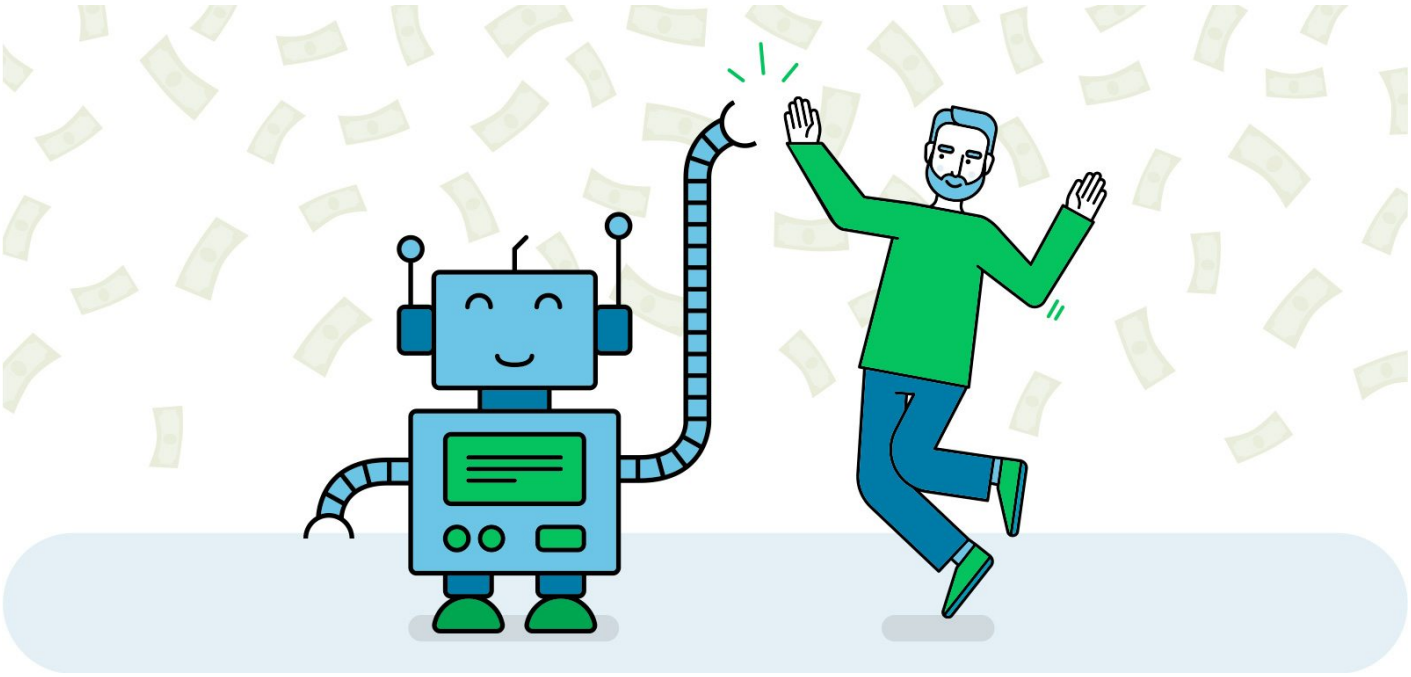
"Robo Advisor" Project

> Prerequisites:
>
> - Networks, and Processing Data from the Internet
> - APIs
> - Environment Variables
> - The "Omniparser" Exercise (specifically the stock data parsing challenge)
> - The "Web Requests" Exercise

Learning Objectives

1. Create a tool to automate manual efforts and streamline business processes.
2. Gain familiarity with APIs and web services, and practice issuing HTTP requests in Python (focusing on GET requests).
3. Increase exposure to JSON-formatted data, and practice processing it in Python.
4. Leverage built-in Python modules and third-party Python packages to speed development and enhance capabilities.

Business Prompt



Assume you own and operate a financial planning business which helps customers make investment decisions.

Your objective is to build yourself a tool to automate the process of providing your clients with stock trading recommendations.

Specifically, the system should accept one or more stock or cryptocurrency symbols as information inputs, then it should request real live historical trading data from the Internet, and finally it should provide a recommendation as to whether or not the client should purchase the given stocks or cryptocurrencies.

Setup

Repo Setup

Use the GitHub.com online interface to create a new remote project repository called something like "robo-advisor". When prompted by the GitHub.com online interface, let's get in the habit of adding a "README.md" file and a Python-flavored ".gitignore" file (and also optionally a "LICENSE") during the repo creation process. After this process is complete, you should be able to view the repo on GitHub.com at an address like `https://github.com/YOUR_USERNAME/robo-advisor`.

After creating the remote repo, use GitHub Desktop software or the command-line to download or "clone" it onto your computer. Choose a familiar download location like the Desktop.

After cloning the repo, navigate there from the command-line:

```
cd ~/Desktop/robo-advisor
```

Use your text editor or the command-line to create a new sub-directory called "app" with a file called "robo_advisor.py", and then place the following contents inside:

```
# app/robo_advisor.py

print("-------------------------")
print("SELECTED SYMBOL: XYZ")
print("-------------------------")
print("REQUESTING STOCK MARKET DATA...")
print("REQUEST AT: 2018-02-20 02:00pm")
print("-------------------------")
print("LATEST DAY: 2018-02-20")
print("LATEST CLOSE: $100,000.00")
print("RECENT HIGH: $101,000.00")
print("RECENT LOW: $99,000.00")
print("-------------------------")
print("RECOMMENDATION: BUY!")
print("RECOMMENDATION REASON: TODO")
print("-------------------------")
print("HAPPY INVESTING!")
print("-------------------------")
```

Make sure to save Python files like this whenever you're done editing them. After setting up a virtual environment, we will be ready to run this file.

Use your text editor or the command-line to create a new file called "requirements.txt", and then place the following contents inside:

```
requests
python-dotenv
```

After setting up a virtual environment, we will be ready to install these packages.

Environment Setup

Create and activate a new Anaconda virtual environment:

```
conda create -n stocks-env python=3.7 # (first time only)
conda activate stocks-env
```

From within the virtual environment, install the required packages specified in the "requirements.txt" file you created:

```
pip install -r requirements.txt
```

From within the virtual environment, demonstrate your ability to run the Python script from the command-line:

```
python app/robo_advisor.py
```

If you see the example output, you're ready to move on to project development. This would be a great time to make any desired modifications to your project's "README.md" file (like adding instructions for how to setup and run the app like you've just done), and then make your first commit, with a message like "Setup the repo".

Basic Requirements

Repository Requirements

Your project repository should contain an "app" directory with a "robo_advisor.py" file inside (i.e. "app/robo_advisor.py").

Your project repository should contain a "README.md" file. The README file should provide instructions to help someone else install, setup, and run your program. This includes instructions for installing package dependencies,

for example using Pip. It also includes instructions for setting an environment variable named `ALPHAVANTAGE_API_KEY` (see "Security Requirements" section below).

Your project repository should contain a file called ".gitignore" which prevents the ".env" file and its secret credentials from being tracked in version control. The ".gitignore" file generated during the GitHub repo creation process should already do this, otherwise you can create your own ".gitignore" file and place inside the following contents:

```
# .gitignore

# ignore secret environment variable values in the ".env" file:
.env
```

Finally, your project repository should contain a "data" directory with another ".gitignore" file inside, with the following contents in it to ignore CSV files which will be written inside the data directory:

```
# data/.gitignore

# h/t: https://stackoverflow.com/a/5581995/670433

# ignore all files in this directory:
*

# except this gitignore file:
!.gitignore
```

Security Requirements

Your program will need an API Key to issue requests to the AlphaVantage API. But the program's source code should absolutely not include the secret API Key value. Instead, you should set an environment variable called `ALPHAVANTAGE_API_KEY`, and your program should read the API Key from this environment variable at run-time.

You are encouraged to use a "dotenv" approach to setting project-specific environment variables by using a file called ".env" in conjunction with the `dotenv` package. Example ".env" contents:

```
ALPHAVANTAGE_API_KEY="abc123"
```

The ".env" file should absolutely not be tracked in version control or included in your GitHub repository. Use a local ".gitignore" file for this purpose (see "Repository Requirements" section above).

Functionality Requirements

Your project should satisfy the functionality requirements detailed in the sections below.

Information Input Requirements

The system should prompt the user to input one stock or cryptocurrency symbol (e.g. `"MSFT"`, `"AAPL"`, etc.). It may optionally allow the user to specify multiple symbols, either one-by-one or all at the same time (e.g. `"MSFT, AAPL, GOOG, AMZN"`). It may also optionally prompt the user to specify additional inputs such as risk tolerance and/or other trading preferences, as desired and applicable.

Validation Requirements

Before requesting data from the Internet, the system should first perform preliminary validations on user inputs. For example, it should ensure stock symbols are a reasonable amount of characters in length and not numeric in nature.

If preliminary validations are not satisfied, the system should display a friendly error message like "Oh, expecting a properly-formed stock symbol like 'MSFT'. Please try again." and stop execution.

Otherwise, if preliminary validations are satisfied, the system should proceed to issue a GET request to the AlphaVantage API to retrieve corresponding stock market data.

When the system makes an HTTP request for that stock symbol's trading data, if the stock symbol is not found or if there is an error message returned by the API server, the system should display a friendly error message like "Sorry, couldn't find any trading data for that stock symbol", and it should stop program execution, optionally prompting the user to try again.

Information Output Requirements

After receiving a successful API response, the system should write historical stock prices to one or more CSV files located in the repository's "data" directory. The CSV file contents should resemble the following example:

```
timestamp, open, high, low, close, volume
2018-06-04, 101.2600, 101.8600, 100.8510, 101.6700, 27172988
2018-06-01, 99.2798, 100.8600, 99.1700, 100.7900, 28655624
2018-05-31, 99.2900, 99.9900, 98.6100, 98.8400, 34140891
2018-05-30, 98.3100, 99.2500, 97.9100, 98.9500, 22158528
```

If the system processes only a single stock symbol at a time, the system may use a single CSV file named "data/prices.csv", or it may use multiple CSV files, each with a name corresponding to the given stock symbol (e.g. "data/prices_msft.csv, "prices_aapl.csv", etc.). If the system processes multiple stock symbols at a time, it should use multiple files, each with a name corresponding to the given stock symbol (e.g. "data/prices_msft.csv", "prices_aapl.csv", etc.).

After writing historical data to a CSV file, the system should perform calculations (see "Calculation Requirements" section below) to produce/print the following outputs:

- The **selected stock symbol(s)** (e.g. "Stock: MSFT")
- The **date and time when the program was executed**, formatted in a human-friendly way (e.g. "Run at: 11:52pm on June 5th, 2018")
- The **date when the data was last refreshed**, usually the same as the latest available day of daily trading data (e.g. "Latest Data from: June 4th, 2018")
- For each stock symbol: its **latest closing price**, its **recent high price**, and its **recent low price**, calculated according to the instructions below, and formatted as currency with a dollar sign and two decimal places with a thousands separator as applicable (e.g. "Recent High: $1,234.56", etc.)
- A **recommendation** as to whether or not the client should buy the stock (see guidance below), and optionally what quantity to purchase. The nature of the recommendation for each symbol can be binary (e.g. "Buy" or "No Buy"), qualitative (e.g. a "Low", "Medium", or "High" level of confidence), or quantitative (i.e. some numeric rating scale).
- A **recommendation explanation**, describing in a human-friendly way the reason why the program produced the recommendation it did (e.g. "because the stock's latest closing price exceeds threshold XYZ, etc...")

> NOTE: the CSV files are information outputs of this system, not information inputs. So it shouldn't be necessary for your program to read a CSV file to perform calculations. The JSON API responses should have all the information your program needs to perform calculations.

Calculation Requirements

The **latest closing price** should be equal to the stock's "close" price on the latest available day of trading data.

The **recent high price** should be equal to the maximum daily "high" price over approximately the past 100 available days of trading data.

The **recent low price** should be calculated in a similar manner as the **recent high price**, but it should instead be equal to the minimum of all daily "low" prices.

> NOTE: By default, the daily data returned by the AlphaVantage API uses an `outputsize` parameter value of `compact`. This "compact" response should provide daily data covering the previous 100 trading days, which is sufficient to use to calculate the **recent high** and **recent low** prices. It is acceptable and recommended to use these default, "compact" responses to calculate these recent prices.

You are free to develop your own custom **recommendation** algorithm. This is perhaps one of the most fun and creative parts of this project. 😀 One simple example algorithm would be (in pseudocode): If the stock's latest closing price is less than 20% above its recent low, "Buy", else "Don't Buy".

Guided Screencast

For a more in-depth guided exercise walkthrough, feel free but not obligated to follow the screencast, but keep in mind a few caveats:

1. Some of the links reference a previous course repository, but you should be able to find related documents in this course repository as well

2. If there are any discrepancies between requirements referenced in the video and requirements stated in this document, defer to the requirements stated in this document

Further Exploration Challenges

If you are able to implement the basic requirements with relative ease, consider addressing one or more of these "Further Exploration Challenges" to enrich and expand your learning experience.

Evaluation

Project submissions will be evaluated according to the requirements set forth above, as summarized by the rubric below:

Category	Requirement	Weight
Repository	Includes README.md file with detailed instructions	7.5%
Security	Excludes secret API Key values from the source code	12.5%
Validations (Prelim)	Prevents an HTTP request if stock symbol not likely to be valid (e.g. symbol of "8888")	5%
Validations	Fails gracefully if encountering a response error (e.g. symbol of "OOPS")	7.5%
Calculations	Displays accurate information	15%
Info Outputs	Displays final recommendation, including justification / context	17.5%
Info Outputs	Writes historical prices to CSV file	10%
Info Outputs	Formats all prices as USD (doesn't apply to CSV file values)	5%
Dev Process	Submitted via remote Git repository which reflects an incremental revision history	20%

This rubric is tentative, and may be subject to slight adjustments during the grading process.

If experiencing execution error(s) while evaluating the application's required functionality, evaluators are advised to reduce the project's grade by between 4% and 25%, depending on the circumstances and severity of the error(s).

In recognition of deliverables which exhibit functionality above and beyond the basic requirements, evaluators are encouraged to award between 4% and 15% "engagement points" to be applied as extra credit.