OPTION B - DATA PROCESSING 1

TRAN DUC ANH DANG

Student ID: 103995439

Requirements

Virtual Environment

- Google Colab
- Jupyter Notebook

Decencies

- numpy
- matplotlib
- mplfinance
- pandas
- scikit-learn
- pandas-datareader
- yfinance
- pandas_ta

Installation

*Note: Anaconda is required unless Google Collab is being used

Anaconda/Virtual Environment

1. Download Anaconda: Go to the Anaconda website (https://www.anaconda.com/products/distribution) and download the appropriate version for your operating system.
2. Install Anaconda: Follow the installation instructions for the OS from the Anaconda website.
3. Open Anaconda Navigator: Launch Anaconda Navigator from your installed applications.
4. Create a New Environment (Required): Create a new environment to isolate Jupyter installation on each project. Click on "Environments" in Navigator and then "Create" to make a new environment.
5. Install Jupyter Notebook: In the selected environment, click on the environment name and select "Open Terminal". In the terminal, type: conda install jupyter.

Dependencies

In Google Colab or Jupyter Notebook, it can directly install the required dependencies using the !pip command in code cells. Here's an example of how to install the dependencies:

!pip install <package> or !pip install -r <text file>

Data Processing 1

Effort to Explain Less Straightforward Lines of Code:

1. Import Statements:

- import os: This module provides a way of using operating system dependent functionality like reading or writing to the file system.
- import sys: This module provides access to some variables used or maintained by the interpreter and to functions that interact with the interpreter.

- from google.colab import drive: This statement allows the script to interact with Google Drive when running on Google Colab. This might be used for loading or saving data.
- import numpy as np: Numpy is a library for numerical computing in Python. It provides a high-performance multidimensional array object and tools for working with these arrays.
- import matplotlib.pyplot as plt: Matplotlib is a plotting library for Python. This line imports the pyplot interface, which provides a MATLAB-like interface for making plots and charts.

2. Complex Lines:

- Lines like DATA_DIR = os.path.join(SKELETON_DIR, "data") and PREPARED_DATA_DIR = os.path.join(SKELETON_DIR, "prepared-data") are constructing directory paths. Here, os.path.join is used to concatenate directory paths in a way that's compatible with the operating system. The variable SKELETON_DIR appears to be a base directory, and the script is defining subdirectories within it.
- Lines such as CSV_FILE = os.path.join(DATA_DIR, f"RawData-from-{start}to-{end}-{ticker}_stock_data.csv") are constructing file paths for CSV files. This line, in particular, seems to be defining a path to a stock data CSV file, and the filename is dynamically constructed using the start, end, and ticker variables. This allows for flexibility in naming files based on date ranges and stock tickers.

Challenges Faced:

- Dynamic File Paths: Understanding how file paths are dynamically constructed using variables can be challenging, especially when multiple variables and string formatting are involved.
- External Libraries: For someone unfamiliar with libraries like Numpy or Matplotlib, understanding functions and methods from these libraries might require external research. The same goes for Google Colab specific functions.

Research:

- Numpy and Matplotlib: For those unfamiliar with these libraries, the official documentation for Numpy and Matplotlib can be invaluable.

Refined Analysis

1. Loading and Processing the Dataset:

The data appears to be read using pd.read_csv() if a local CSV file exists. Otherwise, the script assumes that the data needs to be fetched and uses the yf.download function from Yahoo Finance to download financial data for a given stock ticker and time range.

Once data is loaded or downloaded, it's saved to a new CSV file for future access, ensuring efficiency in subsequent runs.

2. Handling of Start and End Dates:

Variables start and end are defined to set the desired time range for financial data. These are likely passed to the data fetching function to limit the date range of the fetched data.

3. Dealing with NaN Values:

NaN values are handled using df.dropna(inplace=True), which removes any rows containing NaN values from the dataset.

4. Different Methods for Splitting the Data:

The data can be split by date or randomly. If split_by_date is True, the function calculates an index based on a specified ratio and splits the data accordingly. Otherwise, a random split might be applied, although the exact mechanism isn't immediately visible from the highlighted lines.

5. Options for Storing and Loading Data Locally:

Data is saved to a CSV file after loading or downloading. This allows the function to load the processed data directly from the CSV in future runs, saving time and computational resources.

6. Scaling Feature Columns and Storing Scalers:

If the scale parameter is set to True, data scaling is performed using the MinMaxScaler from scikit-learn. The data is scaled to a range between 0 and 1.

Challenges:

- Understanding Data Fetching: Differentiating between when the script fetches data from Yahoo Finance and when it loads local data can be challenging without a deep dive into the logic.
- Handling NaN Values: The script uses a straightforward method to drop rows with NaN values. However, in real-world scenarios, different strategies might be required, such as filling NaN values based on certain criteria or using imputation techniques.
- Data Splitting Logic: Understanding the exact mechanism for splitting data (by date or randomly) requires a closer look at the function's logic and the parameters passed to it.
- Scaling Mechanisms: The script uses the MinMaxScaler for scaling, but understanding why this choice was made over others (like StandardScaler) might require external research or additional context from the script's authors.