data_final_project

October 7, 2022

1 DATA1030 Final Project - Henry Pasts

1.1 Can you create a profitable and feasible stock market trading strategy from a Machine Learning Pipeline?

1.2 Introduction

The problem I want to solve is whether we can create a profitable stock trading strategy using machine learning.

The end goal of the project is to build a stock trading simulation on the test data that will buy or sell (depending on current positioning) a stock market index at the next open.[1] It will use the data available up to the current point of evaluation to make predictions regarding the direction and magnitude of the next price. Different trading simulation hyperparameters will be tuned to find optimal trading rules. The performance of the strategy will be compared against the 'buy/hold' of the security with aspects such as number of trades, winning trades, and losing trades considered.

1. [^](#cite_ref-1) Possible Stock Market Indices: S&P 500, Dow Jones Industrial Average, NASDAQ Compositie.

1.2.1 Target Variable:

I have 3 candidates for the target variable and plan to test all 3 of them separately: 1. The Price of the next day (continuous) 2. The percent change of the next day's price (continuous) 3. The direction of the next day's price, 0 for negative direction and 1 for positive direction (classification)

Thus, this is both a regression and classification problem, as both are relevant. We may want to continue with a trade if say the predicted price decline of the next day does not meet a certain threshold or we may not want to enter a trade if the model has conflicting results for any given day, say if our classificagion model predicts an upward price movement, but our regression model predicts a price lower than the current day or a negative percent change for the next day.

1.2.2 Importance and Interesting Aspects of the Project

We hear a lot about 'quantitative' hedge funds that use sophisticated models to trade stocks very frequently. So called 'algorithmic trading', where models can trade stocks by the second or minute and make large quantities of money are seeming out of reach to the average investor. Thus, the project begs the question whether these models can be 'democratized' to a retail investor to increase accessability and make markets more fair to more people. The project looks at end-of-day daily data, but it is designed to accomodate any time period for the timeseries data, such as minute or hourly data. It might also be applied to other asset classes such as bonds or cryptocurrencies.

1.2.3 The Data

Description of Features:

Original Features:

- Date: The date for the data (Datetime64 Object)
- Open: The Open Price (\$) for the date (float64)
- High: The High Price (\$) for the date (float64)
- Low: The Low Price (\$) for the date (float64)
- Close: The Close Price (\$) for the date (float64)
- Volume: The total dollar amount ($) traded for the date (int64)

Added Features:

- SMA 14 Open: The 14 Day Moving Average of the Open Price (float64)
- SMA 14 High: The 14 Day Moving Average of the High Price (float64)
- SMA 14 Low: The 14 Day Moving Average of the Low Price (float64)
- SMA 14 Close: The 14 Day Moving Average of the Close Price (float64)
- EMA 14 Open: The 14 Day Exponential Moving Average of the Open Price (float64)
- EMA 14 High: The 14 Day Exponential Moving Average of the High Price (float64)
- EMA 14 Low: The 14 Day Exponential Moving Average of the Low Price (float64)
- EMA 14 Close: The 14 Day Exponential Moving Average of the Close Price (float64)
- RSI 14: The 14 Day Relative Strength Index (RSI). A reading below 30 suggests oversold and above 70 suggests overbought (float64)
- CCI 20: The 20 Day Commodity Channel Index (CCI). The CCI is a momentum-based oscillator to measure oversold vs overbought conditions. (float64)
- DMI 14: The 14 Directional Movement Index (DMI). The DMI is a trend based indicator. (float64)
- Prev Open Return 1 Day: The Open Return from 1 Day ago. (open - open_1_day_ago) / open_1_day_ago (float64)
- Prev Open Return 5 Days: The Open Return from 5 Days ago. (float64)
- Prev Open Return 14 Days: The Open Return from 14 Days ago. (float64)
- Prev High Return 1 Day: The High Return from 1 Day ago. (high - high_1_day_ago) / high_1_day_ago (float64)
- Prev High Return 5 Days: The High Return from 5 Days ago. (float64)
- Prev High Return 14 Days: The High Return from 14 Days ago. (float64)
- Prev Low Return 1 Day: The Low Return from 1 Day ago. (low - low_1_day_ago) / low_1_day_ago (float64)
- Prev Low Return 5 Days: The Low Return from 5 Days ago. (float64)
- Prev Low Return 14 Days: The Low Return from 14 Days ago. (float64)
- Prev Close Return 1 Day: The Close Return from 1 Day ago. (close - close_1_day_ago) / close_1_day_ago (float64)
- Prev Close Return 5 Days: The Close Return from 5 Days ago. (float64)
- Prev Close Return 14 Days: The Close Return from 14 Days ago. (float64)

Target Variables:

- Percent Next Day: The Return of the Next Day (next_price - current_price) / current_price (float64)
- Price Next Day: The Price of the Next Day (float64)
- Direction Next Day: The Direction of the Price of the Next Day (int64)

The data was collected using Yahoo Finance's API yfinance[1] (pip install yfinance).

It was collected using the following code:

import pandas as pd
import numpy as np
import yfinance as yf

spy_ohlc_df = yf.download('SPY', start='1993-02-01', end='2022-09-30')
qqq_ohlc_df = yf.download('QQQ', start='1999-05-07', end='2022-09-30')
iyy_ohlc_df = yf.download('IYY', start='2000-06-30', end='2022-09-30')

1.2.4 Other Relevant Research:

1.[^](#cite_ref-1) Yahoo!, Y!Finance, and Yahoo! finance are registered trademarks of Yahoo, Inc. yfinance is not affiliated, endorsed, or vetted by Yahoo, Inc. It's an open-source tool that uses Yahoo's publicly available APIs, and is intended for research and educational purposes. You should refer to Yahoo!'s terms of use (here, here, and here) for details on your rights to use the actual data downloaded. Remember - the Yahoo! finance API is intended for personal use only.

[1]:
```
# all imports
import pandas as pd
import numpy as np
import yfinance as yf

import matplotlib.pyplot as plt
from ta.trend import sma_indicator, ema_indicator, CCIIndicator, ADXIndicator
from ta.momentum import RSIIndicator

import matplotlib.dates as mdates
import matplotlib as mpl
import matplotlib.gridspec as gridspec
```

1.2.5 Number of data points and number of features:

[18]:
```
# Download Data
print("Downloading SPY Price Data...")
spy_ohlc_df = yf.download('SPY', start='1993-02-01', end='2022-09-30')
spy_ohlc_df["Dates"] = spy_ohlc_df.index  # index are the current Daates, so␣
↪set df["Dates"] = index
spy_ohlc_df.reset_index(drop=True, inplace=True)  # reset the index and drop,␣
↪inplace
spy_ohlc_df = spy_ohlc_df.drop(["Adj Close"], axis=1)  # drop Adj Close - not␣
↪of interest
```

```python
print("SPY DF First 5 Rows:")
print(spy_ohlc_df.head())

print("Downloading QQQ Price Data...")
qqq_ohlc_df = yf.download('QQQ', start='1999-05-07', end='2022-09-30')
qqq_ohlc_df["Dates"] = qqq_ohlc_df.index  # index are the current Daates, so␣
↪set df["Dates"] = index
qqq_ohlc_df.reset_index(drop=True, inplace=True)  # reset the index and drop,␣
↪inplace
qqq_ohlc_df = qqq_ohlc_df.drop(["Adj Close"], axis=1)  # drop Adj Close - not␣
↪of interest

print("Downloading IYY Price Data...")
iyy_ohlc_df = yf.download('IYY', start='2000-06-30', end='2022-09-30')
iyy_ohlc_df["Dates"] = iyy_ohlc_df.index  # index are the current Daates, so␣
↪set df["Dates"] = index
iyy_ohlc_df.reset_index(drop=True, inplace=True)  # reset the index and drop,␣
↪inplace
iyy_ohlc_df = iyy_ohlc_df.drop(["Adj Close"], axis=1)  # drop Adj Close - not␣
↪of interest

# print columns and rows:
print("\n")
print("There are " + str(spy_ohlc_df.shape[0]) + " Rows and " + str(spy_ohlc_df.
↪shape[1])
      + " Cols for SPY")
print("\n")
print("There are " + str(qqq_ohlc_df.shape[0]) + " Rows and " + str(qqq_ohlc_df.
↪shape[1])
      + " Cols for QQQ")
print("\n")
print("There are " + str(iyy_ohlc_df.shape[0]) + " Rows and " + str(iyy_ohlc_df.
↪shape[1])
      + " Cols for IYY")
print("\n")
```

```
Downloading SPY Price Data…
[*********************100%***********************]  1 of 1 completed
SPY DF First 5 Rows:
       Open      High       Low     Close  Volume                     Dates
0  43.96875  44.25000  43.96875  44.25000  480500 1993-02-01 00:00:00-05:00
1  44.21875  44.37500  44.12500  44.34375  201300 1993-02-02 00:00:00-05:00
2  44.40625  44.84375  44.37500  44.81250  529400 1993-02-03 00:00:00-05:00
3  44.96875  45.09375  44.46875  45.00000  531500 1993-02-04 00:00:00-05:00
4  44.96875  45.06250  44.71875  44.96875  492100 1993-02-05 00:00:00-05:00
Downloading QQQ Price Data…
[*********************100%***********************]  1 of 1 completed
```

```
Downloading IYY Price Data…
[*********************100%***********************]  1 of 1 completed
```

```
There are 7471 Rows and 6 Cols for SPY
```

```
There are 5889 Rows and 6 Cols for QQQ
```

```
There are 5598 Rows and 6 Cols for IYY
```

1.2.6 Adding Features and Technical Analysis Indicators

Since our original dataset only has 6 features, we add standard technical analysis indicators most of which are standard at around 14 days, as well as the percent change of the current price vs the price 1, 5, and 14 days ago. I give a brief description of the formulas for these technical indicators:

1.2.7 Simple Moving Average (SMA)

Source: Investopedia

The SMA is just an average over the last n days, in our case we use 14 to get the average over the last 14 trading days.

1.2.8 Exponential Moving Average (EMA)

Source: Corporate Finance Institute

1.2.9 Relative Strength Index (RSI)

Source: Stock Charts

1.2.10 Commodity Channel Index (CCI)

Source: Stock Charts

1.2.11 Directional Movement Index (DMI)

Source: Fidelity

I will use the Python Libary 'ta' to add the technical indicators, and I will use the following function to calcuate the target variables:

[3]:
```python
def calc_target_vars(df, column="Close", period=1):
    """
    This Function Calculates the Target Variables Next Day 'Percent Next Day',
    'Price Next Day', and 'next_day_directions' for a column name and period.
```

```
    :param df: a Pandas DataFrame with the specified column
    :param column: default 'Close', the column to run target variable␣
↪calculations
    :param period: default 1, the period in the future to calculate the target␣
↪variable
    calculations
    :return: next_day_percent_change_vals, next_day_prices,␣
↪next_day_directions, which are all
    lists of the target variable values
    """
    next_day_percent_change_vals = []
    next_day_prices = []
    next_day_directions = []
    for i in range(0, len(df)):
        if i == len(df) - 1:
            break
        current_price = df[column].iloc[i]
        next_price = df[column].iloc[i + period]
        # percent change:
        percent_change_of_next_day = (next_price - current_price) /␣
↪current_price
        next_day_percent_change_vals.append(percent_change_of_next_day)
        # next day price:
        next_day_prices.append(next_price)
        # next day direction:
        if next_price > current_price:
            next_day_directions.append(1)
        else:
            next_day_directions.append(0)

    # Can't Calculate the next day target vars for final value:
    next_day_percent_change_vals.append(np.nan)
    next_day_prices.append(np.nan)
    next_day_directions.append(np.nan)

    return next_day_percent_change_vals, next_day_prices, next_day_directions
```

[19]:
```
# add indicators to SPY:

# add 'SMA 14 Open':
spy_ohlc_df['SMA 14 Open'] = sma_indicator(spy_ohlc_df["Open"], window=14)

# add 'SMA 14 High':
spy_ohlc_df['SMA 14 High'] = sma_indicator(spy_ohlc_df["High"], window=14)

# add 'SMA 14 Low':
spy_ohlc_df['SMA 14 Low'] = sma_indicator(spy_ohlc_df["Low"], window=14)
```

```
# add 'SMA 14 Close':
spy_ohlc_df['SMA 14 Close'] = sma_indicator(spy_ohlc_df["Close"], window=14)

# add 'EMA 14 Open':
spy_ohlc_df['EMA 14 Open'] = ema_indicator(spy_ohlc_df["Open"], window=14)

# add 'EMA 14 High':
spy_ohlc_df['EMA 14 High'] = ema_indicator(spy_ohlc_df["High"], window=14)

# add 'EMA 14 Low':
spy_ohlc_df['EMA 14 Low'] = ema_indicator(spy_ohlc_df["Low"], window=14)

# add 'EMA 14 Close':
spy_ohlc_df['EMA 14 Close'] = ema_indicator(spy_ohlc_df["Close"], window=14)

# add 'RSI 14':
spy_ohlc_df['RSI 14'] = RSIIndicator(spy_ohlc_df["Close"], window=14).rsi()

# add 'CCI 20':
spy_ohlc_df['CCI 20'] = CCIIndicator(high=spy_ohlc_df["High"],␣
↪low=spy_ohlc_df["Low"],
                                     close=spy_ohlc_df["Close"], window=20).
↪cci()

# add 'DMI 14':
dmi = ADXIndicator(high=spy_ohlc_df["High"], low=spy_ohlc_df["Low"],
                   close=spy_ohlc_df["Close"], window=14)
dmi_plus = dmi.adx_pos()
dmi_minus = dmi.adx_neg()
dmi_difference = dmi_plus - dmi_minus
spy_ohlc_df['DMI 14'] = dmi_difference


# add 'Prev Open Return 1 Day':
spy_ohlc_df['Prev Open Return 1 Day'] = spy_ohlc_df["Open"].pct_change(1)

# add 'Prev Open Return 5 Days':
spy_ohlc_df['Prev Open Return 5 Days'] = spy_ohlc_df["Open"].pct_change(5)

# add 'Prev Open Return 14 Days':
spy_ohlc_df['Prev Open Return 14 Days'] = spy_ohlc_df["Open"].pct_change(14)


# add 'Prev High Return 1 Day':
spy_ohlc_df['Prev High Return 1 Day'] = spy_ohlc_df["High"].pct_change(1)
```

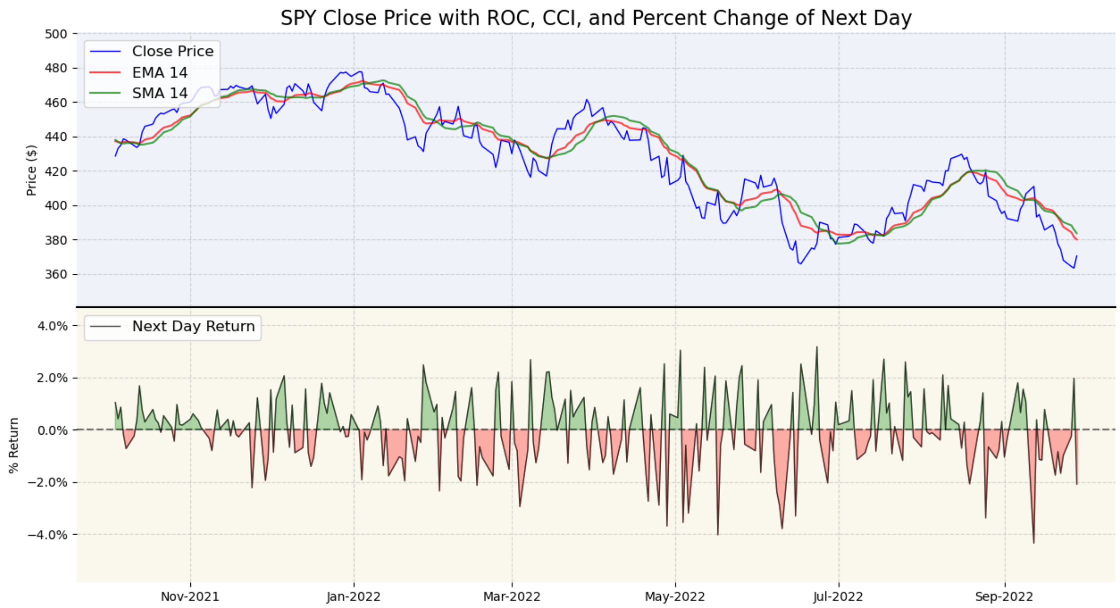
```python
# add 'Prev High Return 5 Days':
spy_ohlc_df['Prev High Return 5 Days'] = spy_ohlc_df["High"].pct_change(5)

# add 'Prev High Return 14 Days':
spy_ohlc_df['Prev High Return 14 Days'] = spy_ohlc_df["High"].pct_change(14)


# add 'Prev Low Return 1 Day':
spy_ohlc_df['Prev Low Return 1 Day'] = spy_ohlc_df["Low"].pct_change(1)

# add 'Prev Low Return 5 Days':
spy_ohlc_df['Prev Low Return 5 Days'] = spy_ohlc_df["Low"].pct_change(5)

# add 'Prev Low Return 14 Days':
spy_ohlc_df['Prev Low Return 14 Days'] = spy_ohlc_df["Low"].pct_change(14)


# add 'Prev Close Return 1 Day':
spy_ohlc_df['Prev Close Return 1 Day'] = spy_ohlc_df["Close"].pct_change(1)

# add 'Prev Close Return 5 Days':
spy_ohlc_df['Prev Close Return 5 Days'] = spy_ohlc_df["Close"].pct_change(5)

# add 'Prev Close Return 14 Days':
spy_ohlc_df['Prev Close Return 14 Days'] = spy_ohlc_df["Close"].pct_change(14)

# calculate the target variables:
percent_next_day, price_next_day, direction_next_day = 
 ↪calc_target_vars(spy_ohlc_df)

# add 'Percent Next Day':
spy_ohlc_df['Percent Next Day'] = percent_next_day

# add 'Price Next Day':
spy_ohlc_df['Price Next Day'] = price_next_day

# add 'Direction Next Day':
spy_ohlc_df['Direction Next Day'] = direction_next_day

# we added 26 columns (23 features, 3 target variables)
assert spy_ohlc_df.shape[1] - 6 == 26
print("Success.")
```

```
Success.
```

1.2.12 Why Might Technical Analysis and other measures of Price Trend and Momentum help us predict the magnitude of the next day's price and its direction?

[48]:
```
# plot Close, RSI, CCI, Percent Change Next Day
indicator_df = spy_ohlc_df[["Dates", "Close", "EMA 14 Close", "SMA 14 Close",␣
 ↪"Percent Next Day"]]
indicator_df = indicator_df.iloc[-250:-1]

fig = plt.figure(figsize=(15,8))
fig.subplots_adjust(hspace=0)

# Add Close:
close_ax = plt.subplot(2, 1, 1)
close_ax.plot(indicator_df["Dates"], indicator_df["Close"], color='blue',␣
 ↪linewidth=1, label="Close Price")
close_ax.set_ylabel("Price ($)")
close_ax.set_title("SPY Close Price with ROC, CCI, and Percent Change of Next␣
 ↪Day", fontsize=16)

# Add EMA 14 Close:
close_ax.plot(indicator_df["Dates"], indicator_df["EMA 14 Close"], color='r',␣
 ↪linewidth = 1.5, alpha=0.7, label="EMA 14")
close_ax.plot(indicator_df["Dates"], indicator_df["SMA 14 Close"], color='g',␣
 ↪linewidth = 1.5, alpha=0.7, label="SMA 14")
close_ax.legend(loc="upper left", fontsize=12)
close_ax.xaxis.set_major_formatter(mdates.DateFormatter('%b-%Y'))  # format␣
 ↪dates

# Add next day return:
return_ax = plt.subplot(2, 1, 2, sharex=close_ax)
return_ax.plot(indicator_df["Dates"], indicator_df["Percent Next Day"] * 100,␣
 ↪color='k', linewidth = 1, alpha=0.7,
               label="Next Day Return")
return_ax.legend(loc="upper left", fontsize=12)
return_ax.set_ylabel("% Return")
return_ax.yaxis.set_major_formatter(mpl.ticker.PercentFormatter())

return_ax.axhline(0, color="black", linestyle = '--', alpha = 0.5)  # add a␣
 ↪horizontal line at 0

return_ax.fill_between(indicator_df["Dates"], 0, indicator_df["Percent Next␣
 ↪Day"] * 100,
                    where=(indicator_df["Percent Next Day"] * 100 <= 0),
                    color='r', alpha=0.3, interpolate=True)  # fill area below␣
 ↪red
```

```
return_ax.fill_between(indicator_df["Dates"], 0, indicator_df["Percent Next␣
 ↪Day"] * 100,
                    where=(indicator_df["Percent Next Day"] * 100 > 0),
                    color='g', alpha=0.3, interpolate=True)  # fill area above␣
 ↪green

return_ax.xaxis.set_major_formatter(mdates.DateFormatter('%b-%Y'))  # format␣
 ↪dates

# add grids to close_ax and return_ax
close_ax.grid(visible=True, linestyle='--', alpha=0.5)
return_ax.grid(visible=True, linestyle='--', alpha=0.5)

# Set facecolor (background) of plots
close_ax.set_facecolor((.94,.95,.98))
return_ax.set_facecolor((.98,.97,.93))

# Add margins around plots
close_ax.margins(0.04, 0.2)
return_ax.margins(0.04, 0.2)

# # Hiding the tick marks from the horizontal and vertical axis:
# close_ax.tick_params(left=False, bottom=False)
# rsi_ax.tick_params(left=False, bottom=False, labelrotation=45)

# Hiding all the spines for the price subplot:
for s in close_ax.spines.values():
    s.set_visible(False)

# Hiding all the spines for the Next Day Return subplot:
for s in return_ax.spines.values():
    s.set_visible(False)

# Reinstate top spine to divide subplots:

return_ax.spines['top'].set_visible(True)
return_ax.spines['top'].set_linewidth(1.5)


# code modified from https://towardsdatascience.com/
 ↪trading-toolbox-04-subplots-f6c353278f78
```



1.2.13 Indicator Optimization - Correlation to Target Variable (train data only ~ 60% of data)

[5]:
```
from scipy.stats import pearsonr

# best_sma = None
# best_corr = -np.inf
# # optimize SMA:
# for i in range(2, 251):
#     spy_ohlc_df['current sma'] = sma_indicator(spy_ohlc_df["Close"], window=i)
#     price_over_ma = list(np.abs(spy_ohlc_df["Close"].iloc[i:-1] -
↪spy_ohlc_df['current sma'].iloc[i:-1]))
#     # price_over_ma = [1 if val > 0 else 0 for val in price_over_ma]
#     next_day_returns_list = list(spy_ohlc_df['Direction Next Day'].iloc[i:-1])
#     result = pearsonr(price_over_ma, next_day_returns_list)
#     corr = result.statistic
#     # print("Corr for SMA " + str(i) + " is " + str(corr))
#     if corr > best_corr:
#         best_corr = corr
#         best_sma = i
# print(best_sma)
# print(best_corr)

# best_cci = None
# best_corr = -np.inf
# # optimize SMA:
# for i in range(3, 251):
```

```python
#     spy_ohlc_df['current cci'] = CCIIndicator(high=spy_ohlc_df["High"],␣
↪low=spy_ohlc_df["Low"],
#                                           close=spy_ohlc_df["Close"], window=i).
↪cci()
#     cci = list(spy_ohlc_df['current cci'].iloc[i:-1])
#     next_day_returns_list = list(spy_ohlc_df['Percent Next Day'].iloc[i:-1])
#     result = pearsonr(cci, next_day_returns_list)
#     corr = result.statistic
#     if corr > best_corr:
#         best_corr = corr
#         best_cci = i
# print(best_cci)
# print(best_corr)

# best_rsi = None
# best_corr = -np.inf
# # optimize SMA:
# for i in range(3, 251):
#     spy_ohlc_df['current rsi'] = RSIIndicator(spy_ohlc_df["Close"], window=i).
↪rsi()
#     rsi = list(spy_ohlc_df['current rsi'].iloc[i:-1])
#     next_day_returns_list = list(spy_ohlc_df['Percent Next Day'].iloc[i:-1])
#     result = pearsonr(rsi, next_day_returns_list)
#     corr = result.statistic
#     if corr > best_corr:
#         best_corr = corr
#         best_rsi = i
# print(best_rsi)
# print(best_corr)

best_dmi = None
best_corr = -np.inf
# optimize DMI:
for i in range(3, 251):
    dmi = ADXIndicator(high=spy_ohlc_df["High"], low=spy_ohlc_df["Low"],
                       close=spy_ohlc_df["Close"], window=i)
    dmi_plus = dmi.adx_pos()
    dmi_minus = dmi.adx_neg()
    dmi_difference = dmi_plus - dmi_minus
    spy_ohlc_df['DMI i'] = dmi_difference
    dmi = list(spy_ohlc_df['DMI i'].iloc[i:-1])
    next_day_returns_list = list(spy_ohlc_df['Direction Next Day'].iloc[i:-1])
    result = pearsonr(dmi, next_day_returns_list)
    corr = result.statistic
    if corr > best_corr:
        best_corr = corr
        best_dmi = i
```

```
print(best_dmi)
print(best_corr)


# ystd = list(spy_ohlc_df["Prev Close Return 1 Day"].iloc[1:-1])
# ystd = [1 if val > 0 else 0 for val in ystd]
# next_day_returns_list = list(spy_ohlc_df['Percent Next Day'].iloc[1:-1])
# corr, p = spearmanr(ystd, next_day_returns_list)
# print(corr, p)

# next_day_dir_list = list(spy_ohlc_df['Direction Next Day'].iloc[1:-1])
# corr, p = spearmanr(ystd, next_day_dir_list)
# print(corr, p)
# dates = spy_ohlc_df["Dates"]
# spy_ohlc_df['current sma'] = ema_indicator(spy_ohlc_df["Close"], window=4188)

# plt.plot(dates, spy_ohlc_df['current sma'])
# plt.plot(dates, spy_ohlc_df['Close'])


# Indicator val of whether Price is above the 222 day SMA has a correlation to
↪the next day's return of 0.0055
# Indicator val of whether Price is above the 222 day SMA has a correlation to
↪the next day's direction of 0.0177
# 249 EMA corr by 0.48 to next day Price
# 249 SMA by 0.415

# squared difference between close and 3 sma has 0.07 corr to next day price
# squared difference between close and 4 ema has 0.07 corr to next day price
```

```
178
0.00558377995404087
```

1.3 Exploratory Data Analysis

1.3.1 General Code to Describe each Column (Code Commented Out)

[6]:
```
for col_name in spy_ohlc_df.columns:
    if col_name == "Dates":
        continue
    print(spy_ohlc_df[col_name].describe())
    print("\n")
```

```
count    7471.000000
mean      160.051747
std        95.540873
min        43.343750
25%       103.122501
```

```
50%       129.990005
75%       203.084999
max       479.220001
Name: Open, dtype: float64


count    7471.000000
mean      161.010462
std        96.041813
min        43.531250
25%       104.047813
50%       130.869995
75%       204.129997
max       479.980011
Name: High, dtype: float64


count    7471.000000
mean      158.982930
std        94.975773
min        42.812500
25%       102.320625
50%       129.190002
75%       201.919998
max       476.059998
Name: Low, dtype: float64


count    7471.000000
mean      160.050293
std        95.545135
min        43.406250
25%       103.344997
50%       130.020004
75%       203.205002
max       477.709991
Name: Close, dtype: float64


count    7.471000e+03
mean     8.454017e+07
std      9.414462e+07
min      5.200000e+03
25%      8.874250e+06
50%      6.101790e+07
75%      1.188418e+08
max      8.710263e+08
Name: Volume, dtype: float64
```

```
count    7458.000000
mean      159.965966
std        95.306109
min        44.136161
25%       102.988281
50%       130.139643
75%       203.306429
max       472.897147
Name: SMA 14 Open, dtype: float64


count    7458.000000
mean      160.923711
std        95.812305
min        44.310268
25%       104.285335
50%       130.823572
75%       204.463215
max       474.860003
Name: SMA 14 High, dtype: float64


count    7458.000000
mean      158.899657
std        94.741769
min        43.877232
25%       102.095000
50%       129.404645
75%       202.159106
max       469.882145
Name: SMA 14 Low, dtype: float64


count    7458.000000
mean      159.965797
std        95.312316
min        44.104911
25%       103.227522
50%       130.119063
75%       203.491607
max       472.662857
Name: SMA 14 Close, dtype: float64


count    7458.000000
mean      159.961512
```

```
std        95.273303
min        44.052407
25%       103.420789
50%       130.218230
75%       203.177684
max       472.742475
Name: EMA 14 Open, dtype: float64
```

```
count    7458.000000
mean      160.919230
std        95.781314
min        44.215226
25%       104.489520
50%       130.868389
75%       204.414687
max       474.693514
Name: EMA 14 High, dtype: float64
```

```
count    7458.000000
mean      158.895040
std        94.706634
min        43.787801
25%       102.251407
50%       129.460520
75%       201.951551
max       469.787593
Name: EMA 14 Low, dtype: float64
```

```
count    7458.000000
mean      159.961154
std        95.279139
min        44.021478
25%       103.620883
50%       130.189675
75%       203.251246
max       472.415930
Name: EMA 14 Close, dtype: float64
```

```
count    7458.000000
mean       54.254178
std        11.177028
min        16.700889
25%        46.279879
50%        54.895537
```

```
75%         62.482791
max         87.030929
Name: RSI 14, dtype: float64
```

```
count    7452.000000
mean       25.022150
std       108.056475
min      -424.095285
25%       -57.110937
50%        49.983152
75%       108.670230
max       310.249307
Name: CCI 20, dtype: float64
```

```
count    7471.000000
mean        0.432695
std        13.255038
min       -56.250845
25%        -8.822050
50%         1.254151
75%        10.258261
max        40.371338
Name: DMI 14, dtype: float64
```

```
count    7470.000000
mean        0.000354
std         0.011790
min        -0.129440
25%        -0.004776
50%         0.000844
75%         0.005964
max         0.115372
Name: Prev Open Return 1 Day, dtype: float64
```

```
count    7466.000000
mean        0.001704
std         0.023935
min        -0.231260
25%        -0.009769
50%         0.003139
75%         0.014540
max         0.158198
Name: Prev Open Return 5 Days, dtype: float64
```

```
count    7457.000000
mean        0.004786
std         0.037959
min        -0.302853
25%        -0.013297
50%         0.008757
75%         0.025641
max         0.214514
Name: Prev Open Return 14 Days, dtype: float64


count    7470.000000
mean        0.000327
std         0.009366
min        -0.073249
25%        -0.003595
50%         0.000602
75%         0.004670
max         0.078880
Name: Prev High Return 1 Day, dtype: float64


count    7466.000000
mean        0.001651
std         0.021581
min        -0.186314
25%        -0.008951
50%         0.003319
75%         0.013441
max         0.142590
Name: Prev High Return 5 Days, dtype: float64


count    7457.000000
mean        0.004693
std         0.035353
min        -0.268162
25%        -0.012560
50%         0.008429
75%         0.024803
max         0.208246
Name: Prev High Return 14 Days, dtype: float64


count    7470.000000
mean        0.000346
std         0.011331
```

```
min        -0.094593
25%        -0.004476
50%         0.000729
75%         0.005606
max         0.089554
Name: Prev Low Return 1 Day, dtype: float64


count    7466.000000
mean        0.001736
std         0.025497
min        -0.237965
25%        -0.010380
50%         0.003295
75%         0.015143
max         0.178947
Name: Prev Low Return 5 Days, dtype: float64


count    7457.000000
mean        0.004854
std         0.039887
min        -0.305583
25%        -0.013971
50%         0.009351
75%         0.026769
max         0.243517
Name: Prev Low Return 14 Days, dtype: float64


count    7470.000000
mean        0.000353
std         0.011911
min        -0.109424
25%        -0.004582
50%         0.000605
75%         0.005894
max         0.145198
Name: Prev Close Return 1 Day, dtype: float64


count    7466.000000
mean        0.001705
std         0.024206
min        -0.197934
25%        -0.009743
50%         0.003272
75%         0.014691
```

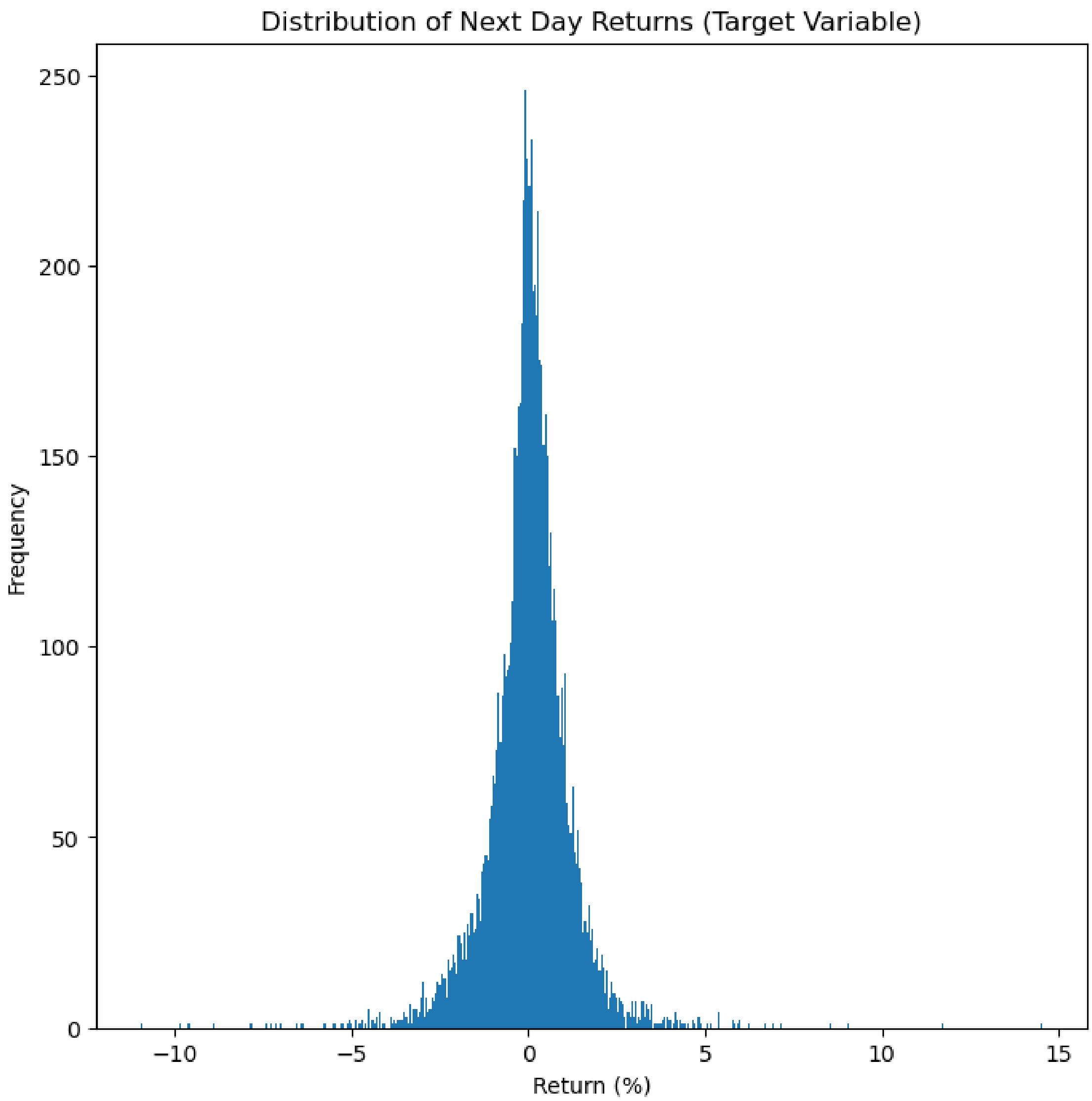
```
max         0.194036
Name: Prev Close Return 5 Days, dtype: float64


count    7457.000000
mean        0.004784
std         0.038064
min        -0.270464
25%        -0.012893
50%         0.008535
75%         0.025821
max         0.236421
Name: Prev Close Return 14 Days, dtype: float64


count    7470.000000
mean        0.000353
std         0.011911
min        -0.109424
25%        -0.004582
50%         0.000605
75%         0.005894
max         0.145198
Name: Percent Next Day, dtype: float64


count    7470.000000
mean      160.065795
std        95.542134
min        43.406250
25%       103.352499
50%       130.025002
75%       203.207504
max       477.709991
Name: Price Next Day, dtype: float64


count    7470.000000
mean        0.534003
std         0.498876
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: Direction Next Day, dtype: float64
```

```
count    7471.000000
mean       -1.316719
std         3.435053
min       -11.721975
25%        -3.646353
50%        -0.875526
75%         1.113563
max         9.214558
Name: DMI i, dtype: float64
```

1.3.2 Visualize Target Variables

[49]:

```
plt.figure(figsize=(8,8))

spy_target_var_df = spy_ohlc_df[["Dates", "Close", "Percent Next Day", "Price␣
↪Next Day", "Direction Next Day",
                                "Prev Close Return 1 Day", "Prev Close Return 5␣
↪Days", "Prev Close Return 14 Days"]].copy()

# Create a histogram of the Percent Next Day:
plt.title("Distribution of Next Day Returns (Target Variable)")
spy_target_var_df["Future Gain"] = spy_target_var_df["Percent Next Day"] * 100
spy_target_var_df["Future Gain"].plot(kind="hist", bins=500)
plt.xlabel("Return (%)")
plt.ylabel("Frequency")
plt.show()
print("Summary Stats For Next Day Returns:")
print("\n")
print(spy_target_var_df["Future Gain"].describe())
```



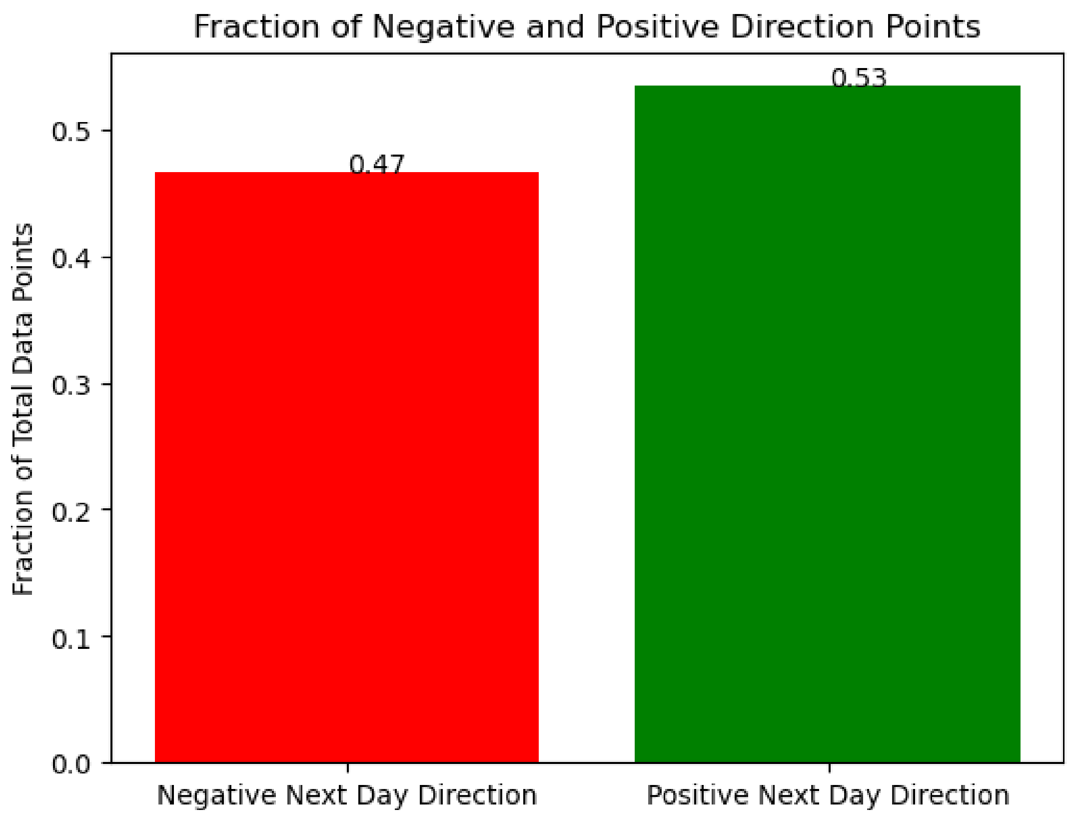

```
Summary Stats For Next Day Returns:


count    7470.000000
mean        0.035270
std         1.191059
min       -10.942374
25%        -0.458201
50%         0.060485
75%         0.589371
max        14.519772
Name: Future Gain, dtype: float64
```

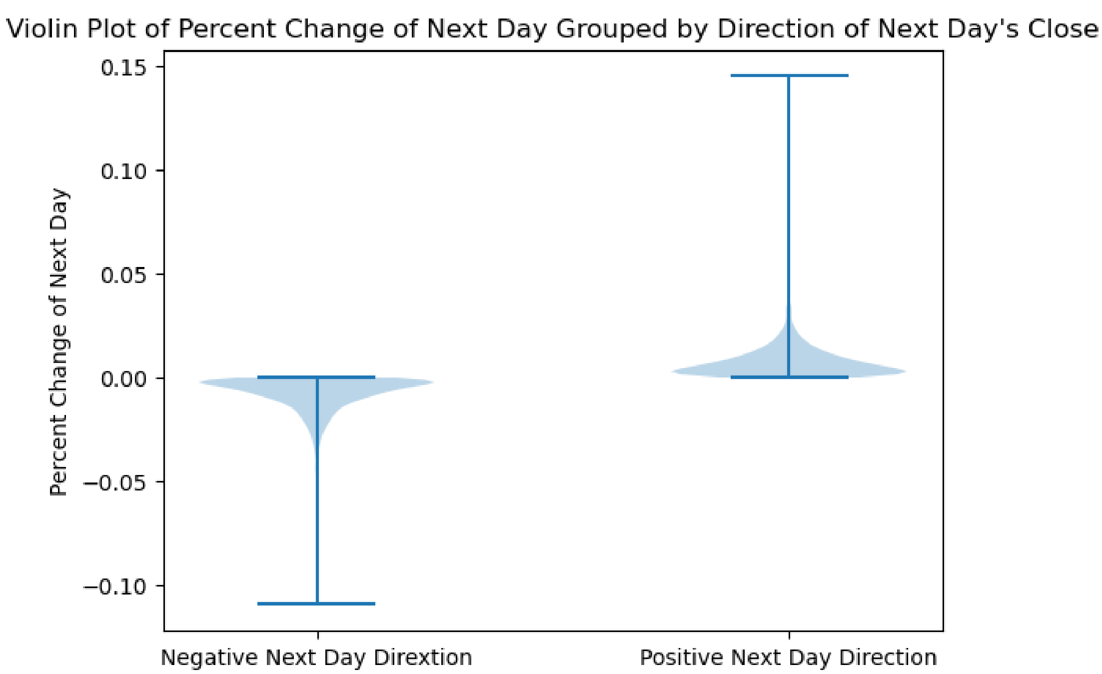
[8]:
```
# Plot Close vs Close Next Day:
fig, ax = plt.subplots()
ax.plot(spy_target_var_df["Dates"].iloc[-50:-2], spy_target_var_df["Close"].
 ↪iloc[-50:-2], color='black', linewidth=0.7,
       label="Close")
ax.plot(spy_target_var_df["Dates"].iloc[-50:-2], spy_target_var_df["Price Next␣
 ↪Day"].iloc[-50:-2], color='red',
        linewidth=0.7, label="Next Close Price")
plt.xlabel("Date")
plt.ylabel("Price")
plt.title("Close vs Next Close Price")
plt.legend(loc=0)  # best loc=0
plt.show()
print("The goal of the project is for a date d to predict the Next Close Price␣
 ↪(red line) given the Close (black line)")
print("and previous values and indicators of trend up to date d.")
```


```
The goal of the project is for a date d to predict the Next Close Price (red
line) given the Close (black line)
and previous values and indicators of trend up to date d.
```

[9]:

```
# Plot balance of Positve and Negative Directions:
direction_df = spy_target_var_df.iloc[0:-1]
num_neg_frac = len(direction_df[direction_df["Direction Next Day"] == 0]) /␣
 ↪len(direction_df)
num_pos_frac = len(direction_df[direction_df["Direction Next Day"] == 1]) /␣
 ↪len(direction_df)
fracs = [num_neg_frac, num_pos_frac]

fig, ax = plt.subplots()
ax.bar(["Negative Next Day Direction", "Positive Next Day Direction"], fracs,␣
 ↪color=["red", "green"])
plt.title("Fraction of Negative and Positive Direction Points")
plt.ylabel("Fraction of Total Data Points")
ax.text(0, num_neg_frac, s=str(round(num_neg_frac, 2)))
ax.text(1, num_pos_frac, s=str(round(num_pos_frac, 2)))
plt.show()

# Plot a Violin plot of the balance of positive and negative directional days␣
 ↪of Direction Next Day:
dataset = [spy_target_var_df[spy_target_var_df['Direction Next Day'] ==␣
 ↪0]['Percent Next Day'].values,
            spy_target_var_df[spy_target_var_df['Direction Next Day'] ==␣
 ↪1]['Percent Next Day'].values]

plt.violinplot(dataset = dataset)
plt.title("Violin Plot of Percent Change of Next Day Grouped by Direction of␣
 ↪Next Day's Close")
plt.xticks([1,2],['Negative Next Day Dirextion','Positive Next Day Direction'])
plt.ylabel("Percent Change of Next Day")
plt.show()
```

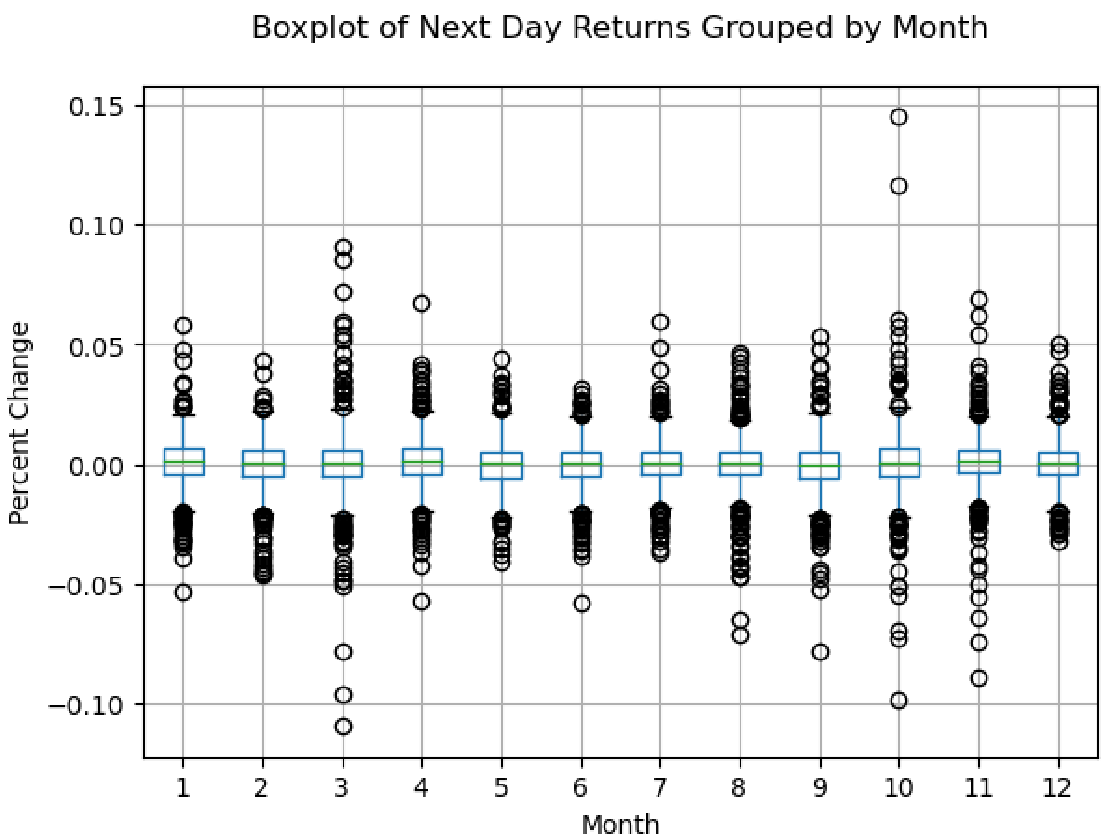

Fraction of Negative and Positive Direction Points
0.47
0.53
Fraction of Total Data Points
0.0
0.1
0.2
0.3
0.4
0.5
Negative Next Day Direction
Positive Next Day Direction

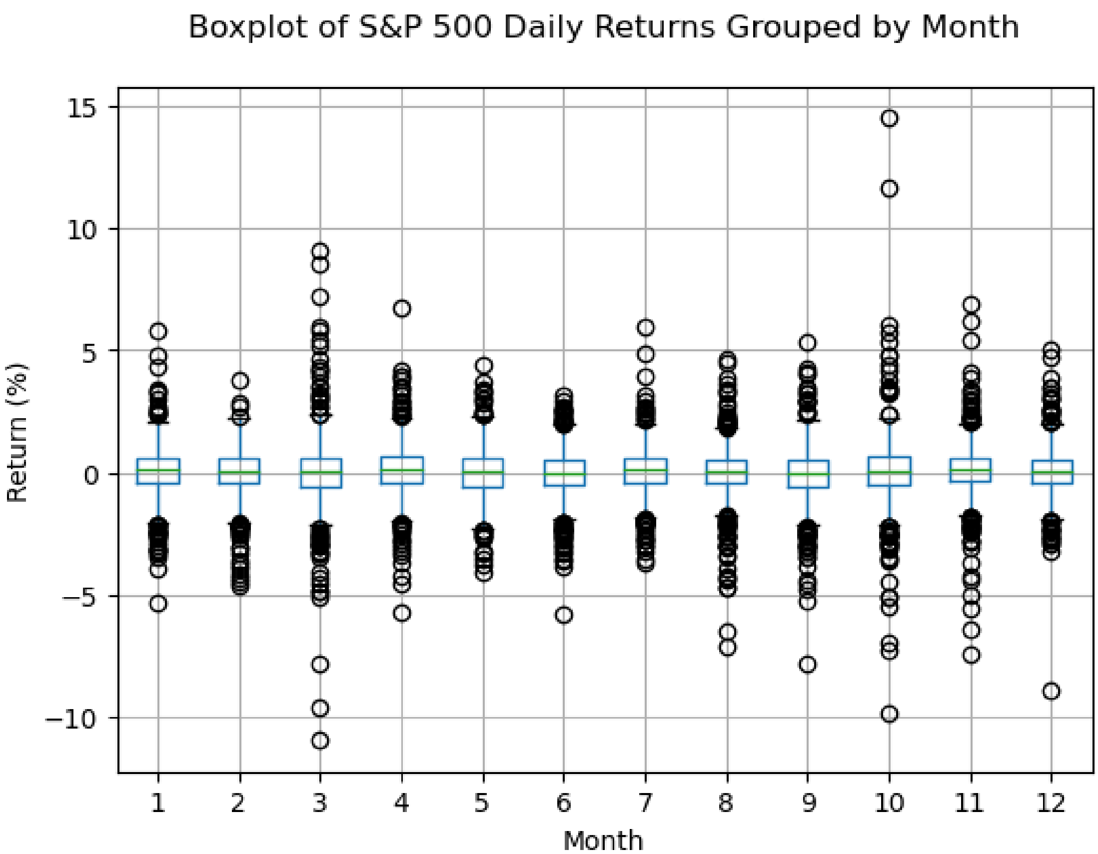

Violin Plot of Percent Change of Next Day Grouped by Direction of Next Day's Close
Percent Change of Next Day
0.15
0.10
0.05
0.00
−0.05
−0.10
Negative Next Day Dirextion
Positive Next Day Direction

[39]:

```python
# plot seasonality concerns: the distribution of percent changes by month
import datetime

date_df = spy_target_var_df.iloc[0:-1].copy()
months = [date.month for date in list(date_df["Dates"])]
date_df["month"] = months

date_df[['Percent Next Day', 'month']].boxplot(by='month')
plt.title("")
plt.suptitle("")
plt.suptitle("Boxplot of Next Day Returns Grouped by Month")
plt.ylabel('Percent Change')
plt.xlabel("Month")
plt.show()

date_df = spy_target_var_df.iloc[0:-1].copy()
date_df['Prev Close Return 1 Day'] = date_df['Prev Close Return 1 Day'] * 100
months = [date.month for date in list(date_df["Dates"])]
date_df["month"] = months
date_df[['Prev Close Return 1 Day', 'month']].boxplot(by='month')
plt.title("")
plt.suptitle("")
plt.suptitle("Boxplot of S&P 500 Daily Returns Grouped by Month")
plt.ylabel('Return (%)')
plt.xlabel("Month")
plt.show()


# plot seasonality concerns: the distribution of percent changes by month

# date_df = spy_target_var_df.iloc[0:-1].copy()
# months = [datetime.datetime.strptime(str(date.month), "%m").strftime('%b')␣
↪for date in list(date_df["Dates"])]
# date_df["month"] = months

# date_df[['Prev Close Return 1 Day', 'month']].boxplot(by='month')
# plt.title("")
# plt.suptitle("")
# plt.suptitle("Boxplot of Daily Returns Grouped by Month")
# plt.ylabel('Return')
# plt.xlabel("Month")
# plt.show()



# fig = plt.figure(figsize=(10,10))
```

```
# date_df = spy_target_var_df.iloc[0:-1].copy()
# months = [date.month for date in list(date_df["Dates"])]
# date_df["month"] = months

# october_df = date_df[date_df["month"] == 10]
# days = [date.day for date in list(october_df["Dates"])]
# october_df["Day"] = days


# october_df[['Prev Close Return 1 Day', 'Day']].boxplot(by='Day')
# plt.title("")
# plt.suptitle("")
# plt.suptitle("Boxplot of Returns Grouped by Day in October")
# plt.ylabel('Return')
# plt.xlabel("Days in October", labelpad=12)



# plt.show()
```



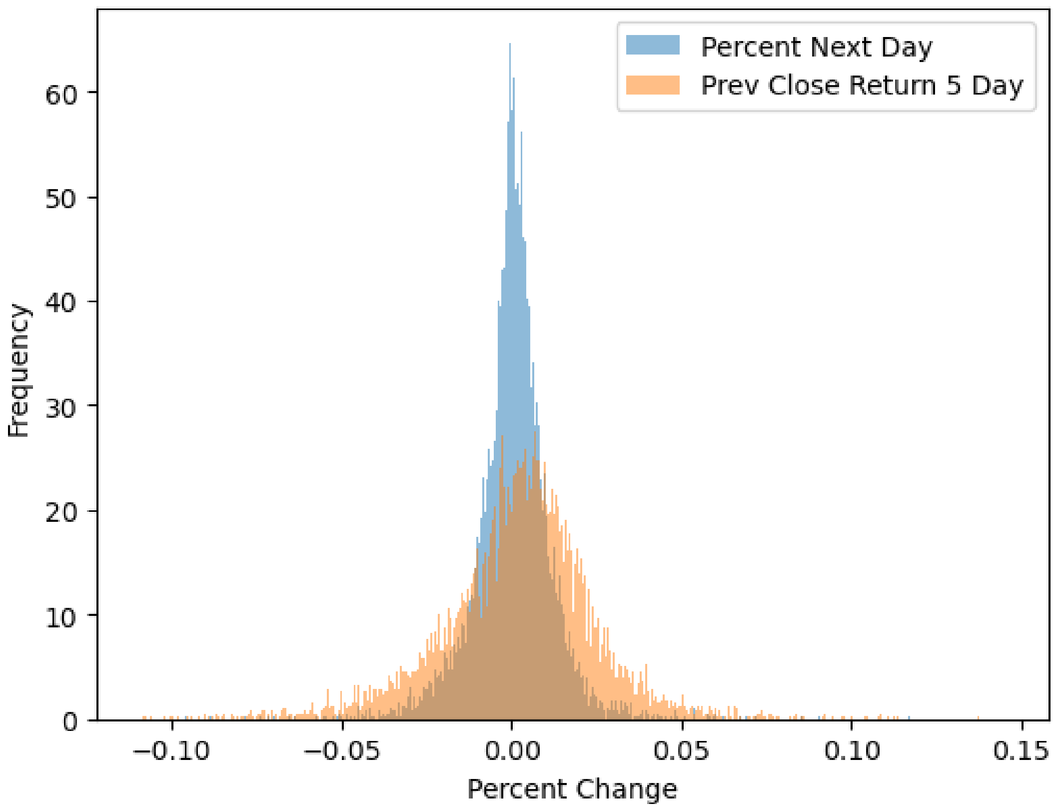


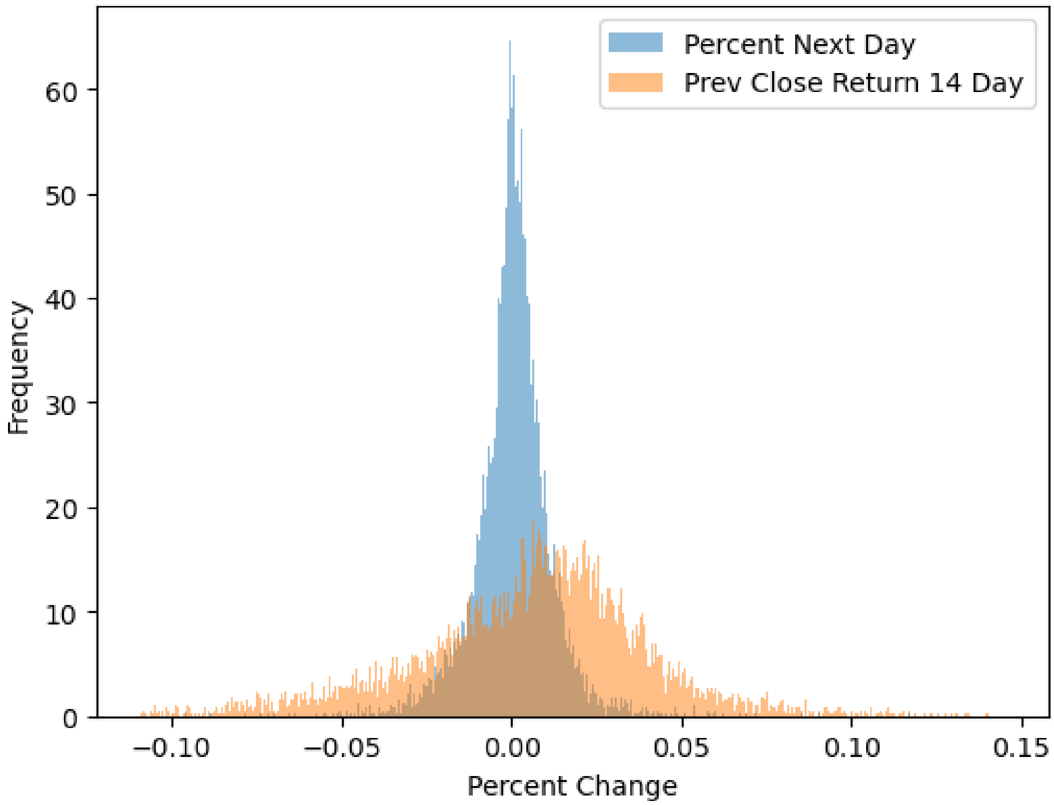
1.3.3 Compare the distributions of previous day returns and the target variable

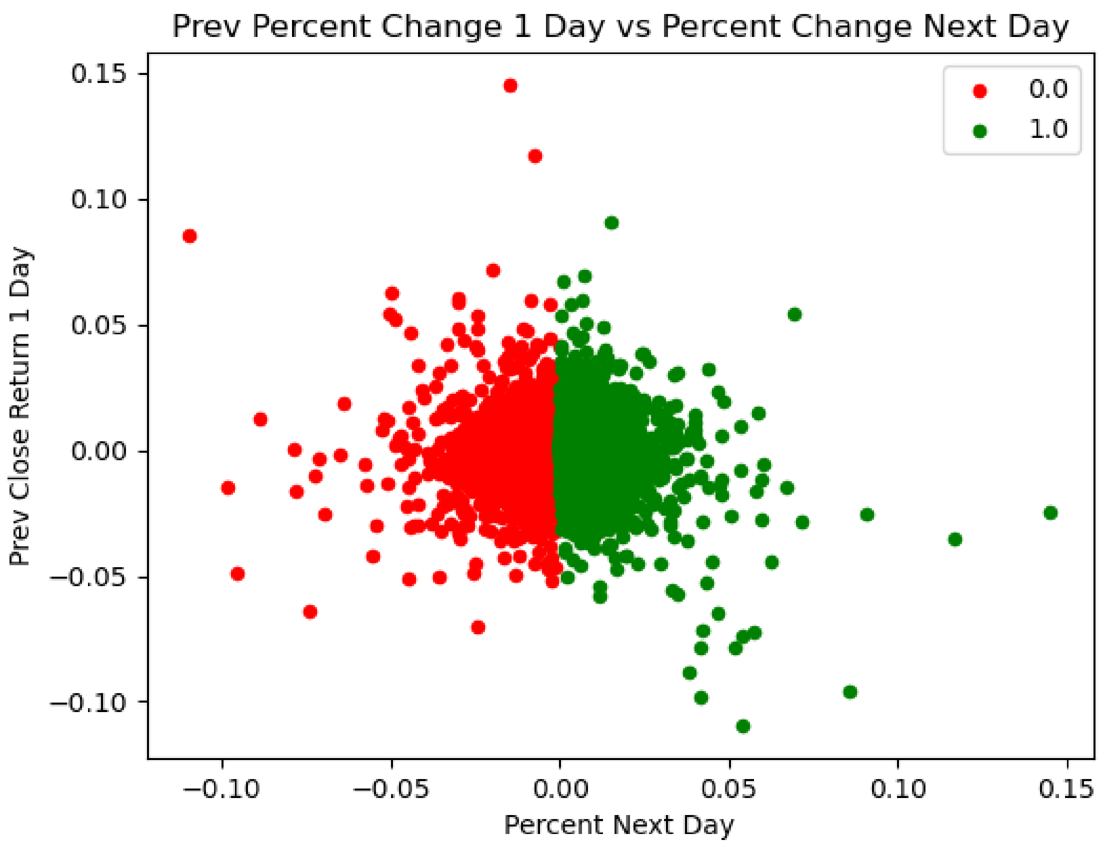
[11]:
```
# get bin range:
min1 = spy_target_var_df["Percent Next Day"].min()
min2 = spy_target_var_df["Prev Close Return 1 Day"].min()
if min1 > min2:
    range_min = min2
else:
    range_min = min1
max1 = spy_target_var_df["Percent Next Day"].max()
max2 = spy_target_var_df["Prev Close Return 1 Day"].max()
if max1 > max2:
    range_max = max1
else:
    range_max = max2
bin_range = (range_min, range_max)

# plot histograms:
```

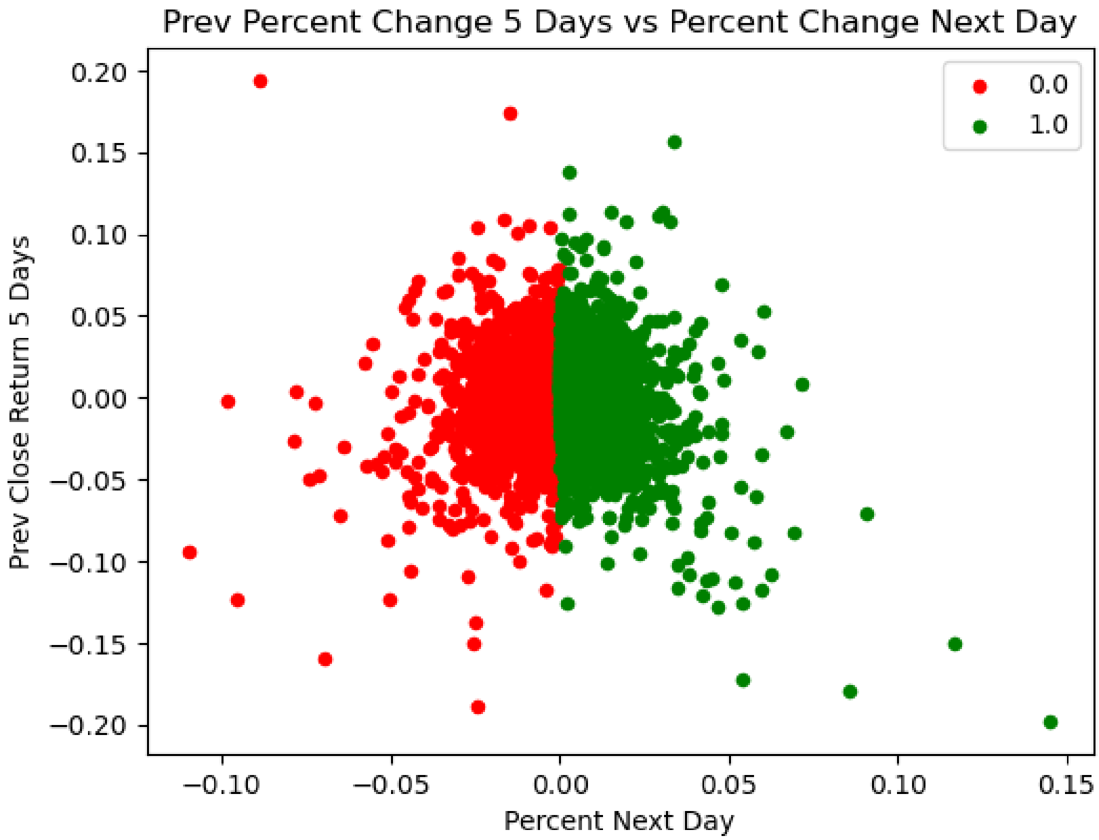
```
plt.hist(spy_target_var_df["Percent Next Day"].iloc[0:-1], alpha=0.5,␣
 ↪label="Percent Next Day",
        range=bin_range, bins=500, density=True)
plt.hist(spy_target_var_df["Prev Close Return 5 Days"].iloc[14:], alpha=0.5,␣
 ↪label="Prev Close Return 5 Day",
        range=bin_range, bins=500, density=True)
plt.legend()
plt.ylabel('Frequency')
plt.xlabel('Percent Change')
plt.show()
plt.hist(spy_target_var_df["Percent Next Day"].iloc[0:-1], alpha=0.5,␣
 ↪label="Percent Next Day",
        range=bin_range, bins=500, density=True)
plt.hist(spy_target_var_df["Prev Close Return 14 Days"].iloc[14:], alpha=0.5,␣
 ↪label="Prev Close Return 14 Day",
        range=bin_range, bins=500, density=True)
plt.legend()
plt.ylabel('Frequency')
plt.xlabel('Percent Change')
plt.show()

# Scatter Plot of Percent Change from 1 day ago vs Percent Change of Next Day:
fig, ax = plt.subplots()
colors = {0: "red", 1: "green"}
group = spy_target_var_df.groupby('Direction Next Day')
for key, group in group:
    group.plot(ax=ax, kind='scatter', x='Percent Next Day', y='Prev Close␣
 ↪Return 1 Day', label=key, color=colors[key])
plt.xlabel("Percent Next Day")
plt.ylabel("Prev Close Return 1 Day")
plt.title("Prev Percent Change 1 Day vs Percent Change Next Day")
plt.show()

# Scatter Plot of Percent Change from 5 days ago vs Percent Change of Next Day:
fig, ax = plt.subplots()
colors = {0: "red", 1: "green"}
group = spy_target_var_df.groupby('Direction Next Day')
for key, group in group:
    group.plot(ax=ax, kind='scatter', x='Percent Next Day', y='Prev Close␣
 ↪Return 5 Days', label=key, color=colors[key])
plt.xlabel("Percent Next Day")
plt.ylabel("Prev Close Return 5 Days")
plt.title("Prev Percent Change 5 Days vs Percent Change Next Day")
plt.show()

# Scatter Plot of Percent Change from 14 days ago vs Percent Change of Next Day:
```

```
fig, ax = plt.subplots()
colors = {0: "red", 1: "green"}
group = spy_target_var_df.groupby('Direction Next Day')
for key, group in group:
    group.plot(ax=ax, kind='scatter', x='Percent Next Day', y='Prev Close␣
↪Return 14 Days', label=key, color=colors[key])
plt.xlabel("Percent Next Day")
plt.ylabel("Prev Close Return 14 Day")
plt.title("Prev Percent Change 14 Days vs Percent Change Next Day")
plt.show()
```

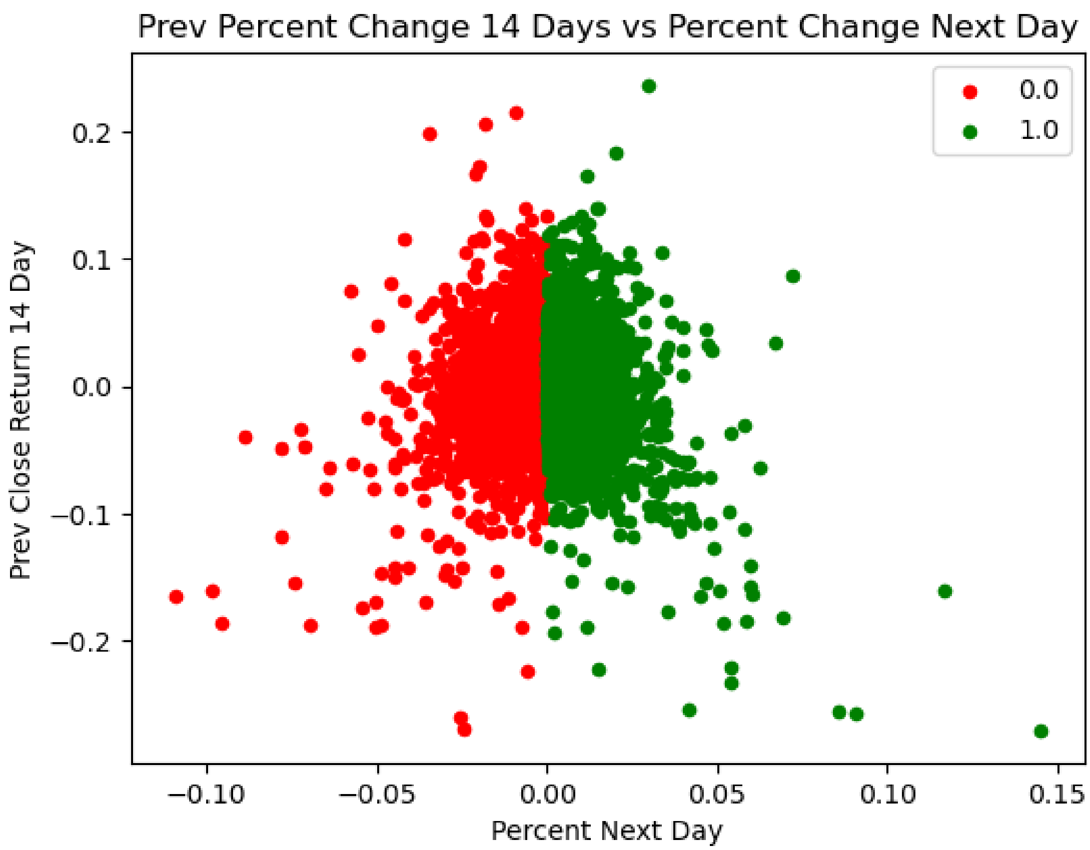



Percent Next Day
Prev Close Return 14 Day
Frequency
Percent Change
0
10
20
30
40
50
60
−0.10
−0.05
0.00
0.05
0.10
0.15


Prev Percent Change 1 Day vs Percent Change Next Day
0.0
1.0
0.15
0.10
0.05
0.00
−0.05
−0.10
Prev Close Return 1 Day
−0.10
−0.05
0.00
0.05
0.10
0.15
Percent Next Day



Prev Percent Change 5 Days vs Percent Change Next Day
0.0
1.0
Prev Close Return 5 Days
Percent Next Day
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.10
−0.05
0.00
0.05
0.10
0.15



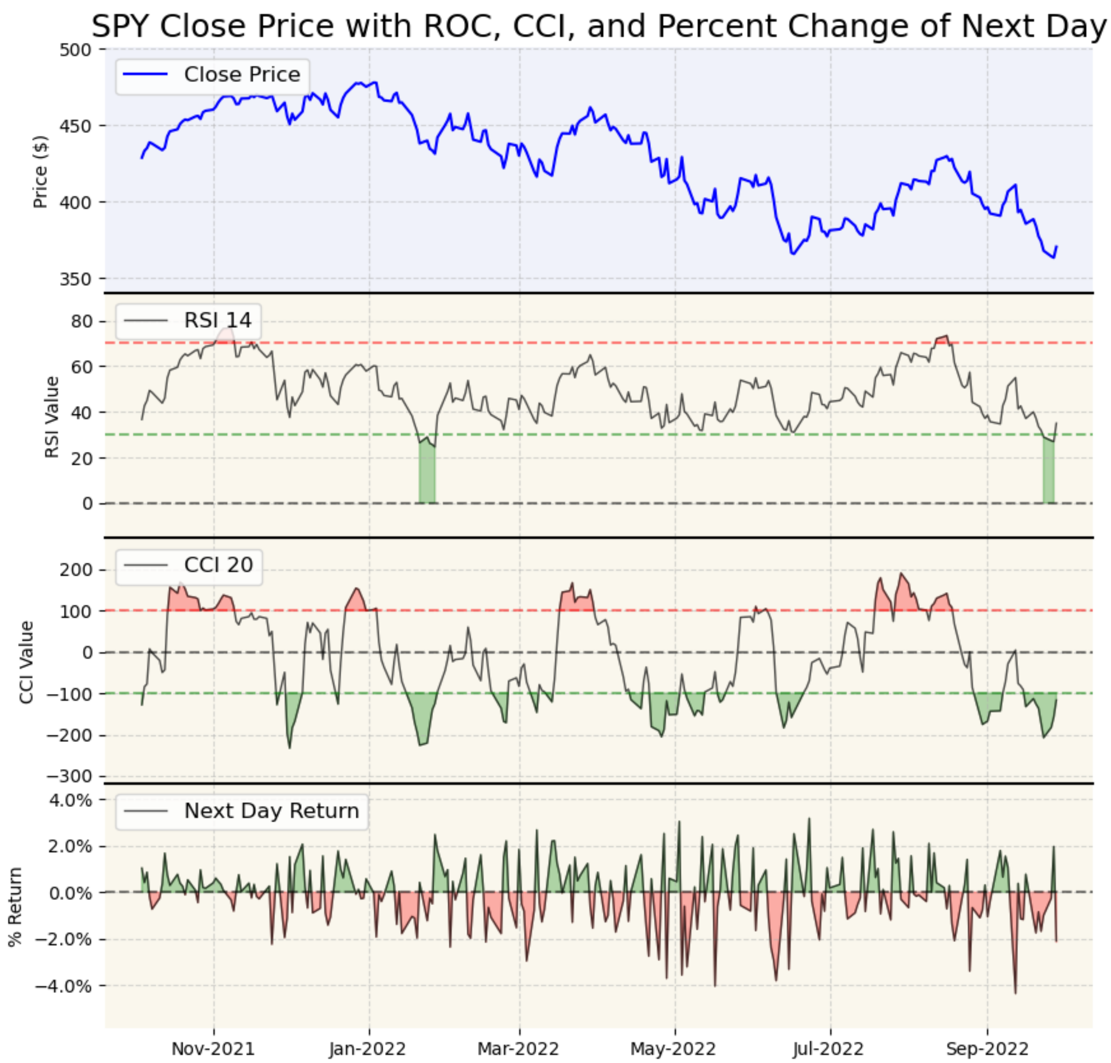
1.3.4 Plot Technical Indicators and Percent Change of the Next Day

[12]:
```
# plot Close, RSI, CCI, Percent Change Next Day
indicator_df = spy_ohlc_df[["Dates", "Close", "RSI 14", "CCI 20", "Percent Next␣
↪Day"]]
indicator_df = indicator_df.iloc[-250:-1]

fig = plt.figure(figsize=(10,10))
fig.subplots_adjust(hspace=0)

# Add Close:
close_ax = plt.subplot(4, 1, 1)
close_ax.plot(indicator_df["Dates"], indicator_df["Close"], color='blue',␣
↪linewidth=1.5, label="Close Price")
close_ax.set_ylabel("Price ($)")
close_ax.legend(loc="upper left", fontsize=12)
close_ax.set_title("SPY Close Price with ROC, CCI, and Percent Change of Next␣
↪Day", fontsize=18)
```

```python
# Add RSI 14:
rsi_ax = plt.subplot(4, 1, 2, sharex=close_ax)
rsi_ax.plot(indicator_df["Dates"], indicator_df["RSI 14"], color='k', linewidth = 1, alpha=0.7, label="RSI 14")
rsi_ax.legend(loc="upper left", fontsize=12)
rsi_ax.set_ylabel("RSI Value")

rsi_ax.axhline(30, color='green', linestyle = '--', alpha = 0.5)  # add a green horizontal line at 30
rsi_ax.axhline(70, color='red', linestyle = '--', alpha = 0.5)  # add a red horizontal line at 70
rsi_ax.axhline(0, color="black", linestyle = '--', alpha = 0.5)  # add a horizontal line at 0

rsi_ax.fill_between(indicator_df["Dates"], 0, indicator_df["RSI 14"], where=(indicator_df["RSI 14"] <=30),
                    color='g', alpha=0.3, interpolate=True)  # fill area below green
rsi_ax.fill_between(indicator_df["Dates"], 70, indicator_df["RSI 14"], where=(indicator_df["RSI 14"]>70),
                    color='r', alpha=0.3, interpolate=True)  # fill area above red

rsi_ax.xaxis.set_major_formatter(mdates.DateFormatter('%b-%Y'))  # format dates

# Add CCI 20:
cci_ax = plt.subplot(4, 1, 3, sharex=close_ax)
cci_ax.plot(indicator_df["Dates"], indicator_df["CCI 20"], color='k', linewidth = 1, alpha=0.7, label="CCI 20")
cci_ax.legend(loc="upper left", fontsize=12)
cci_ax.set_ylabel("CCI Value")

cci_ax.axhline(-100, color='green', linestyle = '--', alpha = 0.5)  # add a green horizontal line at -100
cci_ax.axhline(100, color='red', linestyle = '--', alpha = 0.5)  # add a red horizontal line at 100
cci_ax.axhline(0, color="black", linestyle = '--', alpha = 0.5)  # add a horizontal line at 0

cci_ax.fill_between(indicator_df["Dates"], -100, indicator_df["CCI 20"], where=(indicator_df["CCI 20"] <= -100),
                    color='g', alpha=0.3, interpolate=True)  # fill area below green
cci_ax.fill_between(indicator_df["Dates"], 100, indicator_df["CCI 20"], where=(indicator_df["CCI 20"] > 100),
```

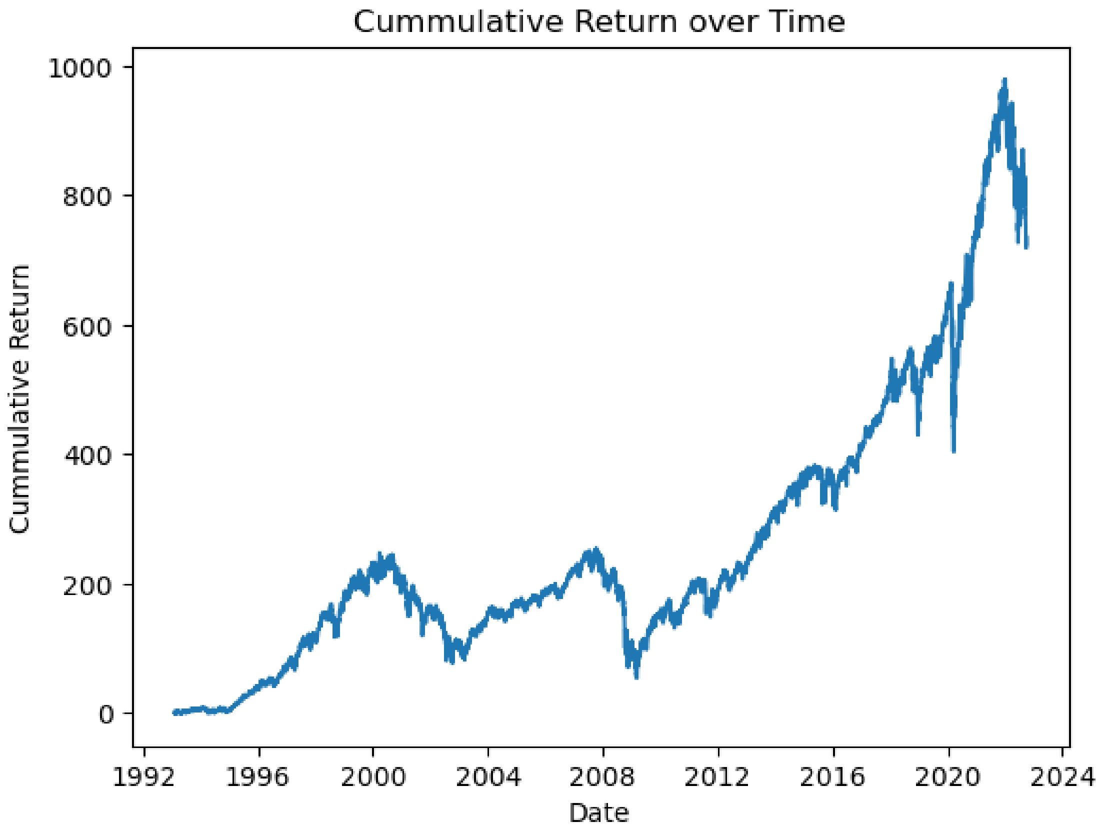
```
                    color='r', alpha=0.3, interpolate=True)  # fill area above␣
 ↪red

cci_ax.xaxis.set_major_formatter(mdates.DateFormatter('%b-%Y'))  # format dates

# Add next day return:
return_ax = plt.subplot(4, 1, 4, sharex=close_ax)
return_ax.plot(indicator_df["Dates"], indicator_df["Percent Next Day"] * 100,␣
 ↪color='k', linewidth = 1, alpha=0.7,
               label="Next Day Return")
return_ax.legend(loc="upper left", fontsize=12)
return_ax.set_ylabel("% Return")
return_ax.yaxis.set_major_formatter(mpl.ticker.PercentFormatter())

return_ax.axhline(0, color="black", linestyle = '--', alpha = 0.5)  # add a␣
 ↪horizontal line at 0

return_ax.fill_between(indicator_df["Dates"], 0, indicator_df["Percent Next␣
 ↪Day"] * 100,
                    where=(indicator_df["Percent Next Day"] * 100 <= 0),
                    color='r', alpha=0.3, interpolate=True)  # fill area below␣
 ↪red

return_ax.fill_between(indicator_df["Dates"], 0, indicator_df["Percent Next␣
 ↪Day"] * 100,
                    where=(indicator_df["Percent Next Day"] * 100 > 0),
                    color='g', alpha=0.3, interpolate=True)  # fill area above␣
 ↪green

return_ax.xaxis.set_major_formatter(mdates.DateFormatter('%b-%Y'))  # format␣
 ↪dates

# add grids to Close, rsi, and cci
close_ax.grid(visible=True, linestyle='--', alpha=0.5)
rsi_ax.grid(visible=True, linestyle='--', alpha=0.5)
cci_ax.grid(visible=True, linestyle='--', alpha=0.5)
return_ax.grid(visible=True, linestyle='--', alpha=0.5)

# Set facecolor (background) of plots
close_ax.set_facecolor((.94,.95,.98))
rsi_ax.set_facecolor((.98,.97,.93))
cci_ax.set_facecolor((.98,.97,.93))
return_ax.set_facecolor((.98,.97,.93))

# Add margins around plots
close_ax.margins(0.04, 0.2)
```

```
rsi_ax.margins(0.04, 0.2)
cci_ax.margins(0.04, 0.2)
return_ax.margins(0.04, 0.2)

# # Hiding the tick marks from the horizontal and vertical axis:
# close_ax.tick_params(left=False, bottom=False)
# rsi_ax.tick_params(left=False, bottom=False, labelrotation=45)

# Hiding all the spines for the price subplot:
for s in close_ax.spines.values():
    s.set_visible(False)
# Hiding all the spines for the rsi subplot:
for s in rsi_ax.spines.values():
    s.set_visible(False)
# Hiding all the spines for the cci subplot:
for s in cci_ax.spines.values():
    s.set_visible(False)
# Hiding all the spines for the Next Day Return subplot:
for s in return_ax.spines.values():
    s.set_visible(False)

# Reinstate top spine to divide subplots:
rsi_ax.spines['top'].set_visible(True)
rsi_ax.spines['top'].set_linewidth(1.5)

cci_ax.spines['top'].set_visible(True)
cci_ax.spines['top'].set_linewidth(1.5)

return_ax.spines['top'].set_visible(True)
return_ax.spines['top'].set_linewidth(1.5)


# code modified from https://towardsdatascience.com/
 ↪trading-toolbox-04-subplots-f6c353278f78
```



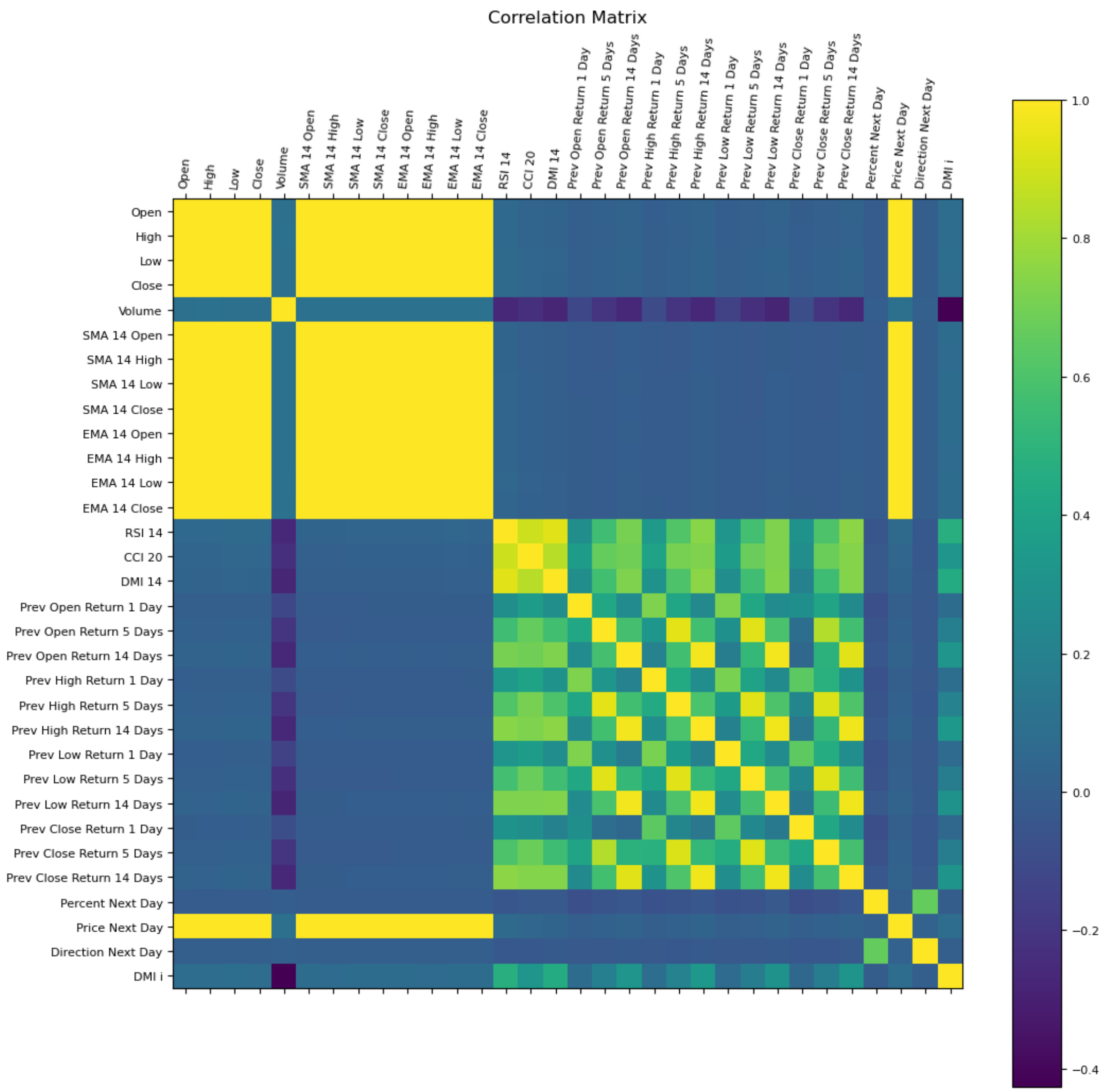
1.3.5 Cumulative Return Graph:

[13]:
```
first_price = spy_ohlc_df["Close"].iloc[0]
percent_changes = [0]
for val in spy_ohlc_df["Close"].iloc[1:]:
    percent_changes.append((val - first_price) / first_price * 100)

dates = spy_ohlc_df["Dates"]

plt.plot(dates, percent_changes)
plt.xlabel("Date")
plt.ylabel("Cummulative Return")
plt.title("Cummulative Return over Time")
plt.show()
```



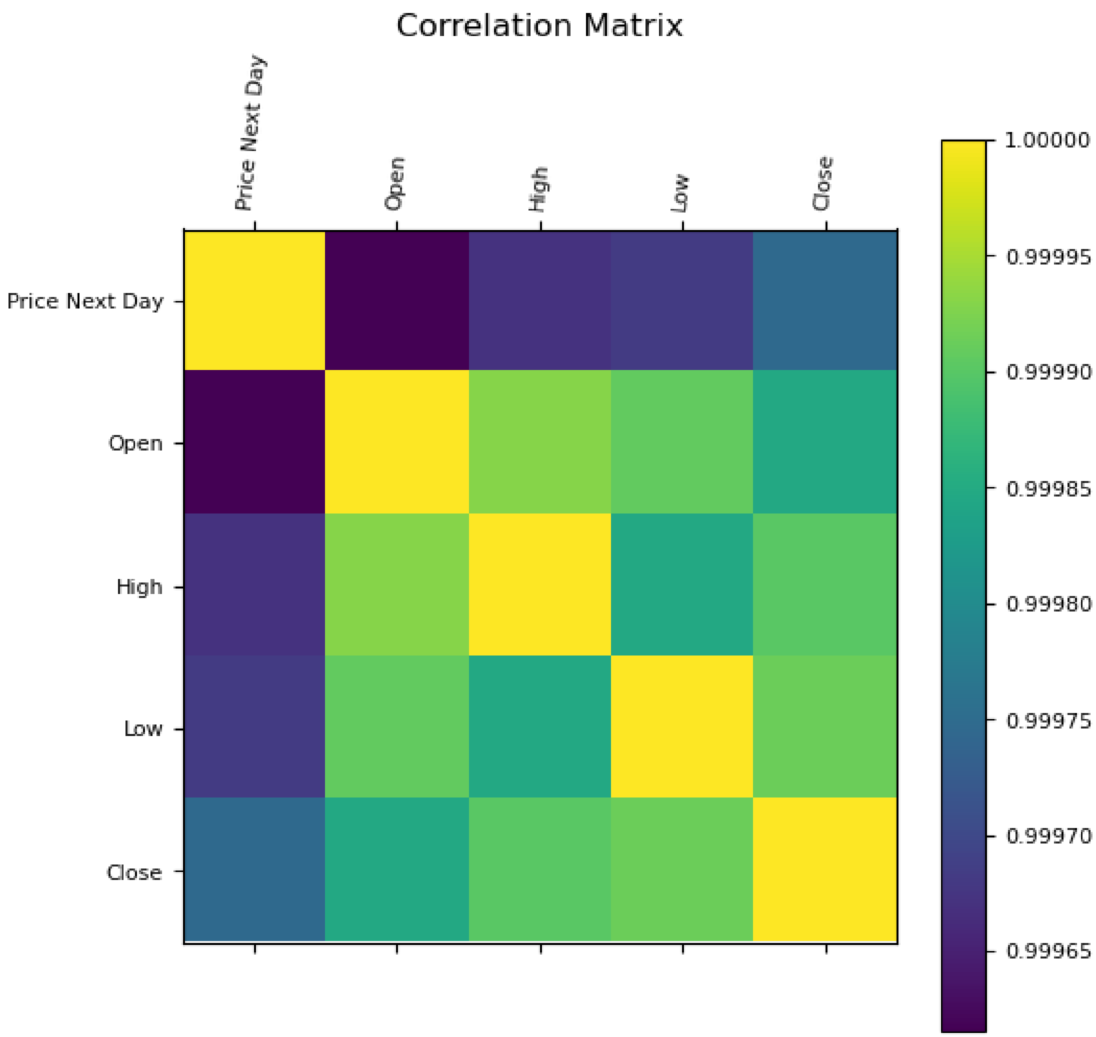
1.3.6 Correlation Between Variables

[14]:
```
f = plt.figure(figsize=(12, 12))
plt.matshow(spy_ohlc_df.corr(), fignum=f.number)

plt.xticks(range(spy_ohlc_df.select_dtypes(['number']).shape[1]), spy_ohlc_df.
 ↪select_dtypes(['number']).columns,
    fontsize=8, rotation=85)
plt.yticks(range(spy_ohlc_df.select_dtypes(['number']).shape[1]), spy_ohlc_df.
 ↪select_dtypes(['number']).columns,
           fontsize=8)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=8)
plt.title('Correlation Matrix', fontsize=12);
```



[47]:
```python
# Correlation Matrix of Interest:
columns = ["Price Next Day", "Open", "High", "Low", "Close",]

f = plt.figure(figsize=(6, 6))
spy_corr_df = spy_ohlc_df[columns]
plt.matshow(spy_corr_df.corr(), fignum=f.number)

plt.xticks(range(spy_corr_df.select_dtypes(['number']).shape[1]), spy_corr_df.
 ↪select_dtypes(['number']).columns,
    fontsize=8, rotation=85)
plt.yticks(range(spy_corr_df.select_dtypes(['number']).shape[1]), spy_corr_df.
 ↪select_dtypes(['number']).columns,
           fontsize=8)
```

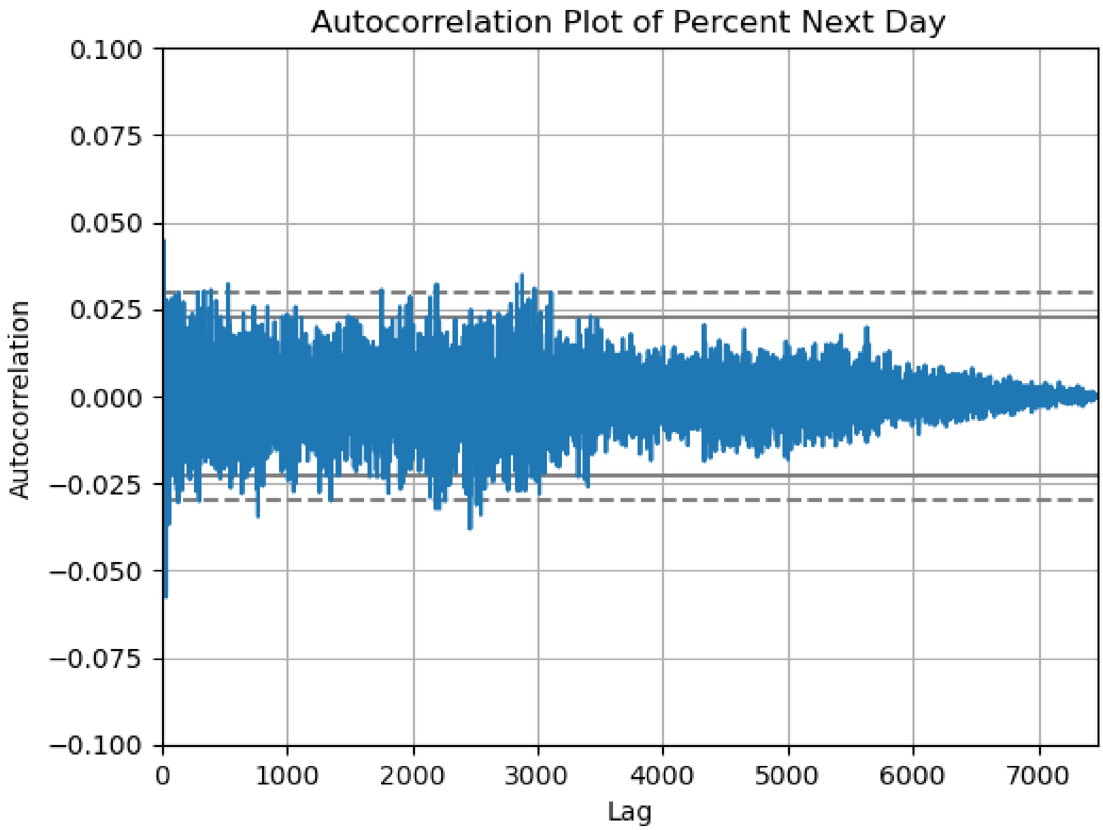
```
cb = plt.colorbar()
cb.ax.tick_params(labelsize=8)
plt.title('Correlation Matrix', fontsize=12);
```



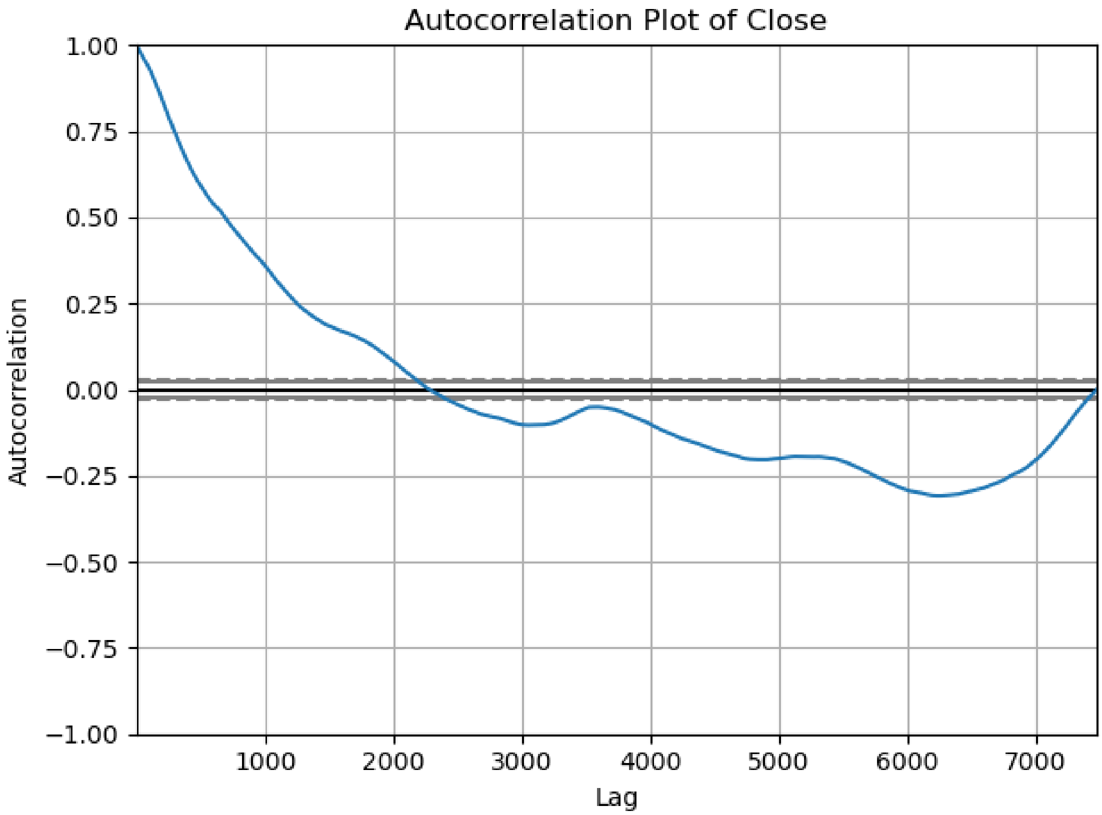
1.3.7 Autocorrelation Plots:

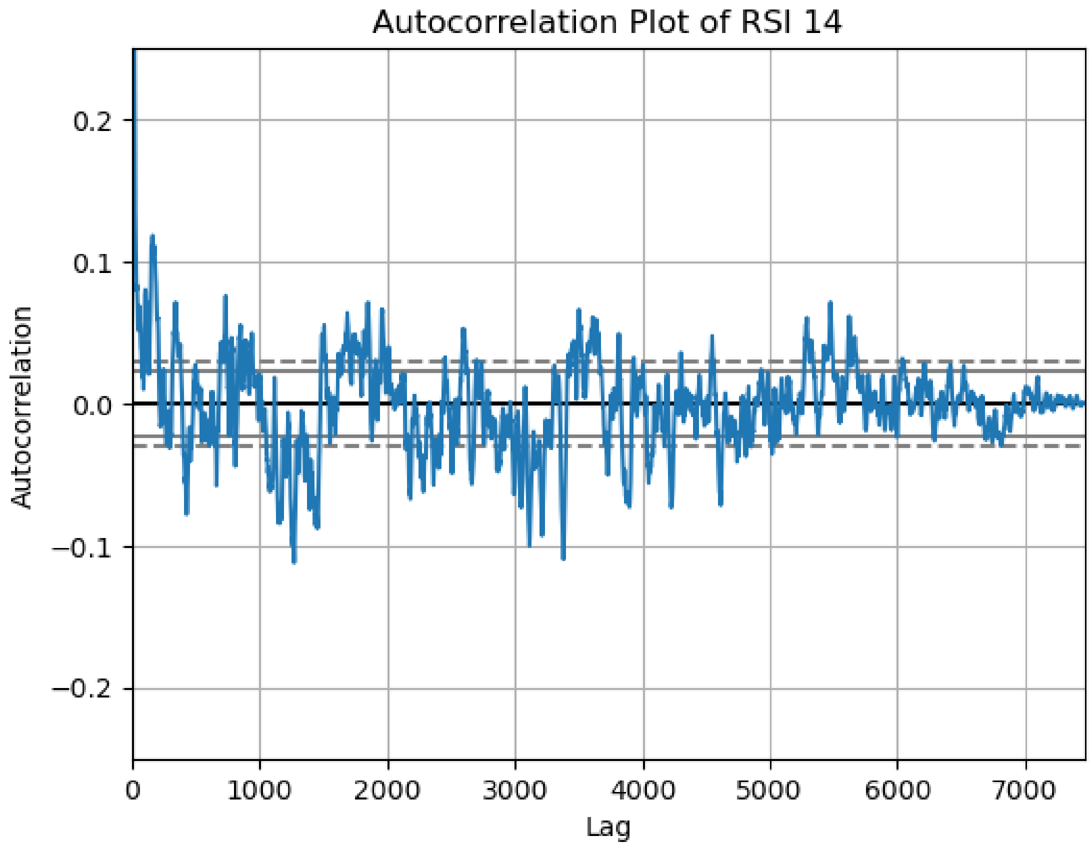
[16]:
```
# Percent Next Day:
pd.plotting.autocorrelation_plot(spy_ohlc_df['Percent Next Day'].iloc[:-1])
plt.title("Autocorrelation Plot of Percent Next Day")
plt.axis([0, len(spy_ohlc_df), -0.1, 0.1])
plt.show()

# Close:
pd.plotting.autocorrelation_plot(spy_ohlc_df['Close'])
```

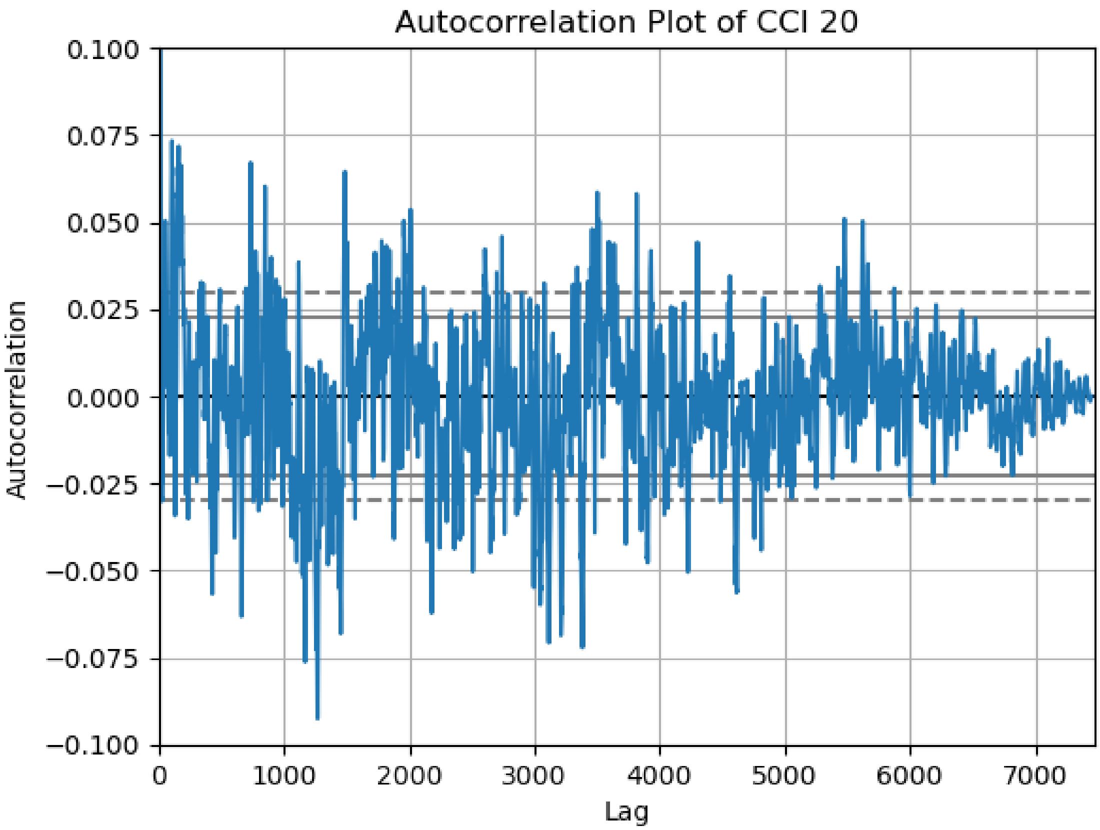
```
plt.title("Autocorrelation Plot of Close")
plt.tight_layout()
plt.show()

# RSI 14:
pd.plotting.autocorrelation_plot(spy_ohlc_df['RSI 14'].iloc[14:])
plt.title("Autocorrelation Plot of RSI 14")
plt.axis([0, len(spy_ohlc_df), -0.25, 0.25])
plt.show()

# CCI 20:
pd.plotting.autocorrelation_plot(spy_ohlc_df['CCI 20'].iloc[20:])
plt.title("Autocorrelation Plot of CCI 20")
plt.axis([0, len(spy_ohlc_df), -0.1, 0.1])
plt.show()

# Prev Low Return 1 Day:
pd.plotting.autocorrelation_plot(spy_ohlc_df['Prev Low Return 1 Day'].iloc[20:])
plt.title("Autocorrelation Plot of Prev Low Return 1 Day")
plt.axis([0, len(spy_ohlc_df), -0.1, 0.1])
plt.show()

# SMA 14 Close:
pd.plotting.autocorrelation_plot(spy_ohlc_df['SMA 14 Close'].iloc[20:])
plt.title("Autocorrelation Plot of SMA 14 Close")
plt.tight_layout()
plt.show()
```



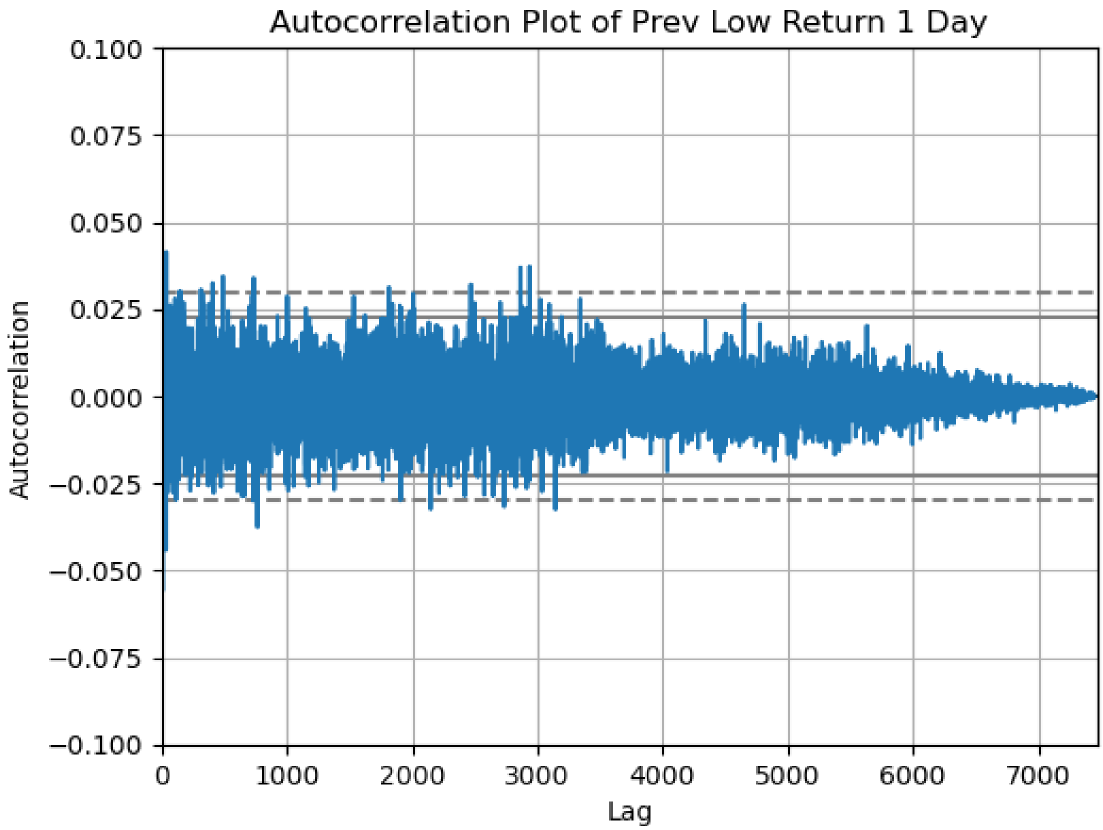
Autocorrelation Plot of Percent Next Day
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
Autocorrelation
0
1000
2000
3000
4000
5000
6000
7000
Lag

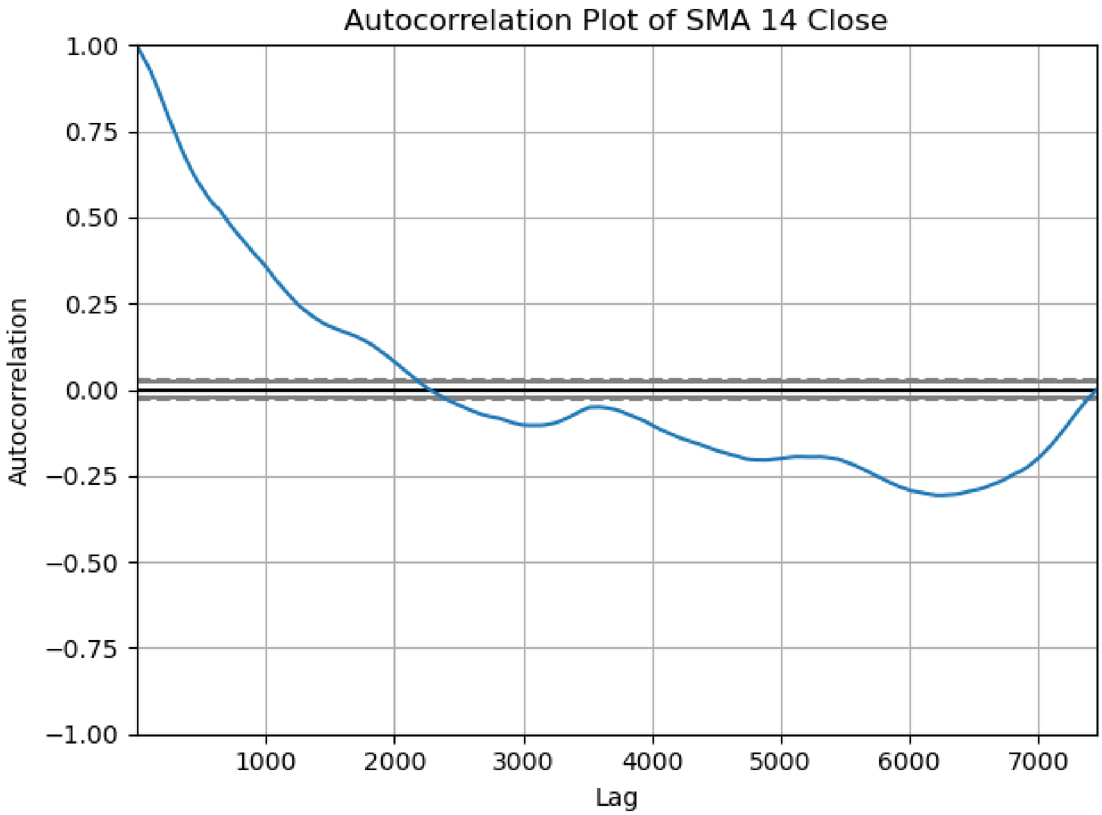

Autocorrelation Plot of Close
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Autocorrelation
1000
2000
3000
4000
5000
6000
7000
Lag


Autocorrelation Plot of RSI 14
Autocorrelation
0.2
0.1
0.0
−0.1
−0.2
0
1000
2000
3000
4000
5000
6000
7000
Lag



Autocorrelation Plot of CCI 20
Autocorrelation
Lag
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
0
1000
2000
3000
4000
5000
6000
7000


Autocorrelation Plot of Prev Low Return 1 Day
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
Autocorrelation
0
1000
2000
3000
4000
5000
6000
7000
Lag


Autocorrelation Plot of SMA 14 Close
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Autocorrelation
1000
2000
3000
4000
5000
6000
7000
Lag

```
[ ]:
```