Week 3, Class 5

Transforming and Creating Variables

Sean Westwood

Today

- Understanding conditional logic step-by-step
- Creating new variables with `mutate()`
- Variable recoding with `case_when()` and `if_else()`
- Calculating proportions with `summarise()`
- Extracting values with `pull()`
- Combining multiple transformation steps

Why Must We Transform Data?

Real Political Science Data is Messy

Example: Survey responses about income:

- Raw: `$35,000`, `$45000`, `45k`, `forty-five thousand`
- Analysis needs: `35000`, `45000`, `45000`, `45000`

Example: Education levels to numeric:

- Raw: `high school`, `HS grad`, `12 years`, `completed secondary`
- Analysis needs: All coded as `12` (years of education)

Example: Age data:

- Raw: Birth years like `1985`, `1992`, `1967`
- Analysis needs: Age groups like `"18-29"`, `"30-44"`, `"45-64"`

The transformation process

1. **Clean** inconsistent formats
2. **Recode** into meaningful categories

3. **Calculate** new measures (ratios, percentages)

The majority of data analysis time is spent on data transformation!

Creating New Variables

Creating Variables with mutate()

The `mutate()` function adds new columns to your dataframe

Basic syntax:

```
data %>%
  mutate(
    new_variable = calculation,
    another_var = some_function(existing_var)
  )
```

summarise() creates a new dataframe with new columns; mutate() alters the original dataframe

What mutate() does:

1. Takes your existing data
2. Adds new column(s) based on calculations
3. Keeps all original columns
4. Returns the expanded dataset

Mathematical Operations in mutate()

Basic math:

```r
mutate(
  total = votes_dem + votes_rep,            # Addition
  margin = votes_dem - votes_rep,           # Subtraction
  vote_share = votes_dem / total_votes,     # Division
  doubled = approval_rating * 2             # Multiplication
)
```

This is row-wise! Each calculation will be independent for each row.

Useful calculations for political science:

```r
mutate(
  turnout_rate = votes_cast / eligible_voters,
  spending_per_vote = total_spent / votes_received,
  ideology_squared = ideology_score^2,
  log_income = log(household_income)
)
```

Understanding Conditional Logic

What is Conditional Logic?

In order to create new variables that are useful for analysis, we often need to use conditional logic.

Conditional logic: Making decisions based on conditions

In everyday life:

- *If* it's raining, *then* bring an umbrella
- *If* you're 21 or older, *then* you can drink alcohol
- *If* it is after 6:00 AM and before 10:00AM, *then* you can order breakfast at McDonalds

In R:

- *If* `party == "Democrat"`, *then* assign `"Liberal"`
- *If* `age >= 65`, *then* assign `"Senior"`
- *If* `approval_rating > 50`, *then* assign `"Majority Approves"`

The Building Blocks: Comparison Operators

Comparison operators:

- `==` (equal to)
- `!=` (not equal to)
- `>` (greater than)
- `>=` (greater than or equal to)
- `<` (less than)
- `<=` (less than or equal to)

The triad of 'equals'

- `<-` used to assign values to objects (make a variable 'get' a value)
- `=` Specify values when using a function (set an argumen to a value)
- `==` Test whether two values are equal (returning a logical result: TRUE, FALSE, or NA)

Why We Want TRUE/FALSE Outcomes

TRUE/FALSE is the foundation of data transformation because:

- All data decisions can ultimately be reduced down to "yes" or "no" questions
- We're essentially asking: "Does this observation meet our criteria?"
- These logical outcomes let us sort, categorize, and transform our data
- This binary logic is what allows us to create meaningful new variables from raw data with simple rules

Examples of TRUE/FALSE Outcomes

Equal to: `==`

```
party_id == "Republican"
# TRUE if Republican, FALSE otherwise
```

Not equal to: `!=`

```
education != "College"
# TRUE if not College, FALSE if College
```

Greater than: `>`

```r
age > 65
# TRUE if older than 65
```

Greater than or equal: `>=`

```r
income >= 50000
# TRUE if 50,000 or more
```

Less than: `<`

```r
approval < 40
# TRUE if less than 40
```

Less than or equal: `<=`

```r
years_in_office <= 2
# TRUE if 2 years or fewer
```

Understanding TRUE and FALSE

Every comparison returns TRUE or FALSE:

```r
# Examples with actual values
age <- 25
age >= 18         # Returns: TRUE
age >= 65         # Returns: FALSE

party <- "Democrat"
party == "Republican"    # Returns: FALSE
party != "Republican"    # Returns: TRUE
```

R treats TRUE as 1 and FALSE as 0:

We can apply mathimatical operators with TRUE/FALSE:

- `sum(TRUE, FALSE, TRUE)` = 2 (counts the TRUEs)
- `mean(TRUE, FALSE, TRUE)` = 0.67 (proportion that are TRUE)

Why mean() gives proportions: Since TRUE = 1 and FALSE = 0, the mean is the sum divided by the total count, which is exactly what a proportion is!

Detailed Operator Examples

Example 1: Creating a New Variable with == (Equal To)

Question: How many survey respondents identify as Democrats?

```r
# Sample data
survey_sample <- tibble(
  respondent = 1:8,
  party_id = c("Democrat", "Republican", "Democrat", "Independent",
               "Democrat", "Republican", "Democrat", "Independent")
)

# Show the data
survey_sample
```

```
# A tibble: 8 x 2
  respondent party_id
       <int> <chr>
1          1 Democrat
2          2 Republican
3          3 Democrat
4          4 Independent
5          5 Democrat
6          6 Republican
7          7 Democrat
8          8 Independent
```

Using == to Find Democrats

```r
# Test each row: is party_id equal to "Democrat"?
survey_sample %>%
  mutate(
    is_democrat = party_id == "Democrat"
  )
```

```
# A tibble: 8 x 3
  respondent party_id    is_democrat
       <int> <chr>       <lgl>
```

```
1           1 Democrat    TRUE
2           2 Republican  FALSE
3           3 Democrat    TRUE
4           4 Independent FALSE
5           5 Democrat    TRUE
6           6 Republican  FALSE
7           7 Democrat    TRUE
8           8 Independent FALSE
```

What happened:

- Row 1: `"Democrat" == "Democrat"` → `TRUE`
- Row 2: `"Republican" == "Democrat"` → `FALSE`

- Row 3: `"Democrat" == "Democrat"` → `TRUE`
- And so on...

Counting with ==

```
survey_sample %>%
  summarise(
    total_respondents = n(),
    democrats = sum(party_id == "Democrat"),
    democrat_percentage = mean(party_id == "Democrat") * 100
  )
```

```
# A tibble: 1 x 3
  total_respondents democrats democrat_percentage
              <int>     <int>               <dbl>
1                 8         4                  50
```

- `sum(party_id == "Democrat")` counts TRUE values
- `mean(party_id == "Democrat") * 100` gives the proportion of Democrats

Example 2: The != (Not Equal To) Operator

Question: How many respondents are NOT Republicans?

```r
# Count non-Republicans
survey_sample %>%
  summarise(
    non_republicans = sum(party_id != "Republican"),
    non_republican_pct = mean(party_id != "Republican") * 100
  )
```

```
# A tibble: 1 x 2
  non_republicans non_republican_pct
            <int>              <dbl>
1               6                 75
```

Using >= to Find Seniors

```r
# Create age sample data
age_sample <- tibble(
  voter_id = 1:10,
  age = c(22, 35, 67, 45, 72, 29, 81, 55, 19, 65)
)

# Show the data
age_sample
```

```
# A tibble: 10 x 2
   voter_id   age
      <int> <dbl>
 1        1    22
 2        2    35
 3        3    67
 4        4    45
 5        5    72
 6        6    29
 7        7    81
 8        8    55
 9        9    19
10       10    65
```

```r
# Count seniors
age_sample %>%
  summarise(
    total_voters = n(),
    seniors = sum(age >= 65),
    senior_percentage = mean(age >= 65) * 100
  )
```

```
# A tibble: 1 x 3
  total_voters seniors senior_percentage
         <int>   <int>             <dbl>
1           10       4                40
```

Example 4: The <= (Less Than or Equal) Operator

Question: How many voters are young adults (30 or younger)?

```r
# Count young adults
age_sample %>%
  summarise(
    young_adults = sum(age <= 30),
    young_adult_pct = mean(age <= 30) * 100
  )
```

```
# A tibble: 1 x 2
  young_adults young_adult_pct
         <int>           <dbl>
1            3              30
```

Multiple Conditions

The world is complex - we often need to examine multiple variables simultaneously

Real-world questions require multiple conditions:

- Young AND Democrat voters (age < 30 AND party == "Democrat")
- High achievers (high IQ AND admitted to Dartmouth)
- At-risk populations (low income AND poor health)
- Swing voters (independent AND frequent voters)

Combining Conditions with AND (&) and OR (|)

AND (`&`): Both conditions must be TRUE

```
age >= 18 & age < 65    # Working age adults
```

OR (`|`): Either condition can be TRUE

```
party == "Democrat" | party == "Republican"    # Major party members
```

Example of multiple conditions with real data

```
# Complex conditions
age_sample %>%
  mutate(
    working_age = age >= 18 & age < 65,
    voting_age = age >= 18
  ) %>%
  summarise(
    working_age_count = sum(working_age),
    voting_age_count = sum(voting_age)
  )
```

```
# A tibble: 1 x 2
  working_age_count voting_age_count
              <int>            <int>
1                 6               10
```

Putting This All Together

Logical Operations in mutate()

Creating TRUE/FALSE variables:

```r
mutate(
  is_competitive = vote_margin < 0.05,      # TRUE if close race
  high_turnout = turnout > 0.7,             # TRUE if high turnout
  experienced = years_in_office >= 4,       # TRUE if experienced
  young_high_turnout = age < 30 & turnout > 0.7,        # Young AND high turnout
  major_party_experienced = (party == "Democrat" | party == "Republican") & years_in_office >
)
```

Why create logical variables?:

- Easy to count: `sum(is_competitive)`
- Easy to get percentages: `mean(high_turnout)`
- Clear for analysis: `filter(major_party)`

The if_else() Function

What if we want to create a new variable that is a categorical variable (e.g., "Senior", "Non-Senior")?

Instead of creating a variable that shows is someone is young or not, we can create a varaible that calls them "Young" or "Old".

For simple binary decisions:

```r
mutate(
  age_group = if_else(age >= 65, "Senior", "Non-Senior"),
  result = if_else(vote_share > 0.5, "Won", "Lost"),
  income_level = if_else(income >= 50000, "High", "Low")
)
```

Syntax: `if_else(condition, value_if_true, value_if_false)`

When to use if_else():

- Only two possible outcomes
- Simple, clear conditions where the labels are sensible

The case_when() Function

Real political data often has **multiple meaningful categories** that can't be captured with simple TRUE/FALSE:

Example 1: Education levels:

- Raw data: Years of education (8, 12, 14, 16, 18, 20)
- Need categories: "Less than HS", "High School", "Some College", "Bachelor's", "Graduate"

Example 2: Congressional ideology scores:

- Raw data: DW-NOMINATE scores (-0.8, -0.2, 0.1, 0.6, 0.9)
- Need categories: "Very Liberal", "Liberal", "Moderate", "Conservative", "Very Conservative"

case_when() handles multiple conditions in order:

Using case_when()

For multiple conditions and outcomes:

What if we wanted to make a new variable called "age_category" by checking each person's age and assigning them to one of four age groups based on which condition they meet first?

```
mutate(
  age_category = case_when(
    age < 30 ~ "Young",
    age < 50 ~ "Middle-aged",
    age < 70 ~ "Older",
    TRUE ~ "Senior"  # catch-all for age >= 70
  )
)
```

Key rules for case_when():

1. Conditions are tested **in order**
2. First TRUE condition wins
3. Use ~ to separate condition from result
4. Always end with `TRUE ~ "catch-all"` for safety

Understanding the ~ Symbol

The tilde (~) means "then":

- `age < 30 ~ "Young"` means "if age < 30, then assign 'Young' "
- `income >= 100000 ~ "High"` means "if income >= 100000, then assign 'High' "

Think of it as an arrow:

- `condition ~ result`
- "When this condition is TRUE, give this result"

Only used in case_when(), not in if_else()

case_when() vs if_else()

Use if_else() for 2 categories:

```
party_type = if_else(party == "Independent", "Independent", "Major Party")
```

Use case_when() for 3+ categories:

```
party_group = case_when(
  party == "Democrat" ~ "Democrat",
  party == "Republican" ~ "Republican",
  party == "Independent" ~ "Independent",
  TRUE ~ "Other"
)
```

Both create new columns, but case_when() is more flexible

Common case_when() Patterns

Age groups:

```
age_group = case_when(
  age < 25 ~ "18-24",
  age < 35 ~ "25-34",
  age < 50 ~ "35-49",
  age < 65 ~ "50-64",
  TRUE ~ "65+"
)
```

Income brackets:

```
income_bracket = case_when(
  income < 25000 ~ "Low",
  income < 75000 ~ "Middle",
  TRUE ~ "High"
)
```

Performance ratings:

```
rating = case_when(
  score >= 90 ~ "Excellent",
  score >= 80 ~ "Good",
  score >= 70 ~ "Fair",
  TRUE ~ "Poor"
)
```

Examples: Summarize Without Mutate

Example 1: Congressional Voting Patterns Analysis

Research Question: What are the basic characteristics of congressional voting behavior by party?

Data: DW-NOMINATE dataset with 1,000 members of Congress, including ideology scores, party affiliation, chamber, and state information.

Code:

```
congress_data %>%
  group_by(party_code) %>%
  summarise(
    count = n(),
    mean_ideology = mean(nominate_dim1, na.rm = TRUE),
    median_ideology = median(nominate_dim1, na.rm = TRUE),
    percentage = n() / nrow(congress_data) * 100
  )
```

Results:

```
# A tibble: 3 x 5
  party_code  count mean_ideology median_ideology percentage
  <chr>       <int>         <dbl>           <dbl>      <dbl>
1 Democrat      451        -0.366          -0.359       45.1
2 Independent    40        -0.150          -0.184        4  
3 Republican    509         0.415           0.407       50.9
```

AI Prompts for This Congressional Analysis

Prompt to Create Code:

```
"I have congressional data that look like this:

Rows: 1000Rows: 1,000
Columns: 5
$ member_id     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1…
$ party_code    <chr> "Democrat", "Republican", "Democrat", "Democrat", "Repub…
...

Create tidyverse code to summarize by party: count of members, mean and median ideology score
```

AI Response:

```
I'll help you create a summary by party using group_by() and summarise(). Here's the code:

[code provided above]
```

Prompt to Explain Code:

```
"Explain this summarise code step by step. What does each function do and why do we use na.rm
```

AI Response:

```
Let me break this down:

1. group_by(party_code): Creates separate
   groups for each party
2. n(): Counts rows in each group
3. mean(nominate_dim1, na.rm=TRUE): Calculates
   average ideology, ignoring missing values
```

```
4. na.rm=TRUE prevents errors if any ideology
   scores are missing (NA)
5. percentage calculation: n() divided by
   total dataset size * 100
```

Example 2: Election Turnout Demographics Analysis

Research Question: How does voter turnout vary across different demographic groups?

Data: Simulated voter turnout dataset with 1,000 respondents, including age groups, education levels, income brackets, and turnout status.

Code:

```
voter_data %>%
  group_by(education_level) %>%
  summarise(
    total_respondents = n(),
    voters = sum(voted == "Yes"),
    non_voters = sum(voted == "No"),
    turnout_rate = mean(voted == "Yes"),
    median_age = median(age)
  )
```

Results:

```
# A tibble: 4 x 6
  education_level total_respondents voters non_voters turnout_rate median_age
  <chr>                       <int>  <int>      <int>        <dbl>      <dbl>
1 College                       280    201         79        0.718       52
2 Graduate                      170    137         33        0.806       52
3 High School                   416    229        187        0.550       53.5
4 Some College                  134     83         51        0.619       55
```

AI Prompts for Turnout Analysis

Prompt to Create Code:

```
"I have voter data tha tlook liek this

Rows: 1,000
```

```
Columns: 4
$ respondent_id   <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,…
$ education_level <chr> "High School", "Graduate", "College", "High School", "…
...

Using tidyverse Create a summary by education level showing: total respondents, number who vo
```

AI Response:

```
I'll create a grouped summary that analyzes turnout by education level:

[code provided above]

This uses:
- group_by() to analyze each education level
- sum(voted == "Yes") to count "Yes" responses
- mean(voted == "Yes") for turnout percentage
- median() for typical age in each group
```

Prompt to Explain Code:

```
"Explain how sum(voted == 'Yes') works to count voters and how mean(voted == 'Yes') calculate
```

AI Response:

```
Great question! This works because R treats TRUE/FALSE as 1/0:

sum(voted == "Yes"):
- Creates TRUE/FALSE for each row
- sum() treats TRUE as 1, FALSE as 0
- Result: count of "Yes" responses

mean(voted == "Yes"):
- Same TRUE/FALSE creation
- mean() of 1s and 0s = proportion
- Result: percentage who voted
```

More Conditional Logic Examples

Example: Income Analysis with Multiple Operators

Research Question: How do income patterns vary in survey data?

```
# A tibble: 12 x 3
   person_id annual_income state
       <int>         <dbl> <chr>
 1         1         25000 CA
 2         2         75000 TX
 3         3         45000 NY
 4         4        120000 CA
 5         5         35000 FL
 6         6         95000 TX
 7         7         15000 NY
 8         8         65000 FL
 9         9         85000 CA
10        10        200000 TX
11        11         42000 NY
12        12         58000 FL
```

Using Different Operators on Income Data

```r
income_demo %>%
  mutate(
    low_income = annual_income <= 30000,         # <= operator
    high_income = annual_income >= 100000,       # >= operator
    middle_income = annual_income > 30000 & annual_income < 100000,  # Combined
    not_wealthy = annual_income != 200000,       # != operator
    exactly_median = annual_income == 60000      # == operator (approximately)
  ) %>%
  select(person_id, annual_income, low_income, high_income, middle_income)
```

```
# A tibble: 12 x 5
   person_id annual_income low_income high_income middle_income
       <int>         <dbl> <lgl>      <lgl>       <lgl>
 1         1         25000 TRUE       FALSE       FALSE
 2         2         75000 FALSE      FALSE       TRUE
 3         3         45000 FALSE      FALSE       TRUE
```

```
4           4        120000 FALSE      TRUE        FALSE
5           5         35000 FALSE      FALSE       TRUE
6           6         95000 FALSE      FALSE       TRUE
7           7         15000 TRUE       FALSE       FALSE
8           8         65000 FALSE      FALSE       TRUE
9           9         85000 FALSE      FALSE       TRUE
10         10        200000 FALSE      TRUE        FALSE
11         11         42000 FALSE      FALSE       TRUE
12         12         58000 FALSE      FALSE       TRUE
```

Income Analysis Summary

```
income_demo %>%
  summarise(
    total_people = n(),
    low_income_count = sum(annual_income <= 30000),
    high_income_count = sum(annual_income >= 100000),
    middle_income_count = sum(annual_income > 30000 & annual_income < 100000),
    median_income = median(annual_income),
    avg_income = mean(annual_income)
  )
```

```
# A tibble: 1 x 6
  total_people low_income_count high_income_count middle_income_count
         <int>            <int>             <int>               <int>
1           12                2                 2                   8
# i 2 more variables: median_income <dbl>, avg_income <dbl>
```

AI Prompts for Income Analysis

Prompt to Create Code:

```
"I have income data with person_id, annual_income, and state variables. I want to create logi

person_id annual_income state
1         25000         CA
2         75000         TX
3         45000         NY

I want tidyverse code to do this."
```

Simulated AI Response:

```
I'll help you create logical variables and summarize the income data:

[code provided above]

Key techniques used:
- <= and >= operators for thresholds
- Combined conditions with & operator
- != operator for exclusions
- == operator for exact matches
- sum() counts TRUE values
```

Prompt to Explain Code:

```
"Explain how the logical operators work in this income analysis. Why do we use sum() to coun
```

Simulated AI Response:

```
Logical operators create TRUE/FALSE values:
- <= 30000 checks if income is at or below threshold
- >= 100000 checks if income meets high threshold
- & requires BOTH conditions to be true
- | requires at least ONE condition to be true

sum() with logical variables:
- TRUE = 1, FALSE = 0 in R
- sum(condition) counts how many TRUEs
- Efficient way to count observations meeting criteria
```

Example 3: Campaign Spending Analysis

Research Question: What are the spending patterns across different types of political campaigns?

Data: Campaign finance dataset with 1,000 candidates, including office type, party, total raised, total spent, and campaign outcome.

Code:

```r
campaign_data %>%
  group_by(office_type, party) %>%
  summarise(
    num_candidates = n(),
    avg_raised = mean(total_raised),
    median_spent = median(total_spent),
    win_rate = mean(outcome == "Won"),
    total_spending = sum(total_spent),
    .groups = "drop"
  )
```

Results:

```
# A tibble: 6 x 7
  office_type party      num_candidates avg_raised median_spent win_rate
  <chr>       <chr>               <int>      <dbl>        <dbl>    <dbl>
1 Governor    Democrat              142   3126128.     2852395.    0.472
2 Governor    Republican            138   3229189.     2855065.    0.551
3 House       Democrat              328    238780.      223938.    0.506
4 House       Republican            299    243071.      229582.    0.482
5 Senate      Democrat               40   1708165.     1576734.    0.425
6 Senate      Republican             53   1776919.     1631585.    0.547
# i 1 more variable: total_spending <dbl>
```

AI Prompts for Campaign Analysis

Prompt to Create Code:

```
"I have campaign data with office_type, party, total_raised, total_spent, and outcome variabl
Columns: 3
$ person_id     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
$ annual_income <dbl> 25000, 75000, 45000, 120000, 35000, 95000, 15000, 65000,…
$ state         <chr> "CA", "TX", "NY", "CA", "FL", "TX", "NY", "FL", "CA", "T…"

I want tidyverse code to do this.
```

Simulated AI Response:

```
I'll create a grouped summary by office type and party. Here's the code:

[code provided above]

This groups by TWO variables simultaneously:
- Each combination gets its own row
- Calculations done within each group
- .groups="drop" removes grouping after
```

Prompt to Explain Code:

```
"Walk me through this grouped summary. How does grouping by two variables work? Why might med
```

Simulated AI Response:

```
Multi-variable grouping creates subgroups:
- group_by(office_type, party) creates 6 groups
- Each office type + party combination
- Statistics calculated within each subgroup

Median vs Mean for spending:
- Campaign spending often has outliers
- Median less affected by extremely high spenders
- Mean pulled up by billionaire self-funders
- Median shows "typical" candidate spending
```

Extracting Values with pull()

When to Use pull()

Purpose: Extract a single column as a vector

Common use cases:

- Get a single statistic for further calculation
- Extract values for comparison
- Create variables from summary statistics

Basic syntax:

```r
data %>%
  pull(column_name)

# Or extract calculated values
data %>%
  summarise(mean_value = mean(variable)) %>%
  pull(mean_value)
```

Example: Using pull() for Calculations

```r
# Extract median income for threshold calculation
median_income <- survey_data %>%
  summarise(median_inc = median(income)) %>%
  pull(median_inc)

median_income
```

```
[1] 51198.99
```

```r
# Use extracted value in new calculation
survey_data %>%
  mutate(
    above_median = income > median_income,
    income_ratio = income / median_income
  ) %>%
  select(respondent_id, income, above_median, income_ratio) %>%
  slice_head(n = 5)
```

```
# A tibble: 5 x 4
  respondent_id  income above_median income_ratio
          <int>   <dbl> <lgl>               <dbl>
1             1  35899. FALSE               0.701
2             2 136369. TRUE                2.66
3             3  43636. FALSE               0.852
4             4  27250. FALSE               0.532
5             5  46590. FALSE               0.910
```

Advanced Conditional Logic

Example: Complex Campaign Classifications

Research Question: How can we classify campaigns by competitiveness and resource level?

```r
# Create sample campaign data with more variables
campaign_complex <- tibble(
  candidate = paste("Candidate", 1:10),
  vote_share = c(0.52, 0.48, 0.67, 0.33, 0.51, 0.49, 0.78, 0.22, 0.55, 0.45),
  total_spent = c(150000, 145000, 300000, 100000, 180000, 175000, 500000, 80000, 200000, 1900
  incumbent = c(TRUE, FALSE, TRUE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE)
)

campaign_complex
```

```
# A tibble: 10 x 4
   candidate    vote_share total_spent incumbent
   <chr>             <dbl>       <dbl> <lgl>
 1 Candidate 1        0.52      150000 TRUE
 2 Candidate 2        0.48      145000 FALSE
 3 Candidate 3        0.67      300000 TRUE
 4 Candidate 4        0.33      100000 FALSE
 5 Candidate 5        0.51      180000 FALSE
 6 Candidate 6        0.49      175000 TRUE
 7 Candidate 7        0.78      500000 TRUE
 8 Candidate 8        0.22       80000 FALSE
 9 Candidate 9        0.55      200000 FALSE
10 Candidate 10       0.45      190000 TRUE
```

Creating Complex Classifications with case_when()

```r
campaign_complex %>%
  mutate(
    # Competitiveness based on vote margin
    competitiveness = case_when(
      abs(vote_share - 0.5) <= 0.02 ~ "Extremely Close",  # Within 2%
      abs(vote_share - 0.5) <= 0.05 ~ "Competitive",      # Within 5%
      abs(vote_share - 0.5) <= 0.10 ~ "Somewhat Safe",    # Within 10%
      TRUE ~ "Safe"                                        # More than 10%
```

```r
  ),

  # Spending level
  spending_level = case_when(
    total_spent >= 400000 ~ "High Budget",
    total_spent >= 200000 ~ "Medium Budget",
    total_spent >= 100000 ~ "Low Budget",
    TRUE ~ "Minimal Budget"
  ),

  # Advantage type using multiple conditions
  advantage = case_when(
    incumbent == TRUE & total_spent >= 200000 ~ "Incumbent + Money",
    incumbent == TRUE & total_spent < 200000 ~ "Incumbent Only",
    incumbent == FALSE & total_spent >= 200000 ~ "Money Only",
    TRUE ~ "Neither Advantage"
  )
) %>%
select(candidate, vote_share, competitiveness, spending_level, advantage)
```

```
# A tibble: 10 x 5
   candidate    vote_share competitiveness spending_level advantage
   <chr>             <dbl> <chr>           <chr>          <chr>
 1 Candidate 1        0.52 Competitive     Low Budget     Incumbent Only
 2 Candidate 2        0.48 Competitive     Low Budget     Neither Advantage
 3 Candidate 3        0.67 Safe            Medium Budget  Incumbent + Money
 4 Candidate 4        0.33 Safe            Low Budget     Neither Advantage
 5 Candidate 5        0.51 Extremely Close Low Budget     Neither Advantage
 6 Candidate 6        0.49 Extremely Close Low Budget     Incumbent Only
 7 Candidate 7        0.78 Safe            High Budget    Incumbent + Money
 8 Candidate 8        0.22 Safe            Minimal Budget Neither Advantage
 9 Candidate 9        0.55 Somewhat Safe   Medium Budget  Money Only
10 Candidate 10       0.45 Competitive     Low Budget     Incumbent Only
```

Understanding Complex Conditions

Breaking down the `advantage` classification:

1. `incumbent == TRUE & total_spent >= 200000`
    - Must be incumbent AND high spending
    - Both conditions must be TRUE

2. `incumbent == TRUE & total_spent < 200000`

   - Must be incumbent AND low spending
   - Checked only if #1 is FALSE

3. `incumbent == FALSE & total_spent >= 200000`

   - Must be challenger AND high spending
   - Checked only if #1 and #2 are FALSE

4. `TRUE`

   - Catches all remaining cases
   - Challenger with low spending

Example 4: Public Opinion Polling

Research Question: How do approval ratings vary across different demographic and geographic groups?

Code:

```r
polling_data %>%
  group_by(region, party_id) %>%
  summarise(
    sample_size = n(),
    avg_approval = mean(approval_rating),
    approval_range = max(approval_rating) - min(approval_rating),
    high_approval_pct = mean(approval_rating > 50),
    response_rate = mean(!is.na(approval_rating)),
    .groups = "drop"
  )
```

```
# A tibble: 12 x 7
   region    party_id  sample_size avg_approval approval_range high_approval_pct
   <chr>     <chr>           <int>        <dbl>          <dbl>             <dbl>
 1 Midwest   Democrat           72         61.8             75             0.681
 2 Midwest   Independ~          88         54.1             60             0.614
 3 Midwest   Republic~          86         45.1             60             0.453
 4 Northeast Democrat           88         57.4             72             0.625
 5 Northeast Independ~          94         52.7             60             0.5
 6 Northeast Republic~          71         44.8             60             0.423
 7 South     Democrat           72         61.8             75             0.681
 8 South     Independ~          87         57.2             60             0.644
```

```
 9 South     Republic~          87          47.9              60              0.448
10 West      Democrat           80          51.5              75              0.5
11 West      Independ~          94          56.2              57              0.606
12 West      Republic~          81          44.9              58              0.395
# i 1 more variable: response_rate <dbl>
```

AI Prompt to Create:

```
"I have polling data with region, party_id, and approval_rating variables. Create a summary 
```

AI Prompt to Explain:

```
"<Paste the results from R.>

Explain how mean(approval_rating > 50) calculates the percentage with high approval, and how
```

Examples: Mutate Operations

Example 5: Creating Campaign Efficiency Metrics

Research Question: How can we create meaningful efficiency metrics for campaign spending?

```r
campaign_data %>%
  mutate(
    spending_efficiency = vote_share / (total_spent / 1000),
    fundraising_ratio = total_raised / total_spent,
    cost_per_vote = total_spent / total_votes,
    budget_category = case_when(
      total_spent < 100000 ~ "Low Budget",
      total_spent < 500000 ~ "Medium Budget",
      TRUE ~ "High Budget"
    )
  ) %>%
  select(candidate_name, spending_efficiency, fundraising_ratio,
         cost_per_vote, budget_category) %>%
  slice_head(n = 5)
```

```
# A tibble: 5 x 5
  candidate_name  spending_efficiency fundraising_ratio cost_per_vote
  <chr>                         <dbl>             <dbl>         <dbl>
1 Anderson, John             0.00128              0.842          5.55
2 Williams, Susan            0.00355              1.19           1.71
3 Davis, John                0.000134             0.867         50.5
4 Smith, David               0.00170              1.19           3.58
5 Miller, Bob                0.000334             1.42          17.7
# i 1 more variable: budget_category <chr>
```

AI Prompt to Create:

```
"I have campaign data with vote_share, total_spent, total_raised, and total_votes. Create new
```

AI Prompt to Explain:

```
"Explain each mutate calculation. Why divide total_spent by 1000 for efficiency? How does cas
```

Example 6: Demographic Recoding and Analysis

Research Question: How can we create standardized demographic categories for analysis?

```
survey_data %>%
  mutate(
    age_group = case_when(
      age < 30 ~ "18-29",
      age < 45 ~ "30-44",
      age < 65 ~ "45-64",
      TRUE ~ "65+"
    ),
    income_quintile = ntile(income, 5),
    education_years_std = (education_years - mean(education_years)) / sd(education_years),
    high_earner = income > median(income),
    political_engagement = (political_interest + civic_participation) / 2
  ) %>%
  select(respondent_id, age_group, income_quintile, education_years_std,
         high_earner, political_engagement) %>%
  slice_head(n = 5)
```

```
# A tibble: 5 x 6
  respondent_id age_group income_quintile education_years_std high_earner
          <int> <chr>               <int>               <dbl> <lgl>
1             1 45-64                   4               0.845 TRUE
2             2 65+                     1               0.583 FALSE
3             3 30-44                   4              -1.25  TRUE
4             4 65+                     5              -0.201 TRUE
5             5 45-64                   5              -0.724 TRUE
# i 1 more variable: political_engagement <dbl>
```

AI Prompt to Create:

```
"I have survey data with age, income, education_years, political_interest, and civic_partici
```

AI Prompt to Explain:

```
"Explain how ntile() creates quintiles, what standardization does to education_years, and hov
```

Example 7: Electoral Performance Metrics

Research Question: How can we measure electoral competitiveness and performance?

```r
election_data %>%
  mutate(
    vote_margin = abs(dem_votes - rep_votes),
    competitive_race = vote_margin < (total_votes * 0.05),
    dem_vote_share = dem_votes / total_votes,
    turnout_rate = total_votes / eligible_voters,
    incumbent_advantage = if_else(incumbent_party == winner, 1, 0),
    upset_victory = (dem_vote_share > 0.5 & dem_expected_share < 0.5) |
                   (dem_vote_share < 0.5 & dem_expected_share > 0.5)
  ) %>%
  select(district, competitive_race, dem_vote_share, turnout_rate,
         incumbent_advantage, upset_victory) %>%
  slice_head(n = 5)
```

```
# A tibble: 5 x 6
  district competitive_race dem_vote_share turnout_rate incumbent_advantage
  <chr>    <lgl>                     <dbl>        <dbl>               <dbl>
1 NV-02    FALSE                     0.537        0.716                   1
```

```
2 ID-03    FALSE                      0.261        0.825                       0
3 VT-06    TRUE                       0.518        0.579                       0
4 IL-02    FALSE                      0.301        0.733                       1
5 MI-01    FALSE                      0.707        0.612                       1
# i 1 more variable: upset_victory <lgl>
```

AI Prompt to Create:

```
"I have election data with dem_votes, rep_votes, total_votes, eligible_voters, incumbent_par
```

AI Prompt to Explain:

```
"Explain the logic for identifying competitive races and upset victories. How do the logical
```

Example 8: Policy Position Scaling

Research Question: How can we create meaningful scales from multiple policy positions?

```r
legislator_data %>%
  mutate(
    economic_conservatism = (tax_cuts_score + deregulation_score +
                            anti_union_score) / 3,
    social_conservatism = (abortion_restrictions + traditional_marriage +
                          religious_freedom) / 3,
    overall_ideology = (economic_conservatism + social_conservatism) / 2,
    ideology_category = case_when(
      overall_ideology < 3 ~ "Liberal",
      overall_ideology < 7 ~ "Moderate",
      TRUE ~ "Conservative"
    ),
    extreme_position = overall_ideology < 2 | overall_ideology > 8,
    policy_consistency = abs(economic_conservatism - social_conservatism)
  ) %>%
  select(legislator_name, economic_conservatism, social_conservatism,
         overall_ideology, ideology_category, policy_consistency) %>%
  slice_head(n = 5)
```

```
# A tibble: 5 x 6
  legislator_name economic_conservatism social_conservatism overall_ideology
  <chr>                           <dbl>               <dbl>            <dbl>
```

```
1 Rep. Brooks                    5                   7                 6
2 Rep. Carter                    3                   3.67              3.33
3 Rep. Harris                    7.33                5.33              6.33
4 Sen. Garcia                    4.67                5                 4.83
5 Rep. Brooks                    2.67                5                 3.83
# i 2 more variables: ideology_category <chr>, policy_consistency <dbl>
```

AI Prompt to Create:

```
"I have legislator data with multiple policy voting scores (1-10 scale). Create composite sco
```

AI Prompt to Explain:

```
"Explain how averaging creates composite scores and how policy_consistency measures the diffe
```

Examples: Variable Recoding

Example 9: Income Bracket Recoding

Research Question: How can we create meaningful income categories from continuous income data?

```r
income_data %>%
  mutate(
    income_bracket = case_when(
      household_income < 25000 ~ "Low Income",
      household_income < 50000 ~ "Lower Middle",
      household_income < 75000 ~ "Middle Class",
      household_income < 100000 ~ "Upper Middle",
      household_income < 150000 ~ "High Income",
      TRUE ~ "Very High Income"
    ),
    poverty_status = case_when(
      household_income < 15000 ~ "Below Poverty",
      household_income < 30000 ~ "Near Poverty",
      TRUE ~ "Above Poverty"
    ),
    economic_security = if_else(
      household_income > 75000 & savings > 10000,
      "Secure", "Insecure"
```

```
    )
  ) %>%
  count(income_bracket, poverty_status, economic_security) %>%
  slice_head(n = 10)
```

```
# A tibble: 10 x 4
   income_bracket   poverty_status economic_security     n
   <chr>            <chr>          <chr>             <int>
 1 High Income      Above Poverty  Insecure             50
 2 High Income      Above Poverty  Secure               91
 3 Low Income       Below Poverty  Insecure             12
 4 Low Income       Near Poverty   Insecure             72
 5 Lower Middle     Above Poverty  Insecure            252
 6 Lower Middle     Near Poverty   Insecure             65
 7 Middle Class     Above Poverty  Insecure            247
 8 Upper Middle     Above Poverty  Insecure             35
 9 Upper Middle     Above Poverty  Secure              118
10 Very High Income Above Poverty  Insecure             16
```

AI Prompt to Create:

```
"I have household income and savings data.
Create income brackets (6 categories), poverty
status (3 categories), and economic security
(combining income and savings thresholds).
Show the distribution across all categories."
```

AI Prompt to Explain:

```
"Explain how case_when processes conditions
in order and why TRUE is used as the final
condition. How does the compound condition
in economic_security work?"
```

Example 10: Geographic Region Consolidation

Research Question: How can we group states into meaningful political regions?

```r
state_data %>%
  mutate(
    political_region = case_when(
      state %in% c("CA", "OR", "WA", "HI") ~ "Pacific Coast",
      state %in% c("NY", "NJ", "CT", "MA", "RI", "VT", "NH", "ME") ~ "Northeast",
      state %in% c("TX", "FL", "GA", "NC", "SC", "VA", "TN", "KY", "WV",
                   "AL", "MS", "LA", "AR", "OK") ~ "South",
      state %in% c("IL", "IN", "OH", "MI", "WI", "MN", "IA", "MO",
                   "ND", "SD", "NE", "KS") ~ "Midwest",
      state %in% c("CO", "UT", "NV", "AZ", "NM", "WY", "MT", "ID") ~ "Mountain West",
      TRUE ~ "Other"
    ),
    battleground_state = state %in% c("PA", "MI", "WI", "AZ", "GA", "NC", "FL"),
    electoral_votes_category = case_when(
      electoral_votes >= 20 ~ "Large State",
      electoral_votes >= 10 ~ "Medium State",
      TRUE ~ "Small State"
    )
  ) %>%
  count(political_region, battleground_state, electoral_votes_category) %>%
  slice_head(n = 10)
```

```
# A tibble: 10 x 4
   political_region battleground_state electoral_votes_category     n
   <chr>            <lgl>              <chr>                    <int>
 1 Midwest          FALSE              Medium State                80
 2 Midwest          FALSE              Small State                120
 3 Midwest          TRUE               Medium State                40
 4 Mountain West    FALSE              Medium State                60
 5 Mountain West    FALSE              Small State                 80
 6 Mountain West    TRUE               Medium State                20
 7 Northeast        FALSE              Medium State                20
 8 Northeast        FALSE              Small State                140
 9 Other            FALSE              Small State                 60
10 Other            TRUE               Small State                 20
```

AI Prompt to Create:

```
"I have state-level data with state abbreviations and electoral_votes. Create political regi
```

AI Prompt to Explain:

"Explain how %in% works to check if a state belongs to a vector of states. Why is this more e

Example 11: Survey Response Cleaning

Research Question: How can we clean and standardize survey response data?

```r
survey_clean_data %>%
  mutate(
    party_clean = case_when(
      str_detect(party_id, regex("dem|liberal", ignore_case = TRUE)) ~ "Democrat",
      str_detect(party_id, regex("rep|conserv|gop", ignore_case = TRUE)) ~ "Republican",
      str_detect(party_id, regex("ind|none|other", ignore_case = TRUE)) ~ "Independent",
      is.na(party_id) ~ "Unknown",
      TRUE ~ "Other"
    ),
    approval_numeric = case_when(
      approval %in% c("Strongly Approve", "Approve") ~ 1,
      approval %in% c("Strongly Disapprove", "Disapprove") ~ 0,
      approval == "Neither" ~ 0.5,
      TRUE ~ NA_real_
    ),
    education_level = case_when(
      education < 12 ~ "Less than High School",
      education == 12 ~ "High School Graduate",
      education %in% 13:15 ~ "Some College",
      education == 16 ~ "College Graduate",
      education > 16 ~ "Post-Graduate",
      TRUE ~ "Unknown"
    )
  ) %>%
  count(party_clean, education_level) %>%
  slice_head(n = 10)
```

```
# A tibble: 10 x 3
   party_clean education_level           n
   <chr>       <chr>                 <int>
 1 Democrat    College Graduate         21
 2 Democrat    High School Graduate     24
 3 Democrat    Less than High School    90
 4 Democrat    Post-Graduate           102
 5 Democrat    Some College             91
```

```
 6 Independent College Graduate         21
 7 Independent High School Graduate     22
 8 Independent Less than High School    99
 9 Independent Post-Graduate            81
10 Independent Some College             66
```

AI Prompt to Create:

```
"I have messy survey data with party_id (text), approval (categories), and education (years)
```

AI Prompt to Explain:

```
"Explain how str_detect with regex works for party identification and why ignore_case = TRUE
```

Example 12: Policy Position Recoding

Research Question: How can we simplify complex policy positions into analyzable categories?

```
voting_data %>%
  mutate(
    healthcare_position = case_when(
      healthcare_vote == "Yes" & medicare_expansion == "Yes" ~ "Progressive",
      healthcare_vote == "Yes" & medicare_expansion == "No" ~ "Moderate",
      healthcare_vote == "No" & medicare_expansion == "No" ~ "Conservative",
      healthcare_vote == "Abstain" | medicare_expansion == "Abstain" ~ "Unclear",
      TRUE ~ "Mixed"
    ),
    environmental_stance = case_when(
      climate_action == "Yes" & clean_energy == "Yes" ~ "Pro-Environment",
      climate_action == "No" & clean_energy == "No" ~ "Anti-Environment",
      TRUE ~ "Mixed Environment"
    ),
    fiscal_ideology = case_when(
      tax_cuts == "Yes" & spending_cuts == "Yes" ~ "Fiscal Conservative",
      tax_cuts == "No" & spending_cuts == "No" ~ "Fiscal Liberal",
      tax_cuts == "Yes" & spending_cuts == "No" ~ "Tax Cutter",
      tax_cuts == "No" & spending_cuts == "Yes" ~ "Spending Cutter",
      TRUE ~ "Fiscal Moderate"
    )
```

```r
  ) %>%
  count(healthcare_position, environmental_stance, fiscal_ideology) %>%
  slice_head(n = 10)
```

```
# A tibble: 10 x 4
   healthcare_position environmental_stance fiscal_ideology         n
   <chr>               <chr>                <chr>               <int>
 1 Conservative        Anti-Environment     Fiscal Conservative     1
 2 Conservative        Anti-Environment     Fiscal Liberal          2
 3 Conservative        Anti-Environment     Fiscal Moderate         8
 4 Conservative        Anti-Environment     Spending Cutter         1
 5 Conservative        Mixed Environment    Fiscal Conservative     4
 6 Conservative        Mixed Environment    Fiscal Liberal         14
 7 Conservative        Mixed Environment    Fiscal Moderate        52
 8 Conservative        Mixed Environment    Spending Cutter        12
 9 Conservative        Mixed Environment    Tax Cutter             14
10 Conservative        Pro-Environment      Fiscal Conservative     1
```

AI Prompt to Create:

```
"I have legislator voting data with Yes/No/Abstain votes on healthcare, medicare_expansion, 
```

AI Prompt to Explain:

```
"Explain how combining multiple vote conditions creates more nuanced categories than single 
```

Complex Examples

Complex Example 1: Comprehensive Electoral Analysis

Research Question: What factors predict electoral success across different types of races and regions?

```r
electoral_analysis <- complex_election_data %>%
  # Filter to competitive races only
  filter(vote_margin < 0.1) %>%

  # Create new variables
  mutate(
```

```r
    spending_advantage = (candidate_spending - opponent_spending) /
                         (candidate_spending + opponent_spending),
    experience_level = case_when(
      prior_office == "None" ~ "Newcomer",
      prior_office %in% c("Local", "County") ~ "Local Experience",
      prior_office %in% c("State", "Federal") ~ "Experienced",
      TRUE ~ "Other"
    ),
    demographic_match = case_when(
      candidate_race == district_majority_race ~ "Matches District",
      TRUE ~ "Different from District"
    ),
    upset_potential = if_else(
      polling_deficit > 5 & spending_advantage > 0.2,
      "High", "Low"
    )
  ) %>%

  # Group analysis by region and office type
  group_by(region, office_type, experience_level) %>%
  summarise(
    total_races = n(),
    avg_spending_advantage = mean(spending_advantage, na.rm = TRUE),
    win_rate = mean(outcome == "Won"),
    upset_rate = mean(upset_potential == "High", na.rm = TRUE),
    .groups = "drop"
  ) %>%
  arrange(desc(win_rate))

electoral_analysis %>% slice_head(n = 8)
```

```
# A tibble: 8 x 7
  region    office_type experience_level total_races avg_spending_advantage
  <chr>     <chr>       <chr>                  <int>                  <dbl>
1 Midwest   Governor    Newcomer                   6                 0.151
2 West      Senate      Local Experience          11                -0.0585
3 South     Governor    Newcomer                  10                -0.132
4 Northeast Governor    Newcomer                   7                 0.150
5 Midwest   Senate      Experienced                9                -0.214
6 West      Governor    Experienced               11                -0.125
7 Midwest   Senate      Local Experience          16                 0.218
8 Northeast House       Experienced               16                 0.0514
```

```
# i 2 more variables: win_rate <dbl>, upset_rate <dbl>
```

AI Prompt to Create:

```
"I have comprehensive election data with candidate info, spending, demographics, and results
```

AI Prompt to Explain:

```
"Explain this complex analysis workflow. How do the filter, mutate, group_by, and summarise 
```

Complex Example 2: Public Opinion Trend Analysis

Research Question: How do public opinion trends vary across demographic groups?

```
opinion_trends <- complex_polling_data %>%
  # Create time-based variables
  mutate(
    year_month = floor_date(survey_date, "month"),
    approval_binary = if_else(approval_rating > 50, 1, 0),
    issue_priority = case_when(
      top_issue %in% c("Economy") ~ "Economic Issues",
      top_issue %in% c("Healthcare") ~ "Social Issues",
      top_issue %in% c("Immigration", "Crime") ~ "Security Issues",
      top_issue %in% c("Environment") ~ "Environmental Issues",
      TRUE ~ "Other Issues"
    ),
    age_generation = case_when(
      age < 30 ~ "Gen Z",
      age < 45 ~ "Millennial",
      age < 65 ~ "Gen X",
      TRUE ~ "Boomer+"
    )
  ) %>%

  # Group by demographics and time
  group_by(year_month, age_generation, party_id) %>%
  summarise(
    sample_size = n(),
    avg_approval = mean(approval_rating),
    approval_rate = mean(approval_binary),
```

```r
    economic_optimism = mean(economic_outlook == "Positive"),
    .groups = "keep"
  ) %>%

  # Calculate trends within groups
  group_by(age_generation, party_id) %>%
  mutate(
    approval_trend = approval_rate - lag(approval_rate),
    trend_direction = case_when(
      approval_trend > 0.05 ~ "Rising",
      approval_trend < -0.05 ~ "Falling",
      TRUE ~ "Stable"
    )
  ) %>%
  ungroup() %>%
  filter(year_month >= "2023-06-01")

opinion_trends %>% slice_head(n = 8)
```

```
# A tibble: 8 x 9
  year_month age_generation party_id    sample_size avg_approval approval_rate
  <date>     <chr>          <chr>             <int>        <dbl>         <dbl>
1 2023-06-01 Boomer+        Democrat             12         52.3         0.583
2 2023-06-01 Boomer+        Independent          12         42.1         0.25
3 2023-06-01 Boomer+        Republican           10         53.9         0.4
4 2023-06-01 Gen X          Democrat              9         48.1         0.444
5 2023-06-01 Gen X          Independent           6         51.8         0.5
6 2023-06-01 Gen X          Republican            5         57.8         0.6
7 2023-06-01 Gen Z          Democrat              3         41           0.333
8 2023-06-01 Gen Z          Independent           6         55.8         0.5
# i 3 more variables: economic_optimism <dbl>, approval_trend <dbl>,
#   trend_direction <chr>
```

AI Prompt to Create:

```
"I have longitudinal polling data with dates, demographics, approval ratings, and issue prio
```

AI Prompt to Explain:

```
"Explain this complex analysis workflow. How do multiple group_by operations work? What does
```

Best Practices

Writing Effective AI Prompts

Structure your requests clearly:

1. **Data description**: "I have a dataset with variables X, Y, Z…"
2. **Goal**: "I want to create/calculate/recode…"
3. **Specific requirements**: "Show me the code and explain each step"

Example prompt structure:

```
"I have campaign finance data with variables: candidate_name, total_spent, total_raised, vote
```

Common Mistakes to Avoid

Logical operator confusion:

- Use `&` for AND, `|` for OR in conditions
- Remember operator precedence with parentheses

case_when() ordering:

- Conditions are evaluated in order
- More specific conditions should come first
- Always include a catch-all with `TRUE`

Missing value handling:

- Use `na.rm = TRUE` in summary functions when needed
- Consider `is.na()` for missing value conditions

Summary

Key Functions Learned

Variable creation:

- `mutate()`: Create new variables with calculations
- `case_when()`: Complex conditional recoding
- `if_else()`: Simple binary recoding

Advanced summarization:

- `summarise()` with proportions and percentages
- `group_by()` with multiple grouping variables
- `pull()` for extracting single values

Mathematical and logical operations:

- Arithmetic calculations and ratios
- Logical comparisons and boolean logic
- Text pattern matching with `str_detect()`