ECP 3004: Python for Business Analytics
Department of Economics
College of Business
University of Central Florida
Spring 2021

Assignment 4

Due Sunday, February 21, 2021 at 11:59 PM
in your GitHub repository

Instructions:

Complete this assignment within the space on your *private* GitHub repo (not a fork of the course repo ECP3004S21!) in a folder called `assignment_04`. In this folder, save your answers to Questions 1 and 2 in a file called `my_A4_functions.py`, following the sample script in the folder `assignment_04` in the course repository. When you are finished, submit it by uploading your files to your GitHub repo using any one of the approaches outlined in Question 3. You are free to discuss your approach to each question with your classmates but you must upload your own work.

Please note: In computer programming, many small details are very important. A file with the wrong name in the wrong folder will not run, even if the functions work perfectly.

Question 1:

Follow the function design recipe to define functions for all of the following Exercises. For each function, create three examples to test your functions. Record the definitions in the sample script `my_A4_functions.py`

Example 1 Write a function `matrix_multiply(mat_1, mat_2)` that replicates the `numpy` method `dot` using loops.

- Your function should take in two numpy arrays and output a numpy array with the same number of rows as `mat_1` and the same number of columns as `mat_2`.
- It should use three nested loops, one for the row of the output, one for the column of the output, and one for the sum taken on the multiplication along the elements the corresponding row of `mat_1` and column of `mat_2`.
- The element in row i and column j of your output array should equal $\sum_{k=1}^{n} mat\_1_{i,k} \times mat\_2_{k,j}$, where the number $A_{s,t}$ is the element in row s and column t of the array A.
- You can use the command `mat_1.dot(mat_2)` to produce the output for a test case.
- Your function should return `None` and print a warning message if the matrices are not conformable; that is, if the number of rows of `mat_2` is not equal to number of columns of `mat_1`.

Example 2 The linear regression model is estimated by minimizing the sum of squared residuals from fitting a linear model $\beta_0 + x_i\beta_1 + \varepsilon$ to the observed variable y_i over two lists of data x and y. Write a function `ssr_loops(y, x, beta_0, beta_1)` that calculates the value of the sum of squared residuals of this regression model.

$$SSR(y, x, \beta_0, \beta_1) = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2.$$

Example 3 Write another version of sum of squared residuals, except this version, called `ssr_vec(y, x, beta_0, beta_1)` should use vector operations with `numpy`. The parameters y and x should be `numpy` arrays and your function should not use loops. You can use the same test cases as you did for `ssr_loops(y, x, beta_0, beta_1)`. Note that some functions such as `sum()` and operations such as addition, subtraction and exponents operate element-by-element on `numpy` arrays. If necessary, experiment with some examples first.

Example 4 The likelihood function of the logistic regression model is used to estimate coefficients in logistic regression. Logistic regression is used to model binary events, i.e. whether or not an event occurred. For each observation i, the observation y_i equals 1 if the event occurred and 0 if it did not. Build on the function `logit_like()` from Assignment 3 and write a python function `logit_like_sum()` that calculates the sum of the log-likelihood across all observations (y_i, x_i), $i = 1, \ldots, n$. That is, it returns the sum of either the log of the function $\ell(x_i; \beta_0, \beta_1)$ if $y_i = 1$ or the log of the function $(1 - \ell(x_i; \beta_0, \beta_1))$ if $y_i = 0$, over all observations $i = 1, \ldots, n$. . For reference, the logit link function is defined as

$$\ell(x_i; \beta_0, \beta_1) = Prob(y = 1|x) = \frac{e^{x_i\prime\beta}}{1 + e^{x_i\prime\beta}} = \frac{e^{\beta_0 + x_i\beta_1}}{1 + e^{\beta_0 + x_i\beta_1}}.$$

This function `logit_like_sum()` will have four arguments, $(y, x; \beta_0, \beta_1)$, in that order, where the sample of observations y and x are both either lists or `numpy` arrays.

Question 2:

For all of the Exercises in Question 1, use your examples to test the functions you defined. Since the examples are all contained within the docstrings of your functions, you can use the `doctest.testmod()` function within the `doctest` module to test your functions automatically. Add some code to the sample program `my_A4_functions.py`, run the lines of code, and paste the output to a file called `my_A4_functions_doctest.txt`.

Don't worry about false alarms: if there are some "failures" that are only different in the smaller decimal places, then your function is good enough. It is much more important that your function runs without throwing an error.

Question 3:

Push your completed files to your GitHub repository following one of these three methods.

Method 1: In a Browser

Upload your code to your GitHub repo using the interface in a browser.

1. Browse to your `assignment_0X` folder in your repository (the "X" corresponds to Assignment X.).

2. Click on the "Add file" button and select "Upload files" from the drop-down menu.

3. Revise the generic message "Added files via upload" to leave a more specific message. You can also add a description of what you are uploading in the field marked "Add an optional extended description..."

4. Press the button "'Commit changes," leaving the buton set to "Commit directly to the `main` branch."

Method 2: With GitHub Desktop

Upload your code to your GitHub repo using the interface in GitHub Desktop.

1. Save your file within the folder in your repository within the folder referenced in GitHub Desktop.

2. When you see the changes in GitHub Desktop, add a description of the changes you are making in the bottom left panel.

3. Press the button "Commit to main" to commit those changes.

4. Press the button "Push origin" to push the changes to the online repository. After this step, the changes should be visible on a browser, after refreshing the page.

Method 3: At the Command Line

Push your code directly to the repository from the command line in a terminal window, such as GitBash on a Windows machine or Terminal on a Mac.

1. Open GitBash or Terminal and navigate to the folder inside your local copy of your git repo containing your assignments. Any easy way to do this is to right-click and open GitBash within the folder in Explorer. A better way is to navigate with UNIX commands, such as `cd`.

2. Enter `git add .` to stage all of your files to commit to your repo. You can enter `git add my_filename.ext` to add files one at a time, such as `my_functions.py` in this Assignment.

3. Enter `git commit -m "Describe your changes here"`, with an appropriate description, to commit the changes. This packages all the `add`ed changes into a single unit and stages them to `push` to your online repo.

4. Enter `git push origin main` to push the changes to the online repository. After this step, the changes should be visible on a browser, after refreshing the page.