SATVI Computational Course

Session Guide

SATVI Computational Group

2024-03-05

Table of contents

Preface

This is a session guide book for the SATVI Computational Course.

This is a version-controlled living document that will be updated as needed as the course progresses. All changes are tracked using git.

1 Introduction

Welcome to the SATVI Computational Course! This course is designed to strengthen fundamental coding skills for SATVI trainees and staff. The curriculum will take you through the basics of R, using the terminal, creating and using git controlled projects, as well as more advanced data analysis methods commonly used at SATVI.

All lessons will be stored on the SATVI GitHub under the repository SATVI_ComputationalCourse. To access all relevant course content, navigate to https://github.com/SATVILab/SATVI_ComputationalCourse.

A static webpage version of the course is also available at https://satvilab.github.io/SATVI_ComputationalCourse/

Your instructors are SATVI members with experience in each topic. For session-specific questions, please contact the relevant instructor:

1.1 Instructor contacts

`Carly Young-Baile: carly.young-bailie@uct.ac.za`

`Monika Looney: monika.looney@uct.ac.za`

`Miguel Rodo: miguel.rodo@uct.ac.za`

`Simon Mendelsohn: simon.mendelsohn@uct.ac.za`

`Munyaradzi Musvosvi: munyaradzi.musvosvi@uct.ac.za`

`Denis Awany: denis.awany@uct.ac.za`

The full curriculum can be found on the "Syllabus" page.

Happy coding!

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

2 Syllabus

2.1 Description

This page serves as a syllabus for the SATVI Computational Course. Details for each session can be found on their dedicated page.

2.2 Module 1: Intro to R and MaRcus Training Course

2.2.1 Session 1: Intro to R and swirl

Topic: Introduction to R language and environments, RStudio, and swirl self-teaching tools.

Instructor: Monika Looney: monika.looney@uct.ac.za

Date: 05 MAR 2024

Time: 10h30 - 11h30

Location: Lekgotla 4A and 4B

Homework: Complete swirl "R Programming" interactive learning sessions at own pace.

2.2.2 Session 2: MaRcus Training Course lesson 1

Topic: Importing data into R environment and basic visualizations with ggplot2

Instructor: Carly Young-Baile: carly.young-bailie@uct.ac.za

Date: 19 MAR 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: See assignment from MaRcus R Training Course

2.2.3 Session 3: MaRcus Training Course lesson 2

Topic: Creating histograms and statistical summaries; combining and exporting plots

Instructor: Carly Young-Baile: carly.young-bailie@uct.ac.za

Date: 26 MAR 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: See assignment from MaRcus R Training Course

2.2.4 Session 4: MaRcus Training Course lesson 3

Topic: Basic data transformation using dplyr

Instructor: Carly Young-Baile: carly.young-bailie@uct.ac.za

Date: 02 APR 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: See assignment from MaRcus R Training Course

Note - MaRcus Training Course lesson 4 was skipped as it covers R Markdown which will be replaced by a session on Quarto later.

2.2.5 Session 5: MaRcus Training Course lesson 5

Topic: Continuation of data transformation using dplyr and data wrangling

Instructor: Carly Young-Baile: carly.young-bailie@uct.ac.za

Date: 09 APR 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: See assignment from MaRcus R Training Course

2.2.6 Session 6: MaRcus Training Course lesson 6

Topic: Clean up data using tidyr

Instructor: Carly Young-Baile: carly.young-bailie@uct.ac.za

Date: 30 APR 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: See assignment from MaRcus R Training Course

2.2.7 Session 7: MaRcus Training Course lesson 7

Topic: Manipulating strings with stringr and intro to regular expressions

Instructor: Carly Young-Baile: carly.young-bailie@uct.ac.za

Date: 07 MAY 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: See assignment from MaRcus R Training Course

2.2.8 Session 8: Exporting data from R

Topic: Exporting data and plots from R in different formats including csv, pdf, and jpeg

Instructor: Monika Looney: monika.looney@uct.ac.za

Date: 21 MAY 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: Load and export data frame to csv and excel. Save plots as pdf and jpeg.

2.3 Module 2: Quarto, GitHub, and GUIs

2.3.1 Intro to Quarto

Topic: Intro to technical publishing using Quarto

Instructor: Monika Looney: monika.looney@uct.ac.za

Date: 04 JUN 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: Initialize a Quarto project for your own study; Make GitHub account, access SATVILab GitHub, and download GitHub Desktop

2.3.2 Intro to GitHub

Topic: Intro to version control using Git and GitHub

Instructor: Monika Looney: monika.looney@uct.ac.za

Date: 18 JUN 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: Set up a version controlled project; Download VS Code

2.3.3 Intro to VS Code

Topic: Intro to VS Code as an alternative GUI to RStudio and git-aware terminals

Instructor: Monika Looney: monika.looney@uct.ac.za

Date: 02 JUL 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework: Write a script in the VS Code GUI

2.4 Module 3: Statistics

2.4.1 Basic statistical tests

Topic: Computing commonly needed statistics and confidence intervals in R

Instructor: Miguel Rodo: miguel.rodo@uct.ac.za

Date: 23 JUL 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework:

2.4.2 Correlations

Topic: Computing correlation metrics in R

Instructor: Miguel Rodo: miguel.rodo@uct.ac.za

Date: 30 JUL 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework:

2.5 Module 4: Commonly needed analyses for Immunology

2.5.1 Heatmaps

Topic: Plotting and manipulating heatmaps

Instructor: Monika Looney: monika.looney@uct.ac.za

Date: 27 AUG 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework:

2.5.2 Dimensionality reduction

Topic: Understanding and conducting dimensionality reduction using PCA and UMAP

Instructor: Monika Looney: monika.looney@uct.ac.za

Date: 03 SEP 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework:

2.5.3 Receiver operating characteristic (ROC) curves

Topic: Understanding and computing ROC curves

Instructor: Simon Mendelsohn: simon.mendelsohn@uct.ac.za

Date: 17 SEP 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework:

2.5.4 Background subtraction

Topic: Learning how to apply a function for background subtraction

Instructor: Miguel Rodo: miguel.rodo@uct.ac.za

Date: 01 OCT 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework:

2.5.5 Basic flow cytometry analysis

Topic: Plotting background subtracted frequencies and MFIs from flow cytometry data

Instructor: Munyaradzi Musvosvi: munyaradzi.musvosvi@uct.ac.za

Date: 15 OCT 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework:

2.5.6 Automatic gating

Topic: Generating inputs for and carrying out automated gating for flow cytometry data

Instructor: Munyaradzi Musvosvi: munyaradzi.musvosvi@uct.ac.za

Date: 29 OCT 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework:

2.6 Module 5: Other coding languages

2.6.1 Intro to Python

Topic: Basics of using python and applications for computational immunology

Instructor: Denis Awany: denis.awany@uct.ac.za

Date: 12 NOV 2024

Time: 11h00 - 12h00

Location: Lekgotla 4A and 4B

Homework:

3 Installations

3.1 Description

This document provides installation guides for basic programming tools.

3.2 R

R is a commonly used coding language for computational biologists and immunologists. Many software packages and analysis pipelines depend on R. R is also a computational environment used for computing and generating graphics.

To install R for Windows or Mac, follow the instructions provided by The Comprehensive R Archive Network (CRAN) found here: https://cran.r-project.org/

It is recommended to download the precompiled binary distribution appropriate for your machine.

To learn more about R, read the following introduction provided by CRAN: https://www.r-project.org/about.html

3.3 R Studio

RStudio is an integrated development environment (IDE) based on R. It provides a user-friendly option for building code and can incorporate multiple languages including python, which is also commonly used by computational immunologists.

To donwload and install RStudio Desktop, follow this link and the provided instructions: https://posit.co/download/rstudio-desktop/#download

3.4 GitHub Desktop

GitHub Desktop is a desktop application that interfaces with version-controlled code, GitHub, and other Git services. It provides a user friendly GUI where you can review changes made to code and perform Git commands. It is open source and free to use.

First sign up for a GitHub account at https://github.com

Now download and install GitHub Desktop, follow this link and the provided instructions: https://desktop.github.com. Sign in with your GitHub account login.

3.5 Visual Studio Code (VS Code)

VS Code is a text and code editor commonly used by developers. It can be used as an alternative for RStudio and supports multiple coding languages and various extensions for debugging and version control.

To download and install VS Code, follow this link and the provided instructions: https://code.visualstudio.com/download

4 swirl

4.1 Description

swirl is an interactive R package that helps you self-teach the basics of R. It is run from directly from the R console.

This session guide follows the instructions provided by swirl. Visit the following link to access the full tutorial.

You can also find the full swirl course tutorial on GitHub.

4.2 Install swirl

swirl requires R 3.1.0 or later installed on your computer. It is also recommended that you have RStudio installed which will provide a user-friendly environment to work with.

For instructions on how to install R and RStudio, visit the Installations session guide page.

Once you have downloaded R and RStudio, perform the following steps:

```
1. Open RStudio.
2. In the RStudio console, type the following where you see the command prompt > :

    install.packages("swirl")
```

4.3 Initialize swirl

Whenever you want to run swirl, you must load and initialize the package.

```
1. In the console, type the following:

    library("swirl")
    swirl()

2. Follow any prompts that come up in the console. i.e. if swirl asks "What shall I call you
```

4.4 Install an interactive course

The first time you initialize swirl, you will need to install a course.

For the SATVI Computational Course, we recommend that those who are new to coding start with "R Programming". This course will cover the basics of programming in R.

There are many courses to choose from, so those who are more advanced may opt for an intermediate or advanced course to work through in their own time. A repository with all available swirl courses can be found here: https://github.com/swirldev/swirl_courses#swirl-courses.

There is also an expansive swirl Network that expands further on open sourse interactive R lessons. You can access the Network and associated courses or become a swirl course author here: https://swirlstats.com/scn/

To install a course that is not part of the swirl course repository, type the following into the console:

```
?InstallCourses
```

4.5 Run swirl

For now, we will assume that we are starting with the basics and have chosen to install the "R Programming" course.

To run the interactive lessons:

Select a new lesson. The R Programming course offers 14 different short interactive lessons. Go through each one in order as the information from earlier lessons is required in later lessons.

4.6 Exit swirl

If at any time you need to exit a swirl lesson before it is complete, simply press the Esc key.

If you need to exit from a prompt, exit and save your work by typing: bye()

4.7 Interactive commands

While you are working in swirl, you may find that you want to skip a section that you are already comfortable with, or to work more on the current topic outside of an interactive session.

Below are some helpful commands for getting the most out of your swirl sessions:

From the R prompt (>):

```
To skip the current question: skip()

To experiment with R on your own without swirl interaction: play()

To re-initiate swirl interaction after playing: nxt()

To exit and save: bye()

To return to swirl's main menu: main()

To display these command options: info()
```

If you see a swirl output followed by … press Enter to continue.

4.8 Homework

As beginners, regular practice is critical! It is recommended that you go through one or two lessons daily to improve and retain these fundamentals.

Over the next week, in your own time, complete the 14 short interactive lessons from the "R Programming" swirl course.

4.9 FAQ

Q1: Can funcitons learned in swirl be applied when writing my own R scripts?

```
A: Absolutely! The functions that you use in swirl are all base R functions that can be used
```

Q2: If I need to use an R package, do I need to install the package each time I start a new session?

A: Nope! Once a package is installed, you do not have to re-install when you open a new R ses

5 MaRcus R Training

5.1 Description

The Marcus R Training program was developed by Hasse Walum of Emory University. The program will cover the following:

1. Importing data
2. Basic data visualization
3. Exporting and saving plots
4. Data transformation
5. R Markdown basics
6. Summarizing data
7. String manipulation and data joining

Rather than reinventing what is covered in the Marcus R Training program, we have been granted permission to use the materials for our SATVI Computational Course.

Over the next 6 weeks, we will refer to the Marcus R Training materials for our sessions.

5.2 Content access

The course and all associated resources are available at:

https://haswal.github.io/MaRcus/index.html

5.3 Homework

Please refer to the MaRcus R Training Program session guides to access your homework assignments.

5.4 FAQ

5.4.0.1 Session 1

Q1: What are the best ways to set your working directory?

A: There are a few ways to do this:

1. If you are using Mac, you can navigate to the directory you would like to work in usi

2. You can also set the working directory using point and click in RStudio. To do so, na

3. A note about setting working directories in scripts. It is good practice to avoid usi

Q2: When generating a plot using ggplot2, does the name used in the script for the row or column we want to plot have to match the col or rowname of the associated dataframe exactly?

A: Yes. The names must match exactly because R searches the dataframe for col or rownames as

Q3: What is the difference between facet_wrap() and facet_grid()?

A: Both are options that can be applied to ggplot2. facet_wrap() wraps a 1d sequence of panel

Q4: When should I specify aes globally vs. locally?

A: In general, specify aes in mapping (global) so that the specifications are applied to all

Q5: What are HEX codes?

A: HEX codes are unique alphanumeric codes assigned to specific colors. They can be used to a

Q6: What are your recommendations for using Chat GPT for help with coding?

A: Chat GPT is a quickly growing tool used by coders. It can be very helpful for designing /

5.4.0.2 Session 2

Q1: What is the difference between top and bottom windows in R Studio?

A: It can help to think of this an analogy: In R Studio, the top left (script) is your recip

Q2: Can you plot confidence intervals automatically using geom_errorbar or do you have to calculate them separately first?

A: Confidence intervals should be calculated separately.

5.4.0.3 Session 3

Q1: How can you save the contents of the R console when I finish a session?

A1: You can save the contents of the base R console using the 'sink()' function. Here you wi

For example:

```
sink("output/console_content.txt")
```

Run code of your choice

```
sink()
```

A2: If using RStudio, you can do this via point and click. Navigate to "History" in the top

Q2: How does 'filter()' work?

A: The 'filter()' function from the 'dplyr' package is used to subset data frames based on s

```
filter(.data, condition)
```

Here .data is any data frame in your environment that you want to filter. Condition needs to

```
filtered.data <- filter(original.data, original.data$frequency > 0.05)
```

filtered.data have rows with frequency > 0.05 removed.

Q3: What is the difference between a function and an operator?

A: A function is a chunk of code that is designed to perform a specific task. They typically

Alternatively, an operator is a simple symbol that is used to perform arithmetic, logical, o

Q4: Why does the 'is.na()' function work if the NA in my data frame is uppercase? Isn't it case-specific?

A: Though most things in R are case specific, is.na() isn't actually looking for the specifi

6 Exporting and Importing Data Formats in R

6.1 Description

This script will demonstrate methods for exporting and importing various data and plot formats from an R script. We will be using the built-in "iris" and "mtcars" datasets available in R. We encourage you to go through these steps with a dataset of your own and export formats that are relevant to your study. This session will cover commonly needed formats, including .xlsx, .csv, .pdf, .png, and .jpeg. However, there are many additional data formats that can be used and we recommend exploring these independently. Keep in mind that there are many different ways to do similar things in R, i.e. multiple packages to export to .xlsx. This script is intended to provide some helpful examples, but is not comprehensive.

6.1.1 Clear environment

```
ls()
rm(list=ls())
```

6.1.2 Set output directory

```
dir.create("output")
dir_save <- "output/"
```

6.1.3 Load libraries

```
library(tidyverse) # Needed for 'glimpse()'
library(openxlsx) # Needed to export data.frame to .xlsx
library(dplyr) # Needed to convert rownames to column and simultaneously delete rownames
library(rio) # Needed for 'import' function
library(readxl) # Needed for alternative method for importing .xlsx
```

6.1.4 Load datasets

We will load the built-in "iris" and "mtcars" datasets for demonstration purposes.

```
data("iris")
data("mtcars")
```

6.1.5 Examine data structure

```
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
glimpse(iris)
```

```
Rows: 150
Columns: 5
$ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
$ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
$ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
$ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
$ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s~
```

```
glimpse(mtcars)
```

```
Rows: 32
Columns: 11
$ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8,~
$ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 8,~
$ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 140.8, 16~
$ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, 180, 180~
$ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92,~
$ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3.150, 3.~
$ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 22.90, 18~
$ vs   <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,~
$ am   <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0,~
$ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3,~
$ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, 1, 1, 2,~
```

```
class(iris)
```

```
[1] "data.frame"
```

```
class(mtcars)
```

```
[1] "data.frame"
```

6.1.6 Export data to .xlsx

Here we will use dir_save to specify where we want to save our files. Alternatively, you can write out the full path to your output directory.

```
# To export a single data.frame to .xlsx

write.xlsx(iris, paste0(dir_save, "iris_data.xlsx"))


# To export multiple data.frames into different sheets, create a list of data.frames to be us

data.frames <- list('Sheet1' = iris, 'Sheet2' = mtcars)
write.xlsx(data.frames, file = paste0(dir_save, "iris_mtcars_data.xlsx"))
```

```
# Write to .xlsx including colnames and rownames for all sheets

write.xlsx(data.frames, file = paste0(dir_save, "iris_mtcars_data_colrow.xlsx"), colNames = T


# Alternatively, convert rownames from specific data.frames to a named column and export with

mtcars <- tibble::rownames_to_column(mtcars, "Model")
data.frames <- list('Sheet1' = iris, 'Sheet2' = mtcars)
write.xlsx(data.frames, file = paste0(dir_save, "iris_mtcars_data_rownamestocol.xlsx"))
```

6.1.7 Export data to .csv

```
# Let's first export iris as is and restore mtcars to its original format before exporting to

write.csv(iris, file = paste0(dir_save, "iris_data.csv"))

mtcars <- column_to_rownames(mtcars, var = "Model")
write.csv(mtcars, file = paste0(dir_save, "mtcars_data.csv"))

# You'll notice that the default for write.csv is to set col.names and row.names = TRUE

write.csv(mtcars, file = paste0(dir_save, "mtcars_data_colrowfalse.csv"), col.names = FALSE,
```

```
Warning in write.csv(mtcars, file = paste0(dir_save,
"mtcars_data_colrowfalse.csv"), : attempt to set 'col.names' ignored
```

```
# When using write.csv, colnames will still be written. If you want to eliminate colnames, us

write.table(mtcars, file = paste0(dir_save, "mtcars_data_colfalse.csv"), col.names = FALSE, r
```

6.1.8 Import data from .xlsx

```
# Import a data.frame from a specific sheet in a .xlsx file

df.iris.xlsx <- read.xlsx(xlsxFile = "output/iris_mtcars_data_colrow.xlsx",
                          sheet = 1,
```

```
                      rowNames = TRUE)

class(df.iris.xlsx)
```

```
[1] "data.frame"
```

```
head(df.iris.xlsx)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
# A common alternative method relies on the 'readxl' package, but functions differently

df.mtcars.xlsx <- read_xlsx("output/iris_mtcars_data_colrow.xlsx",
                      sheet = 2)
```

```
New names:
* `` -> `...1`
```

```
class(df.mtcars.xlsx)
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

```
head(df.mtcars.xlsx)
```

```
# A tibble: 6 x 12
  ...1           mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4     21       6   160   110  3.9   2.62  16.5     0     1     4     4
2 Mazda RX4 W~  21       6   160   110  3.9   2.88  17.0     0     1     4     4
3 Datsun 710    22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
4 Hornet 4 Dr~  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
5 Hornet Spor~  18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
6 Valiant       18.1     6   225   105  2.76  3.46  20.2     1     0     3     1
```

```r
# Using this method, you will need to convert to a data.frame before you can set rownames

df.mtcars.xlsx <- as.data.frame(df.mtcars.xlsx)
rownames(df.mtcars.xlsx) <- df.mtcars.xlsx[[1]]
df.mtcars.xlsx <- df.mtcars.xlsx[-1]
head(df.mtcars.xlsx)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

6.1.9 Import data from .csv

```r
# Import the iris data.frame as is. Below are two alternative methods.

df.iris.csv <- read.csv("output/iris_data.csv")

df.iris.csv <- import("output/iris_data.csv")

# Import and set colnames

df.iris.csv <- read.table("output/iris_data.csv", row.names = 1, header = TRUE, sep = ",")

head(df.iris.csv)
```

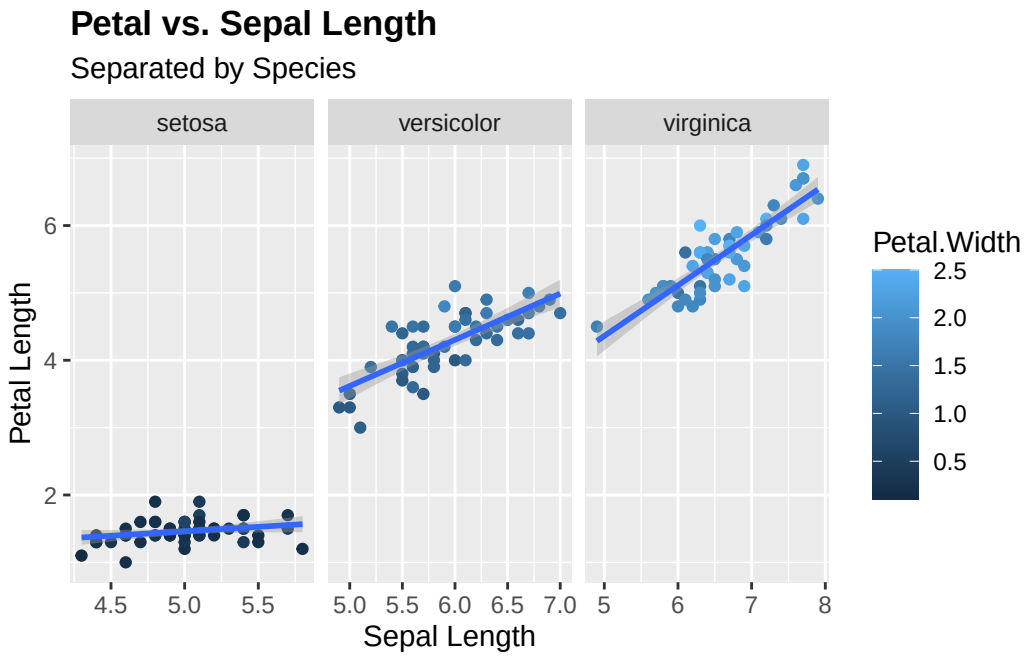
```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```r
df.mtcars.csv <- read.table("output/mtcars_data.csv", row.names = 1, header = TRUE, sep = ",

head(df.mtcars.csv)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

6.1.10 Plot data and export

```
# Create a plot and save using ggplot followed by ggsave

ggplot(data = df.iris.csv,
                mapping = aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(aes(color = Petal.Width)) +
  geom_smooth(method="lm") +
  labs(title = "Petal vs. Sepal Length", subtitle = "Separated by Species", x = "Sepal Length
  facet_wrap(~Species,
             scales = "free_x") +
  theme(plot.title = element_text(face = "bold"))
```



```
ggsave("output/iris_ggplot.pdf", width = 7, height = 7)
ggsave("output/iris_ggplot.png", width = 7, height = 7)
ggsave("output/iris_ggplot.jpeg", width = 7, height = 7)

# Alternatively, assign the plot to an object, then print and dev.off. Whereas the first meth

plot <- ggplot(data = df.iris.csv,
               mapping = aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(aes(color = Petal.Width)) +
  geom_smooth(method="lm") +
  labs(title = "Petal vs. Sepal Length", subtitle = "Separated by Species", x = "Sepal Length
  facet_wrap(~Species,
             scales = "free_x") +
  theme(plot.title = element_text(face = "bold"))

pdf("output/iris_plot.pdf", width = 7, height = 7)
print(plot)
invisible(capture.output(dev.off()))

png(filename = "output/iris_plot.png", width = 1500, height = 1500, res = 300)
print(plot)
invisible(capture.output(dev.off()))

jpeg("output/iris_plot.jpeg", width = 1500, height = 1500, res = 300)
print(plot)
invisible(capture.output(dev.off()))
```

6.1.11 Save what has been done to an .Rdata file

In some cases, it may be helpful to save a specific object or everything in your environment to an .Rdata file that can be imported all at once to be used in a different pipeline or at a later time. You can save as either an RData object or as an RDS object.

```
# To save a specific object

save(df.iris.csv, file = paste0(dir_save, "df.iris.csv.RData"))

# To save all data and values in your R environment to an RData file

save.image(paste0(dir_save, "Data_Export_Tutorial.RData"))
```

You can then load that .RData file back into R and start back up where you left off.

```
# First clear the environment so we can see how RData files are loaded

ls()
rm(list=ls())

# Now load your .RData objects

load("output/Data_Export_Tutorial.RData")
```

You can do the same thing for single objects saved as .RDS

```
saveRDS(df.iris.csv, file = paste0(dir_save, "df.iris.csv.rds"))

ls()
rm(list=ls())

# Now load your .RDS objects

reloaded_data <- readRDS("output/df.iris.csv.rds")
```

There is a workaround to save and reload an entire environment as .RDS, but it is a bit more involved and requires the use of loops, which is beyond the scope of this session. We will cover loops in a later session.

6.2 Homework

For this homework assignment, you will be using a script that you write yourself! If you have data for your own study, we suggest writing a simple script that is relevant to the analyses you will need to do. The only requirements are that you should use data that can be imported / exported in a table or dataframe format and plotted. If you do not have data of your own yet, you can use a built in dataset available from R. To find built in datasets use the following command:

```
data()
```

Now perform the following steps:

1. Clear your environment.

2. Set your working directory. This should be in a location where you perform work related to this course.
3. Set output directory. This should be a subdirectory within your working directory where you want to save any files that you generate. You can create this manually in your normal file finder or create it using R as is done in the script above.
4. Load libraries that are necessary for your script.
5. Load your dataset. Either import your own data or load one of the built in datasets.
6. Examine data structure.
7. Plot your data however you like! Refer to previous sessions for ideas and guidance.
8. Save your plots as pdf, png, and jpeg.
9. Export your data file as .xlsx and .csv. Confirm that your row and colnames are in the correct position.
10. Save a relevant object from your environment as .Rdata and .rds.
11. Load your .Rdata and .rds files back into R.
12. Consult the internet or ChatGPT and find at least one alternative method to import, export, and save your data or plots. Try these out.
13. Save your script.

7 Introduction to Git and GitHub

7.1 Description

This session will cover the basics of using Git and GitHub to create version-controlled analyses and projects. Before this session you should have set up at GitHub account and installed GitHub Desktop. For instructions, visit the "Installations" session document. It is also recommended that you install the GitHub CLI to facilitate streamlined interfacing between github and the Terminal.

7.1.1 What is Git?

Git is the most widely used version control system to date. It is free and open source. Git-based version controlling allows users to track changes that have been made to documents, code, etc. It also gives users the ability to restore version-controlled documents to earlier versions and collaborate with other developers.

Some common vocabulary:

```
1. Directory = Folder
2. Repository = Parent folder which is the top folder for your project
3. Commit = a "saved" snapshot of the repository or files within it
4. Push = upload the current version of your repository to GitHub
5. Pull = download content from a remote GitHub repository and update the local repository to
6. Clone = copy a remote GitHub repository to a local location
7. Staging area = contains information about what you will include in your next commit
8. Terminal = a.k.a. command line, interface for text commands
9. CLI = command line interface, allows certain programs to interface with the command line
```

What is the difference between Git, GitHub, and GitHub Desktop?

Git is the tool that actually tracks changes made to your code over time and allows for version control. **GitHub** is an online website that stores your Git repositories. **GitHub Desktop** is a downloaded software that allows you to work with your Git repositories locally.

7.1.2 Basic git commands:

1. git clone = copy a GitHub repository to your local machine
2. git add = add files to the git staging area
3. git commit = save files (typically with a message describing what was changed)
4. git push = upload files in the commit to a remote repository (GitHub)
5. git pull = download remote repository to local directory and update local repository with

7.1.3 Interface with git from your local machine using the Terminal

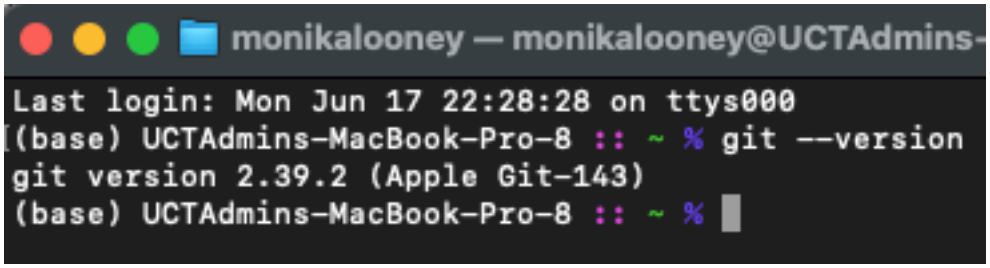
Most of the time, we use git on our local machine to develop the contents of the repo and then push them to GitHub. For this session, we will be interfacing with git from the Terminal to build an understanding of how git operates on a fundamental level. In our next session, we will cover how to develop git-controlled projects using the code editor VS Code.

1. Confirm git is installed. Open Terminal and type the following:

Note - If you can't find your Terminal, (for Mac) you can press Cmd + space to open Spotlight search. Type "Terminal". We recommend adding Terminal to your dock.

```
git --version
```

You should see a readout that looks like this:



2. Configure git

This will allow you to set your user information so it will stay consistent across all git co

```
# To set the name attached to your commits
git config --global user.name "Your Name"

# To set the email attached to your commits
git config --global user.email "youremail@email.com"
```

3. Navigate to your local working directory where you would like to store your git repository

```
cd /PATH/TO/WORKING/DIRECTORY
```

4. Make a directory for your project and initialize git.

```
mkdir [PROJECT NAME]

cd [PROJECT NAME]

git init
```

5. Create your files locally.

    - You can populate the repo with any files that you like. Be it a Quarto project, a si

    - You should always have .gitignore and README.md files in your repository. .gitignore

Note - You do not need to do this manually if creating a Quarto Project as these files are already included

Note - the .gitignore file will be automatically hidden in your file finder. If you want to find and edit your .gitignore, on a Mac, you can use the keyboard shortcut Cmd + shift + . to view it or open the repo in a code editor (this will be covered later).

```
# To create a .gitignore from the Terminal

touch .gitignore
```

```
# To create a README.md from the Terminal

touch README.md
```

It is generally good practice to now set up a basic repo structure. First edit the .gitignore to ignore any large folders or files that you do not want to track (for large analyses, consider ignorming data and output folders)

Make a directory for data. This should generally contain raw unprocessed data.

```
mkdir data
```

Make a directory for output. This should generally contain any of the processed outputs (processed data, figs, etc) generated by your code.

```
mkdir output
```

6. Check the status of your repo.

```
git status
```

7. Add your new or edited files to the staging area.

- The git staging area is an intermediate platform between working files and permanentl

```
# To add one file
git add file.txt

# To add multiple files
git add file.txt file2.csv

# To add files by pattern
git add *.text

# To add all files in the directory recursively
git add .
```

Note - Be careful when adding all files recursively, because it will also add large files if they are not specified in .gitignore

Check status again to confirm correct files have been added to the staging area.

```
git status
```

8. View unstaged changes.

- It is good practice to view the changes that have been made before committing. Norma

```
# To see staged changes
git diff --staged

# To exit, press "q"

#To see unstaged changes
git diff
```

9. If necessary, unstage files.

    - If you view your changes and notice an error (i.e. you have staged a large file that

```
# To unstage a specific file
git restore --staged file.txt

# or
git rm --cached file.txt

# To unstage multiple files
git restore -- staged file.txt file2.csv

# To unstage files by pattern
git restore --staged *.txt

# To unstage everything in the staging area while maintinging changes the files
git reset .
```

10. Commit changes

    - Once you are happy with the files in your staging area you can commit to save the cha

*Note - It is good practice to always include a short "message" with each commit that descril

```
# The -m option will include the message for your commit

git commit -m "Initial commit."
```

11. Push repo to GitHub.

To update the online remote repository, you need to push the repo to GitHub. An easy way to do this is to use the GitHub CLI. If you have not done so already, download and install.

```
# To push using GitHub CLI, authenticate GitHub
gh auth login

# Follow the prompts in the Terminal

# Create a remote repository
gh repo create

# To push the repository we created here, you must select "Push an existing local repository

?Path to local repository (.) /PATH/TO/LOCAL/REPO
```

Note - If you already have a remote repository created that is cloned to a local directory, after making changes locally, you can push the local repository to the remote repository using the following:

```
git push
```

7.1.4 Branching and Merging from the Terminal

One main feature of git repositories is the ability to create and merge different branches. This comes in handy when you have multiple people working on the same project. You can work on different branches that are dedicated to different parts of the project, and then those branches can be merged back into the master branch.

`1. Create and switch to a branch`

```
# To create a new branch called "dev"
git branch dev

# To begin working in the "dev" branch
git checkout dev

# Alternatively, do do this in one step
git checkout -b dev
```

Now you can make your changes and commit on that branch.

2. Merge the branch back into the master branch.

```
git merge dev
```

3. Now return to the main branch

```
git checkout master

# Or
git checkout main
```

7.1.5 Using git commands to navigate through git architecture from the Terminal

This diagram from unstop may be helpful for understanding how git commands can be used to move files around between local and remote directories. The webpage also gives a thorough description of how git and GitHub work.

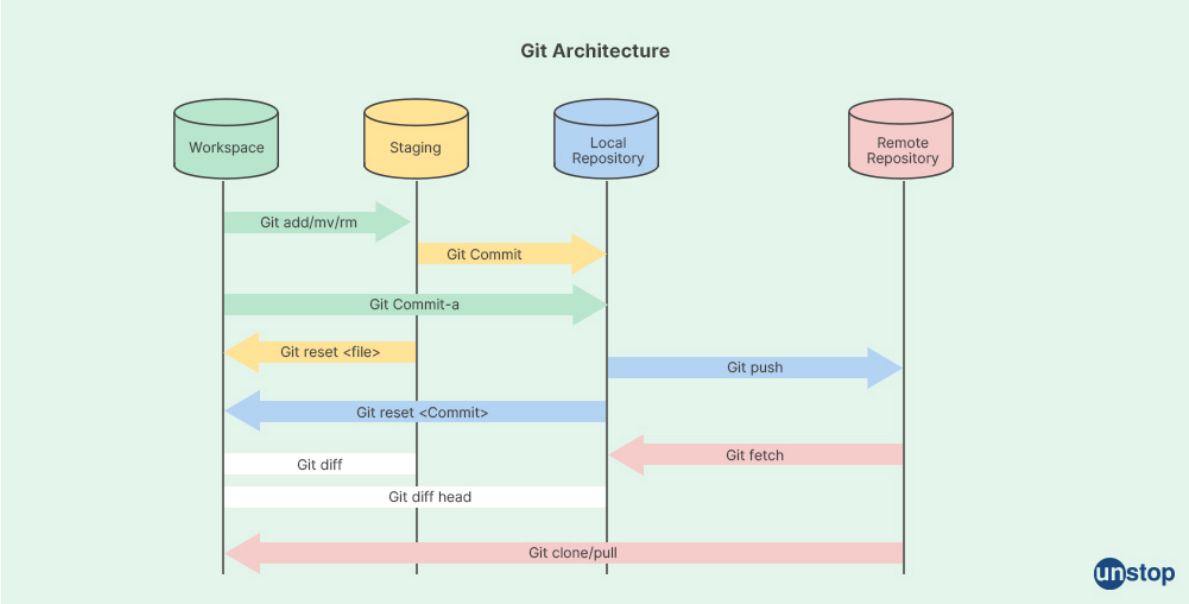
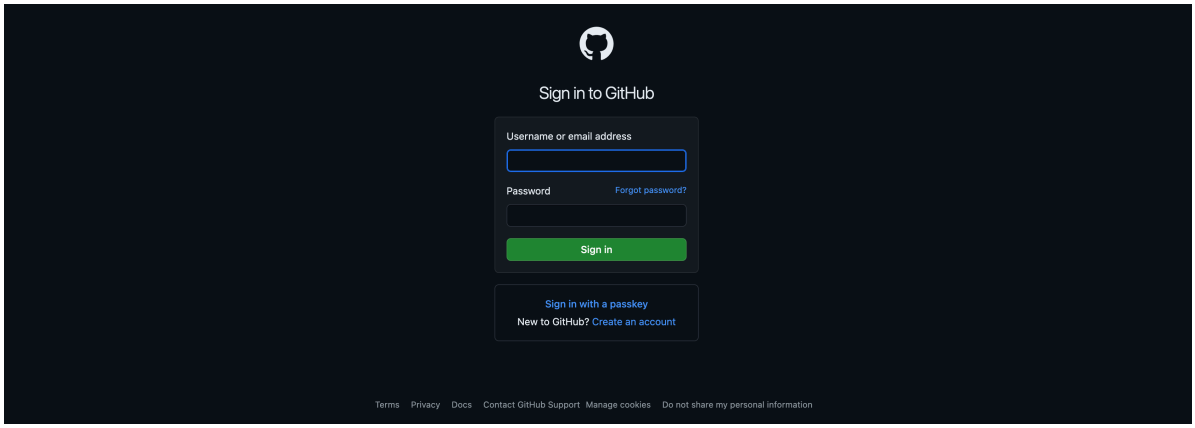


Figure 7.1: https://unstop.com/blog/what-is-git

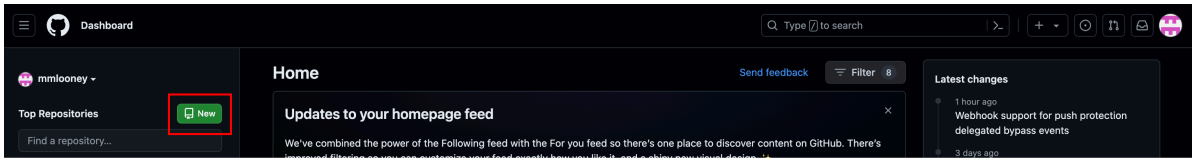
7.1.6 Create, branch, and clone a repository from GitHub

Think of a git repository as a project folder. All of the documents that you will need to run analyses for your project should be stored within the same repo.

1. Sign in to [GitHub](https://github.com/login).



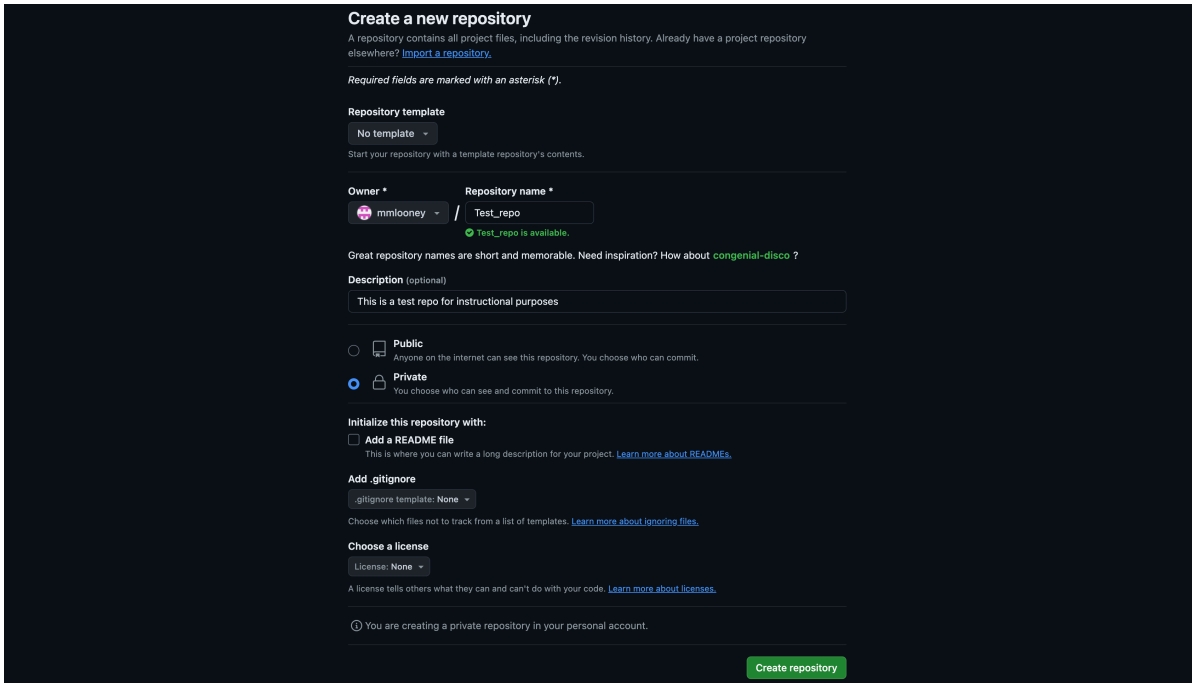
2. From your homepage, click the "New" button to create a new repository.



3. Give the repo a name and description and adjust settings if necessary.

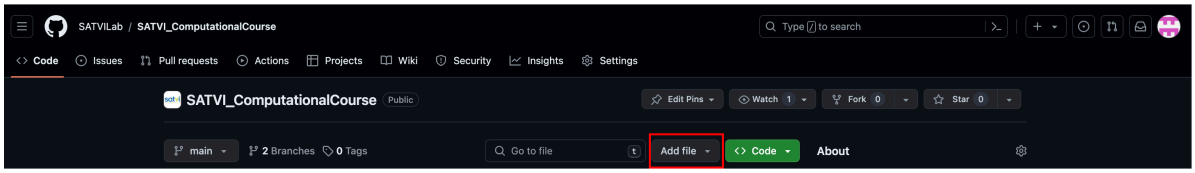
    - We typically set the repo to private until it is ready to share publicly.

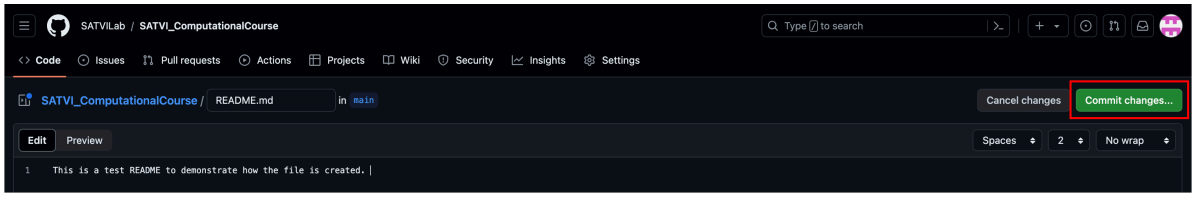
    - You can add README.md and .gitignore files here if you like, or you can add them



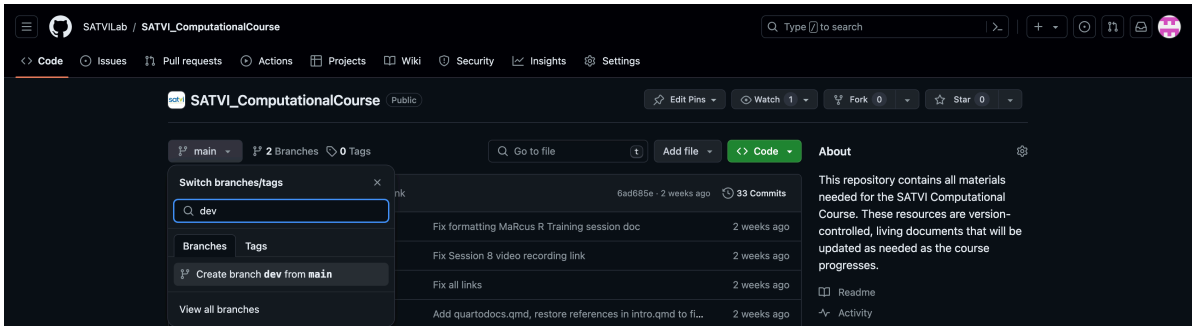
4. If necessary, add README.md on GitHub.

    - All projects MUST have a README file. This is a Markdown file that should contain a





5. Create a new branch.



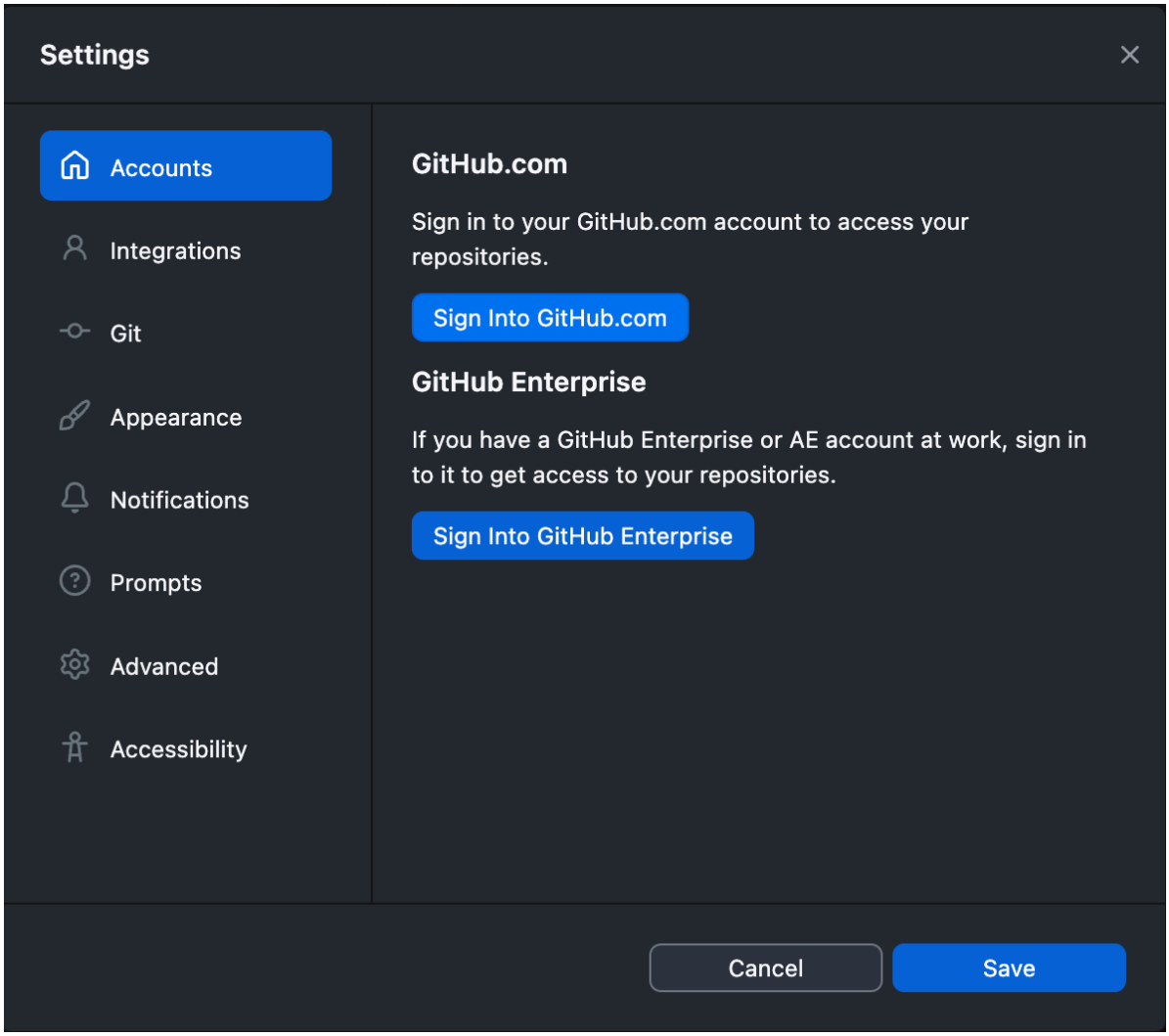
6. Clone your remote repo to your working directory.

    - There are a number of ways to do this. To clone the repo from the command line, go t

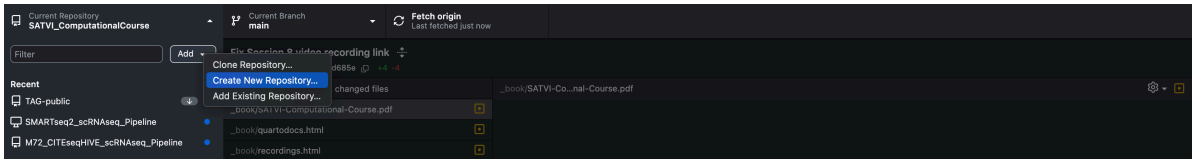
```
git clone https://github.com/SATVILab/SATVI_ComputationalCourse.git
```

7.1.7 Create, branch, and push a repository from GitHub Desktop

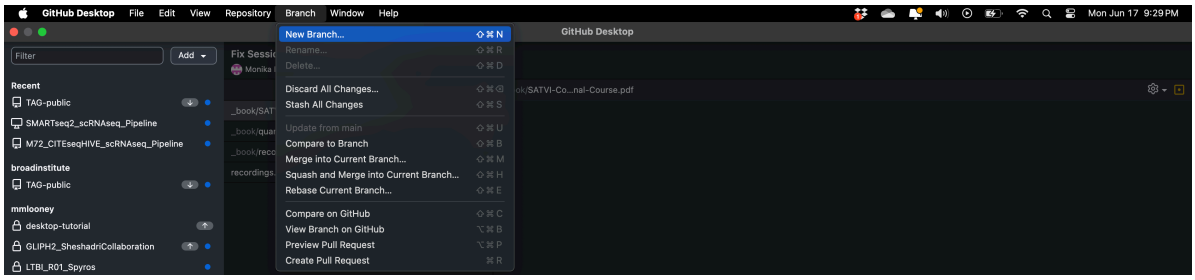
1. Open GitHub Desktop and sign in.



2. Add a repository. This repo can either be cloned from GitHub, created as a new repo, or ac



3. Branch the repo.



4. Push the repo to GitHub.

7.2 Cheatsheets

Here are two helpful git sheat sheets:

Git Cheat Sheet

GitLab Cheat Sheet

7.3 Homework

1. Read the blog post "What is GIT" from unstop

unstop “What is GIT”

2. Set up 2-Factor Authentication on your GitHub account. This will be required to access the

GitHub 2-Factor Authentication

3. Create a git repository from the Terminal and push to GitHub. This can be a test repo or,

4. Download and install VS Code.

5. If it interests you, download and framework for git-aware terminal configuration, such as

oh my zsh

8 Introduction to Visual Studio Code

8.1 Description

This session will cover an introduction to using Visual Studio Code (VS Code). VS Code has many features and functions that we will not have time to cover in this introductory session. It is very versatile and used by developers and computational biologists alike. We encourage you to explore in your own time and consider using VS Code as an additional tool in you computational kit.

8.1.1 What is VS Code?

VS Code is a commonly used code editor that incorporates many of the same features as RStudio, but allows for additional functionalities such as debugging, extensions, and version control interfacing.

What we love about VS Code - one place for everything!

VS Code allows you to create new projects and files, switch between coding languages, create, edit, debug, push, pull, and version control code all from one place.

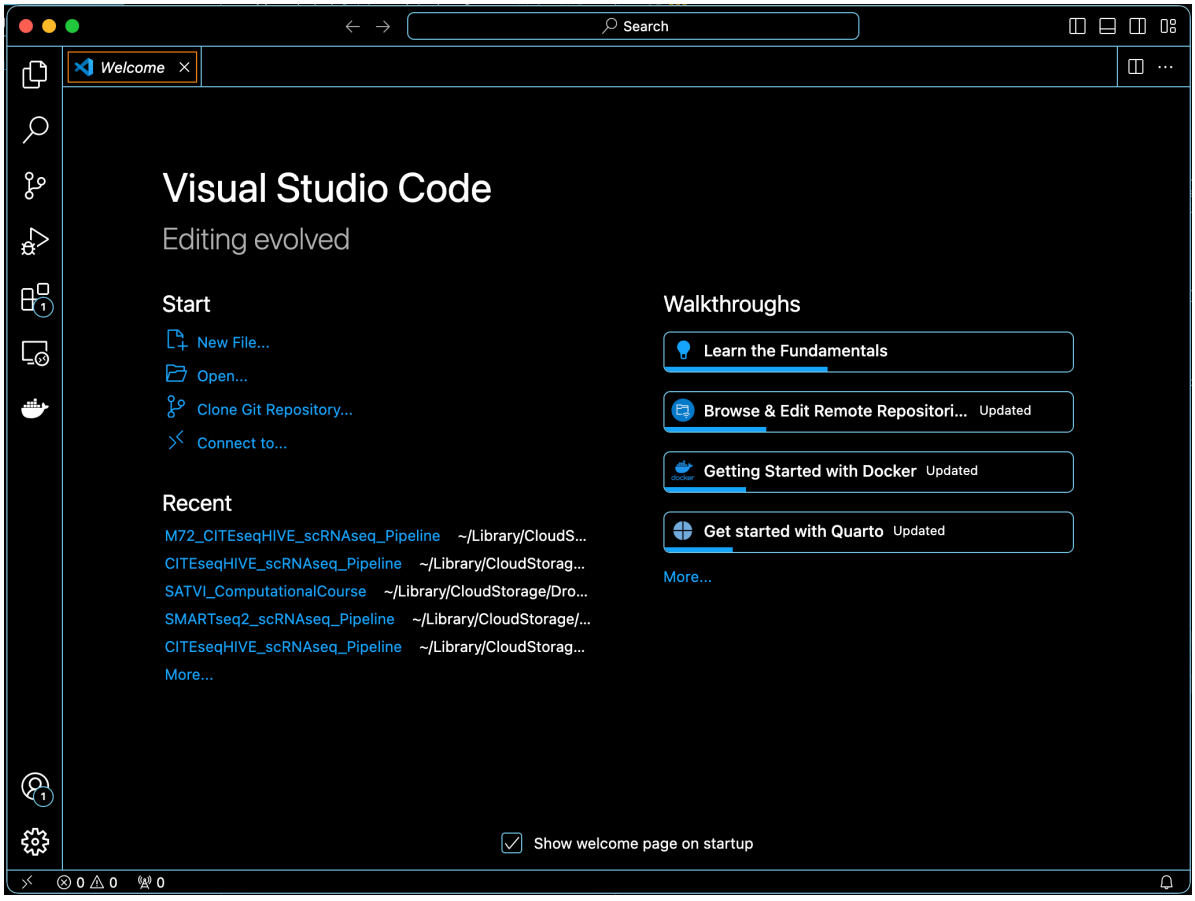
Some common vocabulary:

1. Code editor = a text editor program that is designed for editing source code and can identify code elements and errors in real time.

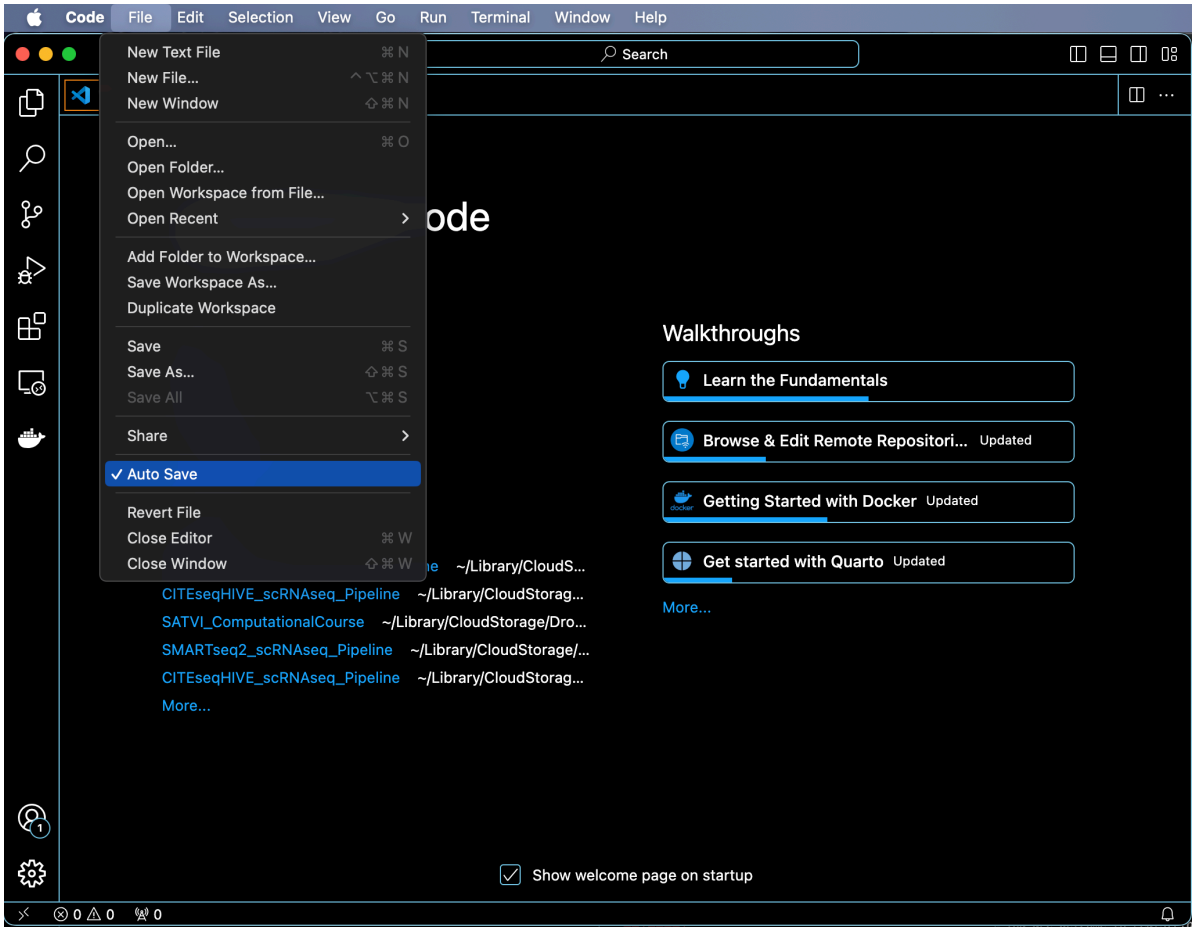
8.1.2 Initialize a project in VS Code

1. Open the VS Code desktop application. You should see the VS Code "Welcome" page.
    - From the Welcome page, you can create a new file, open an existing or recent project or file, clone a git repository, or connect to a remote development workspace. These options can also be found under the "File" tab.
    - The Welcome page also contains liks to helpful "Walkthroughs" which provide tutorials for tasks and functionalities you might find useful while you develop your code.

- Today we will create a new Quarto project, add files, and initialize git, and push to GitHub.

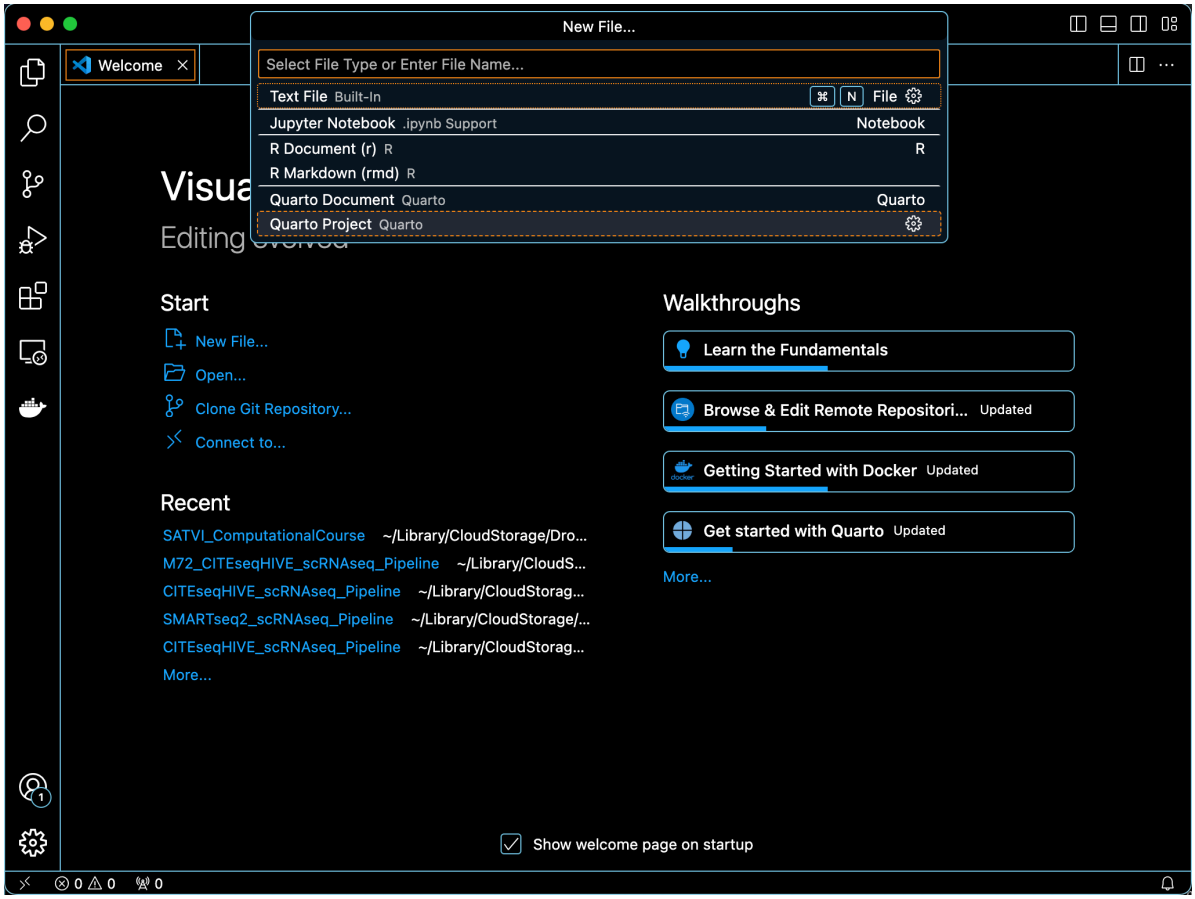


2. Consider checking "Auto Save"

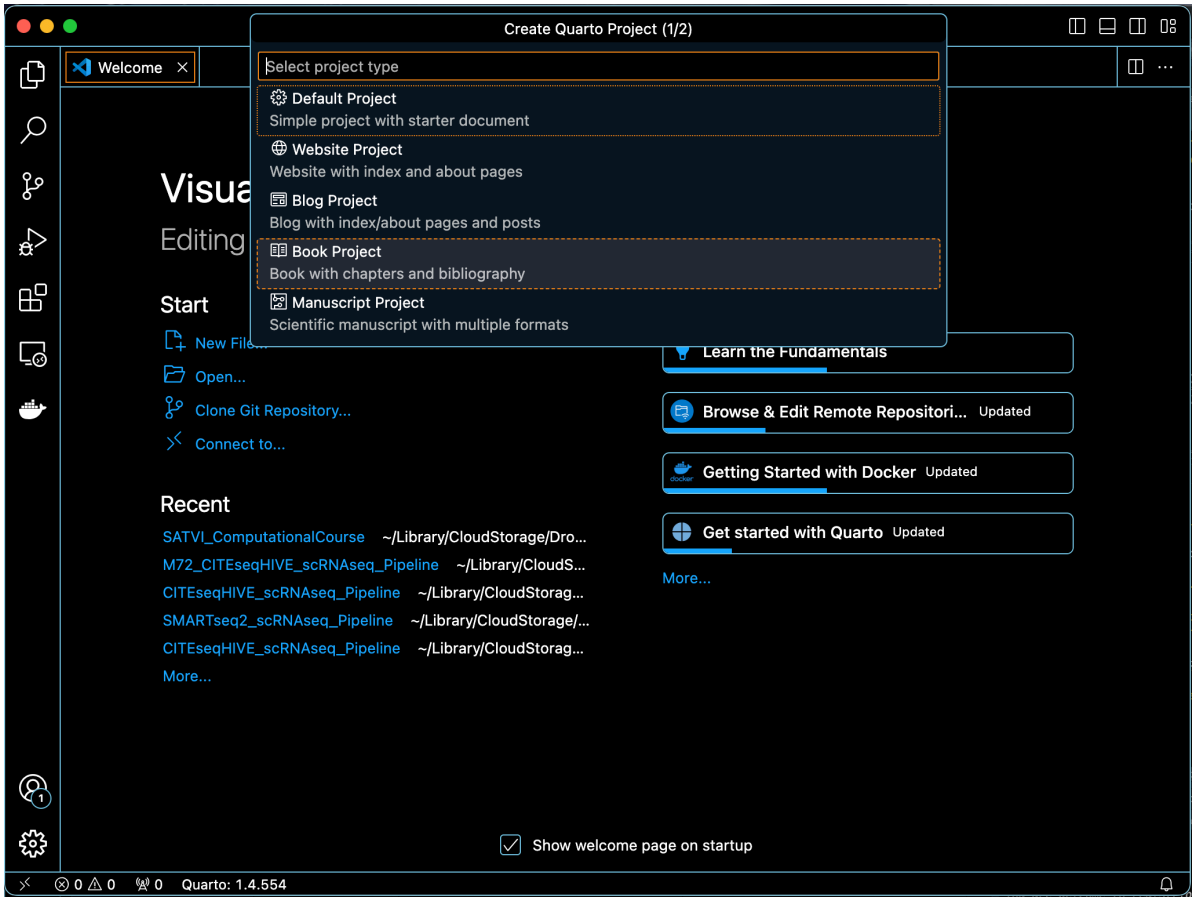


3. Create a new Quarto project

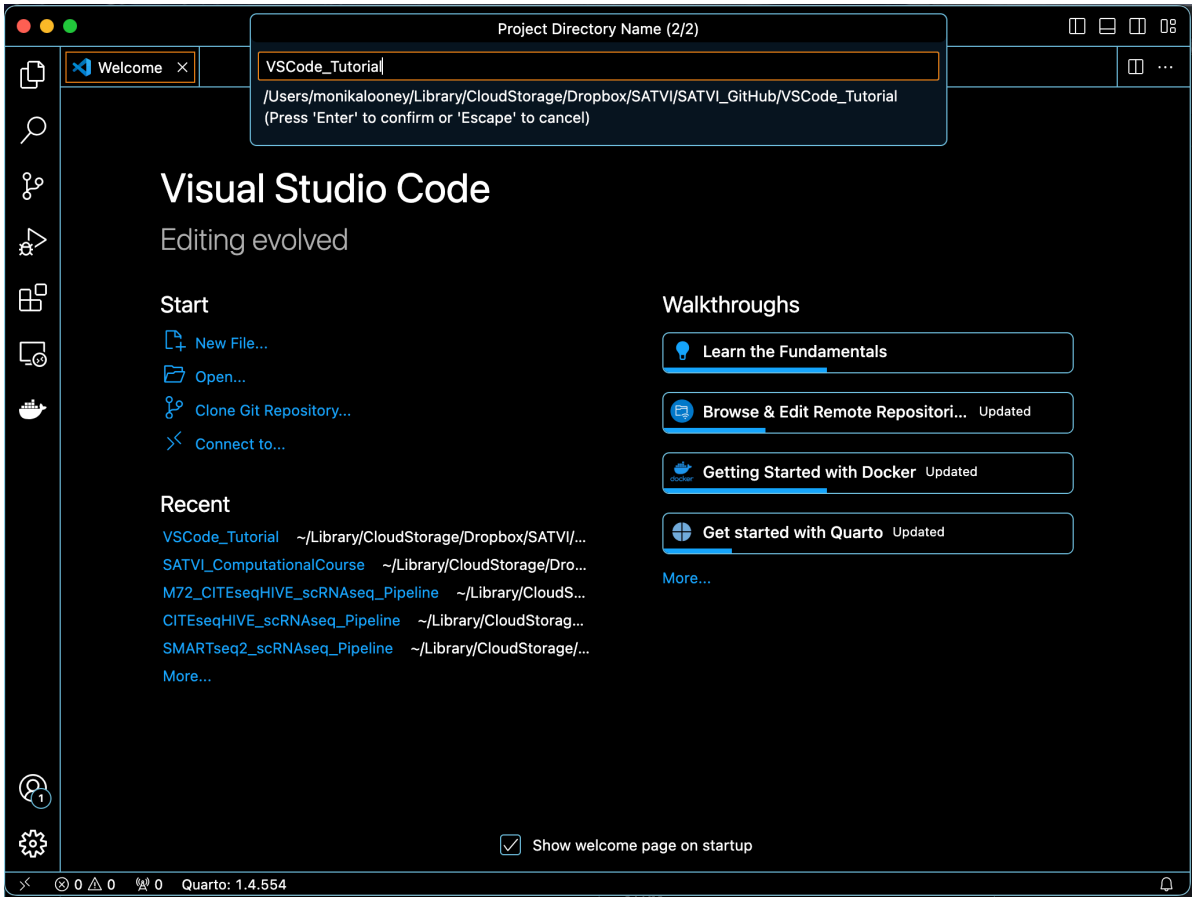
    - When you click "New File", a pop-up will open where you can select the type of file you want to generate. We will select "Quarto Project"



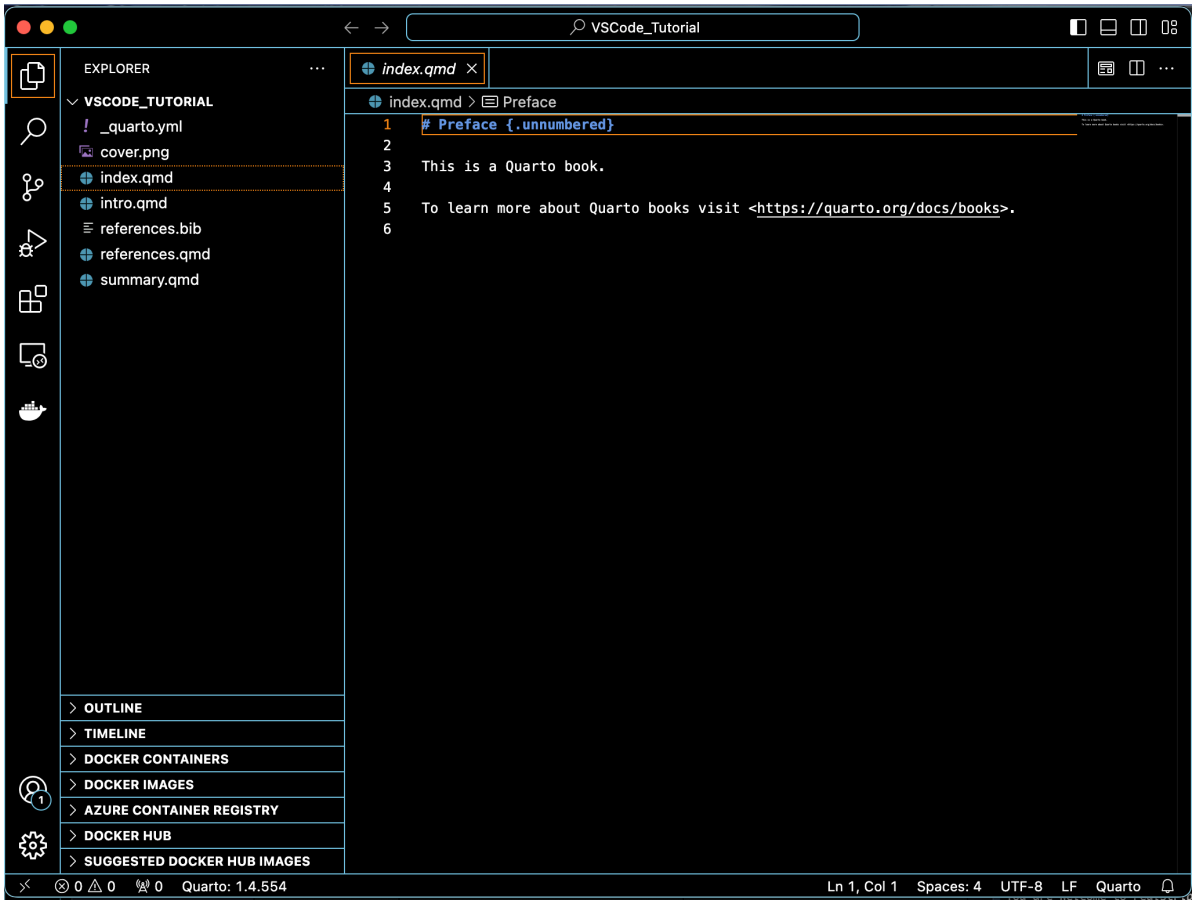
4. Next select the type of project you will want to create. We will create a Quarto Book.



5. Select the directory where you want to save your project and give it a name.



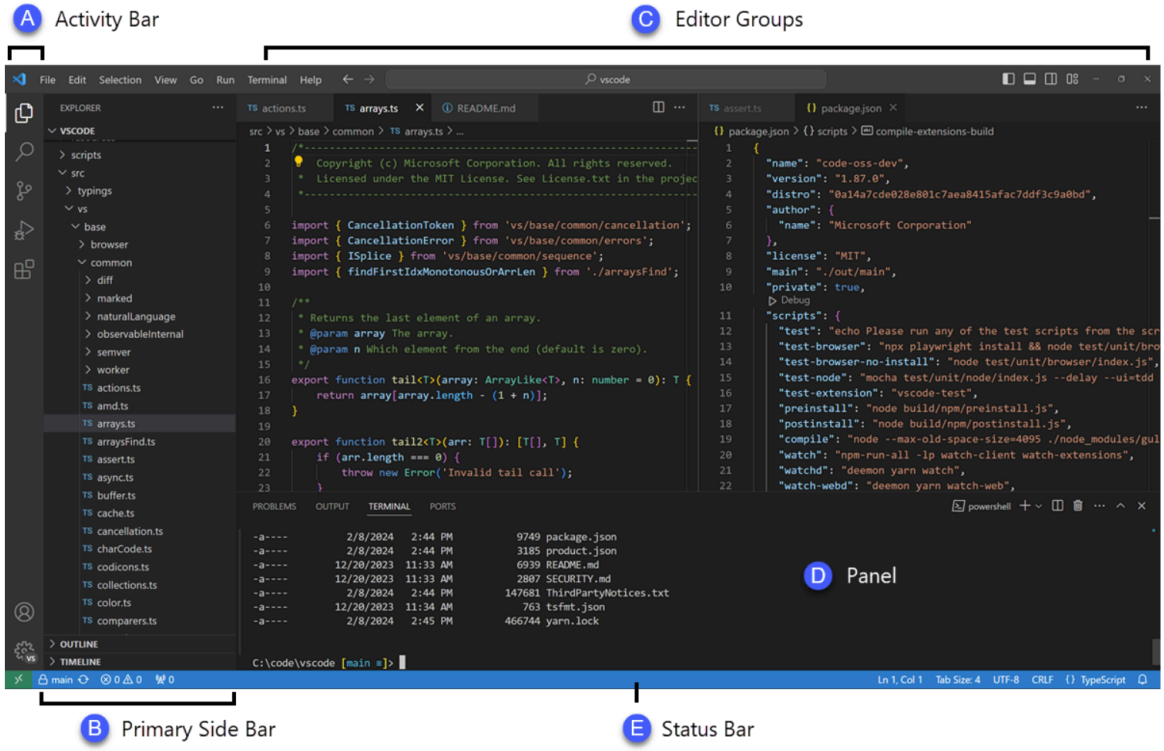
6. You will see your Quarto Book project directory and automatically generated base files appear in the left panel in VS Code.



The panels in VS Code are customizable. Some key features are:

User interface

The VS Code user interface contains all of the necessary components to develop your code. Each component is described below.



Activity Bar

Controls the view of the Primary Side Bar and houses extensions.

Primary Side Bar

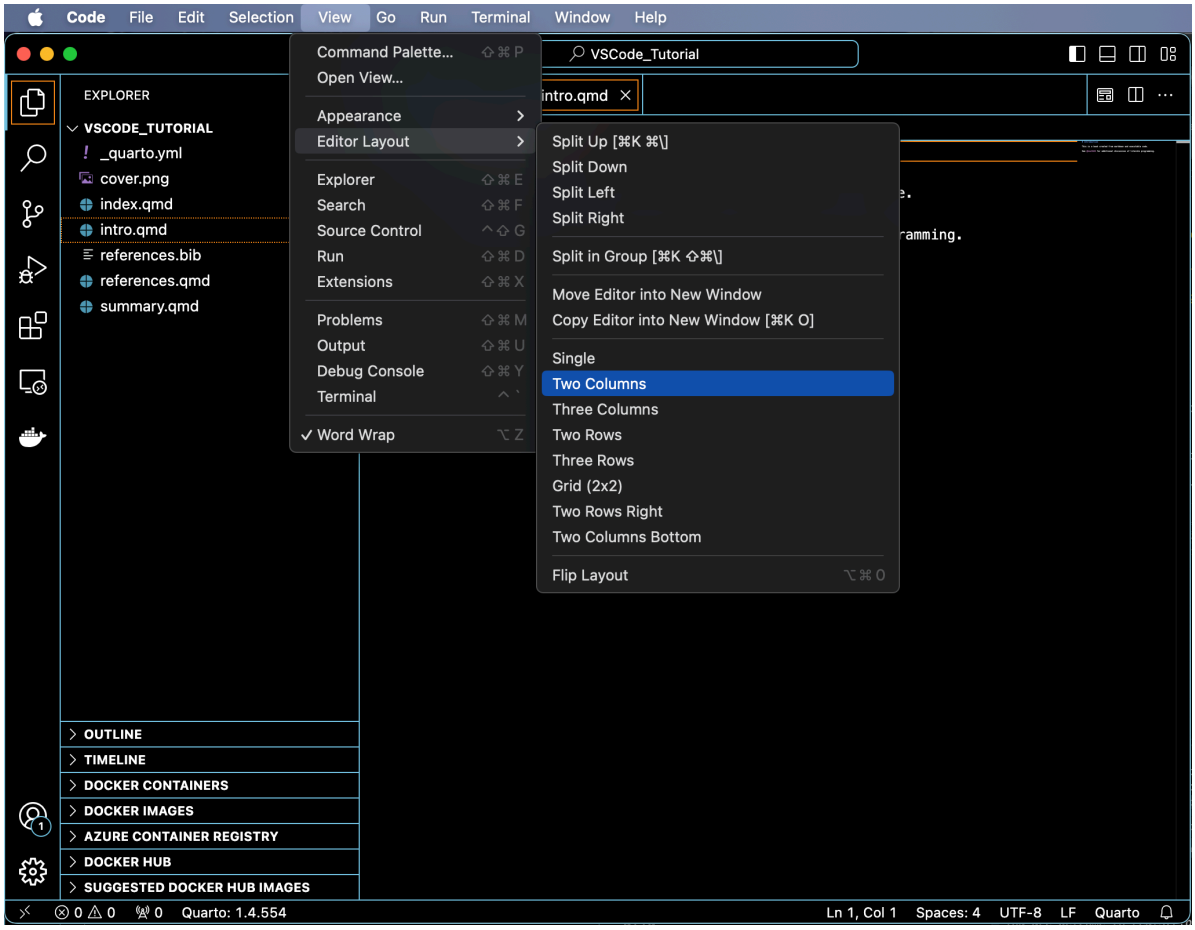
What you see here will depend on what you have selected from the Activity Bar. We often have this set to "Explorer" which shows you the files within your directory and project. The Explorer pane can also show your outline, timeline, containers, etc that are relevant for the open project. These are all collapsable.

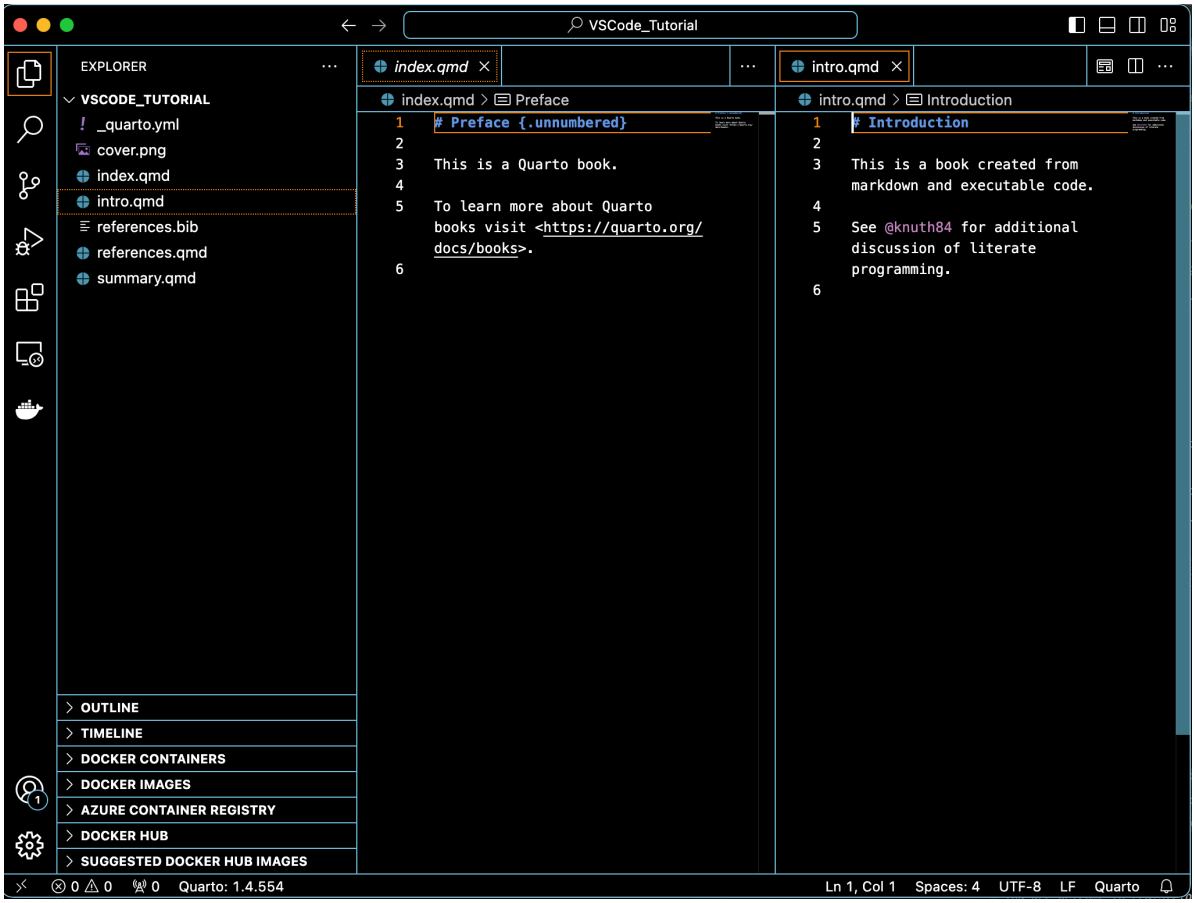
You can also use the Activity Bar to open a Search tool, Source Control, Debugging tool, Extensions, Remote Explorer (i.e. GitHub Codespaces), or Containers in the Primary Side Bar.

Editor

This is where you do your work. Like the scripts panel in RStudio, this is where you will open and edit files.

```
- Multiple panels - Change the Editor Layout from the "View" tab or from the four icons on th
```


Code File Edit Selection View Go Run Terminal Window Help
Command Palette... ⇧⌘P
Open View...
Appearance
Editor Layout
Explorer ⇧⌘E
Search ⇧⌘F
Source Control ^⇧G
Run ⇧⌘D
Extensions ⇧⌘X
Problems ⇧⌘M
Output ⇧⌘U
Debug Console ⇧⌘Y
Terminal ^`
✓ Word Wrap ⌥Z
Split Up [⌘K ⌘\]
Split Down
Split Left
Split Right
Split in Group [⌘K ⇧⌘\]
Move Editor into New Window
Copy Editor into New Window [⌘K O]
Single
Two Columns
Three Columns
Two Rows
Three Rows
Grid (2x2)
Two Rows Right
Two Columns Bottom
Flip Layout ⌥⌘0
VSCode_Tutorial
intro.qmd
EXPLORER
VSCODE_TUTORIAL
_quarto.yml
cover.png
index.qmd
intro.qmd
references.bib
references.qmd
summary.qmd
OUTLINE
TIMELINE
DOCKER CONTAINERS
DOCKER IMAGES
AZURE CONTAINER REGISTRY
DOCKER HUB
SUGGESTED DOCKER HUB IMAGES
Quarto: 1.4.554
Ln 1, Col 1 Spaces: 4 UTF-8 LF Quarto



Panel

This is a versatile additional space where you can run code from the command line, view debugging information, background jobs etc. Importantly, this is where you can access your Terminal directly within VS Code. We will discuss this in further detail below.

Status Bar

The Status Bar provides information about the open files and project, status of git-controlled repositories, and information on if scripts are currently being run.

8.1.3 Install Extensions

When you first install VS Code, you should have no extensions. As different code will require different extensions, you will customize you list of installed extensions as you develop. You can explore available extensions directly from the VS Code Extensions pane, or you can browse the VS Code Marketplace. If you start a script that requires a certain extension, VS Code

will prompt you to install it. However, there are some basic extensions that most users will need.

To install an extension, simply click on the Extensions icon from the Activity Bar and search for your needed extension or choose from the list of recommended extensions. When you click on a desired extension it will display associated documentation in your Editor. Click "Install" and check that it appears in your "Installed" section in the Extensions pane.

Some recommended extensions are below. We have provided links to the documentation on VS Code Marketplace so you can read about these while you are in the process of setting up VS Code on your machine, but we recommend that eventually you install these directly in VS Code desktop, rather than from the links provided here:

Code Runner

R

vscodeR

Quarto

GitHub Repositories

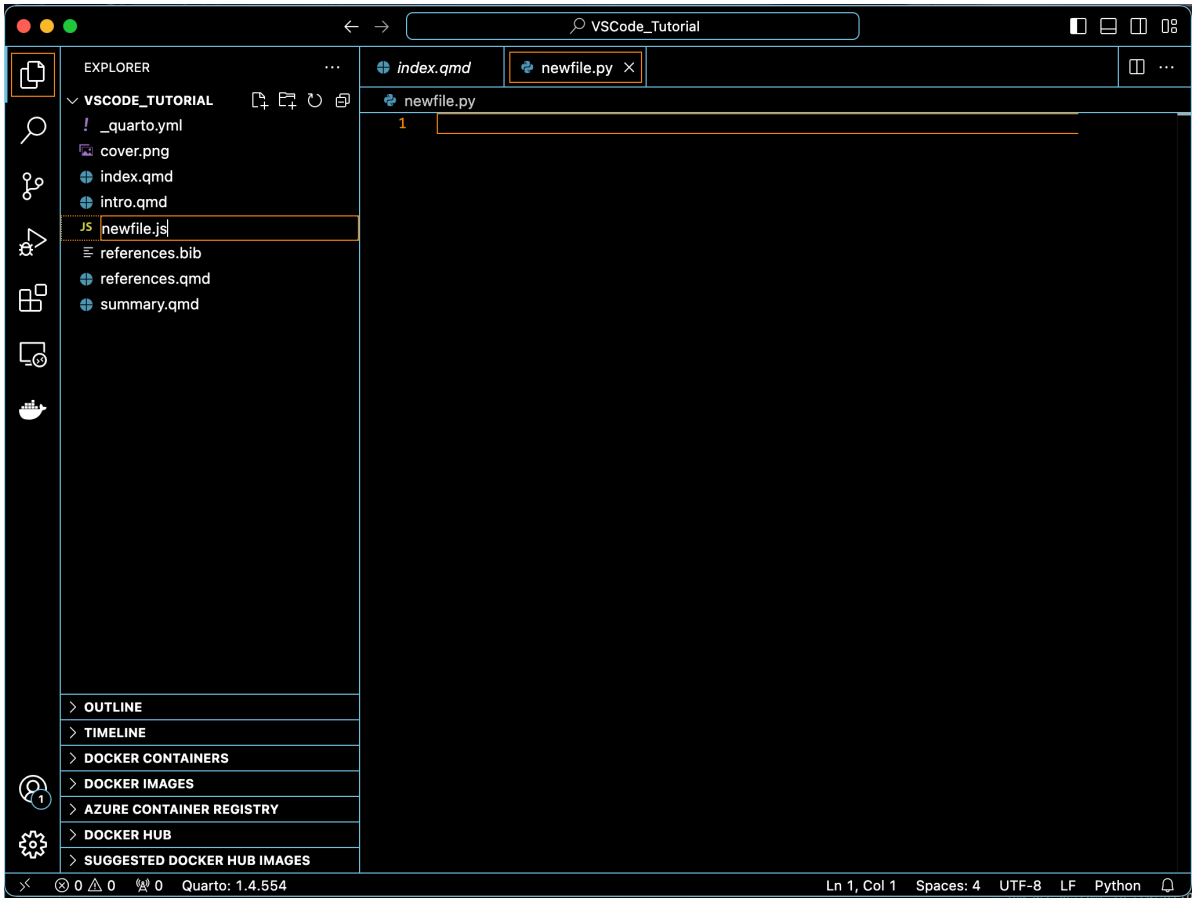
There are also some extensions dedicated to debugging which you may find useful, such as R Debugger.

Use the Extensions pane to manage and uninstall extensions as needed.

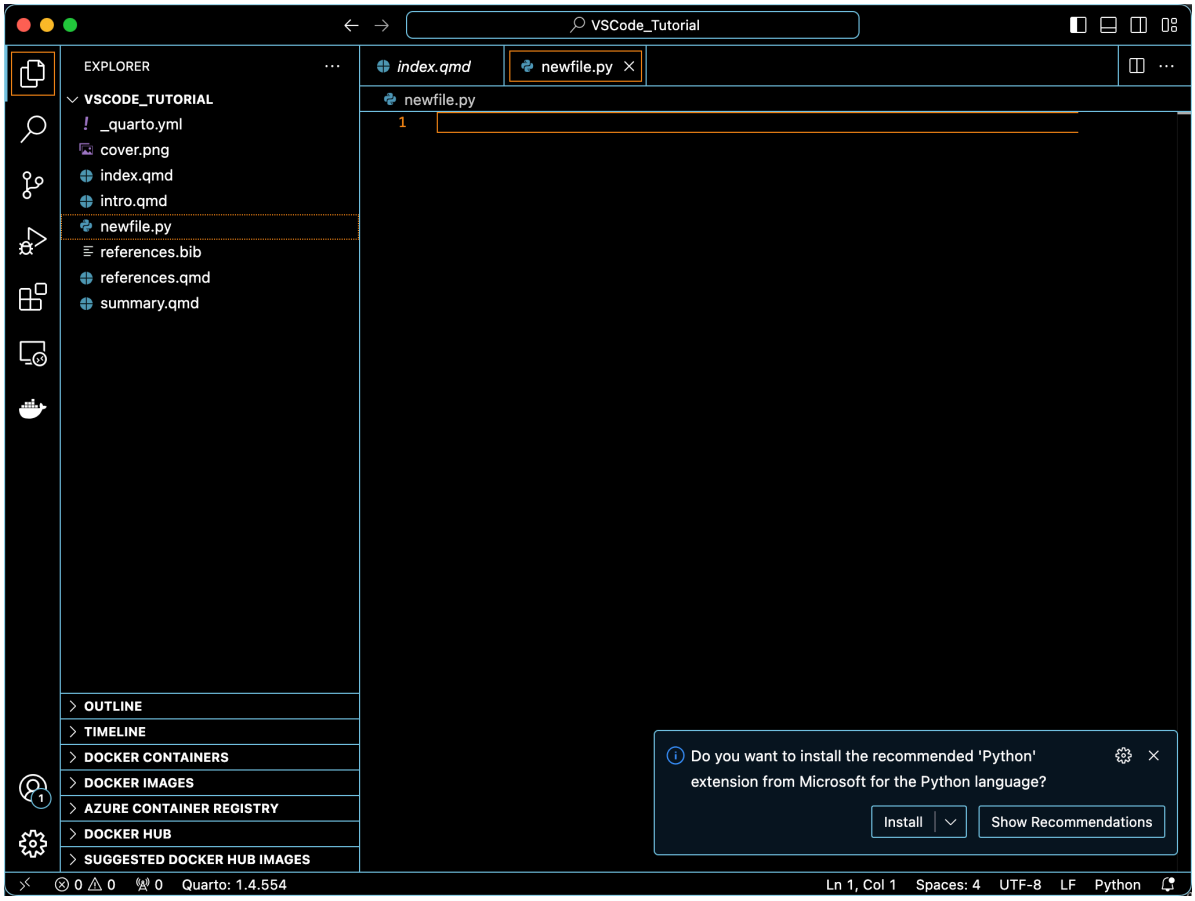
8.1.4 Manage files

To manage the files in your directory, you do not have to exit VS Code and manage files from your standard file finder. The Explorer pane in your Primary Side Bar can serve as a file finder and manager.

1. To rename a file in Explorer:
    - Right click the file and rename
2. To delete a file from Explorer:
    - Right click the file and delete
3. To change the file type in Explorer:
    - Right click and change the file extension. For example if you create a new Python file called newfile.py, but actually wanted it to be JavaScript, you can just change the file extension to .js.



Notice that if you try to create a new file that needs a certain extension to run, VS Code will prompt you to install the recommended extensions.

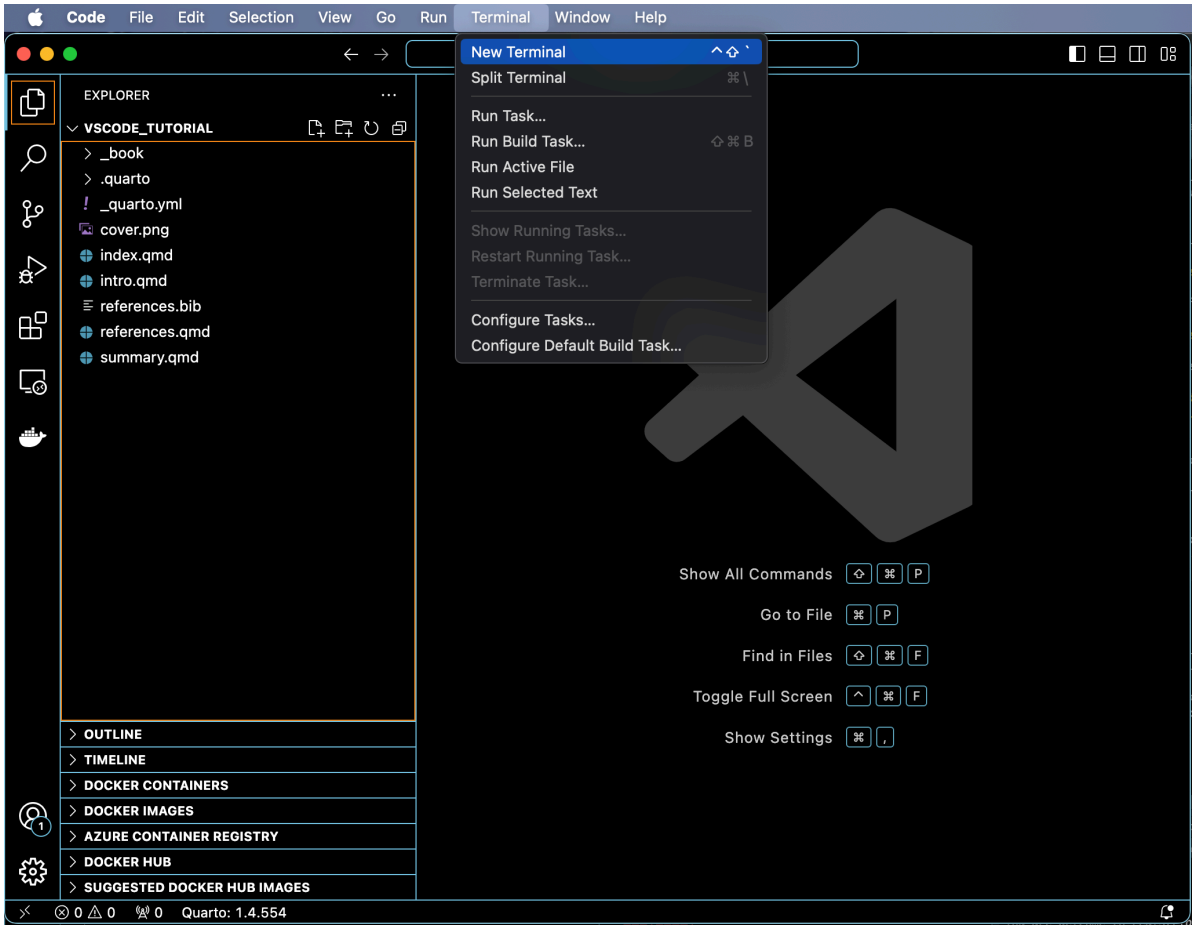


Initialize Git

One major benefit of using VS Code is that it enables integration of your code editor and terminal, streamlining the process for initializing and managing git-controlled projects.

Let's use what we learned from the last session to initialize git for our current directory from VS Code.

1. Open a new Terminal in your VS Code panel.



2. Follow the steps from our session on git and GitHub to initialize git on the project repository, add all files to the staging area, commit, and push to GitHub.

```
# Initialize git
git init

# Create .gitignore
touch .gitignore

# Create README.md
touch README.md

# Check git status
git status

# Add all files in the directory to staging area recursively
```

```
git add .

# Check git status
git status

# Make initial commit
git commit -m "Initial commit."

# Authenticate GitHub
gh auth login

# Create a remote repository
gh repo create

# Set path to existing local repository when prompted
?Path to local repository (.) /PATH/TO/LOCAL/REPO

# Push existing local repository
git push
```

8.2 Code

You can now add and edit your files however you like! Add and edit files, just remember to continue to commit and push as you go. Let's create a simple .qmd file with the code below and save it as a new file to our repository:

8.2.1 Clear environment

```
ls()
rm(list=ls())
```

8.2.2 Set output directory

```
dir.create("output")
dir_save <- "output/"
```

8.2.3 Load libraries

```
library(dplyr)
library(tidyverse)
```

8.2.4 Load dataset

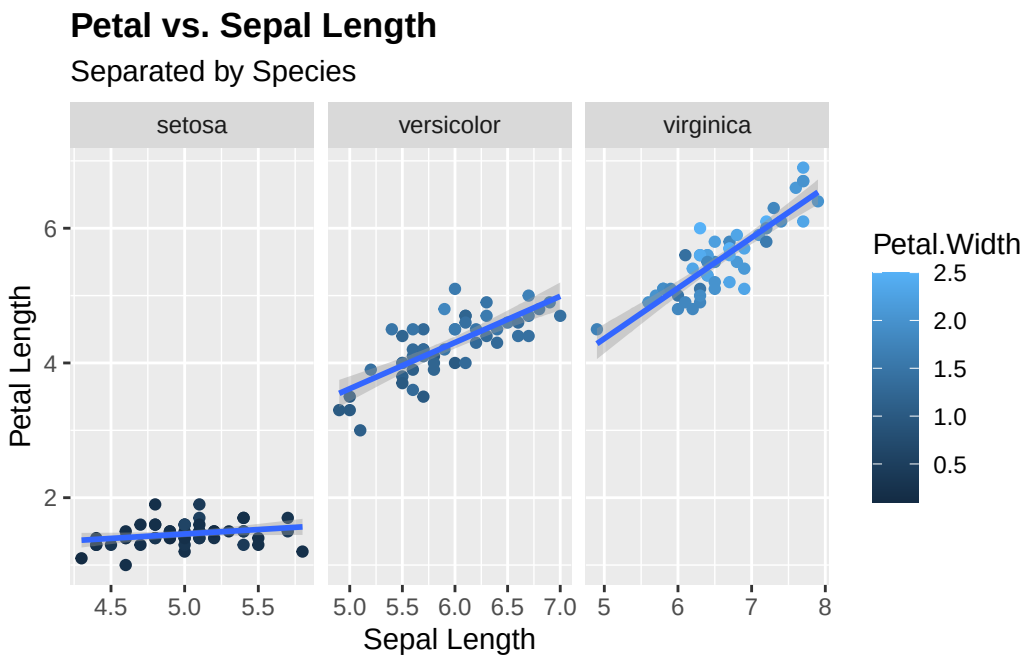
We will load the built-in "iris" R dataset and examine structure.

```
data("iris")
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

8.2.5 Plot data and export

```
ggplot(data = iris,
                mapping = aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(aes(color = Petal.Width)) +
  geom_smooth(method="lm") +
  labs(title = "Petal vs. Sepal Length", subtitle = "Separated by Species", x = "Sepal Lengt
  facet_wrap(~Species,
             scales = "free_x") +
  theme(plot.title = element_text(face = "bold"))
```




```
ggsave("output/iris_ggplot.pdf", width = 7, height = 7)
ggsave("output/iris_ggplot.png", width = 7, height = 7)
ggsave("output/iris_ggplot.jpeg", width = 7, height = 7)
```

Render the .qmd file.

8.3 Debugging

One of the key features VS Code supports is debugging. VS Code has a built in debugger that is compatible with JavaScript and TypeScript, but for other languages, you, like R, you will need to install an extension to help debug the code. For R, the debugging extension is R Debugger.

Follow the instructions in the README.md to install and deploy R Debugger.

Briefly:

1. Install R Debugger extension in VS Code.
2. Install R package vsDebugger().

```
devtools::install_github("ManuelHentschel/vscDebugger")
```

3. Open you R file in the VS Code editor pane.
4. Click the debugger from the Activity Panel so it opens in the Primary Sidebar.
5. Click F5 or the "Launch" button in the Debugger Activity Sidebar.
    - You may also want to open the DEBUG CONSOLE in your panel.
6. Click F5 or "Continue" to debug your code.

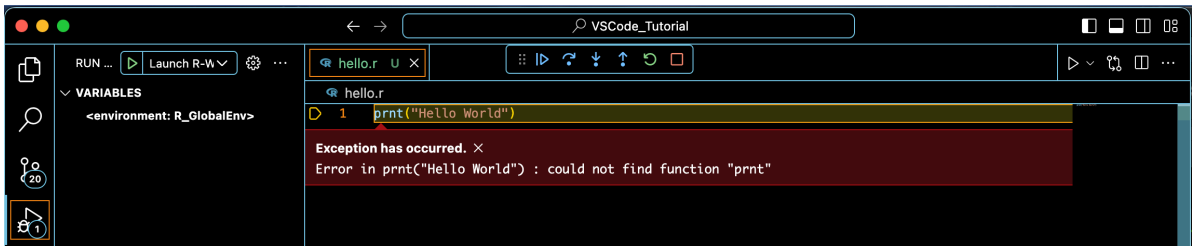
Let's try with a very simple example. Create a simple R script in your open directory:

```
print("Hello World")
```

Run the code to make sure it works. Then introduce an intentional error:

```
prnt("Hello World")
```

Now launch R Debugger and view the errors. You should see that R Debugger has identified the issue so you can fix it:



VS Code also supports extensions for AI assistants that can help debug code as you work, such as GitHub Copilot:

GitHub Copilot

GitHub Copilot allows you to interface with a ChatGPT-like AI tool to help with your code as you develop. It is a paid extension, but there is a free trial.

8.4 Cheatsheets

VS Code Keyboard Shortcuts - macOS

VS Code Keyboard Shortcuts - Windows

8.5 Homework

1. Download and install VS Code.
2. Play around VS Code.
3. Edit your git-controlled repository, commit, and push to GitHub, all within VS Code.

9 Hypothesis testing

```
# attach packages
pkg_vec <- c(
  "ggplot2", "cowplot", "tibble", "dplyr", "knitr", "remotes"
)
for (x in pkg_vec) {
  if (!requireNamespace(x, quietly = TRUE)) {
    install.packages(x)
  }
  library(x, character.only = TRUE)
}
# create clean directory to save figures to
path_dir_fig <- "images/inference"
dir.create(path_dir_fig, recursive = TRUE)
# create custom theme
theme_cowplot_custom <- function(major = "xy", minor = "none") {
  theme_cowplot() +
  theme(
    plot.background = element_rect(fill = "white"),
    panel.background = element_rect(fill = "white")
  ) +
  background_grid(major = major, minor = minor)
}
```

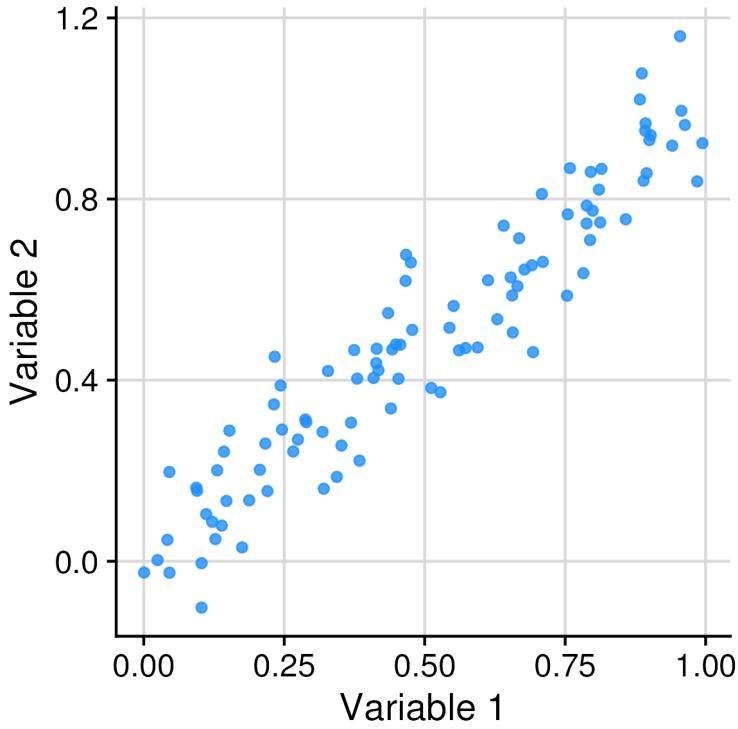
9.1 Why bother with statistics?

Suppose we have a spreadsheet with two variables:

```
set.seed(123)
x_vec <- runif(100, 0, 1)
y_vec <- x_vec + rnorm(100, 0, 0.1)
data_tbl_bother <- tibble::tibble(
  x = x_vec,
```

```
  y = y_vec
)
```

```
p_bother <- ggplot(data_tbl_bother, aes(x = x, y = y)) +
  geom_point(colour = "dodgerblue", alpha = 0.8) +
  theme_cowplot_custom() +
  labs(x = "Variable 1", y = "Variable 2")
# save plot
path_p <- file.path(path_dir_fig, "p-why-bother-init.png")
ggsave(path_p, p_bother, width = 10, height = 10, units = "cm")
```



9.1.1 Performing inference

Today, we'll talk about choosing appropriate statistical approaches to detect effects in our data.

In particular, we'll mention useful `R` functions for performing these tasks, such as the following:

```r
corr_test_obj <- cor.test(data_tbl_bother$x, data_tbl_bother$y)
corr_test_obj
```

```
    Pearson's product-moment correlation

data:  data_tbl_bother$x and data_tbl_bother$y
t = 28.992, df = 98, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9211609 0.9636474
sample estimates:
      cor
0.9463529
```

We will also discuss how to extract the key information (estimate, p-value, etc.) and present it in a more readable format:

Correlation	Confidence interval	P-value
0.946	0.921 to 0.964	0

9.2 Understanding terms

9.2.1 Hypothesis testing

- Purpose: detect differences
- Examples:
  - Is the correlation between two variables different from zero?
  - Is the effect of a drug different from a placebo?
- Primary tool for accounting for uncertainty: P-value

9.2.2 Estimation

- Purpose: obtain best estimate for a given value from the data
- Examples:
  - What is the correlation between two variables?
  - What is the effect of a drug?

- Primary tool for accounting for uncertainty: confidence intervals

::: {.content-visible when-format="html"}

Details

9.3 Relationship to the data

- P-values measure the compatibility of the data with the null hypothesis
- Confidence intervals provide a range of values that will contain the true value with a certain probability
  - For example, a 95% confidence interval [0.04, 0.5] says that 95% of the time, the true value will fall within the interval [0.04, 0.5]

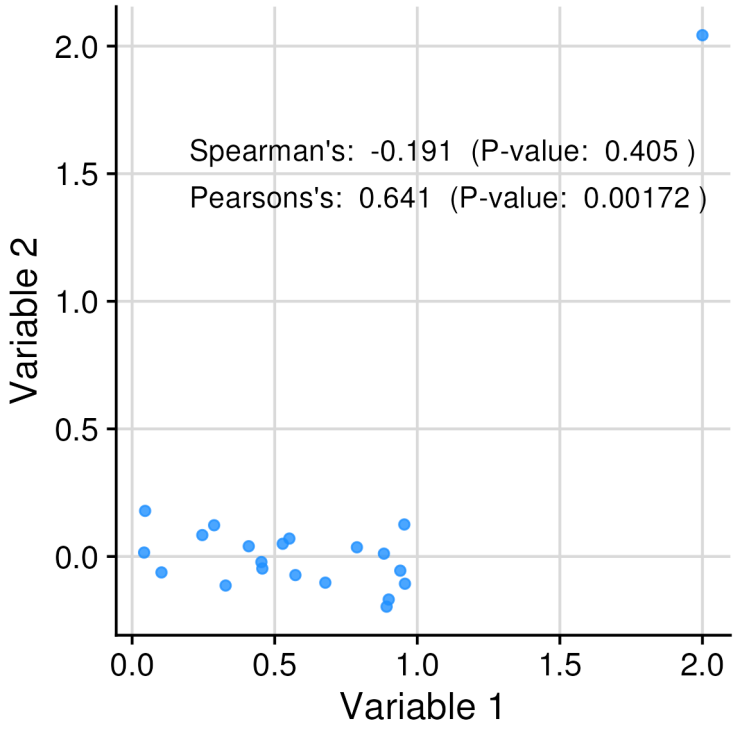
9.3.0.1 Hypothesis testing

- **Null hypothesis** (H_0): The default assumption
  - Typically, that there is no effect/difference
  - For example:
    * The correlation between two variables is zero
    * The effect of a drug is no different from a placebo
- The null hypothesis is tested against the **alternative hypothesis** (H_1)
  - For example:
    * The correlation between two variables is not zero
    * The effect of a drug is different from a placebo
- P-value:
  - The probability of observing a value at least as extreme as what we did observe, given that the null hypothesis is true

:::

9.4 The primary challenge

The main difficulty in performing inference well lies in choosing the appropriate method for the task at hand. Inappropriate choices can be disastrous. Here, for example, two different correlation coefficients will give very different results, because one's assumptions make it vulnerable to outliers:



9.5 Side-skipping the difficulties

The easiest way to avoid making erroneous assumptions is to not make any. This is the reason for the undying (and well-deserved) popularity of non-parametric methods, which make no assumptions about the underlying distribution of the data. Chief among them are:

- Correlation:
  - The Spearman rank correlation
- Hypothesis testing:
  - The Wilcoxon rank-sum test (Mann-Whitney/Mann-Whitney U/Wilcoxon-Mann-Whitney/…)

  - The Kruskal-Wallis test
- Confidence intervals:
  - The bootstrap method

For this section, we will focus on hypothesis testing.

9.6 Spearman rank correlation

The Spearman rank correlation test is a non-parametric test that assesses the strength and direction of association between two ranked variables.

It is robust against outliers because it uses *ranks$ instead of the actual values of the variables.

9.6.1 Ranks

Here is what ranks look like, as a table:

```
rank_tbl <- tibble(
  Value = data_tbl_error$x[seq_len(5)] |> signif(2)
) |>
  mutate(
    Rank = rank(Value)
  )
rank_tbl |> knitr::kable()
```

Value	Rank
0.29	1
0.79	3
0.41	2
0.88	4
0.94	5

Here's what the look like, plotted:

```r
p_error <- ggplot(data_tbl_error, aes(x = x, y = y)) +
  geom_point(colour = "dodgerblue", alpha = 0.8) +
  theme_cowplot_custom() +
  labs(x = "Variable 1", y = "Variable 2")
p_error_ranked <- ggplot(
  data_tbl_error |>
    dplyr::mutate(x = rank(x), y = rank(y)),
  aes(x = x, y = y)) +
  geom_point(colour = "dodgerblue", alpha = 0.8) +
  theme_cowplot_custom() +
  labs(x = "Variable 1 (ranks)", y = "Variable 2 (ranks)")
p_grid_error_rank <- plot_grid(
  p_error + labs(title = "Original"),
  p_error_ranked + labs(title = "Ranked"),
  nrow = 1
)
# save plot
path_p <- file.path(path_dir_fig, "p-error-rank.png")
ggsave(path_p, p_grid_error_rank, width = 14, height = 10, units = "cm")
```

9.6.2 Test

To perform the Spearman rank correlation test, we can use the `cor.test` function with the `method` argument set to `"spearman"`:

```r
cor_test_obj_spearman <- cor.test(
  data_tbl_error$x, data_tbl_error$y, method = "spearman"
)
```

Here are the results, which are quite messy:

```r
cor_test_obj_spearman
```

```
    Spearman's rank correlation rho

data:  data_tbl_error$x and data_tbl_error$y
S = 1834, p-value = 0.4054
alternative hypothesis: true rho is not equal to 0
sample estimates:
```

```
       rho
-0.1909091
```

To extract the correlation and p-value, we can use the following code:

```
corr_spearman <- cor_test_obj_spearman$estimate
corr_spearman
```

```
       rho
-0.1909091
```

```
p_value_spearman <- cor_test_obj_spearman$p.value
p_value_spearman
```

```
[1] 0.4053961
```

It is both difficult and pointless to remember the exact syntax for extracting the correlation and p-value from the `cor_test_obj_spearman` object.

Of course, one could always ask ChatGPT. Here's the answer it gave me.

But typically it is a bit quicker to just look what is in the object and extract it. When you get complicated output like when printing the `cor_test_obj_spearman` object, you can use the following code to see what is in the object:

```
corr_test_obj_spearman |> attributes()
```

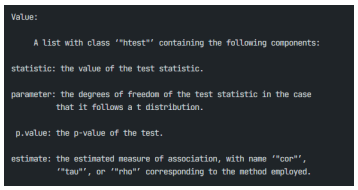
```
$names
[1] "statistic"   "parameter"   "p.value"     "estimate"    "null.value"
[6] "alternative" "method"      "data.name"

$class
[1] "htest"
```

If we are not sure what these names mean exactly, typically they are listed in the help file of the function:

```
?cor.test
```

They're under the `Value` header (would need to scroll down a bit, comes right before examples):

```
knitr::include_graphics("images/inference/help_file.png")
```




```
corr_tbl_spearman <- data.frame(
  `est` = cor_test_obj_spearman$estimate,
  `p_val` = cor_test_obj_spearman$p.value
)
corr_tbl_spearman
```

```
           est     p_val
rho -0.1909091 0.4053961
```

This is not an attractive table. We can make it more presentable with the following code:

```
corr_tbl_spearman <- corr_tbl_spearman |>
  # show only significant digits
  mutate(
    Correlation = est |> signif(3),
    `P-value` = p_val |> signif(3)
  ) |>
  dplyr::select(-c(est, p_val))
# don't display the row names
rownames(corr_tbl_spearman) <- NULL
# display using `kable` function:
corr_tbl_spearman |> kable()
```

Correlation	P-value
-0.191	0.405

Note that for the `kable` function to produce good output, you need to have the chunk option `results: asis` set.

9.6.3 Alternatives

We'll talk about the Pearson correlation coefficient and the Concordance correlation coefficient next week.

9.7 Wilcoxon rank-sum test

The Wilcoxon rank-sum test is a non-parametric test that assesses whether two independent samples come from the same distribution.

As with the Spearman's correlation coefficient, it is robust against outliers because it uses ranks instead of the actual values of the variables.

9.7.1 Example

Suppose that we have twenty samples from two groups:

```
set.seed(4)
x_vec <- rnorm(20, 0, 1)
y_vec <- rnorm(20, 0.5, 1)
sample_tbl_mw <- tibble(group_1 = x_vec, group_2 = y_vec)
sample_tbl_mw |> head()
```

```
# A tibble: 6 x 2
  group_1 group_2
    <dbl>   <dbl>
1   0.217   2.04
2  -0.542   0.665
3   0.891   1.81
4   0.596   1.79
5   1.64    1.09
6   0.689   0.217
```

We can compare if their medians (roughly speaking) are different using the Wilcoxon rank-sum test:

```
wilcox_obj <- wilcox.test(sample_tbl_mw$group_1, sample_tbl_mw$group_2)
wilcox_obj
```

```
    Wilcoxon rank sum exact test

data:  sample_tbl_mw$group_1 and sample_tbl_mw$group_2
W = 145, p-value = 0.1417
alternative hypothesis: true location shift is not equal to 0
```

Again, we don't remember where the output is:

```
wilcox_obj |> attributes()
```

```
$names
[1] "statistic"   "parameter"   "p.value"     "null.value"  "alternative"
[6] "method"      "data.name"

$class
[1] "htest"
```

We extract and format the p-value:

```
wilcox_obj[["p.value"]] |> signif(3)
```

```
[1] 0.142
```

9.8 Paired data

When we have paired data, we typically have *much* more power to detect differences.

For example, suppose we have two measurements from each of twenty people, pre- and post-treatment:

```
set.seed(4)
base_vec <- runif(20, 0, 5)
pre_vec <- base_vec + rnorm(20, 0, 0.5)
post_vec <- base_vec + rnorm(20, 1, 0.5)
paired_tbl <- tibble(
  pre = pre_vec,
  post = post_vec
)
paired_tbl |> head()
```

```
# A tibble: 6 x 2
     pre  post
   <dbl> <dbl>
1 3.21    4.01
2 0.0526  1.57
3 1.66    2.09
4 1.36    1.65
5 4.09    5.50
6 1.39    2.10
```

If we perform the (unpaired) Mann-Whitney, we don't find a significant p-value:

```
wilcox_obj <- wilcox.test(paired_tbl$pre, paired_tbl$post)
wilcox_obj$p.value |> signif(3)
```

```
[1] 0.201
```

But if we use the paired-test equivalent, it is highly significant:

```
wilcox_obj_paired <- wilcox.test(paired_tbl$pre, paired_tbl$post, paired = TRUE)
wilcox_obj_paired$p.value |> signif(3)
```

```
[1] 0.000261
```

The reason is that a lot of the "noise" (variability apart from the treatment) is removed when we use paired data. Sources of such variability include sex, age, income, etc.

9.9 Kruskal-Wallis test

The Kruskal-Wallis test is a non-parametric test that assesses whether three or more independent samples come from the same distribution.

It is a direct extension of the Mann-Whitney test to multiple groups.

9.9.1 Example

Suppose that we have add a third group to the previous example:

```r
set.seed(4)
sample_tbl_kw <- sample_tbl_mw |>
  mutate(
    group_3 = rnorm(20, 4, 1)
  )
sample_tbl_kw |> head()
```

```
# A tibble: 6 x 3
  group_1 group_2 group_3
    <dbl>   <dbl>   <dbl>
1   0.217   2.04     4.22
2  -0.542   0.665    3.46
3   0.891   1.81     4.89
4   0.596   1.79     4.60
5   1.64    1.09     5.64
6   0.689   0.217    4.69
```

We can compare if their medians (roughly speaking) are different using the Kruskal-Wallis test:

```r
kw_obj <- kruskal.test(
  list(
    sample_tbl_kw$group_1,
    sample_tbl_kw$group_2,
    sample_tbl_kw$group_3
  )
)
```

Again, we can extract and format the p-value:

```r
kw_obj[["p.value"]] |> signif(3)
```

```
[1] 1.74e-09
```

9.10 Multiple testing

9.11 Homework

1. Install the package `DataTidy23RodoSTA2005SAssignment`:

```
install_github("MiguelRodo/DataTidy23RodoSTA2005SAssignment@410a5247d600ce462e7941065233b634
```

```
data("data_tidy_yield", package = "DataTidy23RodoSTA2005SAssignment")
data_tidy_yield
```

```
# A tibble: 125 x 13
   FarmID  CropYield Rainfall TempC TempF Fertilizer   Sunlight SoilPH WindSpeed
   <chr>       <dbl>    <dbl> <dbl> <dbl> <chr>           <dbl>  <dbl>     <dbl>
 1 Farm001      23.1    150.   19.2  66.5 CornCare        10.7    4.25      2.72
 2 Farm002      28.6     86.9  16.5  61.7 KernelKindle     8.28   3.21      8.71
 3 Farm003      27.5    107.   20.1  68.2 CornCare        10.7    4.61      6.64
 4 Farm004      28.1    141.   29.9  85.8 CornCare        13.0    4.03      1.17
 5 Farm005      28.0     85.0  18.6  65.6 KernelKindle     7.05   5.69      1.40
 6 Farm006      27.6    102.   20.2  68.3 CornCare        10.7    4.81      1.98
 7 Farm007      29.5    184.   23.5  74.2 KernelKindle    10.7    3.25      2.67
 8 Farm008      23.2    170.   28.5  83.3 KernelKindle     5.81   4.16      4.83
 9 Farm009      28.1    185.   22.5  72.4 KernelKindle     7.05   3.42      4.27
10 Farm010      26.1    167.   20.0  68.0 CornCare        10.7    3.04      8.68
# i 115 more rows
# i 4 more variables: DistanceToWater <dbl>, Altitude <dbl>,
#   PesticideApplied <chr>, Irrigation <chr>
```

It contains (simulated) data on maize crop yield (`CropYield`) under various conditions.

2. Apply the appropriate test and display the results in a table for the following questions:

- Question 1. Does crop yield depend on whether pesticide was applied?
- Question 2. Does crop yield depend on the irrigation type?
- Question 3. Does crop yield depend on rainfall?
- Question 4. Apply the Bonferroni multiple comparison correction to the results of Question 1-3.

10 Inference

```
# attach packages
pkg_vec <- c(
  "ggplot2", "cowplot", "tibble", "dplyr", "knitr", "remotes", "DescTools",
  "cccrm", "ggpubr"
)
for (x in pkg_vec) {
  if (!requireNamespace(x, quietly = TRUE)) {
    install.packages(x)
  }
  library(x, character.only = TRUE)
}
if (!requireNamespace("UtilsGGSV", quietly = TRUE)) {
  renv::install("SATVILab/UtilsGGSV")
}
library(UtilsGGSV)
# create clean directory to save figures to
path_dir_fig <- "images/correlation"
if (!dir.exists(path_dir_fig)) {
  dir.create(path_dir_fig, recursive = TRUE)
}
# create custom theme
theme_cowplot_custom <- function(major = "xy", minor = "none") {
  theme_cowplot() +
  theme(
    plot.background = element_rect(fill = "white"),
    panel.background = element_rect(fill = "white")
  ) +
  background_grid(major = major, minor = minor)
}
n_obs_all <- 1e6
n_boot <- 1e2
```

11 Correlation

What does correlation measure?

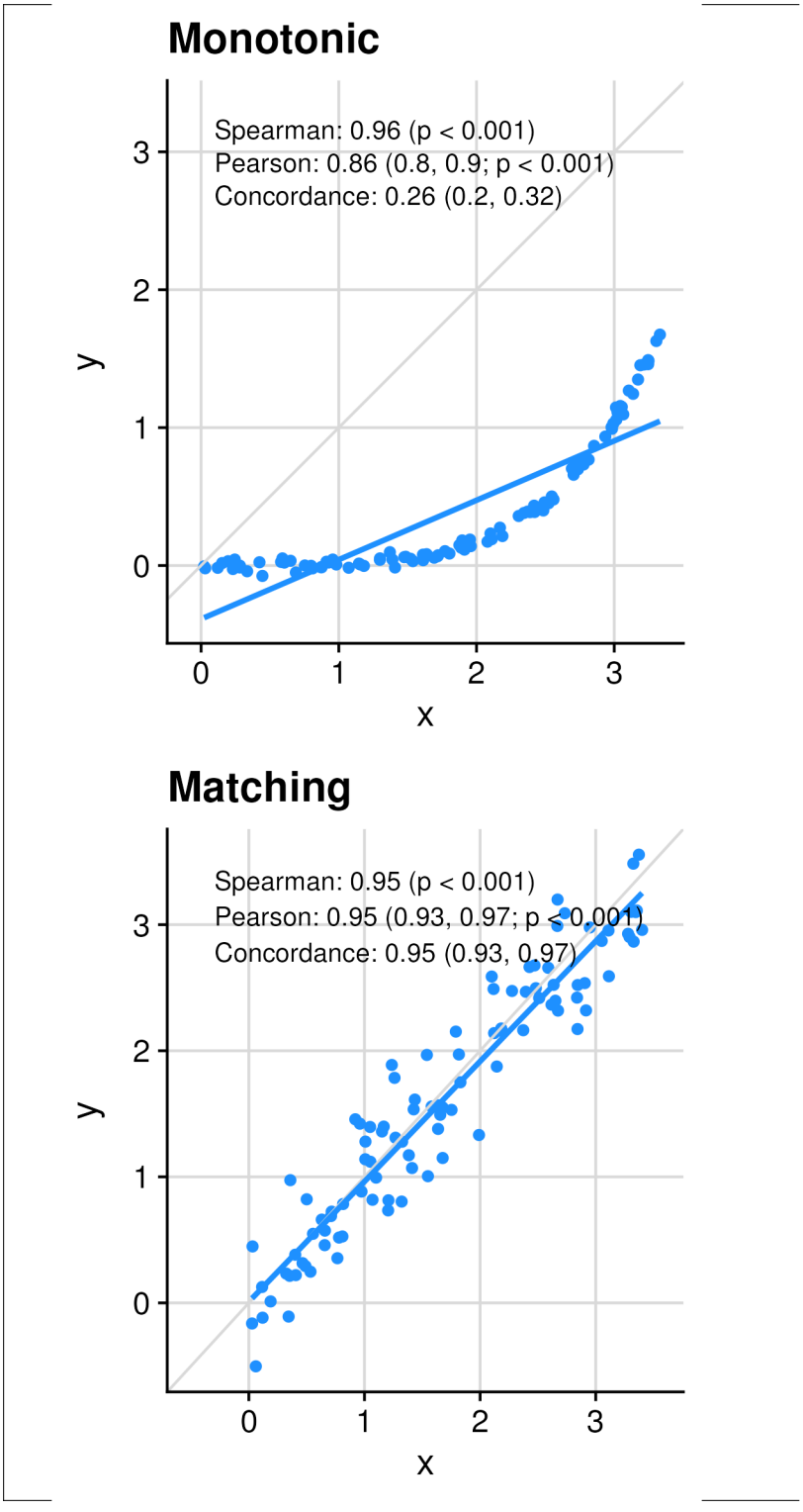
11.1 Different strokes for different folks

```
# generate data for each scenario
set.seed(4)
# Spearman: non-linear monotonic relationship
x_spearman <- runif(100, 0, 10)
y_spearman <- (x_spearman + 1)^5 + rnorm(100, 0, 3e3)

set.seed(123)
# Pearson: linear relationship
x_pearson <- runif(100, 0, 10)
y_pearson <- 2 * x_pearson + rnorm(100)
x_example <- x_pearson
y_example <- y_pearson


# Concordance: linear relationship on y = x
x_concordance <- runif(100, 0, 10)
y_concordance <- x_concordance + rnorm(100, 0, 1)

# Non-monotonic relationship
x_nonmono <- runif(100, 0, 10)
y_nonmono <- sin(x_nonmono) + rnorm(100, 0, 0.2)

data_spearman <- tibble(
  x = x_spearman, y = y_spearman, scenario = "monotonic"
  )
data_pearson <- tibble(
  x = x_pearson, y = y_pearson, scenario = "linear"
)
```

```r
data_concordance <- tibble(
  x = x_concordance, y = y_concordance, scenario = "matching"
)
data_nonmono <- tibble(
  x = x_nonmono, y = y_nonmono, scenario = "non-monotonic"
)
scenario_tbl <- data_spearman |>
  dplyr::bind_rows(
    data_pearson,
    data_concordance,
    data_nonmono
  )
```

```r
p_list <- lapply(unique(scenario_tbl$scenario), function(scenario) {
  data <- scenario_tbl |>
    dplyr::filter(scenario == .env$scenario) |>
    dplyr::mutate(id = as.character(seq_len(dplyr::n()))) |>
    dplyr::mutate(y = y / sd(y), x = x / sd(x)) |>
    tidyr::pivot_longer(
      cols = c(x, y),
      names_to = "grp",
      values_to = "value"
    )
  if (scenario %in% c("linear", "monotonic")) {
    data <- data |>
      dplyr::mutate(value = ifelse(grp == "y", value / 2, value))
  }
  UtilsGGSV::ggcorr(
    data = data,
    grp = "grp",
    y = "value",
    corr_method = c("spearman", "pearson", "concordance"),
    id = "id",
    thm = theme_cowplot_custom(),
    abline = TRUE,
    grp_to_col = "dodgerblue",
    skip = 0.07,
    font_size = 3.5,
    limits_equal = TRUE,
    est_signif = 2,
    pval_signif = 2,
    ci_signif = 2,
```

```
    point_alpha = 0.75
  ) +
  coord_equal() +
  labs(title = switch(scenario,
    monotonic = "Monotonic",
    linear = "Linear",
    matching = "Matching",
    "non-monotonic" = "Non-monotonic"
  ))
})
p_grid <- cowplot::plot_grid(
  plotlist = p_list,
  ncol = 2,
  align = "hv"
) +
  theme(
    panel.background = element_rect(fill = "white")
  )
cowplot::ggsave2(
  filename = file.path(path_dir_fig, "p-correlation_scenarios.png"),
  plot = p_grid,
  width = 20,
  height = 20,
  units = "cm"
)
```


Monotonic
Spearman: 0.96 (p < 0.001)
Pearson: 0.86 (0.8, 0.9; p < 0.001)
Concordance: 0.26 (0.2, 0.32)
Linear
Spearman: 0.99 (p < 0.001)
Pearson: 0.99 (0.98, 0.99; p < 0.001)
Concordance: 0.48 (0.41, 0.55)
Matching
Spearman: 0.95 (p < 0.001)
Pearson: 0.95 (0.93, 0.97; p < 0.001)
Concordance: 0.95 (0.93, 0.97)
Non-monotonic
Spearman: -0.043 (p = 0.67)
Pearson: -0.014 (-0.21, 0.18; p = 0.89
Concordance: -0.0073 (-0.11, 0.096)
x
y

11.2 Relationship to inference

- Hypothesis testing: we can check if the data are compatible with the correlation being zero
- Confidence intervals: we can estimate a range of plausible values for the correlation

11.3 Correlation estimation and inference in R

11.3.1 Spearman and Pearson

In this case, we use the `cor.test` function (available by default in R), as we did last week:

```
cor.test(x = x_pearson, y = y_pearson)
```

```
    Pearson's product-moment correlation

data:  x_pearson and y_pearson
t = 58.247, df = 98, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9790245 0.9904827
sample estimates:
     cor
0.985863
```

By default, the `wilcox.test` function uses the Pearson correlation:

```
cor.test(x = x_example, y = y_example, method = "pearson")
```

```
    Pearson's product-moment correlation

data:  x_example and y_example
t = 58.247, df = 98, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.9790245 0.9904827
sample estimates:
     cor
0.985863
```

We can specify the method to use the Spearman correlation:

```
cor.test(x = x_example, y = y_example, method = "spearman")
```

```
    Spearman's rank correlation rho

data:  x_example and y_example
S = 2296, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.9862226
```

The Spearman method lacks a confidence interval, in this case.

11.3.2 Concordance correlation coefficient

To compute the concordance correlation coefficient, we can use the `cccUst` function from the `cccrm` package:

```
ccc_tbl <- tibble(
  value = c(x_example, y_example),
  grp = rep(c("x", "y"), each = length(x_example))
)
ccc_obj <- cccUst(
  dataset = ccc_tbl, ry = "value", rmet = "grp", cl = 0.95
)
ccc_obj
```

```
CCC estimated by U-statistics:
       CCC  LL CI 95%  UL CI 95%     SE CCC
0.49139477 0.44480249 0.53533453 0.02310291
```

It lacks a p-value, however.

We can also use the `CCC` function from the `DescTools` package:

```
CCC(x_example, y_example)$rho.c
```

```
        est    lwr.ci    upr.ci
1 0.4913948 0.4194006 0.5572464
```

The confidence intervals, interestingly, are clearly narrower than for the `cccrm` package.

11.3.2.1 Comparing confidence interval coverage

Let's compare the actual coverage percentages.

So we'll see what percentage of time the confidence intervals actually contain the true correlation, across sample sizes and methods and nature of relationship.

11.3.2.1.1 Key functions

First, we define a function to calculate the concordance correlation coefficient on large datasets:

```
calc_ccc <- function(x, y, n_test = 1e5) {
  n_test <- 1e5
  n_est <- length(x) / n_test
  est_sum <- rep(0, n_est)
  lb_sum <- rep(0, n_est)
  ub_sum <- rep(0, n_est)
  for (i in seq_len(n_est)) {
    ind_vec <- seq((i - 1) * n_test + 1, n_test * i)
    ccc_obj <- CCC(x[ind_vec], y[ind_vec])$rho.c
    est_sum <- sum(est_sum, ccc_obj[[1]])
    lb_sum <- sum(lb_sum, ccc_obj[[2]])
    ub_sum <- sum(ub_sum, ccc_obj[[3]])
  }
  ccc_obj <- c(est_sum, lb_sum, ub_sum) / n_est
  ccc_obj
}
```

Then we'll find a function to simulate the coverage:

```
simulate_coverage <- function(x,
                              y,
                              lb,
                              ub,
                              seed,
```

```
                              n_boot = 1e3,
                              method = NULL,
                              sample_size = NULL) {
method <- if (is.null(method)) {
  c("cccrm", "z-transform", "asymptotic")
} else {
  method
}
sample_size <- if (is.null(sample_size)) {
  c(5, 10, 20, 50, 100, 200, 500, 1e3, 2e3, 5e3, 1e4)[1:4]
} else {
  sample_size
}
set.seed(seed)
sample_size_ind <- sample_size[1]; method_ind <- "z-transform"
purrr::map_df(sample_size, function(sample_size_ind) {
  print(sample_size_ind)
  purrr::map_df(method, function(method_ind) {
    print(method_ind)
    inc_vec <- purrr::map_lgl(seq_len(n_boot), function(i) {
      ind_vec <- seq((i - 1) * sample_size_ind + 1, sample_size_ind * i)
      boot_vec_x <- x[ind_vec]
      boot_vec_y <- y[ind_vec]
      boot_tbl <- tibble(
        value = c(boot_vec_x, boot_vec_y),
        grp = rep(c("x", "y"), each = sample_size_ind)
      )
      ci <- if (method_ind == "cccrm") {
        cccUst(
          dataset = boot_tbl,
          ry = "value",
          rmet = "grp",
          cl = 0.95
        )[2:3]
      } else {
        CCC(boot_vec_x, boot_vec_y, ci = method_ind)$rho.c[2:3] |> unlist()
      }
      ci_vec <- seq(ci[1], ci[2], length.out = 1e2)
      any(ci_vec >= lb & ci_vec <= ub)
    })
    tibble::tibble(
      sample_size = sample_size_ind, method = method_ind, coverage = mean(inc_vec)
```

```
      )
    })
  })
}
```

11.3.2.1.2 Calculation

We'll now calculate covarage, under three scenarios:

- A matching relationship
- A linear but non-matching relationship
- A monotonic but non-linear relationship

11.3.2.1.2.1 Matching

First, let's generate a large dataset, where can "know" the true correlation:

```
set.seed(4)
x_all_match <- runif(n_obs_all, 0, 10)
y_all_match <- x_all_match + rnorm(n_obs_all, 0, 1)
ccc_vec_match <- calc_ccc(x_all_match, y_all_match)
ccc_vec_match |> signif(4)
```

So we know where the correlation coefficient lies quite precisely. So we'll count any confidence interval that overlaps with this as correct.

```
results_tbl_match <- simulate_coverage(
  x = x_all_match,
  y = y_all_match,
  lb = ccc_vec_match[[2]],
  ub = ccc_vec_match[[3]],
  seed = 4,
  n_boot = n_boot
)
```

11.3.2.1.3 Linear

First, let's generate a large dataset, where we can "know" the true correlation:

```
set.seed(4)
n_obs_all <- 1e6
x_all_linear <- runif(n_obs_all, 0, 10)
y_all_linear <- 2 * x_all_linear + rnorm(100)
ccc_vec_linear <- calc_ccc(x_all_linear, y_all_linear)
ccc_vec_linear |> signif(4)
```

So we know where the correlation coefficient lies quite precisely. So we'll count any confidence interval that overlaps with this as correct.

```
results_tbl_linear <- simulate_coverage(
  x = x_all_linear,
  y = y_all_linear,
  lb = ccc_vec_linear[[2]],
  ub = ccc_vec_linear[[3]],
  seed = 4,
  n_boot = n_boot
)
```

11.3.2.1.4 Monotonic

```
set.seed(4)
n_obs_all <- 1e6
# Spearman: monotonic monotonic relationship
x_all_monotonic <- runif(n_obs_all, 0, 10)
y_all_monotonic <- (x_all_monotonic + 1)^5 + rnorm(n_obs_all, 0, 3e3)
y_all_monotonic <- y_all_monotonic / sd(y_all_monotonic) / 2
ccc_vec_monotonic <- calc_ccc(x_all_monotonic, y_all_monotonic)
ccc_vec_monotonic |> signif(4)
```

So we know where the correlation coefficient lies quite precisely. So we'll count any confidence interval that overlaps with this as correct.

```
results_tbl_monotonic <- simulate_coverage(
  x = x_all_monotonic,
  y = y_all_monotonic,
  lb = ccc_vec_monotonic[[2]],
  ub = ccc_vec_monotonic[[3]],
  seed = 4,
  n_boot = n_boot
)
```

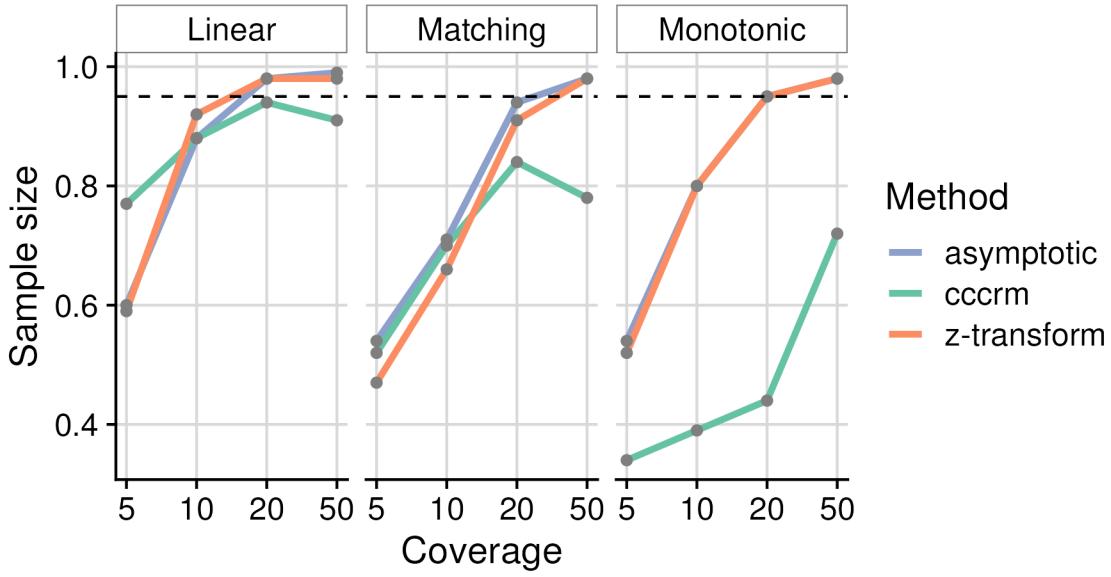
11.3.2.2 Results

Here is a plot of the results:

```
plot_tbl <- results_tbl_match |>
  dplyr::mutate(scenario = "Matching") |>
  dplyr::bind_rows(
    results_tbl_linear |>
      dplyr::mutate(scenario = "Linear")
  ) |>
  dplyr::bind_rows(
    results_tbl_monotonic |>
      dplyr::mutate(scenario = "Monotonic")
  ) |>
  dplyr::mutate(
    sample_size_num = case_when(
      sample_size == 5 ~ 1,
      sample_size == 10 ~ 2,
      sample_size == 20 ~ 3,
      sample_size == 50 ~ 4
    )
  )
p <- ggplot(
  plot_tbl,
  aes(x = sample_size_num, y = coverage, color = method)
) +
  geom_line(aes(color = method), lwd = 1.2) +
  geom_point(colour = "gray50") +
  geom_hline(yintercept = 0.95, linetype = "dashed") +
  cowplot::theme_cowplot() +
  cowplot::background_grid(major = "xy") +
  theme(plot.background = element_rect(fill = "white")) +
  theme(panel.background = element_rect(fill = "white")) +
  scale_x_continuous(labels = c(5, 10, 20, 50)) +
  facet_wrap(~scenario) +
  theme(
    strip.background = element_rect(fill = "white", colour = "gray50")
  ) +
  labs(x = "Coverage", y = "Sample size") +
  scale_colour_manual(
    values = c("cccrm" = "#66c2a5", "z-transform" = "#fc8d62", "asymptotic" = "#8da0cb"),
    name = "Method"
  )
```

```
cowplot::ggsave2(
  filename = file.path(path_dir_fig, "p-correlation_coverage.png"),
  plot = p,
  width = 15,
  height = 8,
  units = "cm"
)
```

```
knitr::include_graphics(file.path(path_dir_fig, "p-correlation_coverage.png"))
```



11.3.3 Conclusion

Always use the `DescTools` package for confidence interval estimation, as it's coverage is practically useful and consistently better than the `cccrm` package. Either the `z-transform` or `asymptotic` methods are fine.

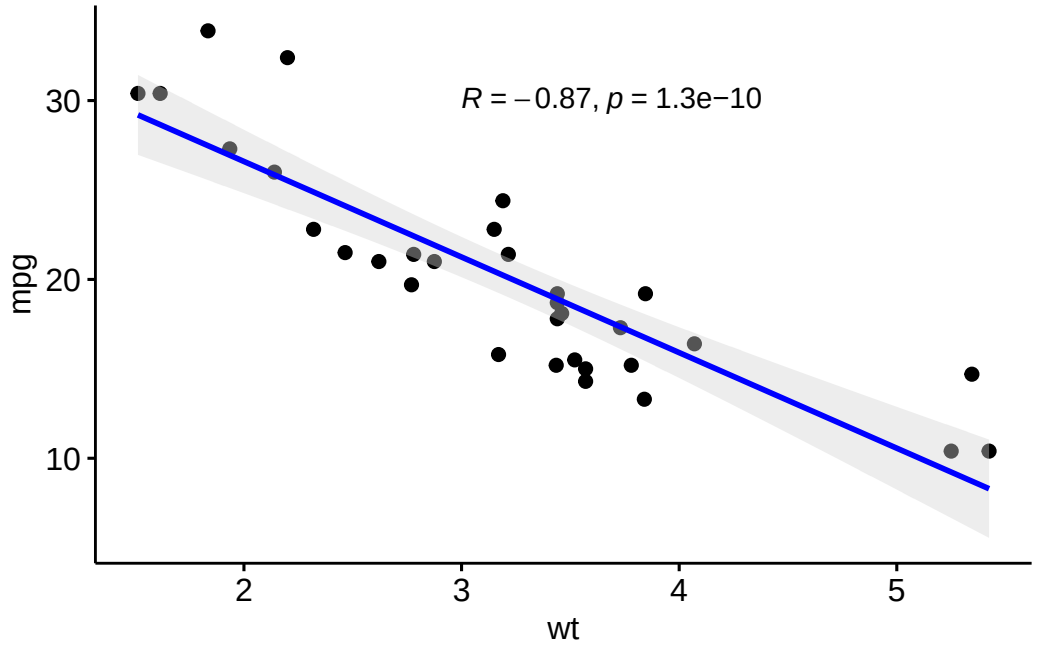
11.4 Plotting correlation coefficients

11.4.1 Straight `ggplot2`

We can plot the raw data, calculate the coefficient and add it manually:

11.4.2 ggpubr

```
# Load data
data("mtcars")
df <- mtcars

# Scatter plot with correlation coefficient
#::::::::::::::::::::::::::::::::::::::::::::::::
sp <- ggscatter(df, x = "wt", y = "mpg",
   add = "reg.line",  # Add regressin line
   add.params = list(color = "blue", fill = "lightgray"), # Customize reg. line
   conf.int = TRUE # Add confidence interval
   )
# Add correlation coefficient
sp + stat_cor(method = "pearson", label.x = 3, label.y = 30)
```



It does not support the concordance correlation coefficient, however.

11.4.3 UtilsGGSV::ggcorr

This is the function I've developed over the years to help with quality assurance at SATVI. When we want to check whether two people running the same assay on the same participants

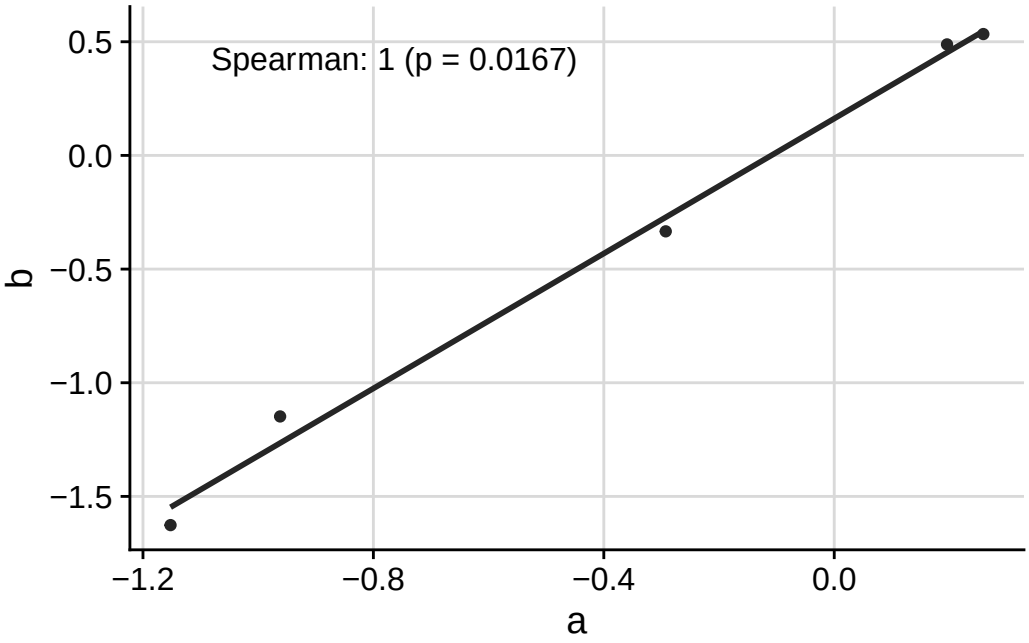
get the same result, we use the concordance correlation coefficient. However, it can also do the Spearman and Pearson correlation coefficients.

Install it using the following:

```
if (!requireNamespace("remotes", quietly = TRUE)) {
  utils::install.packages("remotes")
}
remotes::install_github("SATVILab/UtilsGGSV")
```

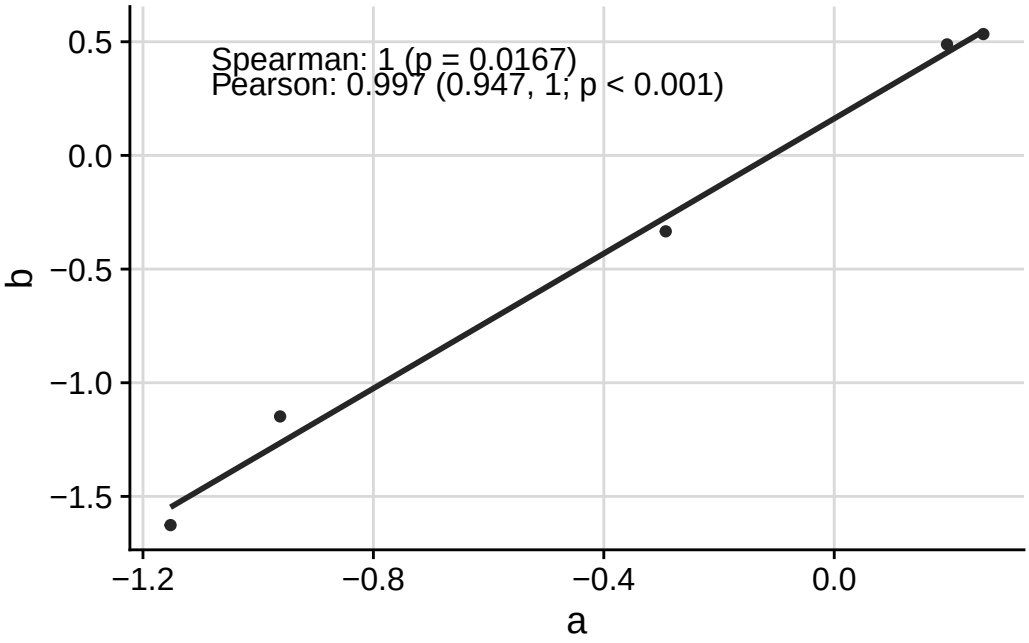
The function `ggcorr` plots correlation coefficients (see `?ggcorr` for more information):

```
set.seed(3)
response_vec_a <- rnorm(5)
response_tbl <- data.frame(
  group = rep(letters[1:3], each = 5),
  response = c(
    response_vec_a,
    response_vec_a * 1.2 + rnorm(5, sd = 0.2),
    response_vec_a * 2 + rnorm(5, sd = 2)
  ),
  pid = rep(paste0("id_", 1:5), 3)
)

ggcorr(
  data = response_tbl |> dplyr::filter(group %in% c("a", "b")),
  grp = "group",
  y = "response",
  id = "pid"
)
```



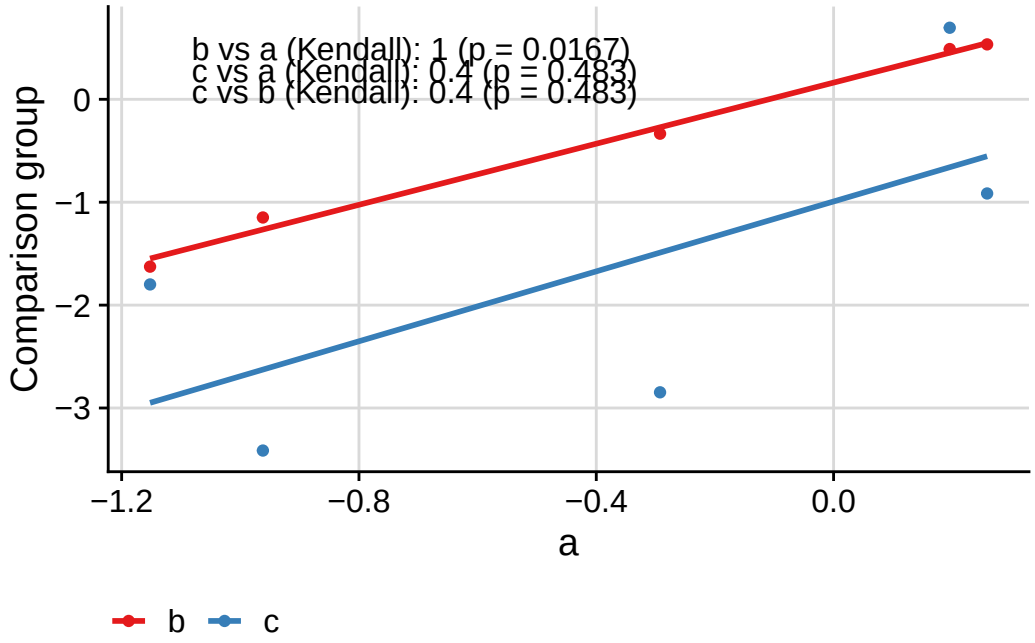
We can display multiple correlation coefficients:

```
ggcorr(
  data = response_tbl |> dplyr::filter(group %in% c("a", "b")),
  grp = "group",
  y = "response",
  id = "pid",
  corr_method = c("spearman", "pearson")
)
```



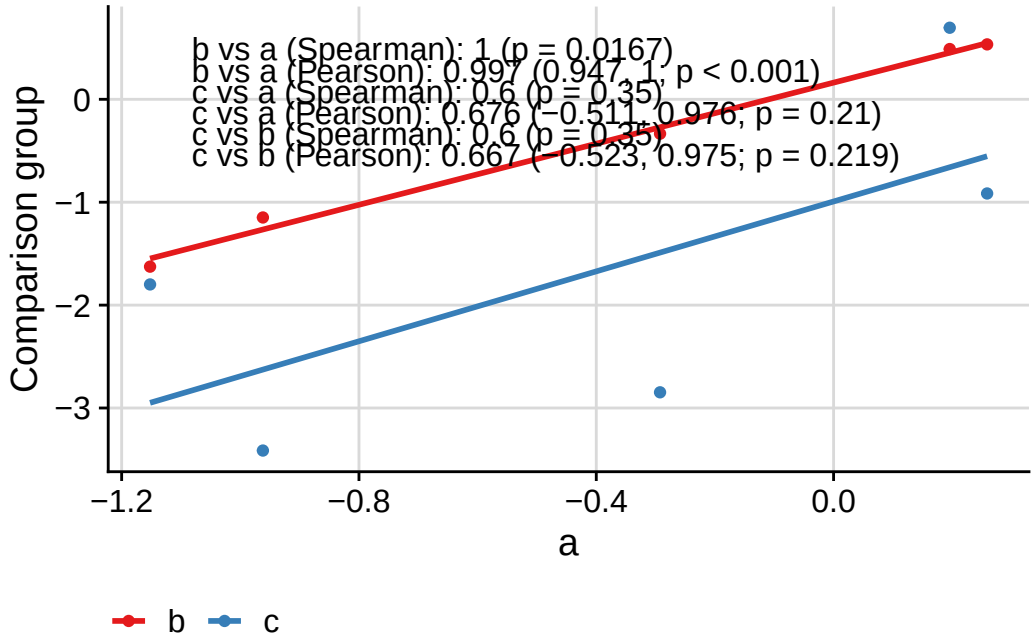
We can compare more than two groups:

```
ggcorr(
  data = response_tbl,
  grp = "group",
  y = "response",
  id = "pid",
  corr_method = "kendall"
)
```



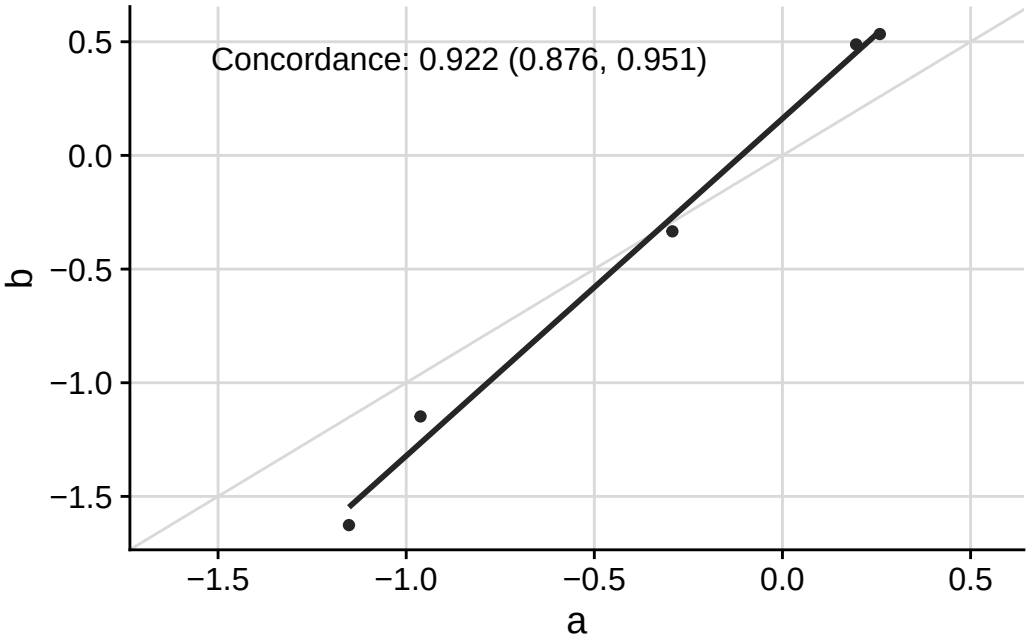
We can compare more than two groups and multiple correlation coefficients:

```
ggcorr(
  data = response_tbl,
  grp = "group",
  y = "response",
  id = "pid",
  corr_method = c("spearman", "pearson")
)
```



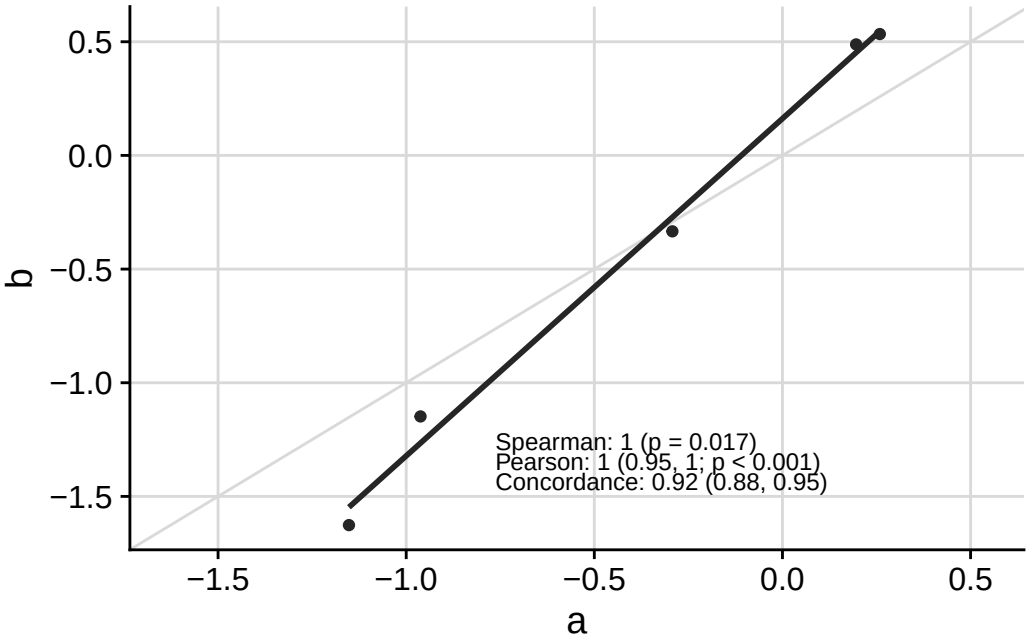
Specific functionality to make appropriate plots for the concordance correlation coefficient is available:

```
ggcorr(
  data = response_tbl |> dplyr::filter(group %in% c("a", "b")),
  grp = "group",
  y = "response",
  id = "pid",
  corr_method = "concordance",
  abline = TRUE,
  limits_equal = TRUE
)
```



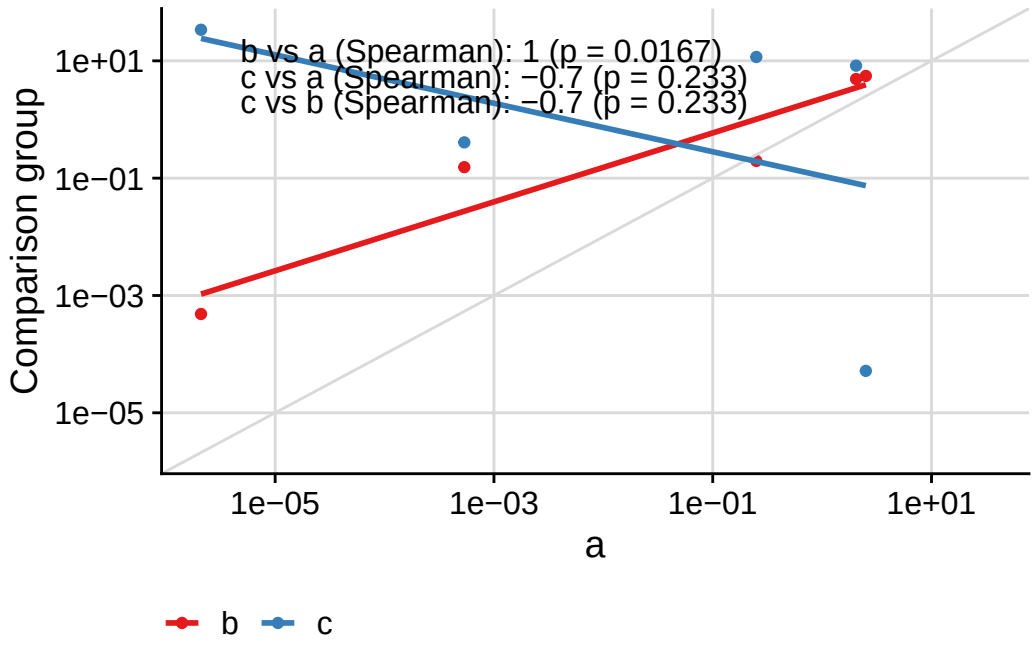
Text in table can be moved around and resized:

```
ggcorr(
  data = response_tbl |> dplyr::filter(group %in% c("a", "b")),
  grp = "group",
  y = "response",
  id = "pid",
  corr_method = c("spearman", "pearson", "concordance"),
  abline = TRUE,
  limits_equal = TRUE,
  coord = c(0.4, 0.17),
  font_size = 3,
  skip = 0.04,
  pval_signif = 2,
  est_signif = 2,
  ci_signif = 2
)
```



Finally, the text placement is kept consistent when the axes are visually transformed:

```
ggcorr(
  data = response_tbl |> dplyr::mutate(response = abs(response + 1)^4),
  grp = "group",
  y = "response",
  id = "pid",
  corr_method = "spearman",
  abline = TRUE,
  limits_equal = TRUE,
  trans = "log10",
  skip = 0.06
)
```



11.4.4 Bootstrapping

Maybe next time!

11.5 Homework

11.5.1 Question one

A lab wants to check that operator `B`, a trainee, can achieve the same results on the same assay as operator `A`. Read in the data by running the following command:

```
operator_tbl <- structure(list(operator = c("A", "A", "A", "A", "A", "A", "A",
"A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A",
"A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "B", "B", "B",
"B", "B", "B", "B", "B", "B", "B", "B", "B", "B", "B", "B", "B",
"B", "B", "B", "B", "B", "B", "B", "B", "B", "B", "B", "B", "B",
"B"), measurement = c(5.86, 0.0895, 2.94, 2.77, 8.14, 2.6, 7.24,
9.06, 9.49, 0.731, 7.55, 2.86, 1, 9.54, 4.16, 4.55, 9.71, 5.84,
9.62, 7.62, 7.15, 9.97, 5.06, 4.9, 6.49, 8.31, 4.82, 8.42, 5.14,
5.3, 7.78, 1.28, 3.77, 3.51, 10.3, 4.93, 9.58, 13.1, 13.6, 1.54,
9.53, 4.97, 2.21, 11.5, 6.64, 6.07, 13.7, 6.84, 11, 10.8, 8.88,
```

```
12.7, 7.52, 5.9, 7.8, 12.1, 6.45, 12.2, 4.99, 6.07)), class = c("tbl_df",
"tbl", "data.frame"), row.names = c(NA, -60L))
```

a. Calculate the appropriate correlation coefficient.
b. Plot the raw data and the appropriate correlation coefficient.
c. Interpret the results.

11.5.2 Question two

We are interested in knowing whether two genes are associated. We do not know the nature of the relationship, if there is one.

Read the data in, as follows:

```
gene_tbl <- structure(list(gene = c("UNDERMINER4", "UNDERMINER4", "UNDERMINER4",
"UNDERMINER4", "UNDERMINER4", "UNDERMINER4", "UNDERMINER4", "UNDERMINER4",
"UNDERMINER4", "UNDERMINER4", "UNDERMINER4", "UNDERMINER4", "UNDERMINER4",
"UNDERMINER4", "UNDERMINER4", "UNDERMINER4", "UNDERMINER4", "UNDERMINER4",
"UNDERMINER4", "UNDERMINER4", "TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7",
"TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7",
"TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7",
"TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7", "TROGLODYTE7",
"TROGLODYTE7", "TROGLODYTE7"), expression = c(5.85800305008888,
0.0894579570740461, 2.93739611981437, 2.77374957920983, 8.13574214931577,
2.60427771368995, 7.24405892658979, 9.06092151300982, 9.49040221050382,
0.73144469410181, 7.54675026983023, 2.8600062080659, 1.00053521571681,
9.5406877505593, 4.15607118513435, 4.55102417618036, 9.71055655973032,
5.83987979684025, 9.6220462443307, 7.6170240319334, 201.591007212395,
0.0164353615310555, 25.7277804258383, 21.2952232089115, 538.541568097937,
17.8319216170362, 381.30708631164, 743.86015840029, 854.678654517749,
0.107886637441148, 431.354200915445, 23.558977358528, 2.3092288669259,
869.726714251939, 72.3804140205039, 93.9770546568619, 916.911928248853,
200.074244601714, 889.917326105173, 443.17271786801)), class = c("tbl_df",
"tbl", "data.frame"), row.names = c(NA, -40L))
```

a. Calculate the most appropriate correlation coefficient. Is it statistically significant?
b. Plot the raw data and the appropriate correlation coefficient.
c. Interpret the results.

11.5.3 Question three

We have been reliably informed that two variables are linearly related (last question imagination levels here). Read the data in as follows:

```
set.seed(4)
n_obs <- 15
x_all_match <- runif(n_obs, 0, 10)
y_all_match <- 5 + x_all_match * 10 + rnorm(n_obs, 0, 10)
var_tbl <- tibble(
  variable = rep(c("exciting-thing-1", "exciting-thing-2"), each = n_obs),
  value = c(x_all_match, y_all_match)
)
dput(var_tbl)
```

```
structure(list(variable = c("exciting-thing-1", "exciting-thing-1",
"exciting-thing-1", "exciting-thing-1", "exciting-thing-1", "exciting-thing-1",
"exciting-thing-1", "exciting-thing-1", "exciting-thing-1", "exciting-thing-1",
"exciting-thing-1", "exciting-thing-1", "exciting-thing-1", "exciting-thing-1",
"exciting-thing-1", "exciting-thing-2", "exciting-thing-2", "exciting-thing-2",
"exciting-thing-2", "exciting-thing-2", "exciting-thing-2", "exciting-thing-2",
"exciting-thing-2", "exciting-thing-2", "exciting-thing-2", "exciting-thing-2",
"exciting-thing-2", "exciting-thing-2", "exciting-thing-2", "exciting-thing-2"
), value = c(5.85800305008888, 0.0894579570740461, 2.93739611981437,
2.77374957920983, 8.13574214931577, 2.60427771368995, 7.24405892658979,
9.06092151300982, 9.49040221050382, 0.73144469410181, 7.54675026983023,
2.8600062080659, 1.00053521571681, 9.5406877505593, 4.15607118513435,
62.4522288284119, 8.01564342234369, 41.4918552015465, 59.8149071489473,
86.1053092224815, 40.6163456528487, 87.4571949887254, 96.3563217676658,
92.8071441398129, 16.2900278303966, 99.4289883901294, 36.7089255281769,
-9.78185363090419, 93.4135192204788, 37.4412720963096)), class = c("tbl_df",
"tbl", "data.frame"), row.names = c(NA, -30L))
```

Read the data in as follows:

```
var_tbl <- structure(list(variable = c("exciting-thing-1", "exciting-thing-1",
"exciting-thing-1", "exciting-thing-1", "exciting-thing-1", "exciting-thing-1",
"exciting-thing-1", "exciting-thing-1", "exciting-thing-1", "exciting-thing-1",
"exciting-thing-1", "exciting-thing-1", "exciting-thing-1", "exciting-thing-1",
"exciting-thing-1", "exciting-thing-2", "exciting-thing-2", "exciting-thing-2",
"exciting-thing-2", "exciting-thing-2", "exciting-thing-2", "exciting-thing-2",
```

```
"exciting-thing-2", "exciting-thing-2", "exciting-thing-2", "exciting-thing-2",
"exciting-thing-2", "exciting-thing-2", "exciting-thing-2", "exciting-thing-2"
), value = c(5.85800305008888, 0.0894579570740461, 2.93739611981437,
2.77374957920983, 8.13574214931577, 2.60427771368995, 7.24405892658979,
9.06092151300982, 9.49040221050382, 0.73144469410181, 7.54675026983023,
2.8600062080659, 1.00053521571681, 9.5406877505593, 4.15607118513435,
62.4522288284119, 8.01564342234369, 41.4918552015465, 59.8149071489473,
86.1053092224815, 40.6163456528487, 87.4571949887254, 96.3563217676658,
92.8071441398129, 16.2900278303966, 99.4289883901294, 36.7089255281769,
-9.78185363090419, 93.4135192204788, 37.4412720963096)), class = c("tbl_df",
"tbl", "data.frame"), row.names = c(NA, -30L))
```

a. Calculate the most appropriate correlation coefficient. Is it statistically significant?
b. Plot the raw data and the appropriate correlation coefficient.
c. Interpret the results.
d. Congratulate yourself for uncovering this important relationship!

11.5.4 Question four

Complete that blank section detailing how to plot the correlation coefficient using `ggplot2` directly above. Bonus marks (and immortality) for contributing the answer via GitHub to the repo (`https://github.com/SATVILab/SATVI_ComputationalCourse`).

12 Session Recordings

12.1 Description

On this page you will find links to the Microsoft Teams recordings for each session. You will only be able to access these recordings if you already have institutional access via Teams. If you would like to access the recordings, but do not have access, please email the course instructors.

12.2 2024 Session Recordings

12.2.1 Session 1: Intro to R and swirl

05 MAR 2024

Session 1 Video Recording

12.2.2 Session 2: MaRcus Training Course lesson 1

19 MAR 2024

Session 2 Video Recording

12.2.3 Session 3: MaRcus Training Course lesson 2

26 MAR 2024

Session 3 Video Recording

12.2.4 Session 4: MaRcus Training Course lesson 3

02 APR 2024

Session 4 Video Recording

Note - MaRcus Training Course lesson 4 was skipped as it covers R Markdown which will be replaced by a session on Quarto later.

12.2.5 Session 5: MaRcus Training Course lesson 5

09 APR 2024

Session 5 Video Recording

12.2.6 Session 6: MaRcus Training Course lesson 6

30 APR 2024

Session 6 Video Recording

12.2.7 Session 7: MaRcus Training Course lesson 7

07 MAY 2024

Session 7 Video Recording

12.2.8 Session 8: Exporting data from R

21 MAY 2024

Session 8 Video Recording

13 Summary

We hope you have enjoyed the course! Remember that the best way to learn how to code is to experiment and use coding languages as much as possible. Just like learning a foreign language, learning to code works best with daily practice.

If you have any questions, feedback, or suggestions, please contact the course instructors.

Best of luck on your computational journey!

References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.