BST273: Final Project (General Specifications)

Version: 2018-10-22

Overview

You will be completing and handing in either 1) the default final project (scatter.py) or 2) a custom project of your own design. This document summarizes the hand-in requirements that are common to both of these final project options, as well as the specific implementation details for the default vs. custom final projects.

Canvas Hand-in

You will hand in a SINGLE ARCHIVE containing multiple files via Canvas. The archive can be in either ZIP or TAR (or TAR.GZ) format. Contact the instructors (Eric and Kevin) if you have trouble producing such an archive. A typical archive will contain four files:

1. A README.txt file
2. Your Python script
3. A sample input file(s)
4. A sample output file(s)

The formats for the individual files are detailed below.

Github Hand-in

You will also hand in your final project files via a PRIVATE repository on Github (you are welcome to make the repository public AFTER the Fall 1 semester has concluded).

- In order to create private repositories, you must sign up for the (free) Github Student Developer Pack.
- You can create the final project online by following Github's instructions for creating a new repository.
- Invite the instructors to join your private repository as collaborators: Under Settings (gear icon) → Collaborators → Add users "kescobo" (Kevin) and "franzosa" (Eric).
- Add files to your repository by cloning the remote repository to your computer ("`$ git clone` *`url`*"), copying files into the newly-created folder, adding those files to the repository ("`$ git add` *`files`*"), making a commit with the files added ("`$ git commit -m` *`'message'`*"), and then pushing the commit to your remote repository ("`$ git push origin master`").
- You are encouraged to make periodic commits and push the changes as you work on the final project. Be sure to commit and push the final version of your project (the same files that are uploaded to Canvas).

NOTE: You are only required to push the FINAL version of your project to your remote repository to receive credit for this portion of the assignment. However, any time you contact the instructors for help with the final project, it will be useful 1) to have committed and pushed the most recent version of your code and 2) to share the repository URL with the instructors.

NOTE: As mentioned in class during our lectures on using git and Github, if you have objections to creating a personal Github account, contact the instructors to make alternative arrangements for this portion of the final project.

Final Project Files

The README.txt File

Your README.txt file provides an explanation of how to run your final project script, as well as answers to a number of questions about the final project and your experience in the course. A template README.txt file is provided with the assignment module. We recommend simply filling out this template and turning it in with your final project. The README.txt file must address the following items:

1. List your name, email address, and the URL for the Github repository that contains you final project files.

2. Summarize your experience working on the final project. For example, you might approximate how many hours you spent on it and how those hours were distributed. If you found some aspects considerably harder than others, list those here. If there are known problems with your final project script, list those here.

3. In a few sentences, describe what your final project script does.

4. List any modules (outside of the Python standard library) that are required to execute your final project script. You may answer "N/A" if no such modules are required.

5. Describe your sample INPUT FILE(S). If you are completing a custom final project that does not require an input file, explain that clearly here.

6. Provide the command used to produce your sample OUTPUT FILE with flags and arguments specified (e.g. "`$ python script_name.py arguments`").

7. Describe your sample OUTPUT FILE(S). If you are completing a custom final project that does not produce an output file, capture the STDOUT of the command specified above and include it here (e.g. "`$ command > sample_stdout.txt`").

8. What was your favorite part of learning to program in BST 273 (i.e. something we should definitely NOT change in future incarnations of the course)?

9. What was your LEAST favorite part of learning to program in BST 273 (i.e. something we should look into changing for future incarnations of the course)?

Your Python Script

Final-project Python scripts have the following general requirements:

- Your Python script should be written using Python 3 syntax.

- If you are using any modules not included in the Python 3 standard library, make sure to specify them in your README.txt file.

- You may title your script anything you like, although our assumption is that students completing the default final project will name the script `scatter.py`.

- All scripts must include an `argparse`-style command-line interface. Details of expected command-line flags are outlined in the specifications of the default final project or in consultation with the instructors (for custom final projects).

- Use the `argparse` description field to indicate what your script does when running with "`--help`". Feel free to use the same description included in your README.txt file.

- Your script MUST execute without errors when run with "`--help`" or when executing the sample command you specified in your README.txt file.

- Your script SHOULD execute without errors when run with reasonable permutations of the sample command.

- Include comments in your code to describe the function of major blocks of code (including functions) as well as "tricky" lines.

Sample Input File(s)

These will vary by project. Use the README.txt file to indicate any specific details of your required input files, including if there is more than one. The format of the input file for the default final project is included in default specifications and a sample input file is provided in the Final Project module on Canvas.

Sample Output File(s)

See above.

Default Final Project Specifications

Version: 2018-10-22

Default Final Project Overview

You will create a Python script called `scatter.py` to generate scatter plots based on columns of data from TSV-format input files. The script will have two "modes": a default mode in which two columns of continuous data are used, and a "stratified" mode in which a third column of categorical data is used. In stratified mode, your script should add a separate "series" (set of dots) to the same plot for each level of the categorical data and note their names in a legend.

Input Data

Your script should accept TSV-format data as input. You can assume that the data conform to the specifications of a "data frame":

- The top row contains the names of "fields" from the data frame.
- Fields (columns) store a particular measurement (property) for a set of observations. Measurements can be continuous (numbers) or categorical (text labels).
- Each row after the first contains the measurements for a particular observation.

You may assume that the data frame is well-formed: i.e. there are no missing values, no text-values in otherwise numerical columns, all columns have the same length, etc.

An example input file (the famous "Iris" dataset) is provided with the Final Project module on Canvas. The first five observations of this dataset are shown here:

sepal width (cm)	sepal length (cm)	petal width (cm)	petal length (cm)	class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5.0	3.6	1.4	0.2	Iris-setosa

The final field ("class") is a categorical field; the first four fields are all continuous.

Command-line Interface Requirements

Your script MUST implement a command-line interface with the following options:

- Path to the input file (always used).
- 1-based index of the column containing continuous x-values to plot (always used).
- 1-based index of the column containing continuous y-values to plot (always used).

- 1-based index of the column containing categorical “series” values on which to stratify (optional: if not specified, plot all x- and y-values as one series).
- Path to the output file (optional: if not specified, save the plot with a reasonable default name in the working directory).

NOTE: You may add additional options with reasonable defaults, for example a flag to set the title of the plot, flags to override the axis labels, flags to specify the x- and y-axis limits, etc.

NOTE: You may assume that the user is picking fields for plotting sensibly. For example, it is the user’s problem if they select a categorical field to use as x-values in plotting.

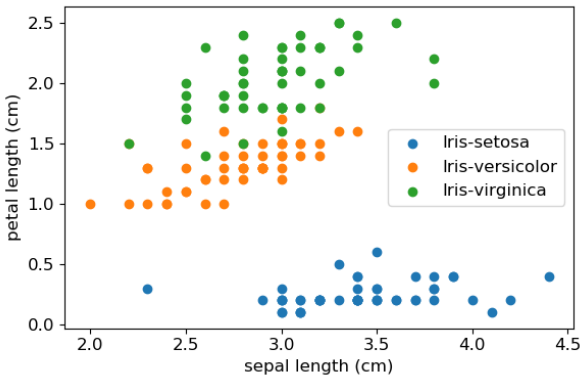
Miscellaneous Requirements

- Use the headers of the selected x and y columns to label the x- and y-axes of the plot.
- If “stratified” mode, your script should include a legend with the plot to associate the colors of plotted points with their labels.

Recommendations

- Use the Python `matplotlib` package (bundled with Anaconda 3) for plotting. There are numerous helpful tutorials available online for `matplotlib`.
- It may be more convenient to think of the script as always acting in stratified mode, adding all points to a single “default” series when not formally stratifying.
- Start by setting up your command-line interface so you have all of its arguments available when needed. Then implement the data loading portion of the script. Then try to produce a basic (default) plot of y- versus x-values. Then implement the stratification option. Add bells and whistles (axis labels, legend, etc.) last.
- Set the default values of optional parameters to **`None`**. Evaluating **`args.param is None`** will then return **`True`** if the parameter was not set.

Example Stratified Output



Custom Final Project Specifications

Version: 2018-10-22

Custom Final Project Overview

Custom final projects are designed in collaboration with the instructors. They can involve a topic in scientific data analysis, or something non-academic (e.g. code to support a hobby or outside interest). Regardless, you MUST seek and receive instructor approval before beginning a custom final project to ensure that your idea is of sufficient yet feasible scope.

To receive instructor approval, email a short description of your project to the instructors (Eric and Kevin) with details of how you will meet the requirements outlined below. It is likely that the instructors will iterate with you one more times to refine the scope of the project.

Requirements

Each custom final project MUST:

- Implement a non-trivial command-line interface.
- Produce some sort of verifiably correct output (at a minimum, some sort of information must be printed to the screen that will vary with choice of input data or parameters).
- Use some element of Python code that was not specifically covered in class.
- Conform to all general Final Project requirements.