Basics of R

Ling 250/450: Data Science for Linguistics

C.M. Downey

Spring 2025

Goals for this quarter

Goals for this quarter

- Welcome back! We'll be switching gears a little now

Goals for this quarter

- Welcome back! We'll be switching gears a little now
- Focuses for the rest of the term:
  - **Term project!** You'll start brainstorming right away in homework 3, and we'll have several check-ins along the way
  - Crash-course in **statistics and probability**, using R to illustrate
  - **Manipulating and visualizing data** in R (with the "tidyverse" libraries & ggplot)
  - **Statistically testing hypotheses**, and applying this to your project

Goals for this quarter

- Welcome back! We'll be switching gears a little now
- Focuses for the rest of the term:
  - **Term project!** You'll start brainstorming right away in homework 3, and we'll have several check-ins along the way
  - Crash-course in **statistics and probability**, using R to illustrate
  - **Manipulating and visualizing data** in R (with the "tidyverse" libraries & ggplot)
  - **Statistically testing hypotheses**, and applying this to your project
- These will all be mixed together as we go

What R is **good** for

What R is **good** for

- Manipulating and analyzing **tabular data** (e.g. CSVs)
  - Has a well-maintained set of libraries called the **tidyverse** for data cleaning and management

What R is **good** for

- Manipulating and analyzing **tabular data** (e.g. CSVs)
  - Has a well-maintained set of libraries called the **tidyverse** for data cleaning and management
- Learning about **statistical concepts**

What R is **good** for

- Manipulating and analyzing **tabular data** (e.g. CSVs)
  - Has a well-maintained set of libraries called the **tidyverse** for data cleaning and management
- Learning about **statistical concepts**
- Running **statistical tests** (usually with a special library)

What R is **good** for

- Manipulating and analyzing **tabular data** (e.g. CSVs)
  - Has a well-maintained set of libraries called the **tidyverse** for data cleaning and management
- Learning about **statistical concepts**
- Running **statistical tests** (usually with a special library)
- Creating **data visualizations** (both with base R and **ggplot**)
  - R libraries are far superior to Python for visualization
  - There are some knock-offs for Python, but they aren't well maintained

What R is **not so good** for

What R is **not so good** for

- Text processing!
  - **String operations are way less intuitive** than in Python
  - I don't think I've heard of anyone using R for this

What R is **not so good** for

- Text processing!
  - **String operations are way less intuitive** than in Python
  - I don't think I've heard of anyone using R for this
- Writing **complicated functions/algorithms**
  - R is specialized for data manipulation; writing other functions from scratch can be cumbersome
  - Python is far more **general purpose**

Installing R(Studio)

- Go to posit.co/download/rstudio-desktop
- Follow the instructions to install **both** R and RStudio (default options for installation should be okay)

1: Install R

RStudio requires R 3.6.0+. Choose a version of R that matches your computer's operating system.

R is not a Posit product. By clicking on the link below to download and install R, you are leaving the Posit website. Posit disclaims any obligations and all liability with respect to R and the R website.

DOWNLOAD AND INSTALL R

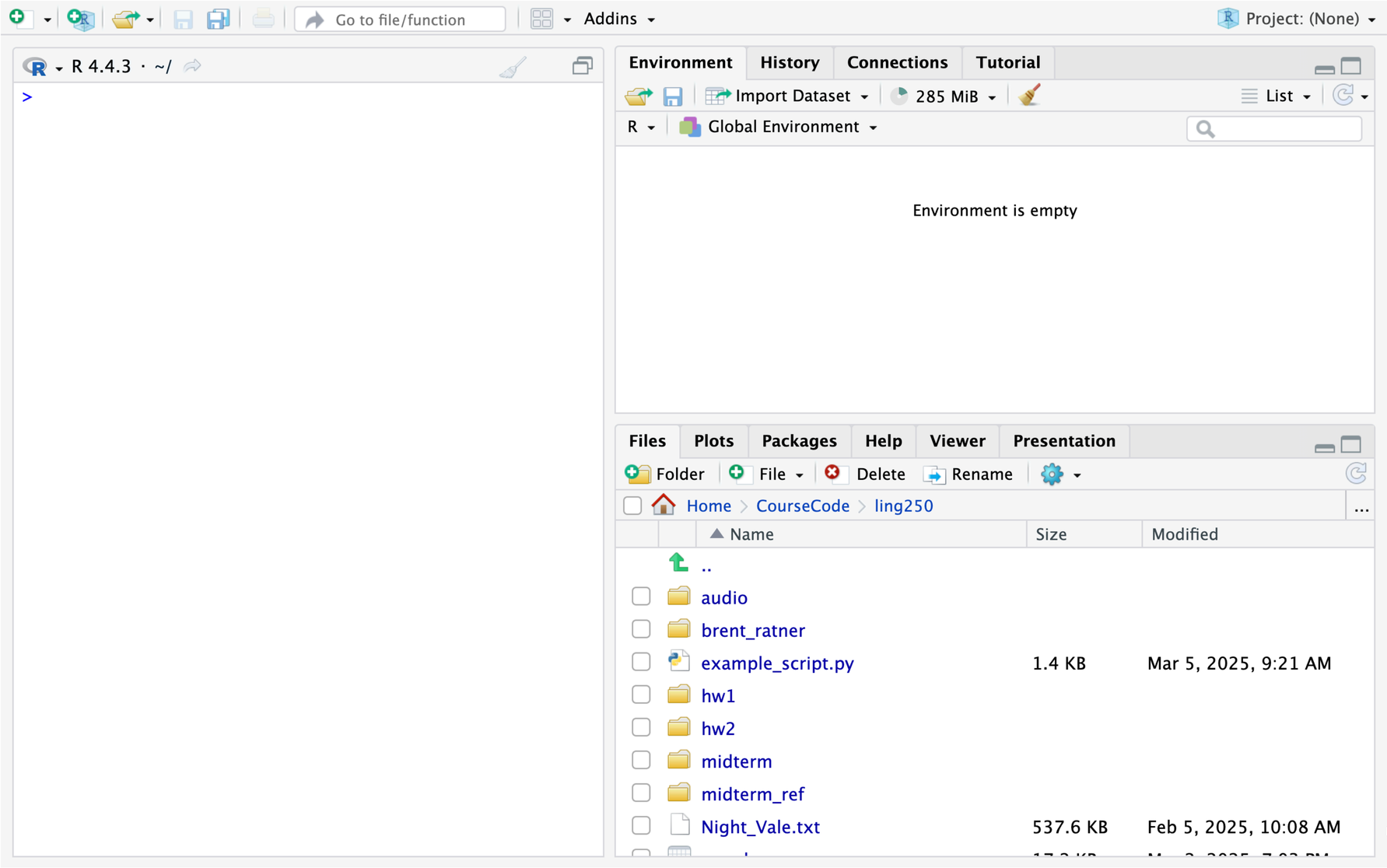
2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR MACOS 13+

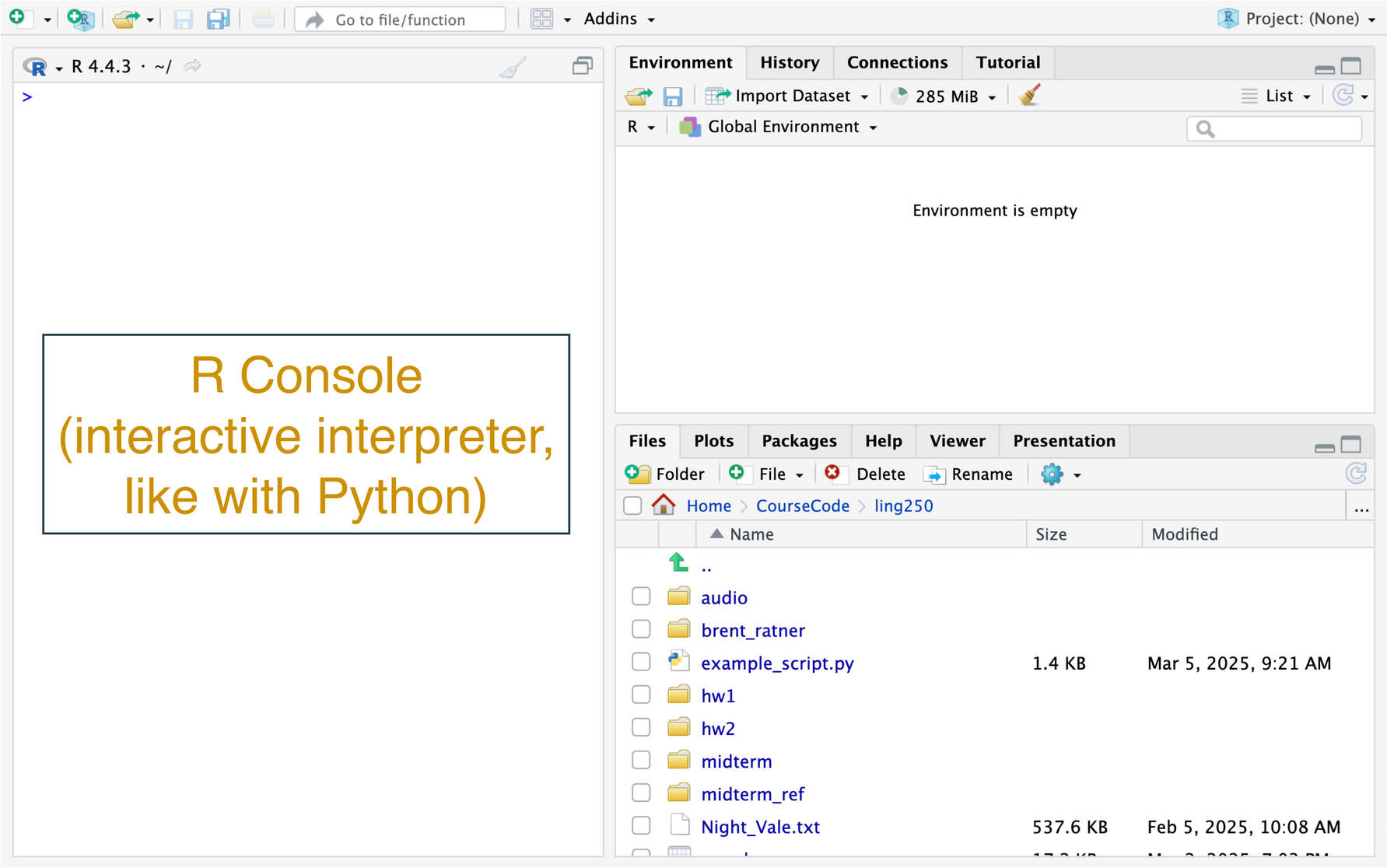
This version of RStudio is only supported on macOS 13 and higher. For earlier macOS environments, please download a previous version.

Size: 557.15 MB | SHA-256: BE73D3A9 | Version: 2024.12.1+563 | Released: 2025-02-13

RStudio tour

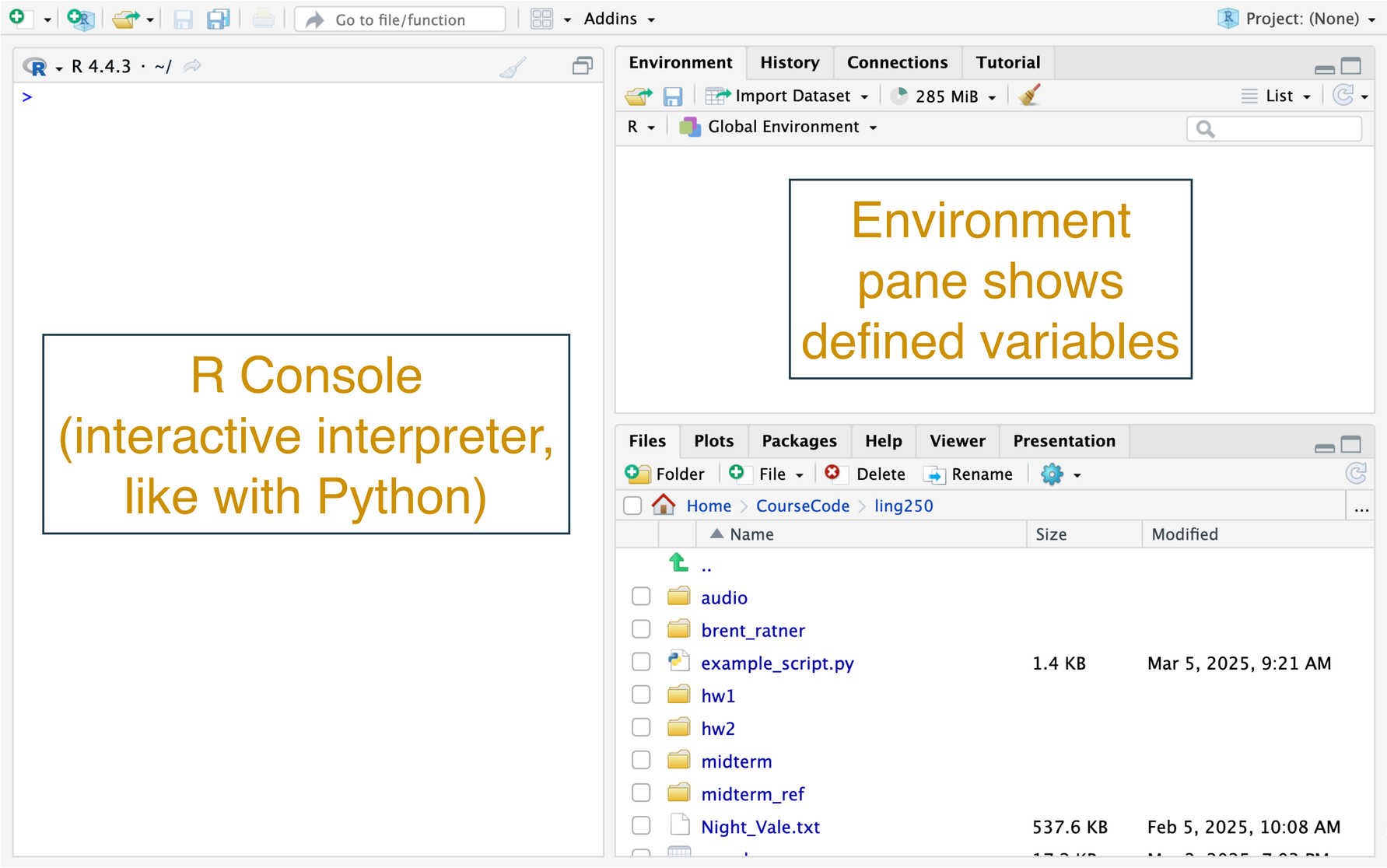


RStudio tour



R Console (interactive interpreter, like with Python)

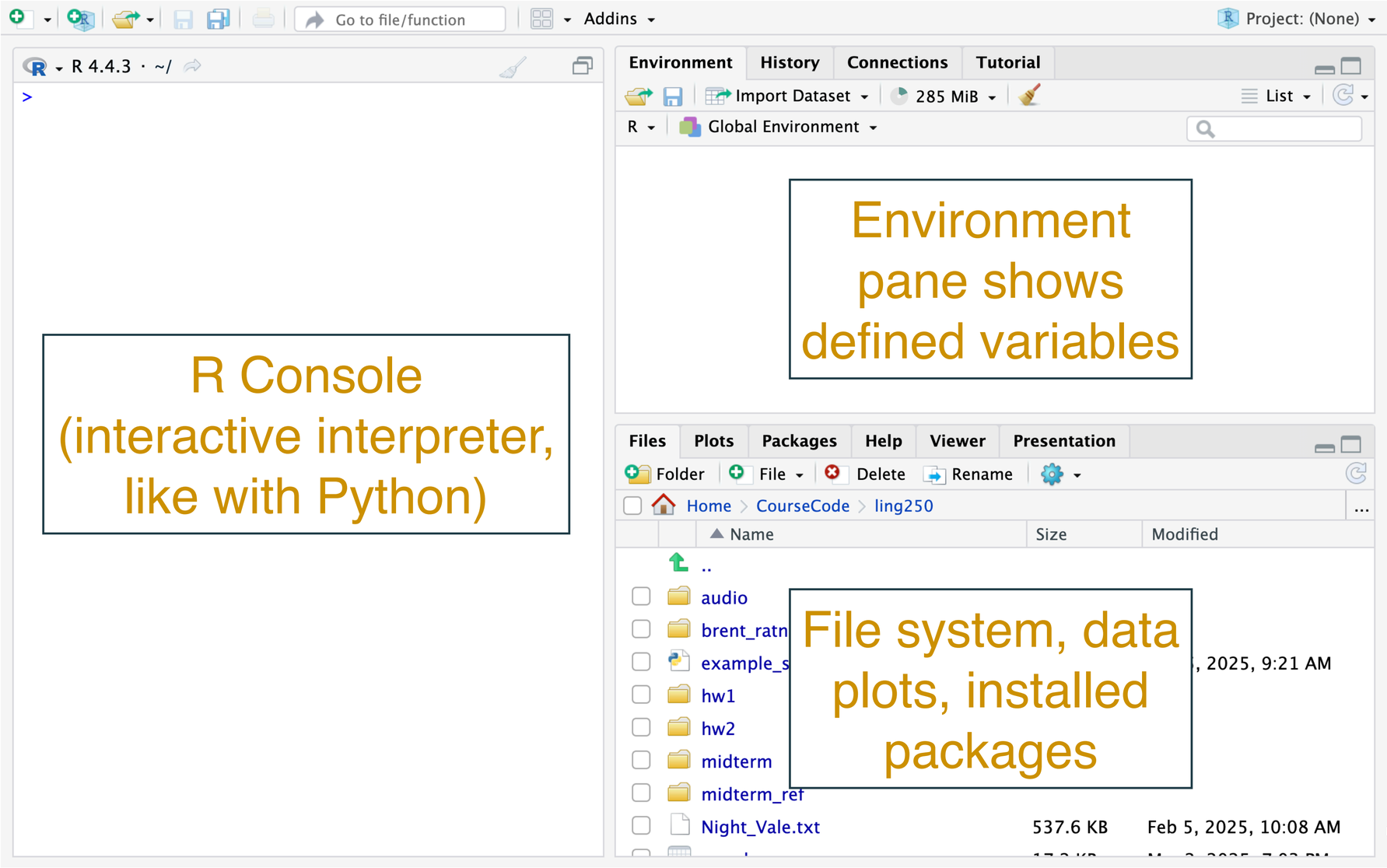
RStudio tour



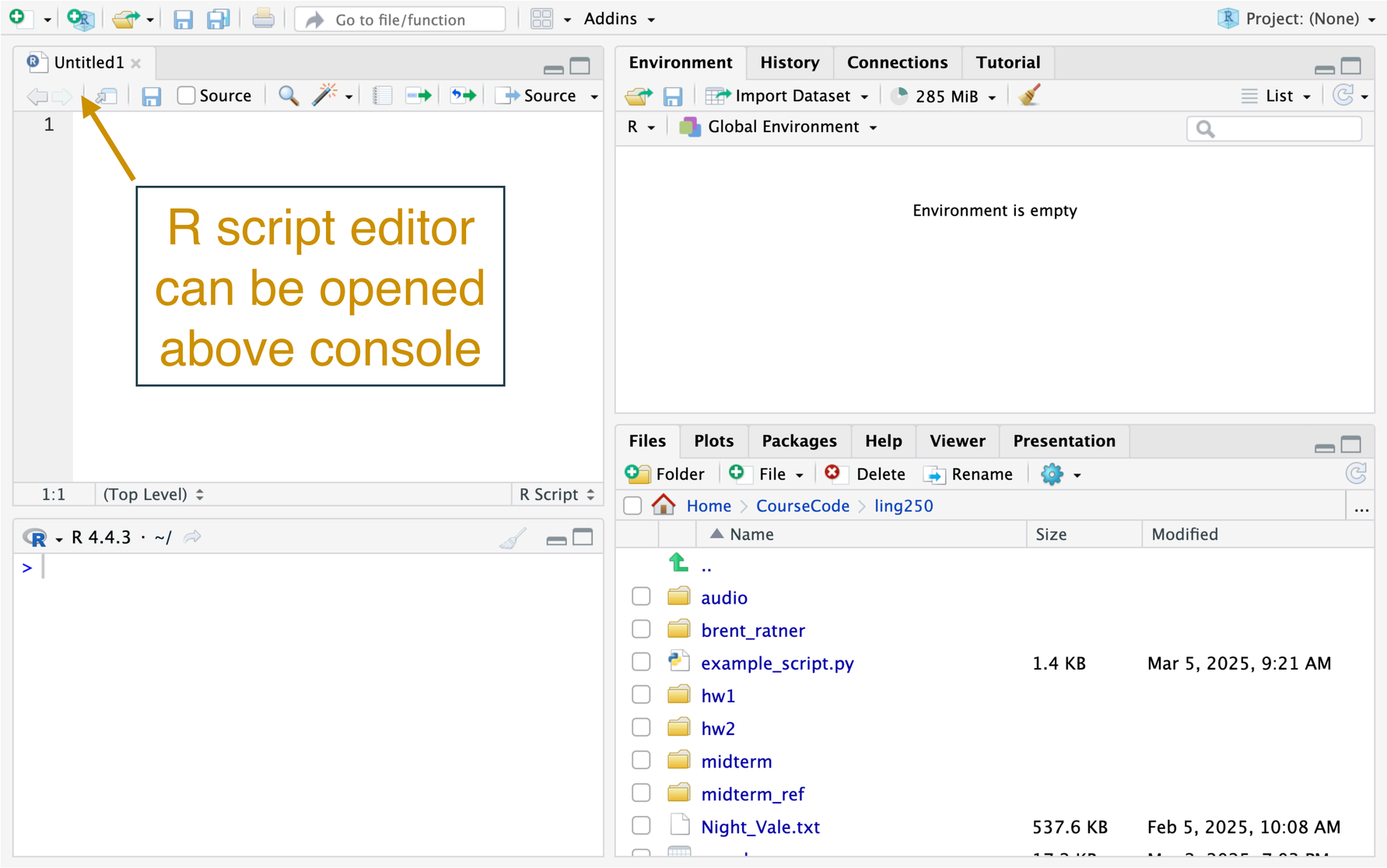
R Console (interactive interpreter, like with Python)

Environment pane shows defined variables

RStudio tour



RStudio tour



R language

Similarities to Python

Similarities to Python

- **Arithmetic operators** are essentially the same
  - Addition and subtraction: `a + b`; `a - b`
  - Multiplication, division, exponents: `a * b`; `a / b`; `a ^ b`
  - (In)equalities: `a > b`; `a >= b`; `a == b`; `a != b`

Similarities to Python

- **Arithmetic operators** are essentially the same
  - Addition and subtraction: `a + b`; `a - b`
  - Multiplication, division, exponents: `a * b`; `a / b`; `a ^ b`
  - (In)equalities: `a > b`; `a >= b`; `a == b`; `a != b`
- **Strings** are written in quotes: `"This is a string variable"`
  - **Note:** strings and characters are the same thing in R, both called characters
  - Strings **do not have the same functionality** as in Python

Similarities to Python

- **Arithmetic operators** are essentially the same
  - Addition and subtraction: `a + b`; `a - b`
  - Multiplication, division, exponents: `a * b`; `a / b`; `a ^ b`
  - (In)equalities: `a > b`; `a >= b`; `a == b`; `a != b`
- **Strings** are written in quotes: `"This is a string variable"`
  - **Note:** strings and characters are the same thing in R, both called characters
  - Strings **do not have the same functionality** as in Python
- **Comments** are demarcated with hashtags: `# this is a comment`

Minor/syntax differences

Minor/syntax differences

- **Variable assignment:** two options that are pretty much the same
  - `my_variable = 10` # newer versions of R allow the equal sign
  - `my_variable <- 10` # R used to only use this "arrow" operator

Minor/syntax differences

- **Variable assignment:** two options that are pretty much the same
  - `my_variable = 10` # newer versions of R allow the equal sign
  - `my_variable <- 10` # R used to only use this "arrow" operator
- **Boolean operators:** function the same as Python, different symbols
  - "and": `a & b`
  - "or": `a | b`
  - "not": `!a`
  - `TRUE, FALSE`
  - Parentheses needed for order of operations: `(2 > 1) & (3 > 2)`

Minor/syntax differences

Minor/syntax differences

- Other concepts that **work similarly** but with **syntactic differences**:
  - Functions
  - If/else statements
  - Loops

Minor/syntax differences

- Other concepts that **work similarly** but with **syntactic differences**:
  - Functions
  - If/else statements
  - Loops
- We won't focus as much on these, but you can read up on them in the Learning Statistics with R book if you need

Substantial differences from Python

Substantial differences from Python

- Probably most substantial: **variable classes** (especially sequential ones)

Substantial differences from Python

- Probably most substantial: **variable classes** (especially sequential ones)
- The closest thing to a Python list is a **vector**

Substantial differences from Python

- Probably most substantial: **variable classes** (especially sequential ones)
- The closest thing to a Python list is a **vector**
  - Formed with the **`c()`** function: `my_vec = c(1, 2, 3, 4, 5)`

Substantial differences from Python

- Probably most substantial: **variable classes** (especially sequential ones)
- The closest thing to a Python list is a **vector**
  - Formed with the **`c()`** function: `my_vec = c(1, 2, 3, 4, 5)`
  - Can be **indexed with brackets**, but **starting with 1:** `my_vec[1] == 1`

Substantial differences from Python

- Probably most substantial: **variable classes** (especially sequential ones)
- The closest thing to a Python list is a **vector**
  - Formed with the **`c()`** function: `my_vec = c(1, 2, 3, 4, 5)`
  - Can be **indexed with brackets**, but **starting with 1:** `my_vec[1] == 1`
  - Elements can be **reassigned:** `my_vec[1] = 0`

Substantial differences from Python

- Probably most substantial: **variable classes** (especially sequential ones)
- The closest thing to a Python list is a **vector**
  - Formed with the **`c()`** function: `my_vec = c(1, 2, 3, 4, 5)`
  - Can be **indexed with brackets**, but **starting with 1:** `my_vec[1] == 1`
  - Elements can be **reassigned:** `my_vec[1] = 0`
  - Can **apply arithmetic** to all elements: `my_vec * 2 == c(2, 4, 6, 8, 10)`

Substantial differences from Python

- Probably most substantial: **variable classes** (especially sequential ones)
- The closest thing to a Python list is a **vector**
  - Formed with the **`c()`** function: `my_vec = c(1, 2, 3, 4, 5)`
  - Can be **indexed with brackets**, but **starting with 1:** `my_vec[1] == 1`
  - Elements can be **reassigned:** `my_vec[1] = 0`
  - Can **apply arithmetic** to all elements: `my_vec * 2 == c(2, 4, 6, 8, 10)`
  - `my_vec + c(1, 2, 3, 4, 5) == c(2, 4, 6, 8, 10)`

Substantial differences from Python

- Probably most substantial: **variable classes** (especially sequential ones)
- The closest thing to a Python list is a **vector**
  - Formed with the **`c()`** function: `my_vec = c(1, 2, 3, 4, 5)`
  - Can be **indexed with brackets**, but **starting with 1:** `my_vec[1] == 1`
  - Elements can be **reassigned:** `my_vec[1] = 0`
  - Can **apply arithmetic** to all elements: `my_vec * 2 == c(2, 4, 6, 8, 10)`
  - `my_vec + c(1, 2, 3, 4, 5) == c(2, 4, 6, 8, 10)`
  - `length(my_vec) == 5`

Data frames

- Data frames **store tabular data** (rows and columns)
  - Can be read from a CSV file with `read.csv(filename)`
  - Can also be created from vectors: `data.frame(vec1, vec2, ...)`
- The first few rows can be viewed with the `head()` function

```
> dataframe = read.csv("~/CourseCode/ling250/vowels.csv")
> head(dataframe)
  SPEAKER WORD VOWEL      F1      F2  SEX HEIGHT
1      S1  bad     æ 848.070 1450.96 male    173
2      S1 bard     ɑ 648.318 1126.22 male    173
3      S1 bead     i 259.000 1834.00 male    173
4      S1  bed     e 578.985 1715.22 male    173
5      S1  bid     ɪ 405.000 1899.00 male    173
6      S1 bird     ɜ 656.600 1414.40 male    173
```

```
> evens = c(2,4,6,8)
> odds = c(1,3,5,7)
> data.frame(evens, odds)
  evens odds
1     2    1
2     4    3
3     6    5
4     8    7
```

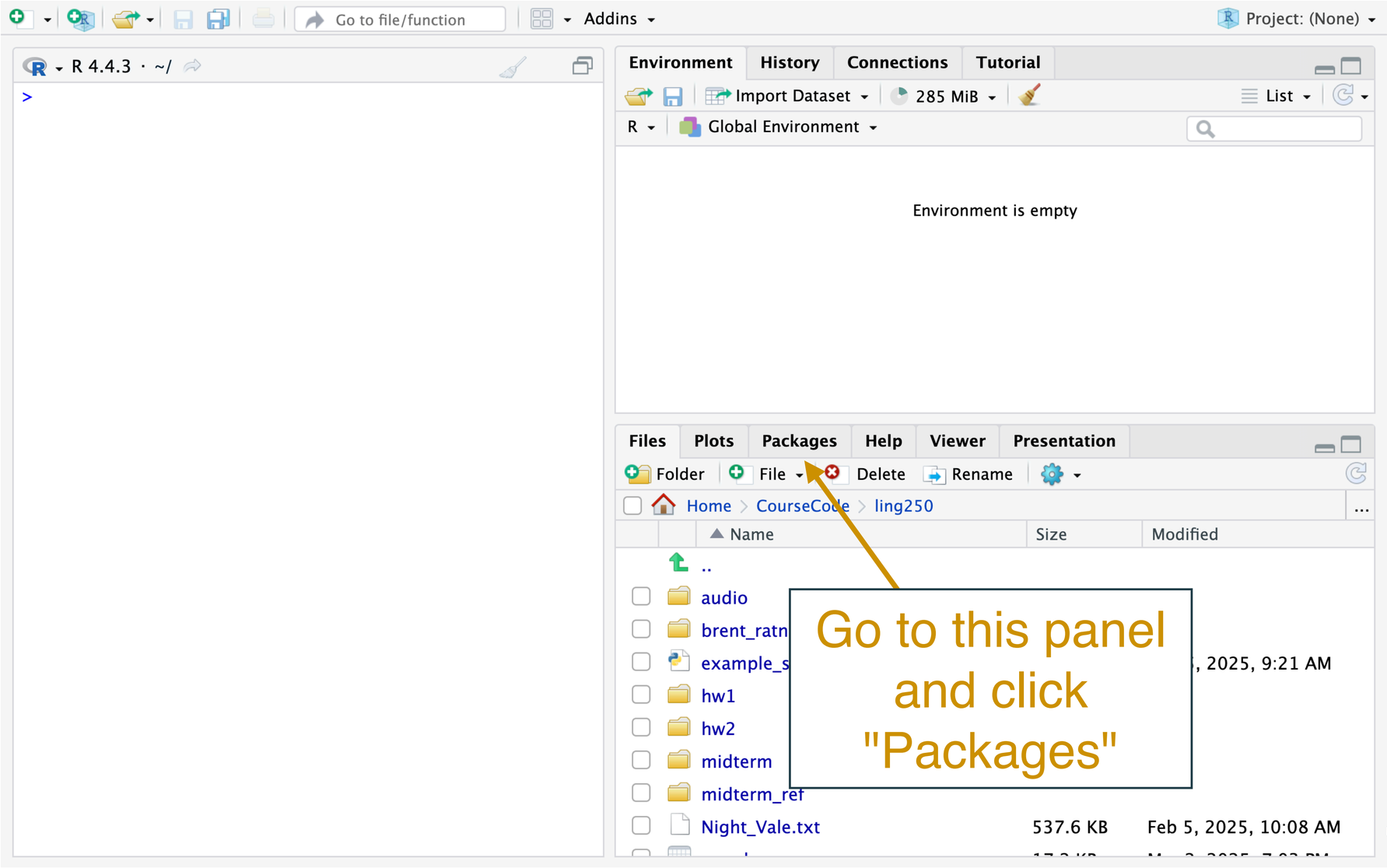
Data frames

- Each **column** is a **vector**, and can be accessed with the **$ operator**
- The **column names** can be accessed with `names(dataframe)`

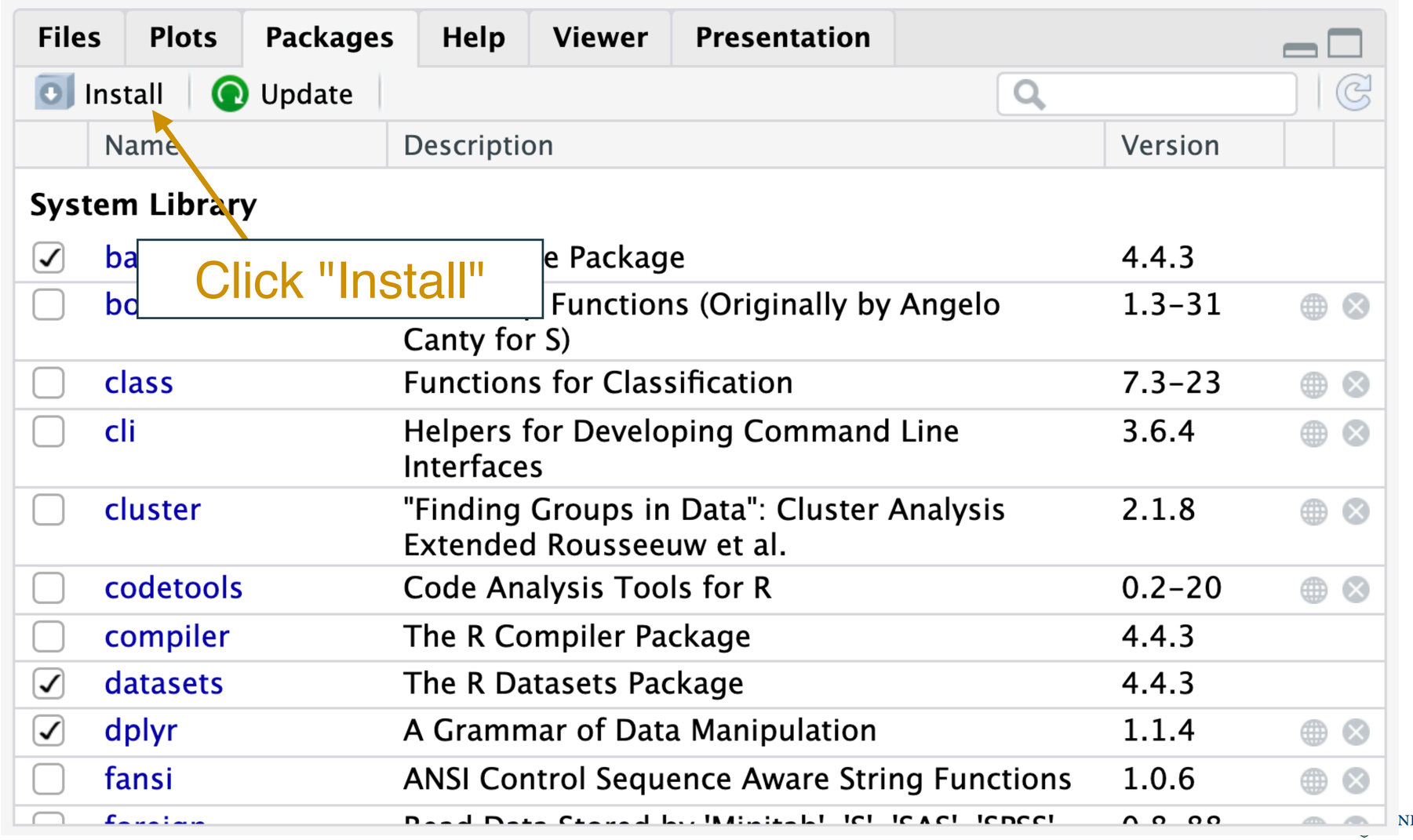
```
> dataframe$VOWEL
 [1] "æ" "ɑ" "i" "e" "ɪ" "ɜ" "ɔ" "ɒ" "u" "ʌ" "ʊ" "æ" "ɑ"
[14] "i" "e" "ɪ" "ɜ" "ɔ" "ɒ" "u" "ʌ" "ʊ" "æ" "ɑ" "i" "e"
[27] "ɪ" "ɜ" "ɔ" "ɒ" "u" "ʌ" "ʊ" "æ" "ɑ" "i" "e" "ɪ" "ɜ"
[40] "ɔ" "ɒ" "u" "ʌ" "ʊ" "æ" "ɑ" "i" "e" "ɪ" "ɜ" "ɔ" "ɒ"
[53] "u" "ʌ" "ʊ" "æ" "ɑ" "i" "e" "ɪ" "ɜ" "ɔ" "ɒ" "u" "ʌ"
[66] "ʊ" "æ" "ɑ" "i" "e" "ɪ" "ɜ" "ɔ" "ɒ" "u" "ʌ" "ʊ" "æ"

> names(dataframe)
[1] "SPEAKER" "WORD"    "VOWEL"   "F1"      "F2"
[6] "SEX"     "HEIGHT"
```

Packages / A little tidyverse

Installing a package (dplyr)

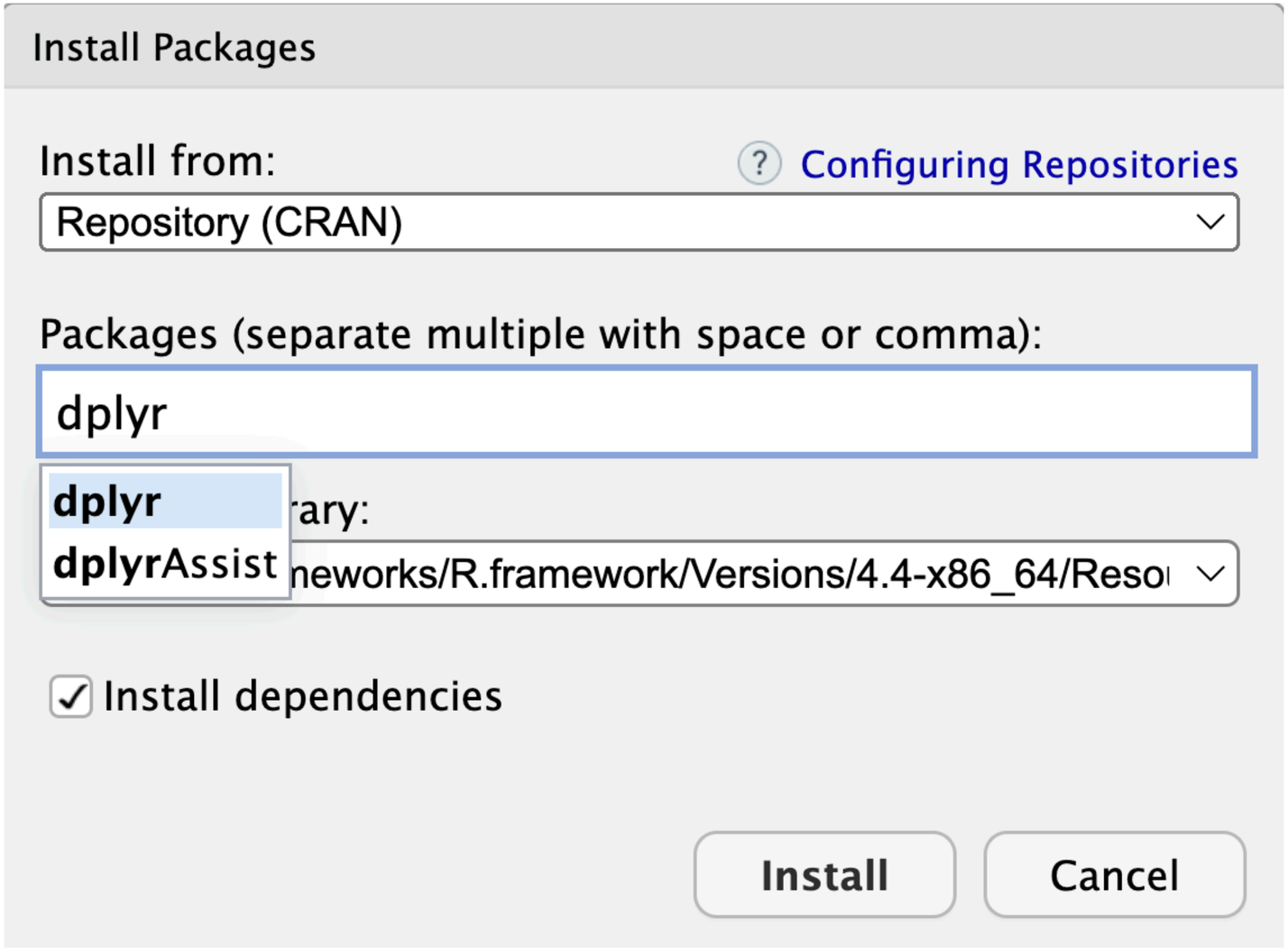


Installing a package (dplyr)



Installing a package (dplyr)

Type in the name of the package and click "Install"



Importing dplyr

R 4.4.3 · ~/

```
> library(dplyr)
> filter(dataframe, VOWEL=="i")
   SPEAKER WORD VOWEL      F1      F2    SEX
1       S1 bead     i 259.000 1834.00   male
2       S2 bead     i 237.000 2258.00   male
3       S3 bead     i 471.265 2791.13 female
4       S4 bead     i 366.237 1841.18 female
5       S5 bead     i 373.277 2602.93 female
6       S6 bead     i 320.294 2048.55   male
7       S7 bead     i 372.307 2874.09 female
8       S8 bead     i 238.480 2269.17   male
9       S9 bead     i 472.915 2440.00 female
10     S10 bead     i 329.000 2875.00 female
```

Importing dplyr

- `library` is similar to Python's import

R 4.4.3 · ~/

```
> library(dplyr)
> filter(dataframe, VOWEL=="i")
   SPEAKER WORD VOWEL      F1      F2    SEX
1       S1 bead     i 259.000 1834.00   male
2       S2 bead     i 237.000 2258.00   male
3       S3 bead     i 471.265 2791.13 female
4       S4 bead     i 366.237 1841.18 female
5       S5 bead     i 373.277 2602.93 female
6       S6 bead     i 320.294 2048.55   male
7       S7 bead     i 372.307 2874.09 female
8       S8 bead     i 238.480 2269.17   male
9       S9 bead     i 472.915 2440.00 female
10     S10 bead     i 329.000 2875.00 female
```

Importing dplyr

- `library` is similar to Python's import
- Unlike Python, imported functions like `filter` don't need to be prefixed with the package name

R 4.4.3 · ~/

```
> library(dplyr)
> filter(dataframe, VOWEL=="i")
   SPEAKER WORD VOWEL      F1      F2    SEX
1       S1 bead     i 259.000 1834.00   male
2       S2 bead     i 237.000 2258.00   male
3       S3 bead     i 471.265 2791.13 female
4       S4 bead     i 366.237 1841.18 female
5       S5 bead     i 373.277 2602.93 female
6       S6 bead     i 320.294 2048.55   male
7       S7 bead     i 372.307 2874.09 female
8       S8 bead     i 238.480 2269.17   male
9       S9 bead     i 472.915 2440.00 female
10     S10 bead     i 329.000 2875.00 female
```

Importing dplyr

- `library` is similar to Python's import
- Unlike Python, imported functions like `filter` don't need to be prefixed with the package name
- `filter` returns a **subset** of the data frame matching a **certain condition**

R 4.4.3 · ~/

```
> library(dplyr)
> filter(dataframe, VOWEL=="i")
   SPEAKER WORD VOWEL      F1      F2    SEX
1       S1 bead     i 259.000 1834.00   male
2       S2 bead     i 237.000 2258.00   male
3       S3 bead     i 471.265 2791.13 female
4       S4 bead     i 366.237 1841.18 female
5       S5 bead     i 373.277 2602.93 female
6       S6 bead     i 320.294 2048.55   male
7       S7 bead     i 372.307 2874.09 female
8       S8 bead     i 238.480 2269.17   male
9       S9 bead     i 472.915 2440.00 female
10     S10 bead     i 329.000 2875.00 female
```

Importing dplyr

- `library` is similar to Python's import
- Unlike Python, imported functions like `filter` don't need to be prefixed with the package name
- `filter` returns a **subset** of the data frame matching a **certain condition**
- `dplyr` contains **many more** functions, but `filter` is extremely useful

R 4.4.3 · ~/

```
> library(dplyr)
> filter(dataframe, VOWEL=="i")
   SPEAKER WORD VOWEL      F1      F2    SEX
1       S1 bead     i 259.000 1834.00   male
2       S2 bead     i 237.000 2258.00   male
3       S3 bead     i 471.265 2791.13 female
4       S4 bead     i 366.237 1841.18 female
5       S5 bead     i 373.277 2602.93 female
6       S6 bead     i 320.294 2048.55   male
7       S7 bead     i 372.307 2874.09 female
8       S8 bead     i 238.480 2269.17   male
9       S9 bead     i 472.915 2440.00 female
10     S10 bead     i 329.000 2875.00 female
```

Using `filter`

R 4.4.3 · ~/

```
> filter(
+   dataframe,
+   SEX=="female",
+   VOWEL=="i" | VOWEL=="u",
+   HEIGHT<=mean(HEIGHT)
+ )
   SPEAKER  WORD VOWEL      F1      F2    SEX HEIGHT
1       S3  bead     i 471.265 2791.13 female    163
2       S3 booed     u 466.943 2360.90 female    163
3       S4  bead     i 366.237 1841.18 female    164
4       S4 booed     u 200.000  379.00 female    164
5       S5  bead     i 373.277 2602.93 female    165
6       S5 booed     u 459.272 1635.41 female    165
7       S7  bead     i 372.307 2874.09 female    159
8       S7 booed     u 382.030 1071.09 female    159
```

Using `filter`

- Can take **any number** of conditions

R 4.4.3 · ~/

```
> filter(
+   dataframe,
+   SEX=="female",
+   VOWEL=="i" | VOWEL=="u",
+   HEIGHT<=mean(HEIGHT)
+ )
   SPEAKER  WORD VOWEL      F1      F2    SEX HEIGHT
1       S3  bead     i 471.265 2791.13 female    163
2       S3 booed     u 466.943 2360.90 female    163
3       S4  bead     i 366.237 1841.18 female    164
4       S4 booed     u 200.000  379.00 female    164
5       S5  bead     i 373.277 2602.93 female    165
6       S5 booed     u 459.272 1635.41 female    165
7       S7  bead     i 372.307 2874.09 female    159
8       S7 booed     u 382.030 1071.09 female    159
```

Using `filter`

- Can take **any number** of conditions
- Conditions can be **any boolean criterion**

R 4.4.3 · ~/

```
> filter(
+   dataframe,
+   SEX=="female",
+   VOWEL=="i" | VOWEL=="u",
+   HEIGHT<=mean(HEIGHT)
+ )
   SPEAKER  WORD VOWEL      F1      F2    SEX HEIGHT
1       S3  bead     i 471.265 2791.13 female    163
2       S3 booed     u 466.943 2360.90 female    163
3       S4  bead     i 366.237 1841.18 female    164
4       S4 booed     u 200.000  379.00 female    164
5       S5  bead     i 373.277 2602.93 female    165
6       S5 booed     u 459.272 1635.41 female    165
7       S7  bead     i 372.307 2874.09 female    159
8       S7 booed     u 382.030 1071.09 female    159
```

Using `filter`

- Can take **any number** of conditions
- Conditions can be **any boolean criterion**
- This example filters for the vowels and u, spoken by females that are shorter than average height
  - (Shorter than average of all speakers, not just females)

R 4.4.3 · ~/

```
> filter(
+   dataframe,
+   SEX=="female",
+   VOWEL=="i" | VOWEL=="u",
+   HEIGHT<=mean(HEIGHT)
+ )
   SPEAKER  WORD VOWEL      F1      F2    SEX HEIGHT
1       S3  bead     i 471.265 2791.13 female    163
2       S3 booed     u 466.943 2360.90 female    163
3       S4  bead     i 366.237 1841.18 female    164
4       S4 booed     u 200.000  379.00 female    164
5       S5  bead     i 373.277 2602.93 female    165
6       S5 booed     u 459.272 1635.41 female    165
7       S7  bead     i 372.307 2874.09 female    159
8       S7 booed     u 382.030 1071.09 female    159
```

Basic plotting

Caveat: base R vs. ggplot

Caveat: base R vs. ggplot

- R has **built-in functions** for plotting and visualizing data
  - These are great for getting a **quick look** at data, but **hard to customize**

Caveat: base R vs. ggplot

- R has **built-in functions** for plotting and visualizing data
  - These are great for getting a **quick look** at data, but **hard to customize**
- In a week or two, we'll introduce the **ggplot package** for visualization
  - ggplot is considered the **gold-standard** for scientific data visualization
  - It's **far more customizable** than the base R package
  - It gives the best results, but is a little **tricky to learn**, so we'll need a dedicated unit

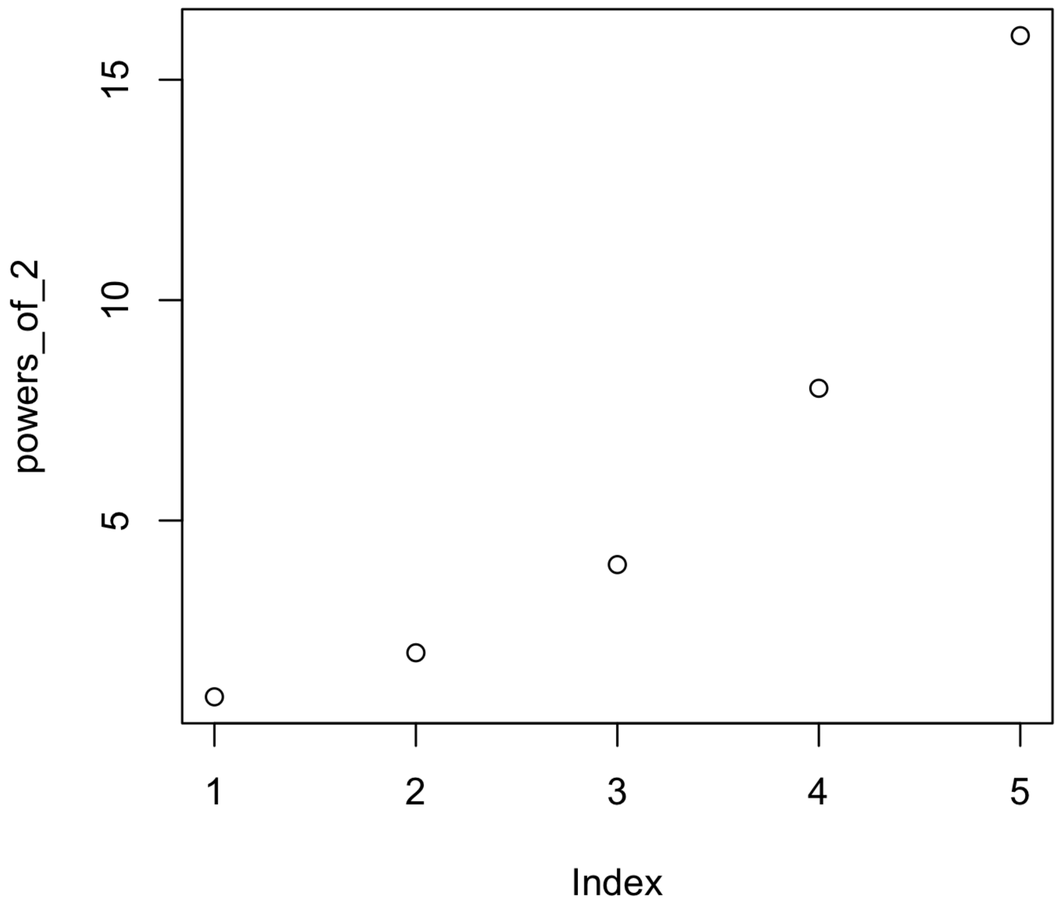
Caveat: base R vs. ggplot

- R has **built-in functions** for plotting and visualizing data
  - These are great for getting a **quick look** at data, but **hard to customize**
- In a week or two, we'll introduce the **ggplot package** for visualization
  - ggplot is considered the **gold-standard** for scientific data visualization
  - It's **far more customizable** than the base R package
  - It gives the best results, but is a little **tricky to learn**, so we'll need a dedicated unit
- Goal for today: **quick and simple** visualization with base R

plot()

R 4.4.3 · ~/

```
> powers_of_2 = c(1,2,4,8,16)
> plot(powers_of_2)
>
```

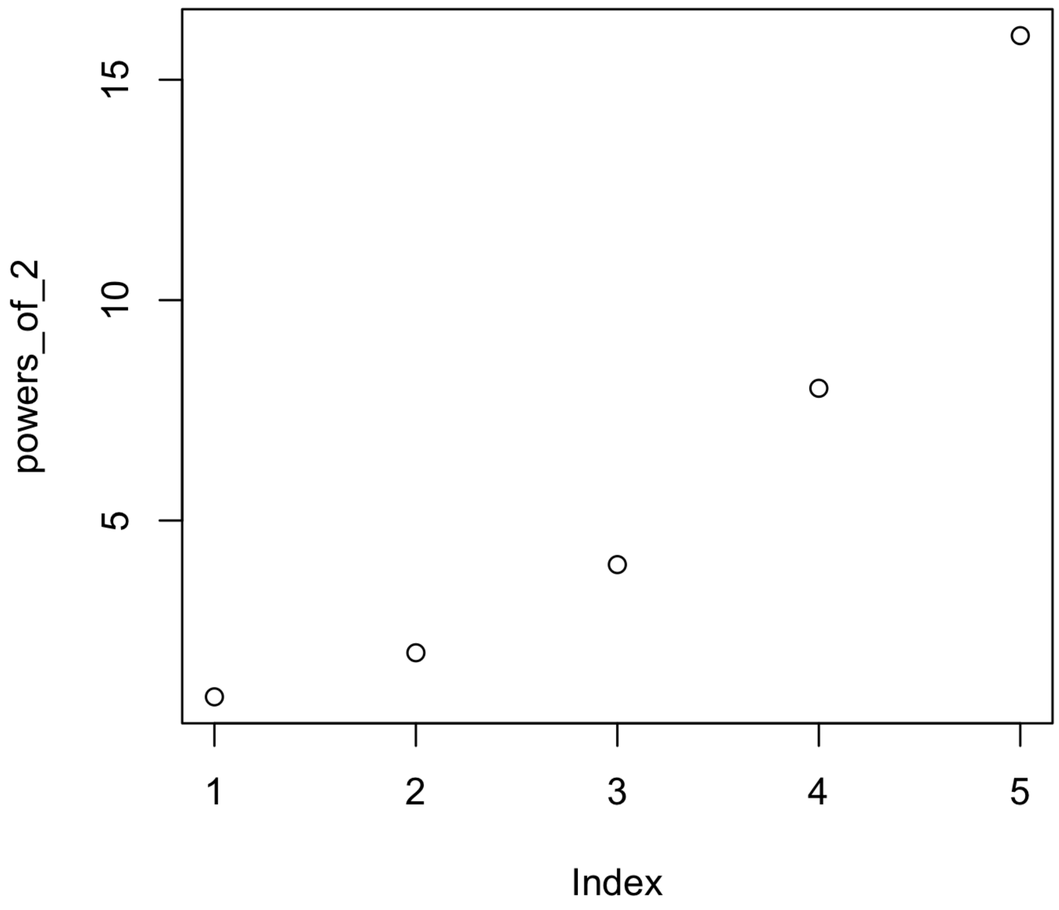


plot()

- `plot(vector)` draws a plot of the datapoints in the vector

R 4.4.3 · ~/

```
> powers_of_2 = c(1,2,4,8,16)
> plot(powers_of_2)
>
```

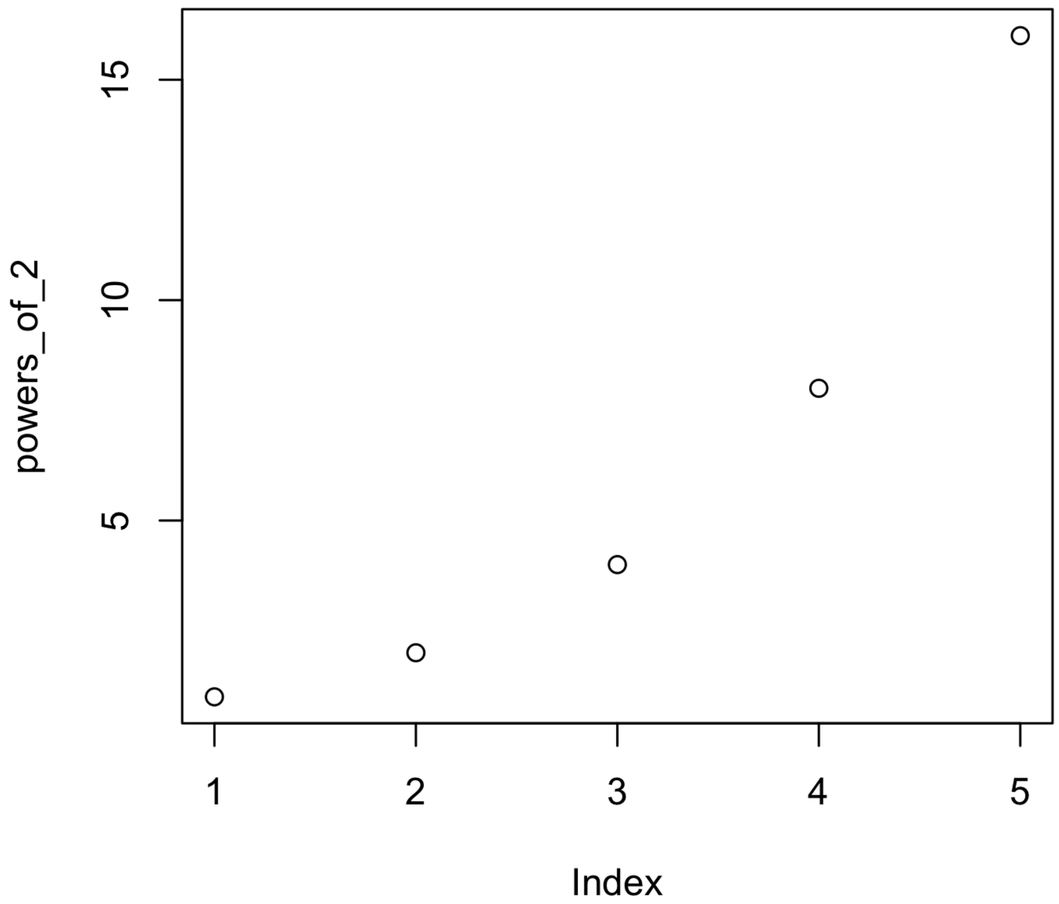


plot()

- `plot(vector)` draws a plot of the datapoints in the vector
- If just **one vector** is given, plots their values on the **y-axis**

R 4.4.3 · ~/

```
> powers_of_2 = c(1,2,4,8,16)
> plot(powers_of_2)
>
```

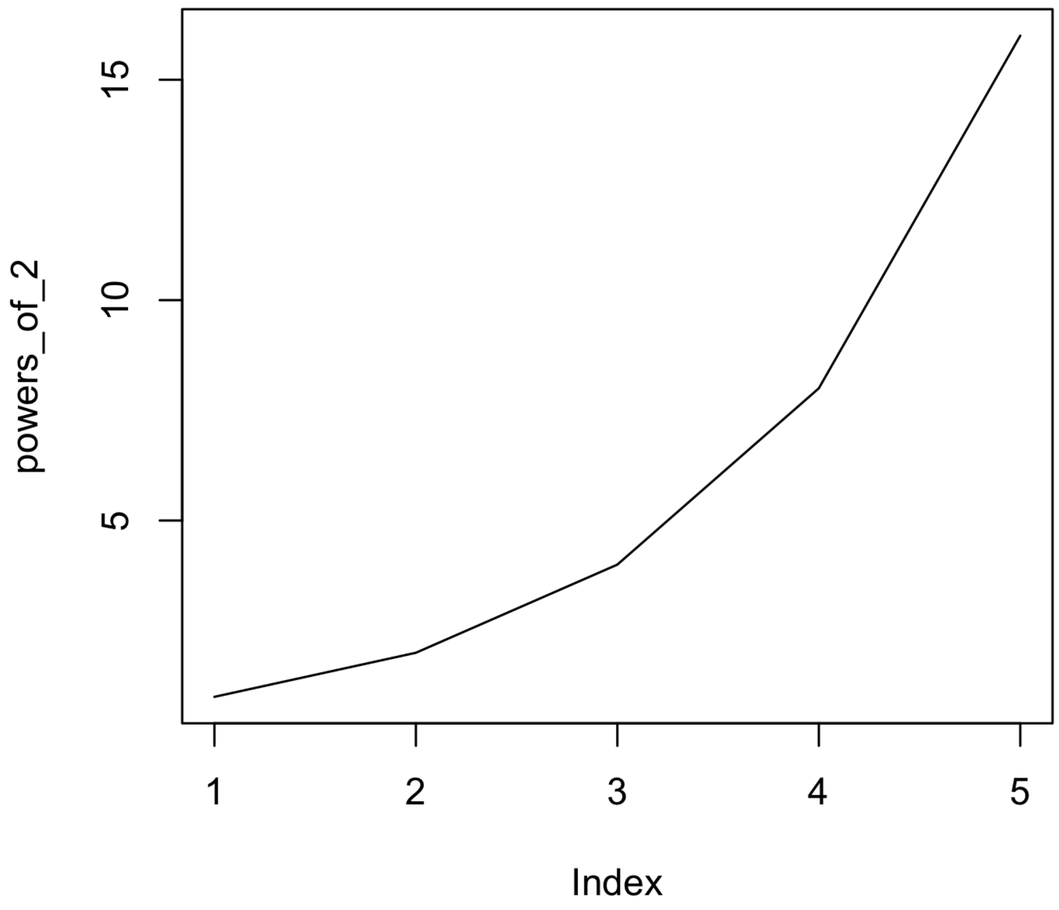


plot()

- `plot(vector)` draws a plot of the datapoints in the vector
- If just **one vector** is given, plots their values on the **y-axis**
  - The `type` argument can change the plot type (e.g. to a **line plot**)

R 4.4.3 · ~/

```
> plot(powers_of_2, type="l")
```

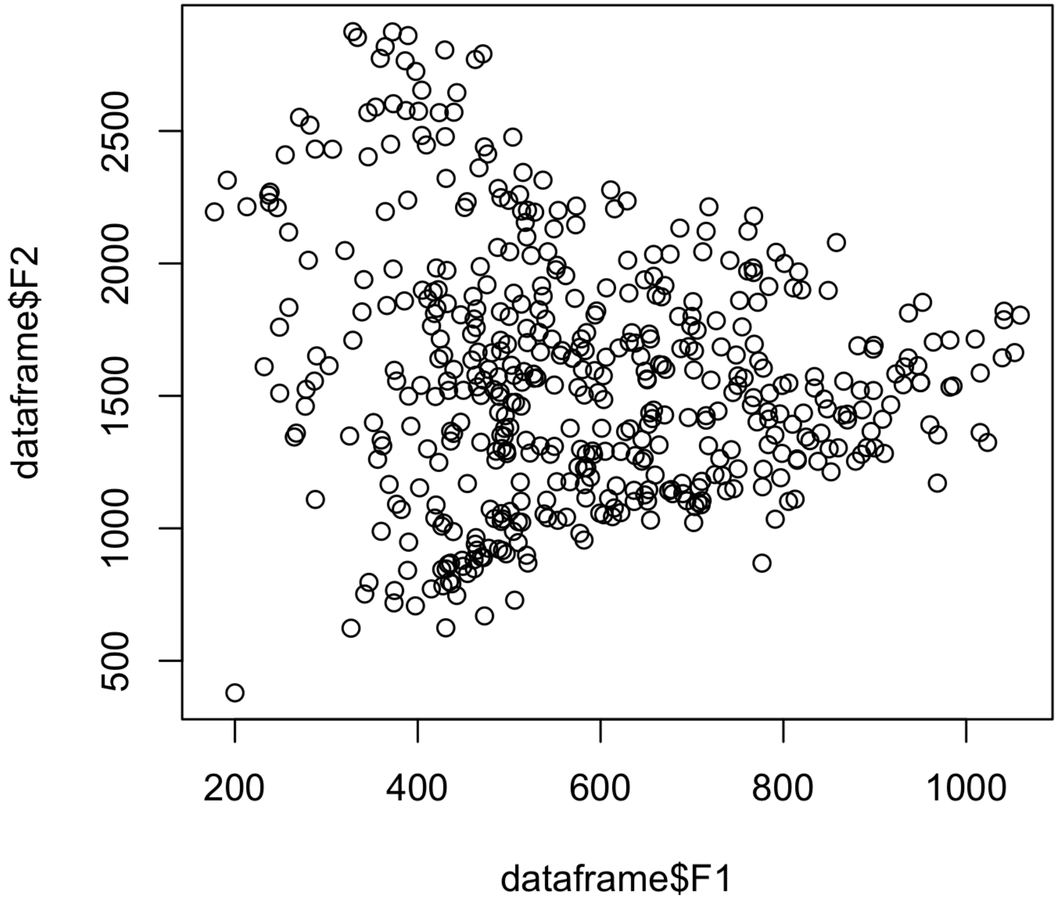


plot()

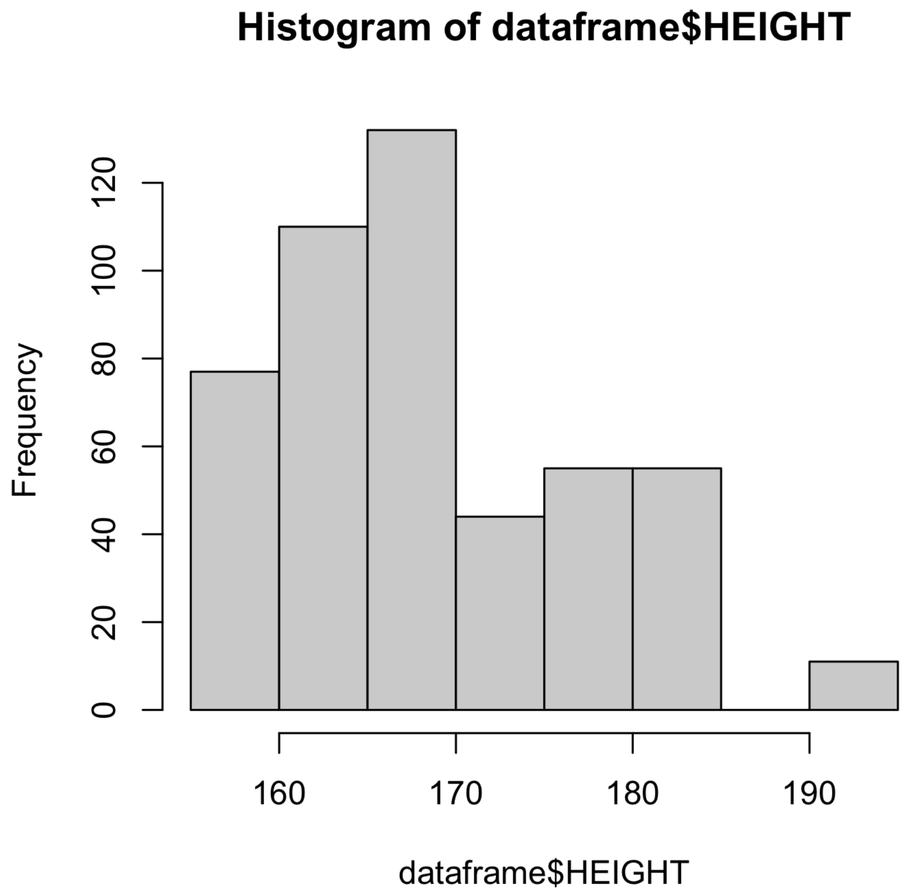
- `plot(vector)` draws a plot of the datapoints in the vector
- If just **one vector** is given, plots their values on the **y-axis**
  - The `type` argument can change the plot type (e.g. to a **line plot**)
- If **two vectors** are given, a **scatterplot** is made with the vectors along the x- and y-axis respectively

R 4.4.3 · ~/

```
> plot(dataframe$F1, dataframe$F2)
```

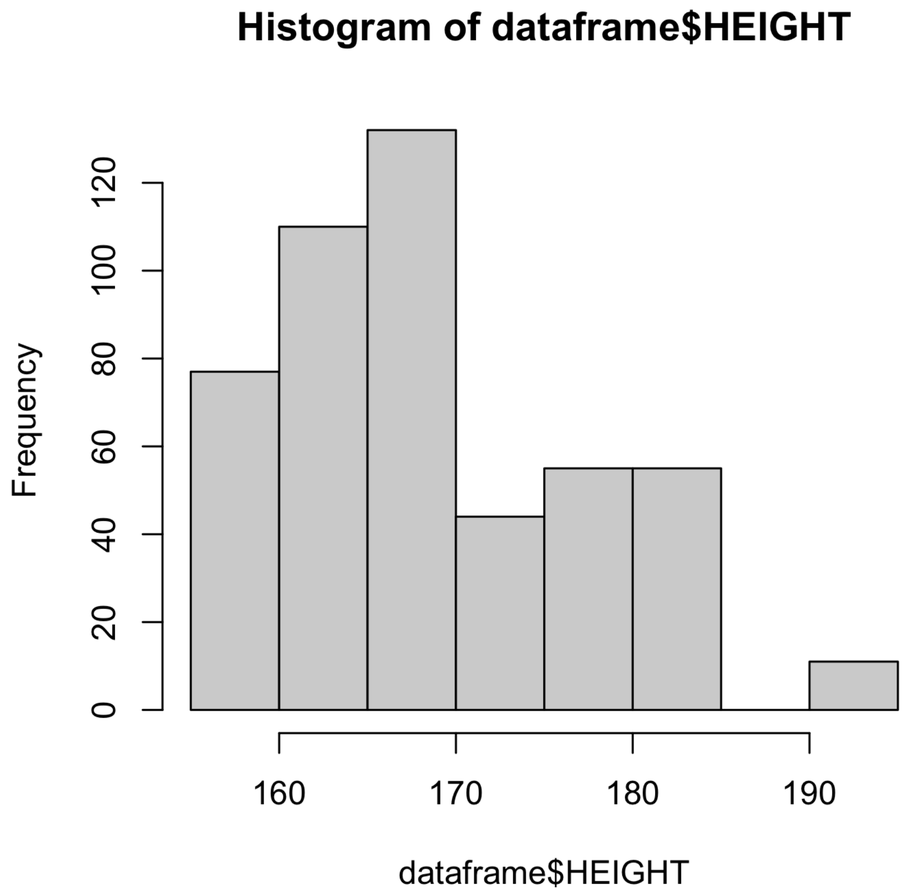


hist()


Histogram of dataframe$HEIGHT
Frequency
0
20
40
60
80
100
120
160
170
180
190
dataframe$HEIGHT

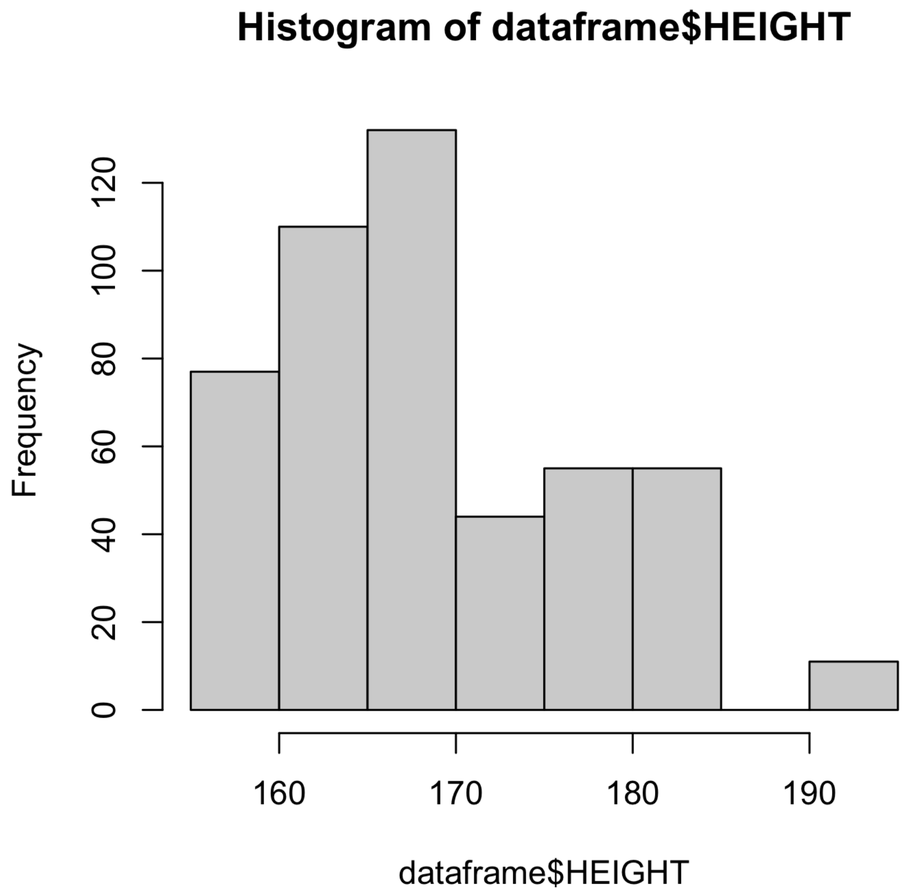
hist()

- `hist(vector)` gives a **histrogram** of the data, which plots the **frequency** of values in certain ranges



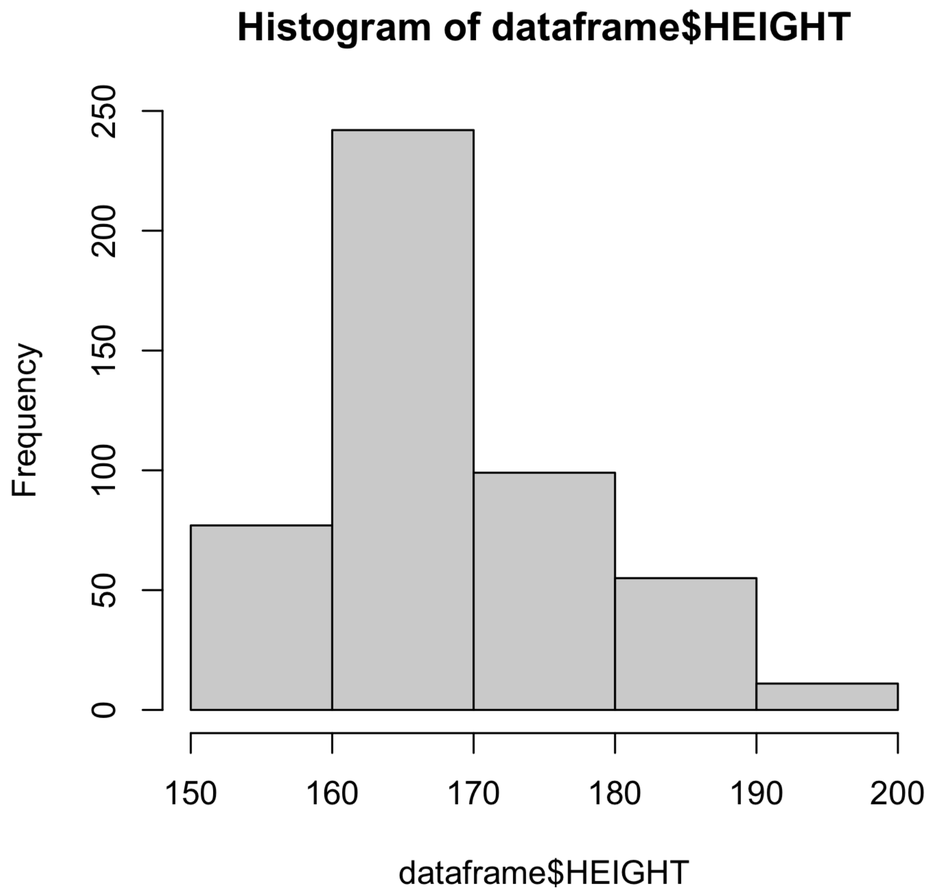
hist()

- `hist(vector)` gives a **histrogram** of the data, which plots the **frequency** of values in certain ranges
- It takes a `breaks` argument that specifies **how many boxes** to split into (approximately)



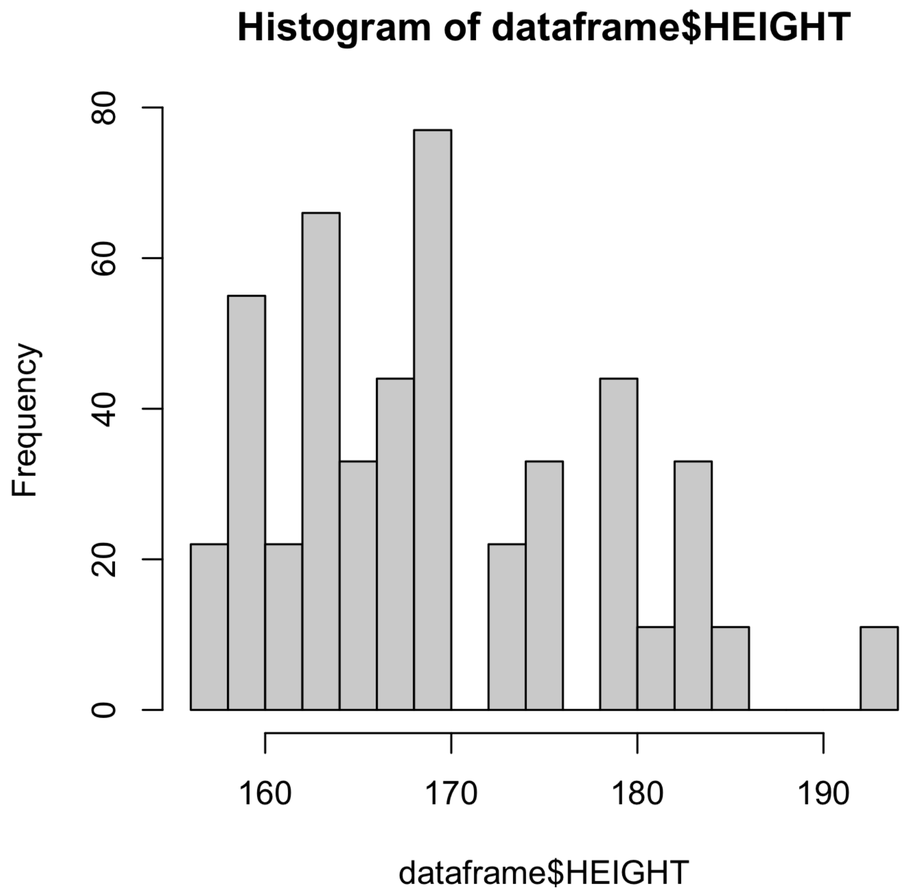
hist()

- `hist(vector)` gives a **histrogram** of the data, which plots the **frequency** of values in certain ranges
- It takes a `breaks` argument that specifies **how many boxes** to split into (approximately)
  - `breaks = 4`



hist()

- `hist(vector)` gives a **histrogram** of the data, which plots the **frequency** of values in certain ranges
- It takes a `breaks` argument that specifies **how many boxes** to split into (approximately)
  - `breaks = 4`
  - `breaks = 16`



boxplot()

- boxplot() draws a plot showing the **inner quartiles** of the data
  - Quartiles show where each **25% of the total data** falls
  - We'll talk more about quartiles later
- Oddly, this function doesn't give axis titles by default