William Santos
https://wsantos.io
william.santos@rhul.ac.uk
School of Engineering, Physical, and Mathematical Sciences
Royal Holloway, University of London

Automated Trading Systems: Support/resistance line detection using mathematical optimisation

ABSTRACT

Automated Trading Systems generate buy / sell orders in response to rapidly changing markets by consulting a decision making engine or knowledge base - which may or may not evolve over time in order to better reflect changing markets. An order, during its lifetime, can be fully filled, partially filled, rejected (by the exchange), revoked (by the system), or simply adjusted. The objective for any trading system is to, of course, manage the changing states of orders, and to correctly forecast the direction of a market - to maximise buy orders at a low price and sell orders at a high price. Rudimentary trading systems use technical indicators and if-then-else logic to determine when to execute orders. Few systems pay attention to market support/resistance levels and volume.

Support and resistance lines define price levels whereby the price of a security tends to stop and reverse. There exists no general algorithm for calculating S/R levels since the lines cannot be calculated in the same way a typical technical indicator can be - and thus are often plotted by hand.

In this paper, I find S/R levels algorithmically using mathematical optimisation, and discuss an approach to integrating S/R levels with an evolutionary trading system. I evaluate the quality of the S/R levels found by the algorithm by comparing them to initial benchmark lines plotted manually.

Keywords and phrases

Support resistance lines, automated trading, machine learning, statistics, mathematical optimisation.

INTRODUCTION

Financial securities, and specifically cryptocurrencies, are traded night and day by forign currency traders and Automated Trading Systems. The objective is to, of course, correctly forecast the direction of a market in order to maximise profits. One places limit, market, or stop orders on a particular exchange at particular price levels at specific times - following a calculated and very much finely-tuned trading strategy. Such decision making strategies fall into one of the two following categories:

Techniques based on fundamental analysis

Fundamental techniques focus purely on the inherent value of a specific currency - which can be affected by many variables *(Garikai 2015)*. For example: current and past trends in the economy, political climate, general sentiment surrounding the currency, inflation, major events that have occurred / are going to occur in the near future - such as natural or biological disasters. It is difficult to measure and model data surrounding these variables using mathematical methods - thus automated systems tend to primarily use technical indicators. One fundamental technique that *can* be automated involves tuning into real-time textual data streams and performing sentiment analysis - to forecast large price movements *(Kim YB, Lee SH, Kang SJ, Choi MJ, Lee J, Kim CH 2015)*. Such data streams could be accessed via online social media platforms, for example.

Techniques based on technical analysis

Technical techniques ignore fundamental information. Technicals generate buy / sell signals based only on historical price data and market activity *(Freire, Rosane & Fernandes, Cristiano & Lorenzoni, Giuliano & Pizzinga, Adrian & Atherino, Rodrigo 2007)*. Most technical

indicators are graph functions - and are plotted against candlestick data. Other technical indicators include fibonacci retracement levels *(Bhattacharya & Kumar 2006)* and support and resistance levels - which are plotted manually. The vast majority of automated trading systems calculate and respond to multiple technical indicators on live exchange data. A simple (and perhaps slightly naive) system could work as follows:

1. calculate two exponential moving averages (EMAs) - one with a small (short-term) window and another with a larger (long-term) window.
2. The short-term EMA passing *above* the long-term EMA indicates a good buy opportunity.
3. The short-term EMA falling *below* the long-term EMA indicates a good sell opportunity.

This is called the crossover strategy *(Praekhaow 2010)* and is relatively straight-forward to implement.

Which techniques and tools to use depends on many different factors, and is ultimately a million dollar question. In this paper, I focus only on technical analysis - specifically the use of support and resistance levels, and defining a method for calculating them automatically. The algorithmically generated S/R lines could then be used by a human trader to make buy/sell decisions.

In this paper, I do not create and deploy a functioning automated trading system. However, I do discuss how an automatic evolutionary system could be implemented. I am

interested in calculating suitable S/R levels algorithmically. I do test my model against live market data, but I do not execute buy / sell orders on a real exchange.

Manual trading with support and resistance levels

Firstly, S/R levels can only be successfully plotted by hand when a market has a clear direction - i.e. not during major upwards / downwards price movements. The market should be either trending upwards, downwards, or sideways - with clear price oscillations. One can plot a support line by drawing a line that intersects *most* of the levels where the market sees support. Likewise, one can plot a resistance line by drawing a line that intersects *most* of the levels where the market sees resistance. In the chart below, the bottom line is a suitable support line, and the top line is a suitable resistance line. The chart shows a market with a slow downwards trend. Possible buy and sell zones - known as entry and exit points - have been highlighted.

Image source: cryptowat.ch charting tool[1]

[1] https://cryptowat.ch/markets/coinbase-pro/ltc/gbp/1h

Until a new upwards or downwards trend causes the price to 'breakout' of the levels, the market will simply oscillate / bounce between the S/R levels. A buy order should be placed when the price dips below the support line, and a sell order should be placed when the price passes above the resistance line.

Building an evolutionary system

Market activity is constantly changing - with trends emerging and quickly disappearing. A train-once model - whose performance does not fluctuate or decrease over time - is not feasible. A more robust and intelligent automated trading system would continuously collect, analyse, and learn from new market data. Theoretically, building such a system would not be too difficult. An evolutionary system could work as follows:

1. Periodically poll an exchange API every hour - and collect the last hour's candlestick data. Add the new candles to the main DataFrame.
2. For the new candles, calculate appropriate S/R lines and prepare the data for the training phase.
3. Train a new model on the last 24 hours of candlestick data. This way, the oldest hour is just ignored.
4. When the model has finished training, swap at the live model with the new model.
5. The new live model more accurately represents the last day's worth of exchange activity. The model is essentially trained on a 24 hour sliding window.ll

METHOD

From hereafter this paper will focus on four key stages. Firstly, historical open, high, low, close, volume (OHLCV) candlestick data for the cryptocurrency Litecoin[2] will be collected and preprocessed. Then, suitable support and resistance levels will be manually identified and plotted - which will serve as a benchmark. After, mathematical optimisation will be used to find optimum S/R levels. Finally, the quality of the S/R levels generated by the algorithm will be evaluated by comparing them to benchmark lines plotted manually.

Data acquisition and preparation

OHLCV candlestick data will be fetched for the LTC-GBP pair from the Coinbase Pro API[3] with a granularity of five minutes and a timeframe spanning 24 hours - 300 candles in total. The candles will be loaded into a Pandas DataFrame DF and a simple moving average (SMA) / rolling mean with a window of three will be calculated over each candle's closing price. DF will then be split into six four-hour groups. For each group G, candles with a closing price above or equal to the SMA, and candles with a closing price below the SMA will be identified and collected into DataFrames $DF_{G}^{+} \subseteq DF$ and $DF_{G^{-}} \subseteq DF$ respectively.

Support and resistance: finding two suitable linear functions

A 'support level' is a line that indicates the price level whereby a security often experiences support - a positive price reversal - over a specific timeframe. Similarly, a 'resistance level' is a line that shows where a security often sees resistance - a negative price reversal - over a specific timeframe. It is important to remember that S/R levels are not 'lines of

[2] https://litecoin.info/index.php/Main_Page
[3] https://docs.pro.coinbase.com

best fit' - thus this isn't a simple linear regression problem. Since S/R levels are just lines, they can be represented by $y = mx + b$, where $x \in \mathbb{N}$ is the current candle's index and $m, b \in \mathbb{R}$ are the values to be optimised. Benchmark S/R levels will be manually plotted on the candlestick data - and will be used to evaluate the quality of the lines produced by the optimisation algorithm.

BFGS: optimisation via hill climbing

The BFGS[4] hill climbing optimisation algorithm will be used to find the optimum $m, b \in \mathbb{R}$ values for both lines. A global optimisation technique is necessary because the search space is large and contains potentially many local optimums. The quality / score of the resistance line at timestep t is given by $S_t = \sum_{i=1}^{|DF_G^+|} d(f(x), DF_{G\,i}^+)$ which is the sum of the distances of each $f(x)$ with the corresponding point in DF_G^+. The algorithm's goal is to minimise S_t by repeatedly adjusting m, b.

Testing and results

The quality of the support/resistance lines found by the optimisation algorithm will be evaluated by simply comparing them (by eye) to the S/R lines that were initially plotted against the data manually. Charts showing price data, manual S/R levels, and automatic S/R levels will be created.

[4] https://en.wikipedia.org/wiki/Limited-memory_BFGS

DATA ACQUISITION AND PREPARATION

Live candlestick data is fetched from Coinbase Pro's HTTP API. Each candlestick contains the open, high, low, close, and volume data for a 300 second (5 minute) period. The data is loaded into a Pandas DataFrame and preprocessed.

Fetching from API

```
In [348]: def fetch_candles():
              now = datetime.now()
              params = {
                  "granularity": "300",
                  "end": now.isoformat(),
                  "start": (now - timedelta(hours=24)).isoformat(),
              }
              r = get("https://api.pro.coinbase.com/products/LTC-GBP/candles", params=params)
              return r.json()
```

The request returns a JSON array of arrays - where each array represents a candle of the form: $[[\textit{opening timestamp},\ o,\ h,\ l,\ c,\ v],\ ...]$. The opening timestamp is a unix epoch timestamp. Candles for the last 24 hour period are retrieved.

Loading into DataFrame

```
In [362]: candles = fetch_candles()
df = pd.DataFrame({
    "open": [c[1] for c in candles],
    "high": [c[2] for c in candles],
    "low": [c[3] for c in candles],
    "close": [c[4] for c in candles],
    "volume": [c[5] for c in candles],
    "timestamp": [c[0] for c in candles]
})
df.head()
```

Out[362]:

	open	high	low	close	volume	timestamp
0	34.34	34.34	34.34	34.34	0.261071	1585099800
1	34.39	34.39	34.39	34.39	2.000000	1585099500
2	34.34	34.45	34.34	34.45	19.087161	1585099200
3	34.44	34.57	34.57	34.44	45.297538	1585098300
4	34.50	34.56	34.50	34.56	3.127961	1585098000

The array of candle data is loaded into a DataFrame - which will allow for easier manipulation and preprocessing later on.

Preprocessing

```
In [364]: df["datetime"] = pd.to_datetime(df["timestamp"], unit="s")
df["i"] = df.apply(lambda x: x.name + 1, axis="columns")
df["close_sma"] = df["close"].rolling(window=3).mean()
df["gt_close_sma"] = df.apply(
    lambda x: 1 if x["close"] >= x["close_sma"] else 0, axis="columns"
)
df["lt_close_sma"] = df.apply(
    lambda x: 1 if x["close"] < x["close_sma"] else 0, axis="columns"
)
df.head()
```

Out[364]:

	open	high	low	close	volume	timestamp	datetime	i	close_sma	gt_close_sma	lt_close_sma
0	34.34	34.34	34.34	34.34	0.261071	1585099800	2020-03-25 01:30:00	1	NaN	0	0
1	34.39	34.39	34.39	34.39	2.000000	1585099500	2020-03-25 01:25:00	2	NaN	0	0
2	34.34	34.45	34.34	34.45	19.087161	1585099200	2020-03-25 01:20:00	3	34.393333	1	0
3	34.44	34.57	34.57	34.44	45.297538	1585098300	2020-03-25 01:05:00	4	34.426667	1	0
4	34.50	34.56	34.50	34.56	3.127961	1585098000	2020-03-25 01:00:00	5	34.483333	1	0

A datetime field is added - which converts each candle unix epoch into a datetime string. An index column is added to the DataFrame. A candle's index is simply its position in the array plus one (not zero indexed). A simple moving average (SMA) with a time period of three is

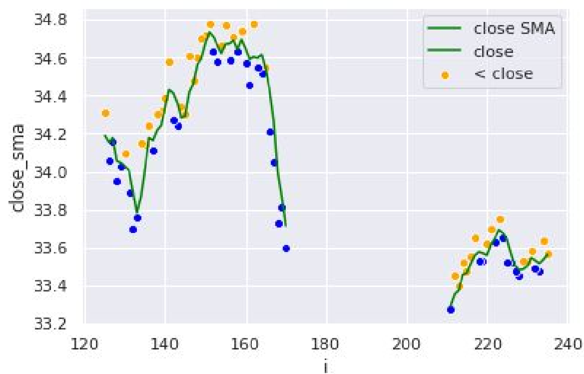
calculated over the closing prices. Closing prices that are above / below the SMA are identified and marked with a 1 by a simple lambda function.

```
In [351]: df_grouped = df.groupby(pd.Grouper(key="datetime", freq="4h"))
```

The main DataFrame *df* is grouped into six four-hour DataFrames.

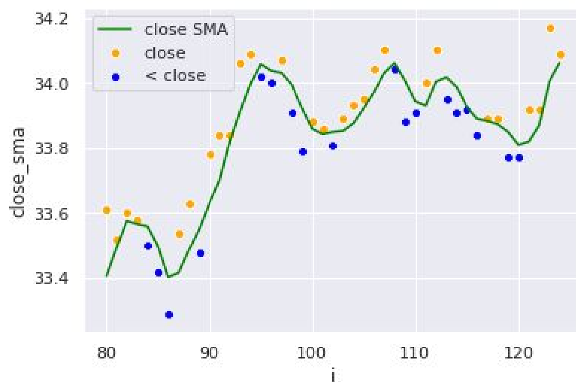
```
In [367]: for i, (_, group) in enumerate(df_grouped):
              above_avg = group.loc[df["gt_close_sma"] == 1]
              below_avg = group.loc[df["lt_close_sma"] == 1]
              sns.scatterplot(x="i", y="close", data=above_avg, color="orange")
              sns.scatterplot(x="i", y="close", data=below_avg, color="blue")
              sns.lineplot(x="i", y="close_sma", data=group, color="green")
              plt.legend(labels=["close SMA", "close", "< close"])
              plt.figure(i)
```

1

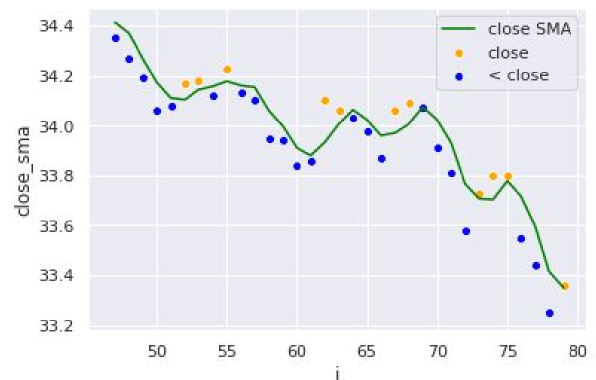


2

3

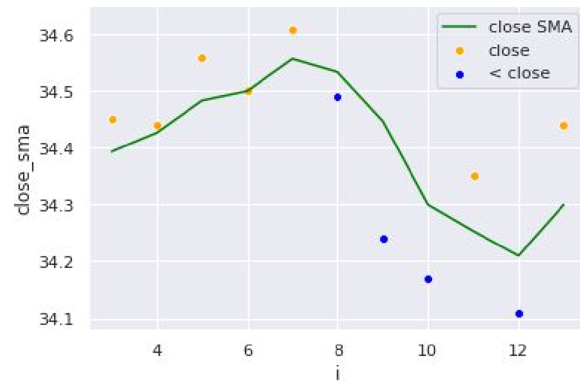


4



5

6



For each candlestick group, the above average and below average points are plotted with the SMA line - and visualised. For each group, two subsets are created - *above_avg* and *below_average* which contain candles that are above the SMA and below the SMA respectively. The points below the SMA will be used to find the support line. The points above the SMA will be used to find the resistance line.

SUPPORT AND RESISTANCE: FINDING TWO SUITABLE LINEAR FUNCTIONS

Support/resistance lines will need to be manually plotted on the data - to serve as benchmarks during the optimisation step. Candlestick data will be visualised and $m, b \in \mathbb{R}$ values will be manually adjusted until suitable S/R lines have been found.

```
In [158]: # linear function
          def f(x, m ,b):
              return m * x + b

          # simple euclidean distance
          def d(p1, p2):
              return math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)
```

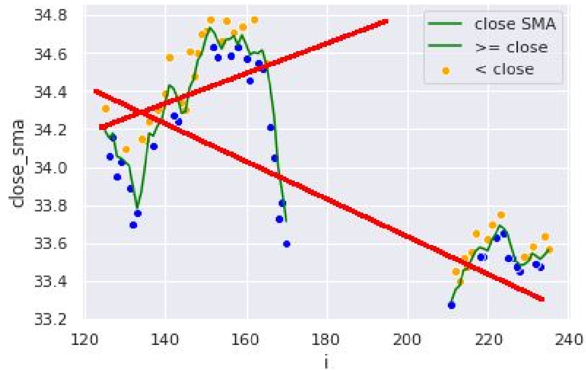
The function $f(x, m, b)$ simply computes the y value for a line at position x. The distance function $d(p1, p2)$ returns the euclidean distance between two points $p1, p2$ where p_1 is a point's x value and p_2 is a point's y value.

```
In [160]: def score(m, b, df):
              tot = 0
              line = []
              for _, r in df.iterrows():
                  y = f(r["i"], m, b)
                  p1 = (r["i"], r["close"])
                  p2 = (r["i"], y)
                  tot += d(p1, p2)
                  line.append(y)
              return tot, pd.DataFrame({ "i": df["i"], "y": line })
```

The $score$ function takes an m, b amd a DataFrame DF and computes the line and score for the provided m, b - by iterating i over all rows in the provided DataFrame. The y value for each candlestick's index x is calculated and added to a list (to be plotted). The distance between the current point $P_i \in DF$ and the candlestick (x, y) is calculated and added to a running sum.

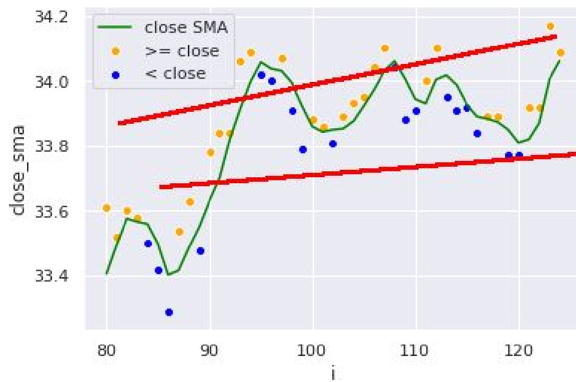
```
In [497]: for i, (_, group) in enumerate(df_grouped):
              above_avg = group.loc[df["gt_close_sma"] == 1]
              below_avg = group.loc[df["lt_close_sma"] == 1]
              s1, line1 = score(m1[i], b1[i], above_avg)
              s2, line2 = score(m2[i], b2[i], below_avg)
              sns.scatterplot(x="i", y="close", data=above_avg, color="orange")
              sns.scatterplot(x="i", y="close", data=below_avg, color="blue")
              sns.lineplot(x="i", y="close_sma", data=group, color="green")
              plt.legend(labels=["close SMA", ">= close", "< close"])
              plt.figure(i)
```

1

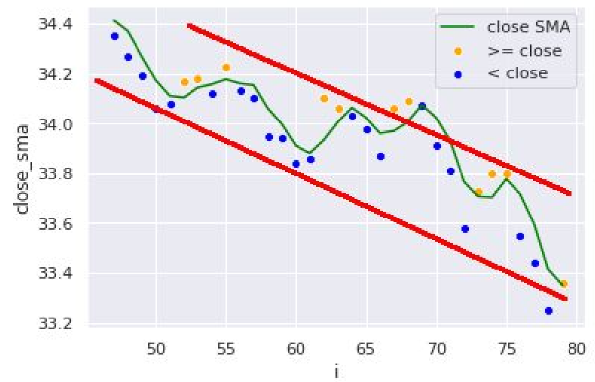


2

3

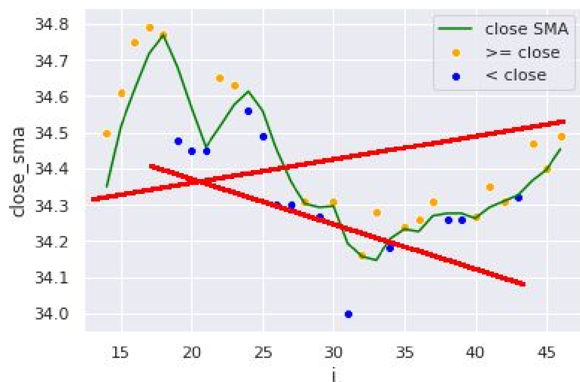


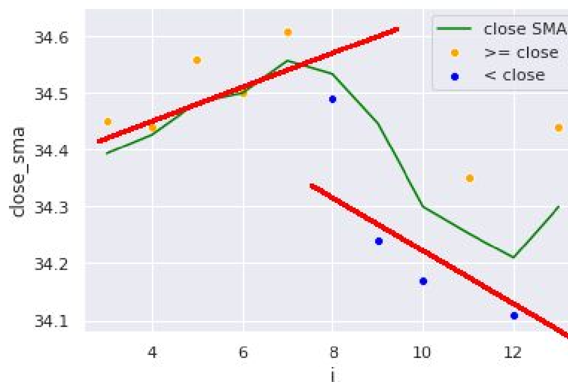
4



5

6





The benchmark support and resistance lines are calculated by manually adjusting variables m, b and trying to minimise the score value returned by *score*. Charts 1, 2, and 6 show no useful S/R levels. However, with charts 3, 4, and 5 the SMA oscillates nicely between the S/R levels.

BFGS: OPTIMISATION VIA HILL CLIMBING

The objective is to find suitable S/R lines computationally. A hill climbing optimisation algorithm is used to do this. The quality / score of the resistance line at timestep t is given by $S_t = \sum_{i=1}^{|DF_G{}^+|} d(f(x), DF_G{}^+{}_i)$. The optimisation algorihtm's goal is to minimise S_t - to produce a support line and resistance line that is nearest to all pointes below and above the SMA line, respectively.

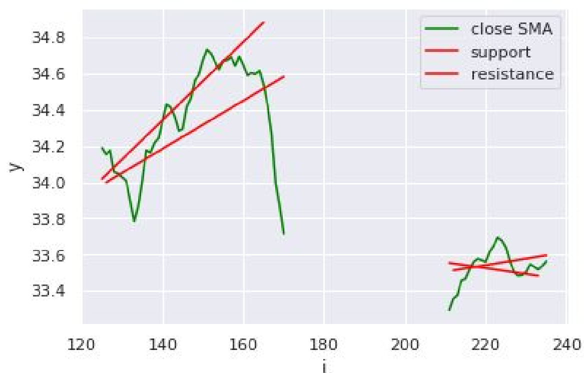
```
In [633]: def find_sr(group):
              above_avg = group.loc[df["gt_close_sma"] == 1]
              below_avg = group.loc[df["lt_close_sma"] == 1]
              steps1 = []
              steps2 = []

              def opt1(x):
                  m, b = x
                  s, _ = score(m, b, above_avg)
                  return s

              def opt2(x):
                  m, b = x
                  s, _ = score(m, b, below_avg)
                  return s

              def cb1(xk):
                  steps1.append(xk)

              def cb2(xk):
                  steps2.append(xk)

              res1 = optimize.minimize(opt1, x0=[1, 1], callback=cb1, method="BFGS")
              res2 = optimize.minimize(opt2, x0=[1, 1], callback=cb2, method="BFGS")

              return (below_avg, res2.x, steps2), (above_avg, res1.x, steps1)
```

Given a group DataFrame, the function $find_sr$ will find the optimum m, b values for a support line and a resistance line. $find_sr$ returns two three-tuples of the form $(avg\ DataFrame,\ (m, b),\ series\ of\ (m,\ b)\ computed\ during\ optimisation)$ where the first tuple is the support line and the second is the resistance line. The $minimize$ function from the Scipy $optimize$ package[5] is used with the $BFGS$ method. All lines start at $y = 1x + 1$.

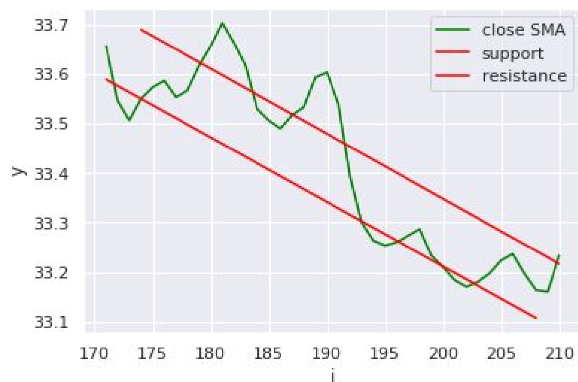
[5] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

```
In [620]: for i, (ts, group) in enumerate(df_grouped):
              s, r = find_sr(group)
              _, line1 = score(s[1][0], s[1][1], s[0])
              _, line2 = score(r[1][0], r[1][1], r[0])
              sns.lineplot(x="i", y="close_sma", data=group, color="green")
              sns.lineplot(x="i", y="y", data=line1, color="red")
              sns.lineplot(x="i", y="y", data=line2, color="red")
              plt.legend(labels=["close SMA", "support", "resistance"])
              plt.figure(i)
```

1

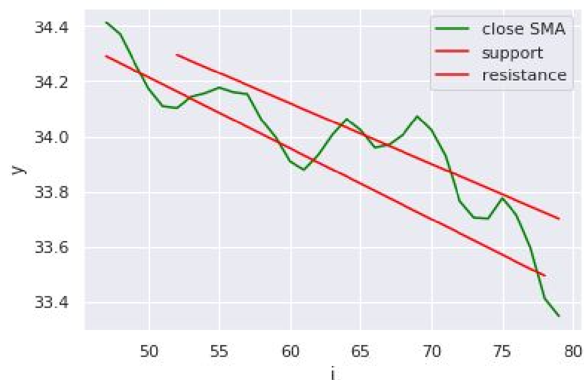


2

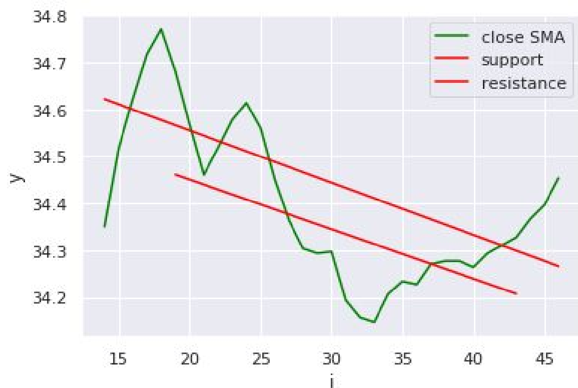


3

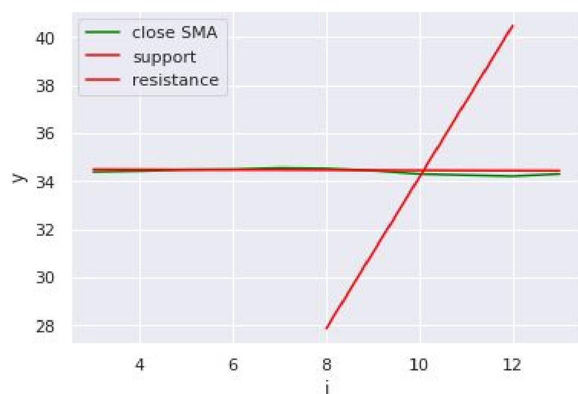
4



5



6



For each of the six four-hour groups, support and resistance lines are calculated and plotted against the SMA line. When the SMA has clear oscillations, the S/R lines wrap the SMA well and closely resemble the lines plotted manually. However, when there is no clear price oscillation - i.e. during a major upwards / downwards movement or when data is missing - no useful lines are found, which is to be expected.

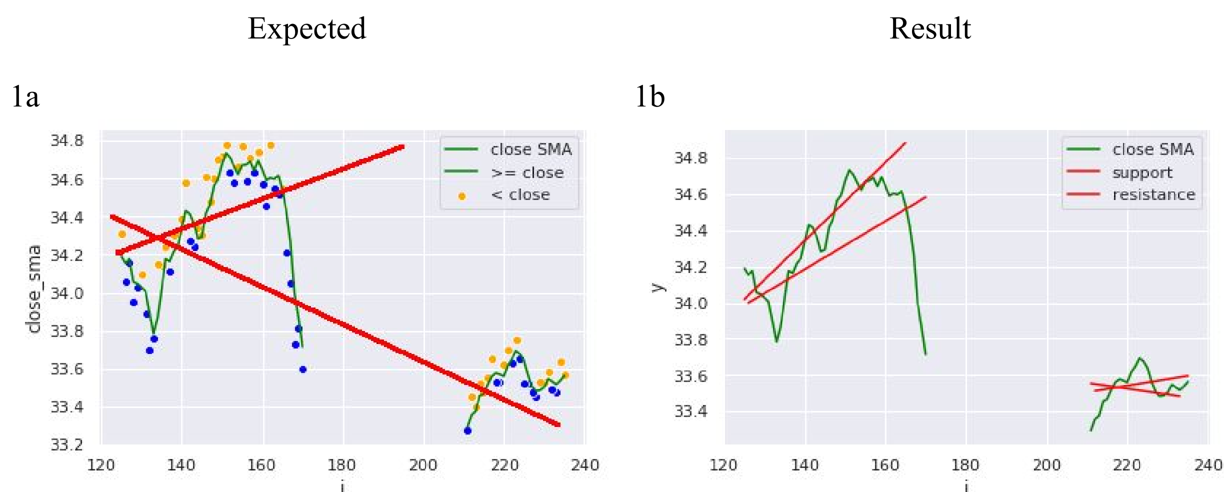
Computing the S/R lines is relatively computationally expensive - taking between three to four seconds to calculate on a Xeon E3 1240v3 at 3.4GHz. Instantaneous plotting in a real-time environment would not be possible.

TESTING AND RESULTS

An expected/result side-by-side comparison is shown below. Additionally, a side-by-side comparison between each result and the closing SMA's distance from the support level and resistance level is shown. When the price is 'bouncing' predictably between the S/R levels, the two distances appear to be *opposite* - the support distance's peaks overlap with the resistance distance's troughs.

Expected Result

1a 1b

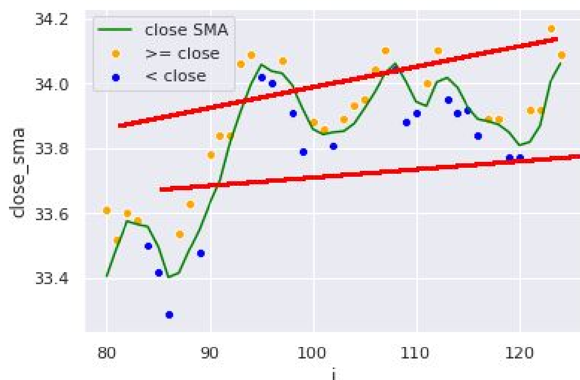


Data is missing and there is no clear up-down price movement in this data group. The benchmark lines are meaningless and were a guess at what the algorithm *would* have found. Surprisingly, the lines produced by the algorithm were somewhat successful - especially for the first chunk of data.

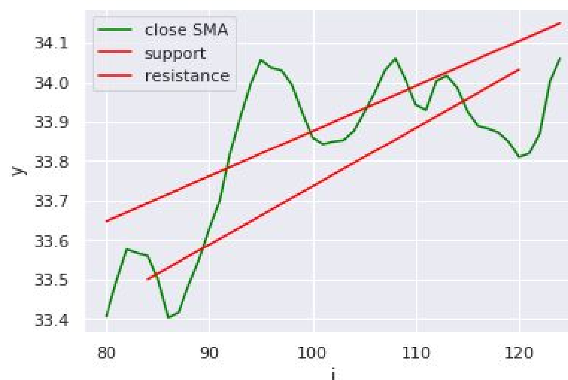
2a 2b

There is an obvious downwards trend in this group - with clear up-down price oscillations. When compared with the result lines, the benchmark support line appears to be slightly too low. The lines found by the algorithm for this group are accurate and better than the benchmark lines.

3a

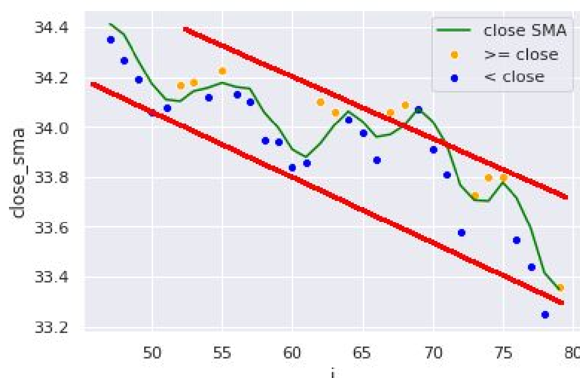


3b

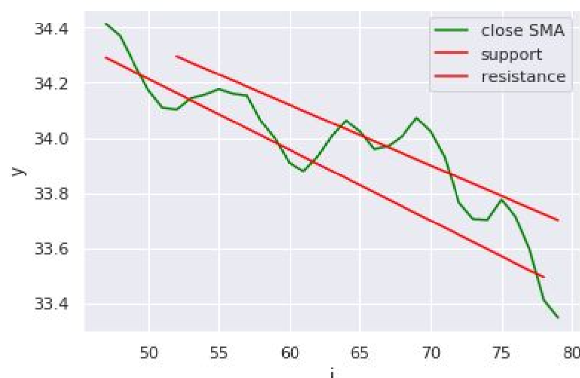


The first quarter of this group shows a sharp upwards movement, and the remaining three quarters show a sidewards trend with clear oscillations. The lines produced by the algorithm are heavily affected by the first quarter of data. Although the algorithm's are accurate, the benchmark lines are more useful in showing support and resistance price levels.

4a



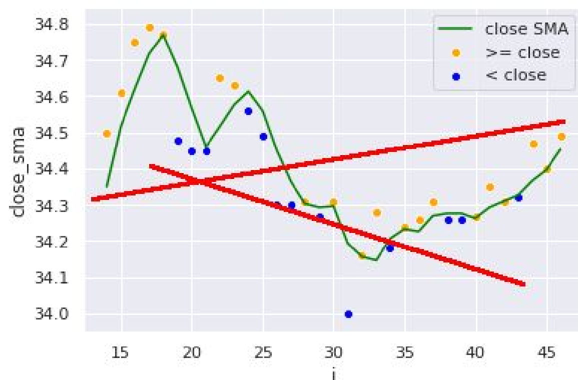
4b

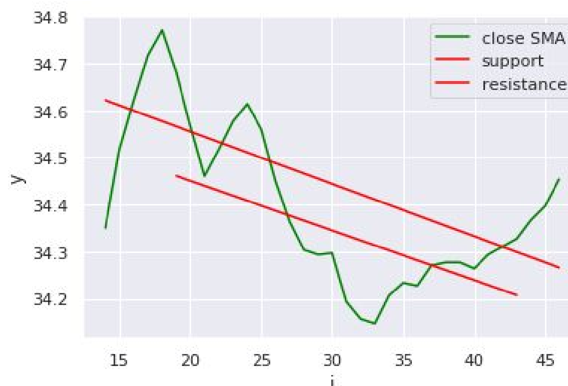


There is a clear downwards trend in this group with price oscillations. The benchmark support line for this group is too low. The lines found by the algorithm are accurate and wrap the SMA line nicely - and are better than the benchmark lines.

5a

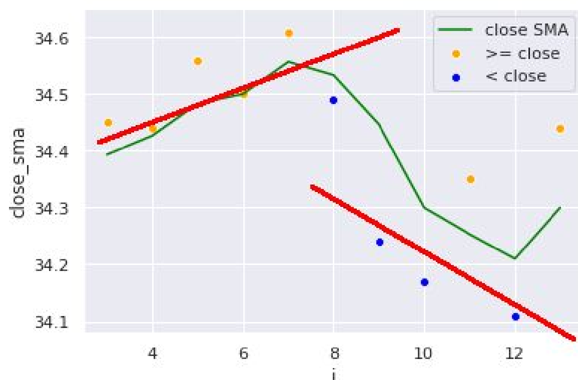
5b



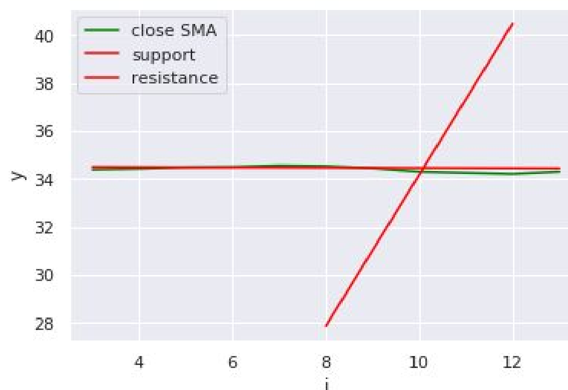


The first half of this group shows a sharp down trend, while the second half shows a slow uptrend. No meaningful support / resistance lines can be drawn for this data. The benchmark lines were a guess at what the algorithm would produce. However, the lines found by the algorithm make more sense and more accurately show the support and resistance levels in the data.

6a



6b

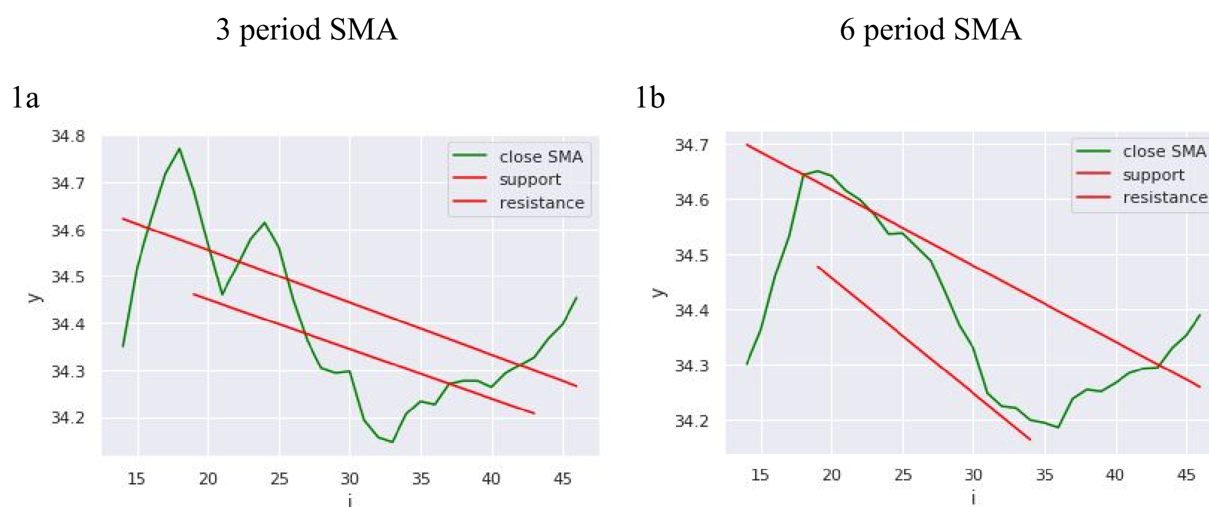


There is no clear trend or price oscillation in this group, and the data is small and quite sparse. The benchmark lines are meaningless and were a guess at what the algorithm would generate. The lines produced by the algorithm are also useless - which is to be expected with this data.

Overall, the support and resistance levels produced by the algorithm are accurate when there is a clear upwards / downwards / sideways trend with obvious price oscillations. Charts 2 and 4 show this well. However, lack of a trend, or when there is a sharp price movement, causes the algorithm to produce less accurate or completely useless lines. This is acceptable since it is

not always possible to plot meaningful support and resistance lines. Charts 5 and 6 illustrate this well.

Improving / adjusting the lines produced by the algorithm could be achieved by increasing or decreasing the SMA window. A larger window will result in a smoother SMA line - which would smooth-out steep price movements, resulting in more useful S/R lines when markets are particularly volatile. An example is shown below.

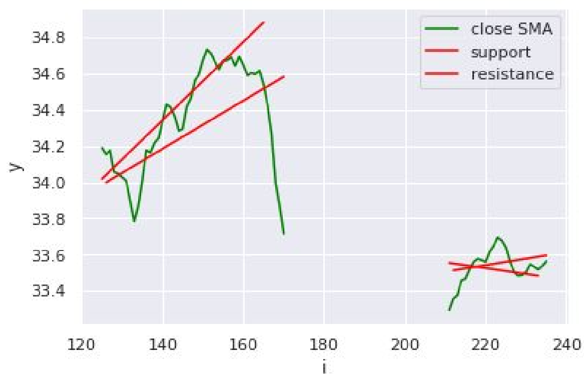


Clearly the 6 period SMA line smooths out the spikes visible in the 3 period line - resulting in more useful support and resistance lines.

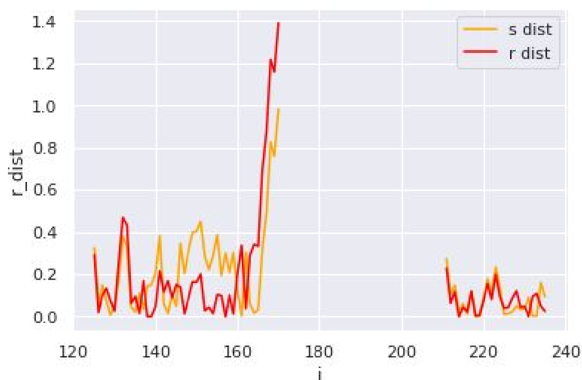
Close SMA with S/R levels

SMA distance from S/R levels

1a

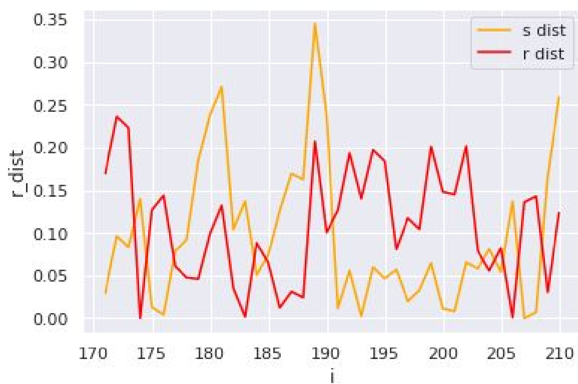


1b

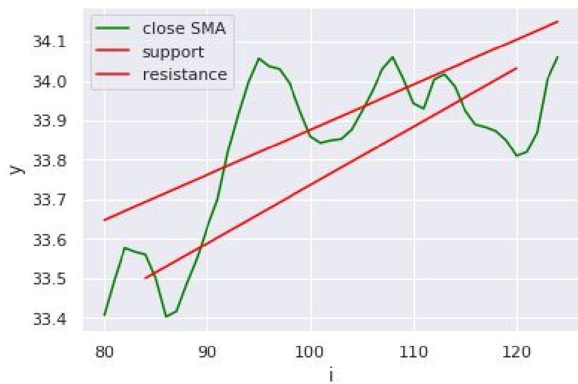


2a

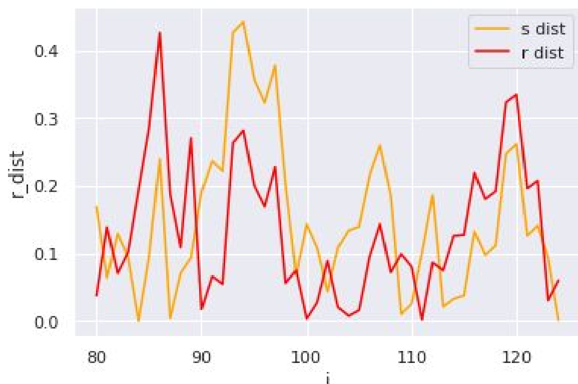
2b



3a

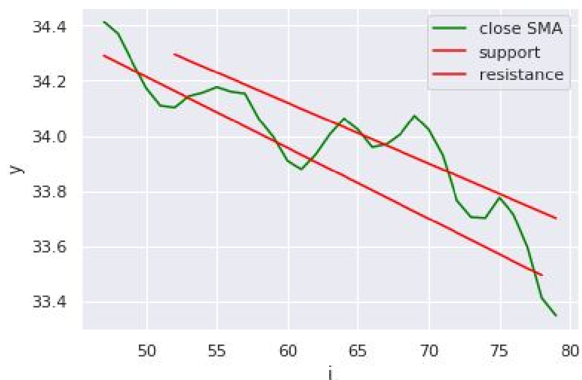


3b

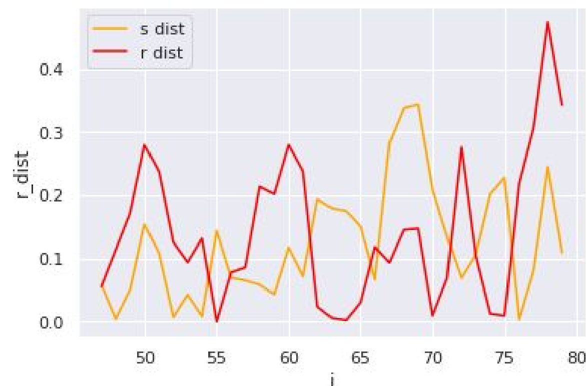


4a

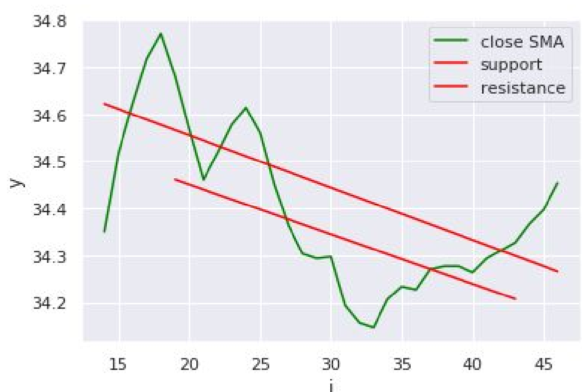
4b



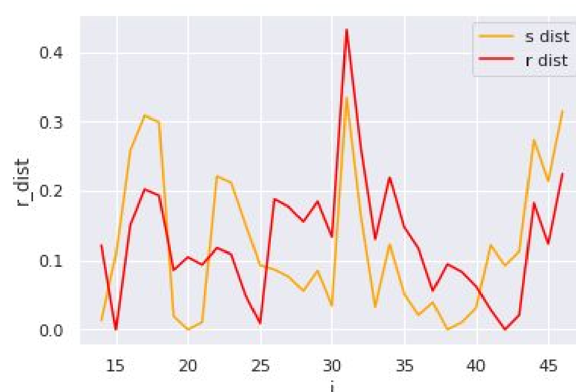
5a



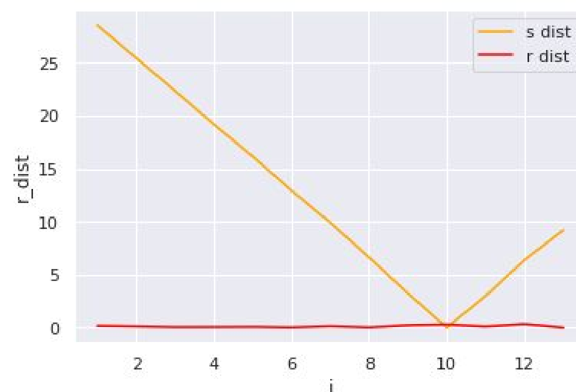
5b



6a



6b



As mentioned above, when the market is oscillating nicely between the S/R levels, the SMA distances, when plotted, appear to be opposite / mirror each other - which is to be expected - and provides an interesting insight into market activity.

CONCLUSION

I generated support and resistance on candlestick data using the following method:

1. Calculate simple moving average over close price - with a window of size 3.
2. Split the dataset into smaller, equally sized groups.
3. For each group, separate candles into two groups: those that are above or equal to the SMA, and those that are below the SMA.
4. For points above the SMA, calculate the resistance line: for each point p along the x axis, find a y value that is as near to the point as possible - such that the sum of the distances is minimal.
5. For points below the SMA calculate the support line: for each point p along the x axis, find a y value that is as near to the point as possible - such that the sum of the distances is minimal.

To conclude, overall, the algorithm works well and is successful in generating suitable support and resistance lines. The algorithm struggles when there is missing data or when there is no clear trend in the market - which is to be expected. It is possible to adjust / tweak the algorithm by changing the following parameters.

Tunable parameters

- SMA window size - a larger window will produce a smoother SMA line, which will remove sharp price changes.
- Candlestick timeframe - shorter candles are more sensitive to sharp changes in price. Like with the SMA window size, a longer candlestick duration will smooth-out market activity.

- Group size - the *group* is the series / subset of candles to calculate the S/R lines over. A larger group may contain multiple trends, while a smaller group may contain no obvious trends. The group size should be a specific timeframe - and thus is closely linked to the candlestick timeframe parameter.
- Distance function - ultimately, the optimisation algorithm is trying to minimise the sum of a series of distance values - 'distance' here refers to the euclidean distance between two points, but any distance function $d(P_1, P_2) \in \mathbb{R}$ can be used.

Limitations to my approach and things to improve

There is much room for improvement and further experimentation. The above algorithm and findings in this paper are limited in the following ways.

- Small dataset - I have only experimented with 5 minute candlesticks spanning a total of 24 hours. The above algorithm may perform better / worse on different timeframes.
- One market on one exchange analysed - The data I analysed was for the cryptocurrency/fiat pair LTC-GBP on the Coinbase exchange - and is not representative of the entire crypto market.
- Distance function - I have only experimented with one distance function. The function could be adjusted or changed completely to measure distance / similarity more effectively and/or efficiently.
- Candlestick groups - I simply split my 24 hour dataset into six four-hour groups,

ignoring start and end times. I have not experimented with different group sizes or accounted for increased trading activity at different times of the day.

Where next?

The findings in this paper will direct and help inform my future

1. Collect and experiment with more data - Specifically, I will experiment with other large crypto markets: Bitcoin and Ethereum.
2. Create and test a probability model - I intend to use the above algorithm, along with volume data and various technical indicators to train a neural network to forecast probable / least probable future price movements.
3. Deploy an automated trading system into a live environment - Ultimately, the probability model described above, along with the evolutionary process discussed in the introduction, will serve as the decision making engine for my own automated trading system.

Resources and tools used

1. Pandas - https://pandas.pydata.org/
2. Scipy - https://www.scipy.org/
3. Coinbase (Pro) - https://pro.coinbase.com/
4. Seaborn and Matplotlib - https://seaborn.pydata.org/, https://matplotlib.org/
5. Python - https://www.python.org/
6. Jupyter Notebook - https://jupyter.org/

Works cited

Wellington Garikai, Bonga. (2015). The Need for Efficient Investment: Fundamental Analysis and Technical Analysis. Finance & development. 10.2139/ssrn.2593315.

Kim YB, Lee SH, Kang SJ, Choi MJ, Lee J, Kim CH (2015) Virtual World Currency Value Fluctuation Prediction System Based on User Sentiment Analysis. PLoS ONE 10(8): e0132944.doi:10.1371/journal.pone.0132944

Freire, Rosane & Fernandes, Cristiano & Lorenzoni, Giuliano & Pizzinga, Adrian & Atherino, Rodrigo. (2007). On the Statistical Validation of Technical Analysis. Revista Brasileira de Finanças. 5.

Bhattacharya, Sukanto & Kumar, Kuldeep. (2006). A computational exploration of the efficacy of Fibonacci Sequences in Technical analysis and trading. Business papers. 7.

Praekhaow, Puchong. (2010). Determination of Trading Points using the Moving Average Methods.

[notebook] Support resistance line detection using mathematical optimisation

March 30, 2020

[830]:
```
import math
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from requests import get
from datetime import datetime, timedelta
from time import sleep
from scipy import optimize
```

[ ]:
```

```

[ ]:
```

```

0.0.1 fetch data from coinbase pro api

The cryptocurrency exchange Coinbase, and their trading platform Coinbase Pro, has a nice publically available data API - which allows fetching of candlestick data with different granularity and start-end timestamps. Documentation can be found at https://docs.pro.coinbase.com/

[831]:
```
def fetch_candles():
    now = datetime.now()
    params = {
        "granularity": "300",
        "end": now.isoformat(),
        "start": (now - timedelta(hours=24)).isoformat(),
    }
    r = get("https://api.pro.coinbase.com/products/LTC-GBP/candles",␣
 ↪params=params)
    return r.json()
```

[ ]:
```

```

[832]:
```
candles = fetch_candles()
df = pd.DataFrame({
    "open": [c[1] for c in candles],
```

```
    "high": [c[2] for c in candles],
    "low": [c[3] for c in candles],
    "close": [c[4] for c in candles],
    "volume": [c[5] for c in candles],
    "timestamp": [c[0] for c in candles]
})
df.head()
```

[832]:

```
    open   high    low  close     volume   timestamp
0  31.32  31.43  31.32  31.43  13.240023  1585554300
1  31.27  31.38  31.27  31.29  11.153097  1585554000
2  31.17  31.17  31.17  31.17   1.000000  1585553700
3  31.20  31.29  31.25  31.20   2.446124  1585553400
4  31.32  31.38  31.32  31.38   2.666017  1585553100
```

[ ]:

[833]:

```
df["datetime"] = pd.to_datetime(df["timestamp"], unit="s")
df["i"] = df.apply(lambda x: x.name + 1, axis="columns")
df["close_sma"] = df["close"].rolling(window=3).mean()
df["volume_pct_chg"] = df["volume"].pct_change()
df["close_pct_chg"] = df["close"].pct_change()
df["gt_close_sma"] = df.apply(
    lambda x: 1 if x["close"] >= x["close_sma"] else 0, axis="columns"
)
df["lt_close_sma"] = df.apply(
    lambda x: 1 if x["close"] < x["close_sma"] else 0, axis="columns"
)
df.head()
```

[833]:

```
    open   high    low  close     volume   timestamp            datetime  i  \
0  31.32  31.43  31.32  31.43  13.240023  1585554300 2020-03-30 07:45:00  1
1  31.27  31.38  31.27  31.29  11.153097  1585554000 2020-03-30 07:40:00  2
2  31.17  31.17  31.17  31.17   1.000000  1585553700 2020-03-30 07:35:00  3
3  31.20  31.29  31.25  31.20   2.446124  1585553400 2020-03-30 07:30:00  4
4  31.32  31.38  31.32  31.38   2.666017  1585553100 2020-03-30 07:25:00  5

   close_sma  volume_pct_chg  close_pct_chg  gt_close_sma  lt_close_sma
0        NaN             NaN            NaN             0             0
1        NaN       -0.157623      -0.004454             0             0
2  31.296667       -0.910339      -0.003835             0             1
3  31.220000        1.446124       0.000962             0             1
4  31.250000        0.089894       0.005769             1             0
```

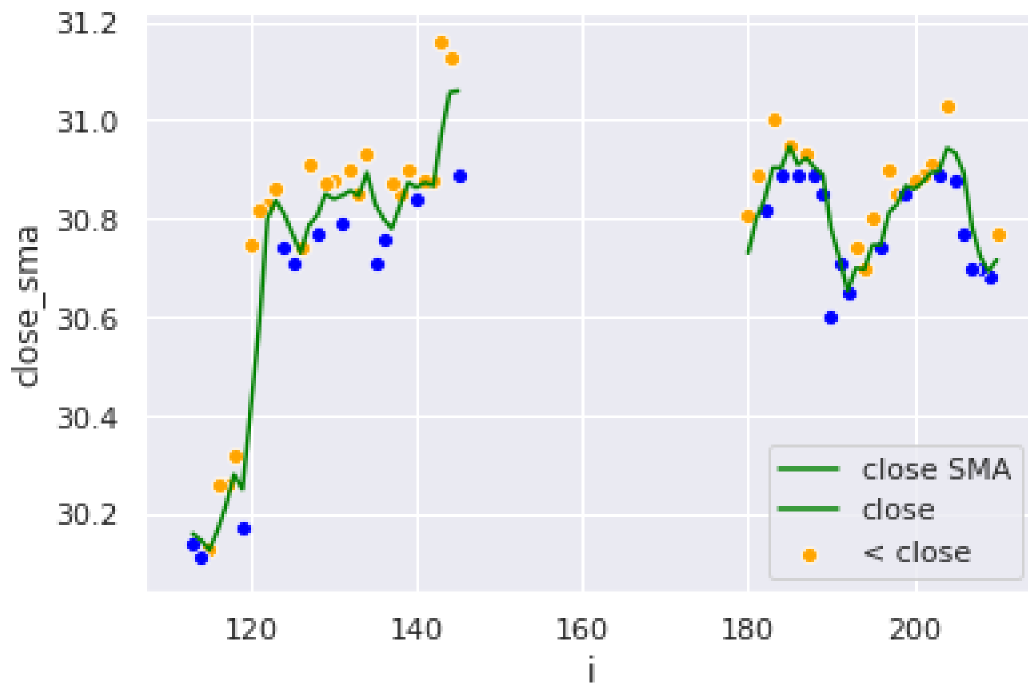
[ ]:

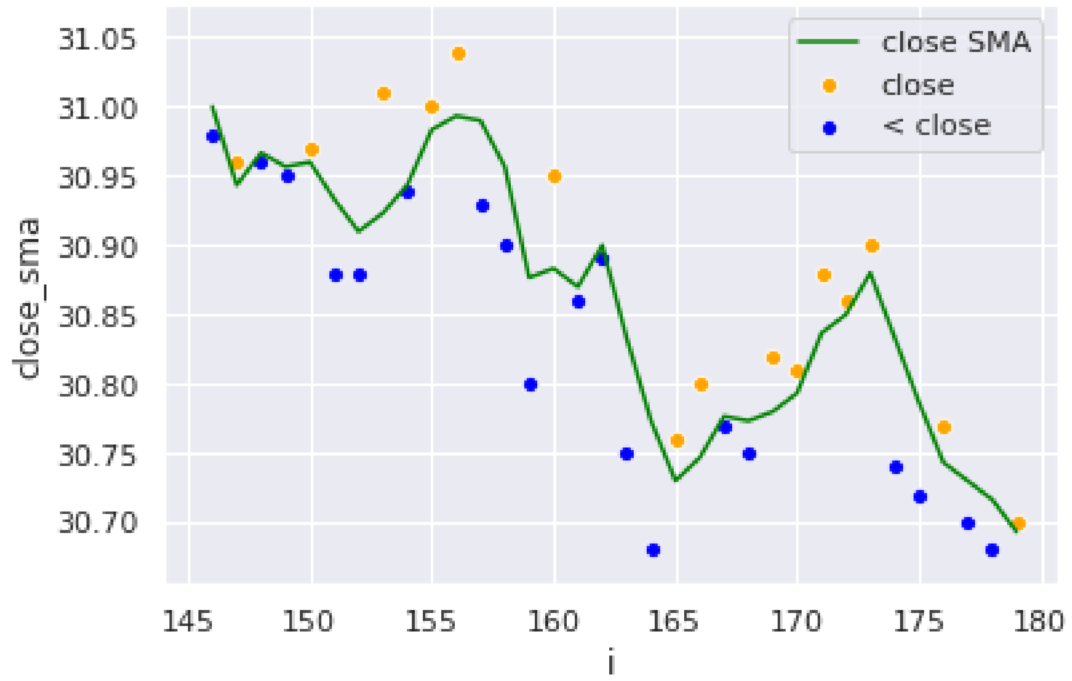
[834]:

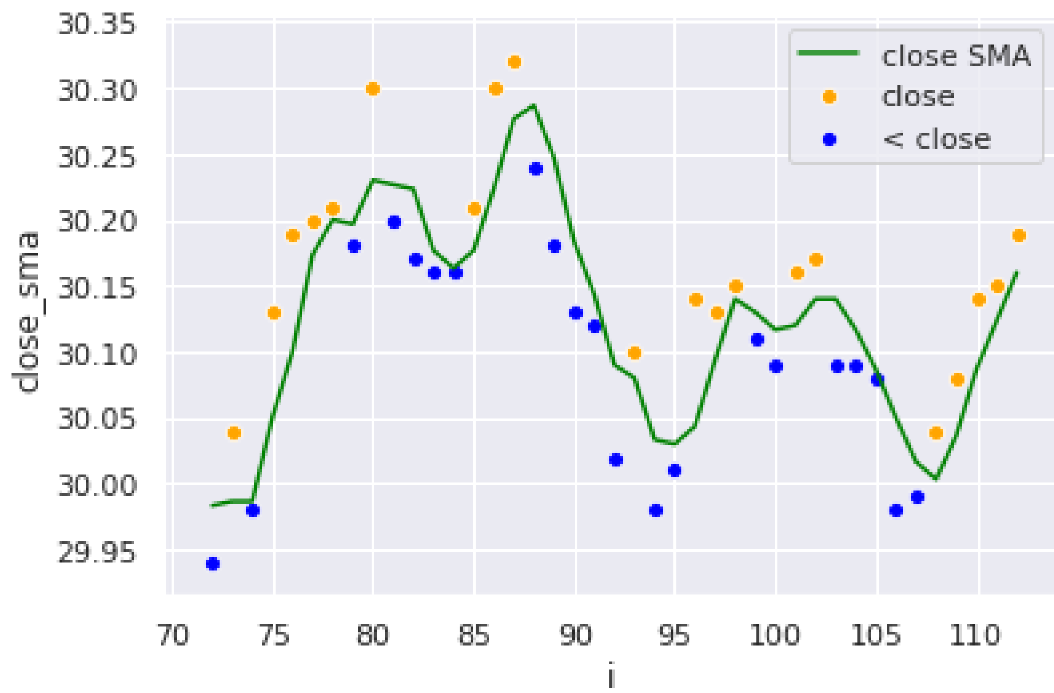
```
df_grouped = df.groupby(pd.Grouper(key="datetime", freq="4h"))
```

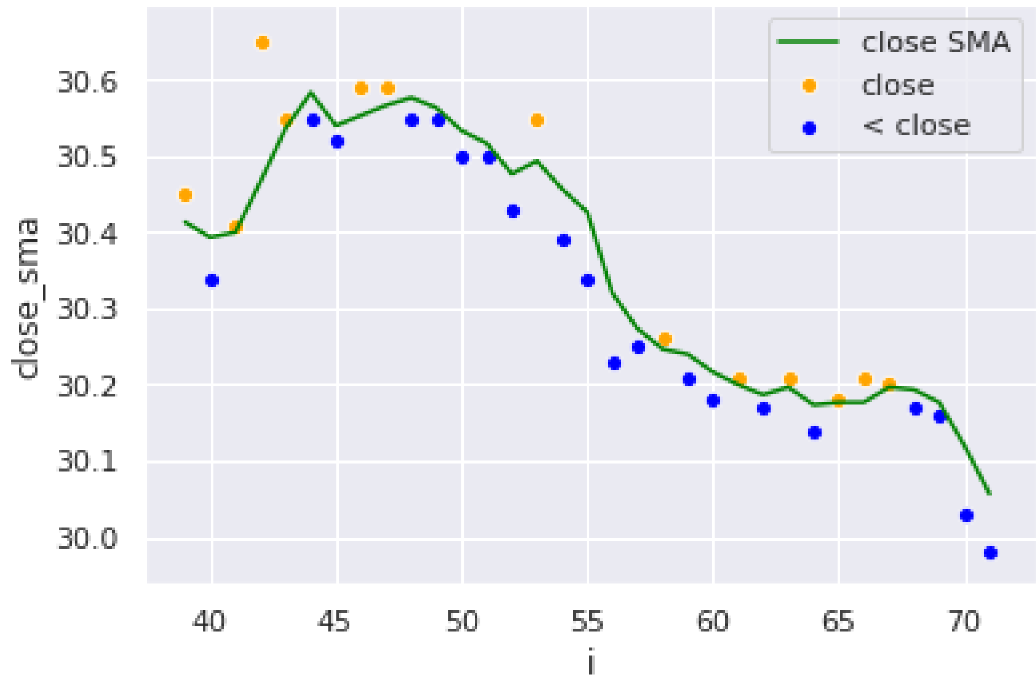
[ ]:

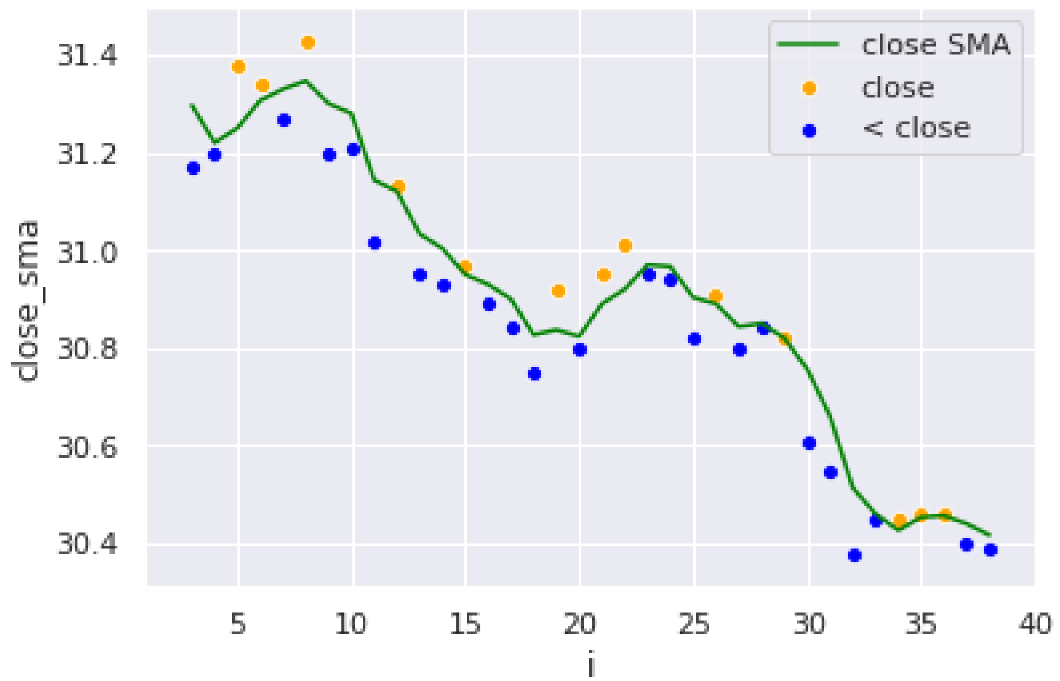
[835]:
```
for i, (_, group) in enumerate(df_grouped):
    above_avg = group.loc[df["gt_close_sma"] == 1]
    below_avg = group.loc[df["lt_close_sma"] == 1]
    sns.scatterplot(x="i", y="close", data=above_avg, color="orange")
    sns.scatterplot(x="i", y="close", data=below_avg, color="blue")
    sns.lineplot(x="i", y="close_sma", data=group, color="green")
    plt.legend(labels=["close SMA", "close", "< close"])
    plt.figure(i)
```




close SMA
close
< close
close_sma
i
31.05
31.00
30.95
30.90
30.85
30.80
30.75
30.70
145
150
155
160
165
170
175
180


close SMA
close
< close
close_sma
i
30.35
30.30
30.25
30.20
30.15
30.10
30.05
30.00
29.95
70
75
80
85
90
95
100
105
110





<Figure size 432x288 with 0 Axes>

[ ]:

```

```

[836]:

```
# linear function
def f(x, m ,b):
    return m * x + b

# simple euclidean distance
def d(p1, p2):
    return math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)
```

[ ]:

```

```

[837]:

```
def score(m, b, df):
    tot = 0
    line = []
    for _, row in df.iterrows():
        y = f(row["i"], m, b)
        p1 = (row["i"], row["close"])
        p2 = (row["i"], y)
        tot += d(p1, p2)
        line.append(y)
    return tot, pd.DataFrame({ "i": df["i"], "y": line })
```

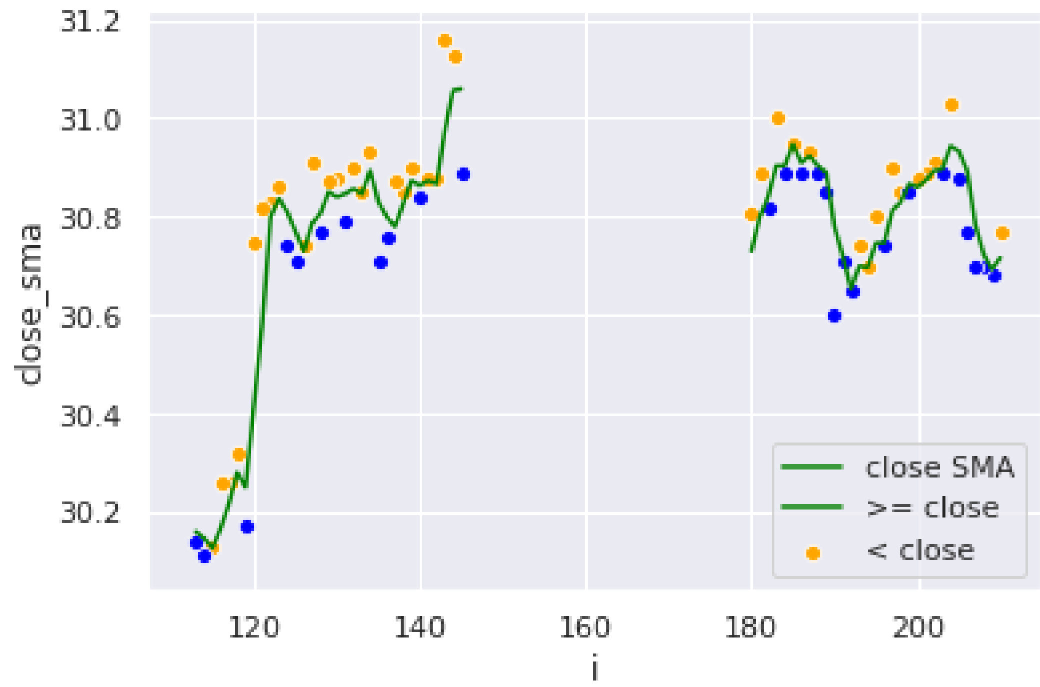
[ ]:

```

```

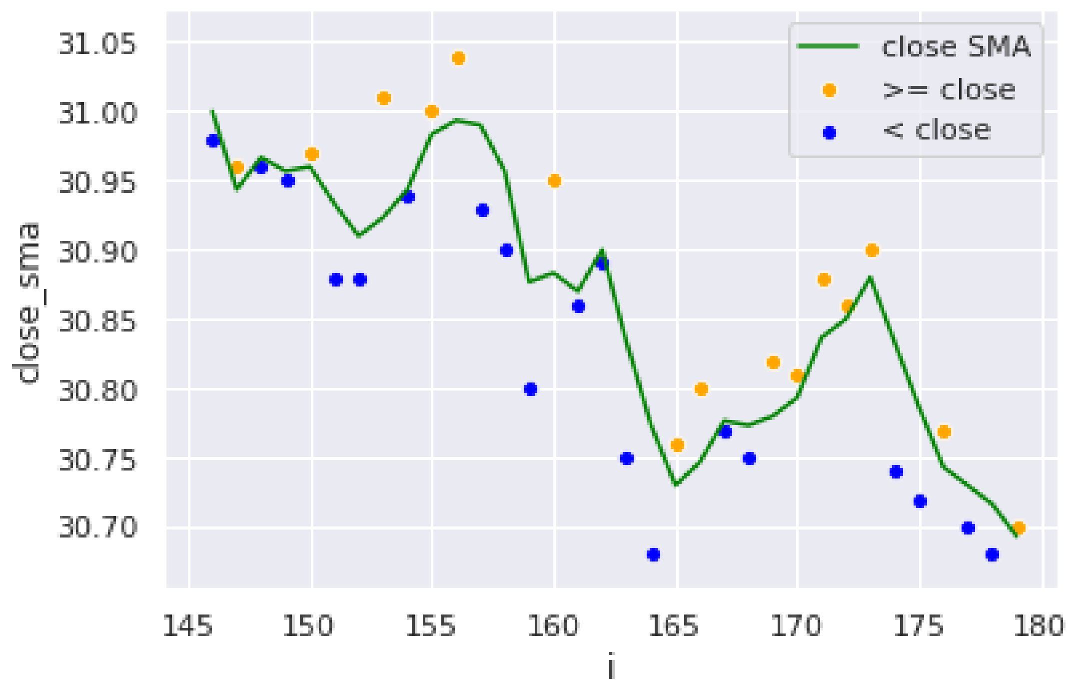
[838]:

```
for i, (ts, group) in enumerate(df_grouped):
    above_avg = group.loc[df["gt_close_sma"] == 1]
    below_avg = group.loc[df["lt_close_sma"] == 1]
    sns.scatterplot(x="i", y="close", data=above_avg, color="orange")
    sns.scatterplot(x="i", y="close", data=below_avg, color="blue")
    sns.lineplot(x="i", y="close_sma", data=group, color="green")
    plt.legend(labels=["close SMA", ">= close", "< close"])
    plt.figure(i)
    print(ts)
```

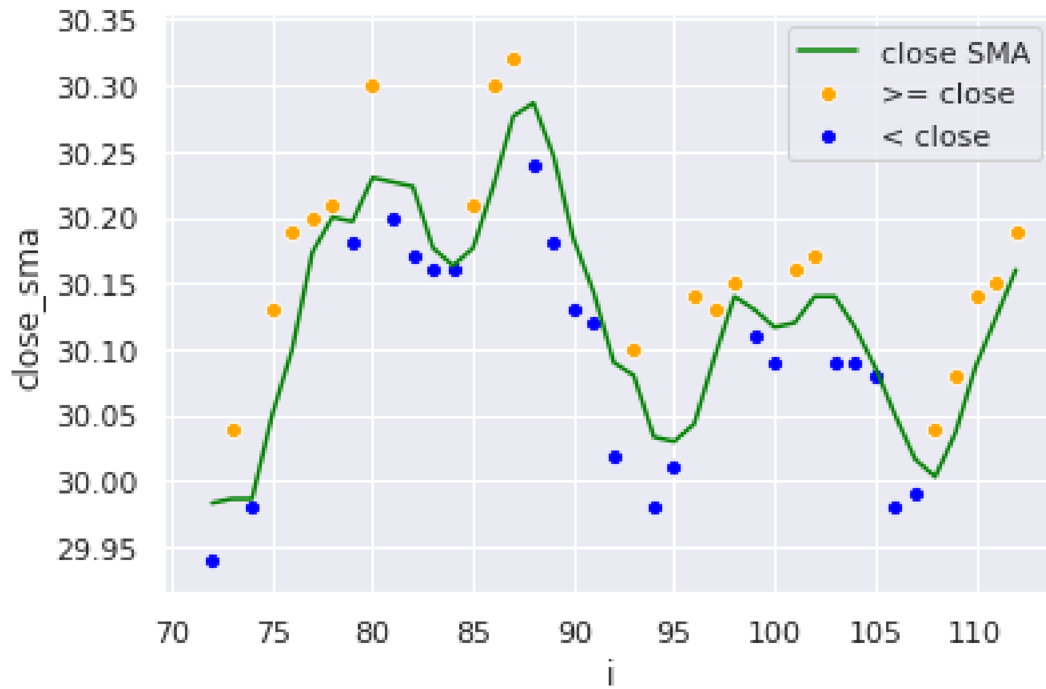
```
2020-03-29 08:00:00
2020-03-29 12:00:00
2020-03-29 16:00:00
2020-03-29 20:00:00
2020-03-30 00:00:00
2020-03-30 04:00:00
```

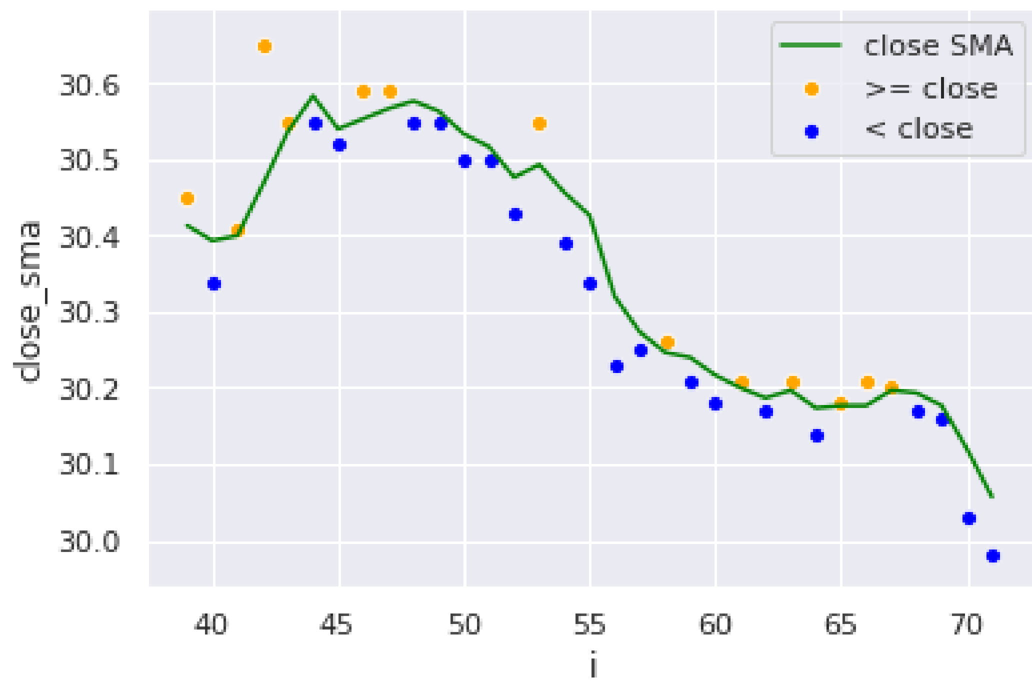


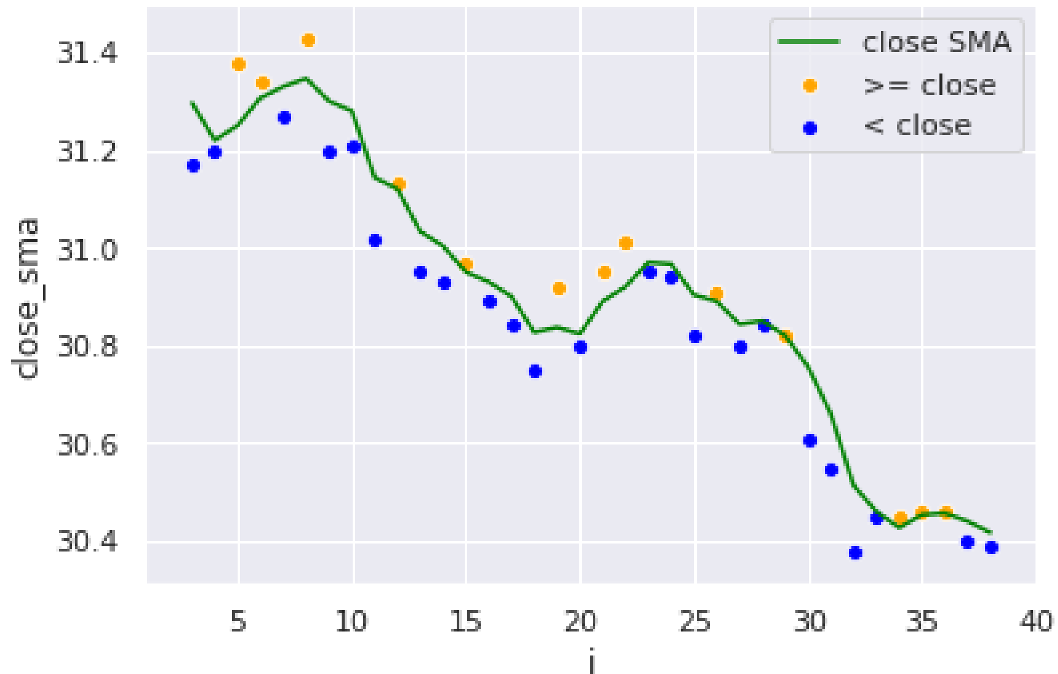
31.2
31.0
30.8
30.6
30.4
30.2
close_sma
120
140
160
180
200
i
close SMA
>= close
< close



31.05
31.00
30.95
30.90
30.85
30.80
30.75
30.70
close_sma
145
150
155
160
165
170
175
180
i
close SMA
>= close
< close


close SMA
>= close
< close
close_sma
i
30.35
30.30
30.25
30.20
30.15
30.10
30.05
30.00
29.95
70
75
80
85
90
95
100
105
110


close SMA
>= close
< close
close_sma
i
30.6
30.5
30.4
30.3
30.2
30.1
30.0
40
45
50
55
60
65
70




```
<Figure size 432x288 with 0 Axes>
```

[ ]:

```

```

[839]:

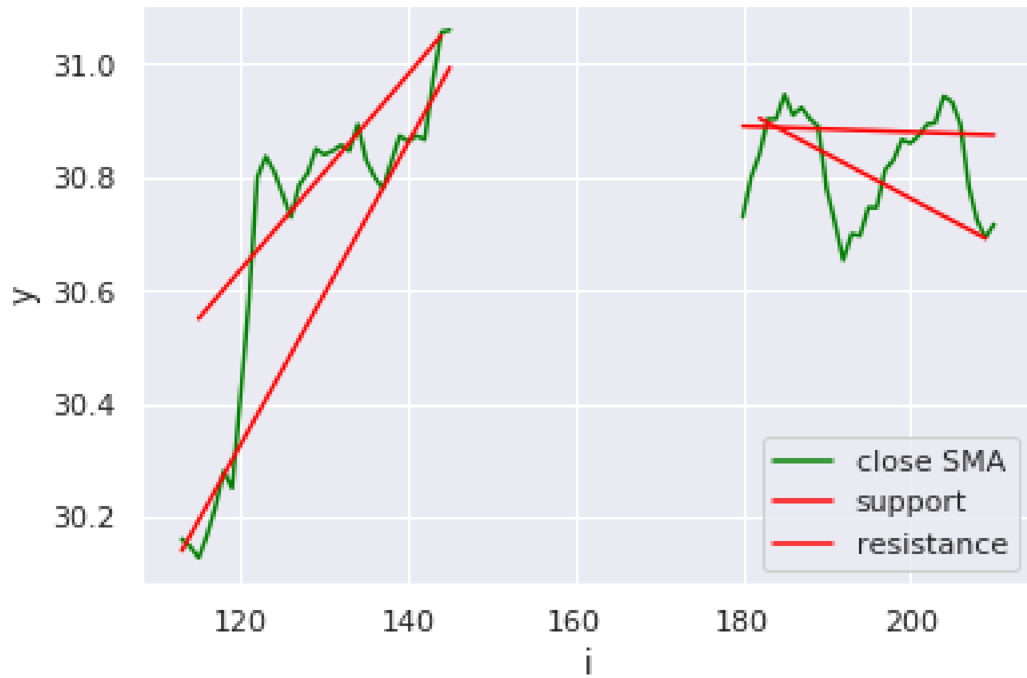
```
def find_sr(group):
    above_avg = group.loc[df["gt_close_sma"] == 1]
    below_avg = group.loc[df["lt_close_sma"] == 1]
    steps1 = []
    steps2 = []

    def opt1(x):
        m, b = x
        s, _ = score(m, b, above_avg)
        return s

    def opt2(x):
        m, b = x
        s, _ = score(m, b, below_avg)
        return s

    def cb1(xk):
        steps1.append(xk)

    def cb2(xk):
```

```
        steps2.append(xk)

    res1 = optimize.minimize(opt1, x0=[1, 1], callback=cb1, method="BFGS")
    res2 = optimize.minimize(opt2, x0=[1, 1], callback=cb2, method="BFGS")

    return (below_avg, res2.x, steps2), (above_avg, res1.x, steps1)
```

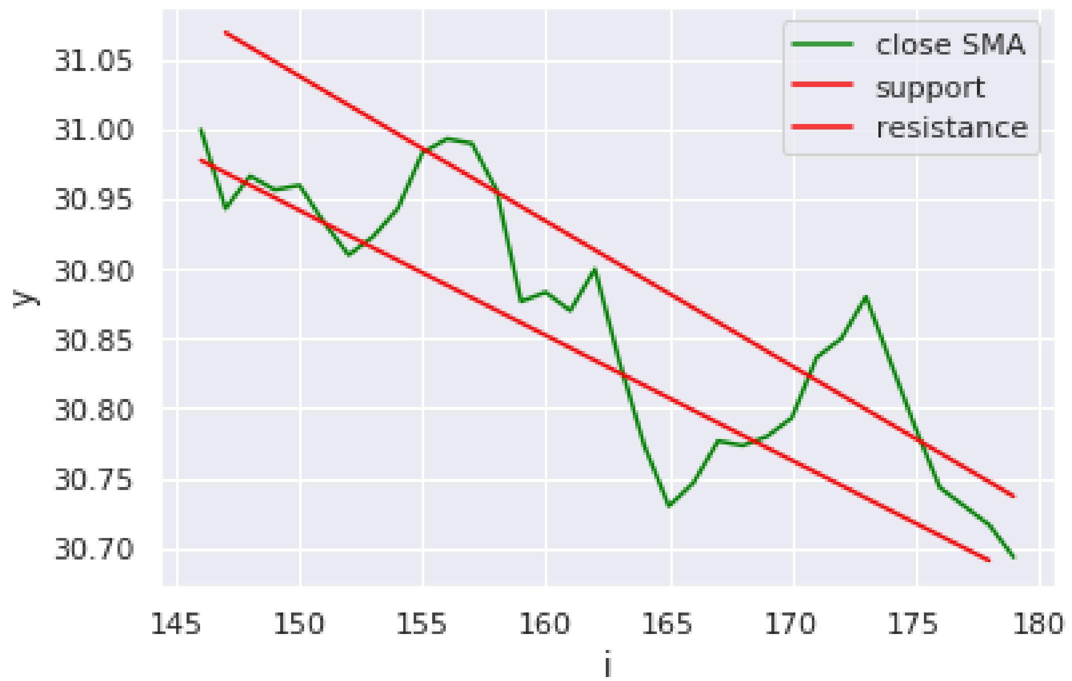
[ ]:

```
[840]: for i, (ts, group) in enumerate(df_grouped):
    s, r = find_sr(group)

    for j, row in group.iterrows():
        df.loc[j, "s_dist"] = d(
            (row["i"], f(row["i"], s[1][0], s[1][1])),
            (row["i"], row["close"])
        )
        df.loc[j, "r_dist"] = d(
            (row["i"], f(row["i"], r[1][0], r[1][1])),
            (row["i"], row["close"])
        )

    _, line1 = score(s[1][0], s[1][1], s[0])
    _, line2 = score(r[1][0], r[1][1], r[0])
    sns.lineplot(x="i", y="close_sma", data=group, color="green")
    sns.lineplot(x="i", y="y", data=line1, color="red")
    sns.lineplot(x="i", y="y", data=line2, color="red")
    plt.legend(labels=["close SMA", "support", "resistance"])
    plt.figure(i)
    print(ts)
```

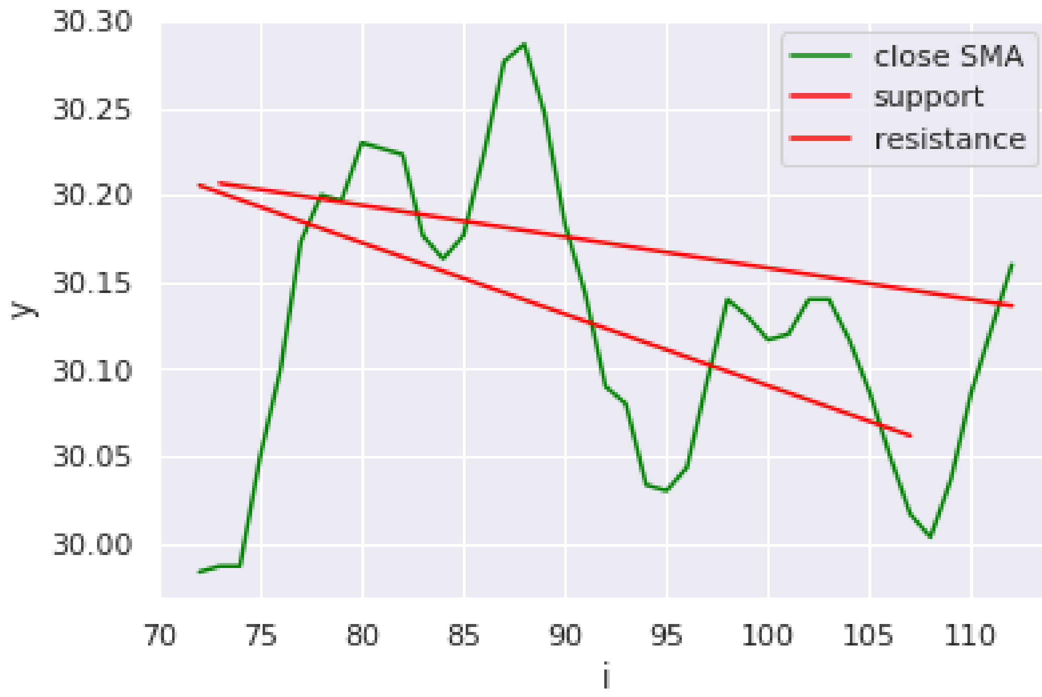
```
2020-03-29 08:00:00
2020-03-29 12:00:00
2020-03-29 16:00:00
2020-03-29 20:00:00
2020-03-30 00:00:00
2020-03-30 04:00:00
```



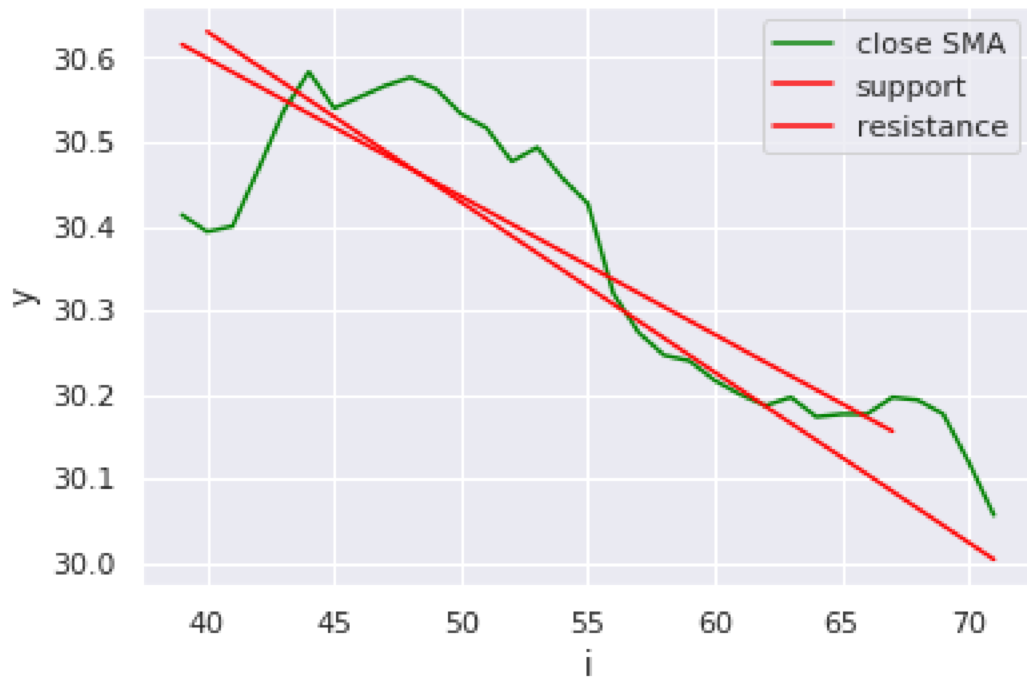
31.0
30.8
30.6
30.4
30.2
y
120
140
160
180
200
i
close SMA
support
resistance



31.05
31.00
30.95
30.90
30.85
30.80
30.75
30.70
y
145
150
155
160
165
170
175
180
i
close SMA
support
resistance



close SMA
support
resistance
30.30
30.25
30.20
30.15
30.10
30.05
30.00
y
70
75
80
85
90
95
100
105
110
i



close SMA
support
resistance
30.6
30.5
30.4
30.3
30.2
30.1
30.0
y
40
45
50
55
60
65
70
i

```
<Figure size 432x288 with 0 Axes>
```

[ ]: