9 Functions

25 Functions

1. Why Write Functions
    - functions avoid repeating the same code
    - they make code cleaner, shorter, and less error-prone
    - they make analyses easier to update and maintain
2. Structure Of A Function
    - define functions with function()
    - functions take inputs called arguments
    - the last line in the body becomes the return value
3. Naming And Style
    - good function names describe the action performed
    - snake_case is the preferred style
    - clear argument names improve readability
4. Function Arguments
    - default arguments make functions flexible
    - arguments allow a single function to handle multiple cases
    - argument order matters unless you name them explicitly
5. Returning Values
    - R returns the last evaluated expression
    - return() can be used for clarity but is not required
    - each function should do one clear thing
6. Debugging Functions
    - print statements can help diagnose issues
    - using smaller helper functions simplifies debugging
    - checking inputs early prevents downstream errors

13.1 if-else

1. Purpose Of If-Else
    - if-else controls the flow of a program
    - code runs differently depending on logical conditions
    - useful for choosing between alternative actions
2. Basic If-Else Structure
    - if() runs code only when a condition is true
    - else() provides an alternative when the condition is false
    - conditions must evaluate to TRUE or FALSE
3. Chained Conditions
    - else if() handles multiple possible branches
    - conditions are evaluated in order

    - the first true condition determines which block runs
4. Vectorized Alternatives
    - ifelse() applies conditional logic element-wise
    - ifelse() returns a vector the same length as the input
    - useful inside mutate() or other data manipulation steps
5. Common Pitfalls
    - using == NA instead of is.na() causes errors
    - conditions returning NA lead to unexpected behavior
    - forgetting braces can cause ambiguous code