FT-100 Stock Screening and ML forecasting Software

Final project submission for Code Labs Academy Data Science and AI Bootcamp

Ginger software available at: https://github.com/colinmoughton/Ginger

Author: Colin Moughton

Date: Dec 24

Rev. 1

Table of Contents

1.0 Introduction

The project goal was to create a predictive model capable of identifying trading opportunities. The model operates on historical daily stock price data that has the data features ‘open’, ‘high’, ‘low’, ‘close’ and ‘volume’ (OHLCV) for stocks within the FT-100 index. This financial index is made up of the largest 100 public companies trading on the London Stock Exchange.

The machine learning (ML) software developed to achieve this goal is called ‘Ginger’. It runs as a FastAPI web application. This report aims to describe what Ginger is and how it works. The software is available on GitHub and can easily be run using Docker.

2.0 Working principle

Ginger operates on a static set of data for 100 stocks. Each stock has approximately 1700 daily OHLCV records. Currently Ginger does not have a ‘real-time’ predictive capability, however, this functionality could easily be added.

The main focus of this work was to understand whether ML models could provide a stock picking advantage using a known historical dataset.

The Ginger workflow is currently made up of three major stages.

1. Stock screening
2. ML model selection and training
3. Back testing and results evaluation

2.1 Stock screening

The first stage of the Ginger workflow involves a stock screening process. This is a simple algorithm and does not involve machine learning, it is primarily to identify and classify successful historic trading events. In this process the user inputs several parameters that define a successful single trading event. Ginger optimistically assumes long-trades; this is when stock is bought at a price, the stock then rises in value and is sold for a profit. The parameters input to the screener are:

- Trade Size – the amount of money being risked (e.g. £10,000)
- Target Trade Profit – if successful, the return achieved (e.g. £500)
- Trade Loss Limit – a stop-loss (sell) would trigger if loss greater than this (e.g. 150)
- Test End Date – this is the last day of testing – governed by the available data.
- Max Trade Duration – the forward time window in which the trade must happen (days).
- Training Duration – the number of days of data the ML model will be trained on (days).
- Test Duration – the number of days the ML model will be tested on (days).

Using these input parameters each of the 100 stocks in the database are evaluated to establish how many successful trading events occur over the entire history considered. (training + test duration).

Each daily OHLCV record of each stock is considered and labeled with a ‘1’ if a successful trade event occurs within the forward window specified, and ‘0’ if not. This classification step only uses the forward ‘high’ and ‘low’ price data to discern whether that day is ‘1’ or ‘0’.

This method of course has knowledge of the future that would not be available in real life, however, the aim of the screener is only to evaluate historic data to identify whether many positive trading events occur and how spread out they are. The screener output enables the user to pick a good stock with many known positive events on which to run machine learning models.

The screener therefore provides information to help identify good stocks to consider further.

Figures 1 & 2 below describe how the screener discerns whether the current day is classified as a ‘1’ or a ‘0’.

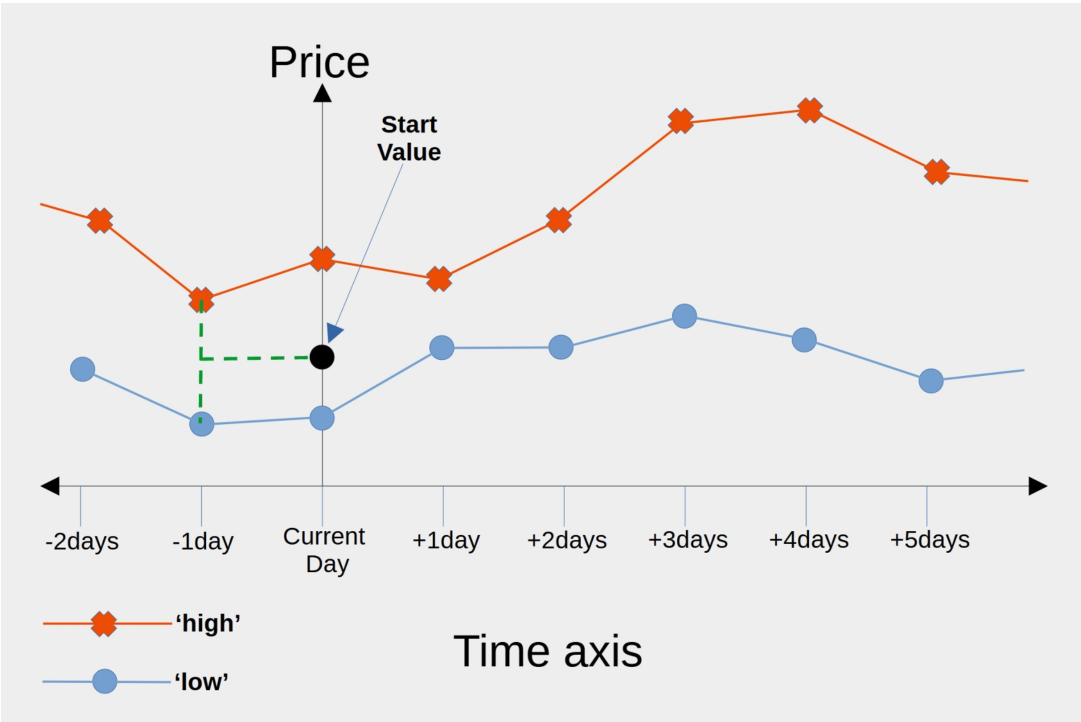


Figure 1: The first step in the screener algorithm is to calculate the mean of the previous day’s high & low. This is used as the start value for the current day.

In this first part of the screener algorithm a ‘0’ label could result, either if the ‘current day’ high is below the ‘start value’ or if the ‘current day’ low is low is above the ‘start value’. In either of these cases the buy order would not fill if placed at the start value, at the start of the current day.

The user input values are used to define three key parameters:

- High threshold
- Low threshold
- Max Trade Duration

these are shown in figure 2 below. The High & Low thresholds are calculated using the Start Value, Trade Size, Target Trade Profit and Trade Loss Limit.

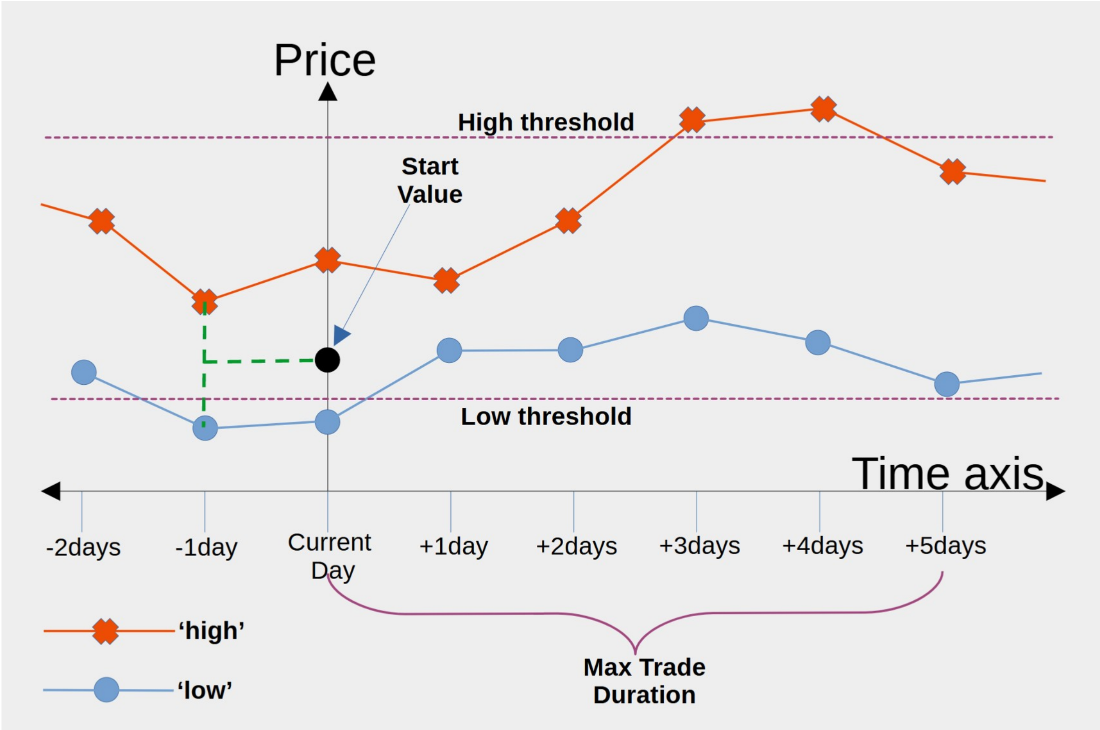


Figure 2: The screener algorithm computes a High & Low threshold and checks whether the high breaks the High threshold within the Max Trade Duration before the Low breaks the Low threshold. Only the first day onward are considered. If this happens (as in this case) a label value of '1' is assigned to the current day. If not, a '0'.

In this way the screener works through all of the days in all of the stocks and then presents a league table of stocks showing the following results for each stock:

- Total Records – the number of days considered.
- Occurrence – how many times a positive event '1' label occurs.
- Occurrence Interval – the averaged time between '1' labels.
- Trade Duration – the average time it took to reach the High threshold.
- Four Sigma – distribution, smaller is better, 95% of trade duration fall within.

The output league table of stocks is presented in order of Occurrence. The more '1' occurrences the better.

To summarize, the screener classifies each day of each stock as a '1' or a '0'. A '1' means if money was invested on this day, it would hit the profit target and not stop out. The stock with the highest number of events ('1's) represents the biggest opportunity if the distribution is not abnormal. So, for the given input parameters, the screener returns a league table of stocks ranked by profit opportunity. This helps the user select a good stock to model with ML.

2.2 Machine learning models

The stock screener provided a classification method for each daily record. This classification method required daily ‘high’ and ‘low’ prices from each record as inputs. Initially the ML problem was approached as a classification model. This approach ran into difficulties, the classification ML model types and approaches are described in a later section.

The approach that was found to work best was to use a Recurrent Neural Network (RNN) to predict actual forward price values of the stocks, rather than predict whether each day was a ‘1’ or a ‘0’. To enable performance measurement, the classification stage was then achieved by post processing the predicted price values.

The best results were achieved using a Gated Recurrent Unit (GRU) model, see figure 3 below, but modified to include attention. Long Short Term Memory (LSTM) models were also explored, however, they appeared to offered no improvement over the GRU model in terms of predictive performance and were slower to run.

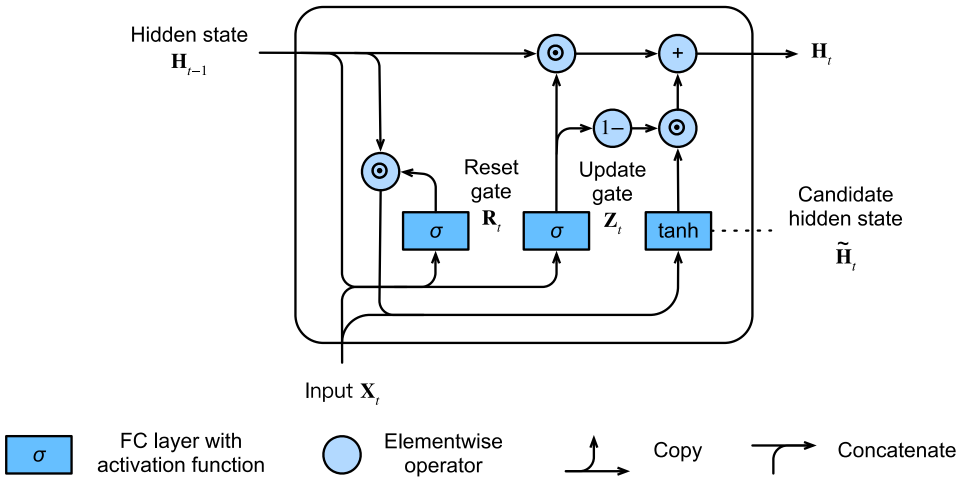


Figure 3: Gated Recurrent Unit (GRU) architecture (https://d2l.ai/chapter_recurrent-modern/gru.html)

Initially the GRU model was used to predict the ‘high’ price time series. It was realized at this point that both the ‘high’ and ‘low’ price series would require predicting if the screener post-processing classification was to be replicated, (as shown in figure 1 & 2), using the predicted values. This approach was named the ‘High-Low’ ML model. This ‘High-Low’ ML model has not been implemented in this software revision but will be included in the next revision.

Instead of the using the ‘High-Low’ ML method, a simplified approach was developed. This new approach was called the ‘High-Low-Mean’ ML method. The intention was to evaluate whether a simplified approach could deliver a faster solution.

The ‘High-Low-Mean’ ML method would pre-process the ‘high’ and ‘low’ price time series data to give the ‘mean price’ time series. This simplification would require only one ML model to be trained, rather than two. However, it did mean that the post-processing classification method would require modification. This modification is covered in the next section (Back testing & results evaluation).

The benefit of the 'High-Low-Mean' model was that a single average price time series curve could be modeled and predicted, then evaluated for accuracy and loss performance.

The 'high-low-mean' GRU ML model was fed daily incremented windows of data, the best performance was obtained using window length values of 25 days of daily time series data. Each window of data used to predict the next days price value. So in this training and test stage, the ability of the model to predict the next day was assessed by comparing the graphs of the actual test time series data for average price and the predicted test data time series curves.

The metrics chosen to assess the training results were:

- An overlaid plot of actual vs predicted average prices – looking at visual fit.
- R^2 score – to assess this numerically over the epochs.
- Training and validation loss graph to understand model convergence.

The training results are shown below for a given stock:

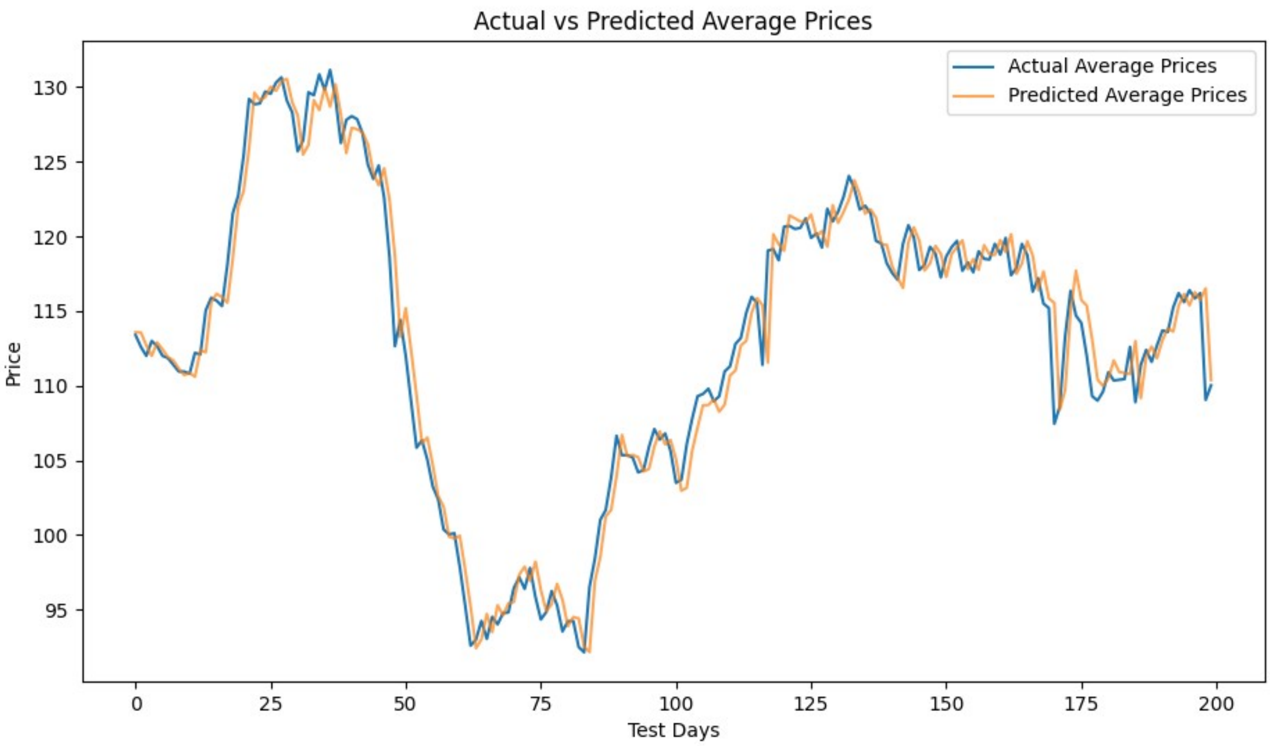


Figure 4: Overlaid plot of actual vs predicted average prices for a given stock

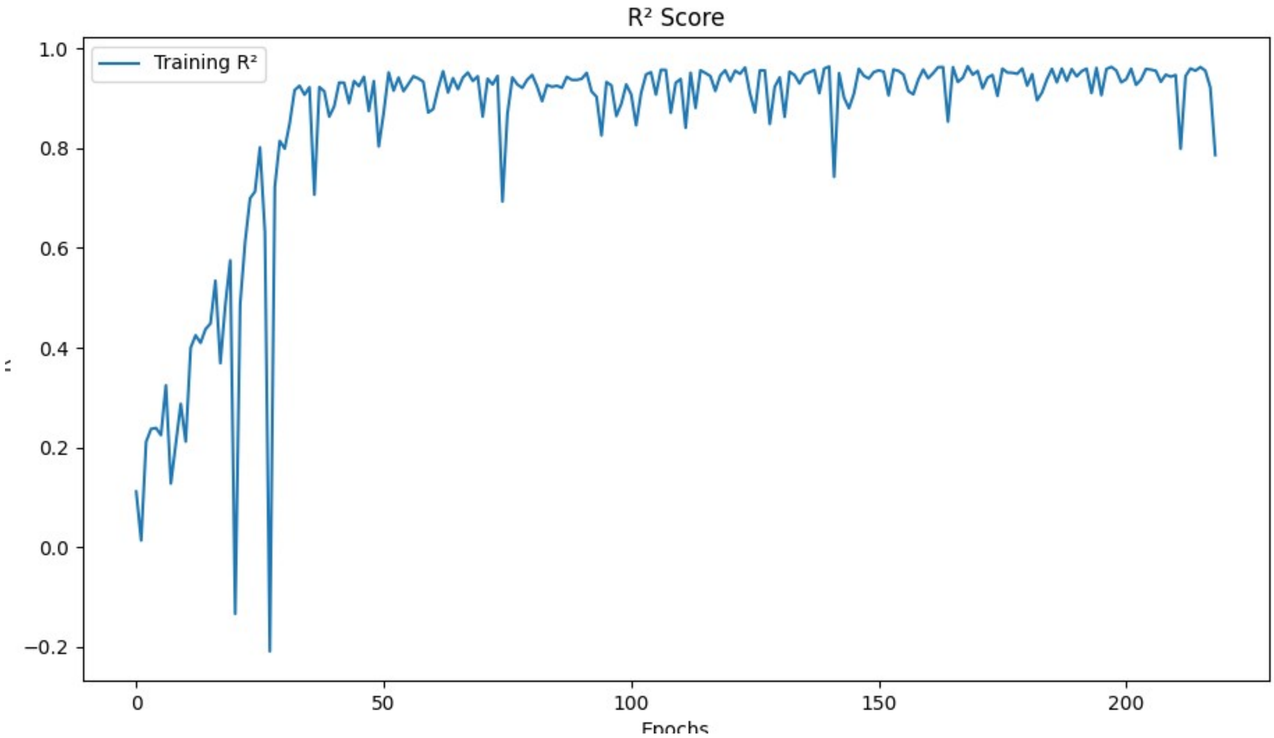


Figure 5: The R^2 score of the actual verses predicted curves.

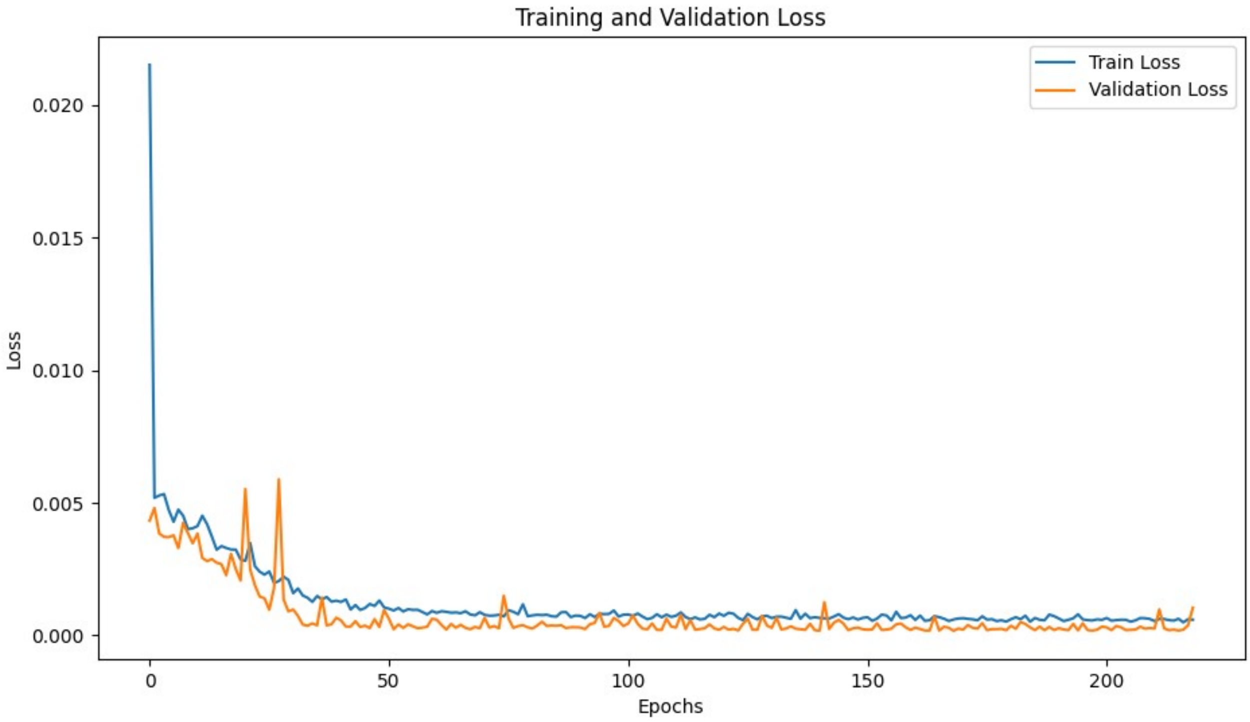


Figure 6: The training and validation loss curves.

2.2.1 Hyper parameter tuning

As mentioned above, the sliding data window found to work best was 25 days. A number of other hyper parameters were also tuned. Different numbers of GRU layers and GRU units per layer were experimented with. Four GRU layers each with 50 GRU units and a 0.2 dropout layer were found to work well. In the experiment these values were doubled and halved to understand the impact on the training results.

The learning rate was also adjusted to understand if this would improve performance. The learning rate was reduced to 0.001. This was found to deliver better performance on the stocks tested.

The final dropout layer was fed into an attention layer, this layer was added to provide attention over different periods of the time series history.

The attention layer was fed into a Global pooling layer and finally into a dense layer with one node to give the predicted price output for the given day.

All of the training day data was used for training, and the test days data used for both validation and test. The number of epochs was set to 500 with an early stopping callback. A further callback was used to pull out the R^2 values during the run.

Several different optimizers listed in the Keras documentation were tried but none performed noticeably better than adam.

A wider sensitivity study would be useful to dial the hyper parameters in further.

The attention-GRU model definition is shown in the listing below:

```
X, y = self.create_sequences_multifeature(features_normalized, target_feature_index, window_size)

# Adjust X shape for the GRU model (samples, timesteps, features)
X = X.reshape((X.shape[0], X.shape[1], X.shape[2]))  # Shape (num_samples, window_size, num_features)

train_days = self.training_duration - window_size

X_train, X_test = X[:train_days], X[train_days:]
y_train, y_test = y[:train_days], y[train_days:]

# Input Layer
input_layer = Input(shape=(X.shape[1], X.shape[2]))

# GRU Layers with Dropout
gru_output = GRU(50, return_sequences=True)(input_layer)
gru_output = Dropout(0.2)(gru_output)

gru_output = GRU(50, return_sequences=True)(gru_output)
gru_output = Dropout(0.2)(gru_output)

gru_output = GRU(50, return_sequences=True)(gru_output)
gru_output = Dropout(0.2)(gru_output)

gru_output = GRU(50, return_sequences=True)(gru_output)
gru_output = Dropout(0.2)(gru_output)

# Attention Layer
attention_output = Attention()([gru_output, gru_output])

# Global Average Pooling Layer
pooled_output = GlobalAveragePooling1D()(attention_output)

# Fully Connected Output Layer
output_layer = Dense(1)(pooled_output)

# Define Model
attention_model = Model(inputs=input_layer, outputs=output_layer)
```

```python
attention_model.compile(optimizer= tf.keras.optimizers.Adam(learning_rate=0.001),
                        loss='mean_squared_error', metrics=['accuracy']
                        )

# Instantiate the R² callback
r2_callback = R2ScoreCallback(validation_data=(X_test, y_test), target_scaler=target_scaler)

# Train the model with the callback
history = attention_model.fit(
        X_train, y_train,
        validation_data=(X_test, y_test),
        epochs=500,
        batch_size=16,
        callbacks=[
                    tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=True),
                    r2_callback  # Add the custom R² callback
                ]
)
```

Figure 7: Attention-GRU model code listing.

2.2.2 Data analysis and feature engineering

The daily OHLCV data was initially sourced from a python module called y-Finance. This program offered free downloads of the data needed for all of the stocks in the FT-100 index. The data was graphed and evaluated. Unfortunately, a number of anomalies were found when spot checks were carried out using data from the exchange website.

Another source of data was found called Alpha Vantage. To obtain data from this source the user is asked to create and identity and get an api key. This key could then be used to query their database and obtain the desired data.

This OHLCV data was also spot checked against data available on the stock exchange website. It was found to match and so was accepted. Further checks for missing values were carried out.

The OHLCV data used on Ginger does not include the impact of splits and dividends. The rational for this is that the trades are short and it thought best to train the models on actual exchange data that was live, rather than post-processed data.

During the development journey of Ginger the initial intention was to first try using a dense neutral network to predict the screener classified labels. To achieve this each row of data would need to have all the information it needed about past price movements. An initial feature correlation matrix showed that even with the addition of some moving averages and calculated features from the screener there was very little variation that a machine learning model could make use of.

The initial feature correlation matrix is shown below in figure 8. A number of features were developed top provide the neural network with more context. Figure 9 shows a more developed feature correlation matrix.

Interestingly, the dense neutral network was poor at predicting the labels, even after adding attention and a rich set of data features.

When testing on the GRU and LSTM models it was found that adding many data features did little to improve performance, likely because they are all derived from the same data. After much experimentation the best input features for the attention-GRU model were found to be the average price time series (also the target) and a feature showing the percent change in that value from the previous day.

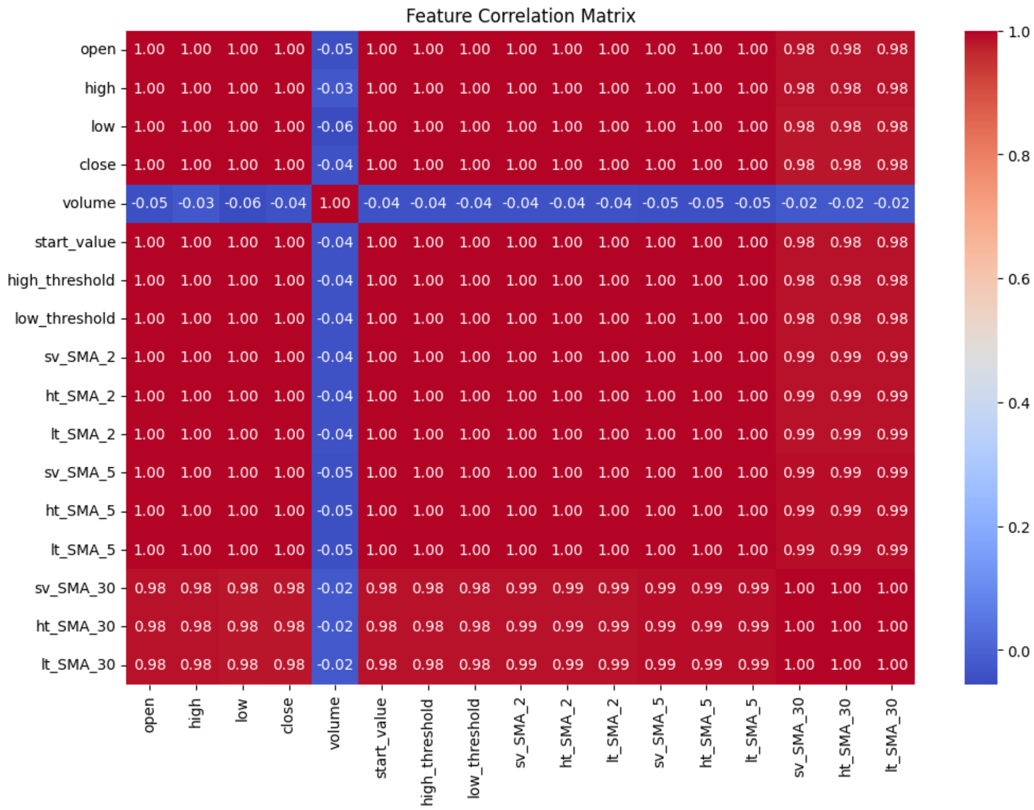


Figure 8: Initial feature correlation matrix.

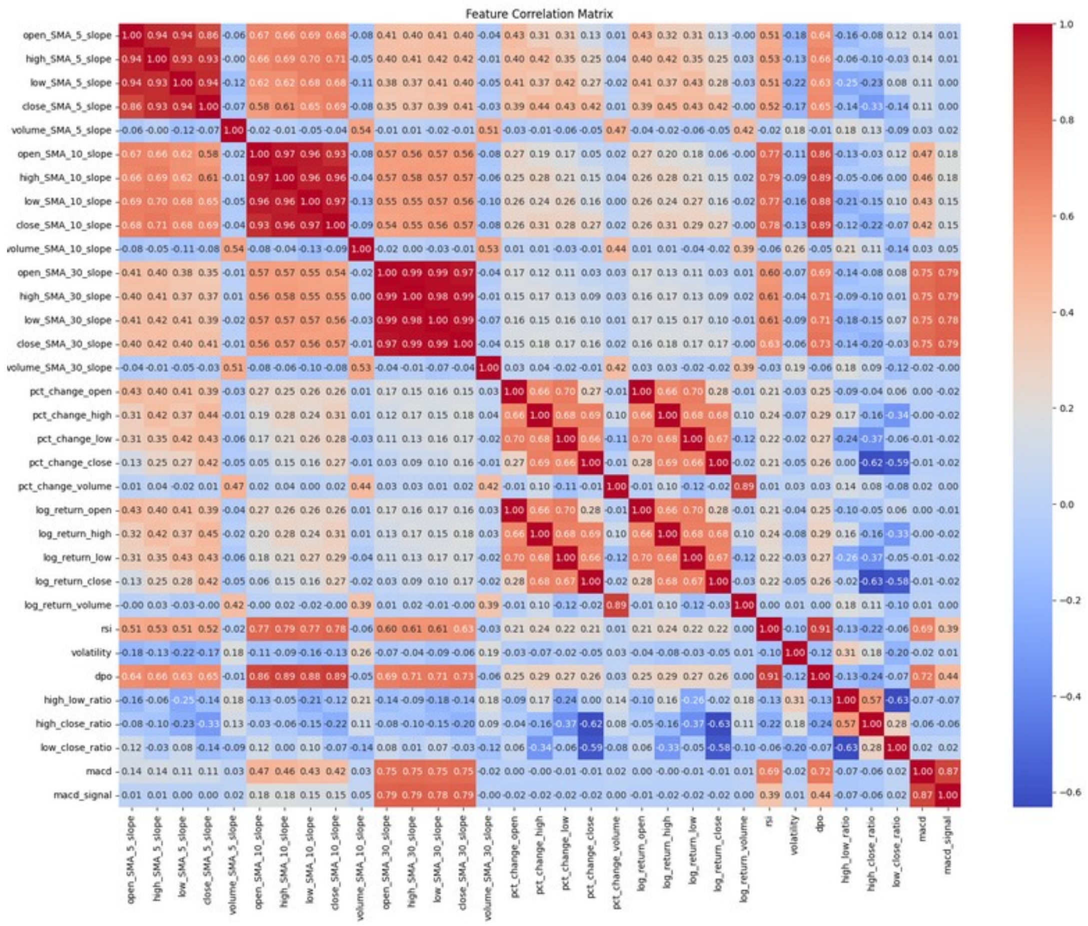


Figure 9: Developed feature correlation matrix.

2.3 Back testing and results evaluation

The weights from the attention-GRU ML model and those of the scalers were saved at the end of the training phase. The next stage of the modeling was to develop a back tester.

2.3.1 Back testing

The aim of the back testing within Ginger is to predict for each day whether that day should be classified as a '1' or a '0'. This is carried out for all of the test days. Only with knowledge of the past, no knowledge of the future is used in this testing. The testing is done in two stages.

1. Average time series price prediction using the attention-GRU model for the forward time window (Max Trade Duration).
2. The screener classification algorithm applied to the predicted price data to discern whether the day is predicted as a '0' or '1'.

In the second stage a key issue with the simplified 'high-low-mean' ML model is that the high and low values of the time series had been averaged for the prediction. This meant that the screener classification algorithm also required simplification. The implications of simplifying the screener classification algorithm from 'high-low', to 'average' had to be considered before using any predicted 'averaged' price values for classification.

Simplified High-Low-Mean screener classification algorithm

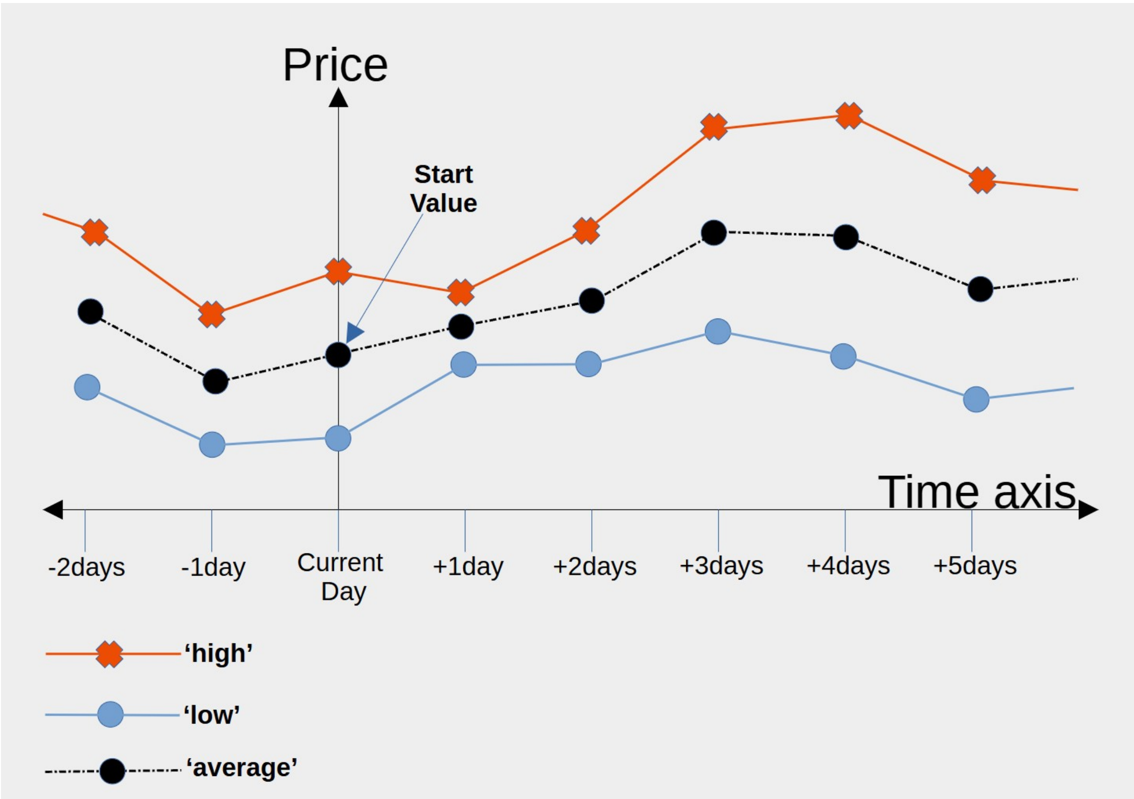


Figure 10: Price chart showing the average of high and low prices

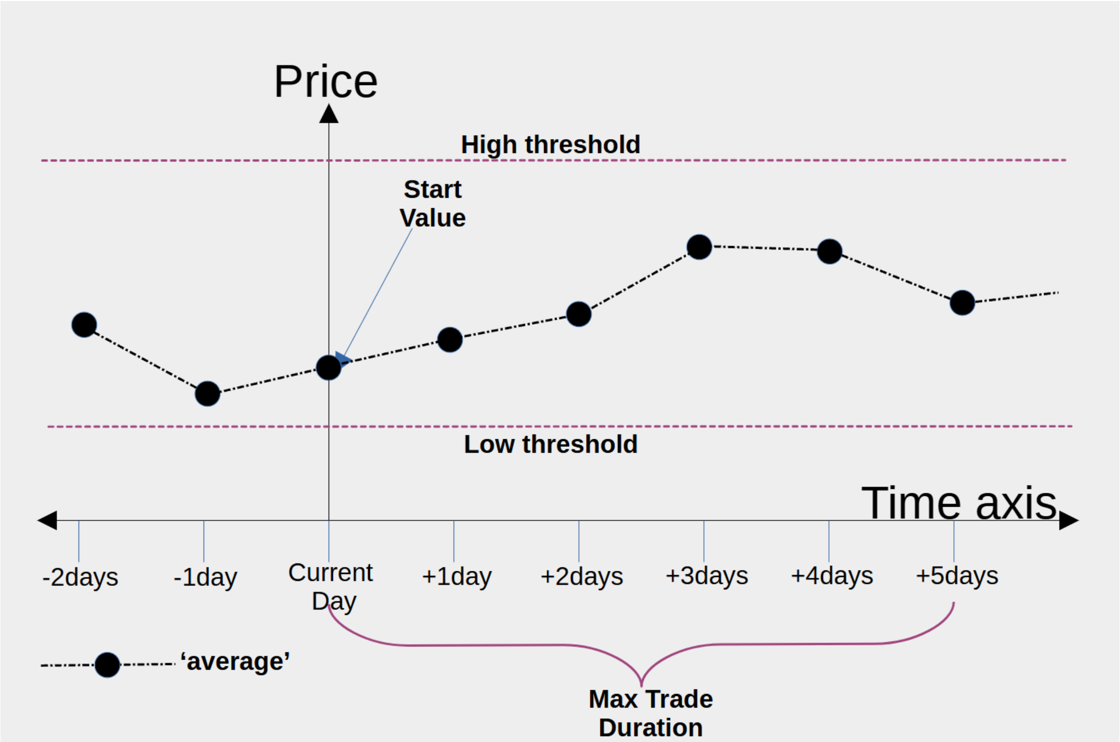


Figure 11: Price chart showing the thresholds generated from the input data and the duration using the simplified screener classification algorithm. With 'High-Low' this would have been a '1', now with 'High-Low-Mean' this is a '0'.

It is clear from figure 10 and 11 that the 'High-Low', and the 'High-Low-Mean' were likely to give different predictions of '1's and '0's.

To address this simplification Ginger shows two back testing graphs. The first graph shows how the averaged 'High-Low-Mean' screener algorithm compares against the original screener algorithm that uses 'high' and 'low' time series curves independently to calculate labels.

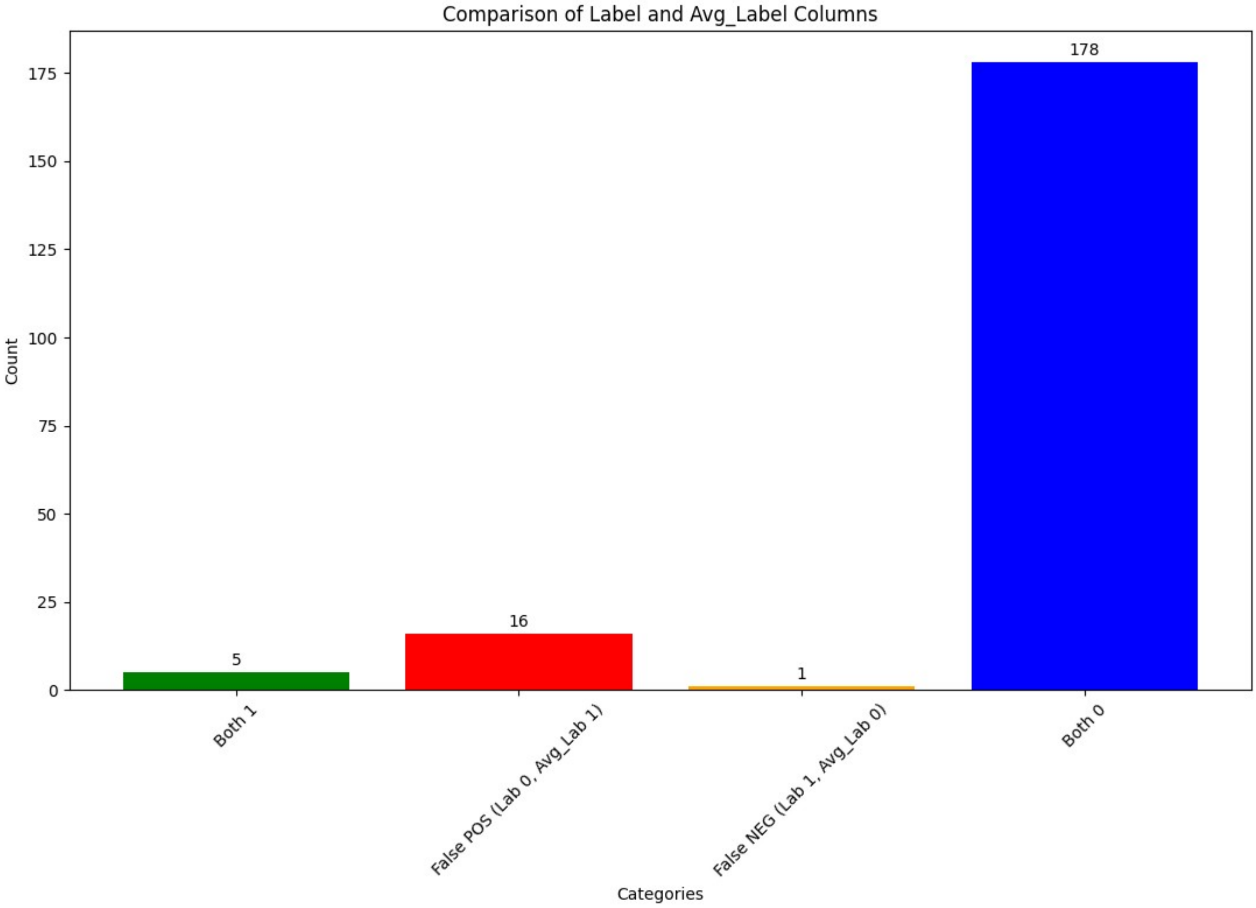


Figure 12: Shows the effect of simplification on the classification algorithm

It can be seen from figure 12 that the simplified 'Avg_label' screener classification algorithm impacts the labeling. It correctly captures 5 out of 6 of the '1' labels. However, it also generates 16 false positives. False positives are the worst results because each of them represents a loosing trade. The real outcome is '0' but the simplified High-Low-Mean' algorithm thinks it is a '1'. This is an important graph because it shows that the averaging simplification has introduced significant degradation in performance even before any predictive modeling is carried out.

The simplified average time series price screener algorithm basically has full knowledge of all of the data, but incorrectly identifies many false positives compared to the 'High-Low' screener classification algorithm because of the averaging.

This graph is presented in the back test results for each stock within Ginger that uses the High-Low-Mean simplified ML algorithm workflow. The reason it is presented is to give context when the predictive 'High-Low-Mean' results are presented.

The simplified screener classification methodology has been presented first here because its assumptions have such a profound effect in terms of false positives.

As mentioned above, the first stage of the back testing is the average time series price prediction using the attention-GRU model for the forward time window (Max Trade Duration). In this first stage the model accepts the previous 25 daily historic records and then predicts the next days average price value. The attention-GRU model is used with the scaling and model weights from the training to achieve this. Then the predicted value of average price is used to calculate the daily percentage change (the other input feature) for that day. The new predicted values of average price and daily percent change are added to the front of the 25 day sliding window with the last values dropped. This updated 25 day window is then used to calculate the next day, and so on until predictions are created for the whole ‘Max Trade Duration’ as specified in the input parameters.

These predicted values for the days in the ‘Max Trade Duration’ are then fed into the simplified screener classifier algorithm (figures 10 & 11) to establish whether each day is a predicted ‘1’ or ‘0’. These results are then presented in the same format as the simplified screener results as below.

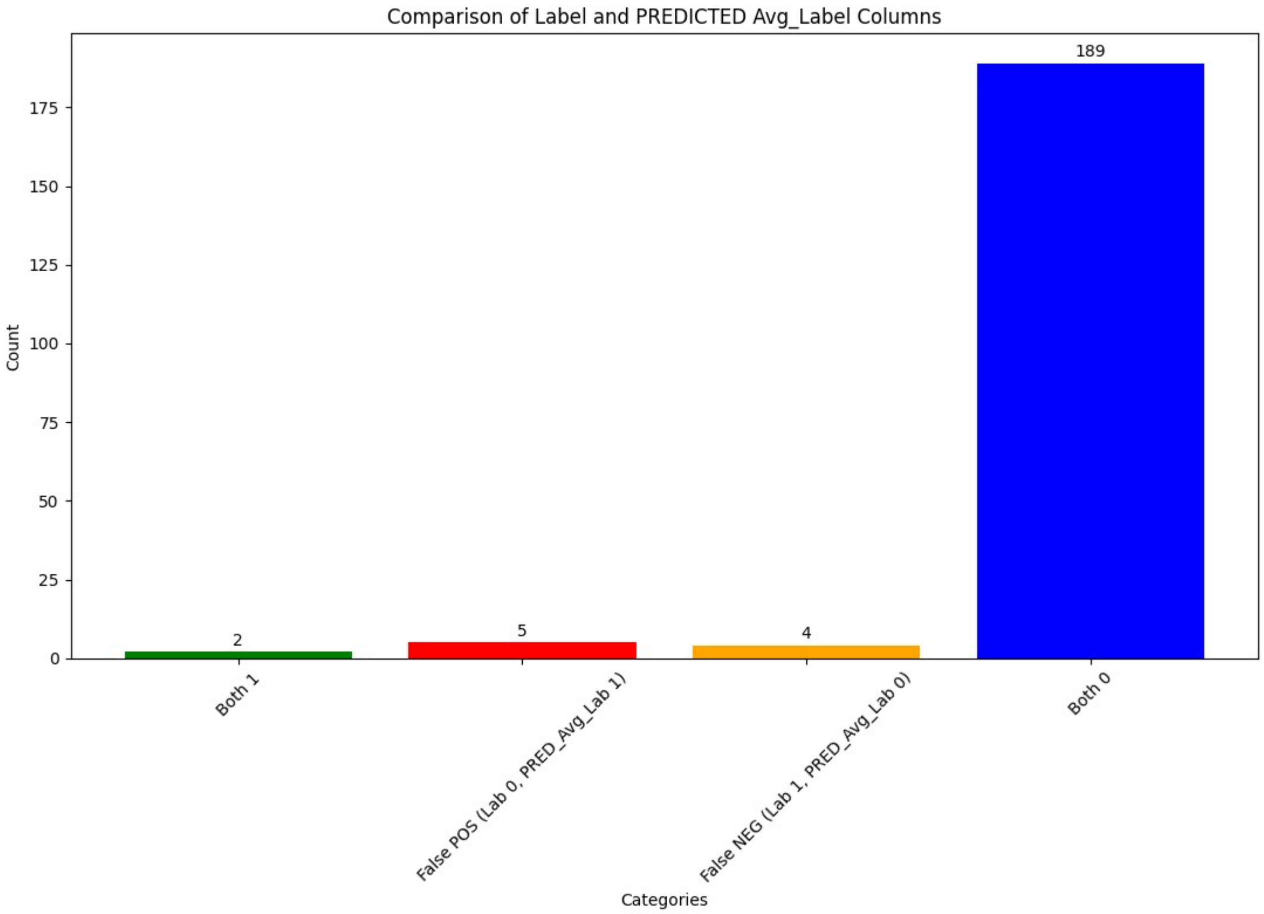


Figure 13: Comparison of original label and predicted ‘High-Low-Mean’ ML model results.

The predicted results in figure 13 above show that the ‘High-Low-Mean’ ML model was able to predict 2 of the 6 positive trade events (‘1’s). It had considerably fewer false positives than the screener alone (5 compared to 16).

If this algorithm was followed it would have yielded a profit. The ratio of wins to losses was 1:2.5. The input parameters were a profit of £700 and a stop loss of £150. So without including trading costs the trades would be +£325 in profit.

However, it would be wise to implement the ‘High-Low’ algorithm prior to trading this model because there is obviously a significant impact due to the averaging.

It would also be sensible to paper trade the model for a significant period to get confidence in its predictive ability before live trading.

Also the fit of the model varies between stocks. Further tuning would be sensible.

3.0 Ginger user workflow

The Ginger workflow has the following stages:

- Model creation and inputs
- Stock screening
- Stock data exploration & selection
- ML Model selection and training
- Back testing and results evaluation

3.1 Model creation and inputs

Ginger is a FastAPI web application. The home page shows the user a list of completed and uncompleted models. It also enables the user to start a new model by writing a name in the text box and pressing the 'Add' button. The home page is shown in figure 14 below:

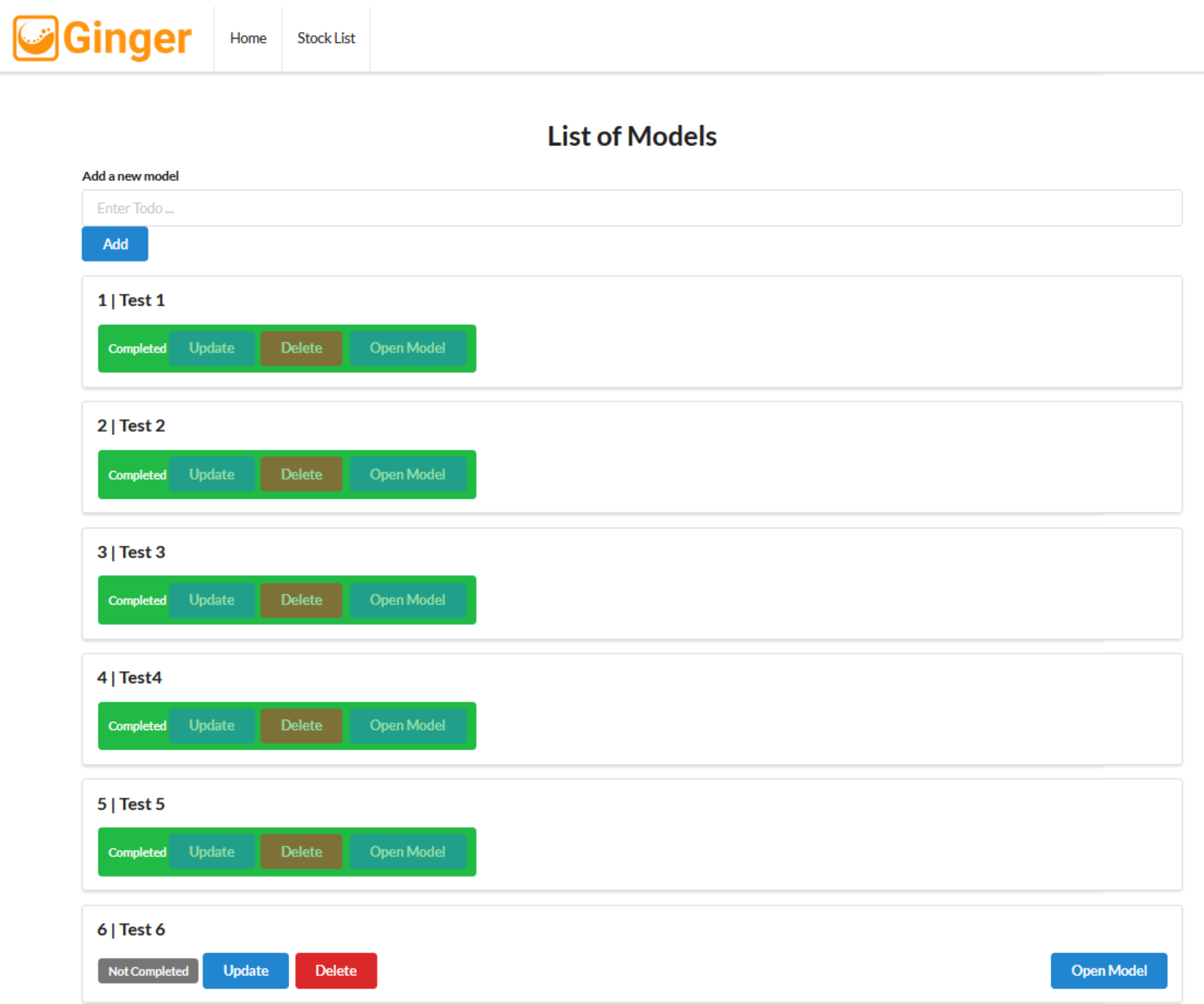


Figure 14: Ginger home page showing a list of models.

To start working with Ginger the user opens a new model and is presented with a form. The user enters a number of parameters into the model details form. These parameters define what a successful trading event looks like and the maximum time span of that event. Also defined are the number of days that will be used by Ginger for training and testing a machine learning model. The parameter input form is shown in figure 15 below:

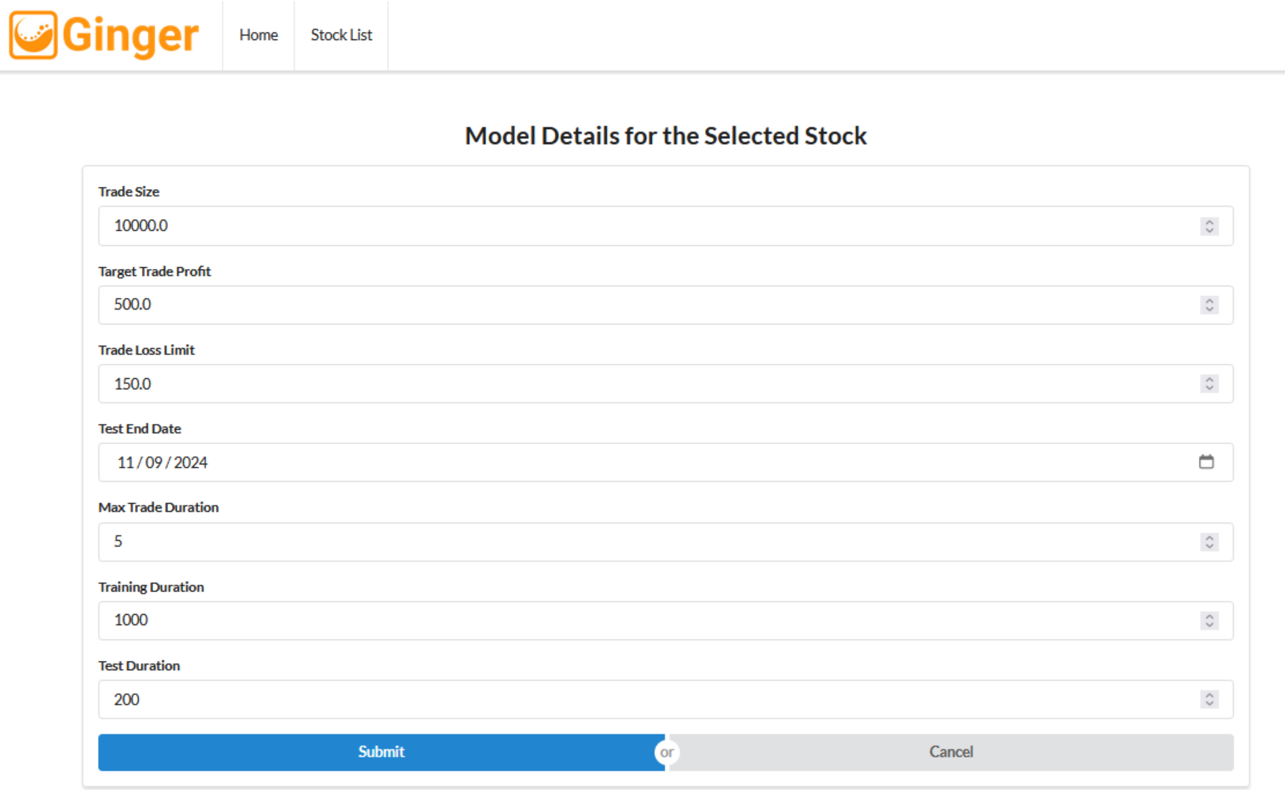


Figure 15: Model parameter input form.

Once the user has filled in the form they hit the 'submit' button and the model input parameters are saved to the model database. This action returns the user to the home page.

Now the parameters are saved the user can return to the model by clicking the 'open model' button on the new model which will now appear in the list (see figure 14 – example Test 6).

Once the model details page has reopened a new full width black button appears labeled 'screen stocks' as shown below:

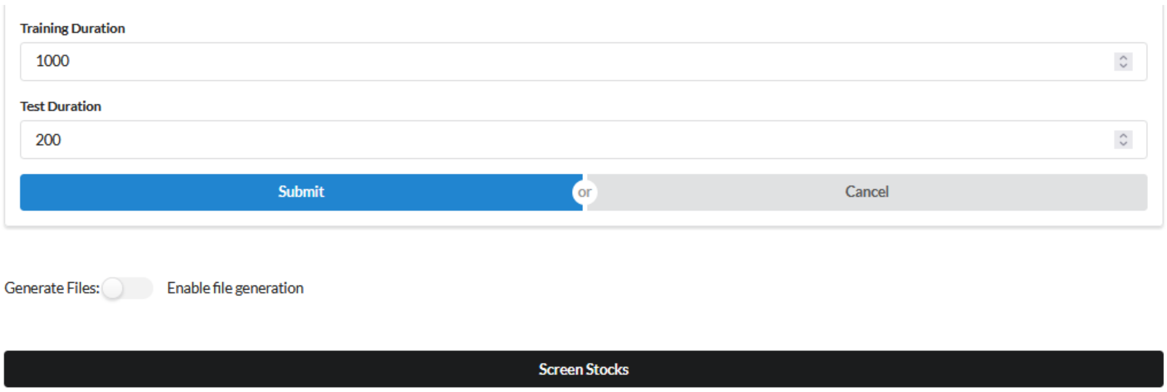
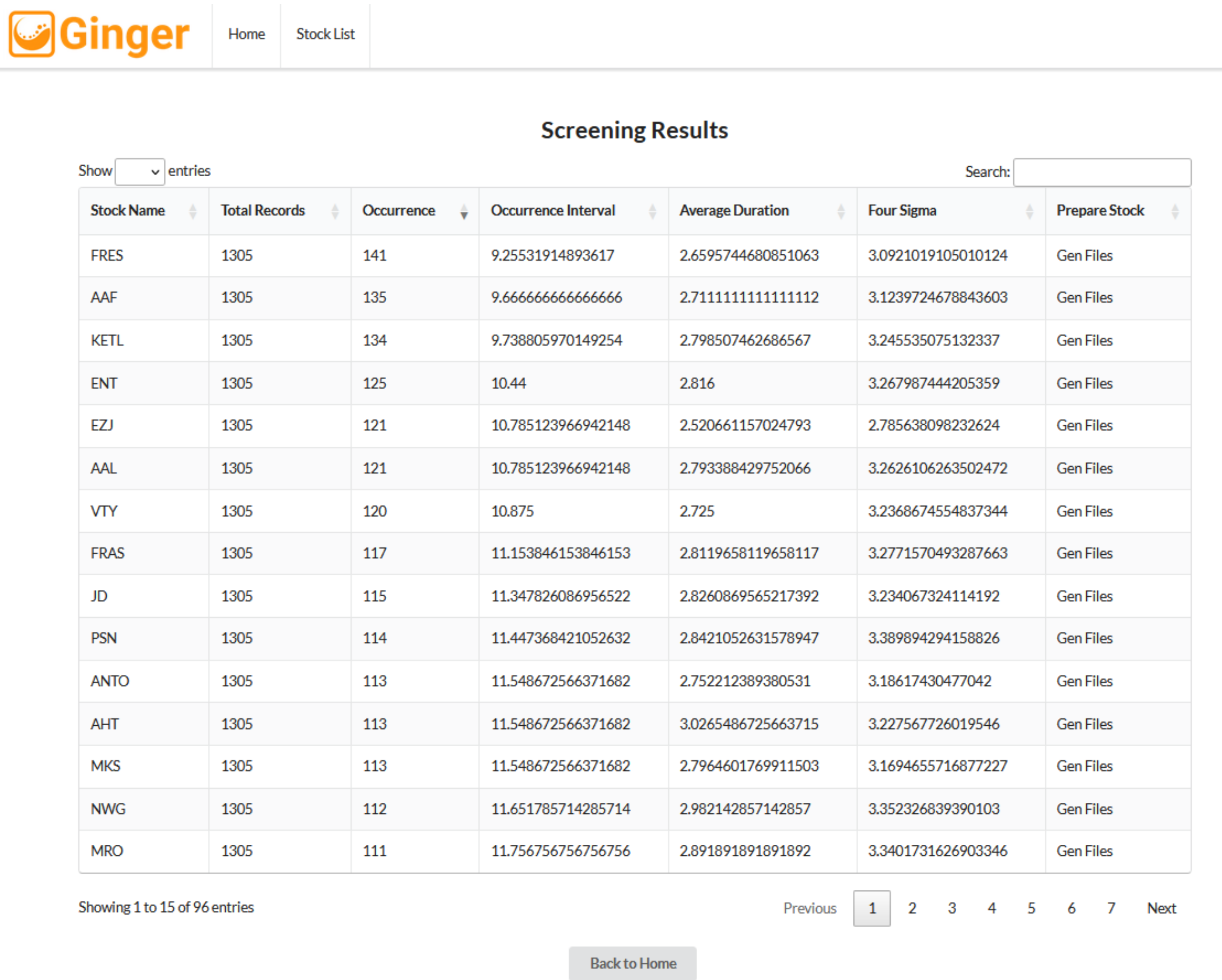


Figure 16: Stock screening option appears.

3.2 Stock screening

When the ‘Screen Stocks’ button is pressed with the ‘Enable file generation’ toggle ‘off’ the trade parameters defined in the ‘model details’ form (figure 15) are applied to all of the stocks in the database. The screener first checks that there is enough time series data for each stock. It then applies the screening classification algorithm to discern and label each day in each time series. A label ‘1’ means that the trade would give a positive outcome if applied on that day. ‘0’ means it would not. A screener results league table is shown to the user at this stage as shown below in figure 17.



Stock Name	Total Records	Occurrence	Occurrence Interval	Average Duration	Four Sigma	Prepare Stock
FRES	1305	141	9.25531914893617	2.6595744680851063	3.0921019105010124	Gen Files
AAF	1305	135	9.666666666666666	2.7111111111111112	3.1239724678843603	Gen Files
KETL	1305	134	9.738805970149254	2.798507462686567	3.245535075132337	Gen Files
ENT	1305	125	10.44	2.816	3.267987444205359	Gen Files
EZJ	1305	121	10.785123966942148	2.520661157024793	2.785638098232624	Gen Files
AAL	1305	121	10.785123966942148	2.793388429752066	3.2626106263502472	Gen Files
VTY	1305	120	10.875	2.725	3.2368674554837344	Gen Files
FRAS	1305	117	11.153846153846153	2.8119658119658117	3.2771570493287663	Gen Files
JD	1305	115	11.347826086956522	2.8260869565217392	3.234067324114192	Gen Files
PSN	1305	114	11.447368421052632	2.8421052631578947	3.389894294158826	Gen Files
ANTO	1305	113	11.548672566371682	2.752212389380531	3.18617430477042	Gen Files
AHT	1305	113	11.548672566371682	3.0265486725663715	3.227567726019546	Gen Files
MKS	1305	113	11.548672566371682	2.7964601769911503	3.1694655716877227	Gen Files
NWG	1305	112	11.651785714285714	2.982142857142857	3.352326839390103	Gen Files
MRO	1305	111	11.756756756756756	2.891891891891892	3.3401731626903346	Gen Files

Figure 17: Screening results shown with ‘Enable generate files’ toggle off.

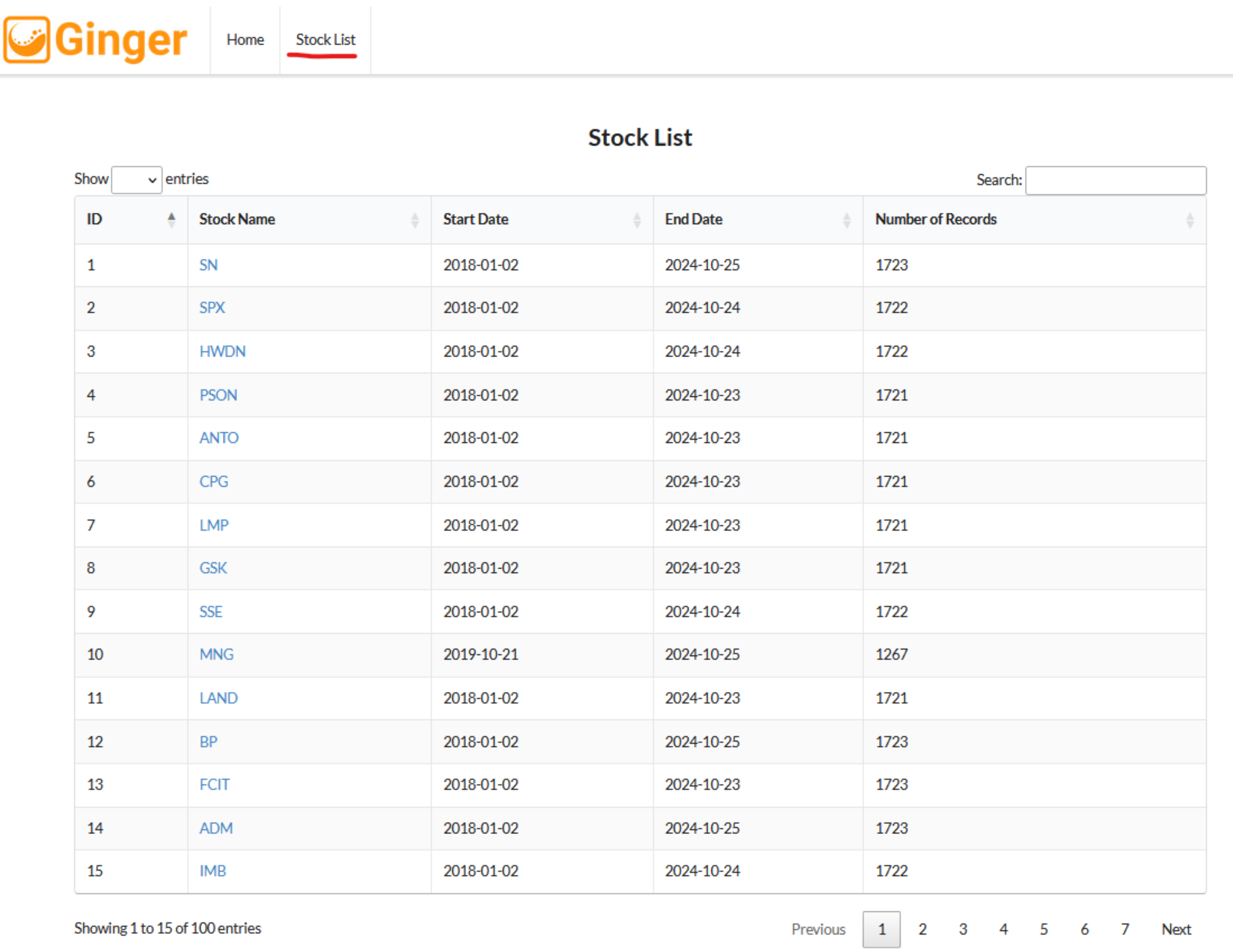
The league table shows a record for each stock. It shows the total number of records required for the ML model. The main column of interest is the ‘Occurrence’ column, the table is sorted on this value in descending order. The Occurrence shows the number of ‘1’ events, or the number of times a positive trade would happen given complete future knowledge of the historic prices its based on.

The aim of Ginger is to predict correctly all of these events, with no false positives, and without any future knowledge. In essence, the Occurrence number represents the size of the opportunity for that particular stock. Stock with low Occurrence will be more difficult to train and offer the lowest return. The other supporting values help the user assess the distribution of the Occurrence duration.

3.3 Stock data exploration & selection

Given this information, the user is likely to want to have a look at the stock price curves for the stocks with the best Occurrence values.

To do this they can note the stocks of interest and hit the home button. Then move to the stock list database to explore these curves.



Stock List

Show entries

Search:

ID	Stock Name	Start Date	End Date	Number of Records
1	SN	2018-01-02	2024-10-25	1723
2	SPX	2018-01-02	2024-10-24	1722
3	HWDN	2018-01-02	2024-10-24	1722
4	PSON	2018-01-02	2024-10-23	1721
5	ANTO	2018-01-02	2024-10-23	1721
6	CPG	2018-01-02	2024-10-23	1721
7	LMP	2018-01-02	2024-10-23	1721
8	GSK	2018-01-02	2024-10-23	1721
9	SSE	2018-01-02	2024-10-24	1722
10	MNG	2019-10-21	2024-10-25	1267
11	LAND	2018-01-02	2024-10-23	1721
12	BP	2018-01-02	2024-10-25	1723
13	FCIT	2018-01-02	2024-10-23	1723
14	ADM	2018-01-02	2024-10-25	1723
15	IMB	2018-01-02	2024-10-24	1722

Showing 1 to 15 of 100 entries

Previous 1 2 3 4 5 6 7 Next

Figure 18: The stock list database table. This database holds the OHLCV data for all stocks and shows how many records are available with the dates they span. The names are hyperlinked to more detailed data.

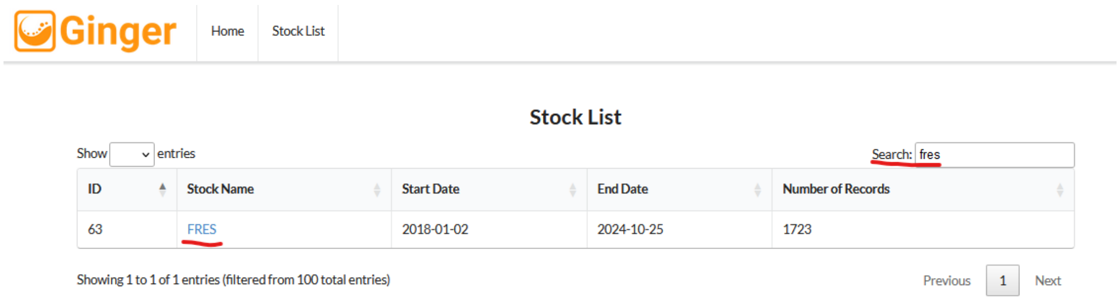


Figure 19: Any stock of interest can easily be selected by searching the name.

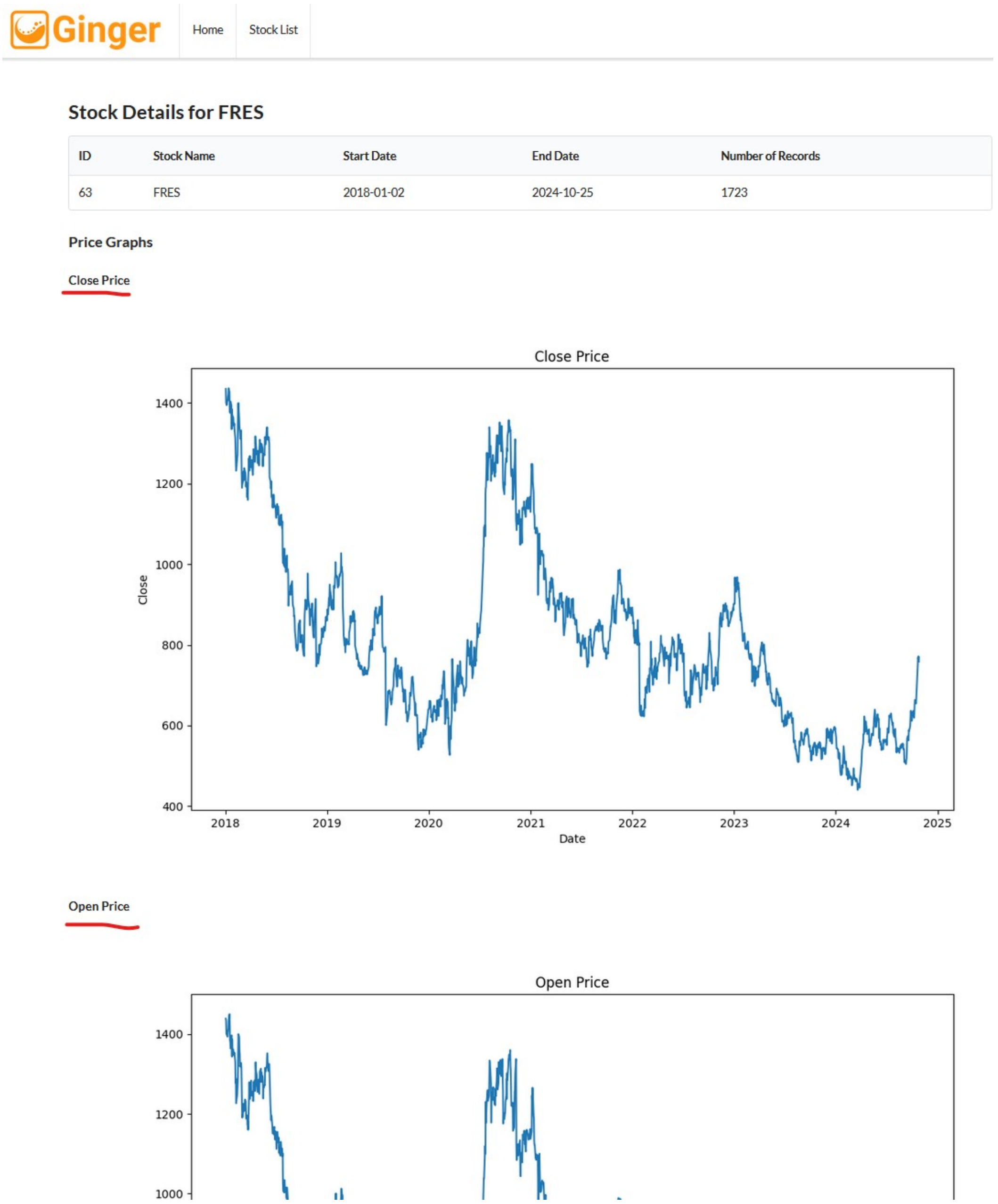


Figure 20: When the stock ticker name hyperlink is clicked the user is taken to a page with details for that particular stock. Graphs showing 'open', 'high', 'low', 'close' and 'volume' are shown.

Once the user has selected a stock they want to model they go back to the 'Home' page and select their open model again from the list. They are presented with the same form as before (figures 15 & 16).

Now that they know which stock they want to take further they switch the 'Enable file generation' toggle to 'on' and hit the black 'Screen Stocks' button again.

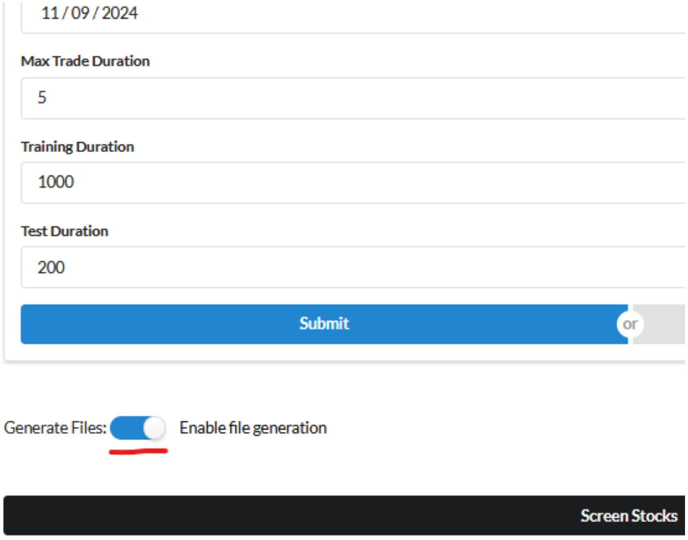
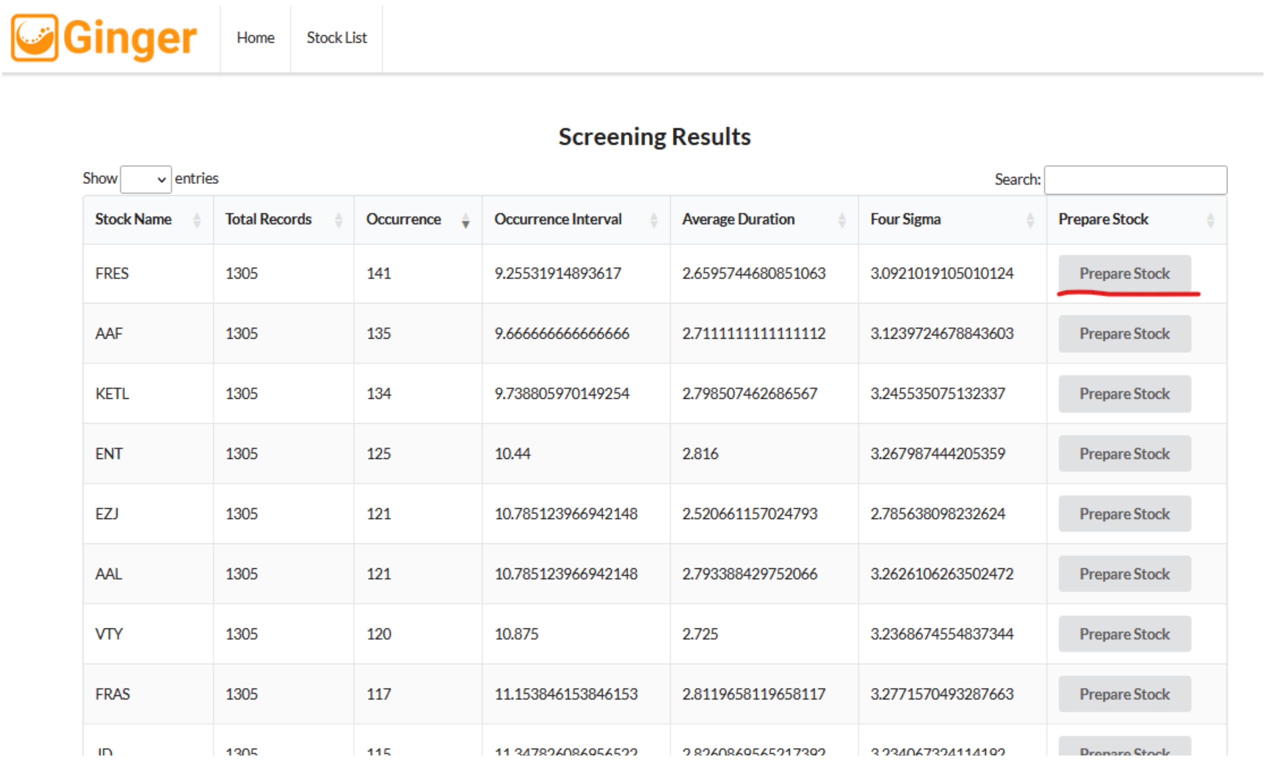


Figure 21: Screening stocks with the 'Enable file generation' toggle switched 'on'.

This time the returned league table has selection buttons in the right hand column.



Stock Name	Total Records	Occurrence	Occurrence Interval	Average Duration	Four Sigma	Prepare Stock
FRES	1305	141	9.25531914893617	2.6595744680851063	3.0921019105010124	Prepare Stock
AAF	1305	135	9.666666666666666	2.7111111111111112	3.1239724678843603	Prepare Stock
KETL	1305	134	9.738805970149254	2.798507462686567	3.245535075132337	Prepare Stock
ENT	1305	125	10.44	2.816	3.267987444205359	Prepare Stock
EZJ	1305	121	10.785123966942148	2.520661157024793	2.785638098232624	Prepare Stock
AAL	1305	121	10.785123966942148	2.793388429752066	3.2626106263502472	Prepare Stock
VTY	1305	120	10.875	2.725	3.2368674554837344	Prepare Stock
FRAS	1305	117	11.153846153846153	2.8119658119658117	3.2771570493287663	Prepare Stock
[illegible]	1305	115	[illegible]	[illegible]	[illegible]	Prepare Stock

Figure 22: Screening results with function to select a single stock for the ML model stage.

Once the user has selected a stock from the screener results they are returned to their open model page. It now has the next workflow step available to them.

3.4 ML Model selection and training

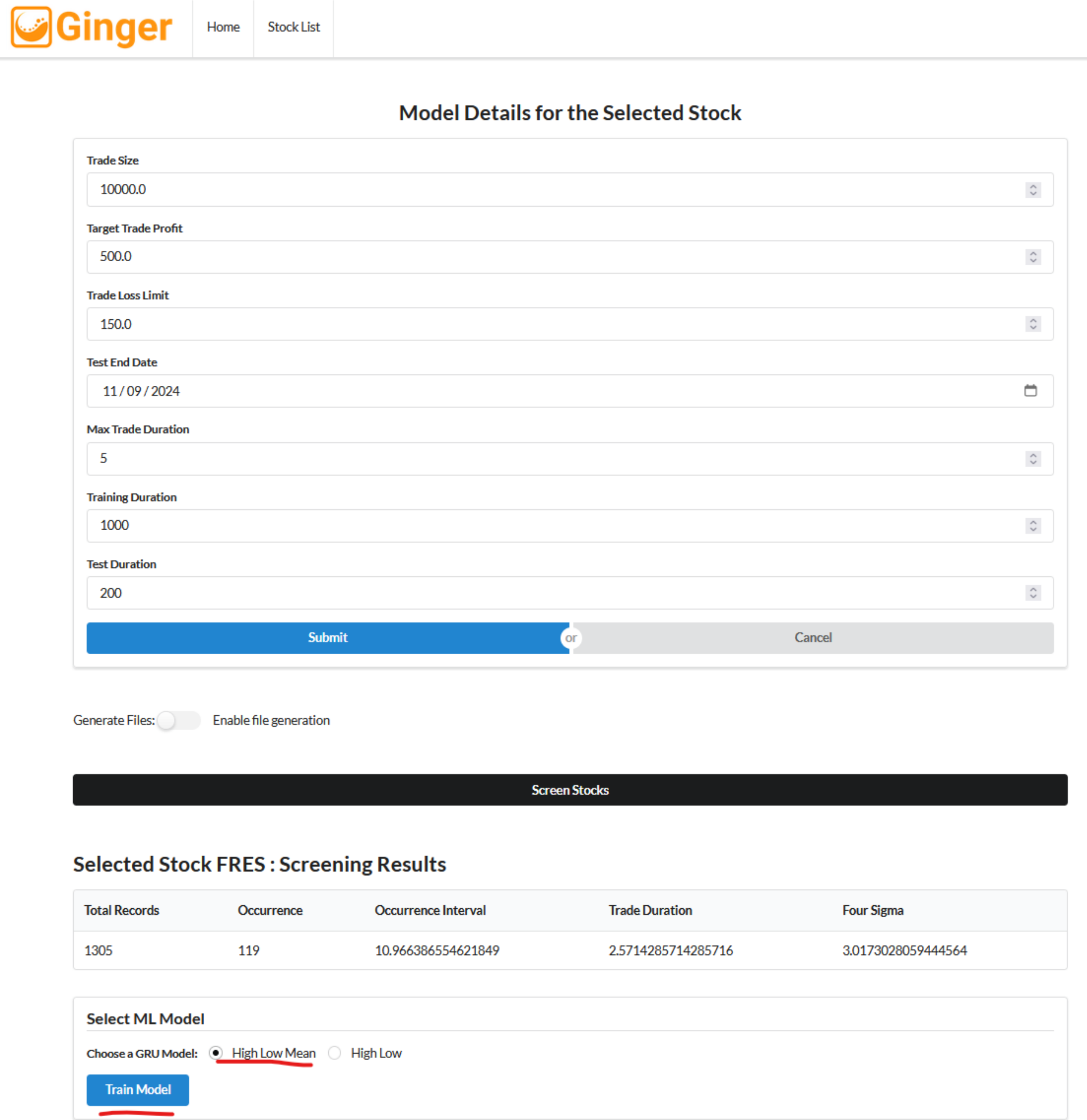


Figure 23: The open model details page now has the selected stock and its details along with the option to train an ML model.

The radio buttons give the option of a ‘High Low Mean’ ML model or a ‘High Low’ ML model.

Currently only the former is available as discussed in the previous section.

Once the user hits the ‘Train Model’ button the selected machine learning model is trained. When it finishes the training the training results are presented.

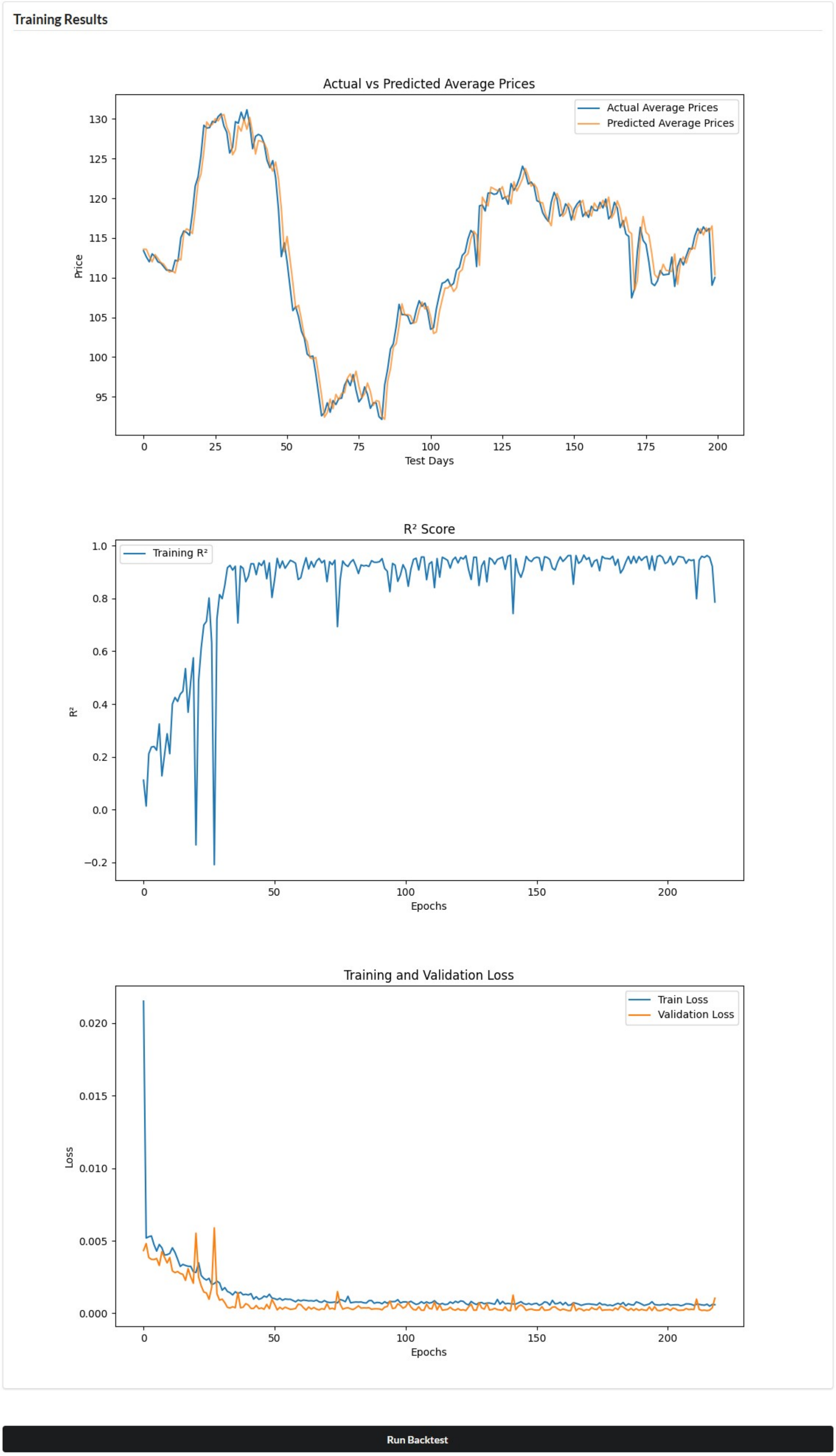


Figure 24: Training results showing actual vs predicted curves, R^2 and loss.

The results from training on the training data days specified shows the performance of the model on the test data days specified on the model detail's form.

3.5 Back testing and results evaluation

If the user is happy with the training results they can move on to back test the test days. The back test operates on each of the test days individually, predicting forward the price values over the length of days specified in the 'Max Trade Duration' field of the model details form. Then the screener style classification algorithm is used to discern whether that day is categorized as a '1' or as a '0'.

When using the 'High Low Mean' ML model two back test results graphs are presented as discussed in the previous section. The first graph is shown below:

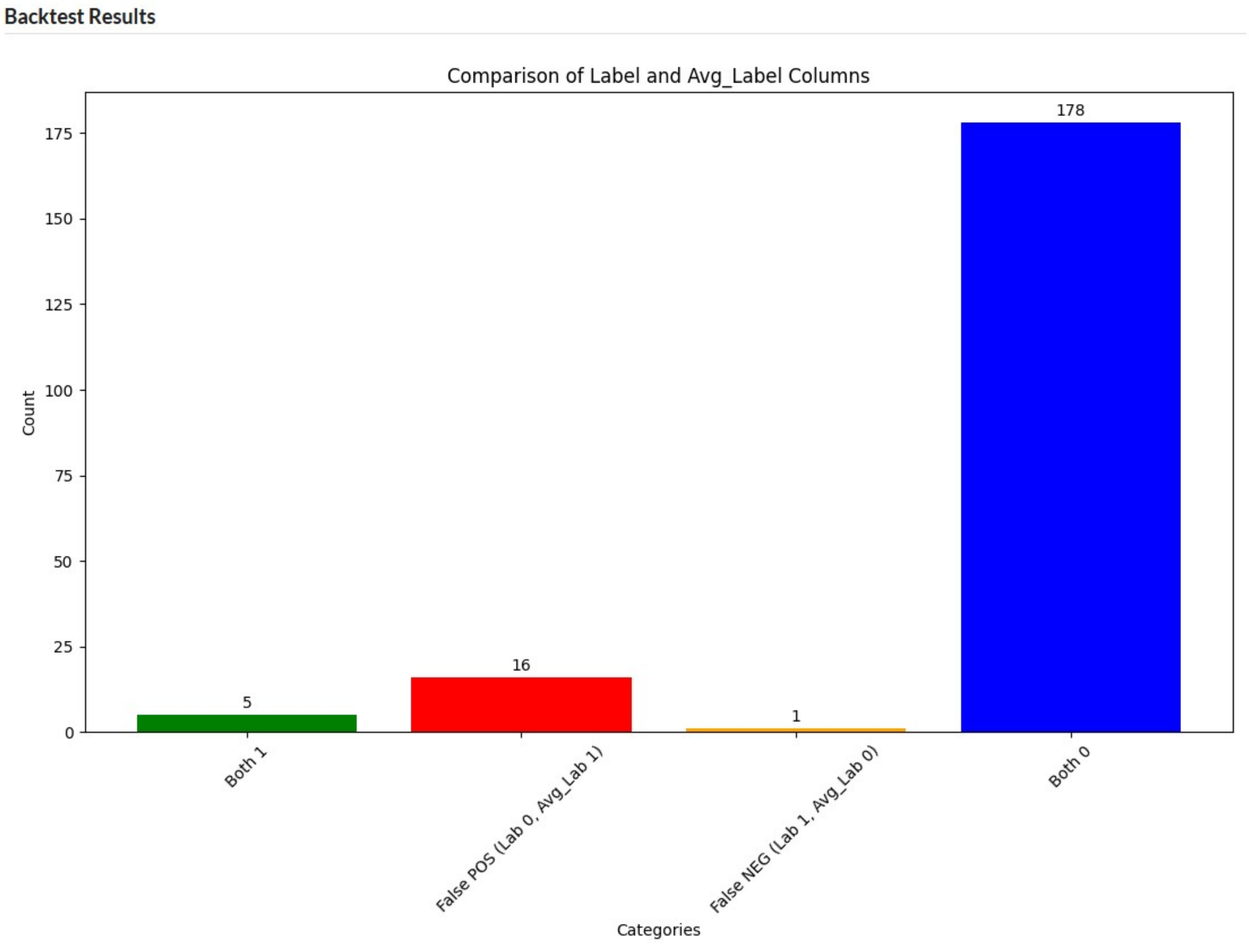


Figure 25: Comparison of label, and average label results.

The results in figure 25 above are not from the ML model, they are simply a way to discern how much the simplification step of averaging the high & low curves to get a single time series has effected the categorization. It can be deduced from the graph above that there were 6 x '1's using the original screener. The simplified categorization algorithm has captured 5 of them, the 6th is shown as a false negative. More concerning is that the simplification also caused 16 false positives to be generated. These are the worst category from a trading point of view because they represent failed trades.

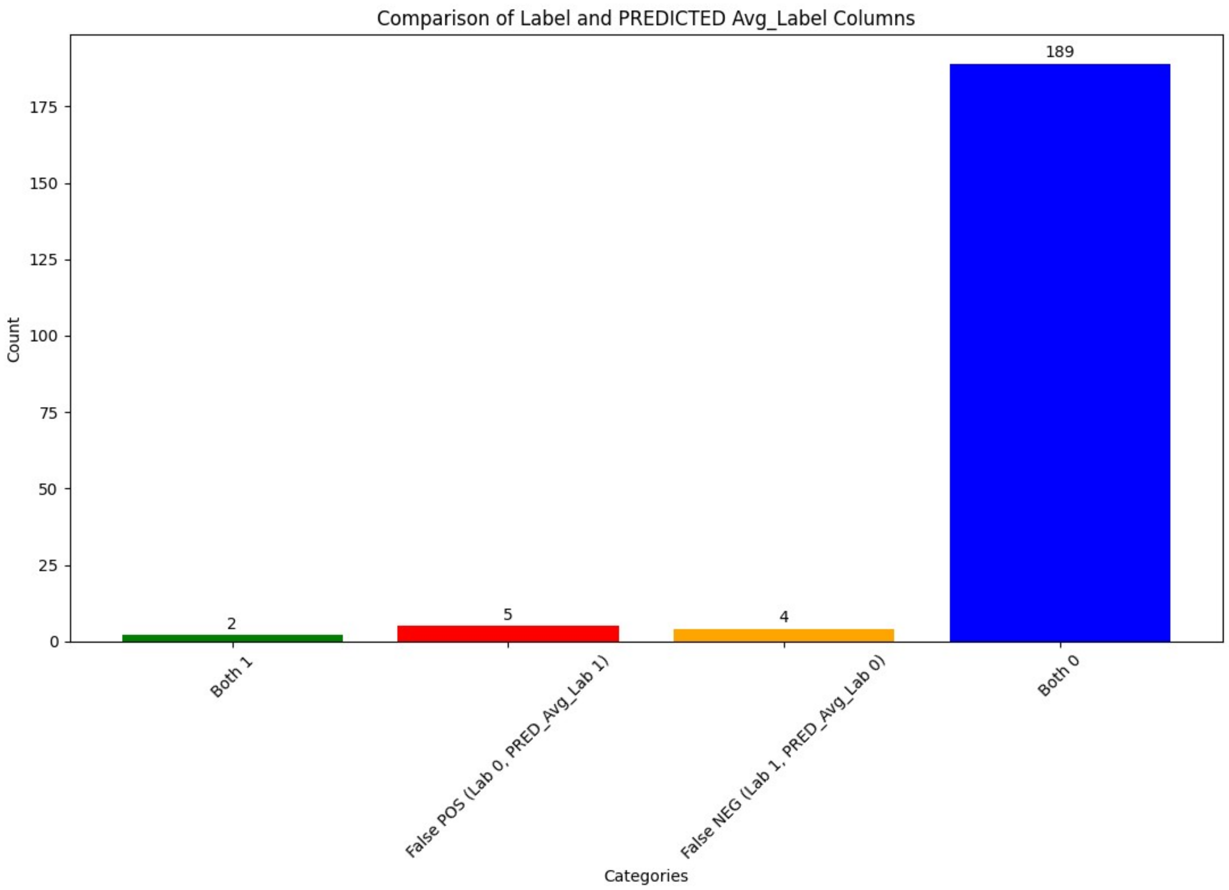


Figure 26: Comparison of original Label and High-Low-Mean ML Predicted Avg_Label results.

The results shown in figure 26 are the main event! They can be used to discern how good the ML model is at predicting the positive trading outcomes, and also how may false positives it picks up. In this set of results there are still 6 original '1' labels (6 good days to trade). The ML model identified two of them (Both 1), and got 4 wrong (False NEG) – this is a little worse than in figure 25, however, these values were predicted rather than calculated. A good thing is that the ML model got less false positives (5) than the average label results (16), again, comparing to figure 25.

The High Low Mean ML model has been tried out on several different stocks. The results vary, however, the better results usually occur when good training results are achieved, this makes sense.

Each stock is different, and the fit of the ML model has a significant impact on the results. Going forward it might be the case that the hyper parameters will require tuning for individual stocks.

4.0 Further work

- Implement the high-low ML model.
- A more in-depth sensitivity study on the hyper parameters do identify optimal values.
- Add indicators to the predicted events to screen out as many false positives as possible.
- Implement the real time data updating and prediction.
- Improve the workflow and user interface to make as friction-less as possible.
- Paper trade the updated model.