ReadMe: Project_2_Team3

**The Title:**
BUILD A TRADING BOT TO BUY AND SELL STOCKS TO OUTPERFORM THE S&P 500

**Executive Summary:**
In this project we assumed the role of an equity stock trader using Fintech investing platform. The platform aims to offer investors an algorithmic machine learning trading bot. By using machine learning to evaluate our trading algorithm written in Python, we aim to remove the human element and human emotion factor to automate investment decision making.
Our goal is to build a machine learning trading bot to buy and sell Apple stock to outperform the S&P 500 human traders.

*How does our project relate to fintech and machine learning?*

We are using the programming language of Python, machine learning and neural network to build a sophisticated algorithmic trading bot. Specifically, we would like to explore stock with buy and sell signals which are automated by a trading performing at a 2% parameter from the baseline mean average.

**Software Version Control:**

Repository ML_Trading_Bot was created on GitHub.

This is our second attempt using the repository. This is the most updated repository:
https://github.com/KaliPatternbraker/ML_Trading_Bot

We had originally committed several times to files to the initial repository: This is not the repo to be graded again, but wanted to include it for examples of commit and progress of code and work:https://github.com/KaliPatternbraker/Project_2_Team_3

We commit messages with detail included with each commit. You can find well commented code in our Jupyter Notebook, Google Colab with analysis and explanations for results.
We organized our repository, with relevant information about the project files included.

**Data Collection and Preparation:**
We sourced our apple data from google sheets and sourced our S&P 500 data from Yahoo Finance. We uploaded the Apple stock data into our github repo and used a url address to call it into the code. For our S&P 500 dataset, we sourced our data from Yahoo Finance and then copied the data and pasted into an excel sheet, and converted the .xls file to a .csv file so that jupyter notebook could read the file. For both Apple Stock data and S&P 500, we used the same date range from January 1, 2021 to April 7, 2022.

Originally we had used the Alpaca API and REST API to source the Apple Stock data, but then when we tried to run the program in Google Colab, it would not work. We then resorted to source the data again from Google Sheets and Yahoo Finance.

To clean and prepare the data, we had to convert the Yahoo Finance columns with 'commas' in the dataset to float integers instead. We used the astype command for this. We also needed to change the titles and do a dropna to drop the none values in the columns.

**The Resources and Libraries:**
Pandas is a software library designed for data analytics, making it easier to work with data from any type of file. Pandas supplies powerful tools for working with data where time series is important. Analysts normally compare  financial assets, from single stocks to large portfolios over a period of time.
Pandas and Jupyter Notebook are efficient in the import, preparation and analysis of data of any type or quantity.
We created a Google Collab group to take advantage of the speed and convenience of collaborative cloud environments.
In order to run this stock machine learning trading model, we used the following machine learning modules as resources for code: Module 14, Module 10 and Import S&P 500 Dataset
We also used Google Search, scikit-learn, sklearn.svm.SVC and matplot lib

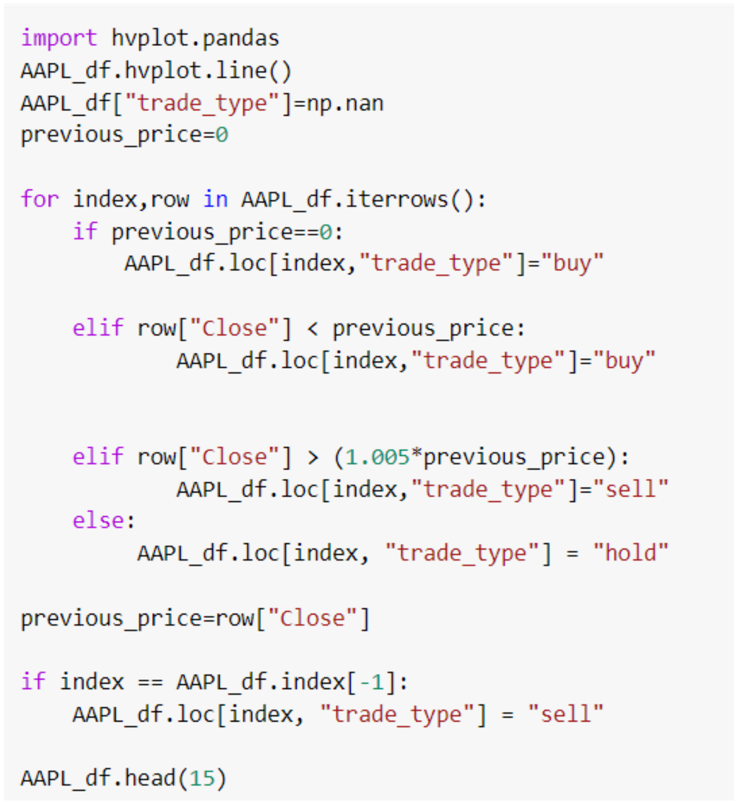
**The Libraries used not covered in Class:**

!pip install finta

**The Approach in Machine Learning:**
The approach we took was to write a program that uses supervised machine learning to analyze the profitability of Apple stock price (labeled AAPL) from January 1, 2021 to April 7, 2022 and calculate its return of investment, then compare it to the return of investment of the S&P 500 stock price for the same time period and compare the results to see which performed better in terms of optimized profitability.

**The Methods In Machine Learning:**
The first method of supervised learning used was to come up with buy, hold or sell signals. The project team decided to set a buy signal if the current day's close price was less than the previous day's close price. The sell signal would be triggered if the current day's close price was 5% higher than the previous day's close price. If none of these conditions were met the trade signal would be to hold.

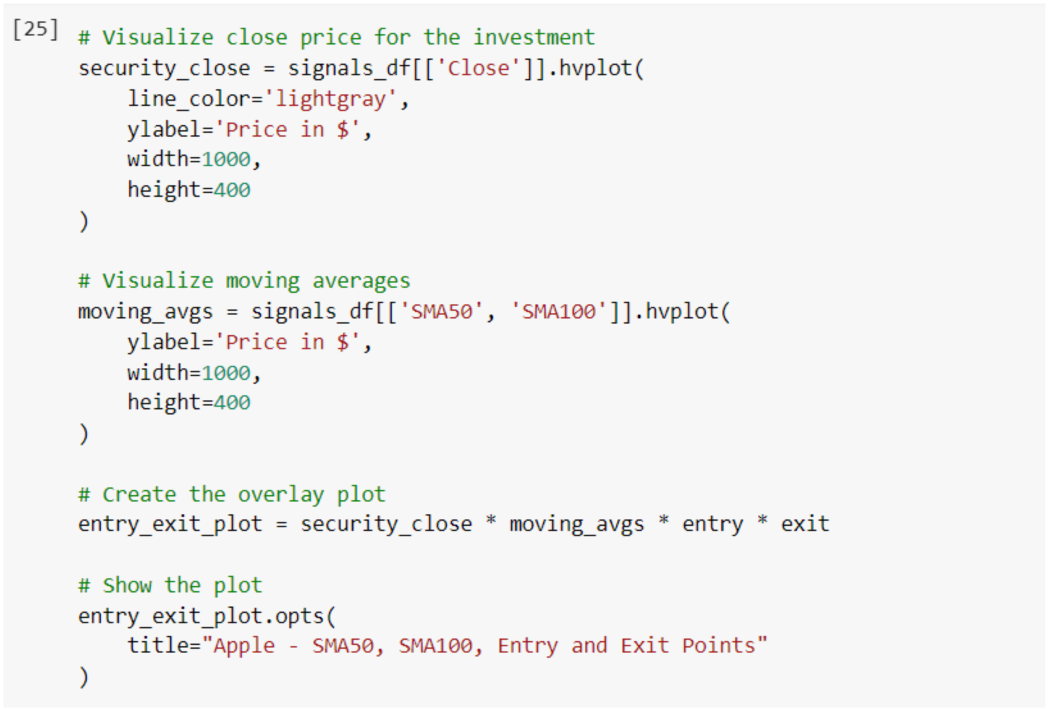
We also ran a different version of the model where we would buy if today's close price is 2% lower than the previous day, sell if today's close price is 2% above the previous day and hold if neither of these conditions were met.



```
import hvplot.pandas
AAPL_df.hvplot.line()
AAPL_df["trade_type"]=np.nan
previous_price=0

for index,row in AAPL_df.iterrows():
    if previous_price==0:
        AAPL_df.loc[index,"trade_type"]="buy"

    elif row["Close"] < previous_price:
            AAPL_df.loc[index,"trade_type"]="buy"


    elif row["Close"] > (1.005*previous_price):
            AAPL_df.loc[index,"trade_type"]="sell"
    else:
         AAPL_df.loc[index, "trade_type"] = "hold"

previous_price=row["Close"]

if index == AAPL_df.index[-1]:
    AAPL_df.loc[index, "trade_type"] = "sell"

AAPL_df.head(15)
```

```
for index, row in AAPL_df.iterrows():

    # buy if the previous_price is 0, in other words, buy on the first day
    if previous_price == 0:
        AAPL_df.loc[index, "trade_type"] = "buy"

        # calculate the cost of the trade by multiplying the current day's price
        # by the share_size, or number of shares purchased
        AAPL_df.loc[index, "cost/proceeds"] = -(row["Close"] * share_size)

        # add the number of shares purchased to the accumulated shares
        accumulated_shares += share_size

    # buy if the current day's price is less than the previous day's price
    elif row["Close"] < previous_price:
        AAPL_df.loc[index, "trade_type"] = "buy"

        # calculate the cost of the trade by multiplying the current day's price
        # by the share_size, or number of shares purchased
        AAPL_df.loc[index, "cost/proceeds"] = -(row["Close"] * share_size)

        # add the number of shares purchased to the accumulated shares
        accumulated_shares += share_size
    elif row["Close"] > (1.005*previous_price):
        AAPL_df.loc[index, "trade_type"] = "sell"
```

```
[25] exit = signals_df[signals_df['Entry/Exit'] == -1.0]['Close'].hvplot.scatter(
         color='yellow',
         marker='v',
         size=200,
         legend=False,
         ylabel='Price in $',
         width=1000,
         height=400
     )
     entry = signals_df[signals_df['Entry/Exit'] == 1.0]['Close'].hvplot.scatter(
         color='purple',
         marker='^',
         size=200,
         legend=False,
         ylabel='Price in $',
         width=1000,
         height=400
     )
```



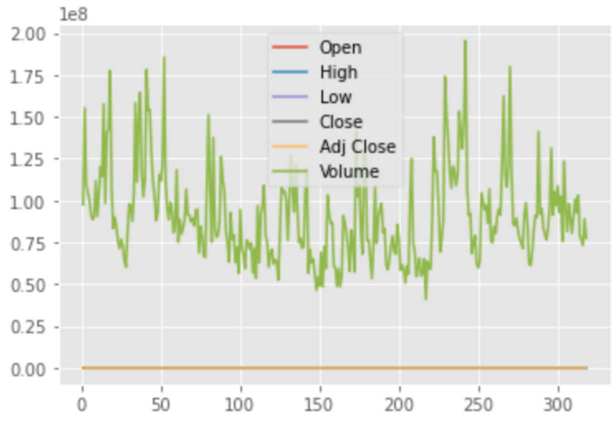
[25]

```python
# Visualize close price for the investment
security_close = signals_df[['Close']].hvplot(
    line_color='lightgray',
    ylabel='Price in $',
    width=1000,
    height=400
)

# Visualize moving averages
moving_avgs = signals_df[['SMA50', 'SMA100']].hvplot(
    ylabel='Price in $',
    width=1000,
    height=400
)

# Create the overlay plot
entry_exit_plot = security_close * moving_avgs * entry * exit

# Show the plot
entry_exit_plot.opts(
    title="Apple - SMA50, SMA100, Entry and Exit Points"
)
```

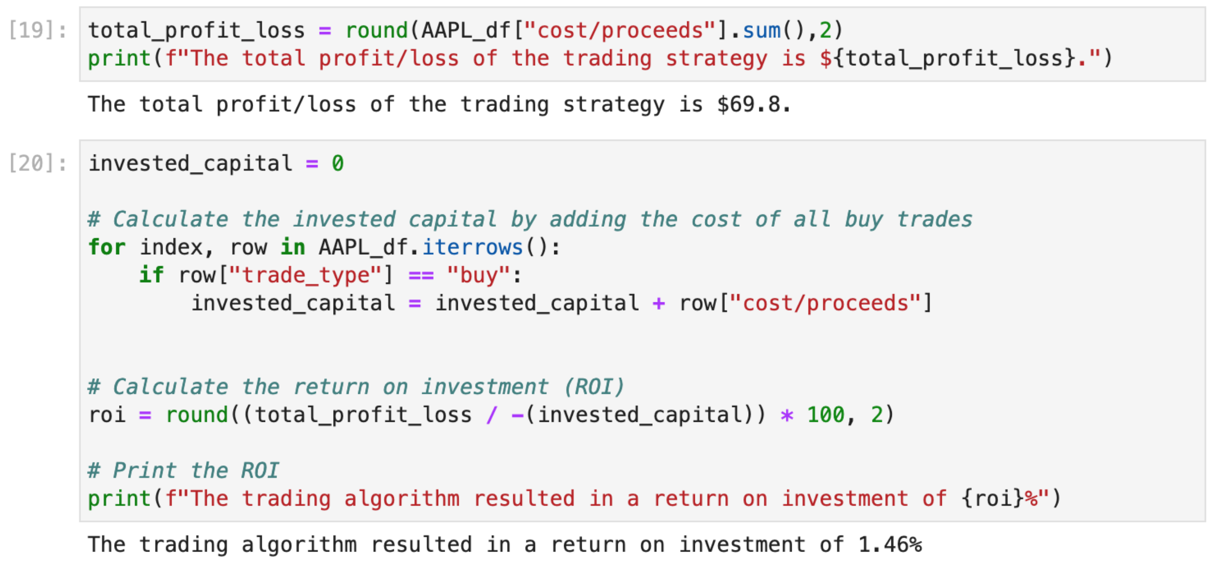
**Instructions:** Trading signals.

```
[16]: import hvplot.pandas

      AAPL_df.hvplot.line()
      AAPL_df["trade_type"]=np.nan
      previous_price=0

      for index,row in AAPL_df.iterrows():
          if previous_price==0:
              AAPL_df.loc[index,"trade_type"]="buy"

          elif row["Close"] < previous_price:
                  AAPL_df.loc[index,"trade_type"]="buy"


          elif row["Close"] > (1.005*previous_price):
                  AAPL_df.loc[index,"trade_type"]="sell"
          else:
               AAPL_df.loc[index, "trade_type"] = "hold"

      previous_price=row["Close"]

      if index == AAPL_df.index[-1]:
          AAPL_df.loc[index, "trade_type"] = "sell"

      AAPL_df.plot()
```

```
[16]: <AxesSubplot:>
```



**Results:**
Shots of dataframes and charts with detailed explanations -
AAPL (Apple) stock data price - Total profit/loss strategy & ROI calculations

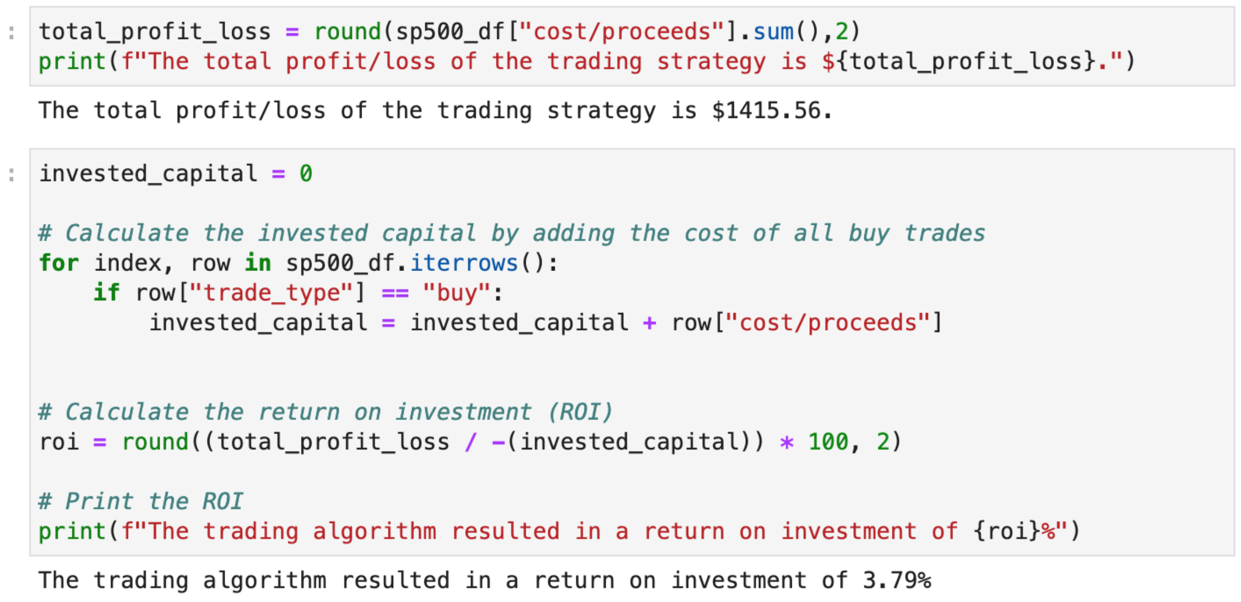


```
[19]: total_profit_loss = round(AAPL_df["cost/proceeds"].sum(),2)
      print(f"The total profit/loss of the trading strategy is ${total_profit_loss}.")

      The total profit/loss of the trading strategy is $69.8.

[20]: invested_capital = 0

      # Calculate the invested capital by adding the cost of all buy trades
      for index, row in AAPL_df.iterrows():
          if row["trade_type"] == "buy":
              invested_capital = invested_capital + row["cost/proceeds"]


      # Calculate the return on investment (ROI)
      roi = round((total_profit_loss / -(invested_capital)) * 100, 2)

      # Print the ROI
      print(f"The trading algorithm resulted in a return on investment of {roi}%")

      The trading algorithm resulted in a return on investment of 1.46%
```

The results were the expected return on investment for the supervised machine learning model, where we gave instructions to our trading bot to either buy or sell based on the stock price increasing or decreasing by 2% from the mean AAPL stock price for the time period of January 1, 2021 to April 7, 2022. The ROI and expected return was 1.51%. This is an annualized return of 552%.

**Results:**
Shots of dataframes and charts with detailed explanations -
S&P 500 stock data price - Total profit/loss strategy & ROI calculations

When we ran the same machine learning algorithmic code with the S&P 500 data, we received an ROI (return of investment) of 3.79%. This made us conclude that our machine learning trading bot did NOT outperform the S&P 500. In our original presentation our colleagues had sources the S&P 500 ROI results from Yahoo Finance and not by writing code to run the S&P 500 data through.We had wrongly concluded that our trading bot had outperformed the S&P 500. In this current second attempt, we had more accurate data results of the S&P 500 ROI since we had run into the exact same algorithm as the Apple stock data.



```
total_profit_loss = round(sp500_df["cost/proceeds"].sum(),2)
print(f"The total profit/loss of the trading strategy is ${total_profit_loss}.")
```

```
The total profit/loss of the trading strategy is $1415.56.
```

```
invested_capital = 0

# Calculate the invested capital by adding the cost of all buy trades
for index, row in sp500_df.iterrows():
    if row["trade_type"] == "buy":
        invested_capital = invested_capital + row["cost/proceeds"]


# Calculate the return on investment (ROI)
roi = round((total_profit_loss / -(invested_capital)) * 100, 2)

# Print the ROI
print(f"The trading algorithm resulted in a return on investment of {roi}%")
```

```
The trading algorithm resulted in a return on investment of 3.79%
```

**Challenges:**
The challenges we had originally was when we had decided to source our apple stock data from the Alpaca API and Rest API (for real time) data feeling that real time data would be more accurate for the trading bot to analyze. We soon learned that our algorithmic trading bot works better with historic data than real time data. We cannot analyze real time data.
A second challenge we had was when Google Colab would not read Alpaca API code or Rest API code. This is when we had to make decisions to source our code from Google Sheets and Yahoo Finance and create csv files manually.

**Next steps:**

Further development of this project would be to test different variations of the various trading signal variables. One example would be to change the buy/sell threshold. Another example would be to set the buy/sell signal based on a popular metric called the Relative Strength Index (RSI).
Additionally we ran the code on Google Colab to demonstrate that the trading bot that we built runs on the cloud. We ran the full code of the Supervised trading bot on Google Colab. This can be useful if we wanted to deploy the code for commercial purposes.
Based on the current results our bot trading AAPL did not outperform the S&P 500. Further testing could be to test an S&P 500 buy and hold strategy versus the active trading bot we built.

For Next steps we also evaluated the Apple Stock data with Unsupervised machine learning code using K-values. We have included the code for this, although the process had several errors and was unsuccessful. We concluded that K-value clustering was very difficult for an unsupervised machine learning trading bot to do with float integers for stock price data which were very close to each other in fluctuations. This most likely made clustering close to impossible to give any results that could be useful for analysis.

For further next steps, we would like to investigate programming a machine learning bot with a 5% buy, sell or hold parameter from the baseline to see if the change in setting the percentage of stock price increase or decrease affects the profitability performance of our trading bot. We also want to investigate other machine learning programs that can study fluctuations in stock price over a longer period of time and made better trading decisions on its own as to when to buy and sell.

For future steps, we would be interested in investigating the possibility of programming our machine learning trading bot to run on the blockchain network.

We would also like to use our trading bot to analyze trading cryptocurrency to see if setting 2% as a parameter would give better results in the crypto market versus the equity market.

**Conclusion:**
Perhaps from this ROI comparison Apple is not a good stock candidate to compare with the S&P 500 given the time range parameter that we chose, since January 1, 2021 to April 7, 2022 was a rather volatile time for equity stocks, namely Apple.
However, for next steps, we would like to use our machine learning trading bot to buy and sell S&P 500 stock since the ROI and profit index was so good.

**Documentation**
GitHub README.md file includes a project overview. We followed step by step Technical requirements
Code in Jupyter Notebook is well commented with relevant notes. Sometimes comments have reference code or notes which helped us follow our work over time.
GitHub README.md file includes examples of the application and the results with ROI.
GitHub README.md file includes detailed usage and installation instructions