

# PS1 Skills Workshop

Gov 51 Section — Week 3



George

Harvard University

February 11, 2026

# Today's Goals

## Everything you need for PS1

1. Fix the `.gitignore` issue — your ACS file is too big for GitHub
2. Master summary statistics — mean, sd, min, max in R
3. Make beautiful tables — `kable()` for professional output
4. Create histograms — `ggplot2` fundamentals
5. Understand  $n-1$  — why we divide by  $n - 1$  for variance

PS1 is due **tomorrow night** (Wednesday 11:59pm)

## Quick Check-In

Where are you with PS1?

- ▷ Haven't started yet?
- ▷ Downloaded the data but stuck?
- ▷ Making progress but have questions?
- ▷ Almost done?

No judgment — that's what section is for!



# Part 1: The `.gitignore` Fix

# The Problem

## GitHub rejects large files

When you try to push your PS1 repository:

```
remote: error: File data/raw/acs2024.csv is 128.45 MB;  
this exceeds GitHub's file size limit of 100.00 MB
```

### Why this happens:

- ▷ The ACS 2024 data file is >100MB
- ▷ GitHub has a hard limit on file sizes
- ▷ Git tried to track the file, now it's stuck

# The Solution

Tell Git to ignore the data file

**Step 1:** Create a `.gitignore` file in your project root

```
# In your project's root folder, create .gitignore  
# Add this line:  
data/raw/acs2024.csv
```

**Step 2:** If you already tried to commit the file:

```
# Remove the file from Git's tracking (keeps the file!)  
git rm --cached data/raw/acs2024.csv  
  
# Now commit the .gitignore  
git add .gitignore  
git commit -m "Add gitignore for large data file"
```

## Verify It Worked

Run `git status` — the data file should NOT appear:

```
$ git status
On branch main
Changes to be committed:
  new file:   .gitignore

# acs2024.csv should NOT be listed here!
```

The file stays on your computer, but Git ignores it.  
Your code still works. GitHub is happy.



## Part 2: Summary Statistics in R

# The Goal

## PS1 Table 1: Clean, Readable, Reproducible

You need to create a summary table like this:

Variable	Mean	Std Dev	Min	Max
Age	41.2	15.3	18	95
Commute Time	27.4	22.1	1	180
Income	52,340	48,210	0	500,000

Your table should be readable *without* looking at your code

# The Basic Functions

## R's built-in statistics

```
# Load your data  
acs <- read.csv("data/raw/acs2024.csv")  
  
# Basic summary functions  
mean(acs$AGE)      # Average  
sd(acs$AGE)        # Standard deviation  
min(acs$AGE)       # Minimum  
max(acs$AGE)       # Maximum
```

Also: `var()`, `sum()`, `length()` for count (n)

# Storing Results

Save statistics to use later

```
# Store results in named objects
avg_age <- mean(acs$AGE)
sd_age <- sd(acs$AGE)
min_age <- min(acs$AGE)
max_age <- max(acs$AGE)

# Now you can use them
cat("The average age is", round(avg_age, 1), "years")
```

## Why store results?

- ▷ Build tables programmatically
- ▷ Use in calculations
- ▷ Reference in your writeup with inline R code

# Watch Out for Missing Values!

The most common error

```
# This will return NA if there are missing values!  
mean(acs$INCTOT) # Returns NA  
  
# Fix: tell R to remove missing values  
mean(acs$INCTOT, na.rm = TRUE) # Works!
```

If your statistics return NA, add `na.rm = TRUE`

# Building a Summary Table

Using tidyverse's `summarize()`

```
library(tidyverse)

# Create summary for commuters
commuters <- acs |> filter(TRANTIME > 0)

summary_stats <- commuters |>
  summarize(
    mean_commute = mean(TRANTIME),
    sd_commute = sd(TRANTIME),
    min_commute = min(TRANTIME),
    max_commute = max(TRANTIME),
    n = n()
  )
```

# Making Tables Beautiful with `kable()`

From data frame to publication-ready

```
library(knitr)

# Turn your summary into a nice table
summary_stats |>
  kable(
    col.names = c("Mean", "Std Dev", "Min", "Max", "N"),
    digits = 2,
    caption = "Summary Statistics for Commute Times"
  )
```

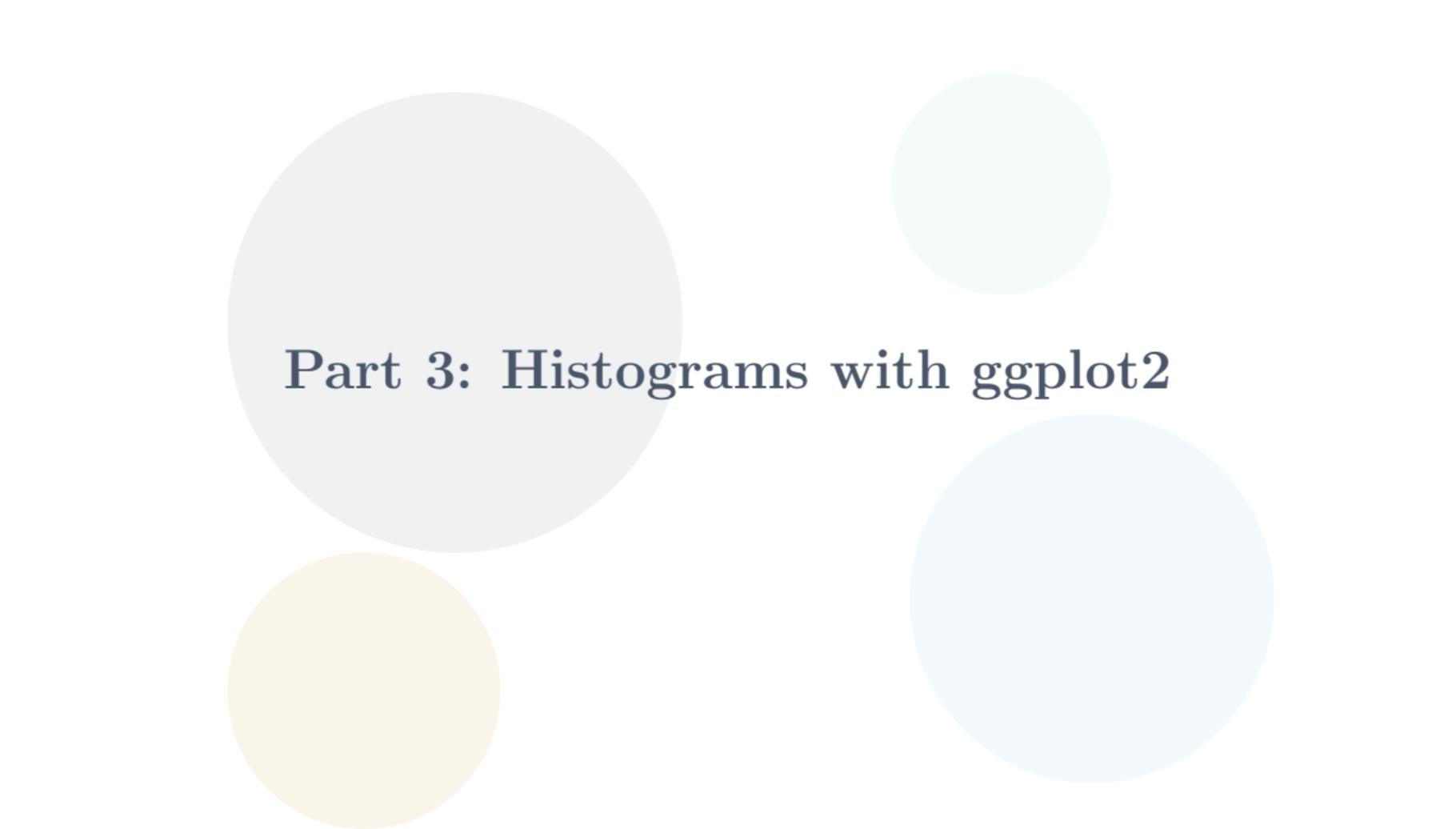
`kable()` is for *display*, not computation.  
Compute first, then make it pretty.

# Your Turn!

## Practice exercise

1. Load the ACS data
2. Filter to commuters only (`TRANTIME > 0`)
3. Calculate the mean and standard deviation of `TRANTIME`
4. Store the results in named objects

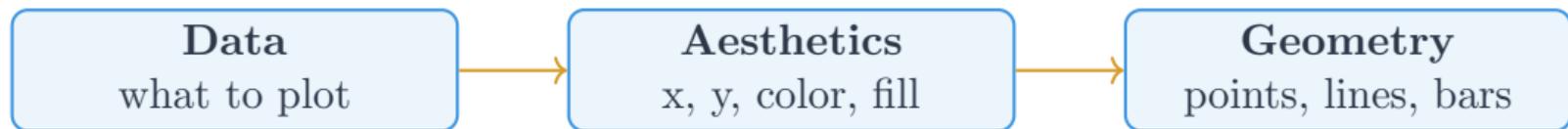
Take 3 minutes — I'll walk around



## Part 3: Histograms with ggplot2

# Anatomy of a ggplot

## Three essential components



```
ggplot(data, aes(...)) + geom_*()
```

# Your First Histogram

## The minimal example

```
library(ggplot2)

ggplot(commuters, aes(x = TRANTIME)) +
  geom_histogram()
```

### What this does:

- ▷ Uses `commuters` data
- ▷ Maps `TRANTIME` to the x-axis
- ▷ Draws a histogram (counts in bins)

# Customizing Your Histogram

Make it informative and attractive

```
ggplot(commuters, aes(x = TRANTIME)) +  
  geom_histogram(  
    binwidth = 5,           # Each bin = 5 minutes  
    fill = "#4299E1",      # Bar color  
    color = "white"        # Bar outline  
  ) +  
  labs(  
    title = "Distribution of Commute Times",  
    x = "Commute Time (minutes)",  
    y = "Number of Workers"  
  ) +  
  theme_minimal()
```

# Saving Your Plot

For your Quarto document

```
# Create the plot
p <- ggplot(commuters, aes(x = TRANTIME)) +
  geom_histogram(binwidth = 5, fill = "#4299E1") +
  labs(title = "Commute Time Distribution") +
  theme_minimal()

# Save to file
ggsave("output/commute_histogram.png", p,
       width = 8, height = 5)
```

In Quarto, the plot renders automatically in code chunks.  
Use `ggsave()` if you need a separate image file.

## Common ggplot Customizations

What you want	How to do it
Change bin width	<code>binwidth = 10</code>
Change bar color	<code>fill = "steelblue"</code>
Add title	<code>labs(title = "...")</code>
Label axes	<code>labs(x = "...", y = "...")</code>
Clean theme	<code>theme_minimal()</code> or <code>theme_bw()</code>
Vertical line	<code>geom_vline(xintercept = 30)</code>

# Your Turn!

## Practice exercise

1. Create a histogram of `TRANTIME`
2. Set `binwidth = 10`
3. Add a title and axis labels
4. Apply `theme_minimal()`

Take 3 minutes

# Why Use Quarto?

Reproducible documents

**What is Quarto?** Modern R Markdown — code + prose in one file.

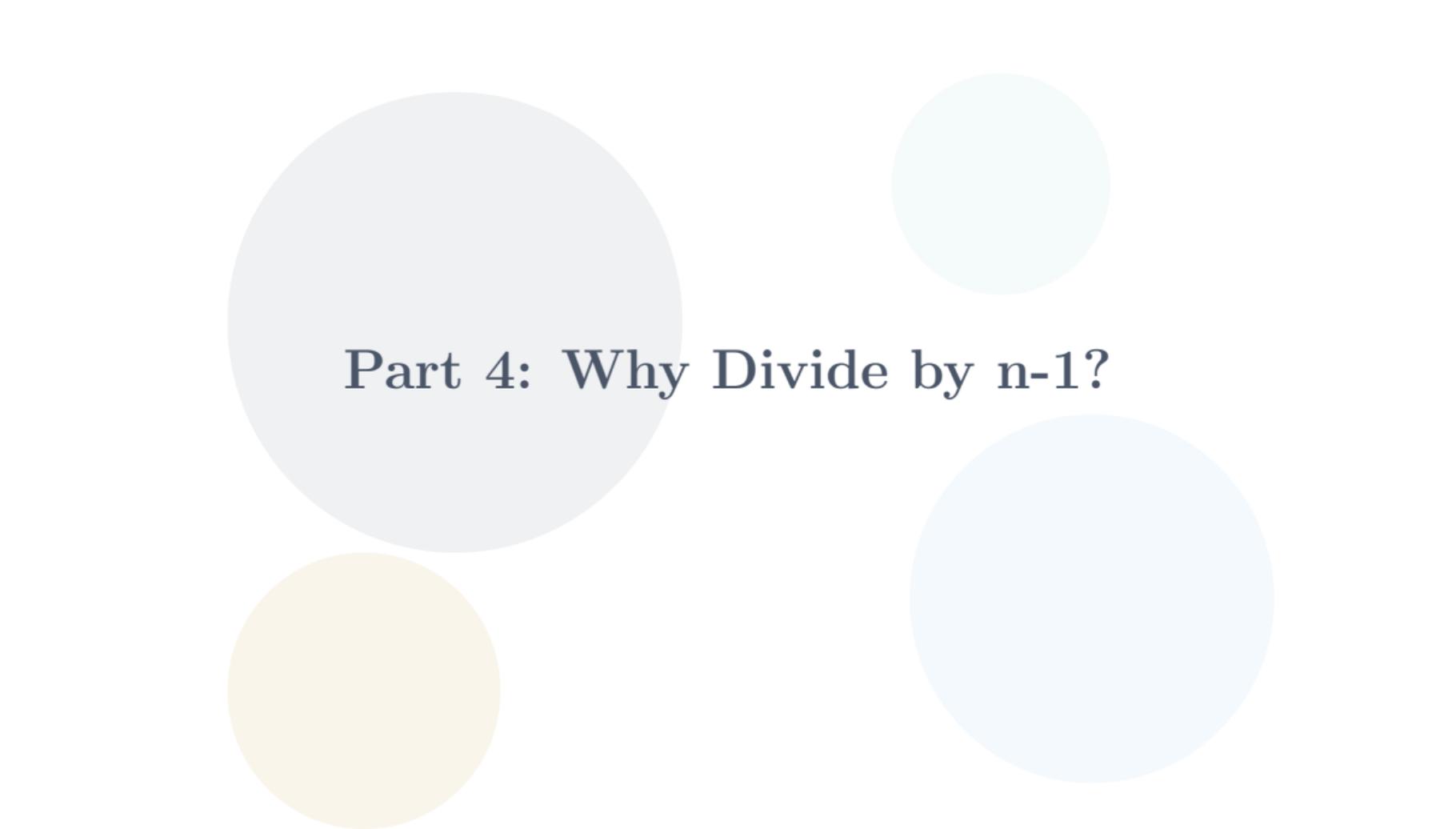
**Show code and output:**

```
```{r}
mean(acs$TRANTIME)
```
```

**Hide code, show output only:**

```
```{r, echo = FALSE}
mean(acs$TRANTIME)
```
```

**Reproducibility** = someone else (or future you) can run your code and get the exact same results



## Part 4: Why Divide by $n-1$ ?

# The Question

A puzzle from the formulas

Sample mean:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Sample variance:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

Why  $n - 1$  and not  $n$ ?

# The Intuition

We're estimating TWO things

## The problem:

- ▷ We want to estimate the *population* variance  $\sigma^2$
- ▷ But we don't know the *population* mean  $\mu$
- ▷ So we use the *sample* mean  $\bar{x}$  instead

## The catch:

- ▷ The sample mean is *too close* to the sample data
- ▷ This makes deviations  $(x_i - \bar{x})$  artificially small
- ▷ Dividing by  $n - 1$  corrects for this bias

# Seeing It With Simulation

Proof by computer

**What we'll do:**

1. Create a population with known variance ( $\sigma^2 = 100$ )
2. Draw thousands of samples
3. Calculate variance using  $1/n$  and  $1/(n - 1)$
4. Compare the averages

If  $1/(n - 1)$  is correct, its average should equal 100.

If  $1/n$  is wrong, its average will be too low.

# The Simulation Code

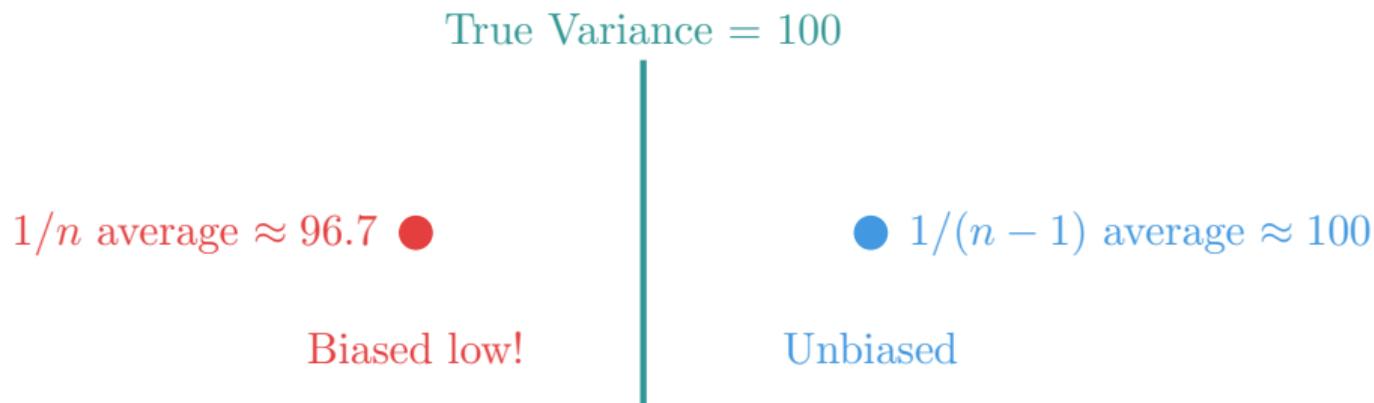
Run this yourself!

```
# True population: variance = 100
pop <- rnorm(100000, mean = 50, sd = 10)

# Take 10,000 samples of size 30
results <- replicate(10000, {
  s <- sample(pop, 30); dev <- (s - mean(s))^2
  c(var_n = sum(dev)/30, var_n1 = sum(dev)/29)
})

mean(results["var_n", ])    # Biased low!
mean(results["var_n1", ])  # Correct!
```

# The Result



Dividing by  $n - 1$  gives an **unbiased** estimate of variance.  
This is called **Bessel's correction**.

# For the Exam

## What you need to know

1. We divide by  $n - 1$  (not  $n$ ) for sample variance
2. This is called the **degrees of freedom correction**
3. It corrects for bias from using the sample mean
4. R's `var()` and `sd()` already use  $n - 1$

You won't need to prove this, but you should know why we don't just divide by  $n$



**Wrapping Up**

# PS1 Checklist

## Everything you need

- `.gitignore` file excludes large data file
- Data loaded and filtered to commuters
- Summary statistics calculated (mean, sd, min, max)
- Table formatted with `kable()`
- Histogram created with `ggplot2`
- Quarto document renders to PDF
- GitHub URL included in document
- Submitted by Wednesday 11:59pm

# Beautiful Output Matters

Not This

```
[1] 27.4  
22.1  
1  
180
```



This

```
Mean      27.4  
Std Dev   22.1  
Min       1  
Max       180
```

Your table should be readable *without* your code.  
This isn't decoration — it's communication.

# Office Hours

## Before the deadline:

- ▷ George's office hours: [time/location]
- ▷ Scott's office hours: [time/location]

Don't wait until Wednesday night!  
If you're stuck, come get help tomorrow.

Questions?

# Truth Over Everything

## The ethics of data science

Data science is **rhetoric** — the art of persuasion.

But the goal is persuasion through **truth**:

- ▷ Not propaganda
- ▷ Not spin
- ▷ Not cherry-picking

**Scientific integrity means:**

- ▷ Report what you find, not what you hoped to find
- ▷ Show the limitations, not just the strengths
- ▷ Let your audience decide voluntarily, based on evidence

Your credibility is your career. Protect it.



Statistics tell stories.  
Your job is to make  
those stories clear.