MY472 Data for Data Scientists
Week 1: Introduction

Friedrich Geiecke

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

27 September 2022

What is this course about?

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

This course is much more about the 80% than the 20%

Course outline

1. Introduction
2. The shape of data
3. Data visualisation
4. Textual data
5. HTML, CSS, and scraping static websites
6. (Reading week)
7. XML, RSS, and scraping non-static website
8. Working with APIs
9. Creating and managing databases
10. Interacting with online databases
11. Cloud computing

Plan for today

- Administration and logistics
- R and RStudio
- Git/GitHub
- Coding

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 each concept three times

1. Introduction to the topic in lecture
2. Lab
3. Course assignments

- No prerequisites for this course, however, we will move relatively fast. Make sure to finish the pre-course soon

Materials

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

Course meetings

- Lecture: Tuesdays 9-11am in MAR.2.08
- Ten one-hour seminars (also called "labs" or "classes") taught by Yuhao and me
  - Group 1: Thursdays 1-2pm in NAB.2.08
  - Group 2: Thursdays 5-6pm in NAB.2.16
  - Group 3: Tba
- No lecture/class in week 6 (Reading Week)
- Office hours: Tuesdays 5-7pm (book via StudentHub)

Assessment

- Four term-time assignments (50%).
  - Submitted via GitHub (more in lab)
  - Only "knitted" R-markdown assignments in HTML accepted
- Final assignment (50%)
  - More extensive and open-ended than term-time assignments

A note on plagiarism and collaboration

- Three individual term-time assignments, one group term-time assignment, one individual final assignment
- **Strictly no discussion and collaboration with others allowed in any individual assignment**
- You can use online resources but always give credit and cite if you borrow code or solutions
- Any forbidden discussion/collaboration or not cited code/solutions/papers/resources are considered plagiarism

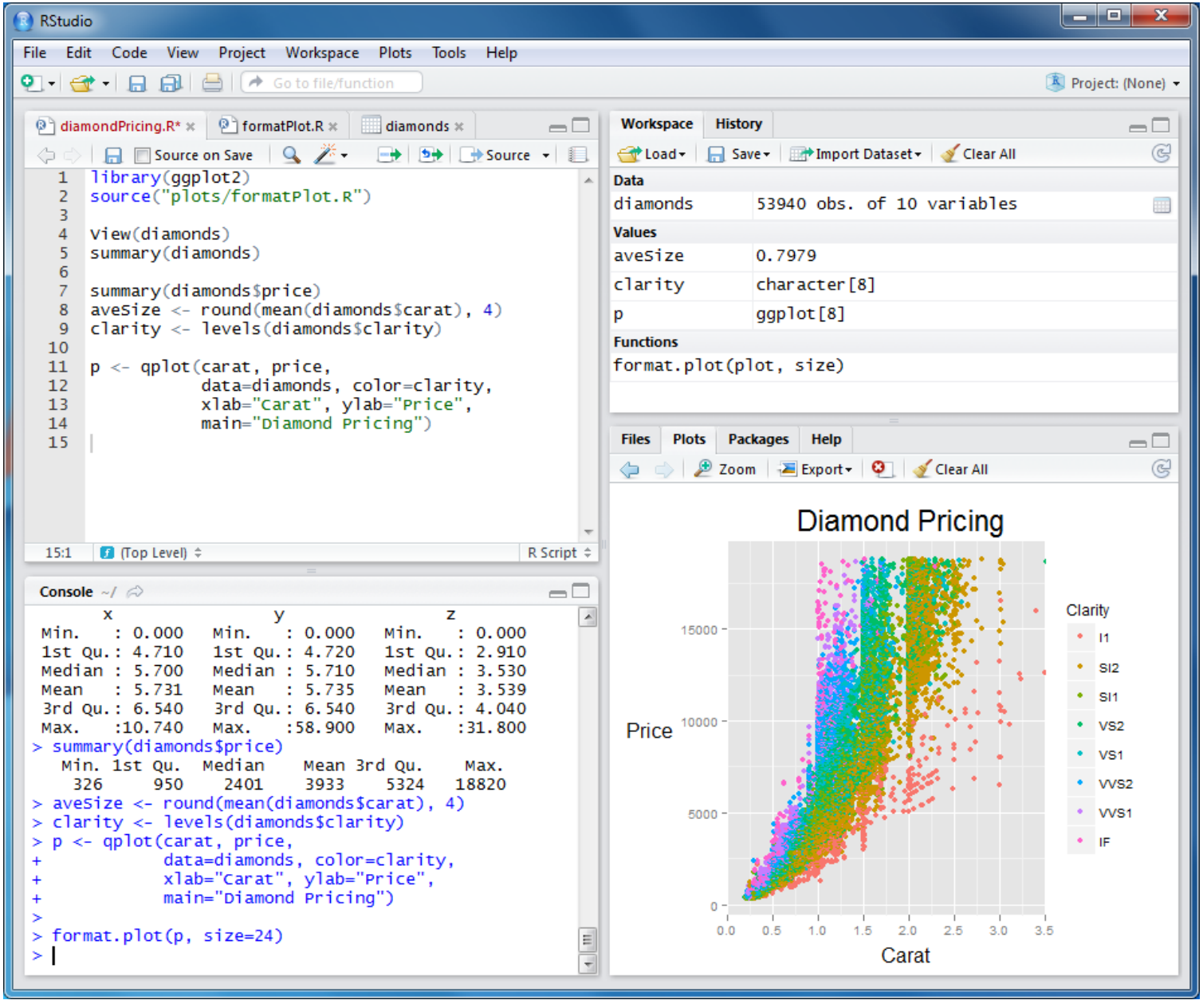
Plan for today

- Administration and logistics
- R and RStudio
- Git/GitHub
- Coding

Why use R?

- It is free and open-source
- Quite accessible also for those students who have never programmed before
- Frequently used in academia and the private sector
- Flexible and extensible through many *packages*
- Excellent documentation and online help resources
- R is also a full programming language; makes it easier to learn other languages, too

RStudio


RStudio
File Edit Code View Project Workspace Plots Tools Help
Go to file/function
Project: (None)
diamondPricing.R*
formatPlot.R
diamonds
Source on Save
Source
1 library(ggplot2)
2 source("plots/formatPlot.R")
3
4 View(diamonds)
5 summary(diamonds)
6
7 summary(diamonds$price)
8 aveSize <- round(mean(diamonds$carat), 4)
9 clarity <- levels(diamonds$clarity)
10
11 p <- qplot(carat, price,
12 data=diamonds, color=clarity,
13 xlab="Carat", ylab="Price",
14 main="Diamond Pricing")
15
15:1 (Top Level) R Script
Console ~/
x y z
Min. : 0.000 Min. : 0.000 Min. : 0.000
1st Qu.: 4.710 1st Qu.: 4.720 1st Qu.: 2.910
Median : 5.700 Median : 5.710 Median : 3.530
Mean : 5.731 Mean : 5.735 Mean : 3.539
3rd Qu.: 6.540 3rd Qu.: 6.540 3rd Qu.: 4.040
Max. :10.740 Max. :58.900 Max. :31.800
> summary(diamonds$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)
>
Workspace History
Load Save Import Dataset Clear All
Data
diamonds 53940 obs. of 10 variables
Values
aveSize 0.7979
clarity character[8]
p ggplot[8]
Functions
format.plot(plot, size)
Files Plots Packages Help
Zoom Export Clear All
Diamond Pricing
Price
Carat
0
5000
10000
15000
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5
Clarity
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF

Installing R and RStudio

- Please install R and RStudio on your laptop and bring it to lectures and labs
- Software:
  - R – Install from `https://www.r-project.org/`
  - RStudio – Install from `https://www.rstudio.com/products/rstudio/download/`

→ *Try to install both before the lab this week. If there are any issues with the installation, we can discuss them in the lab*

Plan for today

- Administration and logistics
- R and RStudio
- Git/GitHub
- Coding

Version control systems

- 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*

Main ideas

- Have code files stored in a folder on own computer
- Record changes in this code with time stamps
- Revert back to earlier code if e.g. something broke
- Store the code from local computer also online such that others can see changes and time stamps
- Create separate versions to try out new ideas without impacting the main code
- Discuss with others whether these modifications should also be included into the main code

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...)

Some 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

Git/GitHub example

- I will now go through a Git/GitHub example
- Please try to repeat these steps yourself after the lecture. Any questions? We can discuss these in the lab this week
- First, install Git:
- Mac: Type `git` into your Terminal and hit enter. Only if not installed already, `https://git-scm.com/download/mac` (use/first install Homebrew)
- Windows: `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`
- **Please let us know your GitHub account name via the form on the MY472 Moodle page**

Creating a repository on GitHub

- First, log on to `https://github.com/` with your account
- Click on the alias in the upper right hand corner -> *Your repositories* -> *New*
- Select a name, e.g. 'firstrepository'
- 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*
- The repo now exists on GitHub

Configuring Git user name 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)
- For some further explanation of command line syntax such as "cd", see the Appendix of these slides

Cloning a repository

- 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)

Git and GitHub example: Creating file

- We will now create a new file in the repository and log these changes
- Change into repo folder with `cd firstrepo` (change directory)
- With RStudio or a text editor (e.g. download VS Code at `https://code.visualstudio.com/`), add a file `somecode.R` into the repo which contains `print("hello world")`
- Now you can commit the changes that were made to the repo. This will become a very familiar workflow

Git and GitHub example: Committing

- 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)
- Commit/log changes with
  `git commit -m "added a code sample"`
- That's it
- To study this again, add another line
  `print("hello london")` to the file and repeat the above
- Run `git log` to see the history of commits

Git and GitHub example: Push, graphical user interface of the website, and pull

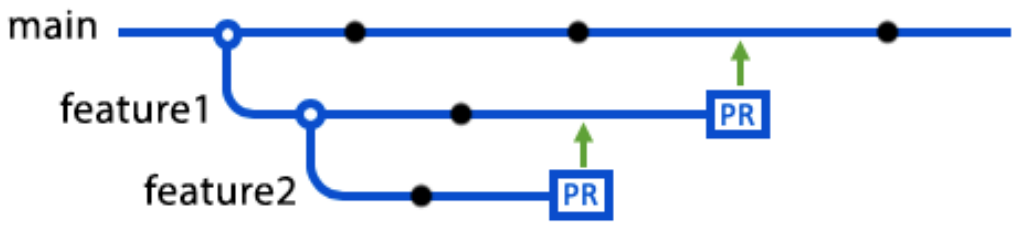
- To store these changes also in the remote repository, run `git push` afterwards
- It is now possible to review the changes in detail also in the browser which is very helpful for large code files. For this, go to the repository page on GitHub and click on the clock symbol next to 'commits' in the upper right hand corner
- If you now click on the key describing a specific commit, which could e.g. look something like '472cb9d', then the GitHub website visualises which lines of code changed through the commit.
- 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 describing edits"`: 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 (discussed in exercises)
- We will discuss these in the lab



Git/GitHub - further options and study

- People often use a combination of Git via the command line and the user interface of the GitHub website
- Yet, 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/en`

Plan for today

- Administration and logistics
- R and RStudio
- Git/GitHub
- Coding

Coding

We will no review some R:

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

Appendix: Some Linux commands for Git Bash and Mac Terminal (1/2)

- `pwd` "Print working directory" prints out the path to the current directory
- `cd` Changes the directory, e.g. `cd code` will try to go to a folder "code" contained in the current directory, `cd ..` goes back one folder level, only `cd` will go back to the home directory, and `cd some/path/to/a/folder` will go to a specific folder. We will use these commands to navigate into Git directories
- `ls` shows all folders and files in the current directory (`ls -a` also shows hidden files)
- Other helpful commands can be `mkdir`, `rmdir`, `rm`, and `touch`

Appendix: Some Linux commands for Git Bash and Mac Terminal (2/2)

- Lastly, one slightly tedious particularity of Git in the command line interfaces can be to get stuck in Vim (an old school command line text editor) when the software wants to edit a text file or give users the option to store a (commit) message
- The trick when using this command line editor is to switch between an *insert* mode to write text and another mode to exit, save etc. Switching between these modes is done with the key `esc`. To exit the editor without writing anything, just press `esc` to leave the insert mode and then `:q` followed by `enter` (or `:q!` to force-quit in some cases)
- If you want to have a closer look at Vim, enter `vim` into your Terminal or Git bash, or alternatively even change Git's default command line editor from Vim to e.g. the more user-friendly Nano with `git config --global core.editor="nano"`. Yet, neither is necessary for this course.