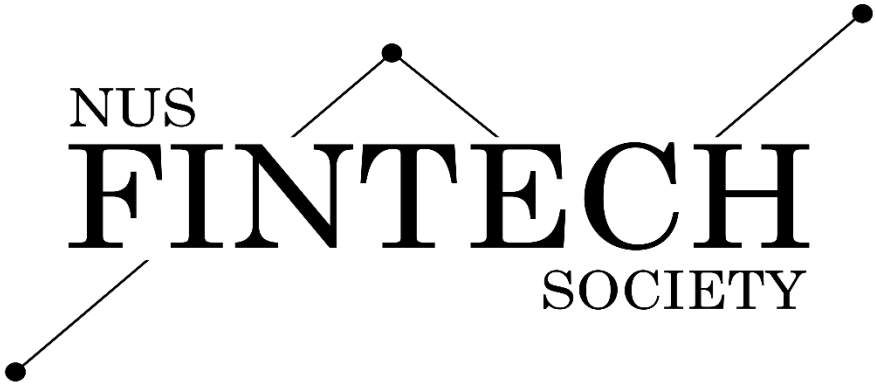


Recurrent Neural Net Trading For Financial Markets

Recurrent Neural Nets | Long-Short Term Memory NNs | XGBoost

Authors

Cai Shuhang | Jeremy Lim | Choon Kiat

NUS FINTECH SOCIETY
nusfintech@gmail.com

NUS FINTECH SOCIETY

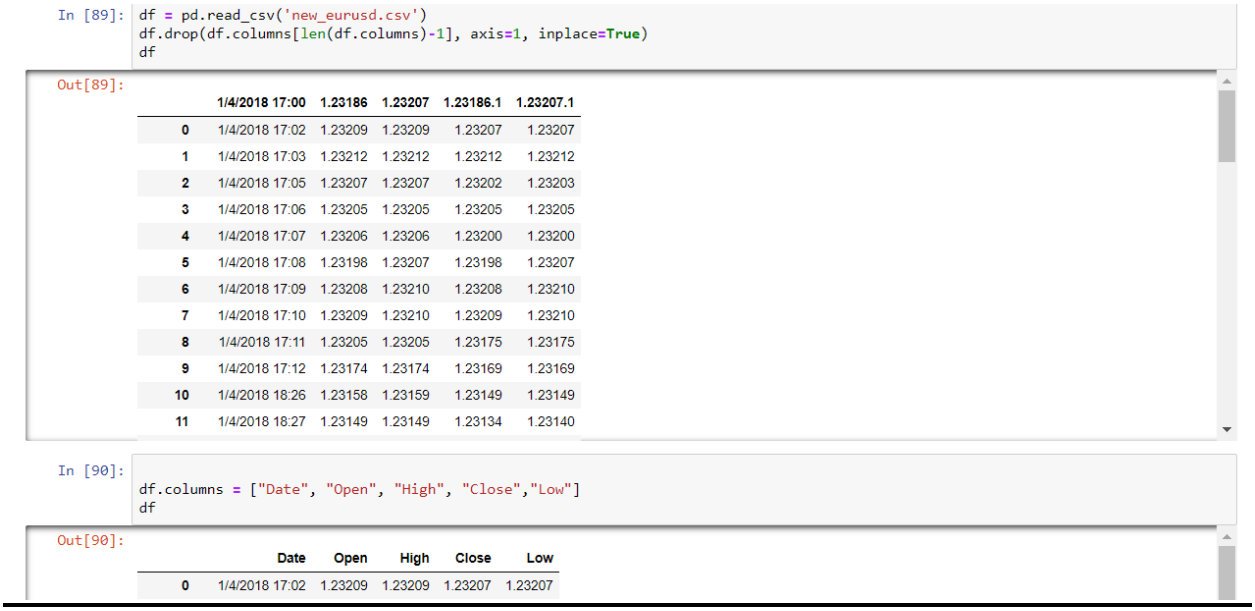
Content Page

Overview

This document summarises the data wrangling/ exploration processes, the performance indicators used and Risk Management done by the team. An instruction on how to run the model and understanding each part of the code in the model is also provided.

Data Wrangling/Exploration

We used a Jupyter notebook to visualise the EURUSD dataset and to do data exploration.



```
In [89]: df = pd.read_csv('new_eurusd.csv')
         df.drop(df.columns[len(df.columns)-1], axis=1, inplace=True)
         df
```

Out[89]:

	1/4/2018 17:00	1.23186	1.23207	1.23186.1	1.23207.1
0	1/4/2018 17:02	1.23209	1.23209	1.23207	1.23207
1	1/4/2018 17:03	1.23212	1.23212	1.23212	1.23212
2	1/4/2018 17:05	1.23207	1.23207	1.23202	1.23203
3	1/4/2018 17:06	1.23205	1.23205	1.23205	1.23205
4	1/4/2018 17:07	1.23206	1.23206	1.23200	1.23200
5	1/4/2018 17:08	1.23198	1.23207	1.23198	1.23207
6	1/4/2018 17:09	1.23208	1.23210	1.23208	1.23210
7	1/4/2018 17:10	1.23209	1.23210	1.23209	1.23210
8	1/4/2018 17:11	1.23205	1.23205	1.23175	1.23175
9	1/4/2018 17:12	1.23174	1.23174	1.23169	1.23169
10	1/4/2018 18:26	1.23158	1.23159	1.23149	1.23149
11	1/4/2018 18:27	1.23149	1.23149	1.23134	1.23140

```
In [90]: df.columns = ["Date", "Open", "High", "Close","Low"]
         df
```

Out[90]:

	Date	Open	High	Close	Low
0	1/4/2018 17:02	1.23209	1.23209	1.23207	1.23207

Figure 1:EURUSD Dataset

Here, we first converted the data into proper data frames as part of data cleaning.

```
In [93]:
df['hour'] = df.index.hour
df['day']  = df.index.weekday
df['week'] = df.index.week
df['avg_price'] = (df['Low'] + df['High'])/2
df['range']     = df['High'] - df['Low']
df['ohlc_price'] = (df['Low'] + df['High'] + df['Open'] + df['Close'])/4
df['oc_diff']    = df['Open'] - df['Close']

In [94]:
df
```

Out[94]:

Date	Open	High	Close	Low	hour	day	week	avg_price	range	ohlc_price	oc_diff
2018-01-04 17:02:00	1.23209	1.23209	1.23207	1.23207	17	3	1	1.232080	0.00002	1.232080	0.00002
2018-01-04 17:03:00	1.23212	1.23212	1.23212	1.23212	17	3	1	1.232120	0.00000	1.232120	0.00000
2018-01-04 17:05:00	1.23207	1.23207	1.23202	1.23203	17	3	1	1.232050	0.00004	1.232048	0.00005
2018-01-04 17:06:00	1.23205	1.23205	1.23205	1.23205	17	3	1	1.232050	0.00000	1.232050	0.00000
2018-01-04 17:07:00	1.23206	1.23206	1.23200	1.23200	17	3	1	1.232030	0.00006	1.232030	0.00006
2018-01-04 17:08:00	1.23198	1.23207	1.23198	1.23207	17	3	1	1.232070	0.00000	1.232025	0.00000
2018-01-04 17:09:00	1.23208	1.23210	1.23208	1.23210	17	3	1	1.232100	0.00000	1.232090	0.00000
2018-01-04 17:10:00	1.23209	1.23210	1.23209	1.23210	17	3	1	1.232100	0.00000	1.232095	0.00000
2018-01-04 17:11:00	1.23205	1.23205	1.23175	1.23175	17	3	1	1.231900	0.00030	1.231900	0.00030
2018-01-04 17:12:00	1.23174	1.23174	1.23169	1.23169	17	3	1	1.231715	0.00005	1.231715	0.00005
2018-01-04 18:26:00	1.23158	1.23159	1.23149	1.23149	18	3	1	1.231540	0.00010	1.231537	0.00009

Figure 2::EURUSD Dataset

Next, we created the columns for hour, day, week and range as attributes so that we can build technical indicators from these attributes later.

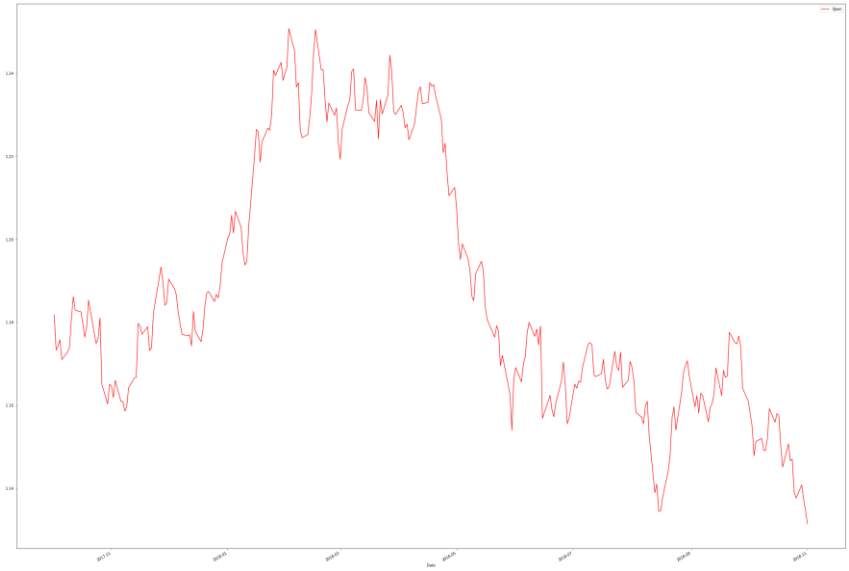


Figure 3: EURUSD price trend over 1 year period

We also plotted the price against time to understand the trend and volatility better. In general, the EUR is weakening against the USD. We can also see that there is high volatility between the two currencies.

Feature	Formula	Description
Average Price	(High + Low) / 2	Average price for the day
Range	High - Low	The trading range for the day
Open-High-Low-Close Price	(Open + High + Low + Close) / 4	Average of the OHLC price for the day
Open Close Price Difference	Open - Close	Difference between the open and close price for the day.
5-day Simple Moving Average of Close Price (SMA)	Sum(Close Price of past 5 days) / 5	Mean of the close price of the past 5 days
Relative Strength Index (RSI)	RSI(Close price,14 days)	RSI oscillates between zero and 100. RSI is considered overbought when above 70 and oversold when below 30. (Investopedia, n.d.)
Stop and Reverse (SAR	SAR(High,Low,acceleration =0.2, maximum=0.2)	SAR is a method to find potential reversals in the market price direction.
Average Directional Index (ADX)	ADX(High,Low,Close,14 days)	ADX is an indicator of trend strength in a series of prices of a financial instrument
Average True Range (ATR)	ATR(High,Low,Close,14)	ATR reflects the volatility of a price in absolute price levels.

Principal Component Analysis	PCA = PCA(n_components=1)	PCA is a method that brings together a measure of how each variable is associated with one another, the directions in which the data are dispersed as well as the relative importance of these different directions.
Intraday Percentage Change	% change$_i$ = (closePrice$_i$ - openPrice$_i$) / openPrice$_i$	The relative change in the price within the current day
Interday Percentage Change	% change$_i$ = (closePrice$_i$ - closePrice$_{i-1}$) / (closePrice$_{i-1}$)	The relative change in the price between day$_i$ and day$_{i-1}$

Figure 4: Feature Engineering

In addition to open, high, low, close price, attributes were constructed from the available data to build the features that comprise the training set. A diverse range of features such as seasonality features, lagged values and technical indicators were explored in the construction of the models to improve the classification capability of the model.

```
df['hour'] = df.index.hour
df['day']  = df.index.weekday
df['week'] = df.index.week
df['avg_price'] = (df['Low'] + df['High'])/2
df['range']     = df['High'] - df['Low']
df['ohlc_price'] = (df['Low'] + df['High'] + df['Open'] + df['Close'])/4
df['oc_diff']    = df['Open'] - df['Close']
#df['RSI'] = ta.RSI(np.array(df['Close']), timeperiod=14)
df['SMA'] = df['Close'].rolling(window=5).mean()
df['percentage_change'] = (df['Close'] - df['Open']) / df['Open']  # Percentage change in price for the day
df['cp_change'] = (df['Close'] - df['Close'].shift(1)) / (
df['Close'].shift(1))  # Change in closing price from the previous day

## to get RSI value
delta = df["Close"].diff()
window = 15
up_days = delta.copy()
up_days[delta<=0]=0.0
down_days = abs(delta.copy())
down_days[delta>0]=0.0
RS_up = up_days.rolling(window).mean()
RS_down = down_days.rolling(window).mean()
rsi= 100-100/(1+RS_up/RS_down)
df['RSI'] = rsi

ATR1 = abs (df['High'] - df['Low'])
ATR2 = abs (df['High'] -
df['Close'].shift())
ATR3 = abs (df['Low'] - df['Close'].shift())
ATR = pd.DataFrame(ATR1)
ATR['ATR2'] = ATR2
ATR['ATR3'] = ATR3
ATR = ATR.max(axis=1)
ATR = ATR.rolling(14).mean()
df['ATR'] = ATR
```

Figure 5: Feature Engineering in Code

Feature Selection

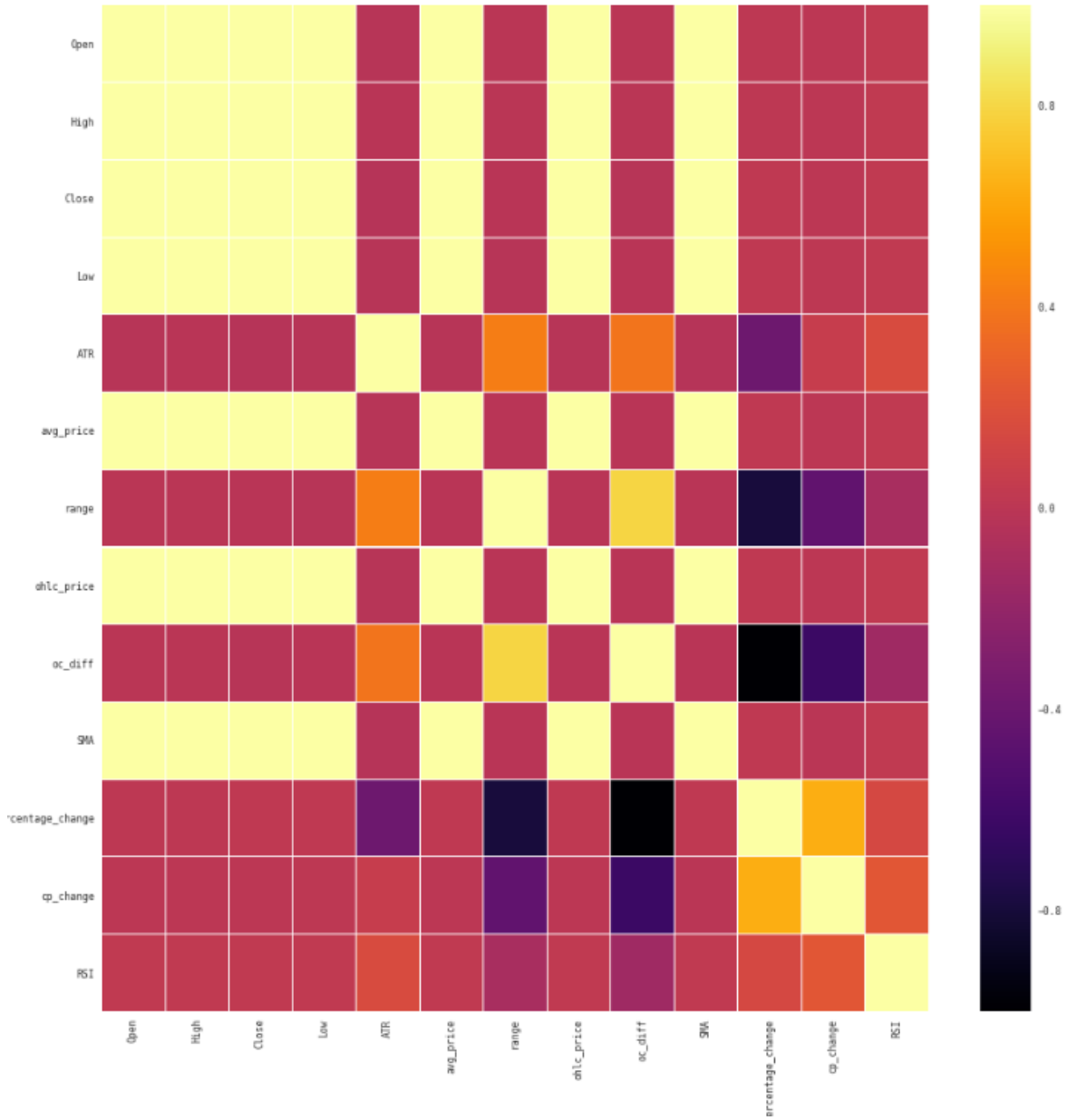
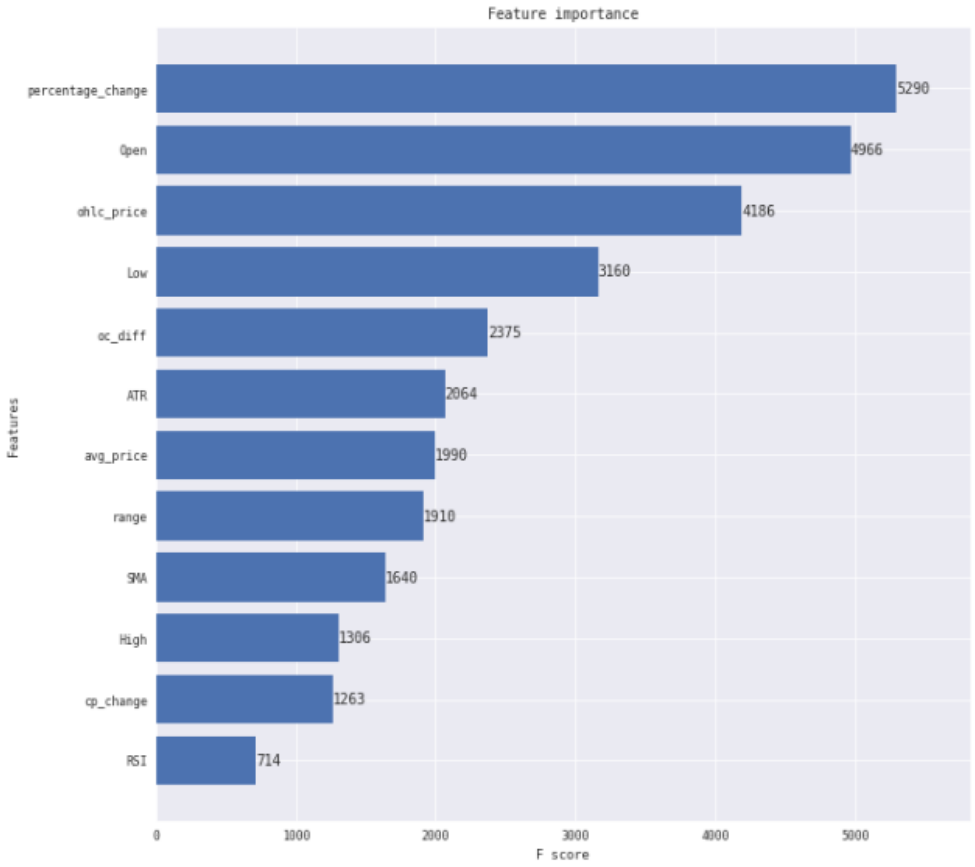


Figure 6: Correlation Matrix of Features

Here, we run a Pearson correlation plot for all the features to determine which features are correlated and important in our prediction. From the colour coding, we can see that features such as Simple Moving Average and the Open, High, Low prices has a high correlation with the ‘Close’ price.




```
]: import xgboost as xgb
xgb_params = {
    'eta': 0.05,
    'max_depth': 10,
    'subsample': 1.0,
    'colsample_bytree': 0.7,
    'objective': 'reg:linear',
    'eval_metric': 'rmse',
    'silent': 1
}
dtrain = xgb.DMatrix(trainX, trainY, feature_names=trainX.columns.values)
model = xgb.train(dict(xgb_params, silent=0), dtrain, num_boost_round=100)
remain_num = 99
```

```
fig, ax = plt.subplots(figsize=(10,10))
xgb.plot_importance(model, max_num_features=remain_num, height=0.8, ax=ax)
plt.show()
```

Figure 7: F-Score of Features

The F-score is calculated as the number of times a variable is selected for splitting, weighted by the squared improvement to the model as a result of each split, and averaged over all trees.

```
[54]: # Add PCA as a feature instead of for reducing the dimensionality. This improves the accuracy a bit.
from sklearn.decomposition import PCA

dataset = df.copy().values.astype('float32')
pca_features = df.columns.tolist()

pca = PCA(n_components=1)
df['pca'] = pca.fit_transform(dataset)
```

Figure 8: Principal Component Analysis

Principal Component Analysis was conducted to reduce dimensionality and improve the accuracy of the model.

```
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)  # Normalise the dataset to a common scale between 0 and 1

y_scaler = MinMaxScaler(feature_range=(0, 1))  # Normalise the dataset to a common scale between 0 and 1
t_y = df['close'].values.astype('float32')
t_y = np.reshape(t_y, (-1, 1))  # Reshape the output data
y_scaler = y_scaler.fit(t_y)  # Normalise the dataset to a common scale between 0 and 1

dataX, dataY = [], []

for i in range(len(dataset) - 10 - 1):  # 10 days lookback/windowing
    a = dataset[i:(i + 10)]
    dataX.append(a)
    dataY.append(dataset[i + 10])
```

Figure 9: Data Preprocessing(Scaling/Reshaping)

For our data preprocessing, we did some scaling and reshaping on our dataset to fit into our LSTM model. A rolling window of 10 days is set up for our features and the labels will be the actual closing price on the 11th day.

```
X, y = np.array(dataX), np.array(dataY)
y = y[:, target_index]

train_size = int(len(X) * 0.95)  # Test-Train split, 98% train and 5% test
trainX = X[:train_size]
trainY = y[:train_size]
testX = X[train_size:]
testY = y[train_size:]
```

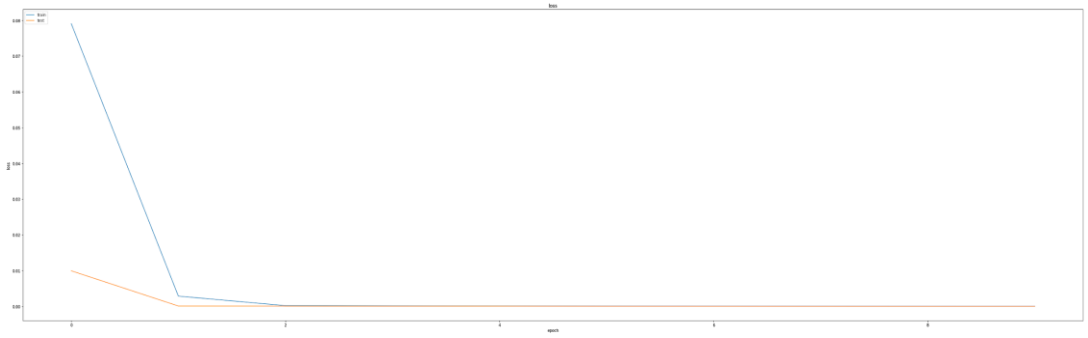
Figure 10: Test Train Split(Time Series Split)

The dataset is split into 95% train and 5% for testing.

Performance Measurements/Indicators

```
# Save the best weight during training.
from keras.callbacks import ModelCheckpoint
checkpoint = ModelCheckpoint("weights.best.hdf5", monitor='val_mean_squared_error', verbose=1, save_best_only=True, mode='min')

# Fit
callbacks_list = [checkpoint]
history = model.fit(trainX, trainY, epochs=200, batch_size=500, verbose=0, callbacks=callbacks_list, validation_split=0.1)
```

```
Epoch 00001: val_mean_squared_error improved from inf to 0.00033, saving model to weights.best.hdf5

Epoch 00002: val_mean_squared_error improved from 0.00033 to 0.00007, saving model to weights.best.hdf5

Epoch 00003: val_mean_squared_error improved from 0.00007 to 0.00004, saving model to weights.best.hdf5

Epoch 00004: val_mean_squared_error did not improve from 0.00004

Epoch 00005: val_mean_squared_error did not improve from 0.00004

Epoch 00006: val_mean_squared_error did not improve from 0.00004

Epoch 00007: val_mean_squared_error did not improve from 0.00004

Epoch 00008: val_mean_squared_error improved from 0.00004 to 0.00003, saving model to weights.best.hdf5

Epoch 00009: val mean squared error did not improve from 0.00003
```



		predicted	actual	diff	low	high
symbol	time					
EURUSD	2018-10-18	1.158362	1.155850	0.002512	1.149425	1.157795
	2018-10-19	1.158154	1.150035	0.008119	1.144915	1.152740
	2018-10-20	1.154455	1.145910	0.008545	1.143300	1.153475
	2018-10-22	1.151238	1.151385	-0.000147	1.149825	1.151980
	2018-10-23	1.153172	1.151120	0.002053	1.145220	1.155035
	2018-10-24	1.153576	1.145560	0.008016	1.143945	1.149385
	2018-10-25	1.150391	1.146835	0.003556	1.137905	1.147690
	2018-10-26	1.150297	1.141245	0.009052	1.135615	1.143275
	2018-10-27	1.147065	1.136850	0.010215	1.133570	1.142130
	2018-10-29	1.144308	1.140425	0.003883	1.138630	1.140405
	2018-10-30	1.145051	1.139195	0.005856	1.136060	1.141650
	2018-10-31	1.145212	1.138095	0.007117	1.133615	1.138795
	2018-11-01	1.144657	1.134215	0.010442	1.130215	1.136040
	2018-11-02	1.142835	1.134325	0.008510	1.133775	1.142420
	2018-11-03	1.142261	1.140895	0.001366	1.137220	1.145595
	2018-11-05	1.145162	1.138810	0.006352	1.138275	1.140440
	2018-11-06	1.145081	1.138480	0.006601	1.135375	1.142410
	2018-11-07	1.144804	1.139985	0.004819	1.139145	1.147315

```
2018-11-08 00:00:00 0.048690780997276306
2018-11-08 00:00:00 MSE4.4619977e-05
2018-11-08 00:00:00 Predicted price is 1.148163914680481
2018-11-08 00:00:00 Prev_price is 1.1399849653244019
2018-11-08 00:00:00 long
```

Figure 11: Model Optimization by MSE

Since our model predicts price, the team has decided to use the mean squared error (MSE) as a metric to evaluate the accuracy of the model. It is calculated by taking the average of the square of the difference between the predicted price and actual price. Since values closer to zero are better, we want to minimise the MSE.

We also conducted a simple grid search to achieve the best hyperparameters for constructing our model. During training, every epoch is being checked for its mean squared error(MSE) and the epoch with the lowest MSE will be saved as a checkpoint. This information served as a metric to tune the hyperparameters of the LSTM.

```
self.model = Sequential()  # Setting up of LSTM Model
self.model.add(LSTM(20, input_shape=(X.shape[1], X.shape[2]), return_sequences=True))
self.model.add(LSTM(10, return_sequences=True))
self.model.add(Dropout(0.2))
self.model.add(LSTM(10, return_sequences=False))
self.model.add(
    Dense(1, kernel_initializer='he_normal', activation='relu'))

self.model.compile(
    loss='mean_squared_error',
    optimizer='adam',
    metrics=['mae', 'mse'])

self.model.fit(
    trainX, trainY, epochs=300, verbose=0)
```

Figure 12: Model Configuration

To prevent overfitting, the team only added two LSTM layers with appropriate dropout rates. The dimension of the input is kept high as well but less than the number of features, which is a good standard practice to prevent overfitting the data.

Did you consider ensemble models to allow voting?

Unfortunately, the team did not consider ensemble models to allow voting due to limitations of QuantConnect. Since we were on a free account, our run time was limited to 10 minutes and the RAM allocated is also limited to 512MB. The entire process of training and testing our LSTM model took up most of our available RAM.

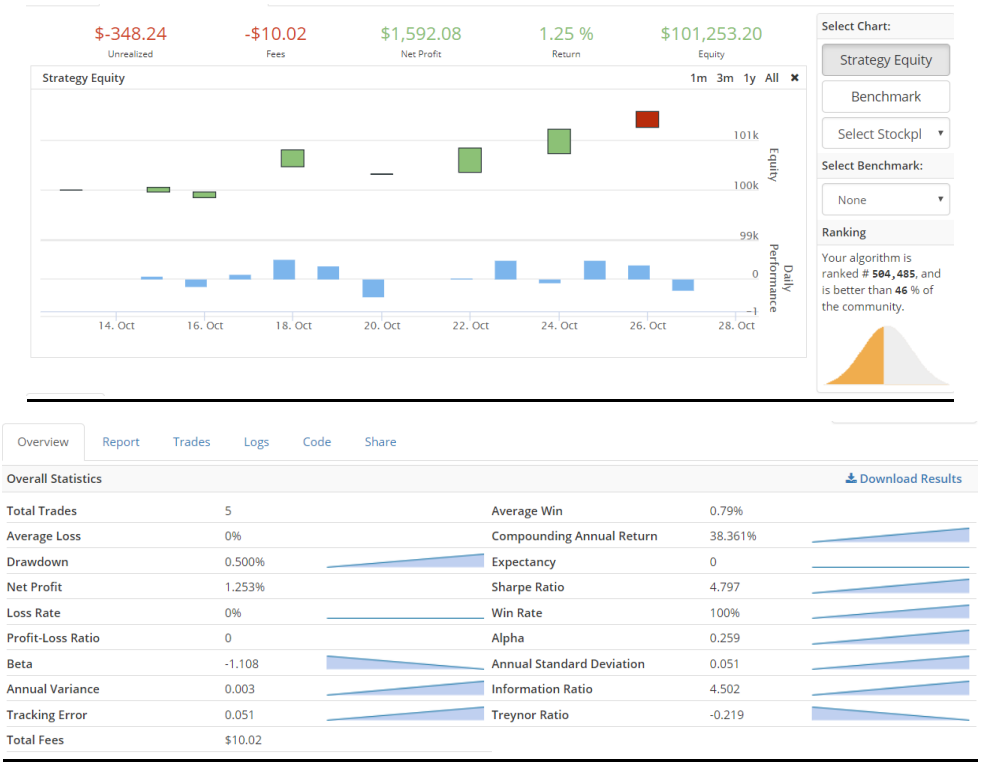


Figure 13: Model Results

However, we did consider the use of other machine learning model as well. Mainly the Support Vector Machine and the Logistic Regression. The other model's results can be found in the appendix and the code is attached in the folder as well for reference.

We measured the performance of each model on its percentage return and net profit. Overall statistics of each model such as the Average Win Percentage, Compounding Annual Return, Sharpe Ratio and Loss rate were used as well. Sharpe ratio is a measure of excess portfolio return over the risk-free rate relative to its standard deviation. It shows how much additional return an investor earns by taking additional risk.

We chose the best model the LSTM as it had successfully managed to make a return of 1.25% with a net profit of $1,592 in the two weeks time period. It also has a positive average win of 0.79% with a compounding annual return of 38.4%. The Sharpe ratio is also high at 4.797 which suggests a high positive return per unit of deviation in our currency pair.

```
# Baby the model a bit
# Load the weight that worked the best
model.load_weights("weights.best.hdf5")

# Train again with decaying learning rate
from keras.callbacks import LearningRateScheduler
import keras.backend as K

def scheduler(epoch):
    if epoch%2==0 and epoch!=0:
        lr = K.get_value(model.optimizer.lr)
        K.set_value(model.optimizer.lr, lr*.9)
        print("lr changed to {}".format(lr*.9))
    return K.get_value(model.optimizer.lr)
lr_decay = LearningRateScheduler(scheduler)

callbacks_list = [checkpoint, lr_decay]
history = model.fit(trainX, trainY, epochs=int(epoch/3), batch_size=500, verbose=0, callbacks=callbacks_list, validation_split=0.
```

Figure 14: Retraining model via callback

In our Jupyter notebook, the Checkback function in keras is used to save the optimal weights used to checkpoint the weights of the model. As a result, the weights of the model are only saved and updated whenever there is an decrease in the MSE of the model.

In our actual model, the model is retrained every time it enters a position. This is done in the rebalance function which is called every six hours. Since market conditions rarely stay the same, it is important to consistently retrain our model to ensure that it recognises the current trend so as to make an accurate prediction.

Risk Management

How did your model take action?

```
pred_data = np.array([dataset[-10:]])  # Prepare data for prediction of next day close price
output = self.model.predict(pred_data)
output = y_scaler.inverse_transform(output)
predictions2 = pd.DataFrame()
predictions2['predicted'] = pd.Series(
    np.reshape(output, (output.shape[0])))
predictions2 = predictions2.astype(float)
output = predictions2['predicted'][0]  # Get predicted value

self.Debug("Predicted price is " + str(output))

prev_price = predictions['actual'][-1]  # Get previous day's value
self.Debug("Prev_price is " + str(prev_price))

# Buy/Sell Execution conditions

if output > prev_price and self.currency not in self.long_list and self.currency not in self.short_list:
    self.SetHoldings(self.currency, 1)
    self.long_list.append(self.currency)
    self.Debug("long")

if output < prev_price and self.currency not in self.long_list and self.currency not in self.short_list:
    self.SetHoldings(self.currency, -1)
    self.short_list.append(self.currency)
    self.Debug("short")
```

Figure 15: Prediction and Buy/Sell Conditions

After training and optimizing our model, we use the model to predict the next day's closing price.

```
self.Schedule.On(
    self.DateRules.Every(DayOfWeek.Monday, DayOfWeek.Tuesday,  # Schedule function to enter a position
                         DayOfWeek.Wednesday, DayOfWeek.Thursday,
                         DayOfWeek.Friday),
    self.TimeRules.Every(TimeSpan.FromMinutes(360)),
    Action(self.Rebalance))

self.Schedule.On(  # Schedule function to exit a position
    self.DateRules.EveryDay(self.symbol),
    self.TimeRules.Every(TimeSpan.FromMinutes(10)),
    Action(self.Rebalance2))
```

Figure 16: Schedule Functions

Two schedule functions are set up to perform trades every day. The first schedule function will be for entering a position. It will be executed every weekday at an interval of six hours to check whether the price is favourable to enter a trade.

The second schedule function is for exiting a trade and will be executed on every trading day of the forex market at an interval of 10 minutes. This ensures that we do not miss the best opportunity to exit the market once the price falls into our Stop Loss-Take Profit range.

For the buy/sell conditions, if the predicted price is higher than yesterday's close price,

we will take a long position, if the predicted price is going to be lower we take a short position. The holdings for the currency pair is set to its respective value and the symbol is added to its respective position list for tracking.

Risk-reward ratio

The decision to buy and sell currency pair is determined by the Average True Range, an indicator that reflects the volatility of the currency pair based on previous days data. The team has decided to go for a conservative risk-reward ratio of 1:1 for our trade in light of the volatility seen in recent weeks as shown in Figure 3. If we had chosen a larger risk/reward ratio such as 1:3 or 1:5, while our losses will remain the same, it can result in fewer trades as the threshold is too high. On the contrary, it can also mean that the model will miss out on smaller profits to accumulate assets.

Below, we show how we incorporated our risk-reward ratio into the model in the form of creating the ATR indicator and in the Stop Loss-Take-Profit conditions.

```
def OnData(self, data):
    if self.IsWarmingUp:
        return

    df1 = self.hist_data  # Calculate and update new atr everyday
    df1 = df1[['open', 'high', 'low', 'close']]
    self.atr = ta.ATR(df1['high'], df1['low'], df1['close'], timeperiod=14)
    self.Debug("atr" + str(self.atr[-1]))
```

Figure 17: ATR Calculation

ATRs are better than using a fixed percentage because they change based on the characteristics of the stock being traded, recognizing that volatility varies across issues and market conditions. It is more reflective of the volatility of the asset and maximises the potential profit to be gain. Here, we chose to use the ATR of the past 14 days.

```
def Rebalance2(self):
    # curr_price = self.Securities["EURUSD"].Price  # Get current market price
    curr_price = pd.DataFrame(
        self.History([self.currency], 120,
                     Resolution.Hour))
    curr_price = curr_price['close'][-1]


    # Exit execution Conditions
    df1 = self.hist_data  # Calculate and update new atr everyday
    df1 = df1[['open', 'high', 'low', 'close']]
    self.atr = ta.ATR(df1['high'], df1['low'], df1['close'], timeperiod=14)
    self.Debug("atr " + str(self.atr[-1]))
    self.Debug("curr_price " + str(curr_price))

    if self.currency in self.long_list:
        cost_basis = self.Portfolio[self.currency].AveragePrice  # To calculate the price of our FOREX currency pair

        if ((curr_price <= float(cost_basis) - float(self.atr[-1]))  # Stop loss-Take Profit conditions
                or (curr_price >=  # Use of Average True Range to determine price to sell
                    float(self.atr[-1] * 1) + float(cost_basis))):
            self.Debug("price is: " + str(curr_price))
            self.SetHoldings(self.currency, 0)
            self.long_list.remove(self.currency)


    if self.currency in self.short_list:
        cost_basis = self.Portfolio[self.currency].AveragePrice

        if ((curr_price <= float(cost_basis) - float(self.atr[-1] * 1))  # Stop loss-Take Profit conditions
                or  # Use of Average True Range to determine price to buy back
                (curr_price >= float(self.atr[-1]) + float(cost_basis))):
            self.Debug("price is: " + str(curr_price))
            self.SetHoldings(self.currency, 0)
            self.short_list.remove(self.currency)
```

Figure 18: Exit Conditions

The Rebalance2 function looks at the current market price and decides based on the Stop Loss-Take Profit (SL-TP) conditions to exit the market.

The ATR indicator as described above is used to determine our stop loss and profit margin and is computed and updated here again.

If we are in a Long position: Our stop-loss threshold is when the current price falls below the price we bought the currency pair at minus the ATR range of the currency pair. For take profit, we would sell the currency pair when the current price is higher than the price we bought at by one time of the ATR of the currency pair. The ratio of the ATR is determined by trial and error to maximise profit.

On the other hand, if we are in a short position, our stop loss threshold is when the current price of the currency pair rises above the price we bought at by the ATR value. For take profit, when the current price falls below the price we bought at by one time of the ATR value, we would then buy the currency pair. Hence, giving us a risk-reward ratio of 1:1.

Instructions On Running Model

```
import numpy as np

from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, Bidirectional

from sklearn.preprocessing import MinMaxScaler

from sklearn.decomposition import PCA

from sklearn.metrics import mean_squared_error

import pandas as pd
import talib as ta
```

Figure 19: Importing Libraries

Firstly, we needed to import the necessary libraries required to run the model. As we are using neural networks, we would need to import the relevant keras libraries.

The sklearn library is used for data preprocessing and feature engineering. On top of that, we used an external library, Talib, to perform technical analysis of the data.

```
def Initialize(self):

    self.SetStartDate(2018, 10, 15)  # Set Start Date
    self.SetEndDate(2018, 10, 26)  # Set End Date
    self.SetCash(100000)  # Set Strategy Cash
    self.currency = "EURUSD"  # Set currency pair
    self.symbol = self.AddForex(self.currency, Resolution.Daily).Symbol
    self.SetBrokerageModel(BrokerageName.InteractiveBrokersBrokerage,
                           AccountType.Margin)  # Set Brokerage Model
    self.SetWarmUp(365)

    self.hist_data = pd.DataFrame(
        self.History([self.currency], 365,
                     Resolution.Daily))  # Asking for past 1 year of historical data
    self.long_list = []  # To track short/long position
    self.short_list = []
    self.model = Sequential()
    self.atr = [0]  # Track current average true range
```

Figure 20: Initialise Model

We initialise our model by setting the conditions for backtesting as well as when it goes live. We have set the start and end date of our backtest to be from 15th October to 26th October. This is because we want to simulate the actual trading time period which is around two weeks. We also want to evaluate the performance of our model in the most recent market conditions.

The initial capital is set to one hundred thousand USD and we add the currency pair of our choice the EURUSD as well as a brokerage to model the trading fees.

Self.history is used to retrieve price data from the past 365 days to train our LSTM model. A long and short list is also initialised to track our current positions. We then initialised the keras sequential model and the average true range (our Stop-Loss Take Profit) indicator.

```
def Rebalance(self):
    self.hist_data = pd.DataFrame(
        self.History([self.currency], 365,
                     Resolution.Daily))  # Updating historical data for training

    if not self.hist_data.empty and self.currency not in self.long_list and self.currency not in self.short_list:
        df = self.hist_data

        df = df[['open', 'high', 'low', 'close']]  # Features to be included
        df['avg_price'] = (df['low'] + df['high']) / 2  # Average Price
        df['range'] = df['high'] - df['low']  # Price Range
        df['ohlc_price'] = (  # Open-High-Low-Close Average Price
                                df['low'] + df['high'] + df['open'] + df['close']) / 4
        df['oc_diff'] = df['open'] - df['close']  # Open Close Difference
        df['percentage_change'] = (df['close'] - df['open']) / df['open']  # Percentage change in price for the day
        df['cp_change'] = (df['close'] - df['close'].shift(1)) / (
            df['close'].shift(1))  # Change in closing price from the previous day
        df['SMA'] = df['close'].rolling(window=5).mean()  # Simple Moving Average of 5 days
        df.dropna(inplace=True)

        dataset = df.copy().values.astype('float32')
        pca_features = df.columns.tolist()

        pca = PCA(n_components=1)  # Price Component Analysis
        df['pca'] = pca.fit_transform(dataset)
```

Figure 21: Rebalance Function

In the first Rebalance function for entering a position, we engineered some of the features to be used in our dataset. This includes the daily Open, High, Low, Close price and also the Average Price(avg_price) , Range, Open-High-Low-Close Price (ohlc-price), Open-Close Difference(oc-diff), the intraday percentage change (percentage change) in the opening and closing price and interday percentage change in close price as well as a Simple Moving Average (SMA) of 5 days. Principal component analysis is also done on the features. For a full explanation of the features, kindly refer to the main report attached.

```
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)  # Normalise the dataset to a common scale between 0 and 1

y_scaler = MinMaxScaler(feature_range=(0, 1))  # Normalise the dataset to a common scale between 0 and 1
t_y = df['close'].values.astype('float32')
t_y = np.reshape(t_y, (-1, 1))  # Reshape the output data
y_scaler = y_scaler.fit(t_y)  # Normalise the dataset to a common scale between 0 and 1

dataX, dataY = [], []

for i in range(len(dataset) - 10 - 1):  # 10 days lookback/windowing
    a = dataset[i:(i + 10)]
    dataX.append(a)
    dataY.append(dataset[i + 10])
```

Figure 22: Data Preprocessing(Scaling/Reshaping)

For our data preprocessing, we did some scaling and reshaping on our dataset to fit into our LSTM model. A rolling window of 10 days is set up for our features and the labels will be the actual closing price on the 11th day.

```
X, y = np.array(dataX), np.array(dataY)
y = y[:, target_index]

train_size = int(len(X) * 0.95)  # Test-Train split, 98% train and 5% test
trainX = X[:train_size]
trainY = y[:train_size]
testX = X[train_size:]
testY = y[train_size:]
```

Figure 23: Test Train Split(Time Series Split)

The dataset is split into 95% train and 5% for testing.

```
self.model = Sequential()  # Setting up of LSTM Model
self.model.add(LSTM(20, input_shape=(X.shape[1], X.shape[2]), return_sequences=True))
self.model.add(LSTM(10, return_sequences=True))
self.model.add(Dropout(0.2))
self.model.add(LSTM(10, return_sequences=False))
self.model.add(
    Dense(1, kernel_initializer='he_normal', activation='relu'))

self.model.compile(
    loss='mean_squared_error',
    optimizer='adam',
    metrics=['mae', 'mse'])

self.model.fit(
    trainX, trainY, epochs=300, verbose=0)
```

Figure 24: Model Configuration

Next, we added layers and dropout to our LSTM model. These are the optimal hyperparameters after some trial and error.

The model is compiled with our loss function being mean squared error (MSE) and optimizer ‘adam’.

The model is then fitted with the training data.

```
pred = self.model.predict(testX)  # Predict on test data
pred = y_scaler.inverse_transform(pred)
close = y_scaler.inverse_transform(
    np.reshape(testY, (testY.shape[0], 1)))
predictions = pd.DataFrame()  # Create dataframe for evaluation of prediction on test data
predictions['predicted'] = pd.Series(
    np.reshape(pred, (pred.shape[0])))
predictions['actual'] = pd.Series(
    np.reshape(close, (close.shape[0])))
predictions = predictions.astype(float)
predictions['diff'] = predictions['predicted'] - predictions[
    'actual']  # Calculate difference between predicted and actual value

p = df[-pred.shape[0]:].copy()
predictions.index = p.index
predictions = predictions.astype(float)
predictions = predictions.merge(
    p[['low', 'high']], right_index=True,
    left_index=True)  # Add in high low price to see whether predicted value falls between this range
self.Debug(predictions)

trainscore = mean_squared_error(close, pred)
self.Debug('MSE' + str(trainscore))
```

```
                     predicted    actual      diff       low      high
symbol time
EURUSD 2018-10-18   1.158362  1.155850  0.002512  1.149425  1.157795
       2018-10-19   1.158154  1.150035  0.008119  1.144915  1.152740
       2018-10-20   1.154455  1.145910  0.008545  1.143300  1.153475
       2018-10-22   1.151238  1.151385 -0.000147  1.149825  1.151980
       2018-10-23   1.153172  1.151120  0.002053  1.145220  1.155035
       2018-10-24   1.153576  1.145560  0.008016  1.143945  1.149385
       2018-10-25   1.150391  1.146835  0.003556  1.137905  1.147690
       2018-10-26   1.150297  1.141245  0.009052  1.135615  1.143275
       2018-10-27   1.147065  1.136850  0.010215  1.133570  1.142130
       2018-10-29   1.144308  1.140425  0.003883  1.138630  1.140405
       2018-10-30   1.145051  1.139195  0.005856  1.136060  1.141650
       2018-10-31   1.145212  1.138095  0.007117  1.133615  1.138795
       2018-11-01   1.144657  1.134215  0.010442  1.130215  1.136040
       2018-11-02   1.142835  1.134325  0.008510  1.133775  1.142420
       2018-11-03   1.142261  1.140895  0.001366  1.137220  1.145595
       2018-11-05   1.145162  1.138810  0.006352  1.138275  1.140440
       2018-11-06   1.145081  1.138480  0.006601  1.135375  1.142410
       2018-11-07   1.144804  1.139985  0.004819  1.139145  1.147315
2018-11-08 00:00:00 0.048690780997276306
2018-11-08 00:00:00 MSE4.4619977e-05
2018-11-08 00:00:00 Predicted price is 1.148163914680481
2018-11-08 00:00:00 Prev_price is 1.1399849653244019
2018-11-08 00:00:00 long
```

Figure 25: Presentation of Training Results(Logs)

Here we convert the training results into a table for future evaluation.

The table can be accessed in the logs and we can use this information to optimise our model. Our aim is to minimize the difference in the actual and predicted price.

Conclusion

This report has covered the instructions to run the model, data wrangling and exploration processes and the performance indicators used to improve and evaluate our model. Detailed explanation and methodology of the model can be found in the main report.

Appendix

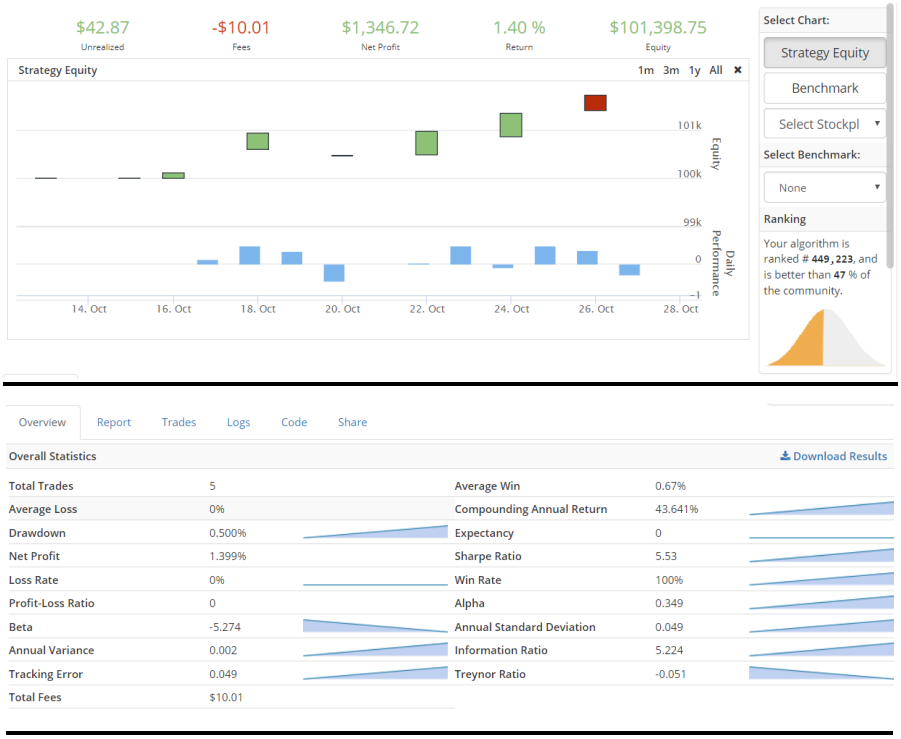
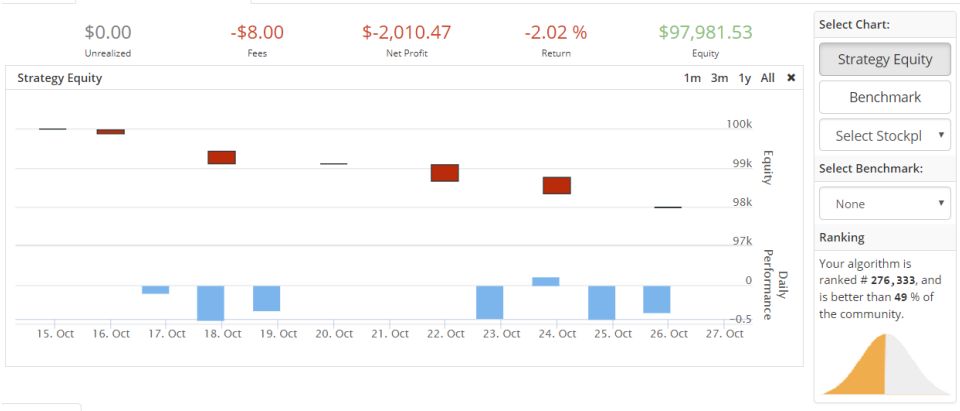
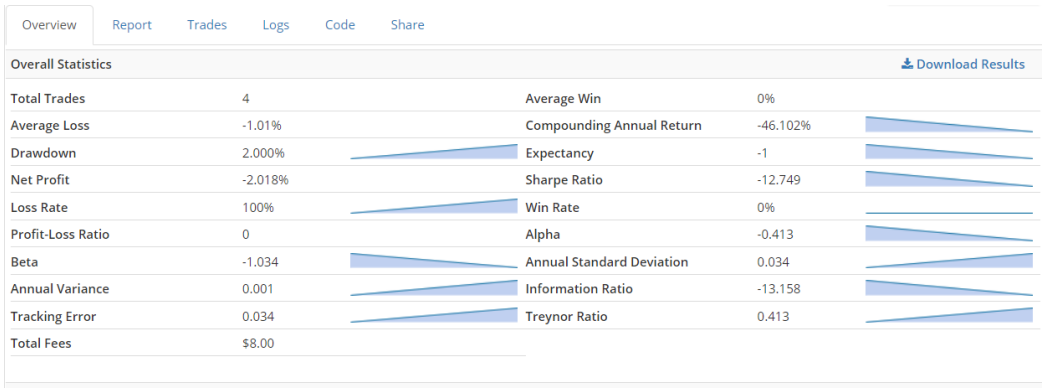


Figure 22: SVM results





Overview | Report | Trades | Logs | Code | Share

Overall Statistics — Download Results

Total Trades	4	Average Win	0%
Average Loss	-1.01%	Compounding Annual Return	-46.102%
Drawdown	2.000%	Expectancy	-1
Net Profit	-2.018%	Sharpe Ratio	-12.749
Loss Rate	100%	Win Rate	0%
Profit-Loss Ratio	0	Alpha	-0.413
Beta	-1.034	Annual Standard Deviation	0.034
Annual Variance	0.001	Information Ratio	-13.158
Tracking Error	0.034	Treynor Ratio	0.413
Total Fees	$8.00		

Figure 5: Backtest for Logistic Regression