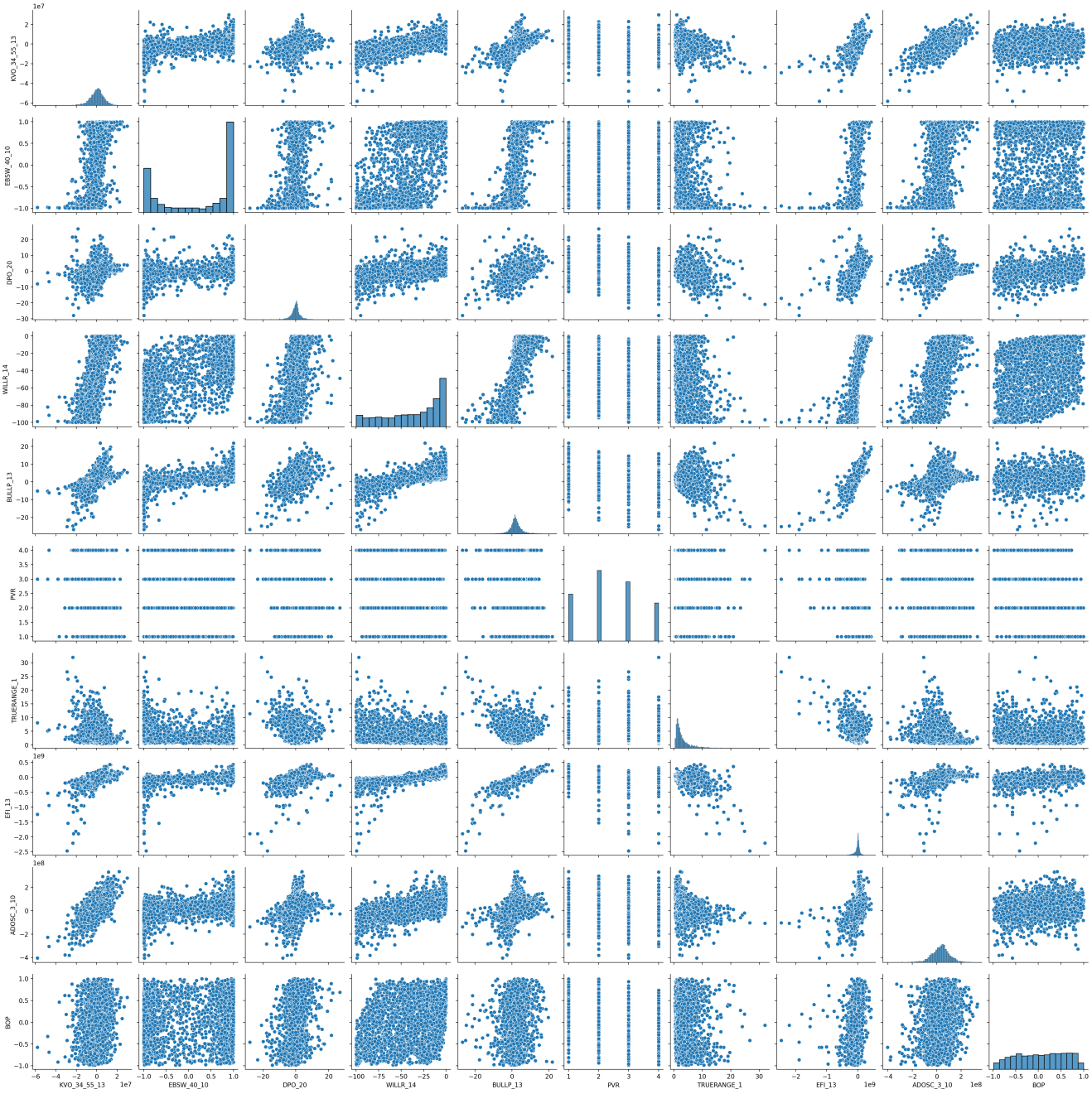
Out[54]: []

4.3 Pair Plot

In [55]:
```
1 sns.pairplot(X_selected)
```

executed in 15.1s, finished 20:09:34 2023-01-23

Out[55]: <seaborn.axisgrid.PairGrid at 0x7fd9742aba00>



4.4 XGBoosting

In [56]:
```
import shap
def XGBoosting_Filter(X_selected, X_train_f, y_train_f, X_test, y_test):
    # Scale and fit the classifier model
    xgbcls =  XGBClassifier(verbosity = 0, silent=True, random_state=101)
    xgbcls.fit(X_train_f, y_train_f)
    # Cross-validation
    tscv = TimeSeriesSplit(n_splits=5, gap=1)
    # Hyper parameter optimization
    param_grid = {'learning_rate': [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
                  'max_depth': [3, 4, 5, 6, 8, 10, 12, 15],
                  'min_child_weight': [1, 3, 5, 7],
                  'gamma': [0.0, 0.1, 0.2 , 0.3, 0.4],
                  'colsample_bytree': [0.3, 0.4, 0.5 , 0.7]}
    # perform random search
    svm_rdm = RandomizedSearchCV(xgbcls, param_grid, n_iter=100, scoring='f1', cv=tscv, verbose=0)
    svm_rdm.fit(X_train_f, y_train_f, verbose=0)

    # print best parameter after tuning
    print("-----------------------------------------------------------")
    print(svm_rdm.best_params_)
    # print best score after tuning
    print("-----------------------------------------------------------")
    print(svm_rdm.best_score_)
    print("-----------------------------------------------------------")

    # Refit the XGB Classifier with the best params
    cls_best = XGBClassifier(**svm_rdm.best_params_)

    cls_best.fit(X_train_f, y_train_f,
            eval_set=[(X_train_f, y_train_f), (X_test, y_test)],
            eval_metric='logloss',
            verbose=True)
    # Plot feature importance
    # feature importance_type = 'gain'

    fig, ax = plt.subplots(figsize=(8, 6))
    feature_imp = pd.DataFrame({'Importance Score': cls_best.feature_importances_,
                                'Features': X_selected.columns}).sort_values(by='Importance Score', ascending=False)

    sns.barplot(x=feature_imp['Importance Score'], y=feature_imp['Features'])
    ax.set_title('Features Importance')
    plt.show()

    plot_importance(cls_best, importance_type='gain', show_values=False)
    plt.show()

    # SHAP
    explainer = shap.TreeExplainer(cls_best)
    shap_values = explainer.shap_values(X_test)
    # future importance summary
    shap.summary_plot(shap_values, X_test, plot_type="bar")
    # interpretation plot
    shap.summary_plot(shap_values, X_test)
```

executed in 2.77s, finished 20:09:37 2023-01-23

4.5 Spliting

In [57]:
```
# Always keep shuffle = False for financial time series
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.3, shuffle=False)

# convert to array
X_train, X_test, y_train, y_test = np.array(X_train), np.array(X_test), np.array(y_train), np.array(y_test)

# Output the train and test data size
print(f"Train and Test Size {len(X_train)}, {len(X_test)}")
```

executed in 8ms, finished 20:09:37 2023-01-23

```
Train and Test Size 2029, 870
```

In [58]:
```
# We scale the data so hasten the LSTM's conversion
scaler = MinMaxScaler()

# Shuffle is set to False because it is a timeseries, and the order matters.
X_train, X_test, y_train, y_test = train_test_split(X_selected, y,
                                                    test_size = 0.30,
                                                    random_state=0, shuffle = False)

# Scaling
X_scaled_train = scaler.fit_transform(X_train)
X_scaled_test = scaler.transform(X_test)
```

executed in 10ms, finished 20:09:37 2023-01-23

In [59]:
```
set_seeds(2023)
X_scaled_train_df = pd.DataFrame(X_scaled_train)
X_scaled_test_df = pd.DataFrame(X_scaled_test)
XGBoosting_Filter(X_selected, X_scaled_train_df, y_train,
                  X_scaled_test_df, y_test)
```

executed in 32.8s, finished 20:10:10 2023-01-23

```
-----------------------------------------------------------
{'min_child_weight': 7, 'max_depth': 3, 'learning_rate': 0.05, 'gamma': 0.0, 'colsample_bytree': 0.7}
-----------------------------------------------------------
0.9203230102808355
-----------------------------------------------------------
[0]     validation_0-logloss:0.65658    validation_1-logloss:0.65313
[1]     validation_0-logloss:0.62371    validation_1-logloss:0.61790
[2]     validation_0-logloss:0.59356    validation_1-logloss:0.58371
[3]     validation_0-logloss:0.57690    validation_1-logloss:0.56552
[4]     validation_0-logloss:0.55099    validation_1-logloss:0.53792
[5]     validation_0-logloss:0.53706    validation_1-logloss:0.52351
[6]     validation_0-logloss:0.51411    validation_1-logloss:0.49722
[7]     validation_0-logloss:0.50220    validation_1-logloss:0.48437
[8]     validation_0-logloss:0.49523    validation_1-logloss:0.47644
[9]     validation_0-logloss:0.47544    validation_1-logloss:0.45377
[10]    validation_0-logloss:0.45714    validation_1-logloss:0.43338
[11]    validation_0-logloss:0.44022    validation_1-logloss:0.41383
[12]    validation_0-logloss:0.43756    validation_1-logloss:0.41082
[13]    validation_0-logloss:0.42204    validation_1-logloss:0.39330
```

In [ ]:
```
1
```

5 LSTM Model

In [60]:
```
# sequence length
seqlen = 60

# number of features
numfeat = X_scaled_train.shape[1]
```
executed in 2ms, finished 20:10:11 2023-01-23

In [61]:
```
g = TimeseriesGenerator(X_scaled_train, y_train.to_numpy(), length=seqlen)
g_ = TimeseriesGenerator(X_scaled_test, y_test.to_numpy(), length=seqlen)
```
executed in 5ms, finished 20:10:11 2023-01-23

5.1 Model 1 LSTM Model

In [62]:
```
# Create a model
def create_model(hu=256, lookback=60, features=numfeat):

    tensorflow.keras.backend.clear_session()

    # instantiate the model
    model = Sequential()
    model.add(LSTM(units=hu, input_shape=(lookback, features), activation = 'relu', return_sequences=False, name='LSTM'))
    model.add(Dense(units=1, name='Output'))              # can also specify linear activation function

    # specify optimizer separately (preferred method))
    opt = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-08)
#     opt = Adam(lr=0.001, epsilon=1e-08, decay=0.0)       # adam optimizer seems to perform better for a single lstm

    # model compilation
    model.compile(optimizer=opt, loss='mse', metrics=['mse'])

    return model
```
executed in 24ms, finished 20:10:11 2023-01-23

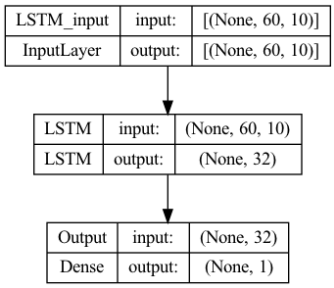
In [63]:
```
# lstm network
model_1 = create_model(hu=32, lookback=seqlen, features=numfeat)
```
executed in 105ms, finished 20:10:11 2023-01-23

In [64]:
```
model_1.summary()
```
executed in 11ms, finished 20:10:11 2023-01-23

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 LSTM (LSTM)                 (None, 32)                5504

 Output (Dense)              (None, 1)                 33

=================================================================
Total params: 5,537
Trainable params: 5,537
Non-trainable params: 0
_________________________________________________________________
```

In [65]:
```
plot_model(model_1, to_file='./img/model_1.png', show_shapes=True, show_layer_names=True)
```
executed in 443ms, finished 20:10:13 2023-01-23

Out[65]:



In [66]:
```
# Specify callback functions
model_path = (results_path / 'model_1.h5').as_posix()
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

my_callbacks = [
    EarlyStopping(patience=10, monitor='loss', mode='min', verbose=1, restore_best_weights=True),
    ModelCheckpoint(filepath=model_path, verbose=1, monitor='loss', save_best_only=True),
    TensorBoard(log_dir=logdir, histogram_freq=1)
]
```
executed in 4ms, finished 20:10:13 2023-01-23

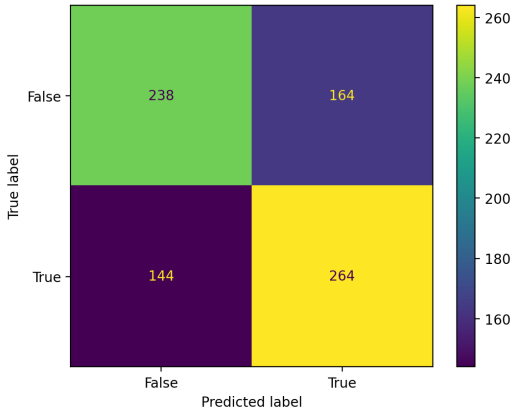
In [67]:
```
# Model fitting
history = model_1.fit(g,
          epochs=500,
          verbose=1,
          callbacks=my_callbacks,
          shuffle=False,
          class_weight=class_weight)
```
executed in 50.7s, finished 20:11:04 2023-01-23

```
Epoch 8: loss improved from 0.24796 to 0.24744, saving model to results/lstm_time_series/model_1.h5
16/16 [==============================] - 0s 17ms/step - loss: 0.2474 - mse: 0.2472
Epoch 9/500
13/16 [=====================>......] - ETA: 0s - loss: 0.2480 - mse: 0.2474
Epoch 9: loss improved from 0.24744 to 0.24692, saving model to results/lstm_time_series/model_1.h5
16/16 [==============================] - 0s 18ms/step - loss: 0.2469 - mse: 0.2467
Epoch 10/500
13/16 [=====================>......] - ETA: 0s - loss: 0.2476 - mse: 0.2470
Epoch 10: loss improved from 0.24692 to 0.24642, saving model to results/lstm_time_series/model_1.h5
16/16 [==============================] - 0s 17ms/step - loss: 0.2464 - mse: 0.2462
Epoch 11/500
13/16 [=====================>......] - ETA: 0s - loss: 0.2472 - mse: 0.2466
Epoch 11: loss improved from 0.24642 to 0.24591, saving model to results/lstm_time_series/model_1.h5
16/16 [==============================] - 0s 17ms/step - loss: 0.2459 - mse: 0.2457
Epoch 12/500
15/16 [===========================>..] - ETA: 0s - loss: 0.2457 - mse: 0.2455
Epoch 12: loss improved from 0.24591 to 0.24541, saving model to results/lstm_time_series/model_1.h5
16/16 [==============================] - 0s 21ms/step - loss: 0.2454 - mse: 0.2452
Epoch 13/500
13/16 [=====================>......] - ETA: 0s - loss: 0.2464 - mse: 0.2459
```

In [68]:
```
ypred_1 = np.where(model_1.predict(g_, verbose=False) > 0.5, 1, 0)
```
executed in 224ms, finished 20:11:04 2023-01-23

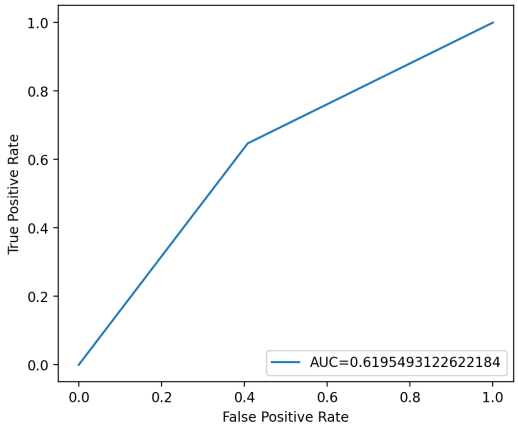
In [69]:
```
confusion_matrix = metrics.confusion_matrix(y_test[seqlen:], ypred_1)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
                                            display_labels = [False, True])

cm_display.plot()
plt.show()
```

executed in 239ms, finished 20:11:05 2023-01-23



In [70]:
```
fpr, tpr, _ = metrics.roc_curve(y_test[seqlen:],  ypred_1)
auc = metrics.roc_auc_score(y_test[seqlen:], ypred_1)

#create ROC curve
plt.figure(figsize=(6,5))
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

executed in 166ms, finished 20:11:05 2023-01-23



In [71]:
```
print(classification_report(y_test[seqlen:], ypred_1))
```

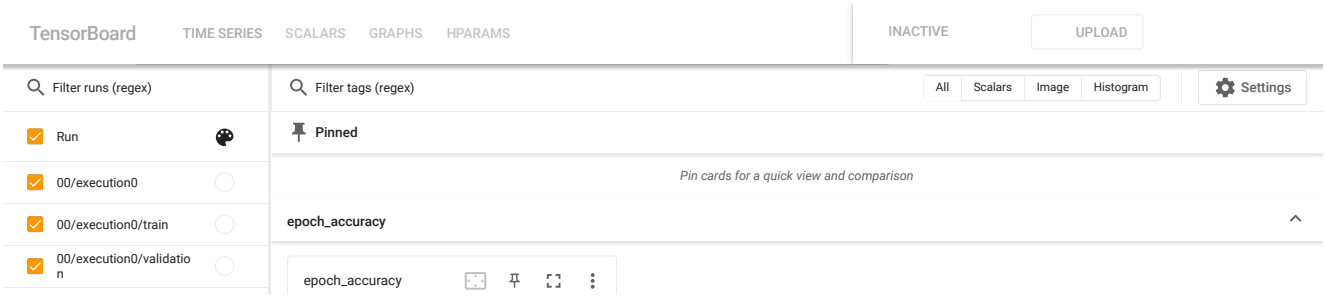
executed in 7ms, finished 20:11:05 2023-01-23

```
              precision    recall  f1-score   support

           0       0.62      0.59      0.61       402
           1       0.62      0.65      0.63       408

    accuracy                           0.62       810
   macro avg       0.62      0.62      0.62       810
weighted avg       0.62      0.62      0.62       810
```

In [72]:
```
%tensorboard --logdir ./tensorboard/bologs
```

executed in 10ms, finished 20:11:06 2023-01-23

```
Reusing TensorBoard on port 6006 (pid 4282), started 1 day, 23:03:04 ago. (Use '!kill 4282' to kill it.)
```



5.2 Model 2 Stacked-LSTM Model

gelu + relu + tanh

In [73]:
```python
# Create a model
def create_model(hu=256, lookback=60, features=numfeat):

    tensorflow.keras.backend.clear_session()

    # instantiate the model
    model = Sequential()
    model.add(LSTM(units=hu, input_shape=(lookback, features), activation = 'relu', return_sequences=True, name='LSTM1'))
    model.add(Dropout(0.4, name='Dropout1'))

    model.add(LSTM(units=hu, activation = 'gelu', return_sequences=True, name='LSTM2'))
    model.add(Dropout(0.4, name='Drouput2'))

    model.add(LSTM(units=hu, activation = 'selu', return_sequences=False, name='LSTM3'))
    model.add(Dense(units=1, activation='sigmoid', name='Output'))              # can also specify linear activation function

    # specify optimizer separately (preferred method))
    opt = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-08)
#     opt = Adam(learning_rate=0.001, epsilon=1e-08, decay=0.0)       # adam optimizer seems to perform better for a single lstm

    # model compilation
    model.compile(optimizer=opt, loss='mse', metrics=['mae'])

    return model
```

executed in 6ms, finished 20:11:06 2023-01-23

In [74]:
```python
# lstm network
model_2 = create_model(hu=32, lookback=seqlen, features=numfeat)
```

executed in 238ms, finished 20:11:07 2023-01-23

In [75]:
```python
model_2.summary()
```

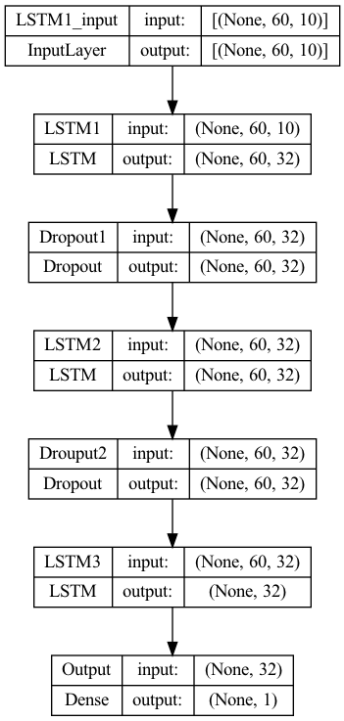
executed in 14ms, finished 20:11:07 2023-01-23

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 LSTM1 (LSTM)                (None, 60, 32)            5504

 Dropout1 (Dropout)          (None, 60, 32)            0

 LSTM2 (LSTM)                (None, 60, 32)            8320

 Drouput2 (Dropout)          (None, 60, 32)            0

 LSTM3 (LSTM)                (None, 32)                8320

 Output (Dense)              (None, 1)                 33

=================================================================
Total params: 22,177
Trainable params: 22,177
Non-trainable params: 0
_________________________________________________________________
```

In [76]:
```python
plot_model(model_2, to_file='./img/model_2.png', show_shapes=True, show_layer_names=True)
```

executed in 520ms, finished 20:11:08 2023-01-23

Out[76]:



In [77]:
```python
# Specify callback functions
model_path = (results_path / 'model_2.h5').as_posix()
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

my_callbacks = [
    EarlyStopping(patience=10, monitor='loss', mode='min', verbose=1, restore_best_weights=True),
    ModelCheckpoint(filepath=model_path, verbose=1, monitor='loss', save_best_only=True),
    TensorBoard(log_dir=logdir, histogram_freq=1)
]
```

executed in 6ms, finished 20:11:08 2023-01-23

In [78]:
```
# Model fitting
history = model_2.fit(g,
          epochs=500,
          verbose=1,
          callbacks=my_callbacks,
          shuffle=False,
          class_weight=class_weight)
```

executed in 2m 38s, finished 20:13:46 2023-01-23

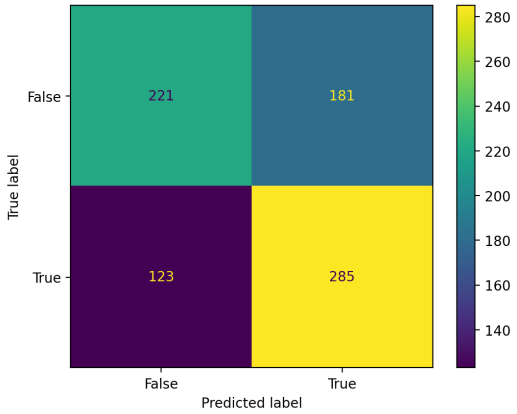
```
Epoch 1/500
15/16 [===========================>..] - ETA: 0s - loss: 0.2515 - mae: 0.5004
Epoch 1: loss improved from inf to 0.25146, saving model to results/lstm_time_series/model_2.h5
16/16 [==============================] - 4s 72ms/step - loss: 0.2515 - mae: 0.5004
Epoch 2/500
15/16 [===========================>..] - ETA: 0s - loss: 0.2502 - mae: 0.4997
Epoch 2: loss improved from 0.25146 to 0.25012, saving model to results/lstm_time_series/model_2.h5
16/16 [==============================] - 1s 75ms/step - loss: 0.2501 - mae: 0.4997
Epoch 3/500
15/16 [===========================>..] - ETA: 0s - loss: 0.2501 - mae: 0.4998
Epoch 3: loss improved from 0.25012 to 0.25006, saving model to results/lstm_time_series/model_2.h5
16/16 [==============================] - 1s 70ms/step - loss: 0.2501 - mae: 0.4998
Epoch 4/500
15/16 [===========================>..] - ETA: 0s - loss: 0.2504 - mae: 0.4999
Epoch 4: loss did not improve from 0.25006
16/16 [==============================] - 1s 66ms/step - loss: 0.2503 - mae: 0.4998
Epoch 5/500
15/16 [===========================>..] - ETA: 0s - loss: 0.2503 - mae: 0.4998
Epoch 5: loss did not improve from 0.25006
```

In [79]:
```
ypred_2 = np.where(model_2.predict(g_, verbose=False) > 0.5, 1, 0)
```

executed in 536ms, finished 20:13:47 2023-01-23

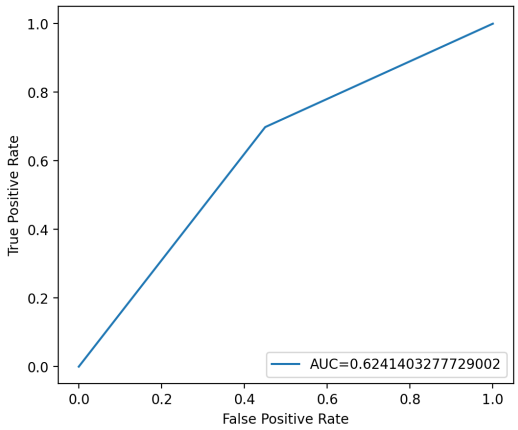
In [80]:
```
confusion_matrix = metrics.confusion_matrix(y_test[seqlen:], ypred_2)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
                                            display_labels = [False, True])

cm_display.plot()
plt.show()
```

executed in 228ms, finished 20:13:48 2023-01-23



In [81]:
```
fpr, tpr, _ = metrics.roc_curve(y_test[seqlen:],  ypred_2)
auc = metrics.roc_auc_score(y_test[seqlen:], ypred_2)

#create ROC curve
plt.figure(figsize=(6,5))
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

executed in 162ms, finished 20:13:48 2023-01-23



In [82]:
```
print(classification_report(y_test[seqlen:], ypred_2))
```

executed in 9ms, finished 20:13:48 2023-01-23

```
              precision    recall  f1-score   support

           0       0.64      0.55      0.59       402
           1       0.61      0.70      0.65       408

    accuracy                           0.62       810
   macro avg       0.63      0.62      0.62       810
weighted avg       0.63      0.62      0.62       810
```

6 Hyperparameter Optimization

In [83]:
```
def build_model(hp):

    tf.keras.backend.clear_session()

    # instantiate the model
    model = Sequential()

    # Tune the number of units in the layers
    hp_units1 = hp.Int('units1', min_value=4, max_value=32, step=4)
    hp_units2 = hp.Int('units2', min_value=4, max_value=32, step=4)
    hp_units3 = hp.Int('units3', min_value=4, max_value=32, step=4)

    # Tune the dropout rate
    hp_dropout1 = hp.Float('Dropout_rate', min_value=0, max_value=0.5, step=0.1)
    hp_dropout2 = hp.Float('Dropout_rate', min_value=0, max_value=0.5, step=0.1)

    # Tune the learning rate for the optimizer
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    # Tune activation functions
    hp_activation1 = hp.Choice(name = 'activation', values = ['relu', 'gelu', 'elu'], ordered = False)
    hp_activation2 = hp.Choice(name = 'activation', values = ['relu', 'gelu', 'elu'], ordered = False)
    hp_activation3 = hp.Choice(name = 'activation', values = ['relu', 'tanh', 'elu'], ordered = False)

    model.add(LSTM(hp_units1, input_shape=(seqlen, numfeat), activation=hp_activation1, return_sequences=True, name='LSTM1'))
    model.add(Dropout(hp_dropout1, name='Drouput1'))

    model.add(LSTM(hp_units2, activation = hp_activation2, return_sequences=True, name='LSTM2'))
    model.add(Dropout(hp_dropout2, name='Drouput2'))

    model.add(LSTM(hp_units3, activation = hp_activation3, return_sequences=False, name='LSTM3'))

    model.add(Dense(units=1, activation='sigmoid', name='Output'))

    # specify optimizer separately (preferred method))
    opt = Adam(learning_rate=hp_learning_rate, epsilon=1e-08)

    # model compilation - 'binary_crossentropy' - 'accuracy' - BinaryAccuracy(name='accuracy', threshold=0.5)
    model.compile(optimizer=opt,
                  loss=BinaryCrossentropy(),
                  metrics=['accuracy',
                           Precision(),
                           Recall()])

    return model
```

executed in 7ms, finished 20:13:49 2023-01-23

In [84]:
```
# initialize an early stopping callback to prevent the model from
# overfitting/spending too much time training with minimal gains
callback1 = [EarlyStopping(patience=5, monitor='loss', mode='min', verbose=1, restore_best_weights=True),
             TensorBoard(log_dir="./tensorboard/rslogs")]

callback2 = [EarlyStopping(patience=5, monitor='loss', mode='min', verbose=1, restore_best_weights=True),
             TensorBoard(log_dir="./tensorboard/hblogs")]

callback3 = [EarlyStopping(patience=5, monitor='loss', mode='min', verbose=1, restore_best_weights=True),
             TensorBoard(log_dir="./tensorboard/bologs")]
```

executed in 4ms, finished 20:13:49 2023-01-23

6.1 Approach 1 : Random Search [...]

6.2 Approach 2 : HyperBand [...]

6.3 Approach 3 : BayesianOptimization

In [85]:
```
# BayesianOptimization algorithm from keras tuner
botuner = BayesianOptimization(
    build_model,
    objective="val_accuracy",
    max_trials=15,
    num_initial_points=2,
    hyperparameters=None,
    tune_new_entries=True,
    allow_new_entries=True,
    overwrite=True,
    directory="./keras",
    project_name="botrial")
```

executed in 221ms, finished 20:13:49 2023-01-23

In [86]:
```
# launch tuning process
botuner.search(g, epochs=50, validation_data=g_, callbacks=callback1, class_weight = class_weight, shuffle=False)
```

executed in 9m 31s, finished 20:23:21 2023-01-23

```
Trial 15 Complete [00h 00m 56s]
val_accuracy: 0.644444465637207

Best val_accuracy So Far: 0.6567901372909546
Total elapsed time: 00h 09m 31s
INFO:tensorflow:Oracle triggered exit
```

In [87]:
```
# display the best hyperparameter values for the model based on the defined objective function
best_bohp = botuner.get_best_hyperparameters()[0]
print(best_bohp.values)
```

executed in 4ms, finished 20:23:21 2023-01-23

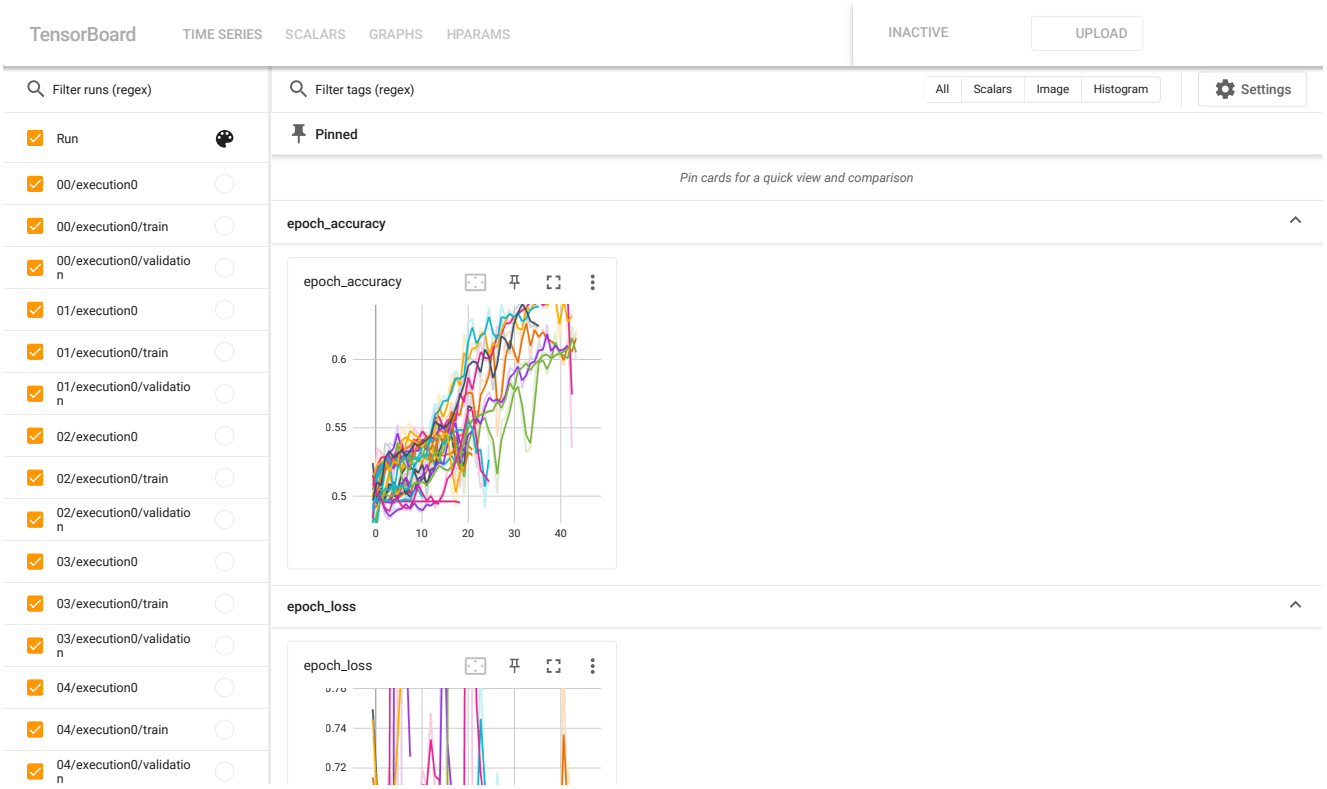
```
{'units1': 32, 'units2': 32, 'units3': 32, 'Dropout_rate': 0.2, 'learning_rate': 0.001, 'activation': 'elu'}
```

In [88]:
```
# display tuning results summary
botuner.results_summary()
```
executed in 5ms, finished 20:23:21 2023-01-23

```
Dropout_rate: 0.1
learning_rate: 0.001
activation: elu
Score: 0.6370370388031006
Trial summary
Hyperparameters:
units1: 4
units2: 28
units3: 16
Dropout_rate: 0.2
learning_rate: 0.001
activation: elu
Score: 0.5962963104248047
Trial summary
Hyperparameters:
units1: 28
units2: 32
units3: 32
Dropout_rate: 0.0
learning_rate: 0.01
```

In [89]:
```
%tensorboard --logdir ./tensorboard/bologs
```
executed in 10ms, finished 20:23:22 2023-01-23

```
Reusing TensorBoard on port 6006 (pid 4282), started 1 day, 23:15:20 ago. (Use '!kill 4282' to kill it.)
```



In [ ]:
```
```

7 Optimal LSTM

In [90]:
```
# Create a model
def create_model(hu1=32, hu2=32, hu3=32, lookback=60, features=10):

    tensorflow.keras.backend.clear_session()

    # instantiate the model
    model = Sequential()
    model.add(LSTM(units=hu1, input_shape=(lookback, features), activation = 'elu', return_sequences=True, name='LSTM1'))
    model.add(Dropout(0.2, name='Dropout1'))

    model.add(LSTM(units=hu2, activation = 'elu', return_sequences=True, name='LSTM2'))
    model.add(Dropout(0.2, name='Drouput2'))

    model.add(LSTM(units=hu3, activation = 'elu', return_sequences=False, name='LSTM3'))
    model.add(Dense(units=1, activation='sigmoid', name='Output'))              # can also specify linear activation function

    # specify optimizer separately (preferred method))
    opt = RMSprop(learning_rate=0.001, rho=0.9, epsilon=1e-08)
#     opt = Adam(learning_rate=0.001, epsilon=1e-08, decay=0.0)       # adam optimizer seems to perform better for a single lstm

    # model compilation
#     model.compile(optimizer=opt, loss='mse', metrics=['mae'])
    model.compile(optimizer=opt,
                  loss=BinaryCrossentropy(),
                  metrics=['accuracy',
                           Precision(),
                           Recall()])

    return model
```
executed in 5ms, finished 20:23:22 2023-01-23

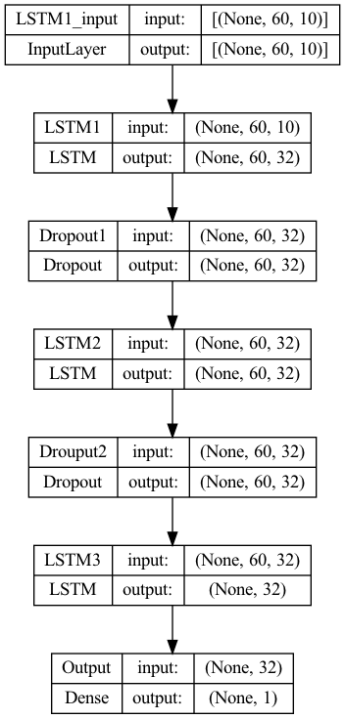
In [91]:
```
# lstm network
model_3 = create_model(hu1=32, hu2=32, hu3=32, lookback=seqlen, features=numfeat)
```
executed in 304ms, finished 20:23:23 2023-01-23

In [92]:
```
model_3.summary()
```
executed in 16ms, finished 20:23:23 2023-01-23

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 LSTM1 (LSTM)                (None, 60, 32)            5504

 Dropout1 (Dropout)          (None, 60, 32)            0

 LSTM2 (LSTM)                (None, 60, 32)            8320

 Drouput2 (Dropout)          (None, 60, 32)            0

 LSTM3 (LSTM)                (None, 32)                8320

 Output (Dense)              (None, 1)                 33

=================================================================
Total params: 22,177
Trainable params: 22,177
Non-trainable params: 0
_________________________________________________________________
```

In [93]:
```
plot_model(model_3, to_file='./img/model_2.png', show_shapes=True, show_layer_names=True)
```
executed in 549ms, finished 20:23:24 2023-01-23

Out[93]:



In [94]:
```
# Specify callback functions
model_path = (results_path / 'model_3.h5').as_posix()
logdir = os.path.join("logs", datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))

my_callbacks = [
    EarlyStopping(patience=10, monitor='loss', mode='min', verbose=1, restore_best_weights=True),
    ModelCheckpoint(filepath=model_path, verbose=1, monitor='loss', save_best_only=True),
    TensorBoard(log_dir=logdir, histogram_freq=1)
]
```
executed in 5ms, finished 20:23:24 2023-01-23

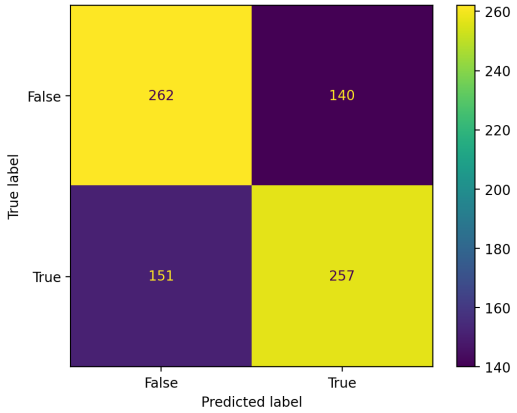
In [96]:
```
random.seed(2023)
# Model fitting
history = model_3.fit(g,
          epochs=500,
          verbose=1,
          callbacks=my_callbacks,
          shuffle=False,
          class_weight=class_weight)
```
executed in 35.6s, finished 20:27:25 2023-01-23

```
16/16 [==============================] - ETA: 0s - loss: 0.6326 - accuracy: 0.6414 - precision: 0.6250 - recall: 0.6143
Epoch 12: loss did not improve from 0.63073
16/16 [==============================] - 1s 83ms/step - loss: 0.6326 - accuracy: 0.6414 - precision: 0.6250 - recall: 0.6143
Epoch 13/500
16/16 [==============================] - ETA: 0s - loss: 0.6298 - accuracy: 0.6425 - precision: 0.6245 - recall: 0.6218
Epoch 13: loss improved from 0.63073 to 0.62981, saving model to results/lstm_time_series/model_3.h5
16/16 [==============================] - 1s 74ms/step - loss: 0.6298 - accuracy: 0.6425 - precision: 0.6245 - recall: 0.6218
Epoch 14/500
15/16 [===========================>..] - ETA: 0s - loss: 0.6333 - accuracy: 0.6375 - precision: 0.6195 - recall: 0.6182
Epoch 14: loss did not improve from 0.62981
16/16 [==============================] - 1s 63ms/step - loss: 0.6329 - accuracy: 0.6374 - precision: 0.6194 - recall: 0.6154
Epoch 15/500
15/16 [===========================>..] - ETA: 0s - loss: 0.6432 - accuracy: 0.6307 - precision: 0.6089 - recall: 0.6269
Epoch 15: loss did not improve from 0.62981
16/16 [==============================] - 1s 62ms/step - loss: 0.6418 - accuracy: 0.6318 - precision: 0.6100 - recall: 0.6250
Epoch 16/500
16/16 [==============================] - ETA: 0s - loss: 0.6290 - accuracy: 0.6486 - precision: 0.6298 - recall: 0.6325
Epoch 16: loss improved from 0.62981 to 0.62905, saving model to results/lstm_time_series/model_3.h5
16/16 [==============================] - 1s 83ms/step - loss: 0.6290 - accuracy: 0.6486 - precision: 0.6298 - recall: 0.6325
Epoch 17/500
```

In [97]:
```
ypred_3 = np.where(model_3.predict(g_, verbose=False) > 0.5, 1, 0)
```
executed in 621ms, finished 20:27:26 2023-01-23

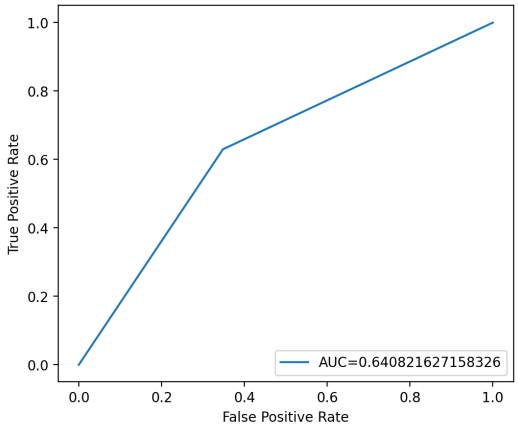
In [98]:
```
confusion_matrix = metrics.confusion_matrix(y_test[seqlen:], ypred_3)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
                                            display_labels = [False, True])

cm_display.plot()
plt.show()
```

executed in 252ms, finished 20:27:27 2023-01-23



In [99]:
```
fpr, tpr, _ = metrics.roc_curve(y_test[seqlen:],  ypred_3)
auc = metrics.roc_auc_score(y_test[seqlen:], ypred_3)

#create ROC curve
plt.figure(figsize=(6,5))
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```

executed in 167ms, finished 20:27:28 2023-01-23



In [100]:
```
print(classification_report(y_test[seqlen:], ypred_3))
```

executed in 7ms, finished 20:27:28 2023-01-23

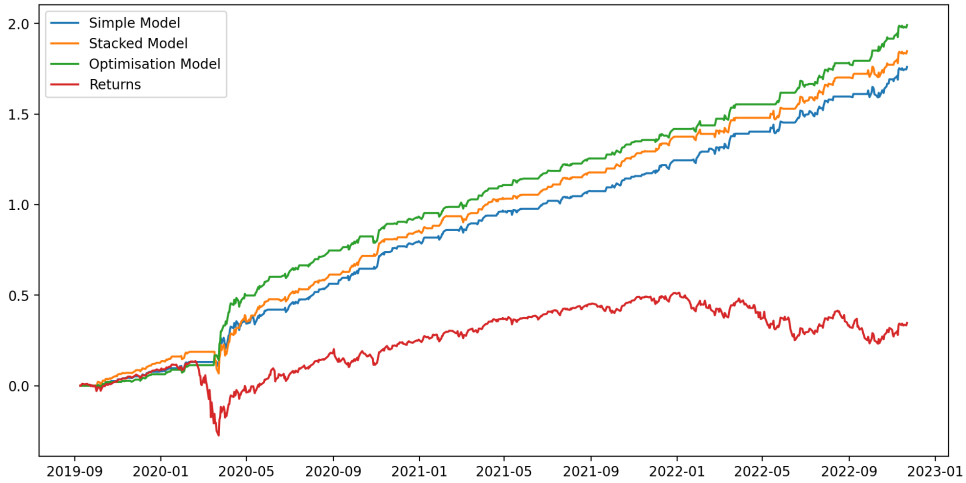
```
              precision    recall  f1-score   support

           0       0.63      0.65      0.64       402
           1       0.65      0.63      0.64       408

    accuracy                           0.64       810
   macro avg       0.64      0.64      0.64       810
weighted avg       0.64      0.64      0.64       810
```

8 Trading Strategy

8.1 Long-Only Strategy

In [101]:
```
bk_lo = stock_data[['Returns']].loc[X_test.iloc[60:,:].index]
bk_lo['Sign_1'] = ypred_1
bk_lo['Sign_2'] = ypred_2
bk_lo['Sign_3'] = ypred_3
bk_lo['Model_1'] = bk_lo['Returns'] * bk_lo['Sign_1'].fillna(0)
bk_lo['Model_2'] = bk_lo['Returns'] * bk_lo['Sign_2'].fillna(0)
bk_lo['Model_3'] = bk_lo['Returns'] * bk_lo['Sign_3'].fillna(0)

plt.figure(figsize=(12,6))
plt.plot(np.cumsum(bk_lo['Model_1']), label='Simple Model')
plt.plot(np.cumsum(bk_lo['Model_2']), label='Stacked Model')
plt.plot(np.cumsum(bk_lo['Model_3']), label='Optimisation Model')
plt.plot(np.cumsum(bk_lo['Returns']), label='Returns')
plt.legend()
plt.show()
```

executed in 323ms, finished 20:27:29 2023-01-23



In [102]:
```
print('Simple LSTM Return:', round(np.cumsum(bk_lo['Model_1'])[-1]*100, 2), '%')
print('Stacked LSTM Return:', round(np.cumsum(bk_lo['Model_2'])[-1]*100, 2), '%')
print('Optimal LSTM Return:', round(np.cumsum(bk_lo['Model_3'])[-1]*100, 2), '%')
print('Holding Return:', round(np.cumsum(bk_lo['Returns'])[-1]*100, 2), '%')
```

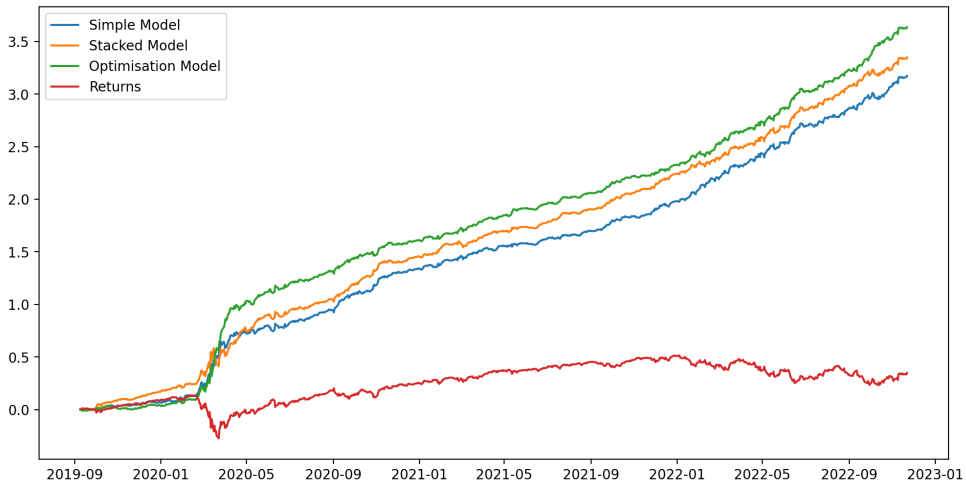
executed in 6ms, finished 20:27:30 2023-01-23

```
Simple LSTM Return: 176.01 %
Stacked LSTM Return: 184.74 %
Optimal LSTM Return: 199.13 %
Holding Return: 34.67 %
```

8.2 Long-Short Strategy

In [103]:
```
bk_ls = stock_data[['Returns']].loc[X_test.iloc[60:,:].index]
bk_ls['Sign_1'] = bk_lo['Sign_1'].apply(lambda sig: 1 if sig == 1 else (-1 if sig == 0 else sig))
bk_ls['Sign_2'] = bk_lo['Sign_2'].apply(lambda sig: 1 if sig == 1 else (-1 if sig == 0 else sig))
bk_ls['Sign_3'] = bk_lo['Sign_3'].apply(lambda sig: 1 if sig == 1 else (-1 if sig == 0 else sig))

bk_ls['Model_1'] = bk_ls['Returns'] * bk_ls['Sign_1'].fillna(0)
bk_ls['Model_2'] = bk_ls['Returns'] * bk_ls['Sign_2'].fillna(0)
bk_ls['Model_3'] = bk_ls['Returns'] * bk_ls['Sign_3'].fillna(0)

plt.figure(figsize=(12,6))
plt.plot(np.cumsum(bk_ls['Model_1']), label='Simple Model')
plt.plot(np.cumsum(bk_ls['Model_2']), label='Stacked Model')
plt.plot(np.cumsum(bk_ls['Model_3']), label='Optimisation Model')
plt.plot(np.cumsum(bk_ls['Returns']), label='Returns')
plt.legend()
plt.show()
```

executed in 301ms, finished 20:27:30 2023-01-23



In [104]:
```
print('Simple LSTM Return:', round(np.cumsum(bk_ls['Model_1'])[-1]*100, 2), '%')
print('Stacked LSTM Return:', round(np.cumsum(bk_ls['Model_2'])[-1]*100, 2), '%')
print('Optimal LSTM Return:', round(np.cumsum(bk_ls['Model_3'])[-1]*100, 2), '%')
print('Holding Return:', round(np.cumsum(bk_ls['Returns'])[-1]*100, 2), '%')
```

executed in 8ms, finished 20:27:31 2023-01-23

```
Simple LSTM Return: 317.34 %
Stacked LSTM Return: 334.8 %
Optimal LSTM Return: 363.58 %
Holding Return: 34.67 %
```

In [ ]:
```

```

In [ ]:
```

```

References

Hilpisch, Y. (2020), *Artificial Intelligence in Finance*, O'Reilly Media.

Kannan Singaravelu, C. (2022*a*), *Feature Engineering : Extraction & Selection*, Machine Learning Workshop Notes.

Kannan Singaravelu, C. (2022*b*), *KMeans Clustering*, CQF Python Lab Notes.

Kannan Singaravelu, C. (2022*c*), *Neural Networks: Deep Sequence Modeling*, Machine Learning Workshop Notes.

Kannan Singaravelu, C. (2022*d*), *Self Organizing Maps*, CQF Python Lab Notes.

Kursa, M. B. & Rudnicki, W. R. (2010), 'Feature selection with the boruta package', *Journal of statistical software* **36**, 1–13.

Lau, K. W., Yin, H. & Hubbard, S. (2006), 'Kernel self-organising maps for classification', *Neurocomputing* **69**(16-18), 2033–2040.

Xu, D. & Tian, Y. (2015), 'A comprehensive survey of clustering algorithms', *Annals of Data Science* **2**(2), 165–193.

Zhou, H. B. & Gao, J. T. (2014), Automatic method for determining cluster number based on silhouette coefficient, *in* 'Advanced materials research', Vol. 951, Trans Tech Publ, pp. 227–230.

Zou, Z. & Qu, Z. (2020), 'Using lstm in stock prediction and quantitative trading', *CS230: Deep Learning, Winter* pp. 1–6.