Chapter 4

INITIAL FINDINGS

4.1 Introduction

In the Chapter 4, the outlines covered the anticipated results and outcome from the implementation of deep reinforcement learning (DRL) for the automated trading in the stock market. Implementation of DRL in financial industry approach to its ability to optimize the trading strategy through the trial and error, learning based on the reward mechanism. In this research is to develop and evaluate the DRL-based trading system that able to autonomously make trading decision based on the real-time market data.

In the research includes performing of exploratory data analysis (EDA) to obtain the initial insights from the dataset (S&P 500) and understand the underlying trends and patterns. EDA is the essential process as can helps to identify the patterns of dataset, anomalies detection, formation of hypotheses and validate assumptions through the descriptive statistics and visual representations. The chapter will start with data preprocessing/data cleaning followed by EDA where including descriptive statistics employed to explore the data and visualization of the dataset. Those are able to provide the information for the feature engineering and the subsequent of training for the DRL models.

The expected outcome of the research is the creation of the robust DRL trading agents that can make the correct decision making based on the real-time market trends. EDA and feature engineering processes able to generate the guideline for the development of the model. Out of that, the potential challenges of the research such as volatility of the market, high adaptability in the different market conditions and overfitting will be addressed. In the expected outcome will including model development structure and the model evaluation. The model development structure and the model evaluation are the expected setting of the model will be used in the research.

4.2 Data Pre-processing/ Data Cleaning

The data will be cleaning by checking the missing data and outliers as the initial step. If the data consists of missing data will be handled with the interpolation method to make sure the data is not missing values as following as Figure 4.1.

```
[ ]  # Check if there are any missing values in the data
     if missing_data.any():
         print("Missing values found. Interpolation will be performed.")

         # Perform linear interpolation to fill missing values
         data_interpolated = data.interpolate(method='time', limit_direction='both')

         print("Interpolation completed.")

     else:
         # If no missing data, skip interpolation
         data_interpolated = data
         print("No missing values found. Skipping interpolation.")

     # Verify that there are no more missing values
     missing_data_after = data_interpolated.isnull().sum()
     print("Missing values after interpolation:\n", missing_data_after)
```

```
No missing values found. Skipping interpolation.
Missing values after interpolation:
 Price   Ticker
Close    ^GSPC     0
High     ^GSPC     0
Low      ^GSPC     0
Open     ^GSPC     0
Volume   ^GSPC     0
dtype: int64
```

Figure 4.1 Track of the missing values after interpolation

After that, the data will continue clean with remove outlier which is remove the abnormal data point to prevent the agents are learning the abnormal condition. The results after remove outliers as the Figure 4.2.

```
[ ] from scipy.stats import zscore

    # Calculate z-scores for the close prices
    data['zscore'] = zscore(data['Close'])

    # Filter out data points where z-score is greater than 3 (outliers)
    data_clean = data[data['zscore'].abs() <= 3]

    # Drop the z-score column for the final cleaned data
    data_clean = data_clean.drop(columns=['zscore'])

    print(data_clean.head())
```

```
Price             Close         High          Low         Open      Volume
Ticker            ^GSPC        ^GSPC        ^GSPC        ^GSPC       ^GSPC
Date
2016-01-04  2012.660034  2038.199951  1989.680054  2038.199951  4304880000
2016-01-05  2016.709961  2021.939941  2004.170044  2013.780029  3706620000
2016-01-06  1990.260010  2011.709961  1979.050049  2011.709961  4336660000
2016-01-07  1943.089966  1985.319946  1938.829956  1985.319946  5076590000
2016-01-08  1922.030029  1960.400024  1918.459961  1945.969971  4664940000
```

Figure 4.2 Results after remove outliers.

Normalize the feature of the ‘Close’ price to a range [0,1]. The Normalized ‘Close’ data will show at Figure 4.3.

```
[ ] # Normalize the 'Close' column to a range [0, 1]
    data['Close Normalized'] = (data['Close'] - data['Close'].min()) / (data['Close'].max() - data['Close'].min())

    # Display the first few rows
    print(data[['Close', 'Close Normalized']].head())
```

```
Price            Close Close Normalized
Ticker           ^GSPC
Date
2016-01-04  2012.660034         0.061864
2016-01-05  2016.709961         0.063229
2016-01-06  1990.260010         0.054315
2016-01-07  1943.089966         0.038420
2016-01-08  1922.030029         0.031323
```

Figure 4.3 Normalize the ‘Close’ column to a range [0,1]

The data will separate into training and test set which is 2016-2022 will be training set and 2023-2024 will be test set. The data will split as the Figure 4.4.

```python
import pandas as pd

# Load your data (replace with the actual file path or DataFrame)
# Assuming your data has a column 'Date' and 'Close' (or other stock-related data)

# Convert the Date column to datetime if not already done
data.index = pd.to_datetime(data.index)

# Calculate the split index
train_size = int(len(data) * 0.8)  # 80% for training

# Split the data into train and test sets
train_data = data.iloc[:train_size]
test_data = data.iloc[train_size:]

# Display the results
print("Train Data (80%):")
print(train_data.head())  # Show the first few rows of the train data
print("\nTest Data (20%):")
print(test_data.head())  # Show the first few rows of the test data
```

```
Train Data (80%):
Price             Close         High          Low         Open      Volume  \
Ticker            ^GSPC        ^GSPC        ^GSPC        ^GSPC       ^GSPC
Date
2016-01-04  2012.660034  2038.199951  1989.680054  2038.199951  4304880000
2016-01-05  2016.709961  2021.939941  2004.170044  2013.780029  3706620000
2016-01-06  1990.260010  2011.709961  1979.050049  2011.709961  4336660000
2016-01-07  1943.089966  1985.319946  1938.829956  1985.319946  5076590000
2016-01-08  1922.030029  1960.400024  1918.459961  1945.969971  4664940000

Price         zscore Close Normalized
Ticker
Date
2016-01-04 -1.495889         0.061864
2016-01-05 -1.491027         0.063229
2016-01-06 -1.522778         0.054315
2016-01-07 -1.579402         0.038420
2016-01-08 -1.604683         0.031323

Test Data (20%):
Price             Close         High          Low         Open      Volume  \
Ticker            ^GSPC        ^GSPC        ^GSPC        ^GSPC       ^GSPC
Date
2022-05-24  3941.479980  3955.679932  3875.129883  3942.939941  4923190000
2022-05-25  3978.729980  3999.330078  3925.030029  3929.590088  4802560000
2022-05-26  4057.840088  4075.139893  3984.600098  3984.600098  4709970000
2022-05-27  4158.240234  4158.490234  4077.429932  4077.429932  4375620000
2022-05-31  4132.149902  4168.339844  4104.879883  4151.089844  6822640000

Price         zscore Close Normalized
Ticker
Date
2022-05-24  0.819501         0.711850
2022-05-25  0.864216         0.724403
2022-05-26  0.959182         0.751062
2022-05-27  1.079704         0.784895
2022-05-31  1.048384         0.776103
```

Figure 4.4 Training and test dataset

After the preprocessing and data cleaning, will have the final check as the confirmation that data is fully cleaned. The cleaned data checking will be shows as Figure 4.5.

```
# Check for remaining missing values
print(data.isnull().sum())

# Check for duplicated rows
print(data.duplicated().sum())

# Display the final cleaned data
print(data.head())
```

```
Price             Ticker
Close             ^GSPC     0
High              ^GSPC     0
Low               ^GSPC     0
Open              ^GSPC     0
Volume            ^GSPC     0
zscore                      0
Close Normalized            0
dtype: int64
0
Price             Close         High          Low         Open      Volume  \
Ticker            ^GSPC        ^GSPC        ^GSPC        ^GSPC       ^GSPC
Date
2016-01-04  2012.660034  2038.199951  1989.680054  2038.199951  4304880000
2016-01-05  2016.709961  2021.939941  2004.170044  2013.780029  3706620000
2016-01-06  1990.260010  2011.709961  1979.050049  2011.709961  4336660000
2016-01-07  1943.089966  1985.319946  1938.829956  1985.319946  5076590000
2016-01-08  1922.030029  1960.400024  1918.459961  1945.969971  4664940000

Price         zscore Close Normalized
Ticker
Date
2016-01-04 -1.495889         0.061864
2016-01-05 -1.491027         0.063229
2016-01-06 -1.522778         0.054315
2016-01-07 -1.579402         0.038420
2016-01-08 -1.604683         0.031323
```

Figure 4.5 Final check for the dataset

4.3 Exploratory Data Analysis (EDA)

EDA is an essential stage in comprehending the fundamental framework of the data, identifying patterns, detecting anomalies, and forming hypotheses. The following subsections describe the visualizations, descriptive statistics, initial insights, and feature engineering performed during the EDA phase. EDA was carried out to understand the data patterns.

4.3.1 Descriptive Analysis

The daily price of the S&P 500 was taken from the Yahoo Finance by using the API of yfinance in Python. The first 75% of the dataset (2016-2022) is used for the training, while the rest is used for the validation and testing (2023-2024). The statistics analysis of the data of S&P 500 from 01/01/2016 to 01/01/2024 data points are shown in Figure 4.6.

```
[ ] # Statistical summary of the dataset
    print(data.describe())
```

```
Price         Close         High          Low         Open        Volume  \
Ticker        ^GSPC        ^GSPC        ^GSPC        ^GSPC         ^GSPC
count   2012.000000  2012.000000  2012.000000  2012.000000  2.012000e+03
mean    3258.800263  3276.243445  3238.939939  3258.235883  4.059945e+09
std      833.250376   838.711695   827.403837   832.944035  1.002561e+09
min     1829.079956  1847.000000  1810.099976  1833.400024  0.000000e+00
25%     2584.929993  2596.832520  2572.749939  2582.687439  3.435872e+09
50%     3004.994995  3016.340088  2990.939941  3004.680054  3.845810e+09
75%     4079.305054  4099.894897  4056.354919  4077.399902  4.448278e+09
max     4796.560059  4818.620117  4780.979980  4804.509766  9.976520e+09

Price         zscore Close Normalized
Ticker
count   2.012000e+03      2012.000000
mean    3.390264e-16         0.481796
std     1.000249e+00         0.280794
min    -1.716262e+00         0.000000
25%    -8.089259e-01         0.254711
50%    -3.046724e-01         0.396267
75%     9.849486e-01         0.758295
max     1.845954e+00         1.000000
```

Figure 4.6 Statistics Analysis for the S&P 500

The data points are also depicted in Figure 4.7 It can be roughly described that generally, the trend is going up, although there is some sudden plummeting in this period, and from the start of 2022, the price is following a downtrend. As for the distribution of the data points, most of the data points fell below 3700 and the maximum is 4796.56 showing that the data is right-skewed.

```
# Plotting the Closing Price
plt.figure(figsize=(12,6))
plt.plot(data['Close'], label='S&P 500 Close')
plt.title('S&P 500 Close Price (2000-2024)', fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Price (USD)', fontsize=12)
plt.legend()
plt.show()
```

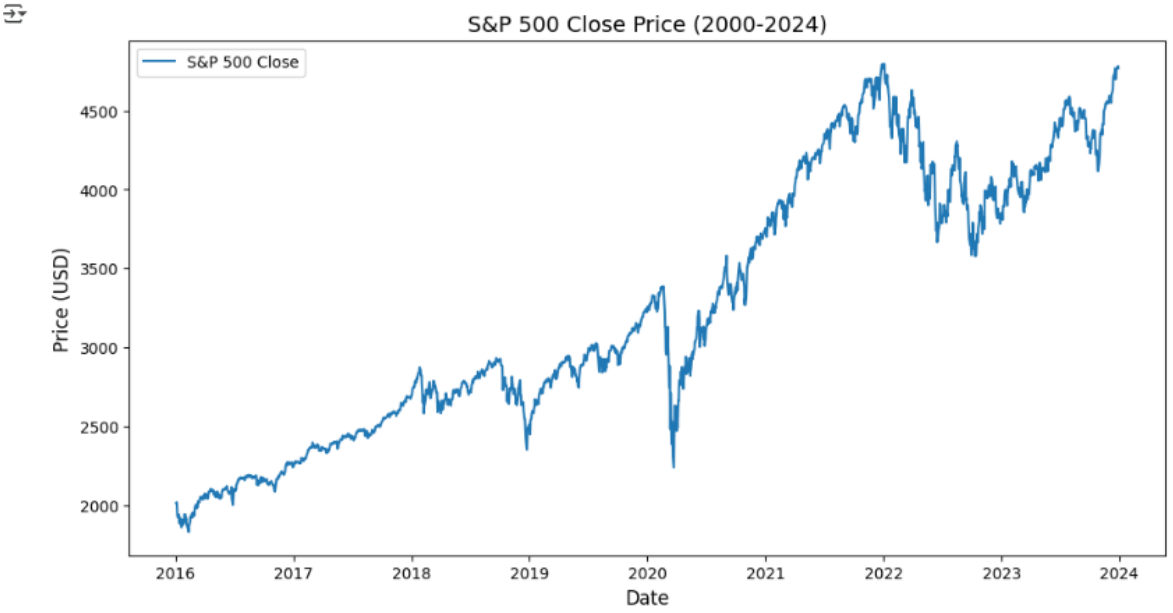


Figure 4.7: S&P 500 closing price from 01/01/2016 to 01/01/2024

Things get complicated when it comes to time-series stock market prediction. That is because the data tends to be non-stationary, that is the data points depend on the time at which they are observed. To get useful information, some of the traditional trading techniques from are being applied in this project such as the 30-day moving average and 100-day moving average as the features engineering. The Figure 4.8 shown the S&P 500 closing price with the 30-day moving average and 100-day moving average. As the Figure 4.8 shown that when the 30-day moving average and 100-day moving average lower than the closing price, which means the market is considered in the uptrend condition. When the 30-day moving average and 100-day moving average higher than the closing price, which means the market is considered in the downtrend condition. When the 30-day moving average crosses above the 100-day moving average, the condition is called Golden Cross which means the trends is potential for upward movement in the price. When the 30-day moving average crosses below the 100-day moving average, the condition is called Death Cross which means the trends is potential for downward movement in the price.

```
[ ]  # Compute 30-day and 100-day rolling mean and standard deviation
     data['30_day_MA'] = data['Close'].rolling(window=30).mean()
     data['100_day_MA'] = data['Close'].rolling(window=100).mean()
     data['30_day_STD'] = data['Close'].rolling(window=30).std()

     # Plotting the closing price along with rolling mean and std
     plt.figure(figsize=(12,6))
     plt.plot(data['Close'], label='S&P 500 Close', color='blue')
     plt.plot(data['30_day_MA'], label='30-Day Rolling Mean', color='orange')
     plt.plot(data['100_day_MA'], label='100-Day Rolling Mean', color='green')
     plt.title('S&P 500 with Rolling Mean (30, 100)', fontsize=14)
     plt.xlabel('Date', fontsize=12)
     plt.ylabel('Price (USD)', fontsize=12)
     plt.legend()
     plt.show()
```

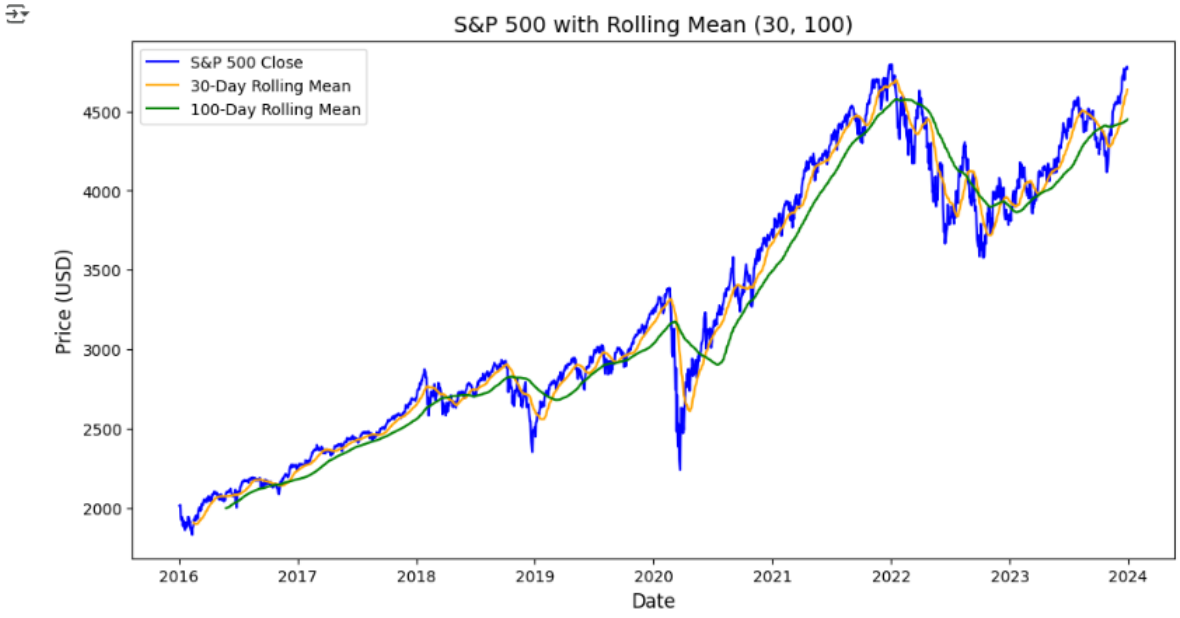


Figure 4.8 S&P 500 closing price with the 30-day moving average and 100-day moving average.

The distribution of the daily returns in the histogram will be plot to identify the frequency of the daily returns rate. In the Figure 4.9 shows the small price changes (in percentages) occurs most frequently but the significant market of fluctuations is still present and occasionally cause extreme returns. The distribution of the daily returns also helps to identify the behaviour of the returns either positive/ negative skew or kurtosis.

```
[ ]  # Compute daily returns
     data['Daily Return'] = data['Close'].pct_change()

     # Plot the distribution of daily returns
     plt.figure(figsize=(10,6))
     sns.histplot(data['Daily Return'], bins=100, kde=True, color='purple')
     plt.title('Distribution of Daily Returns (S&P 500)', fontsize=14)
     plt.xlabel('Daily Return', fontsize=12)
     plt.ylabel('Frequency', fontsize=12)
     plt.show()
```

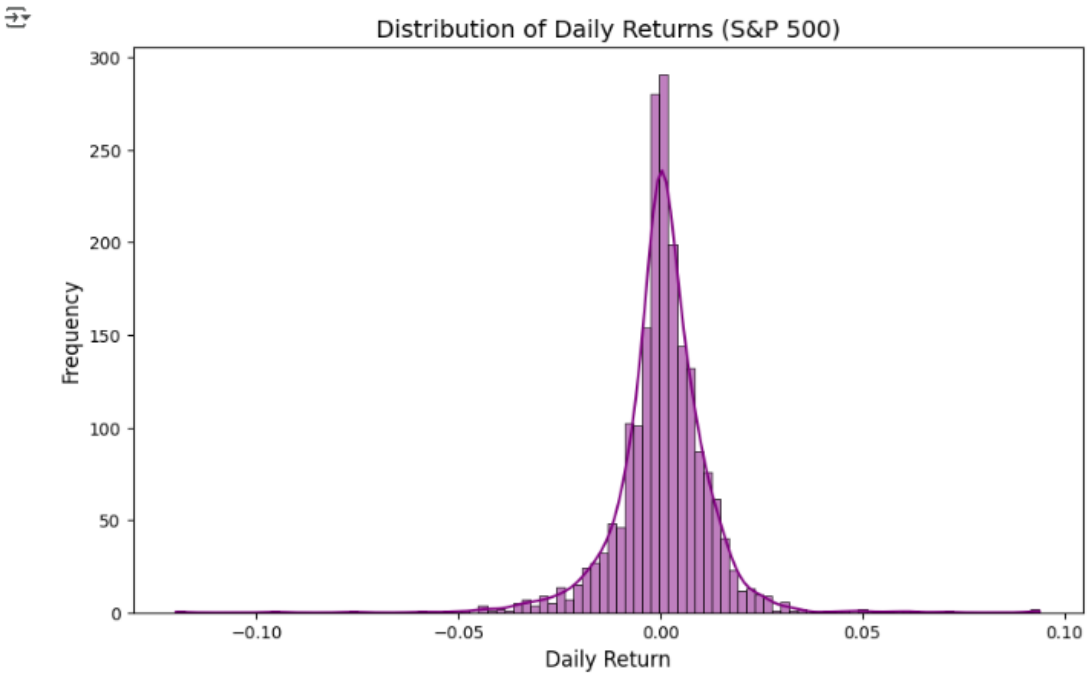


Figure 4.9 Distribution of Daily Returns in percentages (S&P 500)

```
[ ]  # Plotting the histogram for daily price differences
     plt.figure(figsize=(12, 6))
     plt.hist(data['Daily Difference'], bins=50, color='purple', edgecolor='black', alpha=0.7)
     plt.title('Histogram of S&P 500 Daily Price Differences', fontsize=14)
     plt.xlabel('Daily Price Difference (USD)', fontsize=12)
     plt.ylabel('Frequency', fontsize=12)
     plt.grid(True)
     plt.show()
```

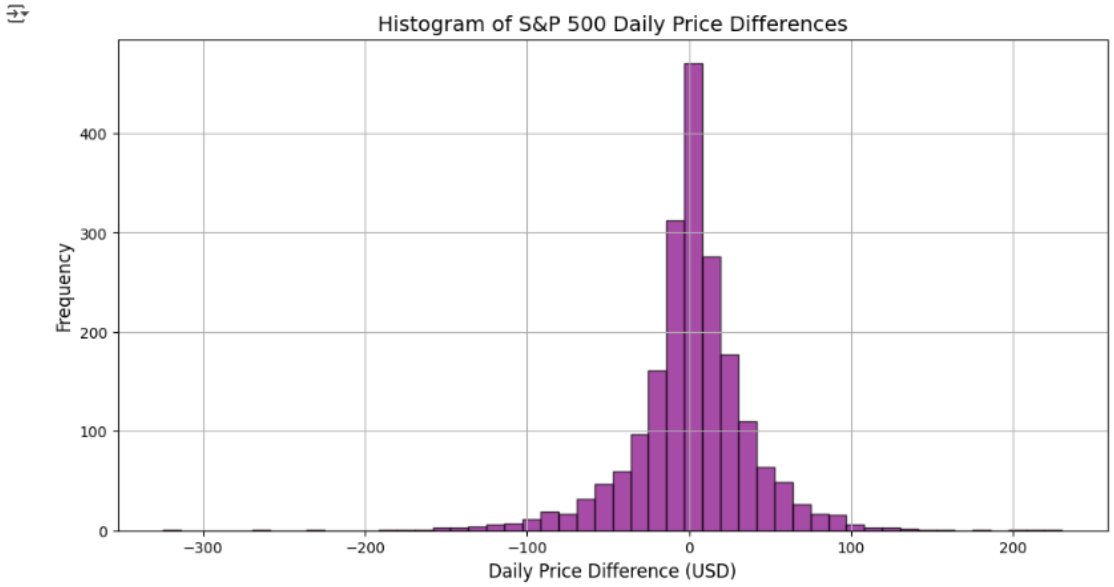


Figure 4.10 Distribution of Daily Returns in USD (S&P 500)

Correlation heatmap helps to identify the relation between the features. In the Figure 4.11 shows that the correlations between different price metrics/features (open, close, high, low and volume). The correlation between the features is strong because all of the metrics are inter-link with others features.

```
# Create a correlation heatmap
corr_matrix = data[['Open', 'High', 'Low', 'Close', 'Volume']].corr()

plt.figure(figsize=(10,6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix for S&P 500 Price Data', fontsize=14)
plt.show()
```

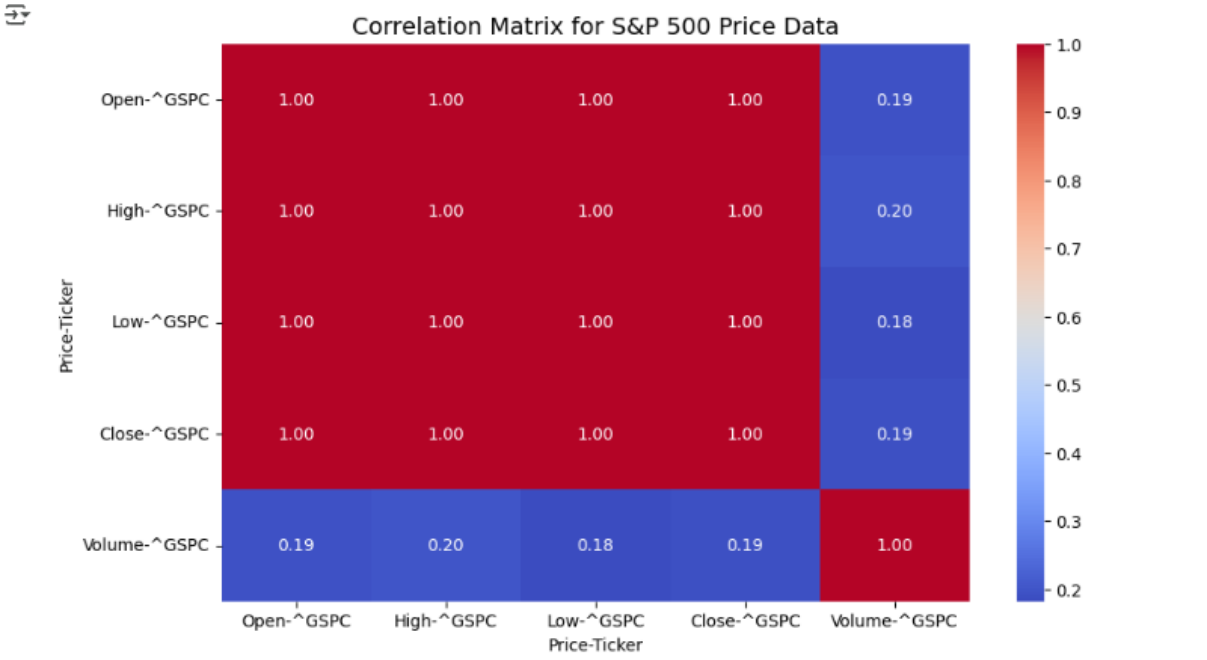


Figure 4.11 Correlation Matrix for S&P 500 Price Data

The volatility of the market over time can identify by using the volatility plot. In the figure 4.12 shows that the S&P 500 30-Day volatility. In 2020, the volatility of the market is sharpest because of the COVID-19 crash. The significant event leads to unexpected economic shutdowns caused by the pandemic. From 2016 until 2019 shows the stable periods in the economic and the confidence among investors. Crisis period during 2020 and recovery and post-COVID volatility after 2020. The figure 4.13 shows the difference in daily price either is positive or negative to identify the distribution of the price.

```
[ ]  # Compute 30-day volatility (standard deviation of daily returns)
     data['30_day_volatility'] = data['Daily Return'].rolling(window=30).std()

     # Plotting volatility
     plt.figure(figsize=(12,6))
     plt.plot(data['30_day_volatility'], label='30-Day Volatility', color='red')
     plt.title('S&P 500 30-Day Volatility', fontsize=14)
     plt.xlabel('Date', fontsize=12)
     plt.ylabel('Volatility', fontsize=12)
     plt.legend()
     plt.show()
```

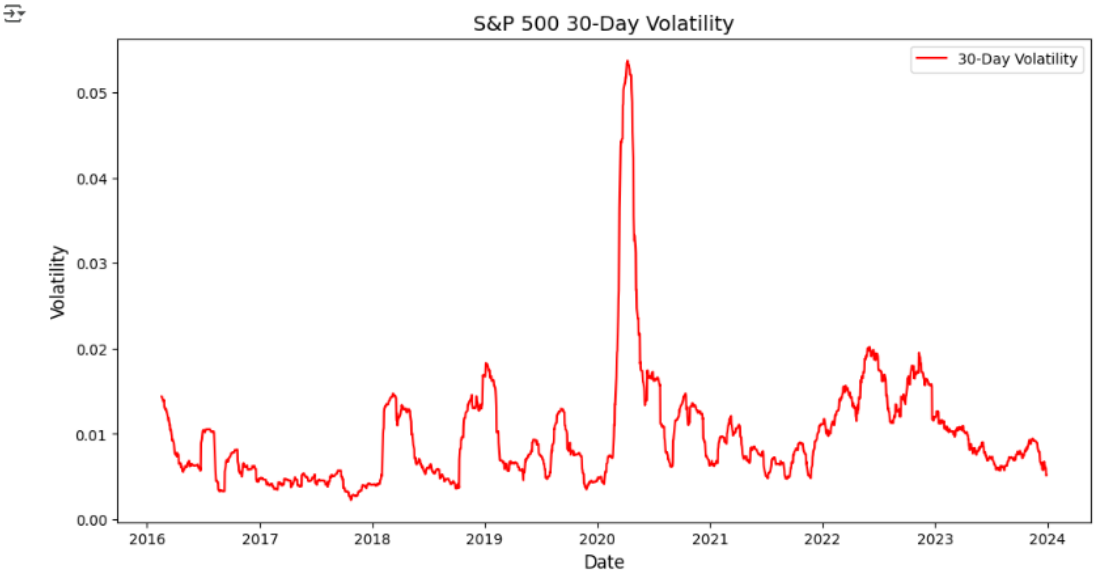


Figure 4.12 S&P 500 30-Day Volatility

```python
# Count the number of positive and negative differences
positive_diff = (data['Difference Type'] == 'Positive').sum()
negative_diff = (data['Difference Type'] == 'Negative').sum()

# Data for the pie chart
labels = ['Positive Difference', 'Negative Difference']
sizes = [positive_diff, negative_diff]
colors = ['#66b3ff', '#ff6666']  # Blue for positive, Red for negative

# Check if there are any positive or negative differences to plot
if sum(sizes) > 0:
    # Plot the pie chart
    plt.figure(figsize=(8, 8))
    plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90, wedgeprops={'edgecolor': 'black'})
    plt.title('Positive vs Negative Daily Price Differences (S&P 500)', fontsize=14)
    plt.axis('equal')  # Equal aspect ratio ensures the pie chart is drawn as a circle.
    plt.show()
else:
    print("No positive or negative daily differences to plot for the pie chart.")
```

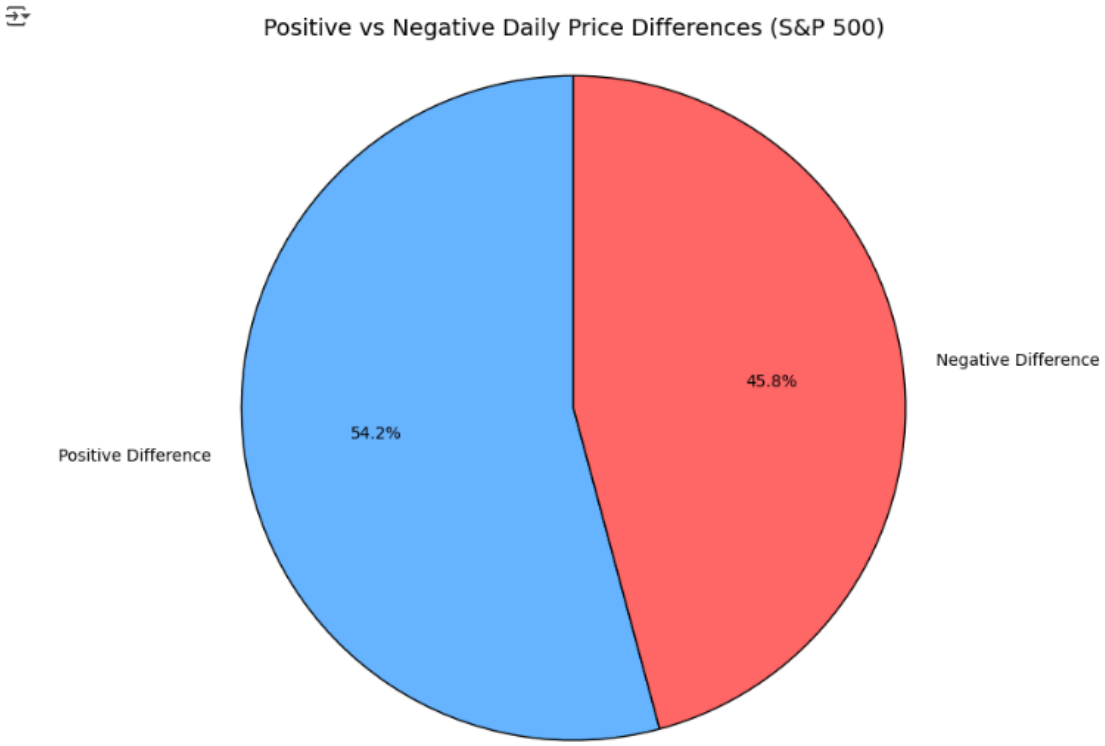


Figure 4.13 Positive vs Negative Daily Price Differences (S&P 500)

4.4 Expected Outcome

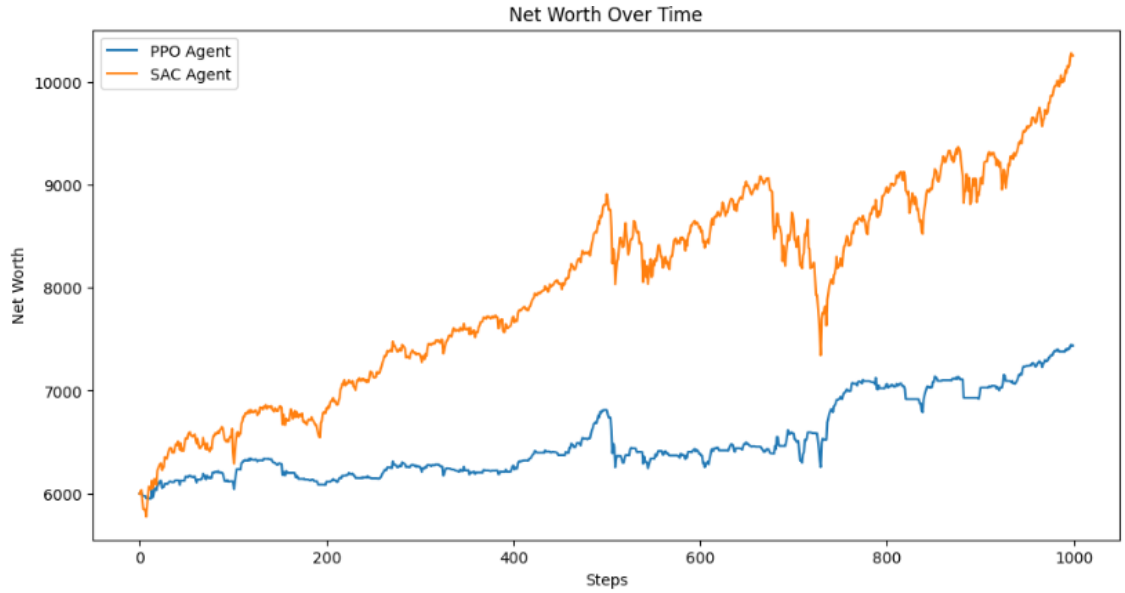
In the research, the expected contribution is to train agent with the high efficiency and stable in sentimental. What kinds of the difference will be brought by using the different model in trading agents which is PPO, SAC and DQN. Indicate the strengths and weaknesses of the PPO, SAC and DQN in trading strategy. Helping stakeholder to harvest more profit without worry and sentimental impact. The trader will be automation decision making based on the policy and reward mechanism. The establish policy and reward mechanism is needed to train

the agent that can fully eliminated the sentimental and the decision-making will be more discipline and the profit will be optimized. Every model has their own strengths and weaknesses; by identify the advantages of model will let the stakeholder apply those models in more appropriately toward the real-time market environment. The model with the higher value in F1-scores and accuracy, and optimized cumulative return rates based on the back testing will be selected for the future trading strategy in decision making.

4.4.1 Model Development Structure

Based on the Figure 4.14 and 4.15, the performance of the DQN, SAC and PPO can be compared with the same input which is S&P 500. For the net worth over time, PPO agent shows a steady increase. There are some of the stagnations or decline periods before the value begins to rise sharply. PPO able to make the profitable decision but encountering some of the limitation when facing with the certain market conditions. For SAC agent, it much smoother and more consistent rise in the net worth over time compared with the PPO agent. SAC agent could be the better at handling fluctuations in the market and more consistent. The DQN shows a more volatile net worth trajectory with the dramatically ups and downs. The indicates will cause the DQN agent has the high returns, it struggles to maintain the consistent growth and lose in stability of the decision-making process. The performance of the DRL models also can compare in the Sharpe ratio. SAC agent has the highest Sharpe ratio around 0.0747, which means it has a favorable balance between return and risk. SAC able to generate return with less volatility and better risk-adjusted performance. PPO agent with the lower Sharper ratio around 0.0342 which means the some of the return bring the risk where the risk-adjusted returns are much lower compared with SAC agent due to the high volatility of market in decision making process. DQN agent with the slightly higher Sharpe ratio to PPO around 0.0736. The return rate of DQN agent is decent, the volatility in the net worth but it loss the stability in decision making.

Based on the performance comparison, SAC outperform than PPO and DQN in terms of net worth growth and Sharpe ratio because SAC has the better ability to handle fluctuation of market and able to generate higher returns with lower risk. PPO has steady progress but faces more volatility and less favorable risk-adjusted returns compared with SAC. DQN with the highest volatility in performance where the significant fluctuations in net worth and a relatively low Sharpe ratio which means DQN hard to maintain the decision making in trading consistently. SAC more suitable for the dynamics stock market.



	Agent	Return	Standard Deviation	Sharpe Ratio
1	SAC Agent	0.000566	0.00757	0.074730
0	PPO Agent	0.000133	0.00388	0.034171

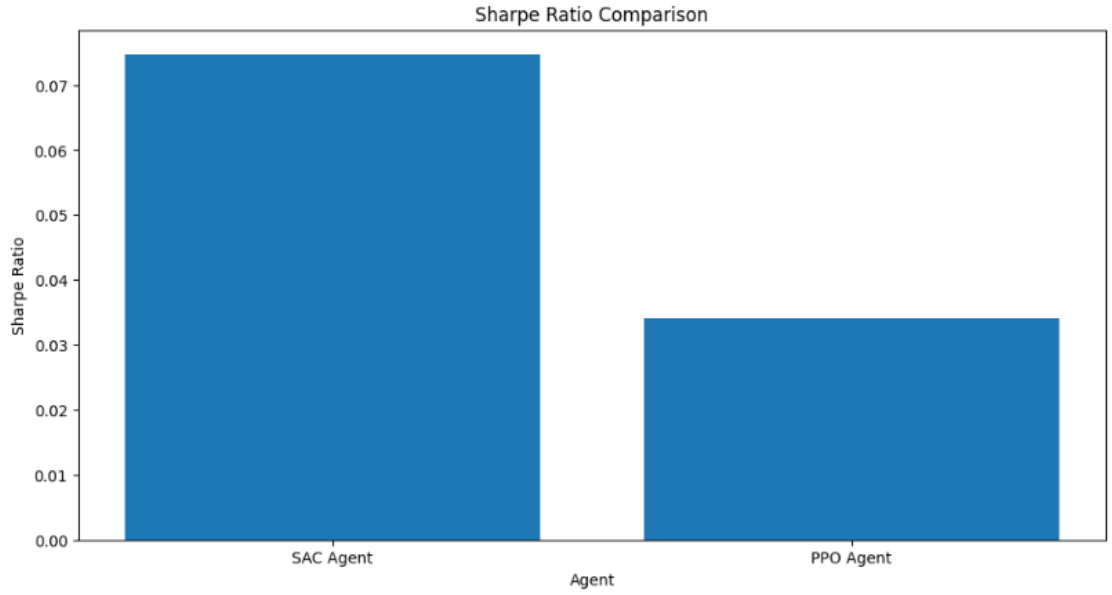
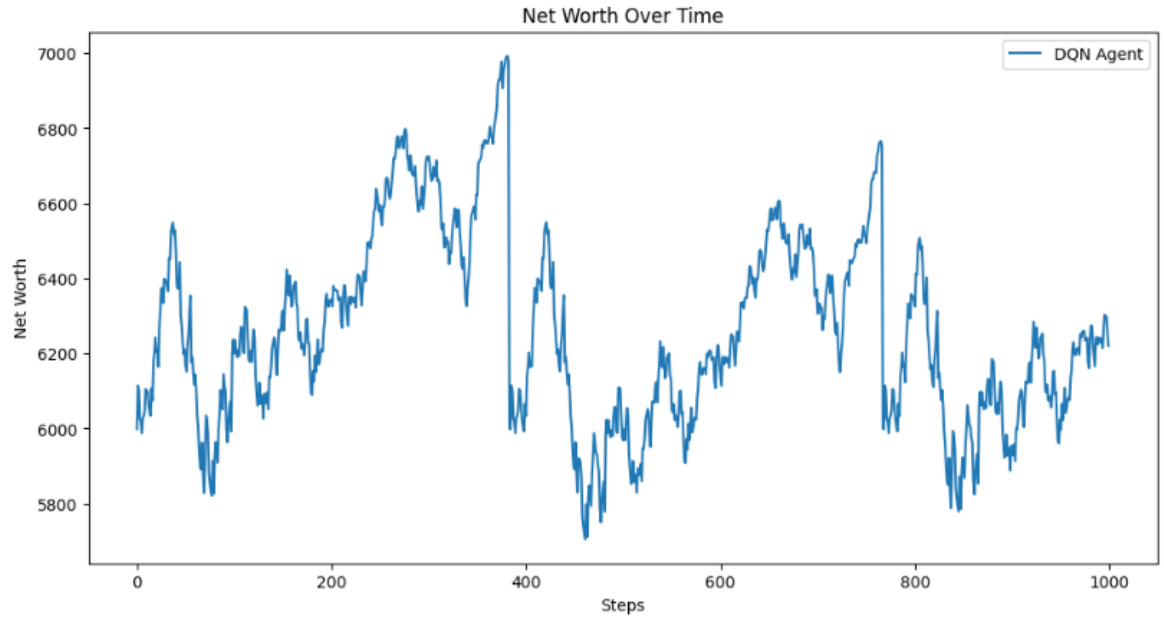


Figure 4.14 Output for PPO and SAC



	Agent	Return	Standard Deviation	Sharpe Ratio
0	DQN Agent	0.000561	0.007621	0.073644

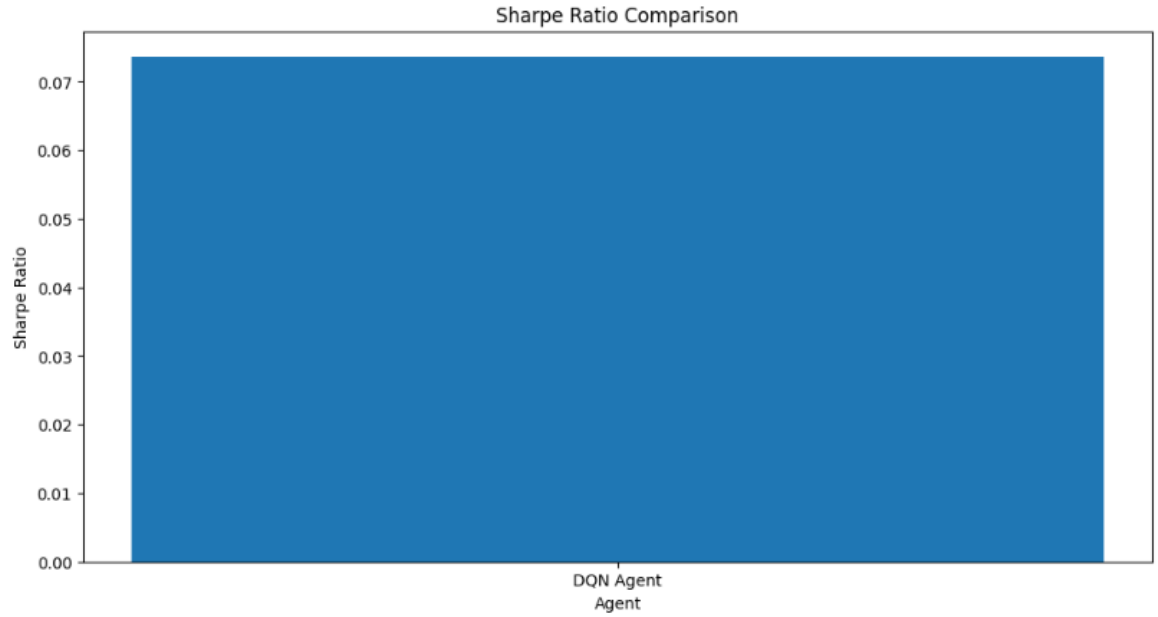


Figure 4.15 Output for DQN model

4.5 Summary

In summary, the chapter 4 will present the primary data sources which is the S&P 500 from the Yahoo Finance. The EDA involved descriptive statistics to identifying patterns, detecting anomalies, and forming hypotheses. Features engineering also added (30-day moving average and 100-day moving average) to let the agent familiar in the real-time stock market conditions.

The expected outcome and the model development structure also included in the chapter as the future results.