

Introducing RStudio

Hands-On Learning and Career Readiness in the Social, Behavioral, and Educational Sciences



Njesa Totty

Introducing RStudio

Hands-On Learning and Career Readiness in the Social, Behavioral, and Educational
Sciences

Njesa Totty

Assistant Professor

Framingham State University

Copyright

Copyright © 2025. First Edition. Updated: August 31st, 2025.

Introducing RStudio: Hands-On Learning and Career Readiness in the Social, Behavioral, and Educational Sciences © 2025 by Njesa Totty is licensed under CC BY-NC 4.0. To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc/4.0/>.

This manual and its companion files may be downloaded from [the author's GitHub page](#).

The creation of this manual was supported by the CA-ROTEL Grant. More information on the project may be found at <https://carotelproject.org/>.

This work was primarily human-created. AI was used to make new content, such as text, images, analysis, and ideas. AI was prompted for its contributions, or AI assistance was enabled. AI-generated content was reviewed and approved.

Contents

| | |
|---|-----------|
| 1 Preface | 6 |
| 1.1 Use of GSS Data | 6 |
| 1.2 Career Readiness and NACE Competencies | 6 |
| 1.3 Use of AI in Development | 7 |
| 1.4 How to Use This Manual | 7 |
| 2 Navigating the RStudio IDE | 8 |
| 2.1 The RStudio Interface | 8 |
| 2.2 Installing and Loading Packages | 8 |
| 2.3 Console vs. Source | 9 |
| 2.4 Common Beginner Mistakes and How to Fix Them | 9 |
| 2.5 Summary | 9 |
| 3 Introduction to the Tidyverse and Core Data Skills | 10 |
| 3.1 Why Use the Tidyverse? | 10 |
| 3.2 Core Packages | 10 |
| 3.3 Installing and Loading the Tidyverse | 10 |
| 3.4 A Note on Tibbles | 11 |
| 3.5 The Pipe Operator (<code> ></code>) | 11 |
| 3.6 The Five Core <code>dplyr</code> Verbs | 12 |
| 3.7 Working with Text: <code>stringr</code> Functions | 16 |
| 3.8 Career Readiness Connection | 17 |
| 4 Importing and Cleaning Data | 18 |
| 4.1 Inspecting Data | 18 |
| 4.2 Handling Missing Data | 19 |
| 4.3 Reshaping Data | 19 |
| 4.4 Recoding Values | 20 |
| 4.5 Career Readiness Connection | 21 |
| 5 Visualizing and Summarizing Data | 22 |
| 5.1 Histograms: Distributions of Numerical Variables | 22 |
| 5.2 Line Graphs: Trends Over Time | 24 |
| 5.3 Scatterplots: Relationships Between Two Numerical Variables | 25 |
| 5.4 Boxplots: Comparing Numerical Distributions Across Groups | 26 |
| 5.5 Bar Charts: Summarizing Categorical Variables | 27 |

| | | |
|----------|--|-----------|
| 5.6 | Bar Charts with Group Comparisons | 28 |
| 5.7 | Faceting: Small Multiples for Subgroup Comparisons | 29 |
| 5.8 | Summary | 30 |
| 6 | Practice Problems | 31 |
| 6.1 | RStudio | 31 |
| 6.2 | Tidyverse | 31 |
| 6.3 | Cleaning | 31 |
| 6.4 | Data Visualization and Summarization | 32 |
| 7 | Appendix A: Shorter GSS Dataset Codebook | 33 |
| 8 | Appendix B: Practice Solutions | 34 |
| 8.1 | RStudio | 34 |
| 8.2 | Tidyverse | 34 |
| 8.3 | Cleaning | 37 |
| 8.4 | Data Visualization and Summarization | 39 |
| 9 | Appendix C: Transitioning from Python to R Using AI | 44 |
| 9.1 | Why Use AI to Bridge R and Python? | 44 |
| 9.2 | How to Prompt AI for Code Translation | 46 |
| 9.3 | Translating Core Data Tasks | 50 |
| 9.4 | Translating Visualizations | 53 |
| 9.5 | Becoming an Independent Learner with AI | 58 |
| 9.6 | Conclusion | 60 |

1 Preface

Data analysis is more than a set of formulaic procedures—it is a method for understanding the world around us. This lab manual was developed to help students begin working in RStudio so that they can analyze data to answer authentic questions in the social, behavioral, and educational sciences. By working directly with real datasets and using professional analytical tools, students will gain practical skills that bridge the gap between classroom learning and workplace application.

The activities in this manual align with the key topics of data visualization and descriptive statistics that are often taught in introductory statistics courses. Each activity invites students to use their skills in meaningful contexts, strengthening both data literacy and problem-solving abilities.

A distinctive feature of this manual is its integration of **RStudio** as the primary computational platform. RStudio provides a modern, open-source environment for data analysis, widely used in research and professional settings. Through repeated hands-on practice, students will learn to write and adapt R code, interpret outputs, create publication-quality graphics, and present results in ways that meet professional standards.

1.1 Use of GSS Data

To ground activities in relevant, real-world examples, this manual draws extensively from the [General Social Survey \(GSS\)](#) for years 2000–2024. The GSS contains a rich array of information on U.S. society, making it particularly valuable for students in the social, behavioral, and educational sciences. A cleaned subset of the data is available at [the author's GitHub page](#) and its codebook is in the appendix. The full dataset for years 1972–2024 is available from the [GSS Data Explorer](#), which instructors may use to create alternative subsets for classroom use.

1.2 Career Readiness and NACE Competencies

In addition to developing data analysis skills, this manual is designed to help students connect their academic work to professional competencies valued in the workplace. The [National Association of Colleges and Employers \(NACE\)](#) identifies eight core career readiness competencies:

1. Career and Self-Development
2. Communication
3. Critical Thinking
4. Equity and Inclusion
5. Leadership
6. Professionalism
7. Teamwork
8. Technology

Each activity in this manual intentionally builds one or more of these competencies. For example, students may develop *Critical Thinking* by interpreting `ggplot` outputs to guide evidence-based decisions, or *Technology* skills by automating data cleaning processes in R. By engaging with these competencies throughout the manual, students can see how the skills they build translate into tangible career preparation.

1.3 Use of AI in Development

The creation of this manual incorporated AI-assisted tools for formatting, content organization, and resource curation. This reflects a commitment to leveraging emerging technologies in educational publishing, while ensuring that all instructional content has been carefully reviewed and aligned with learning objectives.

1.4 How to Use This Manual

Students are encouraged to complete activities sequentially, as later exercises build on skills developed earlier. Instructors may adapt the activities to fit the structure of their courses or the specific needs of their students. Supplementary files are available on the author's GitHub page for easy access and customization.

By the end of this manual, students will not only be able to apply the methods of data visualization and summarization using RStudio, but also recognize the value of these skills in a variety of career paths across the social, behavioral, and educational sciences.

2 Navigating the RStudio IDE

In this chapter, you will learn to navigate RStudio, install and load packages, and avoid common beginner pitfalls. By the end, you will be able to open a new R script, write and run R code, install the packages you need, and troubleshoot common errors. At this point you should have [RStudio downloaded](#) and opened alongside this lab manual.

2.1 The RStudio Interface

RStudio is divided into four panes:

- **Source (Top Left):** Where you write and save R scripts. To open a new script press Ctrl+Shift+N (Windows) or Cmd+Shift+N (Mac)
- **Console (Bottom Left):** Where you run code for immediate execution.
- **Environment/History (Top Right):** Lists objects and variables in memory, plus a history of commands.
- **Files/Plots/Packages/Help/Viewer/Presentation (Bottom Right):** Used to browse files, view plots, documents, or presentations, manage packages, and access help files.

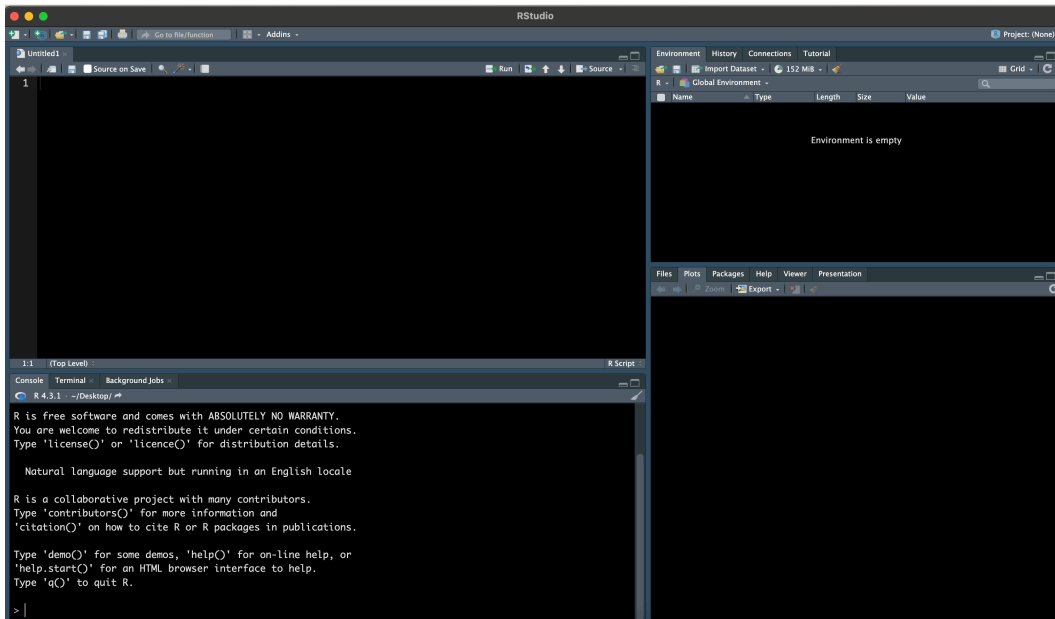


Figure 1: The four-pane layout of RStudio.

2.2 Installing and Loading Packages

R's core functions are extended by packages. We will use `tidyverse` frequently so it is a good idea to install it now. Type the following code into your R script and use Ctrl+R (Windows) or Cmd+R (Mac) to run each line.

- Install once: `install.packages("tidyverse")`
- Load in each session: `library(tidyverse)`

Installing a package is like screwing in a light bulb. Loading a package is like turning on the light switch. You only need to install the package (light bulb) once but you need to load the package (turn on the light) each time you open RStudio (enter the room).

2.3 Console vs. Source

The console is for one-time commands; the source is for writing and saving code.

- Try typing `2 + 2` in both places.
- You can save your R script with `Ctrl+S` (Windows) or `Cmd+S` (Mac).
- You can repopulate the console with the previous line of code using the up arrow
- Code typed into the Console is not saved!

2.4 Common Beginner Mistakes and How to Fix Them

Forgetting quotes around strings: `paste(apple)` causes an error; use `"apple"` for strings.

Running code in the wrong pane: If nothing happens, check if you typed into the source but forgot to press `Ctrl+Enter` (Windows) or `Cmd+Enter` (Mac).

Forgetting to load a package: Restarting R clears loaded packages and their functions will not be found; run `library(package_name)` again.

Overwriting variables: Accidentally reusing names (e.g., `mean <- 5`) will hide the original function. Restart R to reset.

Misplaced working directory: Errors like “file not found” often mean R is looking in the wrong folder. Check with `getwd()`.

Knowing these pitfalls can save hours of frustration. Learning to code in R takes time and practice so be prepared for some frustration and determine now to continue persevering despite this.

2.5 Summary

You now have the essential navigation skills to work effectively in RStudio. The more time you spend working in RStudio the more comfortable you will become using these and other skills. There is a wealth of RStudio documentation online. It is always a good idea to search for help online and bookmark any useful tips for future reference.

3 Introduction to the Tidyverse and Core Data Skills

The `tidyverse` is a collection of R packages designed to make data science easier, more consistent, and more readable. Think of it as a “toolbox” filled with tools that all work in a similar way, so once you learn one, the others feel familiar. We will use the `tidyverse` throughout this manual for data wrangling, visualization, and analysis.

While R has many built-in, or **base R**, functions, the `tidyverse` provides a consistent “grammar” for data analysis that’s easier to learn for beginners and widely used by professionals in research, government, and industry.

3.1 Why Use the Tidyverse?

- **Consistency** – The same style and function names appear across different tasks.
- **Readability** – Code often reads like a sentence describing the analysis.
- **Efficiency** – Complex tasks can often be completed with fewer lines of code.
- **Career relevance** – Tidyverse skills are in demand for data roles across the social, behavioral, and educational sciences.

3.2 Core Packages

When you load the `tidyverse`, you get several packages at once:

| Package | Purpose |
|----------------------|---|
| <code>ggplot2</code> | Create high-quality visualizations |
| <code>dplyr</code> | Manipulate and summarize data (filter rows, select columns, group, arrange) |
| <code>tidyr</code> | Reshape and clean data |
| <code>readr</code> | Import data from CSV and text files |
| <code>stringr</code> | Work with text data (strings) |
| <code>forcats</code> | Manage categorical variables (factors) |

We will mostly focus on `ggplot2`, `dplyr`, `tidyr`, and `readr`.

3.3 Installing and Loading the Tidyverse

You only need to install the `tidyverse` **once** on your computer. The code is included below in case if you have not already done so.

```
install.packages("tidyverse")
```

You must **load** the `tidyverse` each time you open RStudio:

```
library(tidyverse)
```

3.4 A Note on Tibbles

When you work with `tidyverse` functions, you will often see data printed in a slightly different format called a **tibble**. Tibbles are modern versions of data frames:

- They show only the first few rows and columns by default.
- They display column data types under the names.
- They never change variable names or types without your permission.

3.5 The Pipe Operator (`|>`)

A key feature of `tidyverse` workflow is the **pipe** operator, `|>`. It passes the result of one function into the next.

Without the pipe:

```
summarise(group_by(txhousing, year), avg_sales = mean(sales, na.rm = TRUE))
```

```
## # A tibble: 16 x 2
##   year avg_sales
##   <int>   <dbl>
## 1  2000     478.
## 2  2001     496.
## 3  2002     532.
## 4  2003     539.
## 5  2004     577.
## 6  2005     636.
## 7  2006     665.
## 8  2007     621.
## 9  2008     520.
## 10 2009     469.
## 11 2010     440.
## 12 2011     447.
## 13 2012     520.
## 14 2013     608.
## 15 2014     626.
## 16 2015     659.
```

With the pipe:

```
txhousing |>
  group_by(year) |>
  summarise(avg_sales = mean(sales, na.rm = TRUE))
```

```
## # A tibble: 16 x 2
##   year avg_sales
##   <int>   <dbl>
## 1  2000     478.
## 2  2001     496.
## 3  2002     532.
```

```
## 4 2003      539.
## 5 2004      577.
## 6 2005      636.
## 7 2006      665.
## 8 2007      621.
## 9 2008      520.
## 10 2009     469.
## 11 2010     440.
## 12 2011     447.
## 13 2012     520.
## 14 2013     608.
## 15 2014     626.
## 16 2015     659.
```

Think of it as saying:

1. Start with `txhousing`.
2. Group by `year`.
3. Summarize with the average number of sales.

We'll explain what `summarise` and `group_by` do later, but notice that both chunks of code produce the same result. However, code written with the pipe is easier to read.

3.6 The Five Core `dplyr` Verbs

In the following examples, we will use the **General Social Survey** dataset. Read in the dataset from your computer and save it as `gssdat`. The dataset should be saved in the same location on your computer as the R script that you are working in.

```
# read from a downloaded file in your working directory
gssdat <- read.csv("files/gssdat.csv")
```

3.6.1 `select()` — Choose the variables you need

Datasets can be overwhelming. If you check the dimensions of the GSS data with `dim(gssdat)` you will see that it is a large dataset. Selecting specific variables keeps your workspace and analysis focused. For example, you can select the variables `year`, `age`, `degree`, and `income` using the code below. We use `slice_sample(n = 10)` to return only a random sample of ten rows rather than the entire dataset. We will use this throughout and it should be used before knitting to avoid generating cumbersome output.

```
gssdat |>
  select(year, age, degree, income) |>
  slice_sample(n = 10)
```

```
##   year age          degree income
## 1 1984 66  less than high school     3
## 2 2006 34  bachelor's          12
## 3 2012 32  high school          11
## 4 1998 50  bachelor's          12
## 5 2006 30  associate/junior college    11
```

```
## 6 1986 27          bachelor's    10
## 7 2018 64    less than high school  1
## 8 1996 68          high school    12
## 9 1987 87    less than high school  NA
## 10 2000 41          high school    9
```

3.6.2 filter() — Keep only the rows you want

Social science researchers often focus on specific subgroups (e.g., only people of a certain age, education level, or survey year). Filtering lets us zoom in. For example, we may want to study the age, degree, and income of respondents from 2022 who have a bachelor's degree or higher.

```
gssdat |>
  select(year, age, degree, income) |>
  filter(year == 2022, degree %in% c("bachelor's", "graduate")) |>
  slice_sample(n = 10)
```

```
##   year age   degree income
## 1 2022 88 graduate     12
## 2 2022 49 bachelor's    12
## 3 2022 63 bachelor's    12
## 4 2022 34 bachelor's    12
## 5 2022 44 graduate     12
## 6 2022 32 bachelor's    12
## 7 2022 79 graduate     12
## 8 2022 62 bachelor's    12
## 9 2022 49 bachelor's    12
## 10 2022 33 bachelor's    12
```

The `%in%` operator determines whether elements of the vector on the left-hand side are present in the right-hand side vector. There are many different ways to use this operator. A few examples are given below along with the output that they return. The length of the resulting logical vector will be the same as that for the left-hand side vector.

```
# length one vector
5 %in% c(1, 3, 5, 7, 9)
```

```
## [1] TRUE
```

```
# length three vector
c(1, 3, 5) %in% c(1, 2, 4, 6, 3)
```

```
## [1] TRUE TRUE FALSE
```

```
# toy dataset
toydat <- data.frame(letter = c("A", "B", "C", "D", "E", "F", "I", "A"))
```

```
# length two vector against variable
c("A", "F") %in% toydat$letter
```

```
## [1] TRUE TRUE
```

```
# variable against length two vector  
toydat$letter %in% c("A", "F")
```

```
## [1] TRUE FALSE FALSE FALSE FALSE TRUE FALSE TRUE
```

3.6.3 mutate() — Create new variables

Sometimes the variable you need is not readily available in the dataset and you have to create it from other variables in the dataset. For example, we can create a new variable for whether a respondent is considered a “young adult” (18–30) or not.

```
gssdat |>  
  mutate(young_adult = if_else(age >= 18 & age <= 30, "Yes", "No")) |>  
  select(year, young_adult, degree, income) |>  
  slice_sample(n = 10)
```

```
##   year young_adult          degree income  
## 1  1998         Yes      high school     6  
## 2  2000         Yes associate/junior college 12  
## 3  1994         Yes less than high school 12  
## 4  1987         No less than high school 12  
## 5  2024         No      high school 12  
## 6  1994         No less than high school 12  
## 7  1990         No      high school 11  
## 8  2014         No      bachelor's 12  
## 9  1998         Yes      high school  9  
## 10 1975         No      high school 12
```

3.6.4 summarise() — Condense data into summaries

Summarizing helps researchers describe trends, central tendencies, and group differences. For example, we can find the average age of respondents by highest degree. Setting `na.rm = TRUE` removes missing values before R calculates the mean. Missing values are commonly found in all types of data, including survey data where respondents may choose not to answer a question.

```
gssdat |>  
  group_by(degree) |>  
  summarise(avg_age = mean(age, na.rm = TRUE))
```

```
## # A tibble: 6 x 2  
##   degree          avg_age  
##   <chr>          <dbl>  
## 1 associate/junior college 44.0
```

```
## 2 bachelor's          45.2
## 3 graduate            49.9
## 4 high school        44.8
## 5 less than high school 52.5
## 6 <NA>                47.4
```

3.6.5 arrange() — Order the rows

Sorting results can make patterns easier to see and tables easier to interpret. For example, we can arrange degrees by average income bracket from lowest to highest.

```
gssdat |>
  group_by(degree) |>
  summarise(avg_income = mean(income, na.rm = TRUE)) |>
  arrange(avg_income)
```

```
## # A tibble: 6 x 2
##   degree          avg_income
##   <chr>          <dbl>
## 1 less than high school    8.08
## 2 <NA>                 8.45
## 3 high school            10.3
## 4 associate/junior college 11.1
## 5 bachelor's            11.3
## 6 graduate              11.6
```

To see what income brackets 1-12 correspond to monetarily use `unique(gssdat$income_cat)`. Income bracket one is under \$1,000, bracket two is \$1,000 to \$2,999 and so on.

```
unique(gssdat$incomecat)
```

```
## [1] NA "$10,000 to $14,999" "$7,000 to $7,999"
## [4] "$4,000 to $4,999" "$1,000 to $2,999" "$15,000 to $19,999"
## [7] "$5,000 to $5,999" "$20,000 to $24,999" "$3,000 to $3,999"
## [10] "under $1,000" "$8,000 to $9,999" "$25,000 or more"
## [13] "$6,000 to $6,999"
```

To sort the data from highest to lowest income bracket we can use `desc`. Notice that we use the numeric form of income here.

```
gssdat |>
  group_by(degree) |>
  summarise(avg_income = mean(income, na.rm = TRUE)) |>
  arrange(desc(avg_income))
```

```
## # A tibble: 6 x 2
##   degree          avg_income
##   <chr>          <dbl>
```

```
## 1 graduate 11.6
## 2 bachelor's 11.3
## 3 associate/junior college 11.1
## 4 high school 10.3
## 5 <NA> 8.45
## 6 less than high school 8.08
```

3.7 Working with Text: stringr Functions

3.7.1 str_detect() — Find patterns in text

Many survey variables are stored as text (e.g., job titles, open-ended responses). Pattern matching helps you find specific entries. For example, we can find respondents whose job titles contain “teacher”, “nurse”, or “emergency”.

```
gssdat |>
  filter(str_detect(occ10, regex("teacher|nurse|emergency"))) |>
  select(age, incomecat, occ10) |>
  slice_sample(n = 10)
```

```
##   age      incomecat      occ10
## 1  28 $25,000 or more elementary and middle school teachers
## 2  74      <NA>          registered nurses
## 3  50 $25,000 or more preschool and kindergarten teachers
## 4  28 $25,000 or more postsecondary teachers
## 5  42 $25,000 or more secondary school teachers
## 6  56 $25,000 or more special education teachers
## 7  65 $25,000 or more preschool and kindergarten teachers
## 8  57 $25,000 or more postsecondary teachers
## 9  39 $25,000 or more postsecondary teachers
## 10 30 $25,000 or more licensed practical and licensed vocational nurses
```

In the above code the bar (|) is interpreted as or; asking do any of these terms appear? To filter based on the detection of a single string we do not need to use regex as below.

```
gssdat |>
  filter(str_detect(occ10, "teacher")) |>
  select(age, incomecat, occ10) |>
  slice_sample(n = 10)
```

```
##   age      incomecat      occ10
## 1  49 $15,000 to $19,999 secondary school teachers
## 2  23 $25,000 or more other teachers and instructors
## 3  82 $25,000 or more secondary school teachers
## 4  89 $20,000 to $24,999 elementary and middle school teachers
## 5  38 $10,000 to $14,999 teacher assistants
## 6  40 $25,000 or more elementary and middle school teachers
## 7  69 $25,000 or more secondary school teachers
## 8  32 $25,000 or more elementary and middle school teachers
## 9  69 $25,000 or more postsecondary teachers
## 10 84 $3,000 to $3,999 elementary and middle school teachers
```

3.7.2 `str_replace()` — Clean up or standardize text

Standardizing text responses can make them easier to analyze. Cleaning up strings with repetition can also make analysis easier. For example, we can shorten the word `miscellaneous` to `misc` for brevity in the occupation variable.

```
gssdat |>
  filter(str_detect(occ10, "miscellaneous")) |>
  mutate(occ10 = str_replace(occ10, "miscellaneous", "misc")) |>
  select(occ10) |>
  slice_sample(n = 10)
```

```
##                occ10
## 1      misc assemblers and fabricators
## 2          misc agricultural workers
## 3          misc legal support workers
## 4      misc assemblers and fabricators
## 5 misc health technologists and technicians
## 6          misc legal support workers
## 7      misc assemblers and fabricators
## 8      misc assemblers and fabricators
## 9      misc assemblers and fabricators
## 10     misc assemblers and fabricators
```

3.8 Career Readiness Connection

These skills mirror real-world research and data tasks:

- Filtering/selecting data is like narrowing a literature review to relevant studies.
- Creating variables mirrors coding qualitative data into categories.
- String matching is useful in cleaning resumes, job postings, or survey responses.

By starting with tidyverse now, you are learning tools that can directly transfer to internships, research assistant roles, and jobs involving data analysis.

4 Importing and Cleaning Data

In this section, you will learn how to import, inspect, and clean data using the **tidyverse**. These are the first steps in almost every data analysis project, and mastering them will help you prepare datasets for meaningful analysis.

4.1 Inspecting Data

We will continue using the **General Social Survey (GSS)** dataset introduced earlier. If you have not already loaded it, you can do so now. If you closed RStudio since the last section, reload the dataset before proceeding.

If you already have the `gssdat` dataset and `tidyverse` loaded from Section 1.2, you can skip this step. Otherwise, run the following code:

```
# load package
library(tidyverse)

# Load the GSS dataset from your working directory
gssdat <- read.csv("files/gssdat.csv")
```

Before cleaning data, it's important to understand what you have. Use these functions to explore the dataset:

```
# Number of rows and columns
dim(gssdat)
```

```
## [1] 75699    27
```

```
# First ten variable names
names(gssdat)
```

```
## [1] "year"      "age"       "degree"    "income"    "occ10"     "incomecat"
## [7] "trustfam"  "trustdoc"  "wrkstat"   "id"        "marital"   "agewed"
## [13] "conjudge"  "realrinc"  "hrs1"      "conpress"  "race"      "consci"
## [19] "fepol"    "happy"     "childs"    "health"    "realinc"   "coneduc"
## [25] "conlegis"  "sex"       "educ"
```

```
# Summary of one variable
summary(gssdat$age)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's
##  18.00   32.00   44.00  46.72  60.00   89.00   870
```

These simple checks help you identify issues such as missing values, inconsistent variable names, or unusual data entries.

4.2 Handling Missing Data

Missing values are common in survey data. They can appear as NA. Rows with missing data should not be removed unless if information is missing for all variables. Rather we can omit missing values as needed during the analysis. We already saw this in the previous section where we used `na.rm = TRUE` to calculate summary statistics (e.g. `mean`, `median`) on variables with missing values. Below are some examples of how we can also handle missing data:

```
# Count number of missing values in each column
gssdat |>
  is.na() |>
  colSums()
```

```
##      year      age  degree  income  occ10  incomecat  trustfam  trustdoc
##      0       870    199    9329    8447    9329      74321    74334
##  wrkstat     id  marital  ageded  conjudge  realrinc    hrs1    conpress
##      47      0      66    49156    26388    33366    32371    25474
##      race  consci  fepol   happy  childs   health   realinc  coneduc
##      172   27826  39112   4830    292    17251   10787   25135
##  conlegis   sex    educ
##  25853     131    286
```

```
# How many rows do not have a missing income bracket
gssdat |>
  filter(!is.na(income)) |>
  nrow()
```

```
## [1] 66370
```

Removing or recoding missing data helps avoid bias and ensures analyses run correctly. We can also summarize the number of missing values in one variable for each category of another variable. This may help us to discover any patterns in the missingness.

4.3 Reshaping Data

Sometimes variables are stored in a “wide” format but need to be “long” for analysis. The `pivot_longer()` function from `tidyr` changes wide data to long.

Suppose that we want to compare responses to trust questions across types. Below we compare types of trusts between trusting one’s family versus their doctor.

```
# Example with two trust variables
gssdat |>
  select(trustfam, trustdoc) |>
  pivot_longer(cols = c(trustfam, trustdoc),
               names_to = "trust_type",
               values_to = "trust_level") |>
  group_by(trust_type, trust_level) |>
  summarize(count = n())
```

```
## # A tibble: 12 x 3
```

```
## # Groups:   trust_type [2]
##   trust_type trust_level   count
##   <chr>      <chr>        <int>
## 1 trustdoc   a great deal    400
## 2 trustdoc   completely      381
## 3 trustdoc   not at all      49
## 4 trustdoc   only a little   94
## 5 trustdoc   somewhat       441
## 6 trustdoc   <NA>          74334
## 7 trustfam   a great deal    242
## 8 trustfam   completely     1025
## 9 trustfam   not at all     17
## 10 trustfam  only a little   17
## 11 trustfam  somewhat       77
## 12 trustfam  <NA>          74321
```

We can also expand the data in one column based on another column using `pivot_wider()`. Suppose that we want to view the work status of respondents for surveys after 2020.

```
# widen workstatus by year
gssdat |>
  select(id, year, wrkstat) |>
  pivot_wider(names_from = year,
              values_from = wrkstat) |>
  select('2021', '2022', '2024') |>
  slice_sample(n = 10)
```

```
## # A tibble: 10 x 3
##   `2021`                `2022`                `2024`
##   <chr>                 <chr>                 <chr>
## 1 <NA>                 retired                working full time
## 2 in school            working part time      retired
## 3 other                working full time      working full time
## 4 retired              keeping house           working full time
## 5 working full time    working full time      working full time
## 6 working part time    <NA>                  <NA>
## 7 unemployed, laid off, looking for work retired                working full time
## 8 working full time    working part time      working full time
## 9 <NA>                 working full time      working full time
## 10 retired             retired                retired
```

4.4 Recoding Values

Recoding helps combine categories or fix inconsistent entries.

```
# Combine 'separated' and 'divorced' into 'not married'
gssdat |>
  mutate(marital = recode(marital,
                          "separated" = "not married",
                          "divorced" = "not married")) |>
  count(marital)
```

```
##      marital      n
## 1      married 38959
## 2 never married 16963
## 3    not married 12706
## 4      widowed   7005
## 5         <NA>    66
```

```
# compare
gssdat |>
  count(marital)
```

```
##      marital      n
## 1      divorced 10158
## 2      married 38959
## 3 never married 16963
## 4      separated  2548
## 5      widowed   7005
## 6         <NA>    66
```

4.5 Career Readiness Connection

Data cleaning is a critical step in real-world research and professional work. The following career competencies were touched on in this section:

- **Technology:** Learning reproducible, code-based methods for preparing data.
- **Critical Thinking:** Deciding how to handle missing or inconsistent data involves judgment and awareness of the research context.
- **Equity and Inclusion:** Fair handling of missing responses and careful recoding can reduce bias in your analysis.

By mastering these skills, you will be able to prepare datasets for valid, trustworthy analysis. This is an essential ability in both academic research and data-driven careers.

5 Visualizing and Summarizing Data

In this chapter, we learn to visualize and summarize data. You will learn about five types of graphs commonly used in the social sciences:

- Histograms
- Line graphs
- Scatterplots
- Bar charts
- Boxplots

These plots help us identify patterns, compare groups, and communicate findings clearly. Visualizations are especially useful for exploring relationships between variables while tables are useful for summarizing large datasets. We will again use the General Social Survey (GSS) data throughout this chapter.

If you have not already done so you can load the data and the `tidyverse` using the code below.

```
# load the package
library(tidyverse)

# load the data
gssdat <- read.csv("gssdat.csv")
```

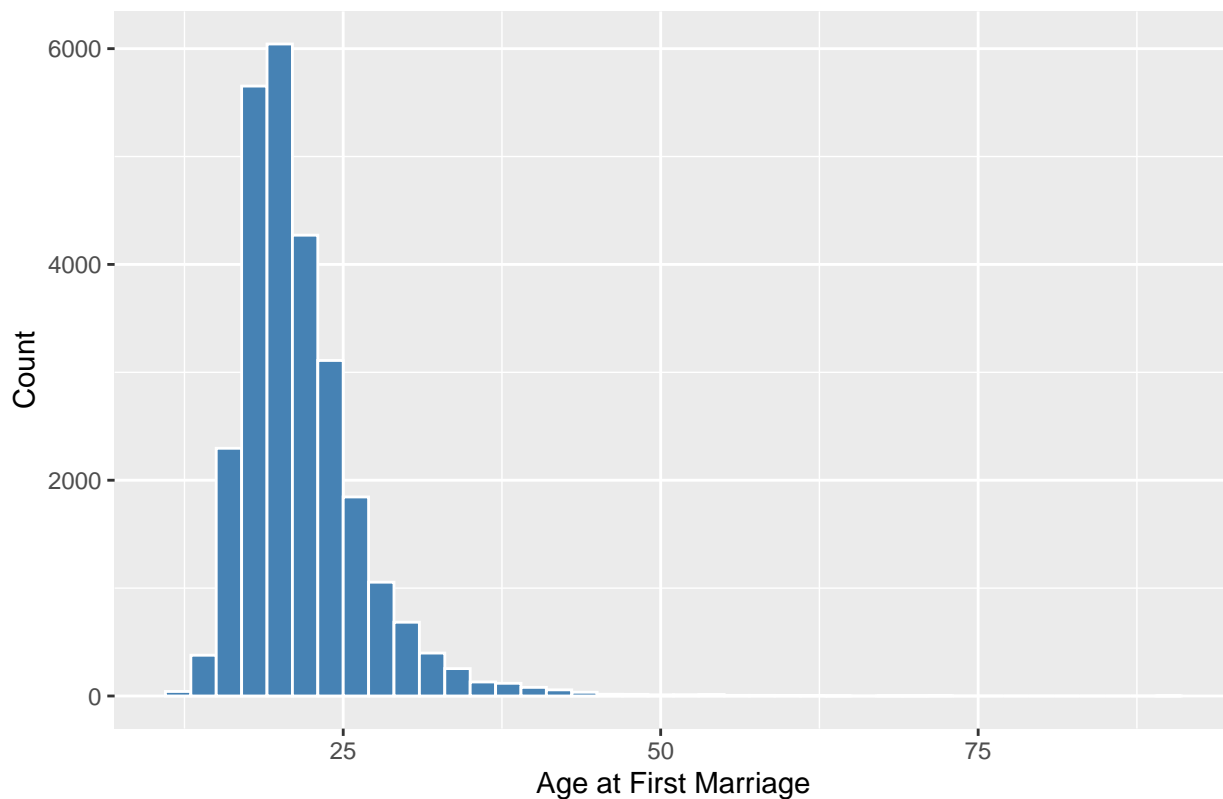
5.1 Histograms: Distributions of Numerical Variables

A **histogram** displays how often each range of values occurs for a numerical variable. They are useful for numerical variables that take a wide range of values. They tell us about the center, shape, and spread of the distribution.

Here we examine the distribution of age at first marriage (`agewed`):

```
gssdat |>
  ggplot(aes(x = agewed)) +
  geom_histogram(binwidth = 2, fill = "steelblue", color = "white") +
  labs(title = "Distribution of Age at First Marriage",
       x = "Age at First Marriage",
       y = "Count")
```

Distribution of Age at First Marriage



We see that most respondents married in their early 20s. The histogram is slightly right-skewed since the outliers lie to the right. In this case the mean age at first marriage will be greater than the median age. We can impose the mean and median on the plots but first we must compute them using `summarize`.

```
agewed_summary <- gssdat |>
  summarize(mean_age = mean(agewed, na.rm = TRUE),
            med_age = median(agewed, na.rm = TRUE))
```

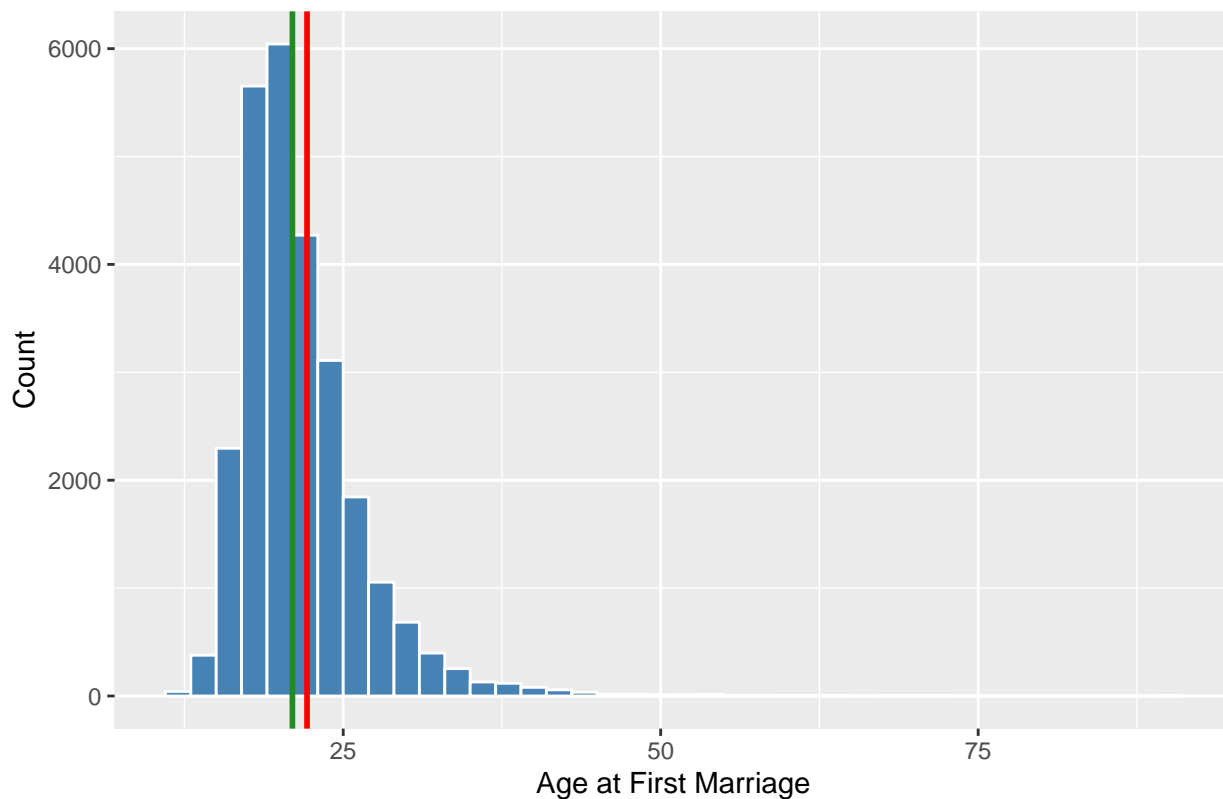
```
agewed_summary
```

```
##   mean_age med_age
## 1 22.15006     21
```

To add the mean and median we use `geom_vline` which adds a vertical line to a plot at the specified x-intercept. For a horizontal line we would use `geom_hline`. Since we are using two different datasets in the same plot we need to specify the `agewed_summary` dataset in the `geom_vline` layer:

```
gssdat |>
  ggplot(aes(x = agewed)) +
  geom_histogram(binwidth = 2, fill = "steelblue", color = "white") +
  geom_vline(data = agewed_summary, aes(xintercept = mean_age), color = "red", linewidth = 1) +
  geom_vline(data = agewed_summary, aes(xintercept = med_age), color = "forestgreen", linewidth = 1) +
  labs(title = "Distribution of Age at First Marriage",
       x = "Age at First Marriage",
       y = "Count")
```

Distribution of Age at First Marriage

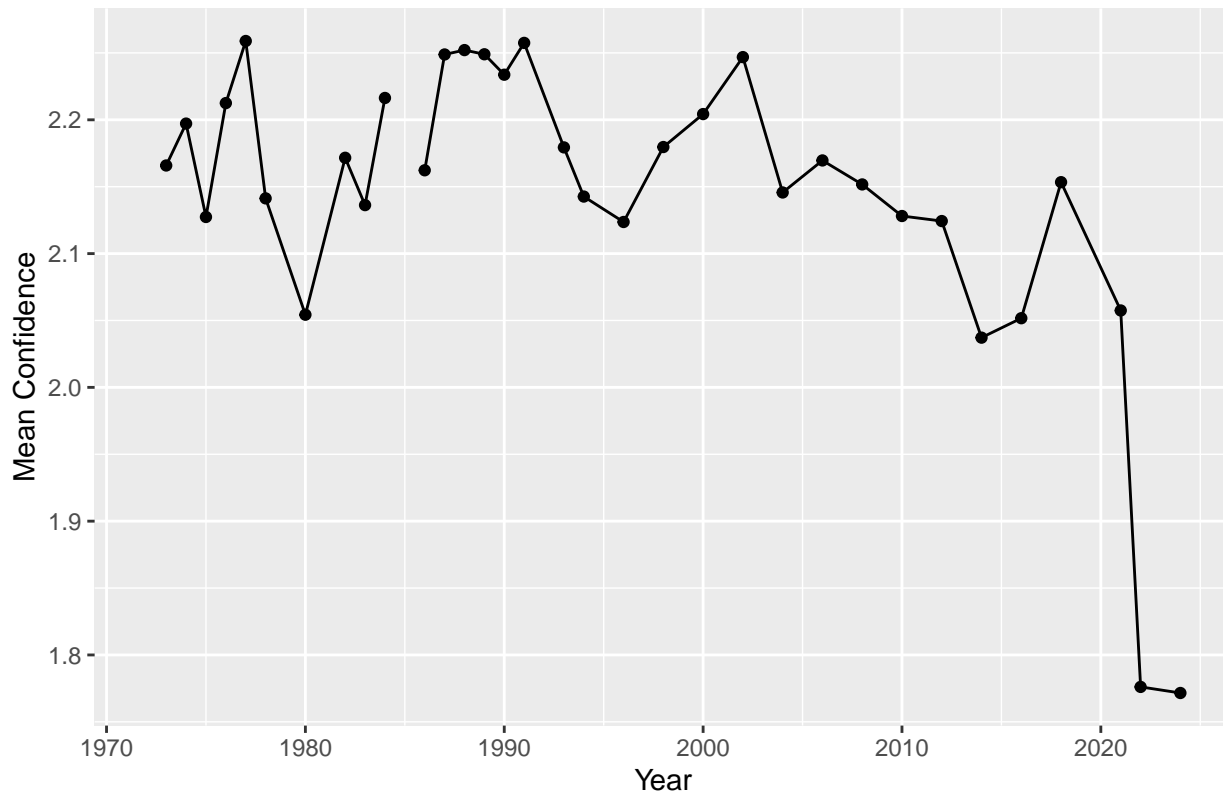


5.2 Line Graphs: Trends Over Time

Line graphs show how the value of a variable changes over time. For example, we can examine how average confidence in the U.S. Supreme Court (`conjudge`) has changed across survey years. We first mutate the data so that a one represents hardly any confidence and a three represent a great deal of confidence. Then, we calculate the summary table and pipe this into `ggplot`:

```
gssdat |>
  group_by(year) |>
  mutate(conjudge = recode(conjudge, `1` = 3L, `3` = 1L, `2` = 2L)) |>
  summarize(mean_conf = mean(conjudge, na.rm = TRUE)) |>
  ggplot(aes(x = year, y = mean_conf)) +
  geom_line() +
  geom_point() +
  labs(title = "Average Confidence in the Supreme Court Over Time",
       x = "Year",
       y = "Mean Confidence")
```

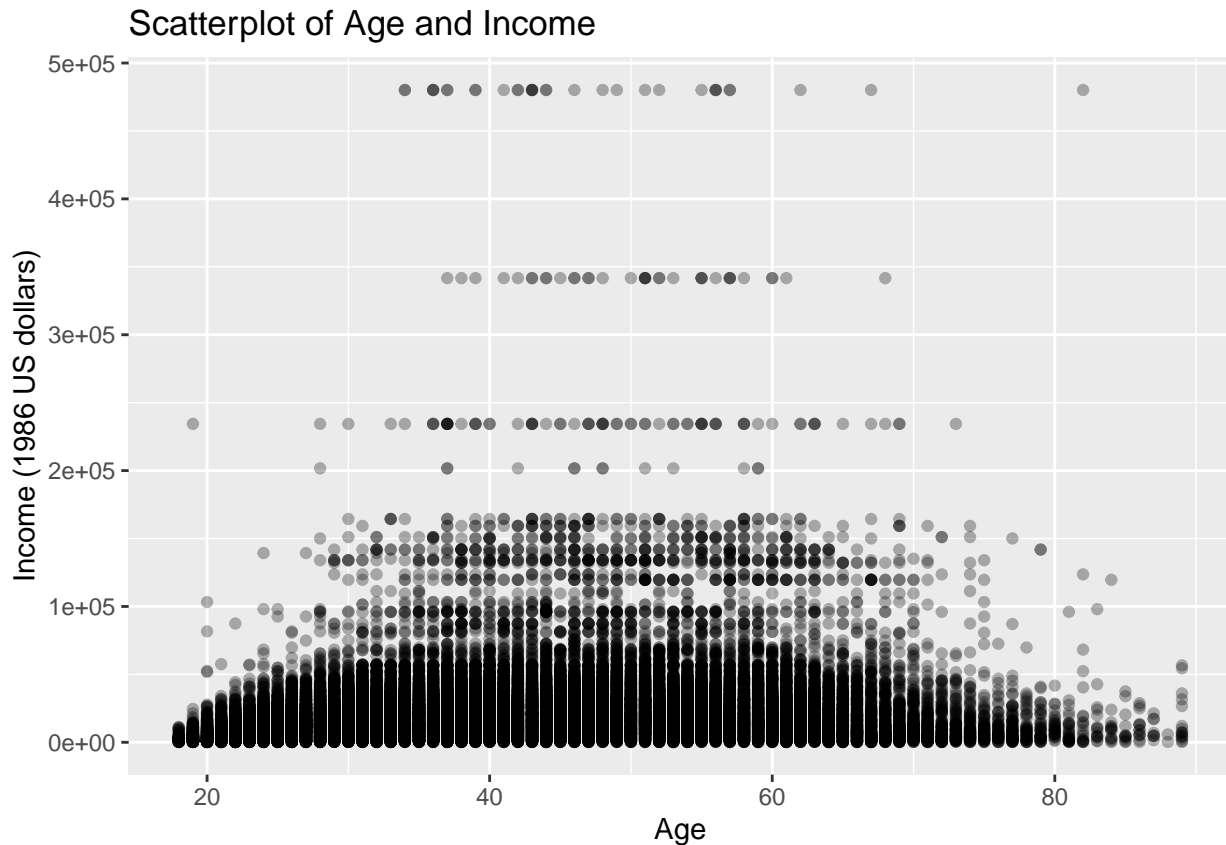
Average Confidence in the Supreme Court Over Time



5.3 Scatterplots: Relationships Between Two Numerical Variables

Scatterplots help us examine relationships between two numerical variables. For example, we can compare age (`age`) and income in constant 1986 US dollars (`realrinc`):

```
gssdat |>
  ggplot(aes(x = age, y = realrinc)) +
  geom_point(alpha = 0.3) +
  labs(title = "Scatterplot of Age and Income",
       x = "Age",
       y = "Income (1986 US dollars)")
```



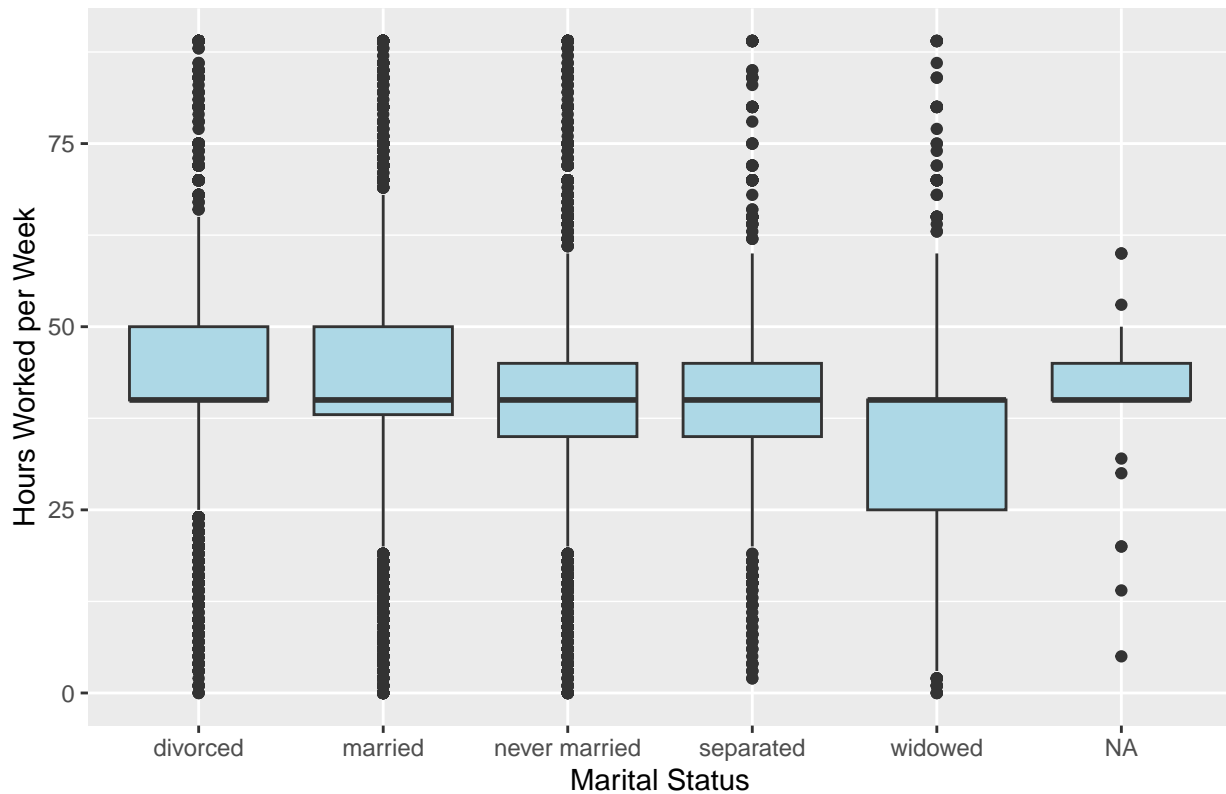
It seems that income is greater for respondents in their middle ages (30-70) while it is less for respondents whose ages are less than 30 or greater than 70.

5.4 Boxplots: Comparing Numerical Distributions Across Groups

Boxplots display the distribution of a numerical variable and are especially useful for comparing groups. We can compare medians (center bar), ranges (sides of box), and variability across categories. Outliers are plotted as actual points. Below we compare hours worked per week (`hrs1`) across marital status:

```
gssdat |>
  ggplot(aes(x = marital, y = hrs1)) +
  geom_boxplot(fill = "lightblue") +
  labs(title = "Hours Worked per Week by Marital Status",
       x = "Marital Status",
       y = "Hours Worked per Week")
```

Hours Worked per Week by Marital Status



The distribution of hours worked is very similar for those never married and separated. It is actually a bit hard to determine which group has the highest work hours. Therefore a plot like this would be well accompanied by the summary table below:

```
gssdat |>
  group_by(marital) |>
  summarize(avg_hrs = mean(hrs1, na.rm = TRUE)) |>
  arrange(avg_hrs)
```

```
## # A tibble: 6 x 2
##   marital      avg_hrs
##   <chr>         <dbl>
## 1 widowed         35.3
## 2 <NA>            39.0
## 3 never married  40.0
## 4 separated      41.0
## 5 married        41.7
## 6 divorced       42.3
```

We see that divorced respondents reported the greatest working hours, on average, and those widowed reported the lowest.

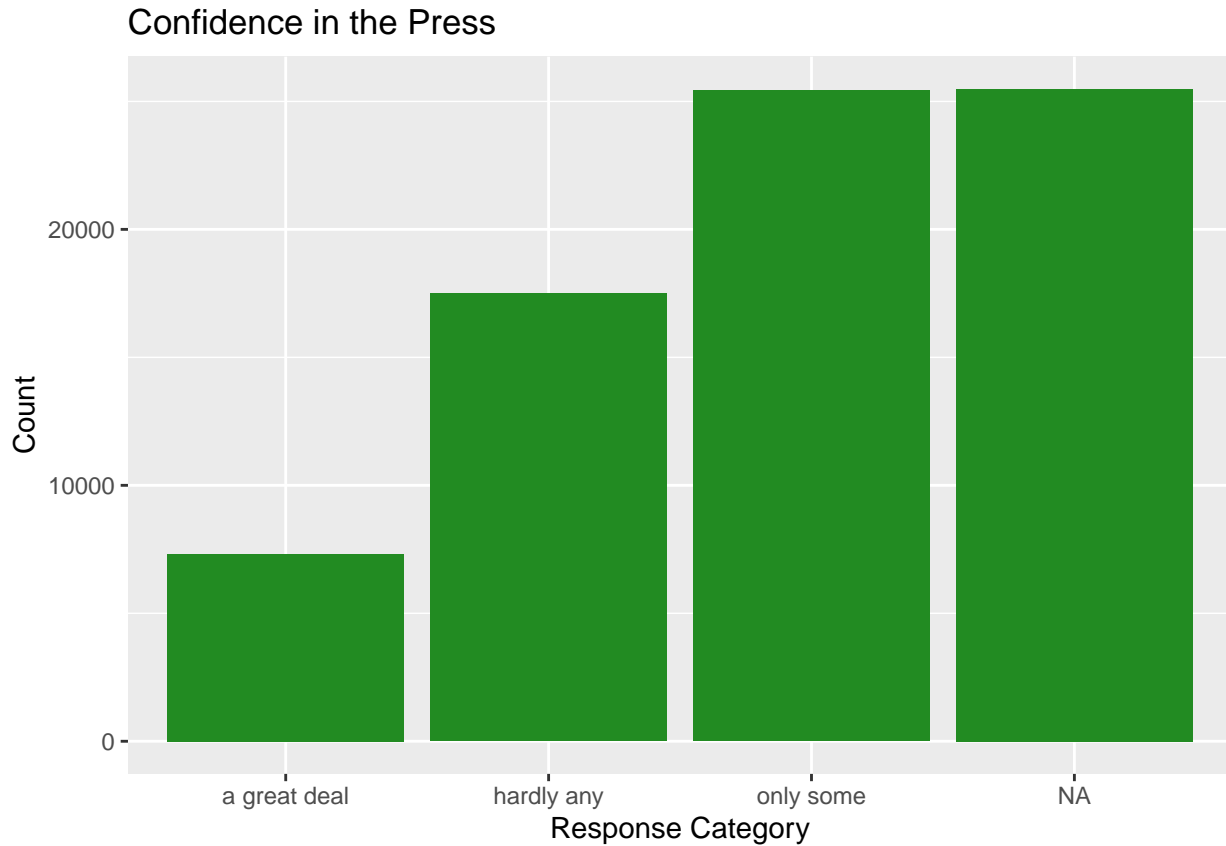
5.5 Bar Charts: Summarizing Categorical Variables

Bar charts summarize counts or proportions for categorical variables. We look at confidence in the press (conpress) below:

```

gssdat |>
  ggplot(aes(x = conpress)) +
  geom_bar(fill = "forestgreen") +
  labs(title = "Confidence in the Press",
       x = "Response Category",
       y = "Count")

```



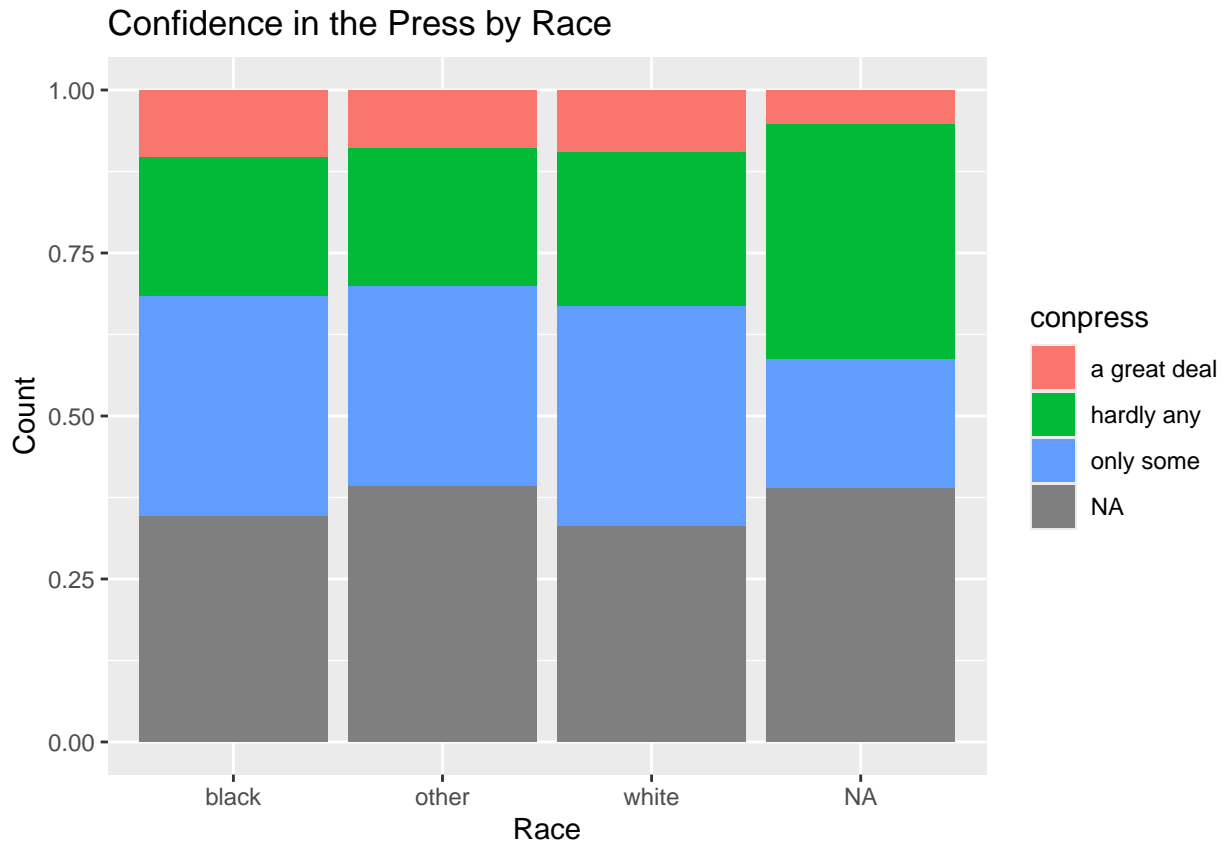
5.6 Bar Charts with Group Comparisons

We can also compare responses across groups using side-by-side bar charts. For example, we examine confidence in the press by race (`race`) below:

```

gssdat |>
  ggplot(aes(x = race, fill = conpress)) +
  geom_bar(position = "fill") +
  labs(title = "Confidence in the Press by Race",
       x = "Race",
       y = "Count")

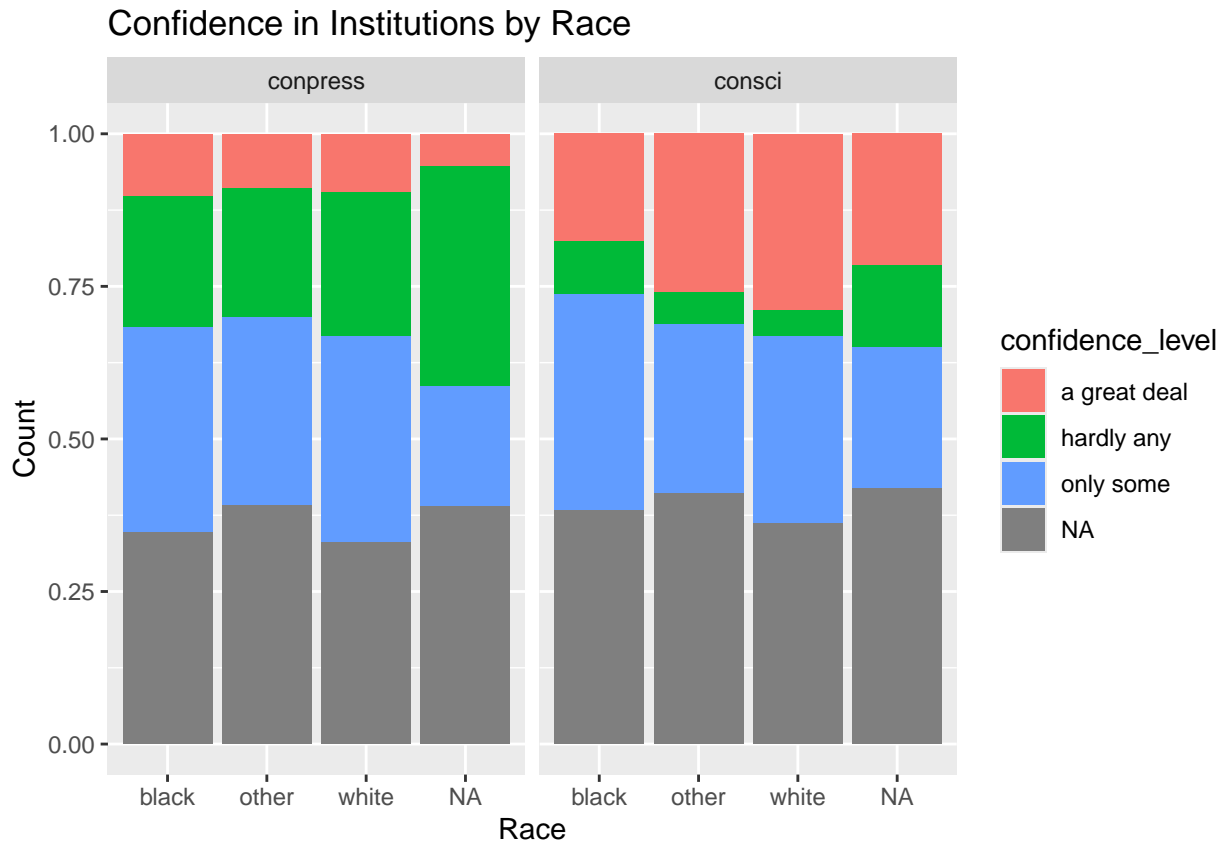
```



5.7 Faceting: Small Multiples for Subgroup Comparisons

Faceting is a technique that creates multiple small plots for subgroups. Faceting is helpful when we want to examine subgroup differences without cluttering a single plot. For example, we reshape the dataset to analyze multiple confidence variables together. Below we use `pivot_longer` to gather several confidence variables into a long format before again making comparisons across races:

```
gssdat |>
  select(conpress, consci, race) |>
  pivot_longer(cols = c(conpress, consci),
               names_to = "confidence_type",
               values_to = "confidence_level") |>
  ggplot(aes(x = race, fill = confidence_level)) +
  geom_bar(position = "fill") +
  facet_wrap(~confidence_type) +
  labs(title = "Confidence in Institutions by Race",
       x = "Race",
       y = "Count")
```



There are higher rates of respondents indicating a great deal of confidence in science in comparison to the press. Individuals identifying as Black responded in this manner at a lower rate than all other groups though. There are quite a few missing values across all racial groups and confidence types (gray colored portion of the bars). The use of faceting has allowed to plot four variable (count, confidence level, confidence type, and race) simultaneously.

5.8 Summary

In this chapter, we practiced creating and interpreting five core visualization types:

- Histograms for distributions of numerical variables.
- Line graphs for trends over time.
- Scatterplots for relationships between two numerical variables.
- Boxplots for comparing numerical distributions across categories.
- Bar charts for summarizing categorical data and comparing groups.

We also learned to:

- Gather related variables into long format for grouped visualizations.
- Use stacked bar charts to compare categories.
- Apply faceting to explore subgroup patterns.

These tools are essential for exploring data in the social sciences and for communicating findings clearly and responsibly.

6 Practice Problems

6.1 RStudio

Complete each task below to familiarize yourself with RStudio:

1. Open the help file for the `mean` function with `?mean`.
2. Create a simple plot with `plot(1:5)`.
3. Change the RStudio theme using `Tools` → `Global Options` → `Appearance`

6.2 Tidyverse

Be sure to use the codebook for the GSS data subset provided at [the author's GitHub page](#). It contains variable names and definitions which will be useful for completing practice activities involving the dataset. At times it may be best to use `slice_sample(n=10)` before knitting to avoid cumbersome output.

4. Select only the variables needed to explore whether people's views on whether women are not suited for politics (`fepol`) have changed over time (`year`). Return a random sample of 10 rows.
5. How many respondents in 2018 reported that they are "very happy"? Keep only that `year` and happiness category (`happy`). Return the resulting dataset with only the variables used for filtering.
6. Create a variable `above_med_inc` for whether a respondent's `income` bracket is above or below the median income bracket in the dataset. Return a sample of 10 rows from the resulting dataset containing only `income` and `above_med_inc`.
7. What is the average number of children (`childs`) for each `marital` status?
8. Arrange `marital` status categories by average `age` (oldest to youngest).
9. Find all respondents whose job title (`occ10`) includes "nurse" (case-insensitive). Return a random sample of 10 rows from the resulting dataset that contains only `occ10`, `income`, and `age`.
10. Replace "telephone" with "phone" in the occupation variable (`occ10`). Return a random sample of 10 rows from the resulting dataset that contains only `occ10` and `income` filtered to only those occupations whose titles include "phone".
11. **Challenge** Define "young professionals" as respondents aged 25–40 whose occupation contains "manager," "teacher," or "engineer." How has the number of young professionals changed over time?

6.3 Cleaning

12. How many rows and columns does the dataset have? What type of variable is `degree`?
13. Create a new dataset that removes any rows with missing values for `income`. Return a random sample of 10 rows with the variables `income` and `age`
14. Use `pivot_longer()` to create a dataset with `happy` and `health` responses in a single column called `response_type`. Return a random sample of 10 rows of the resulting dataset with the two new variables.

15. In the `happy` variable, change “not too happy” to “unhappy” and return the counts for each category using this new level.
16. **Challenge:** Create a dataset with only respondents from 2010–2020, remove any rows with missing age, and then calculate the average age for each marital status.

6.4 Data Visualization and Summarization

17. Use `realinc` (respondent’s inflation-adjusted income) to visualize the income distribution. Set `binwidth = 10000`, use a different fill and color, and add appropriate axis labels and a title. What shape does the income distribution take? Is it symmetric or skewed? Are there clusters or long tails that suggest income inequality?
18. Create a line graph to examine how confidence in the education system (`coneduc`) has changed across survey years. First mutate the data so that one represents hardly any confidence and three represents a great deal of confidence. Then group the data by year and calculate the mean value of `coneduc` for each year. Plot these averages as a line graph with points added along the line. Use a custom color to make the graph visually distinctive. Be sure to include a descriptive title as well as appropriate axis labels so that the plot is easy to interpret. Do you notice any periods where confidence in the education system increased or decreased sharply? How might these changes connect to historical or social events?
19. First create a stacked bar chart that shows the distribution of confidence in Congress (`conlegis`) responses by race, then facet this plot by sex. Do changes in confidence in congress across races remain consistent across sexes?
20. Create a boxplot to compare education across different racial groups. Use years of education (`educ`) as the numerical variable and race (`race`) as the grouping variable. Add a descriptive title and axis labels to clarify the purpose of the visualization. If helpful for interpretation, you may also order the groups in a meaningful way (for example, by median education). Which racial group shows the highest median education level? Are the differences between groups large or small compared to the spread within each group?

7 Appendix A: Shorter GSS Dataset Codebook

| Variable | Label / Meaning | Type | Common Values / Categories |
|-----------|---|-------------|---|
| year | Survey year | Numeric | 1972–2022 |
| age | Respondent's age | Numeric | 18–89+ |
| degree | Highest degree earned | Ordinal | lt high school, high school, junior college, bachelor, graduate |
| income | Respondent's income | Numeric | Continuous, self-reported |
| occ10 | Occupation code (2010 scheme) | Categorical | 4-digit census occupation codes |
| incomecat | Income grouped into categories | Ordinal | low, middle, high (varies by coding) |
| trustfam | Trust in family | Ordinal | not at all, only some, a lot |
| trustdoc | Trust in doctors | Ordinal | not at all, only some, a lot |
| wrkstat | Labor force status | Categorical | working full-time, part-time, unemployed, retired |
| id | Respondent identifier | Numeric | Unique ID |
| marital | Marital status | Categorical | married, widowed, divorced, separated, never married |
| agedwed | Age when first married | Numeric | 15–80+ |
| conjudge | Confidence in courts/judges | Ordinal | not at all, only some, a lot |
| realinc | Respondent's income (1986 constant dollars) | Numeric | Continuous |
| hrs1 | Hours worked last week | Numeric | 0–80+ |
| conpress | Confidence in press | Ordinal | not at all, only some, a lot |
| race | Race of respondent | Categorical | White, Black, Other |
| consci | Confidence in science | Ordinal | not at all, only some, a lot |
| fepol | Attitude: women suited for politics | Ordinal | agree, disagree |
| happy | General happiness | Ordinal | not too happy, pretty happy, very happy |
| childs | Number of children | Numeric | 0–8+ |
| health | Self-rated health | Ordinal | poor, fair, good, excellent |
| realinc | Family income (1986 constant dollars) | Numeric | Continuous |
| coneduc | Confidence in education system | Ordinal | not at all, only some, a lot |
| conlegis | Confidence in Congress | Ordinal | not at all, only some, a lot |
| sex | Respondent's sex | Categorical | male, female |
| educ | Years of education completed | Numeric | 0–20+ |

8 Appendix B: Practice Solutions

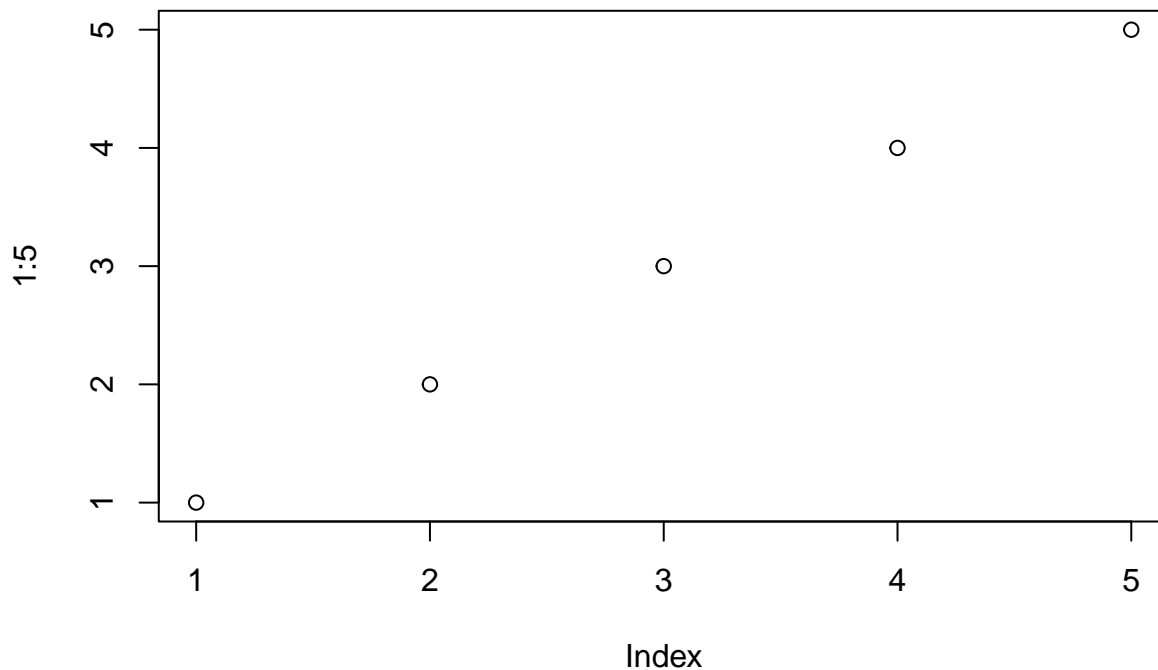
8.1 RStudio

1. Open the help file for the `mean` function with `?mean`.

```
?mean
```

2. Create a simple plot with `plot(1:5)`.

```
plot(1:5)
```



3. Change the RStudio theme using `Tools` → `Global Options` → `Appearance`

Use `Tools` → `Global Options` → `Appearance` to select an editor theme. `Tomorrow Night Bright` is a great option.

8.2 Tidyverse

4. Select only the variables needed to explore whether people's views on whether women are not suited for politics (`fepol`) have changed over time (`year`).

```
gssdat |>  
  select(fepol, year) |>  
  slice_sample(n = 10)
```

```
##      fepol year
## 1      <NA> 2021
## 2 disagree 2010
## 3      agree 1994
## 4      <NA> 2004
## 5      <NA> 2016
## 6      <NA> 2006
## 7      <NA> 1993
## 8 disagree 1998
## 9 disagree 2012
## 10     <NA> 2002
```

5. How many respondents in 2018 reported that they are “very happy”? Keep only that year and happiness category (happy). Return the resulting dataset with only the variables used for filtering.

```
gssdat |>
  filter(year == 2018, happy == "very happy") |>
  select(year, happy) |>
  slice_sample(n = 10)
```

```
##      year      happy
## 1  2018 very happy
## 2  2018 very happy
## 3  2018 very happy
## 4  2018 very happy
## 5  2018 very happy
## 6  2018 very happy
## 7  2018 very happy
## 8  2018 very happy
## 9  2018 very happy
## 10 2018 very happy
```

6. Create a variable `above_med_inc` for whether a respondent’s income bracket is above or below the median income bracket in the dataset. Return the resulting dataset containing only `income` and `above_med_inc`.

```
gssdat |>
  mutate(above_med_inc = income >= median(income, na.rm = TRUE)) |>
  select(above_med_inc, income) |>
  slice_sample(n = 10)
```

```
##      above_med_inc income
## 1          FALSE      4
## 2           TRUE     12
## 3          FALSE      9
## 4            NA     NA
## 5           TRUE     12
## 6           TRUE     12
## 7            NA     NA
## 8           TRUE     12
## 9            NA     NA
## 10           NA     NA
```

7. What is the average number of children (childs) for each marital status?

```
gssdat |>
  group_by(marital) |>
  summarize(mean_child = mean(childs, na.rm = TRUE))
```

```
## # A tibble: 6 x 2
##   marital      mean_child
##   <chr>         <dbl>
## 1 divorced      2.09
## 2 married       2.26
## 3 never married 0.541
## 4 separated    2.60
## 5 widowed      2.73
## 6 <NA>         1.69
```

8. Arrange marital status categories by average age (oldest to youngest).

```
gssdat |>
  group_by(marital) |>
  summarize(mean_age = mean(age, na.rm = TRUE)) |>
  arrange(desc(mean_age))
```

```
## # A tibble: 6 x 2
##   marital      mean_age
##   <chr>         <dbl>
## 1 widowed      70.3
## 2 <NA>        52.0
## 3 divorced    50.1
## 4 married     47.5
## 5 separated   44.5
## 6 never married 33.6
```

9. Find all respondents whose job title (occ10) includes “nurse” (case-insensitive). Return a random sample of 10 rows from the resulting dataset that contains only occ10, income, and age.

```
gssdat |>
  filter(str_detect(occ10, "nurse")) |>
  select(occ10, income, age) |>
  slice_sample(n = 10)
```

```
##           occ10 income age
## 1 registered nurses    12  53
## 2 registered nurses    12  27
## 3 registered nurses    12  41
## 4 registered nurses    12  40
## 5 registered nurses    12  42
## 6 licensed practical and licensed vocational nurses    12  34
## 7 registered nurses    12  35
## 8 registered nurses    12  44
## 9 registered nurses    12  46
## 10 registered nurses    12  30
```

10. Replace “telephone” with “phone” in the occupation variable (`occ10`). Return the resulting dataset that contains only `occ10` and `income` filtered to only those occupations whose titles include “phone”.

```
gssdat |>
  mutate(occ10 = str_replace(occ10, "telephone", "phone")) |>
  select(occ10, income) |>
  filter(str_detect(occ10, "phone")) |>
  slice_sample(n = 10)
```

```
##           occ10 income
## 1 phone operators    10
## 2 phone operators    12
## 3 phone operators     9
## 4 phone operators    NA
## 5 phone operators    10
## 6 phone operators    12
## 7 phone operators    11
## 8 phone operators    10
## 9 phone operators    12
## 10 phone operators     5
```

11. **Challenge** Define “young professionals” as respondents aged 25–40 whose occupation contains “manager,” “teacher,” or “engineer.” How has the number of young professionals changed over time?

```
gssdat |>
  filter((age <= 40 | age >= 25) & str_detect(occ10, regex("manager|teacher|engineer"))) |>
  select(year, occ10) |>
  group_by(year) |>
  summarize(count = n())
```

```
## # A tibble: 34 x 2
##   year count
##   <int> <int>
## 1 1972   189
## 2 1973   171
## 3 1974   180
## 4 1975   147
## 5 1976   191
## 6 1977   183
## 7 1978   159
## 8 1980   191
## 9 1982   212
## 10 1983   205
## # i 24 more rows
```

8.3 Cleaning

12. How many rows and columns does the dataset have? What type of variable is degree?

```
dim(gssdat)
```

```
## [1] 75699    27
```

```
gssdat |>  
  select(degree) |>  
  glimpse()
```

```
## Rows: 75,699  
## Columns: 1  
## $ degree <chr> "bachelor's", "less than high school", "high school", "bachelor~
```

13. Create a new dataset that removes any rows with missing values for income. Return a dataset with the variables income and age.

```
gssdat |>  
  filter(!is.na(income)) |>  
  select(income, age) |>  
  slice_sample(n = 10)
```

```
##   income age  
## 1     9  43  
## 2    11  42  
## 3    12  37  
## 4     8  46  
## 5    12  70  
## 6     9  73  
## 7    12  56  
## 8    12  60  
## 9    12  67  
## 10   12  42
```

14. Use `pivot_longer()` to create a dataset with happy and health responses in a single column called `response_type`. Return the resulting dataset with the two new variables.

```
gssdat |>  
  select(happy, health) |>  
  pivot_longer(cols = c(happy, health),  
               names_to = "response_type",  
               values_to = "response") |>  
  slice_sample(n = 10)
```

```
## # A tibble: 10 x 2  
##   response_type response  
##   <chr>         <chr>  
## 1 happy         very happy  
## 2 health        <NA>  
## 3 health        good  
## 4 health        excellent  
## 5 health        excellent
```

```
## 6 happy          pretty happy
## 7 health         fair
## 8 happy          very happy
## 9 happy          not too happy
## 10 happy         pretty happy
```

15. In the happy variable, change “not too happy” to “unhappy” and return the counts for each category using this new level.

```
gssdat |>
  mutate(happy = recode(happy,
                        "not too happy" = "unhappy")) |>
  count(happy)
```

```
##           happy      n
## 1 pretty happy 39705
## 2      unhappy 10095
## 3 very happy  21069
## 4          <NA>  4830
```

16. **Challenge:** Create a dataset with only respondents from 2010–2020, remove any rows with missing age, and then calculate the average age for each marital status.

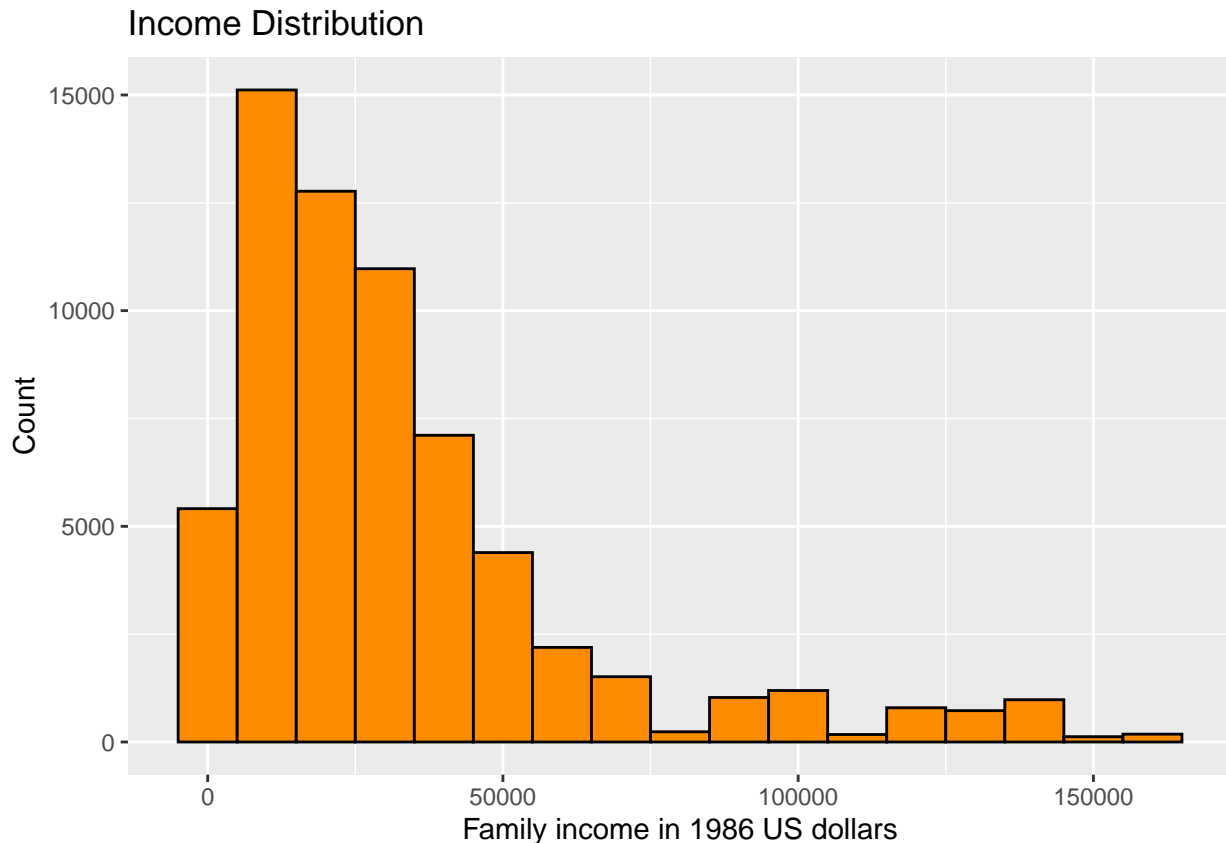
```
gssdat |>
  filter((year <= 2020 | year >= 2010) & !is.na(age)) |>
  group_by(marital) |>
  summarize(mean_age = mean(age))
```

```
## # A tibble: 6 x 2
##   marital      mean_age
##   <chr>         <dbl>
## 1 divorced      50.1
## 2 married       47.5
## 3 never married 33.6
## 4 separated     44.5
## 5 widowed       70.3
## 6 <NA>          52.0
```

8.4 Data Visualization and Summarization

17. Use `realinc` (respondent’s inflation-adjusted income) to visualize the income distribution. Set `binwidth = 10000`, use a different fill and color, and add appropriate axis labels and a title. What shape does the income distribution take? Is it symmetric or skewed? Are there clusters or long tails that suggest income inequality?

```
gssdat |>
  ggplot(aes(x = realinc)) +
  geom_histogram(fill = "darkorange", color = "black", binwidth = 10000) +
  labs(x = "Family income in 1986 US dollars", y = "Count", title = "Income Distribution")
```



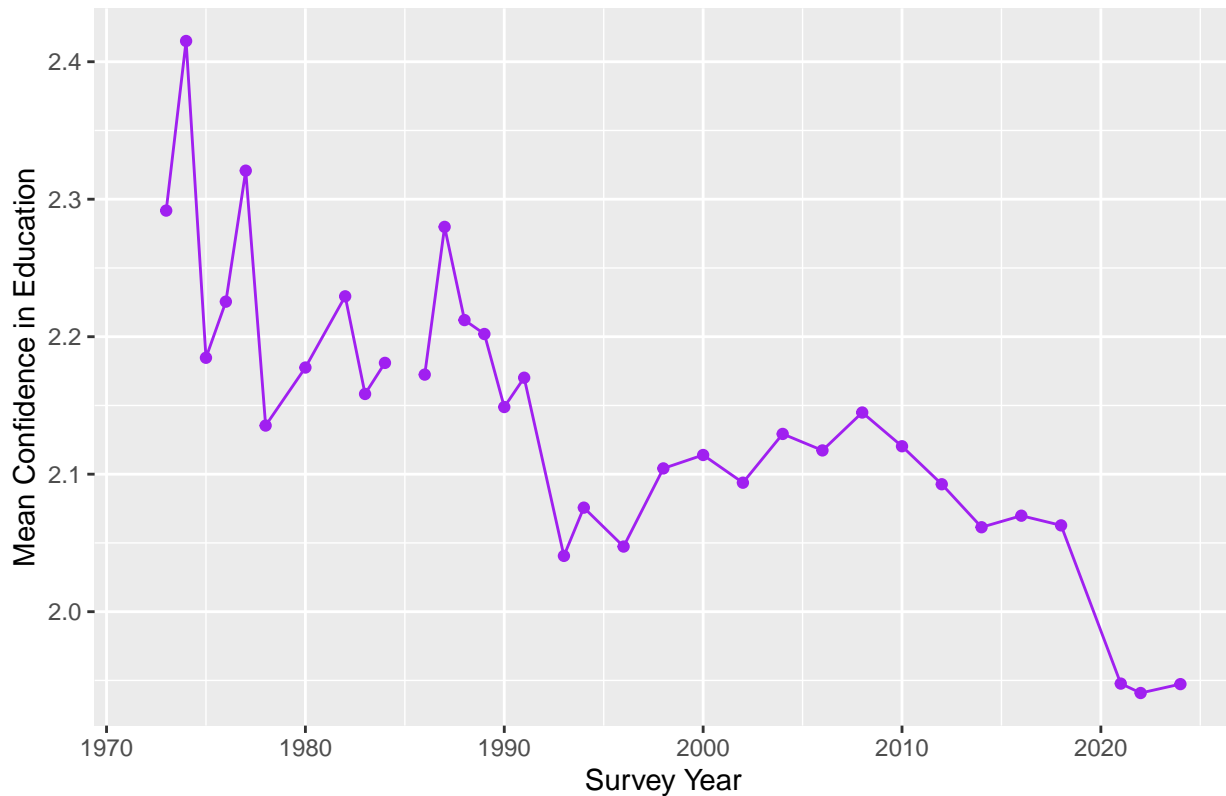
18. Create a line graph to examine how confidence in the education system (`coneduc`) has changed across survey years. First mutate the data so that one represents hardly any confidence and three represents a great deal of confidence. Then group the data by year and calculate the mean value of `coneduc` for each year. Plot these averages as a line graph with points added along the line. Use a custom color to make the graph visually distinctive. Be sure to include a descriptive title as well as appropriate axis labels so that the plot is easy to interpret. Do you notice any periods where confidence in the education system increased or decreased sharply? How might these changes connect to historical or social events?

```

gssdat %>%
  mutate(coneduc = recode(coneduc, `1` = 3L, `3` = 1L, `2` = 2L)) |>
  group_by(year) %>%
  summarise(mean_coneduc = mean(coneduc, na.rm = TRUE)) %>%
  ggplot(aes(x = year, y = mean_coneduc)) +
  geom_line(color = "purple") +
  geom_point(color = "purple") +
  labs(title = "Average Confidence in the Education System Over Time",
       x = "Survey Year",
       y = "Mean Confidence in Education")

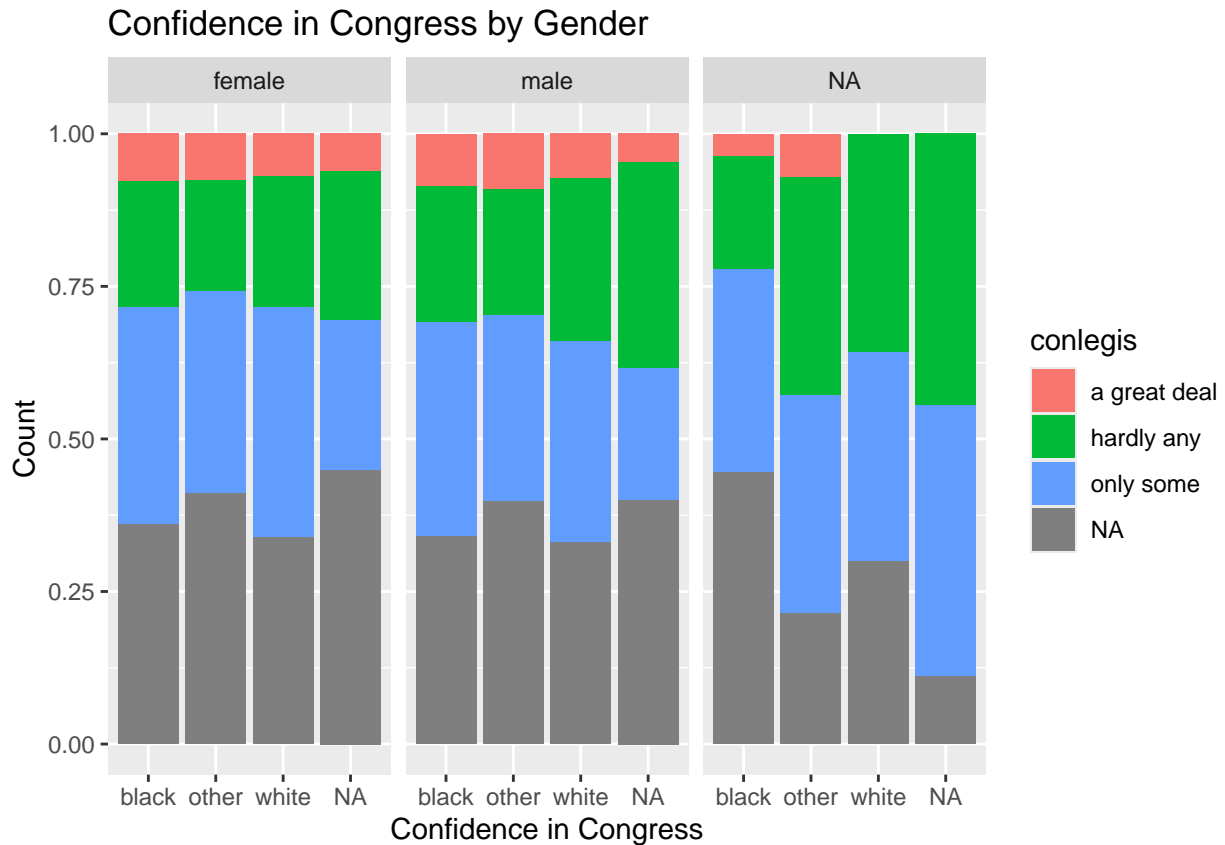
```

Average Confidence in the Education System Over Time



19. First create a stacked bar chart that shows the distribution of confidence in Congress (`conlegis`) responses by race, then facet this plot by gender. Do changes in confidence in congress across races remain consistent across genders?

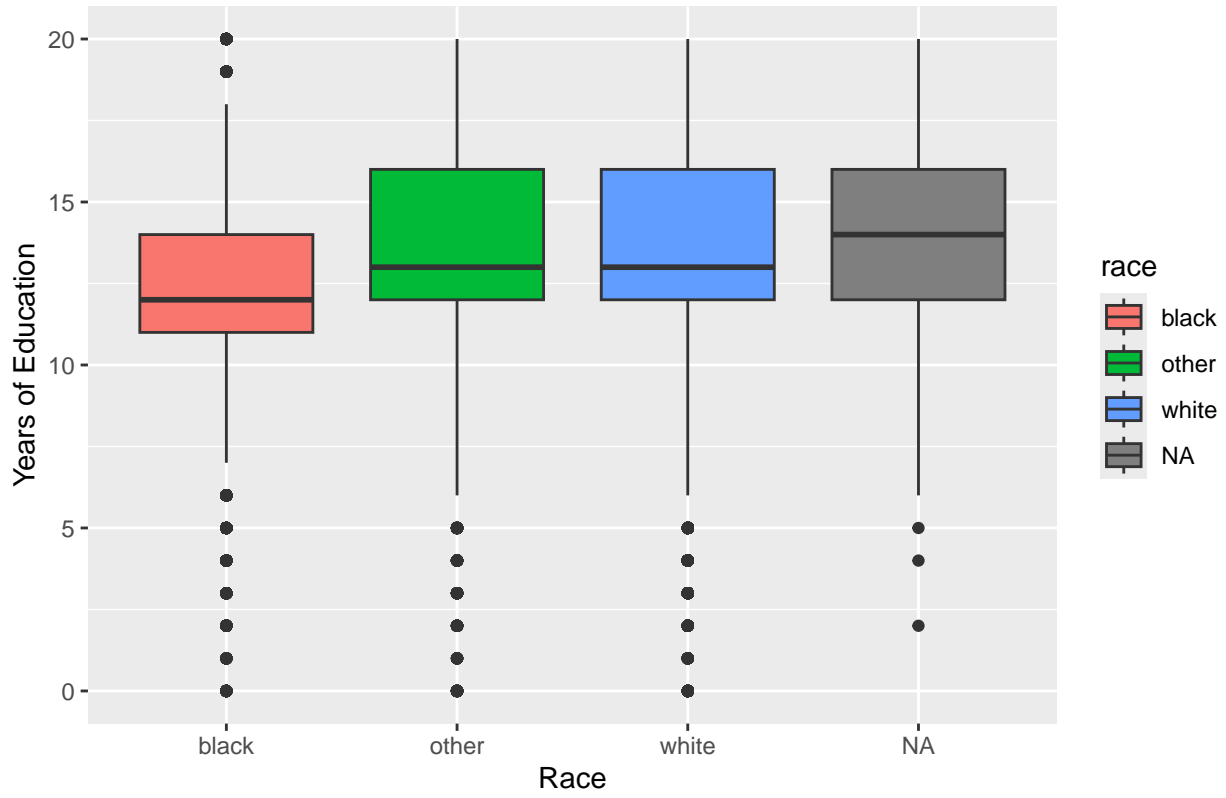
```
gssdat %>%  
  ggplot(aes(x = race, fill = conlegis)) +  
  geom_bar(position = "fill") +  
  facet_wrap(~sex) +  
  labs(title = "Confidence in Congress by Gender", x = "Confidence in Congress", y = "Count")
```



21. Create a boxplot to compare education across different racial groups. Use years of education (`educ`) as the numerical variable and race (`race`) as the grouping variable. Add a descriptive title and axis labels to clarify the purpose of the visualization. If helpful for interpretation, you may also order the groups in a meaningful way (for example, by median education). Which racial group shows the highest median education level? Are the differences between groups large or small compared to the spread within each group?

```
gssdat %>%
  ggplot(aes(x = race, y = educ, fill = race)) +
  geom_boxplot() +
  labs(
    title = "Distribution of Education by Race",
    x = "Race",
    y = "Years of Education"
  )
```

Distribution of Education by Race



9 Appendix C: Transitioning from Python to R Using AI

9.1 Why Use AI to Bridge R and Python?

Users of this manual may be learning programming for the first time, while others may already have experience in Python. Because this lab manual is written in R, those with prior Python experience may find themselves mentally “translating” code as they work. This appendix introduces a structured way to support that process using artificial intelligence (AI).

The goal of this chapter is not to replace learning, nor to encourage copying code without understanding it. Instead, the goal is to help readers use AI as a tool to deepen their understanding of both languages by focusing on the underlying concepts that they share.

9.1.1 R and Python: Different Syntax, Similar Ideas

R and Python are two of the most widely used programming languages in data analysis. While they differ in syntax and style, they rely on many of the same core ideas:

- Importing and inspecting data
- Selecting and filtering observations
- Creating new variables
- Summarizing information
- Visualizing results

For example, the following R code uses the `tidyverse` to filter and summarize data:

```
gssdat |>
  filter(year == 2022) |>
  summarize(avg_age = mean(age, na.rm = TRUE))
```

A Python user might expect to perform the same task using the `pandas` library. Although the syntax looks different, the logic is the same:

- Start with a dataset
- Filter rows
- Compute a summary statistic

Learning to recognize these shared ideas is more important than memorizing exact syntax.

9.1.2 Why People May Prefer Python

Python is widely used across industries such as technology, finance, and data science. Some readers may already be familiar with tools like:

- `pandas` for working with data

- NumPy for numerical operations

Because of this familiarity, translating R code into Python can make new concepts feel more accessible. However, it is important to recognize that R—especially through the tidyverse—offers a powerful and expressive system for data analysis that is widely used in research and the social sciences.

Rather than choosing one language over the other, this chapter encourages readers to build connections between them.

9.1.3 The Role of AI in Learning Code

AI tools such as ChatGPT and Copilot can help translate code between languages, explain unfamiliar syntax, and suggest alternative approaches.

However, AI is most useful when used actively, not passively.

- Passive use: copying AI-generated code without understanding it
- Active use: asking AI to explain, compare, and justify code

In this appendix, readers will learn how to:

- Write prompts that guide AI effectively
- Ask for explanations alongside code
- Check whether AI outputs are correct
- Use AI to strengthen their understanding of data analysis concepts

Think of AI as a tutor or collaborator, not a shortcut.

9.1.4 A Simple Example of AI-Assisted Translation

Suppose you encounter the following R code:

```
gssdat |>
  select(age, income) |>
  filter(age > 30)
```

Instead of manually rewriting this in Python, you could ask an AI tool:

Translate this R tidyverse code into Python using pandas and explain each step.

The key is not just the translation—it is the explanation that helps you understand how the two languages relate.

Later in this chapter, you will learn how to write prompts like this effectively and refine them when needed.

9.1.5 Academic Integrity and Responsible Use

AI is a powerful tool, but it must be used responsibly:

- Try to understand any code submitted
- Use AI to support learning, not replace it
- Always review and test AI-generated code

Using AI effectively is a professional skill. In many careers, knowing how to ask the right questions and evaluate automated outputs is just as important as writing code from scratch.

9.1.6 Key Takeaway

R and Python may look different, but they are built on the same analytical foundations. AI can help bridge the gap between them—but only if you use it thoughtfully.

In the next section, you will learn how to write effective prompts that allow AI to become a powerful learning tool rather than just a code generator.

9.2 How to Prompt AI for Code Translation

Effectively using AI to translate code from R to Python depends less on the tool itself and more on how the prompt is written. Clear, detailed prompts help AI generate more accurate, useful, and interpretable outputs. This section provides a framework for developing strong prompting habits.

9.2.1 The Anatomy of an Effective Prompt

An effective prompt for translating R code into Python typically includes three key elements:

1. Specify the task clearly > Indicate that the goal is to translate code from R to Python.
2. Provide context > Include information about the dataset, variables, or analytical goal.
3. Request explanation > Ask the AI to explain what the translated code is doing.

A general template is shown below:

```
Translate the following R (tidyverse) code into Python using pandas.  
Explain each step of the Python code and how it corresponds to the R code.
```

```
[R CODE HERE]
```

This structure encourages AI to act as both a translator and a teacher.

9.2.2 Example: Weak vs. Strong Prompts

The quality of AI output is highly sensitive to the quality of the prompt. Consider the following comparison:

Weak Prompt

Convert this to Python:

```
gssdat |> filter(year == 2022)
```

Issues:

- Does not specify which Python library to use
- Provides no explanation request
- Lacks context about the dataset or goal

Strong Prompt

Translate the following R tidyverse code into Python using pandas.
The dataset is called gssdat and contains survey data.
Explain how the Python code filters the data and how it compares to the R code.

```
gssdat |> filter(year == 2022)
```

Why this works better:

- Specifies tidyverse → pandas translation
- Provides dataset context
- Requests conceptual explanation
- Encourages understanding, not just output

9.2.3 Adding Context Improves Results

AI performs better when it understands the purpose of the code. For example:

Translate the following R code into Python using pandas.
This code selects age and income variables and filters respondents over age 30.
Explain each step and describe the output.

```
gssdat |>  
  select(age, income) |>  
  filter(age > 30)
```

Including the purpose (“filters respondents over age 30”) helps the AI produce clearer and more accurate explanations.

9.2.4 Asking for Explanation, Not Just Code

One of the most valuable uses of AI is its ability to explain relationships between languages. This can be done using prompts that include phrases such as:

Explain each step

Compare this to the R version

Describe what the output will look like

This shifts the focus from code generation to conceptual understanding, which is especially important when learning how different programming languages implement similar ideas.

9.2.5 Iterative Prompting: Refining the Output

AI prompting is often an iterative process. The first response may not be perfect, and follow-up prompts can improve clarity and accuracy.

Step 1: Initial Prompt

Translate this R code into Python using pandas:

```
gssdat |> group_by(degree) |> summarize(avg_age = mean(age, na.rm = TRUE))
```

Step 2: Follow-Up Prompt

Can you rewrite the Python code using method chaining?

Also explain how missing values are handled in Python compared to R.

Step 3: Clarification Prompt

What does the groupby function return before aggregation?

This step-by-step refinement helps build deeper understanding and mirrors how analysts troubleshoot and learn in real-world settings.

9.2.6 Common Mistakes When Using AI

- Vague prompts
("Convert this") without specifying tools or goals
- No request for explanation
Leads to surface-level learning
- Ignoring errors in output

AI-generated code may contain mistakes or assumptions

- Overly complex inputs at the start

Large code blocks can produce confusing responses; start small

- Assuming one-to-one translation

Not all R functions map directly to Python equivalents

Recognizing these issues can help you develop more effective habits when working with AI.

9.2.7 Mini Activity: Improving a Prompt

Weak Prompt

Turn this into Python:

```
gssdat |> mutate(young = age < 30)
```

Reflection Questions

- What information could make this prompt more effective?
- How might different prompts produce different outputs?
- What additional clarification could be added if the output is unclear?

Revise this prompt using your reflections. When you are done see how your answer compares to the example answer below.

Example Improved Prompt

Translate the following R tidyverse code into Python using pandas.

The dataset `gssdat` contains survey data.

This code creates a new variable indicating whether a respondent is under age 30.

Explain how the new variable is created in Python.

```
gssdat |> mutate(young = age < 30)
```

9.2.8 Key Takeaway

Prompting is a skill that improves with practice. Clear, specific, and context-rich prompts allow AI to function as a powerful tool for translating and understanding code. By emphasizing explanation, iteration, and critical evaluation, instructors can help learners use AI to build transferable programming knowledge across R and Python.

9.3 Translating Core Data Tasks

One of the most effective ways to move between R and Python is to focus on **concepts rather than syntax**. While the code may look different across languages, the underlying tasks—importing data, filtering rows, creating variables, and summarizing results—remain the same.

AI can assist in this process, but its greatest value lies in helping users recognize **how ideas transfer**, not just producing equivalent code. This section demonstrates how to use AI to translate core data tasks while maintaining a focus on conceptual understanding.

9.3.1 Data Import and Inspection

The first step in any data analysis workflow is importing and inspecting data.

R Example

```
gssdat <- read.csv("files/gssdat.csv")

dim(gssdat)
names(gssdat)
summary(gssdat$age)
```

AI Prompt Example

Translate the following R code into Python using pandas.
The dataset is a CSV file called gssdat.csv.
Explain how the Python code imports the data and inspects its structure.

```
gssdat <- read.csv("files/gssdat.csv")

dim(gssdat)
names(gssdat)
summary(gssdat$age)
```

Resulting Python Code (Typical Output)

```
import pandas as pd

gssdat = pd.read_csv("files/gssdat.csv")

gssdat.shape
gssdat.columns
gssdat["age"].describe()
```

| Task | R | Python |
|--------------------|------------|---------------|
| Import data | read.csv() | pd.read_csv() |
| Dimensions | dim() | .shape |
| Column names | names() | .columns |
| Summary statistics | summary() | .describe() |

Key Insight

Although the syntax differs, both languages:

- Load a dataset into memory
- Provide tools to inspect structure
- Allow quick summaries of variables

AI is most useful when asked to explain these parallels, helping learners see that the workflow—not the syntax—is what transfers across languages.

9.3.2 Data Wrangling Concepts

Data wrangling is central to this lab manual. The `tidyverse` provides a consistent grammar for these tasks, while Python commonly uses the `pandas` library.

Rather than memorizing translations, it is helpful to map core ideas.

| Task | R (tidyverse) | Python (pandas) |
|-------------------|---------------------------------------|-------------------------------------|
| Select columns | <code>select()</code> | <code>df[["col1", "col2"]]</code> |
| Filter rows | <code>filter()</code> | <code>df[df["col"] == value]</code> |
| Create variables | <code>mutate()</code> | <code>df["new"] = ...</code> |
| Group + summarize | <code>group_by() + summarise()</code> | <code>groupby().agg()</code> |

Example: Filtering and Selecting

R Code

```
gssdat |>
  select(year, age, income) |>
  filter(year == 2022)
```

AI Prompt

Translate the following R tidyverse code into Python using pandas.

This code selects the variables `year`, `age`, and `income`, then filters the data to only include rows from 2022. Explain how each step works in Python.

```
gssdat |>
  select(year, age, income) |>
  filter(year == 2022)
```

Python Output (Typical)

```
gssdat[["year", "age", "income"]][gssdat["year"] == 2022]
```

Example: Creating a New Variable

R Code

```
gssdat |>
  mutate(young_adult = age >= 18 & age <= 30)
```

Python Output (Typical)

```
gssdat["young_adult"] = (gssdat["age"] >= 18) & (gssdat["age"] <= 30)
```

Example: Grouping and Summarizing

R Code

```
gssdat |>  
  group_by(degree) |>  
  summarise(avg_age = mean(age, na.rm = TRUE))
```

Python Output (Typical)

```
gssdat.groupby("degree")["age"].mean()
```

Reflection: What Stayed the Same?

Across all examples:

- A dataset is the starting point
- Operations are applied step-by-step
- The goal is to transform or summarize data

What changes is **how the steps are written**, not **what the steps are doing**.

Encouraging reflection on these similarities helps build transferable skills. AI can support this by explaining *why* a translation works, not just *what* the translated code is.

9.3.3 Pipes vs. Method Chaining

One of the most noticeable differences between R and Python is how multiple operations are linked together.

R: Pipe Operator

```
gssdat |>  
  filter(year == 2022) |>  
  group_by(degree) |>  
  summarise(avg_age = mean(age, na.rm = TRUE))
```

The pipe (`|>`) passes the result of one step into the next, creating a readable sequence of operations.

Python: Method Chaining

In Python, similar workflows are often written using method chaining:

```
gssdat[gssdat["year"] == 2022] \  
  .groupby("degree")["age"] \  
  .mean()
```

AI Prompt to Highlight Structure

Translate the following R code into Python using pandas method chaining. Explain how each step in the chain corresponds to the pipe operator in R.

```
gssdat |>
  filter(year == 2022) |>
  group_by(degree) |>
  summarise(avg_age = mean(age, na.rm = TRUE))
```

9.3.4 How AI Supports Understanding

When prompted effectively, AI can:

- Break down each step in the pipeline
- Show how intermediate results flow through the code
- Compare readability and structure across languages

This reinforces a key idea: both R and Python support sequential data transformations, even if they express them differently.

9.3.5 Key Takeaway

Translating between R and Python is most effective when approached as concept mapping rather than direct substitution. AI can assist in this process by making relationships between tasks explicit, helping users recognize that:

- Data workflows are consistent across languages
- Syntax differences are secondary to conceptual understanding
- Clear prompts lead to clearer explanations and better learning outcomes

In the next section, this approach will be extended to data visualization, where stylistic differences between languages become more pronounced but the underlying goals remain the same.

9.4 Translating Visualizations

Data visualization is one of the areas where differences between R and Python are most visible. While both languages are capable of producing high-quality graphics, they follow different design philosophies.

In R, visualizations are commonly created using the **grammar of graphics** implemented in `ggplot2`. In Python, visualizations are often created using libraries such as `matplotlib` and `seaborn`.

AI can help translate between these systems, but it is especially useful for explaining how the same visual idea is expressed differently across languages.

9.4.1 Conceptual Differences: Grammar vs Layers

R (`ggplot2`) builds plots layer-by-layer using a consistent grammar:

- Data
- Aesthetic mappings (`aes`)
- Geometric objects (`geom_*`)
- Labels and themes

Python (`matplotlib/seaborn`) is often more flexible but less standardized:

- Plots are created through function calls or method chaining
- Styling and customization may require additional steps

Despite these differences, both approaches aim to:

- Represent data visually
- Highlight patterns and relationships
- Communicate findings clearly

9.4.2 Translating Histograms

R Code

```
gssdat |>
  ggplot(aes(x = age)) +
  geom_histogram(binwidth = 5, fill = "steelblue", color = "white") +
  labs(title = "Age Distribution",
       x = "Age",
       y = "Count")
```

AI Prompt Example

Translate the following R `ggplot2` code into Python using `seaborn` or `matplotlib`. Explain how the histogram is constructed and how the `binwidth` argument is handled in Python.

```
gssdat |>
  ggplot(aes(x = age)) +
  geom_histogram(binwidth = 5, fill = "steelblue", color = "white") +
  labs(title = "Age Distribution",
       x = "Age",
       y = "Count")
```

Python Output (Typical)

```

import seaborn as sns
import matplotlib.pyplot as plt

sns.histplot(data=gssdat, x="age", binwidth=5, color="steelblue", edgecolor="white")

plt.title("Age Distribution")
plt.xlabel("Age")
plt.ylabel("Count")
plt.show()

```

Comparison

- Both specify the variable (age)
- Both control bin size (binwidth)
- Both customize labels and appearance

The main difference is structural:

- R layers components with +
- Python uses sequential function calls

9.4.3 Translating Scatterplots

R Code

```

gssdat |>
  ggplot(aes(x = age, y = realrinc)) +
  geom_point(alpha = 0.3) +
  labs(title = "Age and Income",
       x = "Age",
       y = "Income")

```

AI Prompt Example

Convert this ggplot2 scatterplot into Python using seaborn.
Explain how transparency (alpha) is handled and how the axes are defined.

```

gssdat |>
  ggplot(aes(x = age, y = realrinc)) +
  geom_point(alpha = 0.3)

```

Python Output (Typical)

```

sns.scatterplot(data=gssdat, x="age", y="realrinc", alpha=0.3)

plt.title("Age and Income")
plt.xlabel("Age")
plt.ylabel("Income")
plt.show()

```

Comparison

- The mapping of variables (x, y) is conceptually identical
- Transparency (alpha) exists in both languages
- Labels are added separately in Python

This highlights a key insight: the visual relationship stays the same even when syntax changes.

9.4.4 Translating Bar Charts

R Code

```
gssdat |>
  ggplot(aes(x = race, fill = conpress)) +
  geom_bar(position = "fill") +
  labs(title = "Confidence in the Press by Race",
        x = "Race",
        y = "Proportion")
```

AI Prompt Example

Translate this stacked bar chart from ggplot2 into Python using seaborn. Explain how the "fill" position (proportions) is represented in Python.

```
gssdat |>
  ggplot(aes(x = race, fill = conpress)) +
  geom_bar(position = "fill")
```

Python Output (Typical)

```
import pandas as pd

prop_data = (
    gssdat
    .groupby(["race", "conpress"])
    .size()
    .groupby(level=0)
    .apply(lambda x: x / x.sum())
    .reset_index(name="proportion")
)

sns.barplot(data=prop_data, x="race", y="proportion", hue="conpress")

plt.title("Confidence in the Press by Race")
plt.xlabel("Race")
plt.ylabel("Proportion")
plt.show()
```

Comparison

- R handles proportions automatically with `position = "fill"`
- Python often requires manual calculation of proportions
- Both ultimately display the same conceptual relationship

This example illustrates that translation is not always one-to-one. AI can help explain these differences and suggest appropriate workflows.

9.4.5 Stylistic vs Conceptual Differences

When translating visualizations, it is helpful to distinguish between:

Conceptual Elements (Shared)

- What variables are being plotted
- What relationship is being shown
- What type of graph is used

Stylistic Elements (Different)

- Default colors and themes
- How layers are added
- How labels and legends are formatted

AI can be especially useful for:

- Identifying which parts of a plot are conceptual vs stylistic
- Explaining why a translation may require additional steps
- Suggesting improvements for clarity or aesthetics

9.4.6 Using AI to Compare Outputs

A useful prompting strategy is to ask AI to compare plots directly:

```
Compare the ggplot2 and seaborn versions of this visualization.  
What are the main differences in structure, defaults, and flexibility?
```

This encourages deeper analysis and helps users move beyond surface-level translation.

9.4.7 Key Takeaway

Visualizations in R and Python may look different in code, but they are built on the same goals: representing data clearly and effectively. AI can support translation by:

- Converting syntax between libraries
- Explaining structural differences
- Highlighting conceptual similarities

By focusing on *what the visualization is doing*, rather than just *how it is written*, users can develop flexible skills that transfer across tools, languages, and analytical contexts.

9.5 Becoming an Independent Learner with AI

AI can be a powerful support tool for translating code and learning new programming languages. However, its long-term value depends on how it is used. The goal is not simply to generate correct code, but to develop the ability to **evaluate, refine, and learn from AI-generated outputs**.

This section focuses on building habits that support independent learning across R, Python, and other analytical tools.

9.5.1 Verifying AI-Generated Code

AI-generated code should always be treated as a **first draft**, not a final answer. Verification is an essential step in any workflow.

Some practical strategies include:

- **Run the code and inspect the output**

Does the result match expectations? Are there unexpected values or missing results?

- **Check intermediate steps**

Break the code into smaller parts and examine what each step produces.

- **Compare with known results**

If the same task has already been completed in R, compare outputs between R and Python.

- **Look for warnings or silent errors**

Some issues (such as missing values) may not produce explicit errors but can still affect results.

AI can also assist in verification when prompted carefully:

```
This Python code was generated from R.  
Can you check if it correctly replicates the logic and identify any potential issues?
```

This encourages AI to act as a reviewer rather than just a generator.

9.5.2 Debugging Strategies

Errors are a natural and valuable part of the learning process. AI can support debugging when used interactively.

Ask AI to Explain Errors

Instead of asking for a corrected version immediately, it is often more helpful to ask for an explanation:

```
I received this error in Python.  
Can you explain what it means and what might be causing it?
```

```
[ERROR MESSAGE HERE]
```

Understanding the cause of an error builds transferable problem-solving skills.

Compare Outputs Between R and Python

When translating code, differences in results can reveal important insights.

For example:

- Are missing values handled the same way?
- Are group summaries calculated identically?
- Are categorical variables treated consistently?

A useful prompt might be:

```
The R and Python versions of this analysis produce different results.  
What are possible reasons for the discrepancy?
```

This encourages deeper thinking about how each language processes data.

9.5.3 When Not to Rely on AI

AI is most useful when supporting learning, but there are situations where reliance should be limited:

- When foundational concepts are unclear

It is often more effective to revisit basic ideas before translating code

- When outputs are accepted without question

AI can produce incorrect or incomplete code

- When tasks become overly automated

Over-reliance can reduce opportunities to practice problem-solving

- When documentation provides clearer answers

Official resources are often more precise for specific functions or syntax

Recognizing these situations helps maintain a balance between efficiency and understanding.

9.5.4 Building Long-Term Skills

AI should be integrated into a broader learning strategy rather than used in isolation.

Reading Documentation

Both R and Python have extensive documentation that explains functions, arguments, and examples in detail. Learning to navigate documentation helps users:

- Verify AI-generated suggestions
- Explore alternative approaches
- Build confidence in independent problem-solving

AI can assist here as well:

Can you summarize the key points of the pandas groupby documentation and explain when to use it?

9.5.5 Using AI as a Tutor, Not a Shortcut

AI is most effective when treated as a learning partner. Productive uses include:

- Asking for step-by-step explanations
- Requesting comparisons between methods
- Exploring “what if” scenarios
- Breaking down complex code into smaller pieces

Less productive use involves copying code without engaging with the underlying ideas.

9.5.6 Key Takeaway

AI can accelerate learning and support transitions between programming languages, but its effectiveness depends on active engagement. By verifying outputs, debugging thoughtfully, and using AI as a guide rather than a replacement, learners can build flexible, durable skills that extend beyond any single tool or language.

9.6 Conclusion

The ability to work effectively with AI is increasingly relevant across many fields. This final section encourages the reader to reflect on how learning to use AI as part of the data analysis workflow could help prepare them for their career.

Think about each of the following questions and discuss your answers with a fellow learner:

- How does using AI change the way problems are approached in data analysis?
- What strategies help ensure that AI-generated results are accurate and reliable?
- In what situations might human judgment be more important than automated output?
- How can AI be used to support collaboration and communication in professional settings?
- What skills remain essential, even when AI tools are available?