

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:

```
1 data %>%  
2   mutate(  
3     new_variable = calculation,  
4     another_var = some_function(existing_var)  
5   )
```

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:

```
1 mutate(  
2   total = votes_dem + votes_rep,      # Addition  
3   margin = votes_dem - votes_rep,    # Subtraction  
4   vote_share = votes_dem / total_votes, # Division  
5   doubled = approval_rating * 2      # Multiplication  
6 )
```

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

Useful calculations for political science:

```
1 mutate(  
2   turnout_rate = votes_cast / eligible_voters,  
3   spending_per_vote = total_spent / votes_received,  
4   ideology_squared = ideology_score^2,  
5   log_income = log(household_income)  
6 )
```


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: **==**

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

Greater than or equal: **>=**

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

Not equal to: **!=**

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

Less than: **<**

```
1 approval < 40  
2 # TRUE if less than 40
```

Greater than: **>**

```
1 age > 65  
2 # TRUE if older than 65
```

Less than or equal: **<=**

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

Understanding TRUE and FALSE

Every comparison returns TRUE or FALSE:

```
1 # Examples with actual values
2 age <- 25
3 age >= 18      # Returns: TRUE
4 age >= 65      # Returns: FALSE
5
6 party <- "Democrat"
7 party == "Republican" # Returns: FALSE
8 party != "Republican" # Returns: TRUE
```

R treats TRUE as 1 and FALSE as 0:

We can apply mathematical 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?

```
1 # Sample data
2 survey_sample <- tibble(
3   respondent = 1:8,
4   party_id = c("Democrat", "Republican", "Democrat", "Independent",
5               "Democrat", "Republican", "Democrat", "Independent")
6 )
7
8 # Show the data
9 survey_sample
```

```
# A tibble: 8 × 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

```
1 # Test each row: is party_id equal to "Democrat"?
2 survey_sample %>%
3   mutate(
4     is_democrat = party_id == "Democrat"
5   )
```

```
# A tibble: 8 × 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 ==

```
1 survey_sample %>%
2   summarise(
3     total_respondents = n(),
4     democrats = sum(party_id == "Democrat"),
5     democrat_percentage = mean(party_id == "Democrat") * 100
6   )
```

```
# A tibble: 1 × 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?

```
1 # Count non-Republicans
2 survey_sample %>%
3   summarise(
4     non_republicans = sum(party_id != "Republican"),
5     non_republican_pct = mean(party_id != "Republican") * 100
6   )
```

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

Using `>=` to Find Seniors

```
1 # Create age sample data
2 age_sample <- tibble(
3   voter_id = 1:10,
4   age = c(22, 35, 67, 45, 72, 29, 81, 55, 19, 65)
5 )
6
7 # Show the data
8 age_sample
```

```
# A tibble: 10 × 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
```

```
1 # Count seniors
2 age_sample %>%
3   summarise(
4     total_voters = n(),
5     seniors = sum(age >= 65),
6     senior_percentage = mean(age >= 65) * 100
7   )
```

```
# A tibble: 1 × 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)?

```
1 # Count young adults
2 age_sample %>%
3   summarise(
4     young_adults = sum(age <= 30),
5     young_adult_pct = mean(age <= 30) * 100
6   )
```

```
# A tibble: 1 × 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

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

OR (|): Either condition can be TRUE

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

Example of multiple conditions with real data

```
1 # Complex conditions
2 age_sample %>%
3   mutate(
4     working_age = age >= 18 & age < 65,
5     voting_age = age >= 18
6   ) %>%
7   summarise(
8     working_age_count = sum(working_age),
9     voting_age_count = sum(voting_age)
10  )
```

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


Putting This All Together

Logical Operations in mutate()

Creating TRUE/FALSE variables:

```
1 mutate(  
2   is_competitive = vote_margin < 0.05,      # TRUE if close race  
3   high_turnout = turnout > 0.7,           # TRUE if high turnout  
4   experienced = years_in_office >= 4,      # TRUE if experienced  
5   young_high_turnout = age < 30 & turnout > 0.7, # Young AND high turnout  
6   major_party_experienced = (party == "Democrat" | party == "Republican") & years_in_office >= 4 # Major party AND experienced  
7 )
```

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 if someone is young or not, we can create a variable that calls them “Young” or “Old”.

For simple binary decisions:

```
1 mutate(  
2   age_group = if_else(age >= 65, "Senior", "Non-Senior"),  
3   result = if_else(vote_share > 0.5, "Won", "Lost"),  
4   income_level = if_else(income >= 50000, "High", "Low")  
5 )
```

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?

```
1  
2 mutate(  
3   age_category = case_when(  
4     age < 30 ~ "Young",  
5     age < 50 ~ "Middle-aged",  
6     age < 70 ~ "Older",  
7     TRUE ~ "Senior" # catch-all for age >= 70  
8   )  
9 )
```

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:

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

Use case_when() for 3+ categories:

```
1 party_group = case_when(  
2   party == "Democrat" ~ "Democrat",  
3   party == "Republican" ~ "Republican",  
4   party == "Independent" ~ "Independent",  
5   TRUE ~ "Other"  
6 )
```

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

Common case_when() Patterns

Age groups:

```
1 age_group = case_when(  
2   age < 25 ~ "18-24",  
3   age < 35 ~ "25-34",  
4   age < 50 ~ "35-49",  
5   age < 65 ~ "50-64",  
6   TRUE ~ "65+"  
7 )
```

Income brackets:

```
1 income_bracket = case_when(  
2   income < 25000 ~ "Low",  
3   income < 75000 ~ "Middle",  
4   TRUE ~ "High"  
5 )
```

Performance ratings:

```
1 rating = case_when(  
2   score >= 90 ~ "Excellent",  
3   score >= 80 ~ "Good",  
4   score >= 70 ~ "Fair",  
5   TRUE ~ "Poor"  
6 )
```


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:

```
1 congress_data %>%
2   group_by(party_code) %>%
3   summarise(
4     count = n(),
5     mean_ideology = mean(nominate_dim1, na.rm = TRUE),
6     median_ideology = median(nominate_dim1, na.rm = TRUE),
7     percentage = n() / nrow(congress_data) * 100
8   )
```

Results:

```
# A tibble: 3 × 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 scores, and percentage of total for each
party."
```

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 = TRUE for the mean and median
calculations?"
```

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:

```
1 voter_data %>%
2   group_by(education_level) %>%
3   summarise(
4     total_respondents = n(),
5     voters = sum(voted == "Yes"),
6     non_voters = sum(voted == "No"),
7     turnout_rate = mean(voted == "Yes"),
8     median_age = median(age)
9   )
```

Results:

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

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 voted, number who didn't vote, turnout
rate, and median age."
```

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') calculates the turnout rate. Why does this
work?"
```

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 × 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

```
1 income_demo %>%
2   mutate(
3     low_income = annual_income <= 30000,      # <= operator
4     high_income = annual_income >= 100000,    # >= operator
5     middle_income = annual_income > 30000 & annual_income < 100000, # Combined
6     not_wealthy = annual_income != 200000,   # != operator
7     exactly_median = annual_income == 60000  # == operator (approximately)
8   ) %>%
9   select(person_id, annual_income, low_income, high_income, middle_income)
```

```
# A tibble: 12 × 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

```
1 income_demo %>%
2   summarise(
3     total_people = n(),
4     low_income_count = sum(annual_income <= 30000),
5     high_income_count = sum(annual_income >= 100000),
6     middle_income_count = sum(annual_income > 30000 & annual_income < 100000),
7     median_income = median(annual_income),
8     avg_income = mean(annual_income)
9   )
```

```
# A tibble: 1 × 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 logical variables that categorize people as: low income ($\leq 30K$), high income ($\geq 100K$), middle income (between 30K-100K), and identify anyone earning exactly the median. Then summarize the counts. The data looks like this:

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:

- \leq and \geq operators for thresholds
- Combined conditions with & operator
- \neq 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 count TRUE values? What's the difference between & and | in logical conditions?"

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:

```
1 campaign_data %>%
2   group_by(office_type, party) %>%
3   summarise(
4     num_candidates = n(),
5     avg_raised = mean(total_raised),
6     median_spent = median(total_spent),
7     win_rate = mean(outcome == "Won"),
8     total_spending = sum(total_spent),
9     .groups = "drop"
10  )
```

Results:

```
# A tibble: 6 × 7
  office_type party      num_candidates avg_raised median_spent
  <chr>      <chr>          <int>      <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
```

AI Prompts for Campaign Analysis

Prompt to Create Code:

```
"I have campaign data with office_type, party, total_raised,
total_spent, and outcome variables. Group by office type and party
to show: candidate count, average raised, median spent, win rate,
and total spending. The data look like this: Rows: 12
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 median be better than mean for spending
analysis?"
```

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:

```
1 data %>%  
2   pull(column_name)  
3  
4 # Or extract calculated values  
5 data %>%  
6   summarise(mean_value = mean(variable)) %>%  
7   pull(mean_value)
```

Example: Using pull() for Calculations

```
1 # Extract median income for threshold calculation
2 median_income <- survey_data %>%
3   summarise(median_inc = median(income)) %>%
4   pull(median_inc)
5
6 median_income
```

```
[1] 51198.99
```

```
1 # Use extracted value in new calculation
2 survey_data %>%
3   mutate(
4     above_median = income > median_income,
5     income_ratio = income / median_income
6   ) %>%
7   select(respondent_id, income, above_median, income_ratio) %>%
8   slice_head(n = 5)
```

```
# A tibble: 5 × 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?

```
1 # Create sample campaign data with more variables
2 campaign_complex <- tibble(
3   candidate = paste("Candidate", 1:10),
4   vote_share = c(0.52, 0.48, 0.67, 0.33, 0.51, 0.49, 0.78, 0.22, 0.55, 0.45),
5   total_spent = c(150000, 145000, 300000, 100000, 180000, 175000, 500000, 80000, 200000, 190000),
6   incumbent = c(TRUE, FALSE, TRUE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE)
7 )
8
9 campaign_complex
```

```
# A tibble: 10 × 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()

```
1 campaign_complex %>%
2   mutate(
3     # Competitiveness based on vote margin
4     competitiveness = case_when(
5       abs(vote_share - 0.5) <= 0.02 ~ "Extremely Close", # Within 2%
6       abs(vote_share - 0.5) <= 0.05 ~ "Competitive",      # Within 5%
7       abs(vote_share - 0.5) <= 0.10 ~ "Somewhat Safe",    # Within 10%
8       TRUE ~ "Safe"                                     # More than 10%
9     ),
10
11    # Spending level
12    spending_level = case_when(
13      total_spent >= 400000 ~ "High Budget",
14      total_spent >= 200000 ~ "Medium Budget",
15      total_spent >= 100000 ~ "Low Budget",
16      TRUE ~ "Minimal Budget"
17    ),
18
19    # Advantage type using multiple conditions
20    advantage = case_when(
21      incumbent == TRUE & total_spent >= 200000 ~ "Incumbent + Money",
22      incumbent == TRUE & total_spent < 200000 ~ "Incumbent Only",
23      incumbent == FALSE & total_spent >= 200000 ~ "Money Only",
24      TRUE ~ "Neither Advantage"
25    )
26  ) %>%
27  select(candidate, vote_share, competitiveness, spending_level, advantage)
```

```
# A tibble: 10 × 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:

```
1 polling_data %>%
2   group_by(region, party_id) %>%
3   summarise(
4     sample_size = n(),
5     avg_approval = mean(approval_rating),
6     approval_range = max(approval_rating) - min(approval_rating),
7     high_approval_pct = mean(approval_rating > 50),
8     response_rate = mean(!is.na(approval_rating)),
9     .groups = "drop"
10  )
```

```
# A tibble: 12 × 7
  region party_id sample_size avg_approval approval_range
  <chr>   <chr>      <int>      <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
```

AI Prompt to Create:

```
"I have polling data with region, party_id, and approval_rating
variables. Create a summary by region and party showing: sample
size, average approval, approval range, percentage with high
approval (>50), and response rate with tidyverse. Here is what the
data look like <insert glimpse>"
```

AI Prompt to Explain:

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

```
Explain how mean(approval_rating > 50) calculates the percentage
with high approval, and how mean(!is.na(approval_rating))
calculates response rate."
```

Examples: Mutate Operations

Example 5: Creating Campaign Efficiency Metrics

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

```
1 campaign_data %>%
2   mutate(
3     spending_efficiency = vote_share / (total_spent / 1000),
4     fundraising_ratio = total_raised / total_spent,
5     cost_per_vote = total_spent / total_votes,
6     budget_category = case_when(
7       total_spent < 100000 ~ "Low Budget",
8       total_spent < 500000 ~ "Medium Budget",
9       TRUE ~ "High Budget"
10    )
11  ) %>%
12  select(candidate_name, spending_efficiency, fundraising_ratio,
13         cost_per_vote, budget_category) %>%
14  slice_head(n = 5)
```

```
# A tibble: 5 × 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 variables for: spending efficiency(vote share per \$1000 spent), fundraising ratio, cost per vote, and budget categories with tidyverse. Here is what the data look like <insert glimpse>"

AI Prompt to Explain:

"Explain each mutate calculation. Why divide total_spent by 1000 for efficiency? How does case_when work for creating budget categories?"

Example 6: Demographic Recoding and Analysis

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

```
1 survey_data %>%
2   mutate(
3     age_group = case_when(
4       age < 30 ~ "18-29",
5       age < 45 ~ "30-44",
6       age < 65 ~ "45-64",
7       TRUE ~ "65+"
8     ),
9     income_quintile = ntile(income, 5),
10    education_years_std = (education_years - mean(education_years)) / sd(education_years),
11    high_earner = income > median(income),
12    political_engagement = (political_interest + civic_participation) / 2
13  ) %>%
14  select(respondent_id, age_group, income_quintile, education_years_std,
15         high_earner, political_engagement) %>%
16  slice_head(n = 5)
```

```
# A tibble: 5 × 6
  respondent_id age_group income_quintile education_years_std high_earner
      <int>    <chr>          <int>          <dbl>
1         1 18-29             4          0.845
2         2 30-44             1          0.583
3         3 45-64             4         -1.25
4         4 65+              5         -0.201
5         5 65+              5         -0.724
# i 1 more variable: political_engagement <dbl>
```

AI Prompt to Create:

"I have survey data with age, income, education_years, political_interest, and civic_participation. Create: age groups, income quintiles, standardized education, high earner indicator, and political engagement score with tidyverse. Here is what the data look like <insert glimpse>"

AI Prompt to Explain:

"Explain how ntile() creates quintiles, what standardization does to education_years, and how the political engagement score is calculated."

Example 7: Electoral Performance Metrics

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

```
1 election_data %>%
2   mutate(
3     vote_margin = abs(dem_votes - rep_votes),
4     competitive_race = vote_margin < (total_votes * 0.05),
5     dem_vote_share = dem_votes / total_votes,
6     turnout_rate = total_votes / eligible_voters,
7     incumbent_advantage = if_else(incumbent_party == winner, 1, 0)
8     upset_victory = (dem_vote_share > 0.5 & dem_expected_share < 0
9                     (dem_vote_share < 0.5 & dem_expected_share > 0.
10  ) %>%
11  select(district, competitive_race, dem_vote_share, turnout_rate,
12         incumbent_advantage, upset_victory) %>%
13  slice_head(n = 5)
```

```
# A tibble: 5 × 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_party, winner, and expected vote shares. Create metrics for: vote margin, competitive races, vote shares, turnout, incumbent advantage, and upsets with tidyverse. Here is what the data look like <insert glimpse>"

AI Prompt to Explain:

"Explain the logic for identifying competitive races and upset victories. How do the logical operators & and | work in the upset_victory calculation?"

Example 8: Policy Position Scaling

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

```
1 legislator_data %>%
2   mutate(
3     economic_conservatism = (tax_cuts_score + deregulation_score +
4                             anti_union_score) / 3,
5     social_conservatism = (abortion_restrictions + traditional_mar
6                             religious_freedom) / 3,
7     overall_ideology = (economic_conservatism + social_conservatism) / 2,
8     ideology_category = case_when(
9       overall_ideology < 3 ~ "Liberal",
10      overall_ideology < 7 ~ "Moderate",
11      TRUE ~ "Conservative"
12    ),
13     extreme_position = overall_ideology < 2 | overall_ideology > 8,
14     policy_consistency = abs(economic_conservatism - social_conservatism) / 2
15   ) %>%
16   select(legislator_name, economic_conservatism, social_conservatism,
17          overall_ideology, ideology_category, policy_consistency) %>%
18   slice_head(n = 5)
```

AI Prompt to Create:

"I have legislator data with multiple policy voting scores (1-10 scale). Create composite scores for economic and social conservatism, overall ideology, ideology categories, extreme position indicators, and policy consistency measures with tidyverse. Here is what the data look like <insert glimpse>"

AI Prompt to Explain:

"Explain how averaging creates composite scores and how policy_consistency measures the difference between economic and social positions. Why use abs()?"

```
# A tibble: 5 × 6
  legislator_name economic_conservatism social_conservatism
  <chr>           <dbl>           <dbl>
1 Rep. Brooks      5             7
2 Rep. Carter      3             3.67
3 Rep. Harris      7.33          5.33
4 Sen. Garcia      4.67          5
5 Rep. Brooks      2.67          5
# i 2 more variables: ideology_category <chr>,
# policy_consistency <dbl>
```

Examples: Variable Recoding

Example 9: Income Bracket Recoding

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

```
1 income_data %>%
2   mutate(
3     income_bracket = case_when(
4       household_income < 25000 ~ "Low Income",
5       household_income < 50000 ~ "Lower Middle",
6       household_income < 75000 ~ "Middle Class",
7       household_income < 100000 ~ "Upper Middle",
8       household_income < 150000 ~ "High Income",
9       TRUE ~ "Very High Income"
10    ),
11    poverty_status = case_when(
12      household_income < 15000 ~ "Below Poverty",
13      household_income < 30000 ~ "Near Poverty",
14      TRUE ~ "Above Poverty"
15    ),
16    economic_security = if_else(
17      household_income > 75000 & savings > 10000,
18      "Secure", "Insecure"
19    )
20  ) %>%
21  count(income_bracket, poverty_status, economic_security) %>%
22  slice_head(n = 10)
```

```
# A tibble: 10 × 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?

```
1 state_data %>%
2   mutate(
3     political_region = case_when(
4       state %in% c("CA", "OR", "WA", "HI") ~ "Pacific Coast",
5       state %in% c("NY", "NJ", "CT", "MA", "RI", "VT", "NH", "ME")
6       state %in% c("TX", "FL", "GA", "NC", "SC", "VA", "TN", "KY",
7         "AL", "MS", "LA", "AR", "OK") ~ "South",
8       state %in% c("IL", "IN", "OH", "MI", "WI", "MN", "IA", "MO",
9         "ND", "SD", "NE", "KS") ~ "Midwest",
10      state %in% c("CO", "UT", "NV", "AZ", "NM", "WY", "MT", "ID")
11      TRUE ~ "Other"
12    ),
13    battleground_state = state %in% c("PA", "MI", "WI", "AZ", "GA")
14    electoral_votes_category = case_when(
15      electoral_votes >= 20 ~ "Large State",
16      electoral_votes >= 10 ~ "Medium State",
17      TRUE ~ "Small State"
18    )
19  ) %>%
20  count(political_region, battleground_state, electoral_votes_cate
21  slice_head(n = 10)
```

AI Prompt to Create:

"I have state-level data with state abbreviations and electoral_votes. Create political regions (5 main regions), identify battleground states, and categorize states by electoral vote count with tidyverse. Here is what the data look like <insert glimpse>"

AI Prompt to Explain:

"Explain how %in% works to check if a state belongs to a vector of states. Why is this more efficient than multiple | (OR) conditions?"

```
# A tibble: 10 × 4
  political_region battleground_state electoral_votes_category
  <chr>           <lgl>           <chr>
1 Midwest        FALSE          Medium State
2 Midwest        FALSE          Small State
3 Midwest        TRUE           Medium State
4 Mountain West  FALSE          Medium State
5 Mountain West  FALSE          Small State
6 Mountain West  TRUE           Medium State
```

Example 11: Survey Response Cleaning

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

```
1 survey_clean_data %>%
2   mutate(
3     party_clean = case_when(
4       str_detect(party_id, regex("dem|liberal", ignore_case = TRUE
5       str_detect(party_id, regex("rep|conserv|gop", ignore_case =
6       str_detect(party_id, regex("ind|none|other", ignore_case = T
7       is.na(party_id) ~ "Unknown",
8       TRUE ~ "Other"
9     ),
10    approval_numeric = case_when(
11      approval %in% c("Strongly Approve", "Approve") ~ 1,
12      approval %in% c("Strongly Disapprove", "Disapprove") ~ 0,
13      approval == "Neither" ~ 0.5,
14      TRUE ~ NA_real_
15    ),
16    education_level = case_when(
17      education < 12 ~ "Less than High School",
18      education == 12 ~ "High School Graduate",
19      education %in% 13:15 ~ "Some College",
20      education == 16 ~ "College Graduate",
21      education > 16 ~ "Post-Graduate",
22      TRUE ~ "Unknown"
23    )
24  ) %>%
25  count(party_clean, education_level) %>%
26  slice_head(n = 10)
```

```
# A tibble: 10 × 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). Clean party affiliations using text detection, convert approval to numeric (1=approve, 0=disapprove, 0.5=neutral), and create education levels with tidyverse. Here is what the data look like <insert glimpse>"

AI Prompt to Explain:

"Explain how str_detect with regex works for party identification and why ignore_case = TRUE is important. How does NA_real_ differ from just NA?"

Example 12: Policy Position Recoding

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

```
1 voting_data %>%
2   mutate(
3     healthcare_position = case_when(
4       healthcare_vote == "Yes" & medicare_expansion == "Yes" ~ "Pro-Envir",
5       healthcare_vote == "Yes" & medicare_expansion == "No" ~ "Mod",
6       healthcare_vote == "No" & medicare_expansion == "No" ~ "Cons",
7       healthcare_vote == "Abstain" | medicare_expansion == "Abstain" ~ "Mixed",
8     ),
9     environmental_stance = case_when(
10      climate_action == "Yes" & clean_energy == "Yes" ~ "Pro-Envir",
11      climate_action == "No" & clean_energy == "No" ~ "Anti-Envir",
12      TRUE ~ "Mixed Environment"
13    ),
14     fiscal_ideology = case_when(
15      tax_cuts == "Yes" & spending_cuts == "Yes" ~ "Fiscal Conserv",
16      tax_cuts == "No" & spending_cuts == "No" ~ "Fiscal Liberal",
17      tax_cuts == "Yes" & spending_cuts == "No" ~ "Tax Cutter",
18      tax_cuts == "No" & spending_cuts == "Yes" ~ "Spending Cutter",
19      TRUE ~ "Fiscal Moderate"
20    )
21   ) %>%
22   count(healthcare_position, environmental_stance, fiscal_ideology)
23   slice_head(n = 10)
```

AI Prompt to Create:

"I have legislator voting data with Yes/No/Abstain votes on healthcare, medicare_expansion, climate_action, clean_energy, tax_cuts, and spending_cuts. Create position categories combining related votes into meaningful ideological groups with tidyverse. Here is what the data look like <insert glimpse>"

AI Prompt to Explain:

"Explain how combining multiple vote conditions creates more nuanced categories than single votes. Why use 'Mixed' and 'Unclear' categories instead of forcing binary classifications?"

```
# A tibble: 10 × 4
  healthcare_position environmental_stance fiscal_ideology
  <chr>              <chr>              <chr>
1 Conservative      Anti-Environment    Fiscal Conservative
2 Conservative      Anti-Environment    Fiscal Liberal
3 Conservative      Anti-Environment    Fiscal Moderate
4 Conservative      Anti-Environment    Spending Cutter
5 Conservative      Mixed Environment    Fiscal Conservative
6 Conservative      Mixed Environment    Fiscal Liberal
```

Complex Examples

Complex Example 1: Comprehensive Electoral Analysis

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

```
1 electoral_analysis <- complex_election_data %>%
2   # Filter to competitive races only
3   filter(vote_margin < 0.1) %>%
4
5   # Create new variables
6   mutate(
7     spending_advantage = (candidate_spending - opponent_spending) /
8                           (candidate_spending + opponent_spending),
9     experience_level = case_when(
10      prior_office == "None" ~ "Newcomer",
11      prior_office %in% c("Local", "County") ~ "Local Experience",
12      prior_office %in% c("State", "Federal") ~ "Experienced",
13      TRUE ~ "Other"
14    ),
15     demographic_match = case_when(
16      candidate_race == district_majority_race ~ "Matches District",
17      TRUE ~ "Different from District"
18    ),
19     upset_potential = if_else(
20      polling_deficit > 5 & spending_advantage > 0.2,
21      "High", "Low"
22    )
23   ) %>%
24
25   # Group analysis by region and office type
26   group_by(region, office_type, experience_level) %>%
27   summarise(
28     total_races = n()
```

```
# A tibble: 8 x 7
  region    office_type experience_level total_races
<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
```

AI Prompt to Create:

"I have comprehensive election data with candidate info, spending, demographics, and results. Create an analysis that: 1) filters to competitive races, 2) creates spending advantage and experience metrics, 3) groups by region/office/experience, 4) calculates win rates and other success metrics with tidyverse. Here is what the data look like <insert glimpse>"

AI Prompt to Explain:

"Explain this complex analysis workflow. How do the filter, mutate, group_by, and summarise functions work together? Why use .groups = 'drop' and what does na.rm = TRUE do in the mean calculations?"

Complex Example 2: Public Opinion Trend Analysis

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

```
1 opinion_trends <- complex_polling_data %>%
2   # Create time-based variables
3   mutate(
4     year_month = floor_date(survey_date, "month"),
5     approval_binary = if_else(approval_rating > 50, 1, 0),
6     issue_priority = case_when(
7       top_issue %in% c("Economy") ~ "Economic Issues",
8       top_issue %in% c("Healthcare") ~ "Social Issues",
9       top_issue %in% c("Immigration", "Crime") ~ "Security Issues",
10      top_issue %in% c("Environment") ~ "Environmental Issues",
11      TRUE ~ "Other Issues"
12    ),
13    age_generation = case_when(
14      age < 30 ~ "Gen Z",
15      age < 45 ~ "Millennial",
16      age < 65 ~ "Gen X",
17      TRUE ~ "Boomer+"
18    )
19  ) %>%
20
21  # Group by demographics and time
22  group_by(year_month, age_generation, party_id) %>%
23  summarise(
24    sample_size = n(),
25    avg_approval = mean(approval_rating),
26    approval_rate = mean(approval_binary),
27    economic_optimism = mean(economic_outlook == "Positive"),
28    groups = "keep"
```

# A tibble: 8 × 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				

AI Prompt to Create:

"I have longitudinal polling data with dates, demographics, approval ratings, and issue priorities. Create an analysis that: 1) creates time and demographic categories, 2) groups by multiple variables for monthly trends, 3) calculates trend changes with lag functions with tidyverse. Here is what the data look like <insert glimpse>"

AI Prompt to Explain:

"Explain this complex analysis workflow. How do multiple group_by operations work? What does lag() do for trend calculation? How does the combination of filter, mutate, group_by, and summarise create this analysis?"

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_share, and outcome. I want to create efficiency metrics including cost per vote and spending per percentage point of vote share with tidyverse. Here is what the data look like <insert glimpse>. Show me the mutate code and explain how each calculation works."
```

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()`



Speaker notes