Project 1: Combining Data from Multiple Sources

Introduction

Finance research often requires assembling a data set from different sources. In many cases, the source data is not formatted such that it can be imported easily into Python for analysis. This assessment helps you develop the foundational skills of data acquisition, cleaning, and merging. You will combine stock price information distributed across many files and produce the output as a JSON file. We ask you to write general code, such that it can adapt easily to different file formats.

Writing general code can be a daunting exercise at first. However, it is in your best interest to practice writing functions that can adapt to different configurations. Doing so makes your code more robust, easier to maintain, and upgradeable.

To help you, we have provided a Python *scaffold*. This file, called `zid_project1.py` contains all the functions needed for this project. Each function has a detailed *docstring* describing what the function does, the input parameters, and the object it returns.

The remainder of this document provide information on:

- The data files you will receive.
- Instructions to set up your development environment in PyCharm.
- Detailed step-by-step instructions required to complete the assessment. Please follow these instructions closely. Our ability to evaluate your work requires that you do so.

You should develop your code within PyCharm. Submission, however, will be through *Ed*. You will only need to copy your `zid_project1.py` file into *Ed*. Unlike the code challenges you have done so far, *Ed* will **not** provide you any feedback on your code. You can still submit multiple times before the deadline – Only your final submission on *Ed* will be marked.

The Source Files

All required files are included in a zip folder with the following structure:

```
project1/
|   project_desc.pdf
|   README.txt
|   TICEKRS.txt
|   zid_project1.py         <-- This is the only file you need to submit
|___data/
|   |   <tic>_prc.dat       <-- Several files of this type
```

- `zid_project1.py` contains the functions you will use in this assessment. Some of the functions are already written, while others you will need to write. Please see the instructions below for more information.
- `README.txt` contains information about how the stock data is stored in the `.dat` files. Please use the information contained the `README.txt` file you received (there are different versions for different students).
- `TICKERS.txt` contains a list of tickers and their corresponding exchanges, one per line. These tickers and exchanges may be in upper or lower case.

- `project_desc.pdf` is a PDF version of this document.
- `data` is a sub-folder containing all the data you will need to complete this assessment. Inside this folder you will find many files. Each `<tic>_prc.dat` contains stock price data for the ticker `<tic>`. Every ticker in `TICKERS.txt` will have a corresponding ".dat" file. However, you may have more ".dat" files than tickers in the `TICKERS.txt` file. In other words, you may have more ".dat" files than you will need (but not fewer).

Instructions

To set up your PyCharm for this assessment, please follow the following steps:

1. Unzip the contents of the zip file onto your computer.
2. Copy/move the entire `project1` folder into your PyCharm `toolkit` project folder. Afterwards, your toolkit folder will look like:

```
toolkit/                        <-- Project folder
|   toolkit_config.py           <-- Already created
|   ...
|___project1/                   <-- Contents of the zipped folder
|   |   project_desc.pdf
|   |   README.txt
|   |   TICEKRS.txt
|   |   zid_project1.py
|   |
|   |___data/
|   |   |   <tic>_prc.dat
```

3. Complete the user-written functions in `zid_project1.py`. See the step-by-step instructions in *Completing the Code Scaffold* below.
4. After you have completed the `zid_project1.py` module in PyCharm, copy and paste the entire contents of this module to *Ed*. This is the only file you will need to submit to complete this assessment.
5. Press "Submit" to submit your project. Your project will not be submitted until you do so.

Completing the Code Scaffold

After setting up your PyCharm development environment with the project files (see instructions above), modify the `zid_project1.py` module by taking the the steps described below, in sequence. The completed code will produce a JSON containing the combined contents of several files. You do **not** need to submit this JSON file.

Step 1: Set the location of files and folders (3 marks)

Open the `zid_project1.py` module in PyCharm.

Set the correct expressions for the constants `ROOTDIR`, `DATDIR`, and `TICPATH` as described below. Importantly, you should not include forward slashes or backslashes when defining these variables (so no "C:\User. . .", etc. . .). Instead, you should use the appropriate methods from the `os` module.

- The ROOTDIR variable combines the base location of the toolkit project folder (which is already specified in your toolkit_config.py module) and the project1 package.
- The DATDIR variable combines the location in ROOTDIR above and the data sub-folder. Note that this is a different variable than the DATADIR included in your toolkit_config.py module. The name of the DATDIR variable is a combination of "DAT" and "DIR" (not "DATA" + "DIR"), and it points to a different location in your computer.
- The TICPATH variable combines the location in ROOTDIR and the name of the file with the tickers (TICKERS.txt).

Again, all these paths should be created using the appropriate methods from the os module. If you include any forward or backward slashes in the definition of these variables your code will only run in your computer. A big part of this assessment is to make sure your code is portable.

The diagram below presents the relation between these variables and their locations. Again, do not use full paths (like "C:\Users. . .") to create these variables.

```
toolkit/
|   toolkit_config.py           <-- You already created this module
|   ...
|___project1/                   <-- `ROOTDIR` variable poitns to this folder
|   |   ...
|   |   TICEKRS.txt             <-- `TICPATH` variables |points to this file
|   |   ...
|   |   zid_project1.py
|   |___data/                   <-- `DATDIR` variable |points to this folder
|   |   |   ...
```

Note: All you have to do for this part is to replace the strings `"<COMPLETE THIS PART>"` with the appropriate `expressions`.

Step 2: Set the variables describing the format of the source data (2 marks)

This part of the project is very important! Make sure you follow these instructions closely.

Before we start, open one of the ".dat" files in PyCharm. To do that, just navigate to the **`toolkit/project1/data/`** folder (inside PyCharm) and double click on one of the files. You will notice the following:

1. There are no column headers. Every line in this file (including the first one) contains data.
2. There is no "separator" between columns (e.g., columns are not separated by comma, tabs, etc. . .). Instead, columns have a fixed width – For example, the first 10 characters belong to column 1, the next 8 to column 2, etc. . .

This means we have to create a function to split the lines into columns, so that each "value" is assigned to its correct "data field". The first step is to set the correct expressions for the variables `COLUMNS` and `COLWIDTHS`.

- The `COLUMNS` variable must be a list, where each element represents a source column name in the README.txt file. The order of the elements in this list must match the order of the columns in the `README.txt` file. For instance, suppose you have the following information in the `README.txt` file:

  ```
  # ----------------------------------------------------------------------------
  #   Column information
  # ----------------------------------------------------------------------------
  Close:
    column position: 1
    width: 14
  Date:
    column position: 2
    width: 11
  ```

  In this case, you must set `COLUMNS = ['Close', 'Date']`.

- The `COLWIDTHS` variable must be a dictionary. Each key is a column name in `COLUMNS`. Each value is the width of this column in the `README.txt` file. In the example above, you would set `COLSWIDTH = {'Close': 14, 'Date': 11}`.

Step 3: Complete the `get_tics` function (15 marks)

Complete the indicated part of the function `get_tics`. This function reads a file with tickers and returns a list with formatted tickers. Make sure the function works with the given `pth` variable and **not** the constant `TICPATH` (i.e., there should be no reference to the `TICPATH` constant inside this function). We will test your code using different files. Using `TICPATH` instead of `pth` inside `get_tics` means that your function always returns the same tickers instead of adapting to different possible ticker lists.

Your function must be consistent with the *docstring* provided. In particular, please make sure the body of the function is consistent with the "Parameters" and "Returns" sections of the docstring. The only exception is the optional suggestions provided in the "Hints" section, which you do not need to follow.

Your module also includes a test function called `_test_get_tics`. After you finish creating the get_tics function, it is a good idea to run the `_test_get_tics` function and look at the output. That should give you a good indication if your function is performing as expected. You can uncomment the relevant part of the `if __name__ ...` code block to run this test function. Like all other test functions provided, you can modify or delete these functions – they will not be marked.

Step 4: Complete the `read_dat` function (5 marks)

Complete the indicated part of the function `read_dat`. This function reads a stock price data file for a given ticker and returns its contents as a list of lines. Please make sure the body of the function is consistent with the "Parameters" and "Returns" sections of the docstring. You may choose to follow the proposed steps in the "Hints" section but that is optional.

Remember not to use literals with full paths like "C:\Users. . ." inside the body of the function (or anywhere in the module). You can use the constants you created in step 1 above (e.g., `DATDIR`) and methods from the `os` module to create paths. You can use the corresponding "test" function `_test_read_dat` to test this function once its completed.

Step 5: Complete the `line_to_dict` function (10 marks)

Complete the indicated part of the function `line_to_dict`. The same instructions provided for the `get_tics` and `read_dat` functions above apply to this function as well.

Step 6: Complete the `verify_tickers` function (10 marks)

Complete the indicated part of the function `verify_tickers`. This function takes in a list of tickers to be verified, and raises an `Exception` if any of the tickers provided is not a key of the dictionary returned by the `get_tics` function. Further details on when to raise an `Exception` are provided in the "Notes" section of the *docstring*.

An `Exception` is an action that disrupts the normal flow of a program. This action is often representative of an error being thrown. `Exceptions` are ways that we can elegantly recover from errors.

To learn more about raising an `Exception`, you may refer to the following resource:/ https://www.w3schools.com/python/gloss_python_raise.asp

Step 7: Complete the `verify_cols` function (10 marks)

Complete the indicated part of the function `verify_cols`. This function takes in a list of column names to be verified, and raises an `Exception` if any of the column names provided are not found in `COLUMNS`. Further details on when to raise an `Exception` are provided in the "Notes" section of the *docstring*.

Step 8: Complete the `create_data_dict` function (15 marks)

Complete the indicated part of the function `create_data_dict`. This function is used to transform the data found in the ".dat" files into a single dictionary.

This function takes in 3 arguments:

1. `tic_exchange_dic`
   - A dictionary returned by the `get_tics` function.
2. `tickers_lst`
   - A list containing tickers (as strings) whose data we want to save in the dictionary returned by `create_data_dict`.
3. `col_lst`
   - A list of the columns (as strings) that we want to save in the dictionary returned by `create_data_dict`.

An example of how the returned dictionary should look like when we call
`create_data_dict(tic_exchange_dic, ['aapl', 'baba'], ['Date', 'Close'])` is provided below:

```
{
    'aapl': {
        'exchange': 'nasdaq',
        'data': [
            {
                'Date': '2020-01-01',
                'Close': '8.0927',
            },
            {
                'Date': '2020-01-01',
                'Close': '8.2784',
            },
            ...
        ]
    },
    'baba': {
        'exchange': 'nyse',
        'data': [
            {
                'Date': '2017-05-13',
                'Close': '3.4939',
            },
            {
                'Date': '2017-05-14',
                'Close': '3.5689',
            },
            ...
        ]
    }
}
```

- Each dictionary found in a ticker's `data` list should only contain the columns specified by `col_list`.

- The `data` list for each ticker should contain a dictionary for each line of the ".dat" file for that ticker.

Note: The numbers used in the above example are entirely arbitrary for illustrative purposes only.

Step 9: Complete the `create_json` function (5 marks)

Complete the indicated part of the function `create_json`. This function saves a given dictionary into JSON file. To learn more about how to write data to a JSON file, take a look at the documentation for the following methods from the built-in Python package `json`:

- `json.dump`
- `json.dumps`

Step 10: Open-ended question (25 marks)

- Explain why to configure as in step 1 (5 marks).
- From step 9, we can obtain a JSON file that contains ticker names, listed exchanges, trading volume, and stock prices, and compute stock returns from stock prices. Suppose we have an extra variable to proxy the journalists' views about the firm performance, and we are interested in evaluating 2 alternative hypotheses:
- Whether the articles journalists wrote are based on:
    1. Investors' evaluations of those firms?
    2. Valuable information beyond firm fundamentals?
- Suppose we have observed the following:
    - When the articles contain relatively more negative words,
    - Stock returns decrease in the short run with no reversal in the long run.
- Explain which one of the two hypotheses is more likely to be true. Please evaluate both hypotheses. (15 marks)
- Given the hypothesis you have chosen, evaluate the short-run predictability for the trading volume with reason(s). (5 marks)

Submit your module

Copy and paste the entire content of your `zid_project1.py` to *Ed* and press "Submit". *Ed* will not mark your project automatically or give you any feedback. After you submit, we will be able to mark your project.

Administrative Guidelines, Additional Hints, Marking

Administrative Guidelines

We will enforce the following:

1. This assessment must be completed individually. Failure to complete the assignment on your own may result in a full loss of marks.
2. Late submissions are allowed, but will be penalised according to the rules described in the course outline.

Hints

Your code should be portable, working in a variety of settings. It should be sufficient to copy your code from PyCharm to *Ed* for submission. However, as part of this assessment, you need to make sure that your code works on our computer as well as on yours.

The following hints should help you correct any portability mistakes:

1. The contents of your `zid_project1.py` module **must not contain any direct reference to folders in your computer**. Variables that define paths should **not** contain any forward or backslashes. Of course, the variables you defined in your `toolkit_config.py` module (which you do not have to submit) do contain forward or backslashes (depending on your operating system). This is one of the reasons why we created this file to begin with.
2. Similarly, you should **not** include strings with specific tickers in `zid_project1.py` module (e.g., "TSLA", "AAPL"). For instance, you should not create a variable called tickers and then copy the specific tickers you received in your `TICKERS.txt` file. Instead, your code should read the `TICKERS.txt` file, produce a list of tickers, and store that in a variable.
3. When writing functions in the file `zid_project1.py`:

- Do not modify the function names or the parameters.
- Only modify the parts indicated by the "`<COMPLETE THIS PART>`" tag.
- You do not need to import any other module. Please do not modify the import statements.
- You should not create any additional constants. The constants that exist in the file (`ROOTDIR`, `DATDIR`, `TICPATH`, `COLUMNS`, and `COLWIDTHS`) should be edited as instructed.
- The "test" functions are included to help you test the code as you work through the project. These functions will not be marked, and you may change them as necessary to suit your needs. Test functions are clearly identified in the file with names starting with `_test`.

4. Use all the parameters in a function declaration. For example, the function `get_tics(pth)` has the single parameter `pth`. Make sure that your function uses this parameter and not a global variable.
5. **Only** submit the `zid_project1.py` module. Make sure your code works with this module only. No other modules can be submitted.

How we will mark your assessment

The following parts of this assessment will be marked:

1. Location of files and folders (3 marks)

2. Set the variables describing the format of the source data (2 marks)
3. Complete the `get_tics` function (15 marks)
4. Complete the `read_dat` function (5 marks)
5. Complete the `line_to_dict` function (10 marks)
6. Complete the `verify_tickers` function (10 marks)
7. Complete the `verify_cols` function (10 marks)
8. Complete the `create_data_dict` function (15 marks)
9. Complete the `create_json` function (5 marks)
10. Open-ended question (25 marks)

To receive full credit for parts 1 and 2, your variables must:

1. Have the correct type (e.g., `COLWIDTHS` must be a dictionary)
2. Include the correct values (e.g., the order of the columns in `COLUMNS` must match the one specified in your `README.TXT` file,
3. Follow all the instructions in this file (e.g., no forward or backslashes in `TICPATH`).

To receive full credit in parts 3 - 9, your functions must:

1. Return the correct object type (described in the docstring)
2. Return the correct information from the `TICKERS.txt`, `README.txt`, and “.dat” files you received.
3. Not violate any of the rules we specified in this document or in the docstring
4. If your function opens a file, **you must use a context manager**.