Python Exercises

2022 European Summer School on Computational and Mathematical Modeling of Cognition

Contents

Assignment description

Work through the following Python problems. You are required to work on some individually, but on other questions you can work in pairs or groups of three. Feel free to ask questions in class and on Slack!

You must save your solutions to the problems as 4 Python scripts (one .py file per **Session**). All answers must have comments and be labeled with the problem number. Some answers will only be comments without associated Python code. IPython terminal-specific commands should be reported as a comment. For questions about output, report your output in a comment under the relevant line of code. Note that all comments have # in front of them so that your Python interpreter does not run this line of code.

Note that the final project (last Section) is due at the end of Day 2. So if you are still working through some of the other problems by mid-day on Day 2, skip to the final project.

```
# Q1.0
# This is the answer the the example problem Q1.0 in a comment
this_variable = 1
this_variable
# Out[3]: 1
```

Session 1: The basics

Q1.1

```
my_numpy_array = np.array([5, 6, 1])
```

This python code produces an error that the name 'np' is not defined. What line of code did I need to import numpy before this line or at the beginning of my Python script?

Q1.2

Assume Numpy is loaded before all following questions.

```
another_array = np.zeros((2, 4, 6))
```

Return the first element of the first dimension, the 4th element of the second dimension, and the last element of the 3rd dimension using the Python shortcut for a last element.

Q1.3

```
another_array = np.zeros((2, 4, 6))
new_array = another_array
new_array[1, 2, 2] = 1
print(another_array[1, 2, 2])
```

What went wrong in this code? How is this different from R? How can we make a true new copy of the numpy array?

Q1.4

Note that the iPython terminal is a specific **interactive** terminal program that will save Python variables in a workspace like R. Other Windows, Mac, and Linux terminals can run Python scripts but are not as interactive.

Try copying some code and then pasting it into a iPython terminal using the magic function `%paste`. Does the `%paste` function work in a Python script? Why or why not?

Q1.5

What are the magic functions in the iPython terminal to change the working directory, print the current working directory, list the contents of the working directory, and to list current workspace variables?

Q1.6

What is the command that can be run in a terminal or iPython terminal to install the package `pip-install-test`?

Q1.7

```
sample_scores = np.array([1, 6, 7, 8, 9, np.nan])
print(np.mean(sample_scores))
```

This returns `nan`, why? Can you compute the mean with the missing value removed? Check `?np.mean`

Q1.8

Make a four dimensional numpy array of 2 by 2 by 2 by 2 with as elements the squares of 1 to 16 (e.g., 1^2, 2^2, 3^2, 4^2 etc).

Q1.9

Import the InsectSprays.csv (to be provided) into Python as a Python `dictionary`. Note that one method is to use `pandas.read_csv` with one additional step. How can you return the variables or `keys` of this dictionary in Python?

Q1.10

We used this data to simulate the speed of R and Python on various operations.

```
np.random.seed(1234)  # Set the random seed
speed_sec = np.zeros(10)
sim_speed = np.random.uniform(size=5)  # Speed simulation in seconds
speed_sec[::2] = sim_speed * 10
speed_sec[1::2] = sim_speed
```

Make the following `pandas.DataFrame` without typing out the first two variables. You can use the variable `speed_sec` for the last variable. Note that you can first make a Python `dictionary` before converting to a `pandas.DataFrame`.

```
##   language code_type  speed_sec
## 0        R  forloop1   1.915195
## 1   Python  forloop1   0.191519
## 2        R  forloop2   6.221088
## 3   Python  forloop2   0.622109
## 4        R  forloop3   4.377277
## 5   Python  forloop3   0.437728
## 6        R  forloop4   7.853586
## 7   Python  forloop4   0.785359
## 8        R  forloop5   7.799758
## 9   Python  forloop5   0.779976
```

Q1.11

Using **Python**, make the following `pandas.DataFrame` without typing out the full vector of any of the columns.

```
##    year people government
## 0   -60  Roman   republic
## 1   -40  Roman   republic
## 2   -20  Roman     empire
## 3     0  Roman     empire
```

Q1.12

Write **Python** code that chooses a random positive integer between 50 and 1000 and assigns that value to `sample_size`. The code should then simulate standard normal data with a sample size equal to `sample_size`, and then plot a histogram of the data.

Q1.13

```
grass = np.array(["green", "fake", "yellow"])
leaves = grass
leaves[0] = "brown"
```

What could go wrong when writing more code that uses variables `grass` and `leaves`? How would you fix it?

Session 2: Flow control

Note that in the following problems we assume that the following imports are given at the beginning of the .py file.

```
import numpy as np
from time import time
```

Q2.1

```
two_dice = np.random.choice(np.arange(1,7), 2, replace = True)
```

Consider a game of dice in which two dice are rolled and a score is determined from the roll.

The score is determined as follows: If the sum of the two dice is even, the score equals the sum of the dots (pips) multiplied by the smallest number of the two dice. For example, if you roll a 3 and a 5 then sum, 8, is even. The score is 3 * (3+5)=24.

If the sum of the two dice is odd, your score is the number of dots (pips) multiplied by -3. For example, if you roll a 2 and a 5 then sum, 7, is odd. The score is -3 * (2+5)=-21.

Write an **if-else statement** that calculates your score based on the values of the dice.

Q2.2

Create the numpy array `np.array([3, 5, 7, 9, 11, 13, 15, 17])` using a **for loop**. Start with `double_plus = np.array([])` and fill the vector using the for loop. Hint: See `?np.append`

Q2.3

```
grass = "green"


def color_it(color_me, grass_me):
    grass_me = grass
    color_me = "blue"
    grass = "blue"
    colorful_items = np.array([(color_me, grass_me)])
    return colorful_items


sky = "grey"
ground = "brown"
these_items = color_it(sky, ground)
print(these_items)
```

Remember the discussion about **global** and **local** variables within and outside **functions**. Does this code run in Python? Why or why not?

How is this different to R? Fix the code to make it run.

Q2.4

Run the following code.

```
class MyClass:
    """A simple example class"""
    classnum = 12345

    def famous(self):
        return 'hello world'


new_stuff = MyClass()
new_stuff.classnum
new_stuff.famous()
```

What is a Python **class**? Briefly describe a Python **class** in a couple of sentences.

Read briefly Python **classes** here: https://docs.python.org/3/tutorial/classes.html

Now add a **definition** to the following Python **class** `ComplexNum` to return the phase angle in radians of the complex vector with the command `my_num.phase_angle()`. Try creating a few different complex numbers with this class.

```
# Complex number class
class ComplexNum:
    """Creates a complex number"""
    numtype = 'complex'

    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart

    def vec_length(self):
        return np.sqrt(self.r**2 + self.i**2)
```

```
my_num = ComplexNum(3.0, 4.0)
print(my_num)
print((my_num.r, my_num.i))
print(my_num.numtype)
print(my_num.vec_length())
```

Q2.5

Write a definition that gives the first `n` Fibonacci numbers as a **numpy array**. See https://en.wikipedia.org/wiki/Fibonacci_number

Use the following structure.

```
def fibonacci(n):
    """
    Returns something...
    """
    # Your code here
    return result
```

Q2.6

Change the Docstring `"Returns something..."` in your `fibonacci()` definition into an informative message and usage about your definition.

Now place your `fibonacci` definition into its own file called sequences.py. This is a called a Python module. Now import sequences.py into your IPython console with this line:

```
from sequences import fibonacci
```

Now type `?fibonacci`. Do you see your Docstring? Why might you want to keep a few definitions in their own .py file? Place your answer as a comment in sequences.py outside the definition.

More help: https://docs.python.org/3/tutorial/modules.html

Q2.7

We might want to use the command `assert` to run tests for our program to make sure that it always returns the correct output, even when we change things in our program. `assert` returns an error if the logical is `False`. Running tests for large programs is very useful.

Add some assert programs at the end of your sequences.py file in the following format:

```
if __name__ == '__main__':
    assert fibonacci(1) == 0
    assert fibonacci(10)[-1] == 34
    print("Tests passed")
```

Write additional assert lines for more tests of your module.

Q2.8

Write code that reads the csv file at this location using `pandas` in **Python**:

https://tinyurl.com/pipsTitanic

Write a for loop over the rows (passengers) of the data file's original order. Whenever the current passenger is female and survived, **and also** the previous passenger was male, print: "`I'll never let go, Jack. I'll never let go. I promise.`"

Q2.9

Write a function in **Python** with one argument `start_num`. This function counts up by 0.1 when given a number `start_num` greater than 0, and counts down by one whole number when given a number `start_num` less than 0. The function should stop counting when it reaches 100 or -100. The function should also print "`I'm not good at counting backwards`" before it counts backwards, but not when it counts forwards.

Q2.10 (Somewhat challenging)

Write a definition `nthroot()` that calculates, by iteration, the nth root of a number using the Newton's method (https://en.wikipedia.org/wiki/Newton%27s_method). The definition should take as arguments, `number`, `n`, and a starting value `start`. The default starting value should be 1.

Hint: https://en.wikipedia.org/wiki/Newton%27s_method#Square_root

Q2.11

Use `time()` from the module `time` to test how long `nthroot(4214,5)` takes compared to base Python code: `4214**(1/5)`. Which is faster?

Q2.12 (Challenging)

Quicksort is a famous algorithm. See https://en.wikipedia.org/wiki/Quicksort

Write the Quicksort algorithm in Python. This will likely require at least two **definitions**. See this pseudocode: https://textbooks.cs.ksu.edu/cc310/7-searching-and-sorting/19-quicksort-pseudocode/

Q2.13 (Bonus)

Create a program that allows you to play Hangman in a terminal (e.g. the IPython console). Read more about the Hangman game if you are not already familiar with it here: https://en.wikipedia.org/wiki/Hangman_%28game%29.

Your Python program should pick a random word from a long list of words. The player in the Python console will guess letters in the words. The Hangman word list can be in any human language you choose (e.g. Nederlands, English, Deutsch, etc.)

We are not awarding points based on Hangman graphics. But graphics can be included in your program if you wish, either as text-based graphics in the terminal or in other graphics.

Session 3: Plotting and style in Python

Q3.1

Define a **Python function** that takes two vectors as inputs. The **function** then plots a scatter plot of the first input on the x-axis and the second input on the y-axis. Use the `matplotlib` library to make your plots.

Q3.2

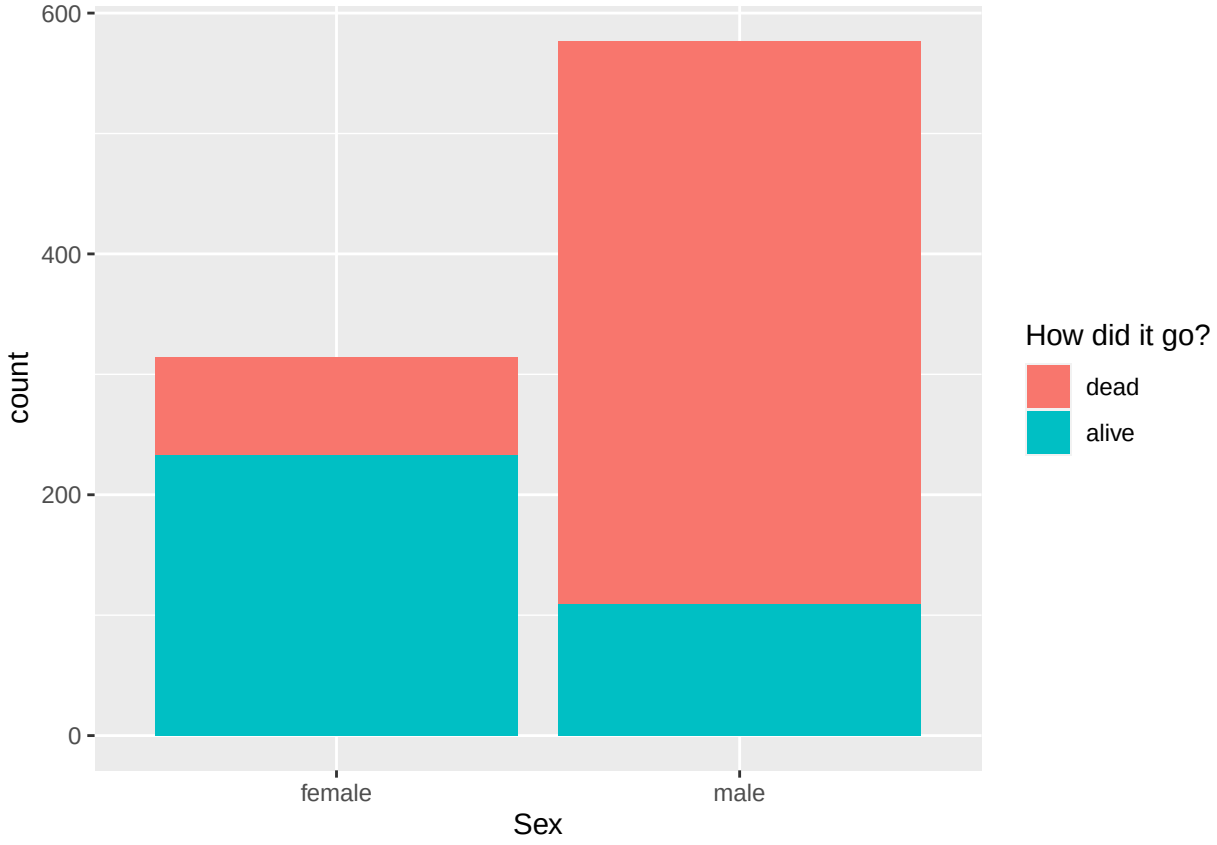
Simulate some data and show them in a `plt.boxplot()` using matplotlib. Now create a `sns.violinplot()` with "jittered" data points using `sns.stripplot()` with the same data using the seaborn package. Make sure you can see the data points!

What does one half of the violin plot represent?

In your option, which plot is a more informative of visualizing your data (this will not graded on correctness)?

Q3.3

The day that the titanic sank, was a bad day in many ways. Most importantly, because it helps present-day men to justify pro-male sexism with plots like this.



Can you recreate the bar plot with matplotlib? The example plot is made in R, so your Python plot will likely end up having a somewhat different style. That is fine as long as the basic setup and logic of the plot is the same.

When reading in the titanic csv file please do not use a path pointing to a location on your computer like "documents/titanic.csv" but rather read the data directly from the web (e.g., here https://raw.githubusercontent.com/hannesrosenbusch/schiphol_class/master/titanic.csv).

Q3.4

Load the wine dataset from the sklearn.datasets module. Make a scatter plot of the wines with `malic_acid` on the x axis and `alcohol` on the y axis. Add a regression line to the plot. Label your x-axis and y-axis with the variable names in large font size.

Q3.5

Make a plots of two subplots next to eachother. Plot a heatmap showing all the correlations between the variables in the wine dataset. Next to the heatmap, add a `kdeplot` with alcohol on the x axis and color_intensity on the y axis.

Q3.6

Define a function my_plots that takes a string as input (i.e., it has one argument) and generates a plot. If the string is "eww" it generates a confusing, non-informative plot. If the string is "yay" it prints a clear, informative plot. All other inputs lead to reminders that the function should be either used with "eww" or "yay".

Q3.7

Look through your solutions from any previous Python exercises where you wrote your own Python code. Improve style of of your code plots by following PEP8 (Python Enhancement Proposal 8) guidelines: https://www.python.org/dev/peps/pep-0008/. Note that you can use a pylint or another Python "linter": https://pylint.org/.

Also change variable names to be more informative and write some comments so that someone else can read your code.

Attach both the old code and new code here that is clearly labeled `# Old bad code` and `# New good code`. Explain what you did to change your code to PEP8 and discuss how else you improved your code. If your old code was already perfect explain why.

Q3.8

Many conventions and style guidelines for Python code are universally accepted. However, sometimes there is controversy about "the best way to do things". Find one of these issues and state in two brief sentences which side you are on.

Q3.9

In terms of **Python style** what are at least two things wrong with the following code (including comments)?

```
import numpy as numpyImport
import pandas as pandasImport

allData = {"year" : numpyImport.arange(-60,1,step=20),
            "people" : numpyImport.repeat(numpyImport.array(['Roman']),4),
           "government" :
             numpyImport.repeat(numpyImport.array(['republic', 'empire']), 2)}
myData = pandasImport.DataFrame(allData)
print(myData)
#    year people government
# 0   -60  Roman   republic
# 1   -40  Roman   republic
# 2   -20  Roman     empire
# 3     0  Roman     empire
```

Q3.10

Michael is a sloppy Python coder: https://github.com/mdnunez/pyhddmjags/blob/master/pyhddmjags utils.py Choose one of his functions to improve by using a pylint and creating better variable names and comments. Explain what you did to change his code to PEP8 and discuss how else you improved his code. If his old code was already perfect explain why.

Final project

Q4.1

General hints:

Boneau (1960) varied n across two levels; 5 and 15. He also varied σ^2 (variance, not SD!) across two levels; 1 and 4. Because n and σ^2 were varied independently for each sample, this yields 9 design cells altogether.

Actually, there is a further wrinkle here.... there is one cell that's special and needs to be considered in two different ways.

Each cell, in turn, involves sampling of data from a rectangular (uniform), Gaussian (normal), or exponential distribution, all of mean 0 and variance determined by level of σ^2. So altogether, this yields 27 different experimental conditions. (Plus the wrinkle just mentioned, which will change that number).

For the project, you should replicate those 27+ cells for starters, and then focus on those that are most interesting (i.e. greatest effects of violation of assumptions) and explore those by accentuating the problems (e.g., by increasing variance or decreasing n or whatever).

Specific hints:

The project can be solved using a t-test function from a Python module. However, it would be particularly meritorious if (time permitting) you also write your own function (e.g., `my_t_test`) that computes the t-statistics and other information and returns it in a same (or similar) structure as the Python module function.