**Graph Data Extraction Process**

### **Overview of the Script**

The script automates the extraction of historical graph data from a financial website. It performs the following tasks:

1. **Load the Webpage**:  
   * The script uses Selenium to open the target webpage and waits for it to fully load.
2. **Select the "All" Tab**:  
   * It clicks the "All" tab to ensure the graph displays all available historical data.
3. **Extract the Graph Data**:  
   * The script executes JavaScript to retrieve the jsonfile variable, which holds the graph's data in JSON format.
4. **Process the Data**:  
   * The script parses the jsonfile data and converts it into a structured format using pandas.
5. **Save Data to a CSV File**:  
   * The extracted data is saved to a CSV file (graph\_data.csv) for further analysis.

### **Where the Data Is Stored**

1. **Original Data Source**:  
   * The data originates from the website’s backend, which could fetch it from:
     + A **database** storing historical fund values.
     + A **third-party API** that provides market or mutual fund data.
2. **Webpage Storage**:  
   * The backend sends the data to the webpage using one of two methods:
     + **Embedded in JavaScript**: The data is directly included in a JavaScript variable (jsonfile) when the page loads.
     + **Dynamic Fetching**: When the user interacts with the graph (e.g., clicks "All"), an **AJAX request** fetches new data and updates jsonfile.
3. **Graph Rendering**:  
   * The charting library (e.g., Chart.js) reads jsonfile to plot the graph:
     + s\_date (dates) for the x-axis.
     + s\_close (values) for the y-axis.
4. **Data Extraction to CSV**:  
   * The script retrieves jsonfile from the browser after the "All" tab is clicked.
   * The data key in jsonfile contains an array of objects (date-value pairs), which is processed and saved as a CSV file.

### **Flow of Graph Data**

#### **1. Original Source (Database or API)**

* The backend retrieves historical mutual fund data from a database or an API.

Example structure at this stage:  
 [

{"date": "2025-01-01", "value": 19.0},

{"date": "2025-01-02", "value": 18.9},

...

]

#### **2. Backend to Webpage**

* The data is either:

**Embedded in JavaScript** as part of the initial page load:  
 var jsonfile = {

"status": "success",

"data": [

{"s\_date": "Jan 01, 2025", "s\_close": "19.0"},

{"s\_date": "Jan 02, 2025", "s\_close": "18.9"},

...

]

};

**Dynamically fetched** via AJAX when a user clicks "All":  
 fetch("https://example.com/api/getData?range=all")

.then(response => response.json())

.then(data => {

jsonfile = data;

});

#### **3. Webpage to Graph**

The graphing library reads jsonfile and renders the graph dynamically:  
 var labels = jsonfile.data.map(item => item.s\_date); // x-axis

var values = jsonfile.data.map(item => parseFloat(item.s\_close)); // y-axis

#### **4. Webpage to CSV File**

* The script:
  + Extracts jsonfile using Selenium’s execute\_script method.
  + Processes the data key in jsonfile.
  + Saves the extracted data as a CSV file (graph\_data.csv).

### **Final Output**

* The graph data flows from its original backend source to the webpage and is visualized.
* The script extracts this same data and saves it to a CSV file for external use.