National University Singapore
Quantitative Finance

Date: 11th April, 2024

Title: Trading Strategies Research

Course: QF2103

Group 11

Team Members:
1. Albert Ariel Widiaatmaja (A0237848M)
2. Nicholas Ng Heok Ping (A0233741L)
3. Akhil Choraria (A0290090L)
4. Stevan Gerard Gunawan (A0244118L)
5. Hamish Burns (A0278303E)

TABLE OF CONTENT:

1. Member Roles and Responsibilities..... 3

2. Preliminary Data Analysis and Data Collection..... 5

2.1 Preliminary Data Analysis. 5

2.2 Data Collection: Financial Indicators. 6

3. Trading Strategy 1: Stepwise Regression 11

3.1 Overview of Strategy Logic. 11

3.2 Preliminary attempt of predicting future log return using lag data..... 12

3.3 Predicting Future Price Using Financial Indicator..... 14

3.3.1 Data Preparation..... 14

3.3.2 Model Execution..... 14

3.4 SMA Trading Strategy Using Predicted Prices 16

3.5 Results of Backtesting and Performance Metrics 16

3.6 Optimizing the SMA Periods..... 19

3.7 Potential Challenges and Risk of Stepwise Regression. 22

3.8 Future Improvements..... 22

4. Trading Strategy 2: Stock Price Direction Prediction using Support Vector Machine .. 24

4.1 Overview of Strategy Logic..... 24

4.2 Machine Learning Model Training and Optimization 24

4.2.1 Feature Engineering 1: Feature Creation 24

4.2.2 Data Optimization: Normalization..... 25

4.2.3 Data Split: Training and Testing Data 26

4.2.4 Machine Learning Model Selection..... 26

4.2.5 Machine Learning Model Optimization using Hyperparameter Tuning 27

4.2.6 Feature Engineering 2: Feature Selection 28

4.3 Python Code 30

4.4 Benchmark Statistics by Backtesting 31

4.5 Potential Risks and Challenges 32

4.6 Future Improvement 33

4.7 Miscellaneous 34

References. 35

1. Member Roles and Responsibilities

Breakdown Overview:

- Preliminary Data Analysis and Data Collection
    - Albert Ariel Widiaatmaja (A0237848M)
- Trading Strategy 1: Stepwise Regression
    - Nicholas Ng Heok Ping (A0233741L)
    - Choraria Akhil (A0290090L)
    - Hamish Burns (A0278303E)
- Trading Strategy 2: Stock Price Direction Prediction using Support Vector Machine
    - Albert Ariel Widiaatmaja (A0237848M)
    - Stevan Gerard Gunawan (A0244118L)

Name	Detailed Roles and Responsibilities
Albert Ariel Widiaatmaja (A0237848M)	• Preliminary Data Analysis and Data Collection • Machine Learning Model Training and Optimization: Model Selection, Feature Engineering, Hyperparameter Tuning • Trade Simulation and Return Calculation • Report and Documentation • Slides

Stevan Gerard Gunawan (A0244118L)	• Machine Learning Model Training and Optimization: Model Selection, Feature Engineering, Hyperparameter Tuning • Trade Simulation and Return Calculation • Testing • Report and Documentation • Graphs for Reports • Slides
Nicholas Ng Heok Ping (A0233741L)	• Trade simulation and return calculation • Optimisation • Report and documentation
Choraria Akhil (A0290090L)	• Preliminary application of Stepwise Regression • Report and documentation • Slides
Hamish Burns (A0278303E)	• Stepwise prediction model • Data preparation • Report and documentation • Slides

2. Preliminary Data Analysis and Data Collection

    a. Preliminary Data Analysis

The initial dataset, sourced from the file "tr_eikon_eod_data.csv," comprises daily end-of-day stock prices of Apple, Microsoft, Intel, Amazon, and Goldman Sachs as shown in Figure 1 below. Our aim is to create prediction for future stock price movements, which will be utilized in our trading strategies.

In this preliminary data analysis, we aim to investigate the correlation strength between current stock price data and future stock price movement. In this analysis, we use the movement direction of Apple stock price to indicate future stock price movement, with 1 indicating a rise in tomorrow's price and -1 indicating a fall in tomorrow's price relative to today's price, as shown by the direction column in Figure 1 below.

Date	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N	SPY	direction
2010-01-04	30.572827	30.950	20.88	133.90	173.08	113.33	1
2010-01-05	30.625684	30.960	20.87	134.69	176.14	113.63	-1
2010-01-06	30.138541	30.770	20.80	132.25	174.26	113.71	-1
2010-01-07	30.082827	30.452	20.60	130.00	177.67	114.19	1
2010-01-08	30.282827	30.660	20.83	133.52	174.31	114.57	-1
...	...	...	...	...	...	...	...
2018-06-25	182.170000	98.390	50.71	1663.15	221.54	271.00	1
2018-06-26	184.430000	99.080	49.67	1691.09	221.58	271.60	-1
2018-06-27	184.160000	97.540	48.76	1660.51	220.18	269.35	1
2018-06-28	185.500000	98.630	49.25	1701.45	223.42	270.89	-1
2018-06-29	185.110000	98.610	49.71	1699.80	220.57	271.28	-1

2138 rows × 7 columns

Figure 1 Initial Dataset from tr_eikon_eod_data.csv and Stock Price Direction of Apple

Upon conducting this analysis, it was observed that the correlation between the given features and the direction was relatively weak, with correlation coefficients consistently below 0.04, as shown in Figure 2 below. This suggests the necessity of identifying more effective features to enhance prediction accuracy.

```
AAPL.O      -0.036055
MSFT.O      -0.016139
INTC.O      -0.021609
AMZN.O      -0.021744
GS.N        -0.009771
SPY         -0.021940
direction    1.000000
Name: direction, dtype: float64
```

Figure 2 Correlation between Direction of Apple Stock Price and Stock Prices of Apple, Microsoft, Intel, Amazon, and Goldman Sachs

Therefore, we gathered more financial indicators to enhance the predictive power of our models.

b. Data Collection: Financial Indicators

We utilized the AlphaVantage API (https://www.alphavantage.co/documentation/) to collect a variety of financial indicators, using a Python script, as illustrated in Figure 3 below.

Date	AAPL.O	SlowK	SlowD	RSI	ADX	CCI	Aroon Down	Aroon Up	OBV	Chaikin A/D	SMA	EMA
2010-04-01	30.572827	78.7484	82.8722	75.0702	38.0954	96.8893	42.8571	100.0000	8.816356e+10	5.524930e+10	6.9552	6.9908
2010-06-01	30.138541	89.5060	82.1665	57.5986	35.3551	151.4643	57.1429	100.0000	8.681483e+10	6.043320e+10	7.5207	7.6163
2010-07-01	30.082827	11.7287	14.7644	38.9499	24.7259	-152.8759	100.0000	42.8571	8.390127e+10	5.897666e+10	8.1018	7.9595
2010-11-01	30.015684	26.0805	26.1764	50.8046	19.7333	-103.8405	0.0000	28.5714	9.091934e+10	6.372971e+10	9.2871	9.2333
2010-12-01	29.674256	71.5220	76.9725	59.2894	18.8722	97.8421	35.7143	0.0000	9.076772e+10	6.438059e+10	9.3769	9.4407
...	...	...	...	...	...	...	...	...	...	...	...	...
2018-06-25	182.170000	24.7765	28.7326	33.5305	28.2742	-132.0057	100.0000	14.2857	9.108196e+10	7.277260e+10	44.7743	44.5714
2018-06-26	184.430000	30.9769	28.9171	32.5318	28.2014	-70.9622	92.8571	7.1429	9.118024e+10	7.276743e+10	44.5744	44.3849
2018-06-27	184.160000	37.4891	31.0808	42.8627	27.4502	-43.5508	85.7143	0.0000	9.107910e+10	7.267438e+10	44.4033	44.3292
2018-06-28	185.500000	55.4647	41.3102	39.4747	26.9167	-30.3030	78.5714	21.4286	9.114856e+10	7.270291e+10	44.2260	44.2348
2018-06-29	185.110000	61.5693	51.5077	48.2783	25.4782	-22.6330	71.4286	14.2857	9.105761e+10	7.270546e+10	44.1370	44.2523

1887 rows × 12 columns

Figure 3 Apple Stock Price Data along with Various Financial Indicators We Collected

The financial indicators data we collected are:

No.	Financial Indicator	Brief Description
1.	Stochs (Stochastic Oscillator)	Stochastic Oscillator is a momentum indicator that compares a particular closing price of a security to its price range over a specified period of time. It measures the level of the closing price relative to the high-low range over a given period, typically 14 days. Stochs help identify potential overbought or oversold conditions in the market, signaling potential reversal points.
2.	RSI (Relative Strength Index)	Relative Strength Index is a momentum oscillator that measures the speed and change of price movements. It ranges from 0 to 100 and is typically used to identify overbought or oversold conditions in a stock. RSI

		compares the magnitude of recent gains to recent losses over a specified period, usually 14 days, and is calculated using a formula that normalizes price changes.
3.	ADX (Average Directional Index)	Average Directional Index is a trend strength indicator that measures the strength of a trend, regardless of its direction. ADX ranges from 0 to 100 and is typically used to determine the strength of a trend, rather than its direction. A high ADX value indicates a strong trend, while a low value suggests a weak trend or a sideways market.
4.	CCI (Commodity Channel Index)	Commodity Channel Index is a momentum-based oscillator used to identify cyclical trends in a security. CCI measures the difference between the current price and its historical average relative to the average deviation over a specified period. Positive CCI values indicate that the price is above the historical average, while negative values suggest the price is below the historical average.
5.	AROON	AROON is a technical indicator used to identify trends in a security's price. It consists of two lines - Aroon Up

		and Aroon Down - which measure the time it takes for the highest high and lowest low to occur over a specified period. Aroon values range from 0 to 100, with higher values indicating stronger trends.
6.	OBV (On Balance Volume)	On Balance Volume is a momentum indicator that measures buying and selling pressure by adding volume on up days and subtracting volume on down days. OBV is used to confirm price trends and identify potential reversals. Rising OBV suggests accumulation (buying) pressure, while falling OBV indicates distribution (selling) pressure.
7.	Chaikin A/D (Chaikin Accumulation/Distribution)	Chaikin Accumulation/Distribution is a volume-based indicator that measures the cumulative flow of money into and out of a security. It combines price and volume to assess the strength of buying and selling pressure. Rising Chaikin A/D suggests accumulation, while falling values indicate distribution.
8.	SMA (Simple Moving Average)	Simple Moving Average is a technical indicator that calculates the average price of a security over a specified period, smoothing out price fluctuations and highlighting trends. SMA is commonly used to identify

		support and resistance levels and to assess the direction of the trend.
9.	EMA (Exponential Moving Average)	Exponential Moving Average is similar to SMA but places greater weight on more recent price data, making it more responsive to recent price changes. EMA reacts more quickly to price movements compared to SMA, making it useful for short-term trend analysis.

Note that for the indicator which is based on a window of values such as SMA and EMA, the right-end of the window is based on today's relevant values to simulate how future values are not available yet.

3. Trading Strategy 1: Stepwise Regression

    a. Overview of strategy logic

Stepwise regression is essentially the combination of Linear Regression with Feature Selection. It is a technique that attempts to optimize a Linear regression model by choosing the most statistically significant predictors by either forward or backward feature selection (or a combination of both). Forward feature selection starts with no predictors and adds them one-by-one based on the most statistically significant contribution to the linear regression model (eg. highest difference in adjusted R-squared). Backward feature selection starts with all the predictors and iteratively removes predictors based on least statistically significant contribution to the model (eg. lowest difference in adjusted R-squared). The linear regression model can thus be used to predict our desired output.

We define the model to perform forward selection below:

```
# define model
from sklearn.linear_model import LinearRegression
from sklearn.feature_selection import SequentialFeatureSelector

LR_model = LinearRegression()
SFS_model = SequentialFeatureSelector(LR_model,
                                      scoring = 'r2')
```

Figure 4 Setting up model

LR_model is the linear regression model used in forward selection of variables. It is applied in the forward selection process and later on to do prediction. *SFS_model* is the model that performs forward selection with linear regression as the unfitted estimator and R-squared as the scoring parameter.

b. Preliminary attempt of predicting future log return using lag data

We first attempt to use lagged data as our variable to predict the log return of Apple stock. First, we prepare the data by finding the 5 period of lagged log returns and the respective direction.

```
Apple = pd.DataFrame(log_ret['AAPL.O'])
#Adding the actual direction of the stock price movement
Apple['Actual_dir'] = np.sign(Apple['AAPL.O'])
for i in range(1,6):
    s = 'lag'+ str(i)
    t = 'lag'+ str(i) + '_dir'
    Apple[s] = Apple['AAPL.O'].shift(i)
    Apple[t] = np.sign(Apple['AAPL.O'].shift(i))
Apple = Apple.dropna()
Apple
```

Date	AAPL.O	Actual_dir	lag1	lag1_dir	lag2	lag2_dir	lag3	lag3_dir	lag4	lag4_dir	lag5	lag5_dir
2010-01-12	-0.011440	-1.0	-0.008861	-1.0	0.006626	1.0	-0.001850	-1.0	-0.016034	-1.0	0.001727	1.0
2010-01-13	0.014007	1.0	-0.011440	-1.0	-0.008861	-1.0	0.006626	1.0	-0.001850	-1.0	-0.016034	-1.0
2010-01-14	-0.005808	-1.0	0.014007	1.0	-0.011440	-1.0	-0.008861	-1.0	0.006626	1.0	-0.001850	-1.0
2010-01-15	-0.016853	-1.0	-0.005808	-1.0	0.014007	1.0	-0.011440	-1.0	-0.008861	-1.0	0.006626	1.0
2010-01-19	0.043288	1.0	-0.016853	-1.0	-0.005808	-1.0	0.014007	1.0	-0.011440	-1.0	-0.008861	-1.0
...	...	...	...	...	...	...	...	...	...	...	...	...
2018-06-25	-0.014983	-1.0	-0.002916	-1.0	-0.005592	-1.0	0.004353	1.0	-0.016292	-1.0	-0.000530	-1.0
2018-06-26	0.012330	1.0	-0.014983	-1.0	-0.002916	-1.0	-0.005592	-1.0	0.004353	1.0	-0.016292	-1.0
2018-06-27	-0.001465	-1.0	0.012330	1.0	-0.014983	-1.0	-0.002916	-1.0	-0.005592	-1.0	0.004353	1.0
2018-06-28	0.007250	1.0	-0.001465	-1.0	0.012330	1.0	-0.014983	-1.0	-0.002916	-1.0	-0.005592	-1.0
2018-06-29	-0.002105	-1.0	0.007250	1.0	-0.001465	-1.0	0.012330	1.0	-0.014983	-1.0	-0.002916	-1.0

2132 rows × 12 columns

Figure 5 Apple stock Lag variables data

With the lagged variables, we split the dataframe for training and testing. Finally we perform a stepwise linear regression to predict the log return of Apple stock for the training data set.

```
# perform linear regression on the selected variables to predict training data
Apple_train['Predict_Step'] = LR_model.fit(sel_X_train, y_train).predict(sel_X_train)
Apple_train
```

Date	AAPL.O	Actual_dir	lag1	lag1_dir	lag2	lag2_dir	lag3	lag3_dir	lag4	lag4_dir	lag5	lag5_dir	Predict_Step
2010-01-12	-0.011440	-1.0	-0.008861	-1.0	0.006626	1.0	-0.001850	-1.0	-0.016034	-1.0	0.001727	1.0	-0.001368
2010-01-13	0.014007	1.0	-0.011440	-1.0	-0.008861	-1.0	0.006626	1.0	-0.001850	-1.0	-0.016034	-1.0	-0.000227
2010-01-14	-0.005808	-1.0	0.014007	1.0	-0.011440	-1.0	-0.008861	-1.0	0.006626	1.0	-0.001850	-1.0	0.003479
2010-01-15	-0.016853	-1.0	-0.005808	-1.0	0.014007	1.0	-0.011440	-1.0	-0.008861	-1.0	0.006626	1.0	-0.000922
2010-01-19	0.043288	1.0	-0.016853	-1.0	-0.005808	-1.0	0.014007	1.0	-0.011440	-1.0	-0.008861	-1.0	-0.000570
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2016-10-11	0.002152	1.0	0.017297	1.0	0.001492	1.0	0.007403	1.0	0.000442	1.0	0.004257	1.0	0.001186
2016-10-12	0.008903	1.0	0.002152	1.0	0.017297	1.0	0.001492	1.0	0.007403	1.0	0.000442	1.0	0.001461
2016-10-13	-0.003073	-1.0	0.008903	1.0	0.002152	1.0	0.017297	1.0	0.001492	1.0	0.007403	1.0	0.000726
2016-10-14	0.005541	1.0	-0.003073	-1.0	0.008903	1.0	0.002152	1.0	0.017297	1.0	0.001492	1.0	-0.000252
2016-10-17	-0.000680	-1.0	0.005541	1.0	-0.003073	-1.0	0.008903	1.0	0.002152	1.0	0.017297	1.0	0.001271

1704 rows × 13 columns

Figure 6 Log return prediction

The predicted step proved to be accurate with a mean squared error of 0.0002766871052795959. We perform a simple trading strategy to buy or sell the stock based on the predicted direction of the stock.

```
#Implementing the trading strategy
position = 0
trades = []
for i in range(len(Apple_train['Actual_dir'])):
    predict_dir = Apple_train['Predict_dir'].iloc[i]
    if predict_dir>0 and position<=0:
        trades.append(('buy', Apple_train['AAPL.O'].iloc[i]))
        position = 1
    elif predict_dir<0 and position>=0:
        trades.append(('sell', Apple_train['AAPL.O'].iloc[i]))
        position = -1
    else:
        trades.append(('hold', 0))
Apple_train['Trades'] = trades
Apple_train
```

Figure 7 Directional trading

This strategy thus gives a strategy return of 2.930424 vs an actual return of 3.916286 in the training data.

However, out of the 1704 predictions, there were a total of 895 trades and 851 false predictions. Such figures are undesirable, leading us to take a different trading strategy. We ultimately decided to use the financial indicators to predict the stock prices, using the stock prices to perform a SMA trading strategy.

c. Predicting future price using financial indicator

i. Data preparation

We first import the relevant files for stock prices and the financial indicators, then we create a dataframe for the respective stock prices along with the indicators at the respective time points. To do that, we perform a left join on the financial indicators to the stock prices. An example is shown for Apple stock:

```
raw = pd.read_csv('tr_eikon_eod_data.csv', index_col = 0, parse_dates = True)
data = raw[['AAPL.O', 'MSFT.O', 'INTC.O', 'AMZN.O', 'GS.N']].dropna()
```

```
aapl = (pd.read_csv('apple_output.csv', index_col = 0, parse_dates = True)).dropna().iloc[:, 1:]
AAPL = (pd.DataFrame(data['AAPL.O']).join(aapl, how = 'left')).dropna()
AAPL
```

Date	AAPL.O	SlowK	SlowD	RSI	ADX	CCI	Aroon Down	Aroon Up	OBV	Chaikin A/D	SMA	EMA
2010-01-04	30.572827	81.3819	78.0429	69.6459	30.2438	87.1666	28.5714	100.0000	7.815824e+10	5.242860e+10	6.2074	6.2604
2010-01-06	30.138541	70.6672	77.9169	70.3530	32.1805	52.0135	14.2857	92.8571	7.820798e+10	5.191009e+10	6.3276	6.3344
2010-01-07	30.082827	47.7707	66.7131	62.5820	30.9912	3.9476	7.1429	85.7143	7.773085e+10	5.192788e+10	6.3649	6.3467
2010-01-11	30.015684	30.4817	37.7213	63.0963	29.4253	-69.1548	0.0000	71.4286	7.771623e+10	5.224445e+10	6.4202	6.3642
2010-01-12	29.674256	27.6022	30.9984	53.7582	27.5182	-162.4372	0.0000	64.2857	7.712177e+10	5.211136e+10	6.4125	6.3569
...	...	...	...	...	...	...	...	...	...	...	...	...
2018-06-25	182.170000	24.7765	28.7326	33.5305	28.2742	-132.0057	100.0000	14.2857	9.108196e+10	7.277260e+10	44.7743	44.5714
2018-06-26	184.430000	30.9769	28.9171	32.5318	28.2014	-70.9622	92.8571	7.1429	9.118024e+10	7.276743e+10	44.5744	44.3849
2018-06-27	184.160000	37.4891	31.0808	42.8627	27.4502	-43.5508	85.7143	0.0000	9.107910e+10	7.267438e+10	44.4033	44.3292
2018-06-28	185.500000	55.4647	41.3102	39.4747	26.9167	-30.3030	78.5714	21.4286	9.114856e+10	7.270291e+10	44.2260	44.2348
2018-06-29	185.110000	61.5693	51.5077	48.2783	25.4782	-22.6330	71.4286	14.2857	9.105761e+10	7.270546e+10	44.1370	44.2523

1868 rows × 12 columns

Figure 8 Apple stock Financial indicator data

After that, we perform a train test split on the respective dataframe to split 80% of the data to train the model and 20% of the model to test our results.

ii. Model execution

We define the functions below to perform stepwise regression on the datasets and subsequently execute the SMA trading strategy on the predicted price.

```
# Stepwise regression to predict training data
def stepwise_regression_train(data):
    df = data.copy()

    # Stepwise regression select features
    X_train = df.iloc[:, 1:]    # training input
    y_train = df.iloc[:, 0]     # training expected output

    sel_X_bool = SFS_model.fit(X_train, y_train).get_support()    # boolean list of variables selected
    sel_X_train = X_train.loc[:, sel_X_bool]                      # filter variables

    # perform linear regression on the selected variables to predict training data
    df['Predict_Step'] = LR_model.fit(sel_X_train, y_train).predict(sel_X_train)

    print('Selected variables are ' + str(list(sel_X_train.columns)))
    return df
```

Figure 9 Stepwise regression on training data

The function *stepwise_regression_train(data)* takes in the underlying training data and predicts the price of the underlying stock. The function first separates the data into the variables and the expected output. The data is then fit into *SFS_model* that returns variables that are significant in predicting the stock price. With the subset of variables, perform linear regression to predict the price in the training data. The function will then return the dataframe with the new predicted value and a string of the variables that were used.

```
# Stepwise regression to predict testing data
def stepwise_regression_test(train, test):
    df_train = train.copy()
    df_test = test.copy()

    # Stepwise regression select features
    X_train = df_train.iloc[:, 1:]    # training input
    y_train = df_train.iloc[:, 0]     # training expected output

    sel_X_bool = SFS_model.fit(X_train, y_train).get_support()    # boolean list of variables selected
    sel_X_train = X_train.loc[:, sel_X_bool]                      # filter variables

    # Filter test data
    X_test = df_test.iloc[:, 1:]                                  # test input data
    sel_X_test = X_test.loc[:, sel_X_bool]                        # variables selected from training data

    # perform linear regression on the selected variables to predict training data
    df_test['Predict_Step'] = LR_model.fit(sel_X_train, y_train).predict(sel_X_test)

    print('Selected variables are ' + str(list(sel_X_train.columns)))
    return df_test
```

Figure 10 Stepwise regression on testing data

The function *stepwise_regression_test(train, test)* is similar to *stepwise_regression_train(data)*, except that this function will be used to predict the stock prices in our testing data set. The

difference lies in the second half of the function where the result from *SFS_model* is passed through the test data set and hence predicting the price of the test data set instead.

d. SMA trading strategy using predicted prices

```
def trade(data):
    df = data.copy()

    SMA_short = 20
    SMA_long = 50

    # Predicted
    df['SMA_Short'] = df['Predict_Step'].rolling(SMA_short).mean()
    df['SMA_Long'] = df['Predict_Step'].rolling(SMA_long).mean()

    df.dropna(inplace = True)
    df['Position'] = np.where(df['SMA_Short'] > df['SMA_Long'], 1, -1)

    df['log_ret'] = np.log(df.iloc[:, 0] / df.iloc[:,0].shift(1))
    df['Predict_log_ret'] = np.log(df['Predict_Step'] / df['Predict_Step'].shift(1))
    df['SMA_Returns'] = df['Position'].shift(1) * df['log_ret']
    return df
```

Figure 11 SMA trading implementation

The function *trade(data)* would take the predicted price and perform a SMA trading strategy. The short and long period used are 20 and 50 respectively in this case. The predicted price is used to determine the position to take at each time period. The function will take a long position when *SMA_Short* > *SMA_Long* and short position when the converse is true. With the position, the trade is performed on the actual stock price. This function also calculates the log return of the actual and predicted price for analysis later.

e. Results of backtesting and performance metrics

The table below summarizes the various indicators that were selected by the model to be used to predict the respective stock price.

	SlowK	**SlowD**	**RSI**	**ADX**	**CCI**	**Aroon Down**	**Aroon Up**	**OBV**	**Chaikin A/D**	**SMA**	**EMA**
AAPL.O											
MSFT.O											
INTC.O											
AMZN.O											
GS.N											

```python
def statistics(data):
    df = data
    # Actual Return
    ret = np.exp(np.sum(df['log_ret']))

    # Predicted Return
    pred_ret = np.exp(np.sum(df['Predict_log_ret']))

    # norminal mean square error of prediction
    df['actual_norm'] = (df[df.columns[0]] - df[df.columns[0]].mean()) / df[df.columns[0]].std()
    df['predicted_norm'] = (df['Predict_Step'] - df['Predict_Step'].mean()) / df['Predict_Step'].std()
    mse = mean_squared_error(df['actual_norm'], df['predicted_norm'])

    # Strategy returns
    Predicted_Step = np.exp(np.sum(df['SMA_Returns']))

    # Number of trades (when position change)
    trades = (df['Position'].diff().dropna() != 0).sum()

    return pd.DataFrame([ret, pred_ret, mse, Predicted_Step, trades],
                        ['Actual return', 'Predicted return', 'Norm Mean Squared error', 'Strategy return', 'Number of trades'],
                        [data.columns[0]])
```

Figure 12 Function generate summary statistics

The function *statistics(data)* will be used to analyse the performance of our trading strategy. This function takes in the data after trading and returns metrics in a pandas dataframe to analyse with. 'Actual return' returns the actual log return of the stock over the period. 'Predicted return' returns the log return of the predicted stock price over the period. 'Norm mean Squared error' normalize the actual and predicted price, then calculate the mean squared error of the regression. 'Strategy return' is the returns of the trading strategy. 'Number of trades' gives the number of times the position changes in the duration.

The metrics can be generated for the outcome on the training and testing data using the code below, along with the outcome:

Summary from predicting training data

```python
statistics(AAPLO_train).join(statistics(MSFTO_train)).join(statistics(INTCO_train)).join(statistics(AMZNO_train)).join(statistics(GSN_train))
```

	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N
Actual return	3.420331	1.952011	1.677116	5.844731	0.959667
Predicted return	3.376963	1.977841	1.727727	5.518039	1.050270
Norm Mean Squared error	0.002817	0.002831	0.019588	0.001309	0.013947
Strategy return	1.373444	0.518951	0.514628	1.822625	1.629921
Number of trades	30.000000	38.000000	34.000000	34.000000	34.000000

Figure 13 Results from training data

Summary from predicting test data

```
statistics(AAPLO_test).join(statistics(MSFTO_test)).join(statistics(INTCO_test)).join(statistics(AMZNO_test)).join(statistics(GSN_test))
```

	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N
Actual return	1.650999	1.616292	1.392437	2.215214	0.913522
Predicted return	1.671735	1.626077	1.423309	2.224121	0.983142
Norm Mean Squared error	0.010511	0.004124	0.010347	0.004215	0.043801
Strategy return	0.712956	1.616292	1.232234	1.843929	1.101074
Number of trades	9.000000	0.000000	5.000000	3.000000	5.000000

Figure 14 Results from testing data

We can see that the expected returns from the model's predicted prices are fairly similar to the actual prices observed. This is supported by the normalized mean square error being close to 0. To visualise this, we can plot the graphs comparing the actual and predicted price of each stock. Below are the graphs comparing the prediction on the test data:

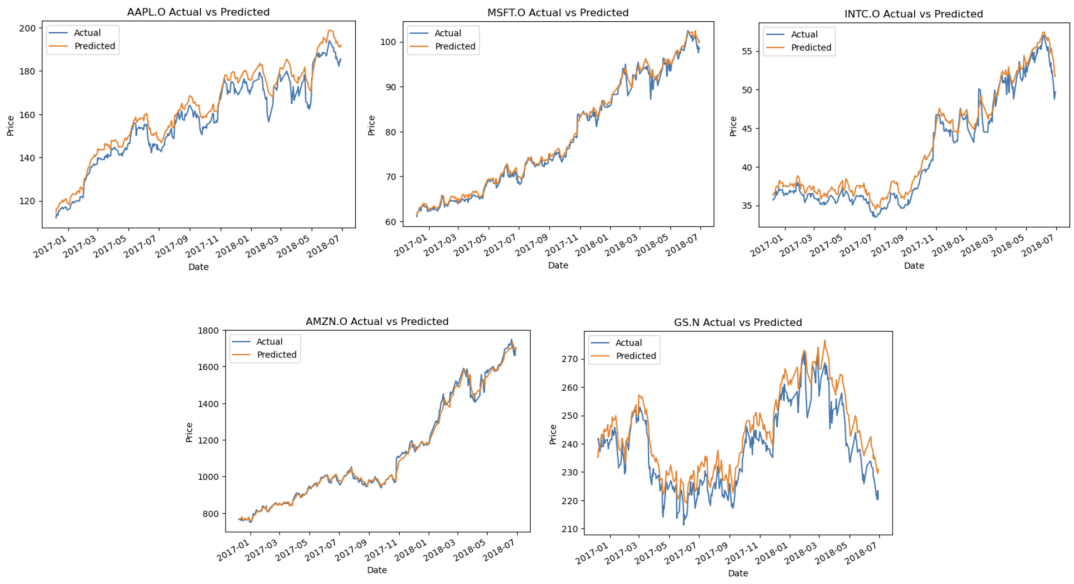


Figure 15 Predicted vs Actual price on test data

To a large degree, stepwise regression is fairly accurate in predicting the stock price. However, we also identify that despite the accurate prediction of prices, the model underperformed in some of its trading strategy. On the training data set, Microsoft and Intel produced returns of less than 1 while Apple, Amazon and Goldman Sachs have returns of greater than 1. Surprisingly, on trading in the test data, Apple and Goldman Sachs stocks underperformed while Microsoft and Intel stocks

performance exceeded expectations and had returns greater than 1. Amazon stocks returns were consistent in both data sets.

The implementation of the trading strategy on the test data set is illustrated below:

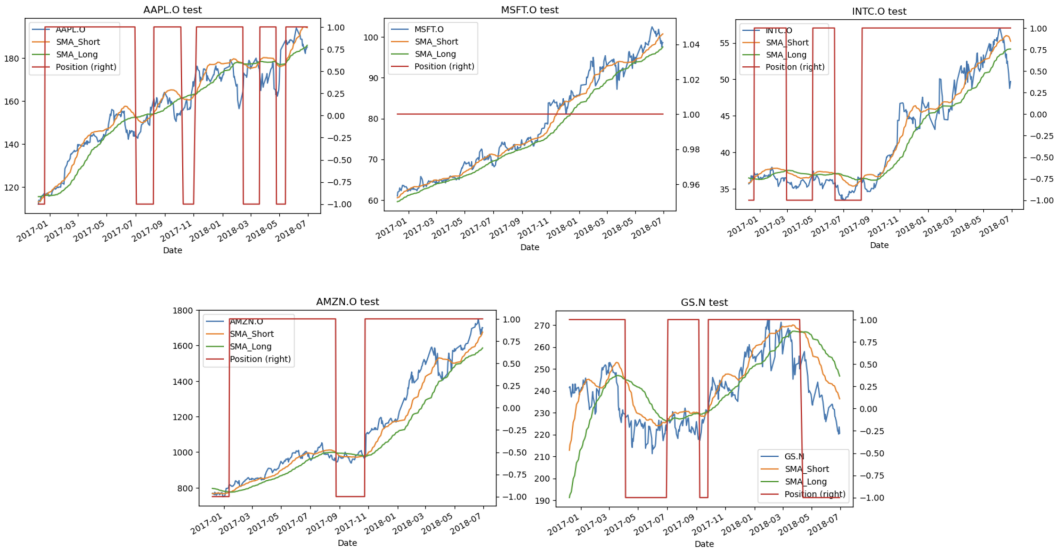


Figure 16 Positions from SMA strategy

In real life applications, we would want to minimize the number of trades as each trade would incur some form of transaction cost. This model performs the trading strategy with acceptable number of trades with less than 10 trades occurring in the duration of the test data (~2 years).

f. Optimizing the SMA periods

Previously the trading strategy fixed the period for SMA trading to be 20 and 50 periods. These periods are chosen as to be the common periods which a crossover is significant. (Maverick, 2023). The use of different SMA periods can lead to different strategy returns. To optimize our returns, the periods chosen for the SMA trading strategy can be further optimised to select the optimal SMA periods to implement the trading strategy.

The function *optimize(data)* would perform brute force optimisation by testing the various combinations of SMA long and short periods with our predicted value and choose the optimal

SMA periods to implement on our trading strategy. The values for the shorter period would range from 1 to 50 and the longer period range from 51 to 200. The function is defined as below:

```
# To find optimum SMA values
def optimize(data):
    df = pd.DataFrame(data['Predict_Step'])
    from itertools import product
    SMA_short = range(1, 51, 1)
    SMA_long = range(50, 201, 1)

    result = pd.DataFrame()
    for sma_short, sma_long in product(SMA_short, SMA_long):

        # compute short term and long term SMA
        df['SMA_short'] = df[df.columns[0]].rolling(sma_short).mean()
        df['SMA_long'] = df[df.columns[0]].rolling(sma_long).mean()

        # compute the returns from the benchmark strategy
        df['Benchmark_Returns'] = np.log(df[df.columns[0]] / df[df.columns[0]].shift(1))

        # compute the position
        df.dropna(inplace = True)
        df['position'] = np.where(df['SMA_short'] > df['SMA_long'], 1, -1)

        # compute the return from the SMA strategy
        df['SMA_Returns'] = df['position'].shift(1) * df['Benchmark_Returns']
        df.dropna(inplace = True)

        # Compare the total return from benchmark and SMA strategy
        Performance = np.exp(df[['Benchmark_Returns', 'SMA_Returns']].sum())

        # Compute by how much SMA strategy outperforms the benchmark
        Out_performance = Performance['SMA_Returns'] - Performance['Benchmark_Returns']

        # Store the result in every run into results dataframe
        result = pd.concat([result, pd.DataFrame({'SMA_short': sma_short,
                                                  'SMA_long': sma_long,
                                                  'Outperformance': Out_performance}, index=[0])], ignore_index = True)
    return result.sort_values('Outperformance', ascending = False)
```

Figure 17 SMA period optimization function

Applying the *optimize* function to the *trade* function, we produce a different set of results. The tables below are the optimal period selected from the model for each stock.

Train data		
	SMA_short	**SMA_long**
AAPL.O	1	60
MSFT.O	26	50
INTC.O	1	68
AMZN.O	1	64
GS.N	1	52

Test data		
	SMA_short	**SMA_long**
AAPL.O	26	50
MSFT.O	26	50
INTC.O	1	55
AMZN.O	1	51
GS.N	1	51

The returns and metrics from the trading strategy are displayed below:

Summary from predicting training data

```
statistics(AAPLO_train).join(statistics(MSFTO_train)).join(statistics(INTCO_train)).join(statistics(AMZNO_train)).join(statistics(GSN_train))
```

	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N
Actual return	3.212816	1.952011	1.610060	5.261776	0.978902
Predicted return	3.093726	1.977841	1.712337	5.035841	1.069074
Norm Mean Squared error	0.002865	0.002831	0.019338	0.001320	0.013835
Strategy return	2.125161	0.689024	1.185780	1.555400	1.923803
Number of trades	86.000000	34.000000	104.000000	80.000000	84.000000

Figure 18 Optimized result on training data

Summary from predicting test data

```
statistics(AAPLO_test).join(statistics(MSFTO_test)).join(statistics(INTCO_test)).join(statistics(AMZNO_test)).join(statistics(GSN_test))
```

	AAPL.O	MSFT.O	INTC.O	AMZN.O	GS.N
Actual return	1.650999	1.616292	1.351182	2.211381	0.912012
Predicted return	1.671735	1.626077	1.381398	2.212890	0.971749
Norm Mean Squared error	0.010511	0.004124	0.010319	0.004234	0.041633
Strategy return	0.912977	1.616292	1.078916	2.085940	0.984166
Number of trades	9.000000	0.000000	21.000000	5.000000	19.000000

Figure 19 Optimized result on testing data

On the training data, we notice that the optimisation step significantly improved the returns on Apple and Intel stocks. Microsoft and Goldman Sachs see slight improvement in returns. Amazon stocks however did not benefit from the optimisation of SMA periods.

On the testing data, Apple, Intel and Amazon stocks have improved returns with the new SMA periods. Microsoft, whose chosen optimal period is close to what we used previously, see similar returns. Goldman Sachs, however, saw a fall in returns in the new period. Apple's returns after optimisation is still less than 1. The new trading strategy can be visualised in the graphs below:

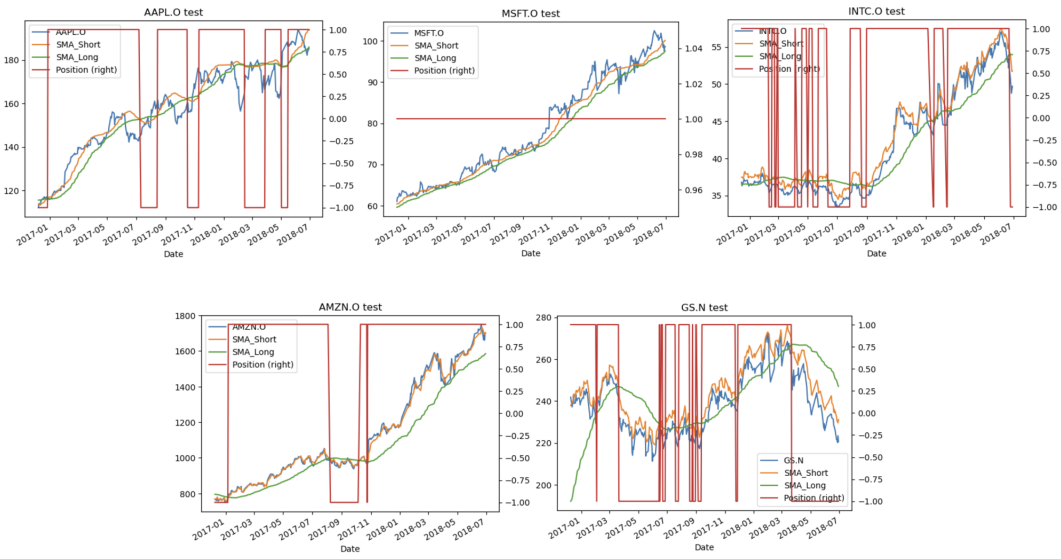


Figure 20 Optimum positions from SMA trading

It is also important to note that the optimal shorter period is mostly 1, compared to the assumption of 20 initially. As a result, we see an increase in the number of trades occurring. This may have an overall impact on the profit from any transaction cost incurred.

g. Potential challenges and risk of stepwise regression

One of the largest problems with stepwise regression is that it is very easy to overfit the model to the data. This is because the more potential predictors we try to add, the higher the chance we get a type 1 error (attributing significance to a model when there is none). This reason could potentially explain the results from Apple and Goldman Sachs returns which underperformed in the test data set as these stocks see greater fluctuation in prices. This means it is important to limit the potential predictors to a smaller group, as well as evaluating these predictors with a more heuristic approach.

h. Future improvements

For future improvements, we can aim to decompose time series into a trend and residual component and model each separately. We can also apply Fourier transforms to identify and model cyclical patterns within the data. Similarly, we also aim to use the Modern Portfolio Theory to

optimize asset allocation and minimize risk. Finally, to prevent overfitting, we can use ensembling methods such as Random Forest, Boosting which combines prediction from multiple models and gives more accurate predictions.

4. Trading Strategy 2: Stock Price Direction Prediction using Support Vector Machine

    a. Overview of Strategy Logic

In this trading strategy, we utilize machine learning, specifically Support Vector Machine (SVM), to classify whether the stock price will rise or fall in the following day, assigning a label of 1 for an anticipated rise or no change and -1 for an expected decline. If the SVM model forecasts a price increase, we will take a long position by buying 1 unit of stock and conversely, if the model predicts a price decrease, we will take a short position by selling 1 unit of stock. For every trade (which occurs once a day), the absolute difference between todays and tomorrow's prices will be the profit in the case of accurate prediction and the loss in the case of inaccurate prediction of the trade. The log returns of the strategy are computed by summing the log return of each trade.

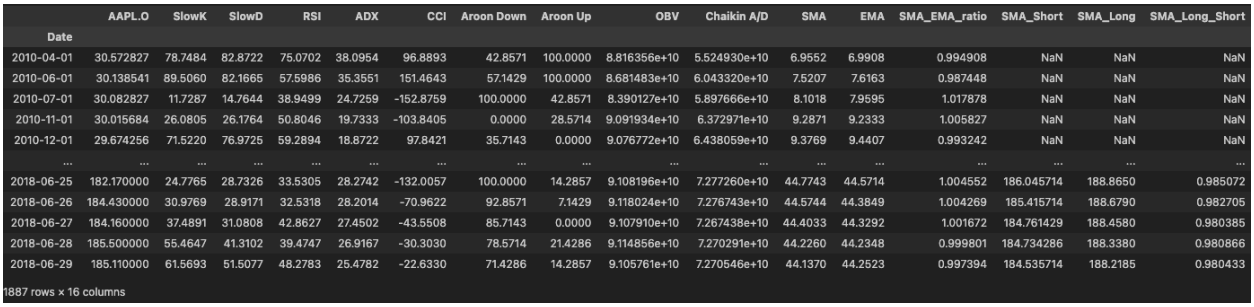
    b. Machine Learning Model Training and Optimization

        i. Feature Engineering 1: Feature Creation

Although certain features may exhibit weak correlation with stock price direction initially, we can enhance their predictive power by creating new features through transformation functions or by combining them with other features. In Figure 21 below, we illustrate the creation of the following features, aiming to strengthen their correlation with stock price direction.

No.	Features	Brief Description
1.	SMA_Short	SMA on stock price with a window with size 7
2.	SMA_Long	SMA on stock price with a window with size 7
3.	SMA_Long_Short	Ratio of the derived SMA Short / SMA Long

4.	SMA_EMA_ratio	Ratio of SMA/EMA



Date	AAPL.O	SlowK	SlowD	RSI	ADX	CCI	Aroon Down	Aroon Up	OBV	Chaikin A/D	SMA	EMA	SMA_EMA_ratio	SMA_Short	SMA_Long	SMA_Long_Short
2010-04-01	30.572827	78.7484	82.8722	75.0702	38.0954	96.8893	42.8571	100.0000	8.816356e+10	5.524930e+10	6.9552	6.9908	0.994908	NaN	NaN	NaN
2010-06-01	30.138541	89.5060	82.1665	57.5986	35.3551	151.4643	57.1429	100.0000	8.681483e+10	6.043320e+10	7.5207	7.6163	0.987448	NaN	NaN	NaN
2010-07-01	30.082827	11.7287	14.7644	38.9499	24.7259	-152.8759	100.0000	42.8571	8.390127e+10	5.897666e+10	8.1018	7.9595	1.017878	NaN	NaN	NaN
2010-11-01	30.015684	26.0805	26.1764	50.8046	19.7333	-103.8405	0.0000	28.5714	9.091934e+10	6.372971e+10	9.2871	9.2333	1.005827	NaN	NaN	NaN
2010-12-01	29.674256	71.5220	76.9725	59.2894	18.8722	97.8421	35.7143	0.0000	9.076772e+10	6.438059e+10	9.3769	9.4407	0.993242	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2018-06-25	182.170000	24.7765	28.7326	33.5305	28.2742	-132.0057	100.0000	14.2857	9.108196e+10	7.277260e+10	44.7743	44.5714	1.004552	186.045714	188.8650	0.985072
2018-06-26	184.430000	30.9769	28.9171	32.5318	28.2014	-70.9622	92.8571	7.1429	9.118024e+10	7.276743e+10	44.5744	44.3849	1.004269	185.415714	188.6790	0.982705
2018-06-27	184.160000	37.4891	31.0808	42.8627	27.4502	-43.5508	85.7143	0.0000	9.107910e+10	7.267438e+10	44.4033	44.3292	1.001672	184.761429	188.4580	0.980385
2018-06-28	185.500000	55.4647	41.3102	39.4747	26.9167	-30.3030	78.5714	21.4286	9.114856e+10	7.270291e+10	44.2260	44.2348	0.999801	184.734286	188.3380	0.980866
2018-06-29	185.110000	61.5693	51.5077	48.2783	25.4782	-22.6330	71.4286	14.2857	9.105761e+10	7.270546e+10	44.1370	44.2523	0.997394	184.535714	188.2185	0.980433

1887 rows × 16 columns

Figure 21 Creation of New Features

Please note that missing data (NA) will be excluded during the training and implementation of the model. This ensures the integrity of the analysis and prevents the inclusion of incomplete or unreliable information in the decision-making process.

ii. Data Optimization: Normalization

Subsequently, we conducted normalization on all feature columns using MinMaxScaler. Normalization is carried out to scale the data within a consistent range, typically between 0 and 1, ensuring that all features contribute equally to the analysis and preventing any particular feature from dominating due to its larger magnitude. It enhances the stability and convergence of machine learning algorithms during training. Note that we retain the original values of price and log returns, as they are essential for return calculation and graph generation in later stages of analysis. Figures 22 and 23 below illustrate the values before and after normalization, respectively.

Date	AAPL.O	SlowK	SlowD	RSI	ADX	CCI	Aroon Down	Aroon Up	OBV	Chaikin A/D	SMA	EMA	SMA_EMA_ratio	SMA_Short	SMA_Long	SMA_Long_Short	price	log_rets	direction
2010-04-01	30.572827	78.7484	82.8722	75.0702	38.0954	96.8893	42.8571	100.0000	8.816356e+10	5.524930e+10	6.9552	6.9908	0.994908	NaN	NaN	NaN	30.572827	-0.014307	-1
2010-06-01	30.138541	89.5060	82.1665	57.5986	35.3551	151.4643	57.1429	100.0000	8.681483e+10	6.043320e+10	7.5207	7.6163	0.987448	NaN	NaN	NaN	30.138541	-0.001850	-1
2010-07-01	30.082827	11.7287	14.7644	38.9499	24.7259	-152.8759	100.0000	42.8571	8.390127e+10	5.897666e+10	8.1018	7.9595	1.017878	NaN	NaN	NaN	30.082827	-0.002234	-1
2010-11-01	30.015684	26.0805	26.1764	50.8046	19.7333	-103.8405	0.0000	28.5714	9.091934e+10	6.372971e+10	9.2871	9.2333	1.005827	NaN	NaN	NaN	30.015684	-0.011440	-1
2010-12-01	29.674256	71.5220	76.9725	59.2894	18.8722	97.8421	35.7143	0.0000	9.076772e+10	6.438059e+10	9.3769	9.4407	0.993242	NaN	NaN	NaN	29.674256	0.014007	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2018-06-25	182.170000	24.7765	28.7326	33.5305	28.2742	-132.0057	100.0000	14.2857	9.108196e+10	7.277260e+10	44.7743	44.5714	1.004552	186.045714	188.8650	0.985072	182.170000	0.012330	1
2018-06-26	184.430000	30.9769	28.9171	32.5318	28.2014	-70.9622	92.8571	7.1429	9.118024e+10	7.276743e+10	44.5744	44.3849	1.004269	185.415714	188.6790	0.982705	184.430000	-0.001465	-1
2018-06-27	184.160000	37.4891	31.0808	42.8627	27.4502	-43.5508	85.7143	0.0000	9.107910e+10	7.267438e+10	44.4033	44.3292	1.001672	184.761429	188.4580	0.980385	184.160000	0.007250	1
2018-06-28	185.500000	55.4647	41.3102	39.4747	26.9167	-30.3030	78.5714	21.4286	9.114856e+10	7.270291e+10	44.2260	44.2348	0.999801	184.734286	188.3380	0.980866	185.500000	-0.002105	-1
2018-06-29	185.110000	61.5693	51.5077	48.2783	25.4782	-22.6330	71.4286	14.2857	9.105761e+10	7.270546e+10	44.1370	44.2523	0.997394	184.535714	188.2185	0.980433	185.110000	NaN	-1

1887 rows × 19 columns

Figure 22 Data Before Normalization of Feature Columns

Date	AAPL.O	SlowK	SlowD	RSI	ADX	CCI	Aroon Down	Aroon Up	OBV	Chaikin A/D	SMA	EMA	SMA_EMA_ratio	SMA_Short	SMA_Long	SMA_Long_Short	price	log_rets	direction
2010-04-01	0.018826	0.797082	0.851941	0.769280	0.413634	0.658869	0.428571	1.000000	0.466485	0.197673	0.021918	0.021476	0.436146	NaN	NaN	NaN	30.572827	-0.014307	-1
2010-06-01	0.016218	0.907128	0.844247	0.536156	0.374250	0.748379	0.571429	1.000000	0.417465	0.386577	0.033742	0.034488	0.269045	NaN	NaN	NaN	30.138541	-0.001850	-1
2010-07-01	0.015884	0.111497	0.109400	0.287327	0.221484	0.249224	1.000000	0.428571	0.311572	0.333500	0.045891	0.041627	0.950701	NaN	NaN	NaN	30.082827	-0.002234	-1
2010-11-01	0.015480	0.258310	0.233819	0.445504	0.149730	0.329648	0.000000	0.285714	0.566643	0.506703	0.070674	0.068125	0.680743	NaN	NaN	NaN	30.015684	-0.011440	-1
2010-12-01	0.013430	0.723159	0.787620	0.558717	0.137354	0.660432	0.357143	0.000000	0.561133	0.530421	0.072551	0.072439	0.398836	NaN	NaN	NaN	29.674256	0.014007	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2018-06-25	0.929087	0.244971	0.261688	0.215016	0.272481	0.283454	1.000000	0.142857	0.572554	0.836229	0.812646	0.803231	0.652194	0.962688	0.996360	0.435173	182.170000	0.012330	1
2018-06-26	0.942657	0.308398	0.263699	0.201690	0.271435	0.383573	0.928571	0.071429	0.576126	0.836040	0.808467	0.799351	0.645859	0.958849	0.995204	0.420941	184.430000	-0.001465	-1
2018-06-27	0.941036	0.375016	0.287289	0.339535	0.260639	0.428531	0.857143	0.000000	0.572450	0.832650	0.804889	0.798193	0.587664	0.954862	0.993830	0.407000	184.160000	0.007250	1
2018-06-28	0.949082	0.558899	0.398814	0.294329	0.252971	0.450259	0.785714	0.214286	0.574974	0.833689	0.801182	0.796229	0.545763	0.954696	0.993085	0.409889	185.500000	-0.002105	-1
2018-06-29	0.946740	0.621347	0.509992	0.411795	0.232297	0.462838	0.714286	0.142857	0.571669	0.833782	0.799321	0.796593	0.491854	0.953486	0.992342	0.407291	185.110000	NaN	-1

1887 rows × 19 columns

Figure 23 Data After Normalization of Feature Columns

iii. Data Split: Training and Testing Data

Following normalization, we partitioned the data sequentially into training and testing sets according to the instructed ratio of 4:1. Subsequently, both the training and testing datasets were further divided into feature matrices (X) and target vectors (y). This separation facilitates the training of the machine learning model on the training data and the evaluation of its performance on the unseen testing data.

iv. Machine Learning Model Selection

Support Vector Machines offer several kernel functions for classification, including Linear, Gaussian (RBF), Polynomial, and Sigmoid. However, Linear kernel is excluded from consideration due to its tendency to produce predictions that are predominantly all 1 or -1, which may result from its unsuitability for handling non-linearly separable data. This observation is supported by the significantly higher occurrence of 1s in the predictions generated by the Linear

model, as depicted in Figure 24. Among the remaining three kernels, the Polynomial kernel demonstrates the highest accuracy and return rates, as evidenced by Figure 24. Consequently, the Polynomial kernel is selected for further optimization and refinement of the model.

```
SVC Linear Model
accuracy: 0.5294117647058824
log_return: 0.4787557026896492
simple_return: 1.6140647757066218
Count of 1: 372 Count of -1: 2

SVC Gaussian Model
accuracy: 0.4893048128342246
log_return: -0.3270948911013484
simple_return: 0.7210153218423989
Count of 1: 105 Count of -1: 269

SVC Poly Model
accuracy: 0.5026737967914439
log_return: -0.05445583600499991
simple_return: 0.9470003312521907
Count of 1: 76 Count of -1: 298

SVC Sigmoid Model
accuracy: 0.5240641711229946
log_return: 0.4622009690552016
simple_return: 1.5875643224699827
Count of 1: 374 Count of -1: 0
```

Figure 24 Performance Statistics of Different Support Vector Machine Kernels

v. Machine Learning Model Optimization using Hyperparameter Tuning

Upon opting for Support Vector Machine (SVM) with the Polynomial (Poly) kernel, we embarked on optimizing its performance by fine-tuning its hyperparameters, namely the degree and C. The degree parameter determines the complexity of the decision boundary, while C regulates the balance between maximizing the margin and minimizing classification errors. Employing a Brute Force Grid Search approach, we systematically explored various combinations of degree and C values to identify the most effective configuration. Figure 25 illustrates the output of running this process, which later reveals that a degree of 2 and a C value of 0.3 emerged as the optimal parameters, exhibiting superior accuracy and return on investment. These findings guide the finalization of the SVM model with the Polynomial kernel, ensuring its efficacy in accurately forecasting stock price movements.

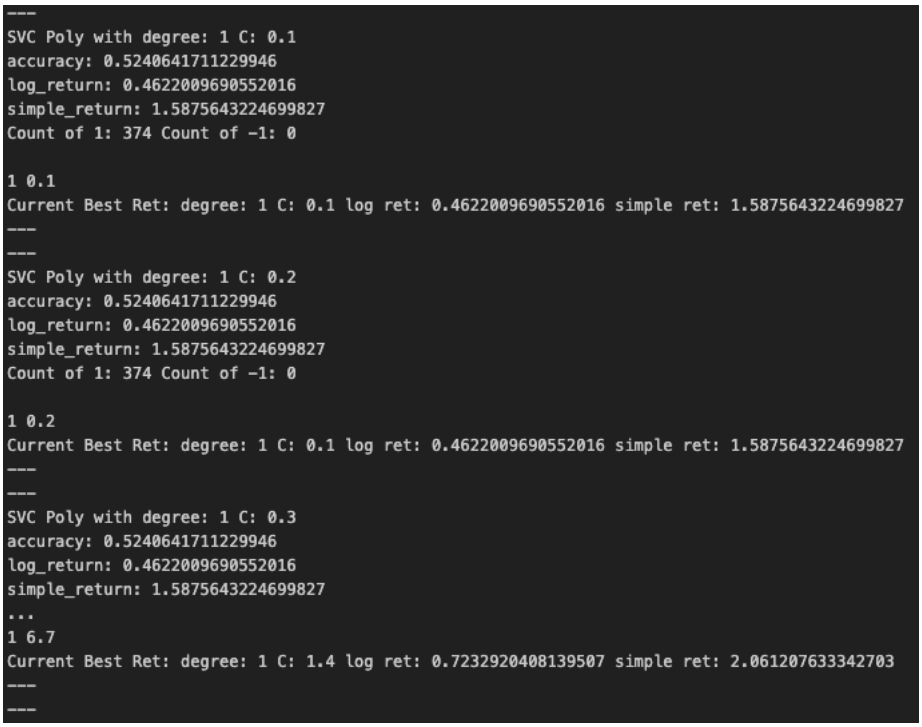


Figure 25 Output of Running Hyperparameters Tuning

vi. Feature Engineering 2: Feature Selection

Using SVM with the Polynomial kernel, specifically with a degree of 2 and C value of 0.3, we conducted feature selection to identify subsets that yield the best returns for each stock. Employing a Brute Force method, we systematically explored every possible combination of features for each stock, selecting the subset that demonstrated the highest return rate. Figure 26 illustrates the output of this feature selection process, where the indices "0, 1, 2, … , 15" correspond to the features in the feature array: ['Stock Price', 'SlowK', 'SlowD', 'RSI', 'ADX', 'CCI', 'Aroon Down', 'Aroon Up', 'OBV', 'Chaikin A/D', 'SMA', 'EMA', 'SMA_EMA_ratio', 'SMA_Short', 'SMA_Long', 'SMA_Long_Short']. This rigorous approach ensures that the selected feature subsets optimize the predictive power of the SVM model for each stock, thereby enhancing the accuracy of forecasting stock price movements.

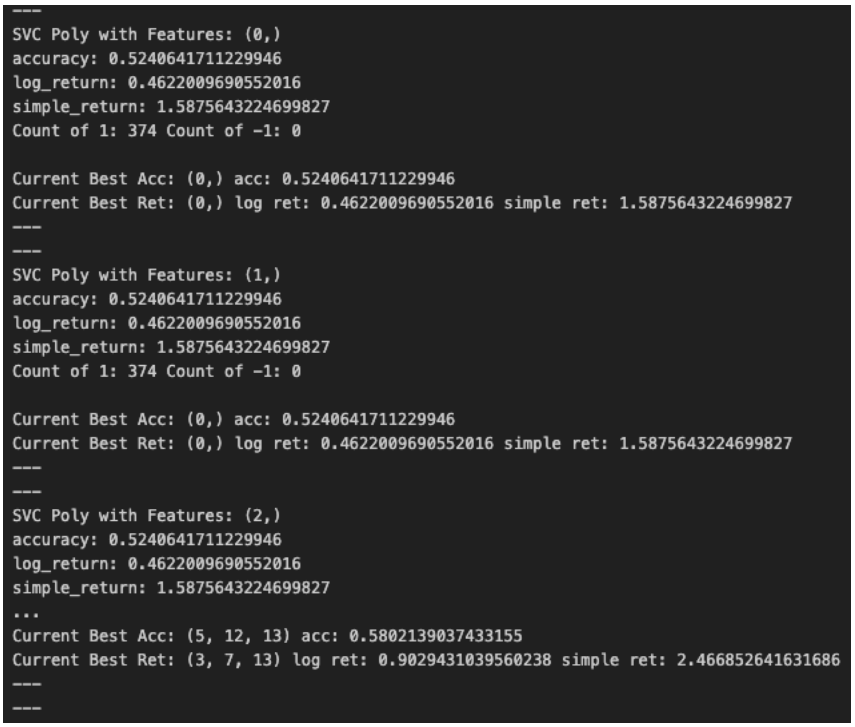


Figure 26 Output of Running Feature Selection

Upon running this process, the best features for each stock are shown in the table below.

No.	Stock Names	Best Features
1.	Apple	'SlowK', 'CCI', 'EMA', 'SMA_Short'
2.	Amazon	'RSI', 'ADX', 'Aroon Down', 'Aroon Up', 'OBV', 'Chaikin A/D', 'SMA', 'EMA', 'SMA_Long_Short'
3.	Microsoft	'MSFT.O', 'SlowK', 'SlowD', 'RSI', 'ADX', 'Aroon Down', 'Aroon Up', 'Chaikin A/D', 'SMA', 'EMA', 'SMA_EMA_ratio', 'SMA_Short', 'SMA_Long', 'SMA_Long_Short'

4.	Intel	'INTC.O', 'SlowD', 'Aroon Down', 'OBV', 'Chaikin A/D', 'SMA_EMA_ratio', 'SMA_Long_Short'
5.	Goldman Sach	'SlowK', 'SlowD', 'RSI', 'CCI', 'Aroon Down', 'OBV', 'Chaikin A/D', 'SMA_Short'

c. Python Code

The general flow of the Python code is as follow:

- For each stock,
    - Prepare a Dataframe with the following format:
        - Column 0-15 are features
        - Column 16 is original stock price
        - Column 17 is log return
        - Column 18 is direction
    - Normalize the feature columns of the Dataframe (Column 0-15)
    - Split the Dataframe into X_train, y_train, X_test and y_test
    - Create a Support Vector Machine model with degree = 2 and C = 0.3
    - Choose the best feature subset for this stock and use it to train the model
    - Use the model to predict test data
    - Calculate return and accuracy of the prediction

Figure 27 below shows a screenshot of the code, which can also be found in result.ipynb.

```python
for i in range(5) :
    curr = stocks[i]

    data = prepare_data(curr[1], curr[0], add_original_price_col=True, add_log_rets=True, add_direction_col=True)
    data = normalize(data, ['price', 'log_rets', 'direction'])
    X_train, y_train, X_test, y_test, log_rets = sequential_train_test_split(data)
    print(curr[0])
    basic_info(data['log_rets'], pd.DataFrame(log_rets).dropna().to_numpy())

    print(data.columns[curr[2]])


    degree = 2
    c = 0.3

    print("---")
    model = SVC(kernel='poly', degree = degree, C = c)
    curr_X_train = X_train[:, curr[2]].reshape(-1, len(curr[2]))
    curr_X_test = X_test[:, curr[2]].reshape(-1, len(curr[2]))
    acc, log_ret, simple_ret = benchmark(f"SVC Poly with Features: {curr[2]}", model, log_rets, curr_X_train, y_train, curr_X_test, y_test)
```

0.7s Python

Figure 27 Screenshot of the Python Code

d. Benchmark Statistics by Backtesting

Following backtesting on the test data, extracted from the sequential train and test split (the last 20% of all data), we derived accuracy, log return, and simple return statistics. Figure 28 below presents the results, highlighting a notable best Simple Return Rate of 3.091643 achieved through our strategy. It is crucial to note that the benchmark return mirrors the benchmark strategy outlined in the lecture, entailing the purchase of 1 unit of stock (long position), and holding it until the conclusion of the period. This comparison elucidates the effectiveness of our strategy in generating superior returns compared to the benchmark approach.

	Stock	Accuracy	Log Return	Simple Return	benchmark ret
0	AAPL.O	0.588235	1.008780	2.742253	1.573396
1	AMZN.O	0.572193	0.852801	2.346210	2.028183
2	MSFT.O	0.617647	0.860116	2.363435	1.616557
3	INTC.O	0.550802	0.880614	2.412381	1.409813
4	GS.N	0.580214	1.128703	3.091643	1.259536

Figure 28 Statistics of Strategy

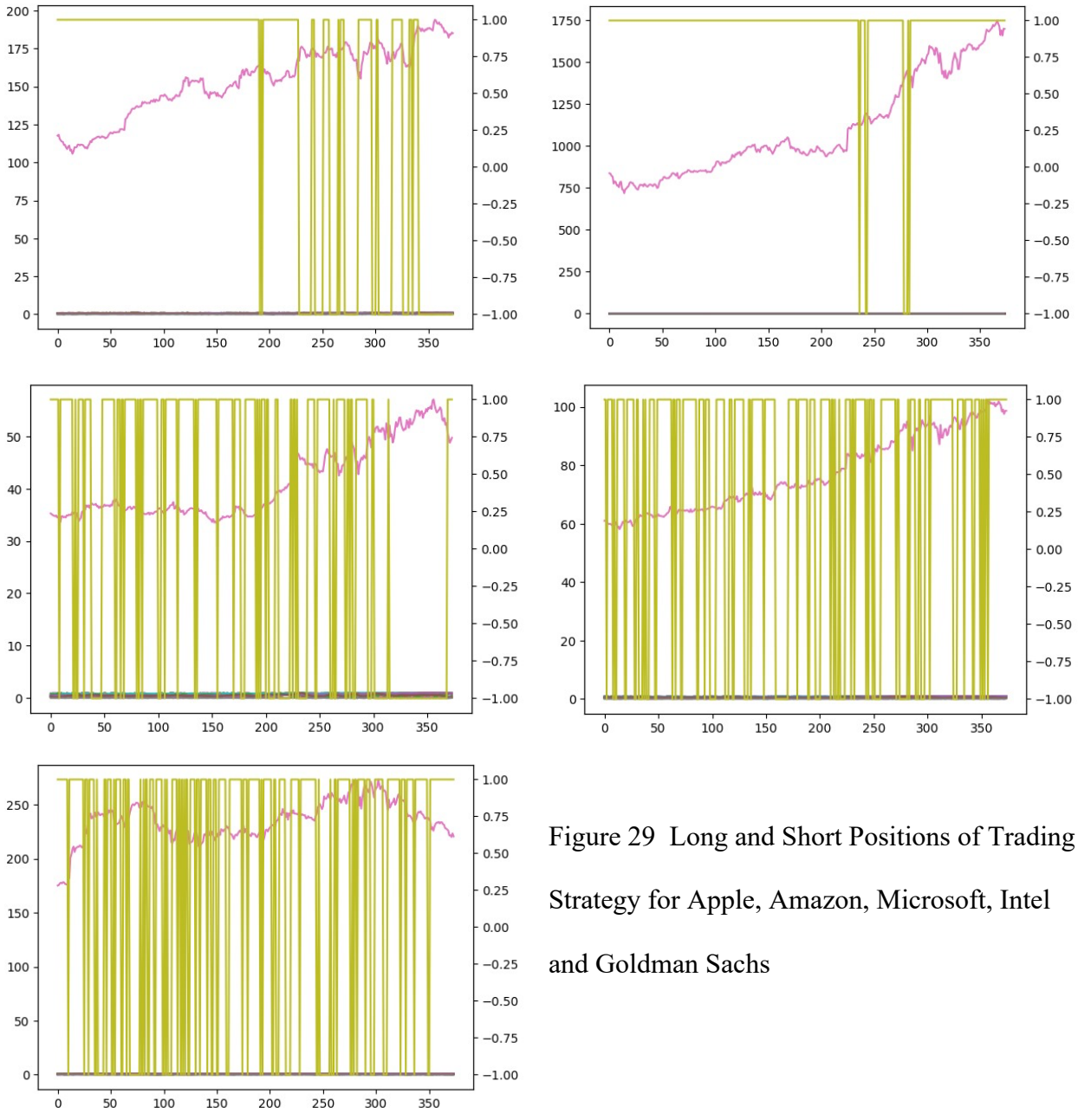


Figure 29 Long and Short Positions of Trading Strategy for Apple, Amazon, Microsoft, Intel and Goldman Sachs

e. Potential Risks and Challenges

Given the complexity and various technical indicators used in the real stock market, overfitting of the model to the given data is a significant risk. This may result in a model which performs well only on historical data, but fails to adapt and utilize new information which may come at anytime in the real-world application and impacts the prediction drastically. Furthermore, there are other

“unseen” information which cannot be solidified into usable data soon enough to be useful. Events such as recession, geopolitical developments, market sentiments and even word of mouth are real time data which can be hard to quantify in a short amount of time for it to be useful.

As external API is used to gain additional data, the quality and consistency can be ascertained. This might lead to reduction in effectiveness and accuracy of the model, and more incorrect prediction to be made.

f. Future Improvement

Exploration of more method such as bagging, boosting and stacking to aggregate prediction from multiple models, in hope of improving the accuracy and robustness of the model. Furthermore, feature engineering is an always an option. Exploration of ways to utilize relevant information such as “unseen” data as mentioned in the previous part or other financial indicators could allow a better model to be created.

Model wise, techniques such as SHAP (Shapley Additive exPlanations) values or plots can provide an insight on why a model does a certain prediction, which improve understanding of the model and increases the possibility of choosing better/improving the model. In addition, continuously training the model or using better model (which comes with advancement of AI/ML) would also a mandatory addition in the future.

Not to forget, better trading strategy can be devised to improve on the returns. Risk management such as stop-loss levels, using options/derivatives, and regular assessments can also be implemented to reduce risk. Backtesting can be conducted thoroughly on various situations, thus giving more confidence on the model or identify area of improvements.

g. Miscellaneous

Trading wise, focus is on the returns. We found that accuracy and returns are not always positively correlated (suppose just comparing 2 data points), even though it is in most cases. Thus, the primary focus is still on the returns, but increasing the accuracy is useful especially before the fine tuning of hyperparameters and models.

References

Maverick, J. (2023, July 24). *Most Commonly-Used Periods in creating Moving Average (MA) lines*. Investopedia. https://www.investopedia.com/ask/answers/122414/what-are-most-common-periods-used-creating-moving-average-ma-lines.asp#:~:text=Common%20periods%20used%20are%20100,exponential%20moving%20average%20(EMA)

Alpha Vantage. (2024, April 10). Documentation. Retrieved from https://www.alphavantage.co/documentation/