STAT 215A general lab instructions

Anthony Ozerov

September 20, 2024

These are the **general** lab instructions, applying to all 215A labs. Each lab will also have lab-specific instructions.

Contents

These instructions are based on previous lab materials by Chengzhong Ye, Theo Saarinen, Omer Ronen, James Duncan, Tiffany Tang, Zoe Vernon, Rebecca Barter, Yuval Benjamini, Jessica Li, Adam Bloniarz, and Ryan Giordano.

This is the first time STAT215A is being taught in Python, so these instructions have been heavily adapted from previous years. They are provisional and will likely be modified throughout the course as we progress through the labs and get feedback.

1 Obtaining lab materials and setting up lab directory

Copy over the lab materials from the `stat-215-a-gsi` repository to your own repository `stat-215-a`. i.e. the `stat-215-a-gsi/labX` directory should be copied to `stat-215-a/labX`. **Do not** do the lab in some other folder on your computer and copy it to your repo at the last moment. It is easiest to just do the whole lab within the repository directory.

2 Installation / environment setup

Unix environment

It is essential that you use a Unix environment for this class. MacOS and Linux are both Unix environments. If you are using Windows, I *highly recommend* installing Windows Subsystem for Linux (WSL) (if you haven't already) and doing everything there. This will give you access to a Unix environment on your Windows machine. If you would like to install WSL, here some resources:

- Installing WSL: `https://learn.microsoft.com/en-us/windows/wsl/install`
- Using VSCode on Windows with your WSL setup: `https://code.visualstudio.com/docs/remote/wsl`

In principle, you don't *have* to use a Unix environment. But using one will make things easier. And if something isn't working in your non-Unix environment, I likely won't be able to help you.

Conda

If you haven't already, install `miniconda` or `conda` (don't install Anaconda).

Making/updating an environment from the lab's environment.yaml file

Set up your 215a environment using the provided `labX/code/environment.yaml`. First, take a look at the file to see how it is formatted; it is just a plaintext list of dependencies. Navigate to your `labX` directory in your own `stat-215-a` repo. If you don't yet have an environment named "215a" (you can check using `conda env list`), run this command:

```
conda env create -f code/environment.yaml
```

This will create a new environment named "215a" with all of the packages you need. If you already have a 215a environment, you can update it with:

```
conda env update -f code/environment.yaml
```

This will update your environment to add the packages listed in the `environment.yaml` file.

Checking your environment

Now let's check that the environment works:

1. Enter your environment using `conda activate 215a`.
2. Run the command `jupyter lab` to create a Jupyter Lab server
3. Open a new notebook
4. Run `import sys` then `print sys.executable`. The output should be a Python executable associated with your `215a` environment, and should end in `215a/bin/python`.

Making a conda environment for R

If you would like to also use `R` (e.g. for figures), you can install it as well. These days you can install `R` and `Rstudio` using `conda`. You don't have to, but I would recommend making a separate conda environment for `R` to avoid any conflicts. There is also an `code/environment-r.yaml` file that you can use to set up an `R` environment. You can create a new `215a-r` environment with:

```
conda env create -f code/environment-r.yaml
```

If you run `conda activate 215a-r` you will be able to use `R` from the command-line. If you install `jupyterlab` in this environment (`pip install jupyterlab`), you'll be able to use it in Jupyter notebooks too. The environment also contains an installation of `Rstudio`, which you can run with `rstudio` from the command line while you are in the `215a-r` environment.

MacOS issues

Note: Recent Apple devices use an arm64 architecture as opposed to an x86_64 architecture. Many conda packages are not compiled for arm64, so you may run into issues when installing packages. You may be able to get around this by running `conda config --env --set subdir osx-64` after creating a new, empty conda environment. e.g. you might be able to do:

```
conda create -n 215a
conda activate 215a
conda config --env --set subdir osx-64
conda env update -f code/environment.yaml
```

3 Do the lab

Come to OH if you have questions, or ask on Ed. The different labs will have specific requirements, in terms of PCS documentation, on what the report needs to contain.

General report-writing guidelines

Note these instructions from the syllabus: "The assignments require two full weeks of work to satisfactorily complete, which requires a very early start.... For the data labs, each student will produce a **12-page (maximum)** report presenting a narrative that connects the motivating questions, the analysis conducted and the conclusions drawn. The labs will be completed in Python (optionally some parts in R, especially for visualization). The reports will be made using Jupyter Notebooks or pure LaTeX and the final pdf output should not contain any code whatsoever."

"The data labs will be done individually, except for one group lab later in the semester. The goal of writing the lab report is not only to gain data analysis experience but is also an exercise in communication. We ask that particular attention is given to the writing of the report, as your peers will be reading them: so

that the students can learn from one another, the labs will be peer-reviewed. Each student will review 2-3 labs from their peers, and will provide feedback and a grade based on several criteria including clarity of writing, validity of analysis and informativeness of visualizations. The final grade of each lab will be decided by the GSI who will use the student grades as a guide."

Also take note of these guidelines:

- You must make an effort to communicate effectively. Think as if you are writing a blog post or an informal journal article.
- The data from each lab comes from an existing research paper, which will be given to you. You must also make an effort to incorporate domain information and knowledge in the writeup to give the report some context. For example, it is good habit to explain in the introduction why your problem is important within the domain, to describe any connections between the statistical models/algorithms and the true phenomenon at hand, and to conclude with a discussion of the impacts of the results within the domain context. Ideally, domain knowledge should be incorporated at all stages of the data science pipeline.
- Favour simplicity over complexity. It is much more important to be thorough and to communicate effectively than to come up with some super fancy modeling idea that no one understands. If a super fancy is needed or justified, then feel free to go for it.
- Keep in mind that there are two types of visualization: *exploratory* and *explanatory*. Exploratory visualizations are graphics that you produce to help *you* understand the data, whereas explanatory visualizations are final versions of a small subset of these figures that you produce to explain to *other people* what is in the data. Typically you will produce many, many exploratory plots and only a few key explanatory plots that answer specific questions. Choose your explanatory plots carefully, and ask the following question of every figure in your report: "Does this figure add anything? Is my story strictly worse when I remove it?" If the answer to either question is "no", then you should remove the figure. Just because you spent a lot of time making a really pretty figure, doesn't mean that it adds anything to your story. There have been many times in my life where I have spent an hour or two making a really awesome plot only to decide the next day that it is actually fairly irrelevant to my main points and removing it.

Jupyter notebook or LaTeX

Your lab report can be written in either a Jupyter notebook (`labX.ipynb`) or a LaTeX document (`labX.tex`). If you are more industry-minded (or adventurous!) I recommend trying in Jupyter notebooks, and if you are more academia minded I recommend trying in LaTeX. In either case, you should convert the final product to a PDF (`report/lab0.pdf`).

If you would like, you can write your report as `.qmd` instead of `.ipynb` or `.tex`, or even as MyST markdown. But I won't be able to help you if you have problems! Please don't use `.Rmd`.

LaTeX instructions

If you do the lab in LaTeX, you can use Overleaf (especially for group labs), but for the individual labs I recommend installing LaTeX locally and editing your `.tex` files locally. This just makes it easier to work with figures generated by your code (if you re-run some code and get a new figure, you can just recompile your `.tex` file to see it in your document). Feel free to ask me about setting up LaTeX on your computer.

Jupyter notebook instructions

If you do the lab in Jupyter notebooks, here are some useful commands (to be run from your Unix command-line) for exporting them to PDF. The first converts a notebook to a pdf without running it again, and the second converts a notebook to a pdf after re-running it. You will need to put the pdf output in the `report` directory of your lab folder manually unless you can figure out some other wizardry.

```
quarto render labX.ipynb --to pdf
quarto render labX.ipynb --to pdf --execute
```

`quarto` is a software which, among other things, can convert Jupyter notebooks to much nicer PDFs than Jupyter's built-in converter. You can install it in your `215a` environment with `conda install -c conda-forge quarto`, or install it in some other way on your system. NOTE: if quarto is used to execute a notebook, it must be run from within the appropriate Conda environment.

Quarto also has a lot of tools for customizing the PDF output, (defining a title, adding figure captions, etc.) which are not otherwise available in Jupyter. You can read more about using Quarto with ipynb files here: `https://quarto.org/docs/tools/jupyter-lab.html`, and here: `https://quarto.org/docs/get-started/authoring/jupyter.html`.

I think converting a Jupyter notebook to PDF (using Quarto or otherwise) requires you to have a local LaTeX installation. I think quarto can install a small LaTeX installation for itself with `quarto install tinytex`. If you have trouble with this, please talk with me.

If you have interactive elements in your Jupyter notebook, feel free to also export it to HTML (`quarto render labX.ipynb --execute`) and include that with your report, so I can play around with it!

There is a small example notebook in `stat-215-a-gsi/disc/week1` which you can look at to see how to set up a .ipynb for use with Quarto and also display figures when rendering with Quarto.

No code in PDF report!

In any case, **Your PDF report must not contain any code**. For converting with Quarto, here is an example first-block in your .ipynb (which should be a "Raw" block as opposed to "Python" or "Markdown") which defines a title and prevents code from being displayed in the PDF:

```
---
title: "STAT 215A Lab X"
execute:
  echo: false
---
```

To clarify, it is okay if your `.ipynb` contains code (it would make sense if it does). The key is that the code should not be displayed in the final PDF report, which we can accomplish by just controlling how Quarto does the rendering.

Coding language

Primarily in Python!

In general, **labs must be completed primarily in Python**. However, you are welcome to do components of your labs in R, especially visualization, if you prefer. Using the right tools for the right tasks is an important skill. For most things in this class, Python is the right tool.

Interplay between R and Python

If you do use both, you should set up the interplay between them cleanly. In general, to avoid confusing bugs and dependency issues, it is good to keep R and Python code completely separate, only passing data between them through intermediate files (e.g. .csv). For example, you can easily make R plots of your Python analyses:

- Export data from Python to a CSV file
- Load it in R, and possibly reshape it somehow (see `dplyr`)
- Use `ggplot` (or other plotting methods) to make a figure.
- Save the figure in the `figs` directory of your lab.

Note that your final R code should be able to run as a script, saving figures on its own in appropriate directories. You **cannot** just load the data in a .Rmd file, manually run cells, and save the plots manually when they appear. See this stackoverflow post for how to save R figures: `https://stackoverflow.com/questions/7144118/how-can-i-save-a-plot-as-an-image-on-the-disk`

Loading figures created in R

If you are using a Jupyter notebook for the report, you can use the following code to display the figure you made in R:

```
from IPython.display import Image
Image(filename='../figs/plot.png')
```

If you are using a LaTeX document for your report, you can include the figure using the following code:

```
\usepackage{graphicx}
...
\begin{document}
...
\begin{figure}
    \centering
    \includegraphics[width=0.5\textwidth]{../figs/plot.png}
\end{figure}
```

Coding requirements

LLM usage rules

Each lab will describe the policy on using LLMs for coding the lab.

Style requirements

Your code should not be ugly. I recommend following the Google Python Style Guide: `https://google.github.io/styleguide/pyguide.html`. Pay special attention to guidelines for the names of variables, functions, and classes. Variable names in Python should be in snake_case, not CamelCase! You don't have to follow the style guide super strictly, but you should write clean, readable code, and the Google one is a good guide to do so. There are also many tools to auto-format the whitespace of your code to make it look nice, like `black`.

Documentation requirements: comments and docstrings

Your code should be well-commented.

- There should be comments throughout which explain what is going on.
- Use docstrings for functions and classes. Feel free to follow any style guide for this.

(Hint: LLMs are pretty good at documenting code you've written!)

4 Submission Requirements

Report length

The report should be a maximum of 12 pages, including figures, tables, etc. and excluding bibliography.

Lab folder structure

Your `labX` folder should have the following structure:

```
labX/
    code/
        run.sh
        environment.yaml
        environment-r.yaml (optional)
        (labX.ipynb)
        (some_script.py)
        (some_other_script.py)
        (some_exploratory_notebook.ipynb)
        ...
    data/
    documents/
    figs/
    other/
    report/
        (labX.tex)
        labX.pdf
```

- `code` should contain all of your code, including any Jupyter notebooks.
- `data` should contain any data that you use, including data which we give you and intermediate outputs from your code. **Do not commit data to your repo**. When testing your code, I will put the data in `stat-215-gsi/labX/data` into your `stat-215-a/labX/data`.

- `documents` should contain any relevant documents, including papers which we give you or which you use.
- `figs` should contain any figures that you generate which are included in a LaTeX report.
- `other` should contain any other relevant files.
- `report` should contain your lab report, in both `.tex` and `.pdf` formats. Any figures in your report should be pulled from the `figs` folder.

If you didn't use them, it is okay if the `documents`, `figs`, and `other` folders are not present.

No name or identifying information

Your lab should not contain any identifying information. However I recommend in this class that you also make versions with your name, so you can show them to other people (employers?) later.

Reproducibility

Your results need to be reproducible. That means I should be able to easily run your code and produce all of the figures/information needed to make your report. **You will be graded on reproducibility**. Below are some specific requirements which ensure I can reproduce your environment and run your code in the correct order. **I will not grade harshly on environment.yaml and run.sh**, they are just there to make sure I can reproduce your results using your code.

environment.yaml

`code/environment.yaml` should be a yaml file listing the dependencies of your code. I have provided one for you to start with, but make sure to add to it any dependencies you install. Any dependencies you installed with `conda` should be added to the list under `dependencies`, and any dependencies you installed with `pip` should be added to the sublist under `pip`. Please do this so that I can recreate your environment and run your code without any issues!

You can test that your environment works to run your code by making and activating a new test environment, then running your code within it:

```
conda env create --name 215a-test -f environment.yaml
conda activate 215a-test
```

You can do likewise with your `215a-r` environment and `environment-r.yaml` if you are using one.

run.sh

`run.sh` should be a shell script which runs all of the code needed to generate your results. It can be something as simple as:

```
#!/bin/bash
conda activate 215a
python main.py
```

This will work if you just have a `main.py` script which produces all of your results, which are then loaded in a .tex file to make your report. If you are using a Jupyter notebook for your report and running the notebook produces all of the intermediate results and figures you need, your `run.sh` could look like:

```
#!/bin/bash
conda activate 215a
jupyter nbconvert --to notebook --execute --inplace labX.ipynb
```

The final command here is equivalent to opening the notebook in Jupyter Lab, hitting "Restart Kernel and Run All Cells", and saving the notebook. The command runs all of the cells in the notebook in order, and saves the output in the notebook itself.

You could also have something more complicated like this, if you are using a mix of Python and R and have some scripts producing intermediate data/results:

```
#!/bin/bash

conda activate 215a               # activates your 215a environment
python process_data.py            # runs your Python data processing script
python main.py                    # runs some main script which produces some outputs
conda deactivate                  # deactivates your 215a environment

conda activate 215a-r             # activates your 215a-r environment
Rscript --no-save make_figures.R  # runs an R script to make some figures
conda deactivate                  # deactivates your 215a-r environment

conda activate 215a               # reactivates your 215a environment
# executes a Jupyter notebook containing some analysis
jupyter nbconvert --to notebook --execute --inplace labX.ipynb
```

Don't worry about including code to make a PDF of your report in `run.sh`.

Again, I will not grade harshly on **run.sh**. The most important aspect of it is that it defines what order to run your code in. If you'd like, feel free to also experiment with using a Makefile to define how your code should be run.

Note: you may have an issue with running `run.sh`, seeing an error like `CondaError: Run 'conda init' before 'conda activate'`. Due to some arcane Conda details, shell scripts containing `conda activate` may not work when you try running them directly with `bash run.sh` or `./run.sh` depending on your particular setup. To get around this, try running it with `source run.sh` or `bash -i run.sh` or `zsh -i run.sh`. If for whatever reason you cannot get it to work, and you are sure you can run all of the commands in `run.sh` manually in the appropriate order, that is okay, as I should be able to run it myself.