CRI Cancer Research Institute™

CRI Bioinformatics Bootcamp 2025

Wyndham Grand Orlando Resort Bonnet Creek

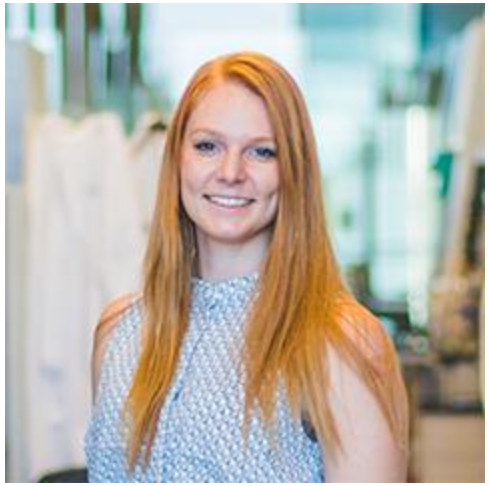
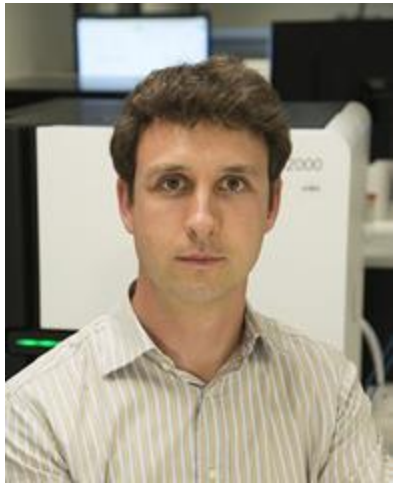
Orlando, Florida

May 17-22, 2025

Introduction

- Faculty
- Overview
- Resources
  - Slack
  - Course website
  - Post-its

Instructors

Introduction to R		RNA sequencing		Spatial
				
Nick Ceglia	Katie Campbell	Malachi Griffith	Obi Griffith	Nataly Naser Al Deen

Teaching Assistants

Christie Chang

Kelsy Cotto

Evelyn Schmidt

Kartik Singhal

Zachary Skidmore

Matthew Zatzman

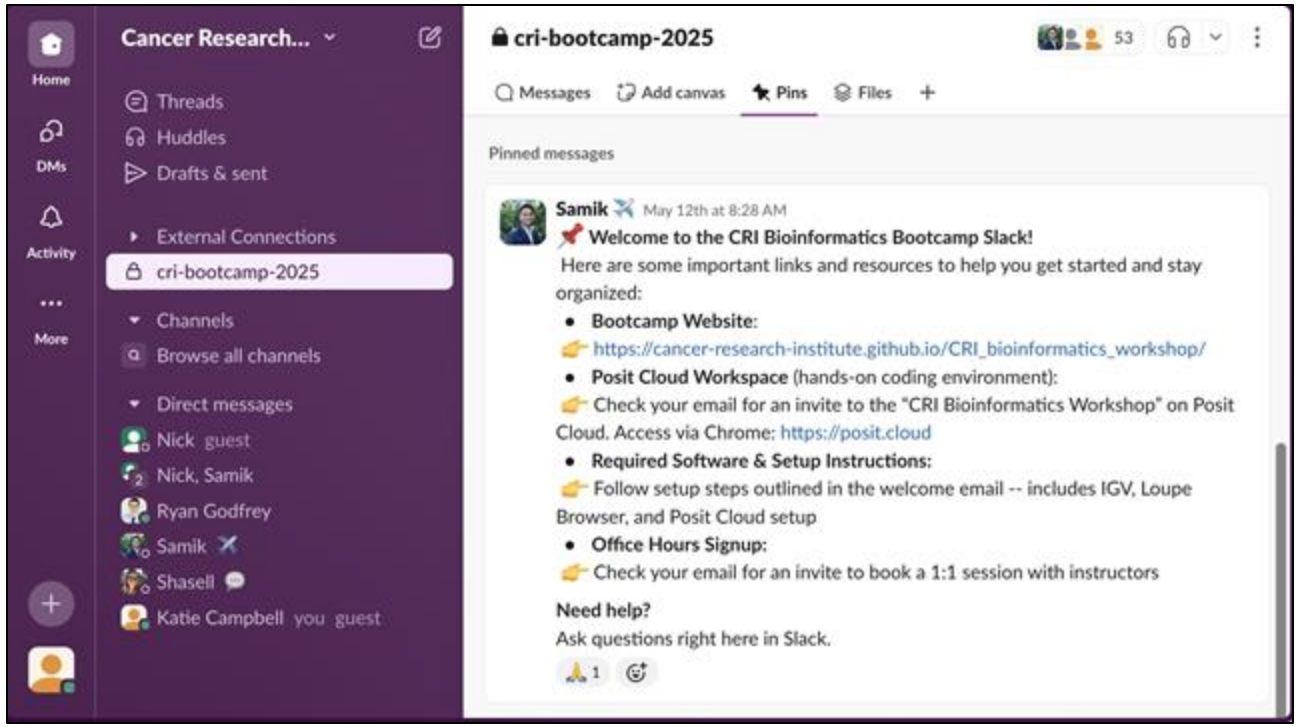
Overview

- Lectures
- Hands-on time
- Office hours

Day	Time	Duration	Module	Topic
1 (Sat)	9:00AM-9:30AM	0:30	R Workshop	Introduction to R and RStudio
	9:30AM-12:00PM	2:30		Hands-on: Reading, writing, and interpreting data structures
	12:00PM-1:30PM	1:30	Office Hours	Lunch
	1:30PM-2:15PM	0:45		Introduction to plotting and statistics
	2:15PM-3:15PM	1:00		Hands-on: Plotting information from data structures for real-time analysis
	3:15PM-3:30PM	0:15		Break
	3:30PM-5:00PM	1:30		Hands-on (cont'd): Plotting

Slack

- **cri-bootcamp-2025** (everyone)
  - Ask anything!
  - Tag @[Name] to call out an instructor or TA
  - Helpful hints
- **Direct messages** (1:1)
  - Directly message each other or faculty members
- **Pins**
  - Syllabus
  - Links to course website and Posit



Course website

CRI Bioinformatics Workshop Home Course Schedule Faculty Resources

Course overview

R Workshop (Day 1, Sat. - Day 2, Sun.)

1. Introduction to R
2. Basic plotting and statistics
3. Clustering concepts and correlation
4. Common challenges and additional resources

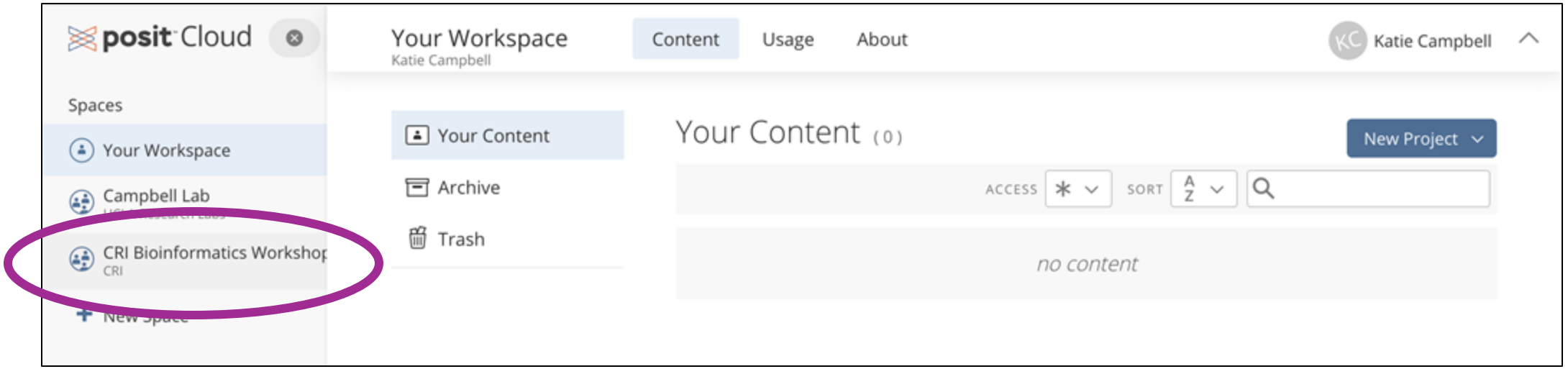
On this page

Use Post-its to ask for help!

If you need a TA, put the pink Post-it as a “flag” on your laptop.

Getting started in Posit (RStudio Server)

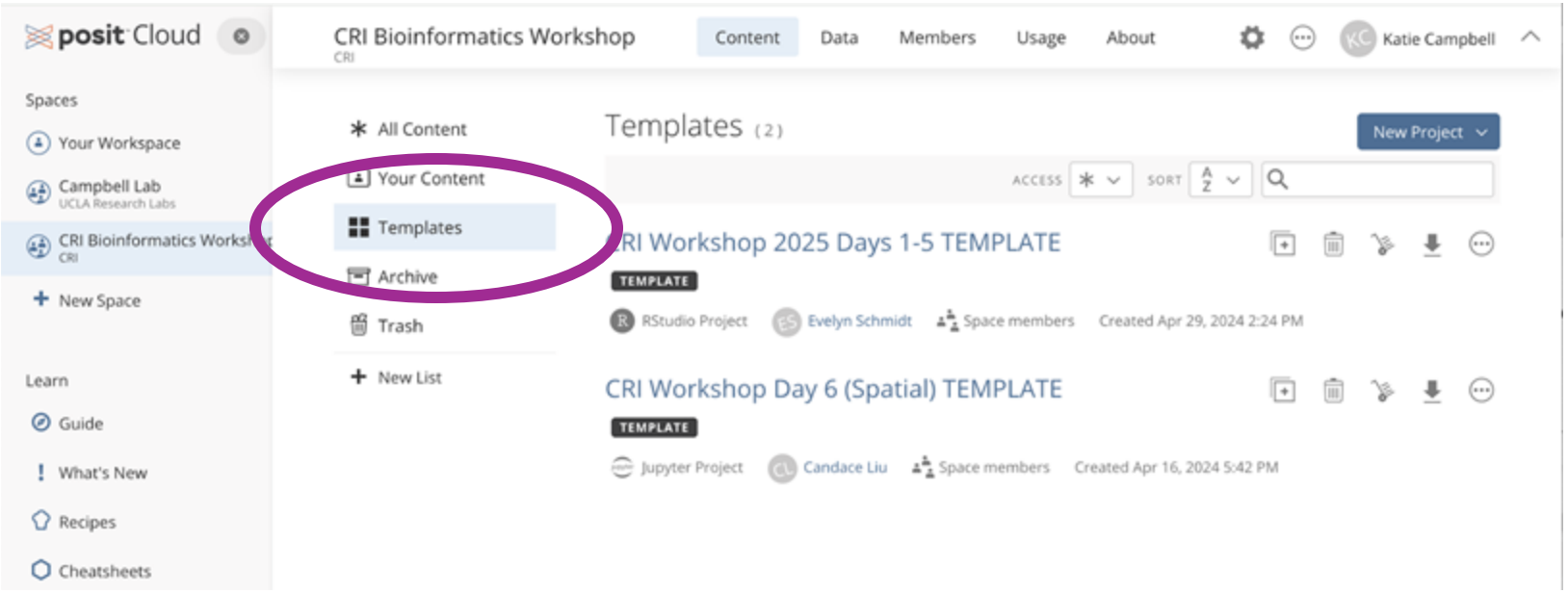
1. Go to posit.cloud and log in



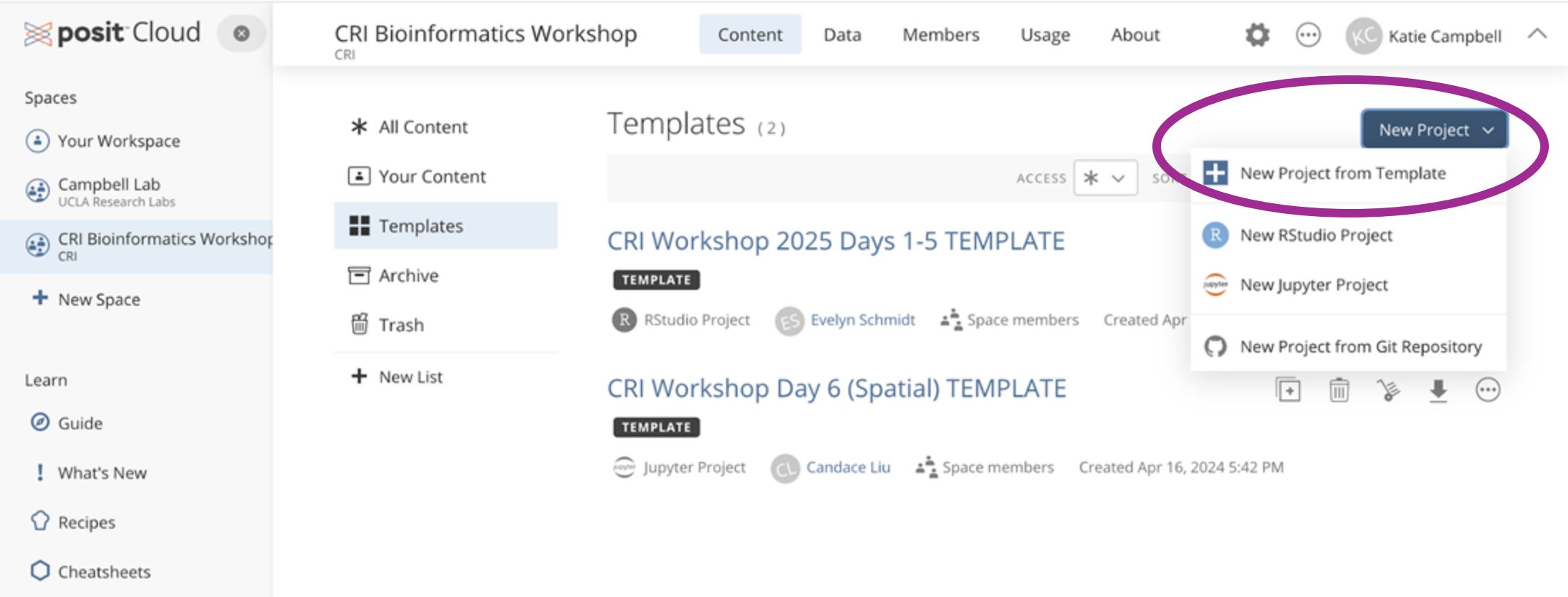
*If the **CRI Bioinformatics Workshop** space is not visible on the lefthand side, slack @Samik your email address.*

2. Find the Day 1-5 template

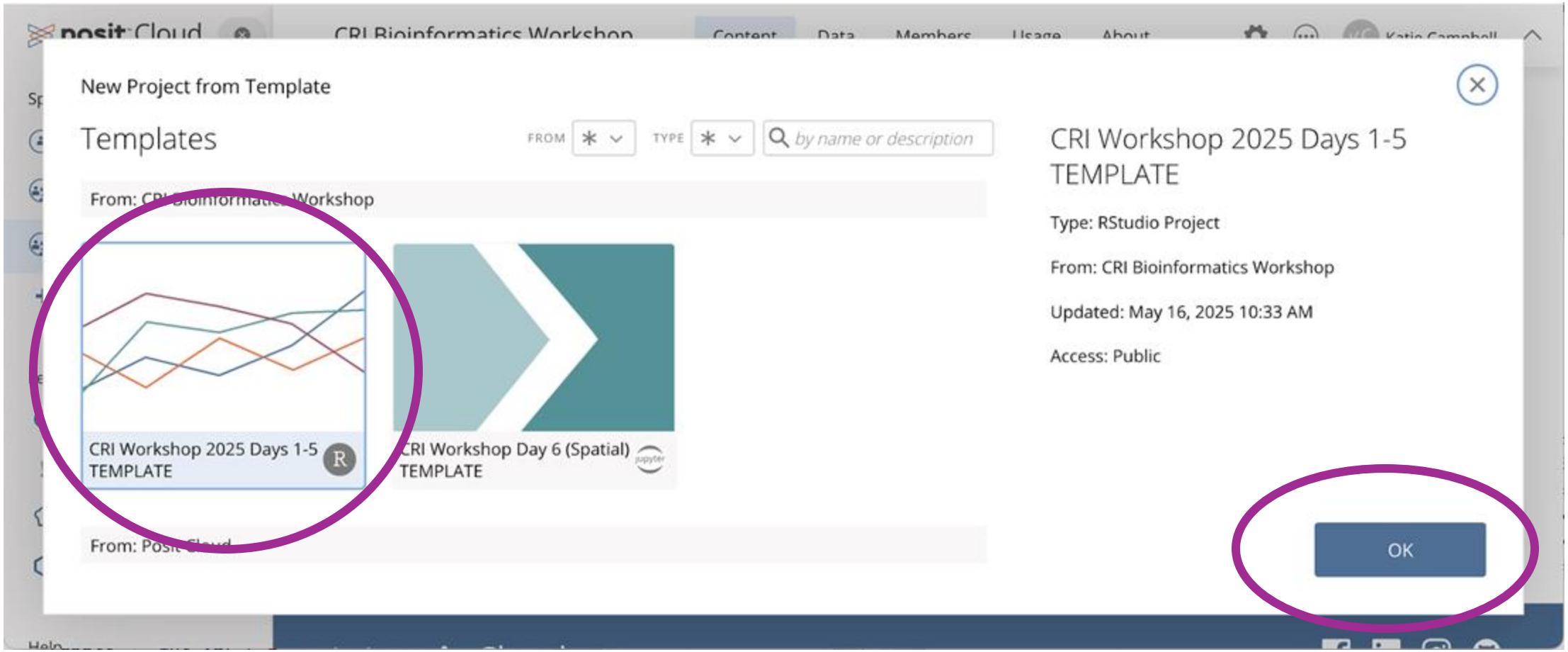
Select Templates



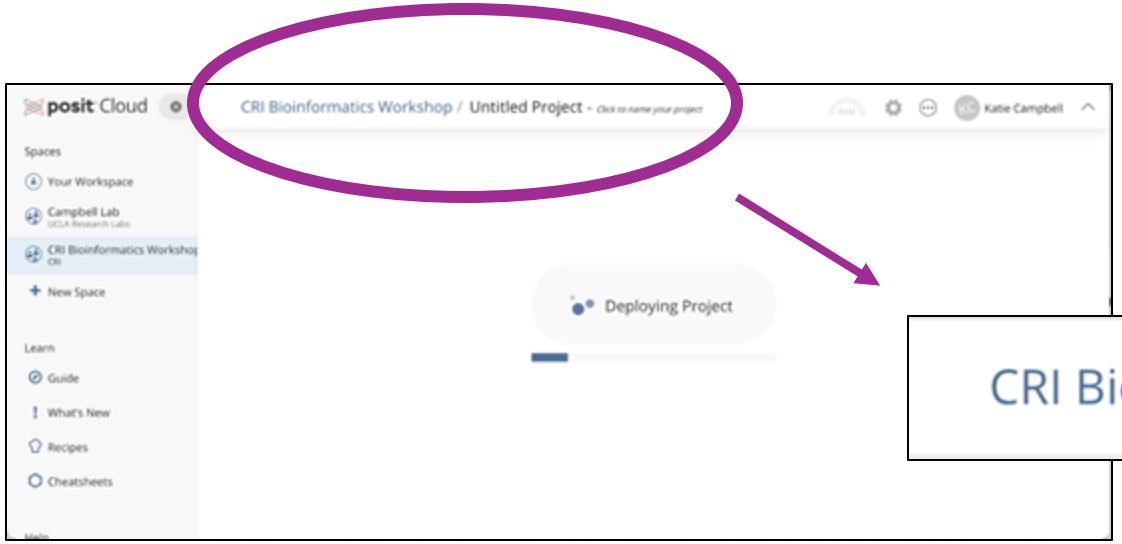
3. New Project □ New Project from Template



4. Select **CRI Workshop 2025 Days 1-5 TEMPLATE**



5. Rename project to **YOUR NAME**

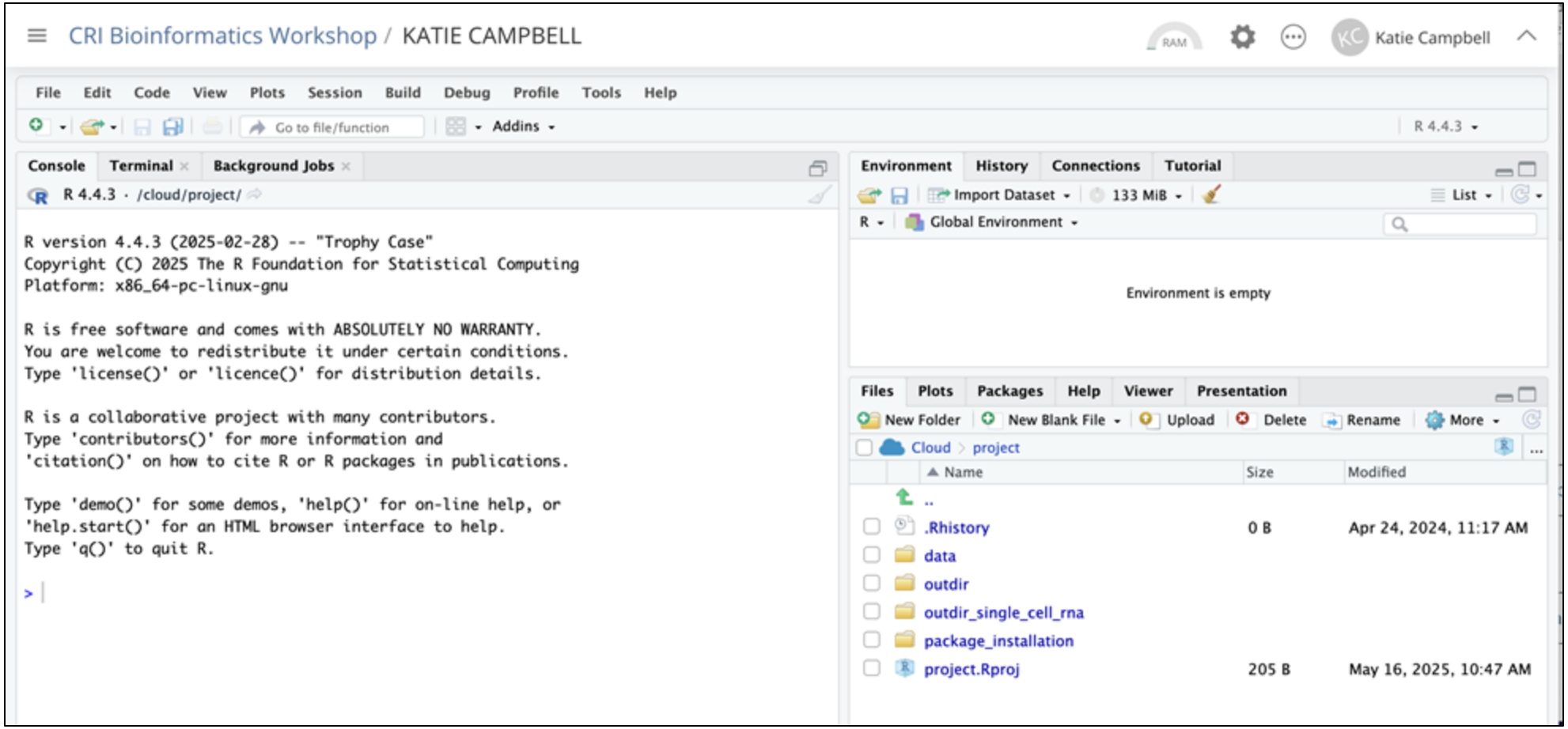


CRI Bioinformatics Workshop / Untitled Project ← *Click to name your project*

Deploying the project will take a minute, then…

CRI Bioinformatics Workshop / KATIE CAMPBELL

Only use **your** project for the course!



Goals for the R Workshop

1. Feel comfortable reading and running commands
2. Figuring out how to figure things out
3. Learn enough to be a little dangerous

Objectives for **Day 1**

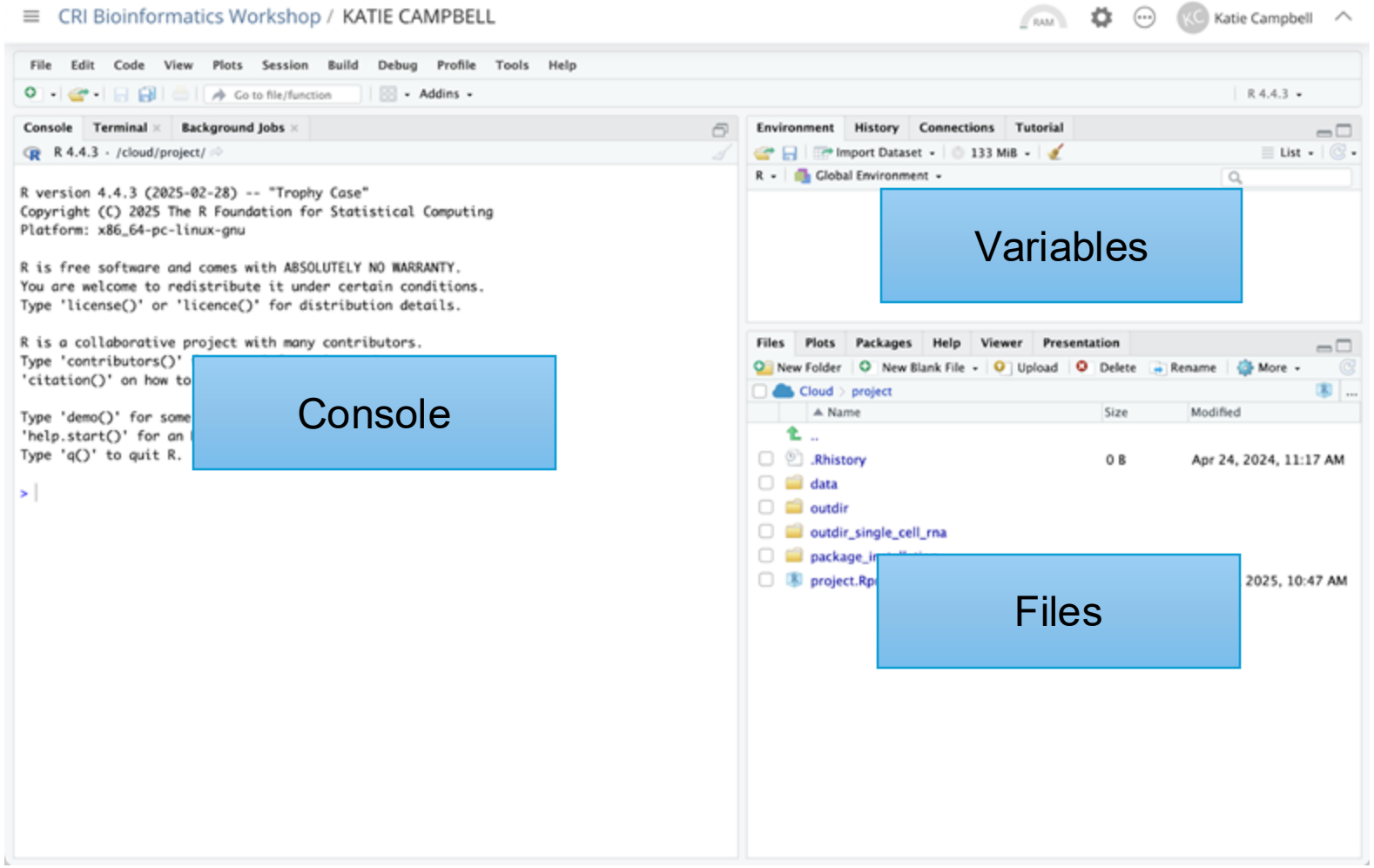
Introduction to R and Rstudio

- **Hands-on**: Reading, writing, and interpreting data structures

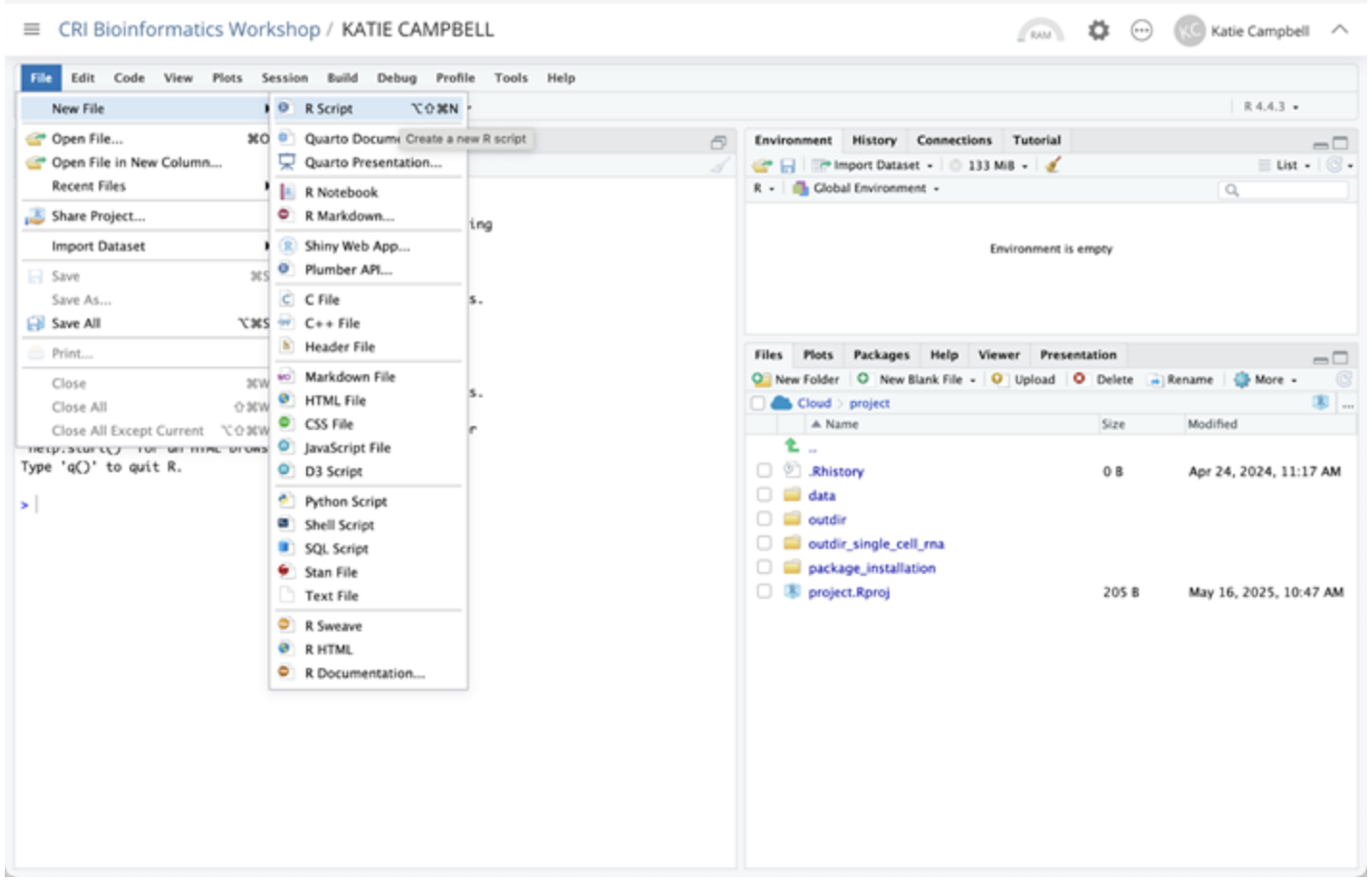
Introduction to plotting and statistics

- **Hands-on**: Plotting information from data structures for real-time analysis

RStudio Interface



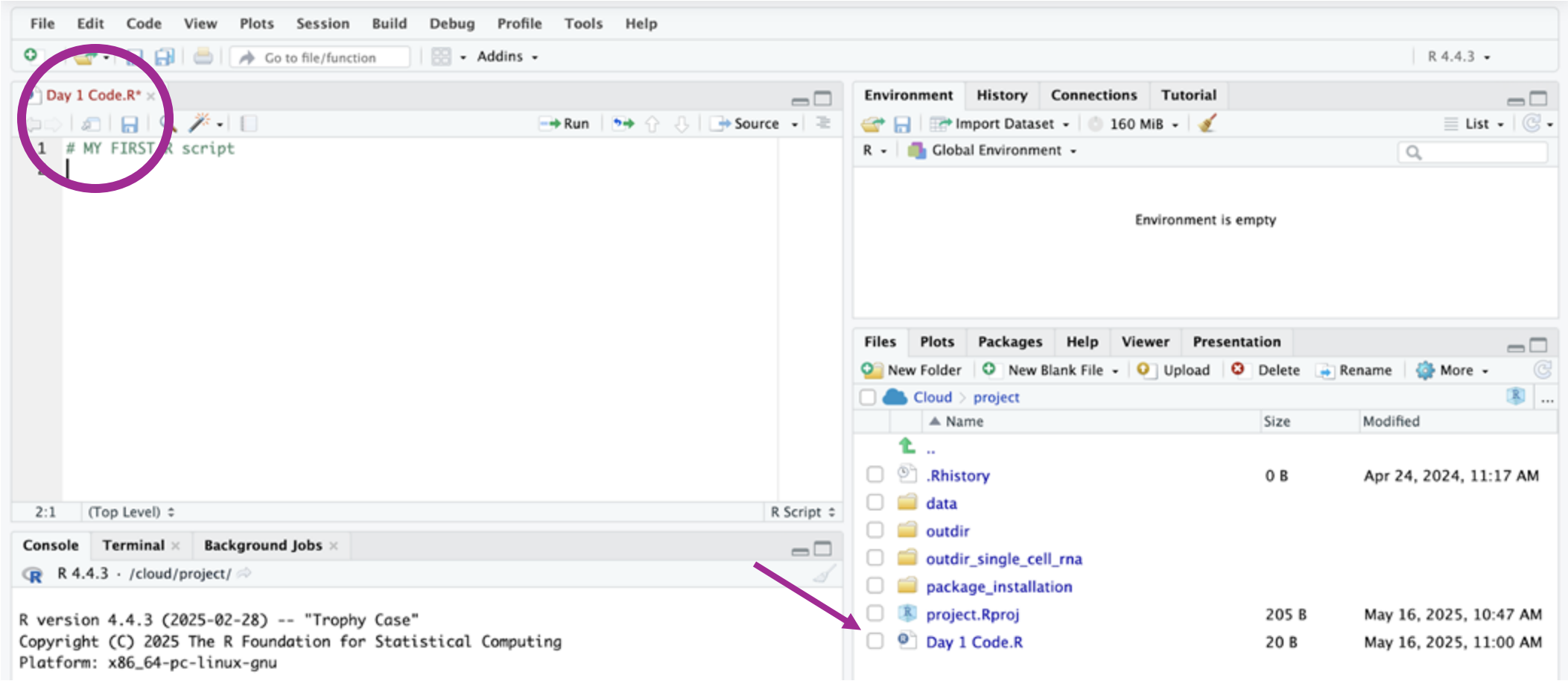
Creating a script



Saving code in the script vs coding in the console

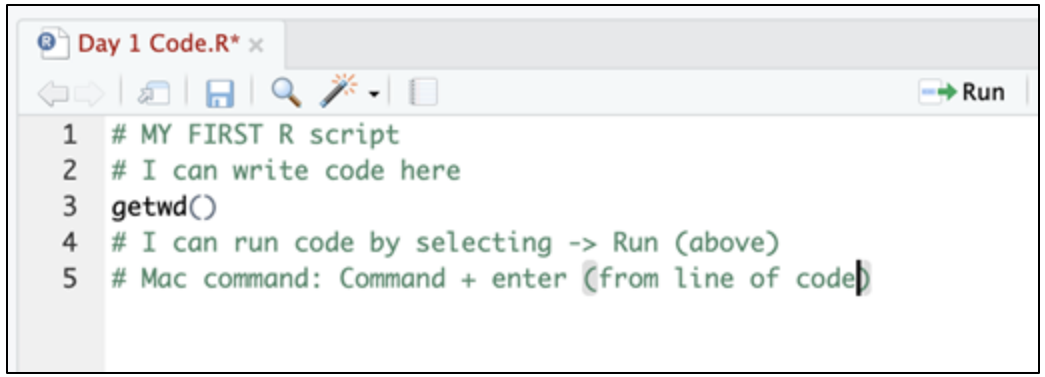
- Script
  - Anything you want to come back to later
  - Storing variables
  - Saving progress
- Console
  - Evaluating sizes, shapes, and summaries of data types
  - Listing
  - Testing code

Saving your first RScript

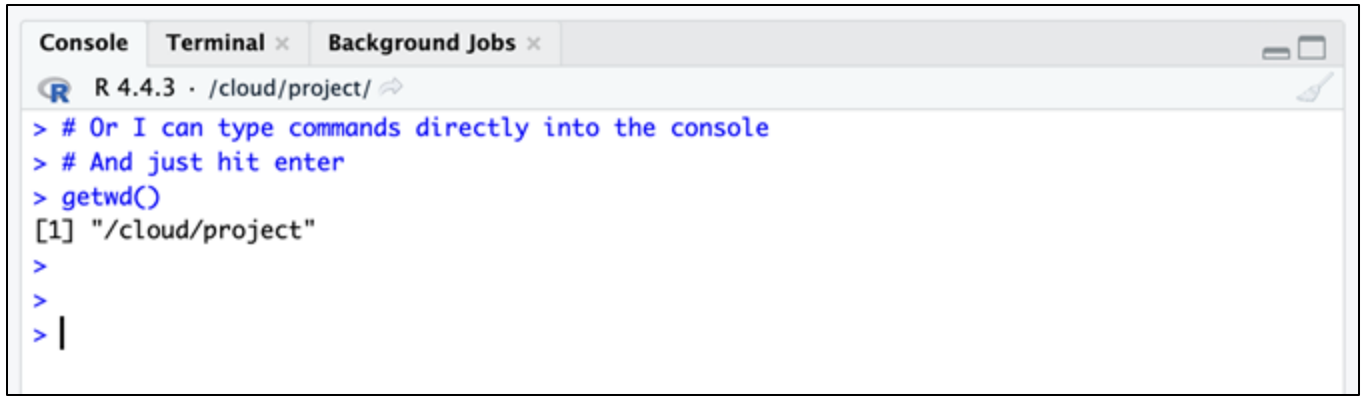


Running pieces of code

From a script



From the console



The working directory: Orienting the filesystem of your computer

```r
# Get the current working directory
current_dir <- getwd()
print(paste("Current directory:", current_dir))

# List files in the current directory
files <- list.files()
print("Files in the current directory:")
print(files)

# Change the current directory
new_dir <- "path/to/new/directory"
setwd(new_dir)
print(paste("Changed directory to:", new_dir))
```

The working directory: Orienting the filesystem of your computer

```r
# Get the current working directory
current_dir <- getwd()
print(paste("Current directory:", current_dir))

# List files in the current directory
files <- list.files()
print("Files in the current directory:")
print(files)

# Change the current directory
new_dir <- "path/to/new/directory"
setwd(new_dir)
print(paste("Changed directory to:", new_dir))
```

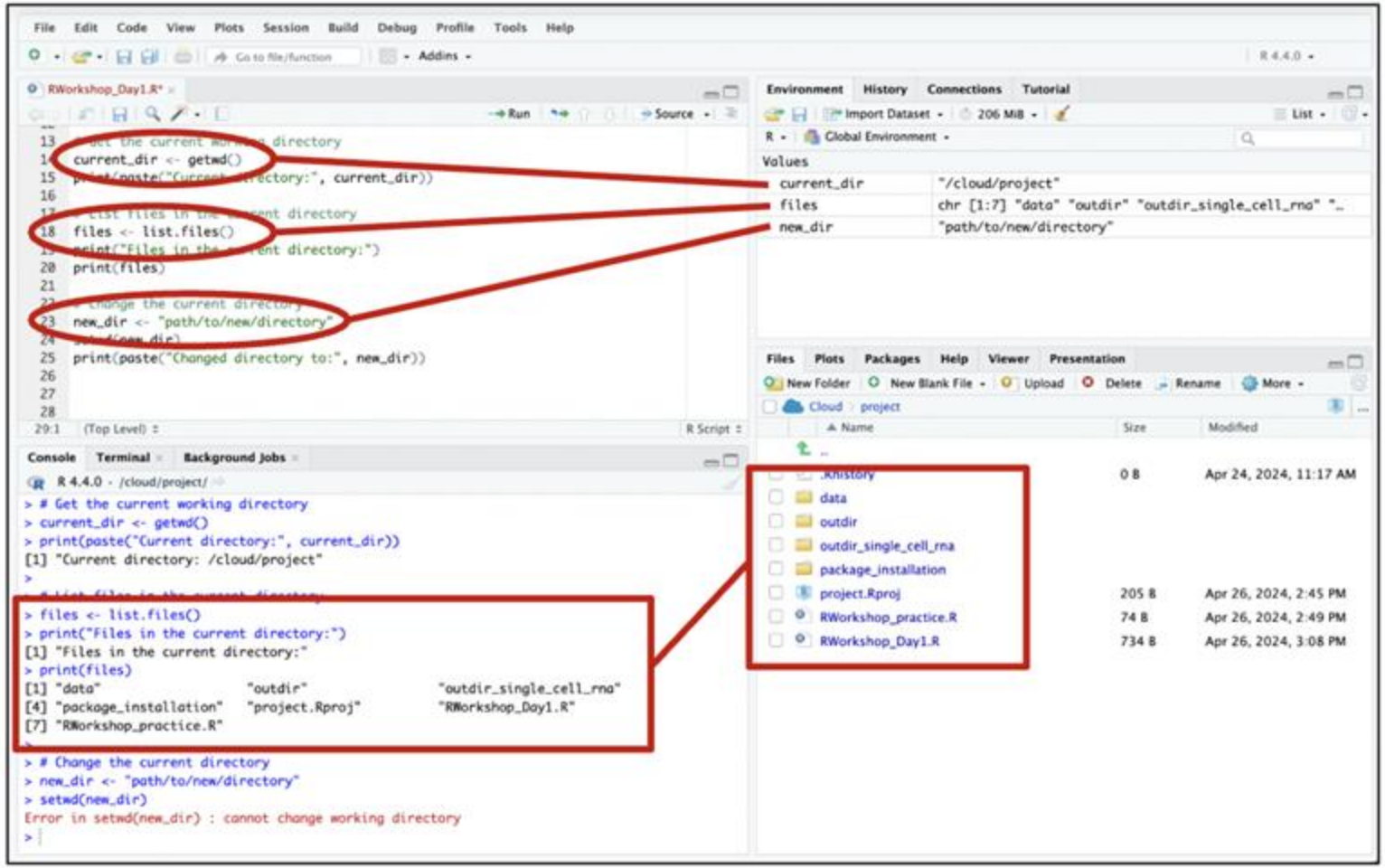
Functions are indicated by function_name(parameters)

paste() can be used to concatenate strings together

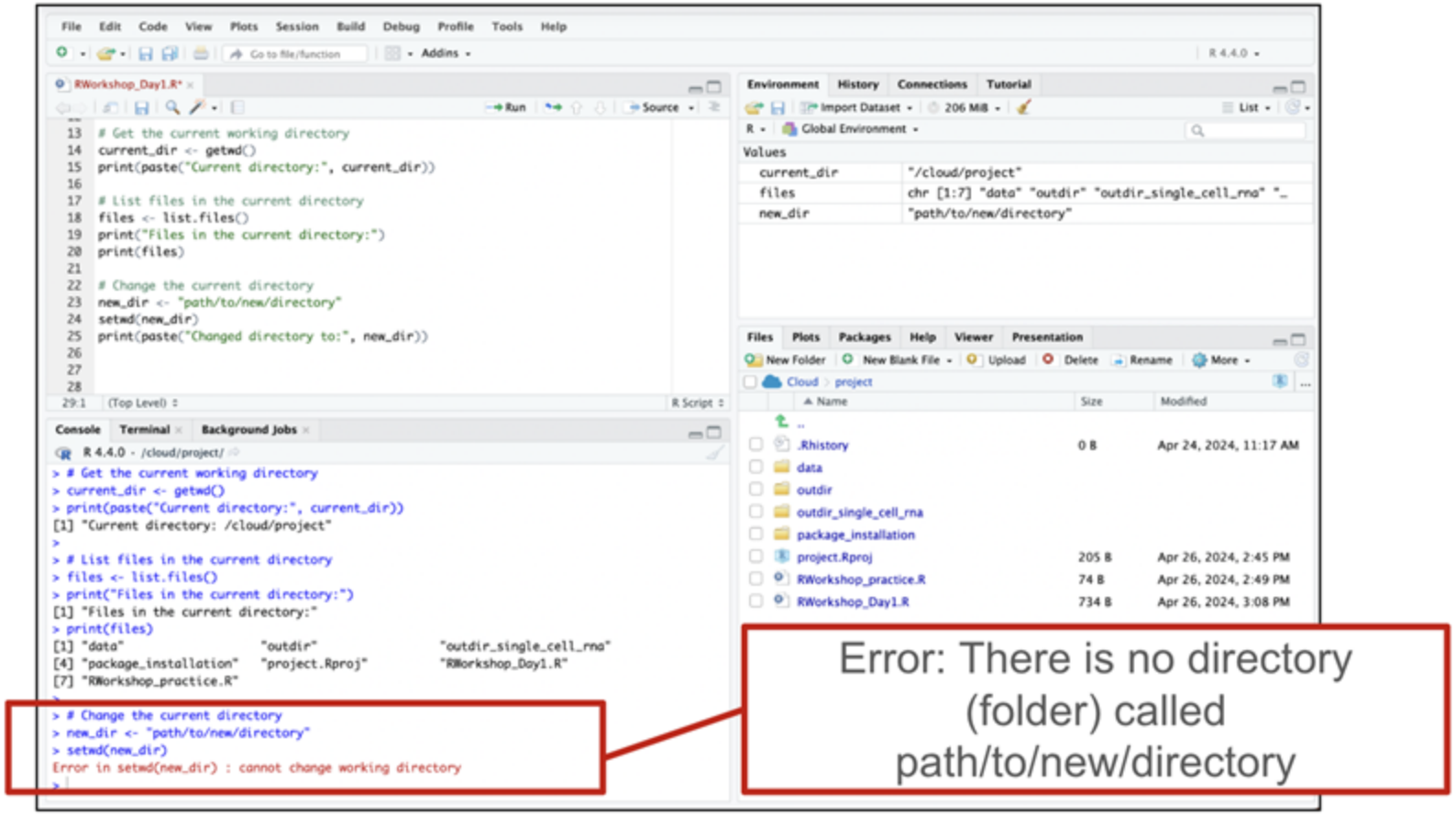
print() will print out what's inside the parentheses into the console

<- assigns the content on the right (the string) to the variable name on the left

Assigning variables saves them in your global environment



Be prepared for Errors…



Packages in R

- Base R ships with many built-in functions.
- Packages (or “libraries”) are collections of functions, data, and documentation bundled together to extend R’s capabilities.
- The Comprehensive R Archive Network (CRAN) hosts packages that can be installed directly.

```
> install.packages("pkgname")
```

- Every CRAN package ships with built-in documentation and vignettes.

```
> ?function_name
> browseVignettes("pkgname")
```

Using R Packages

- **Install** once per machine or environment.
- **Load** with library(pkgname) every new R session to make its functions available

```
## 1. Install  (only once per machine / environment)
install.packages("tidyverse")   # installs in ggplot2, dplyr, readr, etc.

## 2. Load (every new R session)
library(tidyverse)  # attaches the core tidyverse packages to search path
```

- **Common Error:**

```
> mutate()
Error in mutate(): could not find function "mutate"
```

- **Solution:** Install and reload package.

** All of the packages you need for this course are already installed.*

More Package Repositories

Bioconductor

- An open-source ecosystem of 3000+ R packages dedicated to genomics, single-cell, proteomics, epigenetics, etc.

```
if (!requireNamespace("BiocManager")) install.packages("BiocManager")
BiocManager::install("packageName")
```

Devtools

- install packages from **GitHub, GitLab, Bitbucket, local folders, specific versions, or tarballs**—handy for cutting-edge code not yet on CRAN/Bioconductor.

```
devtools::install_github("user/repo")          # latest master/main
devtools::install_github("user/repo@v1.2.0")
```

Variables in R

- A **variable** is a human-readable name you attach to a piece of data so you can use and manipulate that data.
- Each variable is associated with a spot in **memory**; you can swap what's stored there, move it around, or read it back whenever you need.
- Every **algorithm** can be reduced down to creating, updating, and comparing variables.

```
# Basic assignment styles
x <- 10          # left-arrow (most idiomatic)
y = x + 5        # equals works too
99 -> z          # right-arrow (rare, but legal)
x <- x * 2       # Reassignment (overwrite the same name)
```

Primitive Variables

- **Atomic building blocks** – R's *primitive* (atomic) types sit at the bottom of every data structure.
- **Six core kinds** – logical, integer, double (numeric), character, complex, raw.
- Atomic variables are further combined to build higher order complex **data structures** - like matrices and dataframes.
- All represented as length 1 vectors.

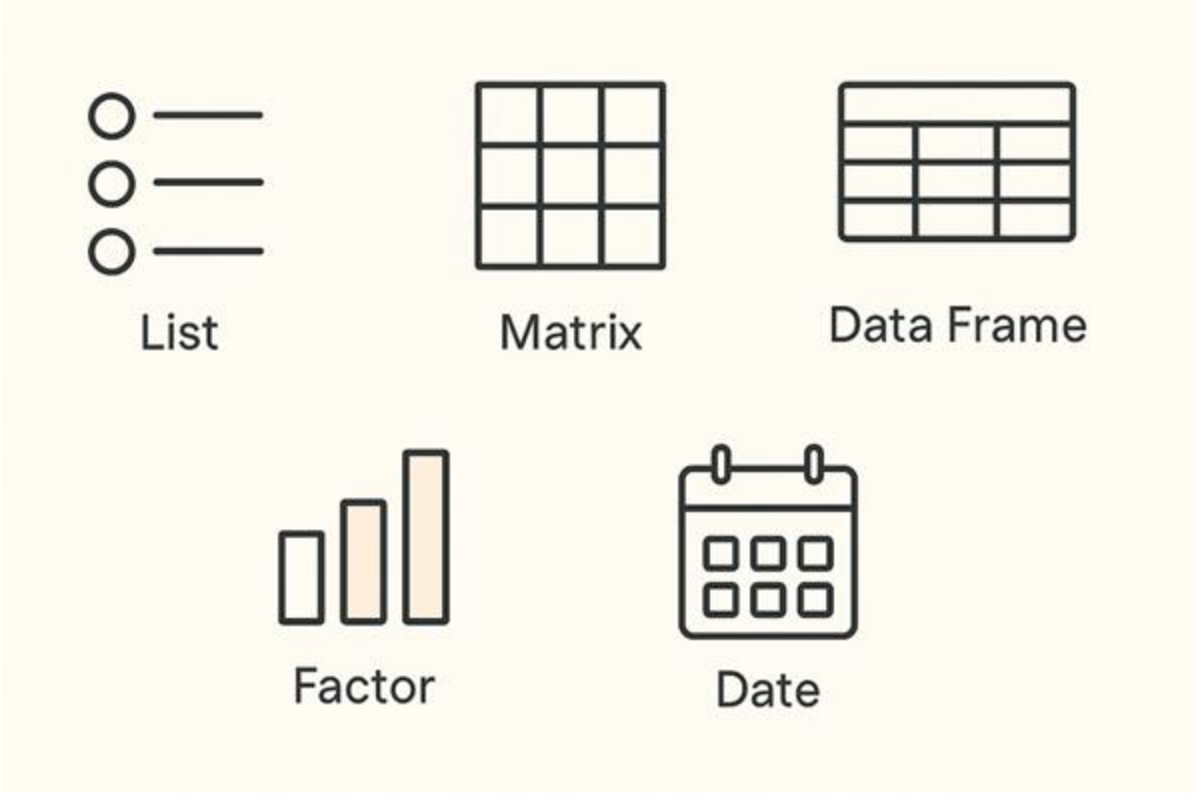
Primitive variables in R

- **Numeric (*Float*)**
  - Default variable for numbers
  - Includes floating point and integer
- **Integer**
  - Whole number
  - Efficient
- **Logical (*Boolean*)**
  - TRUE or FALSE
  - Conditional Statements
- **Character (String)**
  - Text data
- **Other types**
  - Raw (Byte Representation)
  - Complex

```r
# —— Logical: for true/false flags, filtering & control flow ——————————
flag <- TRUE          # common in if-statements, masks, comparisons
print(flag)           # [1] TRUE

# —— Integer: whole-number counts, indices, factors, IDs ——————————————
count <- 42L          # the trailing L forces storage as integer
print(count)          # [1] 42

# —— Double / Numeric: real-valued measurements, calculations ————————
pi_val <- 3.14159     # default numeric type in R is double precision
print(pi_val)         # [1] 3.14159

# —— Character: text labels, categorical names, file paths, messages ——
greet <- "Hello, R!"  # character vectors store strings of any length
print(greet)          # [1] "Hello, R!"
```

Data Structures

- **Aggregates primitives:** higher-order structures (lists, data frames, matrices) *combine* many primitive values into one organized object.
- **Add shape or metadata:** they supply dimensions (rows × cols), names, or class attributes so functions “know” how to treat the data.
- **Enable richer operations:** with structure comes methods—sorting a data frame, subsetting a matrix, plotting a ggplot object—capabilities primitives alone don’t provide.



Lists and Factors

- **Vector**
  - 1 dimensional
  - Single data type
    - Numeric, Character, Logical
  - Efficient (pre-allocation)
  - Building Matrices
- **List**
  - Heterogeneous data
  - Dynamic
  - Creating Dataframes
- **Factor**
  - Categorical data
  - Levels
  - Memory efficient

```r
# Lists
expr_counts <- c(7.2, 5.9, 8.1)
umi_counts  <- c(1500L, 2300L, 1800L)
qc_pass     <- c(TRUE, TRUE, FALSE)
tcr_seq     <- c("CASSLGQGAETQYF", "CASSPTGGYNEQFF", "CASSIRSSYEQYF")

# Lists
cell1 <- list(
  tcr_beta    = "CASSLGQGAETQYF",  # character
  umi_count   = 1500L,             # integer
  expr_avg    = 7.2,               # double
  is_T_cell   = TRUE,              # logical
  phenotype   = factor("Naive", levels = levels(phenotype))
)

# Factors
phenotype <- factor(
  c("Naive", "Memory", "TEMRA", "Memory", "Naive"),
  levels = c("Naive", "Memory", "TEMRA", "Exhausted")  # predefined order
)
```

Matrix

- 2 dimensional
- Single data type
- ***matrix()***
  - data
  - nrow
  - ncol
  - byrow
    - Whether to fill by rows or columns
  - dimnames
- Access elements by row and column
- Matrix operations

```r
################################################################################
# Create a 3 × 3 numeric matrix
################################################################################
mat <- matrix(
  c(1, 2, 3,
    4, 5, 6,
    7, 8, 9),
  nrow = 3, byrow = TRUE
)

################################################################################
# Column sums and row sums
################################################################################
print(colSums(mat))   # [1] 12 15 18
print(rowSums(mat))   # [1]  6 15 24

################################################################################
# Transpose (swap rows ⇔ cols)
################################################################################
t_mat <- t(mat)
print(t_mat)
#      [,1] [,2] [,3]
# [1,]    1    4    7
# [2,]    2    5    8
# [3,]    3    6    9
```

Matrix Arithmetic

- **Element-wise math**: mat + 2, mat * 3, mat1 / mat2 operate cell-by-cell when dimensions match.
- **True matrix multiply**: use %*% (e.g., A %*% B) for linear-algebra dot products; dimensions must align (n × k × k × m → n × m).
- **Transpose first**: t(A) flips rows ↔ cols.
- **Power & outer products**: A ^ 2 squares each entry; outer(x, y, "*") builds a full matrix from two vectors.
- **Element-wise vs. dot**: * is element-wise, %*% is dot—pick the right one to avoid silent shape errors.

```
##############################################################################
# Matrix arithmetic
##############################################################################
# Element-wise operations
print(mat + 10)       # add scalar
#     [,1] [,2] [,3]
# [1,]   11   12   13
# [2,]   14   15   16
# [3,]   17   18   19

print(mat * 2)        # multiply each element
#     [,1] [,2] [,3]
# [1,]    2    4    6
# [2,]    8   10   12
# [3,]   14   16   18

# Matrix multiplication (3×3 %*% 3×3 → 3×3)
prod <- mat %*% t_mat
print(prod)
#      [,1] [,2] [,3]
# [1,]   14   32   50
# [2,]   32   77  122
# [3,]   50  122  194
```

Indexing Matrices

- **[row, col] syntax** – mat[2,3] pulls element at 2nd row, 3rd column; leave one side blank for full row/col (mat[,2]).
- **Slices & sets** – feed vectors: mat[1:2, 2:3] (ranges) or mat[c(1,3), c(1,3)] (non-adjacent picks).
- **Negative indices** – mat[-1,] drops the 1st row; mat[, -c(2,3)] keeps everything except those columns.
- **Logical masks** – apply conditions: mat[mat[,1] > 5 ,] filters rows where column 1 exceeds 5.
- **Named access** – set rownames() / colnames() and use strings: mat["sample3", "geneA"].
- **Dimension recycling** – remember element-wise ops (mat + 1) recycle scalars to fit, but indexing always respects exact row/col vectors—mismatched lengths throw errors, not warnings.

```r
## 1) Single element  (row, col)
mat[2, 3]           # 6      (2nd row, 3rd column)

## 2) Whole row / whole column
mat[1, ]            # 1 2 3  (first row as vector)
mat[ ,2]            # 2 5 8  (second column)

## 3) Multiple, non-adjacent rows/cols
mat[c(1, 3), c(1, 3)]

## 4) Exclude a row or column with negative index
mat[-2, ]           # drop 2nd row
mat[ , -1]          # drop 1st column

## 5) Logical row/col selection
rows_keep <- mat[ ,1] > 3     # TRUE FALSE TRUE   (rows where first col > 3)
mat[rows_keep, ]

## 6) Slice with sequence / colon
mat[1:2, 2:3]       # upper-left 2 × 2 block of cols 2–3

## 8) Name rows/cols, then access by names
rownames(mat) <- paste0("r", 1:3)
colnames(mat) <- paste0("c", 1:3)

mat["r2", "c1"]      # 4
mat[ , c("c2", "c3")]
```

Dataframe

- **Tabular container** – each column is a vector of equal length; rows are observations, columns are variables.
- **Heterogeneous by column** – numeric, character, logical, factor vectors can sit side-by-side.
- **Base-R workhorse** – easy CSV I/O, subsetting, compatible with modeling and plotting functions.

- **Column access** – df$phenotype pulled a full column vector.
- **Element access** – df[1, 2] retrieved the value in row 1, column 2 ("TEMRA").
- **Row filtering** – subset(df, treatment == "Post") returned only “Post” samples.
- **Adding a column** – df$site <- … appended a new vector, instantly widening the table.
- **Appending a row** – df <- rbind(df, new_row) stacked an additional observation, lengthening the table.
- **Editing a cell** – df[4, "site"] <- "Tumor" overwrote a single value by row & column label.

```r
df <- data.frame(
  barcode   = c("AAACCTGAGAGGCCT-1",
                "AAACCTGCAAGTTGTC-1",
                "AAACCTGCACCGATAT-1"),
  phenotype = c("TEMRA", "Naive", "Dysfunctional"),
  treatment = c("Pre",   "Post",  "Pre"),
  response  = c("R",     "NR",    "R"),
  stringsAsFactors = FALSE
)

phenotypes <- df$phenotype          # TEMRA  Naive  Dysfunctional
print(phenotypes)

cell_val <- df[1, 2]                # row 1, column 2  →  "TEMRA"

post_cells <- subset(df, treatment == "Post")

df$site <- c("Tumor", "Tumor", "Blood")

new_row <- data.frame(
  barcode   = "AAACCTGCACCGATAT-1",
  phenotype = "Naive",
  treatment = "Post",
  response  = "R",
  site      = "Tumor",
  stringsAsFactors = FALSE
)
df <- rbind(df, new_row)

df[4, "site"] <- "Tumor"            # overwrite the 4th row's site
```

Cancer Research Institute™

String manipulation

- Strings vs substrings
  - Substring is a sequence of text within a string.
  - “Hello” is a substring of “Hello world!”
- Case conversion
  - Uppercase and lower matter, R is case-sensitive
- Find and replace
  - Find a substring and replace it with another substring
  - Examples: ID matching, motifs, etc.
- Length calculation
  - Sequence length
- Stringr package (tidyverse)
  - str_length
  - str_to_upper
  - str_locate
  - str_sub
  - str_replace and str_replace_all
- Base: grep, grepl, gsub
- Regular expressions
  - Search pattern
  - “hell*olrd!” matches “hello world!”e

```r
library(stringr)

# Example sequence
sequence <- "ATGCGTACGTGACT"

# Length
sequence_length <- str_length(sequence)
print(sequence_length)          # 14

# Case
sequence_upper <- str_to_upper(sequence)
print(sequence_upper)           # "ATGCGTACGTGACT"

sequence_lower <- str_to_lower(sequence)
print(sequence_lower)           # "atgcgtacgtgact"

# Substring Extraction (positions 3 to 8)
extracted_sequence <- str_sub(sequence, 3, 8)
print(extracted_sequence)       # "GCGTAC"

# Replase
dna_sequence <- "ATGCGTACGTTGACT"
rna_sequence <- str_replace_all(dna_sequence, "T", "U")
print(rna_sequence)             # "AUGCGUACGUUGACU"
```

Cancer Research Institute™

Logical operations

- Evaluate to True or False
  - logical data type
- Use case
  - Conditional statements
    - if / else
  - Filtering data
    - subset(df,treatment="post")
- Comparisons
  - Equal: ==
  - Not equal: !=
  - Greater Than: ">"
  - Less Than "<"
  - %in%
    - Test inclusion of one vector in another
- Boolean statements
  - and: both are true "&"
  - or: atleast one is true "|"
  - not: evaluates to false "!"

```r
# Greater than
result_gt <- x > y
print(result_gt)  # Output: FALSE

# Less than
result_lt <- x < y
print(result_lt)  # Output: TRUE

# Equal to
result_eq <- x == y
print(result_eq)  # Output: FALSE

# Not equal to
result_neq <- x != y
print(result_neq)  # Output: TRUE
```

```r
# Logical operations
a <- TRUE
b <- FALSE

# AND operation
result_and <- a & b
print(result_and)  # Output: FALSE

# OR operation
result_or <- a | b
print(result_or)   # Output: TRUE

# NOT operation
result_not <- !a
print(result_not)  # Output: FALSE
```

```r
x <- 3
if (x > 5) {
  print("x is greater than 5")
} else if (x == 5) {
  print("x is equal to 5")
} else {
  print("x is less than 5")
}
```

Writing your own functions

- Encapsulate reusable block of code for a specific task
- Elements of a function
  - Name
  - Input (Arguments)
  - Body
  - Output (Optional)
    - Writing to a file
    - Plotting
- Why write functions?
  - Breaking down larger problems
  - Reusability
  - Minimizing errors
  - Maintainability

```
function_name <- function(argument1, argument2, ...) {
  # Function body
  # Perform operations
  # Return a value (optional)
}
```

```
# Define a function to calculate the square of a number
square <- function(x) {
  return(x^2)
}

# Call the function
result <- square(5)
print(result)  # Output: 25
```

Tidyverse

- **A curated family of R packages that implement the tidy philosophy**
- **What "Tidy" means:**
  - Each variable is a column
  - Each observation is a row
  - A table is a consistent set of observations for a known set of variables.
- **Tidyverse Packages for Data Manipulation:**
  - **dplyr**: Provides a grammar of data manipulation for data frames, enabling easy filtering, selecting, mutating, summarizing, and arranging of data.
  - **tidyr**: Offers tools for reshaping and tidying messy datasets, such as gather() and spread() functions for converting between wide and long formats.
  - **ggplot2**: Allows for the creation of sophisticated data visualizations using a layered grammar of graphics.

Manipulating data frames

- Nice features:
  - Printing
  - Variable information
  - Extracting a column for a tibble yields another tibble
    - data.frame will yield a vector
  - Row index not added by default
    - Explicit id column

```
library(tidyverse)

# Create a tibble (tidyverse version of data.frame)
df <- tibble(
  barcode = c("AAACCTGAGAAGGCCT-1", "AAACCTGCAAGTTGTC-1", "AAACCTGCACCGATAT-1"),
  phenotype = c("TEMRA", "Naive", "Dysfunctional"),
  treatment = c("Pre", "Post", "Pre"),
  response = c("R", "NR", "R")
)
```

Hands-on: Reading, writing, and interpreting data structures

1. Start your first Rscript
2. Work through the blocks of code under the **Course: Introduction to R** page
   - Review the details on how the code works in the Lecture slides for assistance
   - Put a post-it on your laptop if you get stuck, indicating for a TA to come up to you
   - Work through the blocks of code on this page, practicing in both your Rscript and the console
3. Take the next step
   - There are a list of **Additional exercises** at the bottom of the page for you to try on your own

Goal: Start building your R vocabulary

Are your hands as smart as your brain?