# Stock Prediction using MachineLearning

*CS354N: Minor Project*


Team Members:

1) Anjani kumar - 210001004
2) Shreyas Mahesh Honrao - 210001024
3) Priyanshu Jogdand - 210001055

# Introduction

Given the multifaceted nature of financial markets, accurately predicting stock prices is a significant task. This task involves considering factors such as market trends, company performance, and external events, making it inherently complex and uncertain.

This study aims to address the stock price prediction problem by employing three distinct machine learning models: **Long Short-Term Memory (LSTM)**, **Gated Recurrent Unit (GRU),** and **Linear Regression.** These models utilize historical stock prices and relevant features to make informed forecasts about future stock prices.

The core objective of this research is to conduct a comparative analysis of these models to ascertain their efficacy in stock price prediction. By evaluating the performance of LSTM, GRU, and Linear Regression models, we seek to discern which model offers the **highest predictive accuracy**.

# DATA

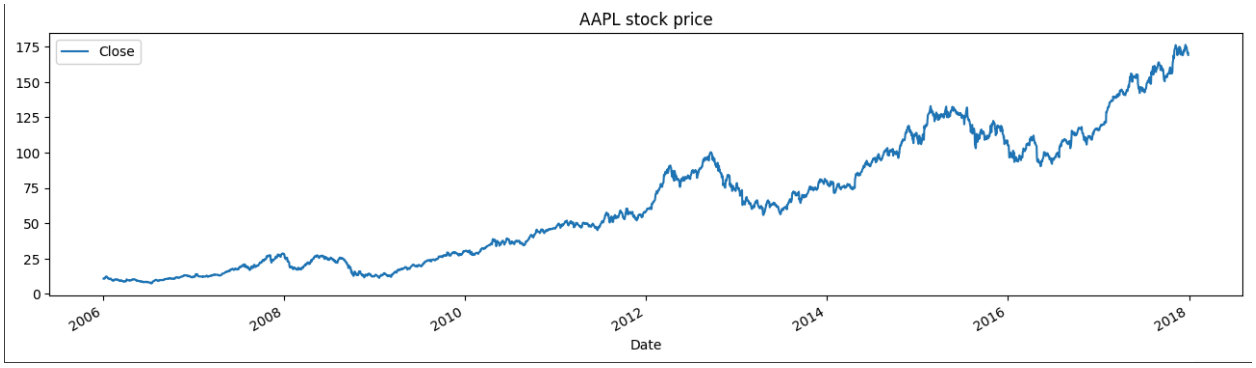
Data is the most essential thing in Stock Market Prediction.

We have extracted data from **Yahoo Finance Data**.

Data includes Stock prices of individual Stocks traded on the **NYSE**(New York Stock Exchange). Ex: **Apple, Amazon,** etc.

Input features include as follows:

- Our data includes seven columns, mainly **OHLC(Open, low, high, close), volume, name and date of that particular Stock.**
- Data has been cross-verified with the original prices of that stock traded on US Stock exchanges.



# DATA PIPELINE

In machine learning projects, following a data pipeline to prepare and preprocess the data to train the models accurately is essential. We followed a data pipeline that involved the following steps:

**Data Split**

We first split our dataset into **training and testing sets**. The training set was used to train the models, and the testing set was used to evaluate the models' performance. We used an 88-11 split, where 88% of the data was used for training, and 11% was used for testing.

**Feature Generation**

We Generated **synthesised data features** from the input data features to provide additional information to our models.

**Feature Scaling**

We Scaled the features to ensure that they had similar ranges and no feature dominated the others. Feature scaling is necessary because it ensures that the machine learning models give equal importance to all features. We used **Min-Max scaling** to scale our features, which scales the values between 0 and 1.

By following this data pipeline, we ensured that **our dataset was well-prepared and processed** and that our machine-learning models would be trained on high-quality and relevant data, leading to accurate predictions.

# MODEL

## 1) LSTM :

- Long Short-Term Memory (LSTM) is a type of **recurrent neural network** (RNN) architecture designed to address the vanishing gradient problem and capture long-term dependencies in sequential data.
- In stock market prediction, LSTM models can analyze historical stock prices along with other relevant features such as **trading volume, market sentiment, and technical indicators.**
- By learning patterns and relationships within this data, LSTM models can predict future stock prices or market trends.
- They achieve this by maintaining a **memory cell** that can retain information over long sequences, enabling them to capture both short-term fluctuations and longer-term trends in the stock market.
- LSTM models have been widely used in financial forecasting because they can handle **sequential data effectively** and uncover complex patterns that may influence stock prices.
- However, it's important to note that predicting stock prices accurately is challenging due to the inherent unpredictability and volatility of financial markets.
- LSTM models should be used as **part of a broader investment strategy** rather than relying solely on their predictions.

**2) GRU:**

- Gated Recurrent Unit (GRU) is another type of **recurrent neural network** (RNN) architecture, similar to LSTM, designed to address the limitations of traditional RNNs in capturing long-range dependencies in sequential data.
- GRU simplifies the LSTM architecture by **combining the forget and input gates into a single update gate** merging the cell and hidden states.
- This simplification makes GRU more computationally efficient while still enabling effective modeling of temporal dependencies.
- In the context of stock market prediction, GRU models function similarly to LSTM models by analyzing historical stock prices and other relevant data to predict future market trends.
- They can **capture both short-term fluctuations and longer-term patterns in the market, leveraging the gating mechanisms** to retain and update information over time selectively.
- GRU models have gained popularity in financial forecasting due to their simpler architecture and comparable performance to LSTM models.
- However, like LSTM models, GRU models should be used cautiously as part of a comprehensive investment strategy, as predicting stock prices accurately remains challenging and subject to various unpredictable factors in the financial markets.

**3) Linear Regression:**

- Linear Regression, a foundational machine learning algorithm, holds significance in stock market prediction due to its simplicity and interpretability.
- **It models the linear relationship between independent variables (historical stock prices, trading volume, and economic indicators) and a dependent variable (future stock prices or market trends).**
- By fitting a linear equation to historical data, Linear Regression aims to **minimize the difference between observed and predicted values.**
- In the context of stock market prediction, it can offer insights into how various factors influence stock prices, aiding in decision-making for investors and traders.
- Despite its simplicity, Linear Regression can provide valuable forecasts when used in conjunction with other techniques and factors.
- However, its performance may be limited by the **assumption of linearity** in the underlying data, and it may not capture complex nonlinear relationships present in financial markets.
- Thus, while Linear Regression serves as a useful tool in stock market prediction, it's essential to consider its limitations and complement it with other methods for more accurate forecasts.

# ALGORITHM

To train our machine learning models, we have used an algorithm that involves splitting the dataset into training and testing sets**.** We have used the training set to train our models and the testing set to evaluate their performance.

## Training

To ensure a fair and accurate fitting result, we have used a **sliding window** approach to training and testing. The default **window size taken is 60 days**.

### i) LSTM

Data Preprocessing:

- The code imports necessary libraries such as **pandas** for data manipulation, **numpy** for numerical operations, and **MinMaxScaler** from sklearn for scaling the data.
- The **Apple Inc stock data** is loaded from a CSV file into a pandas DataFrame (`df`), specifying the 'Date' column as the index and parsing dates.
- It plots the training set (data up to 2016) and the test set (data from 2017 onwards) using matplotlib.

Feature Scaling:

- MinMaxScaler is applied to scale the training set between 0 and 1. This is a common practice in machine learning when working with neural networks, as it helps in **faster convergence** and better performance.

Creating Training and Test Sets:

- The **'Close'** stock prices are extracted from the DataFrame for the training set (`training_set`) and test set (`test_set`).
- A loop is used to create the **input sequences** (`X_train`) and corresponding **output** (`y_train`) for the LSTM model. It takes the previous 60 stock prices to predict the next one.
- The input data is reshaped to be suitable for an LSTM model.

Building the LSTM Model:

- The LSTM model is constructed using **Keras Sequential API.**
- **Four LSTM layers** are stacked, each followed by a Dropout layer to prevent overfitting.
    - The first layer has 100 units, with a dropout rate of 0.3.
    - The second layer has 80 units, with a dropout rate of 0.1.
    - The third layer has 50 units, with a dropout rate of 0.2.

- The fourth layer has 30 units, with a dropout rate of 0.3.
- Finally, a Dense layer with 1 unit is added for the output.

Compiling and Training the Model:

- The model is compiled with **'adam' optimizer** and '**mean_squared_error**' loss, which is often used for regression problems.
- It's trained on the training data (`X_train`, `y_train`) for 5 epochs with a batch size of 32.

Making Predictions:

- For making predictions, the code combines the entire dataset to ensure the continuity of the input data.
- It then takes the last **60 days of scaled data** from the combined dataset (`inputs`), reshapes it, and predicts the stock prices for the test set (`X_test`).
- The predicted stock prices are **inversely transformed** back to their original scale (`predicted_stock_price`).

Visualization and Error Calculation:

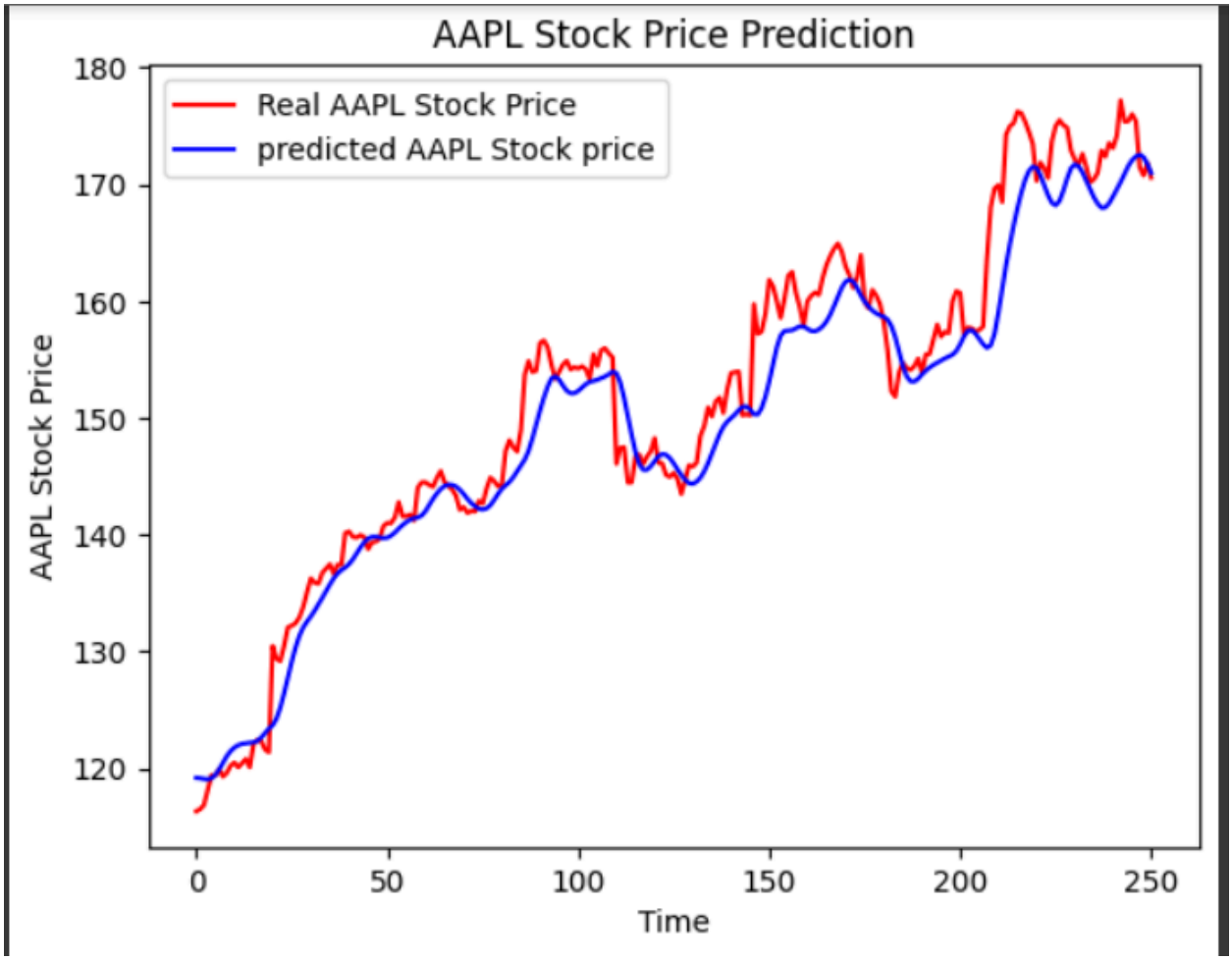
- The `plot_prediction` function is defined to **plot** the real AMZN stock prices (`test_set`) against the predicted prices (`predicted_stock_price`).

- The `return_rmse` function calculates the **root mean squared error** (RMSE) between the real and predicted stock prices.

Final Steps:

- The `plot_prediction` function is then used to visualize the real and predicted stock prices.
- Finally, the RMSE is calculated and printed to evaluate the performance of the model.

This algorithm utilizes LSTM, a type of **recurrent neural network** (RNN) architecture, which is particularly effective for sequence prediction tasks like stock price forecasting. It can capture patterns in the data over time, making it suitable for financial time series data. The model is trained on past stock prices and learns to predict the next day's price based on the patterns it discovers during training.


AAPL Stock Price Prediction
Real AAPL Stock Price
predicted AAPL Stock price
AAPL Stock Price
Time
180
170
160
150
140
130
120
0
50
100
150
200
250

## ii) GRU

Data Preprocessing:

- The code starts by importing necessary libraries: pandas for data manipulation, numpy for numerical operations, and MinMaxScaler from sklearn for scaling the data.
- The Apple stock data is loaded from a CSV file into a pandas DataFrame (`df`), specifying the 'Date' column as the index and parsing dates.
- It plots the training set (data up to 2016) and the test set (data from 2017 onwards) using matplotlib.

Feature Scaling:

- MinMaxScaler is applied to scale the training set between 0 and 1. This helps in improving the model's convergence and performance.

Creating Training and Test Sets:

- The '**High**' stock prices are extracted from the DataFrame for the training set (`training_set`) and test set (`test_set`).
- A loop is used to create the input sequences (`X_train`) and corresponding output (`y_train`) for the GRU model. It takes the previous 60 stock prices to predict the next one.
- The input data is reshaped to be suitable for a GRU model.

Building the GRU Model:

- The GRU model is constructed using **Keras Sequential API.**

- **Four GRU layers** are stacked, each followed by a Dropout layer to prevent overfitting:
    - The first layer has 100 units with a tanh activation function, with a dropout rate of 0.3.
    - The second layer has 80 units with a tanh activation function, with a dropout rate of 0.2.
    - The third layer has 50 units with a tanh activation function, with a dropout rate of 0.1.
    - The fourth layer has 30 units with a tanh activation function, with a dropout rate of 0.2.
- Finally, a Dense layer with 1 unit is added for the output.

Compiling and Training the Model:

- The model is compiled with **'adam' optimizer** and '**mean_squared_error**' loss, which is common for regression problems.
- It's trained on the training data (`X_train`, `y_train`) for 5 epochs with a batch size of 150.

Making Predictions:

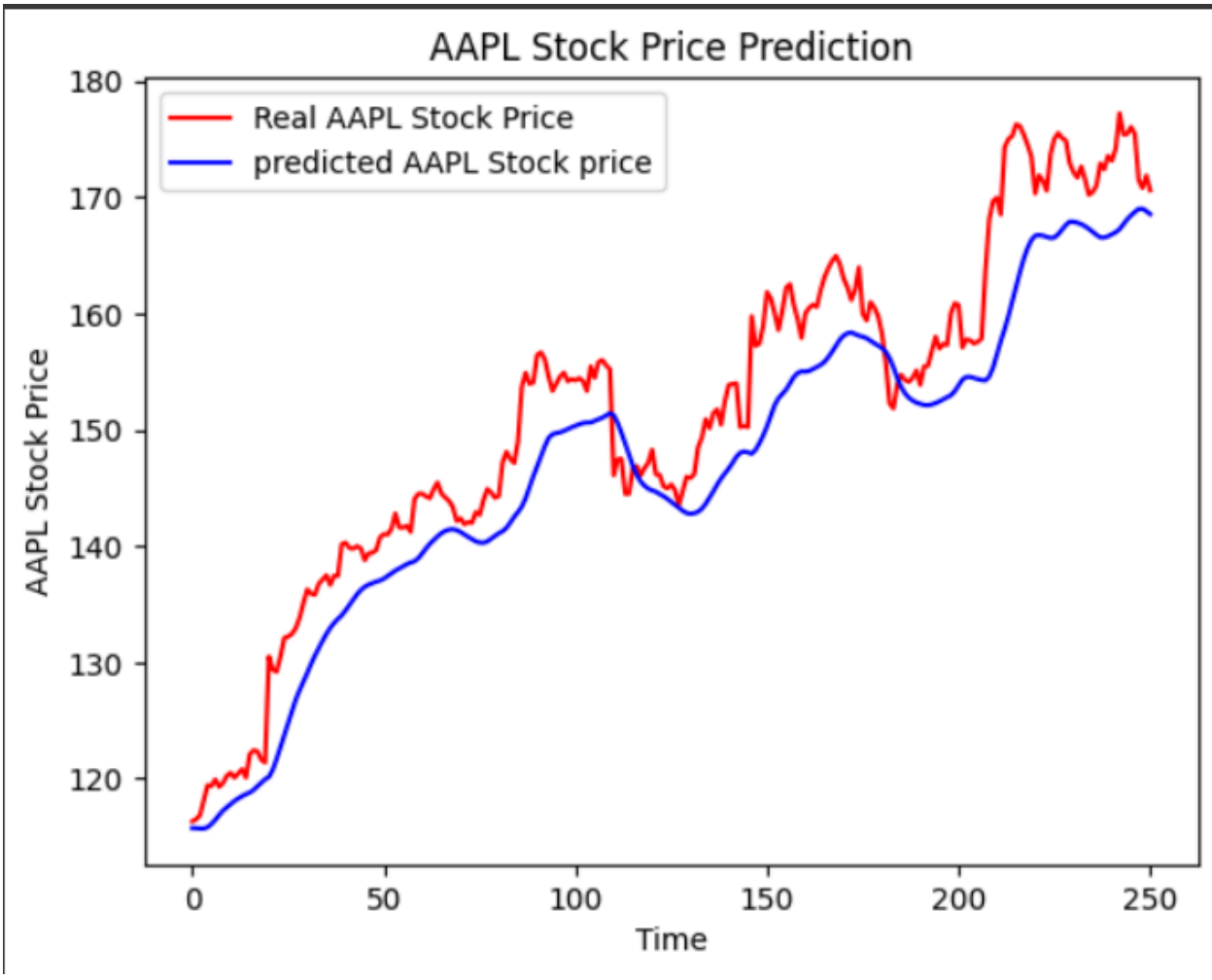
- For making predictions, the code combines the entire dataset to ensure continuity of input data.
- It then takes the last **60 days** of scaled data from the combined dataset (`inputs`), reshapes it, and predicts the stock prices for the test set (`X_test`).
- The predicted stock prices are inversely transformed back to their original scale (`predicted_stock_price`).

Visualization and Error Calculation:

- The `plot_prediction` function is defined to **plot** the real AAPL stock prices (`test_set`) against the predicted prices (`predicted_stock_price`).
- The `return_rmse` function calculates the root mean squared error (RMSE) between the real and predicted stock prices.

Final Steps:

- The `plot_prediction` function is then used to visualize the real and predicted stock prices.
- Finally, the RMSE is calculated and printed to evaluate the performance of the model.


AAPL Stock Price Prediction
Real AAPL Stock Price
predicted AAPL Stock price
AAPL Stock Price
Time
180
170
160
150
140
130
120
0
50
100
150
200
250

## iii) LINEAR REGRESSION

Data Loading and Initial Exploration:

- The code starts by importing necessary libraries such as pandas, matplotlib, numpy, seaborn, etc.
- The Apple stock data is loaded from a CSV file into a pandas DataFrame (`df_Stock`), containing historical stock prices.
- Initial exploratory steps are taken, such as checking the shape of the data, plotting the 'Close' prices over time, and displaying the first few and last few rows of the DataFrame.

Data Preprocessing:

- The 'Date' and 'Name' columns are dropped from the DataFrame since they are not needed for the modeling.
- A function `create_train_test_set` is defined to **split** the data into training, validation, and test sets.
    - Features are defined as all columns except 'Close', which is the target variable.
    - The data is split into roughly 88% training, 10% validation, and 2% test sets.
    - Shapes of the resulting sets are printed for verification.

Training the Linear Regression Model:

- A **Linear Regression model** (`lr`) is created and fitted on the training data (`X_train`, `Y_train`).
- The coefficients and intercept of the linear regression model are printed.

- The performance of the model is evaluated using the R-squared score on the training data.
- A custom function `get_mape` is defined to calculate Mean Absolute Percentage Error (MAPE) for later use.

Predictions and Model Evaluation:

- Predictions are made on the training (`X_train`), validation (`X_val`), and test (`X_test`) sets.
- Performance metrics such as R-squared, Explained Variance, MAPE, Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) are calculated for each set.
- These metrics help in evaluating how well the model is performing on different sets.

Visualization:

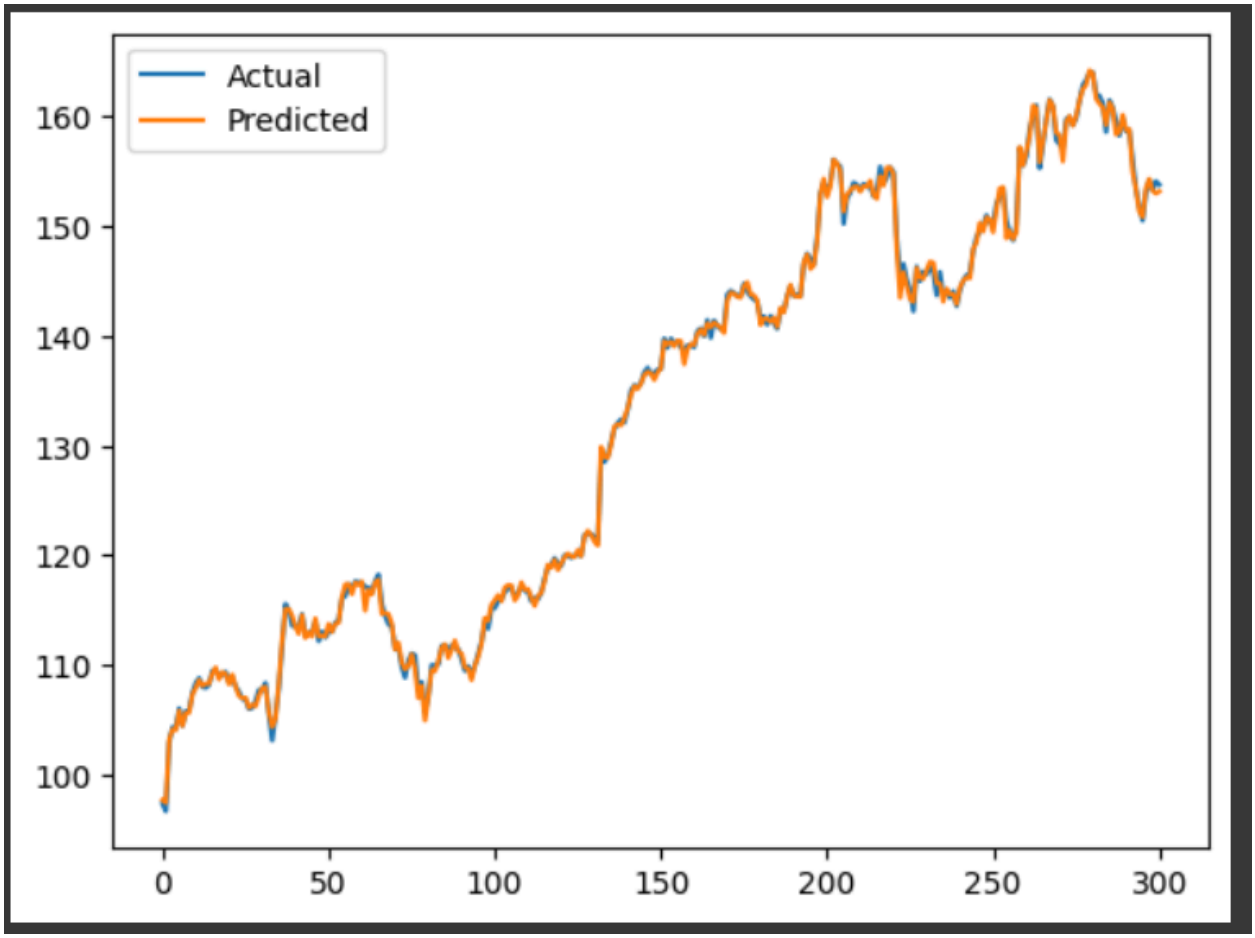
- A DataFrame `df_pred` is created to compare actual vs. predicted values on the validation set.
- The 'Actual' and 'Predicted' values are plotted over time to visually inspect the model's predictions against the true values.

Calculating Returns:

- A function `calculate_returns` is defined to calculate daily returns from the predicted stock prices.
- This function calculates the percentage change between consecutive days' prices and returns a list of daily returns.

The algorithm uses a simple Linear Regression model to predict stock prices based on historical data. Here's a summary of the performance metrics:

- Training Set:
    - R-squared: Indicates how well the model fits the training data.
    - Explained Variance: Measures the proportion of the variance in the target variable that the model explains.
    - MAPE: Mean Absolute Percentage Error, indicating the average percentage error in predictions.
    - MSE, RMSE, MAE: Measures of the error between predicted and actual values.
- Validation Set:
    - Similar metrics as the training set to assess how well the model generalizes to new data.
- Test Set:
    - Final evaluation on a small test set to get an idea of how the model performs on unseen data.


Actual
Predicted
160
150
140
130
120
110
100
0
50
100
150
200
250
300

# RESULT

Among all 3 models Linear Regression performs the best on test data by comparing RMSE  with LSTM and GRU models.

**i) LSTM**

```
The root mean sqaured error is 3.5354974652305122.
```

**ii) GRU**

```
The root mean sqaured error is 5.2903599952189365.
```

**iii) LR**

```
Test R-squared:  0.99
Test Explained Variation:  0.99
Test MAPE: 0.24
Test Mean Squared Error: 0.28
Test RMSE:  0.53
Test MAE:  0.4
```

# Performance Metrics:

**1)Root mean square error:**

Root Mean Square Error (RMSE) is a key metric for assessing model accuracy in stock market prediction. It quantifies the average deviation between predicted and actual values, with lower RMSE indicating closer alignment of predictions with real market behavior. Comparing RMSE values across models helps investors gauge which provides more accurate forecasts for informed investment decisions.

.

**2)Stock Returns on daily basis:**

Stock daily returns represent the percentage change in a stock's price from one trading day to the next. Calculating daily returns is a common practice in financial analysis and is essential for various tasks such as risk assessment, portfolio optimization, and performance evaluation. Daily returns are calculated using the following formula:

$$\text{Daily Return} = \frac{\text{Price}_t - \text{Price}_{t-1}}{\text{Price}_{t-1}} \times 100$$

**Conclusion:**

Used 3 models to compare the metrics for performance and we came to a conclusion that LR works best in comparison with LSTM and GRU.

**Github Repository:**
https://github.com/ShreyasHonrao/Computational-Intelligence-Project

## REFERENCES


https://link.springer.com/article/10.1007/s13042-019-01041-1

https://www.hindawi.com/journals/sp/2021/4055281/

https://search.proquest.com/openview/6d3e7bbf0c63bea87b9e2051f94b3e71/1?pq-origsite=gscholar&cbl=236248

https://medium.com/@anishnama20/understanding-gated-recurrent-unit-gru-in-deep-learning-2e54923f3e2

https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/