Bayesian Statistical Methods and Data Analysis

- Introductions
- Logistics
- Structure of the course
- Intro to Python and JupyterHub

Introductions

Logistics

Slides and notebooks available on Moodle and github.

Attendance is not required but strongly recommended. I will try to keep a zoom connection open but the plan is to keep the course interactive with a lot of hands-on practice, so attending remotely (or not at all) will probably not be as useful.

Grades

If you want credit points, the grade is going to be based on a small research project & report.

- The project should either cover a data analysis or the theory behind a statistical tool.
  - This can be one of the project ideas I will suggest throughout the course or some data or algorithm you find interesting and want to explore more, for example from your own research.
- Format: like a journal article (specifically a Letter):
  - Professional presentation, i.e. using LaTeX, nice plots
  - Proper references and appropriate structure
  - Page limit of 5 (excluding references)
- Group work is allowed. Conditions:
  - Max group size: 4
  - Everyone in the group gets the same grade
  - Everyone needs to contribute to both the report and the analysis
  - The report needs to include a contribution statement that briefly describes what each member has done

Deadline is 2023/07/21.

Structure of the course

- Lesson, work on exercises, discuss solutions
- To get most out of the course, have a look at the material and exercises **before** the lecture
- Learning by doing: we will implement many of the basic methods ourselves to get an intuition for how they work
  - After that using well-developed and robust libraries is usually preferable

- Working in teams is encouraged
- Be ready for Clicker questions

- Feedback is always welcome!
  - EduApp: Course channel "Feedback"
  - Anonymous Google form (link on Moodle)
  - Email
  - In-person
  - Official course evaluation on the last day of the course: Friday 23.06 at 17:00

What is this course about?

Statistical inference

Statistical inference is concerned with drawing conclusions from data about quantities that are not observed.

Bayesian data analysis

The core of Bayesian data analyses is coming up with a probabilistic model of the world. These models describe both the observable and unobservable quantities of a problem in probabilistic terms.

By conditioning the probabilistic model on the observed data, we then obtain the probability distribution of the unobserved quantity of interest: the posterior distribution.

Computational statistics

Outside of simple models and special cases, our models will likely rely on numerical methods. This is especially the case in Bayesian data analysis, where drawing samples from complex distributions is often a key part of the analysis.

Literature

- Practical Statistics for Astronomers, Wall, 2012 ETH library. Short, with a focus on practical applications. Many of the examples and exercises are from astrophysics but are generally

applicable, especially for the physical sciences, which often come a bit short in general statistics textbooks. Solutions and data sets are available online.
- Bayesian Data Analysis, Gelman, 2013 ETH library. The title says it all.
- Information Theory, Inference, and Learning Algorithms, MacKay, 2003 Link. Heavy on the information theory but also covers inference methods nicely. The exercises come with solutions.
- Weighing the odds, a course in probability and statistics, Williams, 2001 ETH library. A good introduction to probability theory and statistics with a high level of mathematical rigour.

Python, Jupyter notebooks, and JupyterLab

The course focusses on computational methods to do data analysis with Bayesian statistics.

We will be using Python. There are other options, such as R and Julia, but once the models become more complex, Python has a much larger ecosystem available. For example astrophysics software or deep learning libraries.

Jupyter notebooks

A key part of a statistical analysis is understanding your data. Data exploration and visualisation is made easy by using Jupyter notebooks, which allow combining code, visualisations, and documentation in one place.

They are not good for complex modelling however. Once your code goes beyond some simple data analysis and plotting, splitting the modelling code into standalone modules and packages makes things more robust and maintainable.

JupyterLab

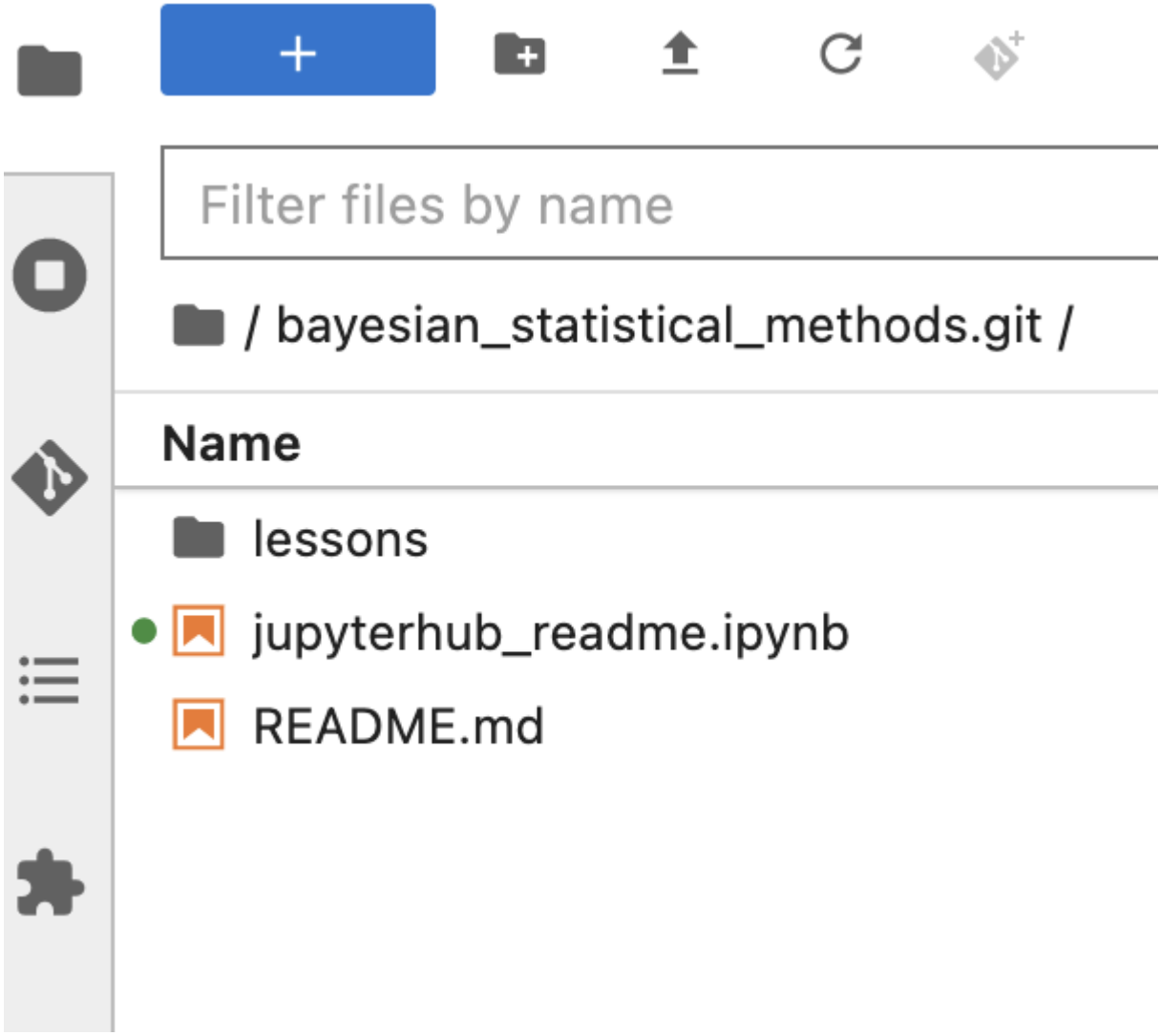
JupyterLab is an environment that centres around editing notebooks in your browser but can do a lot more as well. For example, it provides debugging and version control interfaces, access to a terminal, etc.

ETH provides an instance of JupyterLab for everyone in the course. This is an online development environment and comes with all the necessary packages installed. Changes you make will persist throughout the course. If you set up version control, it is easy to synchronise your on the JupyterLab instance with a local copy on your own device.

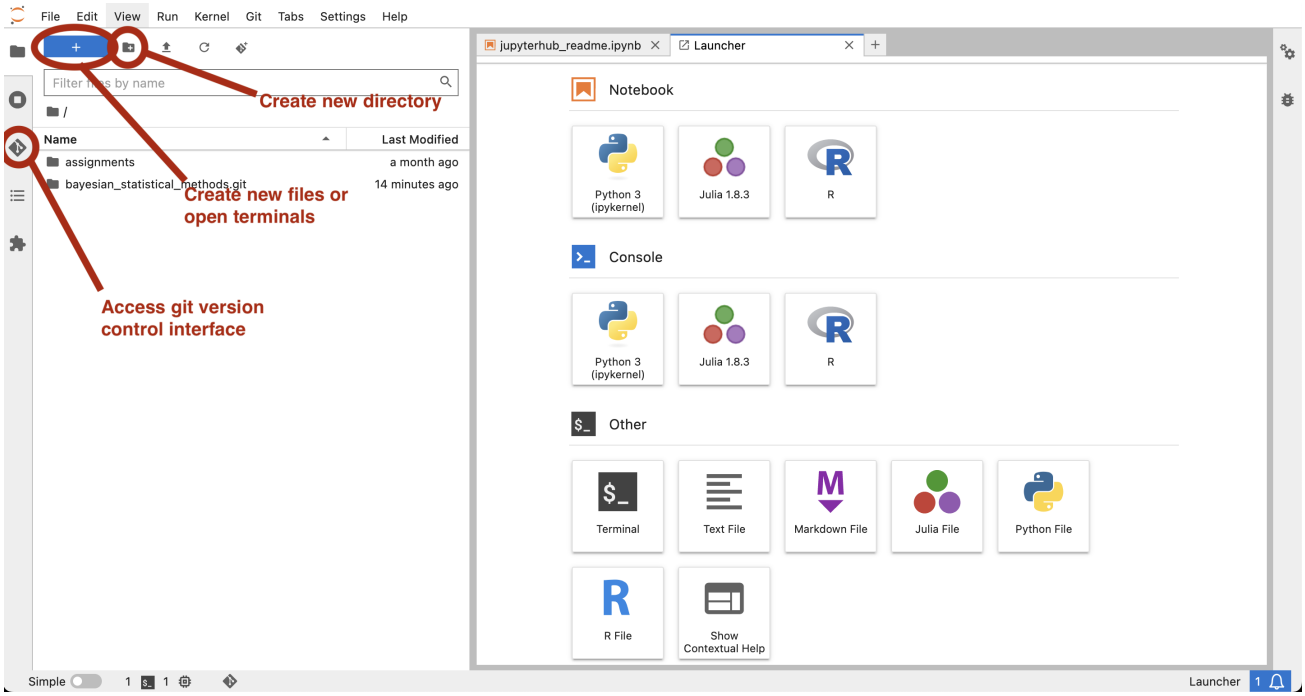
You are free to use your own setup as well.

Using JupyterLab

If you are seeing this in the course JupyterLab, you are probably in the directory with the course materials:



To make your live easier later on with version control, go up one level in the directory structure and make your own directory for the code you are going to write in this course:



On your own device

All the materials can be access outside of JupyterLab as well. The repository with the course materials can be found on github.

To set up your own computing environment I strongly recommend anaconda. Either using miniconda or the much faster mamba implementation.

Useful extensions

Two very useful JupyterLab extensions are enabled on the ETH instance: debugging and version control with git

Debugging

Any code ever written has bugs. Do not trust anyone that claims their code does not contain bugs. That includes yourself!

A common approach to debugging is the sprinkle `print` statements throughout the code and try to figure out when something does not work as it should.

What are some reasons why this approach might be sub optimal?

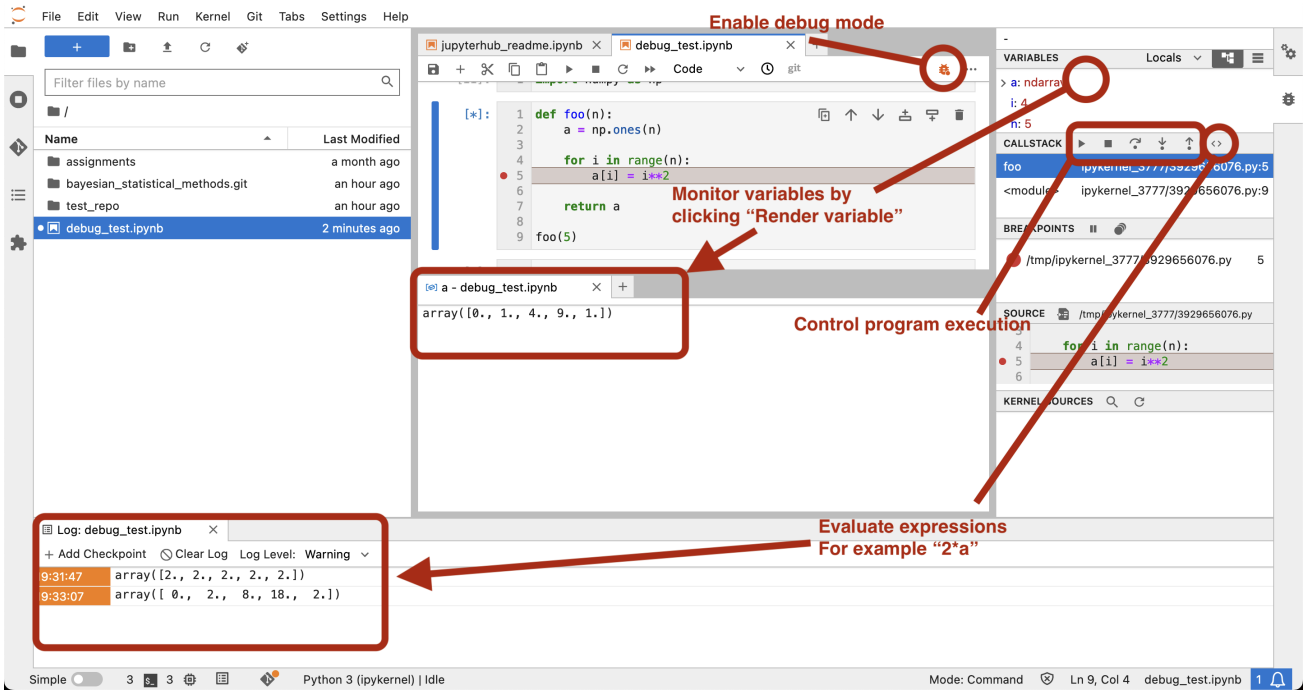
The alternative is using a debugger.

A debugger allows to

- Step through a program line-by-line
- Set breakpoints, so the execution automatically pauses when reaching the breakpoint
- Inspect the values of variables while the program is running and evaluate expressions based on them

One of the biggest downsides to using debugger used to be the set up. But JupyterLab allows debugging of notebook cells, without having to setup `pdb`, `gdb`, etc.

Debuggers built into other tools, such as VS Code, are even more powerful, with features such as conditional breakpoints.



Version control

If you are already familiar with version control and git, great! If not and your version control looks something like

- `my_great_research_project.py`
- `my_great_research_project_ver2.py`
- `my_great_research_project_ver3.py`
- `my_great_research_project_final.py`
- `my_great_research_project_really_final.py`
- `my_great_research_project_really_really_final.py`

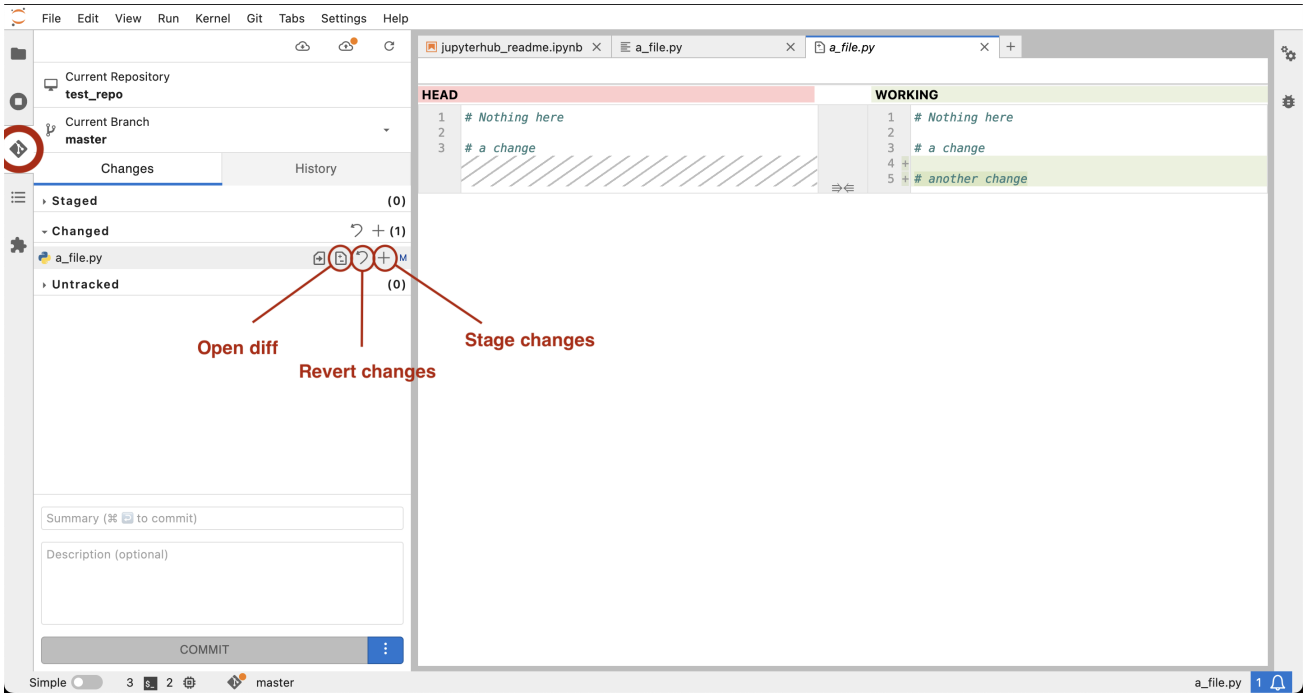
I suggest this introduction. Knowing how to use version control like git is essential if you work on code (or reports) together with other people.

To use git on JupyterLab you can use either the command line interface by opening a terminal or the graphical interface provided by JupyterLab:

- Clone an existing repository:
  1. Create a repository on your provider of choice (github, gitlab, bitbucket, etc)
  2. On JupyterLab, `Git->Clone a Repository`.
- Initialise an empty repository on JupyterLab:
  1. `Git->Initialize a Repository`
  2. If you want to synchronise with a repository outside of JupyterLab, add the remote: `Git->Manage Remote Repositories`

Once the repository is set up on JupyterLab, you can add files by right-clicking on it and selecting `Git->Add` from the context menu.

Opening the git pane on the left allows you to see the files with changes, look at the diffs, stage/unstage files, and commit the changes.



Exercise

1. Create a new directory at the top of the JupyterLab file system
2. Create a new Python notebook

3. Obtain an estimate the value of π
   - From the definition in `numpy`
   - Using the series $\pi = 4 \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$
   - Using a Monte-Carlo estimate:
     A. Sample n points uniformly on the unit square using `numpy.random.uniform`
     B. Count the number n_r of points with distance $r = \sqrt{x^2 + y^2} \leq 1$
     C. Compare the fraction $\frac{n_r}{n}$ to the ratio of the area of a quadrant of a disc to the area of the square to derive an estimate of π

4. Use the debugger to step through some iterations of your code.

5. Create plots showing the estimated value as the number of terms in the series expansion and the number of Monte-Carlo samples is varied

Your python package

The goal of this course is for you to build a toolbox with statistical and computational methods that you can use in your future data analyses.

For that toolbox to be useful it needs to contain reusable code. Notebooks are great for quick development and data exploration but terrible for reusable and maintainable code. As soon as you think about copy & pasting some piece of code from one notebook to another (or within the same notebook), you should instead consider putting that piece of code into your toolbox.

This toolbox can be made into a python package, so that you can import it where ever you need it.

For example, under `course_tools` in the course repository is a small python package with some helper functions, for example for creating the slides.

The package can be installed from the `course_tools` directory by calling `pip install --user --editable .`

I can then `import` the package like any other python package and use it

```
import bayesian_stats_course_tools
bayesian_stats_course_tools.misc.load_tex_defs()

import matplotlib.style
matplotlib.style.use("bayesian_stats_course_tools.light")
```

To create your own package, have a look at the example in `course_tools` and the documentation at https://packaging.python.org/en/latest/tutorials/packaging-projects/.

The key files are

- The `pyproject.toml` file, which specifies that this is a python package
- The `src/bayesian_stats_course_tools/__init__.py`, which makes `bayesian_stats_course_tools` a python module that can be imported.