ShriTeq 2022

Bot Xchange Starter Pack

Thank you for participating in Bot Xchange in ShriTeq 2022. This document contains information and resources you require to compete in the event.

This document is expected to be read by all participating teams. You will not be able to fulfil the event requirements unless you follow the instructions in this document.

Table of Contents

Links

ShriTeq 2022 Discord server: **discord.gg/Gt3ZGvssgH.** Join the server, enter your school password to verify yourself as a participant, and choose the Bot Xchange role. The OC will be directly communicating with participants here and may occasionally provide help to participants.
OC members to contact on the Discord server are:

Advay (A G#4774)

Arjun (Arjun Sharma#5783).

Starter project: Find a starter project which is in the required format for submissions here. Find instructions on how to use the starter project here.

Training dataset: You can find a training dataset with historic prices for a set of stocks here. Use this data to build your price-predicting machine learning model. This is structured in the same way stock market data will be provided to your program during execution. Find more details about the format of the dataset here.

Starter machine learning notebook: You can find a starter notebook for creating the model here.
This contains code that helps you get started with processing the above dataset into something that can be used as training data for your model. It also contains code for building a rudimentary model, as well as code for how to use that model in your program.
Note: While the starter notebook is useful for helping you understand how to process the dataset, and can give you an idea of what features can be used, you should not blindly follow it. Some decisions have been intentionally taken in the starter notebook to avoid ideal performance.

Event Timeline

Tuesday, September 27 - The event opens and starting resources are provided to participants

Saturday, October 1 - Participants will be allowed to submit their code to the OC for a test run of their code. The OC will test whether the code is in the correct format and is functional, and will provide feedback to teams to fix bugs in their code, if any. The code or results from the test run will not be considered during the final evaluation of programs. Any problems with the code will not be considered in the final evaluation as this day should be used to iron out any issues.

Tuesday, October 4th - Submissions due at 11:59 PM.

Code Requirements

Folder Format

It is strongly recommended that you create your project in a local copy of the starter project. This already contains the correct directory structure with the correct files.

The folder should consist of the following files at minimum:

- A **main.py** file, containing a function called **`make_trades`**. Refer to this section for more details.
- A **requirements.txt** file, consisting of third-party dependencies that your code uses.

  Each line in the file should be formatted as `packagename==1.2.3`.
  - Find the package name on PyPi.
  - Write the correct version number of the package you require in place of `1.2.3`. If your code is version-agnostic, simply enter the package name without an == sign or a version number after it.

Try running `pip install -r requirements.txt` in the same directory as `requirements.txt` to ensure that the file is correctly formatted.

Note: if you are working in a virtual environment, enter `pip freeze > requirements.txt` in your command line in the same directory as `requirements.txt` to automatically generate the file in the correct format.

The folder may contain additional files, for example, helper code split across files and folders, or resources such as a model file.

Program flow

A team's program will be given $10,000,000 of virtual money to invest at the beginning. It will then be required to make trades in a virtual stock market every day for 50 days. The stocks that will be allowed in the market will all be from a similar industry.

A coderunner will automatically evaluate team programs automatically, keeping track of all necessary variables.

How make_trades will be called

On each day, the `make_trades` function from a team's `main.py` file will be called, with 4 arguments:

- `df`, a Pandas DataFrame in the below format:

ticker	day_0	day_1	day_2	...	day_{current}
0	10.10	10.20	10.00	...	12.32
1	50.50	50.00	51.00	...	42.42

. . .	. . .	. . .	. . .	. . .	. . .
25	23.00	23.01	23.02	...	25.00

Each row represents one stock. The first column is the ID of the stock, and each successive column is the historic opening price of the stock one day in the market. The last column represents the current day's opening price, at which stocks are currently being traded.

Note: The exact number of days of historic price data will not be disclosed to participants. It is guaranteed that there will be a minimum of 750 days in the dataframe which means you can use at most 750 days for calculations in your model.
Programs will be run multiple times, on a different set consisting of exactly 25 stocks each time. Refer to this section for more details.

Note: The training dataset is in the same format. Refer to it if you want to get a better idea of what this looks like as a DataFrame. The stocks in the training set are different from the stocks in the set the programs will be evaluated on.

- `holdings`, a dictionary where the key is the ID of the ticker and the value is the number of shares the program owns of that ticker. By default, this is 0 for all keys.
  For example, if the program has bought 50 shares of stock 1, `holdings` will be:
  `{0: 0, 1: 50, 2: 0, 3: 0 … }`

- `current_money`, a floating-point number representing the amount of money the program has. At the beginning of the first day, before any trades are made, this is equal to the initial amount, i.e., $10,000,000.

For example, if the only trade made by the program on day 1 was buying 50 shares of stock 2 at $50.0 per share, current_money on day 2 would be equal to (10,000,000.0 - 2500) = 9,997,500.0.

- `day`, an integer representing the number of days since trading started

Note: `day_0` in the historic price dataset is not the same as the first day the program is run - it is the first day for which there is historic data. The last column in the table represents the price of the program on the 'day' on which the function is being called.
For example, assume that `day_1000` is the last column in the table on the first virtual 'day' on which `make_trades` is executed. This means that we have data for the last 0-999 (1000 total) days, and the price of the stock 'today' is in `day_1000`. The price of the stock 'tomorrow' will be on `day_1001`.

Expected output of make_trades

`make_trades` is expected to return a **single dictionary**, where the keys represent the IDs of stocks and the values are integers representing the number of shares of corresponding stock.

If you wish to buy shares of a stock, the number of shares should be **positive**.
If you wish to sell shares of a stock, the number of shares should be **negative**.

You may buy as many shares of a stock as you wish, so long as the total value of those shares multiplied with the day's market price does not exceed the current money you have. You may only sell a number of shares of a stock less than or equal to the number of shares you already own of that stock.

Note: All the stocks to be bought will be bought first and only then will the stocks to be sold be processed irrespective of their order in the returned dictionary. Therefore you cannot use money earned by selling stocks on the same day.

For example, if you wish to buy 10 shares of stock 0, sell 5 shares of stock 1, and buy 20 shares of stock 2, the function should return `{0: 10, 1: -5, 2: 20}`. This assumes that the program currently has enough money to buy these many shares of stock 0 and 1, and owns 5 or more shares of stock 1.

The process by which you decide which stocks to buy or sell, and in what quantity, is **up to you**, and is the main challenge of the event. It is recommended that you use your ML model at this stage to forecast the future stock price(s), and then use a financial trading algorithm, to decide what trades to make.

How output is handled

Assuming there are no logical or runtime errors when `make_trades` is called, the coderunner will first conduct all stock purchases. This will:

1. Subtract the total value of stock purchases from the program's `current_money`
2. Increase the number of shares owned by the program for that stock

The coderunner will then conduct stock sales. This will:

1. Add the total value of stock sales to the program's `current_money`
2. Decrease the number of shares owned by the program for that stock

The coderunner will then call `make_trades` again, with a price dataframe as described here, with a new column at the end which represents the price on the next day. It will do this 49 times more in total, not counting the first day.
After the program has made its trades on the last day, all shares owned by the program will be sold.

Evaluation

As previously noted, programs will be evaluated over multiple trials on different sets of 25 stocks. Teams will be ranked based on the average of the final values of `current_money` across all of these trials.
Note: the sets will be the same for all teams to ensure a fair comparison

Error Handling

In case a logical or runtime error occurs during code execution, the coderunner will skip to the next 'day'. Therefore, any errors will result in the program making no trades on that day.

Logical errors may include, but are not limited to:

- Attempting to buy more shares of stocks than is possible with the value of `current_money`. For example, `current_money` is `200.5`, but your program attempts to buy 10 shares of stock 0 at $25 per share and 20 shares of stock 1 at $50 per share.
- Attempting to sell more shares of a stock than are currently owned. For example, 10 shares of stock 0 are owned by the program but it attempts to sell 20 shares.

Runtime errors include any exceptions thrown by your program during execution.

Make sure to utilise the window on Saturday, 1st October, in which you can submit your program for a test run to ensure it is functioning correctly.

Code execution

- Code will be run on a Linux machine with Python 3.9
- After packages are installed from `requirements.txt`, all code will be run **offline**. Your code should not require the internet in any way.
- Programs will only be allowed to read/write files from the folder submitted.
- Programs will be given 10 minutes to run after which they will be forcefully terminated.
- Any program which contains code which attempts to damage or disable the system in which it is being called will result in immediate disqualification of the school which submitted that program.

Submission Guidelines

Submissions should be emailed to **tsrs.shriteq@gmail.com** by 11:59 PM on Tuesday, 4th October. They should consist of:

- A link to the GitHub repository of your project, or a link to a .zip file of your project. This should include:
    - The main project as described here
    - (New as of 6:20 PM on 27th September) The code you used to train your model.
- The full name of your school, including its branch name
- The full names and grades of all the participants in your team

Submissions will be rejected if they do not contain all of these details.