CS506: Data Wrangling and Management

Fall 2025

Marc Tollis, PhD

Table of contents

Preface

Welcome to CS506: Data Wrangling and Management. This course introduces graduate students to data wrangling and management using R and the Tidyverse ecosystem. Students will learn to import, manipulate, clean, and visualize data with a strong emphasis on practical applications and reproducible workflows.

Please access the course syllabus.

The course will utilize the free textbook R for Data Science by Hadley Wickham and Garrett Grolemund.

Course Objectives: Upon successful completion of the course, students will be able to:

- Develop an understanding of R and the Tidyverse ecosystem
- Import structured and unstructured data into R
- Clean and transform data using `dplyr`, `tidyr`, and other core Tidyverse packages
- Visualize data effectively using `ggplot2`
- Conduct exploratory data analysis (EDA)
- Apply data wrangling techniques to real-world datasets

Textbook: *R for Data Science* by Hadley Wickham & Garrett Grolemund (Available for free: https://r4ds.hadley.nz/)

Software Requirements:

- R (https://www.r-project.org/)
- RStudio (https://posit.co/downloads/)

Footnotes

- This is a Quarto book. To learn more about Quarto books visit https://quarto.org/docs/books.
- This website is published using Github Pages.

1 Software

You will need to have all of the following free software downloaded and in working order on your laptop.

> **!** Prior to first lecture
>
> You must have the following on your laptops prior to the first lecture.

- Compatible version of R software environment
- Latest version of RStudio Desktop IDE
- Quarto publishing system (for documents with integrated code).
- You must have a functional PDF Engine to render Quarto (`.qmd`) documents into PDF. See this section on PDF Engines, and be sure to test whether you can render an example `.qmd` file into a PDF.

2 Introduction to R, RStudio, and Quarto

2.1 Learning Objectives

By the end of this week, you should be able to:

- Install and open R, RStudio, and Quarto
- Navigate the four-pane layout of RStudio
- Create and run R scripts
- Understand the differences between the console, script editor, and environment
- Execute basic R operations and understand data types
- Install and load R packages
- Create and render a Quarto (`.qmd`) document to `.pdf`

2.2 Getting Started

R is a programming language designed for data analysis.
RStudio is an Integrated Development Environment (IDE) that makes working with R easier.
Quarto is a tool for creating reproducible documents that combine code and text.

2.3 Installing R, RStudio, and Quarto

1. Install R: https://cran.r-project.org/
2. Install RStudio: https://posit.co/download/rstudio-desktop/
3. Install Quarto: https://quarto.org/docs/get-started/

When you open RStudio, you'll see four panes:

- Console – runs code interactively
- Source – write and save scripts or Quarto documents

- Environment/History – view and manage objects
- Files/Plots/Packages/Help/Viewer – navigation and visualization tools

---

2.4 Introduction to Quarto

Quarto allows you to create documents that include both text and executable R code.

2.4.1 Your First Quarto Document

1. In RStudio: File → New File → Quarto Document
2. Replace the header with:

```
---
title: "My First Quarto Document"
author: "Your Name"
format: pdf
---
```

3. Below the header, add:

```r
x <- c(1, 2, 3, 4, 5)
mean(x)
```

```
[1] 3
```

4. Click Render to produce a PDF file.

2.4.2 In-Class Quarto Exercise

- Create a new Quarto document with:
  - A title, your name, and the date
  - A short paragraph of text
  - A code chunk that calculates the mean and standard deviation of a numeric vector
- Render it to PDF and verify it works.

2.5 Basic R Concepts

2.5.1 Variables and Assignments

```r
x <- 5
y <- 10
z <- x + y
z
```

```
[1] 15
```

2.5.2 Vectors and Functions

```r
ages <- c(25, 30, 35, 40)
mean(ages)
```

```
[1] 32.5
```

```r
sd(ages)
```

```
[1] 6.454972
```

2.5.3 Data Frames

```r
name <- c("Alice", "Bob", "Charlie")
age <- c(25, 30, 35)
student_data <- data.frame(name, age)
student_data
```

```
     name age
1   Alice  25
2     Bob  30
3 Charlie  35
```

2.5.4 Inspecting Data

```r
str(student_data)
```

```
'data.frame':	3 obs. of  2 variables:
 $ name: chr  "Alice" "Bob" "Charlie"
 $ age : num  25 30 35
```

```r
summary(student_data)
```

```
     name                age      
 Length:3           Min.   :25.0  
 Class :character   1st Qu.:27.5  
 Mode  :character   Median :30.0  
                    Mean   :30.0  
                    3rd Qu.:32.5  
                    Max.   :35.0  
```

```r
head(student_data)
```

```
     name age
1   Alice  25
2     Bob  30
3 Charlie  35
```

2.5.5 Comments and Help

```r
# This is a comment
?mean  # Help for the mean function
```

2.5.6 Using Scripts and Console

- Write your code in the script editor and run lines with Ctrl+Enter (Cmd+Enter on Mac)
- Save scripts with the `.R` extension
- Use the Console for quick exploration

2.5.7 Installing and Loading Packages

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

2.5.8 In-Class R Exercises

1. Create a numeric vector of five numbers and calculate its mean, median, and standard deviation.
2. Create a data frame with three columns (name, age, and major) and print its structure.
3. Import a dataset from a URL using `read.csv()` and summarize it using `summary()`.

```r
my_vec <- c(10, 20, 30, 40, 50)
mean(my_vec)
```

```
[1] 30
```

```r
median(my_vec)
```

```
[1] 30
```

```r
sd(my_vec)
```

```
[1] 15.81139
```

```r
df <- data.frame(
  name = c("Lily", "Mark", "Tom"),
  age = c(21, 22, 23),
  major = c("Biology", "Math", "History")
)
str(df)
```

```
'data.frame':   3 obs. of  3 variables:
 $ name : chr  "Lily" "Mark" "Tom"
 $ age  : num  21 22 23
 $ major: chr  "Biology" "Math" "History"
```

```r
data <- read.csv("https://people.sc.fsu.edu/~jburkardt/data/csv/airtravel.csv")
summary(data)
```

```
    Month               X1958           X1959           X1960      
 Length:12          Min.   :310.0   Min.   :342.0   Min.   :390.0  
 Class :character   1st Qu.:339.2   1st Qu.:387.5   1st Qu.:418.5  
 Mode  :character   Median :360.5   Median :406.5   Median :461.0  
                    Mean   :381.0   Mean   :428.3   Mean   :476.2  
                    3rd Qu.:411.8   3rd Qu.:465.2   3rd Qu.:514.8  
                    Max.   :505.0   Max.   :559.0   Max.   :622.0  
```

2.6 Homework Preview

- Create a `.qmd` document that:
  - Includes a title and your name
  - Demonstrates at least three code chunks
  - Shows basic statistics on a numeric vector
  - Imports a dataset, inspects it with `str()` and `summary()`, and writes one paragraph summarizing your findings
- Render to PDF and submit to Canvas.

2.7 Next Steps

You now know how to run R scripts and render Quarto documents.
Next week, you'll learn how to create data visualizations using `ggplot2`.

3 Data Visualization with ggplot2

3.1 Learning Objectives

By the end of this chapter, you should be able to:

- Create basic scatterplots using `ggplot2`
- Map variables to aesthetics (color, size, shape)
- Use different geoms (points, smooth lines, histograms)
- Create facets to display subsets of data
- Customize plots for clear communication

3.2 Introduction to Data Visualization

This week we begin with **visualization first**, following *R for Data Science (Ch. 2)*.
`ggplot2` is part of the tidyverse and implements the **grammar of graphics**.
We will use the built-in `mpg` dataset for examples.

3.3 ggplot2 Basics

The **template** for a ggplot is:

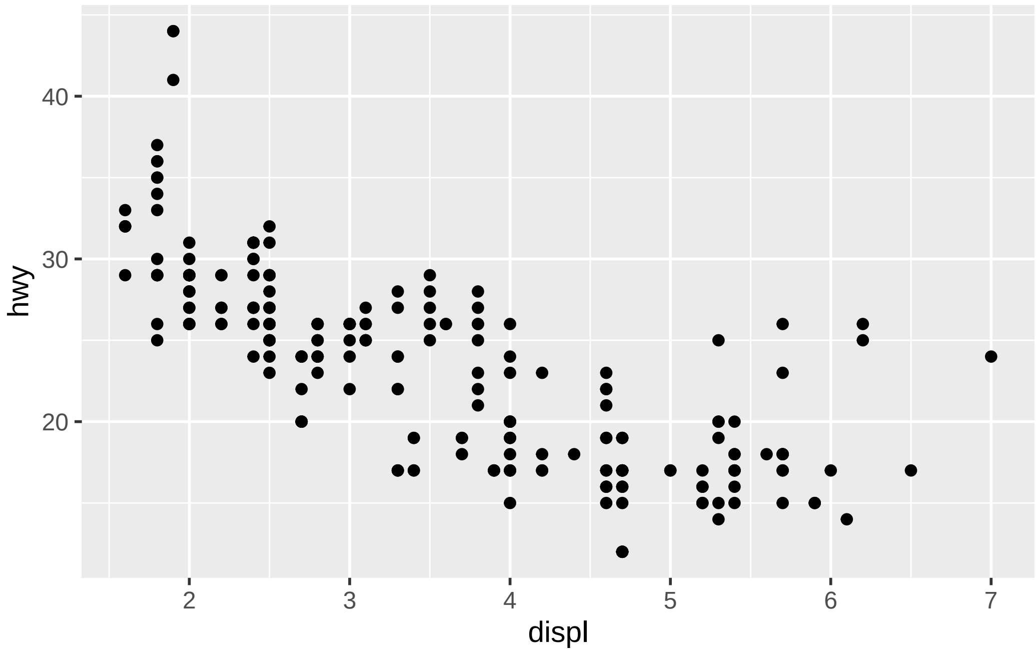
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

3.3.1 Example: Scatterplot of engine size vs. highway mpg

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----------------------- tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



3.3.2 In-Class Exercise 1

1. Create a scatterplot of `cty` (city mpg) vs. `hwy` (highway mpg).

2. What relationship do you see?

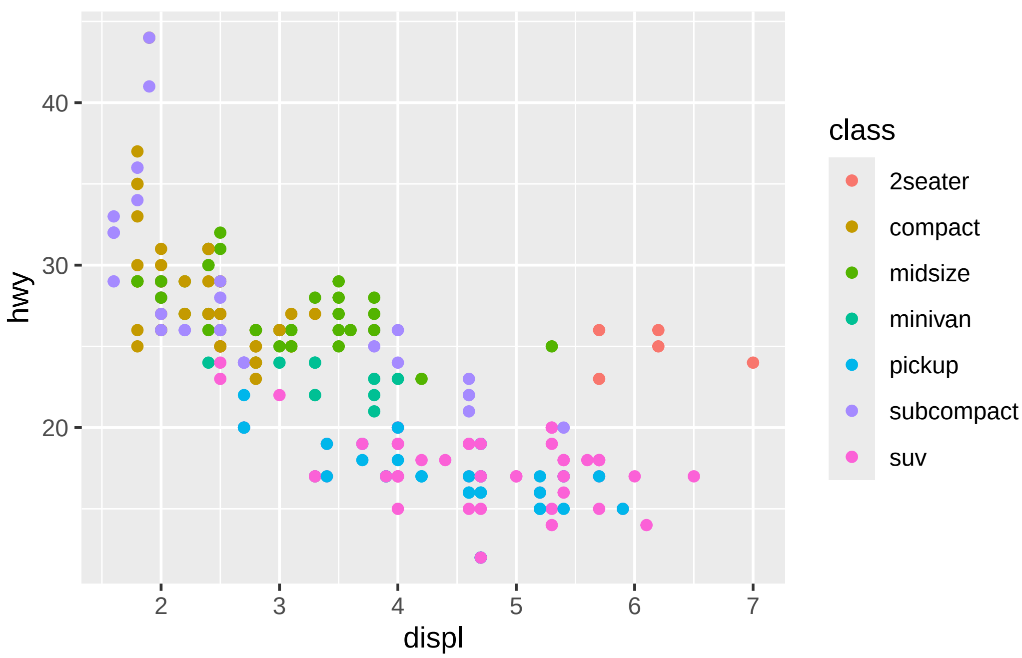
3. Try swapping x and y—does it change the interpretation?

3.3.3 Aesthetic Mappings

You can map variables to visual properties: color, size, shape, alpha.

3.3.4 Example: Color by class

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



3.3.5 In-Class Exercise 2

- Modify the plot to map `size` to `cyl` (number of cylinders).
- Map `shape` to `drv` (drive type).
- Try using both color and shape in one plot.

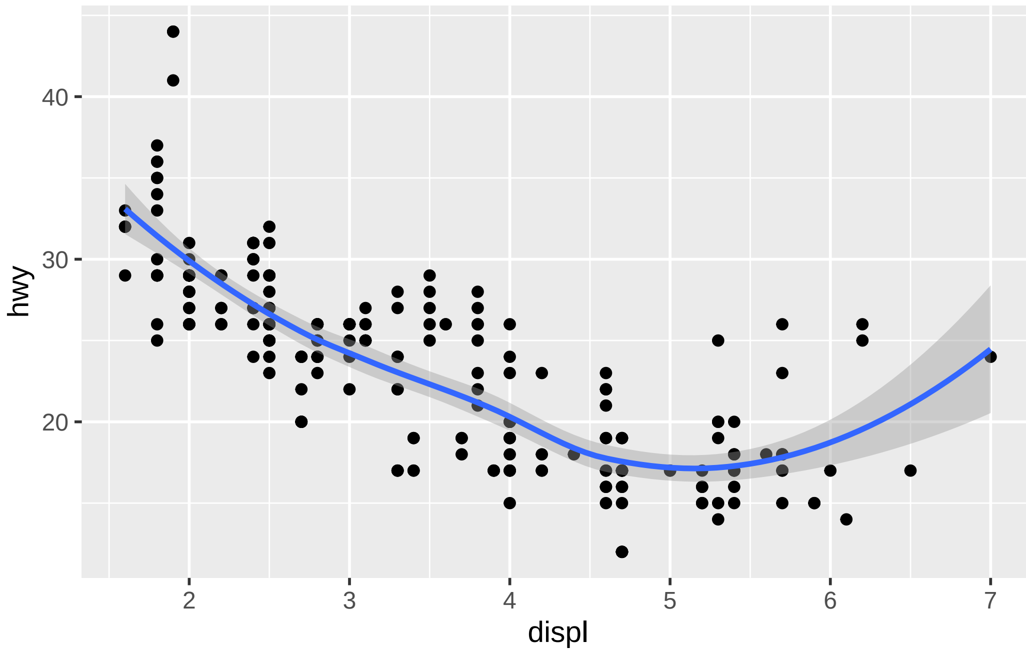
3.4 Adding Geoms

The `geom_point()` function creates a scatterplot, but there are many geoms.

3.4.1 Example: Add a smoothing line

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



3.4.2 In-Class Exercise 3

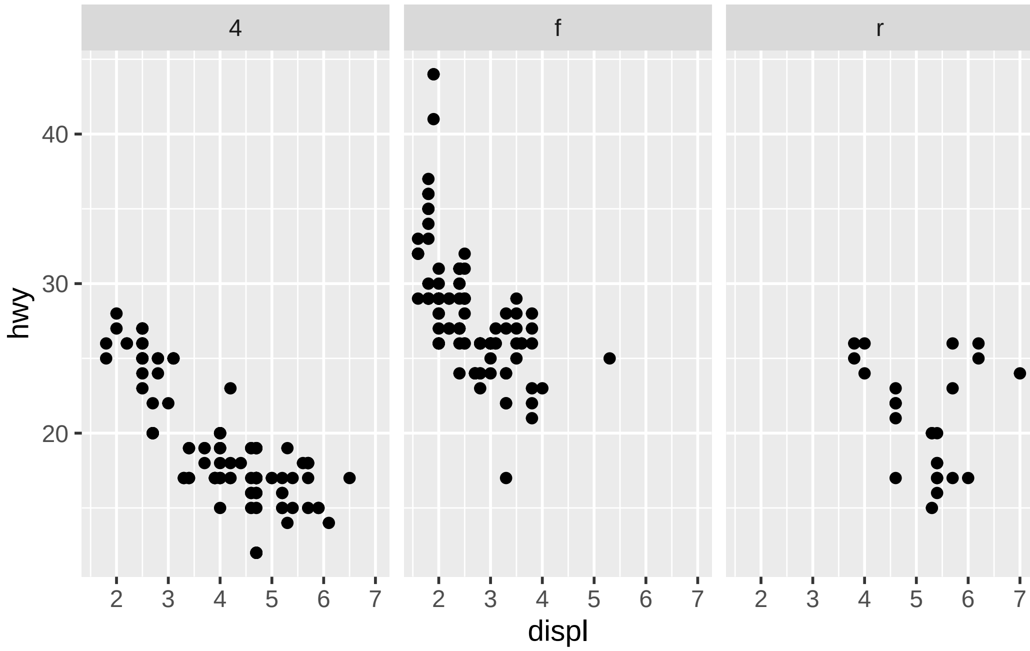
- Add a `geom_smooth()` line to your plot from Exercise 1.
- Try setting `se = FALSE` to remove the confidence band.
- Change the color of the line manually.

3.5 Facets

Facets split the data into subplots based on a variable.

3.5.1 Example: Facet by drive type

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ drv)
```



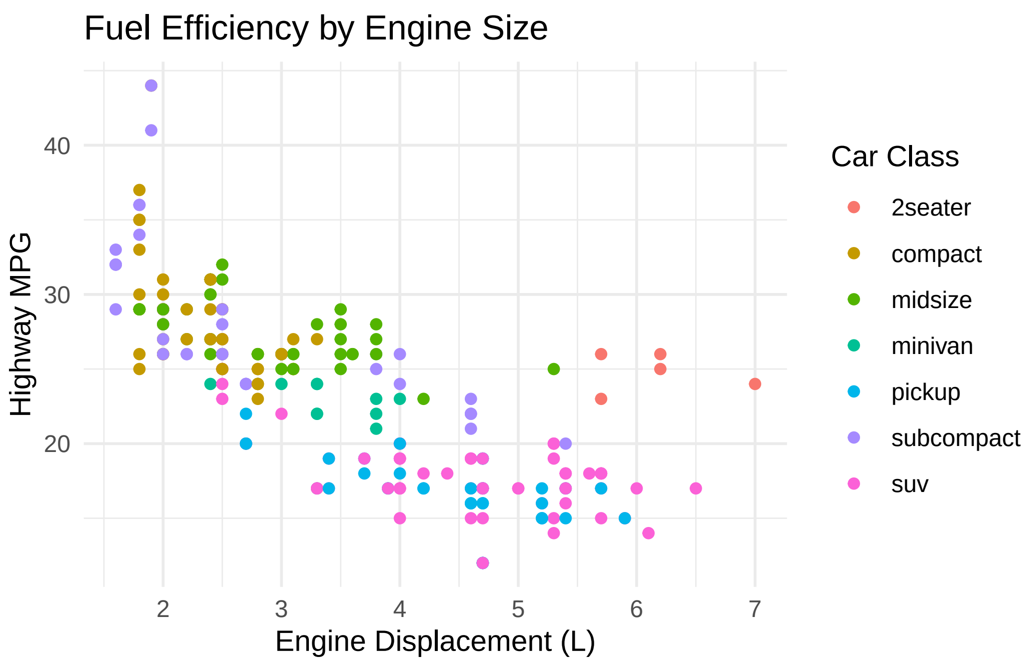
3.5.2 In-Class Exercise 4

- Use `facet_wrap()` to facet the plot by `class`.
- Try `facet_grid(drv ~ cyl)`—what do you observe?

3.6 Customizing Plots

You can add labels, titles, and themes to improve clarity.

```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class)) +
  labs(
    title = "Fuel Efficiency by Engine Size",
    x = "Engine Displacement (L)",
    y = "Highway MPG",
    color = "Car Class"
  ) +
  theme_minimal()
```



3.7 In-Class Challenge

Using the `mpg` dataset:

1. Make a scatterplot of `displ` vs `hwy`.
2. Map a third variable to `color`.

3. Add a smooth line and facet by drive type.

4. Add labels and use a clean theme.

3.8 Homework Preview

For **Homework**, you will:

- Use the `mpg` dataset (or another dataset of your choice).

- Create **three plots**:

  1. A scatterplot with at least one aesthetic mapping
  2. A faceted plot showing subsets of data
  3. A customized plot with titles, labels, and a theme

- Render your `.qmd` to PDF and submit on Canvas.

3.9 Next Steps

Next week, we begin **data transformation** using `dplyr` to manipulate data before plotting.

4 Data Transformation with dplyr (Part 1)

4.1 Learning Objectives

By the end of this chapter, you should be able to:

- Filter rows using `filter()`
- Sort rows using `arrange()`
- Select columns using `select()`
- Create or modify columns using `mutate()`
- Combine multiple transformations using the base R pipe `|>`

4.2 Introduction

This chapter follows *R for Data Science (Ch. 3)* and introduces `dplyr`, a tidyverse package for data transformation.
We will use the `nycflights13::flights` dataset for examples.

4.3 Working with Rows

4.3.1 `filter()`

`filter()` keeps rows that match given conditions.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(nycflights13)

flights |>
  filter(month == 1, day == 1)
```

```
# A tibble: 842 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 832 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

What's a **tibble**? See Appendix C: Tidyverse and Tibbles

4.3.2 `arrange()`

`arrange()` orders rows by a column.

```r
flights |>
  arrange(desc(dep_delay))
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     9      641            900      1301     1242           1530
 2  2013     6    15     1432           1935      1137     1607           2120
 3  2013     1    10     1121           1635      1126     1239           1810
 4  2013     9    20     1139           1845      1014     1457           2210
 5  2013     7    22      845           1600      1005     1044           1815
 6  2013     4    10     1100           1900       960     1342           2211
 7  2013     3    17     2321            810       911      135           1020
 8  2013     6    27      959           1900       899     1236           2226
 9  2013     7    22     2257            759       898      121           1026
10  2013    12     5      756           1700       896     1058           2020
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

4.3.3 In-Class Exercise 1 – Rows

Using the `flights` dataset:

1. Filter for flights departing from **JFK** in **July**.
2. Arrange by **arrival delay** (largest to smallest).
3. Identify the flight with the worst delay.

4.4 Working with Columns

4.4.1 `select()`

`select()` chooses columns.

```r
flights |>
  select(year, month, day, dep_delay, arr_delay)
```

```
# A tibble: 336,776 x 5
    year month   day dep_delay arr_delay
   <int> <int> <int>     <dbl>     <dbl>
 1  2013     1     1         2        11
 2  2013     1     1         4        20
 3  2013     1     1         2        33
 4  2013     1     1        -1       -18
 5  2013     1     1        -6       -25
 6  2013     1     1        -4        12
 7  2013     1     1        -5        19
 8  2013     1     1        -3       -14
 9  2013     1     1        -3        -8
10  2013     1     1        -2         8
# i 336,766 more rows
```

4.4.2 `mutate()`

`mutate()` creates or modifies columns.

```r
flights |>
  mutate(speed = distance / air_time * 60) |>
  select(tailnum, distance, air_time, speed)
```

```
# A tibble: 336,776 x 4
   tailnum distance air_time speed
   <chr>      <dbl>    <dbl> <dbl>
 1 N14228      1400      227  370.
 2 N24211      1416      227  374.
 3 N619AA      1089      160  408.
 4 N804JB      1576      183  517.
 5 N668DN       762      116  394.
 6 N39463       719      150  288.
 7 N516JB      1065      158  404.
 8 N829AS       229       53  259.
 9 N593JB       944      140  405.
10 N3ALAA       733      138  319.
# i 336,766 more rows
```

4.4.3 In-Class Exercise 2 – Columns

1. Select `carrier`, `flight`, `dep_delay`, and `arr_delay`.
2. Create a column `gain = arr_delay - dep_delay`.
3. Display the first 10 rows.

4.5 Using Pipes to Combine Steps

The base R pipe `|>` passes results from one function to the next, making code easier to read.

```
flights |>
  filter(month == 6, origin == "JFK") |>
  select(carrier, flight, dep_delay, arr_delay) |>
  mutate(gain = arr_delay - dep_delay) |>
  arrange(desc(gain)) |>
  head()
```

```
# A tibble: 6 x 5
  carrier flight dep_delay arr_delay  gain
  <chr>    <int>     <dbl>     <dbl> <dbl>
1 B6        2402        -2       142   144
2 DL         706        -3       138   141
3 AA         181        -2       132   134
4 DL        1394       224       350   126
5 B6          83        36       160   124
6 DL         161       278       400   122
```

4.5.1 In-Class Exercise 3 – Pipes

Chain these steps using `|>`:

1. Filter flights from JFK in June.
2. Select `carrier`, `flight`, `dep_delay`, `arr_delay`.
3. Create a column `gain`.
4. Arrange by largest gain and show the top 5.

4.6 Homework Preview

For Homework, you will:

- Use `flights` or another dataset.
- Filter for a subset of interest.
- Create at least two new variables with `mutate()`.
- Sort using `arrange()`.
- Save the transformed dataset and inspect it with `glimpse()` and `summary()`.

Render to PDF and submit on Canvas.

4.7 Next Steps

Next week, we will extend these skills with `group_by()` and `summarize()` to calculate grouped summaries.

5 Data Transformation with dplyr (Part 2)

5.1 Learning Objectives

By the end of this chapter, you should be able to:

- Group data with `group_by()`
- Compute summary statistics with `summarize()`
- Use multiple summaries with grouped data
- Combine multiple datasets using `join` functions
- Practice chaining multiple verbs with the pipe `|>`

5.2 Grouped Summaries

Grouping allows you to calculate statistics **per group**.
We will use the `nycflights13::flights` dataset.

5.2.1 `group_by()` and `summarize()`

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(nycflights13)

flights |>
  group_by(carrier) |>
  summarize(
    delay = mean(dep_delay, na.rm = TRUE)
  )
```

```
# A tibble: 16 x 2
   carrier delay
   <chr>   <dbl>
 1 9E      16.7
 2 AA       8.59
 3 AS       5.80
 4 B6      13.0
 5 DL       9.26
 6 EV      20.0
 7 F9      20.2
 8 FL      18.7
 9 HA       4.90
10 MQ      10.6
11 OO      12.6
12 UA      12.1
13 US       3.78
14 VX      12.9
15 WN      17.7
16 YV      19.0
```

5.3 Multiple Summaries

```r
flights |>
  group_by(dest) |>
  summarize(
    count = n(),
    avg_delay = mean(arr_delay, na.rm = TRUE),
    .groups = "drop"
  )
```

```
# A tibble: 105 x 3
```

```
   dest  count avg_delay
   <chr> <int>     <dbl>
 1 ABQ     254      4.38
 2 ACK     265      4.85
 3 ALB     439     14.4 
 4 ANC       8     -2.5 
 5 ATL   17215     11.3 
 6 AUS    2439      6.02
 7 AVL     275      8.00
 8 BDL     443      7.05
 9 BGR     375      8.03
10 BHM     297     16.9 
# i 95 more rows
```

5.3.1 In-Class Exercise 1 – Grouped Summaries

Using `flights`:

1. Group by `origin` and calculate the average departure delay.

2. Group by `carrier` and find the number of flights and average arrival delay.

3. Which carrier has the highest average arrival delay?

5.4 Grouping with Multiple Variables

You can group by multiple columns at once.

```r
flights |>
  group_by(origin, month) |>
  summarize(
    avg_delay = mean(dep_delay, na.rm = TRUE),
    .groups = "drop_last"
  )
```

```
# A tibble: 36 x 3
# Groups:   origin [3]
   origin month avg_delay
   <chr>  <int>     <dbl>
 1 EWR        1     14.9
 2 EWR        2     13.1
 3 EWR        3     18.1
 4 EWR        4     17.4
 5 EWR        5     15.4
 6 EWR        6     22.5
 7 EWR        7     22.0
 8 EWR        8     13.5
 9 EWR        9      7.29
10 EWR       10      8.64
# i 26 more rows
```

5.4.1 In-Class Exercise 2 – Multiple Grouping

1. Group by `origin` and `carrier`.
2. Summarize with the average `air_time`.
3. Arrange results to see which origin-carrier combination has the longest average flights.

5.5 Joining Datasets

`dplyr` provides functions to join tables by a common key:

- `left_join()`
- `inner_join()`
- `right_join()`
- `full_join()`

Example using `flights` and `airlines`:

```r
flights |>
  left_join(airlines, by = "carrier") |>
  select(name, carrier, flight) |>
  head()
```

```
# A tibble: 6 x 3
  name                   carrier flight
  <chr>                  <chr>    <int>
1 United Air Lines Inc.  UA        1545
2 United Air Lines Inc.  UA        1714
3 American Airlines Inc. AA        1141
4 JetBlue Airways        B6         725
5 Delta Air Lines Inc.   DL         461
6 United Air Lines Inc.  UA        1696
```

5.5.1 In-Class Exercise 3 – Joins

1. Use `left_join()` to add airline names to the flights dataset.
2. Use `count()` to find how many flights each airline operates.
3. Arrange results by the number of flights.

5.6 Chaining with Pipes

We can combine `group_by()`, `summarize()`, and joins in a single pipeline.

```r
flights |>
  left_join(airlines, by = "carrier") |>
  group_by(name) |>
  summarize(
    flights = n(),
    avg_delay = mean(dep_delay, na.rm = TRUE),
```

```r
    .groups = "drop"
  ) |>
  arrange(desc(avg_delay))
```

```
# A tibble: 16 x 3
   name                        flights avg_delay
   <chr>                         <int>     <dbl>
 1 Frontier Airlines Inc.          685     20.2
 2 ExpressJet Airlines Inc.      54173     20.0
 3 Mesa Airlines Inc.              601     19.0
 4 AirTran Airways Corporation    3260     18.7
 5 Southwest Airlines Co.        12275     17.7
 6 Endeavor Air Inc.             18460     16.7
 7 JetBlue Airways               54635     13.0
 8 Virgin America                 5162     12.9
 9 SkyWest Airlines Inc.            32     12.6
10 United Air Lines Inc.         58665     12.1
11 Envoy Air                     26397     10.6
12 Delta Air Lines Inc.          48110      9.26
13 American Airlines Inc.        32729      8.59
14 Alaska Airlines Inc.            714      5.80
15 Hawaiian Airlines Inc.          342      4.90
16 US Airways Inc.               20536      3.78
```

5.7 In-Class Challenge

Using the flights dataset:

- Join airline names
- Group by airline name
- Summarize number of flights, average departure delay, and average arrival delay
- Arrange by average arrival delay
- Identify the airline with the longest delays

5.8 Homework Preview

For homework, extend your data transformation by:

- Grouping data by at least one variable
- Calculating at least two summary statistics
- Joining an additional dataset (e.g., airlines, airports)
- Rendering your results as a table in your PDF

5.9 Next Steps

Next week, we will explore **tidy data principles** and learn how to reshape datasets using `tidyr`.

6 Tidy Data with tidyr

6.1 Learning Objectives

By the end of this chapter, you should be able to:

- Explain why tidy data improves analysis and visualization
- Reshape data between wide and long formats using `pivot_longer()` and `pivot_wider()`
- Separate and unite columns using `separate()` and `unite()`
- Apply tidying techniques to messy real-world datasets
- Prepare datasets for use with `dplyr` and `ggplot2`

6.2 Why Tidy Data?

In Week 6, you performed **EDA** on datasets that were already in a usable format.
Real datasets are often messy. **Tidy data** makes it easy to:

- Use `ggplot2` for visualization

- Use `dplyr` for summaries and transformations

- Combine datasets with joins

Principles of Tidy Data (Hadley Wickham):
1. Each variable is a column
2. Each observation is a row
3. Each value is a cell

6.3 Pivoting: Long vs Wide

6.3.1 pivot_longer()

Converts wide data into long (tidy) format.

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
table4a |>
  pivot_longer(cols = c(`1999`, `2000`),
               names_to = "year",
               values_to = "cases")
```

```
# A tibble: 6 x 3
  country     year   cases
  <chr>       <chr>  <dbl>
1 Afghanistan 1999     745
2 Afghanistan 2000    2666
3 Brazil      1999   37737
4 Brazil      2000   80488
5 China       1999  212258
6 China       2000  213766
```

6.3.2 pivot_wider()

Converts long data back into wide format.

```r
table2 |>
  pivot_wider(names_from = type, values_from = count)
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <dbl>  <dbl>      <dbl>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

6.3.3 In-Class Exercise 1 – Pivoting

1. Use `pivot_longer()` to convert `table4a` to long format.
2. Use `pivot_wider()` on `table2` to create separate columns for `type`.
3. Which format is easier to use with `ggplot2` and `dplyr`?

6.4 Separating and Uniting Columns

6.4.1 `separate()`

Splits a column into multiple columns.

```r
table3 |>
  separate(rate, into = c("cases", "population"), sep = "/")
```

```
# A tibble: 6 x 4
  country      year cases  population
  <chr>       <dbl> <chr>  <chr>
1 Afghanistan  1999 745    19987071
```

```
2 Afghanistan  2000 2666   20595360
3 Brazil       1999 37737  172006362
4 Brazil       2000 80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

6.4.2 `unite()`

Combines multiple columns into one.

```r
table5 |>
  unite(new, century, year, sep = "")
```

```
# A tibble: 6 x 3
  country     new   rate
  <chr>       <chr> <chr>
1 Afghanistan 1999  745/19987071
2 Afghanistan 2000  2666/20595360
3 Brazil      1999  37737/172006362
4 Brazil      2000  80488/174504898
5 China       1999  212258/1272915272
6 China       2000  213766/1280428583
```

6.4.3 In-Class Exercise 2 – Separate and Unite

1. Use `separate()` to split the `rate` column in `table3`.

2. Use `unite()` to combine `century` and `year` into one column.

6.5 Tidying a Real Dataset

The `who` dataset is messy: column names encode multiple variables.

Example tidying workflow:

```
who |>
  pivot_longer(cols = starts_with("new"),
               names_to = "key",
               values_to = "cases",
               values_drop_na = TRUE) |>
  separate(key, into = c("type", "sex_age"), sep = "_") |>
  separate(sex_age, into = c("sex", "age"), sep = 1)
```

```
Warning: Expected 2 pieces. Additional pieces discarded in 73466 rows [1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].
```

```
# A tibble: 76,046 x 8
   country     iso2  iso3   year type  sex   age   cases
   <chr>       <chr> <chr> <dbl> <chr> <chr> <chr> <dbl>
 1 Afghanistan AF    AFG    1997 new   s     p         0
 2 Afghanistan AF    AFG    1997 new   s     p        10
 3 Afghanistan AF    AFG    1997 new   s     p         6
 4 Afghanistan AF    AFG    1997 new   s     p         3
 5 Afghanistan AF    AFG    1997 new   s     p         5
 6 Afghanistan AF    AFG    1997 new   s     p         2
 7 Afghanistan AF    AFG    1997 new   s     p         0
 8 Afghanistan AF    AFG    1997 new   s     p         5
 9 Afghanistan AF    AFG    1997 new   s     p        38
10 Afghanistan AF    AFG    1997 new   s     p        36
# i 76,036 more rows
```

6.5.1 In-Class Exercise 3 – WHO Dataset

1. Pivot `who` longer to create `key` and `cases`.

2. Separate `key` into multiple components.

3. Count total cases by country.
4. Which country has the highest reported cases?

6.6 Tidy Data Workflow

After tidying, you can:

- Use `ggplot2` for visualizations
- Use `group_by()` and `summarize()` for summaries
- Join with other datasets

6.7 Homework Preview

For homework, you will:

- Take a messy dataset (e.g., `table4a`, `table5`, or your own)
- Use `pivot_longer()` and/or `pivot_wider()` to reshape it
- Use `separate()` and `unite()` as needed
- Produce a tidy dataset and create **one visualization** and **one grouped summary**
- Render to PDF and submit on Canvas

---

7 Exploratory Data Analysis (EDA)

7.1 Learning Objectives

By the end of this chapter, you should be able to:

- Understand the purpose of exploratory data analysis (EDA)
- Visualize distributions of single variables
- Examine relationships between variables
- Detect patterns, clusters, and outliers
- Use transformations to clarify patterns

7.2 Introduction to EDA

Exploratory Data Analysis (EDA) is about **looking at your data** to find patterns, spot anomalies, and guide your next steps.
We use `ggplot2` to visualize both **univariate** and **bivariate** relationships.

We will use the `diamonds` dataset.

7.3 Visualizing Single Variables

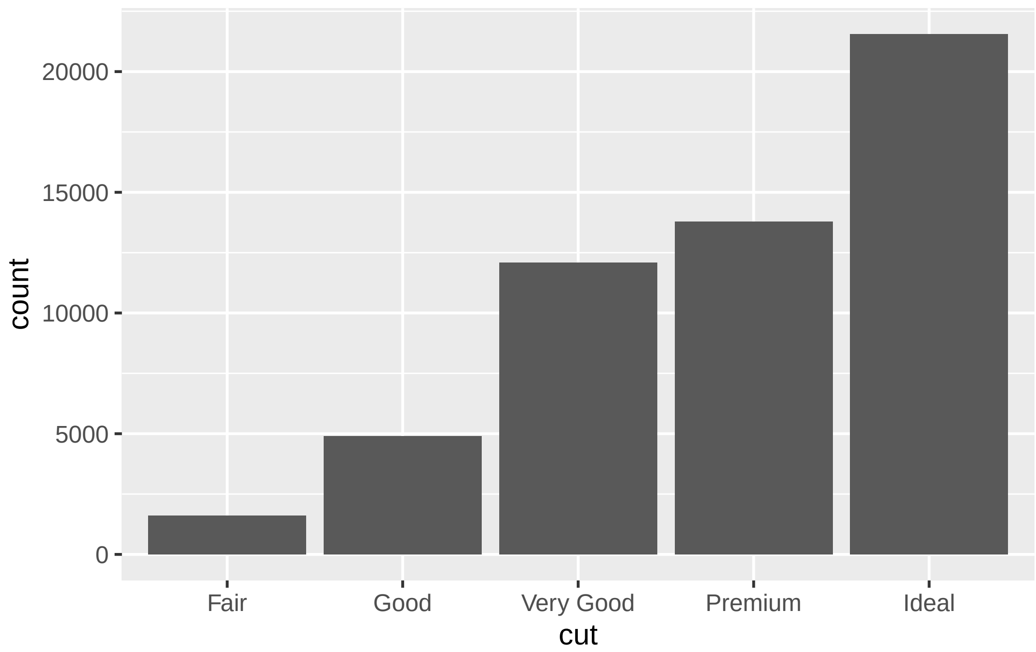
7.3.1 Categorical Variables

Use a bar chart (`geom_bar()`):

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

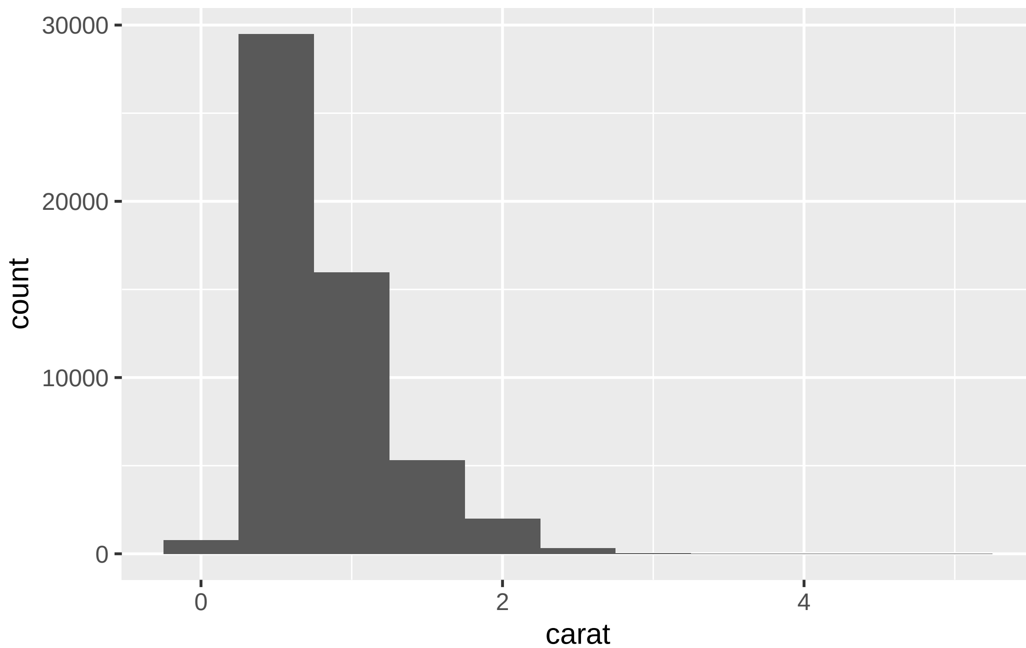
```r
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```



7.3.2 Continuous Variables

Use a histogram (`geom_histogram()`):

```
ggplot(data = diamonds) +
  geom_histogram(mapping = aes(x = carat), binwidth = 0.5)
```



You can also use `geom_freqpoly()` for density curves.

7.3.3 In-Class Exercise 1 – Single Variables

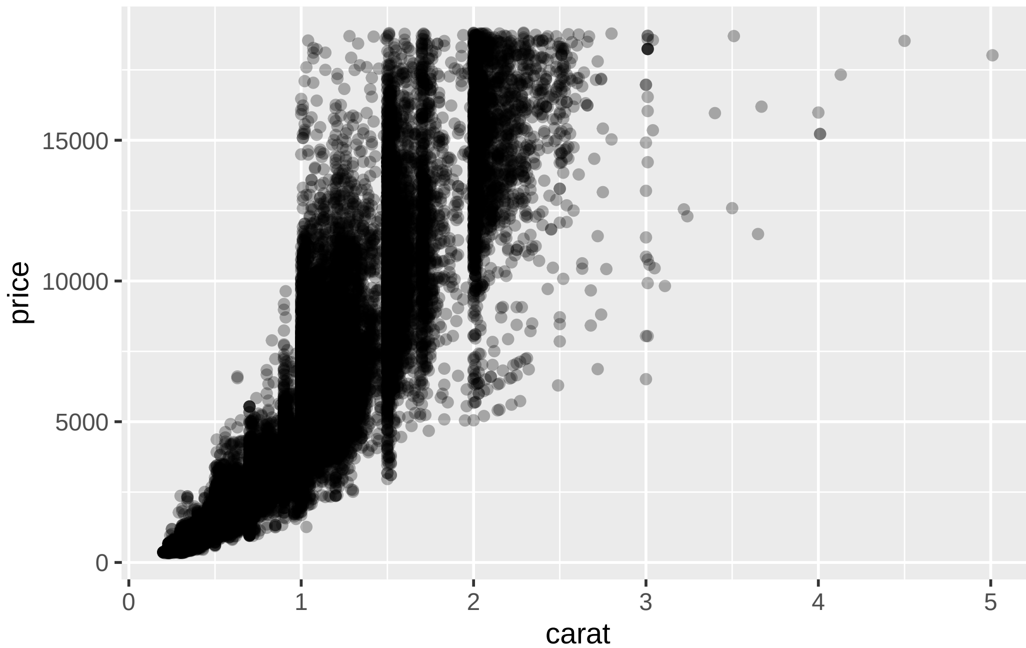
1. Plot the distribution of `color` using a bar chart.
2. Plot a histogram of `price` with a binwidth of 1000.
3. What patterns or anomalies do you see?

7.4 Visualizing Relationships

7.4.1 Two Continuous Variables

Scatterplots show relationships:

```
ggplot(data = diamonds) +
  geom_point(mapping = aes(x = carat, y = price), alpha = 0.3)
```

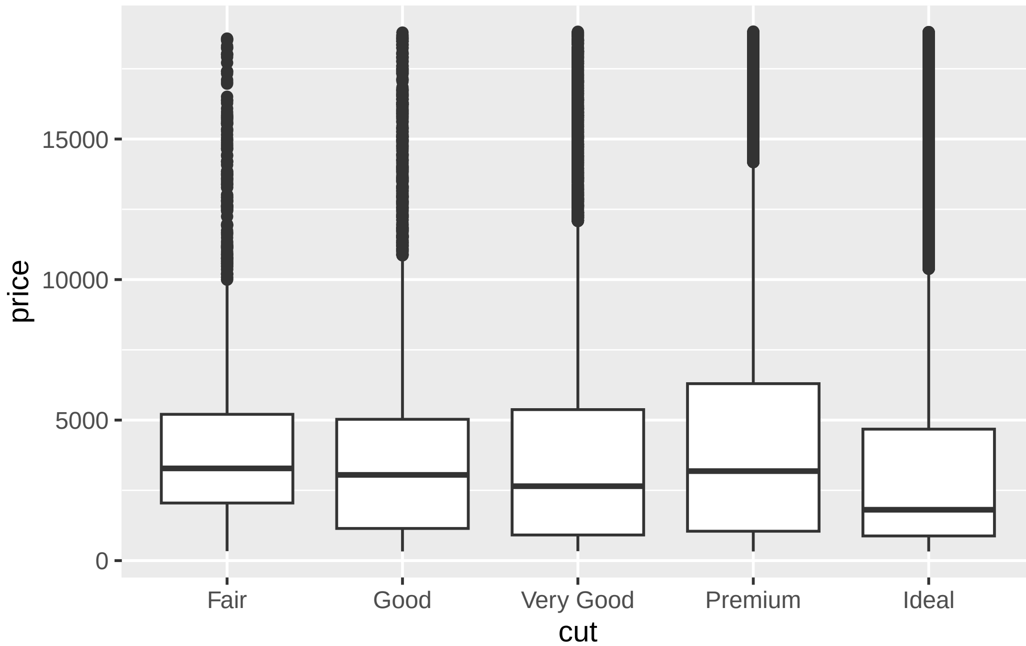


Use `alpha` to reduce overplotting.

7.4.2 Categorical vs. Continuous

Boxplots work well:

```
ggplot(data = diamonds) +
  geom_boxplot(mapping = aes(x = cut, y = price))
```



7.4.3 In-Class Exercise 2 – Relationships

1. Create a scatterplot of `carat` vs `price`.

2. Color the points by `cut`.

3. Make a boxplot of `price` across diamond `color` categories.

7.5 Patterns and Outliers

Look for clusters, gaps, and unusual observations.
You can **filter** or **highlight** outliers.

Example: filter diamonds with unusually high price:

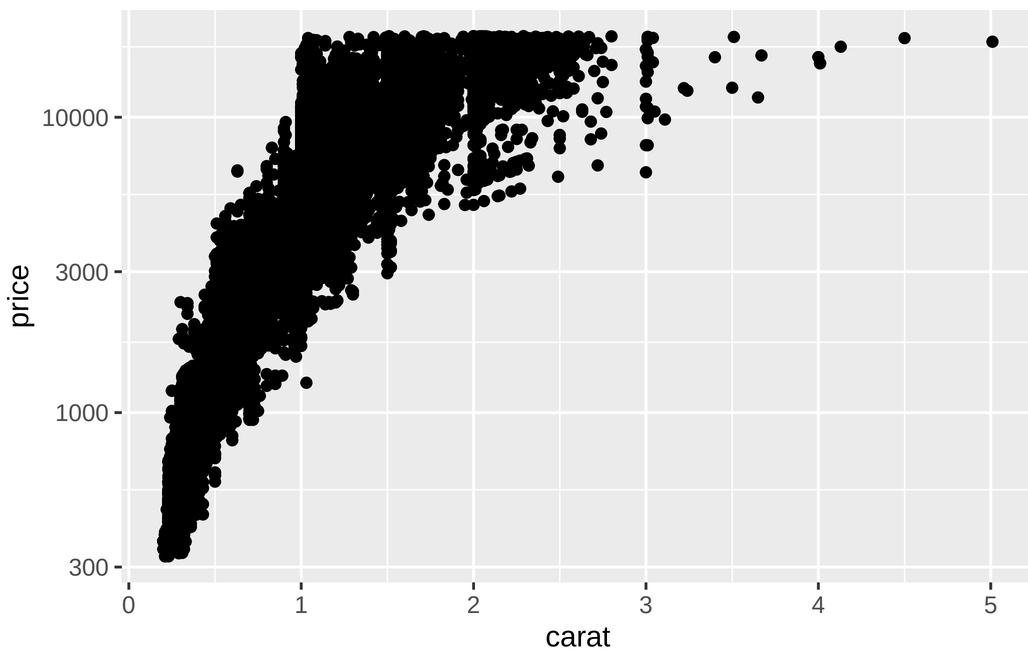
```r
diamonds |>
  filter(price > 15000) |>
  arrange(desc(price)) |>
  head()
```

```
# A tibble: 6 x 10
  carat cut       color clarity depth table price     x     y     z
  <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  2.29 Premium   I     VS2      60.8    60 18823  8.5   8.47  5.16
2  2    Very Good G     SI1      63.5    56 18818  7.9   7.97  5.04
3  1.51 Ideal     G     IF       61.7    55 18806  7.37  7.41  4.56
4  2.07 Ideal     G     SI2      62.5    55 18804  8.2   8.13  5.11
5  2    Very Good H     SI1      62.8    57 18803  7.95  8     5.01
6  2.29 Premium   I     SI1      61.8    59 18797  8.52  8.45  5.24
```

7.5.1 Transformations

Log transformations can reveal patterns in skewed data.

```r
ggplot(data = diamonds) +
  geom_point(mapping = aes(x = carat, y = price)) +
  scale_y_log10()
```



7.5.2 In-Class Exercise 3 – Patterns and Transformations

1. Identify any outliers in the `diamonds` dataset using filters.
2. Apply a log transformation to `price`.
3. Does the relationship between `carat` and `price` become clearer?

7.6 Combining EDA with dplyr

Use `filter()`, `mutate()`, and `group_by()` to enhance your plots.

Example: average price per cut:

```
diamonds |>
  group_by(cut) |>
  summarize(mean_price = mean(price))
```

```
# A tibble: 5 x 2
  cut       mean_price
  <ord>          <dbl>
1 Fair           4359.
2 Good           3929.
3 Very Good      3982.
4 Premium        4584.
5 Ideal          3458.
```

7.6.1 In-Class Challenge – EDA Workflow

- Explore `diamonds` by:
  - Visualizing distributions of at least two variables
  - Plotting relationships between two variables
  - Detecting outliers
  - Applying a transformation to clarify a pattern

7.7 Homework Preview

For homework, you will:

- Choose a dataset (e.g., `diamonds` or your own)
- Create at least **two univariate** visualizations (bar chart, histogram)
- Create at least **two bivariate** visualizations (scatterplot, boxplot)
- Identify any patterns or outliers and describe them in text
- Apply at least one transformation to improve visualization
- Render to PDF and submit

7.8 Next Steps

Next week, we will dive into **Tidy Data** and learn how to reshape messy datasets using `tidyr`.

8 Workflow and Reproducibility

8.1 Learning Objectives

By the end of this chapter, you should be able to:

- Organize your work with R projects
- Use Quarto for reproducible documents
- Follow best practices for naming files and structuring directories
- Incorporate code, text, and output into a single reproducible report
- Use version control with GitHub (optional, for advanced students)

8.2 Why Workflow Matters

Reproducible workflows:

- Make it easy to rerun analyses later
- Allow others to reproduce your results
- Keep projects organized and easy to navigate
- Prevent errors caused by hard-coded file paths and messy code

---

8.3 Organizing Projects in RStudio

8.3.1 RStudio Projects

- Use **File → New Project** for each analysis/course project
- Keep data, scripts, and outputs in **subfolders** (e.g., `data/`, `scripts/`, `figures/`, `docs/`)
- Avoid using absolute paths—use **relative paths** inside the project

---

8.3.2 Example Project Structure

```
my_project/
  data/
    raw_data.csv
  scripts/
    analysis.R
  figures/
    plot1.png
  docs/
    report.qmd
  my_project.Rproj
```

8.3.3 In-Class Exercise 1 – Project Setup

1. Create a new RStudio Project for this course.

2. Make folders: `data`, `scripts`, `outputs`.

3. Save your `.qmd` homework file in the project root.

4. Render your Quarto document and confirm outputs stay organized.

8.4 Quarto for Reproducibility

Quarto allows you to:

- Combine text and code in one document
- Render reports to PDF, HTML, or Word
- Ensure results match the code that generated them

8.4.1 Example Quarto Workflow

```
---
title: "My Analysis"
format: pdf
---
```

```r
library(tidyverse)
data <- read_csv("data/mydata.csv")
summary(data)
```

8.4.2 In-Class Exercise 2 – Quarto Report

1. Create a `.qmd` file that loads a dataset and runs a simple analysis.

2. Add at least one plot and one table.

3. Render to PDF and check the output.

8.5 Best Practices for Reproducibility

- **Use scripts and Quarto documents instead of manual steps**
- **Keep raw data unchanged**; clean data with scripts
- **Document everything**: use comments and text
- **Save figures and tables** programmatically, not manually
- **Render final reports from source code**

8.6 Optional: Version Control with Git and GitHub

For students interested in collaboration and tracking changes:

- Install Git and create a GitHub account
- Use `usethis::use_git()` to initialize Git in a project
- Commit changes regularly and push to GitHub

(We will not cover Git in detail, but this is recommended for your own practice.)

8.6.1 In-Class Challenge – Reproducible Mini-Report

- Set up a project with an organized folder structure
- Create a Quarto document that:
  - Reads a dataset
  - Runs a simple transformation
  - Creates a plot
  - Summarizes the results in text
- Render to PDF and check for a clean, reproducible output

8.7 Homework Preview

For homework, you will:

- Organize your project folder (data, scripts, outputs)
- Create a Quarto report with:
  - One dataset
  - At least one data cleaning step
  - One visualization
  - One table of summary statistics
- Ensure all file paths are relative (not absolute)
- Render to PDF and submit on Canvas

8.8 Next Steps

Next week, we will move into **Data Import** (CSV, Excel, and parsing dates) and continue to build your data wrangling workflow.

9 Data Import with readr and readxl

9.1 Learning Objectives

By the end of this chapter, you should be able to:

- Import CSV and TSV files with `readr`
- Read Excel files with `readxl`
- Understand how column types are parsed
- Parse dates, times, and numbers correctly
- Diagnose and fix import problems

9.2 Reading CSV and TSV Files

The `readr` package (part of the tidyverse) provides fast and friendly functions for reading text data.

9.2.1 Example: Reading a CSV file

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
df <- read_csv("https://people.sc.fsu.edu/~jburkardt/data/csv/airtravel.csv")
```

```
Rows: 12 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (1): Month
dbl (3): 1958, 1959, 1960

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
glimpse(df)
```

```
Rows: 12
Columns: 4
$ Month  <chr> "JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", ~
$ `1958` <dbl> 340, 318, 362, 348, 363, 435, 491, 505, 404, 359, 310, 337
$ `1959` <dbl> 360, 342, 406, 396, 420, 472, 548, 559, 463, 407, 362, 405
$ `1960` <dbl> 417, 391, 419, 461, 472, 535, 622, 606, 508, 461, 390, 432
```

9.2.2 Example: Reading a TSV file

```r
df_tsv <- read_tsv("data/example.tsv")
```

9.2.3 In-Class Exercise 1 – CSV/TSV

1. Download a small CSV file (e.g., from the course repository).
2. Read it into R using `read_csv()`.
3. Inspect its structure with `glimpse()` and `summary()`.
4. What data types were automatically detected?

9.3 Column Types and Parsing

readr automatically guesses column types, but you can override them.

9.3.1 Example: Overriding column types

```
df <- read_csv("data/mydata.csv", col_types = cols(
  id = col_character(),
  date = col_date(format = "%Y-%m-%d")
))
```

You can parse numbers with parse_number(), dates with parse_date(), and times with parse_time().

9.3.2 In-Class Exercise 2 – Parsing

1. Create a vector of messy numbers: c("$100", "250%", "300").
2. Use parse_number() to extract numeric values.
3. Create a vector of dates as strings and use parse_date().

9.4 Importing Excel Files

The readxl package is used to read Excel files (.xls, .xlsx).

9.4.1 Example: Reading an Excel sheet

```
library(readxl)

excel_df <- read_excel("data/example.xlsx", sheet = "Sheet1")
head(excel_df)
```

9.4.2 In-Class Exercise 3 – Excel Import

1. Use a provided Excel file (or download one).
2. Read the first sheet with `read_excel()`.
3. Specify a different sheet and check the result.

9.5 Handling Import Problems

When column parsing fails:

- Use `problems()` to diagnose
- Use `col_types` to fix column types
- Clean data after import using `mutate()`

Example:

```
bad <- read_csv("data/bad.csv")
problems(bad)
```

9.6 Reading Other Formats (Optional)

- `read_delim()` – for custom delimiters
- `read_table()` – for whitespace-delimited files
- `read_lines()` – for line-by-line text
- `jsonlite::fromJSON()` – for JSON files (optional preview)

9.6.1 In-Class Challenge – Import & Clean Workflow

1. Import a messy CSV file with mixed types.

2. Fix incorrect column parsing.

3. Convert a date column to proper Date format.

4. Summarize the data by a grouping variable.

9.7 Homework Preview

For homework, you will:

- Import at least **one CSV** and **one Excel** dataset
- Fix any parsing issues (e.g., column types, dates)
- Clean at least one column with `mutate()`
- Provide a short summary (using `group_by()` and `summarize()`)
- Render to PDF and submit

9.8 Next Steps

Next week, we will learn to **work with text data and regular expressions** using the `stringr` package.

10 Transform: Logical Vectors and Numbers

What are the types of variables we see in data frames, and what are the different tools we can use to work with them?

10.1 Learning Objectives

By the end of this chapter, you should be able to:

- Understand how logical vectors work in R
- Use logical conditions to filter and manipulate data
- Convert between logical, numeric, and character types
- Parse numbers from messy strings

10.2 Logical Vectors

Logical vectors contain only `TRUE`, `FALSE`, or `NA`.

```r
x <- c(TRUE, FALSE, TRUE, NA)
x
```

```
[1]  TRUE FALSE  TRUE    NA
```

10.2.1 Logical comparisons create logical vectors:

```r
nums <- c(2, 5, 8, 1)
nums > 4
```

```
[1] FALSE  TRUE  TRUE FALSE
```

You can use these directly with functions like `sum()` and `mean()`:

```
sum(nums > 4)    # Count how many values are > 4
```

```
[1] 2
```

```
mean(nums > 4)  # Proportion of values > 4
```

```
[1] 0.5
```

10.2.2 In-Class Exercise 1 – Logical Conditions

1. Create a numeric vector with 10 random values.
2. Which values are greater than the mean?
3. What proportion is above the mean?

10.3 Logical Operations

Combine logical vectors with `&` (and), `|` (or), and `!` (not):

```
a <- c(TRUE, FALSE, TRUE)
b <- c(TRUE, TRUE, FALSE)

a & b
```

```
[1]  TRUE FALSE FALSE
```

```
a | b
```

```
[1] TRUE TRUE TRUE
```

```
!a
```

```
[1] FALSE  TRUE FALSE
```

10.3.1 In-Class Exercise 2 – Combining Conditions

1. Using the `mpg` dataset, create a logical condition for cars with `hwy > 30` **and** `cyl == 4`.

2. How many such cars exist?

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
mpg |>
  filter(hwy > 30 & cyl == 4) |>
  nrow()
```

```
[1] 22
```

10.4 Numbers and Coercion

Logical values behave like numbers: `TRUE` = 1, `FALSE` = 0.

```r
as.numeric(c(TRUE, FALSE, TRUE))
```

```
[1] 1 0 1
```

This makes calculations on logical vectors easy.

10.5 Parsing Numbers

Real-world data often stores numbers as text with extra symbols.
Use `readr::parse_number()` to extract numeric values.

```r
library(readr)

x <- c("$100", "200%", "300kg")
parse_number(x)
```

```
[1] 100 200 300
```

10.5.1 In-Class Exercise 3 – Parsing

1. Create a character vector: `c("10 kg", "$50", "30%")`.
2. Use `parse_number()` to convert it to numeric.
3. What happens if there are unexpected characters?

10.6 Dealing with Missing Values

Logical and numeric vectors can contain `NA`.
Handle them with `na.rm = TRUE` or functions like `replace_na()`.

```r
nums <- c(1, 2, NA, 4)
mean(nums, na.rm = TRUE)
```

```
[1] 2.333333
```

10.7 In-Class Challenge – Logical Filtering

- Using `flights` from `nycflights13`, calculate the proportion of flights that departed late (`dep_delay > 0`) **and** arrived on time (`arr_delay <= 0`).

```r
library(nycflights13)

flights |>
  summarize(on_time = mean(dep_delay > 0 & arr_delay <= 0, na.rm = TRUE))
```

```
# A tibble: 1 x 1
  on_time
    <dbl>
1   0.108
```

---

11 Homework Preview

For the next homework, you will:

- Create a numeric vector and use logical comparisons to summarize it
- Filter a dataset using a logical condition
- Parse a messy character column into numeric
- Render to PDF and submit on Canvas

---

12 Next Steps

Next, you'll learn how to manipulate and clean **strings** using the `stringr` package.

13 Strings and Regular Expressions with stringr

13.1 Learning Objectives

By the end of this chapter, you should be able to:

- Manipulate strings using the `stringr` package
- Detect patterns with regular expressions (regex)
- Extract, replace, and split text
- Clean messy text data for analysis

13.2 Introduction to stringr

The `stringr` package provides consistent, simple functions for string operations.

Load the library:

```r
library(stringr)
```

13.3 Creating and Inspecting Strings

```r
fruit <- c("apple", "banana", "pear")
str_length(fruit)
```

```
[1] 5 6 4
```

```r
str_c(fruit, " is tasty")
```

```
[1] "apple is tasty"  "banana is tasty" "pear is tasty"
```

13.3.1 In-Class Exercise 1 – Basic String Operations

1. Create a vector of at least 5 words.
2. Measure their lengths with `str_length()`.
3. Concatenate them with the phrase `" is cool"`.

13.4 Detecting Patterns with Regex

`str_detect()` returns TRUE if a pattern is found.

```r
words <- c("dog", "cat", "parrot", "cow")
str_detect(words, "o")
```

```
[1]  TRUE FALSE  TRUE  TRUE
```

You can use **regular expressions** for more complex patterns.

Examples:

- `^a` – starts with "a"
- `ing$` – ends with "ing"
- `[0-9]+` – one or more digits

```r
animals <- c("ant", "bat", "cat", "dog")
str_detect(animals, "^a")
```

```
[1]  TRUE FALSE FALSE FALSE
```

13.4.1 In-Class Exercise 2 – Pattern Detection

1. Create a vector of email-like strings.
2. Use `str_detect()` to check which contain `"@"`.
3. Write a regex to detect strings ending in `.com`.

13.5 Extracting and Replacing Text

13.5.1 `str_extract()`

Extracts the first match:

```r
str_extract(c("abc123", "xyz789"), "[0-9]+")
```

```
[1] "123" "789"
```

13.5.2 `str_replace()`

Replaces matching patterns:

```r
str_replace("apple pie", "apple", "peach")
```

```
[1] "peach pie"
```

13.5.3 In-Class Exercise 3 – Extraction and Replacement

1. Extract digits from a vector of alphanumeric strings.
2. Replace the word `"dog"` with `"puppy"` in a text vector.

13.6 Splitting and Cleaning Text

13.6.1 str_split()

Splits text into pieces:

```
str_split("a,b,c", ",")
```

```
[[1]]
[1] "a" "b" "c"
```

13.6.2 Cleaning with regex

You can remove unwanted characters:

```
dirty <- c(" price:$100 ", " cost:$200 ")
str_replace_all(dirty, "[$ ]", "")
```

```
[1] "price:100" "cost:200"
```

13.6.3 In-Class Challenge – Text Cleaning

1. Create a vector of messy product names with extra spaces and symbols.
2. Use `str_replace_all()` and `str_trim()` to clean them.
3. Extract numeric prices from the strings.

13.7 Homework Preview

For the next homework, you will:

- Work with a text dataset (e.g., movie titles, email logs, or messy product names)
- Use at least three `stringr` functions to clean or extract information
- Write one regex pattern to detect a specific feature in the data
- Render a short report (with code and results) to PDF and submit

---

13.8 Next Steps

Next, we will learn to **work with factors and categorical data** using the `forcats` package.

14 Factors and Categorical Data with forcats

14.1 Learning Objectives

By the end of this chapter, you should be able to:

- Understand what factors are and why they are used
- Reorder factor levels to improve plots
- Rename factor levels
- Collapse multiple levels into broader categories
- Use `forcats` functions to manipulate categorical variables effectively

---

14.2 Introduction to Factors

Factors are used to work with **categorical data** (variables with a fixed set of possible values). R uses factors to control ordering in plots and summaries.

Example:

```r
x <- factor(c("low", "medium", "high", "medium", "low"))
levels(x)
```

```
[1] "high"   "low"    "medium"
```

14.3 Using forcats

The `forcats` package provides helper functions for factors.

```r
library(forcats)
library(tidyverse)
```

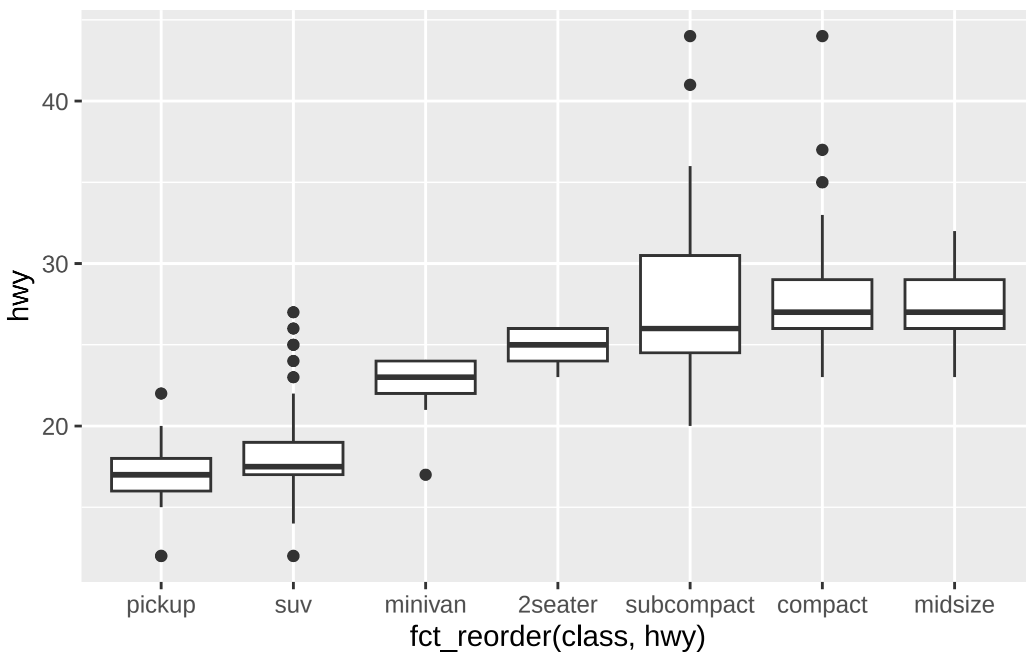
```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v ggplot2   3.5.2     v stringr   1.5.1
v lubridate 1.9.4     v tibble    3.2.1
v purrr     1.0.4     v tidyr     1.3.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

14.4 Reordering Factor Levels

14.4.1 fct_reorder()

Reorders factor levels by another variable (e.g., mean of a numeric variable):

```r
ggplot(mpg, aes(x = fct_reorder(class, hwy), y = hwy)) +
  geom_boxplot()
```



14.4.2 In-Class Exercise 1 – Reordering

1. Use `fct_reorder()` to reorder car classes in the `mpg` dataset by highway mpg.

2. Make a boxplot of `hwy` by class.

3. Which class has the highest median mpg?

14.5 Changing Factor Labels

14.5.1 `fct_recode()`

Renames levels:

```r
mpg |>
  mutate(drv = fct_recode(drv,
    "front-wheel" = "f",
    "rear-wheel"  = "r",
    "4-wheel"     = "4"
  )) |>
  count(drv)
```

```
# A tibble: 3 x 2
  drv             n
  <fct>       <int>
1 4-wheel       103
2 front-wheel   106
3 rear-wheel     25
```

14.5.2 In-Class Exercise 2 – Recoding

1. Recode the `drv` variable to use descriptive names.

2. Count the number of cars in each drive category.

14.6 Collapsing Levels

14.6.1 `fct_collapse()`

Combines multiple levels into broader categories.

```r
mpg |>
  mutate(class_grouped = fct_collapse(class,
    small = c("2seater", "compact", "subcompact"),
    large = c("suv", "pickup", "minivan")
  )) |>
  count(class_grouped)
```

```
# A tibble: 3 x 2
  class_grouped     n
  <fct>         <int>
1 small            87
2 midsize          41
3 large           106
```

14.6.2 In-Class Exercise 3 – Collapsing Levels

1. Create a new variable that collapses `class` into `small` vs. `large`.

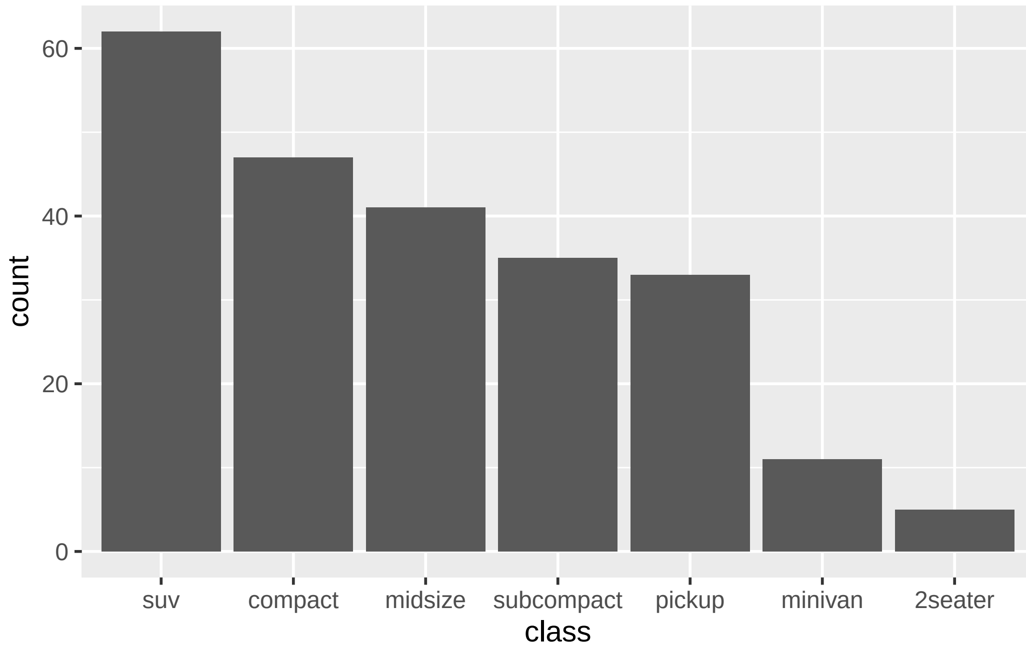
2. Make a bar chart of the collapsed variable.

14.7 Reordering Factors for Plots

14.7.1 `fct_infreq()`

Orders factors by frequency:

```
mpg |>
  mutate(class = fct_infreq(class)) |>
  ggplot(aes(x = class)) +
  geom_bar()
```



14.7.2 In-Class Challenge – Factor Workflow

Using the `mpg` dataset:

- Reorder the `manufacturer` variable by number of cars
- Collapse classes into fewer categories
- Create a bar plot that uses the new ordering and grouping

14.8 Homework Preview

For the next homework, you will:

- Choose a dataset with at least one categorical variable
- Use `forcats` functions to:
  - Reorder levels
  - Recode labels

    - Collapse levels where appropriate

- Produce at least one visualization that uses your factor manipulations
- Render to PDF and submit on Canvas

14.9 Next Steps

Next, we will learn how to work with **relational data** using `dplyr` join functions to combine multiple datasets.

15 Relational Data with dplyr Joins

15.1 Learning Objectives

By the end of this chapter, you should be able to:

- Understand the concept of relational data and keys
- Combine multiple datasets using different join functions
- Use `left_join()`, `inner_join()`, `full_join()`, and `semi_join()`
- Diagnose and handle join problems (missing keys, duplicates)
- Apply joins in analysis workflows

15.2 What is Relational Data?

Relational data consists of **multiple tables** that can be linked by **keys**.

Example tables from `nycflights13`:

- `flights`: flight information
- `airlines`: airline names
- `airports`: airport locations
- `planes`: plane details
- `weather`: weather data

15.3 Keys

- **Primary key**: uniquely identifies each row in a table
- **Foreign key**: column that matches a primary key in another table

Example: `flights$carrier` matches `airlines$carrier`.

15.4 Joins with dplyr

`dplyr` join functions merge tables by keys.

15.4.1 `left_join()`

Keeps all rows from the first table:

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.2     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```r
library(nycflights13)

flights |>
  left_join(airlines, by = "carrier") |>
  rename(airline_name = name) |>
  select(airline_name, carrier, flight) |>
  head()
```

```
# A tibble: 6 x 3
  airline_name           carrier flight
  <chr>                  <chr>    <int>
1 United Air Lines Inc.  UA        1545
2 United Air Lines Inc.  UA        1714
3 American Airlines Inc. AA        1141
4 JetBlue Airways        B6         725
5 Delta Air Lines Inc.   DL         461
6 United Air Lines Inc.  UA        1696
```

15.4.2 `inner_join()`

Keeps only matching rows:

```r
flights |>
  inner_join(airlines, by = "carrier") |>
  rename(airline_name = name) |>
  select(airline_name, carrier, flight) |>
  head()
```

```
# A tibble: 6 x 3
  airline_name           carrier flight
  <chr>                  <chr>    <int>
1 United Air Lines Inc.  UA        1545
2 United Air Lines Inc.  UA        1714
3 American Airlines Inc. AA        1141
4 JetBlue Airways        B6         725
5 Delta Air Lines Inc.   DL         461
6 United Air Lines Inc.  UA        1696
```

15.4.3 `full_join()`

Keeps all rows from both tables:

```r
flights |>
  full_join(airlines, by = "carrier") |>
  rename(airline_name = name) |>
  select(airline_name, carrier, flight) |>
  head()
```

```
# A tibble: 6 x 3
  airline_name           carrier flight
  <chr>                  <chr>    <int>
1 United Air Lines Inc.  UA        1545
2 United Air Lines Inc.  UA        1714
3 American Airlines Inc. AA        1141
4 JetBlue Airways        B6         725
5 Delta Air Lines Inc.   DL         461
6 United Air Lines Inc.  UA        1696
```

15.4.4 `semi_join()` and `anti_join()`

- `semi_join()`: keeps rows in first table with matches in second
- `anti_join()`: keeps rows with no matches

```r
flights |>
  semi_join(airlines, by = "carrier") |>
  head()
```

```
# A tibble: 6 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
1  2013     1     1      517            515         2      830            819
2  2013     1     1      533            529         4      850            830
3  2013     1     1      542            540         2      923            850
4  2013     1     1      544            545        -1     1004           1022
5  2013     1     1      554            600        -6      812            837
6  2013     1     1      554            558        -4      740            728
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

15.4.5 In-Class Exercise 1 – Basic Joins

1. Use `left_join()` to add airline names to `flights` (rename to `airline_name`).
2. Count the number of flights for each airline.
3. Use `inner_join()` and compare the number of rows.

15.5 Joining Multiple Tables

You can chain joins to combine several datasets:

```
flights |>
  left_join(airlines, by = "carrier") |>
  rename(airline_name = name) |>
  left_join(airports, by = c("dest" = "faa")) |>
  select(airline_name, dest, arr_delay) |>
  head()
```

```
# A tibble: 6 x 3
  airline_name           dest  arr_delay
  <chr>                  <chr>     <dbl>
1 United Air Lines Inc.  IAH          11
2 United Air Lines Inc.  IAH          20
3 American Airlines Inc. MIA          33
4 JetBlue Airways        BQN         -18
5 Delta Air Lines Inc.   ATL         -25
6 United Air Lines Inc.  ORD          12
```

15.5.1 In-Class Exercise 2 – Multi-Table Joins

1. Join `flights` with `airports` to add destination airport names.
2. Summarize average arrival delay by airport.

3. Which airport has the longest average delay?

15.6 Handling Join Problems

- Missing keys → results in `NA` values
- Duplicated keys → may create duplicate rows
- Always check results with `count()` or `distinct()`

Example:

```
flights |>
  left_join(airlines, by = "carrier") |>
  rename(airline_name = name) |>
  count(carrier, airline_name)
```

```
# A tibble: 16 x 3
   carrier airline_name                    n
   <chr>   <chr>                       <int>
 1 9E      Endeavor Air Inc.           18460
 2 AA      American Airlines Inc.      32729
 3 AS      Alaska Airlines Inc.          714
 4 B6      JetBlue Airways             54635
 5 DL      Delta Air Lines Inc.        48110
 6 EV      ExpressJet Airlines Inc.    54173
 7 F9      Frontier Airlines Inc.        685
 8 FL      AirTran Airways Corporation  3260
 9 HA      Hawaiian Airlines Inc.        342
10 MQ      Envoy Air                   26397
11 OO      SkyWest Airlines Inc.          32
12 UA      United Air Lines Inc.       58665
13 US      US Airways Inc.             20536
14 VX      Virgin America               5162
15 WN      Southwest Airlines Co.      12275
16 YV      Mesa Airlines Inc.            601
```

15.7 In-Class Challenge – Join Workflow

- Join flights with airlines and airports

- Calculate average arrival delay by airline and destination

- Arrange by delay and identify the worst-performing routes

15.8 Homework Preview

For the next homework, you will:

- Combine at least two datasets using joins
- Use at least two different join types (`left_join()`, `inner_join()`, etc.)
- Handle missing data or duplicates appropriately
- Produce a summary table and one visualization based on the joined data
- Render to PDF and submit on Canvas

15.9 Next Steps

Next, we will introduce **accessing data** using spreadsheets, SQL databases, JSON, and web scraping.

16 Accessing Data: Spreadsheets, Databases, Arrow, JSON, and Web Scraping

16.1 Learning Objectives

By the end of this lecture, you should be able to:

- Import and work with data from Excel and Google Sheets
- Connect to and query relational databases from R
- Use Arrow to work efficiently with parquet files and large datasets
- Access and tidy hierarchical JSON data
- Perform basic web scraping to extract data from web pages

16.2 Spreadsheets (R4DS Chapter 20)

R can read Excel files with the `readxl` package and Google Sheets with `googlesheets4`.

16.2.1 Importing Excel

```
library(readxl)

excel_df <- read_excel("data/example.xlsx", sheet = "Sheet1")
head(excel_df)
```

16.2.2 Importing Google Sheets

```r
library(googlesheets4)
sheet_url <- "https://docs.google.com/spreadsheets/d/your-sheet-id/edit#gid=0"
gs_df <- read_sheet(sheet_url)
```

16.2.3 In-Class Exercise 1 – Spreadsheets

1. Read an Excel file from the course data folder.
2. Load a Google Sheet you create (optional, requires authentication).
3. Summarize one numeric column.

16.3 Databases (R4DS Chapter 21)

Use `DBI` and `RSQLite` to interact with relational databases. You can also use `dplyr` verbs to query tables.

16.3.1 Example: Connecting to SQLite

```r
library(DBI)
con <- dbConnect(RSQLite::SQLite(), "data/mydb.sqlite")

# List tables
dbListTables(con)

# Read a table into R
flights_db <- dbReadTable(con, "flights")

# Or use dplyr to query lazily
library(dplyr)
tbl(con, "flights") |> filter(dep_delay > 60) |> collect() |> head()
```

16.3.2 In-Class Exercise 2 – Databases

1. Connect to the provided SQLite database.
2. List tables with `dbListTables()`.
3. Query the flights table for flights delayed more than 2 hours.

16.4 Arrow (R4DS Chapter 22)

Arrow allows you to read parquet files efficiently without loading everything into memory.

16.4.1 Example: Reading Parquet

```
library(arrow)

dataset <- open_dataset("data/large.parquet")
dataset |> filter(column_x > 10) |> collect() |> head()
```

16.4.2 In-Class Exercise 3 – Arrow

1. Open a parquet dataset using `arrow::open_dataset()`.
2. Run a filter and select query.
3. Compare performance to reading the equivalent CSV.

16.5 Hierarchical Data (R4DS Chapter 23)

Hierarchical data (JSON) often contains nested lists. Use `jsonlite` to load JSON and `tidyr::unnest_wider()` to flatten it.

16.5.1 Example: Reading JSON

```
library(jsonlite)

json_data <- fromJSON("data/example.json")
str(json_data)
```

16.5.2 Flattening Nested Data

```
library(tidyr)
nested_df <- tibble(
  id = 1,
  details = list(tibble(city = "NYC", temp = 75))
)

nested_df |> unnest_wider(details)
```

```
# A tibble: 1 x 3
     id city   temp
  <dbl> <chr> <dbl>
1     1 NYC      75
```

16.5.3 In-Class Exercise 4 – JSON Rectangling

1. Load a nested JSON file.
2. Use `unnest_wider()` or `unnest_longer()` to flatten it.
3. Create a tidy table with one row per observation.

16.6 Web Scraping (R4DS Chapter 24)

Web scraping extracts data from websites. Use `rvest` to read HTML and extract tables or nodes.

16.6.1 Example: Scraping a Table

```r
library(rvest)

url <- "https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)"
page <- read_html(url)

gdp_table <- page |> html_element("table") |> html_table()
head(gdp_table)
```

```
# A tibble: 2 x 1
  X1
  <chr>
1 ""
2 "Largest economies in the world by GDP (nominal) in 2025according to Internat~
```

16.6.2 In-Class Exercise 5 – Web Scraping

1. Use `rvest` to scrape a simple table from Wikipedia.
2. Convert it to a tibble and clean column names.
3. Create a plot of GDP vs. rank.

16.7 In-Class Challenge – Multiple Data Sources

1. Import an Excel dataset, a JSON dataset, and scrape a table from the web.

2. Clean and join at least two sources.

3. Create one visualization combining information.

16.8 Homework Preview

For the next homework:

- Choose **two different data sources** (Excel, database, parquet, JSON, web)
- Import and tidy them
- Join or compare across sources
- Render a short report with one plot and one table
- Submit the rendered PDF

16.9 Conclusion

This session completes the course by showing how to **access data from multiple modern sources**, preparing you to work with **real-world messy data** beyond flat CSV files.

A CS506: Data Wrangling and Management–Syllabus

A.1 Course Overview

INF506: Data Wrangling and Management introduces graduate students to data wrangling and management using **R** and the **Tidyverse** ecosystem. Students will learn to import, manipulate, clean, and visualize data with a strong emphasis on practical applications and reproducible workflows.

- CS 506, Fall 2025, 3 units
- Section 001: TuTh 9:35AM-10:50AM, Learning Resource Ctr Rm 106C
- Prerequisite: Graduate status
- Mode of Instruction: Face-to-face (in person)
- Instructor's Name & Contact:
  - Marc Tollis (marc.tollis@nau.edu)
    - Room 209, SICCS (Building 90, second floor)
    - Office Hours: Tue 11AM-12PM
    - 928-523-3406

A.2 Canvas & Recorded Lectures

We will use the learning management system, Canvas, to conduct some course business, including assignment disbursement and submitting. I will use Canvas to record lectures for future viewing.

A.3 CS506 Book Website

I have compiled a course website that has supplemental text and coded examples that we will walk through in class. This website essentially serves as the course textbook and is required reading. There will be other required reading material.

A.4 Course Objectives

By the end of the course, students will be able to:

- Use R and RStudio for data analysis
- Import structured and unstructured data
- Clean and transform data using `dplyr`, `tidyr`, and other Tidyverse packages
- Create effective visualizations using `ggplot2`
- Perform exploratory data analysis (EDA)
- Apply data wrangling techniques to real datasets

A.4.1 Course Student Learning Outcomes

LO1. Compare and contrast major classes of and techniques for data handling (synthesis).

Students will be able to:
1. Identify various sources of data
2. Identify and utilize tool chains appropriate for accessing data

LO2. Design and enact data manipulation, analysis, and visualization workflows for large, heterogenous datasets (application).

Students will be able to:
1. Aggregate data from multiple sources
2. Reshape data for further analysis
3. Validate data
4. Generate meaningful statistics summarizing the data
5. Visualize trends in data

LO3. Reason about advantages, preferred use cases, and weaknesses of various data manipulation techniques (application)

LO4. Develop a conceptual understanding of how the field of data management is evolving (knowledge).

Students will be able to:
1. Find and employ data management tools in **R**
2. Find and employ data visualization tools in **R**

A.4.2 Program Student Outcomes supported by this class

This course directly supports the following program student outcomes in the Masters of Science in Computational and Applied Data Science program assessment and improvement plan:

SO2. Build the practical skills to explore, analyze, manage, and visualize large data sets using the latest technologies.

SO3. Evaluate and use well accepted methods to obtain, clean, pre-process, and transform data for further processing.

SO4. Apply data science and cutting-edge analytical methods to address data-rich problems from a variety of fields, think critically about data, and drive decision making.

SO7. Identify, appraise, and investigate ethical issues surrounding data collection, use, and data-driven decision making and to act in an informed and conscientious ethical manner.

A.5 Required Materials

- **Textbook:** *R for Data Science* (free online)
- **Software:**
  - R
  - RStudio

A.6 Assessments

Component	Weight
Problem Sets (14 total)	35%
Quizzes (6 total, lowest dropped)	50%
Workshops (2)	10%
Attendance	5%

- Grades will be assigned using the weighted sum described above using this scale: **A** 90%, **B** 80%, **C** 70%, **D** 60%, **F** < 60%.

A.7 Grading and Submission

- **Problem Sets** are simple assignments that will be completed on your own and submitted via Canvas.
- **Problem sets** are marked as **complete** or **incomplete**.
- **Quizzes** are written and completed in-class.
- **The final quiz** is a case study project starting in class and due during finals week.
- **Workshops** will take up class time and attendance is required for the workshop grade.
- **Workshop** assignments will be submitted via Canvas.
- All **Canvas-based assignments are due Sunday 11:59PM** the week they are assigned.

A.8 Course Schedule (Fall 2025)

Week	Dates (T/Th)	R4DS Chapters	Topics	Assignments	Quiz
1	Aug 26 / 28	Ch. 1	**Intro to R, RStudio, and Quarto**: Projects, rendering `.qmd` to `.pdf`	PS1	
2	Sept 2 / 4	Ch. 2 – Data Visualization	**Data Visualization with ggplot2**: Aesthetics, geoms, facets	PS2	

Week	Dates (T/Th)	R4DS Chapters	Topics	Assignments	Quiz
3	Sept 9 / 11	Ch. 3 – Data Transformation	**Data Transformation (Rows)**: `filter()`, `arrange()`	PS3	Quiz 1
4	Sept 16 / 18	Ch. 3 – Data Transformation	**Data Transformation (Columns + Pipes)**: `select()`, `mutate()`, `|>`	PS4	
5	Sept 23 / 25	Ch. 3 – Data Transformation	**Grouping & Summarization**: `group_by()`, `summarize()`	PS5	Quiz 2
6	Sept 30 / Oct 2	Ch. 5 – Tidy Data	**Tidy Data**	PS6	
7	Oct 7 / 9	Ch. 6 – Workflow: Scripts	**Workflow & Reproducibility**: projects, scripts, Quarto best practices	PS7; Mini Hackathon	Quiz 3
8	Oct 14 / 16	Ch. 7 - Data Import	**Data Import**: `readr`, parsing dates, Excel	PS8	
9	Oct 21 / 23	Ch. 10 – Exploratory Data Analysis	**EDA: distributions, patterns, relationships**	PS9	Quiz 4
10	Oct 28 / 30	Ch. 12 through 18 – Transform	**Logical Vectors and Numbers**; **Strings & Regular Expressions**: `stringr`	PS10	
11	Nov 4 / 6	Ch. 12 through 18 – Transform (continued)	**Factors & Categorical Data**: `forcats`	PS11	Quiz 5

Week	Dates (T/Th)	R4DS Chapters	Topics	Assignments	Quiz
12	Nov 11* / 13	Ch. 19 – Joins	**Relational Data**: joining tables (`left_join`, etc.)	PS12	
13	Nov 18 / 20	Ch. 20-24 – Advanced Importing	**Advanced Importing**, databases, web scraping	PS13; Code Review Workshop	
14	Nov 25 / 27	—	**Nov 25:** Catch-Up, Q&A, In-Class Coding Practice **Nov 27:** Thanksgiving – No Class	—	—
15	Dec 2 / 4	—	Course Wrap-up & Final Quiz	PS14	Quiz 6

* **Nov 11 (Veterans Day)** – no class that Tuesday.

A.9 Resources

- RStudio Cheatsheets
- DataCamp & Coursera tutorials for extra practice
- Office hours for additional help

A.10 Policies

A.10.1 Course Policies

Students are encouraged to attend the office hours of the instructor. If a student cannot attend regular office hours with the instructor, an appointment may be considered if made via email with sufficient advanced notice.

- Emails addressed to the instructor must be **respectful and professional**. The instructor will respond to emails promptly, within 2 business days. The instructor will generally not respond to emails on weekends or after working hours (i.e., in the evenings), so please plan accordingly.

- Cheating, including plagiarism of writing or computer code, will not be tolerated. All academic integrity violations are treated seriously. Academic integrity violations will result in penalties including, but not limited to, a zero on the assignment, a failing grade in the class, or expulsion from NAU. The University's Academic Integrity policies will be strictly enforced.
- Each student is required to demonstrate respect towards their peers and the instructor. The instructor is held to the same standard. - The instructor will not provide copies of course notes. These materials should be sought from the students' peers or by watching the recorded lectures.
- Electronic device usage must support learning in the class. All cell phones, PDAs, music players and other entertainment devices must be turned off (or put on silent) during lecture.
- Grades will be entered in Canvas and . Please check LOUIE for your final grade.
- **Attendance:** Active participation in coding activities is expected. Repeated, unexcused absences may affect the student's grade.
- **Late Work:** Accepted only with prior arrangement.
- **Academic Integrity:** Students must adhere to NAU's academic integrity policy.

A.10.2 University Policies

- Please see this document for all of the required Syllabus Policy Statements that equally apply to this course.

  This syllabus is subject to minor adjustments. Updates will be announced in class and posted on Canvas.

B Appendix: Coding Style Guidelines

B.1 Why Style Matters

Consistent code style makes your work:

- **Easier to read** (for you and collaborators)
- **Easier to debug** (clean structure reveals problems quickly)
- **Easier to maintain** (future you will thank present you)

This appendix summarizes the **tidyverse style guide** based on R4DS Workflow: Style.

B.2 File Naming

- Use **lowercase**, **descriptive names**, and **hyphens** (not spaces).
- Good: `data-cleaning.R`, `plot-analysis.R`
- Bad: `Data Cleaning.R`, `final.R`

B.3 Object Naming

- Use **snake_case** for variable and function names.
- Be descriptive, not cryptic.

```r
# Good
daily_sales <- 100
calculate_mean <- function(x) mean(x)

# Bad
ds <- 100
cm <- function(x) mean(x)
```

B.4 Spaces and Indentation

- Use **two spaces** for indentation.
- Always put a space **after commas** and **around operators**.

```
# Good
y <- x + 1
filter(mpg, cyl == 4)

# Bad
y<-x+1
filter(mpg,cyl==4)
```

B.5 Long Lines

- Keep lines **under 80 characters**.
- Use line breaks for long function calls.

```
mpg |>
  filter(cyl == 4, hwy > 30) |>
  arrange(desc(hwy))
```

B.6 Function Formatting

- Use consistent curly brace placement.

```r
# Good
my_function <- function(x) {
  x + 1
}

# Bad
my_function <- function(x){
x+1}
```

B.7 Commenting Code

- Write **comments** to explain why, not what.
- Use `#` for inline comments.

```r
# Calculate average highway mpg for 4-cylinder cars
avg_hwy <- mpg |>
  filter(cyl == 4) |>
  summarize(mean_hwy = mean(hwy))
```

B.8 Piping

- Each step in a pipeline goes on a **new line**.
- Use the pipe `|>` to connect transformations.

```r
mpg |>
  filter(cyl == 4) |>
  group_by(manufacturer) |>
  summarize(mean_hwy = mean(hwy))
```

B.9 Tidyverse Style Summary

- Use `|>` for pipelines, **snake_case** for names
- Indent **two spaces** per level
- Avoid deeply nested code — break into steps
- Write **clear, short, and well-commented** code

B.10 In-Class Exercise

1. Take a messy R script (provided in class).

2. Reformat it to follow these style guidelines.

3. Compare before vs. after readability.

B.11 Conclusion

Good code style is not just aesthetic — it improves **reproducibility** and **collaboration**. Follow these conventions for all homework and projects in this course.

C Appendix: Tidyverse and Tibbles

C.1 Overview

The **Tidyverse** is a collection of R packages designed for **data science**. They share a common design philosophy and work seamlessly together.

Core packages include:
- `ggplot2`: data visualization
- `dplyr`: data manipulation
- `tidyr`: data tidying
- `readr`: data import
- `purrr`: functional programming
- `tibble`: modern data frames
- `stringr`: string manipulation
- `forcats`: working with factors

You load them all with:

```
library(tidyverse)
```

D 1. What Are Tibbles?

Tibbles are **modern replacements** for base R data frames.

D.0.1 Key Features:

- Don't convert strings to factors automatically
- Never change variable names
- Print in a cleaner, more readable way
- Show only the first 10 rows and as many columns as fit on screen

Example:

```r
library(tibble)

tb <- tibble(
  x = 1:5,
  y = x^2,
  z = c("a", "b", "c", "d", "e")
)

tb
```

```
# A tibble: 5 x 3
      x     y z
  <int> <dbl> <chr>
1     1     1 a
2     2     4 b
3     3     9 c
4     4    16 d
5     5    25 e
```

E 2. Differences from Data Frames

- Subsetting with `$` works the same, but `[[` is stricter
- Tibbles don't do **partial matching**
- Printing is **truncated** by default (no flooding the console)

```r
tb$y
```

```
[1]  1  4  9 16 25
```

```r
tb[["z"]]
```

```
[1] "a" "b" "c" "d" "e"
```

F 3. Creating Tibbles

You can create tibbles manually with `tibble()` or convert data frames with `as_tibble()`.

```
df <- data.frame(a = 1:3, b = letters[1:3])
tb2 <- as_tibble(df)
```

G 4. Working with Tibbles

Tibbles work seamlessly with all **dplyr** verbs:

```
tb3 <- tibble(
  x = 1:6,
  y = c("a", "a", "b", "b", "c", "c")
)

tb3 |>
  dplyr::group_by(y) |>
  dplyr::summarize(mean_x = mean(x))
```

```
# A tibble: 3 x 2
  y     mean_x
  <chr>  <dbl>
1 a        1.5
2 b        3.5
3 c        5.5
```

H 5. Best Practices with Tibbles

- Always use `tibble()` for clean, predictable data structures
- Avoid row names; instead, use an explicit column
- Use `glimpse()` for quick inspection
- Use `print(n = Inf)` to see all rows when needed

I 6. When to Convert Back to Data Frames

Some base R functions don't work with tibbles.
Use `as.data.frame()` if you need to revert:

```
df_back <- as.data.frame(tb)
```

I.1 In-Class Exercise

1. Create a tibble with three columns: name, age, and score.
2. Use `mutate()` to add a new column `grade` based on `score`.
3. Group by grade and calculate the average age.

J Conclusion

Tibbles are at the heart of the Tidyverse workflow, offering: - Clean printing - Safer subsetting - Compatibility with the pipe operator and dplyr verbs

Use them as your **default** data structure in this course.