Trading Strategies and Time Series Forecasting

December 21, 2022

0.1 Table of contents

1 Introduction

The paper looks at information related to minute-over-minute data of the spot and futures markets for Bitcoin on the Binance exchange in the period March 1st 2021 to March 1st 2022, and presenting a simple strategy for engaging in trades and how one can test these strategies by using statistics. Based on the time series we will also look at a few different approaches to forecasting future price movements by using ARIMA class models.

1.1 Summary

The paper presents a simple trading strategy based on entering a trade after a *Hammer candle*, and an *Inverted hammer candle*, and exiting after 5 minutes. A hammer candle is a period where the open, close, and high price for the asset within the period are close to eachother, and the lowest price recorded is significantly below these. This pattern is thought of to be a bullish sign for the following periods, as the pattern indicates a significant downturn that turns to a strong recovery within the same period. Similarly an *Inverted hammer candle* is a period where the open, close and low price for the asset within a period are close to eachother, and the highest price recorded is significantly above these. This pattern is thought of to be a bearish sign for the following periods.

The general requirements for a candle to be considered a hammer or inverted hammer candle are somewhat vague, which makes it hard to compare with other data one can find online. In this paper, candles are considered hammer or inverted hammer candles if the following 3 conditions are met:

1. The volatility within the period (minute) is among the top 5% of the recorded volatilities
2. The opening and closing price for the period is within 20% of each other relative to the volatility in the period
3. The closing price is within the top or bottom 20% of the volatility within the period for it to be considered a hammer or inverted hammer candle respectively.

The trading strategy is tested by the means of simulation the difference in means in the futures market, as the sample data from the futures market shows a wider spread than the spot market.

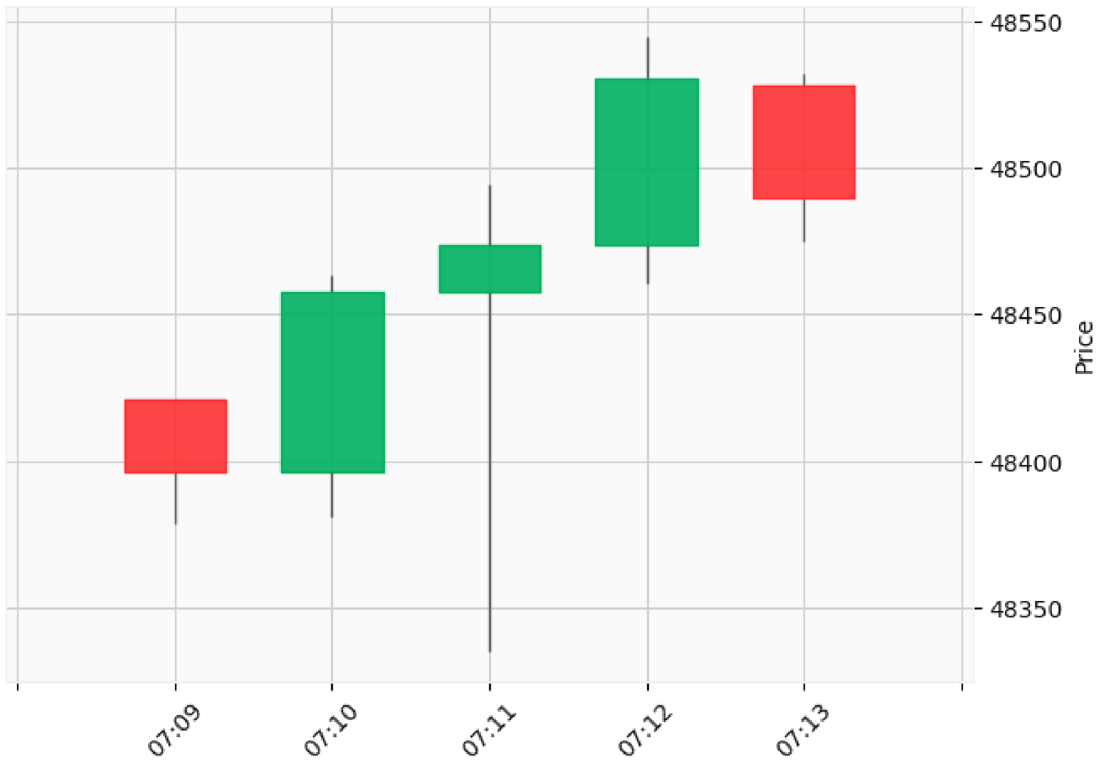
The real mean and standard deviation used for the simulation is based on a simple backtest of the strategy, which introduces some uncertainty in the results due to bias in the sample data and the way the hammer and inverted hammer candles are classified. Despite this, the trading strategy seemed to give reliable returns in the futures market based on the backtest, and the simulation showed a statistically significant difference in the means of the returns in the trailing periods after the initial candles.

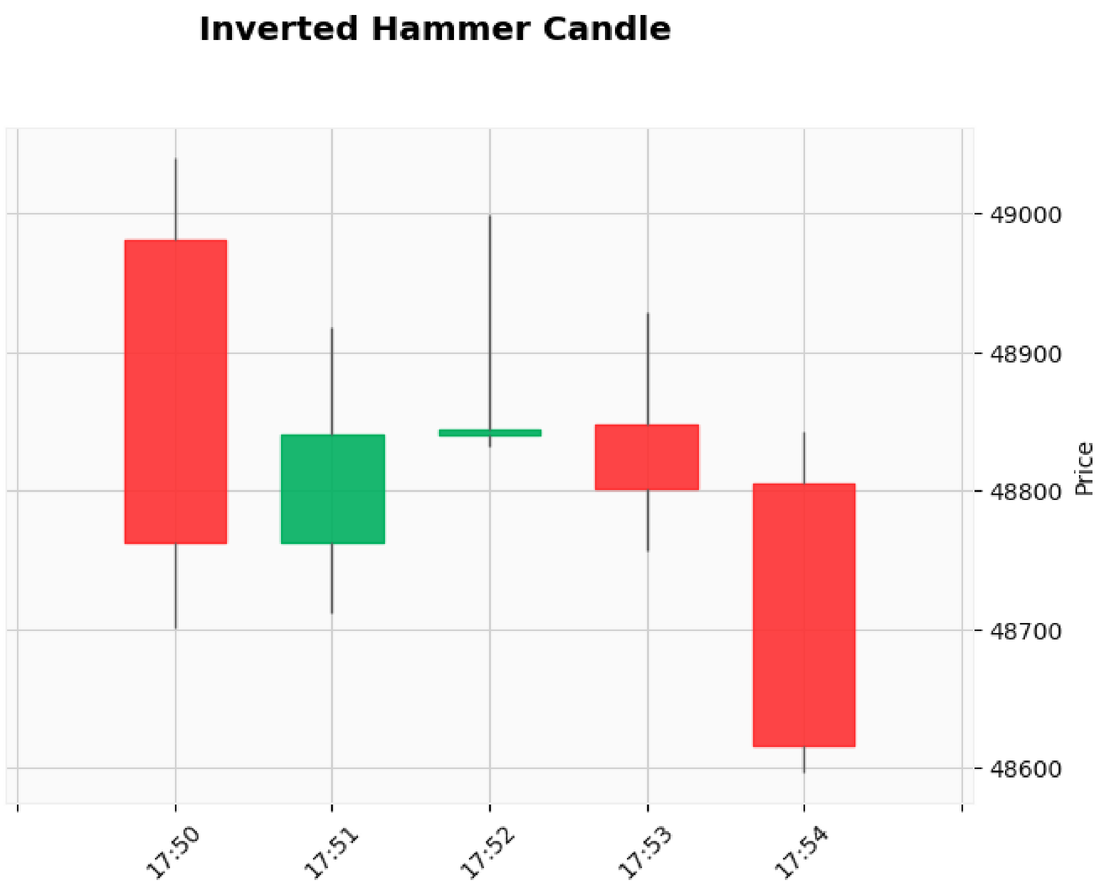
The price forecasts are based on the price data from the spot market. This is due to the slightly lower volatility in the spot market than in the futures market, which should make the forecast marginally easier to predict.

For a visual representation of what a Hammer candle and an inverted hammer candle looks like, the two plots below show just that. The candle of interest in both plots is the middle, or the 3rd one. The line on each candle represents the highest and lowest price recorded in the period, and the wider part represents the opening and closing price in the period - if the candle is green, the closing price is higher than the opening price and vice versa if the candle is red. The first plot shows the regular hammer candle, where the opening, high, and closing price are close, and the low is significantly lower, making it look like a hammer. The second plot shows an inverted hammer which is just that, the hammer upside down. The line showing the low and high price for each candle might be somewhat difficult to see as you can't change the color of it when using the matplotlib finance package, and other packages I tried was not supported by LaTeX and woulld not show up when the file was converted to pdf.

```
[ ]: import matplotlib.pyplot as plt
     import mplfinance as mpl
     import pandas as pd
     import matplotlib.dates as mpl_dates

     # An example of a hammer candle
     df = pd.read_csv("./datasets/hammer.csv")
     df["Date"] = pd.to_datetime(df["Date"])
     df = df.set_index("Date")
     mpl.plot(
         df, type="candle", title = "Hammer Candle", style="yahoo"
         )
     plt.show()

     # An example of an inverted hammer candle
     df = pd.read_csv("./datasets/inverted.csv")
     df["Date"] = pd.to_datetime(df["Date"])
     df = df.set_index("Date")
     mpl.plot(
         df, type="candle", title = "Inverted Hammer Candle", style="yahoo"
         )
     plt.show()
```

Hammer Candle





1.2 Why Bitcoin?

In contrast to the majority of financial assets that are traded with high volume, most cryptocurrencies don't pay dividends. Valuing these kinds of assets can therefore not be done by traditional means of discounting future cash flows, and the valuation of the asset should in theory be based less on fundementals and more on speculation. The data analyzed in the paper is a time series with 1 minute intervals between each entry, which further should reduce the price impact of fundementals and rather emphasize how the price changes in relation to it's returns in previous periods. This is the basis of the paper, which aims to test the profitability of a simple trading strategy, as well as forecasting future price action. Bitcoin is the cryptocurrency that usually trades the highest volume each day (CoinMarketCap, 2022), and it does not pay dividends to the asset holders. The former ensures that both retail and institutional traders can actively engage in trades with varying volume without significant liquidity risk, and the latter satisfies the previously mentioned missing fundementals of this asset class.

The data used in the paper is fetched from Binance's free API that provides market data for hundreds of cryptocurrency pairs with varying time intervals for both spot assets and futures contracts. Binance is the largest cryptocurrency exchange measured by daily volume traded in both the futures and spot market (CoinMarketCap, 2022), and the Bitcoin / USDT trading pair is the highest volume trading pair on the exchange.

1.3 Terminology used in the paper

Some words used in this paper are not regularly used outside of specific applications, and therefore it might be beneficial to explain what these words imply.

- **USDT**

USDT is a stablecoin pegged to the US Dollar. The techincal aspects of it is outside of the scope of this paper, however the basics is that 1 USDT = 1 USD. This means that the quote asset in the BTC/USDT trading pair is essentially the US Dollar. USD is the most used term throughout the paper, but both terms carry the same meaning, and there is not a difference in the value between the two (at the time of writing).

- **Candles**

Candles is a term used to describe individual time periods in a series of trade data for financial instruments. The name *Candle*, or *Candlestick* as it's also called refers to the pattern that the information related to a single period in the time series is visualized on a chart, showing the price at the start of the given period, often called the opening price, the highest price the instrument traded at in the period, the lowest price the instrument traded at in the period, and the price at the end of the period, also called the closing price. References to candles in this paper mainly points at the price at a given time period, or over a set of time periods, where each candle is equal to one minute. The price referenced will be the closing price of each candle, in other words the price at the end of the period.

1.4 Datasets used

- **Minute-over-minute price and volume data for spot trades**

The available data connected to the spot volume of the asset. Spot essentially means instant execution, where a buyer get's control over the asset itself once the trade is executed. Due to the instant settlement of the asset, leverage is limited in this market. This typically means less buying power for each market participant and somewhat tighter spreads on the returns.

- **Minute-over-minute perpetual futures**

Futures are contracts between two parties that enable the settlement of an asset in the future at a given price, and is a commonly used financial instrument for example by companies that deal with commodities like oil or metals to hedge their costs and profits. Perpetual futures are a class of futures contracts used almost exclusively for cryptocurrencies. As the name implies the futures contracts are perpetual, or in other words don't have an expiry date where the spot asset has to be settled between the parties. This makes the valuation of the perpetual contracts different from traditional futures contracts, as there are recurring payments between the parties in the perpetual futures market, where the payment amount is based on the difference in price between the spot asset and the futures contracts. These payments enable market participants to arbitrage the spot and futures market to keep the prices close, and the biggest difference between the markets end up being the amount of leverage available. As mentioned the spot market has limited leverage, but the futures market on Binance offers up to 125x leverage at the time of writing, essentially increasing the size of each trade and somewhat larger spreads on returns compared to the spot asset.

The majority of the functions in this paper use the NatSpec* comment format commonly used in languages such as Vyper and Solidity to provide rich documentation in the codebase.

NatSpec tags used throughout this document:

- @notice - High level function explanation
- @dev - Any extra details for developers
- @param - Documents a single parameter
- @return - Documents the return value(s)

*Ethereum Natural Language Specification Format

More info about the NatSpec format can be found here: https://docs.soliditylang.org/en/v0.8.17/natspec-format.html

2 Working with data

Let's start with importing the necessary packages and adjusting diplay options

[ ]:
```python
# General packages with functions and methods
import pandas as pd
import numpy as np
import datetime as dt
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
import statsmodels as sms
import statsmodels.formula.api as smf
import datetime as dt
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import researchpy as rp
from IPython.display import display
import pmdarima as pm
from pmdarima.arima import auto_arima


# Display values in numpy arrays as floating point numbers
# instead of with scientific notation.
# This does however not seem to work for the Series object
# provided by the pandas package, so not all scientific
# notation is suppressed.
np.set_printoptions(suppress=True)

# Ignore some of the warnings that we might encounter
# Mostly related to boolean functions on pandas dataframes
# That are planned to be deprecated for future version releases
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
```

```python
# General visualization formatting
from cycler import cycler
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
plt.rcParams["axes.labelsize"]= 12
plt.rcParams["figure.facecolor"] = "#f2f2f2"
plt.rcParams['savefig.edgecolor'] = "#f2f2f2"
plt.rcParams['savefig.facecolor'] ="#f2f2f2"
plt.rcParams["figure.figsize"] = [16,10]
plt.rcParams['savefig.bbox'] = "tight"
plt.rcParams['font.size'] = 14
greens = ['#66c2a4','#41ae76','#238b45','#006d2c','#00441b']
multi␣
 ↪=['#66c2a4','#1f78b4','#a6cee3','#b2df8a','#33a02c','#fb9a99','#e31a1c','#fdbf6f']
plt.rcParams["axes.prop_cycle"] = cycler(color=multi)
```

Commonly used functions in this part of the paper:

[ ]:
```python
def get_df(data: pd.DataFrame, column: str, conditional, condition: str):
    """
    @notice Create a copy of the given dataframe with the
            rows that are true according to the conditional
    @dev    Dataframe query alternative
    @param `data` The dataframe containing all the values
    @param `column` The column which should be filtered by
    @param `conditional` The value to filter on
    @param `condition` The symbol for the conditional operation
    @return A new dataframe with only the rows that
            fit the conditional parameter
    """
    if condition == "==":
        return data[data[column] == conditional]
    elif condition == ">=":
        return data[data[column] >= conditional]
    elif condition == "<=":
        return data[data[column] <= conditional]
```

[ ]:
```python
def change(data: pd.Series, n: int=1):
    """
    @notice Calculate the difference of a row element
            compared to the previous row element
            E.g. [0] 100, [1] 200, change = 100
    @dev    Periods equals to amount of rows to calculate over
    @param  `data` The series you wish to calculate diff for
    @param  `n` The amount of rows to calculate the diff over,
            the default is 1
    @return Series containing the change between the rows
    """
    return data.diff(periods=n)
```

[ ]:
```
def close_high_low(df: pd.DataFrame, high: str, low: str, close: str):
    """
    @notice Find out whether the close price is
            closer to the high or low price
    @dev    Using pandas built in vectorized operations
    @param  `df` The dataframe with the columns
    @param  `high` The column name for the high value
    @param  `low` The column name for the low value
    @param  `close` The column name for the close value
    @return A series containing floating point numbers between 0 and 1.
            High number means the closing price is close to the high price
    """
    low_delta = df[close] - df[low]
    volatility = df[high] - df[low]
    return low_delta / volatility
```

[ ]:
```
def close_open_ratio(df: pd.DataFrame, o: str, c: str, vol: str):
    """
    @notice Find how close the open and close price for the period
            is in relation to the volatility within the same period
    @dev    Simple array subtraction and division
    @param  `df` The dataframe with the values
    @param  `o` The column name containing the open price
    @param  `c` The column name containing the close price
    @param  `vol` The column name containing the volatility
    @return An array with the ratio values. 0 is close, 1 is far
    """
    open_close = abs(df[c] - df[o])
    return open_close / df[vol]
```

[ ]:
```
def hammer(df: pd.DataFrame, openPrice: str, closePrice: str, vol: str):
    """
    @notice Add boolean columns that indicate wether a candle is a
            Hammer candle, Inverted hammer candle or neither
    @dev    Numpy filtering based on conditions within the candle
    @param  `df` The dataframe to perform the operations on
    @param  `openPrice` The column name for the column with the opening price
    @param  `closePrice` The column name for the column with the closing price
    @param  `vol` The column name for the column with the
            volatility within the period
    @return A dataframe containing all the new columns
    """
    temp_df = df
    temp_df["VolInterval"] = pd.qcut(df[vol], [0, 0.95, 1.], labels=["Low",␣
↪"High"])
    temp_df["5%"] = np.where(
    temp_df["VolInterval"] == "High", True, False)
    temp_df["Hammer"] = np.where(
    (temp_df["5%"] == True) & (temp_df["COR"] <= 0.2) & (temp_df["CloseVol"] >=␣
↪0.8), True, False)
    temp_df["Inverted"] = np.where(
    (temp_df["5%"] == True) & (temp_df["COR"] <= 0.2) & (temp_df["CloseVol"] <=␣
↪0.2), True, False)
    return temp_df
```

2.1 1m Spot Data:

```
[ ]: spot = pd.read_csv("./datasets/BTCUSDT-spot.csv")
     spot = spot.drop([
         "Unnamed: 0", "KlineOpenTime", "KlineCloseTime",
         "Unused", "NumberOfTrades", "QuoteAssetVolume",
         "TakerBuyQuoteAssetVolume-USD", "TakerBuyBaseAssetVolume-BTC", "Volume"
     ], axis=1)
     print(spot.shape)
     spot.head()
```

```
(524686, 5)
```

```
[ ]:   OpenPrice  HighPrice  LowPrice  ClosePrice                Date
     0  45134.11   45266.77  45130.34    45260.74 2021-03-01 00:00:00
     1  45252.67   45362.07  45250.64    45356.00 2021-03-01 00:01:00
     2  45356.00   45371.41  45104.36    45128.57 2021-03-01 00:02:00
     3  45128.57   45194.65  45020.87    45037.36 2021-03-01 00:03:00
     4  45036.62   45107.01  44977.82    45032.48 2021-03-01 00:04:00
```

Dropping the columns with data that is not going to be used throughout this analysis, we're left with very few variables, all of which are tied to the price of the asset in the period. - OpenPrice: The price at the beginning of the period - HighPrice: The highest price at which a trade occured within the period - LowPrice: The lowest price at which a trade occured within the period - ClosePrice: The price at the end of the period - Date: The year, month, day, hour, and minute for the period

Being left with only the price of the asset might seem like not much to perform analysis on, but creating new variables based on the price increases the scope of the analysis substantially. The fact that the majority of the data provided was dropped is however something to keep in mind. The dropped columns regarding total volume and volume by market participants (Maker / Taker) could for example have been used to analyze future price movements based on the recently traded volume by the active market participants.

Let's expand on the dataframe with some new columns based on the information in the columns above. - Vol: The volatility of the price within the period - CloseVol: The close price in relation to the volatility within the period - Change: USD denominated change in the asset price between periods - Percentage: The percentage change in the asset price between periods - COR: Close-to-open ratio, the absolute value between the open and close price divided by the volatility - Hammer: Boolean value for whether the entry is a hammer candle or not - Inverted: Boolean value for whether the entry is an inverted hammer candle or not

```
[ ]: # Create new dataframe with the desired columns
data = {
    "Date": pd.to_datetime(spot["Date"]),
    "Close": spot["ClosePrice"],
    "High": spot["HighPrice"],
    "Low": spot["LowPrice"],
    "Open": spot["OpenPrice"],
    "Vol": spot["HighPrice"] - spot["LowPrice"],
    "CloseVol": close_high_low(spot, "HighPrice", "LowPrice", "ClosePrice"),
    "Change": change(spot["ClosePrice"]),
    "Percentage": round(spot["ClosePrice"].pct_change() * 100, 3),
    }

spot = pd.DataFrame(data)
spot["COR"] = close_open_ratio(spot, "Open", "Close", "Vol")
spot = hammer(spot, "Open", "Close", "Vol")
# Dropping columns that are not gonna be used anymore
spot = spot.drop(["Open", "High", "Low", "VolInterval", "5%"], axis=1)
print(spot.shape)
spot.head()
```

```
(524686, 9)
```

```
[ ]:                  Date     Close     Vol  CloseVol  Change  Percentage  \
0 2021-03-01 00:00:00  45260.74  136.43  0.955802     NaN         NaN
1 2021-03-01 00:01:00  45356.00  111.43  0.945526   95.26       0.210
2 2021-03-01 00:02:00  45128.57  267.05  0.090657 -227.43      -0.501
3 2021-03-01 00:03:00  45037.36  173.78  0.094890  -91.21      -0.202
4 2021-03-01 00:04:00  45032.48  129.19  0.423098   -4.88      -0.011

        COR  Hammer  Inverted
0  0.928168   False     False
1  0.927309   False     False
2  0.851638   False     False
3  0.524859   False     False
4  0.032046   False     False
```

Out of all the columns in the dataframe, only the *Close* column containing the price at the end of the period is non-stationary.

Let's also do the same transformations to the futures market data so we can compare the two markets. The data for both markets are delivered in the same format from the API, so the process used to transform the spot data is just replicated with a different filename and a different variable name for the dataframe.

2.2 1m Futures Data

[ ]:
```
futures = pd.read_csv("./datasets/BTCUSDT-futures.csv")
futures["Date"] = pd.to_datetime(futures["KlineOpenTime"], unit="ms")
futures = futures.drop([
    "Unnamed: 0", "KlineOpenTime", "KlineCloseTime",
    "Unused", "NumberOfTrades", "QuoteAssetVolume",
    "TakerBuyQuoteAssetVolume-USD", "TakerBuyBaseAssetVolume-BTC", "Volume"
], axis=1)
print(futures.shape)
futures.head()
```

```
(525600, 5)
```

[ ]:
```
   OpenPrice  HighPrice  LowPrice  ClosePrice                Date
0   45162.64   45300.00  45150.13    45273.53 2021-03-01 00:00:00
1   45276.87   45387.06  45273.53    45374.12 2021-03-01 00:01:00
2   45374.11   45389.00  45147.47    45170.62 2021-03-01 00:02:00
3   45172.19   45221.04  45059.77    45066.99 2021-03-01 00:03:00
4   45065.81   45136.79  45011.49    45054.74 2021-03-01 00:04:00
```

[ ]:
```
# Create new dataframe with the desired columns
data = {
    "Date": pd.to_datetime(futures["Date"]),
    "Close": futures["ClosePrice"],
    "High": futures["HighPrice"],
    "Low": futures["LowPrice"],
    "Open": futures["OpenPrice"],
    "Vol": futures["HighPrice"] - futures["LowPrice"],
    "CloseVol": close_high_low(futures, "HighPrice", "LowPrice", "ClosePrice"),
    "Change": change(futures["ClosePrice"]),
    "Percentage": round(futures["ClosePrice"].pct_change() * 100, 3),
    }

futures = pd.DataFrame(data)
futures["COR"] = close_open_ratio(futures, "Open", "Close", "Vol")
futures = hammer(futures, "Open", "Close", "Vol")
# Dropping columns that are not gonna be used anymore
futures = futures.drop(["Open", "High", "Low", "VolInterval", "5%"], axis=1)
print(futures.shape)
futures.head()
```

```
(525600, 9)
```

[ ]:
```
                 Date     Close     Vol  CloseVol  Change  Percentage  \
0 2021-03-01 00:00:00  45273.53  149.87  0.823380     NaN         NaN
1 2021-03-01 00:01:00  45374.12  113.53  0.886021  100.59       0.222
2 2021-03-01 00:02:00  45170.62  241.53  0.095847 -203.50      -0.448
3 2021-03-01 00:03:00  45066.99  161.27  0.044770 -103.63      -0.229
```

```
4 2021-03-01 00:04:00  45054.74  125.30  0.345172  -12.25      -0.027

        COR  Hammer  Inverted
0  0.739908   False     False
1  0.856602   False     False
2  0.842504   False     False
3  0.652322   False     False
4  0.088348   False     False
```

The dataframes are of different size, where the spot market dataframe has 914 less rows than the futures dataframe. These missing rows are randomly scattered, indicating small but frequent periods of outages in the spot market, likely tied to maintenance and upgrades to the matching engine and order book for this market. This amounts to less than 0.2% of the entries missing, which I consider to not be significant for the purpose of this paper.

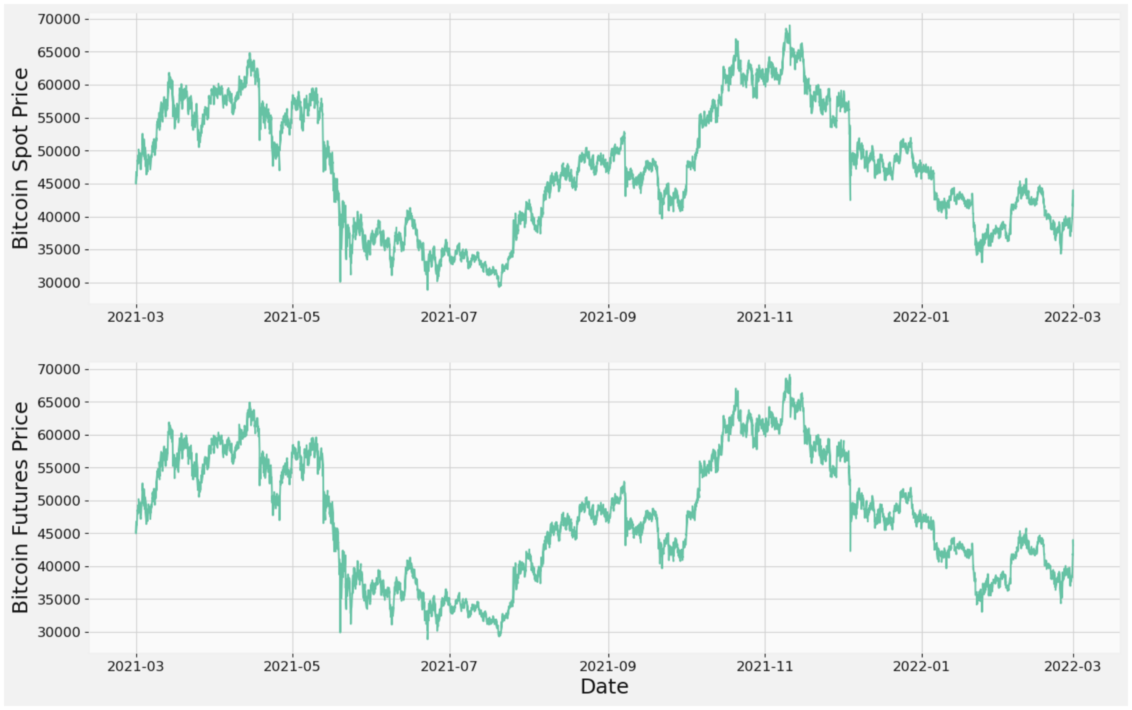
3 Simple analysis

Performing analysis on all types of data requires a starting point, and for time series data, the development of the main variable(s) over time can be a good place to start. As we will later will look at the returns in the trailing periods for hammer and inverted hammer candles in both spot and futures markets, we can start the analysis by looking at both the non-stationary and stationary variables in both of the markets. We can then check the standard deviation in both of the markets and run simple regressions to estimate the price sensitivity in the markets in relation to each other.

3.1 Price Action plot

For the first part of the analysis we can take a look at the non-stationary price action of Bitcoin in both the spot and futures markets. As mentioned in the introduction of the paper, there shouldn't be any visually significant differences between the market prices as inefficiencies are arbitraged quickly by the market participants.

[ ]:
```
fig, axs = plt.subplots(2)
axs[0].plot(spot["Date"], spot["Close"])
axs[0].set_ylabel("Bitcoin Spot Price", fontsize=18)
axs[1].plot(futures["Date"], futures["Close"])
axs[1].set_xlabel("Date", fontsize=18)
axs[1].set_ylabel("Bitcoin Futures Price", fontsize=18)
plt.show()
```



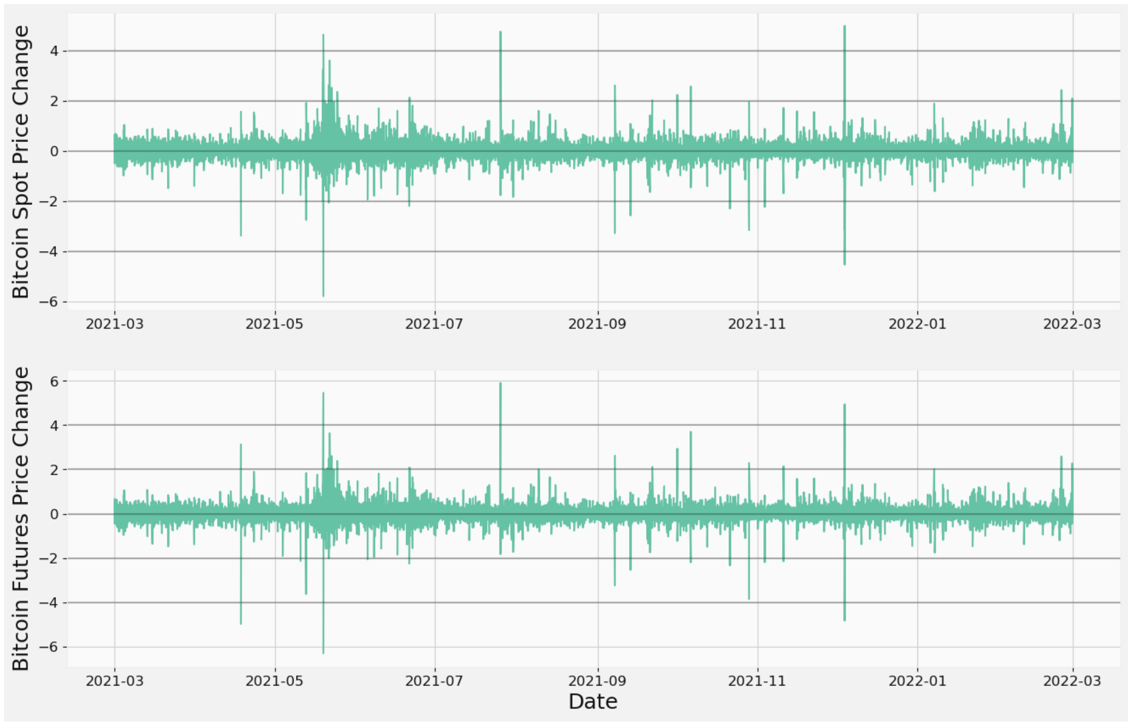
Expectedly, the price of Bitcoin in both the spot and futures markets are closely correlated, with no obvious differences by looking at the prices for the period. If we instead take a look at the stationary series, or the price change between each period, we might be able to spot differences in periods of increased volatility where the arbitrage activity does not keep up with the price change in the different markets.

3.2 Volatility plot

Volatility plots can be made with both the difference between the periods as well as the percentage difference between periods. We can see in the price action plots above that the price varies quite a bit during the period, and by using the percentage return for the volatility plots we can express the volatility over the whole period relative to the price of the underlying asset in both the spot and futures markets.

[ ]:
```
fig, axs = plt.subplots(2)
axs[0].plot(spot["Date"], spot["Percentage"])
axs[0].set_ylabel("Bitcoin Spot Price Change", fontsize=18)
axs[1].plot(futures["Date"], futures["Percentage"])
axs[1].set_xlabel("Date", fontsize=18)
axs[1].set_ylabel("Bitcoin Futures Price Change", fontsize=18)
# Adding horizontal lines so it is easier to spot differences
axs[0].axhline(4, color="black", alpha=0.25)
axs[0].axhline(2, color="black", alpha=0.25)
axs[0].axhline(0, color="black", alpha=0.25)
axs[0].axhline(-2, color="black", alpha=0.25)
```

```
axs[0].axhline(-4, color="black", alpha=0.25)
axs[1].axhline(4, color="black", alpha=0.25)
axs[1].axhline(2, color="black", alpha=0.25)
axs[1].axhline(0, color="black", alpha=0.25)
axs[1].axhline(-2, color="black", alpha=0.25)
axs[1].axhline(-4, color="black", alpha=0.25)
plt.show()
```



By looking at the scale of the y-axes in the plots, it's obvious that the price changes in the futures market can be more volatile than the returns in the spot market. As was stated above, it is easier to spot some differences between the markets with the plot of the stationary data, where some of the spikes that happen at the same time in both markets have different values. Examples of this can be seen on the positive and negative spikes just before May 2021, where the positive spike in the spot market indicates a return of less than 2% while the futures market return was about 3%. Similarly, the negative spike at about the same timestamp seems to be about -2% lower in the futures market than the spot market.

We can also take a look at the standard deviation of percentage returns between periods to see if there are volatility differences throughout the whole year between the spot and futures markets.

```
[ ]: print("Spot:")
     print(spot["Percentage"].std())
     print("Futures:")
     print(futures["Percentage"].std())
```

```
Spot:
0.1160342047680262
Futures:
0.11807782875047175
```

The standard deviations for the returns between periods are very low in both markets, and the futures market seems to be slightly more volatile than the spot market.

To check the sensitivity in the relationship between the markets we can run a regression with the percentage price change of one market as the independent variable on the corresponding variable of the other market as the dependent variable.

```
[ ]: df1 = pd.DataFrame({"Date": pd.to_datetime(spot["Date"]), "Spot":␣
      ↪spot["Percentage"]})
     df1 = df1.set_index("Date")
     df2 = pd.DataFrame({"Date": pd.to_datetime(futures["Date"]), "Futures":␣
      ↪futures["Percentage"]})
     df2 = df2.set_index("Date")
     reg_df = df1.join(df2, on="Date").dropna()
     print(reg_df.shape)
     reg_df.head()
```

```
(524685, 2)
```

```
[ ]:                       Spot  Futures
     Date
     2021-03-01 00:01:00  0.210    0.222
     2021-03-01 00:02:00 -0.501   -0.448
     2021-03-01 00:03:00 -0.202   -0.229
     2021-03-01 00:04:00 -0.011   -0.027
     2021-03-01 00:05:00 -0.146   -0.145
```

By joining the market data from both markets on the *Date* variable, we're left with the same amount of rows as the spot dataframe. The Futures entries with no corresponding spot entry have now been dropped.

As this regression is just to show a relationship between periods that occur at the same time in both the spot and futures markets, and not used to predict future values, we can run the regression with the differents markets as both the independent and dependent variables to compare the model outputs for which coefficient estimation that fits best.

```
[ ]: sr1 = smf.ols(formula='Spot~Futures', data=reg_df).fit()
     print(sr1.rsquared_adj, sr1.aic, sr1.bic)
```

```
0.9654389857769082 -2536793.543892044 -2536771.202785321
```

```
[ ]: sr2 = smf.ols(formula='Futures~Spot', data=reg_df).fit()
     print(sr2.rsquared_adj, sr2.aic, sr2.bic)
```

```
0.9654389857769082 -2518045.1984623554 -2518022.8573556324
```

Both of the models have the same adjusted r squared value, but the AIC and BIC values are different. The AIC and BIC values are measures for the error in the model performance, and a lower error indicates that the model is better at predicting the values of the dependent variable when new independent variable data is given. Based on this it seems like the model with the percentage price change in the spot market as the dependent variable is the better model of the two, as both the AIC and BIC values are lower than the other regression.

We can then look at the summary of the model with the lowest AIC and BIC values to see the coefficient indicating the percentage price change in the spot market given the change in the futures market.

```
[ ]: sr1.summary()
```

```
[ ]: <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Spot   R-squared:                       0.965
Model:                            OLS   Adj. R-squared:                  0.965
Method:                 Least Squares   F-statistic:                 1.466e+07
Date:                Wed, 21 Dec 2022   Prob (F-statistic):               0.00
Time:                        22:07:44   Log-Likelihood:             1.2684e+06
No. Observations:              524685   AIC:                        -2.537e+06
Df Residuals:                  524683   BIC:                        -2.537e+06
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   2.317e-06   2.98e-05      0.078      0.938   -5.61e-05    6.07e-05
Futures        0.9652      0.000   3828.407      0.000       0.965       0.966
==============================================================================
Omnibus:                   905279.665   Durbin-Watson:                   2.791
Prob(Omnibus):                  0.000   Jarque-Bera (JB):    1724090789757.962
Skew:                          -9.855   Prob(JB):                         0.00
Kurtosis:                    8883.457   Cond. No.                         8.47
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

The coefficient is slightly below 1 at 0.9652, with a low standard error and a 95% confidence interval between 0.965 and 0.966. This indicates that the price of Bitcoin in the futures market is more sensitive, or volatile, than the price in the spot market. Given a 1% return in the futures market, one could expect the return in the spot market in the same period to be about 0.965%.

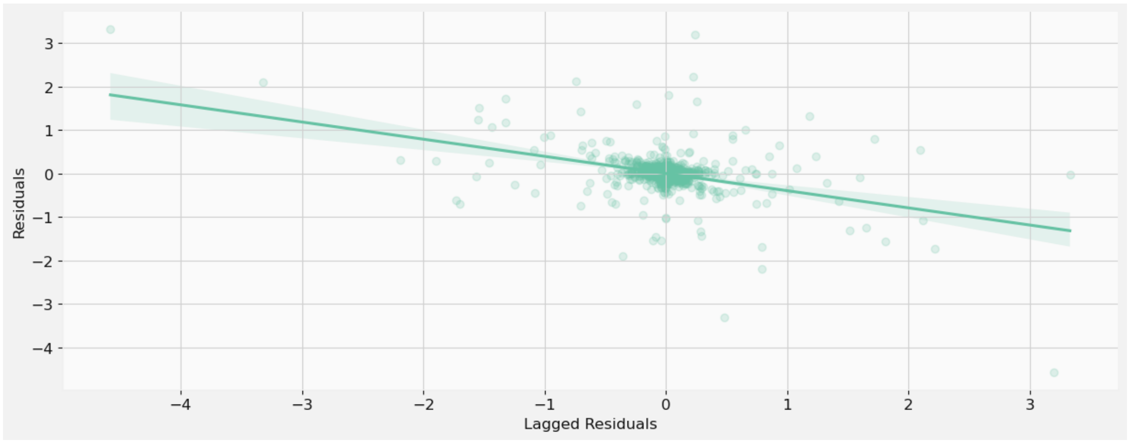
As we were able to visually spot a few differences in the volatility plot, it is likely that the price changes in the futures market are greater relative to the price changes in the spot market during periods of increased volatility. As the standard deviation of the percentage returns in the futures market was only slightly higher than the spot market, it is likely that the price change in both markets are very close in periods of less volatility.

The first requirement that a candle had to meet in order to be classified as a hammer or inverted hammer candle was that the volatility within the period was among the top 5% of the recorded values. If the assumption that the futures market is more sensitive than the spot market in the periods of increased volatility is correct, both the mean and standard deviation of the returns in the trailing period after hammer and inverted hammer candles in the futures market might have higher absolute values than the returns of the same periods in the spot market, which we will look more at in the next section.

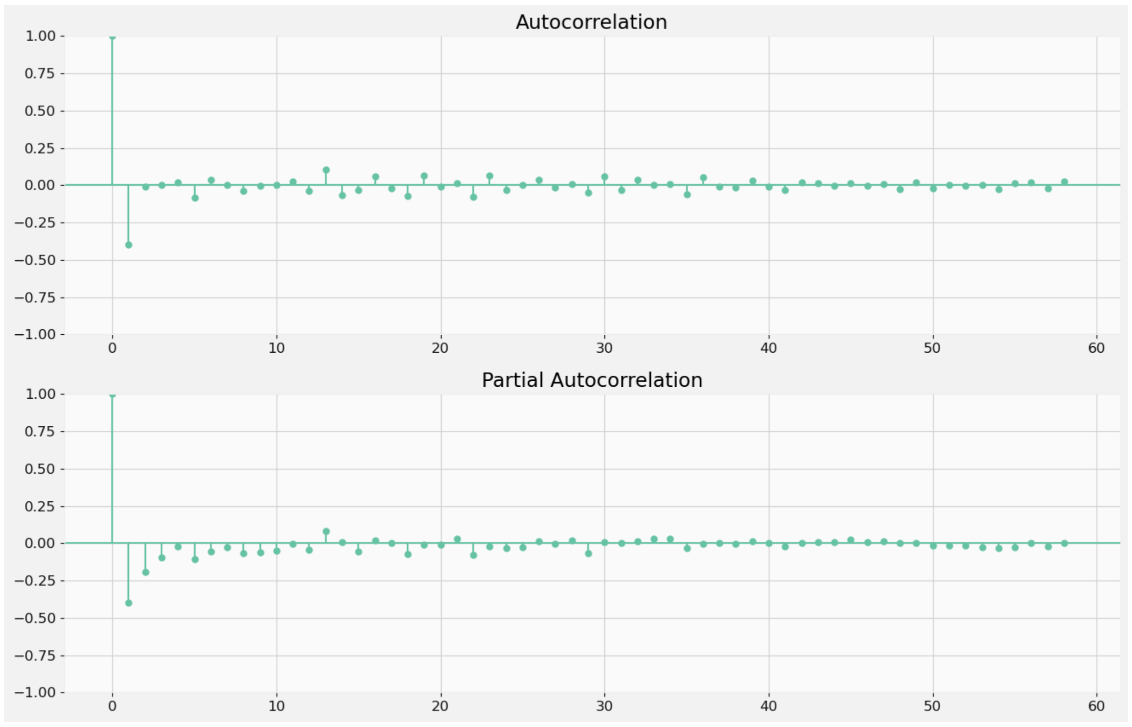
We can also take a look at the relationship between the residuals and lagged residuals of the model to see how it adapts to significant changes or shocks. The scatter plot below shows that if a residual in period $t - 1$ is positive the residual in period t is slightly negative, and similarly if the residual in period $t - 1$ is negative, the residual in period t is slightly positive. This indicates a negative correlation between the two, which can be seen in the autocorrelation and partial autocorrelation plots below.

[ ]:
```
resids = sr1.resid
temporary_df = pd.DataFrame({"Residuals": resids, "Lagged Residuals": resids.
 ↪shift()})
ax = sns.lmplot(x="Lagged Residuals", y="Residuals", data=temporary_df,␣
 ↪height=5, aspect=2.5, scatter_kws={"alpha":0.2})
ax.set(xlabel='Lagged Residuals', ylabel='Residuals')
```

[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x23b93843040>
```



[ ]:
```
fig, ax = plt.subplots(2)
plot_acf(resids, ax=ax[0])
plot_pacf(resids, ax=ax[1])
plt.show()
```



Autocorrelation and partial autocorrelation are measures of association between current and past series values and indicate which past series values are most useful in predicting future values. Both the autocorrelation and the partial autocorrelation show a negative correlation coefficient directly after the initial shock, meaning that a shock in one direction in a period tends to lead a residual in the opposite direction in the next period. In other words, if the residual in period t is positive, the residual in period $t + 1$ is slightly negative.

With this in mind, the volatility plots at the beginning of the simple analysis might be explained a bit better. For the majority of the timestamps where we can see a significant spike in one direction, there is likely a significant spike in the other direction as well. For the most part the plots for both of the spot and futures markets looks to be almost mirrored on the horizontal line at y=0 with somewhat diminished returns, which corresponds well with the negative correlation after a shock.

4 Simulation

Trading strategies are often derived from a hypothesis about the market, and need to be tested before they are implemented in real markets. There are multiple ways of testing such strategies, but the two most popular are *monte carlo simulations* and *backtests*. Backtesting describes a class of models that use previous trading data to see how the strategy would have performed in the given periods of the sample data. These types of tests are good at showing how a strategy would have performed, but they also lack some critical aspects one would like to address when the strategy eventually involves managing real money. The most common flaw that backtest models are criticized for is bias in the sample data. This essentially means that the data you test the strategy on fits the strategy better than how the strategy would have performed on current market data.

Monte carlo simulation of the returns on a strategy is generally considered a better model than regular backtesting, but it is also significantly more complex. Often these types of simulations are based on the probability of where an asset moves, and it captures the possible returns based on a normal distribution of the real mean and standard deviation of the asset. In contrast to the simple backtest, this strategy captures the probability of the possible returns and is therefore significantly better in terms of managing risk than the backtest.

The analysis of the trading strategy presented here, by entering a trade after hammer and inverted hammer candles, uses a simulation based on the data from a simple backtest of the performance of the strategy over the course of the year the data is from. As the backtest is used to derive the values used for the simulation, we're not able to completely address the issues presented with the backtest in the analysis. As with most models, a bad input often leads to a bad output, and this is the case here as well. If the sample data the strategy is tested on is biased in either direction compared to a bigger dataset or current market conditions, this will also be reflected in the simulation. The first requirement presented for candles to be classified as hammer or inverted hammer candles in the introduction was that the volatility within the period was among the top 5% in the dataset, and this is something that will affect the bias in the strategy relative to the sample data as well. If the dataset the backtest is conducted on was larger or from a different period, the values for classifying the hammer and inverted hammer candles would also be different and it might not have been reliable to compare these results. Using the top 5% volatility measure for classifying candles would also mean that it cannot be easily used to classify real-time data unless you keep a rolling database with the constantly adapting values of the rolling period.

For this analysis, the amount of trailing values we will sum and simulate based on will be the 5 periods following the hammer and inverted hammer candles. E.g. if a hammer candle can be spotted in the period t, we will sum the returns in the periods $t+1$ through $t+6$. This number is somewhat randomly picked, but it is based to some extent on the autocorrelation and partial autocorrelation plots in the last section that shows little to no correlation to a shock after 5 periods.

[ ]:
```python
# The amount of trailing candles to sum
Trailing_Candles = 5
```

Commonly used functions in this part of the paper:

[ ]:
```python
def get_trailing(data: pd.DataFrame, indices: list, n: int, column: str):
    """
    @notice Get the sum of all values from `data[column]` that follow
            the index given in `indices` up to `n` values after the index
    @dev    Not vectorized, use with caution and be patient
    @param  `data` The dataframe containing all the values
    @param  `indices` A list of indices to sum trailing values after
    @param  `n` The amount of trailing values (candles) to sum
    @param  `column` The column to sum trailing values from
    @return An array with the sum of trailing values after
            moves specified by the list of indices
    """
    vals = np.array(0)
    for i in indices:
        temp = 0
        for j in range(n):
            if j == 0:
                temp += data.loc[(i + 1), column]
            else:
                temp += data.loc[(i + (1 + j)), column]
        vals = np.append(vals, temp)
    vals = np.delete(vals, 0)
    return vals
```

```
[ ]: def simulate(a: pd.Series, a_desc: str, b: pd.Series, b_desc: str, S: int=150,␣
      ↪N: int=10000):
         """
         @notice Simulate the difference in means of two series of data
         @dev    Call to display statistics and plot. No return values
         @param  `a` A pd.Series for one group
         @param  `a_desc` String with the name for series `a`
         @param  `b` A pd.Series for another group
         @param  `b_desc` String with the name for series `b`
         @param  `S` The sample size, standard 150
         @param  `N` The amount of simulations, standard 10,000
         @return None, displays below the function call
         """
         std_scale = b.std() / a.std()

         est_pos = pd.Series(np.random.normal(a.mean(), a.std(), S))
         meanEst_pos = est_pos.mean()
         meanStd_pos = est_pos.std()

         est_neg = pd.Series(np.random.normal(b.mean(), b.std(), S))
         meanEst_neg = est_neg.mean()
         meanStd_neg = est_neg.std()

         meanDiff = []
         mean_pos = []
         mean_neg = []
         std = []

         for i in range(N):
             sample_pos = np.random.normal(a.mean(), a.std(), S).mean()
             mean_pos.append(sample_pos)
             sample_neg = np.random.normal(b.mean(), b.std(), S).mean()
             mean_neg.append(sample_neg)
             sample = sample_pos - sample_neg
             meanDiff.append(sample.mean())

         meanDiff = pd.Series(meanDiff)
         mean_pos = pd.Series(mean_pos)
```

```
    mean_neg = pd.Series(mean_neg)

    summary, results = rp.ttest(group1= mean_pos, group1_name= a_desc,
                                group2= mean_neg, group2_name= b_desc)
    summary = pd.DataFrame(summary)
    display(summary)
    display(results)

    b_scale = int(50*std_scale)

    fig, ax = plt.subplots(2)
    ax[0].set_title("Difference in means simulation", fontsize=22)
    ax[0].set_ylabel("Frequency", fontsize=18)
    ax[0].set_xlabel("Mean Return", fontsize=18)
    ax[1].set_ylabel("Frequency", fontsize=18)
    ax[1].set_xlabel("Mean Difference", fontsize=18)
    mean_pos.hist(bins=50, ax=ax[0], alpha=.5, color="blue", label=a_desc)
    mean_neg.hist(bins=b_scale,  ax=ax[0], alpha=.5, color="red", label=b_desc)
    meanDiff.hist(bins=50, ax=ax[1], label="Mean difference")
    fig.legend()
```

To conduct this simulation we first have to find all the relevant entries in the spot and futures dataframes. As the dataframes already have columns that indicate whether a candle is a hammer or inverted hammer candle, we can use the *get_df* function presented at the beginning of the previous section to extract the indices of the relevant candles, and then use those to get the total return of the 5 trailing candles.

```
[ ]: # Spot hammer and inverted hammer candles
     hammer_spot = get_df(spot, "Hammer", True, "==")
     inverted_spot = get_df(spot, "Inverted", True, "==")
```

```
[ ]: # Futures hammer and inverted hammer candles
     hammer_futures = get_df(futures, "Hammer", True, "==")
     inverted_futures = get_df(futures, "Inverted", True, "==")
```

```
[ ]: # Spot hammer and inverted hammer trailing sums
     trailing_hammer_spot = get_trailing(spot, hammer_spot.index, Trailing_Candles,␣
      ↪"Change")
     trailing_inverted_spot = get_trailing(spot, inverted_spot.index,␣
      ↪Trailing_Candles, "Change")
```

```
[ ]: # Futures hammer and inverted hammer trailing sums
     trailing_hammer_futures = get_trailing(futures, hammer_futures.index,␣
      ↪Trailing_Candles, "Change")
     trailing_inverted_futures = get_trailing(futures, inverted_futures.index,␣
      ↪Trailing_Candles, "Change")
```

```
[ ]: # Display the sum, mean, std and frequency of the hammer and inverted hammer␣
     ↪candles in each of the two markets
     trailing_data = {
         "Sum": [round(trailing_hammer_spot.sum(), 2), round(trailing_inverted_spot.
     ↪sum(), 2),
                 round(trailing_hammer_futures.sum(), 2),␣
     ↪round(trailing_inverted_futures.sum(), 2)],
         "Mean": [round(trailing_hammer_spot.mean(), 2),␣
     ↪round(trailing_inverted_spot.mean(), 2),
                  round(trailing_hammer_futures.mean(), 2),␣
     ↪round(trailing_inverted_futures.mean(), 2)],
         "Std": [round(trailing_hammer_spot.std(), 2), round(trailing_inverted_spot.
     ↪std(), 2),
                 round(trailing_hammer_futures.std(), 2),␣
     ↪round(trailing_inverted_futures.std(), 2)],
         "Amount": [len(trailing_hammer_spot), len(trailing_inverted_spot),
                    len(trailing_hammer_futures), len(trailing_inverted_futures)]
     }
     display(pd.DataFrame(trailing_data, index=["Spot Hammer", "Spot Inverted",␣
     ↪"Futures Hammer", "Futures Inverted"]))
```

```
                       Sum   Mean     Std  Amount
Spot Hammer       13980.76  27.85  215.81     502
Spot Inverted     -9078.12 -20.54  339.57     442
Futures Hammer    12191.57  31.18  259.97     391
Futures Inverted -18106.11 -48.28  344.21     375
```

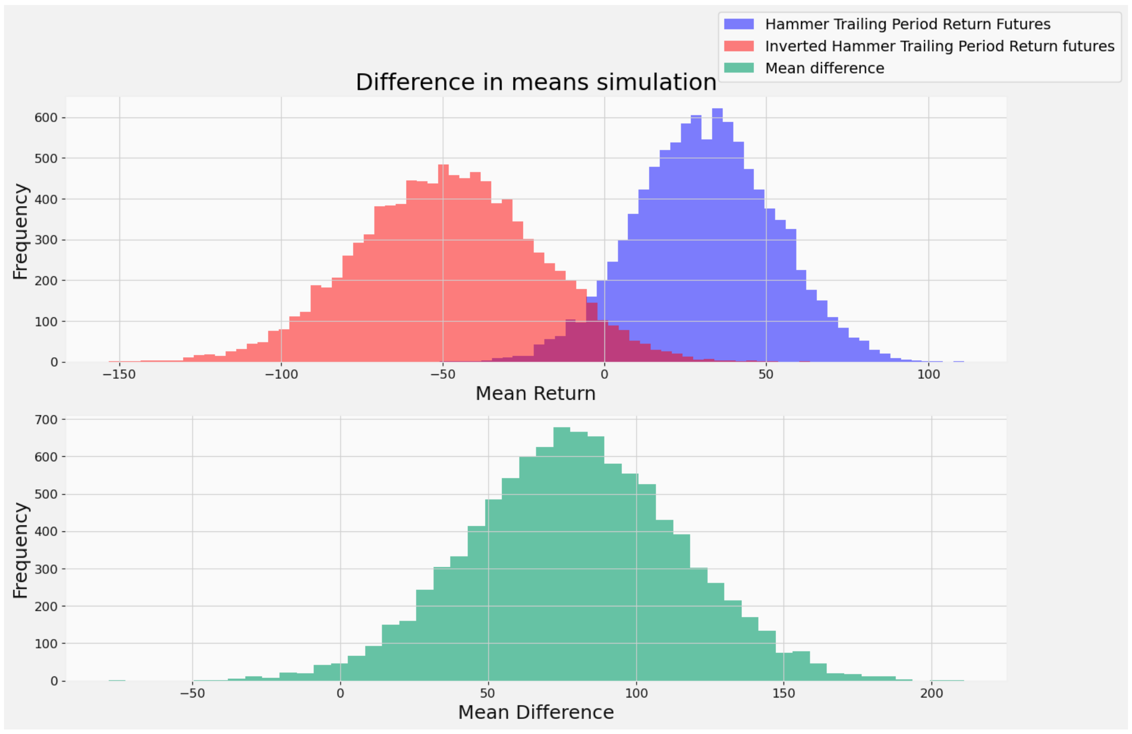
This dataframe sums up the returns of the trailing values in the different markets well, and the most obvious difference between the different candle patterns is that the theory of hammer candles being bullish and inverted hammer candles being bearish seems to be consistent in both the spot and futures markets. The total return for the 5 trailing candles after a hammer candle is relatively close in the spot and futures markets, with less than 15% difference between the two. The total return for the 5 trailing candles after an inverted hammer candle shows a somewhat different relationship in the spot and futures markets, where the latter has about twice the (negative) return of the former, showing that the futures market is significantly more bearish after an inverted hammer candle than the spot market is. This can also be seen by the mean for the two markets, where the mean for the trailing period after inverted hammer candles in the futures market is significantly lower than that of the spot market, while standard deviations are very close.

Another interesting thing to note is the difference in the amount of the patterns that can be observed in the markets, where the spot market saw over 25% more hammer candles than the futures market in the period. The difference between the markets for inverted hammer candles is slightly lower, but there is still over 15% more inverted hammer candles in the spot market than the futures market for the period. The difference between the markets is significant, however it is diffidult to estimate why this is the case based on the current data in the dataframes. One hypothesis might be that since the futures market has slightly higher volatility in general, some of the hammer and inverted hammer candles that was spotted in the spot market, and was likely closely followed in the futures market, might not have been in the top 5% of volatility within periods, and therefore some have been excluded. This could be further investigated by trying different changes to the critical values mentioned in the introduction, but in this case we'll take a look at the data we have currently.

As the dataframe above shows a bigger difference between the returns after the hammer and inverted hammer candles in the futures market, like the hypothesis based on the simple analysis, that is the market we'll simulate the difference in means for and discuss in the rest of this section.

```
[ ]: simulate(trailing_hammer_futures, "Hammer Trailing Period Return Futures",
              trailing_inverted_futures, "Inverted Hammer Trailing Period Return␣
      ↪futures")
```

```
                                         Variable        N       Mean  \
0           Hammer Trailing Period Return Futures  10000.0  31.043041
1  Inverted Hammer Trailing Period Return futures  10000.0 -48.303843
2                                        combined  20000.0  -8.630401

          SD        SE  95% Conf.   Interval
0  21.332258  0.213323  30.624886  31.461196
1  27.993969  0.279940 -48.852581 -47.755105
2  46.833679  0.331164  -9.279510  -7.981292

                                Independent t-test     results
0  Difference (Hammer Trailing Period Return Futu…     79.3469
1                            Degrees of freedom =   19998.0000
2                                             t =     225.4457
3                         Two side test p value =       0.0000
4                        Difference < 0 p value =       1.0000
5                        Difference > 0 p value =       0.0000
6                                     Cohen's d =       3.1883
7                                     Hedge's g =       3.1882
8                                 Glass's delta1 =       3.7196
9                              Point-Biserial r =       0.8471
```



The dataframes printed out shows the summary of the t-test on the two groups, for whether the candle leading the trailing period was a hammer or inverted hammer candle. The main takeaway from the first dataframe is the confidence interval for the mean of both series as well as their standard deviations. Based on the confidence interval for the mean, we can see that it is very narrow for both of the series, which shows a high confidence in the difference between the two groups.

The second dataframe displays the different p-value tests, as well as some other values that represent the results of their respective effect size measure formulas. The two-sided test's p-value is 0, indicating that there is a statistical difference between the means of the groups. The test for whether the difference between the groups is less than 0, measured as the mean return of the trailing candles after the hammer candles minus the mean return of the trailing candles after the inverted hammer candles, is equal to 1. The test for whether the difference between the same groups is greater than 0, has a p-value of 0. This indicates that you can with confidence say that the mean return of the 5 trailing candles after hammer candles is greater than the mean return of the 5 trailing candles after inverted hammer candles.

The effect size measure formulas are used to describe the intensity of the relationship between the two groups. These formulas are more typically used to analyze a test group and a control group, but can also be applied to the test conducted above. *Cohen's d* and *Hedge's g* shows the difference in means of the two groups measured in the weighted standard deviation of both groups. In our case the values given by these are almost identical at roughly 3.2 (might differ slightly as the simulation is conducted again), indicating over 3 standard deviations of difference between the means of the groups. *Glass's delta* measures the same effect as the two other mentioned formulas, but it only uses the standard deviation of the control group. As the standard deviation of the mean for the trailing periods after inverted hammer candles is about 1.33 times higher than the standard deviation of the mean for the trailing periods after hammer candles, the formulas that use the weighted standard deviation shows a better relationship between the two distributions. The values show over a 3 standard deviation difference between the means, which further shows that the result of the simulation is that there is a significant difference between the means of the groups.

Based on the information from the observed returns after the hammer and inverted hammer candles as well as the difference in means simulation above we can create a simple trading strategy that takes advantage of both of the patterns in order to capture the spead between the two.

From the dataframe that displayed the general information about the trailing periods after the two patterns in the different markets, we can see that the futures market had about the same amount of hammer candles as inverted hammer candles, 391 and 375 respectively. During the whole year, we observed about 5% less inverted hammer candles than hammer candles, which is close enough to assume that the two patterns happen as frequently over a period of time for the sake of this example.

If we also assume that over a period of time we will end up with the mean returns that we observed and simulated, the return on the Bitcoin price for the 5 minutes after a hammer candle is 31 USD, and the return for the 5 minutes after an inverted hammer candle is -48 USD. By entering a short position after inverted hammer candles, which consists of borrowing and selling a given amount at time t and buying the same amount back at $t + 5$ to pay back the borrow amount, we can capitalize on the price action even when the price decreases. Similarly by entering a long position after a hammer candle, we can capitalize on the upside in the price action. Combining these two trades and if the patterns indicating hammer and inverted hammer candles occur the same amount of times, the strategy yields about 79 USD on the base asset for every execution of the strategy (2 trades). This is the return on the price of one Bitcoin, so if both the long and short position consisted of one whole Bitcoin, this would be the profit of the trade. The price did however hover between 30,000 and 70,000 over the year, so the average yield in percentage terms is very low, at about 0.16% for every time the strategy is executed if calculated with the price of one Bitcoin at 50,000. This return can be significantly increased by the use of leverage as mentioned in the introduction, as you can essentially increase the position size over 100x the initial investment.

What we need to keep in mind is that if the markets were this easy to predict, these opportunities likely wouldn't exist in the first place. The standard deviation of the returns was close to 10x that of the mean return for the trailing periods of both the hammer and inverted hammer candles in the futures market, so even if the sum of the returns over the whole year seems relatively high and consistent, the reality is that the returns are based on more variables than those presented in this analysis. As an example, the classification used for the patterns does not take into account trend variables like for example the slope of an exponential moving average, and a hammer candle which in theory (and based on our observations) is a bullish sign can be recognized even when the price of Bitcoin tumbles. Another thing to consider is that the numbers used in the example does not take into account trading fees and borrowing fees (for both shorting and leverage), and since the percentage returns on each pair of trades was very low, the strategy is likely less profitable than it seems, even if the assumptions of the mean returns staying consistent is true.

4.1 Decision analysis

Based on the information we got from the backtest we can use Bayes' theorem to conduct a decision analysis for the trading strategy. For the purpose of this we will pair one short and one long trade as one execution of the strategy as discussed in the previous section. By doing this we can identify the probabilities that the individual trades in the strategy are profitable given that the strategy itself is profitable. The formula for Bayes' theorem is written below, and the events A and B are: - A: The short trade in the strategy is profitable. In other words, the return after inverted hammer candles is negative. - B: The strategy is profitable. In other words, the return from the long + the return from the short is positive.

$P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$

[ ]:
```
# Observed probability that the short trade is profitable
a = np.where(trailing_inverted_futures < 0, True, False)
a = a.sum() / len(a)

# Simulated probability that the short trade is profitable
temp = np.array(0)
for i in range(1000):
    i = np.random.normal(-48.28, 344, 1000)
    x = np.where(i < 0, True, False)
    x = x.sum() / len(x)
    temp = np.append(temp, x)
temp = np.delete(temp, 0)
asim = temp.mean()

print(a)
print(asim)
```

```
0.5626666666666666
0.555643
```

[ ]:
```
# Observed probability that the strategy (long hammer + short inverted) is␣
 ↪profitable
b = trailing_hammer_futures[:375] - trailing_inverted_futures
b = np.where(b>0, True, False).sum() / len(b)

# Simulated probability that the strategy (long hammer + short inverted) is␣
 ↪profitable
temp = np.array(0)
for i in range(1000):
    h = np.random.normal(31.2, 260, 1000)
    i = np.random.normal(-48.28, 344, 1000)
    x = np.where((h-i) > 0, True, False)
    x = x.sum() / len(x)
    temp = np.append(temp, x)
temp = np.delete(temp, 0)
bsim = temp.mean()

print(b)
print(bsim)
```

```
0.56
0.573522
```

We can see that the simulated and observed probabilities for the events are very close, with about 0.01 or 1% difference for both of the events. The probabilities for both of the events are also very close, so we can based on the formula already see that P(A | B) is very close to P(B | A). As the simulations above was close to the actual observed values we can simulate the probability for B given A, in other words the probability that the strategy is profitable given that the short is profitable to find out the probability that the short position in the strategy is profitable given that the strategy is profitable.

```
[ ]: temp = np.array(0)
     for i in range(len(trailing_inverted_futures)):
         if (trailing_inverted_futures[i] < 0):
             rng = np.random.normal(31.2, 260, 1000)
             perc = np.where(rng > trailing_inverted_futures[i], True, False)
             perc = perc.sum() / len(perc)
             temp = np.append(temp, perc)
     temp = np.delete(temp, 0)
     basim = temp.mean()
     print(basim)
```

```
0.7602938388625592
```

As this probability was found by simulation, we can use all of the simulated values to find the probability of A given B (numbers might slightly change when the cells are run after I wrote the formula below).

$P(A|B) = \frac{0.76*0.55}{0.57} = 0.7333$

The estimated probability that the short trade is profitable given that the strategy is profitable is 73.33%, and the estimated probability that the strategy is profitable given that the short trade is profitable is 76.09%. These probabilities tells us that the majority of the profitable paired trades consist of both a profitable long and short trade. The small difference between these two probabilities is likely tied to the difference in the mean and standard deviation of the returns after the hammer and inverted hammer candles. The mean return for the short trade is about 48 USD, which is 55% higher than the return of the long trade at 31 USD. The standard deviation for the short trade is 344, which is about 32% higher than the standard deviation of the long trade. As the delta of the returns is greater than the delta of standard deviations, the probability that the strategy is profitable given that the short trade is profitable, P(B | A), is slightly higher than the probability that the short trade is profitable given that the strategy is profitable, P(A | B).

Now that we have looked at how it's possible to test strategies based on previous market conditions, we can also take a look at another way of trading financial instruments that are based on predicting future price movements based on time series forecasting.

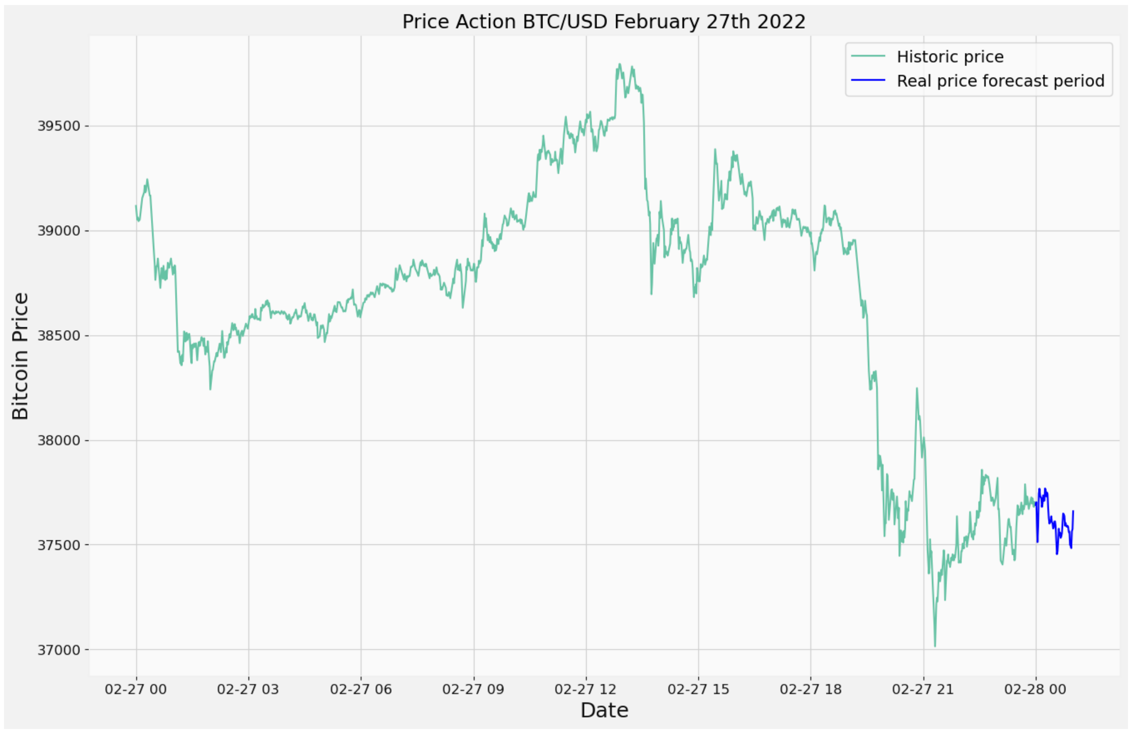
5 Time Series Forecasting

To forecast price movements based on the returns in the previous periods the dataframes needs to be shorter, as my system is not able to run the models on 500,000+ data points due to memory error. Becuase of this, all of the models will run based on 1 day of data, 1,440 data points. The chosen day is the penultimate in the spot dataframe, so it is possible to compare the forecast with the actual price action of the forecasted period which we can find as the last day in the dataframe. Basing a forecast on just one day is however something that could be criticized as the model that fits the best that day might not be the one that fits the best for an extended period of time. For example, one of the auto_arima models I tested on one day of stationary data had the model (1,0,0) as the best fit, while another day had the model (3,0,2) as the best fit.

Time series regression and forecasting can be done based on many different model classes, where ARIMA (autoregressive integrated moving average), (G)ARCH (generalized autoregressive conditional heteroskedasticity), and VAR (vector autoregressive) models are among the most popular classes. The GARCH model class is mainly used for volatility forecasting, and does typically not work as well as other alternative models when the data is in 1 minute intervals. Garch models are popular models for volatility forecasting in the options markets, which has a longer timeframe that typically looks at daily or even weekly trends. VAR models are mostly used for multivariate time series forecasting, which is not the focus of this paper. Hence, we are left with the ARIMA class models.

Let's first gather the data we will use in dataframes for the period as mentioned above

```
[ ]: spot_day = spot[-2880:-1440].set_index("Date")
     spot_forecast = spot[-1440:-1379].set_index("Date")
     fig, ax = plt.subplots()
     ax.plot(spot_day.index, spot_day["Close"], label="Historic price")
     ax.plot(spot_forecast.index, spot_forecast["Close"], color="blue", label="Real␣
      ↪price forecast period")
     ax.set_xlabel("Date", fontsize=18)
     ax.set_ylabel("Bitcoin Price", fontsize=18)
     ax.set_title("Price Action BTC/USD February 27th 2022")
     plt.legend()
     plt.show()
```



The day prior to the forecast period shows a relatively volatile price action, and the standard deviation of returns for the historic price is likely higher than for the forecast period which can be seen as the blue line. The forecast period for all of the models will be 60 periods, or 1 hour. This might be too long of a period to predict prices for, as we saw in the simple analysis section that shocks tend to have little to no correlation after about 5 periods. Using forecasts based on data with 1 minute intervals in a practical application real-time market data would likely aim for a lower forecast period, so this is something to keep in mind.

5.1 Univariate forecasting

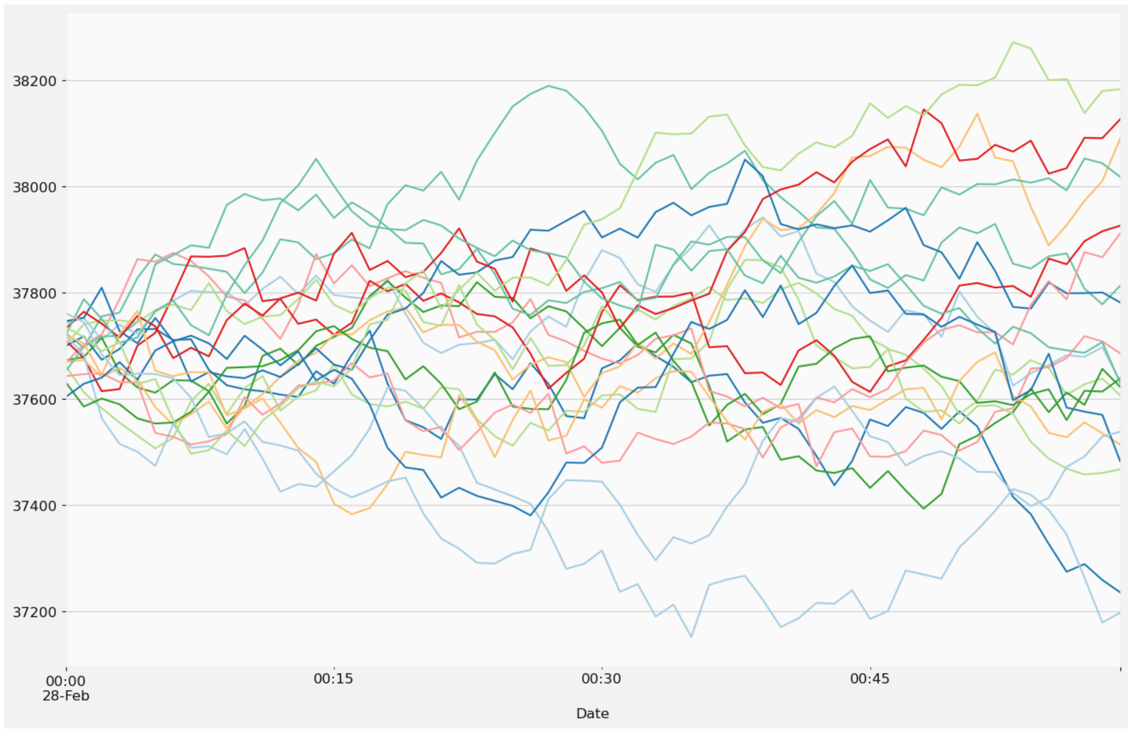
Univariate models are models that only use one variable to forecast, which in this case will be the price of the asset. There are multiple models that can be used to forecast based on this type of data, but we will focus on a random walk as well as ARIMA models.

5.1.1 Random Walk

A Random Walk is a form of autoregressive model where the output shows the cumulative sum of a distribution of returns based on the real mean and standard deviation of the data you want to forecast. The model being autoregressive means that it predicts future values or movements based on past values in the dataset.

[ ]:
```
def random_walk(mean: float, std: float, steps: int, base: float, date: str):
    """
    @notice Generate a random walk
    @dev
    @param  `mean` The mean value for the series
    @param  `std` The standard dev for the series
    @param  `steps` The amount of periods to forecast
    @param  `base` The base number to be added for
            plotting purposes
    @param  `date` The start date of the forecast
    @return A dataframe with all the walks
    """
    dates = []
    C=20
    Y = np.empty(shape=[steps,C])
    for c in range(C):
        e = np.random.normal(mean,std,steps)
        y = np.cumsum(e)
        Y[:,c] = y + base
    for i in range(steps):
        date = pd.to_datetime(date)
        date += dt.timedelta(minutes=1)
        dates.append(date)
    Y = pd.DataFrame(Y)
    Y["Date"] = dates
    Y = Y.set_index("Date")
    return Y
```

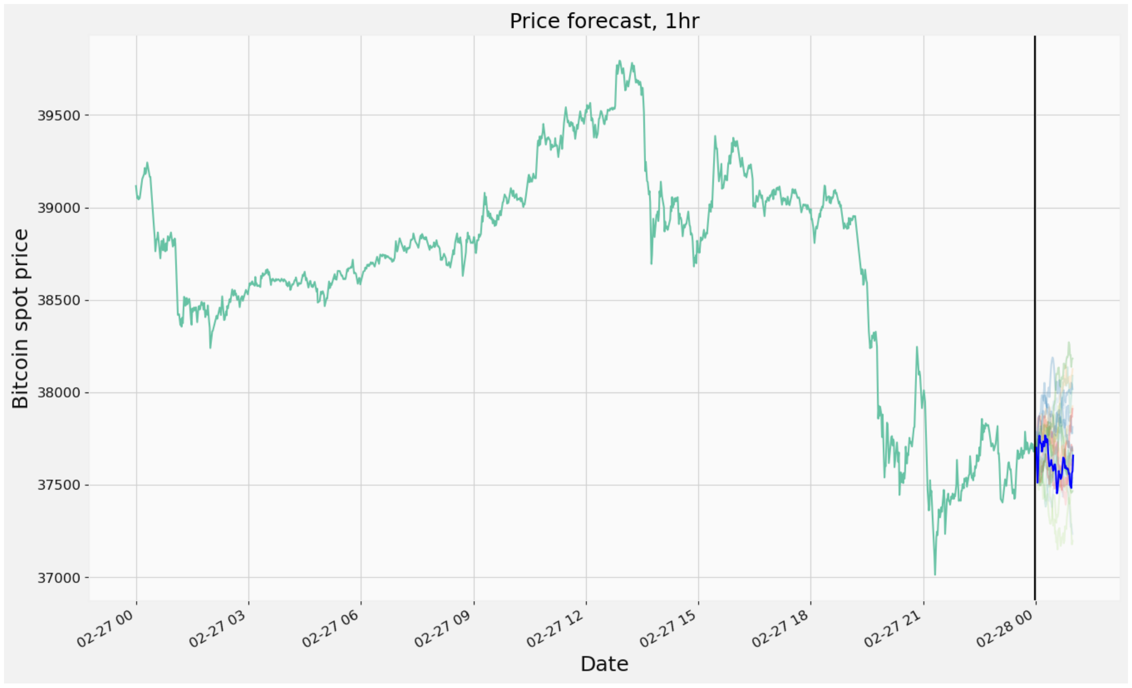
[ ]:
```
# forecast for 1 hour
daily_walk = random_walk(change(spot_day["Close"]).mean(),␣
 ↪change(spot_day["Close"]).std(),
                         60, 37699.07, "2022-02-27 23:59:00")
fig, ax = plt.subplots()
daily_walk.plot(ax=ax)
ax.get_legend().remove()
```



We now have the random walk model output, but as the returns are normally distributed, this plot doesn't tell us much other than that all of the prices within the displayed range are relatively realistic. For the majority of financial assets the historic volatility is lower than the implied volatility, so generally we can expect the actual future price action to not be higher or lower than the model output displayed above. We can append the plots to the original price dataframe for the day to see how the changes look in comparison to both the historic data and the real data for the forecast period.

```
[ ]: fig, ax = plt.subplots()
     ax.plot(spot_day.index, spot_day["Close"])
     daily_walk.plot(ax=ax, alpha=0.25)
     ax.get_legend().remove()
     plt.plot(spot_forecast.index, spot_forecast["Close"], color="blue")
     ax.axvline(spot_day.index[-1], color="black")
     ax.set_xlabel("Date", fontsize=18)
     ax.set_ylabel("Bitcoin spot price", fontsize=18)
     ax.set_title("Price forecast, 1hr", fontsize=18)
```

```
[ ]: Text(0.5, 1.0, 'Price forecast, 1hr')
```



The model highlights the potential price movements of the asset, and as mentioned the actual volatility for the forecast period is lower than the implied volatility in the random walk. However as all of the predicted prices are equally as likely, this way of forecasting time series data is not very good except for specific cases for calculating margin or collateral etc.

Given the spread of the output it seems relatively unlikely that the price is not within the upper and lower bounds of the random walk simulation for the period. If the forecast period started around the local top between 12 and 15pm on the 27th of February, the actual price in the following 60 periods might have ended up below the lowest of the random walk forecasts as the decline in the asset price was unusually steep in this period.

5.1.2 ARIMA

Arima models are used to better understand datasets and predict future trends of the dependent variable based on the strength and weight of the lagged variables of it's previous values.

There are multiple arima functions that can be imported into python, where the most common is probably the statsmodels package, which comes with several functions related to time series regression. By using this package you have to manually input the different AR, I, and MA terms and trying different combinations to find the optimal model. In the R programming language there is a popular package called auto.arima that finds the best suited model for you, and this same functionality is brought to python with the pmdarima package and the auto_arima function. In this paper the model is optimized for the lowest AIC value.

We can start of by using a basic example from the documentation of the package and tweak the parameters to give a better prediction thereafter.

[ ]:
```
# fit stepwise auto-ARIMA
stepwise_fit_day1 = pm.auto_arima(spot_day["Close"], start_p=1, start_q=1,
                             max_p=3, max_q=3, m=12, # m=12 seasonality
                             start_P=0, seasonal=True,
                             d=1, D=1, trace=True, # d=1, non-stationary data
                             error_action='ignore',  # don't want to know if an␣
 ↪order does not work
                             suppress_warnings=True,  # don't want convergence␣
 ↪warnings
                             stepwise=True)  # set to stepwise, not parallell␣
 ↪execution
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=inf, Time=3.55 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=15542.057, Time=0.16 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=15187.760, Time=1.15 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=1.97 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=15542.995, Time=0.24 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=15005.623, Time=2.60 sec
 ARIMA(1,1,0)(2,1,1)[12]             : AIC=inf, Time=8.47 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=inf, Time=3.12 sec
 ARIMA(0,1,0)(2,1,0)[12]             : AIC=15005.203, Time=1.10 sec
 ARIMA(0,1,0)(1,1,0)[12]             : AIC=15186.958, Time=0.58 sec
 ARIMA(0,1,0)(2,1,1)[12]             : AIC=inf, Time=6.46 sec
 ARIMA(0,1,0)(1,1,1)[12]             : AIC=inf, Time=3.53 sec
 ARIMA(0,1,1)(2,1,0)[12]             : AIC=15005.803, Time=2.35 sec
 ARIMA(1,1,1)(2,1,0)[12]             : AIC=15005.344, Time=7.77 sec
 ARIMA(0,1,0)(2,1,0)[12] intercept   : AIC=15007.193, Time=8.86 sec

Best model:  ARIMA(0,1,0)(2,1,0)[12]
Total fit time: 51.905 seconds
```

[ ]:
```
stepwise_fit_day1.summary()
```

[ ]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:                                 y   No. Observations:
1440
Model:             SARIMAX(0, 1, 0)x(2, 1, 0, 12)   Log Likelihood
-7499.601
Date:                           Wed, 21 Dec 2022   AIC
15005.203
Time:                                   21:31:03   BIC
15020.993
```

```
Sample:                              02-27-2022   HQIC
15011.100
                                   - 02-27-2022
Covariance Type:                           opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.S.L12      -0.6363      0.017    -37.595      0.000      -0.670      -0.603
ar.S.L24      -0.3480      0.018    -19.352      0.000      -0.383      -0.313
sigma2      2137.8357     48.864     43.751      0.000    2042.064    2233.607
================================================================================
===
Ljung-Box (L1) (Q):                   1.63   Jarque-Bera (JB):
700.15
Prob(Q):                              0.20   Prob(JB):
0.00
Heteroskedasticity (H):               3.56   Skew:
-0.16
Prob(H) (two-sided):                  0.00   Kurtosis:
6.42
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

The model used is a SARIMAX model which is an extension of the ARIMA class models. The abbreviation stands for Seasonal Auto-Regressive Integrated Moving Average with eXogenous factors, indicating that the model was optimized when it accounted for seasonality found in the dataset.

The optimal model in terms of the lowest possible AIC uses a single Integrated term as the only addition to the seasonal terms, and the seasonal terms consist of 2 autoregressive terms as well as 1 integrated term. Since the data is non-stationary, the integrated terms indicate that the optimal model takes the difference between the periods once and essentially makes the data stationary.

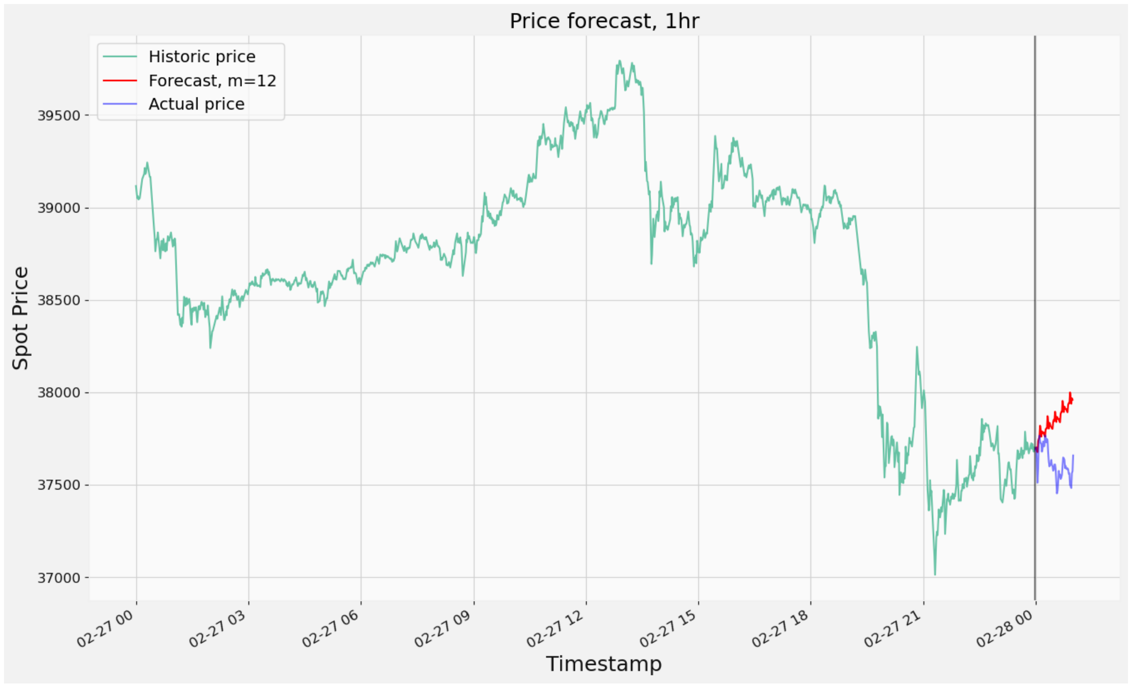
The seasonality argument m in the model is 12, which is commonly used for monthly data with seasonal trends that repeat in the same months every year. Using this value for the seasonal argument on minute-over-minute data is not necessarily better or worse than any other value, but it is a starting point to see how the prediction adapts to the previous price action in relation to the actual price action for the forecast period.

The coefficients for the autoregressive seasonality terms indicate how much the lags in the previous periods affects the model, and the lag for the 12 last periods have a higher coefficient than the preceding 12 periods.

Based on this model we can create a prediction for the forecast period to see how it compares to the actual price in the period.

```
[ ]: ARIMA_pred_day1 = stepwise_fit_day1.predict(n_periods=60)
```

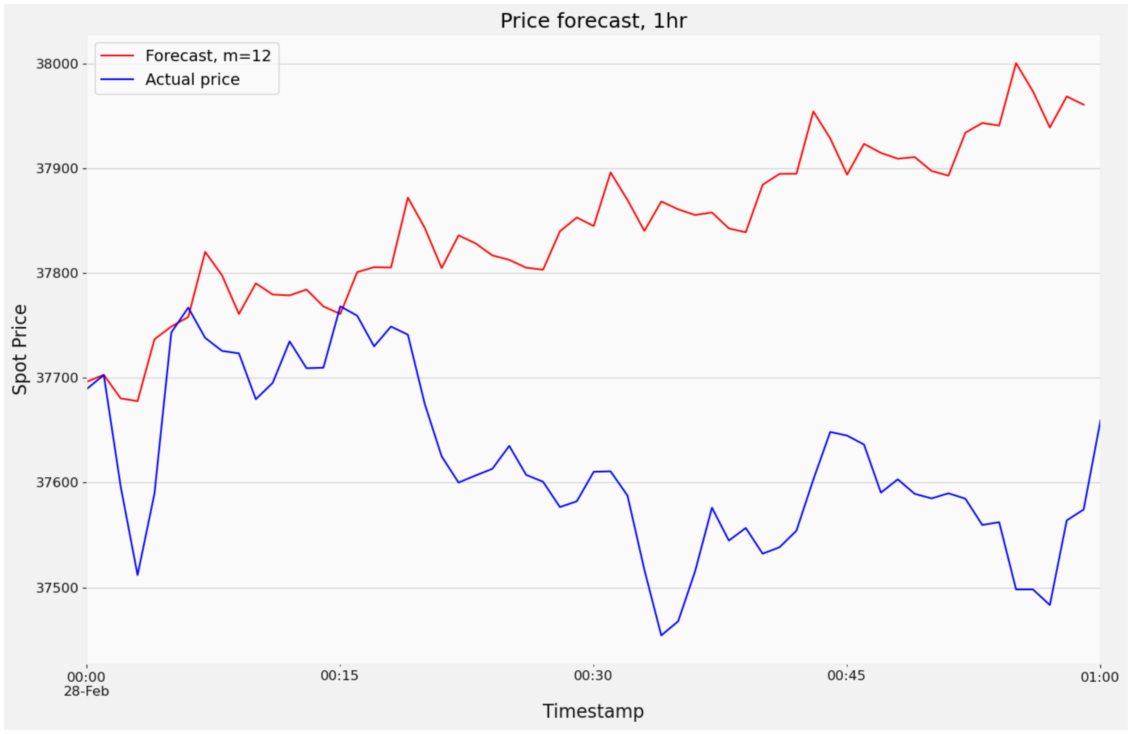
```
[ ]: fig, ax = plt.subplots()
     ax.plot(spot_day.index, spot_day["Close"], label="Historic price")
     ARIMA_pred_day1.plot(ax=ax, color="red", label="Forecast, m=12")
     ax.axvline(spot_day.index[-1], color="black", alpha=0.5)
     ax.plot(spot_forecast.index, spot_forecast["Close"], color="blue", alpha=0.5,␣
      ↪label="Actual price")
     ax.set_ylabel("Spot Price", fontsize=18)
     ax.set_xlabel("Timestamp", fontsize=18)
     ax.set_title("Price forecast, 1hr", fontsize=18)
     plt.legend()
     plt.show()
```



At first glance, the forecast from this model is not very good, and the delta between the forecast and actual value at the end of the forecasting period is quite high. With the seasonal argument m=12, the model adapts to follow the pattern seen in the 12, and 24 candles leading up to the forecasting period (2 seasonal AR terms), and as the forecast is for 60 minutes, this pattern repeats 5 times. It is clear that the model did not do a good job of predicting the price action for the whole forecasting period, however it looks like the delta between the forecast and actual price is relatively low after the first of the five periods in the forecast, which might indicate that the model is able to provide a good forecast in lower timeframes.

If we zoom in and only take a look at the forecast period we can see the differences more clearly:

```
[ ]: fig, ax = plt.subplots()
     ARIMA_pred_day1.plot(ax=ax, color="red", label="Forecast, m=12")
     spot_forecast["Close"].plot(color="blue", label="Actual price")
     ax.set_ylabel("Spot Price", fontsize=16)
     ax.set_xlabel("Timestamp", fontsize=16)
     ax.set_title("Price forecast, 1hr", fontsize=18)
     plt.legend()
     plt.show()
```



The forecast and the actual price was relatively close at the beginning, and the lines crossed 3 times the first 15 minutes, indicating that the price given by the forecast and the actual price was close to the same at these 3 points. This might be a coincidence, but the first of the repeating patterns in the forecast seems to have worked well in this case.

To test if the model is able to predict more accurately for fewer repeating patterns we can forecast for the same timeframe, 1 hour, but increase the seasonality argument m to e.g. 20. By doing this we should get a forecast with a pattern that repeats 3 times.

```
[ ]: # fit stepwise auto-ARIMA
     stepwise_fit_day2 = pm.auto_arima(spot_day["Close"], start_p=1, start_q=1,
                                  max_p=3, max_q=3, m=20,
                                  start_P=0, seasonal=True,
                                  d=1, D=1, trace=True, # d=1, non-stationary data
                                  error_action='ignore',  # don't want to know if an␣
      ↪order does not work
```

```
                                suppress_warnings=True,  # don't want convergence␣
↪warnings
                                stepwise=True)  # set to stepwise, not parallell␣
↪execution
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,1,1)[20]             : AIC=inf, Time=8.29 sec
 ARIMA(0,1,0)(0,1,0)[20]             : AIC=15520.344, Time=0.24 sec
 ARIMA(1,1,0)(1,1,0)[20]             : AIC=15115.801, Time=2.44 sec
 ARIMA(0,1,1)(0,1,1)[20]             : AIC=inf, Time=4.99 sec
 ARIMA(1,1,0)(0,1,0)[20]             : AIC=15520.141, Time=0.49 sec
 ARIMA(1,1,0)(2,1,0)[20]             : AIC=14931.010, Time=5.07 sec
 ARIMA(1,1,0)(2,1,1)[20]             : AIC=inf, Time=20.60 sec
 ARIMA(1,1,0)(1,1,1)[20]             : AIC=inf, Time=7.69 sec
 ARIMA(0,1,0)(2,1,0)[20]             : AIC=14930.783, Time=2.35 sec
 ARIMA(0,1,0)(1,1,0)[20]             : AIC=15115.119, Time=1.40 sec
 ARIMA(0,1,0)(2,1,1)[20]             : AIC=inf, Time=16.14 sec
 ARIMA(0,1,0)(1,1,1)[20]             : AIC=inf, Time=10.62 sec
 ARIMA(0,1,1)(2,1,0)[20]             : AIC=14931.217, Time=5.42 sec
 ARIMA(1,1,1)(2,1,0)[20]             : AIC=14929.906, Time=10.65 sec
 ARIMA(1,1,1)(1,1,0)[20]             : AIC=15117.075, Time=4.54 sec
 ARIMA(1,1,1)(2,1,1)[20]             : AIC=inf, Time=34.08 sec
 ARIMA(1,1,1)(1,1,1)[20]             : AIC=inf, Time=13.55 sec
 ARIMA(2,1,1)(2,1,0)[20]             : AIC=14928.482, Time=15.12 sec
 ARIMA(2,1,1)(1,1,0)[20]             : AIC=15114.302, Time=7.08 sec
 ARIMA(2,1,1)(2,1,1)[20]             : AIC=inf, Time=42.52 sec
 ARIMA(2,1,1)(1,1,1)[20]             : AIC=inf, Time=20.09 sec
 ARIMA(2,1,0)(2,1,0)[20]             : AIC=14926.955, Time=6.53 sec
 ARIMA(2,1,0)(1,1,0)[20]             : AIC=15114.900, Time=2.59 sec
 ARIMA(2,1,0)(2,1,1)[20]             : AIC=inf, Time=25.28 sec
 ARIMA(2,1,0)(1,1,1)[20]             : AIC=inf, Time=10.77 sec
 ARIMA(3,1,0)(2,1,0)[20]             : AIC=14928.490, Time=10.41 sec
 ARIMA(3,1,1)(2,1,0)[20]             : AIC=14930.488, Time=12.87 sec
 ARIMA(2,1,0)(2,1,0)[20] intercept   : AIC=14928.951, Time=17.47 sec

Best model:  ARIMA(2,1,0)(2,1,0)[20]
Total fit time: 319.404 seconds
```

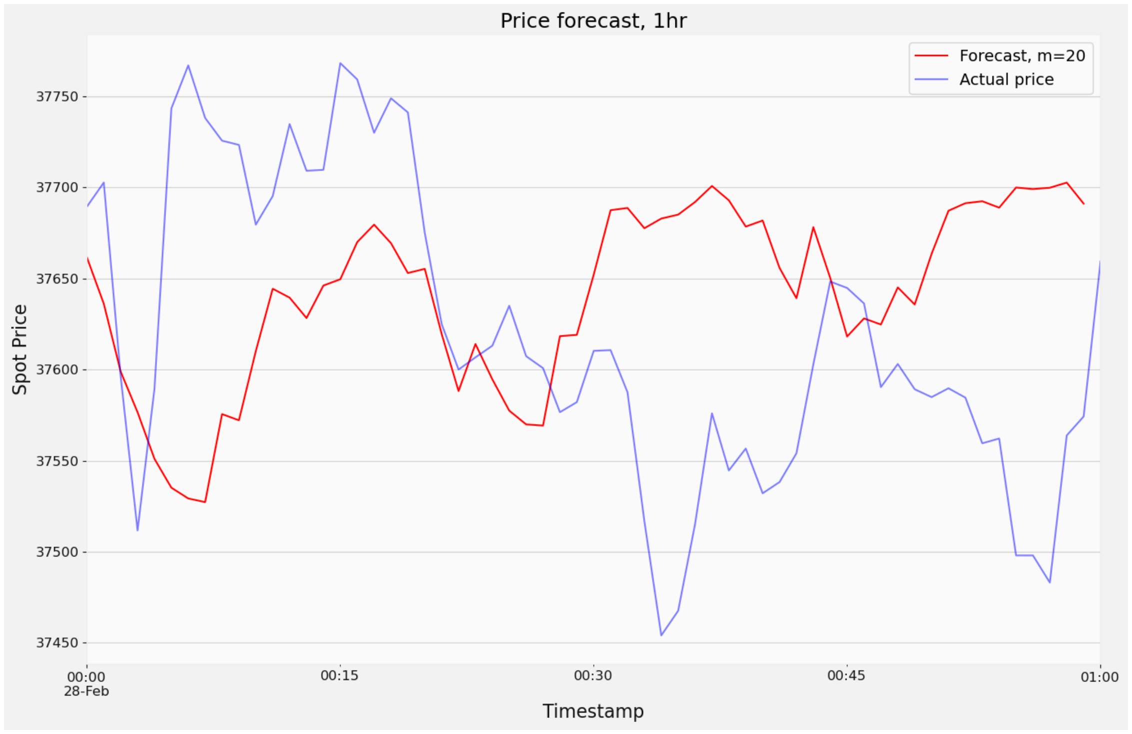
By only changing the seasonality argument from 12 to 20, the optimal model has changed from (0,1,0) to (2,1,0), and the seasonality terms have stayed the same, with 2 autoregressive terms.

```
[ ]: stepwise_fit_day2.summary()
```

```
[ ]: <class 'statsmodels.iolib.summary.Summary'>
     """
                                          SARIMAX Results
     ================================================================================
     ==========
     Dep. Variable:                                  y   No. Observations:
     1440
     Model:             SARIMAX(2, 1, 0)x(2, 1, 0, 20)   Log Likelihood
     -7458.477
     Date:                            Wed, 21 Dec 2022   AIC
     14926.955
     Time:                                    21:36:27   BIC
     14953.243
     Sample:                                02-27-2022   HQIC
     14936.775
                                          - 02-27-2022
     Covariance Type:                              opg
     ==============================================================================
                      coef    std err          z      P>|z|      [0.025      0.975]
     ------------------------------------------------------------------------------
     ar.L1          0.0329      0.019      1.764      0.078      -0.004       0.070
     ar.L2          0.0652      0.018      3.564      0.000       0.029       0.101
     ar.S.L20      -0.6840      0.019    -35.914      0.000      -0.721      -0.647
     ar.S.L40      -0.3609      0.019    -18.619      0.000      -0.399      -0.323
     sigma2      2136.1344     46.070     46.367      0.000    2045.838    2226.431
     ================================================================================
     ===
     Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
     1077.32
     Prob(Q):                              0.93   Prob(JB):
     0.00
     Heteroskedasticity (H):               3.95   Skew:
     -0.35
     Prob(H) (two-sided):                  0.00   Kurtosis:
     7.21
     ================================================================================
     ===

     Warnings:
     [1] Covariance matrix calculated using the outer product of gradients (complex-
     step).
     """
```

In this model summary we can see that both of the autoregressive seasonality coefficients are even lower than the previous model, and the newly introduced AR terms have slight positive coefficients. Something that is really interesting with the autoregressive terms is that the second autoregressive term has a higher coefficient than the first, and the first one has a P-value of 0.078. Time series regression models have a tendency to give low P-values even if the relationship between the dependent and independent variable is questionable, so in this case there might be arguments for not using the first AR term.

```
[ ]: ARIMA_pred_day2 = stepwise_fit_day2.predict(n_periods=60)
[ ]: fig, ax = plt.subplots()
     ARIMA_pred_day2.plot(ax=ax, color="red", label="Forecast, m=20")
     spot_forecast["Close"].plot(color="blue", alpha=0.5, label="Actual price")
     ax.set_ylabel("Spot Price", fontsize=16)
     ax.set_xlabel("Timestamp", fontsize=16)
     ax.set_title("Price forecast, 1hr", fontsize=18)
     plt.legend()
     plt.show()
```



Increasing the seasonality scope fits the current forecast significantly better than the previous forecast. Here the seasonality is set to 20 periods, and the pattern repeats 3 times. By the end of the forecast period, the forecasted price and the actual price seems relatively close, however this might be a coincidence as the forecast is a repeating pattern. Similarly to the previous forecast, the delta between the predicted price action and the actual price action after the first pattern (at 00:20) seems to be low. This is a good sign, but to confidently say that the model is able to predict the price well, this process would have to be repeated several times and an algorithm for scoring the prediction would be needed, which could take a significant amount of time depending on how many iterations one would like to run.

Unfortunately I'm not able to run the regression with an even higher seasonality argument due to memory error. This prevents me from trying to predict the whole hour without a recurring pattern in the predicted price action.

We can also run the model on a stationary version of the series, where the percentage price change is the univariate value. First, we can test for stationarity bu using the Augmented Dickey-Fuller test.

```
[ ]: adf_test= adfuller(spot_day["Percentage"].dropna(), autolag="AIC")
     adf_test
```

```
[ ]: (-10.327341382013193,
      2.902555465284444e-18,
      12,
      1427,
      {'1%': -3.4349408214067227,
       '5%': -2.8635675309927153,
       '10%': -2.5678494453155656},
      -2403.7315250361476)
```

This test returns a tuple containing the values of the test.

The first number is the estimated ADF test statistic, the second is the estimated p value, the third is the chosen number of lags, the fourth is the number of observations, the fifth element is a mapping, or dictionary as it's called in python, containing the critical values for 1%, 5%, and 10%, and the last element is the miximized information criterion id if the autolag is None.

The null hypothesis of the test is that the series is non-stationary, and with a p-value of 2.9*10-18, we can confidently say that the series is stationary.

Now let's take a look at the optimal model for the stationary series when the Integrated term is explicitly set to 0, via the function argument *d=0*

```
[ ]: # fit stepwise auto-ARIMA
     stepwise_fit_day_i = pm.auto_arima(spot_day["Percentage"].dropna(), start_p=1,␣
      ↪start_q=1,
                                  max_p=3, max_q=3, m=12,
                                  start_P=0, seasonal=True,
                                  d=0, D=1, trace=True, #d=0, series is stationary
                                  error_action='ignore',  # don't want to know if an␣
      ↪order does not work
                                  suppress_warnings=True,  # don't want convergence␣
      ↪warnings
                                  stepwise=True)  # set to stepwise, not parallell␣
      ↪execution
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,1,1)[12] intercept   : AIC=inf, Time=14.73 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=-1440.547, Time=1.11 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=-1796.549, Time=3.75 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=inf, Time=11.63 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=-1442.546, Time=0.33 sec
 ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=-1439.456, Time=0.75 sec
 ARIMA(1,0,0)(2,1,0)[12] intercept   : AIC=-1979.329, Time=12.80 sec
 ARIMA(1,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=27.68 sec
 ARIMA(1,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=13.22 sec
 ARIMA(0,0,0)(2,1,0)[12] intercept   : AIC=-1979.907, Time=6.98 sec
 ARIMA(0,0,0)(1,1,0)[12] intercept   : AIC=-1797.527, Time=2.07 sec
 ARIMA(0,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=28.35 sec
 ARIMA(0,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=10.07 sec
 ARIMA(0,0,1)(2,1,0)[12] intercept   : AIC=-1979.169, Time=7.34 sec
 ARIMA(1,0,1)(2,1,0)[12] intercept   : AIC=-1979.450, Time=18.00 sec
 ARIMA(0,0,0)(2,1,0)[12]             : AIC=-1981.899, Time=1.45 sec
 ARIMA(0,0,0)(1,1,0)[12]             : AIC=-1799.527, Time=0.58 sec
 ARIMA(0,0,0)(2,1,1)[12]             : AIC=inf, Time=15.37 sec
 ARIMA(0,0,0)(1,1,1)[12]             : AIC=inf, Time=6.91 sec
 ARIMA(1,0,0)(2,1,0)[12]             : AIC=-1981.322, Time=2.98 sec
 ARIMA(0,0,1)(2,1,0)[12]             : AIC=-1981.162, Time=2.49 sec
 ARIMA(1,0,1)(2,1,0)[12]             : AIC=-1981.446, Time=7.36 sec

Best model:  ARIMA(0,0,0)(2,1,0)[12]
Total fit time: 195.978 seconds
```

As the data used in this regression is already stationary, the Integrated term is 0, and this model is essentially the same as the first one with the same seasonality term *m=12*

```
[ ]: stepwise_fit_day_i.summary()
```

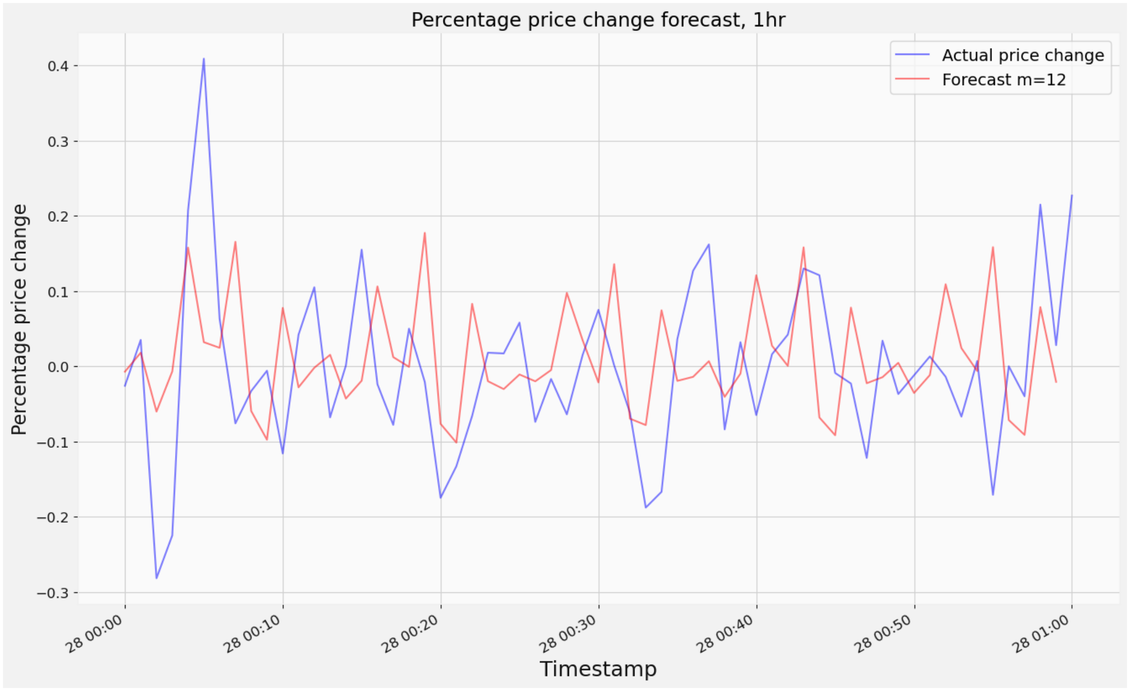
```
[ ]: <class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
==================================================================================
Dep. Variable:                           y   No. Observations:                 1440
Model:                SARIMAX(2, 1, 0, 12)   Log Likelihood                 993.950
Date:                     Wed, 21 Dec 2022   AIC                          -1981.899
Time:                             21:39:46   BIC                          -1966.107
Sample:                         02-27-2022   HQIC                         -1976.002
                              - 02-27-2022
Covariance Type:                       opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.S.L12      -0.6376      0.017    -37.920      0.000      -0.671      -0.605
ar.S.L24      -0.3493      0.018    -19.699      0.000      -0.384      -0.315
sigma2         0.0145      0.000     44.399      0.000       0.014       0.015
==================================================================================
===
Ljung-Box (L1) (Q):                   1.42   Jarque-Bera (JB):
777.69
Prob(Q):                              0.23   Prob(JB):
0.00
Heteroskedasticity (H):               3.77   Skew:
-0.16
Prob(H) (two-sided):                  0.00   Kurtosis:
```

```
6.60
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

This model's coefficients are also very similar to the non-stationary model with the same seasonal term, which is not surprising as it should essentially be the same model, however represented differently which can be seen in the plot below.

[ ]:
```python
ARIMA_pred_day_i = stepwise_fit_day_i.predict(n_periods=60)
```

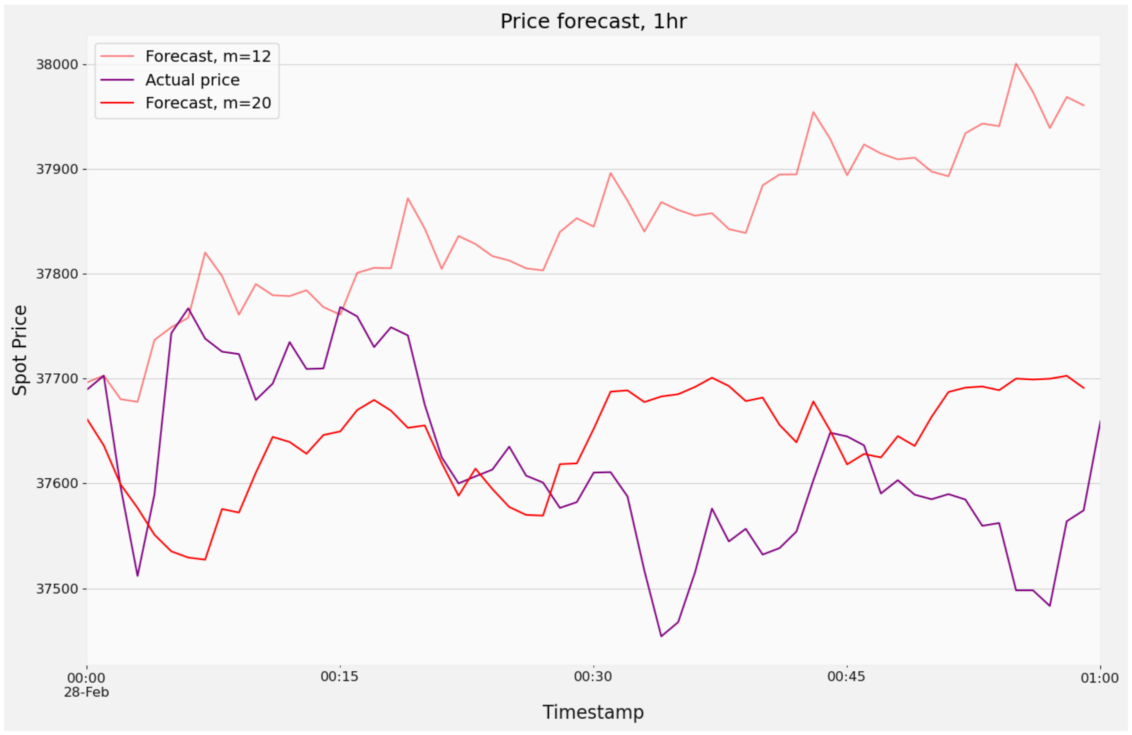
[ ]:
```python
fig, ax = plt.subplots()
ax.plot(spot_forecast.index, spot_forecast["Percentage"], color="blue",
            alpha=0.5, label="Actual price change")
ARIMA_pred_day_i.plot(ax=ax, color="red", alpha=0.5, label="Forecast m=12")
ax.set_ylabel("Percentage price change", fontsize=16)
ax.set_xlabel("Timestamp", fontsize=18)
ax.set_title("Percentage price change forecast, 1hr")
plt.legend()
plt.show()
```



The stationary forecast might look like it fits the actual price for the period better than the non-stationary forecast with the same seasonality terms, but if we add the returns together by using the cumulative sum of the series, it is close to the exact same forecast, as can be seen in the plot below. As the plot above looks better than the plot below, stationary forecasts can be somewhat deceiving for long term forecasts unless you apply a scoring algorithm that looks at the squared difference between the two plots.

What this forecast does show well is the range of expected returns, where both the forecast and actual price seems to have roughly the same variance of returns between 0.2 and -0.1%, indicating that this might serve well as a forecast for predicted risk. In a sense this can be seen as a random walk model with better estimations based on recent data in terms of risk prediction.

[ ]:
```
fig, ax = plt.subplots()
ARIMA_pred_day1.plot(ax=ax, color="red", alpha=0.5, label="Forecast, m=12")
spot_forecast["Close"].plot(color="purple", label="Actual price")
ARIMA_pred_day2.plot(ax=ax, color="red", label="Forecast, m=20")
ax.set_ylabel("Spot Price", fontsize=16)
ax.set_xlabel("Timestamp", fontsize=16)
ax.set_title("Price forecast, 1hr", fontsize=18)
plt.legend()
plt.show()
```



The two forecasts with different seasonality terms shows different results, where the forecast based on the longest previous period, 20, seems to perform better than the shorter term forecast in the long term, however the forecast based on the shorter period is able to better predict the short term move of the asset price.

5.2 Forecasting with exogenous variables

Exogenous variables are variables whose value is determined outside of the model, and are typically used as weighted input for univariate time series analysis.

As the goal in this case is to model and forecast the change in the price of an asset that is highly speculative and does not depend much on other factors (e.g. cash flow as discussed in the introduction), the exogenous variables used here are products of the previous price changes in the asset itself. There are close to an endless amount of different technical indicators, but let's focus on some of the more well known ones, the RSI and MACD.

The RSI, or the Relative Strength Index, is a momentum indicator that measures the speed and magnitude of a security's recent price changes. The values for the RSI is between 0 and 100, with a low value indicating the asset being oversold, and a high value indicating the asset being overbought, which generally are seen as bullish and bearish respectively. The values where the asset is considered either oversold or overbought varies, but below 30 and above 70 are commonly used. Values close to 0 and 100 are not common, and most assets move between the value of 30 and 70 for the majority of the time.

The MACD, or the Moving Average Convergence-Divergence, is a technical indicator used to show how the short and long term exponential moving averages (EMAs) are moving relative to each other. The most commonly used periods for these EMAs are 12 and 26, which is also what is used for the forecasts below. The values of the macd depends on if it is standardized or used as just the difference between the two. In the forecasts below, the difference between the two EMAs will be used.

```
[ ]: def get_macd(price: pd.Series, slow: int, fast: int, smooth: int):
         """
         @notice Find the MACD for the price action of an asset
         @dev    Difference between moving averages
         @param  `price` A series with the price of the asset
         @param  `slow` The longer period MA
         @param  `fast` The shorter period MA
         @param  `smooth` The smooth factor of the calculations
         @return A dataframe with the MACD, signal, and hist
         """
         exp1 = price.ewm(span = fast, adjust = False).mean()
         exp2 = price.ewm(span = slow, adjust = False).mean()
         macd = pd.DataFrame(exp1 - exp2).rename(columns = {'close':'macd'})
         signal = pd.DataFrame(macd.ewm(span = smooth, adjust = False).mean()).
      ↪rename(columns = {'macd':'signal'})
         hist = pd.DataFrame(macd['macd'] - signal['signal']).rename(columns = {0:
      ↪'hist'})
         frames = [macd, signal, hist]
         df = pd.concat(frames, join = 'inner', axis = 1)
         return df
```

To get all of the values needed we can create a new dataframe and use the functions to add the MACD and RSI values to each of the entries in the dataframe.

```
[ ]: df = pd.DataFrame({"Date": spot_day.index, "close": spot_day["Close"]})

     n = 14
     def rma(x, n, y0):
         a = (n-1) / n
         ak = a**np.arange(len(x)-1, -1, -1)
         return np.r_[np.full(n, np.nan), y0, np.cumsum(ak * x) / ak / n + y0*a**np.
      ↪arange(1, len(x)+1)]
     df['change'] = df['close'].diff()
     df['gain'] = df.change.mask(df.change < 0, 0.0)
     df['loss'] = -df.change.mask(df.change > 0, -0.0)
     df['avg_gain'] = rma(df.gain[n+1:].to_numpy(), n, np.nansum(df.gain.to_numpy()[:
      ↪n+1])/n)
     df['avg_loss'] = rma(df.loss[n+1:].to_numpy(), n, np.nansum(df.loss.to_numpy()[:
      ↪n+1])/n)
     df['rs'] = df.avg_gain / df.avg_loss
     df['rsi_14'] = 100 - (100 / (1 + df.rs))
     df = df.drop(labels=["Date", "avg_gain", "avg_loss", "rs",
                          "loss", "gain"], axis="columns")
     df.tail()
```

```
[ ]:                         close  change     rsi_14
     Date
     2022-02-27 23:55:00  37720.92   25.69  56.145255
     2022-02-27 23:56:00  37694.23  -26.69  52.439824
     2022-02-27 23:57:00  37679.94  -14.29  50.517460
     2022-02-27 23:58:00  37703.78   23.84  53.575085
     2022-02-27 23:59:00  37699.07   -4.71  52.879865
```

```
[ ]: macd = get_macd(df["close"], 26, 12, 9)
     macd = macd.drop(labels=["signal", "hist"], axis=1)
     macd.tail()
```

```
[ ]:                           macd
     Date
     2022-02-27 23:55:00  24.929540
     2022-02-27 23:56:00  22.537432
     2022-02-27 23:57:00  19.266493
     2022-02-27 23:58:00  18.385997
     2022-02-27 23:59:00  17.110896
```

```
[ ]: df = df.join(macd, on="Date")
     # remove NaN values in the rsi
     df = df[20:]
     df.head()
```

```
[ ]:                          close  change     rsi_14       macd
     Date
     2022-02-27 00:20:00  39186.44  -33.17  55.226424  29.924769
     2022-02-27 00:21:00  39180.65   -5.79  54.206360  27.345507
     2022-02-27 00:22:00  39165.40  -15.25  51.507816  23.796564
     2022-02-27 00:23:00  39165.71    0.31  51.560606  20.769597
     2022-02-27 00:24:00  39125.51  -40.20  44.756270  14.954513
```

Now we have both the price and the exogenous variables, and we can create the model and calculate the predicted price based on the values we pass to the exogenous variables. To find out the relevant range of values for the variables we can print the maximum and minimum values recorded in the period, and base our forecast on a development in these variables.

```
[ ]: print(df["rsi_14"].max())
     print(df["rsi_14"].min())
     print(df["macd"].max())
     print(df["macd"].min())
```

```
83.30589230680397
10.646288499543175
126.62968586453644
-216.04318074612092
```

I was not able to get pmdarima's auto_arima to work well when adding exogenous variables, so the forecasts will be based on statstool's ARIMA function. To find the optimal model parameters we can perform an auto_arima function on the data without exogenous variables and without the seasonality argument. What we do have to keep in mind however is that this model might not be the one with the best fit when using exogenous variables with the forecast, so there is some room for error if we were to trust this model and the forecast it creates. For the purpose of these forecasts, the seasonal argument was also dropped, so the forecasts will mainly be directional with a confidence interval.

```
[ ]: # fit stepwise auto-ARIMA
     optimal_arima = pm.auto_arima(spot_day["Close"], start_p=1, start_q=1,
                                  max_p=3, max_q=3,
                                  start_P=0, seasonal=True,
                                  d=1, D=1, trace=True,
                                  error_action='ignore',  # don't want to know if an␣
      ↪order does not work
                                  suppress_warnings=True,  # don't want convergence␣
      ↪warnings
                                  stepwise=True)  # set to stepwise, not parallell␣
      ↪execution
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=14685.091, Time=2.44 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=14685.488, Time=0.23 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=14685.602, Time=0.28 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=14685.815, Time=0.56 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=14684.372, Time=0.09 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 3.599 seconds
```

The optimal model for the non-stationary series without using seasonality and exogenous variables is ARIMA(0,1,0).

[ ]:
```
exog = np.column_stack([df["rsi_14"], df["macd"]])
arima_exo = ARIMA(df["close"], exog=exog, order=(0,1,0)).fit()
arima_exo.summary()
```

```
C:\Users\stian\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency T will be used.
  self._init_dates(dates, freq)
C:\Users\stian\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency T will be used.
  self._init_dates(dates, freq)
C:\Users\stian\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency T will be used.
  self._init_dates(dates, freq)
```

[ ]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                  close   No. Observations:                 1420
Model:                 ARIMA(0, 1, 0)   Log Likelihood               -6143.762
Date:                Wed, 21 Dec 2022   AIC                          12293.524
Time:                        21:51:00   BIC                          12309.297
Sample:                    02-27-2022   HQIC                         12299.416
                         - 02-27-2022
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1             6.2953      0.087     72.070      0.000       6.124       6.466
x2             2.2768      0.057     39.976      0.000       2.165       2.388
sigma2       337.4511      6.926     48.722      0.000     323.876     351.026
================================================================================
===
Ljung-Box (L1) (Q):                  20.44   Jarque-Bera (JB):
8026.61
Prob(Q):                              0.00   Prob(JB):
0.00
```

```
Heteroskedasticity (H):                2.74   Skew:
-1.62
Prob(H) (two-sided):                   0.00   Kurtosis:
14.19
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

The main points of interest in this model are the coefficients for the exogenous variables. x1 shows the predicted change in the price based on the RSI value, and that is 6.3. As we saw from the maximum and minimum recorded values, the RSI was between 10 and 83 in the period, and if it were to drop quickly from the max to the min value, the price change is predicted to be about 170 USD. x2 is the MACD coefficient, showing a value of 2.28. As the range of the recorded MACD values is significantly greater than the RSI values, it is somewhat expected that the coefficient also is lower. Even though the coefficient for the MACD is lower than the RSI, the model estimates that the price change based on the MACD is greater than that of the RSI, as the range for the MACD values is more than 4.5 times the RSI values, and the coefficient for the RSI is "only" about 2.7 times the MACD coefficient. We can now create scenarios for the values of the exogenous variables and predict the price change of Bitcoin in the spot market.

```
[ ]: # The last values, where the historical df ended and the forecast starts
     print(df["rsi_14"][-1])
     print(df["macd"][-1])
```

```
52.8798653003075
17.110895559912024
```

We can start by creating a very bullish forecast where both the MACD and RSI values increase from their current levels up to the maximum recorded values in the period.

```
[ ]: high_rsi = np.linspace(52.8, 83, 60)
     high_macd = np.linspace(17, 126, 60)
     exog = np.column_stack([high_rsi, high_macd])
     exo_forecast_high = arima_exo.get_forecast(exog=exog, steps=60).summary_frame()
```

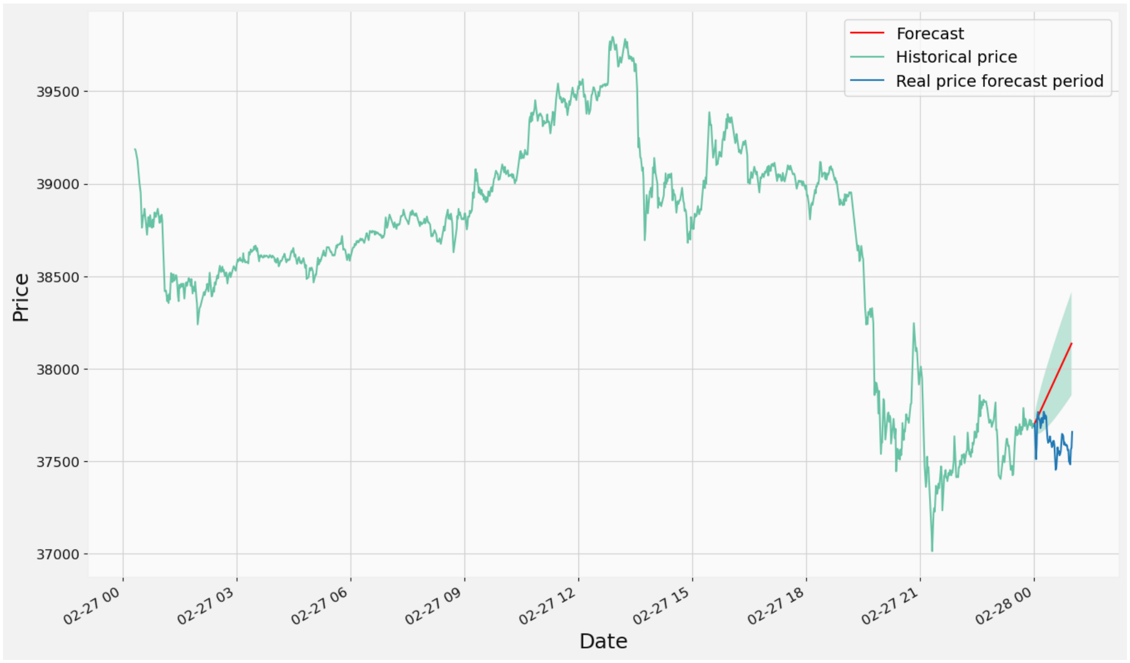
```
[ ]: exo_forecast_high = exo_forecast_high.reset_index()
     exo_forecast_high = exo_forecast_high.rename(columns = {"index":"Date"})
     exo_forecast_high.head()
```

```
[ ]: close                 Date          mean    mean_se  mean_ci_lower  \
     0     2022-02-28 00:00:00  37698.314742  18.369841   37662.310515
     1     2022-02-28 00:01:00  37705.743307  25.978879   37654.825640
     2     2022-02-28 00:02:00  37713.171872  31.817499   37650.810721
     3     2022-02-28 00:03:00  37720.600437  36.739683   37648.591982
     4     2022-02-28 00:04:00  37728.029002  41.076214   37647.521102
```

```
close   mean_ci_upper
0        37734.318970
1        37756.660974
2        37775.533023
3        37792.608892
4        37808.536902
```

[ ]:
```
fig, ax = plt.subplots()
ax.plot(exo_forecast_high["Date"], exo_forecast_high["mean"], color="red",␣
 ↪label="Forecast")
ax.fill_between(exo_forecast_high["Date"], exo_forecast_high["mean_ci_lower"],
                exo_forecast_high["mean_ci_upper"], alpha=.4)
ax.set_xlabel("Date", fontsize=18)
ax.set_ylabel("Price", fontsize=18)
df["close"].plot(ax=ax, label="Historical price")
ax.plot(spot_forecast.index, spot_forecast["Close"], label="Real price forecast␣
 ↪period")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x24825ed63d0>
```



We can see that the forecast misses the actual price significantly, and even the confidence interval is not close to the actual price at the end of the forecast. As mentioned above the plot, this is close to the most bullish scenario we could have expected given the two exogenous variables, and if we compare the forecast to the historical data, the upper confidence interval seems to have close to the same slope as the steepest price increases in the period, indicating that this likely is a decent forecast model given realistic values to the exogenous variables. Finding these realistic values is not easy, and we'll discuss that a bit more after another forecast of a relatively neutral scenario.

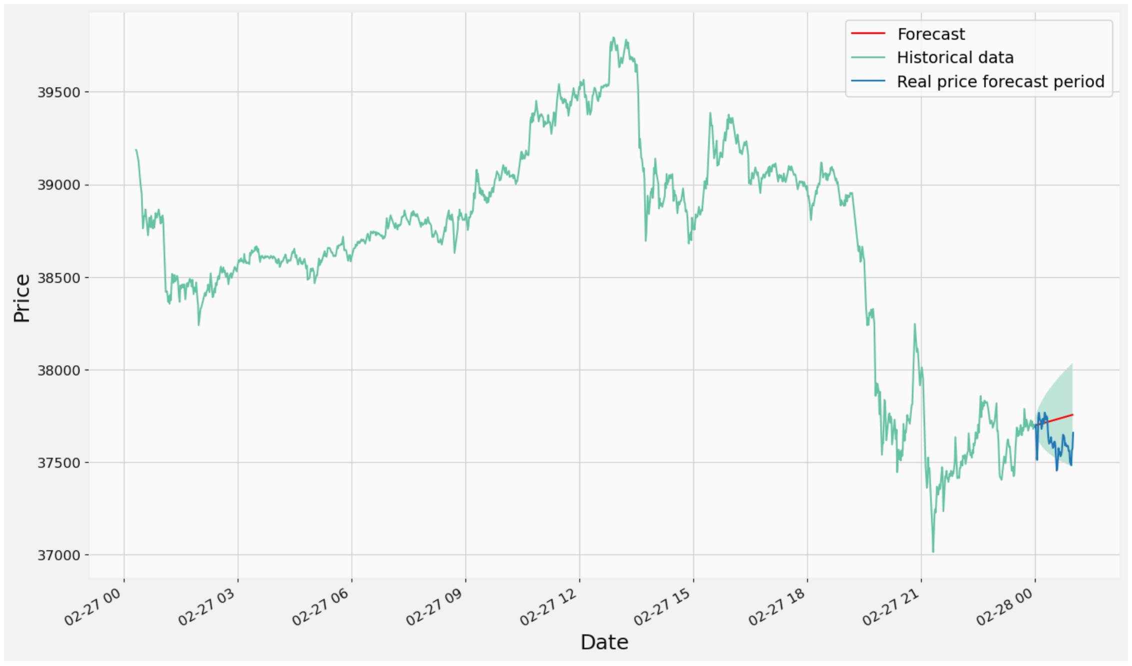
```
[ ]: low_rsi = np.linspace(52.8, 50, 60)
     low_macd = np.linspace(17, 50, 60)
     exog = np.column_stack([low_rsi, low_macd])
     exo_forecast_low = arima_exo.get_forecast(exog=exog, steps=60).summary_frame()
```

```
[ ]: exo_forecast_low = exo_forecast_low.reset_index()
     exo_forecast_low = exo_forecast_low.rename(columns = {"index":"Date"})
     exo_forecast_low.head()
```

```
[ ]: close                  Date          mean    mean_se  mean_ci_lower  \
     0     2022-02-28 00:00:00  37698.314742  18.369841   37662.310515
     1     2022-02-28 00:01:00  37699.289432  25.978879   37648.371765
     2     2022-02-28 00:02:00  37700.264121  31.817499   37637.902970
     3     2022-02-28 00:03:00  37701.238811  36.739683   37629.230356
     4     2022-02-28 00:04:00  37702.213500  41.076214   37621.705600

     close  mean_ci_upper
     0        37734.318970
     1        37750.207098
     2        37762.625272
     3        37773.247266
     4        37782.721400
```

```
[ ]: fig, ax = plt.subplots()
     ax.plot(exo_forecast_low["Date"], exo_forecast_low["mean"], color="red",␣
      ↪label="Forecast")
     ax.fill_between(exo_forecast_low["Date"], exo_forecast_low["mean_ci_lower"],
                     exo_forecast_low["mean_ci_upper"], alpha=.4)
     ax.set_xlabel("Date", fontsize=18)
     ax.set_ylabel("Price", fontsize=18)
     df["close"].plot(ax=ax, label="Historical data")
     ax.plot(spot_forecast.index, spot_forecast["Close"], label="Real price forecast␣
      ↪period")
     plt.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x24825f5dd60>
```



Similarly to the bullish forecast, this forecast is also positive, although slightly, and with a relatively narrow confidence interval compared to some of the rapid price changes in the historical data. Even though the forecasted price does not predict the actual price at the end of the forecast period, it is well within the confidence interval, and the forecast seems to be relatively accurate early on when the recent actual price changes have not changed the RSI and MACD values much from the values at the beginning of the forecast period. Additionally, the confidence interval is significantly more narrow compared to the spread of the random walks we simulated earlier.

If we go back to the criticism mentioned after the bullish plot, we'll realize that none of these two plots are possible to use on real data to forecast prices. The dataframe used for the model had the price, MACD, and RSI values for the same periods in every entry. This means that the RSI and MACD values used to predict the price was changed by the price in the same period, and we encounter a problem about causality for the exogenous variables, which again are used to predict the price in the next period. These models use the RSI and MACD from period t to estimate the price in the same period, but the RSI and MACD values are products of the price in the previous periods as well as period t. In reality we don't know the MACD and RSI values for the next period as they are based on the price, which means we have to use the MACD and RSI from the period $t - 1$ to estimate the price in period t.

With this in mind we can add a 1 period lag to the exogenous variables, and use the lagged variables for the model instead of those derived from the price in the same period as the previous model did.

```
[ ]: df["rsi_l1"] = df["rsi_14"].shift()
     df["macd_l1"] = df["macd"].shift()
     df.head()
```

[ ]:
```
                         close  change     rsi_14       macd     rsi_l1  \
Date
2022-02-27 00:20:00  39186.44  -33.17  55.226424  29.924769        NaN
2022-02-27 00:21:00  39180.65   -5.79  54.206360  27.345507  55.226424
2022-02-27 00:22:00  39165.40  -15.25  51.507816  23.796564  54.206360
2022-02-27 00:23:00  39165.71    0.31  51.560606  20.769597  51.507816
2022-02-27 00:24:00  39125.51  -40.20  44.756270  14.954513  51.560606

                       macd_l1
Date
2022-02-27 00:20:00        NaN
2022-02-27 00:21:00  29.924769
2022-02-27 00:22:00  27.345507
2022-02-27 00:23:00  23.796564
2022-02-27 00:24:00  20.769597
```

[ ]:
```
exog = np.column_stack([df["rsi_l1"].dropna(), df["macd_l1"].dropna()])
arima_exo = ARIMA(df["close"][1:], exog=exog, order=(0,1,0)).fit()
arima_exo.summary()
```

```
C:\Users\stian\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency T will be used.
  self._init_dates(dates, freq)
C:\Users\stian\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency T will be used.
  self._init_dates(dates, freq)
C:\Users\stian\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency T will be used.
  self._init_dates(dates, freq)
```

[ ]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                  close   No. Observations:                 1419
Model:                 ARIMA(0, 1, 0)   Log Likelihood               -7237.575
Date:                Wed, 21 Dec 2022   AIC                          14481.150
Time:                        21:51:04   BIC                          14496.921
Sample:                    02-27-2022   HQIC                         14487.042
                         - 02-27-2022
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.0585      0.272     -0.215      0.830      -0.592       0.475
```

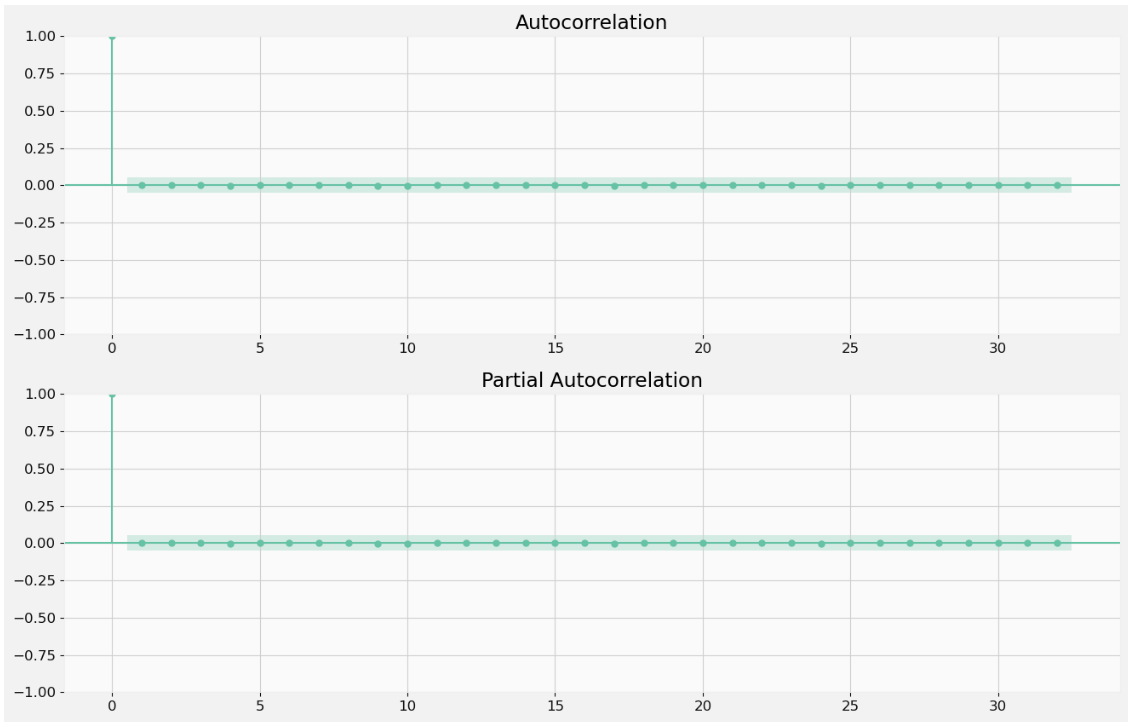
```
x2             0.4025      0.143      2.820      0.005       0.123       0.682
sigma2      1588.6025     31.225     50.877      0.000    1527.403    1649.802
================================================================================
===
Ljung-Box (L1) (Q):                   0.11   Jarque-Bera (JB):
1939.09
Prob(Q):                              0.74   Prob(JB):
0.00
Heteroskedasticity (H):               3.70   Skew:
-0.71
Prob(H) (two-sided):                  0.00   Kurtosis:
8.55
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

With the new model the coefficients for the exogenous variables are significantly different, and they have larger confidence intervals indicating less accurate prediction. To compare with the previous model, the RSI value was 6.3, and is now slightly negative at -0.06. The P value for this coefficient indicates that it is only about 17% certain that the coefficient is not 0, and it is therefore not close to being significant. The MACD coefficient is still significant, but the value has dropped a lot compared to the previous model, which was at 2.28. It is already clear that the forecasts based on this model will be less directional than the previous model as well as showing less certainty about realistic values, given by the confidence interval.

By checking the autocorrelation and partial autocorrelation of the model we can determine whether a shock shows correlation with residuals in later periods.

[ ]:
```
resids = arima_exo.resid
```

[ ]:
```
fig, ax = plt.subplots(2,1)
plot_acf(resids, ax=ax[0])
plot_pacf(resids, ax=ax[1])
plt.show()
```



The plots show no correlation, and this indicates that the model residuals are random. The residual from a price shock in period t does not show a correlation that persists over multiple periods, whether it be $t + 1$, $t + 2$, or $t + 10$.

Now we can use the same scenarios for the forecasts as with the previous model, first one that is very bullish and one that is relatively neutral.

```
[ ]: high_rsi = np.linspace(52.8, 83, 60)
     high_macd = np.linspace(17, 126, 60)
     exog = np.column_stack([high_rsi, high_macd])
     exo_forecast_high = arima_exo.get_forecast(exog=exog, steps=60).summary_frame()
```

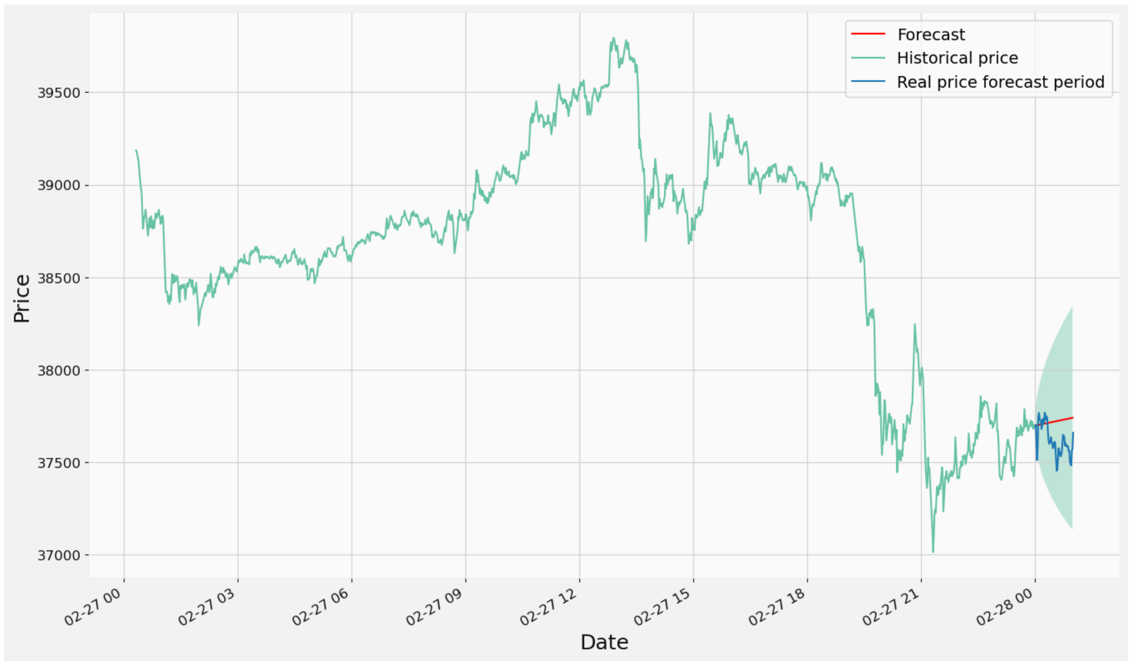
```
[ ]: exo_forecast_high = exo_forecast_high.reset_index()
     exo_forecast_high = exo_forecast_high.rename(columns = {"index":"Date"})
     exo_forecast_high.head()
```

```
[ ]: close                 Date          mean    mean_se  mean_ci_lower  \
     0     2022-02-28 00:00:00  37698.557484  39.857276   37620.438657
     1     2022-02-28 00:01:00  37699.271130  56.366701   37588.794426
     2     2022-02-28 00:02:00  37699.984775  69.034828   37564.678999
     3     2022-02-28 00:03:00  37700.698421  79.714553   37544.460768
     4     2022-02-28 00:04:00  37701.412067  89.123580   37526.733061

     close  mean_ci_upper
     0       37776.676310
```

```
1        37809.747833
2        37835.290552
3        37856.936074
4        37876.091073
```

[ ]:
```
fig, ax = plt.subplots()
ax.plot(exo_forecast_high["Date"], exo_forecast_high["mean"], color="red",␣
↪label="Forecast")
ax.fill_between(exo_forecast_high["Date"], exo_forecast_high["mean_ci_lower"],
                exo_forecast_high["mean_ci_upper"], alpha=.4)
ax.set_xlabel("Date", fontsize=18)
ax.set_ylabel("Price", fontsize=18)
df["close"].plot(ax=ax, label="Historical price")
ax.plot(spot_forecast.index, spot_forecast["Close"], label="Real price forecast␣
↪period")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x248260f6fa0>
```



By using this model, the most bullish forecast looks more like the neutral forecast from the last model, but with a significantly larger confidence interval. The spread of the confidence interval looks more like the spread of the random walks presented earlier, and this points at the model not being very good at predicting the price in the future.

Let's also take a look at how the neutral forecast looks when using this model.

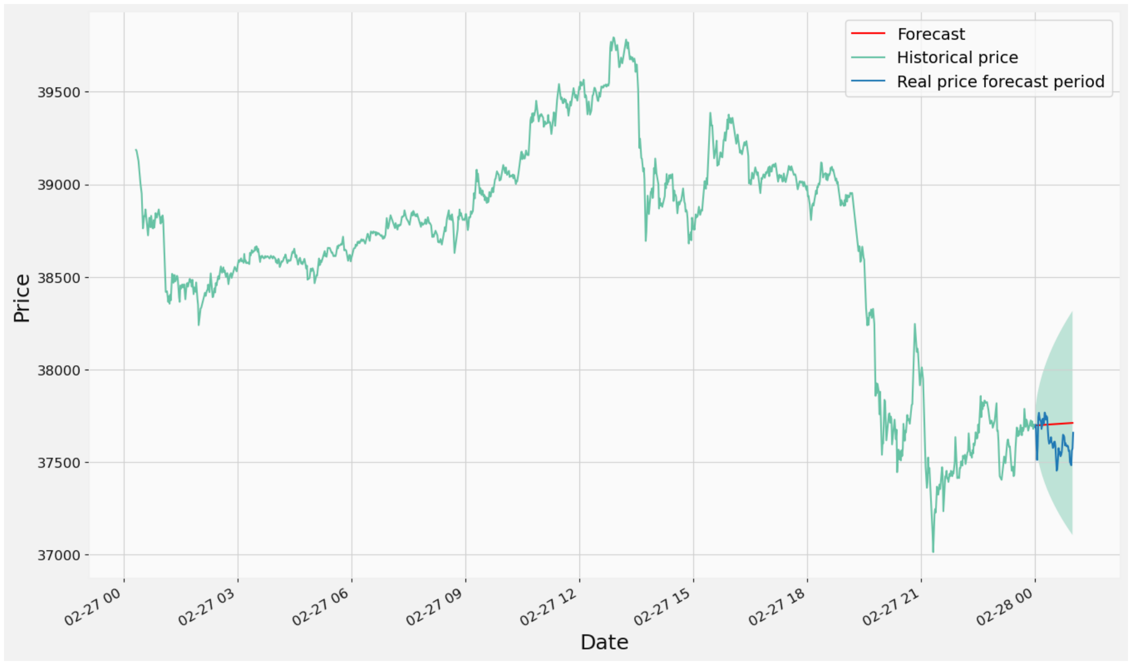
```
[ ]: low_rsi = np.linspace(52.8, 50, 60)
     low_macd = np.linspace(17, 50, 60)
     exog = np.column_stack([low_rsi, low_macd])
     exo_forecast_low = arima_exo.get_forecast(exog=exog, steps=60).summary_frame()
```

```
[ ]: exo_forecast_low = exo_forecast_low.reset_index()
     exo_forecast_low = exo_forecast_low.rename(columns = {"index":"Date"})
     exo_forecast_low.head()
```

```
[ ]: close                Date          mean    mean_se  mean_ci_lower  \
     0     2022-02-28 00:00:00  37698.557484  39.857276   37620.438657
     1     2022-02-28 00:01:00  37698.785382  56.366701   37588.308678
     2     2022-02-28 00:02:00  37699.013279  69.034828   37563.707503
     3     2022-02-28 00:03:00  37699.241177  79.714553   37543.003524
     4     2022-02-28 00:04:00  37699.469075  89.123580   37524.790069

     close  mean_ci_upper
     0       37776.676310
     1       37809.262085
     2       37834.319056
     3       37855.478830
     4       37874.148081
```

```
[ ]: fig, ax = plt.subplots()
     ax.plot(exo_forecast_low["Date"], exo_forecast_low["mean"], color="red",␣
      ↪label="Forecast")
     ax.fill_between(exo_forecast_low["Date"], exo_forecast_low["mean_ci_lower"],
                     exo_forecast_low["mean_ci_upper"], alpha=.4)
     ax.set_xlabel("Date", fontsize=18)
     ax.set_ylabel("Price", fontsize=18)
     df["close"].plot(ax=ax, label="Historical price")
     ax.plot(spot_forecast.index, spot_forecast["Close"], label="Real price forecast␣
      ↪period")
     plt.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x248260e8430>
```



The neutral forecast based on the current model doesn't look much different than the bullish forecast, which is due to the low and uncertain coefficients for the exogenous variables. The main takeaway from the model used to make these forecasts is that predicting the future price action based on the MACD and RSI values in the previous period is difficult. This is kind of suprising given that the RSI is a non-stationary variable and the MACD indicates the difference between the short and long period EMAs. These two regressors should make a relatively good estimation for the trend of the price as was seen with the first model, however with the 1 period lag the confidence intervals of the forecast look like the random walk, which essentially just indicates the implied volatility for the forecast period. One way to address the estimation error with the model on the lagged variables could be to transform them further to indicate a trend that could possibly be used to estimate short forecast periods, as the first autocorrelation plots showed some negative correlation in the periods following a significant price change.

In addition to the criticism about using non-lagged technical indicators that create causality in the model which we saw in the first forecast model with the exogenous regressors, the exogenous regressors themselves can also be criticized. Exogenous variables are exploratory variables that are not correlated with the error term of the model. In the case of the lagged model we can see that the autocorrelation plots show no correlation between the residuals, but at the same time the difference between the bullish and neutral forecasts is almost negligible, indicating that the exogenous regressors does not affect the price prediction of the model significantly. If these variables were transformed in a way that improved the prediction quality of the model, there is a non-zero chance that the correlation is not 0, and the variables can therefore not be considered exogenous. Additionally the way the values for the exogenous variables are created in the forecasts are not realistic as neither of them show linear developments in real markets, however as we saw with the lagged model that is actually able to be used on real market data, the coefficients are so low that however realistic these variables are, it won't affect the prediction much.

6 Summary

Analyzing trading strategies and predicting price changes on financial assets on a relatively high-frequency scale presents some difficult to address aspects that need to be considered, whether it be which variables are used, or bias and overfitting when testing the strategies. The paper does however find that trading with fixed time horizons after hammer and inverted hammer candles was positive over the year the data was tested on based on the raw numbers. Taking into account fees and a difficult to estimate volatility requirement for classifying these candle patterns in advance, the strategy might not be as profitable if applied to current real-time market data

7 Future research

For the hammer and inverted hammer candles: - Look at more variables than just the pattern when predicting whether the return from the trailing periods will be positive or negative. - Try using different trailing periods, or analyze returns for specific candles after the pattern. - Create a more standardized way of classifying and comparing the volatility within the periods.

Forecasting price: - Function to calculate the price in period t based on technical indicators in period $t-1$. Based on the predicted price it is possible to calculate the technical indicators for period t, and it is possible to calculate the price in period $t+1$. This can then be repeated to simulate a forecast, and tweaked until it tracks the future price movements better. - Add more supporting indicators to the indicators already used, E.g. a standardized value for the slope of the exogenous regressors.

8 References

- CoinMarketCap, 2022 https://coinmarketcap.com/
- CoinMarketCap, 2022 https://coinmarketcap.com/rankings/exchanges/