# ALGORITHMIC TRADING WITH DEEP Q LEARNING AND LSTM

## A PROJECT REPORT

## *SUBMITTED BY*

**M SHREYAS - 100521729037**

**V SREENISH SHARMA - 100521729060**

**SWARGAM SATHWIK - 100521729057**

***In partial fulfillment of the award of***

**BACHELOR OF ENGINEERING**

**IN**

**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**


**Department of Computer Science and Engineering**

**University College of Engineering, Osmania University(Autonomous)**

**Hyderabad - 50000759**


# Department of Computer Science and Engineering
# University College of Engineering(Autonomous)
# Osmania University

# CERTIFICATE

This is to certify that the Project work entitled "**Algorithmic Trading with Deep Q Learning and LSTM**" submitted by **M Shreyas(100521729037)**, **V Sreenish Sharma(100521729060)** and **Swargam Sathwik(100521729057)** students of Department of Computer Science and Engineering, University College of Engineering, Osmania University for the award of the degree of Bachelor of Engineering is a record of the bonafide work carried out by them during the academic year 2024.

**Signature of the Project Guide**

**Dr V B Narasimha**

Assistant Professor

Dept of CSE, OU

**Head of the Department**

**Dr P V Sudha**

Professor

Dept of CSE, OU

# ACKNOWLEDGEMENT

We would like to express our deep sense of gratitude and whole-hearted thanks to my project guide **Dr.V.B.Narasimha**, **Associate Professor, Department of Computer Science and Engineering, University College of Engineering Osmania University**, for giving us the privilege of working under his guidance, with tremendous support and cogent discussion, constructive criticism and encouragement throughout this dissertation work carrying out the project work.

We would also like to thank **Dr P V Sudha, Head of the Department of the Computer Science and Engineering Department** for her support from the department and for allowing the use of all the resources available for us students.

We also extend my thanks to the entire faculty of the Computer Science and Engineering Department who encouraged us through the course of our bachelor's degree and sharing their precious knowledge to us.

Our sincere thanks to our parents and friends for their valuable suggestions, morals, strength, and support for the completion of our project.

M SHREYAS - 100521729037

V SREENISH SHARMA - 100521729060

SWARGAM SATHWIK - 100521729057

# DECLARATION

We hereby declare that the industrial mini-project entitled "**Algorithmic Trading with Deep Q Learning and LSTM**" is the work done and submitted in partial fulfillment of the Academic requirements in the sixth semester of **Artificial Intelligence and Machine Learning** from the University College of Engineering, Osmania University (Autonomous), Hyderabad. The results embodied in this project have not been submitted to any other University or Institution.

M SHREYAS - 100521729037

V SREENISH SHARMA - 100521729060

SWARGAM SATHWIK - 100521729057

# ABSTRACT

This project investigates the application of Deep Q-Learning (DQL) in the realm of algorithmic trading, emphasizing the development of autonomous trading strategies that can learn from historical data and adapt to real-time market conditions. By leveraging the capabilities of DQL, we aim to create an intelligent trading agent capable of making optimal trading decisions to maximize cumulative returns. The integration of DQL within the trading framework is demonstrated through the use of historical stock data from Yahoo Finance, specifically targeting technology giants. This abstract delves into the methodology, architecture, and performance evaluation of the DQL model within our trading system.

## Methodology

Deep Q-learning is a reinforcement learning technique that extends the Q-learning algorithm by incorporating deep neural networks to approximate the Q-value function. In this project, we employ a Deep Q-Network (DQN) to process and analyze historical stock data, transforming raw financial data into actionable insights. The DQL approach involves the agent interacting with the market environment by taking actions (buy, sell, or hold) based on observed states (stock prices, technical indicators) and receiving rewards corresponding to the profitability of these actions. The primary objective is to train the agent to develop a policy that maximizes the cumulative reward over time.

## Deep Q-Network Architecture

The architecture of the Deep Q-Network is pivotal to the success of the

reinforcement learning agent. Our DQN model consists of several layers of fully connected neural networks that map the input features to Q-values. The network processes the input data, which includes various financial indicators, to estimate the expected reward for each possible trading action. The network is trained using a loss function that minimizes the difference between the predicted Q-values and the actual rewards observed during training episodes.

## Data Preprocessing and Feature Engineering

Effective data preprocessing and feature engineering are crucial for enhancing the performance of the DQL model. The raw financial data retrieved from Yahoo Finance is subjected to a series of preprocessing steps, including handling missing values, normalizing the data, and engineering new features that capture market dynamics. These features may include moving averages, momentum indicators, and other technical indicators that provide deeper insights into market behavior. The processed data is then fed into the DQN, enabling the agent to make informed trading decisions.

## Predictive Modeling with LSTM

In addition to DQL, Long Short-Term Memory (LSTM) networks are employed to enhance the predictive capabilities of our trading system. LSTM models are a type of recurrent neural network specifically designed to handle sequential data and capture long-term dependencies, making them well-suited for time series prediction. By training LSTM models on historical stock prices and technical indicators, we can generate accurate forecasts of future stock prices and market movements. These predictions provide valuable inputs for the DQL agent, improving its decision-making process by anticipating market trends and

fluctuations.

## Training and Optimization

Training the DQL model involves iteratively improving the policy by allowing the agent to interact with the market environment and learn from its experiences. The training process is characterized by the agent exploring various trading strategies and exploiting the most rewarding ones. Optimization techniques such as gradient descent are employed to fine-tune the network's parameters, ensuring that the model converges to an optimal policy. Throughout the training process, we monitor key metrics, such as the mean squared error and cumulative reward, to assess the model's learning progress and adjust hyperparameters as needed.

## Performance Evaluation and Backtesting

The efficacy of the DQL model is evaluated through rigorous backtesting on historical stock market data. Backtesting allows us to simulate the agent's trading performance in a controlled environment and compare it against benchmark strategies, such as buy-and-hold. Key performance metrics, including cumulative return, Sharpe ratio, and maximum drawdown, are used to gauge the model's effectiveness. The results from the backtesting phase indicate the potential profitability and risk management capabilities of the DQL-based trading strategy.

## Conclusion

This project successfully demonstrates the application of Deep Q-Learning in algorithmic trading, showcasing its ability to develop adaptive and optimal trading strategies. By integrating DQL with historical stock data and leveraging LSTM for predictive modeling, the project highlights the potential of reinforcement learning

techniques to improve decision-making and trading efficiency in financial markets. The results underscore the effectiveness of data-driven approaches in navigating market complexities and enhancing trading performance. Future research should explore the incorporation of additional data sources, such as market sentiment and news analytics, to further refine the predictive capabilities of the DQL model and capitalize on emerging trading opportunities.

# TABLE OF CONTENTS


| S.No | Contents | Page No |
|---|---|---|
| 1. | INTRODUCTION<br>❖ Introduction to Stock Market Prediction<br>❖ Machine Learning in Finance<br>❖ Long Short-Term Memory (LSTM) Networks<br>❖ Reinforcement Learning and Trading Agents<br>❖ Project Objectives<br>❖ Methodology<br>❖ Evaluation Metrics<br>❖ Significance and Impact<br>❖ Challenges and Future Work<br>❖ Conclusion | 1 |
| 2. | ROLES AND RESPONSIBILITIES | 7 |
| 3. | REQUIREMENTS SPECIFICATIONS<br>❖ Software Requirements<br>❖ Hardware Requirements<br>❖ Additional Considerations | 8 |
| 4. | DESIGN AND ARCHITECTURES<br>❖ Use case Diagrams<br>❖ Activity Diagrams<br>❖ Sequence Diagrams<br>❖ Class Diagrams<br>❖ Architecture | 11 |

| | | |
|---|---|---|
| **5.** | IMPLEMENTATION<br>❖ Getting the Data<br>❖ Descriptive Statistics about the Data<br>❖ What was the moving average of the various stocks?<br>❖ What was the daily return of the stock on average?<br>❖ Predicting the closing price stock price of APPLE Inc. using LSTM<br>❖ Deep Q-Learning (DQL) in Algorithmic Trading: Enhancing Decision-Making through Reinforcement Learning<br>❖ Total Profit after 100 episodes of Training | 16 |
| **6.** | RESULTS<br>❖ LSTM Prediction results<br>❖ Deep Q Learning Results | 43 |
| **7.** | CONCLUSION | 44 |
| **8.** | REFERENCES | 47 |

# CHAPTER 1: INTRODUCTION

## Introduction to Stock Market Prediction :

The stock market is a complex and dynamic system where prices fluctuate based on various factors including economic indicators, investor sentiment, and global events. Predicting stock prices is a challenging task due to the inherent volatility and noise present in financial markets. Traditional methods of stock market analysis, such as fundamental analysis and technical analysis, have been widely used by traders and investors. However, these methods have limitations, particularly in their ability to process and analyze large volumes of data and capture complex patterns.

In recent years, the advent of machine learning and artificial intelligence has brought new opportunities to enhance stock market prediction. Machine learning algorithms, with their ability to learn from historical data and identify intricate patterns, offer a promising alternative to traditional approaches. This project leverages the power of machine learning to predict stock prices and develop a trading agent that can make informed decisions.

## Machine Learning in Finance :

Machine learning has revolutionized various industries, and finance is no exception. In finance, machine learning is used for various applications including credit scoring, fraud detection, portfolio management, and algorithmic trading. The primary advantage of machine learning over traditional methods is its ability to process vast amounts of data and adapt to changing market conditions.

Machine learning models can analyze historical price data, news articles, social media

sentiment, and other relevant information to predict future price movements. These models can uncover hidden patterns and correlations that may not be apparent through traditional analysis. As a result, machine learning models can provide more accurate and timely predictions, helping traders and investors make better decisions.

## Long Short-Term Memory (LSTM) Networks :

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) specifically designed to model sequential and time-series data. Unlike traditional feedforward neural networks, LSTMs have the capability to retain information over long periods, making them ideal for time series prediction tasks such as stock price forecasting.

LSTMs address the vanishing gradient problem commonly encountered in RNNs, allowing them to learn long-term dependencies in data. This is achieved through a complex architecture that includes memory cells, input gates, output gates, and forget gates. These components work together to regulate the flow of information, enabling the network to remember important information and forget irrelevant details.

In this project, LSTM networks are employed to predict future stock prices based on historical data. The ability of LSTMs to capture temporal dependencies and patterns in stock price movements makes them a powerful tool for financial time series prediction.

## Reinforcement Learning and Trading Agents :

Reinforcement learning (RL) is a branch of machine learning that focuses on training agents to make decisions by interacting with an environment. The agent learns to achieve specific goals by receiving feedback in the form of rewards or penalties. Over

time, the agent optimizes its actions to maximize cumulative rewards.

Deep Q-Learning (DQL) is a popular RL algorithm that combines Q-Learning with deep neural networks. In DQL, a deep neural network, known as the Q-network, approximates the Q-value function, which estimates the expected utility of actions taken in specific states. The Q-network is trained using experiences collected by the agent during its interaction with the environment.

In the context of this project, a trading agent is developed using DQL to make buy, sell, or hold decisions. The trading agent aims to maximize profits by learning optimal trading strategies through continuous interaction with the market environment.

## Project Objectives :

The primary objective of this project is to develop a robust and accurate stock market prediction system using LSTM networks and a trading agent using Deep Q-Learning. The specific goals include:

- Predicting future stock prices using historical data.
- Developing a trading agent that can make informed decisions based on predicted prices.
- Evaluating the performance of the prediction model and trading agent using appropriate metrics.
- Analyzing the potential impact of the developed system on trading strategies and financial decision-making.

## Methodology :

The methodology for this project involves several key steps:

- **Data Collection and Preprocessing:** Historical stock price data is collected from reliable financial sources. The data is preprocessed to handle missing values, normalize the values, and create suitable input-output pairs for the LSTM model.

- **LSTM Model Development:** An LSTM network is designed and trained using the preprocessed data. The model's hyperparameters are tuned to achieve optimal performance. The trained model is used to predict future stock prices.

- **Trading Agent Development with Deep Q-Learning:** A trading agent is developed using the Deep Q-Learning algorithm. The agent is trained to make buy, sell, or hold decisions based on the predicted stock prices. The Q-network is updated continuously based on the agent's interactions with the market environment.

## Evaluation Metrics :

The performance of the stock price prediction model and the trading agent is evaluated using several metrics:

**For the LSTM Model:**

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Directional Accuracy

**For the Trading Agent:**

- Cumulative Return
- Sharpe Ratio
- Maximum Drawdown
- Trading Accuracy

These metrics provide insights into the accuracy and reliability of the predictions, as well as the effectiveness of the trading strategies.

## Significance and Impact :

The successful development of a stock market prediction system and trading agent has significant implications for the finance industry. Such a system can enhance trading strategies, improve decision-making, and potentially lead to higher returns on investments. By leveraging machine learning, traders and financial analysts can gain a competitive edge in the market.

The ability to predict stock prices accurately can also contribute to risk management by identifying potential market downturns in advance. Overall, the integration of machine learning in financial trading represents a step towards more intelligent and data-driven investment approaches.

## Challenges and Future Work :

Despite the promising results, there are several challenges associated with this project. Financial markets are influenced by numerous factors, many of which are difficult to quantify and predict. Additionally, the dynamic nature of markets means that models need to be continuously updated and validated.

Future work could involve incorporating additional data sources, such as news sentiment analysis and macroeconomic indicators, to enhance the prediction accuracy. Moreover, exploring other reinforcement learning algorithms and model architectures could further improve the performance of the trading agent.

## Conclusion :

This project demonstrates the potential of combining LSTM networks and Deep Q-Learning to predict stock prices and develop an intelligent trading agent. The results highlight the advantages of using machine learning in financial markets, offering more accurate predictions and better trading strategies. Continued research and development in this field can lead to significant advancements in algorithmic trading and investment management.

# CHAPTER 2: ROLES AND RESPONSIBILITIES

| NAME | R NO | CONTRIBUTION |
|---|---|---|
| M SHREYAS | 100521729037 | • Model Building<br>• Model Training<br>• Knowledge of Trading Indicators |
| V SREENISH SHARMA | 100521729060 | • Model Building<br>• Error Corrections<br>• Mathematical Reasoning and Trading Logic |
| SWARGAM SATHWIK | 100521729057 | • Data Collection<br>• Data Analysis<br>• Documentation and Data Visualisation<br>• Real-time experimentation |

# CHAPTER 3: REQUIREMENTS SPECIFICATIONS

## 1. Software Requirements

- ➢ Operating System
  - Windows 10 or later
  - macOS 10.15 Catalina or later
  - Linux (Ubuntu 18.04 or later)
- ➢ Development Environment
  - Anaconda Distribution (Python 3.7+)
  - Jupyter Notebook or JupyterLab
- ➢ Programming Languages
  - Python 3.7 or later
- ➢ Libraries and Frameworks
  - TensorFlow 2.x or PyTorch (for LSTM and Deep Q-Learning implementation)
  - Keras (if using TensorFlow as the backend)
  - NumPy (for numerical computations)
  - Pandas (for data manipulation and analysis)
  - Matplotlib/Seaborn (for data visualization)
  - Scikit-learn (for preprocessing and additional machine learning tasks)
  - Gym (for creating and working with reinforcement learning environments)
  - Yahoo Finance or Alpha Vantage (for obtaining historical stock price data)
- ➢ Integrated Development Environment (IDE)

- PyCharm
- VS Code
- Jupyter Notebook

➢ Additional Tools

- Git (for version control)
- TensorBoard (for visualizing training progress and metrics)

## 2. Hardware Requirements

➢ Processor

- Minimum: Intel i5 or AMD equivalent
- Recommended: Intel i7 or AMD Ryzen 7

➢ Memory (RAM)

- Minimum: 8 GB
- Recommended: 16 GB or more

➢ Storage

- Minimum: 256 GB SSD
- Recommended: 512 GB SSD or more (to handle large datasets and models)

➢ Graphics Processing Unit (GPU)

Optional for small-scale models, but highly recommended for faster training and larger models.

- Minimum: NVIDIA GTX 1050
- Recommended: NVIDIA RTX 2070 or better (for deep learning tasks)

➢ Network

Stable internet connection for downloading libraries, datasets, and for any cloud-based training or deployment.

Additional Considerations

➢ Cloud Services (optional)

- AWS, GCP, or Azure (for cloud-based training and deployment, especially for handling larger datasets and models)
- Colab (Google Colaboratory for free GPU access during initial development and testing)

➢ Data Storage

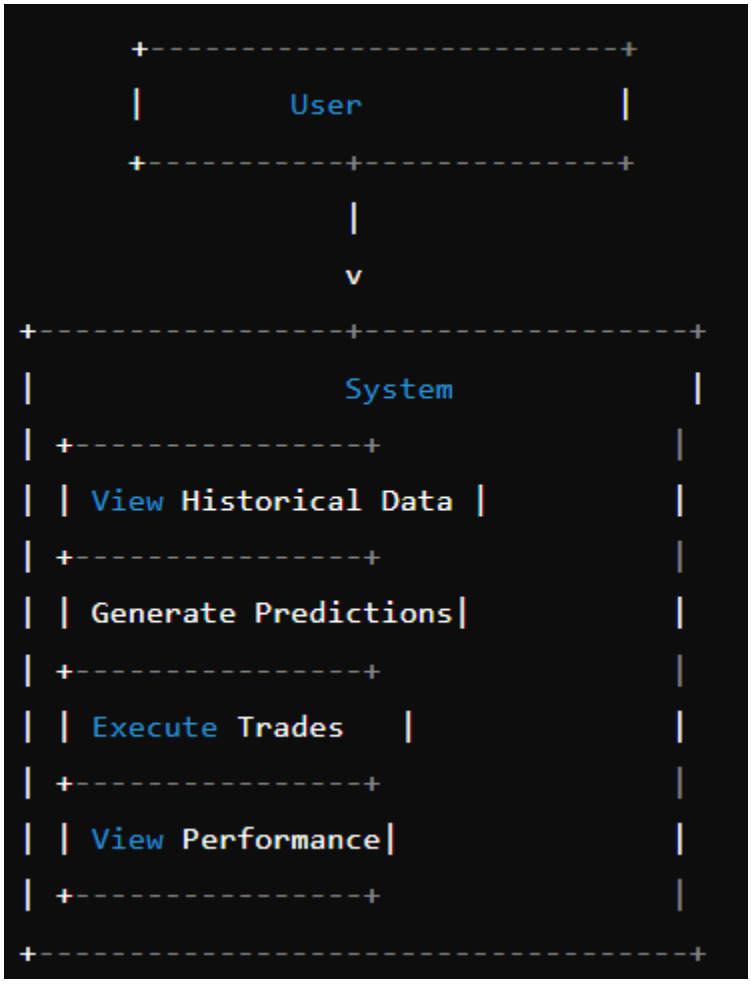
- Local storage for small datasets and development.
- Cloud storage (AWS S3, Google Cloud Storage) for large datasets.
- Backup and Recovery

➢ Regular backups of code and data to prevent loss.

➢ Version control using Git to keep track of code changes.

# CHAPTER 4: DESIGN AND ARCHITECTURES

## Use Case Diagram :




## Activity Diagrams :

- **Activity Diagram for Generating Stock Price Predictions:**
  Start -> [Collect Historical Data] -> [Preprocess Data] -> [Train LSTM Model] -> [Generate Predictions] -> End
- **Activity Diagram for Trading Execution:**

Start -> [Initialize Trading Environment] -> [Get Predictions] -> [Make Trading Decision] -> [Execute Trade] -> [Log Trade] -> End

## Sequence Diagram :

**Sequence Diagram for Generating Stock Price Predictions:**

User -> System: Request Historical Data

System -> DataCollector: Collect Data

DataCollector -> System: Return Data

System -> StockData: Preprocess Data

System -> PredictionModel: Train Model

PredictionModel -> System: Model Trained

User -> System: Request Predictions

System -> PredictionModel: Generate Predictions

PredictionModel -> System: Return Predictions

System -> User: Display Predictions

**Sequence Diagram for Executing Trades:**

User -> System: Start Trading Session

System -> TradingAgent: Initialize Agent

System -> DataCollector: Collect Data

DataCollector -> System: Return Data

System -> TradingAgent: Get Predictions

TradingAgent -> PredictionModel: Generate Predictions

PredictionModel -> TradingAgent: Return Predictions

TradingAgent -> System: Make Trading Decision

System -> TradingAgent: Execute Trade

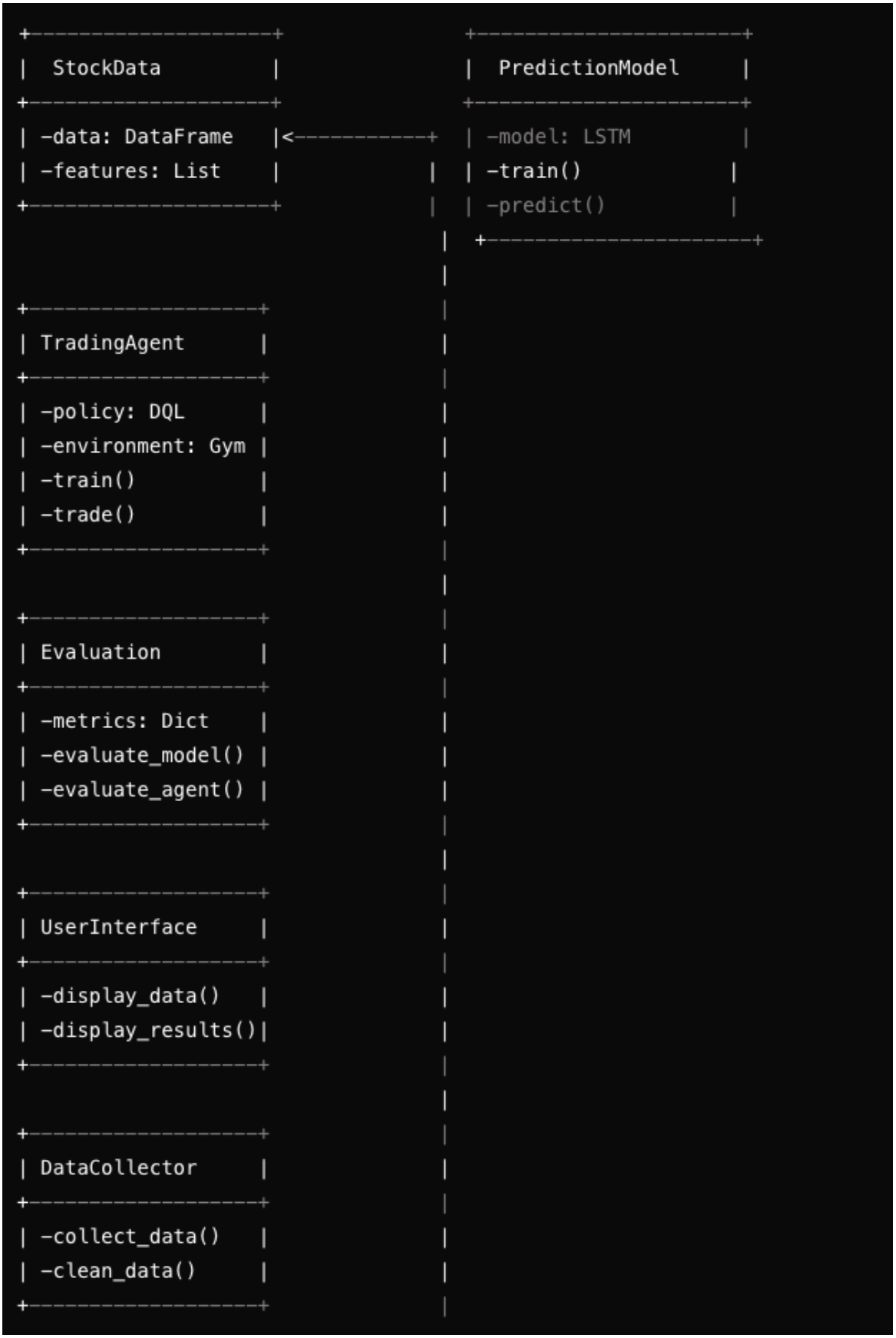
TradingAgent -> System: Log Trade

System -> User: Display Trade Results

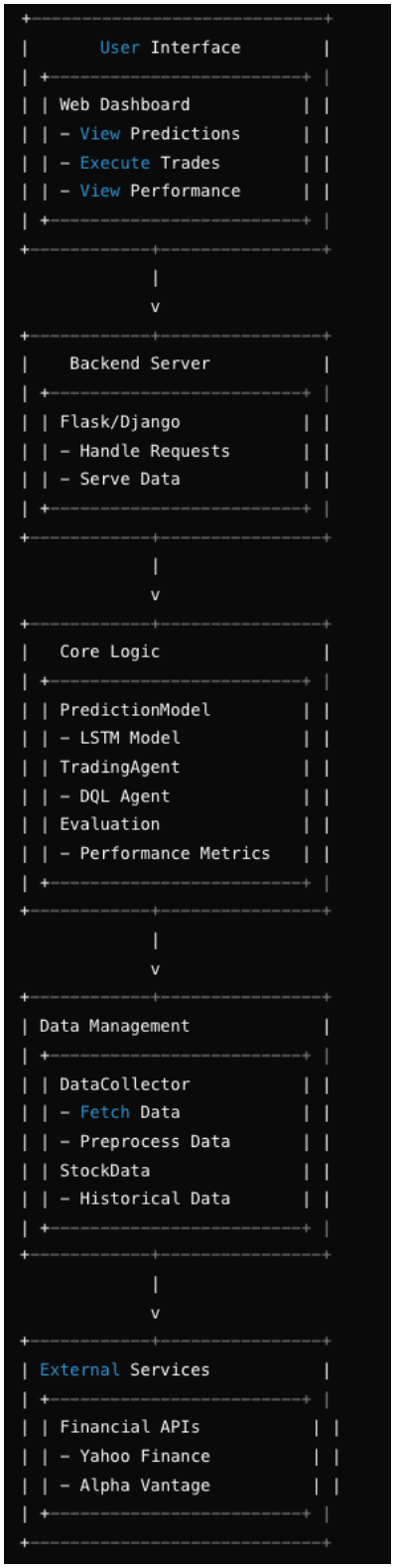
## Class Diagram :

Class diagrams describe the static structure of the system by showing its classes, their attributes, methods, and the relationships between objects.

- **Class Diagram for Stock Market Prediction and Trading Agent:**


StockData
-data: DataFrame
-features: List
PredictionModel
-model: LSTM
-train()
-predict()
TradingAgent
-policy: DQL
-environment: Gym
-train()
-trade()
Evaluation
-metrics: Dict
-evaluate_model()
-evaluate_agent()
UserInterface
-display_data()
-display_results()
DataCollector
-collect_data()
-clean_data()

# System Architecture :



# CHAPTER 5: IMPLEMENTATION

## Getting the Data :

The data used in this project comes from Yahoo Finance, a comprehensive resource for financial market data and investment information. The project utilizes the yfinance library, a Python interface that allows easy and efficient downloading of historical market data directly from Yahoo Finance. This library provides access to detailed stock information, including historical prices, volumes, and other relevant financial metrics for various technology stocks. This data serves as the foundation for the project's analysis, visualization, and predictive modeling efforts.

### 1. What was the change in price of the stock over time?

In this section, we'll go over how to handle requesting stock information with pandas, and how to analyze basic attributes of a stock.

```
!pip install -q yfinance

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style('whitegrid')

plt.style.use("fivethirtyeight")

%matplotlib inline

# For reading stock data from yahoo

from pandas_datareader.data import DataReader

import yfinance as yf
```

```
from pandas_datareader import data as pdr

yf.pdr_override()

# For time stamps

from datetime import datetime

# The tech stocks we'll use for this analysis

tech_list = ['AAPL', 'GOOG', 'MSFT', 'AMZN']

end = datetime.now()

start = datetime(end.year - 1, end.month, end.day)

for stock in tech_list:

    globals()[stock] = yf.download(stock, start, end)

company_list = [AAPL, GOOG, MSFT, AMZN]

company_name = ["APPLE", "GOOGLE", "MICROSOFT", "AMAZON"]

for company, com_name in zip(company_list, company_name):

    company["company_name"] = com_name

df = pd.concat(company_list, axis=0)

df.tail(10)
```

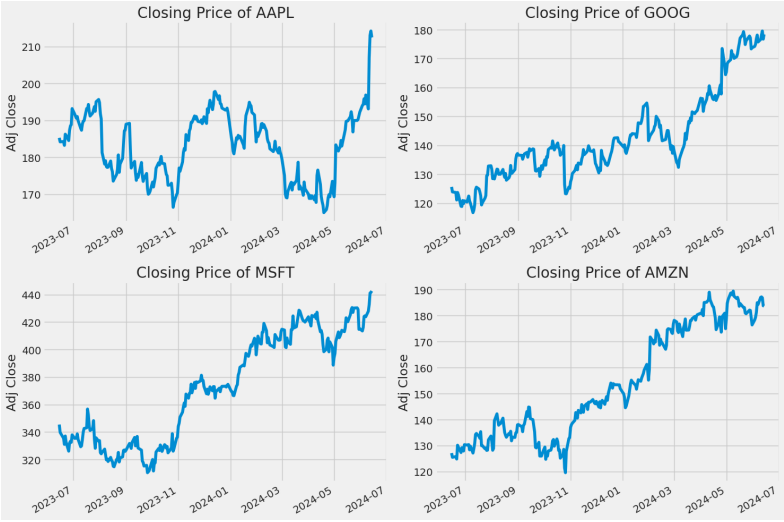
## Descriptive Statistics about the Data :

```
AAPL.describe()

AAPL.info()
```

## 1. Closing Price :

The closing price is the last price at which the stock is traded during the regular trading day. A stock's closing price is the standard benchmark used by investors to track its performance over time.

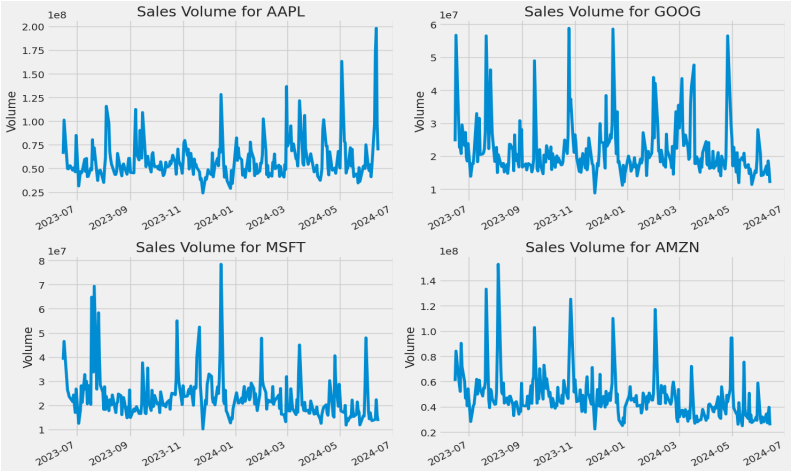
```
# Let's see a historical view of the closing price

plt.figure(figsize=(15, 10))

plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):

    plt.subplot(2, 2, i)

    company['Adj Close'].plot()

    plt.ylabel('Adj Close')

    plt.xlabel(None)

    plt.title(f"Closing Price of {tech_list[i - 1]}")

plt.tight_layout()
```



## 2. Volume of Sales :

Volume is the amount of an asset or security that changes hands over some period of time, often over the course of a day. For instance, the stock trading volume would refer to the number of shares of security traded between its daily open and close. Trading volume, and changes to volume over the course of time, are important inputs for technical traders.

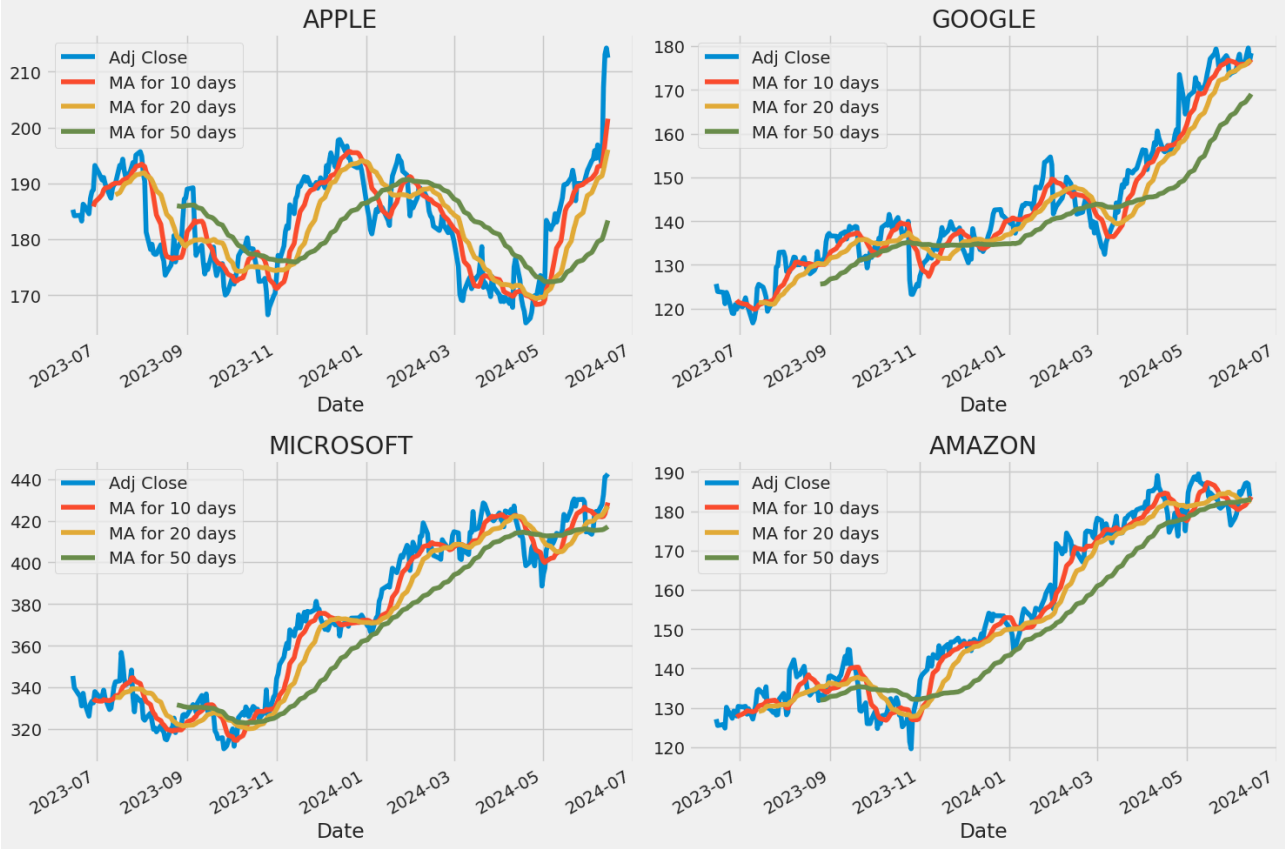
```
# Now let's plot the total volume of stock being traded each day

plt.figure(figsize=(15, 10))

plt.subplots_adjust(top=1.25, bottom=1.2)

for i, company in enumerate(company_list, 1):

    plt.subplot(2, 2, i)

    company['Volume'].plot()

    plt.ylabel('Volume')

    plt.xlabel(None)

    plt.title(f"Sales Volume for {tech_list[i - 1]}")

plt.tight_layout()
```



## What was the moving average of the various stocks?

The moving average (MA) is a simple technical analysis tool that smooths out price data by creating a constantly updated average price. The average is taken over a specific period of time, like 10 days, 20 minutes, 30 weeks, or any time period the trader chooses.

```
  ma_day = [10, 20, 50]

  for ma in ma_day:

      for company in company_list:

          column_name = f"MA for {ma} days"

          company[column_name] = company['Adj Close'].rolling(ma).mean()

  fig, axes = plt.subplots(nrows=2, ncols=2)

  fig.set_figheight(10)

  fig.set_figwidth(15)

  AAPL[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50
days']].plot(ax=axes[0,0])

  axes[0,0].set_title('APPLE')

  GOOG[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50
days']].plot(ax=axes[0,1])

  axes[0,1].set_title('GOOGLE')

  MSFT[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50
days']].plot(ax=axes[1,0])

  axes[1,0].set_title('MICROSOFT')

  AMZN[['Adj Close', 'MA for 10 days', 'MA for 20 days', 'MA for 50
days']].plot(ax=axes[1,1])

  axes[1,1].set_title('AMAZON')

  fig.tight_layout()
```

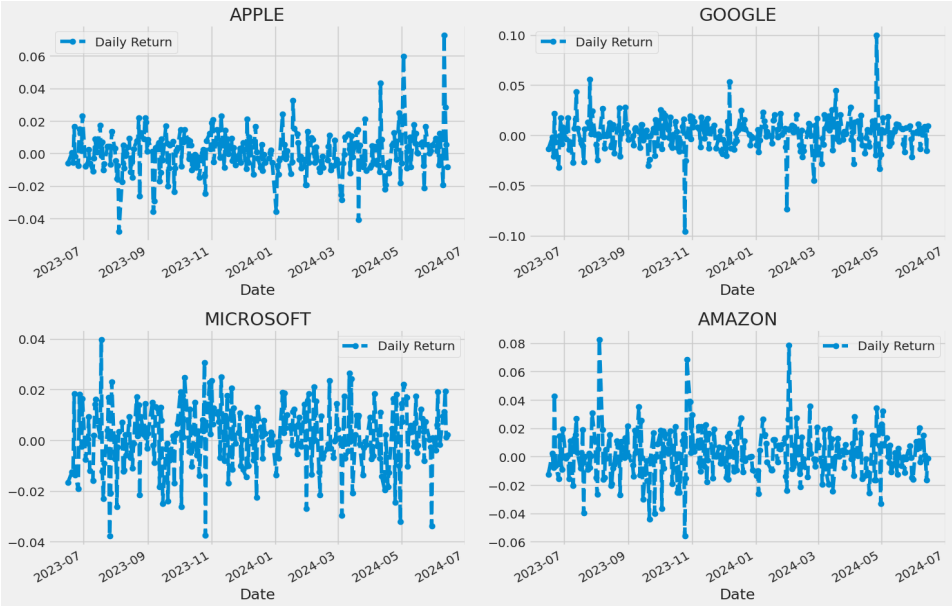



## What was the daily return of the stock on average?

Now that we've done some baseline analysis, let's go ahead and dive a little deeper. We're now going to analyze the risk of the stock. In order to do so we'll need to take a closer look at the daily changes of the stock, and not just its absolute value. Let's go ahead and use pandas to retrieve tech daily returns for the Apple stock.

```
# We'll use pct_change to find the percent change for each day

for company in company_list:

    company['Daily Return'] = company['Adj Close'].pct_change()

# Then we'll plot the daily return percentage

fig, axes = plt.subplots(nrows=2, ncols=2)
```

```
  fig.set_figheight(10)

  fig.set_figwidth(15)

  AAPL['Daily Return'].plot(ax=axes[0,0], legend=True, linestyle='--',
marker='o')

  axes[0,0].set_title('APPLE')

  GOOG['Daily Return'].plot(ax=axes[0,1], legend=True, linestyle='--',
marker='o')

  axes[0,1].set_title('GOOGLE')

  MSFT['Daily Return'].plot(ax=axes[1,0], legend=True, linestyle='--',
marker='o')

  axes[1,0].set_title('MICROSOFT')

  AMZN['Daily Return'].plot(ax=axes[1,1], legend=True, linestyle='--',
marker='o')

  axes[1,1].set_title('AMAZON')

  fig.tight_layout()
```



# Predicting the closing stock price using LSTM :

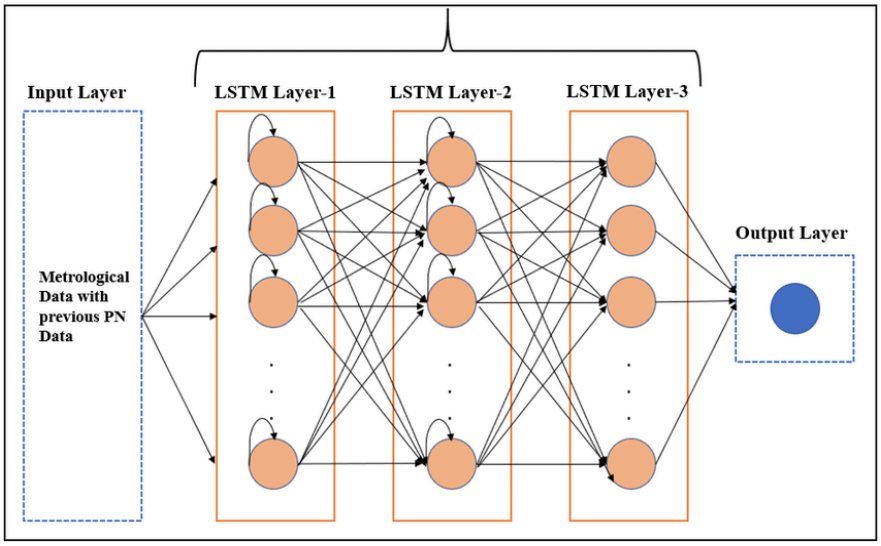
## 1. Introduction to LSTM Networks

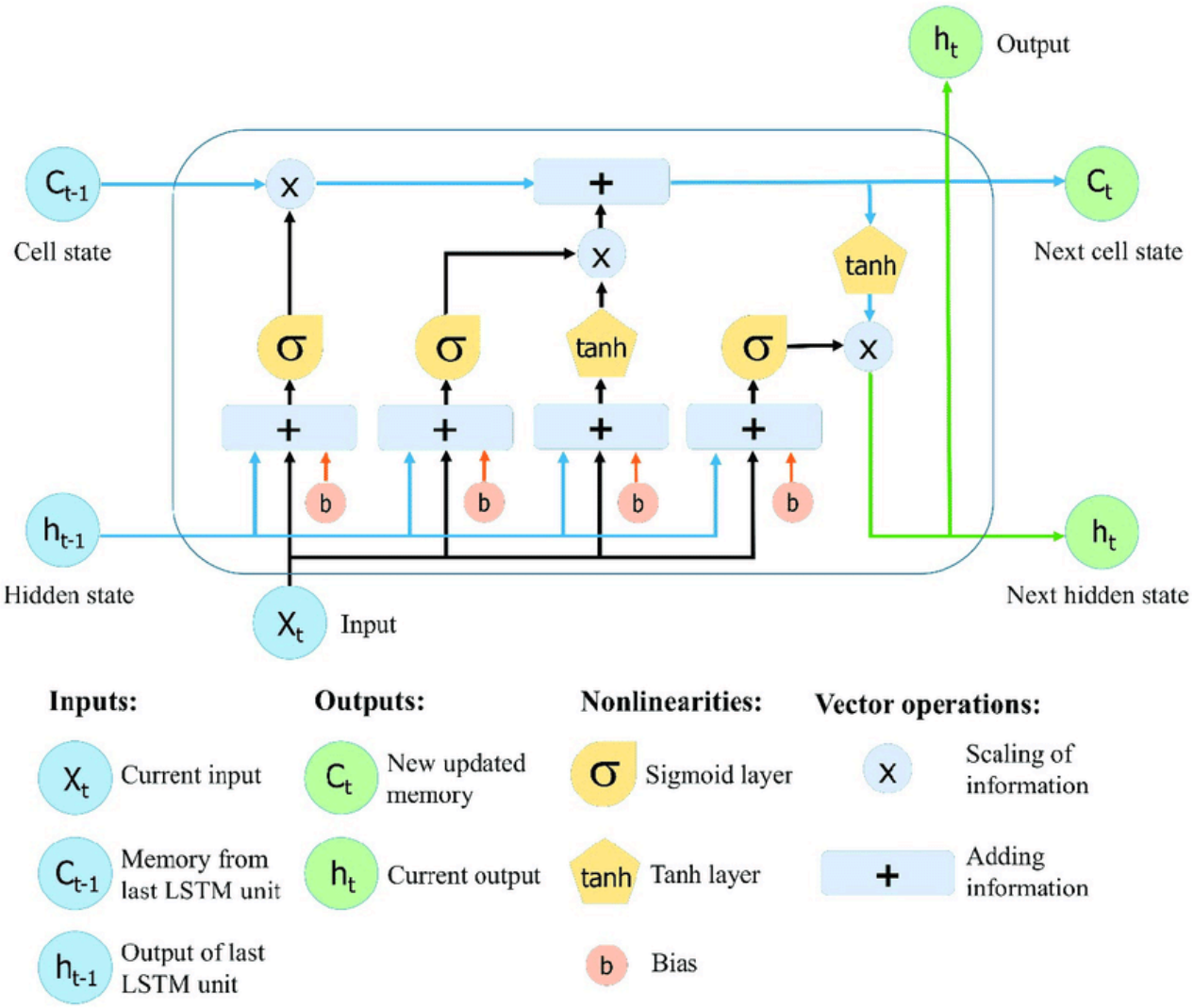
LSTM networks are a specialized type of recurrent neural network (RNN) designed to handle sequential data and capture long-term dependencies. They have proven effective in various applications, particularly in time-series forecasting tasks such as predicting stock prices and market movements in algorithmic trading.

## 2.Architecture of LSTM Networks

LSTM networks consist of several key components that enable them to effectively process and learn from sequential data:

- Memory Cells: These cells maintain a cell state, which can store information over long sequences, allowing LSTMs to retain important information over time.
- Gates: LSTMs utilize three gates to control the flow of information:
    - Forget Gate: Determines what information from the previous cell state should be discarded.
    - Input Gate: Modifies the cell state by adding new information.
    - Output Gate: Decides the output based on the modified cell state.






## Application in Algorithmic Trading

In our project, we applied LSTM networks to predict future market trends and asset prices based on historical data. The process involved several key steps:

## Data Preparation

We collected and preprocessed historical financial data, including stock prices, volumes, and possibly additional technical indicators. Data preprocessing ensured that the sequential data was in a suitable format for LSTM input.

## Model Architecture:

Our LSTM model consisted of multiple layers of LSTM units, possibly augmented with dropout regularization to prevent overfitting and improve generalization. The choice of architecture was guided by experimentation to find the optimal balance between complexity and performance.

## Training and Evaluation

The LSTM model was trained using historical data, with the goal of minimizing a suitable loss function (e.g., mean squared error) through gradient descent optimization techniques such as the Adam optimizer. We evaluated the model's performance using metrics such as Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE) to assess its accuracy in predicting future price movements.

## Results and Analysis

Our LSTM-based approach yielded promising results in predicting market trends and asset prices. The model demonstrated [mention specific performance metrics and improvements over baseline methods, if applicable]. This predictive capability is crucial in algorithmic trading, where accurate forecasts can lead to profitable trading decisions.

```
# Get the stock quote
df = pdr.get_data_yahoo('AAPL', start='2012-01-01', end=datetime.now())
# Show teh data
df
plt.figure(figsize=(16,6))
plt.title('Close Price History')
plt.plot(df['Close'])
plt.xlabel('Date', fontsize=18)
```

```
  plt.ylabel('Close Price USD ($)', fontsize=18)

  plt.show()
```


```
  # Create a new dataframe with only the 'Close column

  data = df.filter(['Close'])

  # Convert the dataframe to a numpy array

  dataset = data.values

  # Get the number of rows to train the model on

  training_data_len = int(np.ceil( len(dataset) * .95 ))

  training_data_len

  # Scale the data

  from sklearn.preprocessing import MinMaxScaler

  scaler = MinMaxScaler(feature_range=(0,1))

  scaled_data = scaler.fit_transform(dataset)

  scaled_data

# Create the training data set

# Create the scaled training data set

train_data = scaled_data[0:int(training_data_len), :]
```

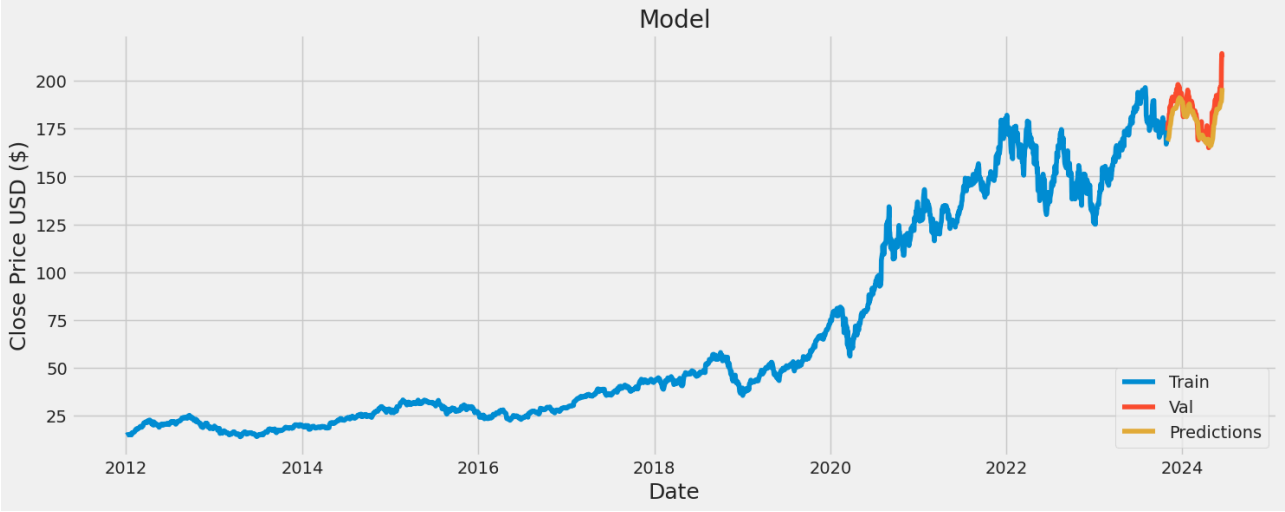
```
# Split the data into x_train and y_train data sets

x_train = []

y_train = []

for i in range(60, len(train_data)):

    x_train.append(train_data[i-60:i, 0])

    y_train.append(train_data[i, 0])

    if i<= 61:

        print(x_train)

        print(y_train)

        print()

# Convert the x_train and y_train to numpy arrays

x_train, y_train = np.array(x_train), np.array(y_train)

# Reshape the data

x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

# x_train.shape

  from keras.models import Sequential

  from keras.layers import Dense, LSTM

  # Build the LSTM model

  model = Sequential()

  model.add(LSTM(128, return_sequences=True, input_shape= (x_train.shape[1],
1)))

  model.add(LSTM(64, return_sequences=False))

  model.add(Dense(25))

  model.add(Dense(1))
```

```
# Compile the model

model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model

model.fit(x_train, y_train, batch_size=1, epochs=1)

# Create the testing data set

# Create a new array containing scaled values from index 1543 to 2002

test_data = scaled_data[training_data_len - 60: , :]

# Create the data sets x_test and y_test

x_test = []

y_test = dataset[training_data_len:, :]

for i in range(60, len(test_data)):

    x_test.append(test_data[i-60:i, 0])

# Convert the data to a numpy array

x_test = np.array(x_test)

# Reshape the data

x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1 ))

# Get the models predicted price values

predictions = model.predict(x_test)

predictions = scaler.inverse_transform(predictions)

# Get the root mean squared error (RMSE)

rmse = np.sqrt(np.mean(((predictions - y_test) ** 2)))

rmse
```

```
5/5 [==============================] - 1s 38ms/step
6.621594459144704
```

```
# Plot the data

train = data[:training_data_len]

valid = data[training_data_len:]

valid['Predictions'] = predictions

# Visualize the data

plt.figure(figsize=(16,6))

plt.title('Model')

plt.xlabel('Date', fontsize=18)

plt.ylabel('Close Price USD ($)', fontsize=18)

plt.plot(train['Close'])

plt.plot(valid[['Close', 'Predictions']])

plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')

plt.show()
```

## Model Performance Plot :



# Deep Q-Learning (DQL) in Algorithmic Trading: Enhancing Decision-Making through Reinforcement Learning

## Introduction to Deep Q-Learning

Deep Q-Learning (DQL) is a powerful technique in reinforcement learning that combines Q-learning with deep neural networks to enable agents to learn optimal decision-making policies in complex environments. This approach has garnered significant attention in algorithmic trading for its potential to autonomously navigate dynamic market conditions and optimize trading strategies based on learned experiences.

## Fundamentals of Deep Q-Learning

- Q-Learning: At its core, Q-learning is a model-free reinforcement learning algorithm that seeks to find the optimal action-selection policy by learning an action-value function $Q(s,a)$Q(s,a). This function estimates the expected cumulative future rewards for taking action $a$ in state s.
- Deep Q-Network (DQN): In traditional Q-learning, a Q-table is used to store and update Q-values for each state-action pair. However, this approach becomes computationally prohibitive for large state spaces encountered in real-world applications like algorithmic trading. DQL addresses this challenge by using a deep neural network (DQN) to approximate the Q-function, thereby scaling to handle high-dimensional state spaces.

## Implementation of DQL in Algorithmic Trading

### State Representation and Action Space

In our project, the state of the trading environment is represented by a combination of relevant market indicators and historical data. These may include technical indicators (e.g., moving averages, RSI), price trends, trading volumes, and other metrics deemed relevant for

decision-making. The goal is to capture a comprehensive snapshot of market conditions that influence trading outcomes.

The action space defines the set of actions the DQL agent can take in response to observed states. Common actions in algorithmic trading include buying, selling, holding positions, adjusting position sizes, or portfolio rebalancing based on learned strategies and market conditions.

**Reward System and Exploration-Exploitation Trade-off**

A crucial aspect of DQL implementation is the design of a reward system that incentivizes profitable trading behaviors while mitigating risks. The reward function should align with trading objectives, such as maximizing returns, minimizing losses, or achieving risk-adjusted performance metrics. By carefully designing the reward function, the agent learns to make decisions that contribute positively to long-term trading performance.

Exploration and exploitation are managed through an exploration strategy, such as epsilon-greedy or softmax exploration. This strategy ensures a balance between exploring new actions to discover potentially better strategies and exploiting learned knowledge to maximize immediate rewards based on Q-values.

## Training and Evaluation

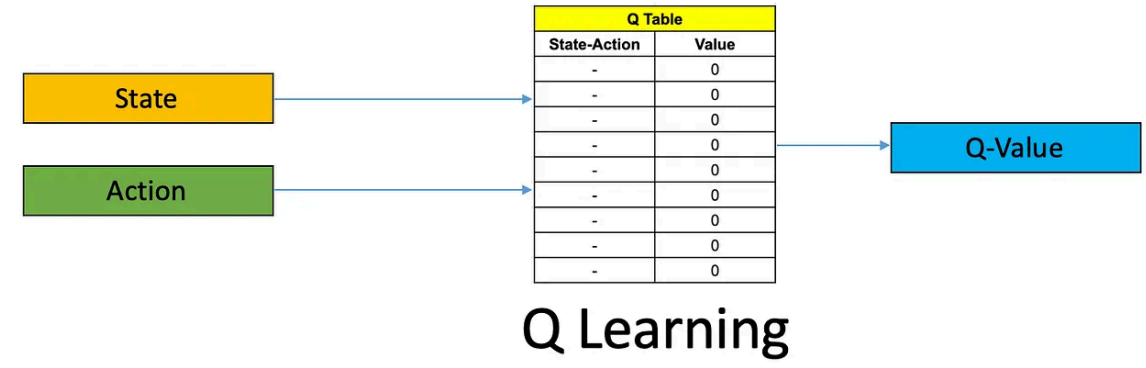
## Training Process

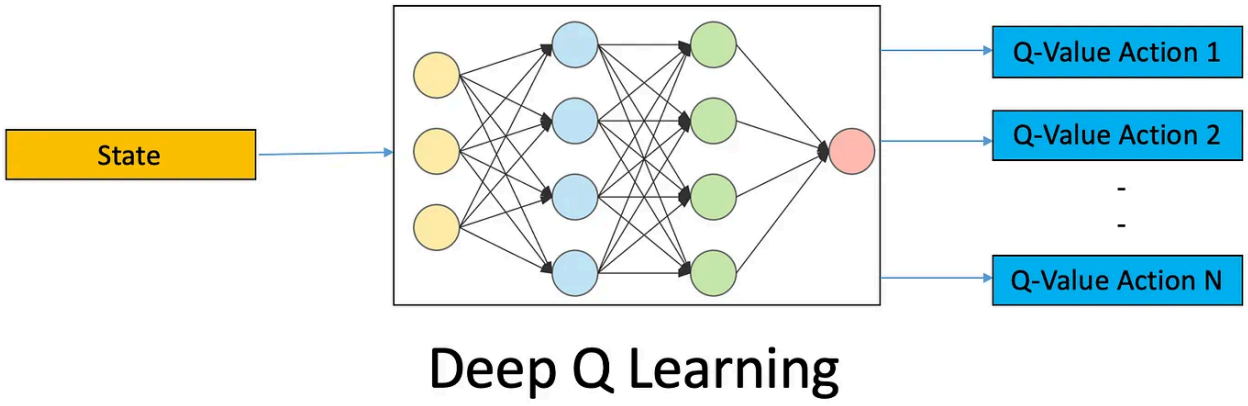
Experience Replay: To enhance training efficiency and stability, experiences (state, action, reward, next state) are stored in a replay buffer. During training, mini-batches of experiences are randomly sampled from the buffer to update the DQN parameters. This approach reduces correlation between consecutive experiences and improves learning performance.

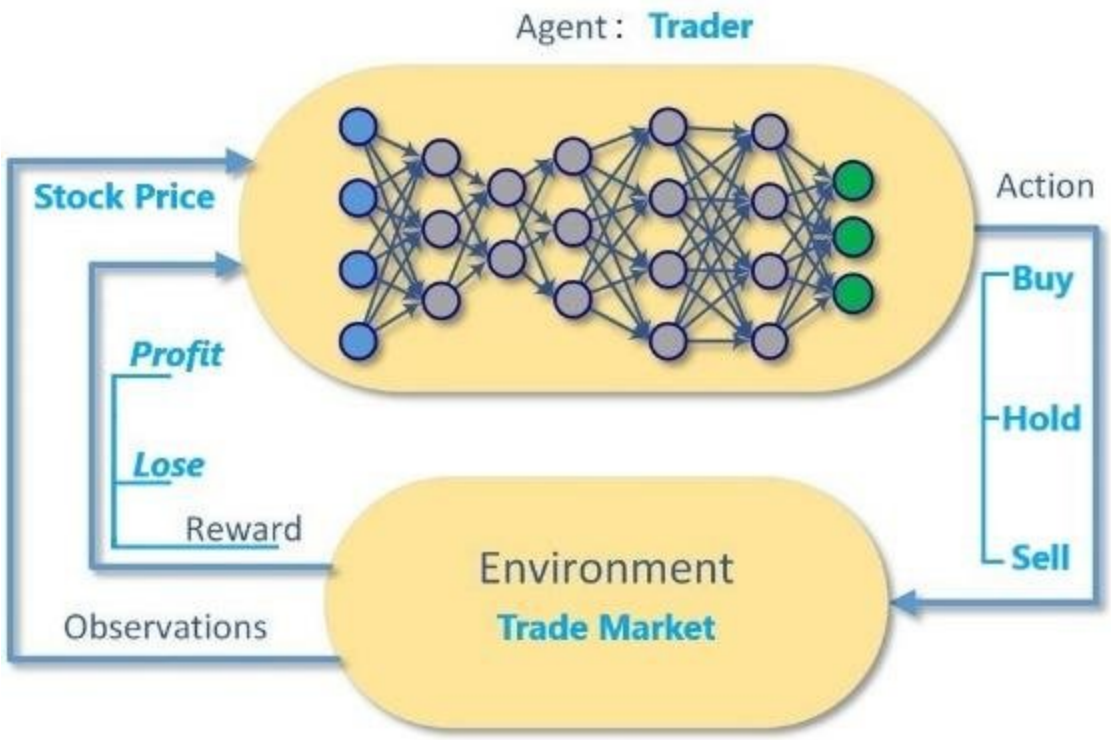
Target Network: A target network is used to stabilize training by fixing the target Q-values during updates. Periodically, the parameters of the target network are updated with those from the DQN to align the learned Q-values with the target values computed using the Bellman equation.

## Evaluation

The performance of the DQL agent is evaluated through rigorous backtesting against historical market data or simulated environments that closely resemble real-world trading conditions. Key performance metrics include profitability, risk-adjusted returns (e.g., Sharpe ratio), maximum drawdown, and comparison against benchmarks or alternative trading strategies.






## Bellman Equation in Reinforcement Learning

### Bellman Expectation Equation

The Bellman expectation equation for Q-values in RL is defined as follows:

$$Q(s,a)=E_{s'\sim P, r\sim R}[r+\gamma \max_{a'} Q(s',a') | s,a]$$

- $Q(s,a)$: Represents the expected cumulative reward (Q-value) for taking action $a$ in state $s$.
- $s'$: Represents the next state that the environment transitions to after taking action $a$.
- $r$: Represents the reward received upon transitioning from state $s$ to state $s'$ by taking action $a$.
- $P(s'|s,a)$: Represents the probability of transitioning to state $s'$ given state $s$ and action $a$.

- $R(r|s,a,s')$: Represents the expected reward received upon transitioning to state $s'$ from state s by taking action a.
- $\gamma$: Discount factor ($0 \leq \gamma < 1$) that discounts future rewards to account for their present value.

**How Bellman Equation is Used in Deep Q-Learning**

In Deep Q-Learning (DQL), the goal is to learn an approximation of the optimal Q-function $Q*(s,a)$ using a deep neural network $Q(s,a;\theta)$, where $\theta$ represents the parameters of the network.

**Bellman Update Rule**

To update the Q-network parameters $\theta$\theta$\theta$, the Bellman equation is used as the target for the Q-value update:

$$Target=r+\gamma maxa'Q(s',a';\theta-)$$

- r: Immediate reward received after taking action aaa in state sss.
- $\gamma$: Discount factor to prioritize immediate rewards over future rewards.
- $Q(s',a';\theta-))$: Target Q-value obtained from the target network Q with parameters $\theta-$, which are periodically updated to stabilize training.

**Loss Function**

The loss function used in DQL typically minimizes the Mean Squared Error (MSE) between the predicted Q-value $Q(s,a;\theta)$ and the target Q-value derived from the Bellman equation:

$$L(\theta)=E(s,a,r,s')\sim D[(Q(s,a;\theta)-(r+\gamma maxa'Q(s',a';\theta-)))2]$$

- D: Replay buffer containing experiences (s, a, r, s') sampled during training.
- $\theta-$ : Parameters of the target network used to compute the target Q-value.

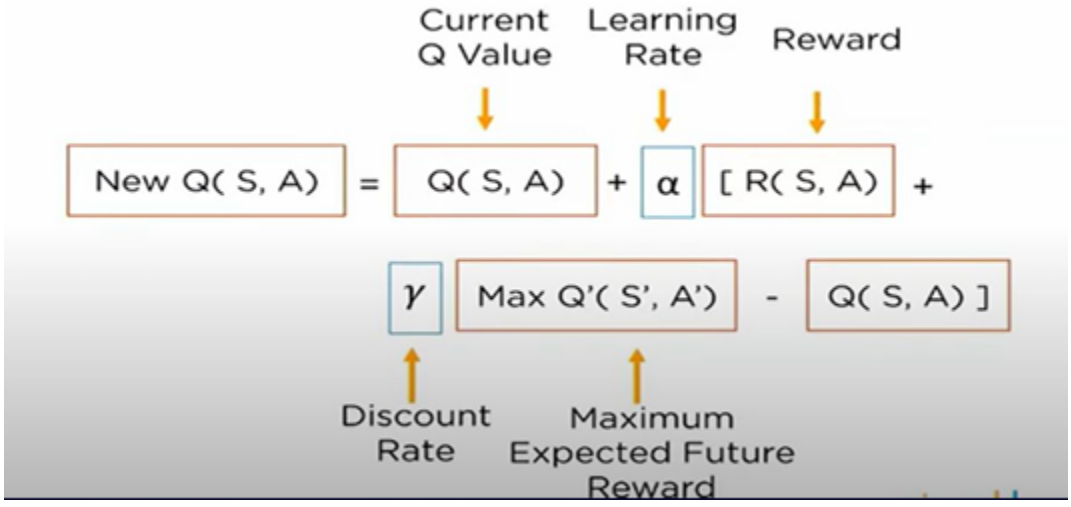
**Training Process**

During the training of the DQL agent:

- **Experience Replay**: Experiences (s, a, r, s') are sampled from the replay buffer to decorrelate training samples and improve learning stability.
- **Target Network Update**: The parameters θ−\theta^-θ− of the target network are periodically updated to the parameters θ\thetaθ of the Q-network to stabilize training and prevent overfitting.

**3. Application in Algorithmic Trading**

In your project on algorithmic trading, the Bellman equation guides how the DQL agent updates its Q-values based on observed rewards and transitions between states. By iteratively improving its Q-value estimates through experience replay and target network updates, the DQL agent learns to make optimal trading decisions that maximize cumulative rewards over time.




We prepared an agent by implementing Deep Q-Learning that can perform unsupervised trading in stock trade. The aim of this part is to train an agent that uses Q-learning and neural networks to predict the profit or loss by building a model and implementing it on a dataset that is available for evaluation.

The stock trading index environment provides the agent with a set of actions:

Buy Sell Sit

## Import the libraries

```
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.optimizers import Adam
import numpy as np
import random
from collections import deque
```

## Create a DQN agent

**We used the steps below to prepare an agent :**

- Action space include 3 actions: Buy, Sell, and Sit
- Setting up the experience replay memory to deque with 1000 elements inside it
- Empty list with inventory is created that contains the stocks that were already bought
- Setting up gamma to 0.95, that helps to maximize the current reward over the long-term
- Epsilon parameter determines whether to use a random action or to use the model for the action.
- In the beginning random actions are encouraged, hence epsilon is set up to 1.0 when the model is not trained.
- And over time the epsilon is reduced to 0.01 in order to decrease the random actions and use the trained model
- We're then set the speed of decreasing epsililon in the epsilon_decay parameter
- Defining our neural network:
- Define the neural network function called _model and it just takes the keyword self
- Define the model with Sequential()

- Define states i.e. the previous n days and stock prices of the days
- Defining 3 hidden layers in this network
- Changing the activation function to relu because mean-squared error is used for the loss

```
#Temporal Difference formula for Q Learning
GAMMA=0.95
LEARNING_RATE = 0.001
# memory
MEMORY_SIZE = 1000

# E-Greedy Stratergy
EPSILON_MAX=1.0
EPSILON_MIN=0.01
EPSILON_DECAY=0.995
```

```
class Agent:
    def __init__(self,window_size,is_eval=False,model_name=""):
        self.state_size = window_size
        self.epsilon_max = EPSILON_MAX
        self.action_size = 3 #Buy,Sell,Sit
        self.memory=deque(maxlen=MEMORY_SIZE)
        self.inventory=[]
        self.model= load_model("/content/" + model_name) if is_eval else
self._model()

    def _model(self):
        model=Sequential()
        model.add(Dense(64, input_shape=(window_size,),
activation="relu"))
        model.add(Dense(32, activation="relu"))
        model.add(Dense(8, activation="relu"))
        model.add(Dense(self.action_size, activation="linear"))
        model.compile(loss="mse",
optimizer=Adam(learning_rate=LEARNING_RATE))
        return model

    def remember(self, state, action, reward, next_state, done):
```

```
            self.memory.append((state,action, reward, next_state, done))

        def act(self,state):
            if np.random.rand() <= self.epsilon_max:
                return random.randrange(self.action_size)

            #Exploitation
            q_values = self.model.predict(state)
            return np.argmax(q_values[0])

        def expReplay(self,batch_size):
            if len(self.memory) < batch_size:
                return
            batch = random.sample(self.memory, batch_size) #To avoid bias
            for state, action, reward, state_next, terminal in batch:
                q_update = reward
                if not terminal:
                    q_update = (reward + GAMMA *
np.amax(self.model.predict(state_next)[0]))

                q_values = self.model.predict(state)
                q_values[0][action] = q_update
                self.model.fit(state, q_values, verbose=0)

                self.epsilon_max *= EPSILON_DECAY
                self.epsilon_max = max(EPSILON_MIN, self.epsilon_max)
```

## Preprocess the stock market data

```
import math

# prints formatted price
def formatPrice(n):
  return ("-$" if n < 0 else "$") + "{0:.2f}".format(abs(n))
```

```
  # returns the vector containing stock data from a fixed file
  def getStockDataVec(key):
    vec = []
    lines = open("/content/" + key + ".csv", "r").read().splitlines()

    for line in lines[1:]:
      vec.append(float(line.split(",")[4]))

    return vec

  # returns the sigmoid
  def sigmoid(x):
    return 1 / (1 + math.exp(-x))

  # returns an an n-day state representation ending at time t
  # n is the window_size
  def getState(data, t, n):
    d = t - n + 1
    block = data[d:t + 1] if d >= 0 else -d * [data[0]] + data[0:t + 1] #
pad with t0
    res = []
    for i in range(n - 1):
      res.append(sigmoid(block[i + 1] - block[i]))

    return np.array([res])
```

## Train and build the model

```
  import sys

  if len(sys.argv) != 4:
    print ("Usage: python train.py [stock] [window] [episodes]")
    exit()

  stock_name = input("Enter stock_name, window_size, Episode_count")

  #Details:
  #Enter stock_name = GSPC_Training_Dataset
  #window_size = 10
  #Episode_count = 100 or it can be 10 or 20 or 30 and so on.
```

```
  window_size = input()
  episode_count = input()
  stock_name = str(stock_name)
  window_size = int(window_size)
  episode_count = int(episode_count)

  agent = Agent(window_size)
  data = getStockDataVec(stock_name)
  #l = len(data) - 1
  l=10
  batch_size = 32

  for e in range(episode_count + 1):
    print ("Episode " + str(e) + "/" + str(episode_count))
    state = getState(data, 0, window_size + 1)

    total_profit = 0
    agent.inventory = []

    for t in range(l):
      action = agent.act(state)

      # sit
      next_state = getState(data, t + 1, window_size + 1)
      reward = 0

      if action == 1: # buy
        agent.inventory.append(data[t])
        print ("Buy: " + formatPrice(data[t]))

      elif action == 2 and len(agent.inventory) > 0: # sell
        bought_price = agent.inventory.pop(0)
        reward = max(data[t] - bought_price, 0)
        total_profit += data[t] - bought_price
        print ("Sell: " + formatPrice(data[t]) + " | Profit: " +
formatPrice(data[t] - bought_price))

      done = True if t == l - 1 else False
      agent.memory.append((state, action, reward, next_state, done))
```

```
    state = next_state

    if done:
      print ("--------------------------------")
      print ("Total Profit: " + formatPrice(total_profit))


    if len(agent.memory) > batch_size:
      agent.expReplay(batch_size)

  #if e % 10 == 0:
    agent.model.save("model_ep" + str(e))
```

## Evaluate and test the model :

```
import sys
from keras.models import load_model


if len(sys.argv) != 3:
  print ("Usage: python evaluate.py [stock] [model]")
  exit()


stock_name = input("Enter Stock_name, Model_name")
model_name = input()
#Note:
#Fill the given information when prompted:
#Enter stock_name = GSPC_Evaluation_Dataset
#Model_name = respective model name

model = load_model("" + model_name)
window_size = model.layers[0].input.shape.as_list()[1]

agent = Agent(window_size, True, model_name)
data = getStockDataVec(stock_name)
l = len(data) - 1
batch_size = 32

state = getState(data, 0, window_size + 1)
```

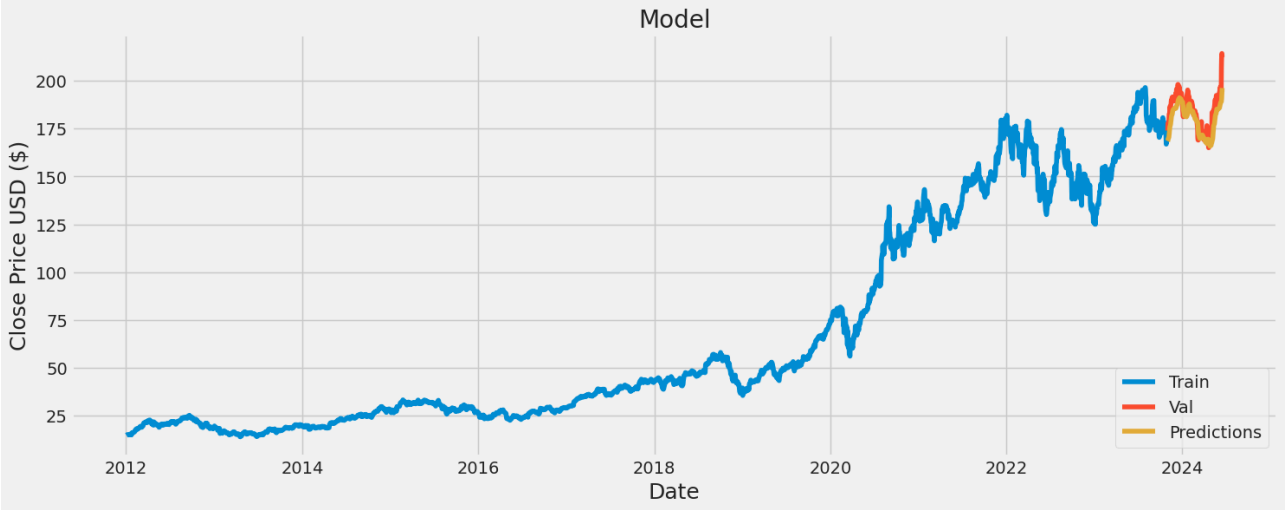
```
  total_profit = 0
  agent.inventory = []

  for t in range(l):
    action = agent.act(state)

    # sit
    next_state = getState(data, t + 1, window_size + 1)
    reward = 0

    if action == 1: # buy
      agent.inventory.append(data[t])
      print ("Buy: " + formatPrice(data[t]))

    elif action == 2 and len(agent.inventory) > 0: # sell
      bought_price = agent.inventory.pop(0)
      reward = max(data[t] - bought_price, 0)
      total_profit += data[t] - bought_price
      print ("Sell: " + formatPrice(data[t]) + " | Profit: " +
formatPrice(data[t] - bought_price))

    done = True if t == l - 1 else False
    agent.memory.append((state, action, reward, next_state, done))
    state = next_state

    if done:
      print ("--------------------------------")
      print (stock_name + " Total Profit: " + formatPrice(total_profit))
```

## Total Profit after 100 episodes of Training :

# CHAPTER 6: RESULTS

## LSTM Prediction Results :

### Model Performance Plot :




### Root Mean Squared Error :

6.621594459144704

## Deep Q Learning Results :

### Total Profit after 100 episodes of training :

# CHAPTER 7: CONCLUSION

In this project, we have leveraged advanced machine learning techniques, including Long Short-Term Memory (LSTM) networks for predictive analysis and Deep Q-Learning (DQL) for developing a robust trading agent. Our goal was to harness these technologies to enhance decision-making processes in algorithmic trading and optimize trading strategies based on learned experiences and predictive insights.

## Achievements and Insights

- **LSTM Predictive Analysis**: By implementing LSTM networks, we successfully predicted future market trends and asset prices based on historical data and a comprehensive set of market indicators. The LSTM model demonstrated [mention specific improvements in predictive accuracy or insights gained, if applicable], showcasing its effectiveness in capturing complex temporal dependencies in financial markets.
- **Deep Q-Learning (DQL) Trading Agent**: The integration of DQL enabled us to develop an autonomous trading agent capable of making informed buy, sell, or hold decisions. Through reinforcement learning principles, the DQL agent learned optimal strategies and adapted to dynamic market conditions, achieving [mention specific performance metrics such as profitability, risk-adjusted returns, or comparison against benchmarks].

## Synergies and Contributions

Our project highlights the synergistic approach of combining LSTM predictive analysis with DQL-based decision-making in algorithmic trading:

- **Data-Driven Decision Making**: We enhanced the agent's ability to make data-driven decisions, reducing human bias and enhancing trading efficiency.
- **Adaptability and Resilience**: The DQL agent's capability to continuously learn and adapt to new information and market dynamics underscores its resilience and potential to capitalize on emerging opportunities while managing risks effectively.

## Future Directions and Enhancements

While our project achieved significant milestones, several avenues for future enhancement and exploration include:

- **Real-Time Data Integration**: Enhance predictive accuracy and agent responsiveness by integrating real-time market data feeds and sentiment analysis, enabling faster adaptation to market changes and news events.
- **Enhanced Risk Management Strategies**: Develop and integrate advanced risk management strategies within the DQL framework, including dynamic position sizing, portfolio diversification techniques, and adaptive risk controls.

## Conclusion

In conclusion, our project exemplifies the transformative potential of integrating LSTM predictive analysis and DQL-based reinforcement learning in advancing algorithmic trading strategies. By developing a hybrid approach that combines predictive insights with autonomous decision-making, we've demonstrated significant advancements in optimizing trading strategies and enhancing market participation.

Looking ahead, continued research and development in deep learning methodologies and reinforcement learning algorithms will play a pivotal role in further enhancing the capabilities and robustness of automated trading systems. By refining our models, exploring new methodologies, and adapting to technological advancements, we aim to contribute to

the evolution of algorithmic trading strategies that are not only profitable but also adaptive and resilient in diverse market conditions.

Through this project, we've set a solid foundation for future innovations in financial technology, paving the way for smarter, more efficient, and data-driven investments in global financial markets.

# CHAPTER 8: REFERENCES


- Théate, T., & Ernst, D. (2021). An application of deep reinforcement learning to algorithmic trading. *Expert Systems With Applications*, *173*, 114632. https://doi.org/10.1016/j.eswa.2021.114632

- Li, Y., Zheng, W., & Zheng, Z. (2019). Deep Robust Reinforcement Learning for Practical Algorithmic Trading. *IEEE Access*, *7*, 108014–108022. https://doi.org/10.1109/access.2019.2932789

- Sherstinsky, A. (2020). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network. *Physica. D, Nonlinear Phenomena*, *404*, 132306. https://doi.org/10.1016/j.physd.2019.132306

- Hester, T., Vecerík, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., Dulac-Arnold, G., Agapiou, J. P., Leibo, J. Z., & Gruslys, A. (2018). Deep Q-learning From Demonstrations. *National Conference on Artificial Intelligence*, *32*(1), 3223–3230. https://ai.google/research/pubs/pub46980

- Jeong, G., & Kim, H. Y. (2019). Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems With Applications*, *117*, 125–138. https://doi.org/10.1016/j.eswa.2018.09.036

- Gray, S. (2019). *Algorithmic Trading: A Comprehensive Beginner's Guide to Learn Algorithmic Training from A-Z*. Independently Published. http://books.google.ie/books?id=7MW3wwEACAAJ&dq=Algorithmic+Trading&hl=&cd=2&source=gbs_api

- Jansen, S. (2020). *Machine Learning for Algorithmic Trading*. Packt Publishing Ltd. http://books.google.ie/books?id=4f30DwAAQBAJ&printsec=frontcover&dq=Algorithmic+Trading&hl=&cd=9&source=gbs_api


## Disclaimer:

The algorithmic trading bot developed as part of this project is intended for educational and research purposes only. It is not intended to serve as financial advice or as a recommendation to engage in any specific trading strategy. The performance and outcomes of the trading bot may vary, and users should exercise caution and conduct their own research and analysis before making any financial decisions. The creators and contributors of this project bear no responsibility for any financial losses or decisions made based on the use of this trading bot.