420-SN1-RE Programming in Science - Lab Exercise 17

October 27, 2024

Introduction

The purpose of this lab is to learn more about `matplotlib`, including best practices for plotting data. Goals for this lab:

- Work with data in various formats
- Plot data with `matplotlib`
- Best practices for creating readable plots
- More practice with nested data structures (lists and dictionaries)

Files distributed with this lab:

- `lab17.pdf` - These lab instructions
- `top_languages.py` - Program to be modified for Exercise 1
- `top_languages.csv` - Data for Exercise 1
- `planets_graph.py` - Program to be modified for Exercise 2

Exercise 1: Plotting most popular world languages

Creating plots is an iterative process that combines technical Python knowledge with aesthetic design principles. In other words, sometimes we have to generate a bunch of plots and see what looks good!

Our goal for the first exercise is to generate a plot like Figure 1 by modifying and adding code to `top_languages.py`. Follow the steps given to produce this plot. These precise steps may not apply to all plots you create, however, they give you an idea of things to think about when generating a plot.

We have prepared some code (`top_languages.py`) to help you get started. However, certain pieces of this program will have to be modified as specified in the steps below.

1. **Understand the data.** We have provided you a dataset of the top-10 world languages by number of speakers as a CSV file (source: Wikipedia). CSV files are a common way to represent tabular data. They can be opened in a spreadsheet program like Microsoft Excel. However, they are ultimately text files. Lines in the text file delineate rows, and commas delineate columns.

   Open the text file in Notepad or in IDLE to view the data in its raw form. Now open the file in Microsoft Excel. See if you can understand how Microsoft Excel arranges the data into rows and columns.

   The data is organized into the following columns.

   - **Language (string):** Language name
   - **Family (string)**: Language families categorize languages that descend from a common language

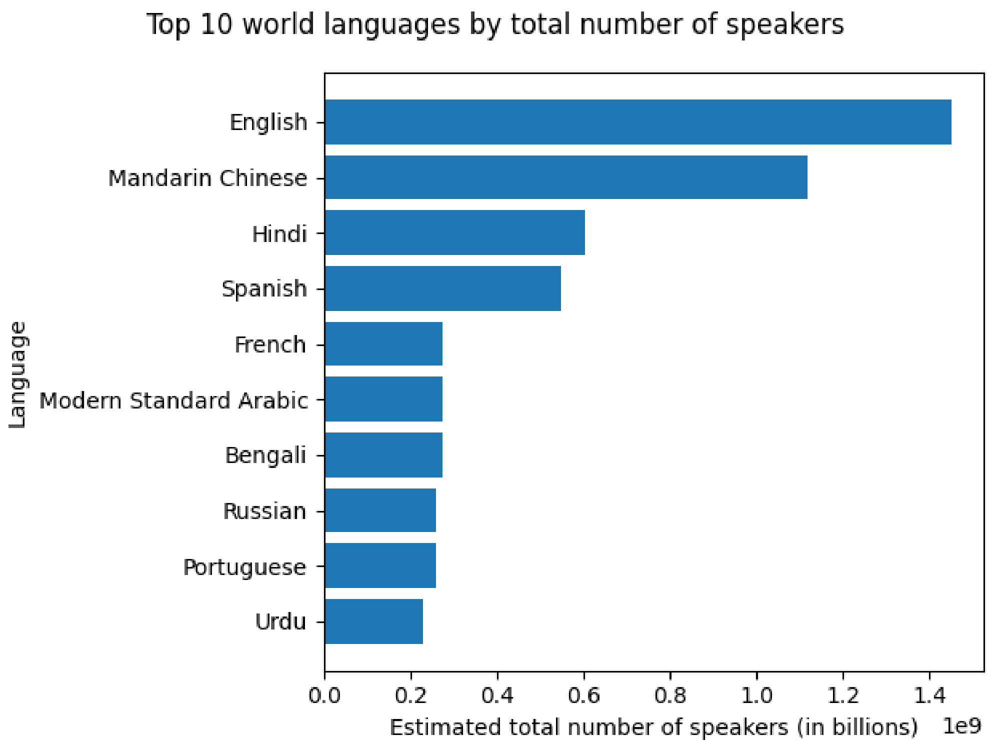


Figure 1: The desired graph for this exercise

- **Branch (string)**: A further division of language families into smaller units.
- **L1 Speakers (numeric)**: Estimated number of people who speak the language as a first language
- **L2 Speakers (numeric)**: Estimated number of people who speak the language as a second language
- **Total Speakers (numeric)**: Estimated number of people who speak the language in total (i.e., as either a first or second language).

Our goal is to create a bar chart depicting the total number of speakers for each language.

2. **Fix the function `top_languages_data()` that reads the data into Python**. We have provided you with a function definition `top_languages_data()` which, when called, should read the CSV file in to Python. The function is missing two crucial lines of code. Add them to finish the function.

3. **Call the function to read the data.** In the main section of the program (see comment specifying main section), write a line of code to call the function that you just finished. The line should give you the ability to use the data throughout the rest of the program.

4. **Visually inspect the data.** You want to ensure the data were correctly read into Python, otherwise any chart you produce will be meaningless or misleading. Compare to what you see in Excel.

   What data structure are we using to represent the data?

5. **Prepare the data for `matplotlib`.** Unfortunately, `matplotlib` cannot work with the data as currently structured. Recall that the `matplotlib` `plot()` function expects a list of values for the `x` (horizontal) axis and a paired list of values for the `y` (vertical) axis.

Write a function `extract(data, column_number)` that accepts two arguments: a data structure in the format you identified above, and a specific column number (integer) that should be extracted into to a list.

An example of the function call on an unrelated but similarly structured dataset is shown below:

```
prices = [["Power Drill", 119.99],
         ["Screwdriver", 20.99],
         ["Cordless Angle Grinder", 299.99]]
print(extract(prices, 0))
print(extract(prices, 1))
```

Should produce the following outputs:

```
['Power Drill', 'Screwdriver', 'Cordless Angle Grinder']
[119.99, 20.99, 299.99]
```

6. **Generate a first-draft plot.** As a first draft, we want to plot language on the x-axis and total number of speakers on the y-axis.

   (a) Call the function you just wrote (`extract()`) to extract the column pertaining to "Language" as a list. Call the function again to extract the column pertaining to "Total Speakers".

   We have provided you with constant names (in uppercase letters) at the beginning of the program. Recall, best programming practices dictate that you should use variables whenever possible (as opposed to hard-coding values such as column numbers)!

   (b) Double-check that your function worked correctly and that each list is in the correct order. In other words, the language at element 0 should correspond to the observation of total speakers at element 0. Errors can occur at any step of this process, and we want to be sure we are plotting the correct data.

   (c) Write a line of code to plot the two variables as a barplot. `matplotlib` includes a method `.bar()`. To understand how to use this method, try running `help(plt.bar)` in IDLE and reading the output. Make sure you have imported `matplotlib.pyplot` as `plt` first!

   (d) Remember, to see the plot, you must run `plt.show()`

   Is this plot readable? Think about what kind of improvements we could make before moving on to the next instruction.

7. **Flip the axes to increase readability.** When generating a plot where the x-axis depicts categories with potentially long names, the plot can be difficult to read. We can increase readability by flipping the x- and y-axes.

   You might think you could just change the order of arguments to `plt.bar()` to flip the axes. However, we need a different plot method here. Discover how to use this new plot by entering the `help(plt.barh)` command in IDLE and reading the output.

   Now, create the horizontal barplot. Is this easier to read?

8. **Good plots tell a concise story.** What story do we want to tell with the illustration of the data?

   Our viewer probably wants to see which language(s) have the most number of speakers and easily compare the number of speakers across the most popular languages.

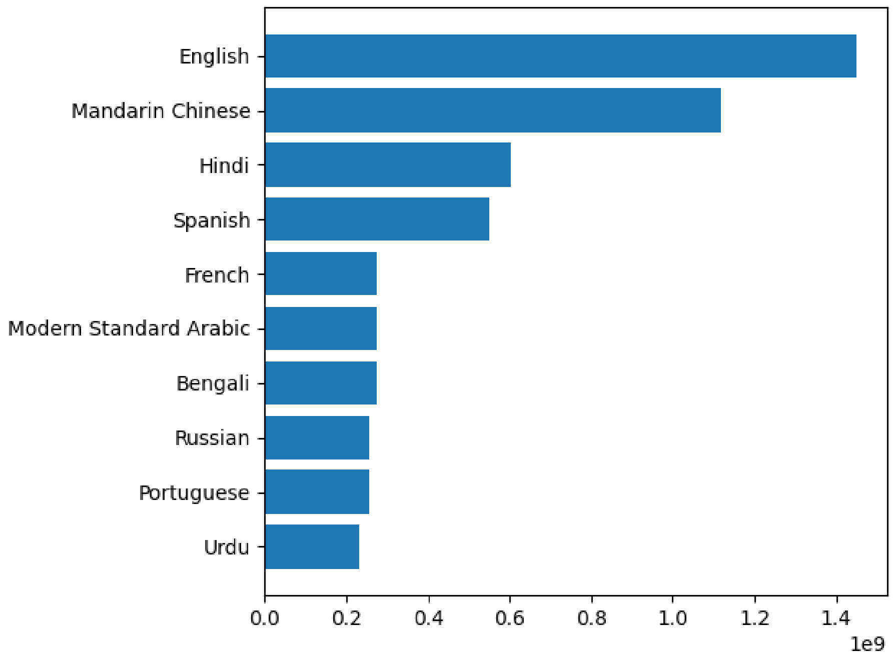
This story is contained within our plot in its current state, but it is not readily accessible to out viewer. Let's reorder data to give our story structure: a beginning, middle, and end. We will arrange our data in ascending order, such that languages with the fewest speakers come first.

Importantly, in order to do this correctly, we must sort the languages list according to the sorted order of the list holding total number of speakers. If we sort the lists independently, we break the relationship between our two lists.

We have provided you a function, `sort_2_lists()` that sorts two lists. Use this function to sort your lists:

```
y, x = sort_2_lists(y, x)
```

At this point, your plot should look like the following.



9. **Finishing touches: Labels**. Our plot is looking pretty good! It tells a clear story about popular world languages. However, there are some finishing touches to add. Good plots have axis labels (with units) and titles. Add some lines of code to add these elements.

   `matplotlib` has the following methods to add these elements to your plot. Remember to understand how they work, you can call the `help()` function on them.

   - `plt.xlabel()`
   - `plt.ylabel()`
   - `plt.legend()`
   - `plt.title()`
   - `plt.suptitle()`

10. **Finishing touches: Font size**. Good plots have readable axes and titles. Often, the default font size is appropriate for viewing on the screen, but may not be appropriate for displaying on a scientific poster or manuscript. You can alter the default font size of a plot by calling the `plt.rcParams.update` method as follows:

    ```
    lt.rcParams.update({'font.size': 14})
    ```

11. **Finishing touches: Final layout**. Sometimes calling the `plt.tight_layout()` method just before calling `plt.show()` can improve the appearance of your plot.

Exercise 2: Graphing Planetary Orbits in our Solar System

You have been provided a file, `planets_graph.py`, that includes a dictionary with some important data about the major planets in our solar system and their orbits around the Sun.

```
# "radius": km ,"distance_sun": km and "orbit_period":days
planets = {
    "mercury":{"radius": 2439.7,"distance_sun": 5.800e7,"moons": 0,"gas_planet": False, "orbit_period":88.0},
    "venus":{"radius": 6051.8,"distance_sun": 1.082e8,"moons": 0,"gas_planet": False,"orbit_period":224.7},
    "earth":{"radius": 6371,"distance_sun": 1.496e8,"moons": 1,"gas_planet": False,"orbit_period":365.2},
    "mars":{"radius": 3389.5,"distance_sun": 2.279e8,"moons": 2,"gas_planet": False,"orbit_period":687.0},
    "jupiter":{"radius": 69911,"distance_sun": 7.785e8,"moons": 79,"gas_planet": True, "orbit_period":4331},
    "saturn":{"radius": 58232,"distance_sun": 1.433e9,"moons": 83,"gas_planet": True,"orbit_period":10747},
    "uranus":{"radius": 25362,"distance_sun": 2.871e9,"moons": 27,"gas_planet": True,"orbit_period":30589},
    "neptune":{"radius": 24622,"distance_sun": 4.495e9,"moons": 14,"atmosphere": True,"gas_planet": True,"orbit_period":59800}
}
```

Note that the units for this dictionary are *kilometers* for `'radius'` and `'distance_sun'` and *days* (that is, Earth days) for `'orbit_period'`.

To analyse the relationship between the orbital period (T) of each planet's orbit against its average distance from the Sun (a), we will start by graphing one vs the other. When studying motion in our solar system, it is common to use some specific units of measurements:

- The average distance of a planet from the Sun is measured in *astronomical units* (AU), which is the average distance from the Earth to the Sun. 1 AU = 1.496×10^8 km
- The amount of time a planet takes to orbit the Sun is measured in Earth years. 1 year = 3.156×10^7 s

Modify the `planets_graph.py` program so that it creates the lists needed for plotting from the dictionary and uses `matplotlib` to generate a plot of T in years as a function of a in AU. You can use the dictionary's values for Earth's data to convert the units to AU and years.

- Annotate the graph with appropriate titles, and add a legend if necessary.
- `matplotlib` changes the axes to make the data a readable as possible. Sometimes this is a problem for seeing the shape of the correlation between datasets. In this case use the `plt.axis('square')` command before you call the plot to make sure the axes are of the same size.

Kepler's third law

Note that the graph that you produced is not linear. In orbital dynamics, Kepler's third law states that the square of T, the period of a planet's orbit around a star is proportional to the cube of a, the average distance from the star. Mathematically, it can be written as:

$$T^2 \propto a^3$$

The symbol $\propto$ means "is proportional to". To verify Kepler's third law, create a second graph. Add code so that you also plot T^2 versus a^3.

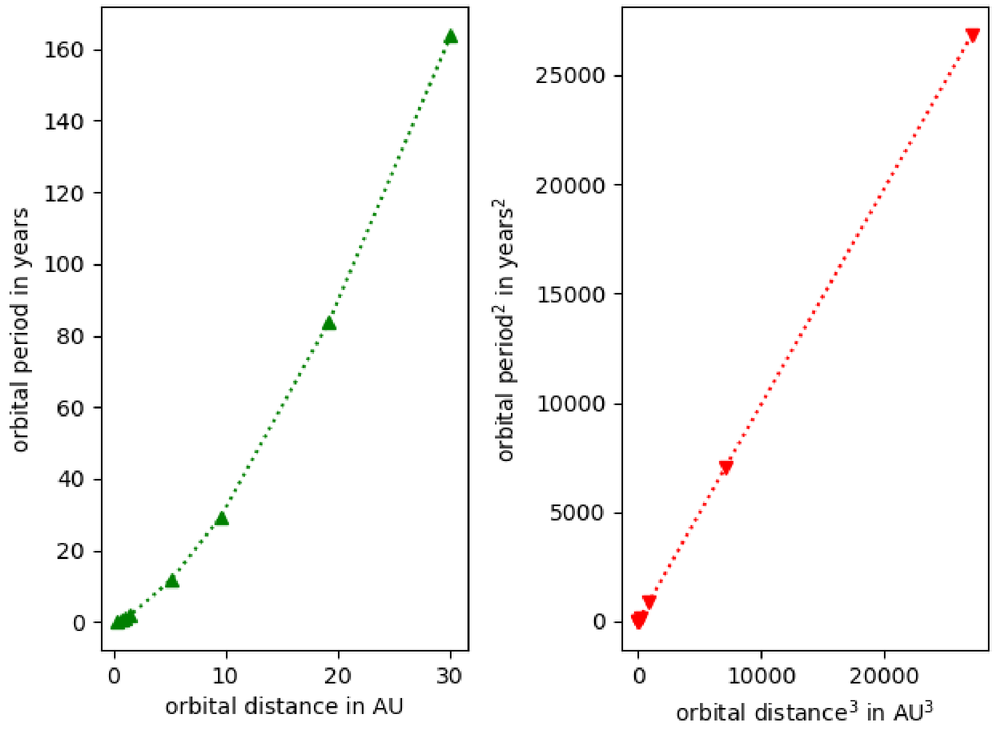
Displaying both graphs at the same time.

When you run your code, note that it creates one graph and waits until you close it to show you the other graph. To make it easier to analyse the results it is interesting to show both graphs at the same time. To do this we will make both of them subgraphs.

1. Remove the `plt.show()` from the first graph and only have it once at the end of your code.
2. Before you call `plt.plot(x, y)` for each dataset include either `plt.subplot(1, 2, 1)` for the first graph or `plt.subplot(1, 2, 2)` for the second graph.
3. The graphs overlaps each other, call `plt.tight_layout()` before calling `plt.show()`.

Expected Outcome

The first graph should show an accelerating trend as distance from the Sun increases. The second graph should be a straight line, confirming that T^2 is proportional to a^3. It should resemble the following:



For more information

For more information on `matplotlib` and `pyplot`, see `https://matplotlib.org/stable/tutorials/pyplot.html#sphx-glr-tutorials-pyplot-py`.

Submission requirements

Combine your Python files into a single zip file and upload them to Omnivox. Be sure to submit a *single* file containing all of your work.