

Introduction to R

Department of Sociology | University of Texas at Austin

Casey Breen

2026-01-22

Welcome to “Intro to R”

- Course website (if you want to learn more):
 - www.github.com/caseybreen/intro_r
 - Slides, exercises, and solutions

Session goals

- Overview: why R is a powerful tool for social science research
- Introduction to R syntax, data types, and data structures
- Basic understanding of data manipulation and visualization

Course agenda

- Module 1: Introduction to R, RStudio, and code formats
- Module 2: R programming fundamentals (syntax, operators, data types, data structures, sequencing)
- Module 3: Working with data

Module 1

Introduction to R, RStudio, and code formats

Learning objectives:

- Installing R and RStudio
- Why R?
- Understanding R Scripts, R notebooks, Quarto documents

R and RStudio

- R is a statistical programming language
 - Download: <https://cloud.r-project.org>
- RStudio is an integrated development environment (IDE) for R programming
 - Download: <http://www.rstudio.com/download>

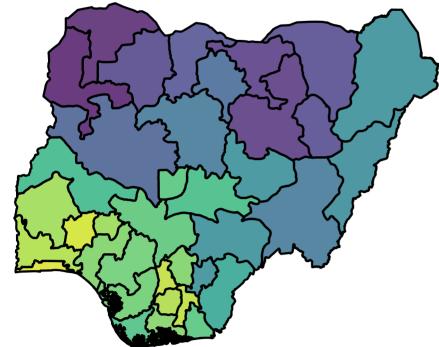
Why R?

- Free, open source – great for reproducibility and open science
- Powerful language for data manipulation, statistical analysis, and publication-ready data visualizations
- Excellent community, lots of free resources

Data visualization

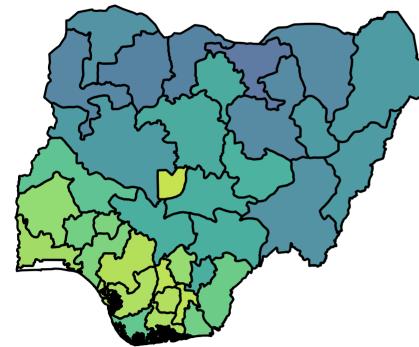
A

Observed



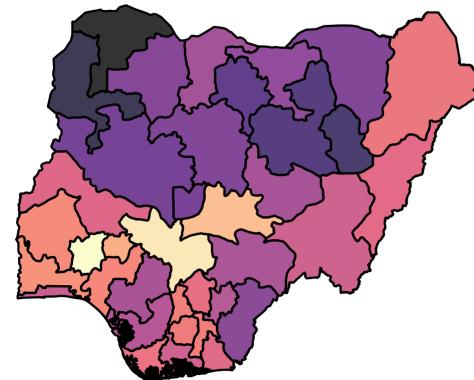
B

Predicted

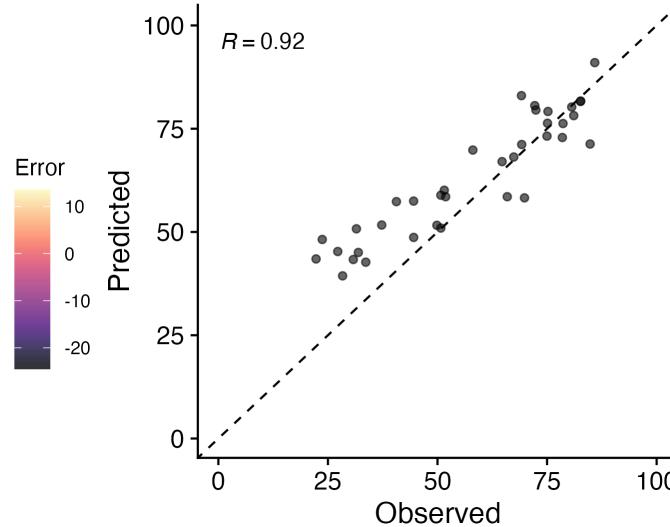


C

Error

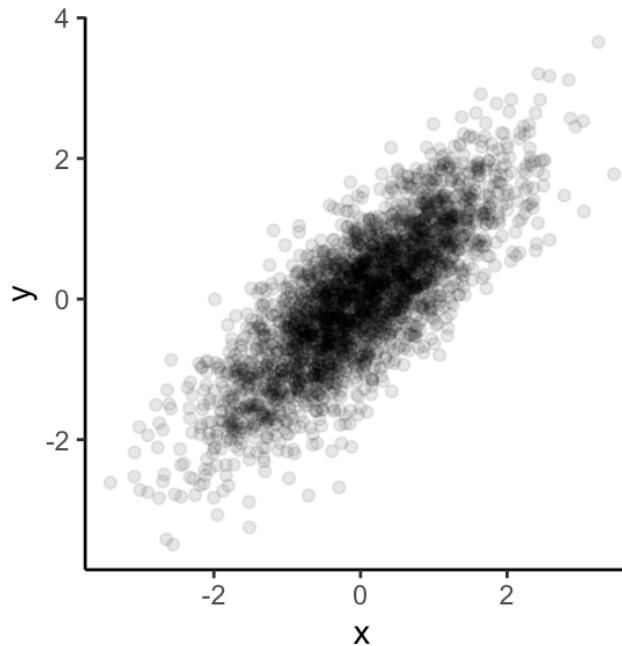


D **Mobile Usage, Women (Error)**

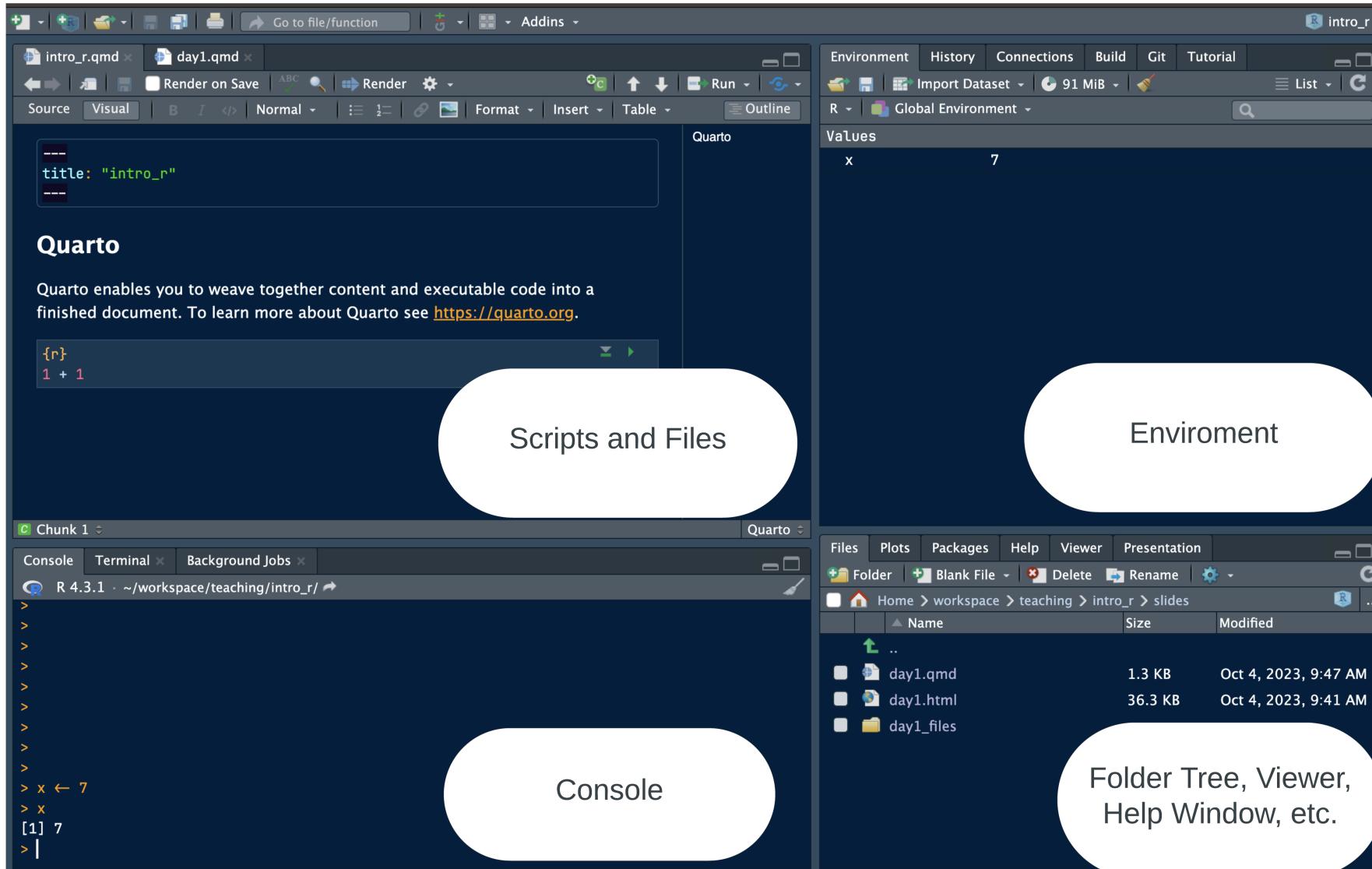


Easy to simulate + plot data

```
1 # Generate random data for x
2 x <- rnorm(n = 3000)
3 y <- 0.8 * x + rnorm(3000, 0, sqrt(1 - 0.8^2))
4
5 # Create data.frame
6 data_df <- data.frame(x = x, y = y)
7
8 # Generate visualization
9 data_df %>%
10   ggplot(aes(x = x, y = y)) +
11   geom_point(alpha = 0.1) +
12   theme_classic()
```



RStudio panes



Why RStudio?

- **All-in-one development environment:** streamlines coding, data visualization, and workflow
- **Extensible:** supports R — but also Python, SQL, and Git
- **Rich community:** eases learning and problem-solving

Code formats: R Scripts vs. R Notebooks

- **R Scripts**
 - Simple: just code
 - Best for simple tasks (and multi-script pipelines)
- **R Notebooks (Quarto, R Notebook)**
 - Integrated: Mix of code, text, and outputs for easy documentation
 - Interactive: real-time code execution and output display

Quarto documents

- “Notebook” Style: supports interactive code and text
 - Code cells: segments for code execution
 - Text chunks: annotations or explanations in Markdown format.
- Inline output: figures and code output display directly below the corresponding code cell

The screenshot shows the Quarto editor interface with the following components:

- YAML Header:** A box pointing to the top section of the document containing YAML metadata:

```
---  
title: "Example Quarto Document"  
format: pdf  
name: Casey Breen  
---
```
- Text Chunks:** A box pointing to the "Quarto" section, which contains introductory text about Quarto and its features.
- Code Chunk:** A box pointing to the "Running Code" section, which shows an R code chunk:

```
{r}  
1 + 1
```

 and its output:

```
[1] 2
```

. Below it, another code chunk is shown:

```
{r}  
# echo: false  
2 * 2
```

 and its output:

```
[1] 4
```

.

Annotations with arrows point from the labels to their respective parts in the Quarto interface.

Installing packages

- Packages: pre-built code and functions.
- Packages are generally installed from the Comprehensive R Archive Network (CRAN)

Install: new packages

```
1 install.packages("tidyverse")
```

Library: load installed packages

```
1 library(tidyverse)
```

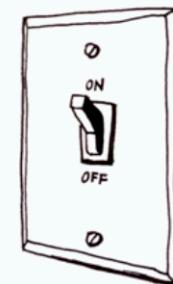
Installing a package

```
install.packages('my.package')
```



Loading a package

```
library('mypackage')
```



YaRrr! The Pirates Guide to R. Nathaniel D. Phillips, 2018.

Running code

- Run all code in a quarto document (or R script, or R notebook)
 - Exception: install packages, quick checks in console
- To run a single line of code in a code cell
 - Cursor over line, **Ctrl + Enter** (Windows/Linux) or **Cmd + Enter** (Mac).
- To run a full code cell (or script)
 - **Ctrl + Shift + Enter** (Windows/Linux) or **Cmd + Shift + Enter** (Mac).

Live coding demo

- Demo of creating a new Quarto document and running code in a code cell
- Your turn next...

In-class exercise 0

- Create a new quarto document
 - File → New File → Quarto Document → Create
- Create a new code cell
 - Insert → Executable cell → R
- Practice running code below

```
1 3+3
```

```
[1] 6
```

```
1 print("Thank you for attending the intro to R session!")
```

```
[1] "Thank you for attending the intro to R session!"
```

Module 2

R programming fundamentals

Learning objectives:

- Comprehend R objects and functions
- Master basic syntax, including comments, assignment, and operators
- Understand data structures and types in R

Objects

- Everything in R is an object
 - **Vectors:** Ordered collection of same type
 - **Data Frames:** Table of columns and rows
 - **Function:** Reusable code block
 - **List:** Ordered collection of objects

```
1 ## Objects in R
2
3 ## Numeric like `1`, `2.5`
4 x <- 2.5
5
6 ## Character: Text strings like '"hello"'
7 y <- "hello"
8
9 ## Boolean: `TRUE`, `FALSE`
10 z <- TRUE
11
12 ## Vectors
13 vec1 <- c(1, 2, 3)
14 vec2 <- c("a", "b", "c")
15
16 ## data.frames
17 df <- data.frame(vec1, vec2)
```

Functions

- Built-in “base” functions

```
1 ## Functions in R
2 result_sqrt <- sqrt(25)
3 result_sqrt
```

[1] 5

- Custom, user-defined functions

```
1 # User-Defined Functions: Custom functions
2 my_function <- function(a, b) {
3   return(a^2 + b)
4 }
5
6 my_function(2, 3)
```

[1] 7

- Functions from packages

```
1 # User-Defined Functions: Custom functions
2
3 library(here) ## library package here
4 here() ## run custom "here" function to print out working directory
```

[1] "/Users/cb48679/workspace/intro_r"

Comments

- Use **#** to start a single-line comment
- Comments are an important way to document code

```
1 ## Add comments
2
3 x <- 7 # assigns 1 to x
4
5 ## the line below won't assign 12 to x because it's commented out
6 # x <- 12
7
8 x
```

```
[1] 7
```

Assignment operators

- Use `<-` or `=` for assignment
 - `<-` is preferred and advised for readability
- Formally, assignment means “assign the result of the operation on the right to object on the left”

```
1 ## Add comments
2
3 x <- 7 # assigns 7 to x
4
5 ## Question: what does this do?
6 y <- x
```

Arithmetic operators

- Addition / Subtraction

```
1 ## R as a calculator (# adds a comment)
2 ## Addition
3 10 + 3
```

```
[1] 13
```

```
1 ## Subtraction
2 4 - 2
```

```
[1] 2
```

- Multiplication / division

```
1 ## Multiplication
2 4 * 3
```

```
[1] 12
```

```
1 ## Division
2 12 / 6
```

```
[1] 2
```

- Exponents

```
1 ## exponents
2 10^2 ## or 10 ** 2
```

```
[1] 100
```

Comparison and logical operators

Operators

Operator	Symbol
AND	&
OR	
NOT	!
Equal	==
Not Equal	!=
Greater/Less Than	> or <
Greater/Less Than or Equal	>= or <=
Element-wise In	%in%

Examples

```
1 ## Logical operators  
2  
3 10 == 10
```

```
[1] TRUE
```

```
1 9 == 10
```

```
[1] FALSE
```

```
1 9 < 10
```

```
[1] TRUE
```

```
1 "apple" %in% c("bananas", "oranges")
```

```
[1] FALSE
```

```
1 "apple" %in% "bananas" | "apple" %in% "apple"
```

```
[1] TRUE
```

```
1 "apple" %in% "bananas" & "apple" %in% "apple"
```

```
[1] FALSE
```

Data structures

- There are lots of data structures; we'll focus on **vectors** and **data frames**.
 - **Vectors**: One-dimensional arrays that hold elements of a single data type (e.g., all numeric or all character).
 - **Data frames**: Two-dimensional tables where each column can have a different data type; essentially a list of vectors of equal length.

Vectors and data frames

- Vector example

```
1 ## Vector Example
2 vec_example <- c(1, 2, 3, 4, 5)
3
4 vec_example ## prints out vec_example
```

```
[1] 1 2 3 4 5
```

- Data frame example

```
1 # Data.frame example
2 example_df <- data.frame(
3   ID = c(1, 2, 3, 4),
4   Name = c("Alice", "Bob", "Charlie", "David"),
5   Age = c(25, 30, 35, 40),
6   Score = c(90, 85, 88, 76)
7 )
8
9 example_df ## prints out df_example
```

	ID	Name	Age	Score
1	1	Alice	25	90
2	2	Bob	30	85
3	3	Charlie	35	88
4	4	David	40	76

Data types

- Each `vector` or `data frame` column can only contain one data type:
 - **Numeric**: Used for numerical values like integers or decimals.
 - **Character**: Holds text and alphanumeric characters.
 - **Logical**: Represents binary values - TRUE or FALSE.
 - **Factor**: Categorical data, either ordered or unordered, stored as levels.

```
1 ## generate vectors
2 vec1 <- c(1, 2, 3)
3 vec2 <- c("a", "b", "c")
4
5 ## check type
6 class(vec1)
```

```
[1] "numeric"
```

```
1 class(vec2)
```

```
[1] "character"
```

NA (missing) values in R

- NA represents missing or undefined data.
 - Can vary by data type (e.g., NA_character_ and NA_integer_)
- NA values can affect summary statistics and data visualization.
- What happens when you run the code below?

```
1 vec <- c(1, 2, 3, NA)
2 mean(vec)
```

Generating sequences in R

- Method 1: Manually write out sequence using `c()`

```
1 ## Basic  
2 c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
  
[1] 1 2 3 4 5 6 7 8 9 10
```

- Method 2: Colon operator (`:`), creates sequences with increments of 1

```
1 c(1:10)  
  
[1] 1 2 3 4 5 6 7 8 9 10
```

- Method 3: `seq()` Function: More flexible and allows you to specify the `start`, `end`, and `by` parameters.

```
1 ## seq 1-10, by = 2  
2 seq(1, 10, by = 2)  
  
[1] 1 3 5 7 9
```

Functions

- Function: Input arguments, performs operations on them, and returns a result
- For each of the below functions, what are the:
 - Input arguments?
 - Operations performed?
 - Results?

```
1 ## hint: rnorm simulates random draws from a standard normal distribution
2 random_draws <- rnorm(n = 5,
3   mean = 0,
4   sd = 1)
5
6 ## find the mean
7 mean(random_draws)
```

```
[1] 0.03602792
```

```
1 ## find the median
2 median(random_draws)
```

```
[1] -0.1229579
```

```
1 ## find the standard deviation
2 sd(random_draws)
```

```
[1] 0.4797366
```

Keyboard shortcuts

Insert new code cell

- macOS: `Cmd + Option + I`
- Windows/Linux: `Ctrl + Alt + I`

Run full code cell or script

- macOS: `Cmd + Shift + Enter`
- Windows/Linux: `Ctrl + Shift + enter`

Assignment operator (creates `<-`)

- macOS: `option + -`
- Windows/Linux: `option + -`

Live coding demo

- Assignment (e.g., `x <- 4`)
- Logical expressions (e.g., `x > 10`)
- Creating a basic sequence
- Your turn next...

In-class exercise 1

1. Assign `x` and `y` to take values 3 and 4.
2. Assign `z` as the product of `x` and `y`.
3. Write code to calculate the square of 3. Assign this to a variable `three_squared`.
4. Write a logical expression to check if `three_squared` is greater than 10.
5. Write a logical expression testing whether `x` is *not* greater than 10. Use the `negate` symbol (`!`).

Exercise 1 solutions

1. Assign `x` and `y` to take values 3 and 4.

```
1 x <- 3  
2 y <- 4
```

2. Assign `z` as the product of `x` and `y`.

```
1 z <- x * y
```

3. Calculate the square of 3 and assign it to a variable called `three_squared`.

```
1 three_squared <- 3^2
```

4. Write a logical expression to check if `three_squared` is greater than 10.

```
1 three_squared > 10
```

[1] FALSE

5. Write a logical expression to test whether `three_squared` is *not* greater than 10. Use the `negate` symbol (`!`).

```
1 !three_squared > 10
```

[1] TRUE

Module 3

Working with **vectors** and **data frames**

Learning objectives

- Select elements from **vectors** and columns from **data frames**
- Subset **data frames**
- Investigate characteristics of **data frames**

Indexing vectors

- Basic indexing

```
1 vec <- c(1, 2, 3, 4, 5)
2 first_element <- vec[1]
3 first_element
```

```
[1] 1
```

```
1 third_element <- vec[3]
2 third_element
```

```
[1] 3
```

- Conditional indexing

```
1 vec <- seq(5, 33, by = 2)
2 vec[vec > 25]
```

```
[1] 27 29 31 33
```

Working with data frames

- Data frames are the most common and versatile data structure in R
- Structured as rows (observations) and columns (variables)

```
1 test_scores <- data.frame(  
2   id = c(1, 2, 3, 4, 5),  
3   name = c("Alice", "Bob", "Carol", "Dave", "Emily"),  
4   age = c(25, 30, 22, 28, 24),  
5   gender = c("F", "M", "F", "M", "F"),  
6   score = c(90, 85, 88, 92, 89)  
7 )  
8  
9 knitr::kable(test_scores)
```

	id	name	age	gender	score
	1	Alice	25	F	90
	2	Bob	30	M	85
	3	Carol	22	F	88
	4	Dave	28	M	92
	5	Emily	24	F	89

Working with data frames

- `head()`- looks at top rows of the `data frame`
- `$` operator - access a column as a `vector`

```
1 ## print first two rows first row
2 head(test_scores, 2)
```

```
id name age gender score
1 1 Alice 25      F    90
2 2   Bob 30      M    85
```

```
1 ## access name column
2 test_scores$name
```

```
[1] "Alice" "Bob"   "Carol" "Dave"  "Emily"
```

Subsetting data frames

- Methods:
 - `$`: Single column by name.
 - `df[i, j]`: Row `i` and column `j`.
 - `df[i:j, k:l]`: Rows `i` to `j` and columns `k` to `l`.
- Conditional Subsetting: `df[df$age > 25,]`.

```
1 ## all rows, columns 1-3
2 test_scores[,1:3]
```

```
id  name age
1  1 Alice 25
2  2 Bob  30
3  3 Carol 22
4  4 Dave  28
5  5 Emily 24
```

```
1 ## all columns, rows 4-5
2 test_scores[4:5, ]
```

```
id  name age gender score
4  4 Dave  28      M    92
5  5 Emily 24      F    89
```

Quiz

Which rows and will this return?

```
1 test_scores[1:3,]
```

- Which rows and which columns will this return?

```
1 test_scores[test_scores$score >= 90, ]
```

Answers

```
1 test_scores[1:3,]
```

```
  id  name age gender score
1 1 Alice 25      F     90
2 2 Bob   30      M     85
3 3 Carol 22      F     88
```

```
1 test_scores[test_scores$score >= 90, ]
```

```
  id  name age gender score
1 1 Alice 25      F     90
4 4 Dave  28      M     92
```

Explore data frame characteristics

Check number of rows

```
1 ## check number of rows (observations)
2 nrow(test_scores)
```

[1] 5

Check number of columns

```
1 ## check number of columns (variables)
2 ncol(test_scores)
```

[1] 5

Check column names

```
1 names(test_scores)
```

[1] "id" "name" "age" "gender" "score"

Module 5

Data manipulation and visualization

Learning objectives

- Overview of `tidyverse` suite of packages
- Fundamentals of data manipulation with `dplyr`
- Data visualization with `ggplot`

Tidyverse

- Packages: Collection of R packages designed for data science.
- Data manipulation: Simplifies data cleaning and transformation with **dplyr**.
- Data Visualization: Enables advanced plotting with **ggplot2**.



Data Manipulation using dplyr

filter: Select rows based on conditions.

```
1 filtered_df <- filter(df, age > 21)
```

select: Choose specific columns

```
1 filtered_df <- select(df)
```

mutate: Add or modify columns

```
1 df <- mutate(df, age_next_year = age + 1)
```

summarize or **summarise**: Aggregate or summarize data based on some criteria

```
1 filtered_df <- summarize(df, mean(age))
```

group_by: Group data by variables. Often used with **summarise()**.

```
1 filtered_df <- df %>%
2   group_by(gender) %>%
3   summarize(mean(age))
```

The Pipe Operator `%>%` (or `|>`) in R

- Takes the output of one function and passes it as the first argument to another function
 - “And then do...”
- What’s the below code doing?

```
1 filtered_df <- df %>%
2   group_by(gender) %>%
3   summarize(mean(age))
```

Recoding values in R

- Sometime you want to recode a variable to take different values (e.g., recoding exact income to binary high/low income variable)
- The `case_when()` function in R is part of the `dplyr` package and is used for creating new variables based on multiple conditions:

```
1 df_new <- df %>%
2   mutate(new_var = case_when(
3     condition1 ~ value1,
4     condition2 ~ value2,
```

Live coding demo

- Filter data
- Selecting data
- Calculating summary statistics by group
- Creating and recoding variables

Your turn

```
1 # install.packages(tidyverse) ## you only need to do this once!
2 library(tidyverse)
```

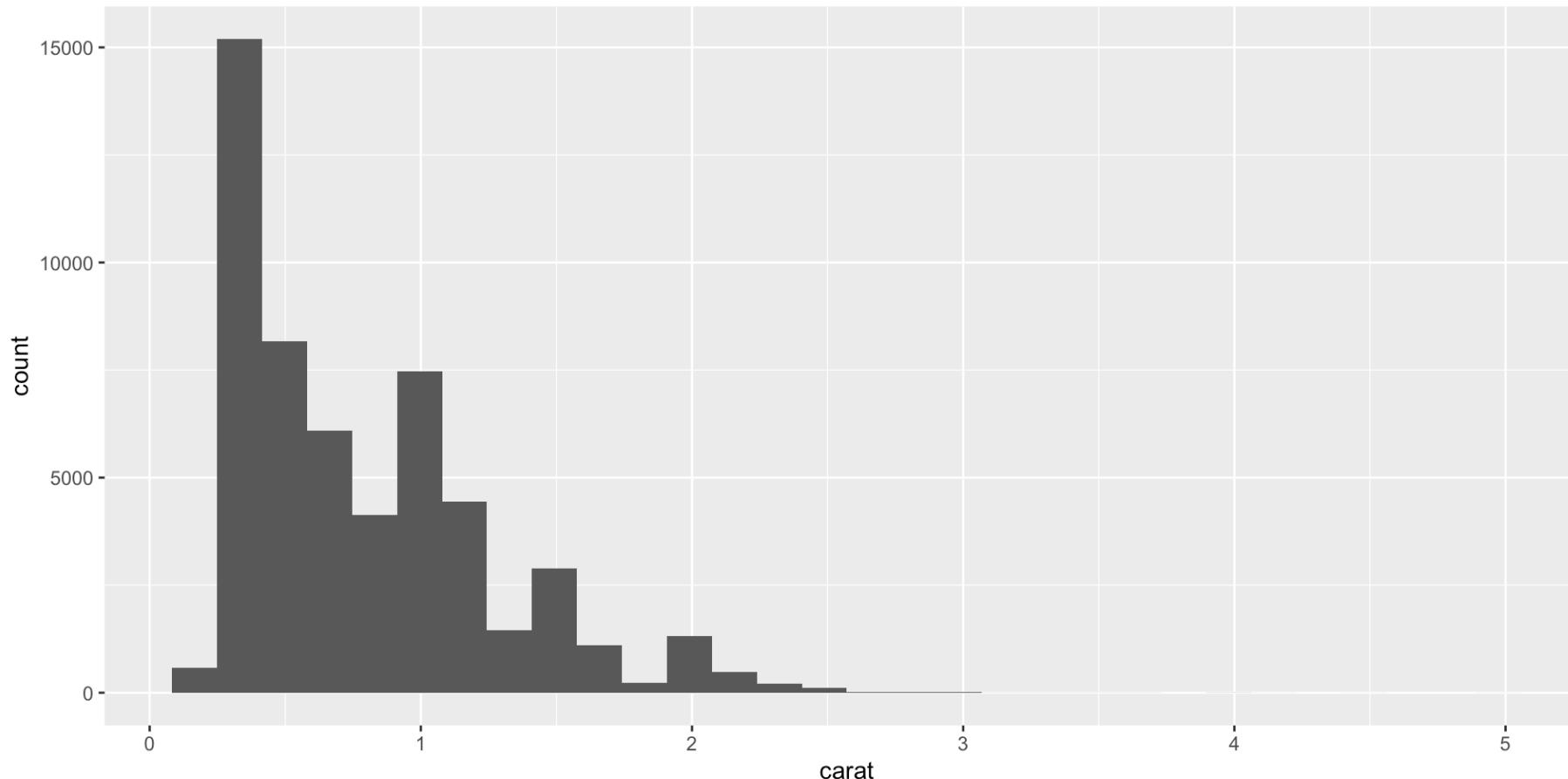
```
1 ## load a built in dataset
2 data(diamonds, package = "ggplot2")
3
4 ## print first few rows
5 head(diamonds)
```

A tibble: 6 × 10

	carat	cut	color	clarity	depth	table	price	x	y	z
1	<dbl>	<ord>	<ord>	<ord>	<dbl>	<dbl>	<int>	<dbl>	<dbl>	<dbl>
1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58	334	4.2	4.23	2.63
5	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48

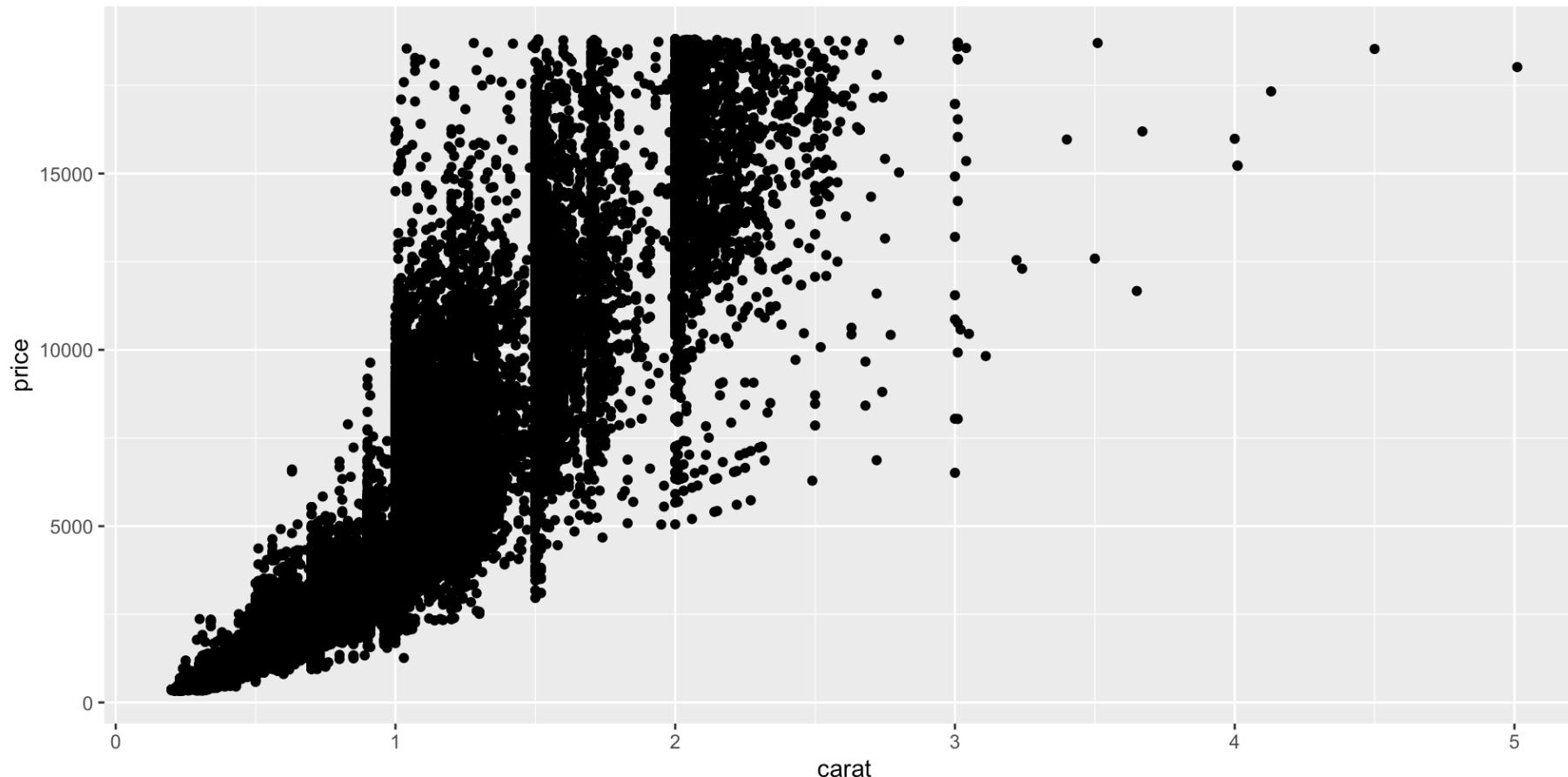
Data Visualization (Distn of carats)

```
1 ## make a histogram of that distribution of carats  
2 ggplot(data = diamonds) +  
3   geom_histogram(aes(x = carat))
```



Data Visualization (Carat vs. Price)

```
1 ## plot relationship between carats and price  
2 ggplot(data = diamonds) +  
3   geom_point(aes(x = carat, y = price))
```

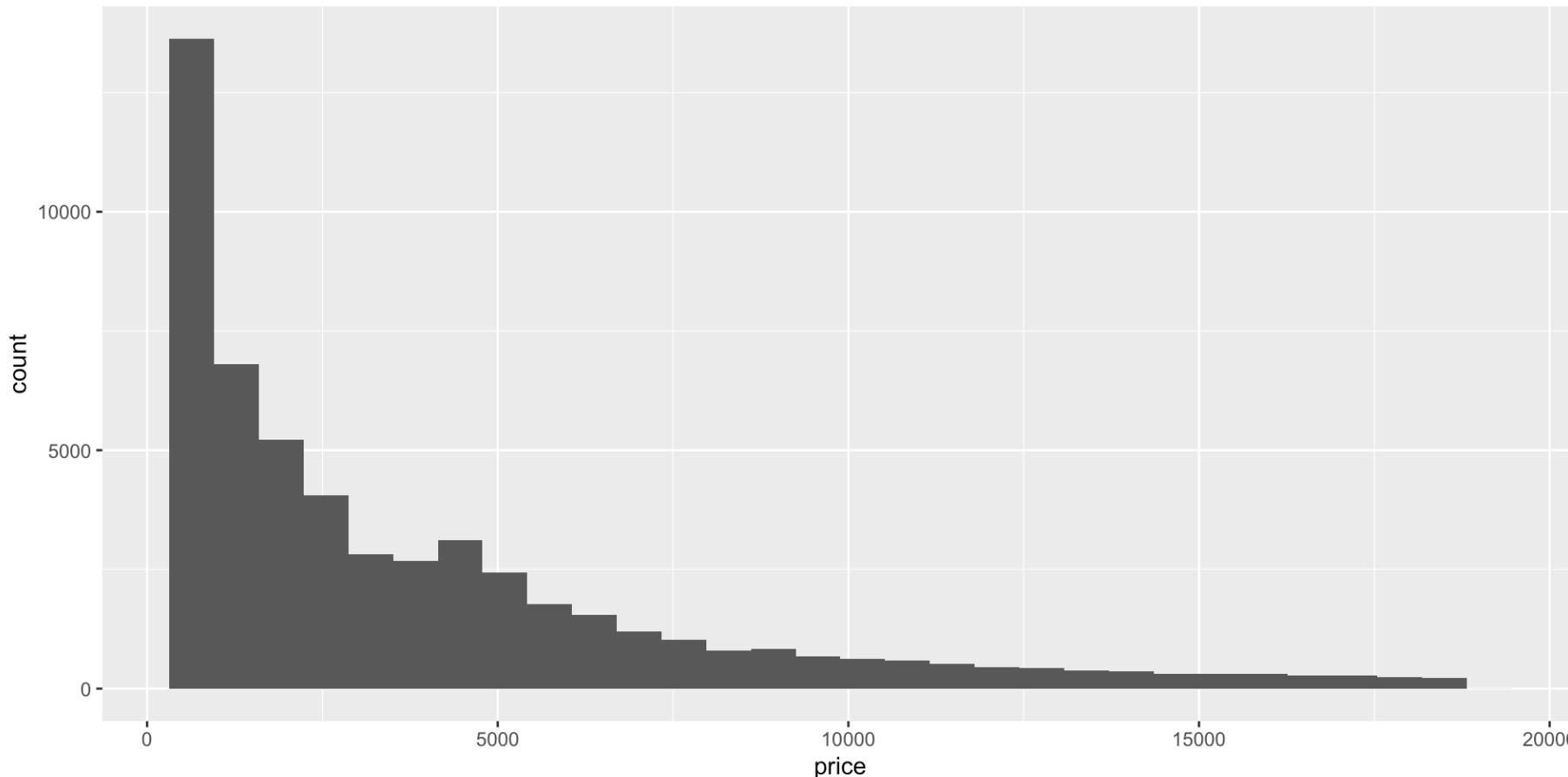


Exercise

1. Make a histogram of the price of diamonds
2. For diamonds great than 1 carat (hint: `filter()`), what is average price by `cut` (hint: `group_by` + `summarize`)?
3. Assign your answer from (2) to a data.frame called `price_by_cut`. Now use `ggplot()` + `geom_col` to visualize this.

Solutions Q1

```
1 ## Price of diamonds  
2 ggplot(data = diamonds) +  
3   geom_histogram(aes(x = price))
```

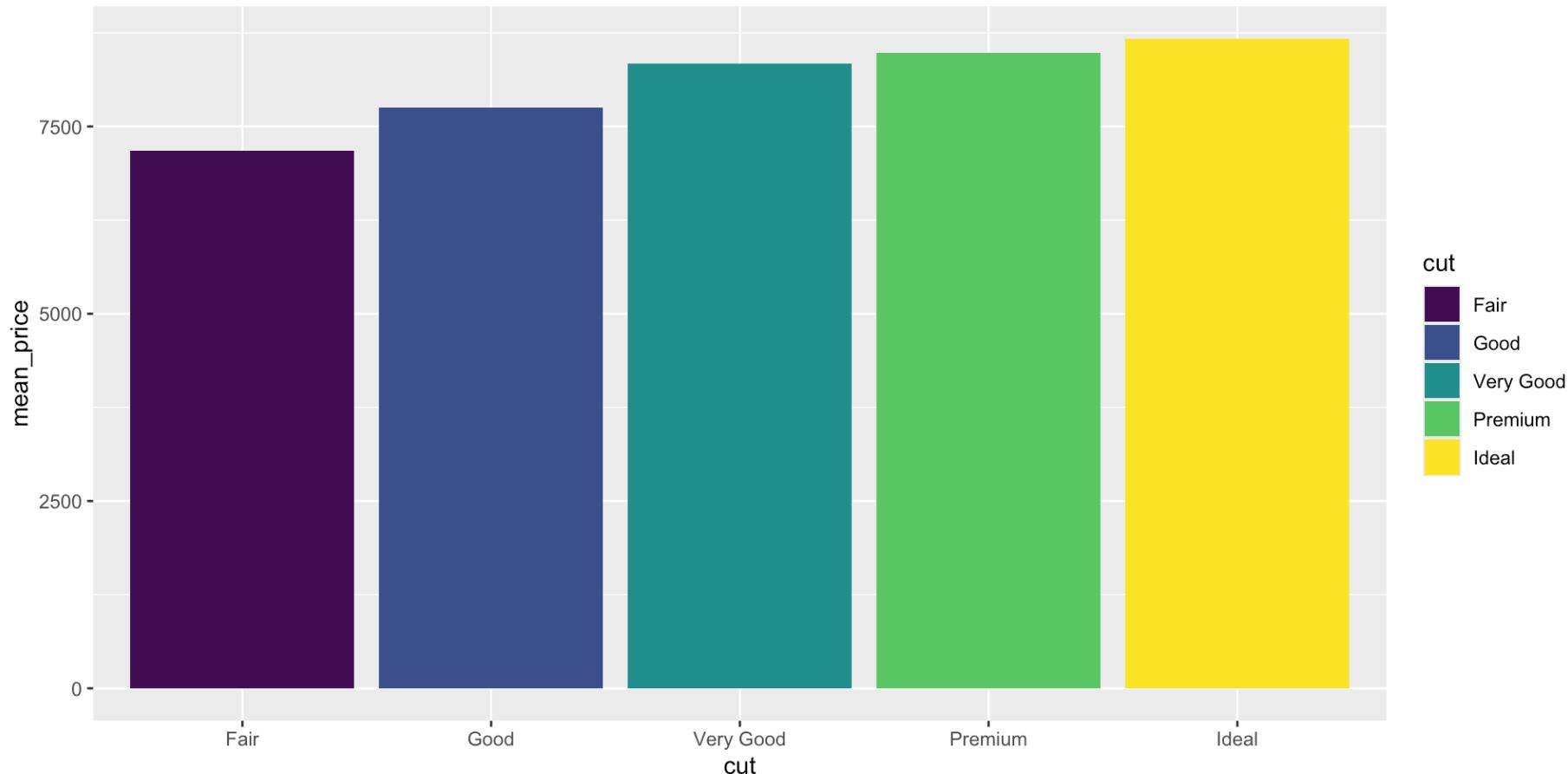


Solutions Q2

```
1 ## Price by cut
2 price_by_cut <- diamonds %>%
3   filter(carat > 1) %>%
4   group_by(cut) %>%
5   summarize(mean_price = mean(price))
```

Solutions Q3

```
1 ## Visualize  
2 ggplot(data = price_by_cut) +  
3   geom_col(aes(y = mean_price, x = cut, fill = cut))
```



Resources for learning more

1. R for data science (<https://r4ds.hadley.nz/>)
2. Data visualization: a practical introduction
(<https://socviz.co/>)

Turn in your lab!

Please turn in your Qmd file (whatever you have completed) on Canvas so you can get credit.