University of Oxford International Health and Tropical Medicine

Proochista Ariana and Ernest Guevarra

2023-01-25

Contents

Open Science and Reproducible Research Handbook

Chapter 1

Introduction

1.1 All about R

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, etc.) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

R is unique in that it is not general-purpose. It does not compromise by trying to do a lot of things. It does a few things very well, mainly statistical analysis and data visualization. While you can find data analysis and machine learning libraries for languages like Python, R has many statistical functionalities built into its core. No third-party libraries are needed for much of the core data analysis you can do with the language.

But even with this specific use case, it is used in every industry you can think of because a modern business runs on data. Using past data, data scientists and data analysts can determine the health of a business and give business leaders actionable insights into the future of their company.

Just because R is specifically used for statistical analysis and data visualization doesn't mean its use is limited. It's actually quite popular, ranking 12th in the TIOBE index of the most popular programming languages.

Academics, scientists, and researchers use R to analyze the results of experiments. In addition, businesses of all sizes and in every industry use it to extract insights from the increasing amount of daily data they generate.

1.2   Open and reproducible science

Open Science is the practice of science in such a way that others can collaborate and contribute, where research data, lab notes and other research processes are freely available, under terms that enable reuse, redistribution and reproduction of the research and its underlying data and methods. Reproducible research means that research data and code are made available so that others are able to reach the same results as are claimed in scientific outputs. Closely related is the concept of replicability, the act of repeating a scientific methodology to reach similar conclusions. These concepts are core elements of empirical research.

This lecture series is designed to give Oxford IHTM students a foundational understanding and appreciation of the pillars of Open Science more broadly and within that the concepts, methods and tools for Reproducible Research more specifically. To further the students' learning, practical examples and exercises will be discussed and walked through using the R language for statistical computing as a way to practically demonstrate these concepts.

Chapter 2

Installing R and RStudio

2.1 Setup for Windows machines

2.1.1 Install R for Windows machines

Following is a simple guide to installing R and RStudio and other required software on Windows machines.

Important that R is installed first. R is the main software and is needed for RStudio to work properly. R should always be installed first.

Go to https://cran.r-project.org

Click on link that says Download R for Windows

Then click on the link that says install R for the first time

Then click on Download R-4.X.X for Windows. This will start the download process.

Once downloaded, go to the `.exe` file in your `Downloads` folder, double-click and follow all the install prompts, selecting recommended options all the time

2.1.2 Install RStudio

This step requires that step 1 has been done and was successful.

Go to https://posit.co/download/rstudio-desktop/

Select the download specific for your Windows machine.

Once downloaded, double-click on `.exe` file downloaded to your `Downloads` folder and then follow all install prompts, always selecting recommended options.

2.1.3   Install `Rtools`

For the things that you will be taught in the **Open and Reproducibile Science Lecture Series**, you we will need to expand the installation of R by installing the `Rtools` software from R.

The download link is https://cran.r-project.org/bin/windows/Rtools/rtools42/rtools.html if you have installed R version 4.2.0 (or higher).

Once you have downloaded the `.exe` file, double-click on the `.exe` file and follow all install prompts. Choose all the recommended options.

2.1.4   Install Git for Windows

For the things that you will be taught in the **Open and Reproducibile Science Lecture Series**, you we will need to install **Git for Windows**.

If your machine is 64-bit machine (most likely the case if your computer is new and running Windows 10 or later) -

Use this download link - https://github.com/git-for-windows/git/releases/latest - to download the latest version of git. Make sure to choose the appropriate download for your machine (i.e., if you have a 64-bit machine then select the download for 64-bit machines; if you have a 32-bit machine then select the download for 32-bit machines)

Once you have downloaded the `.exe` file, double-click it and then follow all install prompts. Choose all recommended options.

2.2 Setup for macOS machines

2.2.1 Install R for macOS machines

Following is a simple guide to installing R and RStudio and other required software on macOS machines.

Important that R is installed first. R is the main software and is needed for RStudio to work properly. R should always be installed first.

Go to https://cran.r-project.org

Click on link that says Download R for macOS

You will have two choices of R versions to install. Make sure to install the appropriate version for your macOS version (Apple Silicon vs Apple Intel version). Click on the download link for your macOS version. This will start the download process of the `.pkg` file specific for installing in macOS computers.

Once downloaded, go to the `.pkg` file in your `Downloads` folder, double-click and follow all the install prompts, selecting recommended options all the time

2.2.2 Install RStudio

This step requires that step 1 has been done and was successful.

Go to https://posit.co/download/rstudio-desktop/

Select the download specific for your macOS machine.

Once downloaded, double-click on `.dmg` file downloaded to your `Downloads` folder and then follow all install prompts, always selecting recommended options.

2.2.3 Install git for macOS

For the things that you will be taught in the **Open and Reproducibile Science Lecture Series**, you we will need to install **git for macOS**. Apple machines are already pre-installed with `git` but it is usually an Apple specific version of git and tends to be older and not configured in the way we need

it. So we need to install another version of it that comes with Apple's `Xcode`
`command line tools`.

To install, go to the macOS terminal and type the following command:

```
xcode-select --install
```

Chapter 3

Introduction to R and RStudio

3.1 What is R?

R is a system for data manipulation, calculation, and graphics. It provides:

- Facilities for data handling and storage

- A large collection of tools for data analysis

- Graphical facilities for data analysis and display

- A simple but powerful programming language

R is often described as an environment for working with data. This is in contrast to a statistical *package* which is a collection of very specific tools. R is not strictly a statistics system but a system that provides many classical and modern statistical procedures as part of a broader data-analysis tool. This is an important difference between R and other statistical systems. In R a statistical analysis is usually performed as a series of steps with intermediate results being stored in objects. Systems such as SPSS and SAS provide copious output from (e.g.) a regression analysis whereas R will give minimal output and store the results of a fit for subsequent interrogation or use with other R functions. This means that R can be tailored to produce exactly the analysis and results that you want rather than produce an analysis designed to fit all situations.

R is a language based product. This means that you interact with R by typing

commands such as:

```
table(SEX, LIFE)
```

rather than by using menus, dialog boxes, selection lists, and buttons. This may seem to be a drawback but it means that the system is considerably more flexible than one that relies on menus, buttons, and boxes. It also means that every stage of your data management and analysis can be recorded and edited and re-run at a later date. It also provides an audit trail for quality control purposes.

R is available under UNIX (including Linux), the Macintosh operating system OS X, and Microsoft Windows. The method used for starting R will vary from system to system. On UNIX systems you may need to issue the R command in a terminal session or click on an icon or menu option if your system has a windowing system. On Macintosh systems R will be available as an application but can also be run in a terminal session. On Microsoft Windows systems there will usually be an icon on the Start menu or the desktop.

3.2   Why use R?

R is an open source system and is available under the *GNU general public license* (GPL) which means that it is available for free but that there are some restrictions on how you are allowed to distribute the system and how you may charge for bespoke data analysis solutions written using the R system. Details of the general public license are available from http://www.gnu.org/copyleft/gpl.html.

R is available for download from http://www.r-project.org/.

This is also the best place to get extension packages and documentation. You may also subscribe to the R mailing lists from this site. R is supported through mailing lists. The level of support is at least as good as for commercial packages. It is typical to have queries answered in a matter of a few hours.

Even though R is a free package it is more powerful than most commercial

packages. Many of the modern procedures found in commercial packages were first developed and tested using `R` or **S-Plus** (the commercial equivalent of `R`).

3.3   What is RStudio

RStudio is an **integrated development environment (IDE)** for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Workbench (Debian/Ubuntu, Red Hat/CentOS, and SUSE Linux).

RStudio IDE Overview

Chapter 4

Introduction to git and GitHub

4.1   All about git

`git` is a version control system for software development. It allows developers to keep track of changes made to their code and collaborate with other developers on a project. `git` also allows for easy rollbacks and branch management. It is widely used in the software industry and is considered one of the best version control systems available.

`git` was developed by Linus Torvalds in 2005. He created `git` as a replacement for the proprietary version control system he was using at the time. The development of `git` was driven by the need for a distributed version control system, which allows multiple developers to work on a project simultaneously, without the need for a central server. Linus Torvalds is also known for creating the Linux operating system kernel.

To use `git` in your machines, you will need to install it as described in the previous chapter (Chapter 2).

4.2   All about GitHub

GitHub is a web-based platform that provides hosting for software development and a community of developers to collaborate, share and learn from each other. It is built on top of `git`, which is the version control system used for managing

and tracking changes to the code. Developers can use GitHub to store and manage their code, collaborate with other developers, and track and manage issues and bugs. It also provides tools for code review, project management, and documentation. It is widely used by developers and organizations to host and share code, as well as to build and maintain open-source software.

GitHub is not a software you need to install. Rather it is a remote or cloud-based server that holds its users' code versioned using the `git` version control system and to which a user's local, git-versioned code syncs/communicates with.

A good illustration of the `git` and `GitHub` relationship can be viewed below:
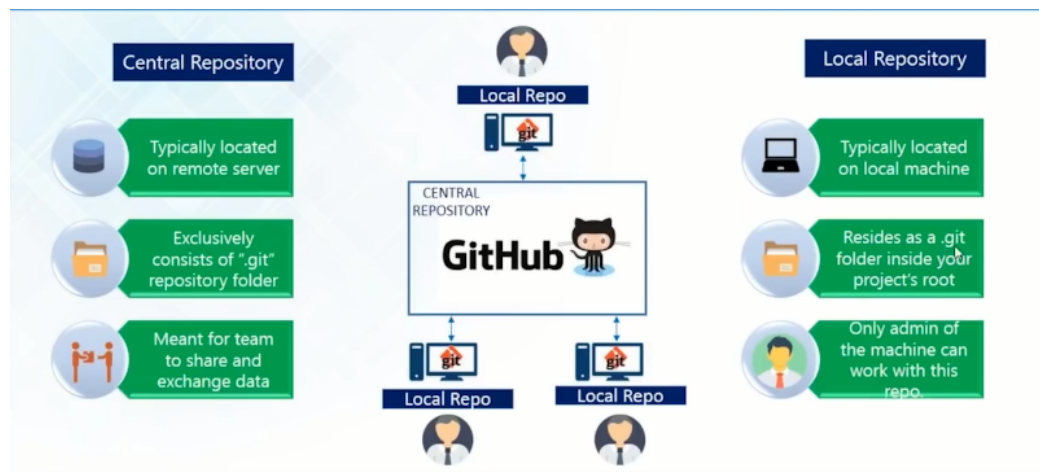
To use GitHub, you will need create an account at https://github.com.

4.2.1   Creating a GitHub account

1. Head to https://github.com

2. On the upper right hand corner of the page, click on **Sign-up** button
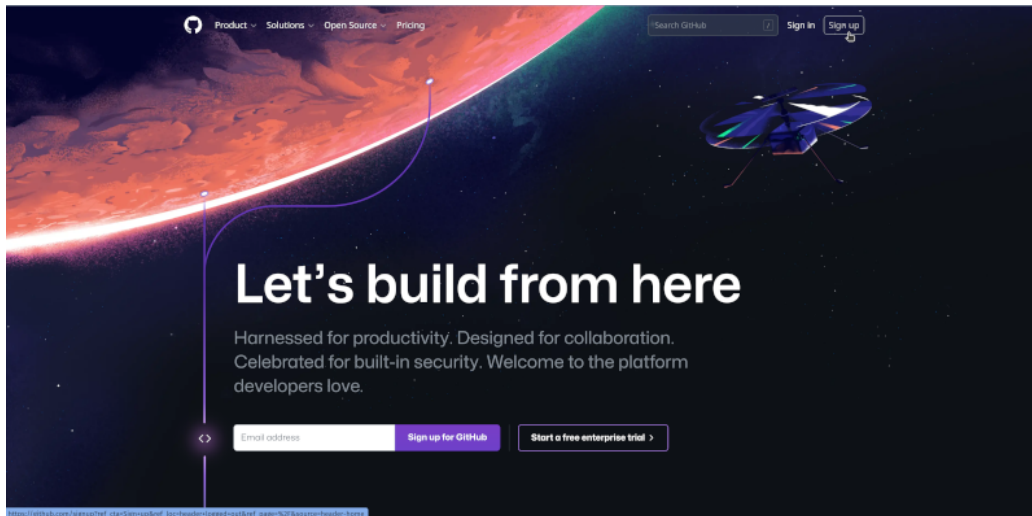
3. You will be then prompted to provide an email address to register your account with.
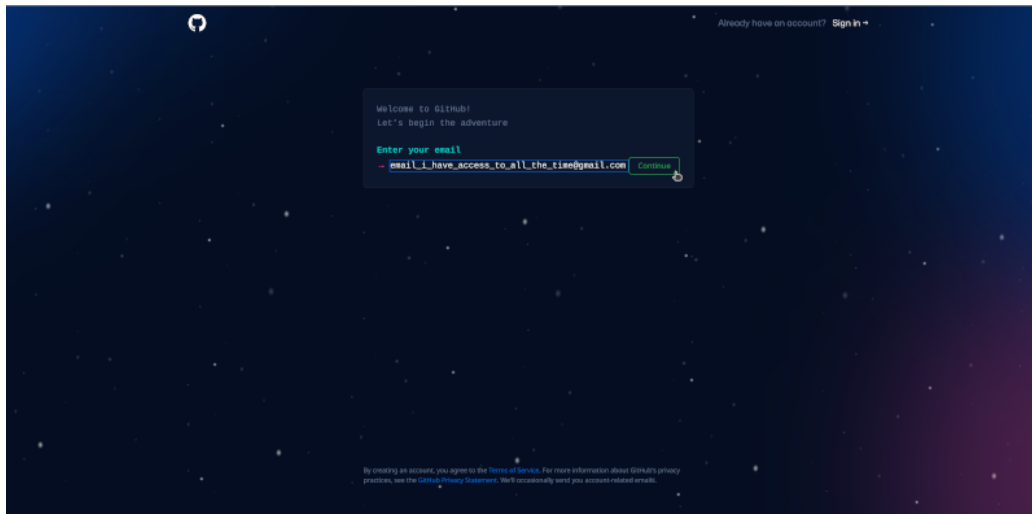
With regard to the email address to use for creating a GitHub account, a best practice recommendation is to use an email address that you will have access to all the time. Email addresses such as those for school (if you are a student) or for your current work may not always be the best email address to use as these email addresses tend to be time-limited (i.e., you lose the email address once you graduate or once you leave your current work).

4. You will then be prompted for a password.
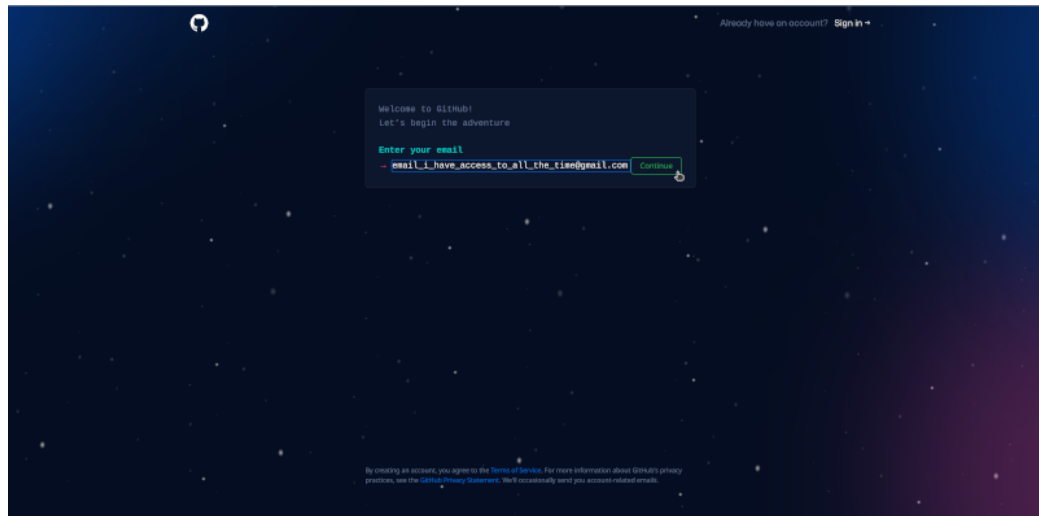
5. Then follow all other prompts after this including confirmation of your email and creating a GitHub username.

With regard to creating/selecting a GitHub username, following are some best practice recommendations:

- Incorporate your actual name! People like to know who they're dealing with. Also makes your username easier for people to guess or remember.

- Pick a username you will be comfortable revealing to your future boss and/or to future collaborators.

- Shorter is better than longer.

- Be as unique as possible in as few characters as possible. In some settings GitHub auto-completes or suggests usernames.

- Make it timeless. Don't highlight your current university, employer, or place of residence.

- Avoid the use of upper vs. lower case to separate words. We highly recommend all lowercase. GitHub treats usernames in a case insensitive way, but using all lowercase is kinder to people doing downstream regular expression work with usernames, in various languages. A better strategy for word separation is to use a hyphen -.

- You can change your username later, but better to get this right the first time.

https://help.github.com/articles/changing-your-github-username/

https://help.github.com/articles/what-happens-when-i-change-my-username/

6. Once you have created a GitHub account, please do share your username to the lecture seriese presenter so he can add your username to the Oxford IHTM GitHub account and add you to the Oxford IHTM GitHub Classroom.

Chapter 5

Connecting RStudio with GitHub

Once RStudio is installed and a GitHub account has been created and registered, the final step in the R, RStudio, git, and GitHub dance is to create git-related settings on your machine that will identify you as a unique/specific git user and then creating a GitHub **personal access token (PAT)** which will serve as the key or proof that you will use to identify yourself as the git user you claim to be whenever you try to syn/connect to your GitHub account via RStudio or any other tool on your local computer.

Follow are the steps you will need to perform:

1. On your local machine, open RStudio and then select the `Terminal` tab in the console pane.

The `Terminal` is the tool used to interface with your computer through commands written in a programming language called `bash`. Most of you would have never used the `Terminal` because the computers you use have different kinds of software built-in that provide a **graphical user interface (GUI)** that you can use to perform operations and tasks.

RStudio comes with the `Terminal` tool built in and usually is already available from the console pane as a separate table along side the R console (see below). By default, whenever you open RStudio, this tab for the `Terminal` tool should already be available.

If if is not present, you can easily open a new `Terminal` tab in the console pane by going to the RStudio menu ribbon and clicking on:

`Tools --> Terminal --> New Terminal`

or you can use the `ALT + SHIFT + R` keyboard shortcut.

   2. Introduce yourself to git

In this step, you are basically going to issue a set of commands to your computer to save and store specific settings for the `git` software that you have installed.

Specifically, you are going to let git know who you are (your name) and what your email address is **that is associated with your GitHub account**.

The commands will be issued on the `Terminal` that we mentioned in step 1. The commands are:

```
git config --global user.name 'Jane Doe'
git config --global user.email 'jane@example.com'
```

Make sure to supply your full proper name (and not your username you created in GitHub).

Make sure that the email you provide is the email address you used to register and create an account with GitHub.

Unless there was an error in your syntax, you should not expect any output on the `Terminal` after you issue the commands. To check that your name and email address have been recorded and/or that the name and email address recorded is correct, you can issue the following command:

```
git config --global --list
```

Here is an example of what you will see after issue this command:

```
$ git config --global --list

user.name=Jane Doe
user.email=jane@example.com
```

Check this output to what you expect it to be specified as. If the information is correct, then you've completed this step. If you need to correct any of

this information, then repeat this step making sure that the name and email address you provide is correct.

3. Create a GitHub personal access token (PAT)

Now that you have introduced yourself to the `git` that is installed in your local machine/computer, you should now visit GitHub on a browser (https://github.com) and create a **personal access token (PAT)**.

When you communicate/sync/interact with a remote git server, such as GitHub, you have to include credentials in the communication you are sending. These credentials prove that you are a specific GitHub user, who's allowed to do whatever you're asking to do.

`git` can communicate with a remote server using one of two protocols, HTTPS or SSH, and the different protocols use different credentials.

Here we describe the credential setup for the HTTPS protocol, which is what we recommend if you have no burning reason to pick SSH. With HTTPS, we will use a **personal access token (PAT)**.

Please note that the **personal access token (PAT)** is not the same as the password you provided when registering for your GitHub account. Also, in performing a connection to GitHub via HTTPS protocol, your password is not an acceptable credential for communicating with GitHub as a server.

To create a GitHub **personal access token (PAT)**, you need to:

3a. Go to Settings from your GitHub account menu

Login to your GitHub account. On the upper right hand corner of the GitHub page you will see your account icon. Click on it to reveal a drop down menu as shown below. Select the **Settings** option.
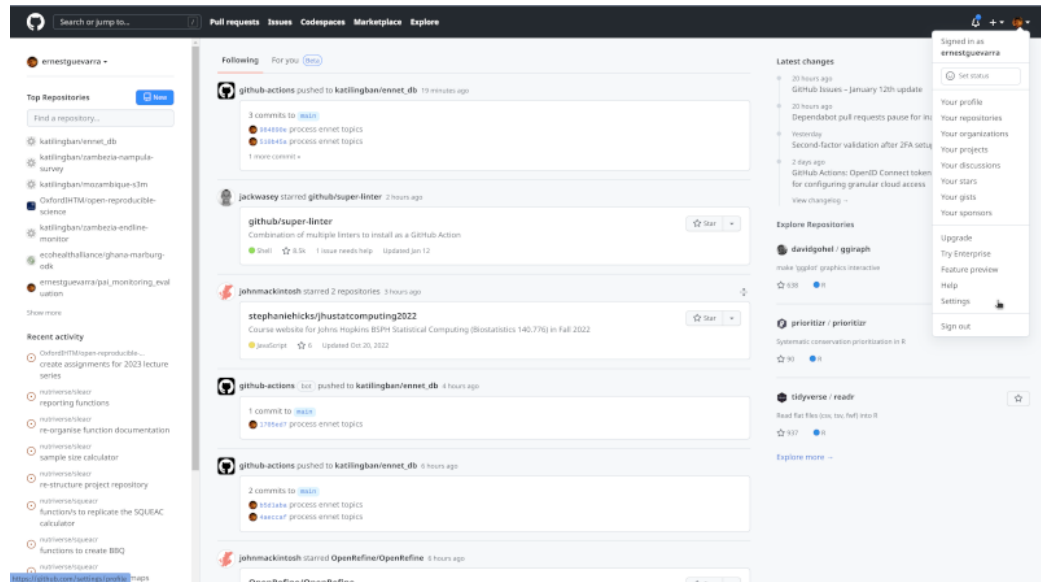
3b. From Settings navigate to Developer Settings

In the **Settings** page, find the **Developer Settings** option on the left hand sidebar as shown below:
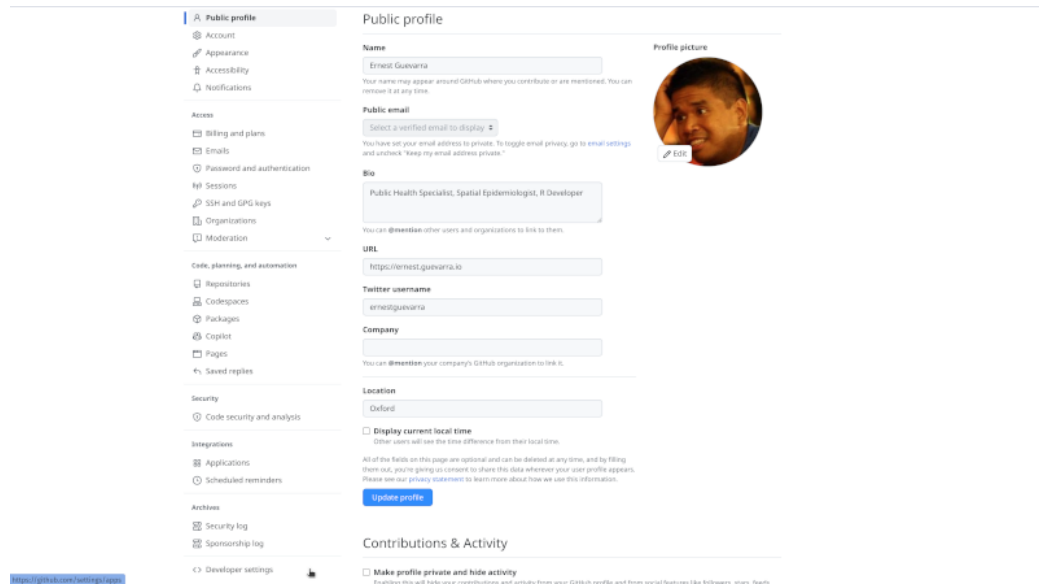
3c. From Developer Settings navigate to Personal Access Token

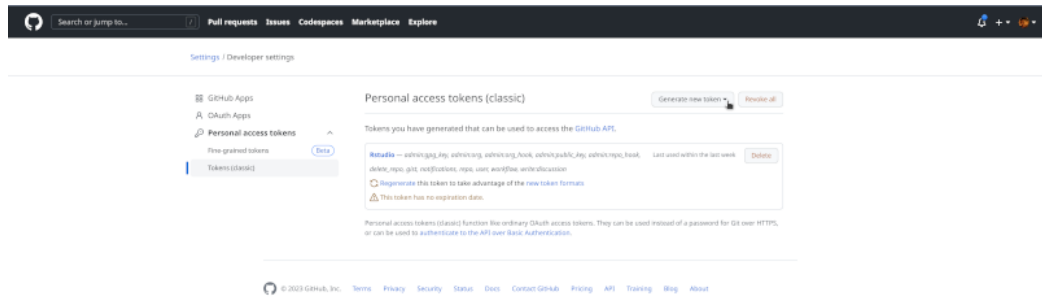In the **Developer Settings** page, find the **Personal Access Token** option

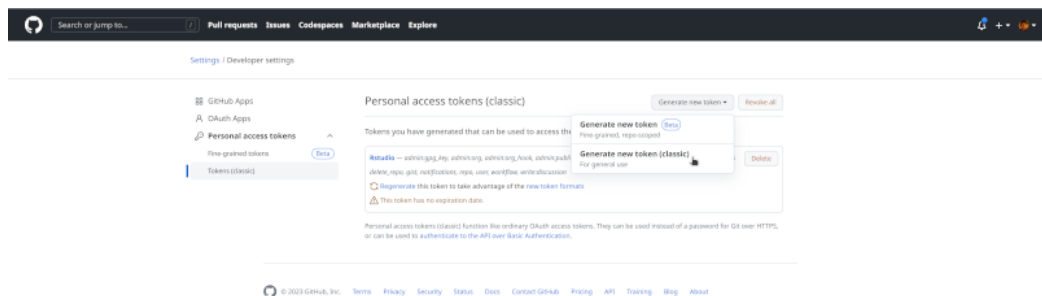on the left hand sidebar as shown below:
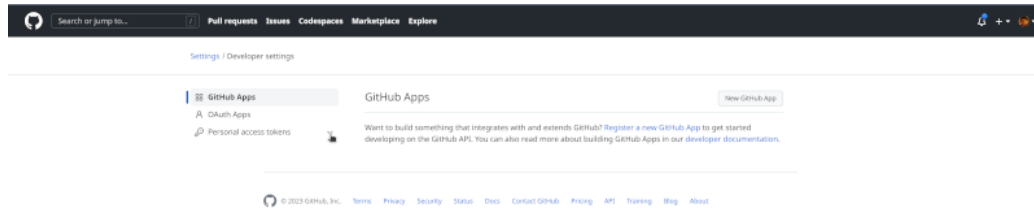
3d. Select Tokens (classic)
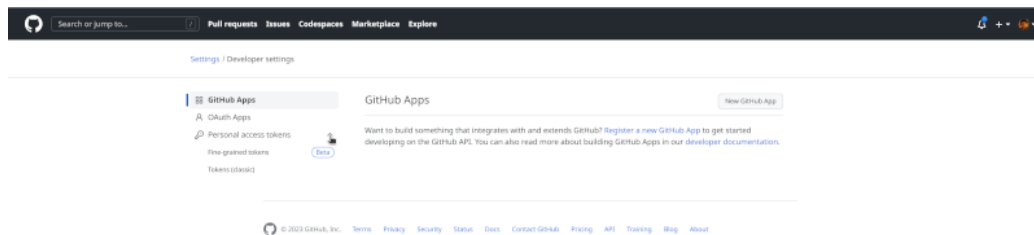
3e. Click on Generate new token

3f. Select Generate new token (classic)

3g. Give your token a name

The token name should be short but descriptive of the where or how you will use the token. Since we are using this to connect and allow communication between RStudio and GitHub, **RStudio** can be a good name that would remind you that this is what you are using to secure the connection/communication between your RStudio and your GitHub account.

3f. Set an expiry date for the token

By default, GitHub will set a **30 day validity** for any new token created. Clicking on the option menu will show the other possible time periods to choose from including **No expiry date**.

It is best practice to assign an expiry date for security tokens such as the GitHub PAT. And a 30 day validity is standard practice. However, in reality, it is cumbersome to be creating new tokens frequently and for beginners, having to go through these steps again can be quite a chore. For the purposes of this lecture series, we would recommend setting the expiry for about 90 days to cover the whole period and then as a group, we'll have a **renew GitHub PAT party** on our last session.

3g. Set scopes

Scopes are the types of permissions that you are attaching the token you are generating. This is again a security feature as takens should only be given specific and limited permissions based on what you intend the token to be used for. It is not good practice to give a token complete or unlimited permissions as you are exposing your account to high risk if and when your token gets compromised.

For general R users, the following scopes are currently recommended:

- repo

- workflow
- gist
- user

3h. Click on Generate token

After clicking you will now see a long string of characters and numbers which is your GitHub PAT. It is important to remember that once you see the generated GitHub PAT, you should copy this right away and store it securely. Treat your GitHub PAT in the same way you would treat your password for online accounts. The best way to securely store the GitHub PAT is using a password manager (LastPass, 1Password, BitWarden). If you have a macOS computer, you can save your GitHub PAT into your computer's keychain.

Non-secure password/token storage practices that has been done by other students before are:

- Email their password/token to themselves

If you are using free email services such as Gmail, then this is a hihly non-secure method. Others use their University of Oxford email address and argue that this is secure compared to using the free email services. Whilst it is true that a university email account is more secure, email communications and email storage is still one of the most vulnerable places to keep something that

is meant to be kept secret.

- Paste the token into a Word document and save in personal computer with the filename GITHUB_PAT.docx

- Paste the token into a Word document and save in Dropbox or in Google Drive

Please **AVOID** these methods.

Chapter 6

Initiating an R/RStudio project

Following are the steps to initiating your own R/RStudio project.

6.1   1. Create a new project in RStudio

6.2   2. Structure/organise your new project appropriately

6.3   3. Start coding and create an R script for your project's workflow

The next steps are specific for when you decide to use git for versioning your project.

6.4   4. Commit changes

6.5   5. Push your changes to your remote git

6.6   6.

Chapter 7

Writing functions

For this topic, we will use data on weight and height to calculate body mass index. As a refresher, body mass index is calculated as follows:

$$\text{Body mass index} = \frac{weight\ (kgs)}{height\ (m)^2}$$

For this topic on writing functions in R, we will use BMI as an example to explore and demonstrate how we can create our own functions in R.

Let's say for example that you have been doing a research on children aged 11 years and older in 3 schools and you have collected the following data:

School 1516

```
school1516
```

```
##     school sex ageMonths weight height
## 427   1516   1       138   24.5  126.0
## 428   1516   1       150   28.3  136.3
## 429   1516   1       162   32.2  143.5
## 430   1516   1       162   32.7  143.5
## 431   1516   1       150   28.6  137.0
## 432   1516   2       138   26.5  134.0
## 433   1516   1       150   29.9  139.2
## 434   1516   1       150   30.0  139.5
## 435   1516   1       162   34.0  148.0
```

```
## 436    1516    1        138    25.4   135.7
## 437    1516    1        150    32.3   143.0
## 438    1516    2        174    38.3   153.5
## 439    1516    2        162    41.6   151.0
## 440    1516    1        150    30.7   145.0
## 441    1516    2        186    46.8   155.2
## 442    1516    1        186    46.6   163.4
## 443    1516    1        150    33.5   145.5
## 444    1516    1        186    47.0   164.0
## 445    1516    1        174    41.1   159.5
## 446    1516    2        162    39.1   152.2
## 447    1516    2        174    40.9   155.5
## 448    1516    2        162    39.7   153.0
## 449    1516    2        162    40.9   153.2
## 450    1516    1        150    34.2   147.5
## 451    1516    2        150    41.8   149.4
## 452    1516    1        138    28.0   141.5
## 453    1516    1        138    30.0   142.0
## 454    1516    1        138    33.1   142.0
## 455    1516    1        186    46.1   167.5
## 456    1516    1        150    36.2   149.0
## 457    1516    2        162    47.4   156.0
## 458    1516    1        150    30.3   150.2
## 459    1516    2        150    36.4   152.1
## 460    1516    2        150    36.4   155.0
## 461    1516    2        150    44.1   155.0
## 462    1516    2        162    42.3   160.1
## 463    1516    2        179    50.4   163.5
## 464    1516    1        150    37.6   155.0
## 465    1516    2        138    36.0   154.5
## 466    1516    2        138    46.1   156.0
```

School 1522

```
school1522
```

```
##      school sex ageMonths weight height
## 646    1522    1        203    30.6   140.5
## 647    1522    1        174    30.8   140.0
```

```
## 648   1522   1       162    29.3   136.3
## 649   1522   1       150    24.0   132.0
## 650   1522   1       150    28.1   132.1
## 651   1522   2       150    27.2   134.9
## 652   1522   1       162    34.2   139.2
## 653   1522   1       150    25.5   134.2
## 654   1522   1       138    24.6   129.0
## 655   1522   1       174    36.4   147.5
## 656   1522   1       150    28.7   137.5
## 657   1522   1       186    45.8   155.6
## 658   1522   1       174    36.3   151.6
## 659   1522   1       150    31.0   139.5
## 660   1522   1       138    29.0   134.3
## 661   1522   1       179    38.3   155.5
## 662   1522   2       138    31.3   138.4
## 663   1522   1       162    36.5   148.8
## 664   1522   1       155    36.8   145.2
## 665   1522   1       138    28.3   136.8
## 666   1522   1       138    26.8   137.3
## 667   1522   2       138    32.6   141.4
## 668   1522   2       138    31.9   143.0
## 669   1522   1       174    42.6   160.7
## 670   1522   2       198    57.8   158.0
## 671   1522   2       162    43.9   153.5
## 672   1522   2       150    35.1   150.6
## 673   1522   2       186    52.6   159.6
## 674   1522   2       150    45.1   152.8
## 675   1522   2       138    34.6   147.2
## 676   1522   2       150    45.3   153.1
## 677   1522   1       186    51.8   170.2
## 678   1522   2       150    57.1   154.2
## 679   1522   2       138    33.5   149.2
## 680   1522   1       150    36.3   154.1
## 681   1522   1       174    44.0   169.1
## 682   1522   2       150    44.5   158.3
## 683   1522   2       150    51.5   159.1
## 684   1522   2       138    47.4   157.8
## 685   1522   2       138    36.8   158.5
```

```
## 686    1522    2        138    52.0   161.0
```

School 1525

```
school1525
```

```
##        school sex ageMonths weight height
## 752    1525    1        186    26.2    137
## 753    1525    1        186    32.7    138
## 754    1525    1        150    25.9    130
## 755    1525    1        162    30.4    137
## 756    1525    2        138    24.4    129
## 757    1525    2        138    23.8    130
## 758    1525    1        150    26.1    133
## 759    1525    1        150    26.4    135
## 760    1525    1        174    35.1    148
## 761    1525    1        162    28.7    142
## 762    1525    1        150    28.0    136
## 763    1525    1        174    34.0    149
## 764    1525    1        186    40.6    155
## 765    1525    2        150    35.8    142
## 766    1525    1        150    35.4    140
## 767    1525    2        138    27.8    137
## 768    1525    2        138    28.2    137
## 769    1525    2        138    29.7    139
## 770    1525    2        138    30.9    139
## 771    1525    1        138    28.2    137
## 772    1525    2        138    26.2    140
## 773    1525    2        138    26.6    140
## 774    1525    1        138    27.2    138
## 775    1525    2        138    27.0    141
## 776    1525    1        150    31.3    145
## 777    1525    2        162    33.9    152
## 778    1525    2        162    42.0    153
## 779    1525    2        185    38.3    157
## 780    1525    2        138    31.0    145
## 781    1525    2        138    32.3    145
## 782    1525    1        139    35.1    144
## 783    1525    2        150    36.4    152
```

```
## 784    1525    2        138    32.7    147
## 785    1525    1        174    44.9    166
## 786    1525    2        138    32.2    148
## 787    1525    2        138    36.4    148
## 788    1525    1        138    31.4    146
## 789    1525    2        138    45.0    149
## 790    1525    2        162    49.4    160
## 791    1525    2        138    34.3    150
## 792    1525    1        138    30.0    148
## 793    1525    2        150    37.0    156
## 794    1525    2        162    52.2    165
## 795    1525    2        138    42.9    158
```

In this dataset, the units of the height measurement is in centimetres.

Using what we have learned earlier on calculating BMI using R, I can perform the following R commands to get the BMI for each child in each of the schools:

```
## Calculate BMI for children in school 1516
school1516$weight / (school1516$height / 100) ^ 2

## Calculate BMI for children in school 1516
school1522$weight / (school1516$height / 100) ^ 2

## Calculate BMI for children in school 1516
school1525$weight / (school1516$height / 100) ^ 2
```

Because the commands are repetitive, I can easily copy and paste my initial line of code to calculate BMI for children in school 1516 and then just change the object names accordingly to calculate the BMI for children in the two other schools.

When I run these lines of code, I get the following results:

```
## Calculate BMI for children in school 1516
school1516$weight / (school1516$height / 100) ^ 2
```

```
##  [1] 15.43210 15.23333 15.63695 15.87976 15.23789 14.75830 15.43095 15.41604 15.52
## [15] 19.42954 17.45347 15.82409 17.47472 16.15550 16.87903 16.91463 16.95929 17.42
## [29] 16.43128 16.30557 19.47732 13.43083 15.73414 15.15088 18.35588 16.50280 18.85
```

```
## Calculate BMI for children in school 1516
school1522$weight / (school1516$height / 100) ^ 2
```

```
## Warning in school1522$weight/(school1516$height/100)^2: longer object lengt
```

```
##  [1] 19.27438 16.57903 14.22865 11.65487 14.97150 15.14814 17.65012 13.1036
## [15] 12.03967 14.34481 14.78490 13.57079 14.46527 12.21679 11.08343 13.9262
## [29] 16.07485 15.58488 18.61440 22.96095 24.68185 13.94381 15.10926 17.1660
```

```
## Calculate BMI for children in school 1516
school1525$weight / (school1516$height / 100) ^ 2
```

```
## Warning in school1525$weight/(school1516$height/100)^2: longer object lengt
```

```
##  [1] 16.50290 17.60176 12.57755 14.76284 13.00016 13.25462 13.46983 13.5661
## [15] 14.69670 10.41216 13.32058 11.04253 12.14611 12.17362 10.83529 11.3631
## [29] 11.04923 14.54889 14.42308 16.13472 14.13479 18.68887 13.40271 14.2009
## [43] 25.34934 20.83308
```

The calculation for the BMI of children in school 1516 seems to have completed without issues and a vector of BMI results have been produced. However, for school 1522 and school 1525, there is a warning saying:

```
## Warning in school1522$weight/(school1516$height)^2: longer object length
## of shorter object length
```

Although a result has been provided, the warning gives me an indication that someting is not quite right with my calculation and when I inspect further, I notice that in my formula for school 1522 and for school 1525, my denominator is still using data for school 1516 and this is most likely what is causing the warning message.

So, to correct this I go back to my lines of code and edit the denominators for school 1522 and school 1525 as follows:

```
## Calculate BMI for children in school 1516
school1516$weight / (school1516$height / 100) ^ 2
```

```
## Calculate BMI for children in school 1516
school1522$weight / (school1522$height / 100) ^ 2
```

```
## Calculate BMI for children in school 1516
school1525$weight / (school1525$height / 100) ^ 2
```

which gives me:

```
## Calculate BMI for children in school 1516
school1516$weight / (school1516$height / 100) ^ 2
```

```
##  [1] 15.43210 15.23333 15.63695 15.87976 15.23789 14.75830 15.43095 15.41604 15.52
## [15] 19.42954 17.45347 15.82409 17.47472 16.15550 16.87903 16.91463 16.95929 17.42
## [29] 16.43128 16.30557 19.47732 13.43083 15.73414 15.15088 18.35588 16.50280 18.85
```

```
## Calculate BMI for children in school 1516
school1522$weight / (school1522$height / 100) ^ 2
```

```
##  [1] 15.50132 15.71429 15.77161 13.77410 16.10277 14.94669 17.65012 14.15908 14.78
## [15] 16.07852 15.83937 16.34076 16.48493 17.45479 15.12217 14.21653 16.30492 15.59
## [29] 19.31656 15.96837 19.32626 17.88178 24.01416 15.04898 15.28626 15.38741 17.75
```

```
## Calculate BMI for children in school 1516
school1525$weight / (school1525$height / 100) ^ 2
```

```
##  [1] 13.95919 17.17076 15.32544 16.19692 14.66258 14.08284 14.75493 14.48560 16.02
## [15] 18.06122 14.81166 15.02477 15.37188 15.99296 15.02477 13.36735 13.57143 14.28
## [29] 14.74435 15.36266 16.92708 15.75485 15.13258 16.29409 14.70051 16.61797 14.73
## [43] 19.17355 17.18475
```

I now do not get the warning message and the expected length of BMI values for each school has now been produced.

From this short example above, we realise how tedious a task it is to type in the code above every time we need to calculate BMI. Also, it becomes even challenging to debug issues with the code because we have to review and edit (as needed) each iteration of the calculation to see where it may have gone wrong (especially when doing a cut and paste approach).

It would be better (and easier) to have a function that calculates and displays the BMI values automatically. Fortunately, R allows us to do just that.

The `function()` function allows us to create new functions in R with the following generic syntax:

```
function_name <- function(argument1, argument2, ...) {
  ## Your code here
}
```

Using this template/generic syntax, we apply it to create a function called
`calculate_bmi` as follows:

```
calculate_bmi <- function(weight, height) {
  weight / height ^ 2
}
```

We now have a function for calculating and outputing BMI values.

Let us now test it with our 3 sets of data:

School 1516

```
calculate_bmi(
  weight = school1516$weight,
  height = school1516$height / 100
)
```

```
##  [1] 15.43210 15.23333 15.63695 15.87976 15.23789 14.75830 15.43095 15.4160
## [15] 19.42954 17.45347 15.82409 17.47472 16.15550 16.87903 16.91463 16.9592
## [29] 16.43128 16.30557 19.47732 13.43083 15.73414 15.15088 18.35588 16.5028
```

School 1522

```
calculate_bmi(
  weight = school1522$weight,
  height = school1522$height / 100
)
```

```
##  [1] 15.50132 15.71429 15.77161 13.77410 16.10277 14.94669 17.65012 14.1590
## [15] 16.07852 15.83937 16.34076 16.48493 17.45479 15.12217 14.21653 16.3049
## [29] 19.31656 15.96837 19.32626 17.88178 24.01416 15.04898 15.28626 15.3874
```

School 1525

```
calculate_bmi(
  weight = school1525$weight,
  height = school1525$height / 100
```

```
)
```

```
##  [1] 13.95919 17.17076 15.32544 16.19692 14.66258 14.08284 14.75493 14.48560 16.02
## [15] 18.06122 14.81166 15.02477 15.37188 15.99296 15.02477 13.36735 13.57143 14.28
## [29] 14.74435 15.36266 16.92708 15.75485 15.13258 16.29409 14.70051 16.61797 14.73
## [43] 19.17355 17.18475
```

In our example here, the `calculate_bmi()` function helped a little bit in making the code to calculate BMI for each student in each school more efficient. But the efficiency that functions provide become more evident when you need to make more complex operations. For example, what if you need to get the mean BMI for students in each school? Without a function, we will have to do the following script for each school:

School 1516

```
## Calculate BMI for children in school 1516
bmi_school1516 <- school1516$weight / (school1516$height / 100) ^ 2

## Get the mean BMI for children in school 1516
mean_bmi_school1516 <- mean(bmi_school1516)

mean_bmi_school1516
```

```
## [1] 16.28491
```

School 1522

```
## Calculate BMI for children in school 1522
bmi_school1522 <- school1522$weight / (school1522$height / 100) ^ 2

## Get the mean BMI for children in school 1522
mean_bmi_school1522 <- mean(bmi_school1522)

mean_bmi_school1522
```

```
## [1] 16.89955
```

School 1525

```r
## Calculate BMI for children in school 1525
bmi_school1525 <- school1525$weight / (school1525$height) ^ 2

## Get the mean BMI for children in school 1525
mean_bmi_school1525 <- mean(bmi_school1525)

mean_bmi_school1525
```

```
## [1] 0.001564695
```

As the operations/calculations we want to perform become more complex, the copy and paste method becomes more and more tedious. With the function approach, we can use the following:

```r
calculate_mean_bmi <- function(weight, height) {
  bmi <- weight / height ^ 2

  mean_bmi <- mean(bmi)

  return(mean_bmi)
}
```

Applying the function to the datasets, we get:

School 1516

```r
calculate_mean_bmi(
  weight = school1516$weight,
  height = school1516$height / 100
)
```

```
## [1] 16.28491
```

School 1522

```r
calculate_mean_bmi(
  weight = school1522$weight,
  height = school1522$height / 100
)
```

```
## [1] 16.89955
```

School 1525

```
calculate_mean_bmi(
  weight = school1525$weight,
  height = school1525$height / 100
)
```

```
## [1] 15.64695
```

Chapter 8

Cloning a GitHub repository into your local computer using RStudio

This tutorial is a summary of the the instructions described here - https://do
cs.github.com/en/repositories/creating-and-managing-repositories/cloning-
a-repository.

8.1   Copy the repository URL of the repository you want to clone from GitHub
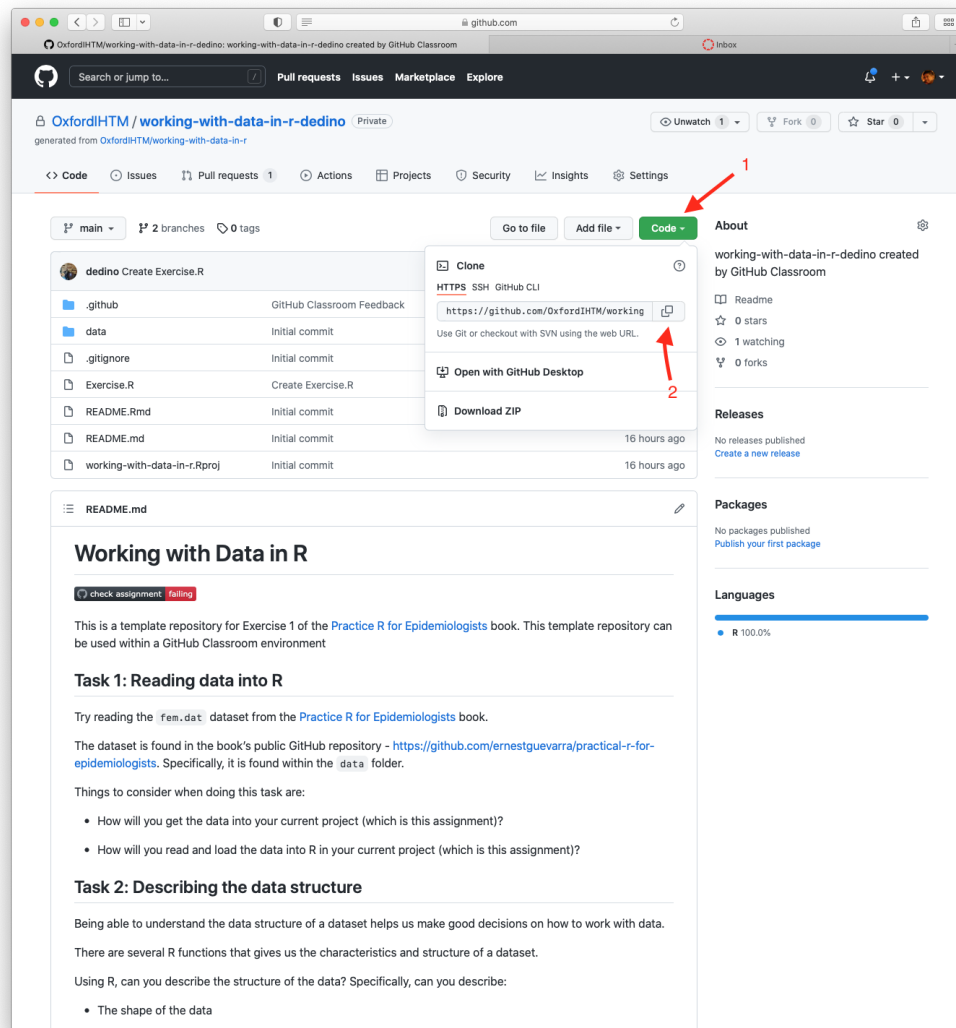
7.1.1 Go to the GitHub page of the repository you own and click on the green button that is labeled code

7.1.2 Click on the copy to clipboard icon to copy the repository URL.

8.2 Go to RStudio and create new project
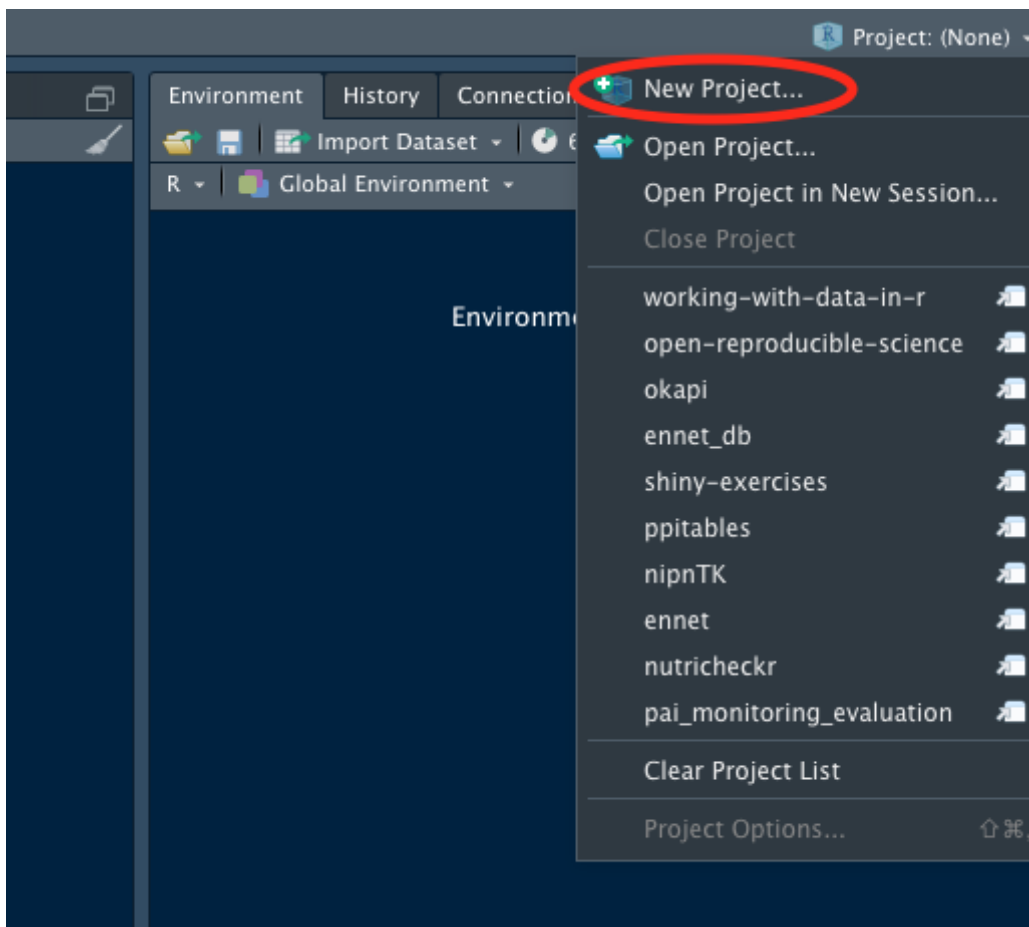
8.3   Choose Version Control
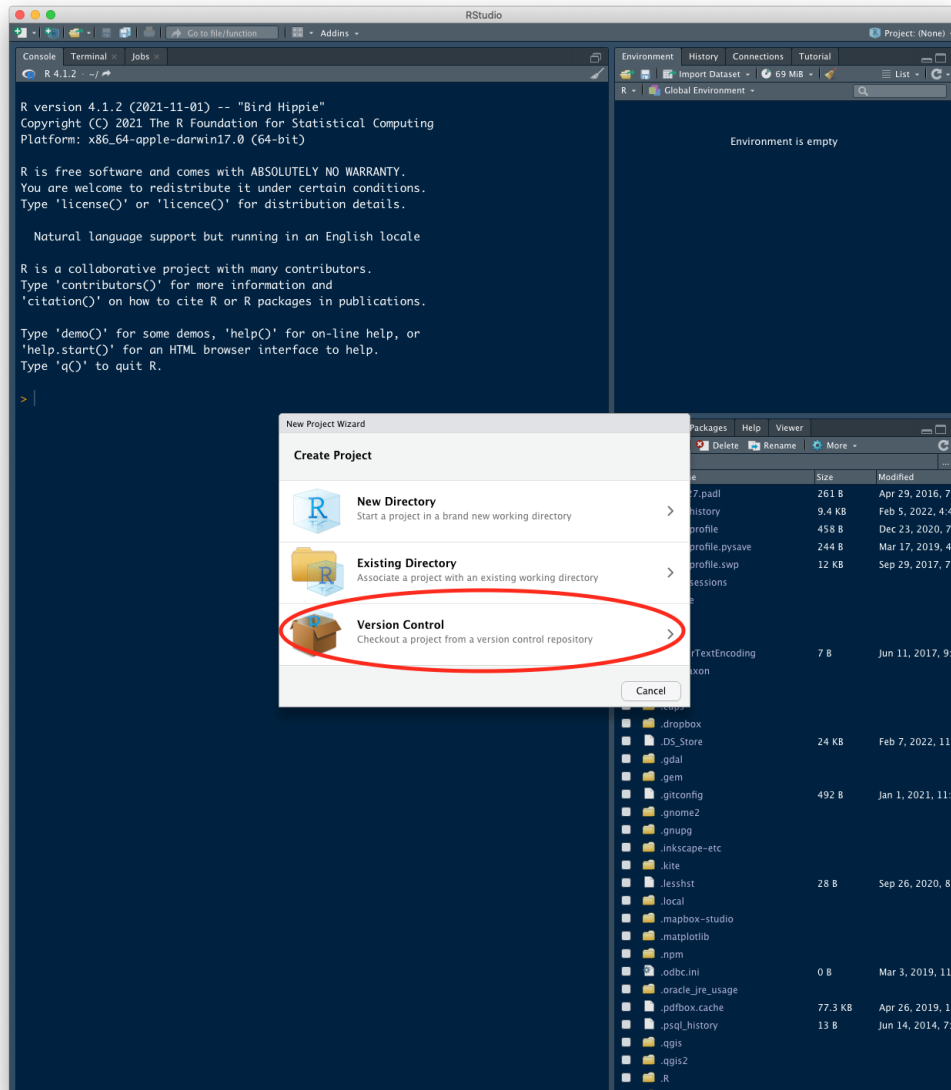
8.4 Select Git
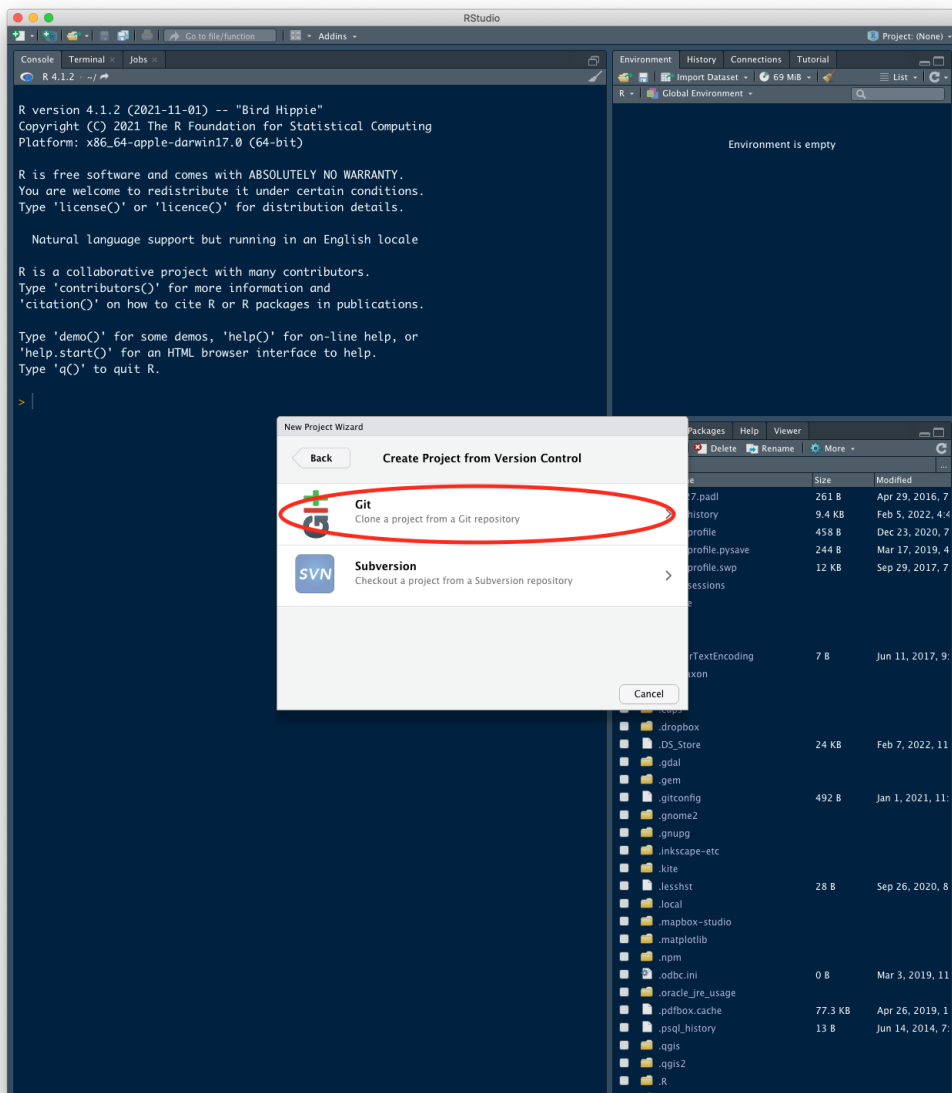
8.5 Setup repository settings
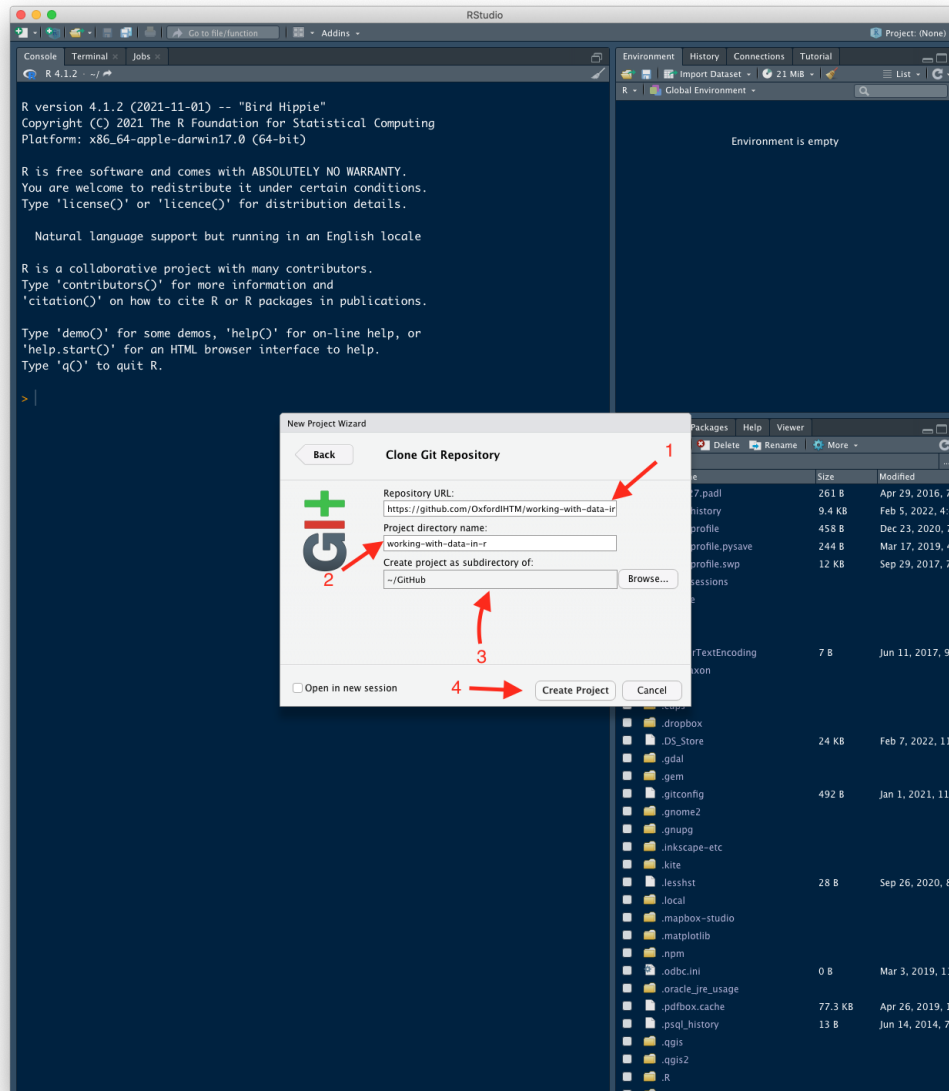
7.5.1 Paste the repository URL you copied earlier.

7.5.2 The Project directory name should be specified already after you paste the repository URL. Use the suggested directory name.

7.5.3 Browse for the directory on your local computer where you want to save the files for the specified project.

7.5.4 Click on the `Create Project` button/icon.

You will now have the GitHub repository in your local computer.

Chapter 9

Participating in an existing R/RStudio project

Following are the general steps to take when participating in an existing R/RStudio project that has been initiated and led by someone else.

There can be different scenarios

9.1 Steps when existing project is not an RStudio project

Chapter 10

Committing your changes and pushing them to GitHub

Following are the steps to take in committing your changes in RStudio and pushing them to GitHub.

10.1   Click on `Commit` in the Git tab on RStudio
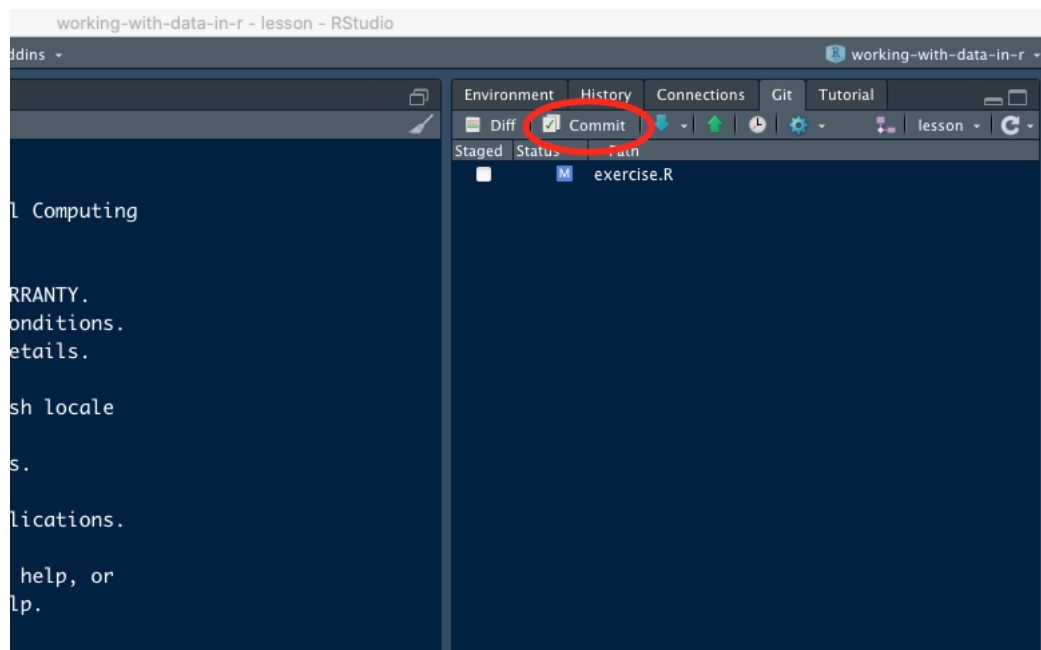
10.2 Getting changes saved and push to GitHub
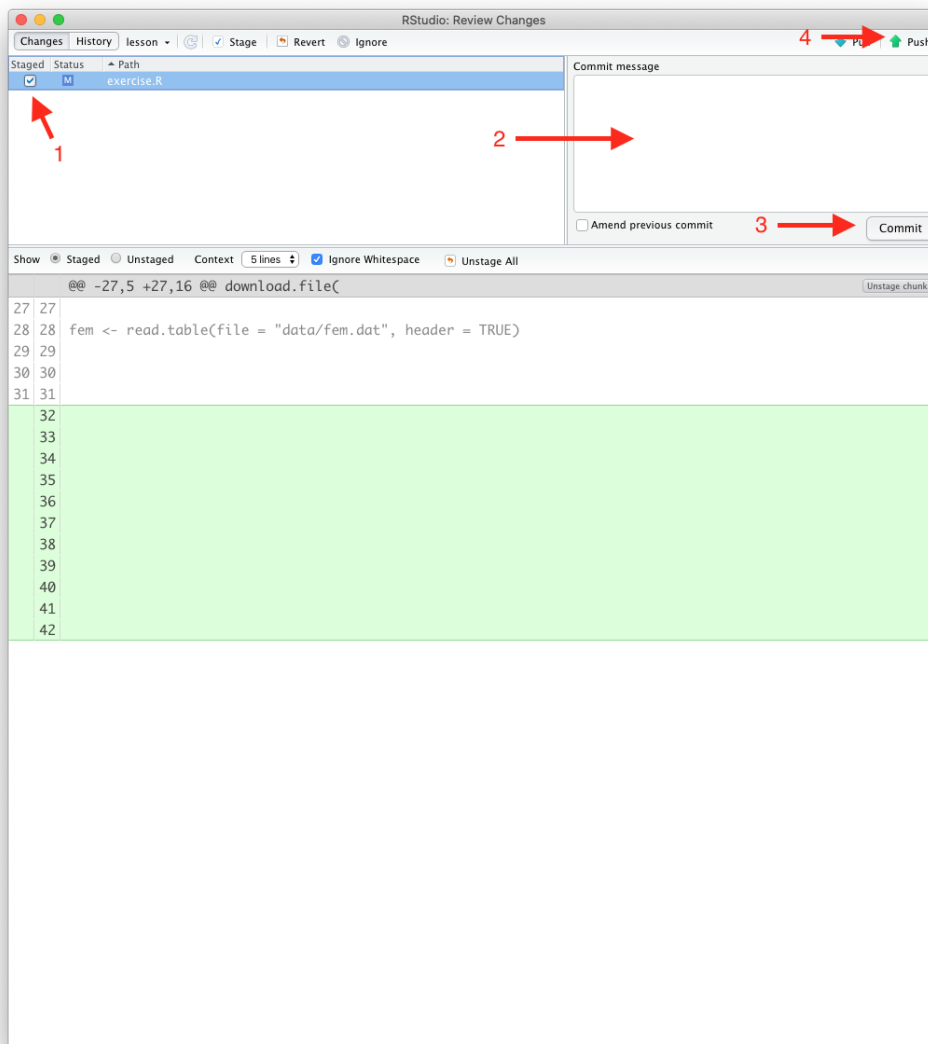
8.2.1 Tick the box beside the file that has changed to stage the changes.

8.2.2 Write a commit message in the `Commit message` dialog box. In the commit message, describe the changes that you made.

8.2.3 Click on the `Commit` button.

8.2.4 Click on the `Push` button.

Chapter 11

Creating portable and reproducible scientific workflows

At this point, you would have written your own R code and R scripts and saved these within an R file (`.R`).

By now, you would have also appreciated how extensible R is through built-in packages and/or through functions that you have created yourself.

So far, in the examples that we have worked on, the operations and the problems have been quite straightforward. But from your own experience dealing with your own data, real world data is far from straightforward and far from simple. Complexity is almost always a given.

R's scripting capability and R's extensibility are its main characteristics that make R a good tool for creating robust scientific workflows particularly for complex data and research projects.

A typical scientific workflow would have the following steps:
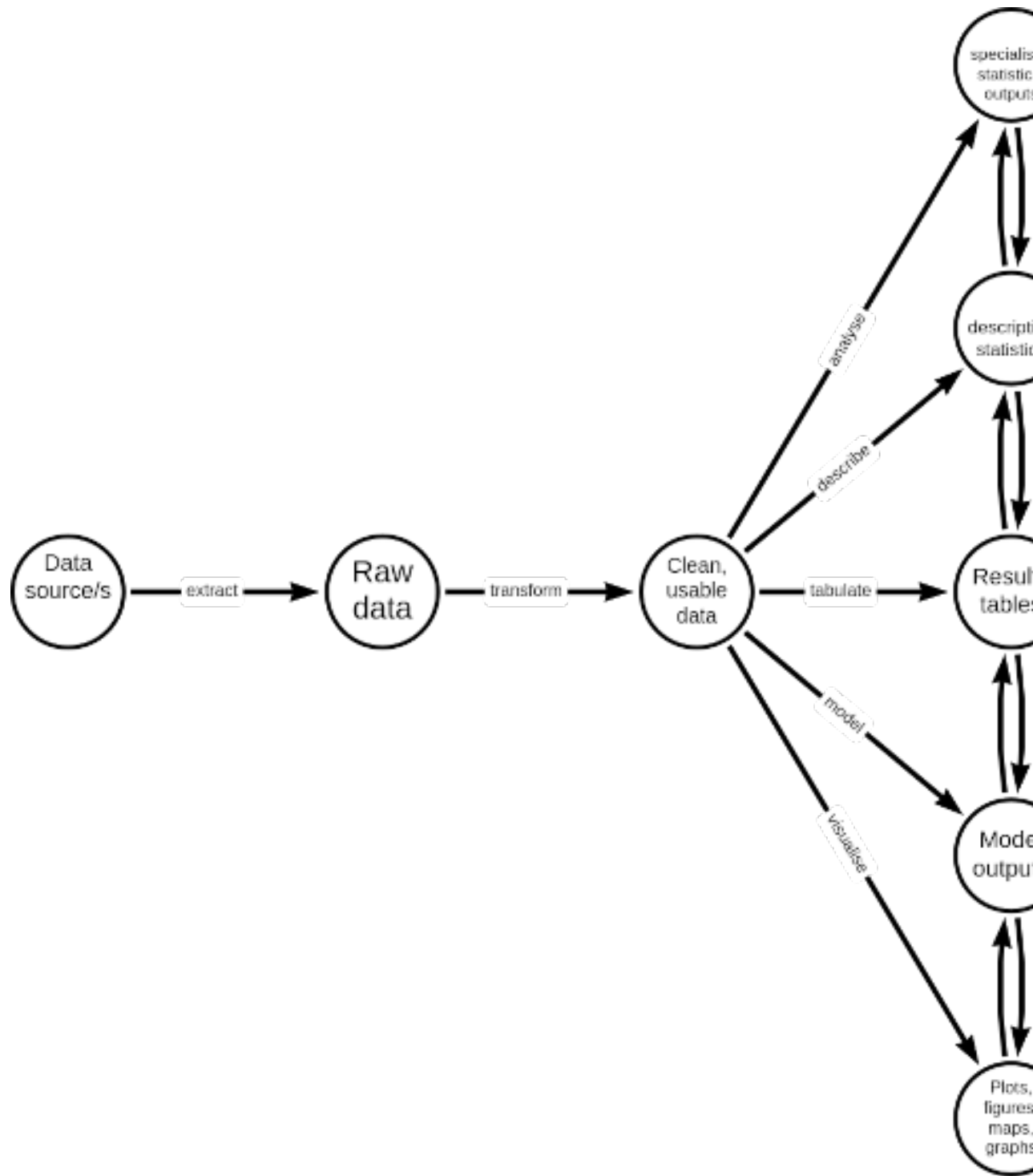
In general, an R script should reflect the different steps outlined above. Hence, an R script of a scientific workflow would tend to look like this:

```
## Load libraries


## Retrieve and read data


## Process data


## Analyse data

### Descriptive analysis

### Statistical tests

### Model specifications


## Outputs

### Tabulation of results

### Model outputs

### Plots


## Report
```

In this chapter, we will go through a step-by-step walkthrough of how to build a robust scientific workflow in R. A robust worklow is one that is **portable** i.e., not dependent on hardware and software and instead can be run on almost any machine with very minimal, if any, additional setup or configuration required, and one that is **reproducible** i.e., can be run over and over again without issues, providing the expected results with the same

data or providing updated results with new and/or updated data.

11.1   Step 1: Create a new RStudio project

- Open RStudio

- Click on the `File` option in the RStudio menu. In the dropdown menu, select `New Project`

- In the menu window, select `New directory` option.

- In the next menu window, select `New project` option.

- In the next menu window, enter the following details:

  - Name of the project - important to make the project name as short as possible but descriptive of the project you are creating; don't use spaces, instead use dash (or underscore) and avoid using capital letters;

  - Select the directory in your computer in which you want to save the project in. Click on `Browse` to open your computers file manager and navigate to the directory you want to save your project in;

  - Tick the selection box to make this project a git repository (whilst this is not necessary, this is highly recommended especially if you are collaborating with others);

  - Tick the selection box to enable `renv` in this project (this is what mainly contribute to the portability of your project); and,

  - Click on `Create project`

11.2   Step 2:   Create an R file called `packages.R`

- Click on the `File` option in the RStudio menu. In the dropdown menu, select `New File` and then in the next dropdown menu, select `R script`.

- A new tab will open in your text editor pane of RStudio (upper left pane) with the name *Untitled1*. Save this file by clicking on the disk

icon on the text editor menu or do a keyboard shortcut with `CTRL + s`. Give this empty R script the filename `packages.R`.

- You should now see a file in the main directory/root directory of your project named `packages.R`

- Add code in the `packages.R` file specifying the packages you will be using in this project. There will be standard packages that we will always use with this type of workflow. So a template/generic `packages.R` file will contain the following:

```
################################################################################
#
#'
#' General packages needed for a targets workflow
#'
#
################################################################################

library(targets)
library(tarchetypes)
library(here)
library(rmarkdown)
library(knitr)
library(kableExtra)
library(dplyr)
library(openxlsx)
library(ggplot2)


################################################################################
#
#'
#' Add other packages that will be used in the project below
#'
#
################################################################################
```

11.3   Step 3: Create placeholder directories for different components of workflow

- In the lower right pane of RStudio (the file manager pane), find the menu button labelled `Folder`.

- Give this new folder the label of `R`. This filder will hold all bespoke functions that we will create to use for this project workflow;

- Repeat these steps to create new folders with the following labels:

  - `data` - This folder will hold any data that we retrieve as part of this workflow.

  - `outputs` - This folder will hold all our workflow outputs such as plots/figures, tables (in Excel or CSV files), HTML and/or Word and/or PDF outputs

  - `reports` - This folder will hold all our RMarkdown report (`.Rmd`) files

  - `docs` - This folder will hold any of our deployed outputs such as HTML report, dashboard, etc.

These are placeholder directories which we will populate as we work through the workflow for this project.

11.4   Step 4: Create the target script file (`_targets.R`)

The next task is to create a {targets} script file (`_targets.R`) which is the file that will define the workflow that we will be creating.

Create the file by:

- Clicking on File –> New File –> R Script in RStudio.

- A new tab will show in your Source window on the top left quadrant of your RStudio screen. This tab will usually be called `Untitled1`.

- Save this file first and change its name to `_targets.R`. Make sure to save it in the current project directory.

- You know that you were successful in doing this once you see a file called `_targets.R` in the File system window in the lower right quadrant of your RStudio screen.

11.5 Step 5: Edit the `_targets.R` script file

Now, the next step is to edit your script file by adding sets of R code that does the following:

- Loads the packages required
- Loads custom functions (if any)
- Defines individual targets using `tar_targets` function
- Ends with a list of targets objects

A basic `{targets}` workflow will look like this:

```
## Load libraries ---------------------------------------------------
library(targets)



## Load custom functions --------------------------------------------
for (f in list.files("R", full.names = TRUE)) source (f)
for (f in list.files(here::here("R"), full.names = TRUE)) source (f)



## Create targets and list targets objects --------------------------
```