Week 1: Introduction

LSE MY472: Data for Data Scientists
https://lse-my472.github.io/

Autumn Term 2024

Ryan Hübert

Slides last updated: 30 September 2024

What is this course about?

80/20 rule of data science:
80% data manipulation, 20% data analysis

MY472 is about the 80%

Course outline

1. Introduction
2. Tabular data
3. Data visualisation
4. Textual data
5. HTML, CSS, and scraping static pages
6. Reading week
7. XML, RSS, and scraping non-static pages
8. Working with APIs
9. Other data types
10. Creating and managing databases
11. Interacting with online databases

Plan for today

- Administration and logistics
- A little about me
- R and RStudio
- Git/Github for version control

Administration and logistics

Prerequisites and software

- Introductory course – no prerequisites (only completion of R preparatory course required!)
- You should bringing your own laptop to lectures and to seminars
- Required software:
  - R – Install from https://www.r-project.org/
  - RStudio – Install from https://www.rstudio.com/products/rstudio/download/
  - Git – Install the GitHub Desktop application from https://desktop.github.com/
  - *Please install before your seminar session tomorrow*
- Mirrors similar tool usage and learning in other Methodology courses

Course philosophy

Who is this course for?

- Two audiences: researchers and "industry"
- Practically speaking, the course was designed by researchers... this is a feature, not a bug!
- You have a wide range of skillsets, that's great!

Course philosophy

How to learn the techniques in this course?

- ➜ Lecture approach: not ideal for learning how to code
- ➜ You can only **learn by doing**
- ➜ We will cover concepts three times
  1. Introduction to the topic in lecture
  2. Guided coding session in lecture and seminar
  3. Course assignments
- ➜ We will **move relatively fast**

Materials and resources

Course website: https://lse-my472.github.io/

- Mixed set of readings, very specific to each week
  - Often freely available online, otherwise, available for purchase (often in electronic versions)
  - Some books are (freely) available online and in print, and the online version may be more recent

Teaching team

- Ryan Hübert, Associate Professor (Methodology), course convenor/lecturer and your primary point of contact
- Daniel de Kadt, Assistant Professor (Methodology)
- Charlotte Kuberka, PhD Student (Government)

Office hour slots are booked in StudentHub.

Course meetings

- Weekly lectures: Wednesdays 13:00–15:00 (CLM.2.02)
- Ten one-hour seminars (“labs”) *starting this week*
  - Group 1: Thursdays 13:00–14:00 (CLM.2.05)
  - Group 2: Thursdays 17:00–18:00 (CBG.2.05)
  - Group 3: Thursdays 14:00–15:00 (CLM.2.05)
- No lecture/seminar in Week 6
- Office hours (book via StudentHub)

Assessment

- Formative in-class exercises
  - Will be completed in weekly seminars with solutions provided
- Formative practice problem set
  - Opportunity to practise format and style of response for the summative assessments
  - Due Friday, 1st November at 5pm
- Summative mid-term problem set (50% of final mark)
  - Due Friday, 22nd November at 5pm
- Summative final take-home assessment (50% of final mark)
  - Due Wednesday, 15th January at 5pm

Submission logistics will be announced when the practice problem set is assigned

A note on collaboration

- All assignments are individual unless we instruct you otherwise
- You may discuss with others, but your submission (including code) should be your own
- You can use online resources but always give credit in comments if you borrow code/solutions
- Any uncited code/solutions/papers/resources, or shared code, are considered plagiarism

ChatGPT (and other generative assistants)

You are allowed to use ChatGPT for your assignments

- ➜ Ignoring the presence/possibilities of ChatGPT is unwise
- ➜ An opportunity to learn how to integrate these tools into your workflow

But beware:

- ➜ We will assess your ability to "deploy" these tools
- ➜ You need proficiency to recognise good code and fix broken code (useful analogy: learning to speak a non-native language)
- ➜ ChatGPT routinely produces bad code, incorrect information
- ➜ You are sharing your thoughts, ideas, and work with models that are proprietary (yikes!)
- ➜ You will need to do more than simple coding exercises; you're learning to be a data *scientist*

About me

Who am I?

- Associate Professor at the London School of Economics
  - PhD in Political Science and MA in Economics from the University of California, Berkeley
  - Assistant Professor of Political Science at the University of California, Davis (2016–2024)
- Research uses game theory and computational/quantitative methods to study U.S. political institutions, especially U.S. federal courts (more: https://ryanhubert.github.io/)
- Email me at r.hubert@lse.ac.uk
- Schedule office hours at https://studenthub.lse.ac.uk/
- I prefer that you call me Ryan

Let me tell you a story about my journey to data wrangling. . .

R and RStudio

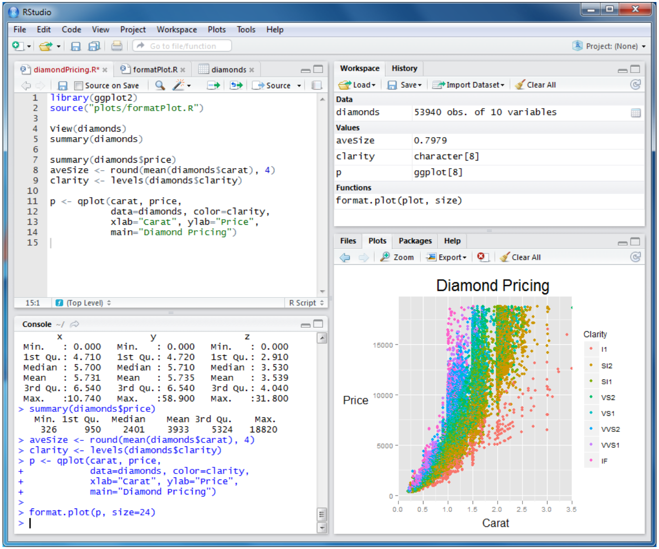
Why use R?

- It's free and open-source
- Quite accessible even to novice coders
- Frequently used in academia and the private sector
- Flexible and extensible through many *packages*
- Excellent online documentation and troubleshooting resources
- A fully-fledged programming language, making it easier to transition to/from other languages

What about python?

- Python is a “similar” coding language popular in industry and gaining some traction in academia – also free and open-source
- It has some advantages and disadvantages relative to R
- If you want to do data science, you’ll probably (eventually) become bilingual
- I may occasionally show you similarities/differences (optional)
- TLDR: if you learn one, that’s good enough and you can learn the other quickly

RStudio



```r
library(ggplot2)
source("plots/formatPlot.R")

View(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")
```

```
       x                y                z
 Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
 Median : 5.700   Median : 5.710   Median : 3.530
 Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
 3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
 Max.   :10.740   Max.   :58.900   Max.   :31.800
> summary(diamonds$price)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(p, size=24)
> 
```

Data	
diamonds	53940 obs. of 10 variables
Values	
aveSize	0.7979
clarity	character[8]
p	ggplot[8]
Functions	
format.plot(plot, size)	

Installing R and RStudio

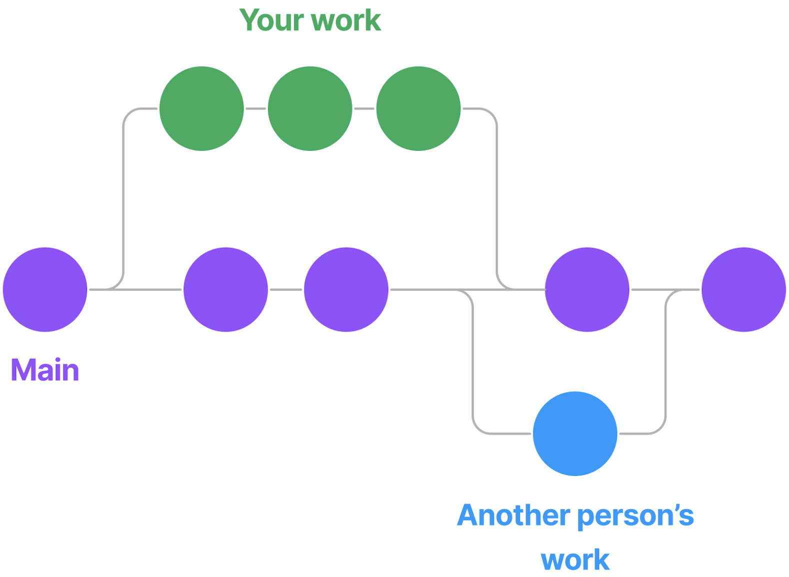
- Please install R and RStudio on your laptop and bring it to lectures and seminars
- Software:
  - R – Install from https://www.r-project.org/
  - RStudio – Install from https://posit.co/download/rstudio-desktop/
- *Try to install both before seminar this week. If there are any issues with installation, we can discuss them in seminar*

Git/GitHub

Version control

- A version control system (VCS) is key when working on code, particularly when collaborating
- It keeps records of changes in files - *who* made *which* changes *when*
- Possibility of reverting changes and going back to previous states
- When a VCS keeps the entire code and history on each collaborator's machine, it is called distributed

The main idea(s)



Git/GitHub

- **Git**: A very popular distributed version control system
- Created by Linus Torvalds in 2005 to facilitate Linux kernel development
- Other options e.g. Mercurial, Subversion
- **GitHub**: Service to host collections of code online with many extra functionalities (UI, documentation, issues, user profiles. . .)

Terminology

- *Repository/repo*: A collection of code and other files
- *Clone*: Download a repo to a computer
- *Commit*: Create a snapshot of (code) files and describe how they have changed
- *Push*: Update changes made locally on a computer also in the remote repository
- *Pull*: Obtain changes made by others which are stored in the remote repository

Installing Git

- Mac:
  - Type `git` into your Terminal and hit enter
  - In most cases, this will automatically install `git` if it's not already installed
  - If it doesn't work, go to https://git-scm.com/download/mac (use Homebrew)
- Windows:
  - Downlod from https://git-scm.com/download/win
- Register a GitHub account at https://github.com/
  - You can apply for student benefits via https://education.github.com/benefits?type=student
- If you install the GitHub Desktop app, then it should install `git` automatically

Creating a repository

- First, log on to https://github.com/ with your account
- Click on your alias in the upper right hand corner $->$ Your repositories $->$ New
- Select a name, e.g. ‘firstrepo’
- Select private to make it visible only to you and accounts you can select
- For the .gitignore choose the R pre-set
- Add an empty README
- Click on ‘Create repository’
- The repo now exists on GitHub

Configuring Git user and email

- ➜ Next, you will once need to configure Git on your computer and link it to GitHub
- ➜ Open Mac Terminal or Windows Git Bash
- ➜ Set your username in Git by pasting in Terminal/Git Bash: `git config --global user.name "Your Name"` (replace with your name before hitting enter)
- ➜ Set your commit email in Git: `git config --global user.email your@email.com`
- ➜ Then navigate to the folder where you would like to locate the repository on your computer with cd (change directory)

Cloning a respository

- The next step is to copy (clone) the online repository to your computer
- On your repository page on GitHub, click on Code and copy the URL (https)
- In the command line, enter `git clone ...` and replace ... with the copied url
- You will now be asked to enter your user name and password, for this we will have to create an access token as the last step in this setup (**note: some users are instead asked at this point to enter their password via a pop-up window - in this case, no access token has to be created manually and you can skip the next slide**)

GitHub authentication

- On GitHub, click on the alias in the upper right hand corner $->$ Settings $->$ Developer settings $->$ Personal access tokens $->$ Generate new token
- Pick a name, e.g. “command line”, choose an expiration, select “repo” (this will allow to access private and public repos from the command line), and click Generate token
- Copy the token (it will only be visible once)
- Now go back to the command line, enter your GitHub user name and as password paste the token
- That’s it, the setup of Git & GitHub is done and the repository was copied as well (no need to repeat the authentication until the token expires)

Creating a file

- We will now create a new file in the repository and log these changes
- With RStudio or a text editor (e.g. download VS Code at https://code.visualstudio.com/), add a file somecode.R into the repo folder
- At the command line, change into the repo folder with `cd firstrepo` (change directory)
- Now you are ready to commit the changes that were made to the repo

Committing changes

- First check whether anything changed with `git status` (make sure you are in the repo folder on your computer)
- Next add all untracked changes to the so-called staging area with `git add .` (we can also add only specific files)
  - WARNING: be very careful using the dot to add files!
- Commit/log changes with `git commit -m "added a code sample"`
- That's it!
- To study this again, add another line of code to the file and repeat the above steps
- Run `git log` to see the history of commits

Pushing changes to the remote repository

- To store these changes also in the remote repository, run `git push` afterwards
- It is now possible to review the changes in the browser which is very helpful for large code files
  - First, go to the repository page on GitHub and click on the clock symbol next to 'commits' in the upper right hand corner
  - Click on the key describing a specific commit, which could e.g. look something like '472cb9d', then you will see which lines of code changed
- If someone else has changed the online repository, run `git pull` to obtain the newest files

Review of key commands

- `git clone ...`: Download online repository to local computer
- `git status`: See status of files in repository
- `git add .`: Stage all changes made (alternatively add distinct file names to be staged)
- `git commit -m "some message "`: Commit (i.e. record) staged changes
- `git push`: Upload local changes to remote repository
- `git pull`: If files changed online, update local repository first

Some further concepts

- *Fork*: Own copy of a repository (pushed changes to this copy do not affect the original remote repository - different from git clone)
- *Branch*: A parallel version of the code originating from a duplication at one point
- *Merge*: Combine branches
- *Pull request*: GitHub based request to merge a branch or a fork into other code
- We will discuss these in the seminar

Extensions for Git/GitHub

- People often use a combination of Git via the command line and the user interface of the GitHub website
- There is also a graphical user interface from GitHub to replace the command line (GitHub Desktop), or Git can be used directly through RStudio as an R-specific alternative to using the more general command line
- For detailed online manuals and books that discuss Git, see e.g. https://git-scm.com/book/en/v2
- To review GitHub, see e.g. https://docs.github.com/e

Useful command line prompts for Mac/Linux

- `pwd` – “Print working directory”
- `cd` – “Change directory” using relative filepaths
  - `cd ..` goes back one folder level
- `ls` – “lists” all folders and files in the current directory
- Other helpful commands can be `mkdir`, `rmdir`, `rm`, and `touch`

Before we get to coding

Homework before seminar:

1. Install R and RStudio
2. Create a GitHub account and register for student benefits (if you want)
3. Install the GitHub Desktop app
4. Check that `git` is installed (see previous instructions)
5. Make a `firstrepo` repository using the instructions above

If you have any issues, we can troubleshoot in seminar.

Coding

Coding

Let's review some R code:

- `01-rmarkdown.Rmd`
- `02-vector-lists-dfs.Rmd`