QMB 3311: Python for Business Analytics
Department of Economics
College of Business
University of Central Florida
Spring 2022

Final Examination

Due Monday, May 2, 2022 at 6:59 PM
in your GitHub repository

Instructions:

Complete this examination within the space of your private GitHub repo in a folder called `final_exam`. In this folder, save your answers in Python scripts, as described in each question. When you are finished, submit it by uploading your files to your GitHub repo using any one of the approaches outlined in Question 7. Since this is an examination, you are NOT free to discuss your approach to each question with your classmates and you must upload your own work.

Question 1:

Module `my_CES_final.py`:
Collect the following functions from your previous assignments into a module called `my_CES_final.py`.

- `CES_utility()` from Assignment 2.
- `CES_utility_valid()` from Assignment 3.
- `CES_utility_in_budget()` from Assignment 3.
- `CESdemand_calc()` from Assignment 5.
- `max_CES_xy()` from Assignment 5.

Module `my_logit_final.py`:
Collect the following functions from your previous assignments into a module called `my_logit_final.py`.

- `logit()` from Assignment 3.
- `logit_like()` from Assignment 3.
- `logit_like_sum()` from Assignment 4.
- `logit_di()` from Assignment 5.
- `logit_dLi_dbk()` from Assignment 5.
- `logit_like_grad()` from Assignment 7.

Question 2:

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 Python scripts as described in Question 1.

Under time constraints, the emphasis is on creating function definitions and testing with examples. You are not responsible for additional error handling and can assume that the user will follow the preconditions and type contract.

Most importantly, take any shortcut you can, using the code from the course repository and any solutions from your previous assignments. The first step is often to locate a similar function and copy it to your module. Then you can begin to either re-purpose that code or call the function in a new function, and save yourself some time.

Exercise 1 Write a function definition
`CES_multi(x, r)` that returns the value of utility from the CES utility function:

$$u(x, r) = \left(\sum_{i=1}^{n} x_i^r \right)^{\frac{1}{r}},$$

where x_i is the amount of each good i purchased by the consumer. Consequently, the arguments `x` is a vector or list of length n. Add this function to your module `my_CES_final.py`.

Exercise 2 Now write a function that calculates the optimal bundle of goods for the CES utility function, maximizing

$$u(x, r) = \left(\sum_{i=1}^{n} x_i^r \right)^{\frac{1}{r}}, \qquad \text{subject to} \qquad \sum_{i=1}^{n} p_i \times x_i \leq w.$$

where p_i is the price of each good i and x_i is the amount of each good i purchased by the consumer with given wealth w. Using calculus, and assuming $0 < r < 1$, when the goods are complements, you can show that the following choices maximize the consumer's utility:

$$x_i^* = \frac{w \cdot p_i^{\frac{-1}{1-r}}}{\sum_{j=1}^{n} p_j^{\frac{-r}{1-r}}}.$$

It is possible, however, that the goods are substitutes, in the case that $r >= 1$, in which case, it would be better to spend all the money on a good with the lowest price (you may assume that no prices are equal, or pick the first, since the optimal utility will be the same). In this case, the solution is

$$x_i^{max} = \begin{cases} \frac{w}{p_i}, & \text{if } p_i = \min\{p_1, \ldots, x_n\}, \\ 0, & \text{if otherwise.} \end{cases}$$

Write a function that returns a list of floats x^{max} called `max_CES_multi_calc(r, p, w)`, where the arguments `(r, p, w)` represent the parameters in the utility function and the consumer's budget constraint. Add this function to your module `my_CES_final.py`.

Exercise 3 Create a penalized utility function that subtracts from the value of the utility if the solution goes out of bounds. That is, it applies a penalty if any of the x_i are negative or if the spending goes out of budget.

$$\tilde{u}(x, r, p, w, \lambda) = \left(\sum_{i=1}^{n} |x_i|^r \right)^{\frac{1}{r}} - \lambda \left[\sum_{i=1}^{n} |x_i| \times 1_{\{x_i < 0\}} + \left(\sum_{i=1}^{n} p_i x_i - w \right) \times 1_{\{\sum_{i=1}^{n} p_i x_i > w\}} \right],$$

where $|x_i|$ is the absolute value of x_i and the indicator function $1_{\{A\}}$ is a boolean indicator that the condition A is true. In words, the penalty is λ times the absolute value of any negative x_i and λ times the amount that the consumer goes over budget but zero otherwise. Write this in a function `CES_multi_penalty(x, r, p, w, penalty)` and add this function to your module `my_CES_final.py`.

Exercise 4 Now verify the derivation of the solution in Exercise 2. Use any method available to find a numerical solution to optimize `CES_multi_penalty(x, r, p, w, penalty)`. Write this in a function called `max_CES_multi_penalty(r, p, w, penalty)` that returns a list of floats x^{max} the same length as p and add this function to your module `my_CES_final.py`. A solution that accommodates two goods is acceptable, however, a solution that accommodates three or more is better.

Question 3:

For all of the Exercises in Question 2, and the functions listed above, use examples to test the functions you defined. Since the examples will all be contained within the docstrings of your functions, you can use the `doctest.testmod()` function within the `doctest` module to test your functions automatically at the bottom of the modules. Use an `if` statement to determine whether the tests will be run using `testmod`: if the script is executed as a script, i.e. as the `__main__` script, then run the tests; if it is imported, then skip the tests.

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 4:

The sample script `logit_calculation_final.py` uses the `statsmodels` module to estimate a model for the probability that borrowers will default on their loans. You will calculate the parameter estimates by applying root-solving methods to estimate these parameters. The dataset `credit_data.csv` in `demo_19_Classification` includes the following variables.

default:	1 if borrower defaulted on a loan
bmaxrate:	the maximum rate of interest on any part of the loan
amount:	the amount funded on the loan
close:	1 if borrower takes the option of closing the listing until it is fully funded
AA:	1 if borrowers FICO score greater than 760
A:	1 if borrowers FICO score between 720 and 759
B:	1 if borrowers FICO score between 680 and 719
C:	1 if borrowers FICO score between 640 and 679
D:	1 if borrowers FICO score between 600 and 639

In the script `logit_calculation_final.py`, these data are loaded in and used to estimate a model using `statsmodels`, with only the variable `bmaxrate`.

a) Run the script `logit_calculation_final.py` up to line 150 to obtain the results for the estimation with `statsmodels`. The goal is to achieve the maximum value of the log-likelihood function shown in the output from `logit_model_fit_sm.summary()` and match the parameter estimates in `logit_model_fit_sm.params`.

b) For Assignment 7, you wrote a function `logit_like_grad(beta, y, x)` to calculate the gradient vector of the likelihood function `logit_like_sum_opt()`. If the value of the parameter `beta` is an optimum, the value of the gradient should be zero, since the gradient is the slope of the objective function and the slope of a function is zero at an optimum. Evaluate the gradient at the estimates from the `statsmodels` module to verify that the gradient is (near) zero in both elements.

c) Calculate the parameter estimates by solving for a root of `logit_like_grad(beta, y, X)`. Use any method you like, including built-in functions from modules such as `scipy`.

Question 5:

The folder `final_exam` contains three `.csv` files: `applications.csv`, `credit_bureau.csv`, and `demographic.csv`. The first dataset `applications.csv` contains the following variables.

`app_id`	=	a unique key for each customer who applied for credit
`ssn`	=	the social security number
`zip_code`	=	the the zip code in which the applicant resides
`income`	=	the applicant's reported income
`homeownership`	=	a categorical variable that indicates whether an applicant owns or rents a home
`purchases`	=	the monthly value of purchases on the account
`credit_limit`	=	the maximum amount that an applicant is approved to spend

The dataset `credit_bureau.csv` contains the following variables.

`ssn`	=	the consumers unique social security number
`zip_code`	=	the zip code in which the consumer resides
`fico`	=	the consumer's credit score
`num_late`	=	the number of number of times a consumer has made a payment after the due date
`past_def`	=	the number of number of times a consumer has defaulted on a line of credit
`num_bankruptcy`	=	the number of number of times a consumer has filed for bankruptcy

The dataset `demographic.csv` contains the following variables.

`zip_code`	=	the zip code to indicate each geographic region
`avg_income`	=	the average income in each zip code
`density`	=	the population density in each zip code

a) Create a new database called `credit_final.db`.

b) `CREATE` the following three `TABLE`s with schema that are appropriate for the variables.

   i) A table called `Applications` from the dataset `applications.csv`.

   ii) A table called `CreditBureau` from the dataset `credit_bureau.csv`.

   iii) A table called `Demographic` from the dataset `demographic.csv`.

c) Use an SQL query to calculate the minimum, average and maximum `purchases` for each of the following groups.

   i) Customers who own a home.

   ii) Customers who own a home and have an income above $100,000.

   iii) Customers who do not own a home but have a `fico` score above 700.

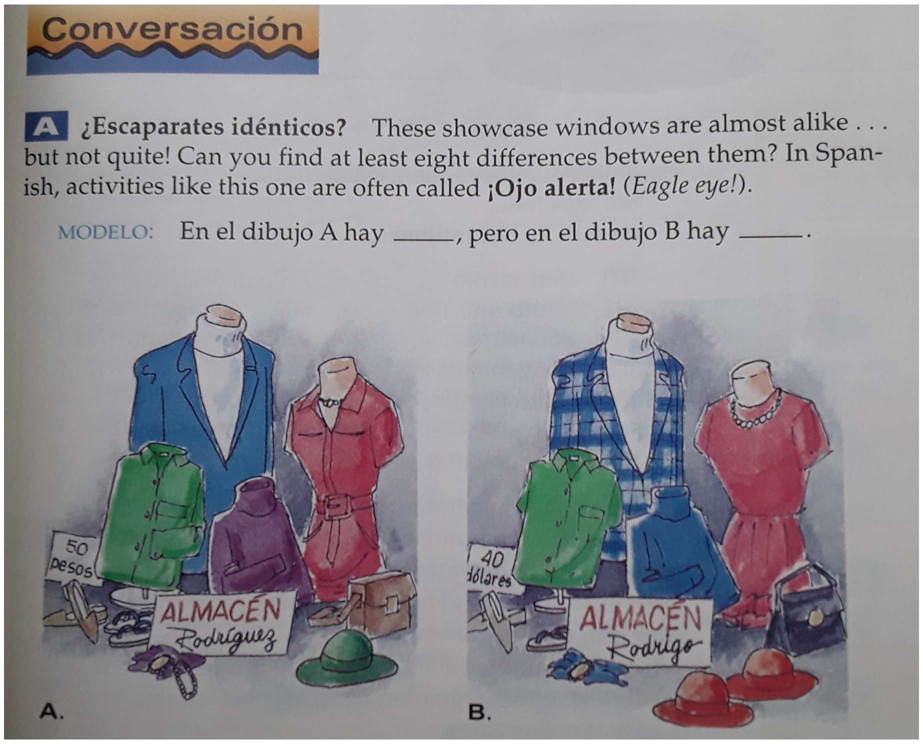
   iv) Customers who own a home and and have an income above the average in their zip code.

Question 6:

Bonus Question

In Figure 1, the images in panel A and B have several differences between them. List as many differences as you can.

Seriously! Save the list in a script where you could use the extra points.



Conversación

A **¿Escaparates idénticos?** These showcase windows are almost alike . . . but not quite! Can you find at least eight differences between them? In Spanish, activities like this one are often called **¡Ojo alerta!** (*Eagle eye!*).

MODELO: En el dibujo A hay ____, pero en el dibujo B hay ____.

Figure 1: Find the Differences

Question 7:

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 ("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 "'Commit changes," leaving the button 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 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.