Week 1, Class 2

Introduction to R and Data Frames

Sean Westwood

Today’s Learning Objectives

By the End of Class You Will:

- Understand basic R operations with variables and vectors
- Know how to create and examine data frames
- Load CSV files using `read_csv()`
- Use essential tidyverse functions for data exploration
- Apply AI assistance effectively for R programming

The AI Revolution in Data Analysis

What’s Changed Recently

Traditional Workflow:

❓ Research Question → 📚 Learn Programming → 💻 Write Code → 🐛 Debug Errors → 📊 Analyze Results → 🧠 Interpret Findings

AI-Assisted Approach:

❓ Research Question → 🤖 Prompt AI → 👀 Review Code → ▶️ Run Analysis → 📊 Interpret Results → ✅ Check Logic

Key Difference

- We focus on understanding and interpretation, not memorizing syntax
- AI handles the programming details
- You verify the logic and meaning

Working with Objects in R

Basic Variable Assignment

R stores information as **objects**. We create objects using the assignment operator `<-`:

```r
# Store a number
electoral_votes_to_win <- 270

# Store text
candidate_name <- "Biden"

# Store the result of a calculation
margin_2020 <- (81283501 - 74223975) / 155507476
```

Variable Types

R automatically determines the type of data you’re storing:

- **Numeric**: Numbers for calculations (`270`, `3.14`, `-5.2`)
  - Used for vote counts, percentages, ages, income data
- **Character**: Text enclosed in quotes (`"Biden"`, `"Democratic"`, `"2024"`)
  - Used for names, party affiliations, state labels, survey responses
- **Logical**: True/false values (`TRUE`, `FALSE`)
  - Used for yes/no questions, whether conditions are met

```r
# These are different!
number_value <- 270
text_value <- "270"
```

Working with Vectors

Creating Vectors

We often want to store multiple values in a single variable. For example, we might want to store the electoral votes for swing states in 2020. We can do this by creating a vector.

> Think of them as lists of values.

Vectors are one-dimensional collections of related values. We create them using `c()`.

```r
# Electoral votes for swing states in 2020
swing_states <- c("Pennsylvania", "Michigan", "Wisconsin",
                  "Arizona", "Georgia", "Nevada")

electoral_votes <- c(20, 16, 10, 11, 16, 6)
```

Basic Vector Operations

How many swing states?

```r
length(swing_states)
```

```
[1] 6
```

Total electoral votes in swing states?

```r
sum(electoral_votes)
```

```
[1] 79
```

What is the maximum number of electoral votes in a swing state?

```r
max(electoral_votes)
```

```
[1] 20
```

Mathematical Operations

Vectors make calculations across multiple values easy:

What if we want to increase the electoral votes in each swing state by 5%?

```r
# If turnout increased by 5% in each state
new_turnout <- electoral_votes * 1.05

new_turnout
```

```
[1] 21.00 16.80 10.50 11.55 16.80  6.30
```

Essential Functions for Working with Data

Basic Summary Functions

```r
# Presidential approval ratings (simulated)
approval_ratings <- c(45, 42, 48, 51, 44, 47, 43, 49, 46, 50)

# Central tendency
mean(approval_ratings)    # Average approval
```

```
[1] 46.5
```

```r
median(approval_ratings)  # Middle value
```

```
[1] 46.5
```

```r
# Spread
range(approval_ratings)   # Min and max
```

```
[1] 42 51
```

```r
length(approval_ratings)  # Number of observations
```

```
[1] 10
```

The `round()` Function

Make numbers readable:

```r
# Calculate Biden's 2020 vote percentage
biden_percentage <- 81283501 / 155507476

# Round to 2 decimal places
round(biden_percentage, 2)
```

```
[1] 0.52
```

```r
# Round mean approval to 1 decimal place
round(mean(approval_ratings), 1)
```

```
[1] 46.5
```

Introduction to Data Frames

What Are Data Frames?

Data frames are like spreadsheets that store data in rows and columns.
Key features:

- **Rows**: Observations (countries, voters, elections)
- **Columns**: Variables (population, party, vote share)
- **Names**: Each column has a descriptive name

Loading Data with Tidyverse

Setting Up: Why Tidyverse?

The **tidyverse** is a collection of R packages designed for data science that share a common philosophy and grammar.

Why use tidyverse over base R?

- **Consistent syntax**: Functions work similarly across packages
- **Readable code**: Operations read like English sentences
- **Better error messages**: More helpful when things go wrong
- **Modern approach**: Designed for contemporary data analysis workflows

First, we load the tidyverse package:

```r
library(tidyverse)
```

Loading CSV Files

What is a CSV file?

CSV stands for “Comma-Separated Values” - it’s a simple text format where:

- Each row represents one observation
- Columns are separated by commas
- First row usually contains variable names

Other file formats: R can load Excel files (.xlsx), SPSS files (.sav), Stata files (.dta), and many others, but for this course, all our data will be in CSV format.

To load CSV files, we use `read_csv()`. This is a function from the tidyverse package that reads in a CSV file and returns a data frame.

```r
# Load UN population data
UNpop <- read_csv("../../data/UNpop.csv")
```

Examining Data Frames

Essential Exploration Functions

```r
# Quick overview of structure (what you will send to AI to help you)
glimpse(UNpop)
```

```
Rows: 7
Columns: 2
$ year      <dbl> 1950, 1960, 1970, 1980, 1990, 2000, 2010
$ world.pop <dbl> 2525779, 3026003, 3691173, 4449049, 5320817, 6127700, 6916183
```

Understanding the Output

`glimpse()` shows:

- Number of rows and columns
- Column names and types
- First few values in each column

This gives you a complete picture of your data structure.

Understanding Data Types

When we use `glimpse()`, we see information about each column’s data type. Come common data types:

- `<dbl>` means “double precision floating point number” (a decimal number)
  - Examples: population counts, vote shares, income amounts
- `<chr>` means “character” (text data)
  - Used for text
  - Examples: candidate names, party affiliations, state names
- `<int>` means “integer” (whole numbers)
  - Examples: number of votes, year, district numbers
- `<lgl>` means “logical” (TRUE/FALSE values)
  - Examples: incumbent status, ballot measure results
- `<date>` means date values (a date)
  - Examples: election dates, survey dates

More Essential Exploration Functions: Head

The `head()` function shows the first few rows of your data frame, which is useful for:

- Seeing what your actual data looks like (not just the structure)
- Checking if data loaded correctly
- Understanding the format and content of each column
- Getting a quick preview before doing analysis

Note: There's also a `tail()` function that shows the last few rows, which can be helpful for checking if your data is complete or seeing the most recent observations.

```r
head(UNpop)
```

```
# A tibble: 6 × 2
   year world.pop
  <dbl>     <dbl>
1  1950   2525779
2  1960   3026003
3  1970   3691173
4  1980   4449049
5  1990   5320817
6  2000   6127700
```

More Essential Exploration Functions: Summary

The `summary()` function provides descriptive statistics for each column in your data frame:

- For **numeric columns** (like population counts):
    - `Min.`: Minimum value
    - `1st Qu.`: First quartile (25th percentile)
    - `Median`: Middle value (50th percentile)
    - `Mean`: Average value
    - `3rd Qu.`: Third quartile (75th percentile)
    - `Max.`: Maximum value

```r
# Summary statistics
summary(UNpop)
```

```
      year        world.pop      
 Min.   :1950   Min.   :2525779  
 1st Qu.:1965   1st Qu.:3358588  
 Median :1980   Median :4449049  
 Mean   :1980   Mean   :4579529  
 3rd Qu.:1995   3rd Qu.:5724258  
 Max.   :2010   Max.   :6916183  
```

More Essential Exploration Functions: Names

The `names()` function shows the names of the columns in your data frame:

- Useful for:
    - Checking column names
    - Understanding variable labels

```r
# Column names
names(UNpop)
```

```
[1] "year"      "world.pop"
```

Working with Real Data: UN Population

The UNpop Dataset

Source: United Nations population estimates
Time period: 1950-2010 (10-year intervals)
Variables:

- `year`: Year of measurement
- `world.pop`: World population (in thousands)

Simple Calculations

Population growth over time

```r
UNpop$world.pop[7] - UNpop$world.pop[1]  # Growth from 1950 to 2010
```

```
[1] 4390404
```

Average population across all years

```r
# Average population across all years
mean(UNpop$world.pop)
```

```
[1] 4579529
```

Convert from thousands to billions

```r
# Convert from thousands to billions
UNpop$world.pop / 1000000
```

```
[1] 2.525779 3.026003 3.691173 4.449049 5.320817 6.127700 6.916183
```

AI Integration for R Programming

Effective AI Prompts

For syntax help:

> “I have a data frame called ‘leaders’ with columns ‘name’, ‘party’, and ‘years_served’. How do I calculate the average years served using tidyverse functions?”

For error messages:

> “I got this error: ‘Error: could not find function “read_csv”’. What does this mean and how do I fix it? I am using Positron”

What AI Does Well

- Generating correct syntax
- Explaining error messages
- Suggesting appropriate functions
- Checking assumptions and limitations

You Still Must

- Understand the logic behind the code
- Verify results make sense
- Make sure AI is doing what you want it to do

Exercise: Congressional Leadership Data

Create and Analyze

Run this code:

```r
# Create the data frame
congressional_leaders <- data.frame(
  name = c("Pelosi", "Schumer", "McConnell", "McCarthy"),
  party = c("Democratic", "Democratic", "Republican", "Republican"),
  chamber = c("House", "Senate", "Senate", "House"),
  years_served = c(36, 24, 38, 16),
  age = c(82, 72, 80, 58)
)
```

Your tasks:

1. Calculate average years of service
2. Find the oldest leader
3. Count Democrats vs Republicans
4. Which chamber has more experienced leaders on average?

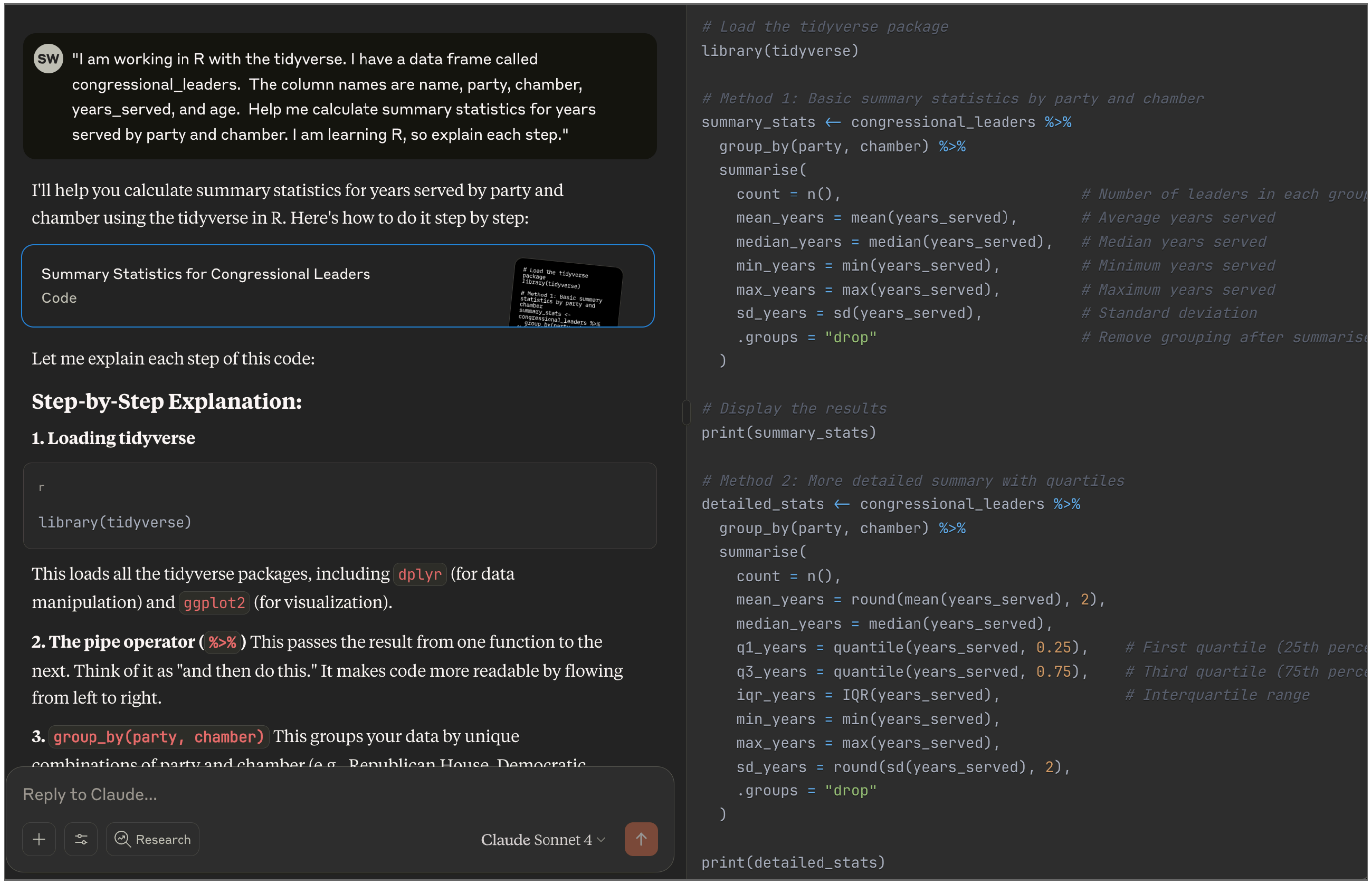
With AI Assistance

Try this prompt:

> “I am working in R with the tidyverse. I have a data frame called congressional_leaders. The column names are name, party, chamber, years_served, and age. Help me calculate summary statistics for years served by party and chamber. I am learning R, so explain each step.”

What you should get back

Something like this:



SW "I am working in R with the tidyverse. I have a data frame called congressional_leaders. The column names are name, party, chamber, years_served, and age. Help me calculate summary statistics for years served by party and chamber. I am learning R, so explain each step."

I'll help you calculate summary statistics for years served by party and chamber using the tidyverse in R. Here's how to do it step by step:

Summary Statistics for Congressional Leaders
Code

Let me explain each step of this code:

Step-by-Step Explanation:

1. Loading tidyverse

```r
library(tidyverse)
```

This loads all the tidyverse packages, including `dplyr` (for data manipulation) and `ggplot2` (for visualization).

2. The pipe operator (`%>%`) This passes the result from one function to the next. Think of it as "and then do this." It makes code more readable by flowing from left to right.

3. `group_by(party, chamber)` This groups your data by unique combinations of party and chamber (e.g., Republican House, Democratic

```
# Load the tidyverse package
library(tidyverse)

# Method 1: Basic summary statistics by party and chamber
summary_stats ← congressional_leaders %>%
  group_by(party, chamber) %>%
  summarise(
    count = n(),                           # Number of leaders in each grou
    mean_years = mean(years_served),       # Average years served
    median_years = median(years_served),   # Median years served
    min_years = min(years_served),         # Minimum years served
    max_years = max(years_served),         # Maximum years served
    sd_years = sd(years_served),           # Standard deviation
    .groups = "drop"                       # Remove grouping after summaris
  )

# Display the results
print(summary_stats)

# Method 2: More detailed summary with quartiles
detailed_stats ← congressional_leaders %>%
  group_by(party, chamber) %>%
  summarise(
    count = n(),
    mean_years = round(mean(years_served), 2),
    median_years = median(years_served),
    q1_years = quantile(years_served, 0.25),    # First quartile (25th perc
    q3_years = quantile(years_served, 0.75),    # Third quartile (75th perc
    iqr_years = IQR(years_served),              # Interquartile range
    min_years = min(years_served),
    max_years = max(years_served),
    sd_years = round(sd(years_served), 2),
    .groups = "drop"
  )

print(detailed_stats)
```

When we run the code

```r
congress_summary <- congressional_leaders %>%
  group_by(party, chamber) %>%
  summarise(
    n                 = n(),
    mean_years        = mean(years_served, na.rm = TRUE),
    median_years      = median(years_served, na.rm = TRUE),
    sd_years          = sd(years_served, na.rm = TRUE),
    min_years         = min(years_served, na.rm = TRUE),
    max_years         = max(years_served, na.rm = TRUE),
    .groups           = "drop"
  )

congress_summary
```

```
# A tibble: 4 × 8
  party      chamber     n mean_years median_years sd_years min_years max_years
  <chr>      <chr>   <int>      <dbl>        <dbl>    <dbl>     <dbl>     <dbl>
1 Democratic House       1         36           36       NA        36        36
2 Democratic Senate      1         24           24       NA        24        24
3 Republican House       1         16           16       NA        16        16
4 Republican Senate      1         38           38       NA        38        38
```

Common Mistakes and Solutions

Error: Object Not Found

```
Error: object 'data' not found
```

Cause: Variable name misspelled or not created yet

Solution: Check spelling! Make sure you are using the correct case.

Error: Could Not Find Function

```
Error: could not find function "read_csv"
```

Cause: Package not loaded

Solution: Run `library(tidyverse)` first

Error: Wrong Data Type

The Problem: When data looks like numbers but is stored as text (character strings), mathematical functions won’t work.

```r
# This won't work as expected
numbers <- c("1", "2", "3")
mean(numbers)  # Error!

# Instead:
numbers <- c(1, 2, 3)
mean(numbers)  # Works!
```

Best Practices

Variable Naming

Good names:

- `electoral_votes`
- `swing_states`
- `approval_ratings`

Poor names:

- `x`, `data`, `stuff`
- `electoralVotes` (inconsistent style)

A full script

```r
# Load packages first
library(tidyverse)

# Load data
UNpop <- read_csv("../../data/UNpop.csv")

# Examine data
glimpse(UNpop)
```

```
Rows: 7
Columns: 2
$ year      <dbl> 1950, 1960, 1970, 1980, 1990, 2000, 2010
$ world.pop <dbl> 2525779, 3026003, 3691173, 4449049, 5320817, 6127700, 6916183
```

```r
# Analyze
mean(UNpop$world.pop)
```

```
[1] 4579529
```

Comments

Comments are lines that start with # and are ignored by R. They are useful for:

- Explaining what the code does
- Adding notes to yourself
- Making the code more readable

```r
# Calculate population growth rate

growth_summary <- UNpop %>%
  slice(c(1, 7)) %>%                              # pick rows 1 and 7
  summarise(
    growth_rate = (last(world.pop) - first(world.pop)) / first(world.pop) * 100
  )

growth_summary
```

```
# A tibble: 1 × 1
  growth_rate
        <dbl>
1        174.
```

Looking Ahead

Next Class Preview

We’ll learn how to:

- Filter and select specific rows and columns
- Sort data in meaningful ways
- Handle missing values
- Work with larger, more complex datasets

Key Concepts to Remember

- R stores information as objects
- Vectors hold multiple related values
- Data frames organize data in rows and columns
- `read_csv()` loads external data files
- AI helps with syntax; you provide the thinking

Questions?

Key takeaway: You don’t need to memorize R syntax. You need to understand data concepts and how to work with AI to implement your ideas.

Speaker notes