Assignment 3

A Summary of Chapters 8–20

R for Data Science by Hadley Wickham and Garrett Grolemund

Enock Mwanzia

Date: 2025-12-14

University:	University of Nairobi
Reg No:	SDS6/48097/2025
Programme:	MSc. Data Science
Course:	SDS 6103 (Statistical Computing)
Year of Study:	Year 1 – 2025/2026

Chapter 8: Workflow – Getting Help

This chapter addresses a critical but often underestimated aspect of data science: *how to learn effectively and solve problems independently.* Rather than focusing on syntax or functions, this chapter emphasizes mindset, habits, and professional development.

The authors highlight that *most real-world data science time is spent debugging, researching, and refining*, not writing new code from scratch. Consequently, learning how to ask good questions and seek help efficiently becomes a core technical skill.

A central idea is the use of *targeted searching. This* significantly improves the relevance of results and leads users to authoritative sources such as official documentation, Stack Overflow, GitHub issues and community forums.

Additionally, the authors introduce the concept of a *reprex (reproducible example).* A reprex is a minimal, self-contained piece of code that demonstrates a problem clearly. Reprex enhances clarity of thought, reduces ambiguity, and increases the likelihood of receiving helpful answers from others.

Example

```r
library(reprex)

x <- c(3, 2, 7, 4)
mean(x)
```

```
[1] 4
```

Chapter 9: Layers

This chapter deepens understanding of ggplot2 by introducing the layered grammar of graphics, which underpins all ggplot visualizations. The chapter explains that every plot is constructed by stacking layers, each responsible for a specific aspect of the visualization.

The authors explain how *aesthetic mappings* link variables in a dataset to visual properties such as position, color, size, and shape. These mappings can be global (applied to all layers) or local (applied to specific geoms), giving the practitioner precise control over plot behavior.

The chapter then explores *geometric objects (geoms),* which define how data are represented visually (points, lines, bars, boxplots, etc.). This abstraction allows the same data to be viewed from multiple perspectives by changing the geom rather than the dataset itself. This is facilitated by:

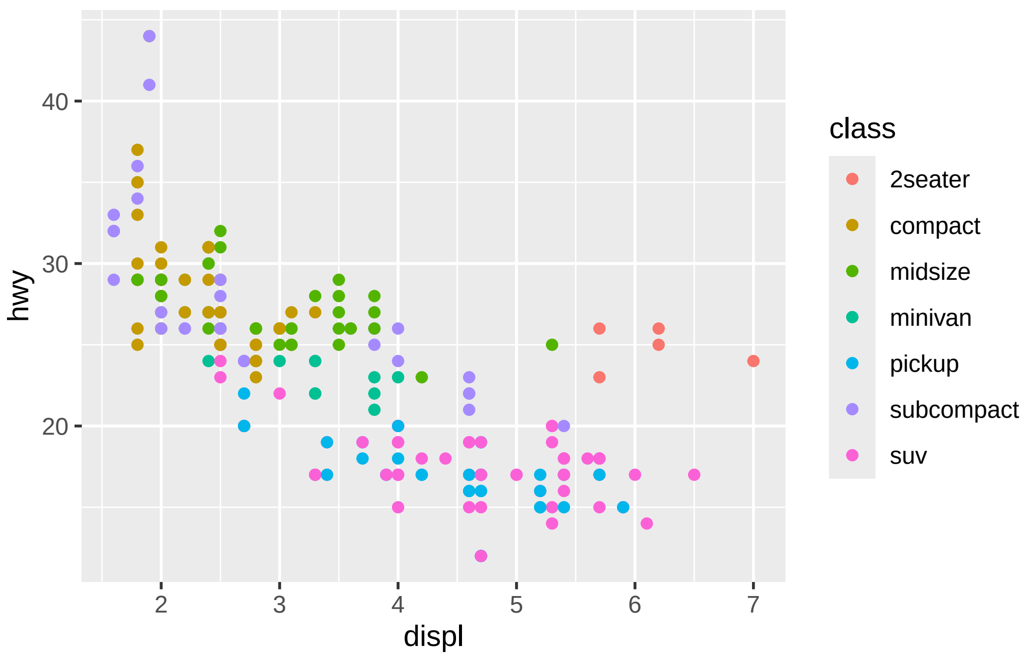
- **Facets**, which split data into subplots for comparison,
- **Statistical transformations**, which summarize or model data before plotting,
- **Positional adjustments,** which affects how the data is positioned.
- **Coordinate systems**, which affect how data are projected onto the plotting space.

Example 1

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.6
v forcats   1.0.1     v stringr   1.6.0
v ggplot2   4.0.1     v tibble    3.3.0
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.2.0
-- 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
```

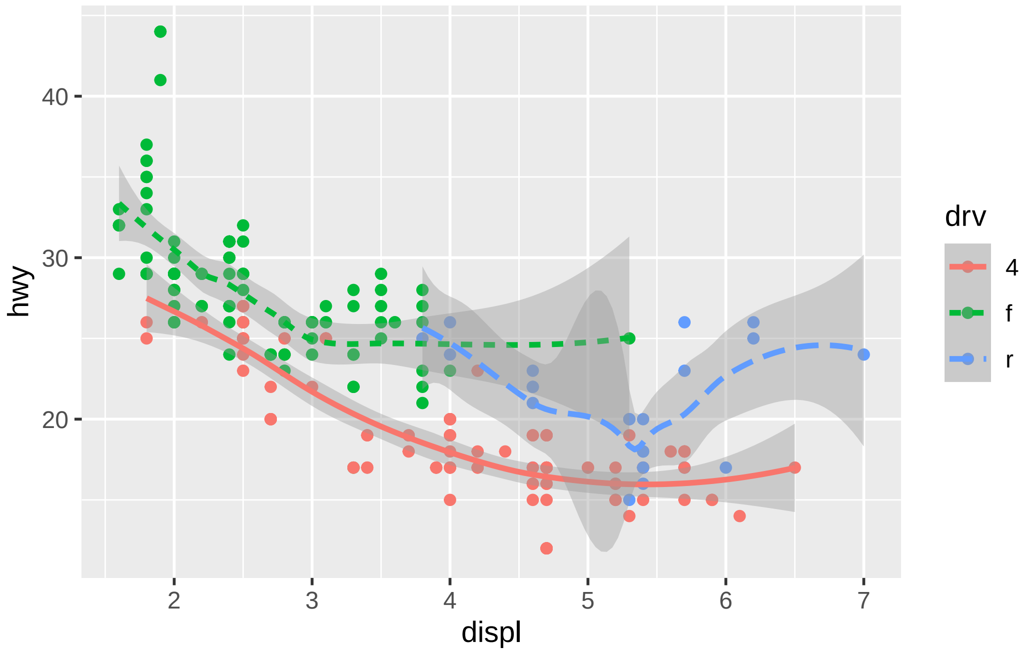
```r
ggplot(mpg, aes(x = displ, y = hwy, color = class)) +
  geom_point()
```



Example 2

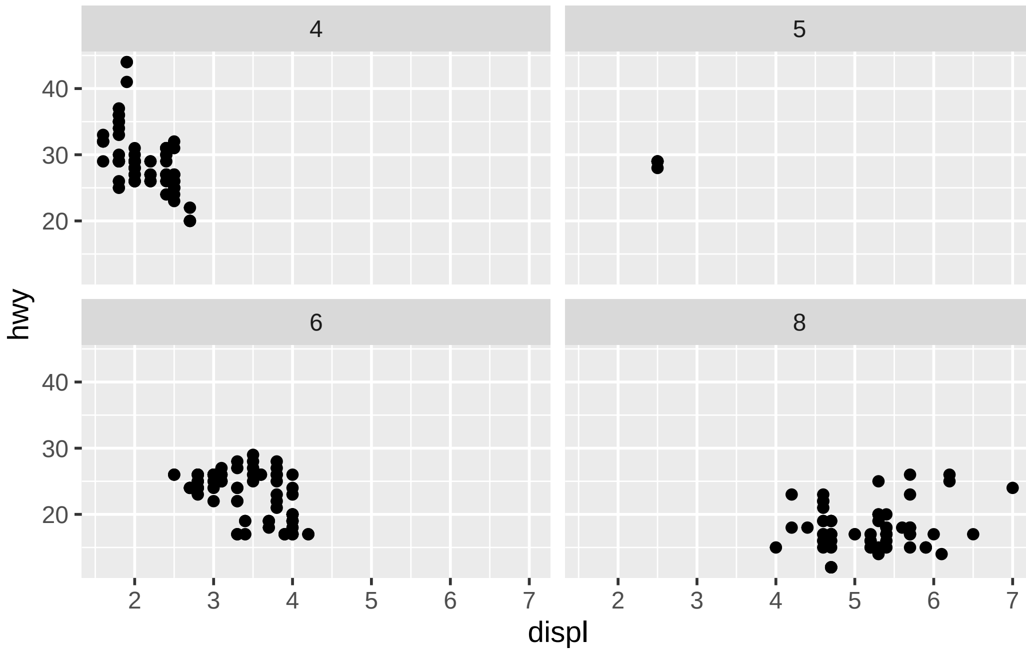
```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(aes(linetype = drv))
```

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



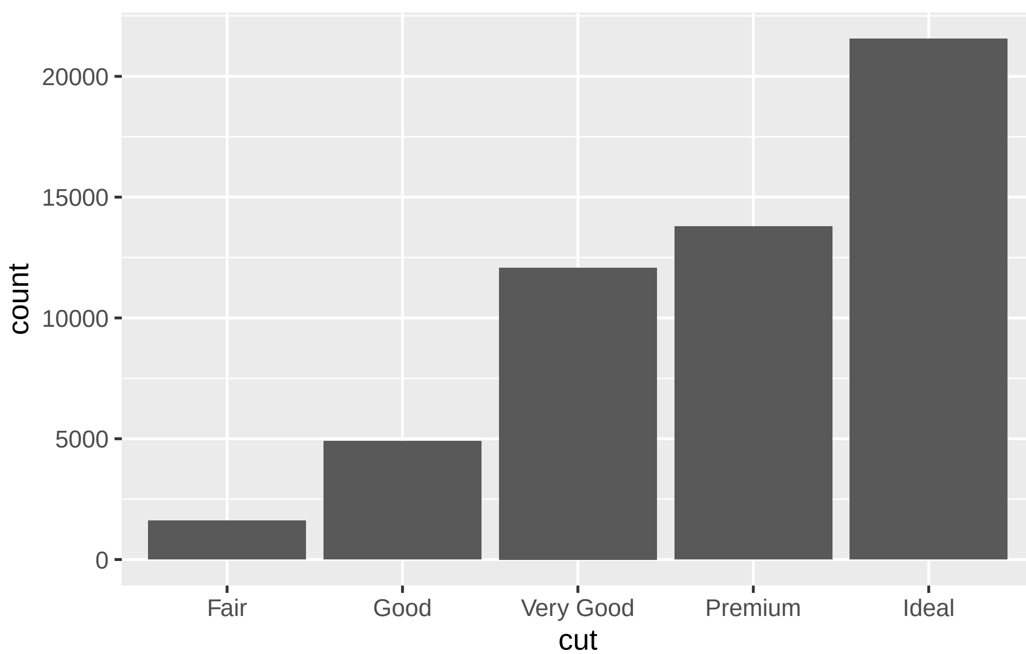
Example 3

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(~cyl)
```



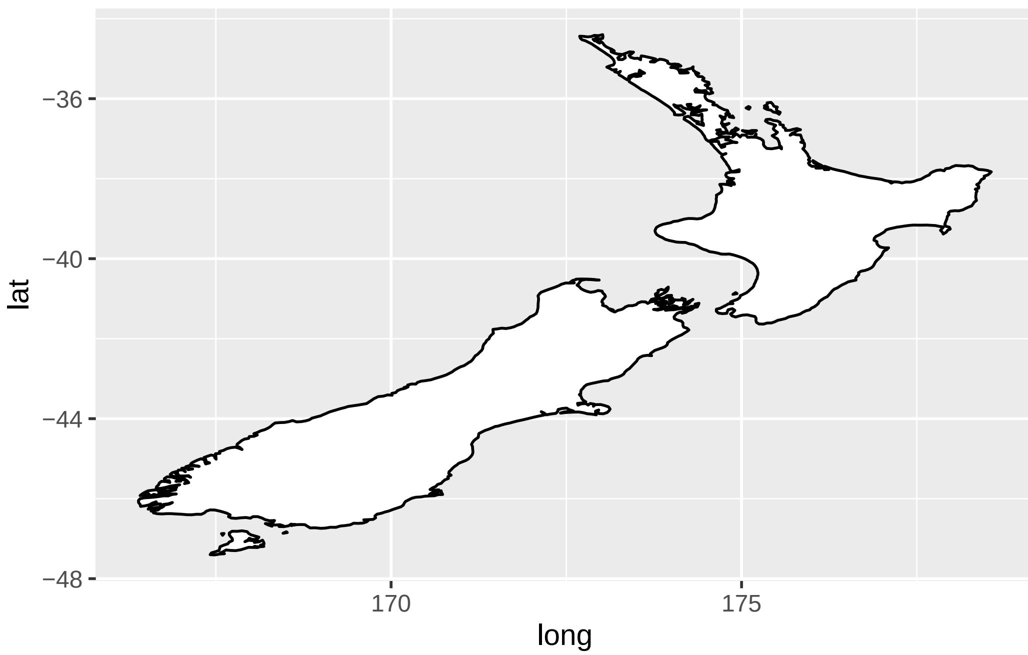
Example 4

```
ggplot(diamonds, aes(x = cut)) +
  geom_bar()
```

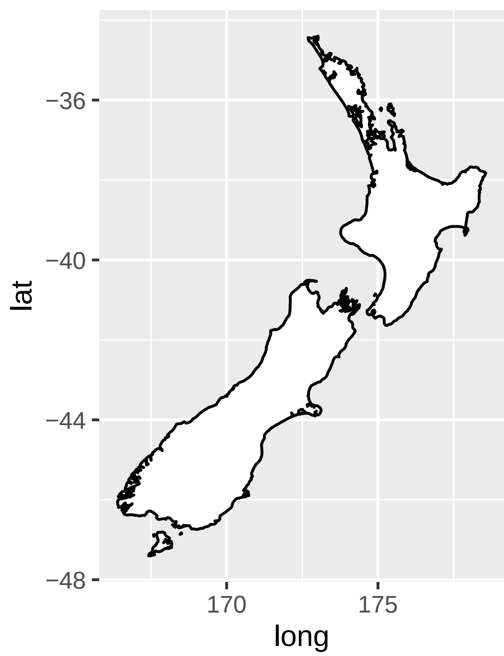


Example 5

```r
nz <- map_data("nz")
ggplot(nz, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "white", color = "black")
```



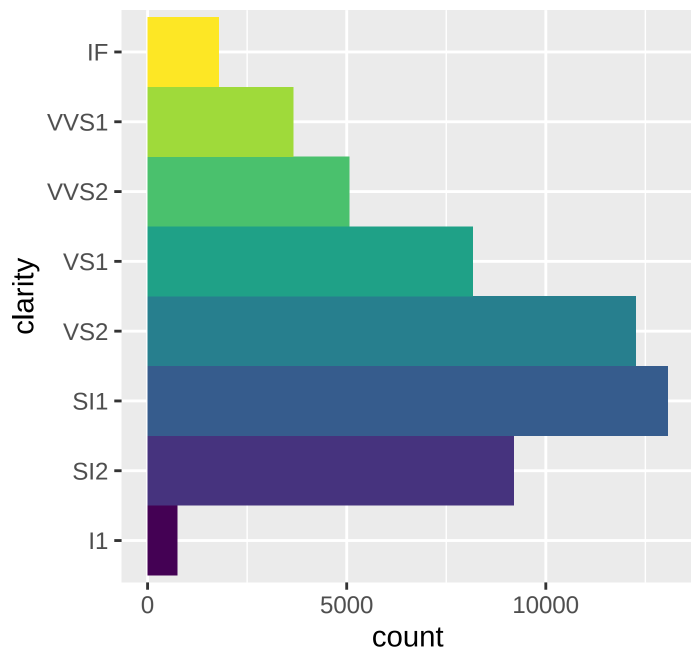

```r
ggplot(nz, aes(x = long, y = lat, group = group)) +
  geom_polygon(fill = "white", color = "black") +
  coord_quickmap()
```



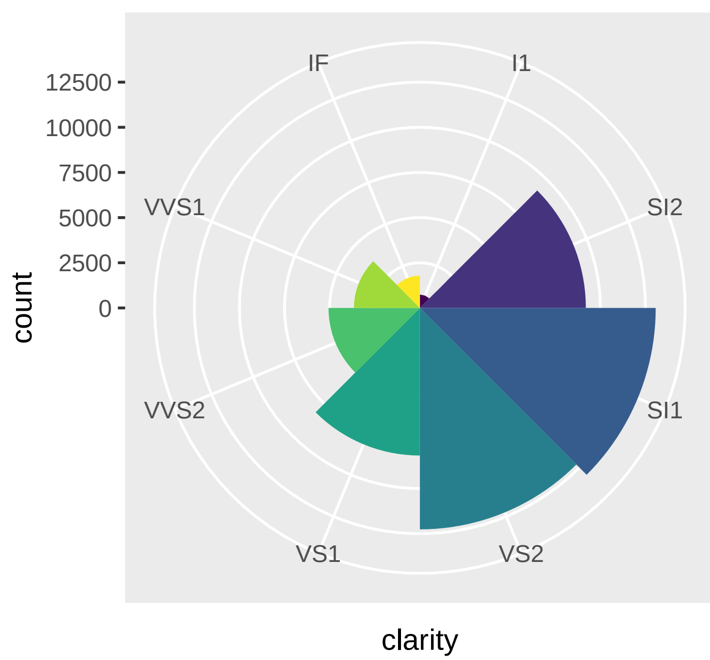
Example 6

```
bar <- ggplot(data = diamonds) +
  geom_bar(
    mapping = aes(x = clarity, fill = clarity),
    show.legend = FALSE,
    width = 1
  ) +
  theme(aspect.ratio = 1)

bar + coord_flip()
```




```
bar + coord_polar()
```



Chapter 10: Exploratory Data Analysis (EDA)

This chapter focuses on exploratory data analysis, positioning visualization as a thinking tool rather than a presentation device. The chapter emphasizes that EDA is an iterative, creative and question-driven process.

The authors structure EDA around three core questions:

1. How do variables vary?
2. Are there unusual values?
3. How do variables covary?

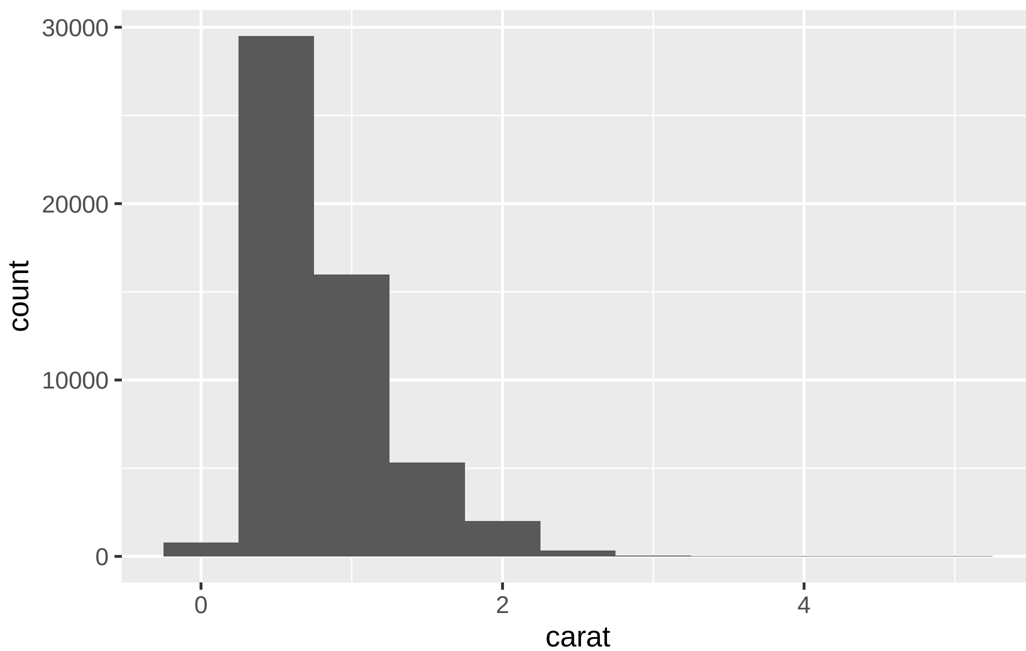
Through histograms, boxplots, scatterplots, and smooths, the chapter demonstrates how visualizations can uncover:

- Distributional shapes,
- Outliers and anomalies,
- Relationships between variables.

The chapter discourages premature modeling and instead promotes understanding the structure and quality of data first as reinforces the idea that good analysis begins with curiosity and skepticism, not assumptions.

Example 1

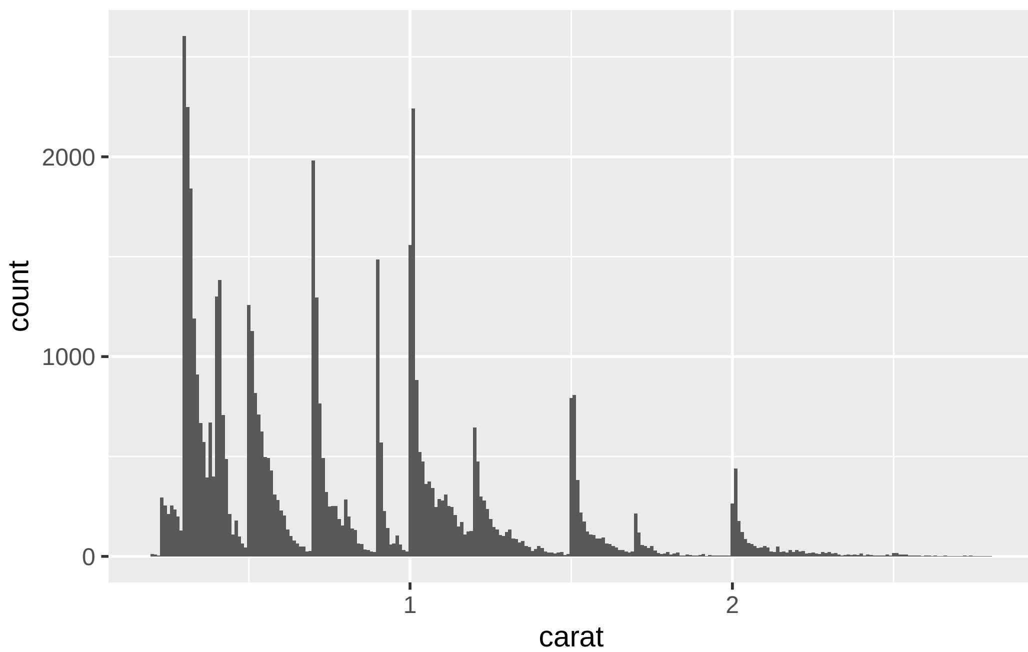
```
ggplot(diamonds, aes(x = carat)) +
  geom_histogram(binwidth = 0.5)
```



Example 2

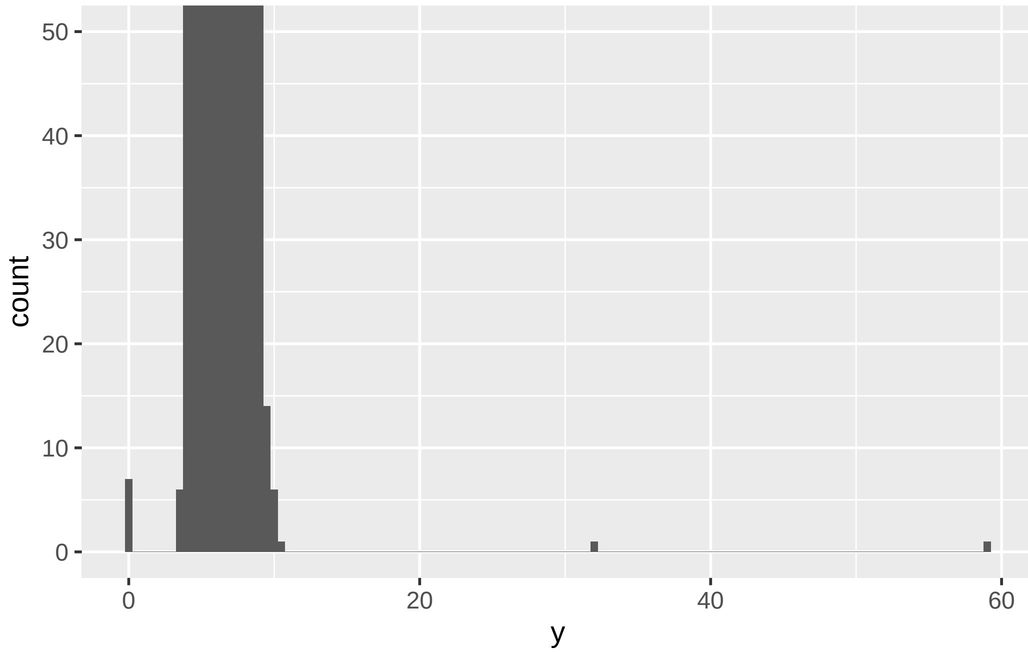
```
smaller <- diamonds |>
  filter(carat < 3)

ggplot(smaller, aes(x = carat)) +
  geom_histogram(binwidth = 0.01)
```



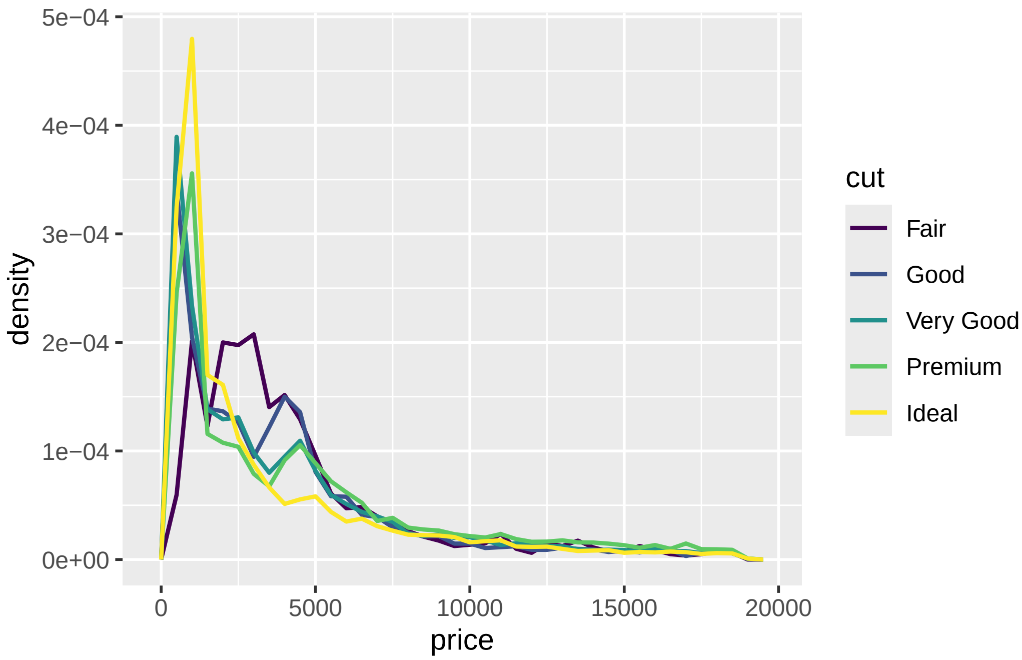
Example 3

```
ggplot(diamonds, aes(x = y)) +
  geom_histogram(binwidth = 0.5) +
  coord_cartesian(ylim = c(0, 50))
```



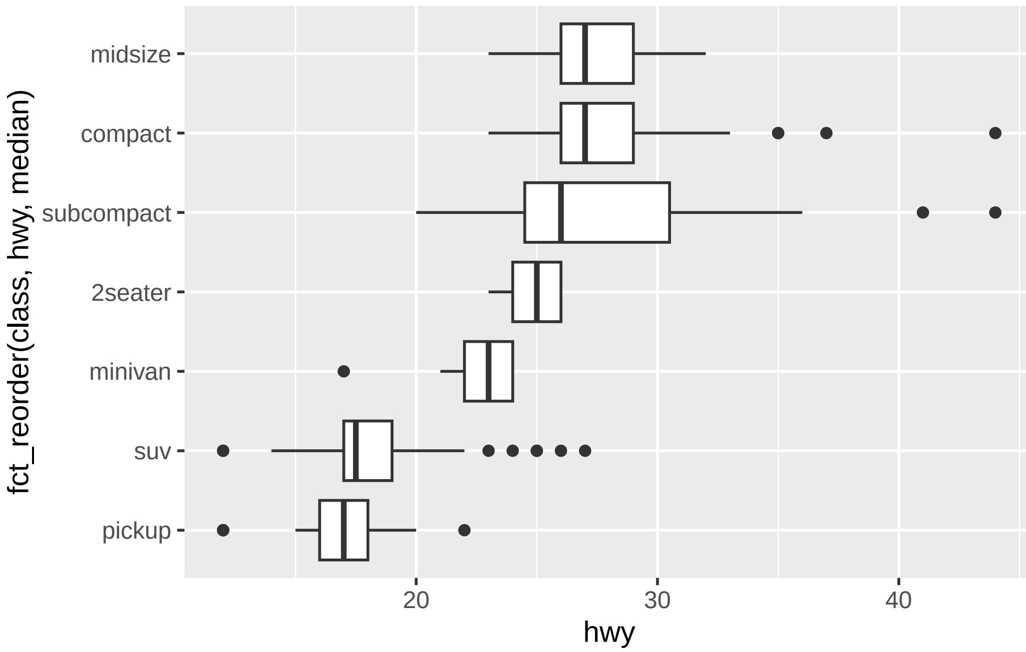
Example 4

```
ggplot(diamonds, aes(x = price, y = after_stat(density))) +
  geom_freqpoly(aes(color = cut), binwidth = 500, linewidth = 0.75)
```



Example 5

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



Chapter 11: Communication

This chapter explores effective communication by making insights understandable to others. The authors explain that communication-oriented graphics require different design choices than exploratory plots, placing more emphasis on:

- Clear and informative labels,
- Meaningful titles and subtitles,
- Thoughtful scale choices,
- Appropriate themes and layouts.

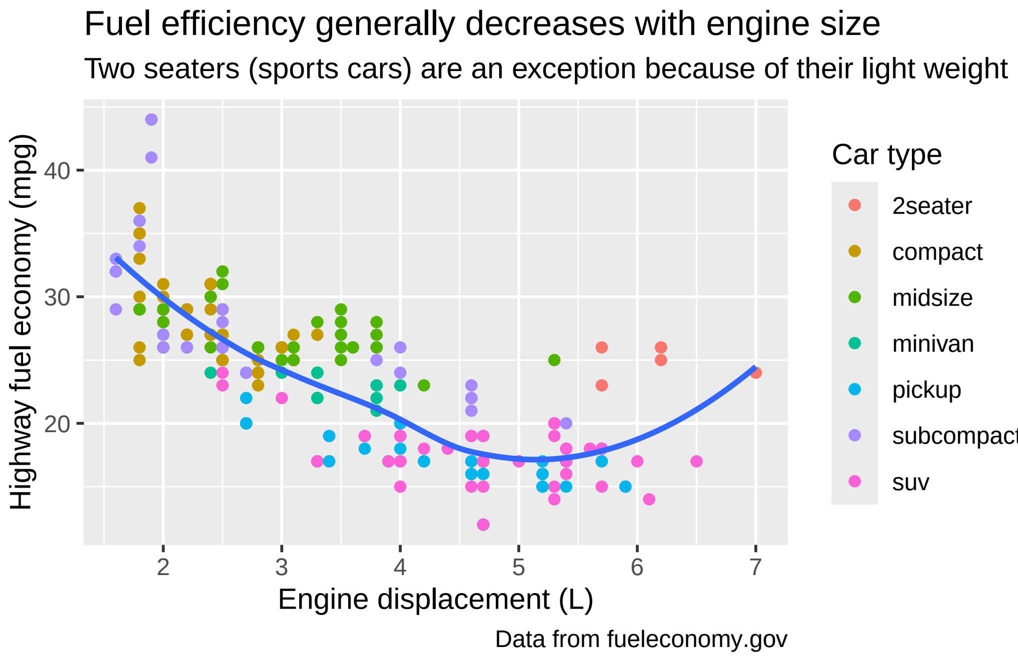
Annotations are introduced as tools for guiding interpretation, highlighting key patterns, or explaining anomalies. The chapter also discusses how consistent visual styling improves credibility and professionalism.

Ultimately, the chapter reinforces that analysis has little value if it cannot be communicated clearly, making visualization a core storytelling device in data science.

Example 1

```r
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    x = "Engine displacement (L)",
    y = "Highway fuel economy (mpg)",
    color = "Car type",
    title = "Fuel efficiency generally decreases with engine size",
    subtitle = "Two seaters (sports cars) are an exception because of their light weight",
    caption = "Data from fueleconomy.gov"
  )
```

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



Example 2

```
label_info <- mpg |>
  group_by(drv) |>
  arrange(desc(displ)) |>
  slice_head(n = 1) |>
  mutate(
    drive_type = case_when(
      drv == "f" ~ "front-wheel drive",
      drv == "r" ~ "rear-wheel drive",
      drv == "4" ~ "4-wheel drive"
    )
  ) |>
  select(displ, hwy, drv, drive_type)

label_info
```

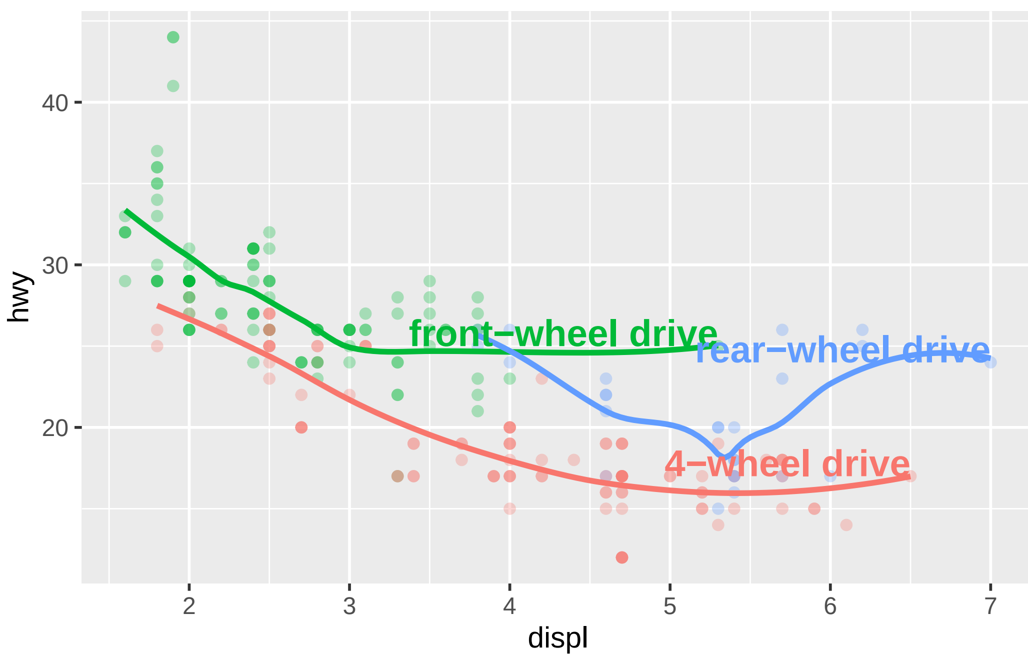
```
# A tibble: 3 x 4
# Groups:   drv [3]
  displ   hwy drv   drive_type
  <dbl> <int> <chr> <chr>
1   6.5    17 4     4-wheel drive
2   5.3    25 f     front-wheel drive
```

```
3   7        24 r      rear-wheel drive
```

Example 3

```
ggplot(mpg, aes(x = displ, y = hwy, color = drv)) +
  geom_point(alpha = 0.3) +
  geom_smooth(se = FALSE) +
  geom_text(
    data = label_info,
    aes(x = displ, y = hwy, label = drive_type),
    fontface = "bold", size = 5, hjust = "right", vjust = "bottom"
  ) +
  theme(legend.position = "none")
```

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



Chapter 12: Logical Vectors

This chapter focuses on logical vectors, which are fundamental to filtering, subsetting, and conditional logic in R. They are the simplest type of vectors because each element can only be of three possible values: TRUE, FALSE, and NA.

The chapter explains how comparisons produce logical values (TRUE or FALSE) and how these values can be combined using Boolean operators such as AND (&), OR (|), and NOT (!). These operations allow analysts to express complex rules for selecting data.

Logical vectors are then applied to summaries and conditional transformations, such as categorizing observations based on thresholds or conditions.

This chapter is crucial because logical thinking underpins nearly all data manipulation tasks. It formalizes the reasoning processes analysts intuitively use and translates them into precise, reproducible code.

Example 1

```
library(tidyverse)
library(nycflights13)
x <- c(1, 2, 3, 5, 7, 11, 13)
x * 2
```

```
[1]  2  4  6 10 14 22 26
```

```
df <- tibble(x)
df |>
  mutate(y = x * 2)
```

```
# A tibble: 7 x 2
      x     y
  <dbl> <dbl>
1     1     2
2     2     4
3     3     6
4     5    10
5     7    14
6    11    22
7    13    26
```

Example 2

```r
flights |>
  mutate(
    daytime = dep_time > 600 & dep_time < 2000,
    approx_ontime = abs(arr_delay) < 20,
  ) |>
  filter(daytime & approx_ontime)
```

```
# A tibble: 172,286 x 21
    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      601            600         1      844            850
 2  2013     1     1      602            610        -8      812            820
 3  2013     1     1      602            605        -3      821            805
 4  2013     1     1      606            610        -4      858            910
 5  2013     1     1      606            610        -4      837            845
 6  2013     1     1      607            607         0      858            915
 7  2013     1     1      611            600        11      945            931
 8  2013     1     1      613            610         3      925            921
 9  2013     1     1      615            615         0      833            842
10  2013     1     1      622            630        -8     1017           1014
# i 172,276 more rows
# i 13 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>, daytime <lgl>,
#   approx_ontime <lgl>
```

Example 3

```r
flights |>
  filter(arr_delay > 0) |>
  group_by(year, month, day) |>
  summarize(
    behind = mean(arr_delay),
    n = n(),
    .groups = "drop"
  )
```

```
# A tibble: 365 x 5
    year month   day behind     n
   <int> <int> <int>  <dbl> <int>
 1  2013     1     1   32.5   461
```

```
 2  2013     1     2   32.0   535
 3  2013     1     3   27.7   460
 4  2013     1     4   28.3   297
 5  2013     1     5   22.6   238
 6  2013     1     6   24.4   381
 7  2013     1     7   27.8   243
 8  2013     1     8   20.8   275
 9  2013     1     9   25.6   287
10  2013     1    10   27.3   220
# i 355 more rows
```

Chapter 13: Numbers

This chapter explores numeric data, focusing on how numbers are created, transformed, and summarized. Thus, strengthening quantitative literacy and prepares readers for both descriptive and inferential analysis.

The authors cover common numeric operations such as scaling, ranking, rounding, and aggregating. They emphasize the importance of understanding numeric precision, handling extreme values, and choosing appropriate summary statistics.

Numeric transformations are not purely mechanical; they encode analytical decisions. For instance, choosing a mean versus a median reflects assumptions about distribution and robustness.

Example 1

```
library(tidyverse)
library(nycflights13)
```

```
flights |> count(dest)
```

```
# A tibble: 105 x 2
   dest      n
   <chr> <int>
 1 ABQ     254
 2 ACK     265
 3 ALB     439
 4 ANC       8
 5 ATL   17215
 6 AUS    2439
 7 AVL     275
 8 BDL     443
 9 BGR     375
10 BHM     297
# i 95 more rows
```

Example 2

```
flights |>
  group_by(dest) |>
  summarize(
    n = n(),
```

```
    delay = mean(arr_delay, na.rm = TRUE)
  )
```

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

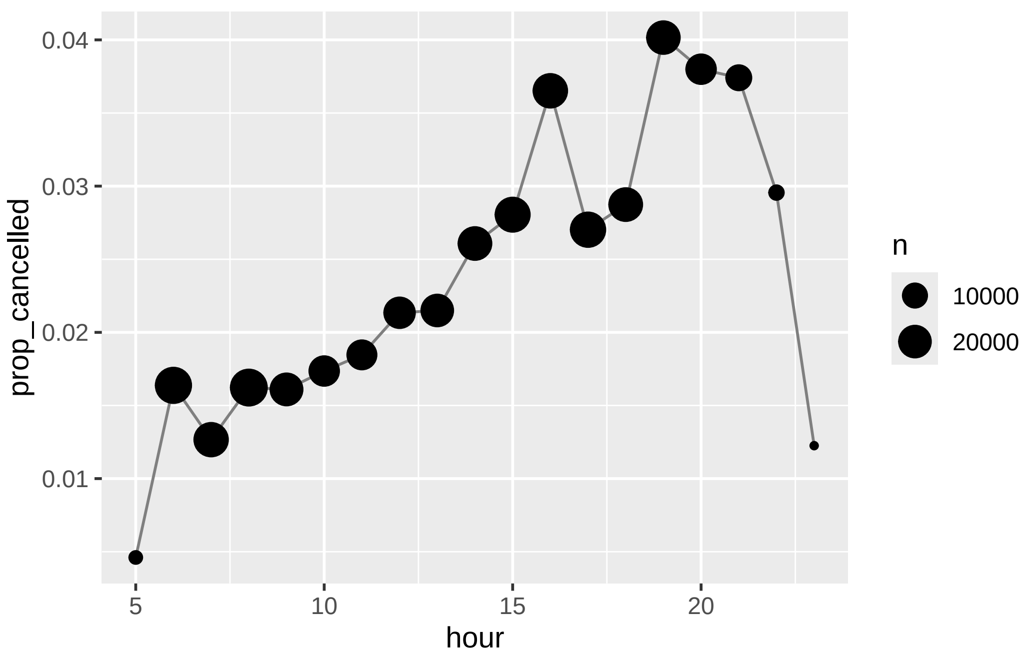
Example 3

```
flights |>
  group_by(dest) |>
  summarize(carriers = n_distinct(carrier)) |>
  arrange(desc(carriers))
```

```
# A tibble: 105 x 2
   dest  carriers
   <chr>    <int>
 1 ATL          7
 2 BOS          7
 3 CLT          7
 4 ORD          7
 5 TPA          7
 6 AUS          6
 7 DCA          6
 8 DTW          6
 9 IAD          6
10 MSP          6
# i 95 more rows
```

Example 4

```
flights |>
  group_by(hour = sched_dep_time %/% 100) |>
  summarize(prop_cancelled = mean(is.na(dep_time)), n = n()) |>
  filter(hour > 1) |>
  ggplot(aes(x = hour, y = prop_cancelled)) +
  geom_line(color = "grey50") +
  geom_point(aes(size = n))
```



Chapter 14: Strings

This chapter addresses string data, which is pervasive in real-world datasets but often messy and inconsistent.

The chapter introduces the stringr package, which provides a consistent and readable interface for string operations. Topics include:

- Creating and combining strings,
- Detecting patterns,
- Extracting and modifying substrings.

The authors stress that text manipulation is essential for tasks such as cleaning names, parsing identifiers, and standardizing categorical values.

Example 1

```
library(tidyverse)
library(babynames)
```

```
string1 <- "R is fascinating"
string2 <- 'It is best for statistical analysis'
string1
```

```
[1] "R is fascinating"
```

```
string2
```

```
[1] "It is best for statistical analysis"
```

Example 2

```
str_c("x", "y")
```

```
[1] "xy"
```

```
str_c("x", "y", "z")
```

```
[1] "xyz"
```

```r
str_c("Hello ", c("John", "Susan"))
```

```
[1] "Hello John"  "Hello Susan"
```

Example 3

```r
str_flatten(c("x", "y", "z"))
```

```
[1] "xyz"
```

```r
str_flatten(c("x", "y", "z"), ", ")
```

```
[1] "x, y, z"
```

```r
str_flatten(c("x", "y", "z"), ", ", last = ", and ")
```

```
[1] "x, y, and z"
```

Example 4

```r
df1 <- tibble(x = c("a,b,c", "d,e", "f"))
df1 |>
  separate_longer_delim(x, delim = ",")
```

```
# A tibble: 6 x 1
  x
  <chr>
1 a
2 b
3 c
4 d
5 e
6 f
```

Example 5

```r
df2 <- tibble(x = c("1211", "131", "21"))
df2 |>
  separate_longer_position(x, width = 1)
```

```
# A tibble: 9 x 1
  x
  <chr>
1 1
2 2
3 1
4 1
5 1
6 3
7 1
8 2
9 1
```

Chapter 15: Regular Expressions

This chapter builds on chapter 14 by introducing regular expressions (regex), a powerful language for pattern matching.

Rather than presenting regex as an abstract formalism, the authors focus on practical usage: detecting patterns, extracting components, and validating text formats.

The chapter also acknowledges the cognitive difficulty of regex and encourages gradual mastery through practice and reference rather than memorization. Regular expressions are positioned as a force multiplier, enabling analysts to solve complex text problems concisely and efficiently.

Example 1

```r
library(tidyverse)
library(babynames)
```

```r
str_view(fruit, "berry")
```

```
 [6] | bil<berry>
 [7] | black<berry>
[10] | blue<berry>
[11] | boysen<berry>
[19] | cloud<berry>
[21] | cran<berry>
[29] | elder<berry>
[32] | goji <berry>
[33] | goose<berry>
[38] | huckle<berry>
[50] | mul<berry>
[70] | rasp<berry>
[73] | salal <berry>
[76] | straw<berry>
```

Example 2

```r
str_view(c("a", "ab", "ae", "bd", "ea", "eab"), "a.")
```

```
[2] | <ab>
[3] | <ae>
[6] | e<ab>
```

Example 3

```r
str_view(c("a", "ab", "abb"), "ab?")
```

```
[1] | <a>
[2] | <ab>
[3] | <ab>b
```

Example 4

```r
str_detect(c("a", "b", "c"), "[aeiou]")
```

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

Example 5

```r
babynames |>
  filter(str_detect(name, "x")) |>
  count(name, wt = n, sort = TRUE)
```

```
# A tibble: 974 x 2
   name            n
   <chr>       <int>
 1 Alexander  665492
 2 Alexis     399551
 3 Alex       278705
 4 Alexandra  232223
 5 Max        148787
 6 Alexa      123032
 7 Maxine     112261
 8 Alexandria  97679
 9 Maxwell     90486
10 Jaxon       71234
# i 964 more rows
```

Chapter 16: Factors

This chapter focuses on categorical data, represented in R as factors.

The chapter explains why factors exist: they encode not just values, but also order and reference levels, which directly affect summaries, models, and visualizations.

It teaches us how to:

- Reorder factor levels,
- Relabel categories,
- Work with ordered factors.

Example 1

```
library(tidyverse)
```

```
x1 <- c("Dec", "Apr", "Jan", "Mar")
x1
```

```
[1] "Dec" "Apr" "Jan" "Mar"
```

```
sort(x1)
```

```
[1] "Apr" "Dec" "Jan" "Mar"
```

```
factor(x1)
```

```
[1] Dec Apr Jan Mar
Levels: Apr Dec Jan Mar
```

```
fct(x1)
```

```
[1] Dec Apr Jan Mar
Levels: Dec Apr Jan Mar
```

Example 2

```r
month_levels <- c(
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
)
```

```r
y1 <- factor(x1, levels = month_levels)
y1
```

```
[1] Dec Apr Jan Mar
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

```r
sort(y1)
```

```
[1] Jan Mar Apr Dec
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

Example 3

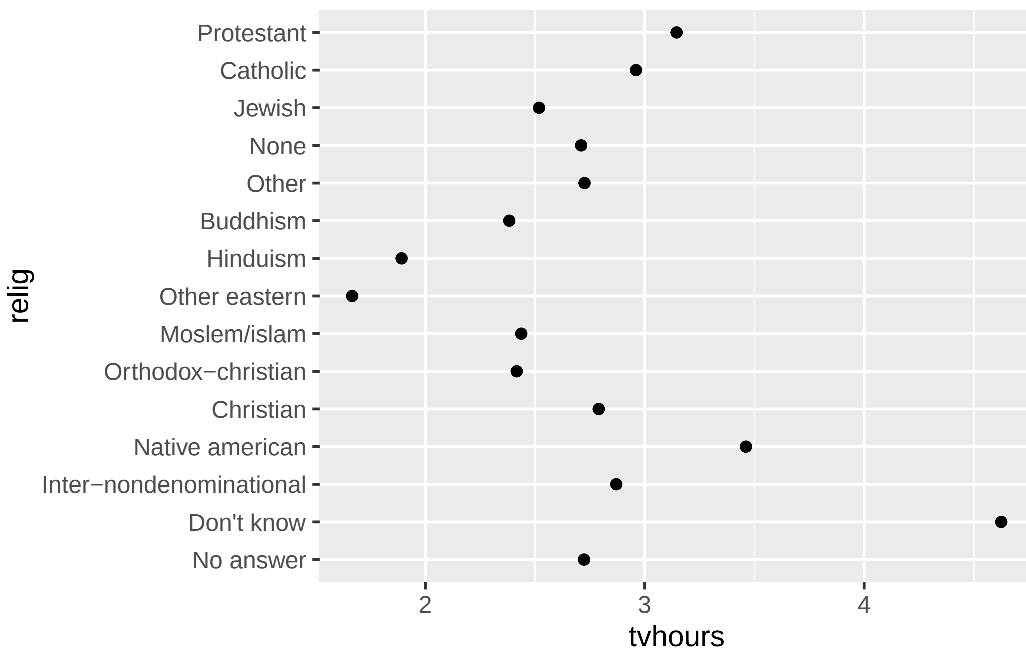
```r
gss_cat |>
  count(race)
```

```
# A tibble: 3 x 2
  race      n
  <fct> <int>
1 Other  1959
2 Black  3129
3 White 16395
```

Example 4

```r
relig_summary <- gss_cat |>
  group_by(relig) |>
  summarize(
    tvhours = mean(tvhours, na.rm = TRUE),
    n = n()
  )

ggplot(relig_summary, aes(x = tvhours, y = relig)) +
  geom_point()
```


Protestant
Catholic
Jewish
None
Other
Buddhism
Hinduism
Other eastern
Moslem/islam
Orthodox−christian
Christian
Native american
Inter−nondenominational
Don't know
No answer
relig
2
3
4
tvhours

Chapter 17: Dates and Times

This chapter addresses temporal data, one of the most complex data types to handle correctly.

Using the `lubridate` package, the authors explain how to parse dates and datetimes, extract components, calculate time spans, and handle time zones.

The chapter highlights common pitfalls, such as ambiguous date formats and daylight saving time issues, reinforcing the need for explicit and careful handling of time.

This chapter equips us with expertise to analyze trends, durations, and seasonality with confidence.

Example 1

```r
library(tidyverse)
library(nycflights13)
```

```r
today()
```

```
[1] "2025-12-14"
```

```r
now()
```

```
[1] "2025-12-14 20:35:40 UTC"
```

Example 2

```r
csv <- "
  date,datetime
  2022-01-02,2022-01-02 05:12
"
read_csv(csv)
```

```
Rows: 1 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: ","
dttm (1): datetime
date (1): date

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.
```

```
# A tibble: 1 x 2
  date       datetime
  <date>     <dttm>
1 2022-01-02 2022-01-02 05:12:00
```

Example 3

```
csv <- "
  date
  01/02/15
"

read_csv(csv, col_types = cols(date = col_date("%m/%d/%y")))
```

```
# A tibble: 1 x 1
  date
  <date>
1 2015-01-02
```

```
read_csv(csv, col_types = cols(date = col_date("%d/%m/%y")))
```

```
# A tibble: 1 x 1
  date
  <date>
1 2015-02-01
```

```
read_csv(csv, col_types = cols(date = col_date("%y/%m/%d")))
```

```
# A tibble: 1 x 1
  date
  <date>
1 2001-02-15
```

Example 4

```
ymd("2017-01-31")
```

```
[1] "2017-01-31"
```

```r
mdy("January 31st, 2017")
```

```
[1] "2017-01-31"
```

```r
dmy("31-Jan-2017")
```

```
[1] "2017-01-31"
```

Example 5

```r
flights |>
  select(year, month, day, hour, minute)
```

```
# A tibble: 336,776 x 5
    year month   day  hour minute
   <int> <int> <int> <dbl>  <dbl>
 1  2013     1     1     5     15
 2  2013     1     1     5     29
 3  2013     1     1     5     40
 4  2013     1     1     5     45
 5  2013     1     1     6      0
 6  2013     1     1     5     58
 7  2013     1     1     6      0
 8  2013     1     1     6      0
 9  2013     1     1     6      0
10  2013     1     1     6      0
# i 336,766 more rows
```

Chapter 18: Missing Values

This chapter focuses entirely on **missing data**, treating it as an analytical issue rather than a nuisance.

The authors distinguish between:

- Explicit missing values (NA),
- Implicit missing values (absent combinations).

They explain how missingness can bias summaries and models if handled carelessly and demonstrate tidyverse tools for identifying, removing, or explicitly representing missing data.

The chapter emphasizes transparency and intentionality in handling missing values.

Example 1

```
library(tidyverse)
```

```
treatment <- tribble(
  ~person,           ~treatment, ~response,
  "Derrick Whitmore", 1,         7,
  NA,                 2,         10,
  NA,                 3,         NA,
  "Katherine Burke",  1,         4
)
```

Example 2

```
treatment |>
  fill(everything())
```

```
# A tibble: 4 x 3
  person           treatment response
  <chr>                <dbl>    <dbl>
1 Derrick Whitmore         1        7
2 Derrick Whitmore         2       10
3 Derrick Whitmore         3       10
4 Katherine Burke          1        4
```

Example 3

```r
x <- c(1, 4, 5, 7, NA)
coalesce(x, 0)
```

```
[1] 1 4 5 7 0
```

Example 4

```r
x <- c(1, 4, 5, 7, -99)
na_if(x, -99)
```

```
[1]  1  4  5  7 NA
```

Chapter 19: Joins

This covers **joining data tables**, a fundamental operation in relational data analysis.

The chapter explains keys, relationships, and different types of joins (left, right, inner, full). It also explores how joins work conceptually and why they sometimes produce unexpected results.

Non-equi joins are introduced for advanced matching scenarios.

This chapter equips readers to combine datasets accurately and confidently, a critical skill in applied data science.

Example 1

```
library(tidyverse)
library(nycflights13)
```

```
planes |>
  count(tailnum) |>
  filter(n > 1)
```

```
# A tibble: 0 x 2
# i 2 variables: tailnum <chr>, n <int>
```

```
weather |>
  count(time_hour, origin) |>
  filter(n > 1)
```

```
# A tibble: 0 x 3
# i 3 variables: time_hour <dttm>, origin <chr>, n <int>
```

Example 2

```
planes |>
  filter(is.na(tailnum))
```

```
# A tibble: 0 x 9
# i 9 variables: tailnum <chr>, year <int>, type <chr>, manufacturer <chr>,
#   model <chr>, engines <int>, seats <int>, speed <int>, engine <chr>
```

```
weather |>
  filter(is.na(time_hour) | is.na(origin))
```

```
# A tibble: 0 x 15
# i 15 variables: origin <chr>, year <int>, month <int>, day <int>, hour <int>,
#   temp <dbl>, dewp <dbl>, humid <dbl>, wind_dir <dbl>, wind_speed <dbl>,
#   wind_gust <dbl>, precip <dbl>, pressure <dbl>, visib <dbl>,
#   time_hour <dttm>
```

Example 3

```
flights |>
  count(time_hour, carrier, flight) |>
  filter(n > 1)
```

```
# A tibble: 0 x 4
# i 4 variables: time_hour <dttm>, carrier <chr>, flight <int>, n <int>
```

```
airports |>
  count(alt, lat) |>
  filter(n > 1)
```

```
# A tibble: 1 x 3
    alt   lat     n
  <dbl> <dbl> <int>
1    13  40.6     2
```

Chapter 20: Spreadsheets

This chapter addresses importing data from spreadsheets, acknowledging that Excel and Google Sheets remain dominant data storage formats.

The chapter explains how to read, write, and manage spreadsheet data using R, while also warning about common spreadsheet issues such as hidden types, merged cells, and inconsistent formatting.

It bridges the gap between real-world data practices and reproducible analysis, reinforcing R's role as an analytical backbone rather than a replacement for spreadsheets.

Example 1

```
library(readxl)
library(tidyverse)
library(writexl)
```

```
students <- read_excel("students.xlsx")
students
```

```
# A tibble: 6 x 5
  `Student ID` `Full Name`      favourite.food     mealPlan            AGE
         <dbl> <chr>            <chr>              <chr>               <chr>
1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          4.0
2            2 Barclay Lynn     French fries       Lunch only          5.0
3            3 Jayendra Lyne    N/A                Breakfast and lunch 7.0
4            4 Leon Rossini     Anchovies          Lunch only          <NA>
5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
6            6 Güvenç Attila    Ice cream          Lunch only          6.0
```

Example 2

```
read_excel(
  "students.xlsx",
  col_names = c("student_id", "full_name", "favourite_food", "meal_plan", "age"),
  skip = 1
)
```

```
# A tibble: 6 x 5
  student_id full_name        favourite_food     meal_plan           age
       <dbl> <chr>            <chr>              <chr>               <chr>
```

```
1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          4.0
2            2 Barclay Lynn     French fries       Lunch only          5.0
3            3 Jayendra Lyne    N/A                Breakfast and lunch 7.0
4            4 Leon Rossini     Anchovies          Lunch only          <NA>
5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
6            6 Güvenç Attila    Ice cream          Lunch only          6.0
```

Example 3

```
students <- read_excel(
  "students.xlsx",
  col_names = c("student_id", "full_name", "favourite_food", "meal_plan", "age"),
  skip = 1,
  na = c("", "N/A"),
  col_types = c("numeric", "text", "text", "text", "text")
)

students <- students |>
  mutate(
    age = if_else(age == "five", "5", age),
    age = parse_number(age)
  )

students
```

```
# A tibble: 6 x 5
  student_id full_name        favourite_food     meal_plan             age
       <dbl> <chr>            <chr>              <chr>               <dbl>
1          1 Sunil Huffmann   Strawberry yoghurt Lunch only              4
2          2 Barclay Lynn     French fries       Lunch only              5
3          3 Jayendra Lyne    <NA>               Breakfast and lunch     7
4          4 Leon Rossini     Anchovies          Lunch only             NA
5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch     5
6          6 Güvenç Attila    Ice cream          Lunch only              6
```

Example 4

```
bake_sale <- tibble(
  item     = factor(c("brownie", "cupcake", "cookie")),
  quantity = c(10, 5, 8)
)
```

```r
bake_sale
```

```
# A tibble: 3 x 2
  item    quantity
  <fct>      <dbl>
1 brownie       10
2 cupcake        5
3 cookie         8
```

```r
write_xlsx(bake_sale, path = "bake-sale.xlsx")
```

```r
read_excel("bake-sale.xlsx")
```

```
# A tibble: 3 x 2
  item    quantity
  <chr>      <dbl>
1 brownie       10
2 cupcake        5
3 cookie         8
```