# Programming Practices for Research in Economics

## Introduction & Motivation

Ulrich Bergmann  Matteo Courthoud  Lachlan Deer

Department of Economics, Univeristy of Zurich

Winter 2020

# Welcome!

# Introductions: Who We Are

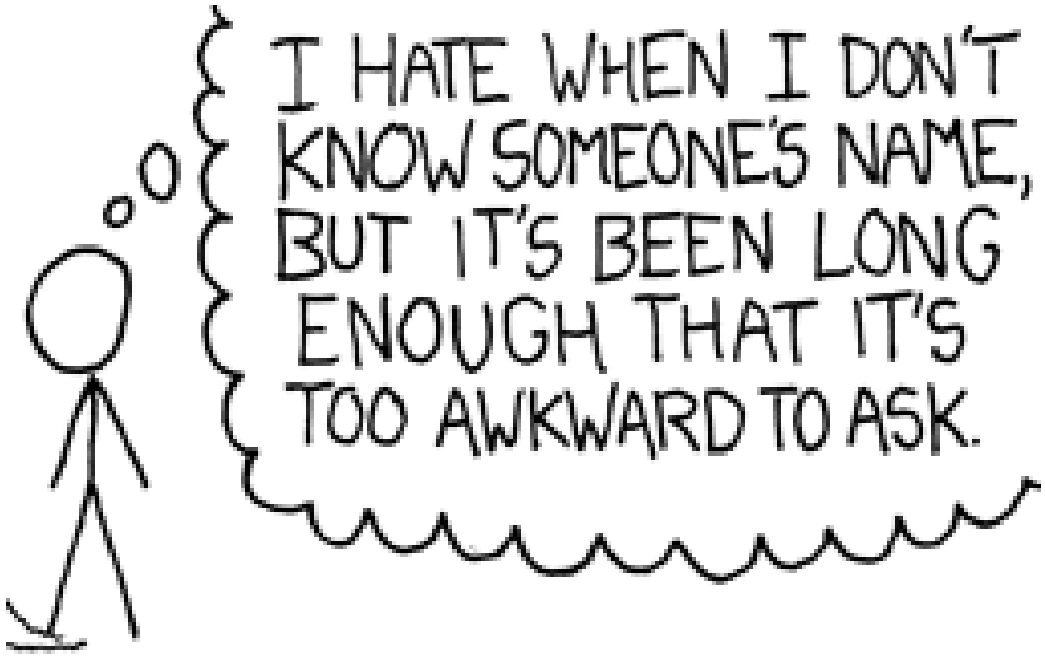
2 PhD students

- Matteo
- Uli

1 Post Doc:

- Lachlan

# Introductions: Who You Are



# Logistics: Audit / Credit Students

audit students:

- enjoy your time
- take skills home

for credit students, also need to

- enrol using sheet we will pass around in the last week
- register for course on `UZH module booking`
- submit an assignment

# Logistics: Classes

sessions are designed to be interactive

- mix of *live coding* & *exercises*
- we want to get you comfortable using your computing environment to solve problems
  - bring your laptop!
  - we expect you have completed the installation guide and have all software installed.
  - ask questions!

# Logistics: Structure of each day

- session 1: 9.30 - 12.30
- session 2: 14.00 - 17.00
- expect coffee breaks in each session
  - exactly when depends on the instructor, and the material
- talk to us during the day
  - no scheduled office hours
  - email for appointment after class if want to discuss assignment

# Logistics: Where to Find Information

- Course website:
  - `pp4rs.github.io/2020-uzh`
- Installation Guide:
  - `pp4rs.github.io/installation-guide`
- Course Chatter:
  - `pp4rs.slack.com/`, #general-2020
- GitHub repositories:
  - `github.com/pp4rs`
- Terminal data for today:
  - `https://bit.ly/38FCQ9R`

# Logistics: Assignment

The basics

- One final assignment
- Can be submitted in groups of 1-3 people
- Due 4 weeks *after* last class
- Propose to us an idea before you start

Use what you learn in this course to solve a non-trivial economic problem

- Code must be in split into meaningful sub-files
- Solution must be submitted using GitHub
- Solution must be executable using a single line of code via `Snakemake`

# Logistics: Social Event

- Join us for casual drinks
- When: This Friday (January, 31st), after class
- Location: TBA

Section 1

# Motivation

***We will cover things that we wish someone had taught us when we were starting out in graduate school***

# Why this course exists

We teach data science skills which fill gaps left by traditional econometrics and methods classes

- general skills
  - how to write clean code
  - how to organize & track evolution of projects
  - how to work on projects with others
  - how to make workflows reproducible
- practical skills
  - data cleaning
  - data wrangling
  - data visualization
  - simulate data
  - solving economic models computationally

# Why? – Practical reasons

Broad Goals for the Course

1. Improve computing skills, so you can do things you could not do before
2. Show how to do things you know with less effort
3. Increase the confidence in results that are produced this way (both your and others' results)

# Why? – Academia Research and Open Science

EU policy requires all publicly funded research to be *open* by 2020

- 4Rs (Pagan and Torgler, Nature 2015)
  - Reproduction: Can others reproduce your results using the same data?
  - Replication: Can others replicate your results using new data?
  - Robustness: Do your results depent on the assumptions you made?
  - Revelation: Do you communicate the reasoning for your conclusions transparently?

# Why? – Software Used in the Industry

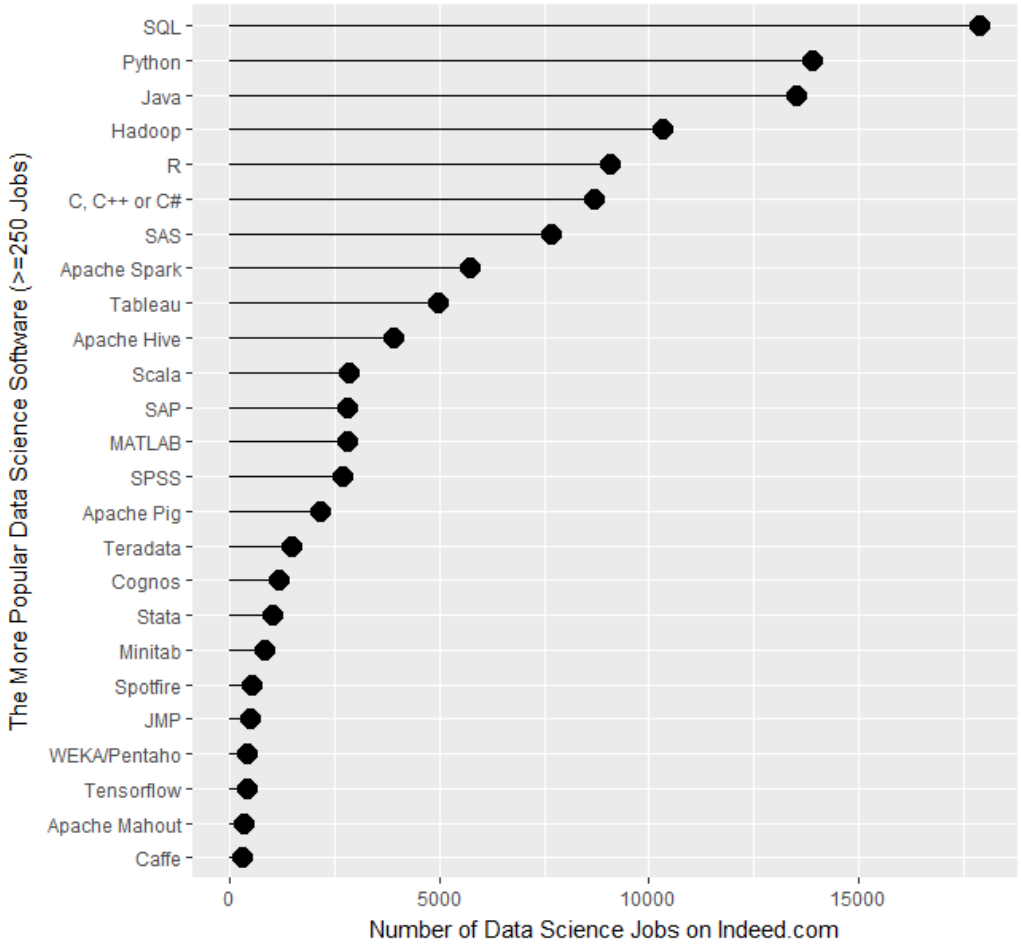



**Figure 1:** Required software for data-science jobs

# Why? – Economics PhDs aren't Prepared for Industry Jobs

Econ PhD graduates report that

- R or Python required in their first job
- Need to cooperate on coding tasks with others

Tech companies who hire PhD economists report that:

- Econ PhDs are expensive to train because they don't have the right software skills
- Most don't `catch up`
- This impacts tech companies willingness to hire us
- (Summary of a discussion by senior economists empolyed by tech companies at an academic conference)

# What We Teach

Core topics:

1. Terminal: The Unix shell
   - Text based interface to computing
   - Automate repetitive tasks
2. Git
   - Track, control, and share work
3. Snakemake
   - Automate the execution of your research project
4. Python/R
   - build modular code to solve typical economics problems

# We Cannot Cover Everything

We miss (important) topics such as:

- Databases: SQL, mySQL, SQLite etc.
- Unit testing
- Complete documentation of Research Projects
- High Performance / distributed Computing
- big data X, Y, Z

Section 2

# Guiding Principles

# ToDo

- First short overview
- One slide per point

# Rules to Code By

1. Write Programs for people, not computers
2. Define things once and only once
3. Use a version control system
4. Optimize software only after it works correctly

# Rule 1: Write Programs for People Not Computers

Make your code easy to understand for humans. If your code looks very complex or messy, you're probably doing it wrong.

Organization:

- Define `functions` that do one mayor step each
- Many short `scripts` that do one task each
- document what your code **doesn't** say

Style:

- Use meaningful variable names (price > p, i_subject > i)
- Use consistent code and formatting styles
  - `camelCase`, `snake_case`, `kebab-case`
- indent
  - your
    - code

# 2. Define things once and only once

Let Computers repeat and execute tasks

- Rule of 3:
  - if you copy-paste code 3 times, write a function instead
- If you do things often, automate them
  - use scripts, macros, aliases
  - write a dictionary with definitions
- Use build tools to automate workflows
  - Makes it easy and consistent to execute tasks

# 3. Use a Version Control System

- Add all inputs but no outputs
  - **DO:** Everything created by humans, data inputs
  - **DON'T:** things created by the computer from your inputs. Those will be reproduced via a workflow
- Work in small changes
  - create snapshots in small and logical steps
  - allows to go each point in time if necessary and to understand progression
- Use an issue tracking tool to document problems
  - **Email is not an issue tracker**

# 4. Optimize software only after it works correctly

Even experts find it hard to predict performance bottlenecks

- get it right then make it fast
- small changes can have dramatic impact on performance
- use a profiler to report how much time is spent on each line of code

# Rules to Code By

1. Write Programs for people, not computers
2. Define things once and only once
3. Use a version control system
4. Optimize software only after it works correctly # A Warning

# A Warning

15 days $\times$ 6 hours/day $=$ 90 hours of content

- that's a lot! ... and fast
- You (and we) **will be tired** at various points

Nobody can transform their practices overmight ...

- but persistance will make your programming life much, much more efficient
- think of us as a 'kick in the arse' to get you started

# Let's Get Started!

# Acknowledgements

This module is based on the 2016 and 2017 versions of the course:

- `Programming Practices For Economists`, by Lachlan Deer, Adrian Etter, Julian Langer & Max Winkler

It is designed after and borrows a lot from:

- `Effective Programming Practices for Economists`, a course by Hans-Martin von Gaudecker
- Software Carpentry's `Managing Software Research Projects` lesson

`Guiding Principles` borrows a lot from the paper

- Wilson G, Aruliah DA, Brown CT, Chue Hong NP, Davis M, Guy RT, et al. (2014) `Best Practices for Scientific Computing`. PLoS Biol 12(1): e1001745.

# License



Suggested Citation:

- Ulrich Bergmann, Matteo Courthoud, Lachlan Deer, 2020, `Introduction and Motivation`, Programming Practices for Research in Economics, University of Zurich