

SATVI Computational Course

Session Guide

SATVI Computational Group

2024-03-05

Table of contents

| | |
|--|-----------|
| Preface | 3 |
| 1 Introduction | 4 |
| 1.1 Instructor contacts | 4 |
| 2 Syllabus | 5 |
| 2.1 Description | 5 |
| 2.2 Module 1: Intro to R and MaRcus Training Course | 5 |
| 2.2.1 Session 1: Intro to R and swirl | 5 |
| 2.2.2 Session 2: MaRcus Training Course lesson 1 | 5 |
| 2.2.3 Session 3: MaRcus Training Course lesson 2 | 6 |
| 2.2.4 Session 4: MaRcus Training Course lesson 3 | 6 |
| 2.2.5 Session 5: MaRcus Training Course lesson 5 | 6 |
| 2.2.6 Session 6: MaRcus Training Course lesson 6 | 7 |
| 2.2.7 Session 7: MaRcus Training Course lesson 7 | 7 |
| 2.2.8 Session 8: Exporting data from R | 7 |
| 2.3 Module 2: Quarto, GitHub, and GUIs | 8 |
| 2.3.1 Intro to Quarto | 8 |
| 2.3.2 Intro to GitHub | 8 |
| 2.3.3 Intro to VS Code | 8 |
| 2.4 Module 3: Statistics | 9 |
| 2.4.1 Basic statistical tests | 9 |
| 2.4.2 Correlations | 9 |
| 2.5 Module 4: Commonly needed analyses for Immunology | 9 |
| 2.5.1 Heatmaps | 9 |
| 2.5.2 Dimensionality reduction | 10 |
| 2.5.3 Receiver operating characteristic (ROC) curves | 10 |
| 2.5.4 Background subtraction | 10 |
| 2.5.5 Basic flow cytometry analysis | 11 |
| 2.5.6 Automatic gating | 11 |
| 2.6 Module 5: Other coding languages | 11 |
| 2.6.1 Intro to Python | 11 |
| 3 Installations | 12 |
| 3.1 Description | 12 |

| | | |
|----------|---|-----------|
| 3.2 | R | 12 |
| 3.3 | R Studio | 12 |
| 3.4 | GitHub Desktop | 13 |
| 3.5 | Visual Studio Code (VS Code) | 13 |
| 4 | swirl | 14 |
| 4.1 | Description | 14 |
| 4.2 | Install swirl | 14 |
| 4.3 | Initialize swirl | 14 |
| 4.4 | Install an interactive course | 15 |
| 4.5 | Run swirl | 15 |
| 4.6 | Exit swirl | 15 |
| 4.7 | Interactive commands | 16 |
| 4.8 | Homework | 16 |
| 4.9 | FAQ | 16 |
| 5 | MaRcus R Training | 18 |
| 5.1 | Description | 18 |
| 5.2 | Content access | 18 |
| 5.3 | Homework | 18 |
| 5.4 | FAQ | 19 |
| 6 | Exporting and Importing Data Formats in R | 22 |
| 6.1 | Description | 22 |
| 6.1.1 | Clear environment | 22 |
| 6.1.2 | Set output directory | 22 |
| 6.1.3 | Load libraries | 22 |
| 6.1.4 | Load datasets | 23 |
| 6.1.5 | Examine data structure | 23 |
| 6.1.6 | Export data to .xlsx | 24 |
| 6.1.7 | Export data to .csv | 25 |
| 6.1.8 | Import data from .xlsx | 25 |
| 6.1.9 | Import data from .csv | 27 |
| 6.1.10 | Plot data and export | 28 |
| 6.1.11 | Save what has been done to an .Rdata file | 29 |
| 6.2 | Homework | 30 |
| 7 | Introduction to Quarto | 32 |
| 7.1 | Description | 32 |
| 7.1.1 | Install R Markdown | 32 |
| 7.1.2 | Create a Quarto document | 32 |
| 7.1.3 | Set the YAML header | 35 |
| 7.1.4 | Create a code chunk | 35 |

| | | |
|-----------|---|-----------|
| 7.1.5 | Tailor the code chunk output | 36 |
| 7.2 | View outline | 38 |
| 7.3 | Now let's test a simple script in Quarto | 39 |
| 7.3.1 | Clear environment | 39 |
| 7.3.2 | Load libraries | 39 |
| 7.3.3 | Load dataset | 39 |
| 7.3.4 | Examine data structure | 39 |
| 7.3.5 | Plot data | 42 |
| 7.4 | Import an image | 42 |
| 7.5 | Render the document | 43 |
| 7.6 | Cheatsheets | 44 |
| 7.7 | Homework | 45 |
| 8 | Introduction to Git and GitHub | 46 |
| 8.1 | Description | 46 |
| 8.1.1 | What is Git? | 46 |
| 8.1.2 | Basic git commands: | 47 |
| 8.1.3 | Interface with git from your local machine using the Terminal | 47 |
| 8.1.4 | Branching and Merging from the Terminal | 51 |
| 8.1.5 | Using git commands to navigate through git architecture from the Terminal | 52 |
| 8.1.6 | Create, branch, and clone a repository from GitHub | 52 |
| 8.1.7 | Create, branch, and push a repository from GitHub Desktop | 55 |
| 8.2 | Cheatsheets | 57 |
| 8.3 | Homework | 57 |
| 9 | Introduction to Visual Studio Code | 59 |
| 9.1 | Description | 59 |
| 9.1.1 | What is VS Code? | 59 |
| 9.1.2 | Initialize a project in VS Code | 59 |
| 9.1.3 | Install Extensions | 68 |
| 9.1.4 | Manage files | 69 |
| 9.2 | Code | 73 |
| 9.2.1 | Clear environment | 73 |
| 9.2.2 | Set output directory | 73 |
| 9.2.3 | Load libraries | 74 |
| 9.2.4 | Load dataset | 74 |
| 9.2.5 | Plot data and export | 74 |
| 9.3 | Debugging | 75 |
| 9.4 | Cheatsheets | 76 |
| 9.5 | Homework | 77 |
| 10 | Session Recordings | 78 |
| 10.1 | Description | 78 |

| | |
|---|-----------|
| 10.2 2024 Session Recordings | 78 |
| 10.2.1 Session 1: Intro to R and swirl | 78 |
| 10.2.2 Session 2: MaRcus Training Course lesson 1 | 78 |
| 10.2.3 Session 3: MaRcus Training Course lesson 2 | 78 |
| 10.2.4 Session 4: MaRcus Training Course lesson 3 | 79 |
| 10.2.5 Session 5: MaRcus Training Course lesson 5 | 79 |
| 10.2.6 Session 6: MaRcus Training Course lesson 6 | 79 |
| 10.2.7 Session 7: MaRcus Training Course lesson 7 | 79 |
| 10.2.8 Session 8: Exporting data from R | 79 |
| 11 Summary | 80 |
| References | 81 |

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-baile@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 tidyverse

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: 16 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 download 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 source 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 functions 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 session.

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 using the terminal.
2. You can also set the working directory using point and click in RStudio. To do so, navigate to the "Session" tab and click "Set Working Directory".
3. A note about setting working directories in scripts. It is good practice to avoid using relative paths and instead use absolute paths or the `getwd()` function to get the current working directory.

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 column names.

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 panels.

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 layers.

Q5: What are HEX codes?

A: HEX codes are unique alphanumeric codes assigned to specific colors. They can be used to assign colors to variables in R.

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 as an analogy: In R Studio, the top left (script) is your recipe book.

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

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 will learn how to do this.

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 menu bar.

Q2: How does 'filter()' work?

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

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

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

```
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 do one thing and do it well. Functions can take arguments, return values, and be combined with other functions.

Alternatively, an operator is a simple symbol that is used to perform arithmetic, logical, or comparison operations.

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 specific string “NA”.

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, 4, 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, 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, 1, 1, 1, 0, 0, ~
$ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 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, 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 used

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 = TRUE

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

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_rownamesstocol.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 CSV

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, use write.table

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

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>
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

```

```
# 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

```
# 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 |

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

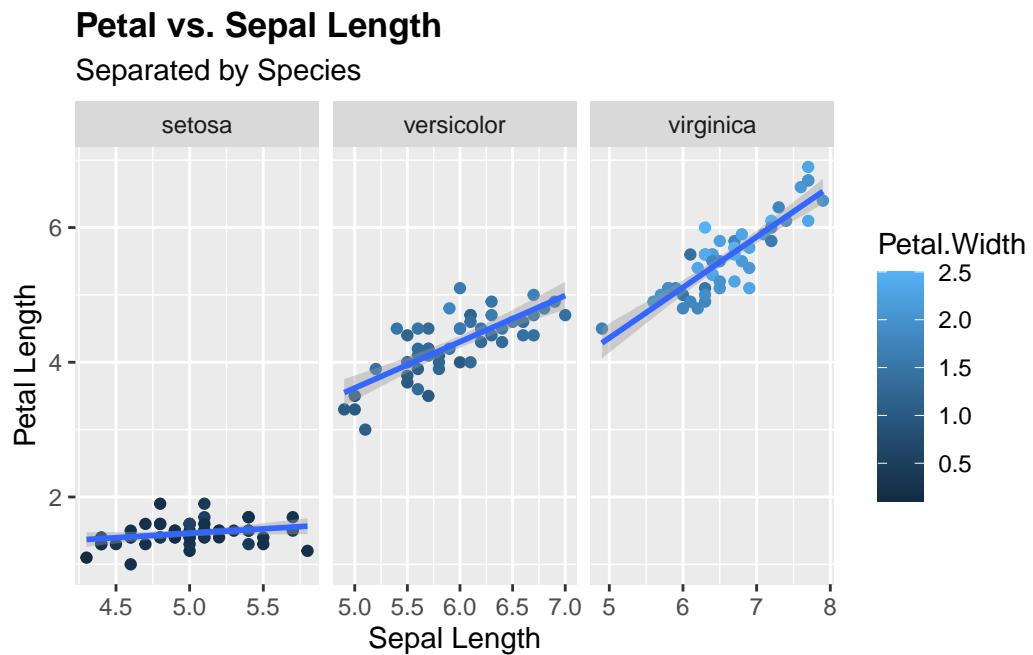
head(df.mtcars.csv)
```

| | 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 method

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 Quarto

7.1 Description

Quarto is an open-source new evolution of R Markdown. Quarto supports development in various coding languages and includes publishing and authoring features for individual documents, books, presentations, and websites.

Using Quarto can provide a helpful structure to organize, annotate, and share your scripts. Well-annotated Quarto documents can also promote code literacy as they are easy to read, write, and modify.

Quarto is very well documented. This tutorial will provide an introduction to some basic features, but we recommend that you explore independently. These links below may be a good place to start:

[Tutorial: Hello, Quarto](#)

[Markdown Basics](#)

7.1.1 Install R Markdown

To use Quarto with R you will need to install R Markdown with the rmarkdown R package. This will also install knitr which is used for rendering. After installing rmarkdown, install the quarto R package.

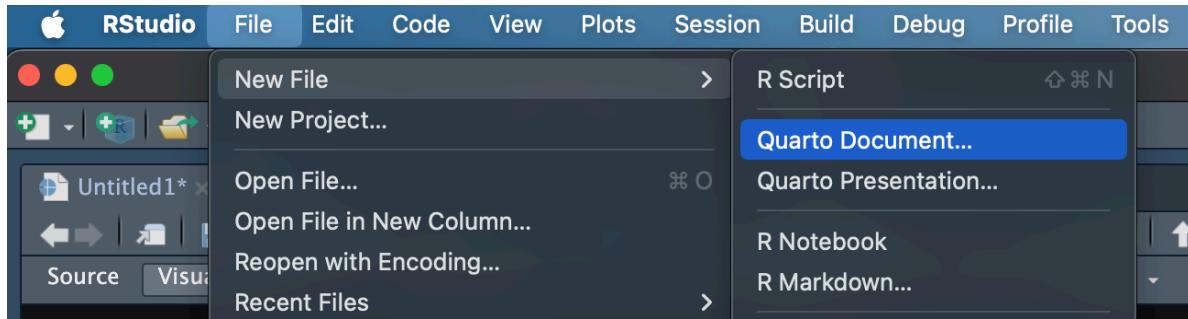
```
install.packages("rmarkdown")
install.packages("quarto")
```

7.1.2 Create a Quarto document

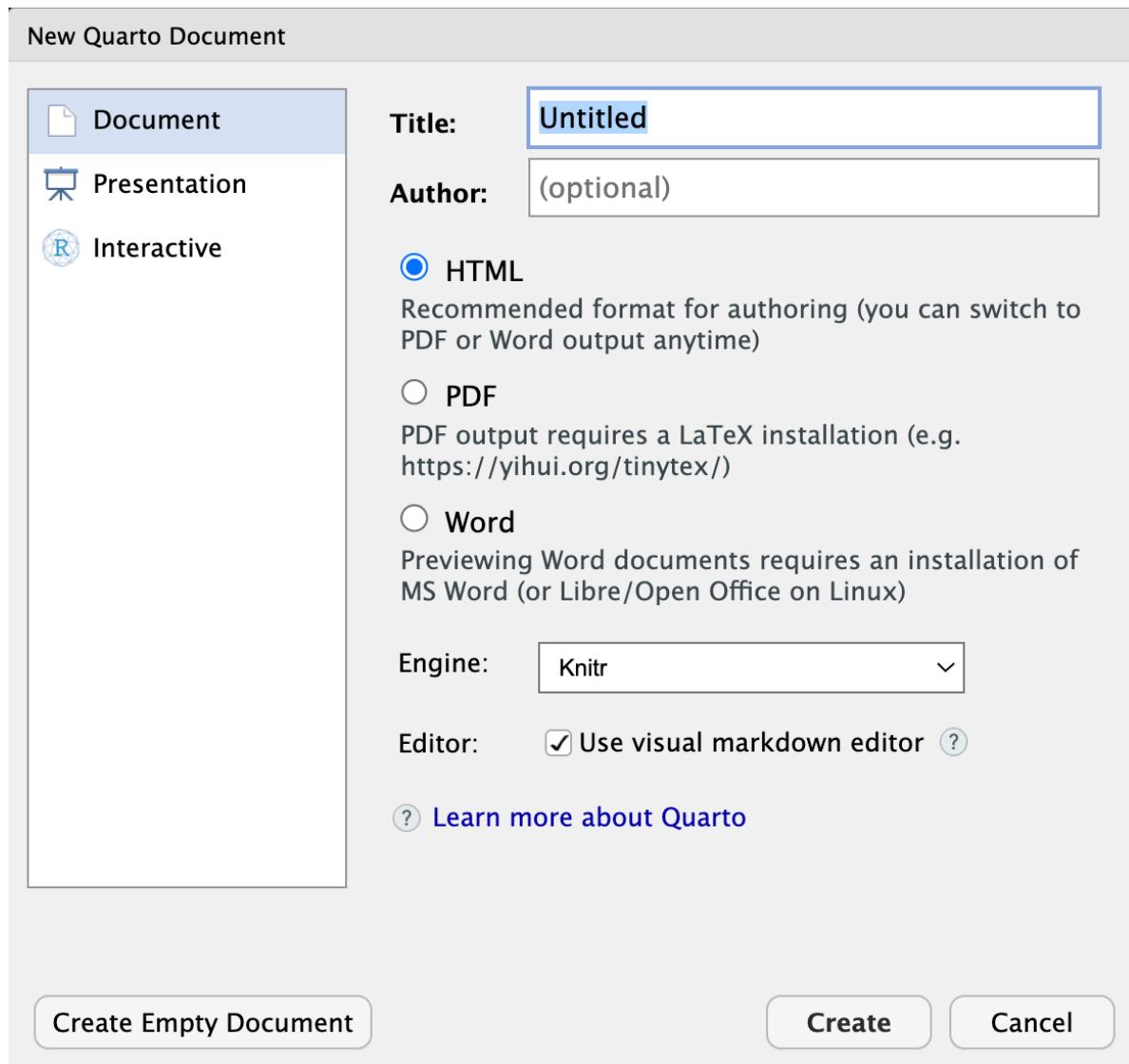
For the purposes of this session, we will introduce using Quarto documents using R. You can also create a Quarto project, which knits together multiple Quarto documents, Quarto presentations, Quarto interactive documents, or Quarto websites. Quarto is also supported by different visual editors including Visual Studio Code (VS Code) which we will cover in a later session.

To begin:

1. Open RStudio
2. Create a new Quarto document. Go to File -> New File -> Quarto Document



3. Enter a name and select your preferred output format and engine. Typically we leave Knitr as the default, but you can change this if necessary.



4. Save the quarto document to your working directory.

Now we can begin with editing the document itself. The most obvious difference in features from a traditional R script is that a Quarto document incorporates code chunks which separate major segments of code into different chunks. Each chunk can be formatted and run separately as well as part of the full script. We recommend that each chunk performs a specific function that can be annotated.

Sections of code containing one or more code chunks that perform a specific task can also be denoted in an outline with headings.

7.1.3 Set the YAML header

At the top of the Quarto document, you will see a header demarcated by — at the top and bottom. What is a YAML? YAML (“Yet Another Markup Language” or “YAML Ain’t Markup Language,” depending on who you ask) is the metadata that tells R Markdown, Quarto, pandoc, and other software exactly how to process or display the document.

```
---
```

```
title: "Untitled"
format: html
editor: visual
---
```

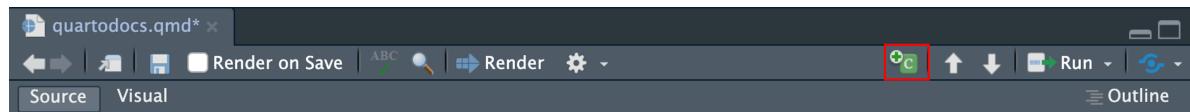
Edit the YAML header to include relevant information.

```
---
```

```
title: "Introduction to Quarto"
format: html
author: SATVI Computational Group
date: today
---
```

7.1.4 Create a code chunk

You can do this manually by clicking the green chunk icon on the top right of the toolbar or you can use the keyboard shortcut option + cmd + I. This will produce an empty code chunk with R as the default language.



```
```{r}
#| label: R chunk

print("Hello, Quarto!")
```
```

```
[1] "Hello, Quarto!"
```

If you want to change the language, simply specify the language you would like to use in the yellow opening delimiter.

```

```{python}
This is a Python code chunk
import math

Calculate the square root of 16
sqrt_16 = math.sqrt(16)

Print the result
print(f"The square root of 16 is {sqrt_16}")
```

```

The square root of 16 is 4.0

7.1.5 Tailor the code chunk output

Add options to each code chunk delimiter to show or hide specific information.

1. warning = FALSE means do not show any warnings generated by the code in the output
2. message = FALSE means do not show any messages generated by the code in the output
3. echo = FALSE means do not show the code in the output

```

```{r}
Generate a warning
x <- -1
if (x < 0) {
 warning("x is negative!")
}
```

```

Warning: x is negative!

```

```{r}
Generate a message
y <- 10
if (y > 5) {
 message("y is greater than 5")
}
```

```

```
y is greater than 5
```

Hide the warning from the output

```
```{r, warning = FALSE}
Generate a warning
x <- -1
if (x < 0) {
 warning("x is negative!")
}

Generate a message
y <- 10
if (y > 5) {
 message("y is greater than 5")
}
```

```

```
y is greater than 5
```

Hide the message from the output

```
```{r, message = FALSE}
Generate a warning
x <- -1
if (x < 0) {
 warning("x is negative!")
}
```

```

```
Warning: x is negative!
```

```
```{r, message = FALSE}
Generate a message
y <- 10
if (y > 5) {
 message("y is greater than 5")
}
```

```

Hide the warning and the message from the output. You can achieve the same effect using Markdown syntax instead of editing the opening delimiter.

```
```{r}
#| warning: false
#| message: false

Generate a warning
x <- -1
if (x < 0) {
 warning("x is negative!")
}

Generate a message
y <- 10
if (y > 5) {
 message("y is greater than 5")
}
```
```

Hide the code from the output

Warning: x is negative!

y is greater than 5

More OPML options can be found [here](#).

7.2 View outline

As you build your script with multiple chunks, an outline will be generated which can be viewed alongside your script or hidden.



7.3 Now let's test a simple script in Quarto

7.3.1 Clear environment

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

7.3.2 Load libraries

```
library(knitr) # Needed to embed an external image within a code chunk  
library(quarto) # Needed to render Quarto document from R console  
library(ggplot2) # Needed to plot data
```

7.3.3 Load dataset

```
data("esoph")
```

7.3.4 Examine data structure

```
head(esoph)
```

| | agegp | alcgp | tobgp | ncases | ncontrols |
|---|-------|-----------|----------|--------|-----------|
| 1 | 25-34 | 0-39g/day | 0-9g/day | 0 | 40 |
| 2 | 25-34 | 0-39g/day | 10-19 | 0 | 10 |
| 3 | 25-34 | 0-39g/day | 20-29 | 0 | 6 |
| 4 | 25-34 | 0-39g/day | 30+ | 0 | 5 |
| 5 | 25-34 | 40-79 | 0-9g/day | 0 | 27 |
| 6 | 25-34 | 40-79 | 10-19 | 0 | 7 |

```
summary(esoph)
```

| agegp | alcgp | tobgp | ncases | ncontrols |
|----------|--------------|-------------|----------------|-----------------|
| 25-34:15 | 0-39g/day:23 | 0-9g/day:24 | Min. : 0.000 | Min. : 0.000 |
| 35-44:15 | 40-79 :23 | 10-19 :24 | 1st Qu.: 0.000 | 1st Qu.: 1.000 |
| 45-54:16 | 80-119 :21 | 20-29 :20 | Median : 1.000 | Median : 4.000 |
| 55-64:16 | 120+ :21 | 30+ :20 | Mean : 2.273 | Mean : 8.807 |
| 65-74:15 | | | 3rd Qu.: 4.000 | 3rd Qu.: 10.000 |
| 75+ :11 | | | Max. :17.000 | Max. :60.000 |

If you want to display your dataframe in a more aesthetic way, you can use the DT R package.

Note - This display option is supported by html. If you want to display tables this way AND render to html and non-html formats, you will also need to install “webshot” or “webshot2” packages that will allow non-html formats to be populated with a snapshot of the table, or include “prefer-html: true” in your YAML front-matter.

```
library(DT)

# Now just use the datatable function to display your table.

datatable(head(esoph), class = 'cell-border stripe')
```

Show entries Search:

| | agegp | alcgp | tobgp | ncases | ncontrols |
|---|-------|-----------|----------|--------|-----------|
| 1 | 25-34 | 0-39g/day | 0-9g/day | 0 | 40 |
| 2 | 25-34 | 0-39g/day | 10-19 | 0 | 10 |
| 3 | 25-34 | 0-39g/day | 20-29 | 0 | 6 |
| 4 | 25-34 | 0-39g/day | 30+ | 0 | 5 |
| 5 | 25-34 | 40-79 | 0-9g/day | 0 | 27 |
| 6 | 25-34 | 40-79 | 10-19 | 0 | 7 |

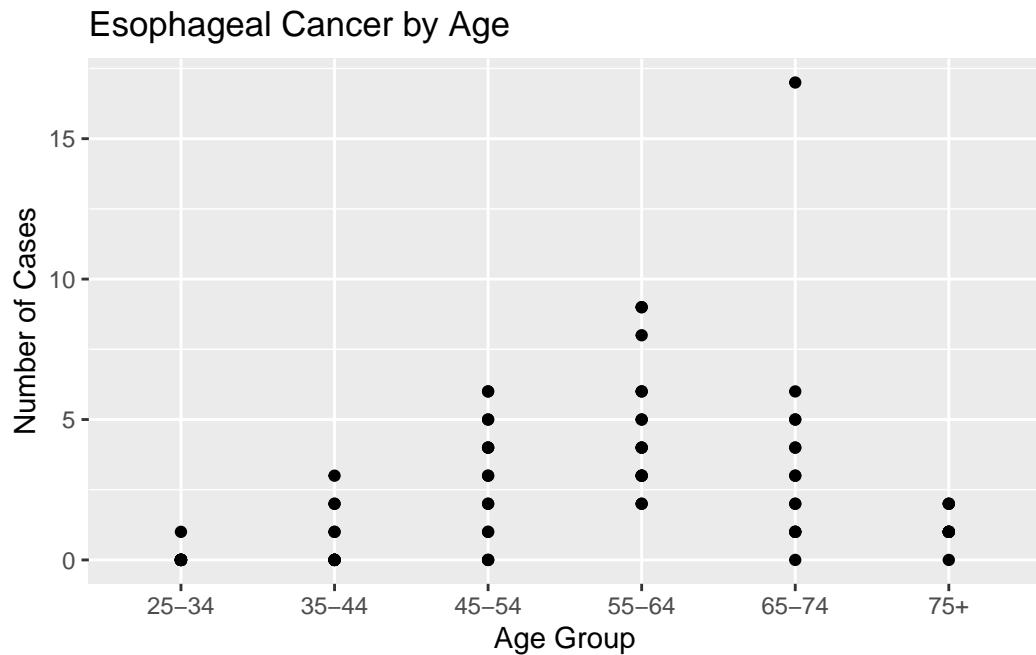
Showing 1 to 6 of 6 entries

Previous Next

Find additional datatable display options [here](#).

7.3.5 Plot data

```
ggplot(data = esoph, aes(x = agegp, y = ncases)) +  
  geom_point() + # Add points  
  labs(title = "Esophageal Cancer by Age", x = "Age Group", y = "Number of Cases")
```



Quarto docs also support the import of image files stored externally. These can be arranged inline with your code or text.

7.4 Import an image

The easiest way to do this is by using R Markdown syntax outside of a code chunk.

```
! [SATVI logo] (images/satvi_logo.png)
```



Figure 7.1: SATVI logo

If you prefer to import from within a Quarto code chunk, you can do so using knitr

```
knitr::include_graphics("images/satvi_logo.png")
```



Documentation on embedding images can be found [here](#).

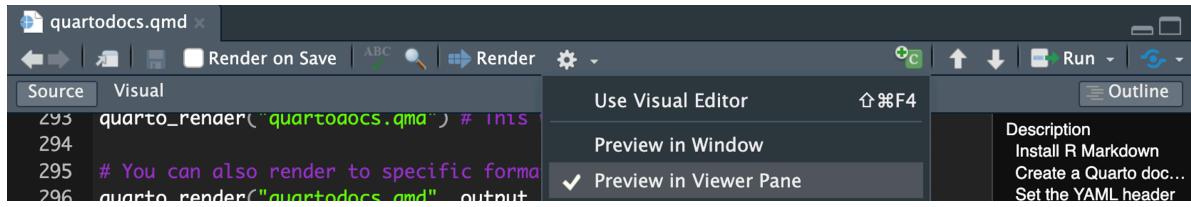
7.5 Render the document

When you render your document, all code chunks and inline code will be executed automatically. You can render the document with point-and-click, directly from the RStudio console, or from the command line in the Terminal. To render your Quarto document using point-and-click:

1. Render using point-and-click



This will open a preview of your rendered document automatically. You can select the type of preview you prefer using the drop down menu. If you select “Preview in Window,” a separate window with your rendered file will open in your browser. If you select “Preview in Viewer Pane,” a viewer pane will open within RStudio. This viewer pane will automatically update as you edit your code, so it can be helpful to select this option as you modify your documents as it will give you real-time visual updates of changes made.



You can also choose to render on every save by checking the box in the toolbar:



2. Render from the RStudio console

You will need to download and install the Quarto command line interface [Quarto CLI](#). Now type the following directly into the RStudio console.

```
quarto_render("quartodocs.qmd") # This will render all formats

# You can also render to specific formats
quarto_render("quartodocs.qmd", output_format = "html")
quarto_render("quartodocs.qmd", output_format = "pdf")
quarto_render("quartodocs.qmd", output_format = "docx")
```

3. Render from the command line

Open the Terminal.

```
quarto render quartodocs.qmd # This will render all formats

# You can also render to specific formats

quarto render quartodocs.qmd --to html
quarto render quartodocs.qmd --to pdf
quarto render quartodocs.qmd --to docx
```

Further information on rendering can be found [here](#).

7.6 Cheatsheets

Here are 2 helpful cheatsheets for working with Quarto in R:

[Quarto by RStudio](#)

7.7 Homework

Get familiar with Quarto

1. Set up a Quarto document and save to your course working directory.
2. Populate the Quarto document with your own code, make sure each chunk and the full script runs without errors.
3. Render the Quarto document to html and pdf (docx if you choose).
4. Go back to the script and tailor the code chunk options to suit your visual preferences.
5. Render the Quarto document again.
6. Experiment with other Quarto features. We recommend initializing a Quarto project, which can knit together multiple Quarto documents into a book, website, presentation, or interactive document.

Remember you can always find help using the R package documentation for rmarkdown and quarto. To access the package documentation, type the following:

```
?rmarkdown  
?quarto
```

Prepare for our next session

Instructions for downloads can be found on the installations page.

1. If you have not done so already, make an online GitHub account.
2. Access the SATVILab GitHub.
3. Download and install GitHub Desktop and login with your GitHub account.

8 Introduction to Git and GitHub

8.1 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.

8.1.1 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.

8.1.2 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

8.1.3 Interface with git from your local machine using the Terminal

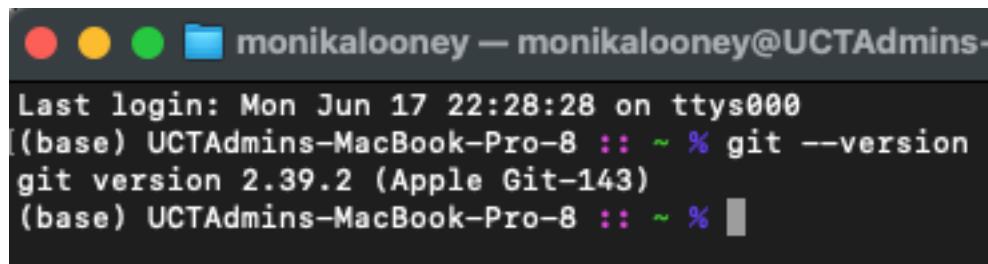
Most of the time, we use git on our local machine to develop the contents of the repo and then push them to GitHub. For this session, we will be interfacing with git from the Terminal to build an understanding of how git operates on a fundamental level. In our next session, we will cover how to develop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



The screenshot shows a terminal window with a dark background. At the top, there are three colored icons (red, yellow, green) followed by the text "monikalooney — monikalooney@UCTAdmins-". Below this, the terminal displays the following text:
Last login: Mon Jun 17 22:28:28 on ttys000
[(base) UCTAdmins-MacBook-Pro-8 :: ~ % git --version
git version 2.39.2 (Apple Git-143)
(base) UCTAdmins-MacBook-Pro-8 :: ~ %]

2. Configure git

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

```
# 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 simple text file or anything else.
- You should always have `.gitignore` and `README.md` files in your repository. `.gitignore` lists files and folders that you do not want to track.

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 ignoring 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 permanent

```
# 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. Normal

```
# 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 maintaining 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 changes

*Note - It is good practice to always include a short "message" with each commit that describes what was changed.

```
# 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
```

8.1.4 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 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
```

8.1.5 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.

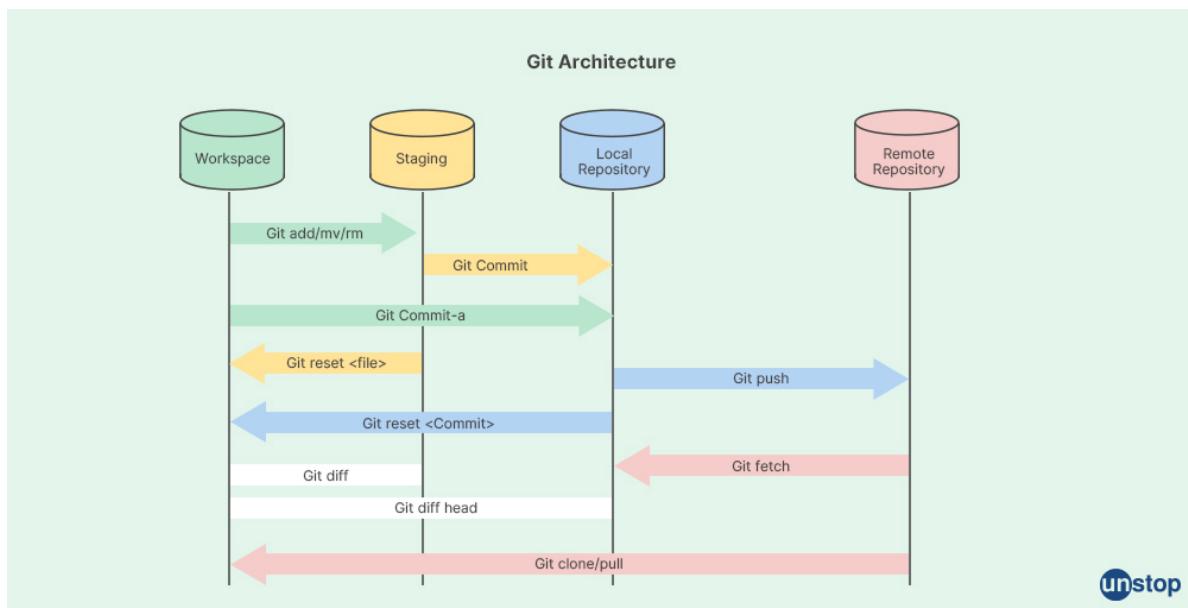
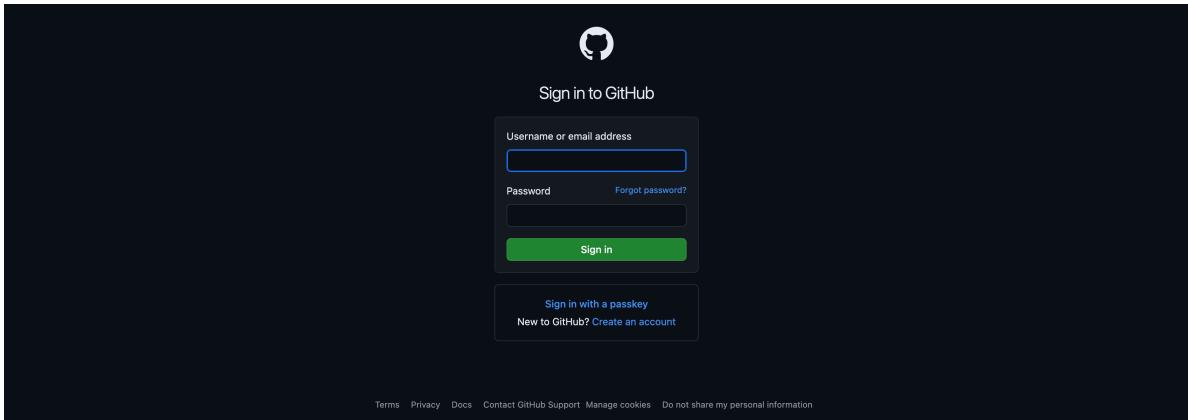


Figure 8.1: <https://unstop.com/blog/what-is-git>

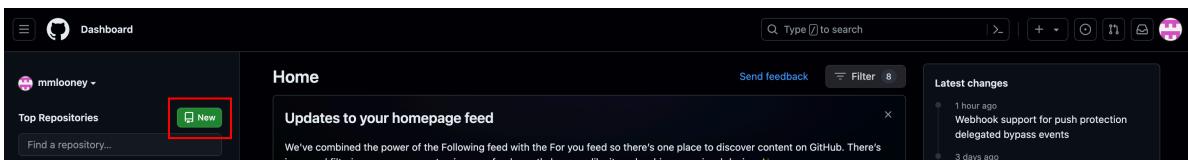
8.1.6 Create, branch, and clone a repository from GitHub

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

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

Repository template

No template

Start your repository with a template repository's contents.

Owner * Repository name *

mmllooney / Test_repo Test_repo is available.

Great repository names are short and memorable. Need inspiration? How about [congenial-disco](#) ?

Description (optional)

This is a test repo for instructional purposes

Public Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

License: **None**

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

(i) You are creating a private repository in your personal account.

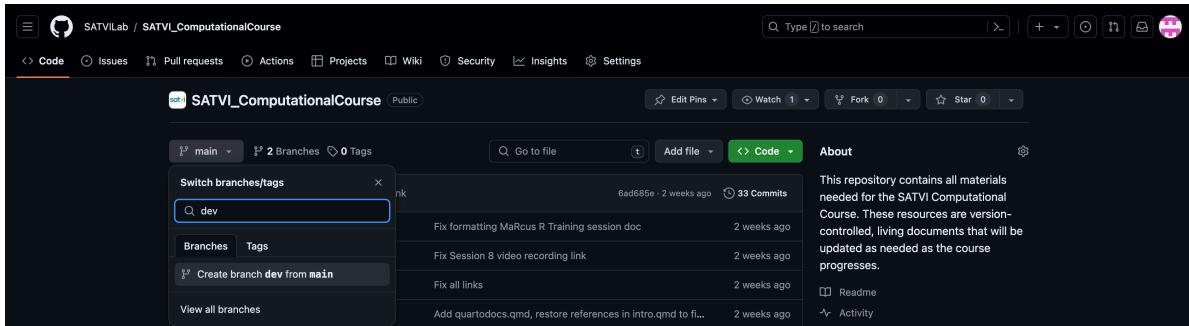
Create repository

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

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

The image consists of two vertically stacked screenshots of the GitHub interface. Both screenshots show the same repository page for 'SATVILab / SATVI_ComputationalCourse'. The top screenshot shows the 'Code' tab selected. In the top right corner, there is a navigation bar with several icons. Below the navigation bar, there is a search bar and a 'Type [] to search' placeholder. To the right of the search bar are several buttons: 'Edit Pins', 'Watch 1', 'Fork 0', 'Star 0', and a gear icon. In the center of the page, there is a button labeled 'Add file' with a red box drawn around it. To the right of the 'Add file' button are 'Code', 'About', and a gear icon. The bottom screenshot shows the same repository page, but now the 'README.md' file is listed under the 'Code' tab. The file has a status of 'in main'. At the bottom right of the page, there is a 'Commit changes...' button with a red box drawn around it. The rest of the interface remains the same, including the navigation bar, search bar, and other repository details.

5. Create a new branch.



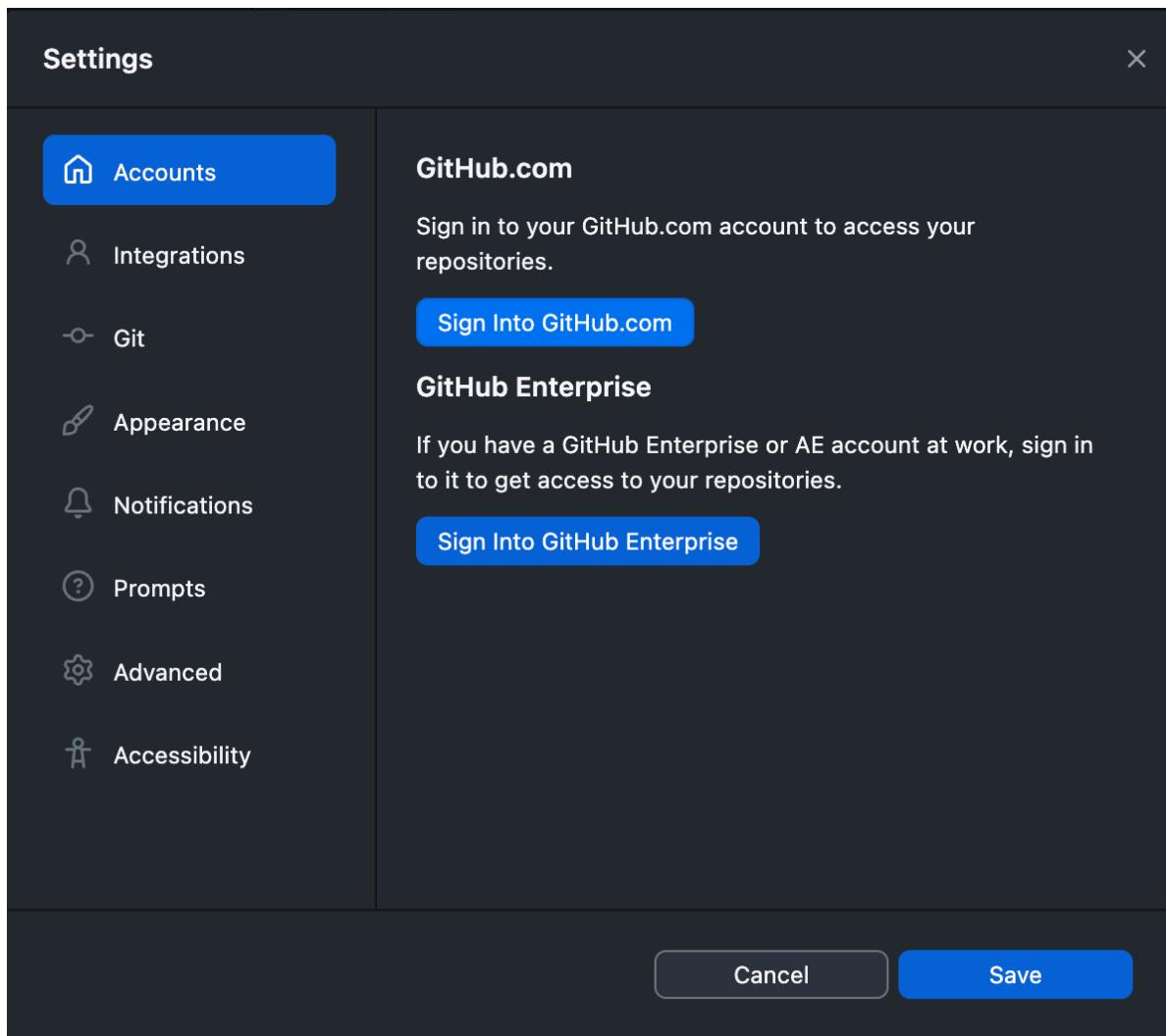
6. Clone your remote repo to your working directory.

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

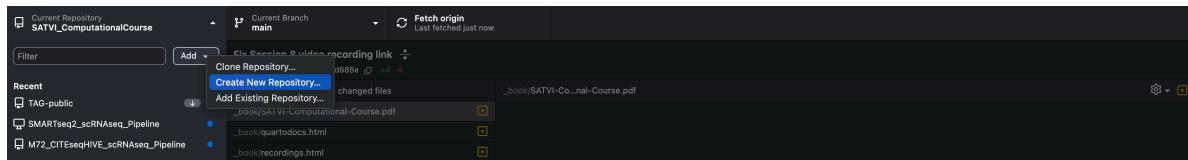
```
git clone https://github.com/SATVILab/SATVI_ComputationalCourse.git
```

8.1.7 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 ad-



3. Branch the repo.



4. Push the repo to GitHub.



8.2 Cheatsheets

Here are two helpful git sheet sheets:

[Git Cheat Sheet](#)

[GitLab Cheat Sheet](#)

8.3 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

9 Introduction to Visual Studio Code

9.1 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 your computational kit.

9.1.1 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.

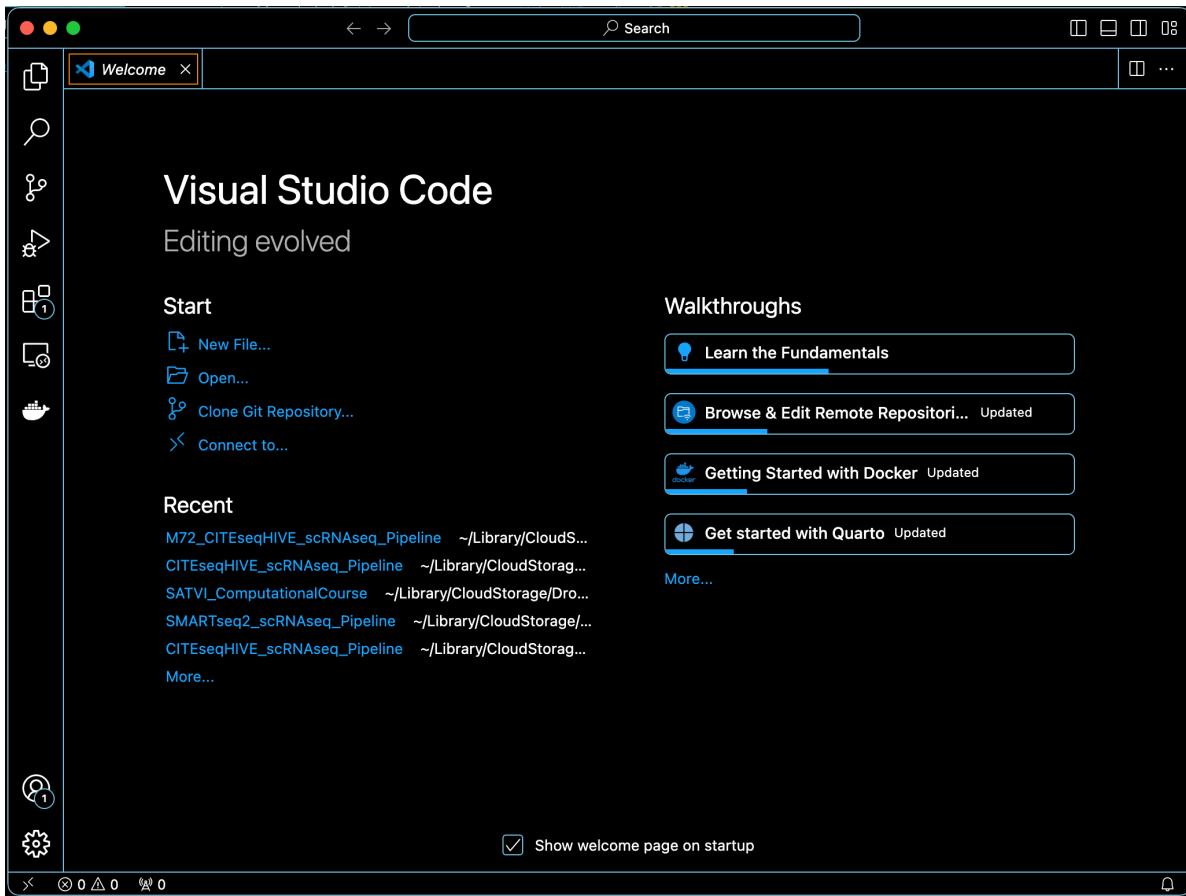
Some common vocabulary:

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

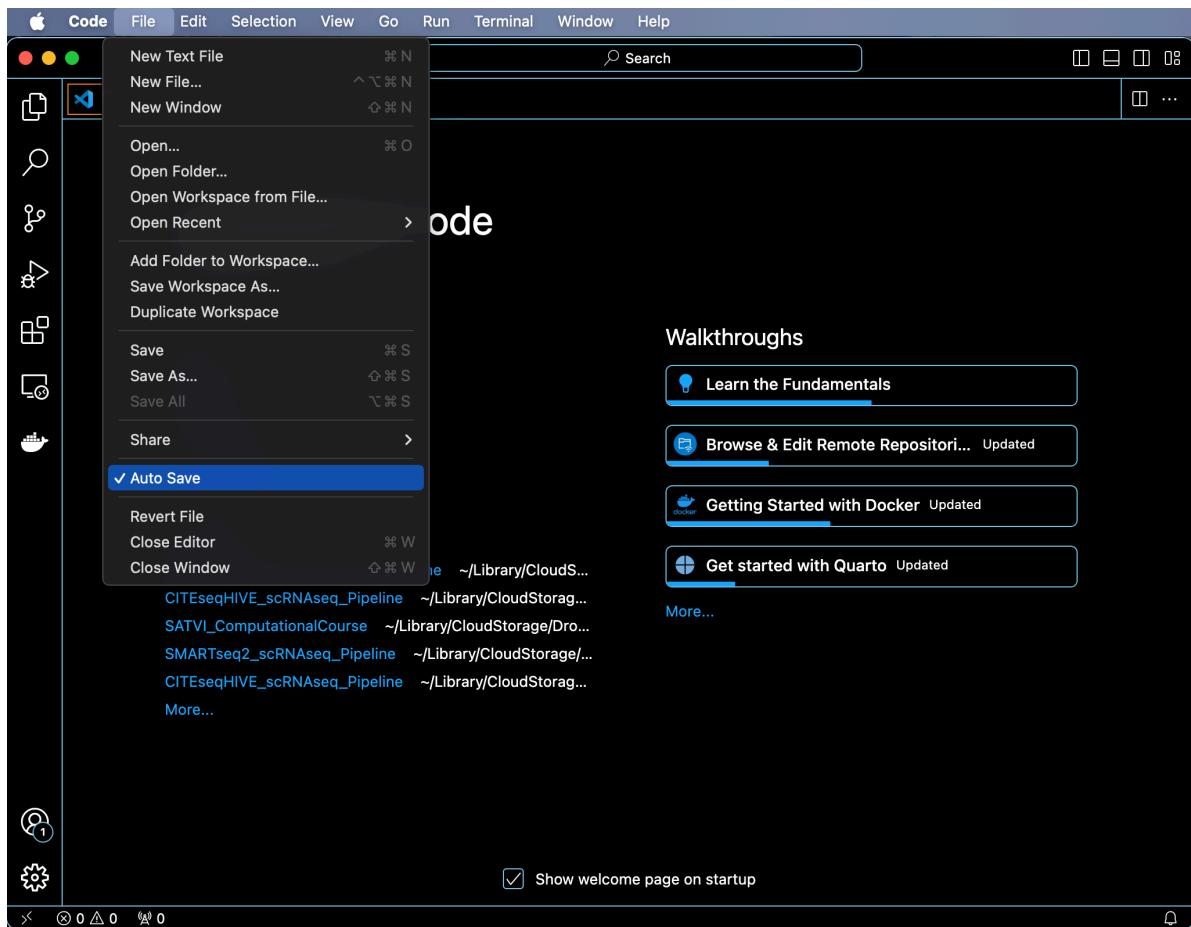
9.1.2 Initialize a project in VS Code

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 links 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.

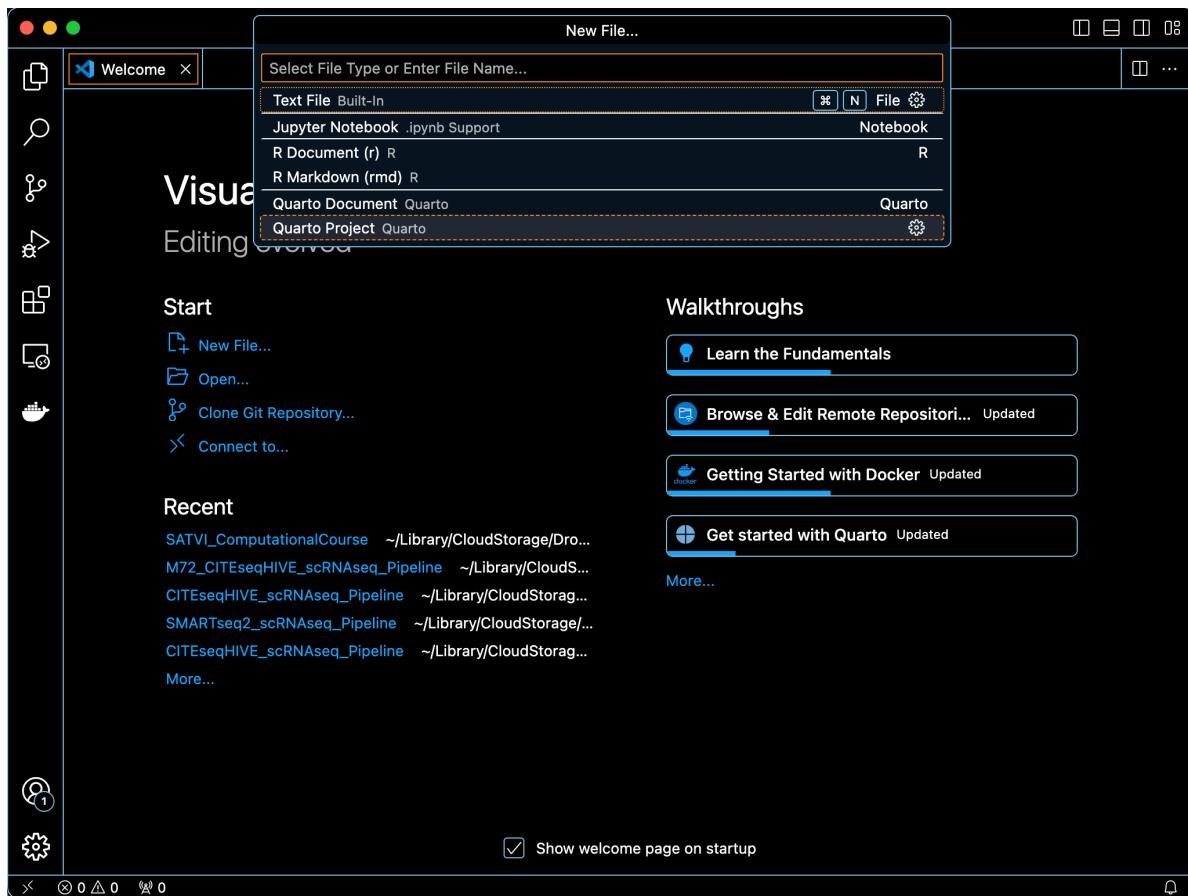


2. Consider checking “Auto Save”

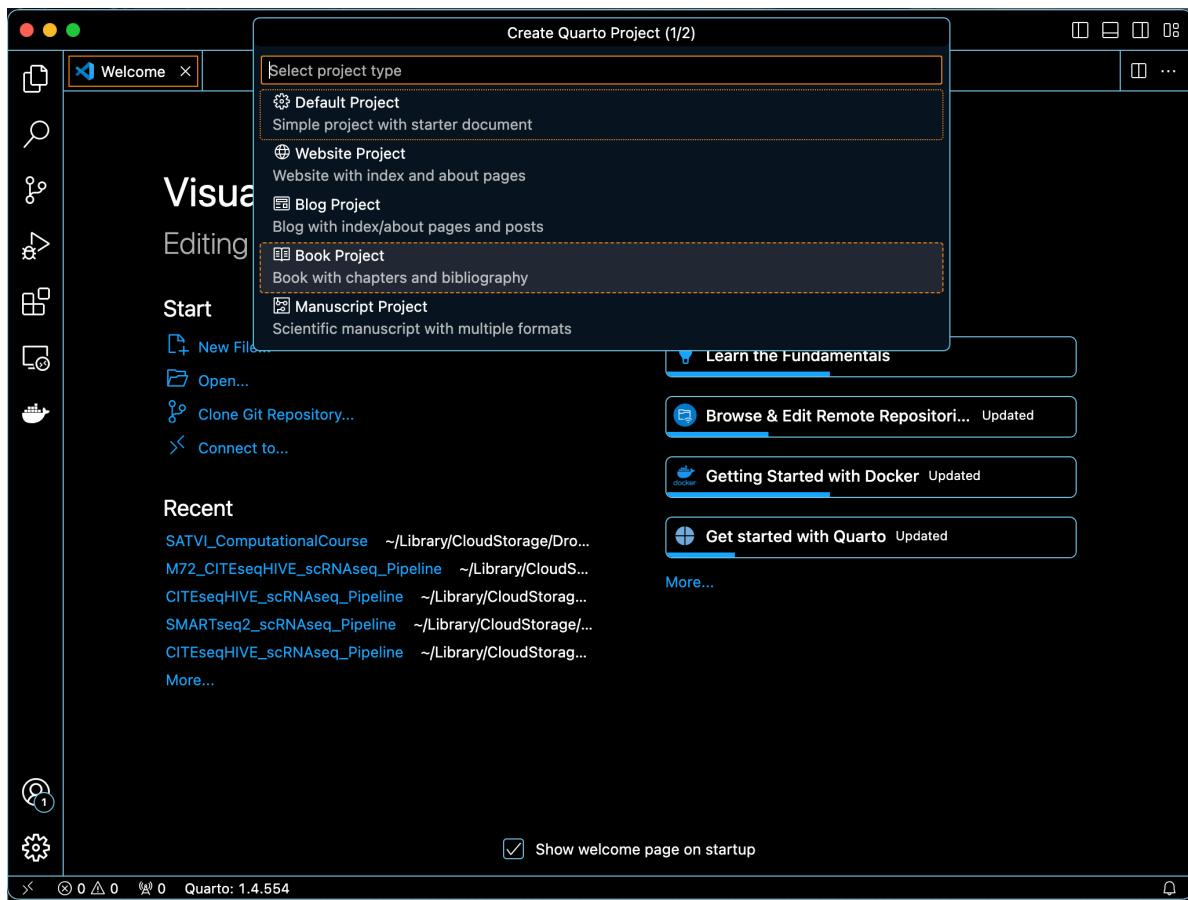


3. Create a new Quarto project

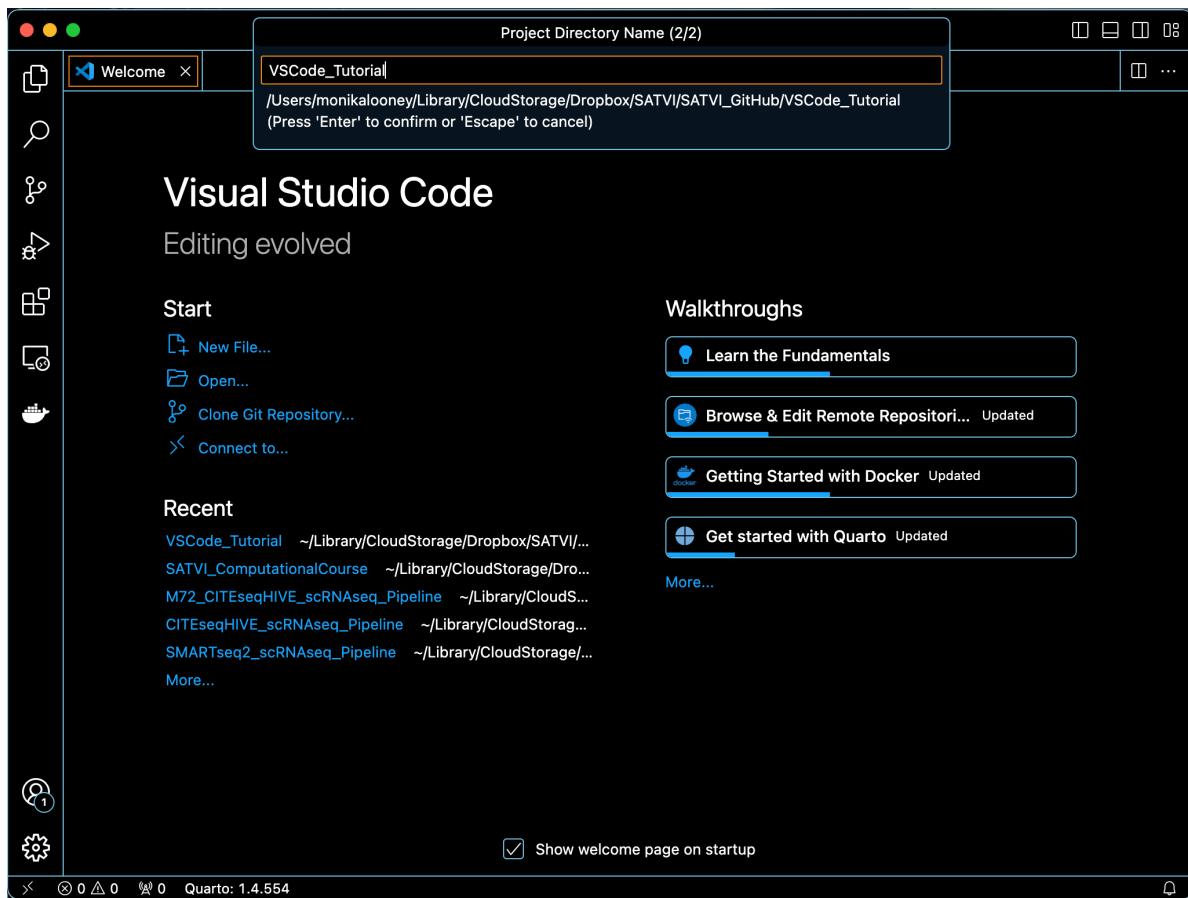
- When you click “New File”, a pop-up will open where you can select the type of file you want to generate. We will select “Quarto Project”



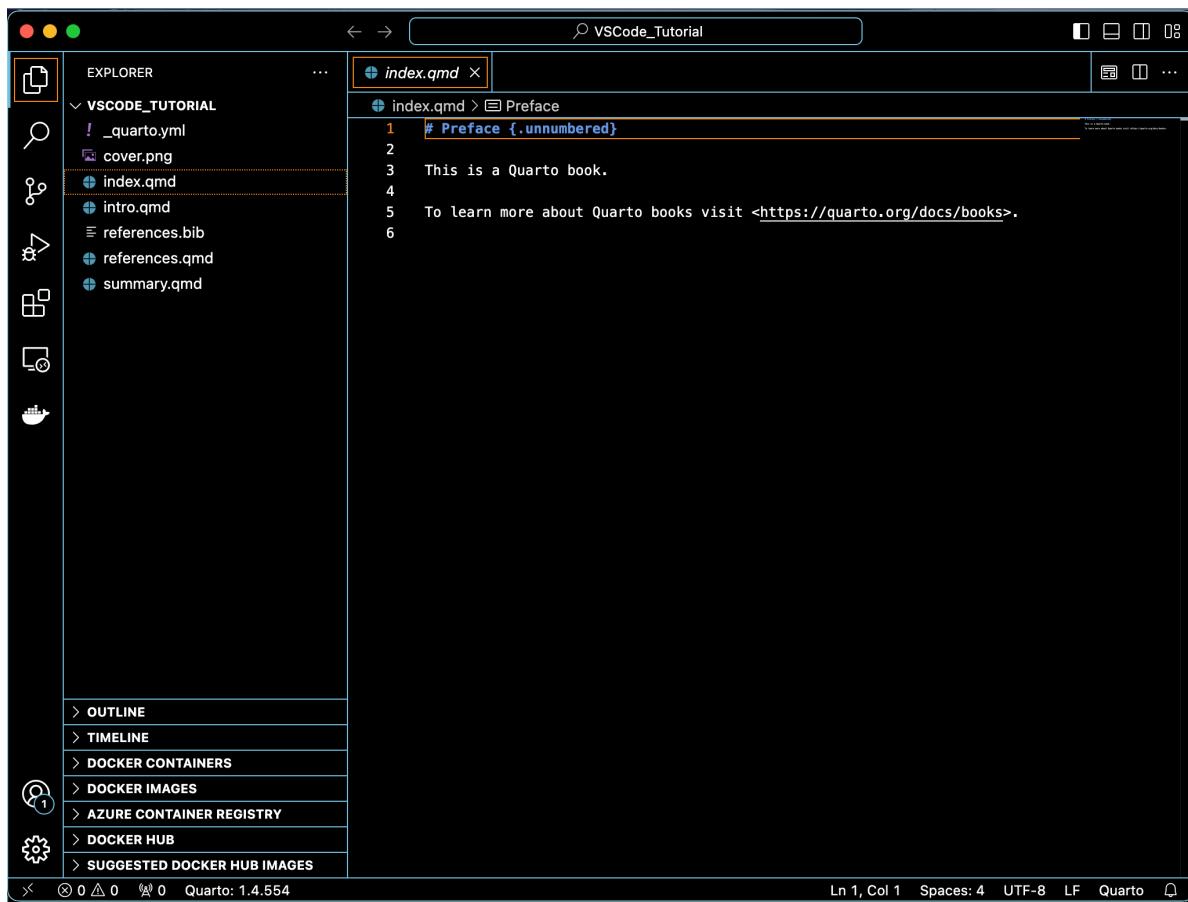
4. Next select the type of project you will want to create. We will create a Quarto Book.



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



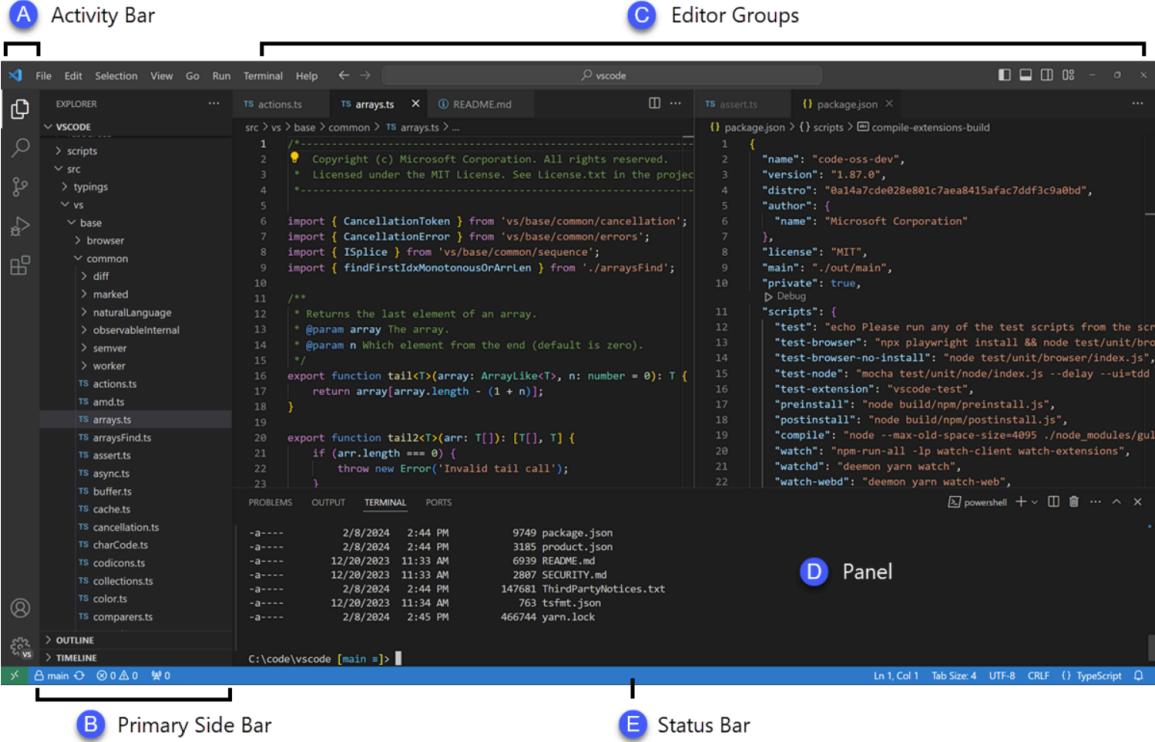
6. You will see your Quarto Book project directory and automatically generated base files appear in the left panel in VS Code.



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

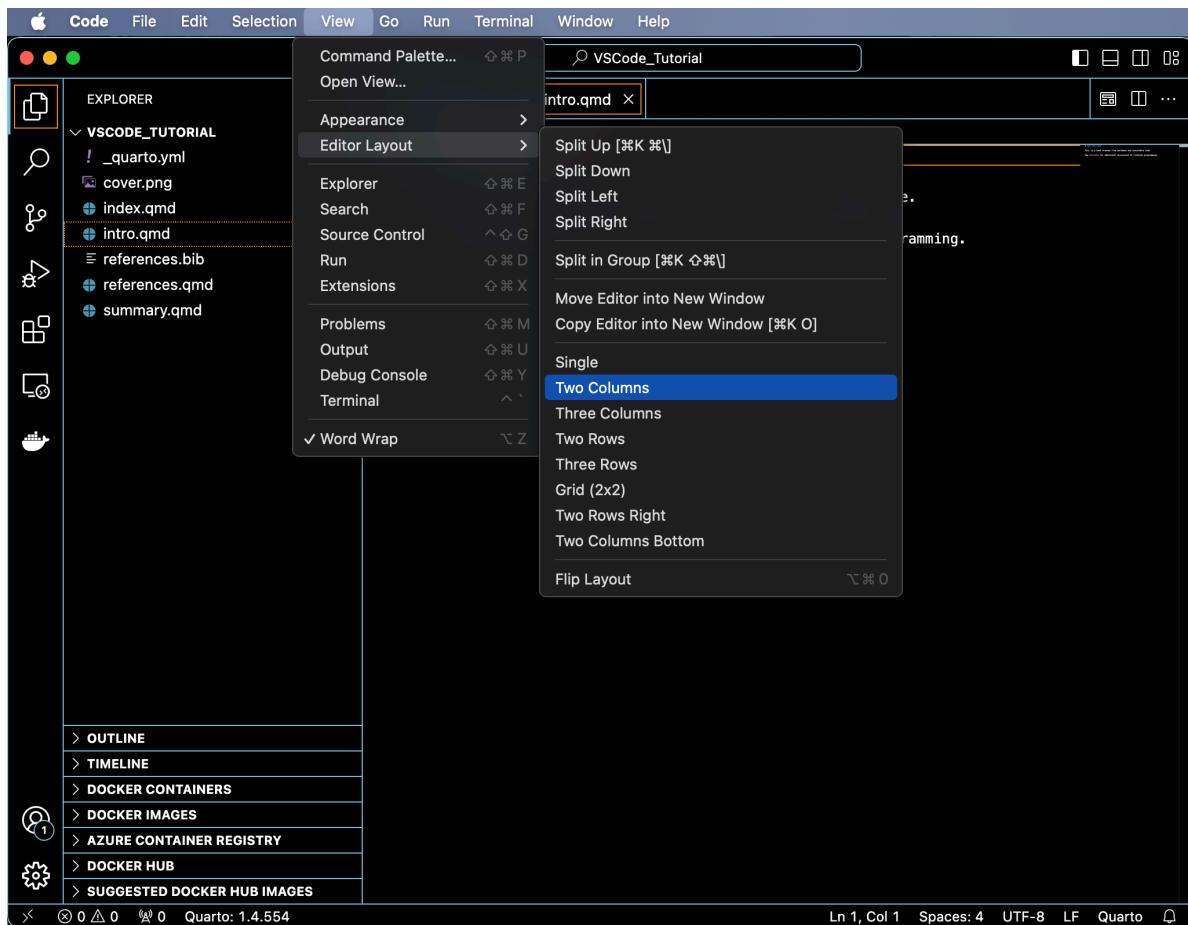
What you see here will depend on what you have selected from the Activity Bar. We often have this set to “Explorer” which shows you the files within your directory and project. The Explorer pane can also show your outline, timeline, containers, etc that are relevant for the open project. These are all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

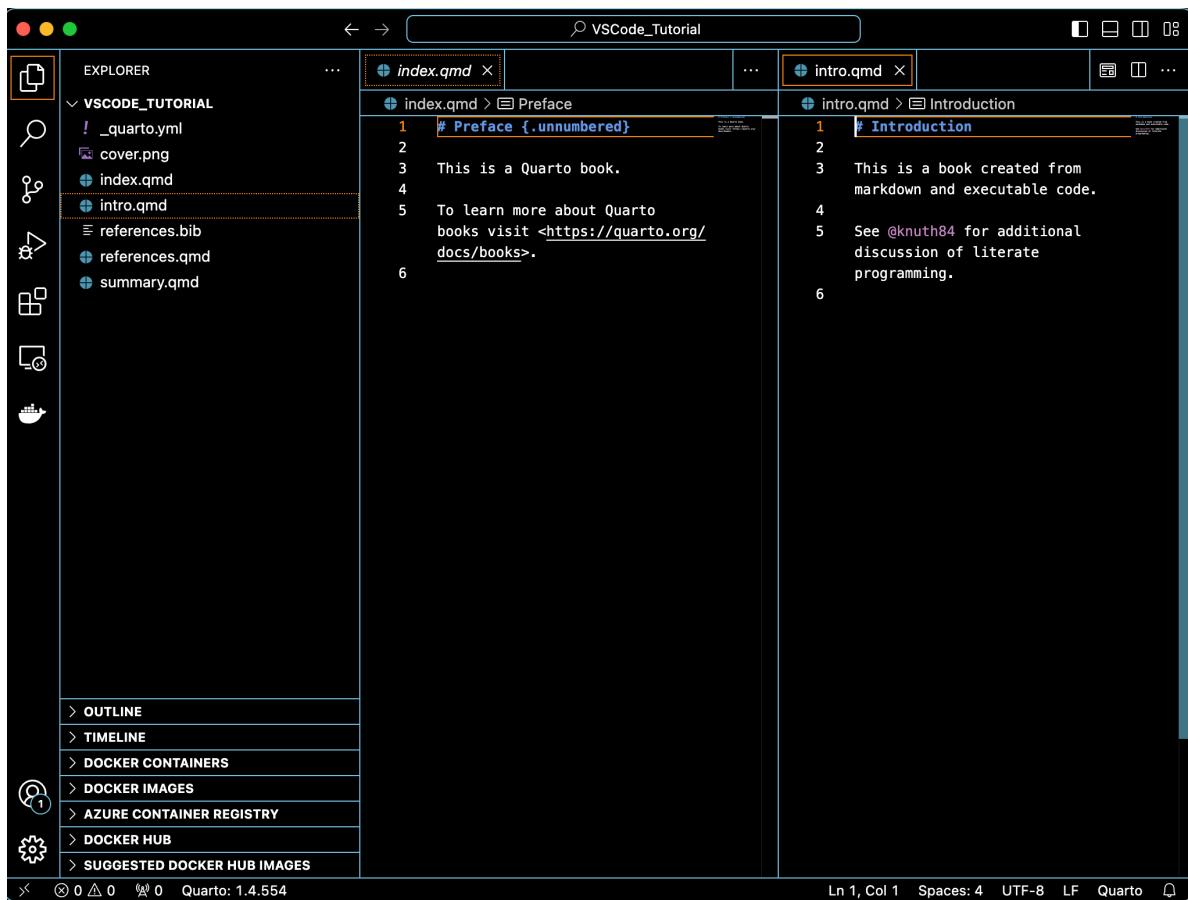
You can also use the Activity Bar to open a Search tool, Source Control, Debugging tool, Extensions, Remote Explorer (i.e. GitHub Codespaces), or Containers in the Primary Side Bar.

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 the right side of the Editor Groups bar.





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.

9.1.3 Install Extensions

When you first install VS Code, you should have no extensions. As different code will require different extensions, you will customize your 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](#)

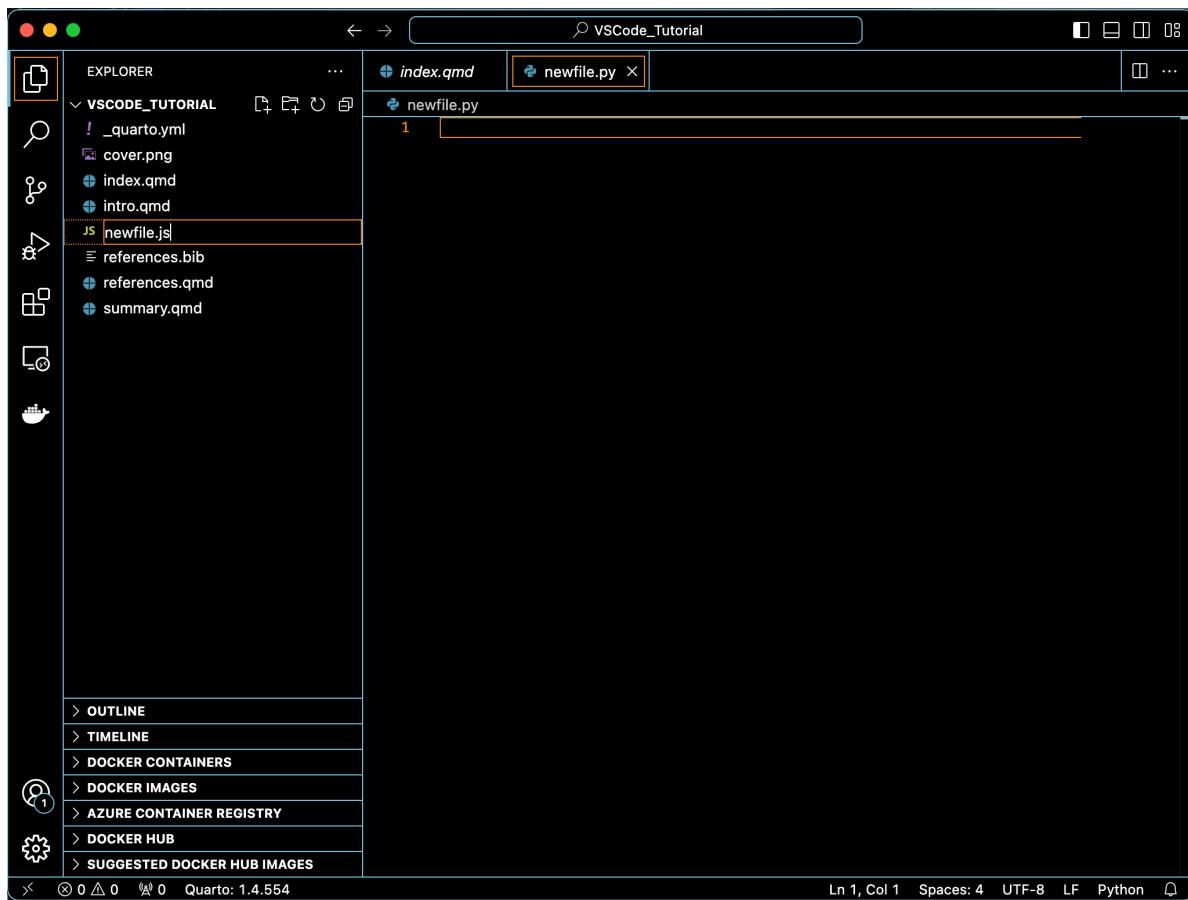
There are also some extensions dedicated to debugging which you may find useful, such as [R Debugger](#).

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

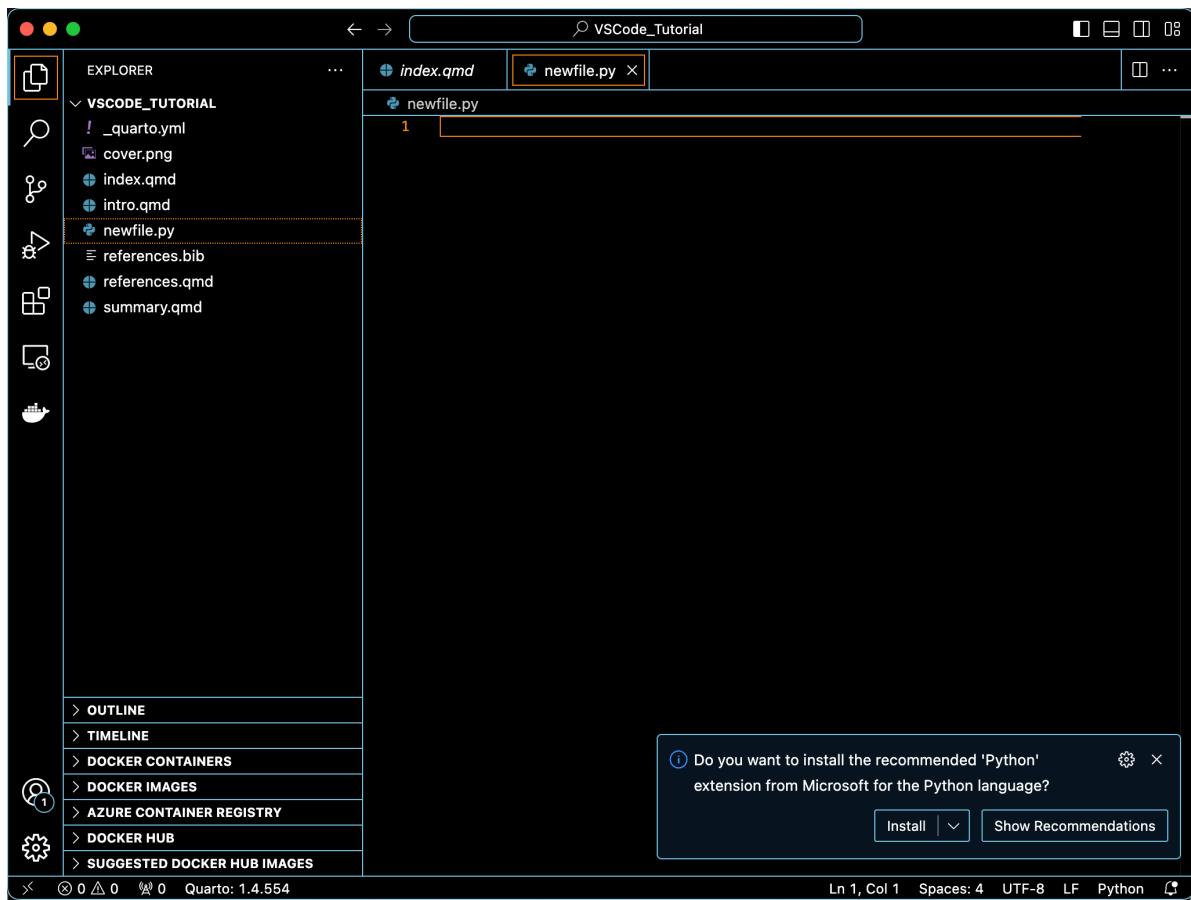
9.1.4 Manage files

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.

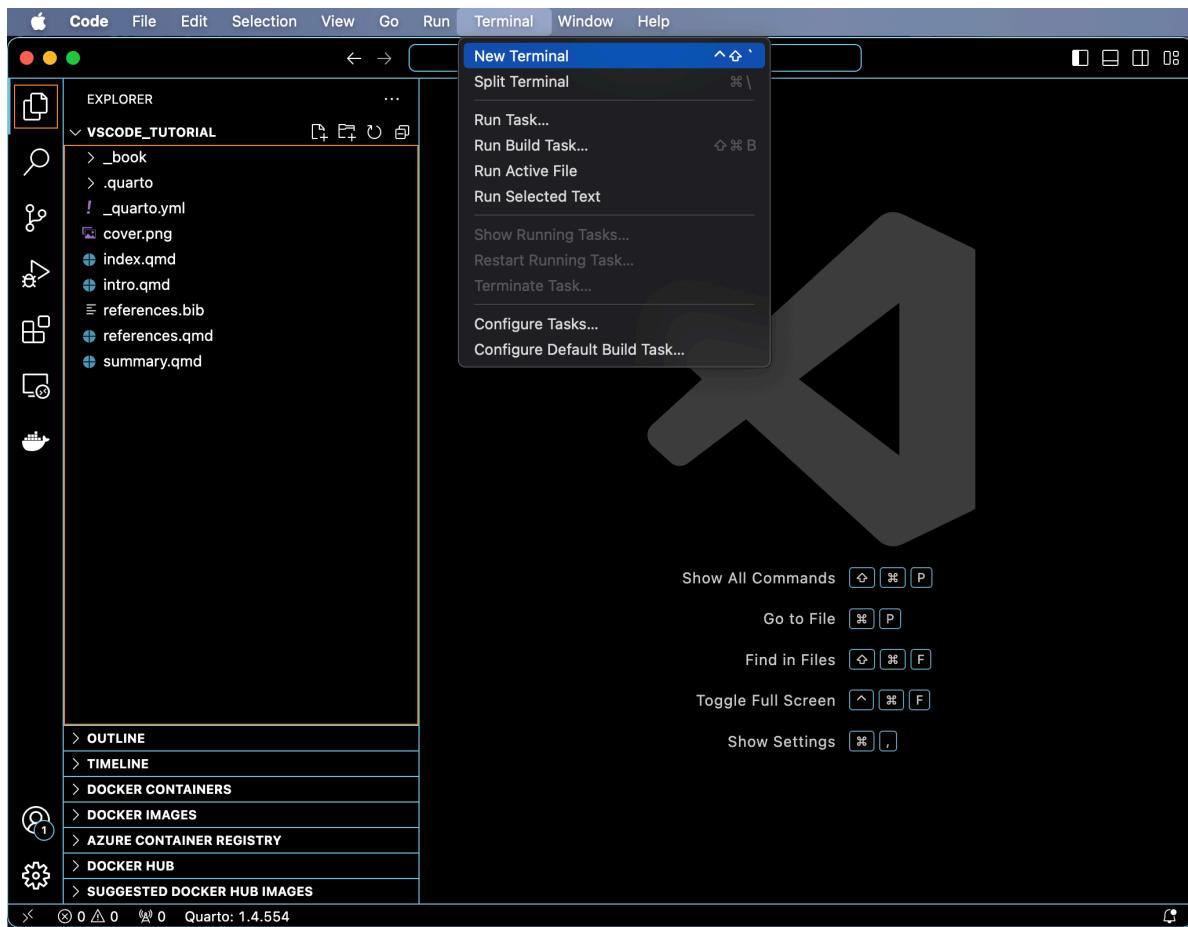


```
## Initialize Git
```

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
```

9.2 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:

9.2.1 Clear environment

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

9.2.2 Set output directory

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

9.2.3 Load libraries

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

9.2.4 Load dataset

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

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

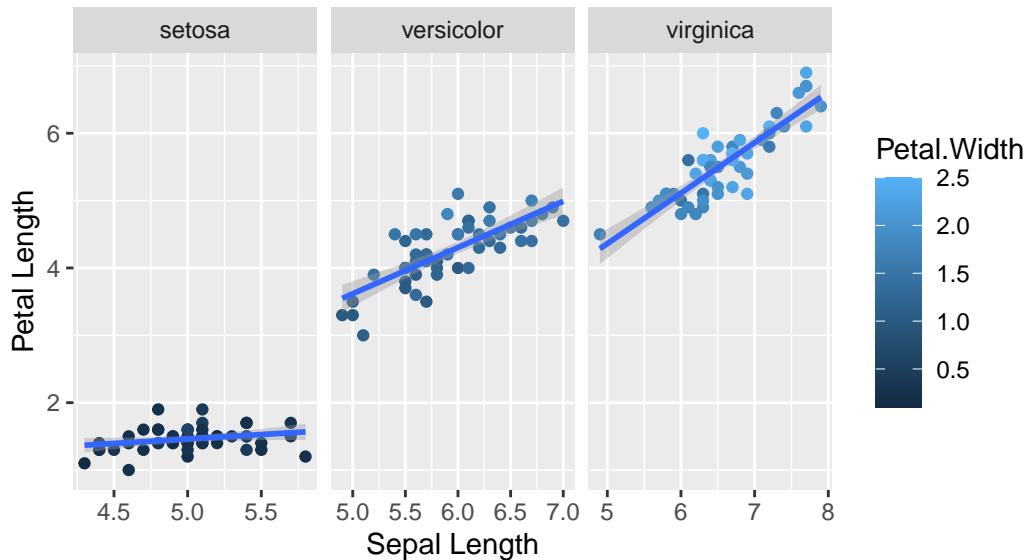
| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|--------------|-------------|--------------|-------------|---------|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |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

9.2.5 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 Length",
       facet_wrap(~Species,
                  scales = "free_x") +
  theme(plot.title = element_text(face = "bold"))
```

Petal vs. Sepal Length

Separated by Species



```
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.

9.3 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.

VS Code supports extensions for AI assistants that can help debug code as you work. Some options are below:

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.

Briefly:

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

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

3. Open your 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.

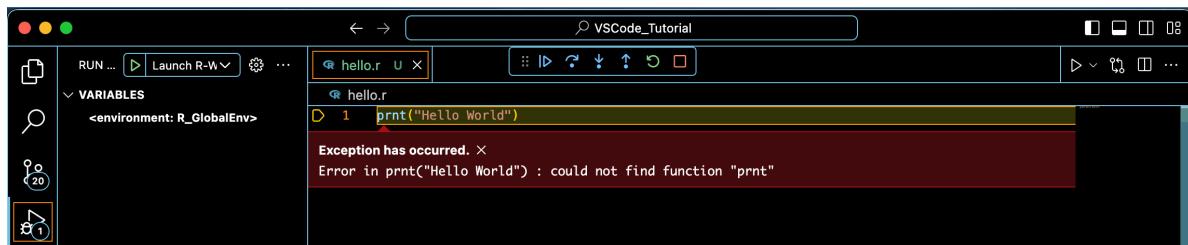
Let's try with a very simple example. Create a simple R script in your open directory:

```
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:



9.4 Cheatsheets

[VS Code Keyboard Shortcuts - macOS](#)

[VS Code Keyboard Shortcuts - Windows](#)

9.5 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.

10 Session Recordings

10.1 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.

10.2 2024 Session Recordings

10.2.1 Session 1: Intro to R and swirl

05 MAR 2024

[Session 1 Video Recording](#)

10.2.2 Session 2: MaRcus Training Course lesson 1

19 MAR 2024

[Session 2 Video Recording](#)

10.2.3 Session 3: MaRcus Training Course lesson 2

26 MAR 2024

[Session 3 Video Recording](#)

10.2.4 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.

10.2.5 Session 5: MaRcus Training Course lesson 5

09 APR 2024

[Session 5 Video Recording](#)

10.2.6 Session 6: MaRcus Training Course lesson 6

30 APR 2024

[Session 6 Video Recording](#)

10.2.7 Session 7: MaRcus Training Course lesson 7

07 MAY 2024

[Session 7 Video Recording](#)

10.2.8 Session 8: Exporting data from R

21 MAY 2024

[Session 8 Video Recording](#)

11 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>.