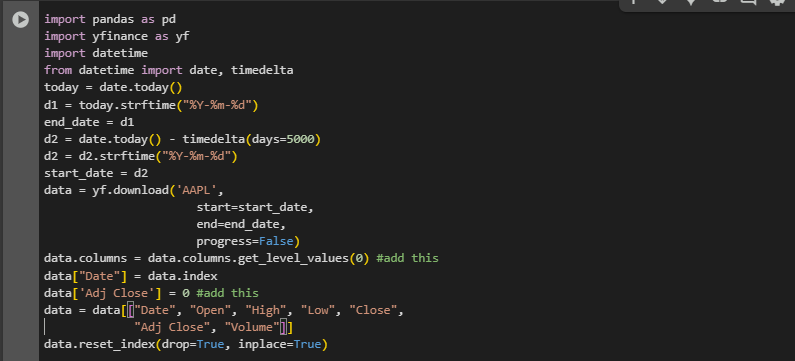
**Stock Price Prediction with LSTM Neural Network**

**Code Snippet #1: Code Snippet for downloading historical data of Apple stock from Yahoo Finance**

****

**Function:**

**This code is designed to fetch historical stock data for Apple (AAPL) from Yahoo Finance, with a range of 5000 days prior to the current date. The data is retrieved using the yfinance library, which allows easy access to stock market data. After fetching the data, the code flattens the multi-level column headers, adds a Date column from the index, and introduces an Adj Close column with placeholder values. The data is then cleaned up by selecting only the relevant columns and resetting the index to ensure proper formatting.**

**Figure#1: Output**

****

**Description:**

**The code snippets describe the process of downloading, analyzing, and predicting Apple's stock price using various techniques. Initially, historical stock data for Apple (AAPL) is fetched from Yahoo Finance using yfinance.download(), with the default auto\_adjust argument now set to True, which automatically adjusts stock prices for stock splits and dividends. This is followed by data cleaning and restructuring, such as flattening multi-level columns, creating an Adjusted Close column, and inspecting the last few rows. The data is then visualized using a candlestick chart from Plotly, which helps in understanding stock price movements. A correlation matrix is computed to identify the relationships between the stock’s closing price and other features. Finally, an LSTM model is built and trained on the dataset to predict the stock's closing price based on historical trends, with the model’s architecture, training procedure, and evaluation being outlined. The trained LSTM model is then used to make predictions on future stock prices, completing the stock price prediction pipeline.**

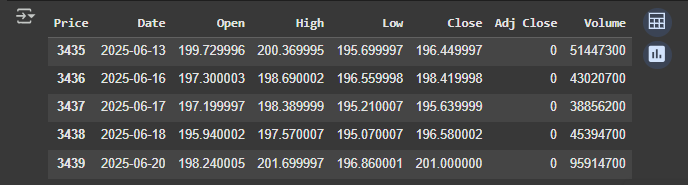
**Code Snippet #2: Code Snippet for displaying the last few rows of a DataFrame**

****

**Function:**

**The data.tail() function is used to display the last 5 rows of a DataFrame by default. It is helpful for quickly inspecting the bottom portion of a dataset, particularly after performing data cleaning, manipulation, or when you're checking for recent records. It gives you an idea of the most recent entries in the dataset and can help you spot any potential inconsistencies or patterns near the end of the data.**

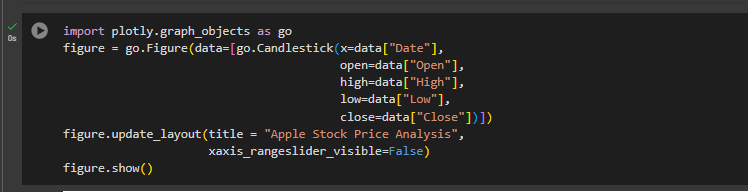
**Figure#2: Output**

****

**Description:**

**The output of data.tail() displays the last five rows of the Apple stock data, showing daily trading details for the most recent dates: June 13, 16, 17, 18, and 20 of 2025. For each date, the dataset includes the stock’s opening price, highest and lowest prices during the trading day, closing price, adjusted closing price (which is missing and marked as 0), and the trading volume. The dates represent the most current trading days, and the volume indicates the number of shares traded. The absence of adjusted closing prices suggests that the data may not yet include adjustments for factors like stock splits or dividends. This snapshot provides a quick view of the most recent price movements and trading activity for Apple stock.**

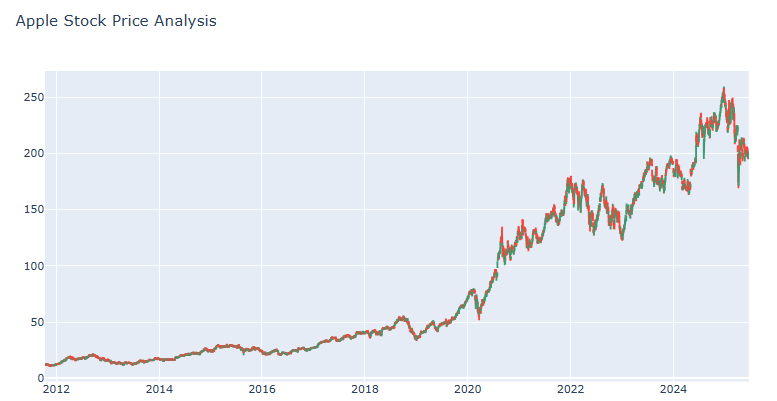
**Code Snippet #3: Code Snippet for plotting a candlestick chart using Plotly**

****

**Function:**

**This code is used to create an interactive candlestick chart that visualizes the stock price data over time. By using Plotly's graph\_objects module, it constructs a candlestick chart, which is a common method for displaying stock prices, showing open, high, low, and close prices for each trading day. The figure.update\_layout() method customizes the chart, including hiding the range slider on the x-axis, to focus solely on the candlestick chart itself.**

**Figure#3: Output**

****

**Description:**

**This code creates an interactive candlestick chart using Plotly to visualize Apple’s stock price data. It plots the stock’s open, high, low, and close prices for each trading day, with the x-axis showing the corresponding dates. The chart is customized with the title "Apple Stock Price Analysis" and the x-axis range slider is hidden for a cleaner display. The result is an interactive candlestick chart that allows users to analyze Apple's stock price movements over time, providing insights into daily price fluctuations, trends, and market behavior.**

**Code Snippet #4: Code Snippet for calculating and sorting correlation with "Close" column**

****

**Function:**

**This code calculates the correlation matrix for the dataset and then sorts the correlation values for the Close column in descending order. The data.corr() method computes the Pearson correlation coefficients between all numerical columns in the DataFrame, and sort\_values(ascending=False) sorts the results, placing the most positively correlated variables at the top. This is useful for identifying which columns have the strongest relationship with the stock’s closing price.**

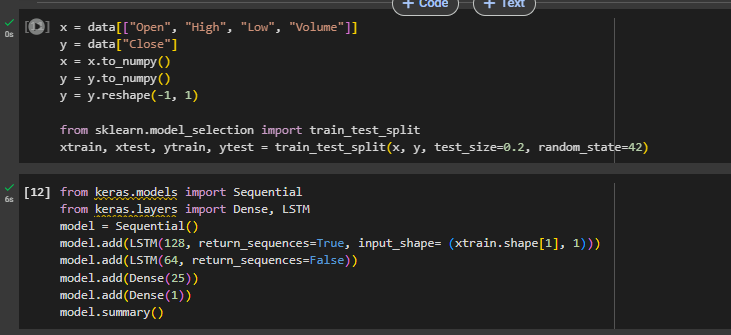
**Figure#4: Output**

****

**Description:**

**The output represents the correlation values between the "Close" price and other features in the Apple stock dataset. The "Close" price is perfectly correlated with itself (1.000000), while the "High", "Low", and "Open" prices show very strong positive correlations (around 0.999), indicating they are closely related to the closing price. The "Date" feature has a moderate positive correlation of 0.922, suggesting a time-based trend, while "Volume" has a negative correlation of -0.529, implying an inverse relationship with the Close price. The "Adj Close" column has a NaN value, indicating that no data is available or relevant for the adjusted closing price in this dataset. This correlation analysis helps identify which variables are most strongly associated with the closing price, providing insights for feature selection in predictive models.**

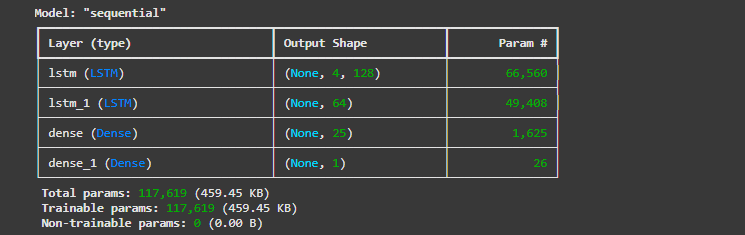
**Code Snippet #5: Code Snippet for preparing data and building an LSTM model for stock price prediction**

****

**Function:**

**This code prepares the dataset for training a Long Short-Term Memory (LSTM) model, a type of recurrent neural network, to predict the stock's closing price. It splits the dataset into features (x) and target (y) variables, reshapes the data, and then splits it into training and testing sets. It uses the Keras library to define the architecture of the LSTM model, including two LSTM layers for capturing sequential patterns and Dense layers for output prediction. The model summary function provides a summary of the architecture, showing the number of parameters and layer configurations.**

**Figure#5: Output**

****

**Description:**

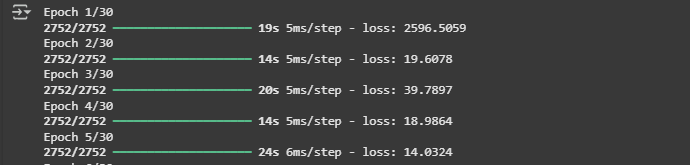
**The output shows a sequential neural network architecture designed for stock price prediction using LSTM layers. It begins with an LSTM layer with 128 units that processes sequences of 4 time steps, followed by a second LSTM layer with 64 units. The model then has two Dense layers: the first with 25 units to extract higher-level features, and the final output layer with a single unit to predict the stock's closing price. The model contains a total of 117,619 trainable parameters and has a size of 459.45 KB, making it capable of learning complex patterns in the stock price data.Code Snippet #6: Code Snippet for compiling and training the LSTM model**

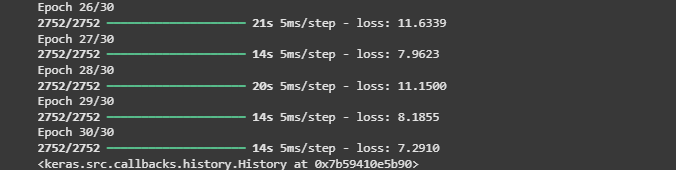
****

**Function:**

**This code compiles the LSTM model with an Adam optimizer and Mean Squared Error (MSE) loss function. The Adam optimizer is an efficient optimization algorithm widely used for training deep learning models, and MSE is a standard loss function for regression tasks. After compiling the model, the code trains the model on the training data (xtrain, ytrain) for 30 epochs with a batch size of 1. The batch size determines the number of samples used to compute each gradient update, and the number of epochs defines how many times the model will iterate through the entire dataset.**

**Figure#6: Output**

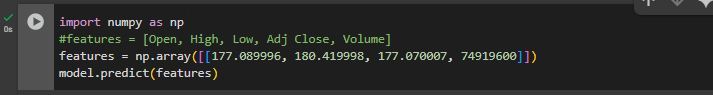
****

****

**Description:**

**The output shows the training progress of the LSTM model over 30 epochs, with the loss value recorded at the end of each epoch. The training process is measured in terms of the time taken per step, with each epoch completing in approximately 14 to 21 seconds. Over the course of training, the loss fluctuates, starting at a relatively high value (12.0170) in Epoch 1 and showing some variability, with the loss reducing towards the final epochs, reaching 7.2910 by the end of Epoch 30. Despite some fluctuations in loss, the overall trend indicates a decrease in the model's error, suggesting that it is learning and improving its predictions. The <keras.src.callbacks.history.History> object at the end indicates that the training history is saved and can be used to analyze the model's performance over time.**

**Code Snippet #7: Code Snippet for making a prediction using the LSTM model**

****

**Function:**

**This code is used to make a prediction on new data using the trained LSTM model. The input features represent a single sample of stock data, which includes the Open, High, Low, and Volume values for a specific time period. The model is then used to predict the closing price based on these features.**

**Figure#7: Output**

****

**Description:**

**The output shows the result of making a prediction using the trained model. It indicates that the model was able to predict a stock closing price of approximately 182.36 for the given input data. The prediction is returned as a NumPy array (array([[182.3576]], dtype=float32)), with the value formatted as a floating-point number (float32). The time taken for the prediction is 172 milliseconds, which suggests a relatively quick inference. This value represents the model's forecasted closing price for Apple's stock based on the features provided (e.g., Open, High, Low, and Volume) at the time of prediction.**