Setting up a Data Science Project
For more reproducible/robust analysis

Léo Gorman

University of BRISTOL
Jean Golding Institute

This talk is for you if…

- You are using (or are interested in using) programming languages for research
- You want an overview of things like testing, documentation… (not too much depth!)

- You are interested in making your research more:
  - **Open** (other people can see what you're doing)
  - **Reproducible** (other people can do repeat what you've done)
  - **Robust** (people can use/modify the things you've made for more general use)
  - This stuff also makes your work
- **easier/more efficient!!!**

- **It's not for you if you want an in depth demo of testing, documentation….**

Caveats

- **Don't expect to learn everything at once.** It's useful to know that these things exist, and slowly you can start implementing them into your work
- There are many tools you can use; I have tried my best to focus on concepts rather than tooling
- How much you use these approaches is up to debate. Simple analyses may not need a complicated setup!
- Best practice is highly opinionated – Some people may disagree

The issue to start with

- You could have a project that looks like this, some challenges you may face:
  - **Hard to maintain:** Make one change at the start, whole thing breaks. Simple changes seem to require changes to whole code base
  - **Hard to run somewhere else:** Got a new laptop? Running on HPC? Getting a colleague to run it? You try and run it, and your code breaks
  - **Hard to understand**: You come back to your code after a holiday, and forget which file to run first

```
My Project
├── scripts_01_01_2022_for_old_data
├── scripts_10_01_2022_experimental
├── scripts_20_01_2022_final
├── scripts_30_01_2022_these_worked
├── scripts_31_01_2022_experimental
├── cleaning.ipynb
├── cleaning2.ipynb
├── cleaning_final_2.ipynb
├── analysis.ipynb
```

Why bother making your project more reproducible/robust?

- Better science (transparency, accountability...)

Selfish Reasons

- **Saving your future self:** Writing code as if others will use it, will make life easier for future you
- **Speeding up**: Setting up project as a pipeline can mean quickly reproducing results
- **More confidence:** You will get a better understanding of how your code works and where it fails
- **Publication:** Sharing is an increasingly common requirement, making sure that code is legible and can run makes your results more trustworthy
- **Employment opportunities:** If you are seeking a job outside of academia, showing you can write good code will help!

```
📁 My Project
├── 📁 scripts_01_01_2022_for_old_data
├── 📁 scripts_10_01_2022_experimental
├── 📁 scripts_20_01_2022_final
├── 📁 scripts_30_01_2022_these_worked
├── 📁 scripts_31_01_2022_experimental
├── 📄 cleaning.ipynb
├── 📄 cleaning2.ipynb
├── 📄 cleaning_final_2.ipynb
├── 📄 analysis.ipynb
```

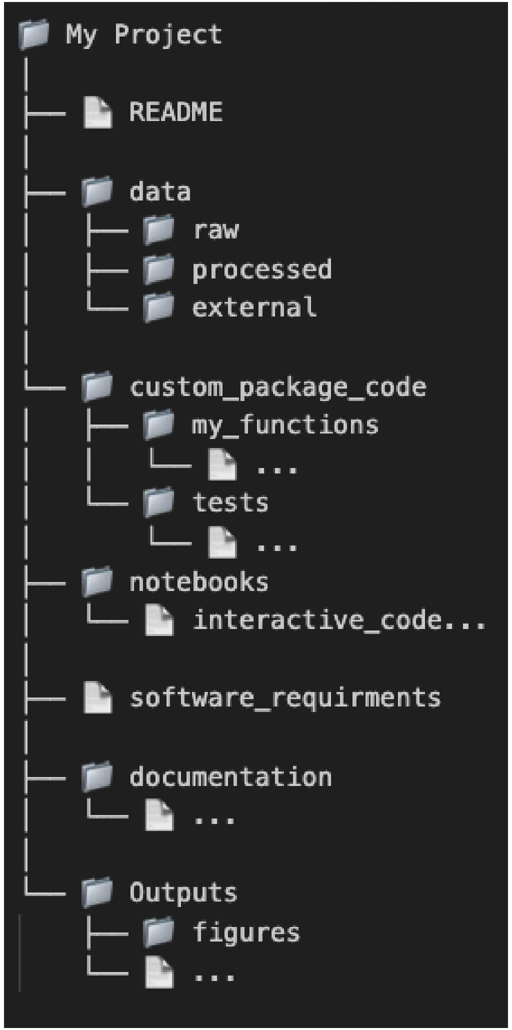
Suggested project layout

- Issues are a combination of
  - Tools/Strategies for managing a project
  - Organisation (project structure conventions)

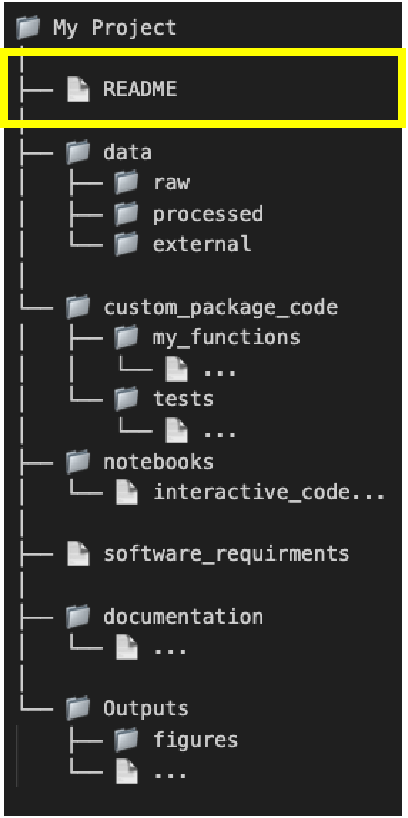
- This is a suggested project layout that I often find helpful for data science projects

- All approaches/strategies is **not always necessary, but often helpful**

- I will go through this folder:
  - explain each concept
  - explain when/how to use it
  - suggest learning materials to follow up



README


My Project
README
data
raw
processed
external
custom_package_code
my_functions
...
tests
...
notebooks
interactive_code...
software_requirments
documentation
...
Outputs
figures
...

README

- A text file, **the first thing you read,** helps:
  - **You** in 6 months time, trying to figure out what is going on in this project
  - **Anybody else** who needs to see what you've done, ore reproduce it.
- Things to include:
  - Summary of the project (including status, e.g. ongoing)
  - A quickstart: what are the most important files. If there is code, in what order should things be run and how?
  - Setup: A guide on what needs to be installed before running code: Describe what the folder contains

Resource:

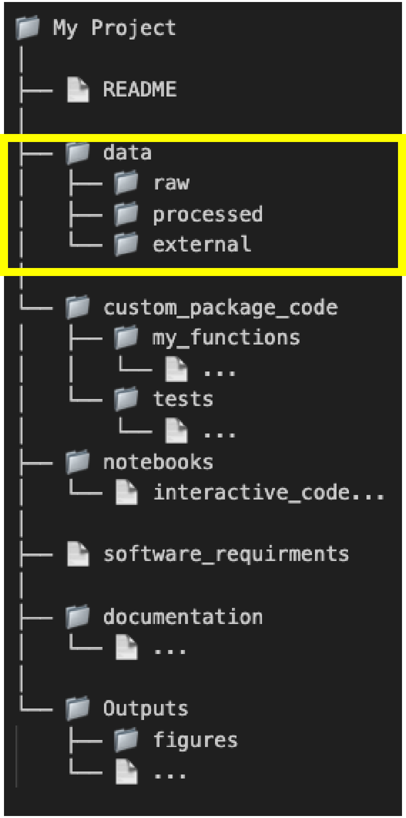
Turing Way README guide

https://book.the-turing-way.org/project-design/pd-design-overview/project-repo/project-repo-readme

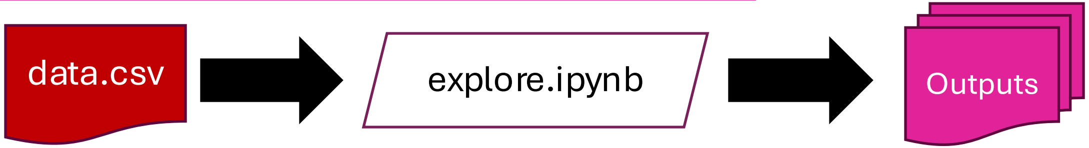
Example Data Science README:

https://github.com/sfbrigade/data-science-wg/blob/master/dswg_project_resources/Project-README-template.md

Data Files


My Project
README
data
raw
processed
external
custom_package_code
my_functions
...
tests
...
notebooks
interactive_code...
software_requirments
documentation
...
Outputs
figures
...

Starting Scenario

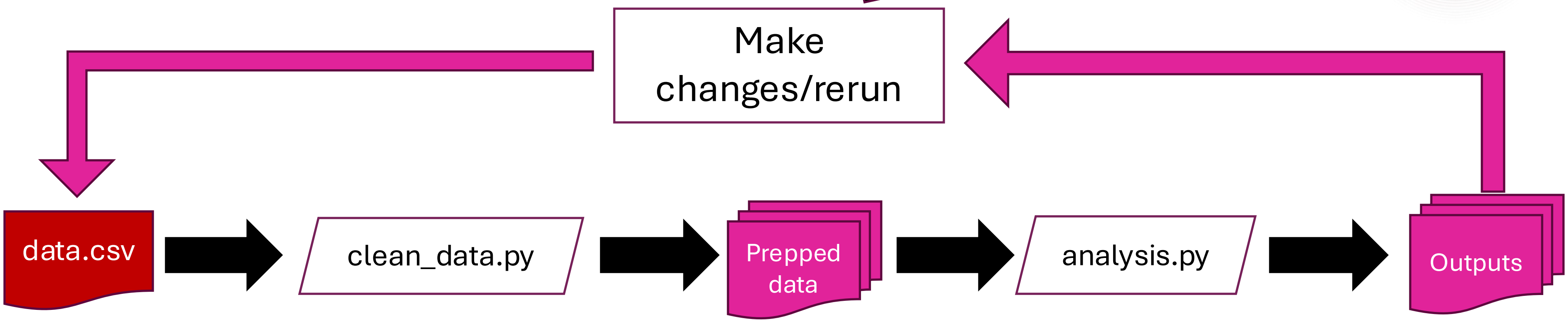


1. **Exploration:** A notebook/excel sheet/script where you explore the dataset
2. **Growth:** Analyses quickly grow in terms of both size/complexity
3. **Pipeline breaks:** Difficult to see how to get from raw data -> results

First Analysis

Second Analysis

Solution 1: Pipeline



Tip: Manage Data

Never overwrite your original data file, and save useful intermediate outputs

Resources:

Turing Way Research Data Management:
https://book.the-turing-way.org/reproducible-research/rdm

Turing Way Project Structure:
https://book.the-turing-way.org/project-design/project-repo/project-repo-advanced#example-for-a-research-project

Code

```
My Project
│
├── README
│
├── data
│   ├── raw
│   ├── processed
│   └── external
│
├── custom_package_code
│   ├── my_functions
│   │   └── ...
│   └── tests
│       └── ...
├── notebooks
│   └── interactive_code...
│
├── software_requirments
│
├── documentation
│   └── ...
│
└── Outputs
    ├── figures
    └── ...
```

Problem: Maintenance

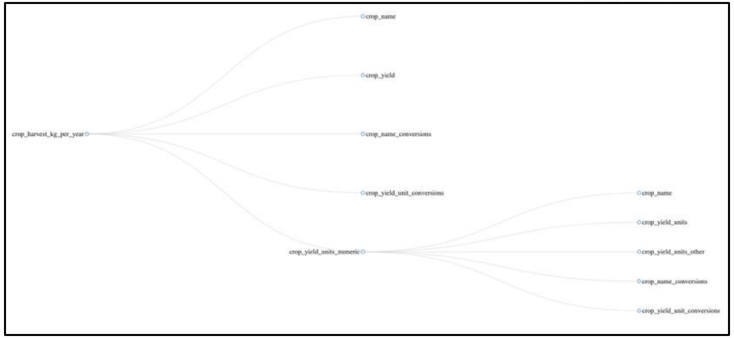
Problem

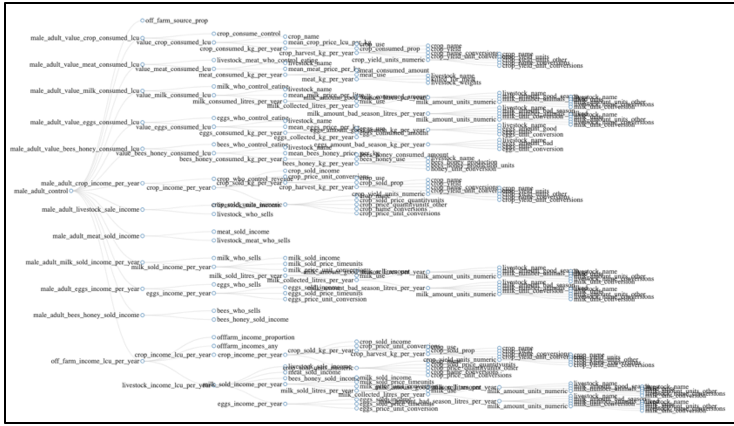
- People start interactive (e.g notebooks/single script), things grow and get **difficult to maintain**

I find this helpful:

1. Start off interactive ("play phase")
2. Breakdown code into smaller functions/modules
3. Test those smaller functions on cases where you know the answer

- Break your process into small chunks that you can check easily
- Combine the small chunks at the end for a more legible final script





Example: Breaking Code Up

Analysis.py

```
# calculate mean
sum = 0
for number in data:
   sum += number
mean = sum/length(data)

# calculate variance
sum_of_squares = 0
for number in data:
   sum_of_squares +=  (number-mean)**2

variance = sum_of_squares/length(data)

...

write_to_file({
   “mean”: mean,
   “variance”: variance
})
```

stats.py

```
def mean(data):
   sum = 0
   for number in data:
      sum += number
   mean = sum/length(data)
   return(mean)

def variance(data):
   sum_of_squares = 0
   for number in data:
   sum_of_squares +=  number-mean)^2
   variance = sum_of_squares/length(data)
   return(variance)
```

Analysis.py

```
from stats import mean, variance

mean = mean(data)
variance = variance(data)


write_to_file({
   “mean”: mean,
   “variance”: variance
})
```

Tip:
Breaking your code into small functions means your main script is easier to read

Example: Testing Chunks

stats.py

```
def mean(data):
   sum = 0
   for number in data:
      sum += number
   mean = sum/length(data)
   return(mean)

def variance(data):
sum_of_squares = 0
for number in data:
   sum_of_squares +=  number-mean)**2
variance = sum_of_squares/length(data)
return(variance)
```

Tip:
Most testing tools allow you to run all your tests in one command

Tip:
Using simulated data is a good way to check your model works!

test_stats.py

```
from stats import mean

def test_mean(data):
   test_data = [1, 2, 3, 4, 5]
   result = mean(test_data)
   expected_result = 3

   assert result == expected_result

...
```

Testing

Key point: Does my code do what I expect/want, for example:

1. **Unit:** Am I correctly linking datasets, lets try with a mini-example?
2. **Integration:** When I make a change to this function, do all my cleaning steps still work together?

When do I test?

Rule of thumb – if I make a change to code, and I am unsure what the consequences are later down the line, I probably should have a test for that!

Resources:

Functions and to testing:

Functions R: https://bristol-training.github.io/intermediate-r/

Functions Python: https://bristol-training.github.io/intermediate-python/

Python testing (Bristol): https://bristol-training.github.io/best-practices-software-engineering/

Python Testing (Turing): https://alan-turing-institute.github.io/rse-course

Testing in R: https://r-pkgs.org/testing-basics.html

Why/When to test:

Why unit test for analysis (R): https://www.r-bloggers.com/2023/04/unit-testing-analytics-code/#google_vignette

Turing Way Testing: https://book.the-turing-way.org/reproducible-research/testing/testing-guidance

Environments

```
My Project
│
├── README
│
├── data
│   ├── raw
│   ├── processed
│   └── external
│
└── custom_package_code
│   ├── my_functions
│   │   └── ...
│   └── tests
│       └── ...
├── notebooks
│   └── interactive_code...
│
├── software_requirments
│
├── documentation
│   └── ...
│
└── Outputs
    ├── figures
    └── ...
```

Environments

- **Working with yourself:**
    1. Start project 1, install packages a, b, c
    2. A few months later, install packages b, c, d for project 2
    3. Go back to rerun code for project 1, it now no longer
- **Avoid:** "But it works on my computer"
- **Environments:** Specify what you need a file, and use tools that make sure you use only what you need when working on that project
- **Disposable**: An environment should be disposable (i.e. you should be able to take the folder full of code, and with limited effort, all the correct packages are installed)

Old Project:

Program: R 4.3.0

Packages:
tidyr 1.2.1
ggplot2 3.3.5
…

New Project:

Program: R 4.3.0

Packages:
tidyr 1.3.1
ggplot2 3.5.0
…

Environments

- **When I use this:** Every project!
- **Tools:**
  - **R:** renv (manage packages), rig (to manage versions of R), conda (manage versions of R and packages), pixi
  - **Python**: venv, uv, conda
- **Resources:**
  - **venv/conda:** https://realpython.com/effective-python-environment/
  - **pixi/uv:** https://jatonline.github.io/managing-dependencies-using-uv-and-pixi/
  - **renv:** https://rstudio.github.io/renv/articles/renv.html

Old Project:

Program: R 4.3.0

Packages:
  tidyr 1.2.1
  ggplot2 3.3.5
…

New Project:

Program: R 4.3.0

Packages:
  tidyr 1.3.1
  ggplot2 3.5.0
…

Documentation

```
My Project
│
├── README
│
├── data
│   ├── raw
│   ├── processed
│   └── external
│
└── custom_package_code
│   ├── my_functions
│   │   └── ...
│   └── tests
│       └── ...
├── notebooks
│   └── interactive_code...
│
├── software_requirments
│
├── documentation
│   └── ...
│
└── Outputs
    ├── figures
    └── ...
```

Documentation

Types of documentation

1. Comments in code
   - In scripts/notebooks
   - Function/doc strings
2. Files in repository
   - README
3. External documentation
   - website

Motivation

1. Comments in code
   - Should explain/justify things not immediately obvious in code
2. Files in repository
   - Describe how to use the code
3. External documentation
   - Discuss what you are doing to a less technical audience.
   - Could include quickstart guides

Documentation

Types of documentation

1. Comments in code
   - In scripts/notebooks
   - Function/doc strings
2. Files in repository
   - README
3. External documentation
   - website

Audience

1. Comments in code
   - other developers/researchers
2. Files in repository
   - other developers/researchers
   - people installing/using tool
3. External documentation
   - people installing/using tool
   - less technical people

Resources

Motivation

1. Comments in code
   - Should explain/justify things not immediately obvious in code
2. Files in repository
   - Describe how to use the
3. External documentation
   - Discuss what you are doing to a less technical audience.
   - Could include quickstart guides

Resources

- Turing Way (what/why): https://book.the-turing-way.org/reproducible-research/code-documentation
- Sphinx (Online documentation, Bristol Training): https://bristol-training.github.io/best-practices-software-engineering/pages/appendix_sphinx.html?q=environment
- Quarto (Online documentation, useful for Python/R): https://quarto.org/docs/websites/

Summary

Getting balance right is hard, just do the best you can

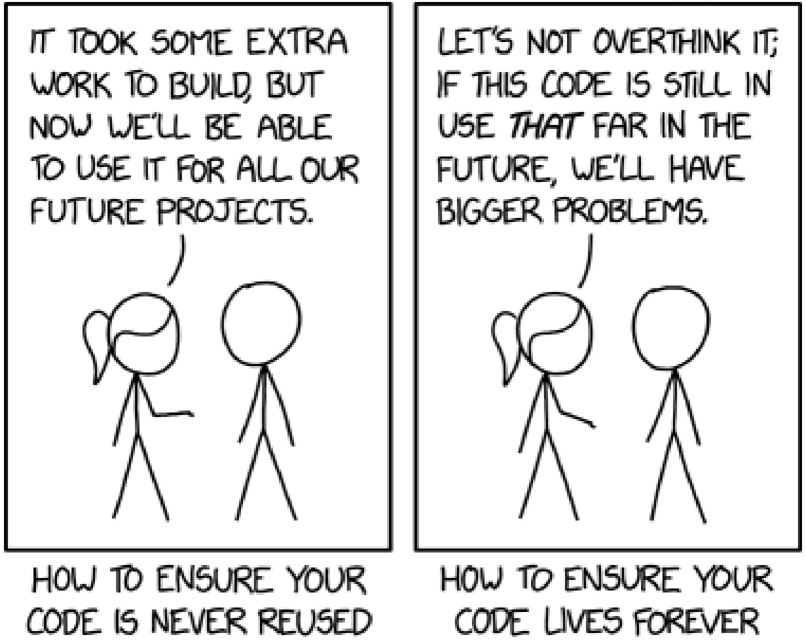
You will only learn by doing, try and implement some of these in your next project

Resources:

[Turing Way]: Impressive community effort, covering these topics and more (in detail)

[Bristol courses]

[Turing/UCL Software Engineering Course (Python)]



Any Questions?

Email: leo.gorman@bristol.ac.uk

Extras! Version Control

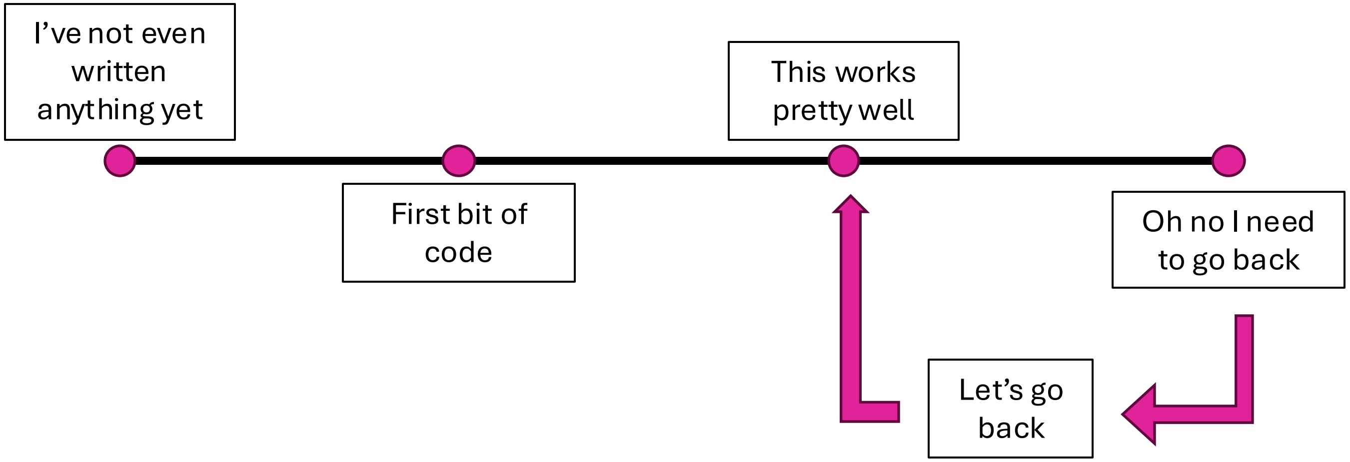
I’ve spent 4 days making changes

1. I need to go back to the last version that worked
2. I’m completely reworking my analysis, but my supervisor is interested in replotting the old results
3. I have a feature in the old version of the scripts that I wish were in this new version

```
📁 My Project
├── 📁 scripts_01_01_2022_for_old_data
├── 📁 scripts_10_01_2022_experimental
├── 📁 scripts_20_01_2022_final
├── 📁 scripts_30_01_2022_these_worked
├── 📁 scripts_31_01_2022_experimental
...
```

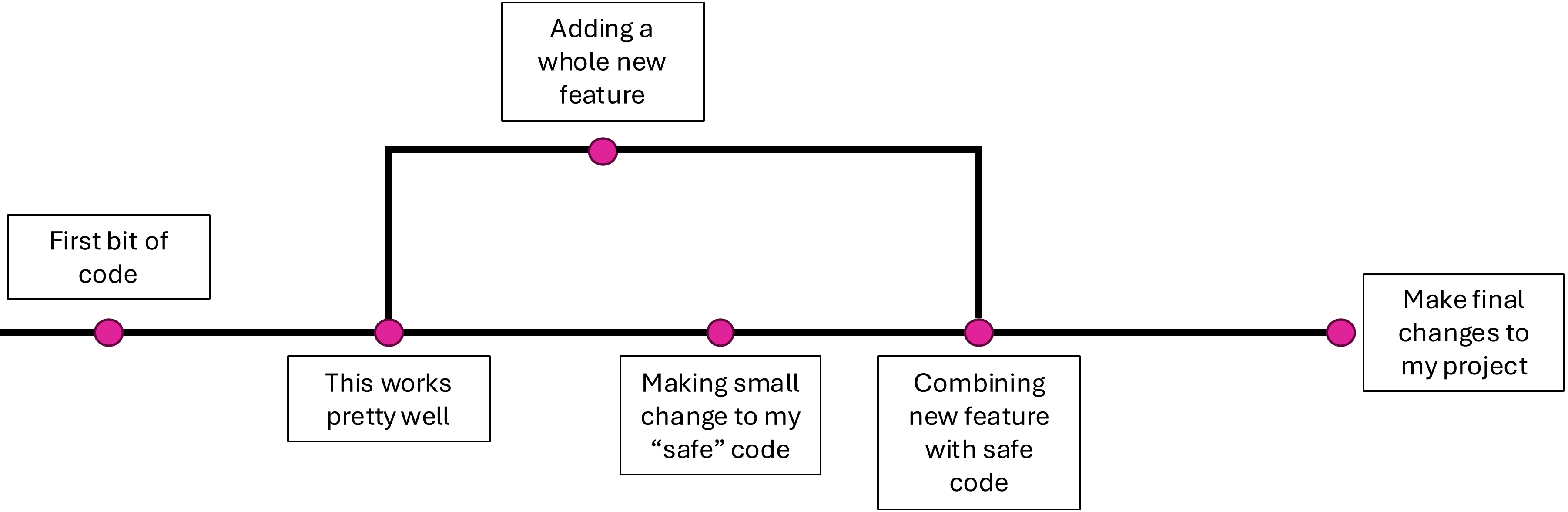
Extras! Version Control

Version control allows you to go back to move between versions easily



Extras! Version Control

- Branches allow you to develop experimental features, whilst having/modifying a stable version of your code



- This is the same approach that allows people to work collaboratively!

Version Control

Key point:

1. Switch between snapshots of a project
2. Develop parallel versions of a project
3. Allows collaborative development

Resources:

Introduction to git (Bristol Course):
https://chryswoods.com/introducing_git/

Turing RSE Course:

https://alan-turing-institute.github.io/rse-course/html/index.html

When do I use version control?

Every project!