

**Real-Time Stock Data Processing**

***Overview***

This project creates a system to collect and analyze stock market data and news in real time. It uses Apache Spark to process data from Kafka and saves the results in Cassandra. The goal is to provide quick alerts based on stock price changes and analyze the sentiment of news articles.

**Description**

**1. Data Ingestion**

* Kafka Integration: The system reads stock and news data from Kafka, allowing it to process information as it arrives.
* Schema Definition: We set up a clear structure for the incoming data to ensure it is properly processed.

**2. Data Processing**

* Spark Streaming: We use Apache Spark to handle data continuously, processing it in real time.
* Sentiment Analysis: We use the TextBlob library to analyze the feelings expressed in news summaries. A custom function calculates how positive or negative the news is.

**3. Data Storage**

* Cassandra and MongoDB Integration: The processed data is saved in Cassandra and MongoDB, a NoSQL database, which allows for fast storage and easy retrieval of information.

**4. Real-Time Alerts**

* The system checks for significant changes in stock prices and sends alerts if prices change a lot. This is useful for traders who need to act quickly.

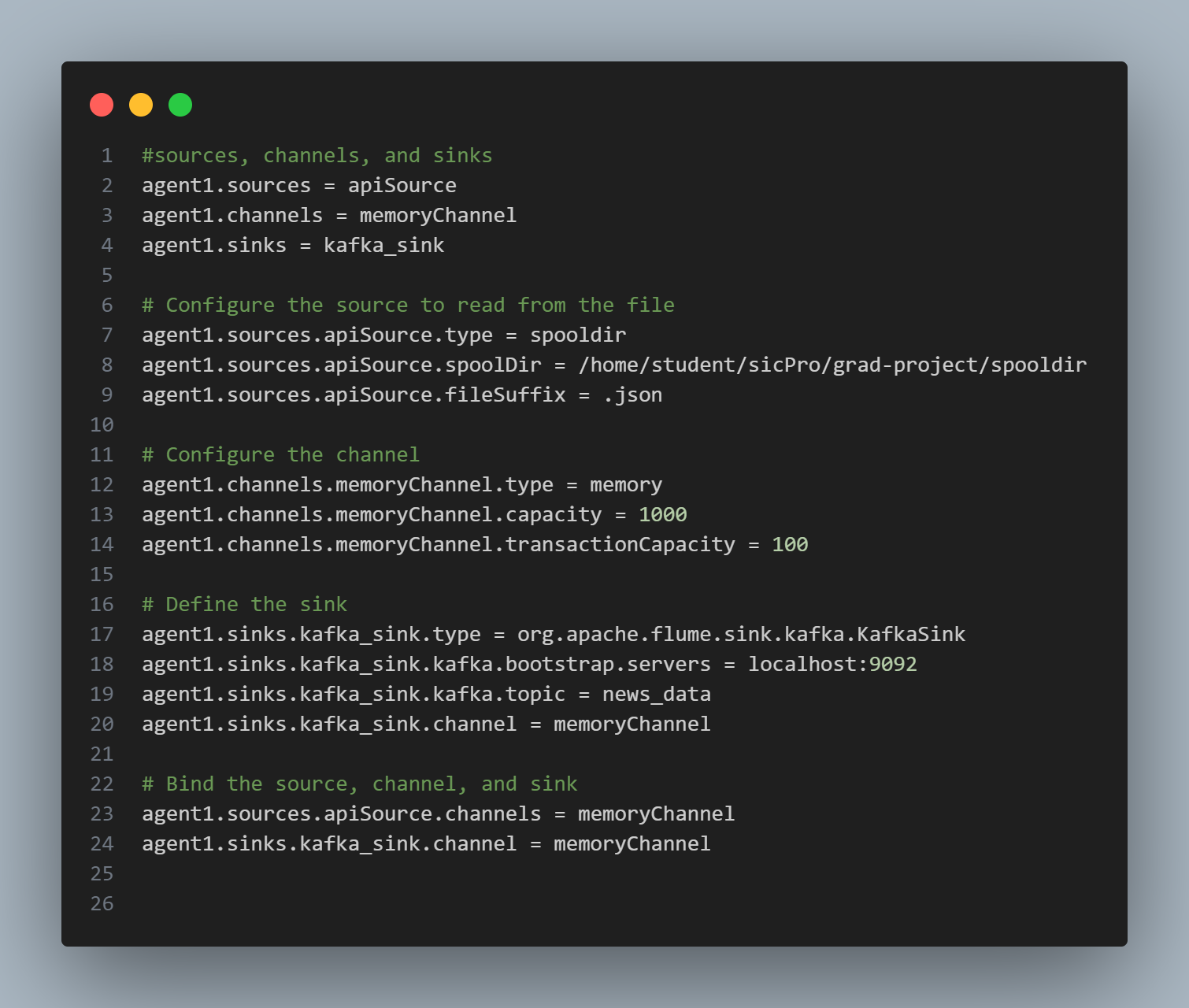
**Stock Market Data Streaming to Kafka**

A screen shot of a computer program

Description automatically generated

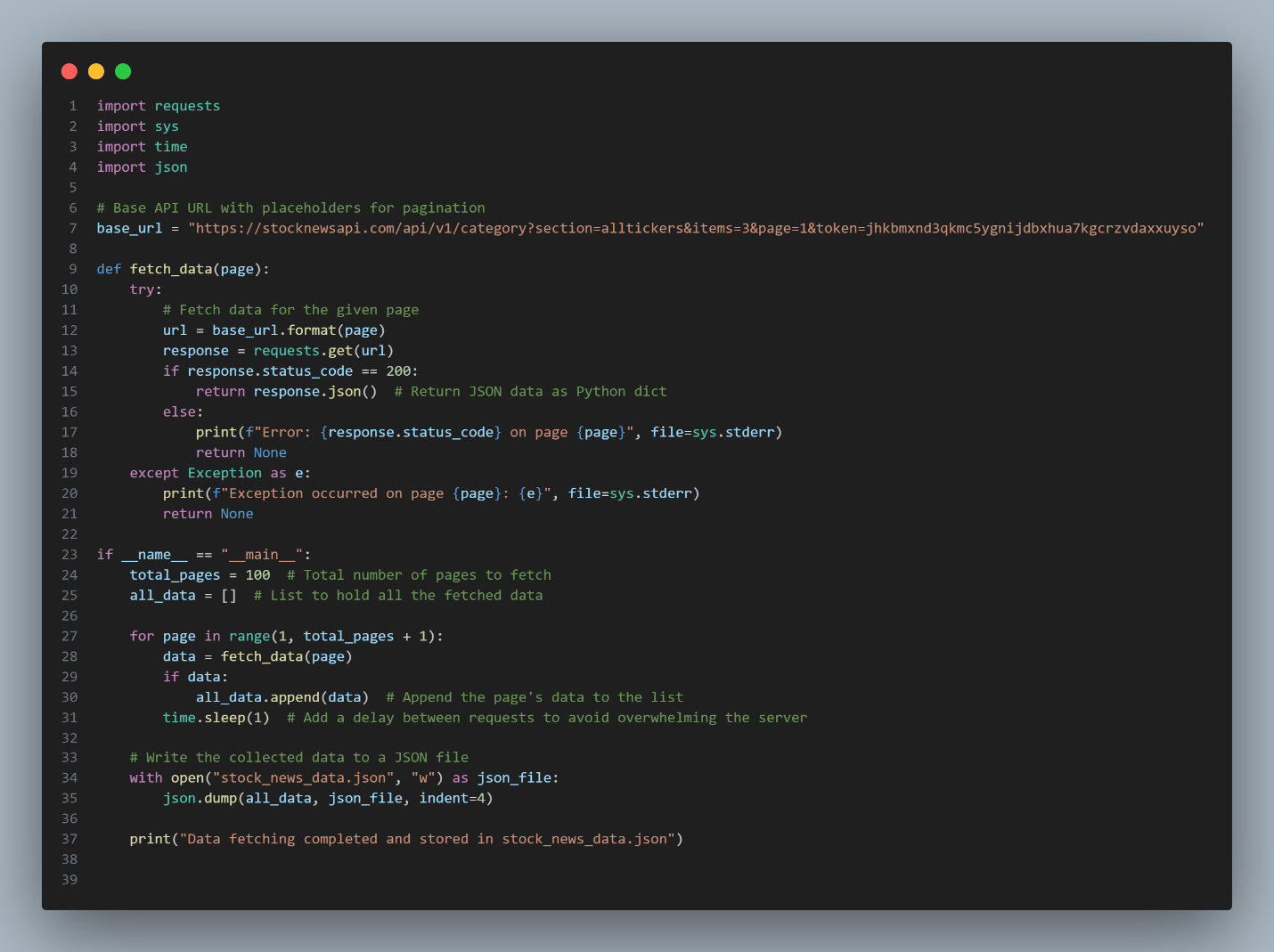
* **Import Libraries:**  
  Uses websocket, json, and Kafka.
* **Kafka Setup:**  
  Sets up Kafka server and topic.
* **API Setup:**  
  Configures Finnhub API token and URL.
* **Create Kafka Producer:**  
  Prepares to send messages to Kafka.
* **Send Data Function:**  
  Sends data to Kafka topic.
* **Message Handler:**  
  Receives and sends messages to Kafka.
* **WebSocket Connection:**  
  Connects to Finnhub and subscribes to stocks.
* **Run Connection:**  
  Keeps the connection open for data.

**News Data Streaming Configuration**



* Defines agent1 with sources, channels, and sinks.
* Reads .json files from /home/student/sicPro/grad-project/spooldir.
* Uses memory storage with a capacity of 1000 events.
* Sends data to Kafka on localhost:9092, topic news\_data.
* Connects source, channel, and sink together.

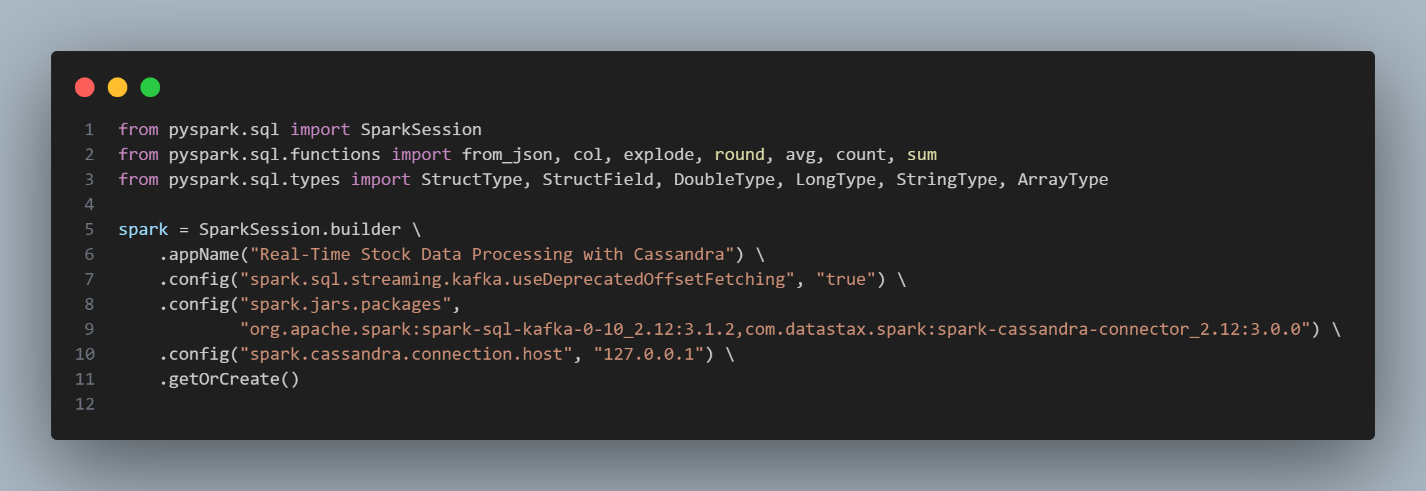
**Fetching Stock News Data from API and Storing in JSON Format**



This script fetches stock news data from an API and stores it in a JSON file. It iterates through multiple pages of API responses, appending the data to a list. The data is then saved as a JSON file for further use. A delay between requests helps prevent overwhelming the server.

Stock Market Trades Processing (Consumer)  
**Setup and Initialization**

* **Import necessary libraries.**
* **Initialize Spark session with Kafka and Cassandra support.**



**Data Reading and Processing**

* **Define Kafka parameters and schema for stock data.**
* **Read stock data from Kafka and parse it using the defined schema.**
* **Flatten the data structure and select relevant fields.**

****

**Price Alert Functionality**

* **Create a dictionary to track the last prices of stocks.**
* **Define a function to check for significant price changes and print alerts.**
* **Define a function to write the cleaned stock data to Cassandra.**

****

**Streaming and Aggregation**

* **Write the cleaned stock data to Cassandra using a streaming query.**
* **Group data by stock symbol and timestamp, performing aggregations.**
* **Write the aggregated data to Cassandra.**

****

**Stock Market News Processing (Consumer)**

**Setup and Data Ingestion**

* **Import necessary libraries.**
* **Initialize Spark session with Kafka support.**
* **Set up Kafka source and read data from the specified topic.**

**A screen shot of a computer

Description automatically generated**

**Sentiment Analysis and Data Writing**

* **Define a UDF for sentiment analysis using TextBlob.**
* **Apply the UDF to calculate sentiment from the summary.**
* **Select relevant columns and write the results to Cassandra.**

****

**Transfer Data from Cassandra to MongoDB**

This script connects to a Cassandra database, fetches cleaned stock data and stock analysis data, and then transfers that data to a MongoDB database.

****

* The script connects to the local Cassandra database and retrieves cleaned stock data and stock analysis data.
* It then establishes a connection to the MongoDB server, specifying the stock\_data\_mongodb database.
* Cleaned stock data is formatted into a dictionary and inserted into a collection in MongoDB.
* Stock analysis data is also formatted and inserted into its respective MongoDB collection.
* Once the data transfer is complete, the connections to both Cassandra and MongoDB are closed.
* This process enables efficient storage, analysis, and visualization of stock data in MongoDB.
* A list of stock symbols (AAPL, AMZN, IC MARKETS:1, BINANCE:BTCUSDT, MSFT) is defined.
* The code replaces the 'symbol' column in both DataFrames with random symbols from the predefined list.

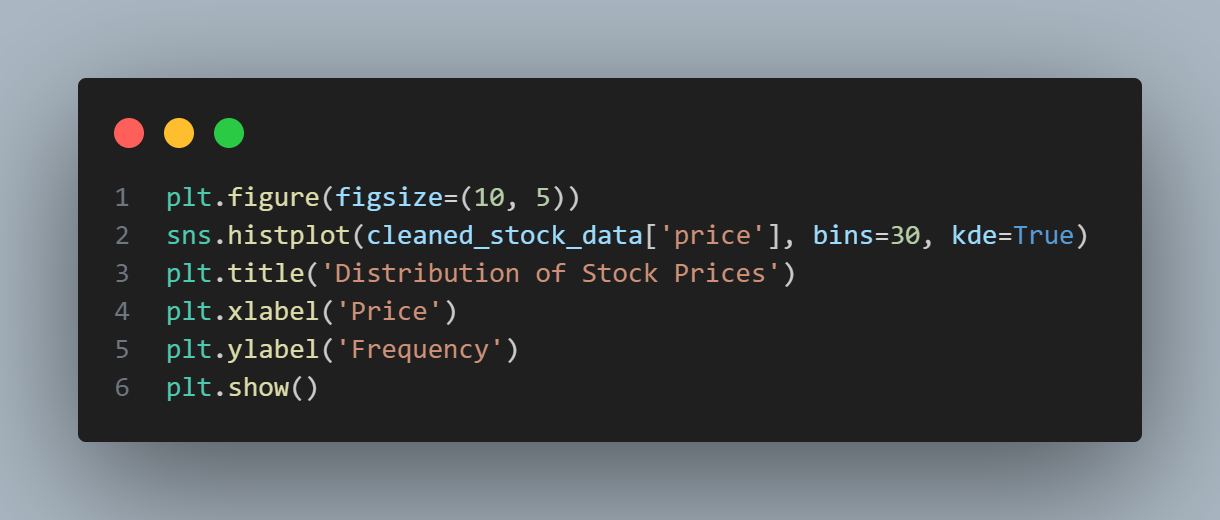


Data Visualization



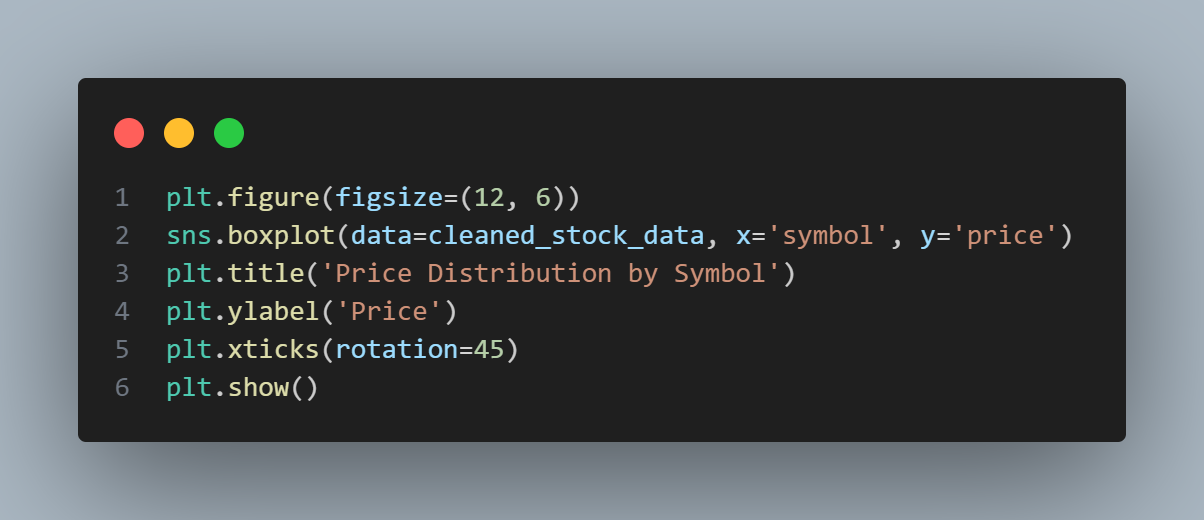
This code visualizes stock price trends over time for different symbols, helping to identify patterns and fluctuations in the market. By plotting the prices against time, it provides a clear and intuitive way to analyze how each stock has performed.

Distribution Analysis

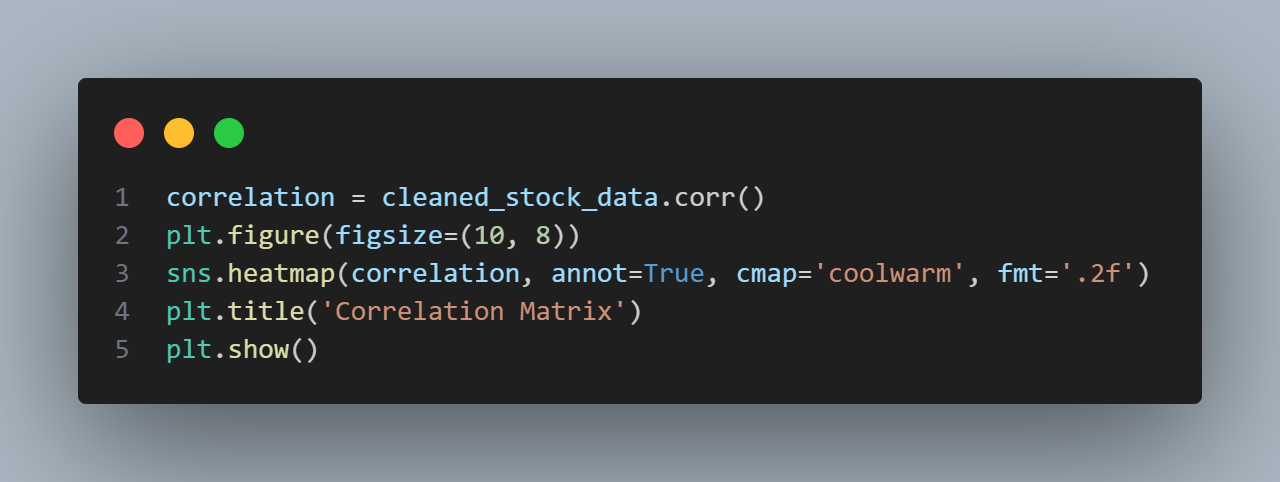


This code generates a histogram to visualize the distribution of stock prices. By plotting the frequency of prices across specified bins, it allows for an easy understanding of how stock prices are spread and whether they follow a particular pattern.

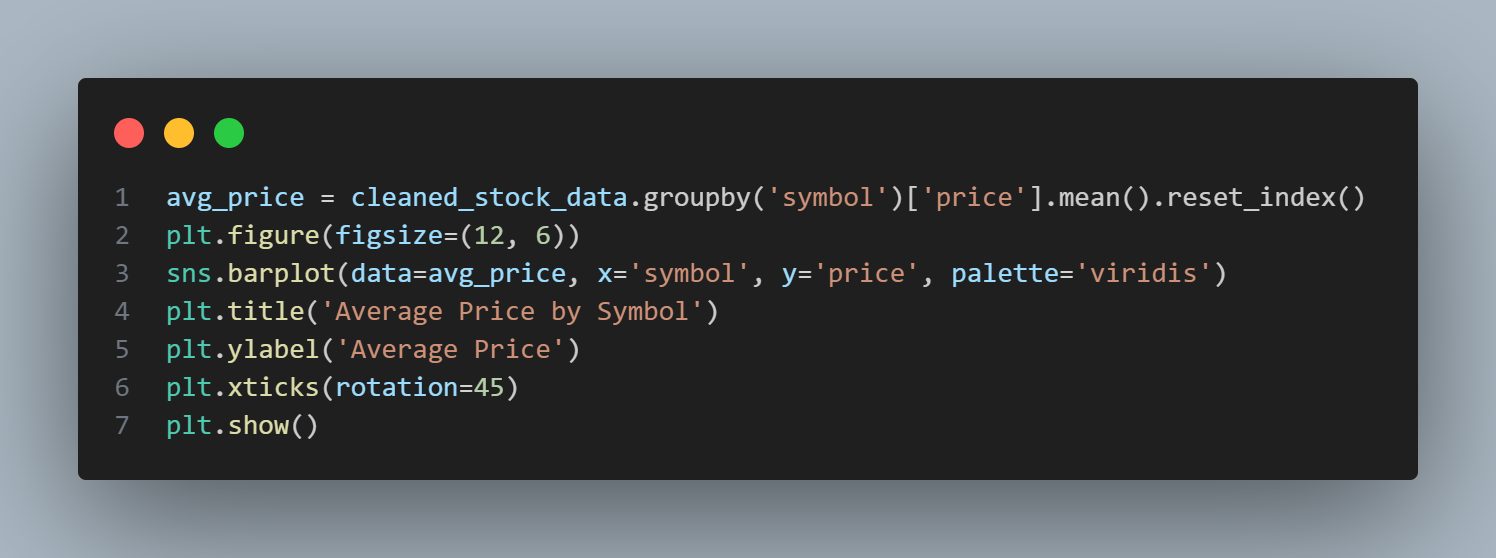
**Box Plot: Compare the price distributions across different symbols to identify outliers and variations.**



**Correlation Analysis**



**Bar Plot: Show average prices or total volume by symbol to compare performance**



**Scatter Plot: Plot price vs. volume to identify any relationship between them.**

