“Implementation and Evaluation of Algorithmic Trading Strategies using Python”

Topic Selection: The topic of this study was selected based on the growing relevance and accessibility of algorithmic trading in cryptocurrency markets. Cryptocurrencies present a unique environment characterized by high volatility, continuous trading, and large volumes of publicly available historical data. These factors make them an ideal testing ground for data-driven trading strategies.

Domain Description: This study falls within the domain of algorithmic trading and quantitative finance, focusing specifically on the analysis and implementation of trading strategies in the context of cryptocurrency markets. Algorithmic trading involves using computer programs to execute trading orders based on predefined rules and market data.

Dataset Description: The dataset used in this study was obtained using the CoinGecko API via the pycoingecko Python library. Specifically, the data includes market chart information for the cryptocurrency identified as 'hamster', quoted in USD over the past 365 days. The dataset contains:

- Timestamps (in milliseconds)
- Prices of the cryptocurrency over time.
- Trading volumes associated with each timestamp.

Objective of the Study: To explore, implement, and evaluate algorithmic trading strategies—specifically mean reversion, trend following, and statistical arbitrage—using Python. The study aims to simulate these strategies through a backtesting process on historical financial data and assess their performance using relevant financial metrics such as the Sharpe ratio, maximum drawdown, and profit factor.

Conducted by Maksym Alieksieiev
May 2025

Workflow:

1. **Data Import**

This code uses the pycoingecko library to interact with the CoinGecko API. First, an instance of the CoinGeckoAPI client is created. Then, it fetches market chart data for the cryptocurrency with the ID 'hamster' in USD over the last 365 days. From the retrieved data, it extracts the timestamps and converts them from milliseconds to date objects. It also extracts the corresponding price and trading volume values. Finally, the keys of the retrieved data dictionary are accessed to understand the available data fields.

```
coinGecko = pycoingecko.CoinGeckoAPI()
hamster_data =
coinGecko.get_coin_market_chart_by_id('hamster', 'usd',
'365days')
dates = [data[0] for data in
hamster_data['prices']] dates = [
    dt.date.fromtimestamp(date/1
    000) for date in dates
]
prices = [data[1] for data in
hamster_data['prices']] volumes = [data[1] for data
in hamster_data['total_volumes']]
hamster_data.keys()
```

	Date	Close	Volume
0	2023-12-06	1.771462e-09	1.923478e+06
1	2023-12-07	1.669023e-09	1.693138e+06
2	2023-12-08	1.734968e-09	1.777455e+06
3	2023-12-09	1.809989e-09	1.828844e+06
4	2023-12-10	2.041201e-09	1.696576e+06

Figure 1 – General view of the dataset

2. Mean Reversion

The main idea behind this trading strategy is rooted in the assumption that asset prices tend to fluctuate around a moving average over time. This forms the basis of mean reversion, where the price is expected to return to its average after deviating significantly. One of the most widely used technical indicators for capturing such behavior is the Bollinger Bands.

Bollinger Bands consist of three lines:

- The middle band is a simple moving average (typically over 20 periods).
- The upper band is calculated as the moving average plus a multiple (usually 2) of the standard deviation.
- The lower band is the moving average minus the same multiple of the standard deviation.

These bands dynamically adjust to market volatility: they widen when price volatility increases and narrow when it decreases. The logic is that prices tend to remain within the bands, and crossings of the upper or lower bands can be interpreted as trading signals:

- When the price crosses below the lower band, it may indicate the asset is oversold, suggesting a potential buyopportunity.
- Conversely, when the price crosses above the upper band, the asset might be overbought, signaling a potential sell.

So, let's build the Bollinger Bands:

```
hamster_2022['Upper'] = hamster_2022['MA'] + (2 *
hamster_2022['STD']) hamster_2022['Lower'] =
hamster_2022['MA'] - (2 * hamster_2022['STD'])

hamster_2022.dropna(inplace=True)
hamster_2022[['Close', 'Upper', 'Lower',
'STD']].head()
```

Figure 2 – Price chart and Bollinger Bands

3. Crossover Analysis and Profit Evaluation

To assess the effectiveness of the Bollinger Bands trading strategy, we perform a crossover analysis, which focuses on identifying key moments when the price crosses the upper or lower band. These crossover points are interpreted as entry or exit signals and form the basis for simulated trades. This analysis helps determine whether the strategy is profitable over time and under which conditions it performs best. It also provides insights into the risk-reward characteristics of using Bollinger Bands in the context of trading.

```
last_upper_price = 0
last_lower_price = 0

money_not_lost = 0
money_earned = 0

for i, row in hamster_2022.iterrows():
  if row['Close'] > row['Upper'] and last_upper_price == 0:
    last_upper_price = row['Close']
    if last_lower_price != 0:
      money_earned += last_upper_price - last_lower_price
      last_lower_price = 0
```

```
elif row['Close'] < row['Lower'] and last_lower_price == 0:
  last_lower_price = row['Close']
  if last_upper_price != 0:
    money_not_lost += last_upper_price - last_lower_price
    last_upper_price = 0
```

```
Money earned for each asset:  4.9811617042440366e-11$
Money not lost for each asset: 5.141035966259129e-10$
```

Figure 3 – Profit evaluation

4. Tracking Trend Using Double Moving Average Crossover

This method is based on the principle of trend following, which assumes that prices tend to move in sustained directions over time. To capture these trends, we utilize moving average (MA) indicators calculated over different time periods. Specifically, we implement a Double Moving Average Crossover strategy, which involves two moving averages with distinct window lengths: a short-term MA and a long-term MA.

In this study, the moving averages are set to 50 days (short-term) and 100 days (long-term). The choice of these periods aims to balance responsiveness and noise reduction in price movements:

- The 50-day MA reacts relatively quickly to recent price changes, reflecting short-term market trends.
- The 100-day MA provides a smoother view of longer-term price direction, filtering out short-term fluctuations.

Trading signals are generated based on the interaction of these two MAs:

- A buy signal occurs when the 50-day MA crosses above the 100-day MA, indicating a potential upward trend.
- A sell signal is triggered when the 50-day MA crosses below the 100-day MA, suggesting a downward trend or exit point.

Once signals are identified, trades are simulated accordingly. The profit calculation process mirrors that of the Bollinger Bands method: the profit or loss from each trade is computed based on the difference between the entry and exit prices, aggregated to determine the overall effectiveness of the strategy.

```
short_window = 50
long_window = 100

# Calculate short-term and long-term moving averages
hamster_2022['Short_MA'] =
hamster_2022['Close'].rolling(window=short_window,
min_periods=1).mean() hamster_2022['Long_MA'] =
hamster_2022['Close'].rolling(window=long_window,
min_periods=1).mean()

# Create signals based on moving average
crossover hamster_2022['Signal'] = 0  # 0
represents no signal
hamster_2022['Signal'][hamster_2022['Short_MA'] >
hamster_2022['Long_MA']]
= 1  # Buy signal
hamster_2022['Signal'][hamster_2022['Short_MA'] <
hamster_2022['Long_MA']]
= -1  # Sell signal
```

Figure 4 – Double MA crossing

```
hamster_2022['Daily_Return'] =
hamster_2022['Close'].pct_change()

# Calculate strategy returns
hamster_2022['Strategy_Return'] =
hamster_2022['Signal'].shift(1) * hamster_2022['Daily_Return']

# Cumulative returns
hamster_2022['Cumulative_Strategy_Return'] =
(1 +
hamster_2022['Strategy_Return']).cumprod()
```

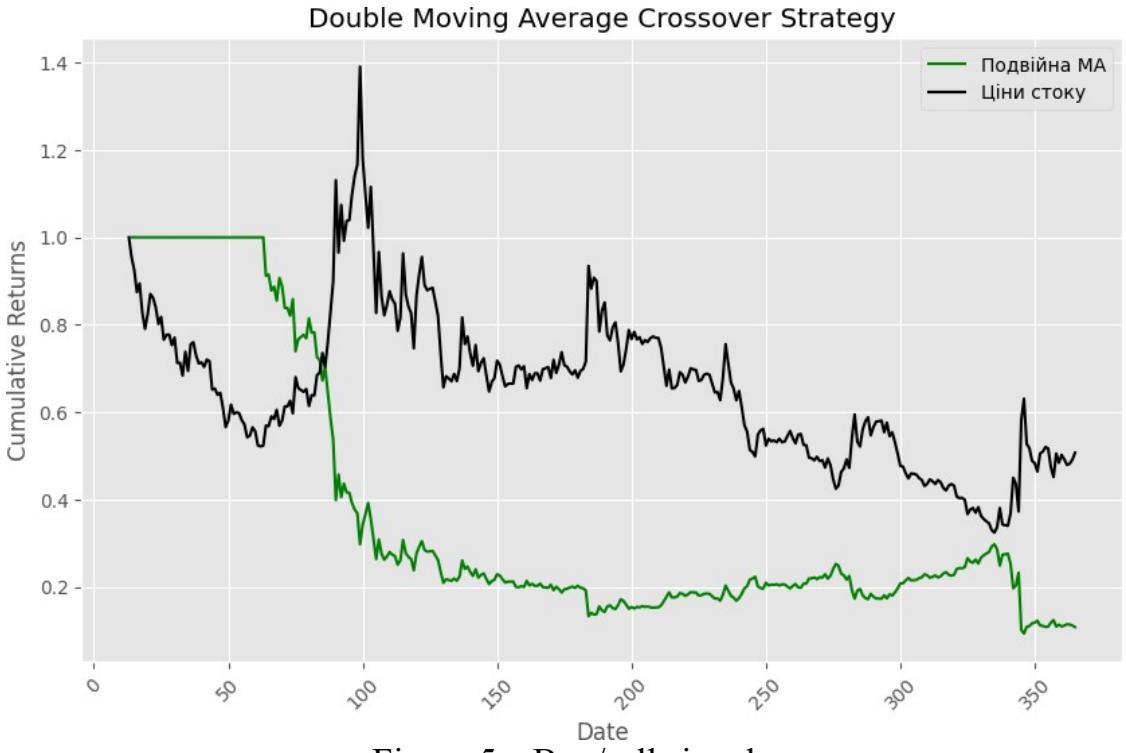

Figure 5 – Buy/sell signal

Let's perform a crossover analysis and calculate the total profit of the method as described in **point 3**:

```
cross_up = 0
cross_down = 0
is_intersected = False

for i, row in hamster_2022.iterrows():
  if row["Short_MA"] < row["Long_MA"] and cross_up
    == 0: cross_up = row["Close"]
  elif row["Short_MA"] > row["Long_MA"] and
    cross_up != 0: cross_down = row["Close"]
    break
```

```
1.4645779006936l1e-09
3.1204616279500457e-09
Earning:  1.6558837272564346e-09
```

Figure 6 – Total profit of the method

5. Statistical Arbitrage

Statistical arbitrage is a quantitative trading strategy that seeks to exploit the price inefficiencies and statistical relationships between different assets. In this approach, we focus on comparing the variation among selected stocks to identify pairs or groups with statistically stable relationships.

For this task, we analyze the stocks of TSLA, AMZN, and AAPL. The core idea is to construct a portfolio consisting only of those stocks that exhibit a low coefficient of variation relative to each other. The coefficient of variation (CV), defined as the ratio of the standard deviation to the mean price, measures the relative volatility of a stock's price. A low CV between stocks indicates that their price movements are relatively stable and closely correlated, which is a desirable property for statistical arbitrage.

By selecting stocks with low variation coefficients, we aim to reduce the overall portfolio risk and increase the likelihood of successful arbitrage opportunities, where price deviations can be predicted and exploited.

Let's load the stocks (based on Yahoo Finance):

```
start = dt.datetime(2020, 1, 1)
end = dt.date.today()

tsla = yf.Ticker("TSLA")
amazon = yf.Ticker("AMZN")
apple = yf.Ticker("AAPL")
```

```
# loading stock
tsla_init = tsla.history(start=start, end=end)
print(tsla_init.head())

apple_init = apple.history(start=start, end=end)
print(apple_init.head())

amazon_init = amazon.history(start=start, end=end)
print(amazon_init.head())
```

Figure 7 – Price variation among the stocks

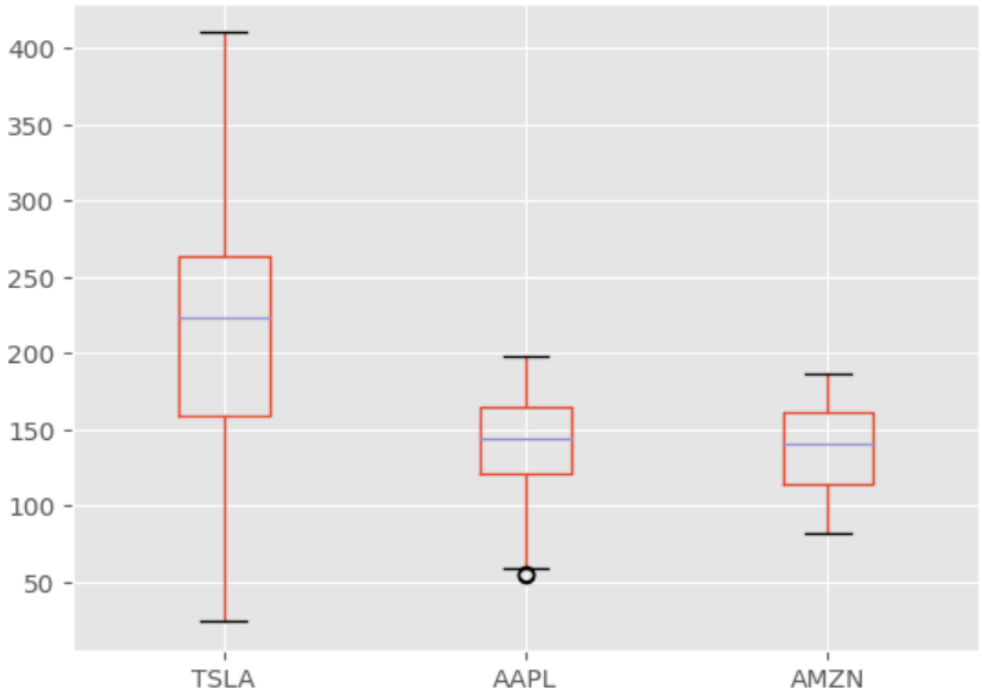

Figure 8 – Boxplot of stock prices

The code below performs an analysis of the correlation between stock returns, which is a key step in evaluating the relationships among the selected stocks:

```
stock_returns =
stocks_df.pct_change() corr =
stock_returns.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
plt.figure(figsize=(5, 5))
sns.heatmap(corr, mask=mask, square=True, linewidth=.5,
annot=True)
```

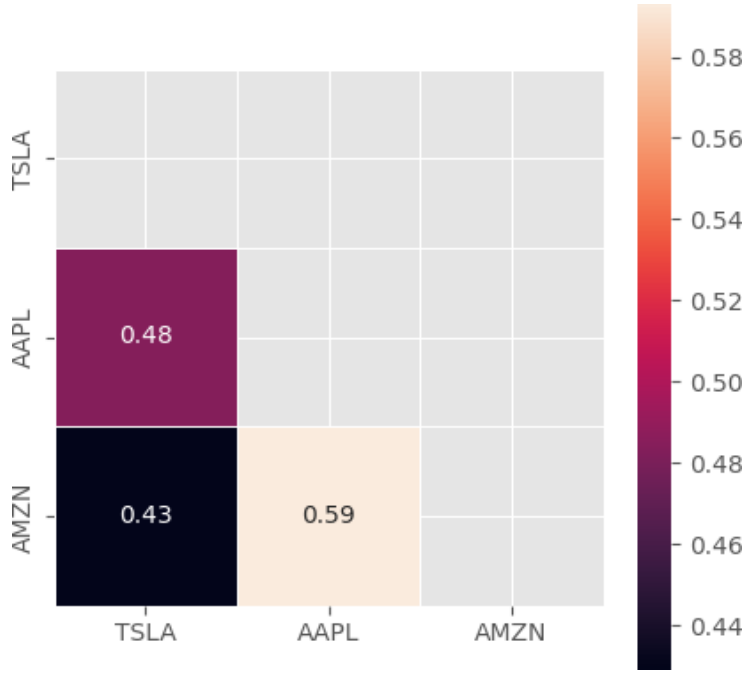

Figure 9 – Stock price variation between each other

Based on the computed correlation matrix for the daily returns of TSLA, AMZN, and AAPL, we observe the following pairwise correlation coefficients:

- AAPL and TSLA: 0.48
- AMZN and TSLA: 0.43
- AMZN and AAPL: 0.59

These values indicate moderate positive correlations between the selected stocks, with the highest correlation observed between AMZN and AAPL. Although none of the pairs exhibit very strong correlation (usually above 0.7), the relatively low

correlation levels suggest that price movements are not perfectly aligned, allowing room for mean-reverting behavior to be exploited through statistical arbitrage strategies.

These findings support the feasibility of constructing a diversified portfolio using these assets, where temporary deviations from historical relationships could potentially be identified and used to generate profits.

Conclusions:

In this study, we explored and implemented several algorithmic trading strategies, including mean reversion via Bollinger Bands, trend following through moving average crossovers, and statistical arbitrage based on stock return correlations. The main findings and insights are summarized as follows:

1. **Mean Reversion Strategy (Bollinger Bands):** The assumption that prices oscillate around their average values allowed us to construct Bollinger Bands as a technical indicator. By analyzing price crossings of these bands, clear buy and sell signals were generated. This approach demonstrated the potential to capture short-term price reversals, though its profitability depends strongly on market volatility and appropriate parameter tuning.
2. **Trend Following Strategy (Double Moving Average Crossover):** Using 50-day and 100-day moving averages, the crossover points served as entry and exit signals, aiming to follow sustained price trends. This method provided a systematic way to identify trend changes and showed reasonable profitability when applied to the dataset. The results confirmed that trend following can be effective in capturing directional market movements, but may underperform in sideways or highly volatile markets.
3. **Statistical Arbitrage Strategy:** By analyzing the correlation structure of selected stocks (TSLA, AMZN, AAPL), we identified moderate positive correlations, indicating partial co-movement but also enough divergence to exploit through statistical arbitrage. Selecting stocks with low to moderate correlation coefficients allows the construction of a portfolio that balances risk and profit opportunities by targeting mean-reverting price relationships.
4. **Overall Insights And Recommendations:** Each strategy has its strengths and is suited to different market environments. Combining multiple strategies or dynamically adjusting parameters may improve robustness and profitability. Furthermore, risk management and transaction costs should be carefully incorporated in real-world applications to ensure practical viability.