Comparison Of MACD And RNN Based Stock Pricing Against Buy & Hold Stable Baseline

CM3070 Computer Science Final Project

1. Introduction

This work is a supporting document for the University of London's CM3070 Computer science Final Project.
This project goal is to compare an average baseline return from US stock markets from the 2000's with a traditional financial analysis based indicator trading strategy the Moving Average Convergence/Divergence (MACD) and a Recurrent Neural Network based Deep LSTM price prediction.

1.1. Baseline

The baseline calculated from the average return of Nasdaq traded ETFs and Stocks. Usually this generic baseline is the S&P 500 index [3] and this work also uses that as the baseline and confirms that it is representing the actual mean return of the market. And that should not be any surprise as the index follows the 500 biggest US (most of them Global) companies.

1.2. MACD Strategy

MACD [4] is a very old strategy created by Gerald Appel in the 1970s. It is a relatively simple trend following strategy.

1.3. Recurrent Neural Network Strategy

RNNs [5] are a type of neural networks where the data flows bi-directionally and that makes them good for time-series forecasting or analysis as every subsequent iteration feeds back its previous state; making it appear as the network has memory and remembers what were the previous states.
LSTMs [6] are the type of RNNs where this memory capabilities are enhanced with not only a simple feedback loop but with rate of forgetting and more control on the feedback itself.

2. Dataset

The Dataset contains several thousands of Nasdaq and other stock exchange traded financial items (stocks and ETFs). The list of symbol is acquired from www.nasdaqtrader.com via the dynamic/SymDir/nasdaqtraded.txt resource. This resource is a CSV file separated by the "|" pipe symbol.

Snippet from nasdaqtraded.txt:

```
Nasdaq Traded|Symbol|Security Name|Listing Exchange|Market Category|ETF|Round Lot Size|Test Issue|Financial Status|CQS Symbol|NASDAQ Symbol|NextShares
Y|A|Agilent Technologies, Inc. Common Stock|N| |N|100|N||A|A|N
Y|AA|Alcoa Corporation Common Stock |N| |N|100|N||AA|AA|N
```

Based on the financial symbols found in the file we download the historical daily price information from Yahoo Finance with the yfinance python module. The symbols are either ETFs or Stocks in this dataset. During the download, we download the maximum amount of data available on Yahoo Finance for each available symbol.
The dataset is based on the Kaggle Stock Market Dataset [1] .
The dataset overview could be found in the 2_Dataset PDF or jupyter notebook found in the notebooks folder.

2.1. Generating Dataset

The Stock Market Dataset is generated by the dataset.py found under the data folder. There is a tests.py script too, that contains high level tests using Python's unittest framework.

The data is stored into two directories etfs and stocks. The directories are created by the dataset.py script and emptied at the beginning of all execution to ensure that the directories contain only fresh data.

Run tests.py to verify that dataset.py script will be able to perform its tasks.

```
UoL_final_project % . ./venv/bin/activate
(venv) UoL_final_project % cd dataset
(venv) dataset % python3 tests.py
[**********************100%%**********************]  1 of 1 completed
...
----------------------------------------------------------------------
Ran 7 tests in 2.729s

OK
```

Run dataset.py to build the Dataset. This process can take up to an hour or more depending on the internet connection and computer speed.

```
UoL_final_project % . ./venv/bin/activate
(venv) UoL_final_project % cd dataset
(venv) dataset % python3 dataset.py
total number of symbols traded = 11258

Fetching: A
[**********************100%%**********************]  1 of 1 completed
```

```
 OK , took 0.42402100563049316 seconds

Fetching: AA
[**********************100%%**********************]  1 of 1 completed
 OK , took 0.6745872497558594 seconds

Fetching: AAA
[**********************100%%**********************]  1 of 1 completed
 OK , took 0.14063096046447754 seconds

Fetching: AAAU
[**********************100%%**********************]  1 of 1 completed
 OK , took 0.19086694717407227 seconds

..........

Fetching: ZWS
[**********************100%%**********************]  1 of 1 completed
 OK , took 0.2697122097015381 seconds

Fetching: ZYME
[**********************100%%**********************]  1 of 1 completed
 OK , took 0.16972684860229492 seconds

Fetching: ZYNE
[**********************100%%**********************]  1 of 1 completed
 OK , took 0.22720599174499512 seconds

Fetching: ZYXI
[**********************100%%**********************]  1 of 1 completed
 OK , took 0.3615880012512207 seconds

Fetched 10476 symbol(s) from 10476 symbols. Overall time took 2967.2017810344696 seconds
```

The script downloaded 10476 symbols' data from 11258 in about 50 minutes.

2.2. High Level Exploration Of Dataset

The Dataset contains 10476 symbols and the total data size is 2.9 Gb that the script downloaded. There are 3170 ETFs, weighing 579 Mb and 7312 Stocks weighing 2.4 Gb. The number of symbols and size could change based on the date of the data acquisition.

```
(venv) dataset % du -h -d 1 .
2.4G    ./stocks
8.0K    ./__pycache__
579M    ./etfs
2.9G    .
(venv) % ls -al etfs | wc -l
    3170
(venv) % ls -al stocks | wc -l
    7312
```

Every symbol has 7 data columns.

Column Name	Description	Usability
Date (also used	The date of the trading day.	Date used as steps.

as index)	All price information is collected on this specific date in the row.	Fundamental to analysis and time-series based analysis.
Open	The opening price for the day.	This work does not focus on intra day price forecasting. Therefore this column is not useful.
High	The highest recorded price for the day.	The High price is not useful on its own as this work does not focus on intra day pricing. Yet it could be useful for feature engineering.
Low	The lowest recorded price for the day.	The Low price is not useful on its own as this work does not focus on intra day pricing. Yet it could be useful for feature engineering.
Close	The price at the end of the day.	As the Close price may fluctuate “randomly” on stock splits and dividends models cannot rely on that column.
Adj Close (Adjusted Close)	The adjusted closing prices from the first appearance of the symbol taking into account all dividends and stock splits. This is basically a stable pricing of the symbol throughout its life.	The Adj Close price is the main focus of this work. The Adjusted close price and its fluctuation will be used as inputs for all models.
Volume	The number of shares changed hands during the trading day.	Volume could be useful on its own or used for feature engineering.

The Dataset contains well over 10000 symbols not every symbol is usable. Doing some quick exploration there are only 32 ETFs and 1856 Stocks that have their first data on or before 2000 1st of January. The earliest ETF is CEF - Sprott Physical Gold and Silver Trust with a starting data of 1986 3rd of April and earliest Stock is MRO - Marathon Oil Corporation with a starting data of 1962 2nd of January. The models training and validation will happen on the symbols that have data at least back to 2000 1st of January. Symbols with

less data might be good for additional training or as additional test and validation data.
The high level exploration process could be found in the 2_2_high_lelvel_exploration_of_dataset PDF or jupyter notebook found in the notebooks folder.

2.3. Properties Of The Data That Worth Attention To

2.3.1. Wide Ranges Of Pricing

The price ranges in the Dataset are from negative numbers to fractions of the Dollar to billions of Dollars. The prices will need transformation, scaling or percentage of change to be usable training machine learning models.

2.3.2. Extreme Outliers

There are very big swings in highest and lower prices ranging from negative hundreds to trillions of Dollars. To ensure a more uniform training of the machine learning models, extreme outliers will need to be removed from the data.

2.3.3. Irregularities Due To Extreme Events

The main interest for this work are symbols traded between the time of 2000 1st of January till today. In that almost 23 years there were numerous extreme global events that have a big impact on the global financial markets. These irregularities will definitely challenge the prediction models.

2.3.3.1. Early 20`0s Recession, Dot-com Bubble

In the early 2000s there was a slowdown in global economics. About the same time fueled by the new hype, the internet, Dot-com Bubble, was growing and popped in 2002 October. Early 2000s Recession, Dot-com Bubble.

2.3.3.2. Great Recession 2007-2009

The Great Recession happening from 2007 was mainly caused by an overheated real estate market based speculative financial assets that had its effects on the global financial markets via disrupting the lending processes and general financial systems. Wikipedia source.

2.3.3.3. COVID Pandemic 2019-Present Day

One of the worst pandemics in the modern time. Starting from China the SARS-CoV-2 virus shook the whole globe with a global death count of over 6 million people till 2023 August. It has disrupted global markets and global supply chains causing

the biggest financial crash since the great depression. Wikipedia source.

2.3.4. Survivorship Bias

The Dataset contains information only about businesses that are survivors from the day they have been listed on the financial markets. Considering the fact that most businesses fail and that a large number of publicly traded companies were failing in the past this poses a threat to how the models trained on the Dataset would perform in a real-world scenario. This is a limitation that requires further development that is not the goal of this work.

2.3.5. Symbol Selection Based On Historical Performance

The symbols used for benchmarking and training are hand selected and selected based on past performance. Past performance never guarantees future performance and could skew the accuracy of the trained models towards unrealistic average performance in real-life production scenarios.

2.3.6. Naive Backtesting Because Of Limited Scope And Data

Backtesting for trading simulation is a very complex challenge. It is very important though to do it properly as in reality trades are not always happening as they have been planned and not always on the planned prices plus there are always some kind of fees attached to every trade or holding assets for a certain amount of time (Like interests on borrowing). However this work does not focus on building or using a complex backtesting system, therefore the very simple backtesting.py framework is used. Also considering that there is not enough data in the dataset as there are only daily prices; all trades will be fulfilled on the planned prices and some fee penalty will be added to the final result.

3. Baseline

In order to determine the performance of the different financial forecast techniques a few simple baselines will be chosen and every model performance will be compared directly against the baselines. There will be at least one baseline for the ETFs and at least one baseline for the Stocks will be selected and quantified.

Why to use multiple baselines? ETFs and Stocks are providing information about different market segments, geographical locations, financial assets classes and companies performance. Generally the S&P 500 in the US is a general benchmark for trading performance, yet it is not representing the whole US or Global economy directly. Having multiple baselines allows the

possibility to correlate the performance of different sectors and measure if a model is more successful in different market conditions.

3.1. Exploratory Data Analysis EDA

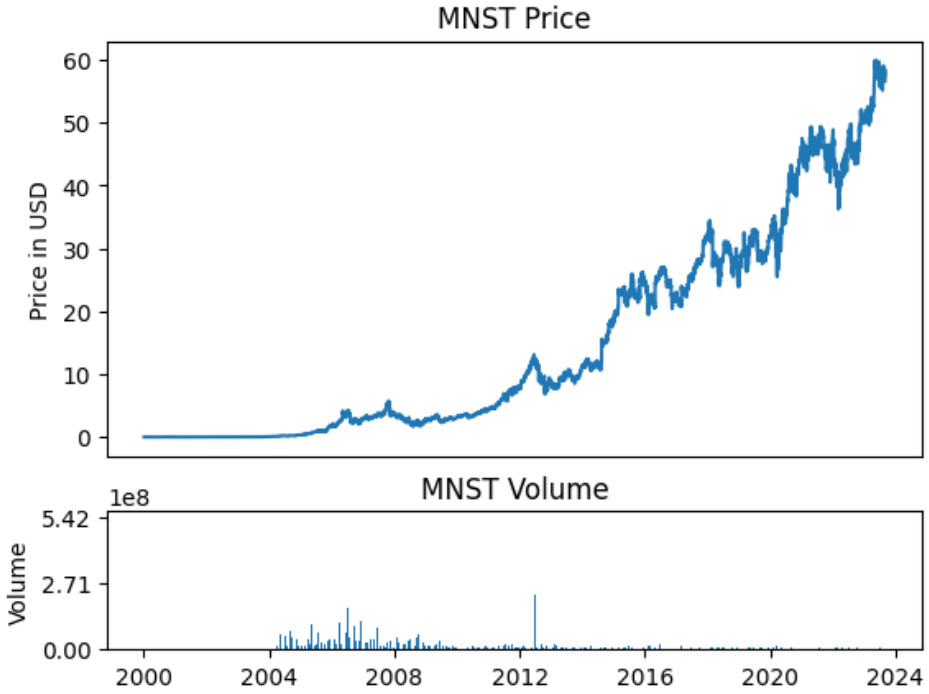
There are well over 12000 symbols in the Dataset, however only the symbols having data at least from 1st of January 2000 (That will be 3rd of January 2000 Monday) are useful symbols for this work. After filtering the symbols, there are 32 interesting ETFs and there are 1856 interesting Stocks remaining.

During the EDA process there were several symbols found that had negative price values. It might be an error in the data or those symbols were really traded on negative prices. Unfortunately zero and negative prices would interfere with the baseline selection process. Because of this all of those symbols got removed.

These symbols were: ['SID', 'VHI', 'SEE', 'ELA', 'NVO', 'AIV', 'ERIC']. After their removal, There are 32 ETFs remaining and there were no ETF affected by zero or negative prices. And there are 1849 Stocks remaining telling that all 7 symbols were Stock symbols. This totals to 1881 symbols in total.

The first trading is different for all symbols and the interesting time frame starts at 1st of January 2000. After dropping all data rows that contain data before 1st of January 2000 the dataset became uniform with each symbol having 5955 data rows.

It is interesting to see that the best performing security was not a tech stock for the almost 23 years period. The best performing stock was MNST - Monster Beverage Corporation with a maximum advance of 1333x (133,331% growth) this is a 40% yearly return.



The full EDA process could be found in the 3_1_exploratory_data_analysis PDF or jupyter notebook found in the notebooks folder.

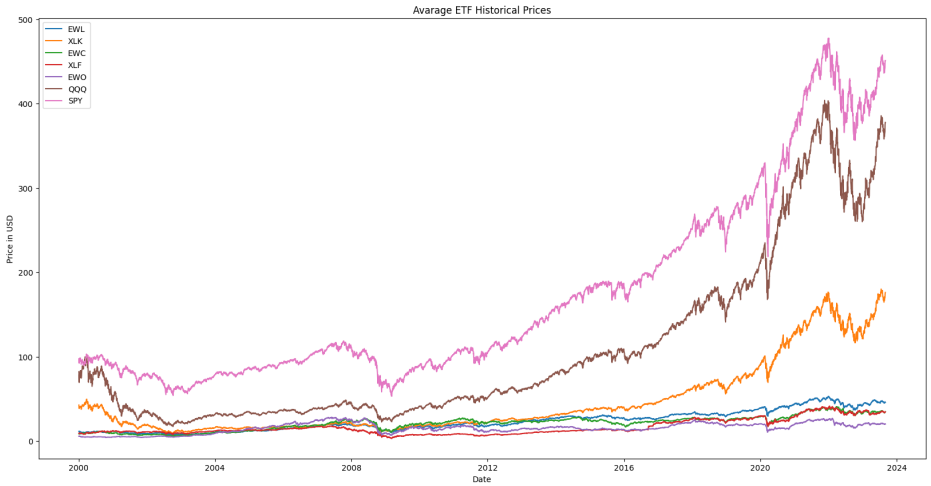
3.2. Determine The Reference Performance

To determine the reference performance that will be used for the comparison of the results of the different pricing/trading strategies, the preselected average ETFs and Stocks will be investigated with a few additional stocks that are more mainstream and known. The additional stocks are KO - Coca-Cola Company, AAPL - Apple Inc., SHEL - Shell PLC American Depositary Shares, IBM - International Business Machines Corporation. The former securities were selected only because they are mainstream companies known by the majority of the western world.

3.2.1. ETFs' Performance

The 32 ETFs on average had a 8% yearly return for the time period. The 7 ETFs closest to that return got selected as baseline candidates.

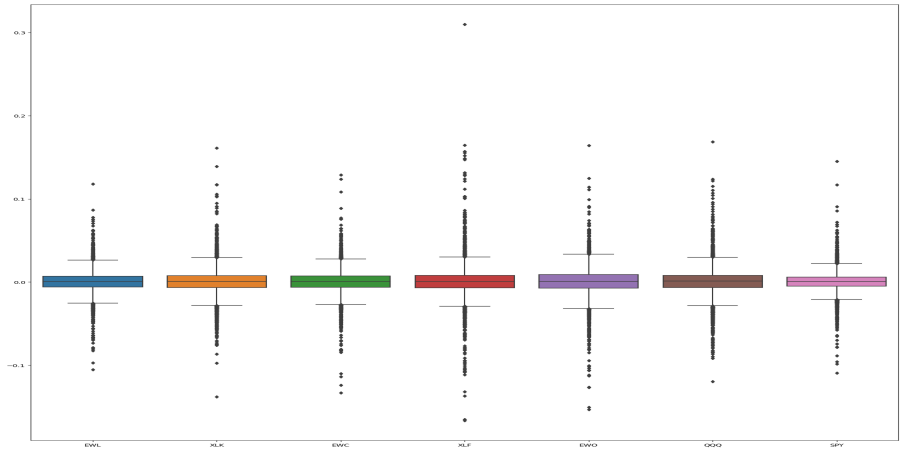
Symbol	**Security Name**
EWL	iShares Inc iShares MSCI Switzerland ETF
XLK	SPDR Select Sector Fund - Technology
EWC	iShares MSCI Canada Index Fund
XLF	SPDR Select Sector Fund - Financial
EWO	iShares Inc iShares MSCI Austria ETF
QQQ	Invesco QQQ Trust, Series 1
SPY	SPDR S&P 500



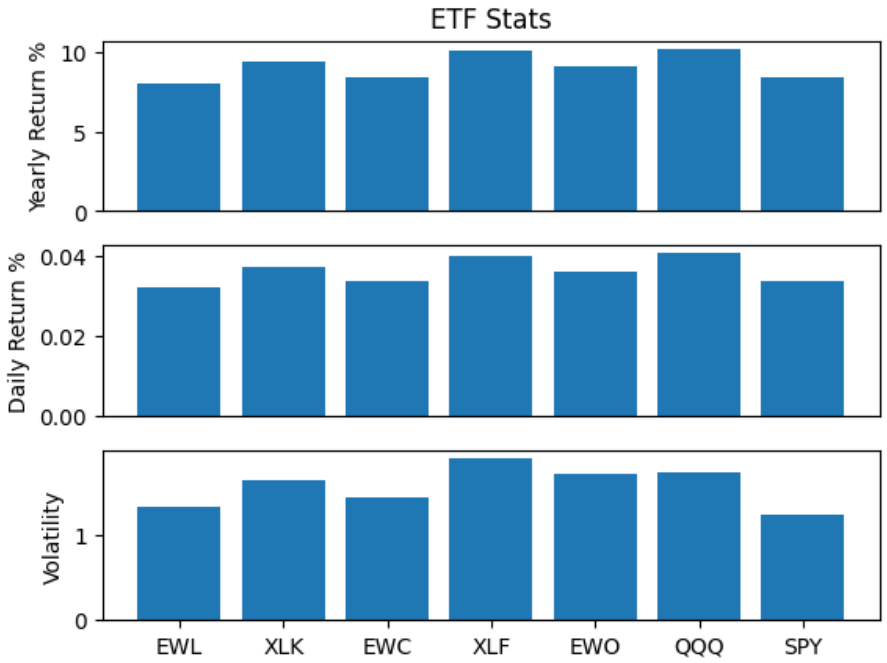
Plotting the data shows the 3 big crashes mentioned in section 2.3.3.

The correlation matrix confirms what is visible on the previous line chart that QQQ and XLK, the two technology ETFs, correlate the best with each other. Otherwise there is some weak correlation that reflects how these big mainly Index funds follow the same assets with minor differences.

Based on the plots above it is easy to see how technology and finance related ETFs generated the best return from the average ETFs yet they have very different risk profiles as it will be visible from the following plots and table.



QQQ and XLF have a higher variance, a higher risk associated with them.



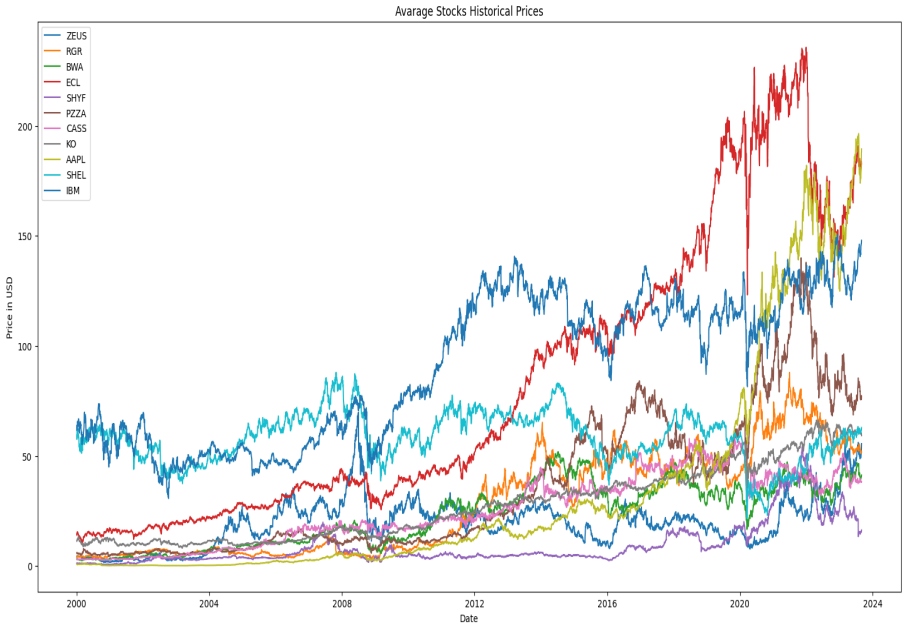
Symbol	Yearly Return %	Daily Return %	Volatility
EWL	8.07571%	0.032046%	1.317572
XLK	9.371595%	0.037189%	1.639194
EWC	8.456826%	0.033559%	1.437909
XLF	10.092181%	0.040048%	1.897438
EWO	9.096845%	0.036099%	1.705665
QQQ	10.210945%	0.04052%	1.72907
SPY	8.428791%	0.033448%	1.241737

Although all of the above volatilities are very low and therefore all of them are very safe assets. However because of the generality of SPY and non be vulnerable to any specific sector (QQQ technology, XLF finance) the baseline will be determined by SPY at a 8,5% yearly return on average.

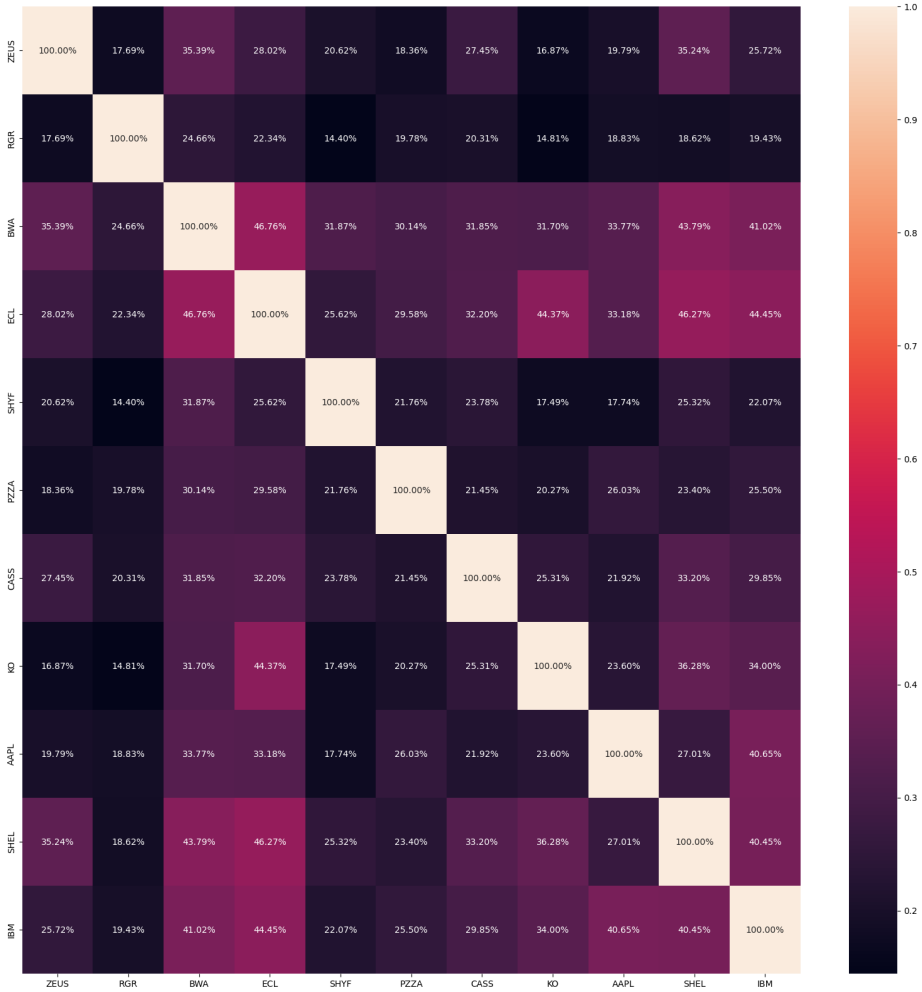
3.2.2. Stocks' Performance

From the original data there were 1849 stocks that had enough data to keep them in the dataset. With the same process as done on ETFs 7 symbols with the most average yearly return around 11% got selected. Additionally 4 additional well known securities symbols got added.

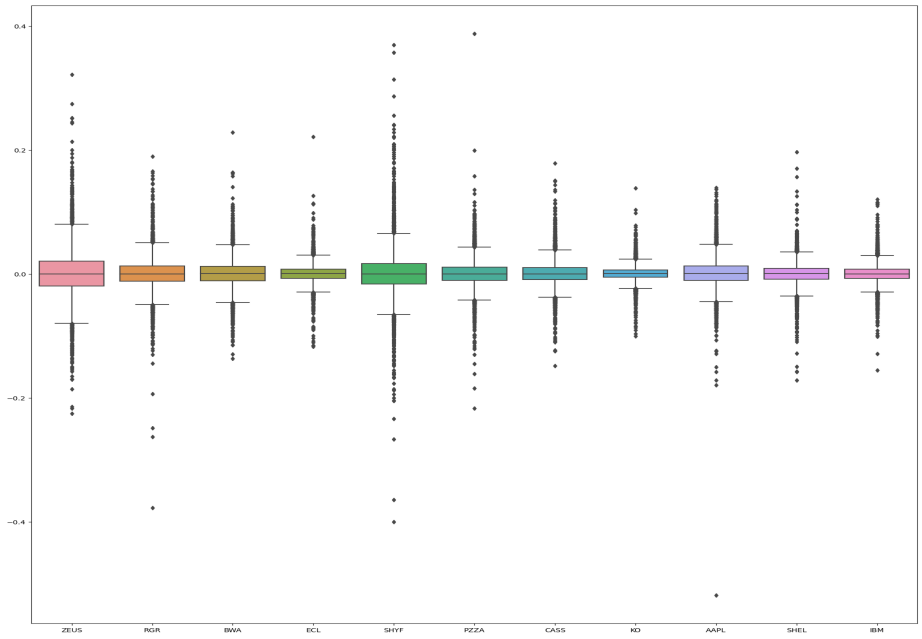
Symbol	Security Name
ZEUS	Olympic Steel, Inc. - Common Stock
RGR	Sturm, Ruger & Company, Inc. Common Stock
BWA	BorgWarner Inc. Common Stock
ECL	Ecolab Inc. Common Stock
SHYF	The Shyft Group, Inc. - Common Stock
PZZA	Papa John's International, Inc. - Common Stock
CASS	Cass Information Systems, Inc - Common Stock
KO	Coca-Cola Company (The) Common Stock
AAPL	Apple Inc. - Common Stock
SHEL	Shell PLC American Depositary Shares (each rep...
IBM	International Business Machines Corporation Co...

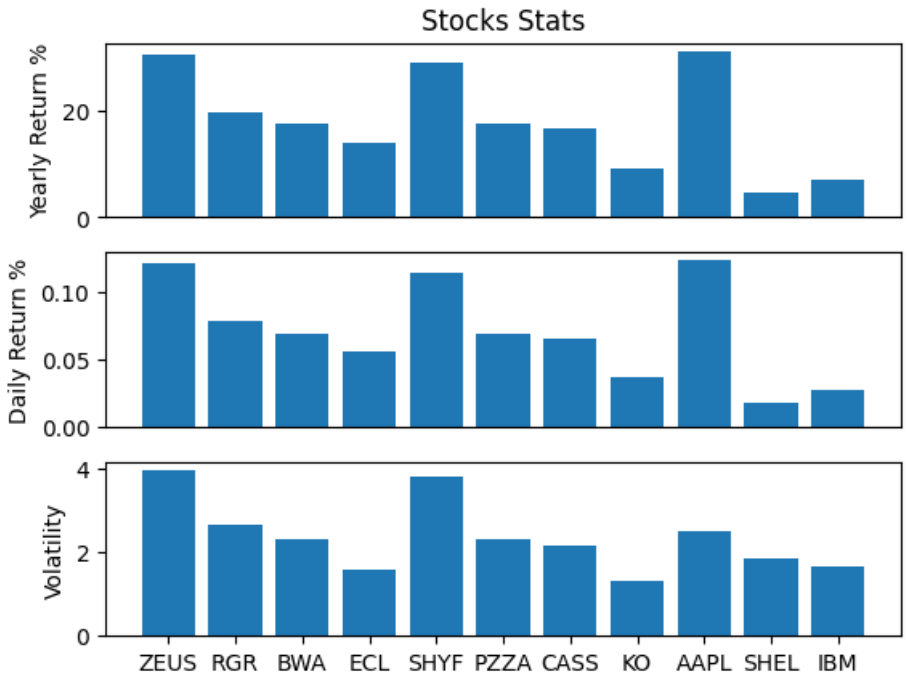


The big crashes are less visible than on the ETFs chart.



Not surprisingly there is almost no correlation between these symbols except the general global economic growth that has an increasing tendency for all symbols. Here it is very important to point back to 2.3.4 about survivorship bias as all these companies have survived this time period and there is no asset that went bankrupt.





Symbol	Yearly Return %	Daily Return %	Volatility
ZEUS	30.343988%	0.120413%	3.95851
RGR	19.643053%	0.077949%	2.633768
BWA	17.382415%	0.068978%	2.296525
ECL	13.900117%	0.055159%	1.584878
SHYF	28.783838%	0.114222%	3.791752
PZZA	17.4286%	0.069161%	2.303858
CASS	16.497573%	0.065467%	2.144626
KO	9.165557%	0.036371%	1.312269
AAPL	30.968136%	0.122889%	2.486965
SHEL	4.556985%	0.018083%	1.847852
IBM	6.952034%	0.027587%	1.639524

Interesting to see how much higher the average yearly return for stocks and how much higher the average return for the stocks.
First reason is that ETFs are derived from other assets and that smooths out sharp changes making the losses smaller yet also reducing the possible gains. Another reason is that the ETFs examined in this work are long running mature ETFs that will not hold small cap or risky assets but rather mature companies that have much smaller growth potential. The growth of penny stocks like MNST -

Monster Beverage Corporation with its exceptional results will drag the average upwards yet as we have a massive survivorship bias as mentioned earlier there is nothing to pull down that mean value.
Another way would be to base the measured stocks around the median overall return or removing the outlier above 2 standard deviations from the dataset.
Based on the data inspected, set the average yearly return for 15% that any trading method should beat to be meaningful as focusing on single stocks highly increases the overall risk.

The full reference exploration process could be found in the 3_2_determine_the_reference_performance PDF or jupyter notebook found in the notebooks folder.

3.3. The Baselines

All the models will be evaluated for two different metrics. As a general baseline of **8,5%** to beat the SPY index. And **15%** to beat the average winning stock performance.

4. Traditional Indicator and strategy

4.1. Moving Average Convergence/Divergence (MACD) Strategy

MACD is a simple strategy that is designed to measure the changes of a trend. The strategy calculated from three time-series that are derived from the historical price data.
To create three series needed to be calculated. MACD series, that is a difference between two Exponential Moving Average or in Pandas term ewm function (exponentially weighted (EW) calculations) and its mean, a fast and slow variant. And the signal that is the Exponential Moving Average of the signal itself.
The strategy should act when the MACD and the signal series are crossing each other. In case when the signal comes top on the MACD we predict that the price will drop and we predict a price increase when the MACD comes top the signal.

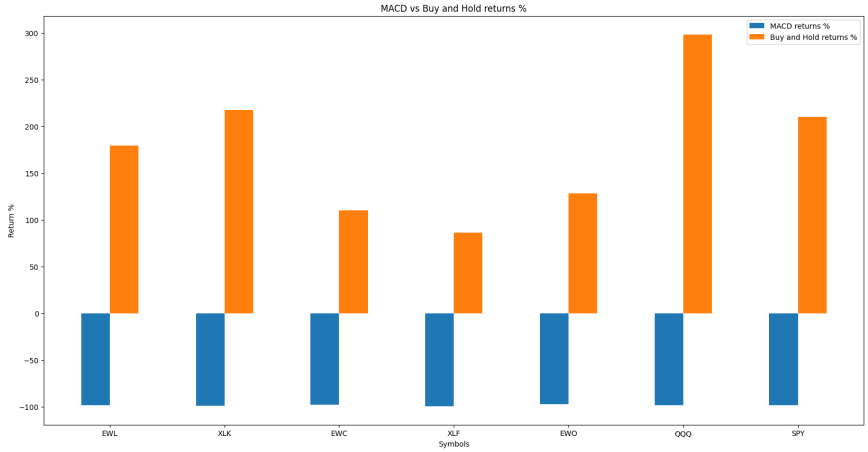
The implementation of the MACD strategy is done with the backtesting.py python package. The MACD helper function calculates the different series used by the strategy.

```
from backtesting import Backtest
from backtesting import Strategy
from backtesting.lib import crossover

# Define function for MACD
```

```
def MACD(series, long=24, short=12, signal=9):
    shortEMA = series.ewm(span=short, adjust=False).mean()
    longEMA = series.ewm(span=long, adjust=False).mean()
    MACD = shortEMA - longEMA
    MACD_signal = MACD.ewm(span=signal, adjust=False).mean()
    return (MACD, MACD_signal)

class MACDCrossover(Strategy):
    # set defaults
    long = 31
    short = 12
    signal = 9
    short_sell = False

    def init(self):
        # create MACD and MACD_signal series
        (self.macd, self.macd_signal) = self.I(MACD,
pd.Series(self.data["Adj Close"]), long=self.long,
short=self.short, signal=self.signal)

    def next(self):
```

4.1.1. Testing MACD Against The Selected ETFs

Running the base MACD strategy against the ETFs produces very unsatisfying results.



It loses basically all of the initial investment for all traded assets.

Symbol	Return %	Buy & Hold Return %
EWL	-98.44919680603029	179.42205508852186
XLK	-98.98498429967884	217.83539472061727
EWC	-97.88843586021429	110.10526499353854

XLF	-99.25493713322659	86.68148413193785
EWO	-96.93459817108169	128.39437135508362
QQQ	-98.50742363651275	298.51186948590043
SPY	-98.25684076385511	210.22948169585308

It is clear to see that it would have been a very bad strategy to rely on and based on that result MACD should be avoided.

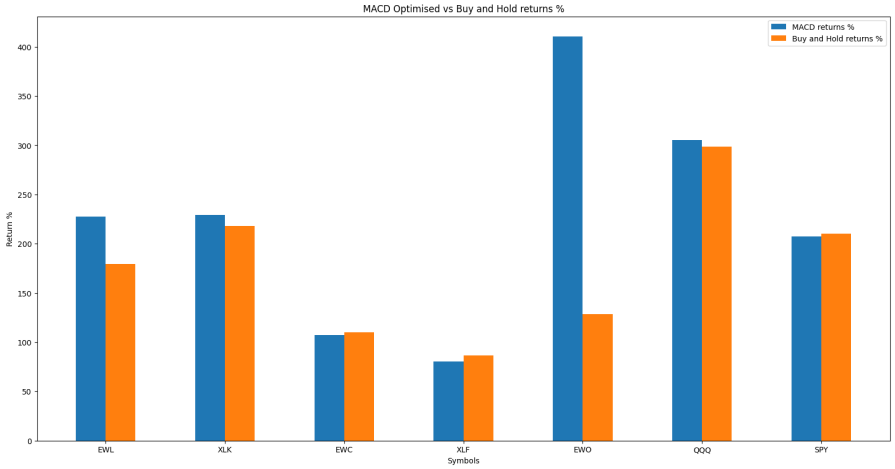
4.1.2. Optimising MACD Against The Selected ETFs

Optimisation with backtesting.py is a grid search process where all combinations of a predefined parameter set are tested and measured against a specific metric like the total return percentage.

```
stock_bt_stats = {}
for symbol in Stock_symbols:
    bt = Backtest(Stocks[symbol], MACDCrossover, cash=100000,
commission=0.02)
    stats, heatmap = bt.optimize(
        long = range(31, 252, 5),
        short = range(12, 63, 5),
        signal = range(9, 63, 5),
        short_sell = [0, 1],
        maximize="Equity Final [$]",
        return_heatmap=True
        )
    stock_bt_stats[symbol] = {
        "bt": bt,
        "stats": stats,
        "heatmap": heatmap
        }
```

Optimize testing through all the different combinations of the long, short, signal and short_sell options. It is a very CPU intensive process and with a step of 5 for the parameters the backtester has to run 10090 simulations.

The results are much better than with the default parameter.



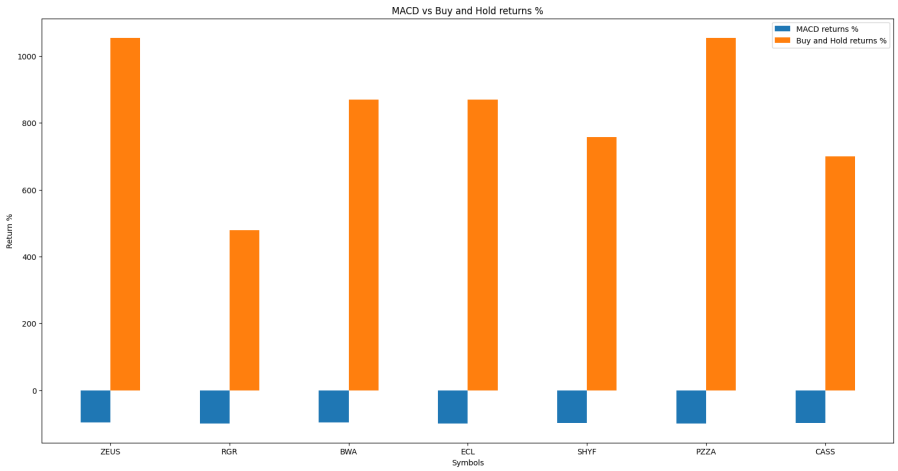
The optimised strategies are able to keep up with the Buy & Hold strategy, the baseline or even outperform in some cases. Yet there are many caveats that are visible on the next table.

Symbol	Return % VS B&H %	long	short	signal	short_sell
EWL	26.9196050254027	36	22	34	1
XLK	5.10345469608488	31	17	14	1
EWC	-2.58418400620938	146	27	9	1
XLF	-7.25420235019794	71	62	59	1
EWO	219.501561714372	166	62	49	0
QQQ	2.22027361098567	36	12	9	1
SPY	-1.41534209086522	31	12	9	1

From the table it is visible that most of the parameters were within a close range around the Buy & Hold Strategy. However all results came from a very different parameter set and most of them have short selling enabled and that increases the risk of the strategy.
Yet one strategy did very well, basically tripling the original Buy & Hold return without the use of short selling.

4.1.3. Testing MACD Against The Selected Stocks

Running the base MACD strategy against the Stocks produces the same result as with the ETFs.



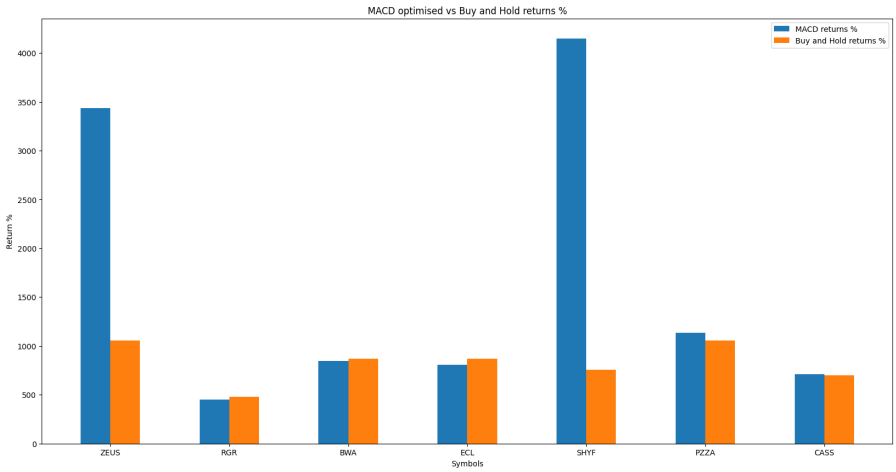
Generally losing all the invested money.

Symbol	Return %	Buy & Hold Return %
ZEUS	-96.70439065071116	1054.9091289569806
RGR	-98.97848049416548	479.3566296984267
BWA	-96.71922138119723	869.9348387782582
ECL	-98.92166444984434	869.5522226109038
SHYF	-97.26235523212448	757.6933011787811
PZZA	-99.14073104592327	1053.644882629965
CASS	-98.21441769619965	699.6538482642471

This would have been a very bad strategy to run.

4.1.4. Optimising MACD Against The Selected Stocks

Optimisation with backtesting.py on the selected Stocks dataset provides similar results as the optimised ETFs MACD tests and the same caution applies to these results as well.



These results are similar to what the ETFs optimisation achieved however with the stocks the two outliers are even better than with ETFs.

Symbol	Return % VS B&H %	long	short	signal	short_sell
ZEUS	225.624346069384	121	47	34	0
RGR	-6.03845829303592	51	47	9	1
BWA	-2.7540152344401	31	57	9	1
ECL	-7.0816064727002	31	12	9	1
SHYF	447.109061177292	36	47	39	1
PZZA	7.95107057179447	31	17	44	1
CASS	1.54610156414505	31	12	9	1

Checking the results, the result of ZEUS actually looks promising without any short selling. Though all the rest except SHYF make no gain with a lot of introduced risk with shorting the assets.

4.1.5. Additional MACD Tests And Full Notebook

Some additional tests were done with the handpicked symbols though they have been very chaotic. It would be a great research on its own how those assets were handpicked alters the way MACD behaves.

The full reference exploration process could be found in the 4_1_MACD_strategy PDF or jupyter notebook found in the notebooks folder.

4.2. Results

The traditional MACD strategy results are up for some interpretation and the results could be interpreted both ways.
With the default MACD Crossover parameters the strategy has lost almost all the money in all cases. This is a very simple result to interpret as a massive failure and declare based only on that MACD strategy does not even come close to the baselines.

The optimised versions of MACD strategy had some good results, yet those results massively overfit the actual data and there is no single case where different best returns are using the same parameters.

Another thing to mention is that there are hundreds of technical indicators and strategies not only MACD and with time and enough testing there could be parameters and combinations of different strategies that should be able to generate at least the return of the Buy & Hold Strategy. However even in this case the risk profile might be very different and make those strategies inferior to a Buy & Hold strategy.

As a final result this MACD strategy was not up to the level of the baseline.

5. Artificial Intelligence Based Indicator And Strategy

5.1. Recurrent Neural Network Based Indicator And Strategy

Recurrent Neural Networks especially LSTMs (Long Short-Term Memory) networks are ideal for time-series analysis and forecasting tasks.
Train multiple RNN models on the SPY data and observe how it is capable of forecasting future price and if it is useful for forecasting future price. As in every previous part the training and testing will happen on the adjusted daily closing price of the symbol.

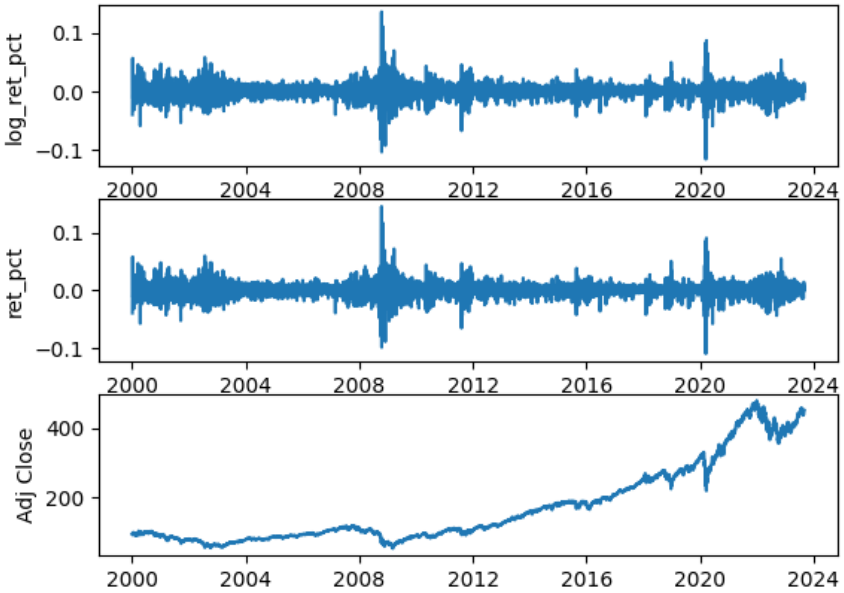
5.1.1. Prepare Data

In the dataset prices are having a very high range with too high fluctuation and that is not ideal for neural networks. To tackle this problem the models will be trained on engineered features next to the scaled prices. The engineered feature is logarithmic scaled daily return percentage change. This is done to introduce log normality [2] into the price changes.
Observe how adjusted close prices have a very high standard deviation and mean and logarithmic return percentage mean is very close to zero and has a very low standard deviation.

```
SPY.describe()
```

	Adj Close	ret_pct	log_ret_pct
count	5954.000000	5954.000000	5954.000000
mean	171.338935	0.000334	0.000257
std	112.609567	0.012417	0.012426
min	53.155308	-0.109424	-0.115887
25%	88.294670	-0.004804	-0.004815
50%	112.451721	0.000651	0.000650
75%	233.685982	0.006019	0.006001
max	477.709991	0.145198	0.135577

This is very well visible on the following plots as well.



The data got scaled with MinMaxScaler to set all the two used feature ranges between zero and one.

LSTMs need their input in the form of (number of observations, time window (the past data), number of features). In this case the selected history is 63 datapoints. This roughly along with an average of 3 months of data points. Therefore the RNN model will have 3 months of previous daily prices and daily logarithmic return percentage to estimate the future time step price.

```
time_step = 63
```

[16]:
```
y = SPY_scaled[time_step: ,0]
```

[17]:
```
X = []
for i in range(time_step, len(SPY_scaled)):
    X.append(SPY_scaled[i - time_step: i])

X = np.array(X)
```

[18]:
```
X.shape
```

[18]: (5891, 63, 2)

5.1.2. Create RNN Models

The model is a Deep Neural Network with 2 LSTM layers and 2 Dense layers.

```
observation, step, features = X.shape
rnn = Sequential([
    LSTM(units=32,
         input_shape=(step, features),
         name="LSTM_1",
         dropout=0.2,
         recurrent_dropout=0.2,
         return_sequences=True),
    LSTM(units=16,
         name="LSTM_2",
         dropout=0.2,
         recurrent_dropout=0.2),
    Dense(10, name="DENSE_1"),
    Dense(10, name="DENSE_2"),
    Dense(1, name="Output")
])
```

```
rnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 LSTM_1 (LSTM)               (None, 63, 32)            4480

 LSTM_2 (LSTM)               (None, 16)                3136

 DENSE_1 (Dense)             (None, 10)                170

 DENSE_2 (Dense)             (None, 10)                110

 Output (Dense)              (None, 1)                 11

=================================================================
Total params: 7907 (30.89 KB)
Trainable params: 7907 (30.89 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

This results in a small model with 7907 parameters.

5.1.3. Train Models

Two copies were created from the previous RNN model to train them with different loss functions and optimizers.

```
rnn_1 = keras.models.clone_model(rnn)
rnn_2 = keras.models.clone_model(rnn)

rnn_1.compile(loss='mean_squared_error', optimizer="adam")
rnn_2.compile(loss='mae', optimizer="RMSProp")
```

The training process happened with TensorBoard Checkpointing and Earlystopping.

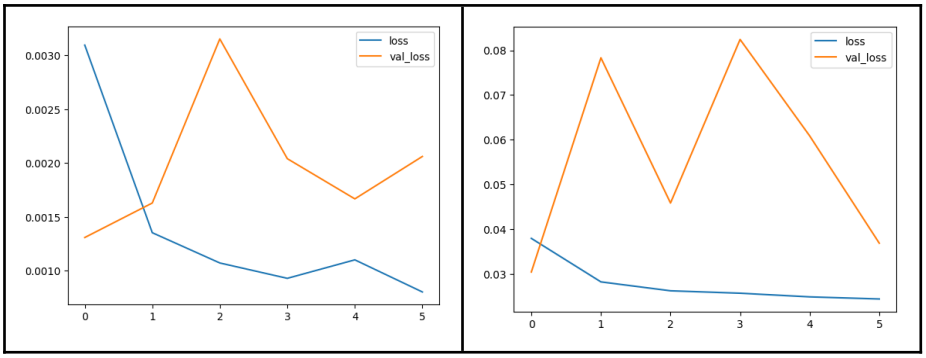
```
metric = 'val_loss'
rnn_path = "../models/rnn_1_spy_2.weights.best"
tb_path = "../tensorboard/rnn_1_spy"

tensorboard = TensorBoard(log_dir=tb_path,
                          histogram_freq=0,
                          write_graph=True,
                          write_images=True)

early_stopping = EarlyStopping(monitor=metric,
                               patience=5,
                               restore_best_weights=True)
```

```
checkpointer = ModelCheckpoint(filepath=rnn_path,
                               monitor=metric,
                               save_best_only=True,
                               save_weights_only=True,
                               save_freq="epoch")

result_1 = rnn_1.fit(X_train,
                 y_train,
                 epochs=100,
                 batch_size=32,
                 validation_data=(X_val, y_val),
                 callbacks=[checkpointer, early_stopping,
tensorboard],
                 verbose=1)

Epoch 1/100
166/166 [==============================] - 6s 26ms/step - loss:
0.0031 - val_loss: 0.0013
Epoch 2/100
166/166 [==============================] - 4s 24ms/step - loss:
0.0014 - val_loss: 0.0016
Epoch 3/100
166/166 [==============================] - 4s 24ms/step - loss:
0.0011 - val_loss: 0.0032
Epoch 4/100
166/166 [==============================] - 4s 24ms/step - loss:
9.3095e-04 - val_loss: 0.0020
Epoch 5/100
166/166 [==============================] - 4s 24ms/step - loss:
0.0011 - val_loss: 0.0017
Epoch 6/100
166/166 [==============================] - 4s 24ms/step - loss:
8.0455e-04 - val_loss: 0.0021
```

5.1.4. Training Results

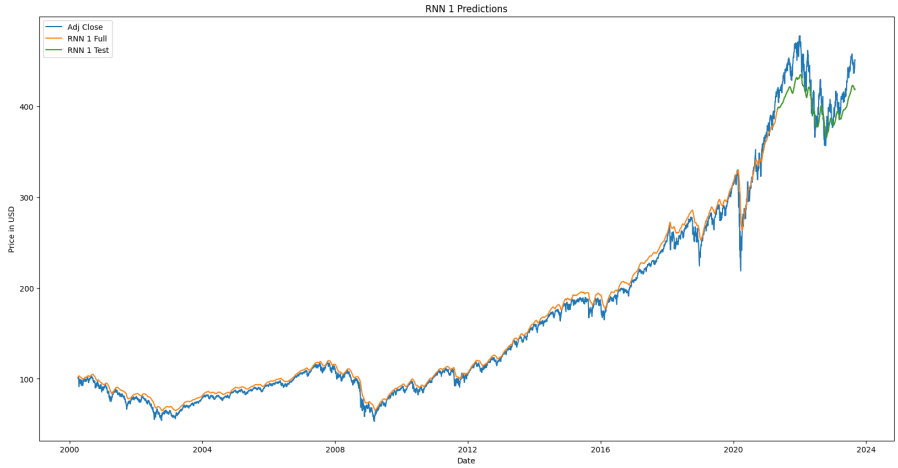
RNN 1 with the Mean Squared Error loss function and Adam optimizers performed better in all training cases.
RNN 1 vs RNN 2 training history

RNN 1	RNN 2
Training Error 1: 0.0170	Training Error 2: 0.1275



5.1.5. Testing RNN 1 Predictions

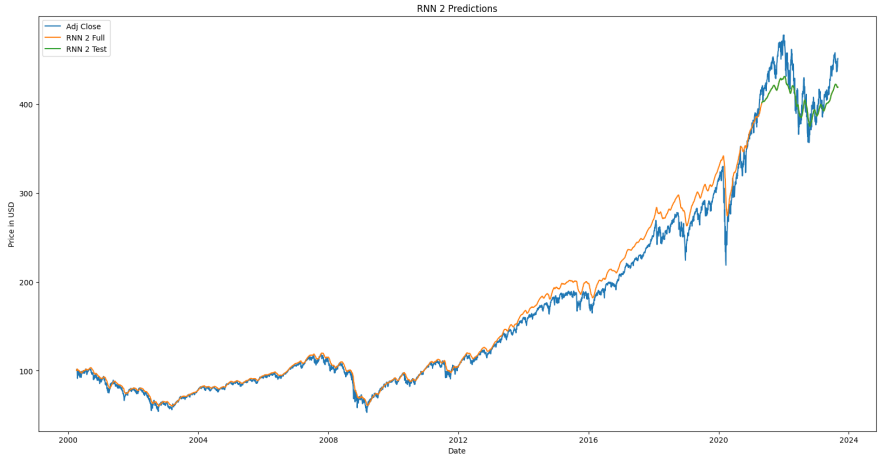
RNN 1 resulted in 0.0170 training error and 0.0536 test error.



RNN 1's predictions follow fairly well the curvature of the real data.

5.2. Testing RNN 2 Predictions

RNN 2 resulted in 0.1275 training error and 0.2090 test error.



It is clearly visible by the error scores and graphs that RNN 2 follows the curvature of the original data but performs much worse than RNN 1 does.

5.2.1. Compare Predictions

The following graphs show that both models achieve similar results but RNN 1 is slightly better and it was consistently better throughout all tests.

5.3. Results

It is easy to see that LSTMs are easy to fit and use to predict short-term forecasts; however, it would need much more training and extensive grid searching and comparison of different neural network architecture to achieve better results. Based on the predictions it is assumable that with the current prediction capabilities it would not outperform the baseline significantly.

6. Conclusion

Predicting the markets is not an easy task. Even some could argue it is practically not possible. Burton Malkiel, an economist from Princeton University write in his book from 1973 “A Random Walk Down Wall Street,” that if the market is truly efficient and a share price reflects all factors immediately as soon as they’re made public, a blindfolded monkey throwing darts at a newspaper stock listing should do as well as any investment professional.
However not all market participants are fully informed and not all market participants are capable to oversee (if any capable) all the intricate connections in our global economy therefore it is possible to uncover special relations within the global economy.

In this work only a few techniques were experimented and tested however it is more than likely that with the combination of the tools and testing the whole dataset together and looking for correlations would generate useful result and would be able to unlock metric that could indicate market movement better than the single financial measures.

6.1. Final Conclusion

In this work none of the methods were able to generate a consistent, lower risk and better result than the Buy And Hold strategy baseline return.

7. References

[1] Kaggle Stock Market Dataset - https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset
[2] Log Normality - https://quantivity.wordpress.com/2011/02/21/why-log-returns/
[3] S&P 500 - https://en.wikipedia.org/wiki/S%26P_500
[4] MACD - https://www.investopedia.com/articles/forex/05/macddiverge.asp
[5] RNN - https://en.wikipedia.org/wiki/Recurrent_neural_network
[7] LSTM - https://en.wikipedia.org/wiki/Long_short-term_memory