R for Korean Studies: A Gentle Introduction to Computational Social Science

Draft Version: 0.0.2

Kadir Jun Ayhan, Ph.D.

2024-10-18

Table of contents

Preface

Korean Studies is traditionally dominated by scholars of history and literature. It's relatively rare to see R, Python, or other computational social science tools being used or taught in this field.

I believe computational social science offers huge opportunities for Korean Studies, not only for quantitative research but also for qualitative studies, including those on history and literature!

In this book, I aim to increase data literacy and convince as many Korean Studies scholars and students as possible about the relative ease of learning R with code samples, and motivational case studies about Korea.

This book is supposed to be a gentle introduction, so I do not go into the details of the R language. You can refer to the links that I provide in this book for more information. Furthermore, I also strongly encourage you to use `Github's Copilot` which is free for academic use, `Chatgpt` which is not necessarily a coding bot, but still helpful especially for simple tasks, `Stackoverflow`, and `Google` for help whenever you are stuck or come across an error.

I also encourage you to join our bootcamps for problem solving! You can sign up for my newsletter to get updates on the workshops.

Current Status of the book

- 0.Preface: Done
- 9.Text Analysis: 50% Done
- 14.Making Korean Data Visualization Social: Done
- 15.Bootcamp: Done

I will complete the Text Analysis chapter and then move on to the next chapters. Subscribe to my newsletter to get updates on the book and the bootcamps.

How to Cite This Book

You can freely use this book, but you must cite my work to avoid plagiarism. You can cite it in the following way: (Ayhan 2024).

```
@book{ayhan_2024_r4ks,
  title = {R for {Korean Studies}: A Gentle Introduction to {Computational Social Science}},
  author = {Ayhan, Kadir Jun},
  year = {2024},
  month = {May},
  edition = {Draft Version 0.0.1},
  url = {https://r4ks.com}
}
```

1 Introduction

> Recently, I had to repeat myself while talking to a few students about Ewha GSIS Computational Social Science Workshop(s). Now, you and future students have this post instead!

This is what I wrote in my blog post about Ewha GSIS Computational Social Science Workshops that I have organized.

Following the same spirit in David Robinson's tweet, I decided to write this book.

1.1 "Why Do I Need Computational Tools in Korean Studies?"

Simply put, there is so much more data out there that is useful for Korean Studies research, and we have faster computers, and handy tools to analyze such data.

Korean Studies curricula across the world are quite rich and interdisciplinary. Those courses often equip students with the history, culture, literature, and language of Korea to understand the country better. Yet, Korean Studies scholars and students are not exposed to computational methods/ tools that can handle big or complex data as much.

If you are already here, it probably means that you appreciate the increasing importance of the computational tools in your research. This book, and bootcamps based on this book, will teach you the basics of R, and give you sample codes based on Korean Studies-related examples.

In the age that we live in, I strongly believe that these computational methods/ tools will empower you in your research as well as in the job market given wide range of prospective jobs Korean Studies graduates seek and find (corporations, international organizations, think tanks, NGOs, media, academia etc.).

1.2 "Why R?"

R is free! There are so many packages that are rich with a wide range of functions that you would need in all kinds of research, analysis, and reporting. Many more are being built as you read this book! You can do from simple math to data pre-processing, from data visualization to regressions, from building your CV to building your website, from analyzing tweets to machine learning.

Python is probably getting more popular in the industry jobs in recent years. Yet, I think, for the time being, R is better suited for social science research. At least there are more books, tutorials, examples that you can learn from in terms of social sciences.

Once exposed to R, you may also consider learning Python as well if it seems more attractive for you.

1.3 "I don't know anything about coding! Indeed, I am frustrated about coding!"

Then this book, and the bootcamps, are very much for you! I don't expect the readers, and bootcamp participants, to have any prior knowledge of R, coding, or other statistical software.

This book is supposed to be a gentle introduction, so I do not go into the details of the R language. You can refer to the links that I provide in this book for more information. Furthermore, I also strongly encourage you to use `Github's Copilot` which is free for academic use, `ChatGPT` which is not necessarily a coding bot, but still helpful especially for simple tasks, `Stackoverflow`, and `Google` for help whenever you are stuck or come across an error.

Learning curve is steep in the beginning. So you may need a trigger to begin and NOT GIVE UP. This book plays this trigger role. So, there is no need to be intimidated by R, or your lack of background with coding. I got you covered!

2 Setting Up

Both R and Rstudio are free to use, and setting them up is quite straightforward.

R is a programming language and software environment, produced mainly for statistical computing and graphics. RStudio is an integrated development environment (IDE) for R (as well as for other programming languages including Python, Stan, Julia and others).

In order to use R, installing RStudio is not enough. Installing R is enough, but RStudio is recommended for a better experience.

You can use R in other IDEs, such as VS Code as well, but RStudio is the most popular and widely used IDE for R.

2.1 Installing R

You need to install R on your computer. You can download the latest version of R from the CRAN website by clicking one of the mirror links in a location that is close to you. In the next page, you can download the installer for your operating system (Windows, Mac, or Linux).

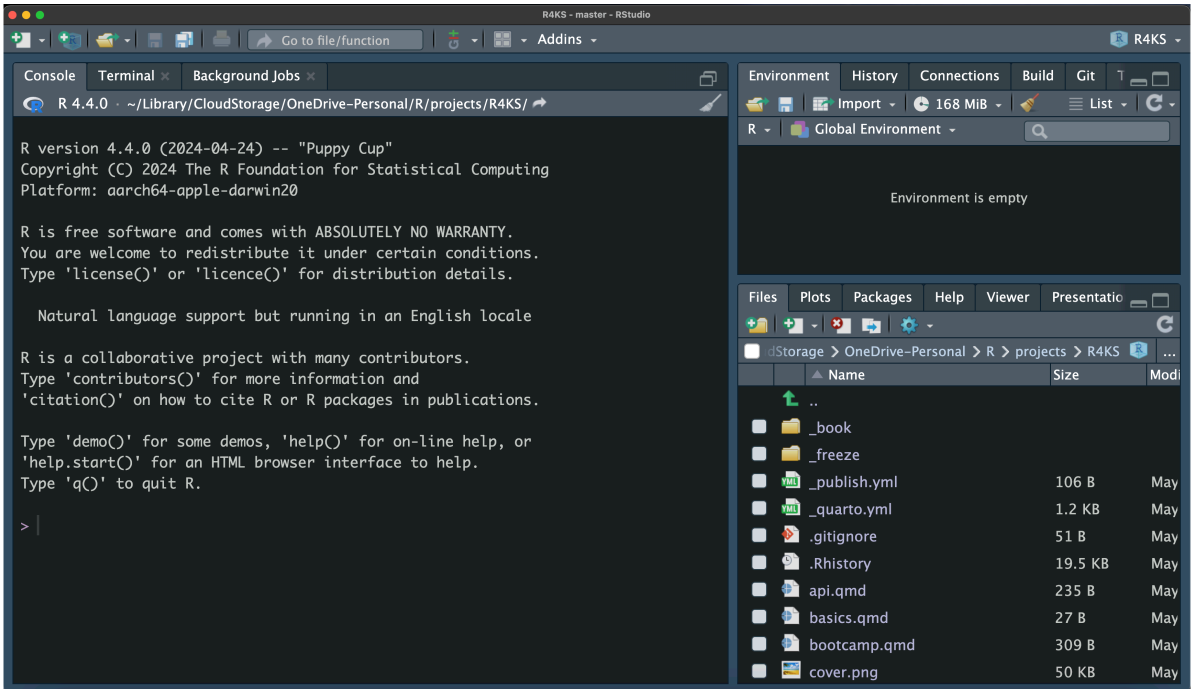
2.2 Installing RStudio

After installing R, you can download RStudio from the RStudio website. You can download the free version of RStudio Desktop by clicking 2: Install RStudio on the right. It automatically recognizes your operating system and downloads the correct installer for you.

2.3 Running R on RStudio

After installing R and RStudio, you can open RStudio and start using R.

When you open RStudio, you will see something like in the Figure 2.1.



```
R version 4.4.0 (2024-04-24) -- "Puppy Cup"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Name	Size	Modi
..		
_book		
_freeze		
_publish.yml	106 B	May
_quarto.yml	1.2 KB	May
.gitignore	51 B	May
.Rhistory	19.5 KB	May
api.qmd	235 B	May
basics.qmd	27 B	May
bootcamp.qmd	309 B	May
cover.png	50 KB	May

Figure 2.1: RStudio

Well, what you will see will be a default white screen, but you can customize it to look like the one in the image. You can change the theme of RStudio by going to `Tools > Global Options > Appearance` and selecting the theme you like.

For now, pay attention to the two panes in RStudio:

1. Console: This is where you can write your R code. For example try writing `1+1` and clicking enter there. You will see the result in the console.
2. Environment: This is where you can see the objects you have created in your R session. For example, if you write `x <- 5` (that is assigning the number '5' to an object named 'x') in the source pane, you will see `x` in the environment pane.

Check out the The Basics of R chapter to learn the basics of R.

2.4 Further Information

You can refer to the following video for further help on installing R and RStudio, unless above information is enough.

https://youtu.be/uIIv0NiVTs4

3 Korean Studies Data Sources

3.1 Statistical Data

3.2 Text Data

4 The Basics of R

In this chapter, we learn the basics of R.

4.1 Creating a Project

For each of your new projects, you should create a new project in RStudio. To do this, click on the “File” menu, then “New Project”. You will be asked to choose a directory for your project. Choose a directory where you want to store your project files. You can also create a new directory for your project. After you have chosen a directory, click on “Create Project”. You will see a new RStudio window with your project. You can now start working on your project.

For now, this is all you need to know about creating a project. We learn more about projects that are connected with Github in the Productivity Tools chapter.

When you work within a project, managing the files for your project becomes easier. When you work within the project, you don’t need to worry about getting and setting your working directory. To give you an idea, this is how you can find the working directory for your project:

```
getwd()
```

```
[1] "/Users/pd/Library/CloudStorage/OneDrive-Personal/R/projects/R4KS"
```

You can also set the working directory to some other path using the `setwd()` function. But you don’t need to do this when you work within a project. On another note, when you are writing code script in an R script file or within a code chunk, you can add non-code comments like this by adding a # sign at the beginning of the line.

```
# You can uncomment a comment line and make it a code line by removing the '#' sign at the be

# Replace "path/to/your/directory" with the actual path to your directory (folder) that you w

# Try removing the '#' sign at the beginning of the line and running the code.

# setwd("path/to/your/directory")
```

When you work within a project, you don't need to worry about the working directory. You can store all the files for your project in the project directory. You can also save your R scripts in the project directory. This way, you can easily find the files for your project.

4.2 Scripting in R

You can simply type your R code in the console and press Enter to run the code. But this is not a good practice. You should write your code in a script file and then run the script file. This way, you can save your code and run it again whenever you want. You can also share your code with others.

One of the most important advantages of R, for example over Excel, is that you can reproduce your results. That's why you should write your code in a script file. Every time you exit R, you should save your R script(s) and then rely on them next time you work on the same project.

4.2.1 Creating a New R Script

The most basic way to create a new R script is to click on the "File" menu, then "New File", and then "R Script". You will see a new R script file in the RStudio editor. You can now write your R code in this file.

4.2.2 Creating a Quarto File

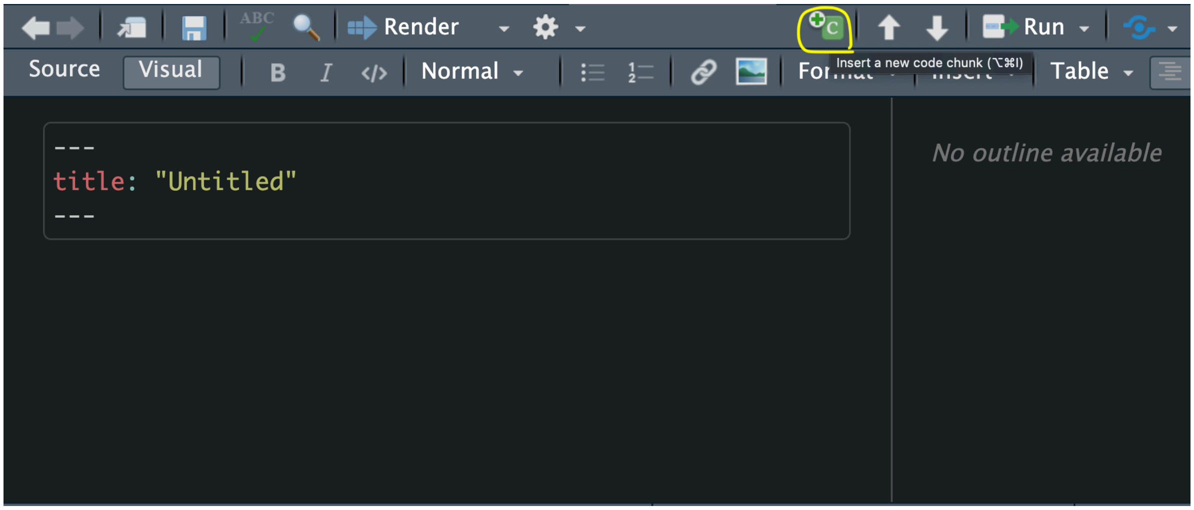
You can also create a new Quarto file by clicking on the "File" menu, then "New File", and then "Quarto File". You will see a new Quarto file in the RStudio editor. You can now write your R code in this file.

Quarto allows you to write your code in chunks. In between chunks, you can have other text, images, and other content. You can also run the code in each chunk and see the output in the document. This is a great way to write reports, papers, and books.

Personally I prefer to write my code in Quarto files. When you click to create a new Quarto file, it will ask you to add a title and author, and select a format for your Quarto file. I explain Quarto further in the Storytelling with Quarto chapter. For now, click "Create Empty Document" on the left bottom. Click File > Save and save you Quarto document in your project directory.

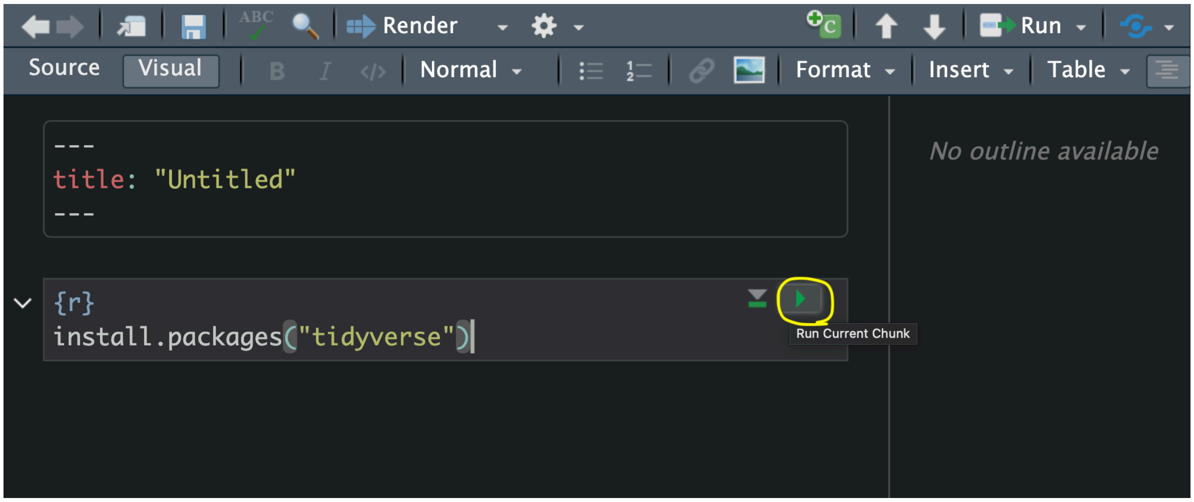
On the top right of the RStudio editor, you can see a green C button with a + sign. That button allows you to insert a code chunk in your document. See Figure 4.1.

Then after you write your code and when you want to run the code in a chunk, you can click on the green Run button on the right side of the chunk. See Figure 4.2.



```
---
title: "Untitled"
---
```

Figure 4.1: Quarto: Inserting a New Code Chunk



```
---
title: "Untitled"
---
```

```
{r}
install.packages("tidyverse")
```

Figure 4.2: Quarto: Running a Code Chunk

When you are working in a simple R script, you don't need to worry about chunks. You can simply write your code in the script file and select the lines of code you want to run and click on the Run button.

4.3 Installing Packages

Packages in R are like apps for your phone. Just like your phone comes with some basic apps, R comes with 14 base packages (as of May 11, 2024) including `base`, `utils`, and `stats`. But you can, and you will need to, install other packages to do different things just like you install apps on your phone.

You can install a package using the `install.packages()` function. This book uses the `tidyverse` package, which is a universe of packages that follow a common "tidy" data philosophy.

You can install the `tidyverse` package using the following command:

```
# uncomment the following line by removing "#" and run the code to install the tidyverse pac

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

You need to install the packages you need only once, and then you can use them whenever you want.

4.4 Loading Packages

Just like apps on your phone, you need to load the packages you need every time you start a new R session. You can load the package using the `library()` function. For example, to load the `tidyverse` package, you can use the following command:

```
library(tidyverse)
```

The `[tidyverse](https://tidyverse.org/)` package is a collection of packages including `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, `stringr`, `forcats`, `rvest`, `lubridate`, and a few other packages. We learn some of these packages in this book. Once you load the `tidyverse` package, you can use all the functions in these packages. In other words, you don't need to load, for example, the `ggplot2` package separately by running `library(ggplot2)`.

4.5 Assigning Values to Variables

You can assign values to variables in R using the <- operator. For example, you can assign the value 14 to a variable x using the following command:

```
x <- 14
```

After assigning 14 to x, you can use x in your code. For example, you can print the value of x using the following command:

```
print(x)
```

```
[1] 14
```

See, x is now 14. You can also see the value of x by typing x in the console and pressing Enter. Try it.

```
x
```

```
[1] 14
```

You can also make additional data manipulations using x and assign it to another variable y using the following command:

```
y <- x + 3
```

Let's see the value of y:

```
y
```

```
[1] 17
```

R is an advanced calculator. You can do all kinds of calculations using R. For example, check out the following calculations:

```
# Square root of 16
sqrt(16)
```

```
[1] 4
```

```
# 2 to the power of 8
2^8
```

```
[1] 256
```

```
# Logarithm of 100
log(100)
```

```
[1] 4.60517
```

```
# Exponential of previously assigned value y, i.e. 17, times x, i.e. 14.
exp(y) * x
```

```
[1] 338169339
```

You can also assign a character string to a variable. For example, you can assign the string "한국학 학자들 및 학생들도 R 좀 배웠으면 좋겠다." to a variable z using the following command:

```
z <- "          R       ."
```

You can print the value of z using the following command:

```
z
```

```
[1] "          R       ."
```

4.6 Make sure to get the spelling right!

As a novice R user, more often than not, you will get error messages because you make mistakes in spelling. For example, we assigned the value 14 to a variable x. If you try to print the value of X instead of x, you will get an error message. Try it.

```
# Uncomment the following line by removing "#" and run the code to see the error message.

# X
```

You will get an error message saying that “Error: object ‘X’ not found”. This is because R is case-sensitive. `X` is not the same as `x`. Make sure to get the spelling right.

If you want to use more than two words for a variable name, you can use an underscore `_` (`my_variable`) or a dot `.` (`my.variable`) to separate the words; or you can write the words together with each new word beginning with a capital letter (`myVariable`). You better be consistent with your naming convention, although technically you can name your variables however you want. For example, you can assign `c(" ", " ", " ", " ")` to a variable `my_variable` using the following command:

```
my_variable <- c(" ", " ", " ", " ")
```

4.7 Data Types

R has several data types including numeric, character, logical, date, list, dataframe and so on. Let’s see the data types of the variables we created above. We can use the `class()` function to see the data type of a variable. For example, you can see the data type of `x`, `y`, and `z` using the following commands:

```
class(x)
```

```
[1] "numeric"
```

```
class(y)
```

```
[1] "numeric"
```

```
class(z)
```

```
[1] "character"
```

The data type of `x` and `y` is numeric, and the data type of `z` is character.

If we write numbers in quotes, they become character strings. For example, you can assign the string “14” to a variable `w` using the following command:

```
w <- "14"
```

You can see the data type of `w` using the following command:

```r
class(w)
```

```
[1] "character"
```

The data type of `w` is character. You can also turn it into a numeric value using the `as.numeric()` function. For example, you can turn `w` into a numeric value using the following command:

```r
w <- as.numeric(w)
```

4.8 Vectors

A vector is a collection of elements of the same data type. You can create a vector using the `c()` function. For example, you can create a vector `v` with the elements “서울”, “부산”, “대구”, “인천”, and “대전” using the following command:

```r
v <- c(" ", " ", " ", " ", " ")
```

You can print the vector `v` using the following command:

```r
v
```

```
[1] " " " " " " " " " "
```

You can see the data type of `v` using the following command:

```r
class(v)
```

```
[1] "character"
```

The data type of `v` is character. You can also create a numeric vector. For example, you can create a vector `numbers` with the elements 1, -2, 3.1, 49, and 0 using the following command:

```r
numbers <- c(1, -2, 314, -49, 0)
```

You can print the vector `numbers` using the following command:

```
numbers
```

```
[1]   1  -2 314 -49   0
```

You can see the data type of `numbers` using the following command:

```
class(numbers)
```

```
[1] "numeric"
```

The data type of `numbers` is numeric.

4.9 Dataframes

A dataframe is a collection of vectors of the same length. You can create a dataframe using the `data.frame()` function. For example, you can create a dataframe `df` with three columns `city_name_en`, `city_name_kr`, and `population` using the following command:

```
# Relying on this link for the population data (in millions):
# https://kosis.kr/statHtml/statHtml.do?orgId=101&tblId=DT_1B040A3

df <- data.frame(city_name_en = c("Busan", "Daegu", "Incheon", "Seoul", "Daejeon"),
                 city_name_kr = c(" ", " ", " ", " ", " "),
                 population = c(3.3, 2.4, 3, 9.4, 1.4))
```

You can print the dataframe `df` using the following command:

```
df
```

```
  city_name_en city_name_kr population
1        Busan                     3.3
2        Daegu                     2.4
3      Incheon                     3.0
4        Seoul                     9.4
5      Daejeon                     1.4
```

You can see the data type of `df` using the following command:

```
class(df)
```

```
[1] "data.frame"
```

The data type of `df` is dataframe. We can reach the columns of the dataframe using the `$` sign. For example, you can see the column `city_name_en` using the following command:

```
df$city_name_en
```

```
[1] "Busan"   "Daegu"   "Incheon" "Seoul"   "Daejeon"
```

4.10 Some Basic Functions

You can use the `head()` function to see the first few rows of a dataframe. For example, you can see the first few rows of the dataframe `df` using the following command:

```
head(df)
```

```
  city_name_en city_name_kr population
1        Busan                     3.3
2        Daegu                     2.4
3      Incheon                     3.0
4        Seoul                     9.4
5      Daejeon                     1.4
```

By default, the `head()` function shows the first 6 rows of the dataframe. You can also specify the number of rows you want to see. For example, you can see the first 3 rows of the dataframe `df` using the following command:

```
head(df, 3)
```

```
  city_name_en city_name_kr population
1        Busan                     3.3
2        Daegu                     2.4
3      Incheon                     3.0
```

You can use the `tail()` function to see the last few rows of a dataframe. For example, you can see the last few rows of the dataframe `df` using the following command:

```
tail(df)
```

```
  city_name_en city_name_kr population
1        Busan                  3.3
2        Daegu                  2.4
3      Incheon                  3.0
4        Seoul                  9.4
5      Daejeon                  1.4
```

In this example, both the `head()` and `tail()` functions show the entire dataframe because the dataframe `df` has only 5 rows. But, in longer dataframes, you can see the first or last few rows using these functions.

In a similar vein, `glimpse()` function from the `dplyr` package is a function to see the structure of a dataframe. For example, you can see the structure of the dataframe `df` using the following command:

```
glimpse(df)
```

```
Rows: 5
Columns: 3
$ city_name_en <chr> "Busan", "Daegu", "Incheon", "Seoul", "Daejeon"
$ city_name_kr <chr> " ", " ", " ", " ", " "
$ population   <dbl> 3.3, 2.4, 3.0, 9.4, 1.4
```

The `nrow()` function gives the number of rows in a dataframe. For example, you can see the number of rows in the dataframe `df` using the following command:

```
nrow(df)
```

```
[1] 5
```

Likewise, the `ncol()` function gives the number of columns in a dataframe. For example, you can see the number of columns in the dataframe `df` using the following command:

```
ncol(df)
```

```
[1] 3
```

The dim() function gives the dimensions of a dataframe. For example, you can see the dimensions of the dataframe df using the following command:

```
dim(df)
```

```
[1] 5 3
```

The summary() function gives a summary of a dataframe. For example, you can see the summary of the dataframe df using the following command:

```
summary(df)
```

```
 city_name_en       city_name_kr         population 
 Length:5           Length:5           Min.   :1.4  
 Class :character   Class :character   1st Qu.:2.4  
 Mode  :character   Mode  :character   Median :3.0  
                                       Mean   :3.9  
                                       3rd Qu.:3.3  
                                       Max.   :9.4  
```

4.11 Rows and Columns

You can select rows and columns of a dataframe using the [] operator. The first argument of the [] operator is the row index, and the second argument is the column index. df[row, column] selects the row with the index row and the column with the index column.

For example, you can select the first row and the second column of the dataframe df using the following command:

```
df[1, 2]
```

```
[1] "  "
```

You can select the first row of the dataframe df using the following command:

```
df[1, ]
```

```
  city_name_en city_name_kr population
1        Busan                      3.3
```

You can select the second column of the dataframe `df` using the following command:

```
df[, 2]
```

```
[1] " " " " " " " " " "
```

4.12 Piping

The pipe operators `|>` and `%>%` are powerful tools in R.[1] The pipe allows you to write code in a more readable way. You can use the pipe operator to pass the output of one function to the input of another function. For example, you can use the pipe operator to pass the dataframe `df` to the `head()` function. You can see the first few rows of the dataframe `df` using the following command:

```
df |> head()
```

```
  city_name_en city_name_kr population
1        Busan                     3.3
2        Daegu                     2.4
3      Incheon                     3.0
4        Seoul                     9.4
5      Daejeon                     1.4
```

We can arrange the dataframe `df` using the `arrange()` function from the `dplyr` package. For example, you can arrange the dataframe `df` by the `population` column using the following command with a pipe:

```
df |> arrange(population)
```

```
  city_name_en city_name_kr population
1      Daejeon                     1.4
2        Daegu                     2.4
3      Incheon                     3.0
4        Busan                     3.3
5        Seoul                     9.4
```

[1] In most cases, these two pipes work the same way. Refer to this link for more explanation on the difference between the `base` pipe `|>` and the `magrittr` pipe `%>%`. For now, you can simply ignore the difference.

`arrange()` function arranges the dataframe by the selected numeric column in ascending order by default. If it is a character column, it arranges the dataframe in alphabetical order. You can arrange the dataframe in descending order by using the `desc()` function. For example, you can arrange the dataframe `df` by the `population` column in descending order using the following command:

```
df |> arrange(desc(population))
```

```
  city_name_en city_name_kr population
1        Seoul                  9.4
2        Busan                  3.3
3      Incheon                  3.0
4        Daegu                  2.4
5      Daejeon                  1.4
```

We can also assign `df` to the rearranged dataframe. For example, you can assign the arranged dataframe to `df` using the following command:

```
df <- df |> arrange(desc(population))
```

Now, `df` is arranged by the `population` column in descending order. Let's check out:

```
df
```

```
  city_name_en city_name_kr population
1        Seoul                  9.4
2        Busan                  3.3
3      Incheon                  3.0
4        Daegu                  2.4
5      Daejeon                  1.4
```

Good. We learned the basics of R. In the next chapter, we learn about data wrangling using mainly the `dplyr` package.

5 Data Wrangling

In this chapter, we will learn how to wrangle data mainly using the `dplyr` package. We will learn how to select, filter, arrange, mutate, group, and summarize data. We will learn how to join data from different sources, working with dates, and converting data to long and wide formats.

As an example for this chapter, we will use Korea's trade data, `trade_data`, from the `kdiplo` package. Let's install the package. You can install the development version from GitHub with:

```
# install.packages("devtools") # if you haven't installed the devtools package yet, remove t

devtools::install_github("kjayhan/kdiplo")
```

Let's load the libraries and the data.

```
library(tidyverse) # load the tidyverse package which includes dplyr, ggplot2, tidyr, readr,

library(kdiplo) # load the kdiplo package
```

Let's take a quick look at the data.

```
head(trade_data)
```

```
# A tibble: 6 x 18
  iso3c country  year export import total_export total_import export_kosis
  <chr> <chr>   <dbl>  <dbl>  <dbl>        <dbl>        <dbl>        <dbl>
1 ABW   Aruba    1965     NA     NA    175082000    463442000           NA
2 ABW   Aruba    1966     NA     NA    250334000    716441000           NA
3 ABW   Aruba    1967     NA     NA    320229000    996246000           NA
4 ABW   Aruba    1968     NA     NA    455400000   1462873000           NA
5 ABW   Aruba    1969     NA     NA    622516000   1823611000           NA
6 ABW   Aruba    1970     NA     NA    835185000   1983973000           NA
# i 10 more variables: import_kosis <dbl>, export_cow <dbl>, import_cow <dbl>,
#   index <dbl>, cpi <dbl>, export_cons_2015 <dbl>, import_cons_2015 <dbl>,
#   total_export_cons_2015 <dbl>, total_import_cons_2015 <dbl>,
#   updated_at <date>
```

We can read the data's documentation using the `?` function.

```
?trade_data
```

Let's assign the data to a new object.

```
trade_data <- trade_data
```

5.1 Selecting columns

We do not need all the columns in the data. We can select the columns we need using the `select()` function. For now, I will select only five columns: `iso3c` (country code), `country` (country name), `year` (year), `export_kosis` (Korea's exports as reported by Korean Statistical Information Service (KOSIS)), and `import_kosis` (Korea's exports as reported by KOSIS).

We can either assign the updated object with the selected columns to the same object or a new object. Here, I will assign the updated object to a new object.

```
trade <- trade_data |>
  select(iso3c, country, year, export_kosis, import_kosis)
```

Let's see how many rows and columns `trade_data` and `trade` have.

```
nrow(trade_data) # number of rows in trade_data
```

```
[1] 16511
```

```
ncol(trade_data) # number of columns in trade_data
```

```
[1] 18
```

```
nrow(trade) # number of rows in trade
```

```
[1] 16511
```

```
ncol(trade) # number of columns in trade
```

```
[1] 5
```

`trade_data` has 16511 rows and 18 columns. `trade` has 16511 rows and 5 columns.

5.2 Filtering rows

We can filter rows based on a condition using the `filter()` function. Here, I will filter rows where the year is larger than 1964. Indeed, KOSIS data starts from 1965. This time, I will assign the updated object to the same object. We need a condition for filtering. In this case, the condition is `year > 1964`. It is the same as `year >= 1965`.

```
trade <- trade |>
  filter(year > 1964)
```

Let's create a new object with the data from only 2019. == is the condition for equality. We need to use == instead of = for equality condition, and we need to be careful about it.

```
trade_2019 <- trade |>
  filter(year == 2019)

# let's see what the data looks like:
head(trade_2019)
```

```
# A tibble: 6 x 5
  iso3c country         year export_kosis import_kosis
  <chr> <chr>          <dbl>        <dbl>        <dbl>
1 ABW   Aruba           2019     10396000         1000
2 AFG   Afghanistan     2019     49930000        38000
3 AGO   Angola          2019    236830000     16733000
4 AIA   Anguilla        2019       817000         1000
5 ALA   Åland Islands   2019           NA            0
6 ALB   Albania         2019     20744000      3357000
```

Let's create a new object with the data from only three countries: United States, China, and Japan. We need to use `%in%` as a condition for multiple values that we look for in the dataframe.

```
trade_us_china_japan <- trade |>
  filter(country %in% c("United States", "China", "Japan"))

# let's see what the data looks like:
head(trade_us_china_japan)
```

```
# A tibble: 6 x 5
  iso3c country  year export_kosis import_kosis
  <chr> <chr>   <dbl>        <dbl>        <dbl>
1 CHN   China    1965           NA           NA
2 CHN   China    1966           NA           NA
3 CHN   China    1967           NA           NA
4 CHN   China    1968           NA           NA
5 CHN   China    1969           NA           NA
6 CHN   China    1970           NA           NA
```

We can filter the rows for multiple years using the `%in%` operator as well. Let's create a new object with the data from 2015, 2016, 2017, and 2018. `:` is used to create a sequence of numbers. `2015:2018` creates a sequence of numbers from 2015 to 2018.

```
trade_2015_2018 <- trade |>
  filter(year %in% 2015:2018)
```

We can also filter rows based on multiple conditions. Let's create a new object with the data from 2015, 2016, 2017, and 2018 using the `&` operator, which means "and".

```
trade_2015_2018_backup <- trade |>
  filter(year >= 2015 # year is greater than or equal to 2015
         & # and
         year <= 2018 # year is less than or equal to 2018
         )
```

Let's check if `trade_2015_2018` and `trade_2015_2018_backup` are the same.

```
identical(trade_2015_2018, trade_2015_2018_backup)
```

```
[1] TRUE
```

Now, let's filter the data for 2015, 2016, 2017, and 2018 for the United States, China, and Japan, this time using country codes.

```
trade_us_china_japan_2015_2018 <- trade |>
  filter(year %in% 2015:2018 # included years are 2015, 2016, 2017, and 2018
         & # and
         iso3c %in% c("USA", "CHN", "JPN") # included country codes are USA, CHN, and JPN
         )

# let's see what the data looks like:
head(trade_us_china_japan_2015_2018)
```

```
# A tibble: 6 x 5
  iso3c country  year export_kosis import_kosis
  <chr> <chr>   <dbl>        <dbl>        <dbl>
1 CHN   China    2015 137123934000  90250275000
2 CHN   China    2016 124432941000  86980135000
3 CHN   China    2017 142120000000  97860114000
4 CHN   China    2018 162125055000 106488592000
5 JPN   Japan    2015  25576507000  45853834000
6 JPN   Japan    2016  24355036000  47466592000
```

Two other operators that we can use for filtering are `|` and `!`. `|` means “or” and `!` means “not”. Let’s create a new object with the data for 2015, 2016, 2017, and 2018 or the export volume is larger than 100 billion USD.

```
trade_2015_2018_or_export <- trade |>
  filter(year %in% 2015:2018 # included years are 2015, 2016, 2017, and 2018
         | #or
         export_kosis > 110000000000 # export volume is larger than 110 billion USD
         )
```

Let’s see what else is included that is not in the years 2015, 2016, 2017, and 2018.

```
trade_2015_2018_or_export |>
  filter(!year %in% 2015:2018) #  excluded years are 2015, 2016, 2017, and 2018
```

```
# A tibble: 11 x 5
   iso3c country        year export_kosis import_kosis
   <chr> <chr>         <dbl>        <dbl>        <dbl>
 1 CHN   China          2010 116837833000  71573603000
 2 CHN   China          2011 134185009000  86432238000
 3 CHN   China          2012 134322564000  80784595000
 4 CHN   China          2013 145869498000  83052877000
 5 CHN   China          2014 145287701000  90082226000
 6 CHN   China          2019 136202533000 107228736000
 7 CHN   China          2020 132565445000 108884645000
 8 CHN   China          2021 162912974000 138628127000
 9 CHN   China          2022 155789389000 154576314000
10 CHN   China          2023 124817682000 142857338000
11 USA   United States  2023 115696334000  71272030000
```

5.3 Arranging rows

We can arrange rows based on a column using the `arrange()` function. Let's arrange the data by year in ascending order.

```
trade <- trade |>
  arrange(year)

head(trade)
```

```
# A tibble: 6 x 5
  iso3c country         year export_kosis import_kosis
  <chr> <chr>          <dbl>        <dbl>        <dbl>
1 ABW   Aruba           1965           NA           NA
2 AFG   Afghanistan     1965           NA           NA
3 AGO   Angola          1965           NA           NA
4 AIA   Anguilla        1965           NA           NA
5 ALA   Åland Islands   1965           NA           NA
6 ALB   Albania         1965           NA           NA
```

We can arrange by year in descending order.

```
trade <- trade |>
  arrange(desc(year))

head(trade)
```

```
# A tibble: 6 x 5
  iso3c country         year export_kosis import_kosis
  <chr> <chr>          <dbl>        <dbl>        <dbl>
```

```
1 ABW   Aruba          2023     21005000        121000
2 AFG   Afghanistan    2023     25079000       1045000
3 AGO   Angola         2023    474761000         11000
4 AIA   Anguilla       2023        96000         10000
5 ALA   Åland Islands  2023        15000             0
6 ALB   Albania        2023    142311000      11053000
```

We can arrange alphabetically by country codes in ascending order.

```r
trade <- trade |>
  arrange(iso3c)

head(trade)
```

```
# A tibble: 6 x 5
  iso3c country  year export_kosis import_kosis
  <chr> <chr>   <dbl>        <dbl>        <dbl>
1 ABW   Aruba    2023     21005000       121000
2 ABW   Aruba    2022     24954000        15000
3 ABW   Aruba    2021     11612000     93314000
4 ABW   Aruba    2020      3070000     83864000
5 ABW   Aruba    2019     10396000         1000
6 ABW   Aruba    2018     14807000      2935000
```

5.4 Mutating columns

We can create new columns or update existing columns using the `mutate()` function. Let's create a new column, `trade_kosis`, which is the total trade volume of Korea with a country in a year. The total trade volume is the sum of exports and imports.

```r
trade <- trade |>
  mutate(trade_kosis = export_kosis + import_kosis)

head(trade)
```

```
# A tibble: 6 x 6
  iso3c country  year export_kosis import_kosis trade_kosis
  <chr> <chr>   <dbl>        <dbl>        <dbl>       <dbl>
1 ABW   Aruba    2023     21005000       121000    21126000
```

```
2 ABW   Aruba    2022     24954000        15000    24969000
3 ABW   Aruba    2021     11612000     93314000   104926000
4 ABW   Aruba    2020      3070000     83864000    86934000
5 ABW   Aruba    2019     10396000         1000    10397000
6 ABW   Aruba    2018     14807000      2935000    17742000
```

5.5 Grouping and summarizing data

We can group data based on one or more columns using the `group_by()` function. We can summarize data based on the groups using the `summarize()` function. Let's group the data by year and summarize the total trade volume of Korea in each year.

We need to be careful about one thing. There are missing values in the data. We need to ignore them (in other words treat them as zero) when we calculate the total trade volume. Otherwise, the total trade volume will be `NA` if there is at least one missing value in the data for a year. We can use the `na.rm = TRUE` argument in the `sum()` function to remove missing values.

```
trade_volume <- trade |>
  group_by(year) |>
  summarize(total_trade_kosis = sum(trade_kosis, na.rm = TRUE)) |>
  arrange(desc(total_trade_kosis))

head(trade_volume)
```

```
# A tibble: 6 x 2
   year total_trade_kosis
  <dbl>             <dbl>
1  2022     1400216998000
2  2023     1270073156000
3  2021     1248778081000
4  2018     1127928070000
5  2014     1092728073000
6  2011     1077938860000
```

We can also group the data by country. Let's summarize the total trade volume of Korea with each country since 1965.

```
trade_country <- trade |>
  group_by(country) |>
  summarize(total_trade_kosis = sum(trade_kosis, na.rm = TRUE)) |>
  arrange(desc(total_trade_kosis))

head(trade_country)
```

```
# A tibble: 6 x 2
  country              total_trade_kosis
  <chr>                            <dbl>
1 China                    4455699092000
2 United States            3179689314000
3 Japan                    2424243884000
4 Vietnam                   773845848000
5 Hong Kong SAR China       759431632000
6 Saudi Arabia              753711941000
```

5.6 Conditional Mutating

We can conditionally mutate columns using the `case_when()` function. Let's create a new column, `trade_status`, which is "surplus" if the export volume is larger than the import volume, "deficit" if the import volume is larger than the export volume, and "balanced" if the export volume is equal to the import volume. If the export or import volume is missing, we will make the trade status "unknown". We can use `is.na()` to check if a value is missing.

```
trade <- trade |>
  mutate(trade_status = case_when(
    export_kosis > import_kosis ~ "surplus", # export volume is larger than import volume
    export_kosis < import_kosis ~ "deficit", # export volume is less than import volume
    export_kosis == import_kosis ~ "balanced", # export volume is equal to import volume
    is.na(export_kosis) | is.na(import_kosis) ~ "unknown", # export or import volume is miss
    TRUE ~ "everything else" # in this instance, we do not need "TRUE ~" since we cover all
  ))

head(trade)
```

```
# A tibble: 6 x 7
  iso3c country  year export_kosis import_kosis trade_kosis trade_status
  <chr> <chr>   <dbl>        <dbl>        <dbl>       <dbl> <chr>
```

```
1 ABW    Aruba    2023     21005000        121000    21126000 surplus
2 ABW    Aruba    2022     24954000         15000    24969000 surplus
3 ABW    Aruba    2021     11612000      93314000   104926000 deficit
4 ABW    Aruba    2020      3070000      83864000    86934000 deficit
5 ABW    Aruba    2019     10396000          1000    10397000 surplus
6 ABW    Aruba    2018     14807000       2935000    17742000 surplus
```

In this instance, we do not need “TRUE ~” since we cover all `case_when()` options above. But in other cases, you may need it. “TRUE ~” basically helps you assign a new value for every other condition that is not mentioned above.

We can create a table using the `table()` function for the trade status of Korea since 1965.

```
table(trade$trade_status)
```

```

balanced  deficit  surplus  unknown
     143     3134     6567     5444
```

5.7 Merging datasets

Right now, we only have one dataset. Let’s get another dataset from the `WDI` package, which includes World Bank’s World Development Indicators data. Let’s install the package if you do not have it yet.

```
# install.packages("WDI") # if you haven't installed the WDI package yet, remove the # sign.

library(WDI) # load the WDI package
```

Let’s get the data for the GDP of all countries since 1965. You can search for indicators from the World Bank’s World Development Indicators database here or using the `WDIsearch` function in the `WDI` package. For details, you can check out `WDI`’s documentation using the `?` function or its Github page.

```
wdi <- WDI(country = "all", # all countries
           indicator = c("gdp" = "NY.GDP.MKTP.KD", # GDP at constant 2015 US dollars
                         "gdp_pc" = "NY.GDP.PCAP.KD"), # GDP per capita at constant 2015 US
           start = 1965, # start year
           end = 2024, # end year
           extra = TRUE, # include extra columns included in the WDI package defaults
```

```
            language = "en" # language is English
            )

head(wdi)
```

```
      country iso2c iso3c year status lastupdated        gdp   gdp_pc
1 Afghanistan    AF   AFG 1965         2024-09-19         NA       NA
2 Afghanistan    AF   AFG 2003         2024-09-19 7867263256 347.4152
3 Afghanistan    AF   AFG 1966         2024-09-19         NA       NA
4 Afghanistan    AF   AFG 2005         2024-09-19 8874480196 363.5415
5 Afghanistan    AF   AFG 1971         2024-09-19         NA       NA
6 Afghanistan    AF   AFG 2002         2024-09-19 7228795919 344.2242
      region capital longitude latitude     income lending
1 South Asia   Kabul   69.1761  34.5228 Low income     IDA
2 South Asia   Kabul   69.1761  34.5228 Low income     IDA
3 South Asia   Kabul   69.1761  34.5228 Low income     IDA
4 South Asia   Kabul   69.1761  34.5228 Low income     IDA
5 South Asia   Kabul   69.1761  34.5228 Low income     IDA
6 South Asia   Kabul   69.1761  34.5228 Low income     IDA
```

We wanted extra WDI data, but we don't need all. Let's select the ones we need. This time, let's exclude the columns we do not need by using the - sign. Then let's exclude non-country groups (e.g., "High income", "Not classified") by filtering out rows where the `iso3c` column is missing. Then let's arrange the data by country code and year.

```
wdi <- wdi |>
  #select(-iso2c, -status, -lastupdated, -capital, -lending, -longitude, -latitude) |> # excl
  filter(!is.na(iso3c)) |> # exclude the rows that are missing country codes (in other words
  arrange(iso3c, year) # arrange the data by country code and year

head(wdi)
```

```
              country iso2c iso3c year status lastupdated          gdp
1         High income    XD       1965         2024-09-19 1.214382e+13
2          Low income    XM       1965         2024-09-19           NA
3 Lower middle income    XN       1965         2024-09-19 5.474380e+11
4      Not classified    XY       1965         2024-09-19           NA
5 Upper middle income    XT       1965         2024-09-19 1.481616e+12
6         High income    XD       1966         2024-09-19 1.281436e+13
      gdp_pc region capital longitude latitude income lending
1 12593.9054   <NA>    <NA>      <NA>     <NA>   <NA>    <NA>
```

```
2          NA   <NA>    <NA>      <NA>     <NA>    <NA>    <NA>
3    584.5733   <NA>    <NA>      <NA>     <NA>    <NA>    <NA>
4          NA   <NA>    <NA>      <NA>     <NA>    <NA>    <NA>
5   1171.4299   <NA>    <NA>      <NA>     <NA>    <NA>    <NA>
6  13154.2957   <NA>    <NA>      <NA>     <NA>    <NA>    <NA>
```

This did not work out. Probably these entries are not missing, but instead simply empty! Let's check that. Let's try filtering out empty country codes (instead of missing country codes which we checked with `is.na()`).

```r
wdi <- wdi |>
  filter(iso3c != "") # exclude the rows that have empty country codes. We check it as an emp

head(wdi)
```

```
  country iso2c iso3c year status lastupdated gdp gdp_pc
1   Aruba    AW   ABW 1965          2024-09-19  NA     NA
2   Aruba    AW   ABW 1966          2024-09-19  NA     NA
3   Aruba    AW   ABW 1967          2024-09-19  NA     NA
4   Aruba    AW   ABW 1968          2024-09-19  NA     NA
5   Aruba    AW   ABW 1969          2024-09-19  NA     NA
6   Aruba    AW   ABW 1970          2024-09-19  NA     NA
                     region    capital longitude latitude      income
1 Latin America & Caribbean Oranjestad  -70.0167  12.5167 High income
2 Latin America & Caribbean Oranjestad  -70.0167  12.5167 High income
3 Latin America & Caribbean Oranjestad  -70.0167  12.5167 High income
4 Latin America & Caribbean Oranjestad  -70.0167  12.5167 High income
5 Latin America & Caribbean Oranjestad  -70.0167  12.5167 High income
6 Latin America & Caribbean Oranjestad  -70.0167  12.5167 High income
         lending
1 Not classified
2 Not classified
3 Not classified
4 Not classified
5 Not classified
6 Not classified
```

Yes, that was it. Instead of NA, those country code columns were empty for those rows. Now that we successfully filtered out the rows with empty country codes, let's join Korea's trade data with the WDI data. There are different types of joins. I will explain five of them. To make things easier, I will create smaller datasets for the demonstration. We will have only the data for the United States, China, and Japan in the trade data. We will have only the data for the United States, Japan and Italy in the WDI data.

```
trade_df <- trade |>
  filter(iso3c %in% c("USA", "CHN", "JPN"))

head(trade_df)
```

```
# A tibble: 6 x 7
  iso3c country  year export_kosis import_kosis  trade_kosis trade_status
  <chr> <chr>   <dbl>        <dbl>        <dbl>        <dbl> <chr>
1 CHN   China    2023 124817682000 142857338000 267675020000 deficit
2 CHN   China    2022 155789389000 154576314000 310365703000 surplus
3 CHN   China    2021 162912974000 138628127000 301541101000 surplus
4 CHN   China    2020 132565445000 108884645000 241450090000 surplus
5 CHN   China    2019 136202533000 107228736000 243431269000 surplus
6 CHN   China    2018 162125055000 106488592000 268613647000 surplus
```

```
wdi_df <- wdi |>
  filter(iso3c %in% c("USA", "JPN", "ITA"))
```

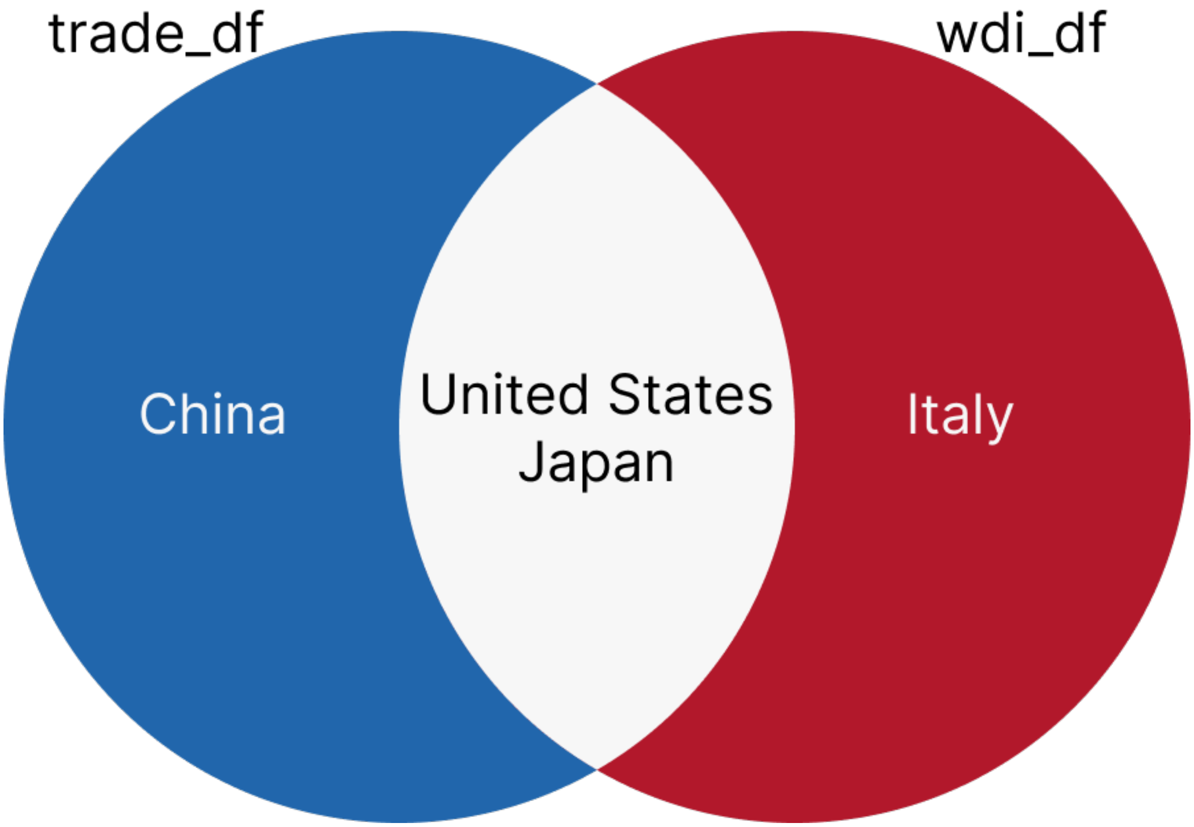


Figure 5.1: Dataframes

5.7.1 inner_join

`inner_join` returns only the rows that have matching values in both datasets. Let's join the `trade_df` and `wdi_df` datasets using the `iso3c` and `year` columns.

```
inner_df <- inner_join(trade_df, wdi_df, by = c("iso3c", "year"))

# you can also write it like this:

# inner_df <- trade_df |> inner_join(wdi_df, by = c("iso3c", "year"), suffix = c("_trade", "


head(inner_df)
```

```
# A tibble: 6 x 19
```

```
  iso3c country.x  year export_kosis import_kosis trade_kosis trade_status
  <chr> <chr>     <dbl>        <dbl>        <dbl>       <dbl> <chr>
1 JPN   Japan      2023  29000616000  47656468000 76657084000 deficit
2 JPN   Japan      2022  30606278000  54711795000 85318073000 deficit
3 JPN   Japan      2021  30061806000  54642165000 84703971000 deficit
4 JPN   Japan      2020  25097651000  46023036000 71120687000 deficit
5 JPN   Japan      2019  28420213000  47580853000 76001066000 deficit
6 JPN   Japan      2018  30528580000  54603749000 85132329000 deficit
# i 12 more variables: country.y <chr>, iso2c <chr>, status <chr>,
#   lastupdated <chr>, gdp <dbl>, gdp_pc <dbl>, region <chr>, capital <chr>,
#   longitude <chr>, latitude <chr>, income <chr>, lending <chr>
```

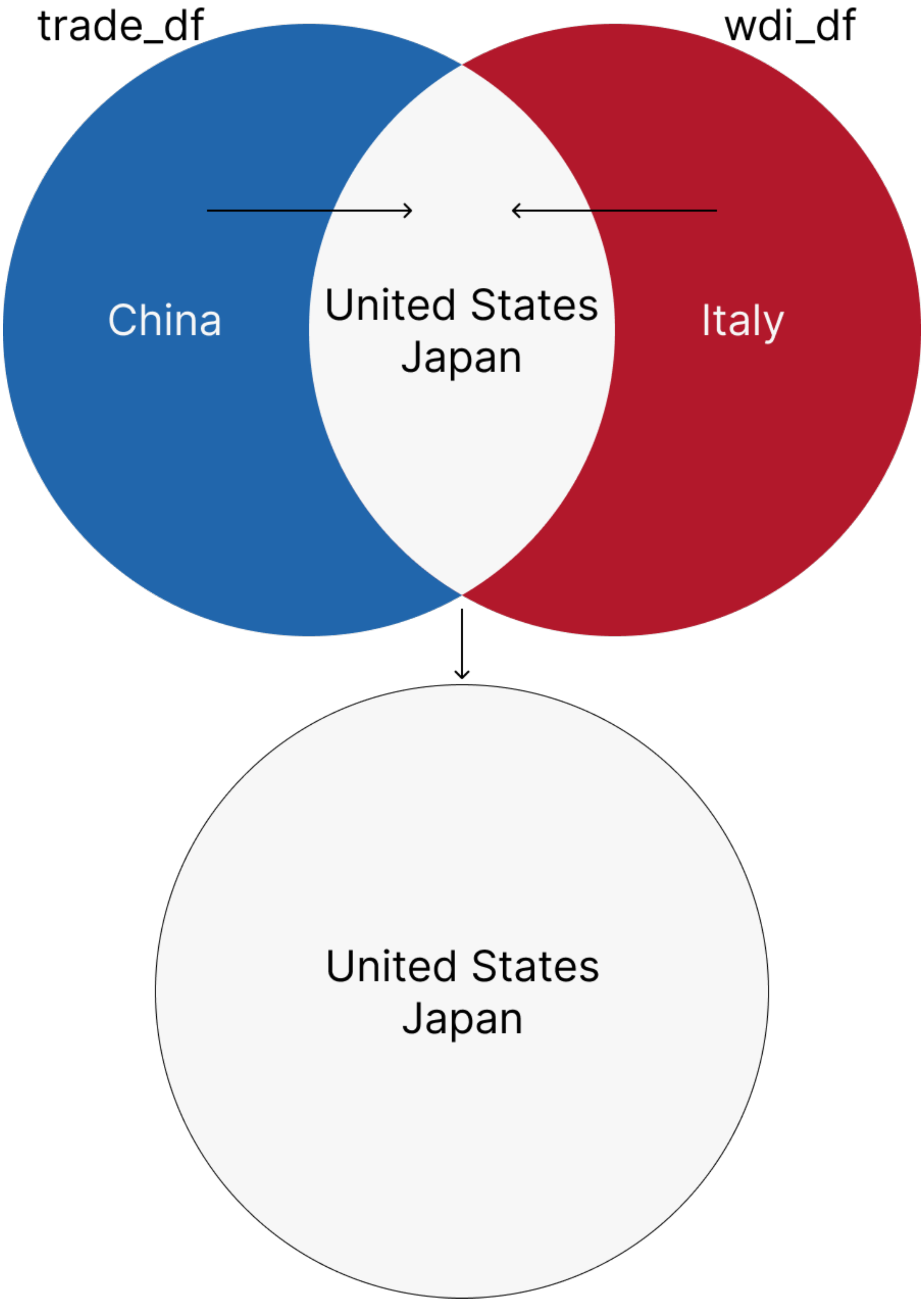


Figure 5.2: inner_join

The column names that we will join by are the same in both dataframes (“iso3c” and “year”). If it was not the same, we could write the code as follows:

```
inner_df <- inner_join(trade_df, wdi_df, by = c("iso3c" = "iso3c", "year" = "year")) # the fi
```

If, for example, the country code column name was “country_code” and the year column was “Year” in `trade_df`, you would replace the first “iso3c” with “country_code” and the first “year” with “Year”.

If there are columns with the same name in both dataframes other than the columns you use to join them, you can use the `suffix` argument to add a suffix to the column names. For example, in this case, we have columns named “country” in both dataframes. Since we didn’t have suffix in the above code, we have two columns “country.x” and “country.y”. If you want to add suffices, you can do it as follows:

```
inner_df <- inner_join(trade_df, wdi_df, by = c("iso3c", "year"), suffix = c("_trade", "_wdi"

head(inner_df)
```

```
# A tibble: 6 x 19
  iso3c country_trade  year export_kosis import_kosis trade_kosis trade_status
  <chr> <chr>         <dbl>        <dbl>        <dbl>       <dbl> <chr>
1 JPN   Japan          2023  29000616000  47656468000 76657084000 deficit
2 JPN   Japan          2022  30606278000  54711795000 85318073000 deficit
3 JPN   Japan          2021  30061806000  54642165000 84703971000 deficit
4 JPN   Japan          2020  25097651000  46023036000 71120687000 deficit
5 JPN   Japan          2019  28420213000  47580853000 76001066000 deficit
6 JPN   Japan          2018  30528580000  54603749000 85132329000 deficit
# i 12 more variables: country_wdi <chr>, iso2c <chr>, status <chr>,
#   lastupdated <chr>, gdp <dbl>, gdp_pc <dbl>, region <chr>, capital <chr>,
#   longitude <chr>, latitude <chr>, income <chr>, lending <chr>
```

5.7.2 left_join

`left_join` returns all the rows from the left dataset and the matched rows from the right dataset. If there is no match, the result is `NA`. Let’s join the `trade_df` and `wdi_df` datasets using the `iso3c` and `year` columns.

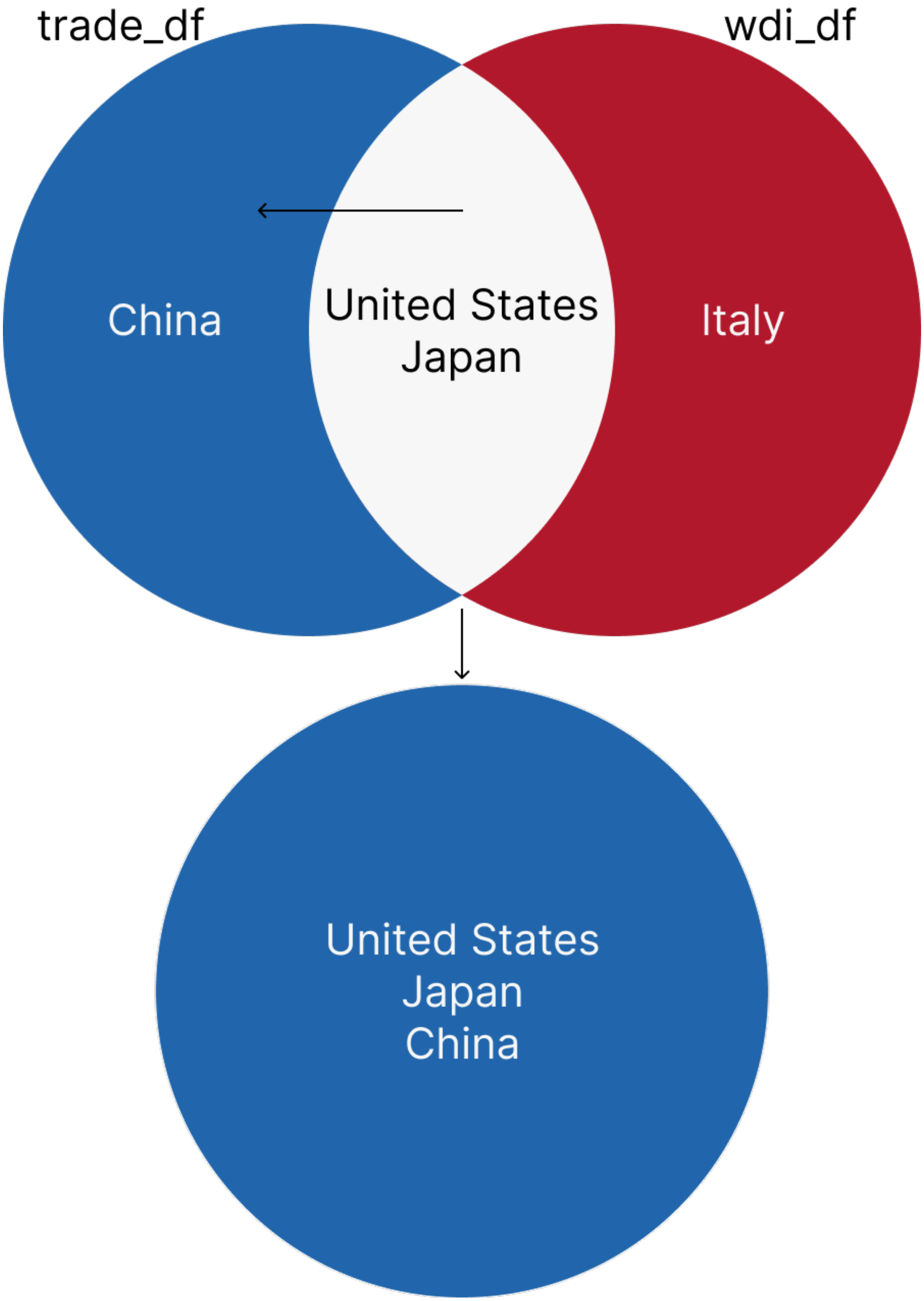


Figure 5.3: left_join

```
left_df <- left_join(trade_df, wdi_df, by = c("iso3c", "year"))

head(left_df)
```

```
# A tibble: 6 x 19
  iso3c country.x  year export_kosis import_kosis  trade_kosis trade_status
  <chr> <chr>     <dbl>        <dbl>        <dbl>        <dbl> <chr>
1 CHN   China      2023 124817682000 142857338000 267675020000 deficit
2 CHN   China      2022 155789389000 154576314000 310365703000 surplus
3 CHN   China      2021 162912974000 138628127000 301541101000 surplus
4 CHN   China      2020 132565445000 108884645000 241450090000 surplus
5 CHN   China      2019 136202533000 107228736000 243431269000 surplus
6 CHN   China      2018 162125055000 106488592000 268613647000 surplus
# i 12 more variables: country.y <chr>, iso2c <chr>, status <chr>,
#   lastupdated <chr>, gdp <dbl>, gdp_pc <dbl>, region <chr>, capital <chr>,
#   longitude <chr>, latitude <chr>, income <chr>, lending <chr>
```

5.7.3 right_join

right_join returns all the rows from the right dataset and the matched rows from the left dataset. If there is no match, the result is NA. Let's join the trade_df and wdi_df datasets using the iso3c and year columns.

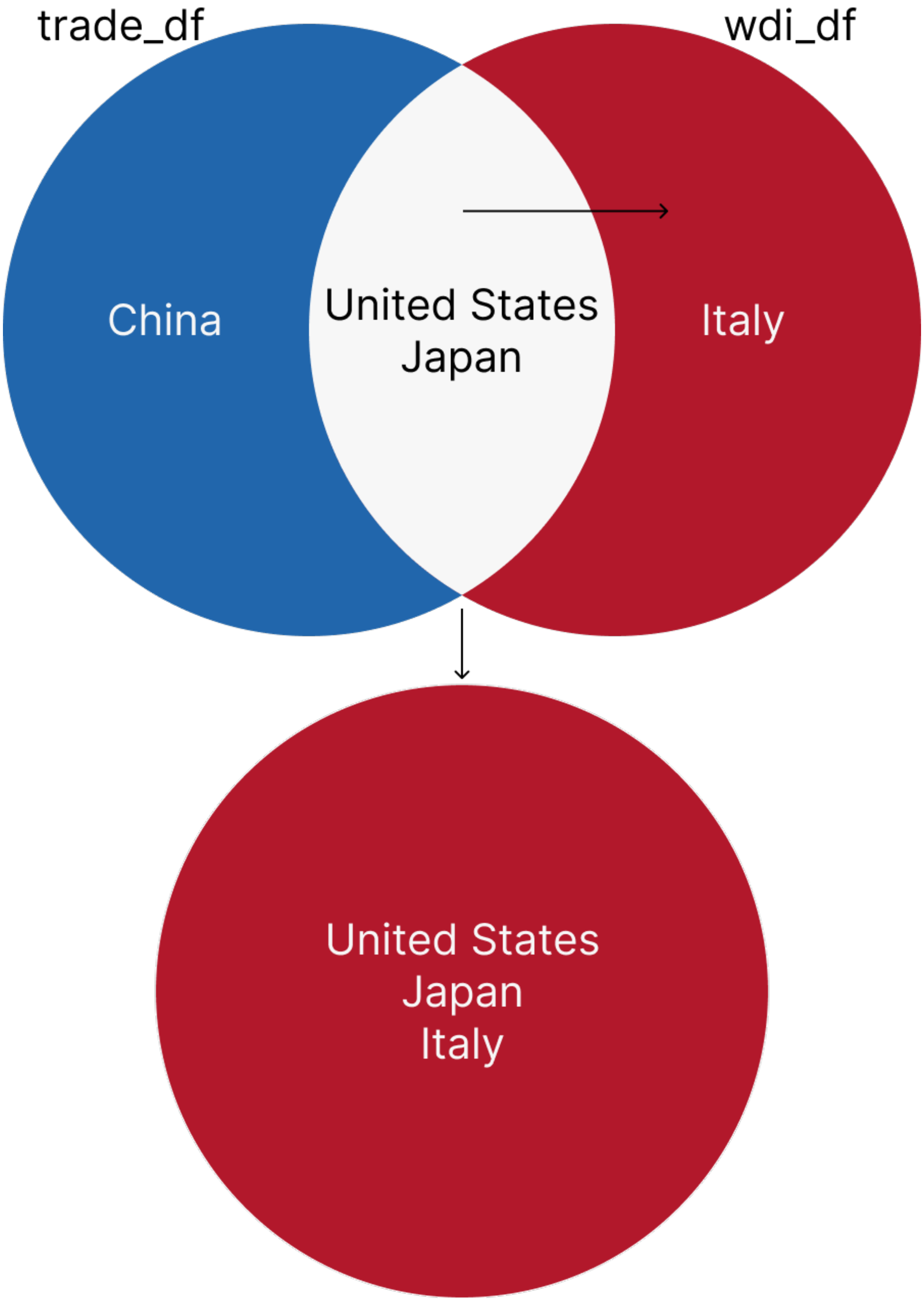


Figure 5.4: right_join

```
right_df <- right_join(trade_df, wdi_df, by = c("iso3c", "year"), suffix = c("_trade", "_wdi'

head(right_df)
```

```
# A tibble: 6 x 19
  iso3c country_trade  year export_kosis import_kosis trade_kosis trade_status
  <chr> <chr>         <dbl>        <dbl>        <dbl>       <dbl> <chr>
1 JPN   Japan          2023  29000616000  47656468000 76657084000 deficit
2 JPN   Japan          2022  30606278000  54711795000 85318073000 deficit
3 JPN   Japan          2021  30061806000  54642165000 84703971000 deficit
4 JPN   Japan          2020  25097651000  46023036000 71120687000 deficit
5 JPN   Japan          2019  28420213000  47580853000 76001066000 deficit
6 JPN   Japan          2018  30528580000  54603749000 85132329000 deficit
# i 12 more variables: country_wdi <chr>, iso2c <chr>, status <chr>,
#   lastupdated <chr>, gdp <dbl>, gdp_pc <dbl>, region <chr>, capital <chr>,
#   longitude <chr>, latitude <chr>, income <chr>, lending <chr>
```

5.7.4 full_join

`full_join` returns all the rows from both datasets. If there is no match, the result is `NA`. Let's join the `trade_df` and `wdi_df` datasets using the `iso3c` and `year` columns.

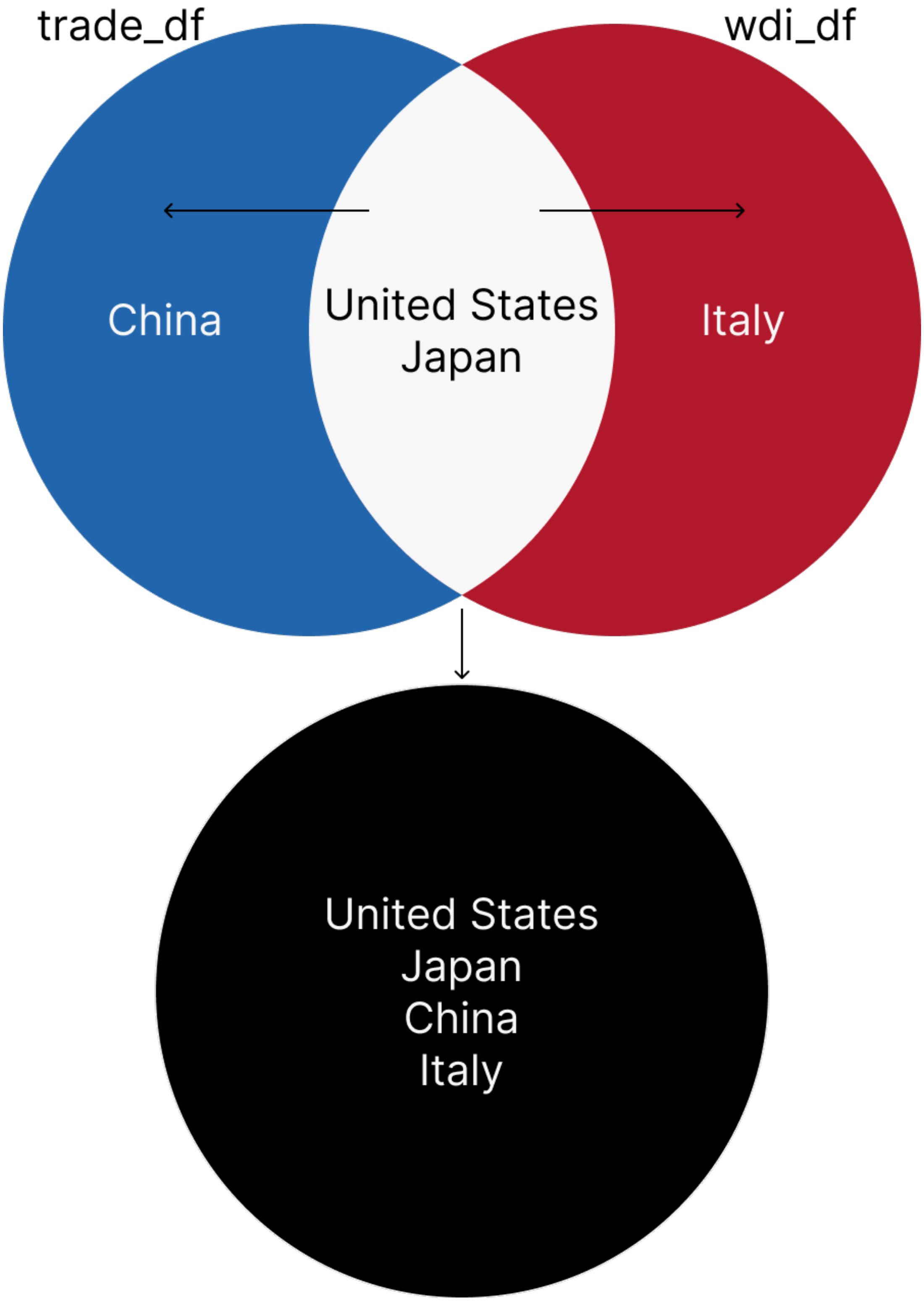


Figure 5.5: full_join

```
full_df <- full_join(trade_df, wdi_df, by = c("iso3c", "year"), suffix = c("_trade", "_wdi"))

head(full_df)
```

```
# A tibble: 6 x 19
  iso3c country_trade  year export_kosis import_kosis  trade_kosis trade_status
  <chr> <chr>         <dbl>        <dbl>        <dbl>        <dbl> <chr>
1 CHN   China          2023 124817682000 142857338000 267675020000 deficit
2 CHN   China          2022 155789389000 154576314000 310365703000 surplus
3 CHN   China          2021 162912974000 138628127000 301541101000 surplus
4 CHN   China          2020 132565445000 108884645000 241450090000 surplus
5 CHN   China          2019 136202533000 107228736000 243431269000 surplus
6 CHN   China          2018 162125055000 106488592000 268613647000 surplus
# i 12 more variables: country_wdi <chr>, iso2c <chr>, status <chr>,
#   lastupdated <chr>, gdp <dbl>, gdp_pc <dbl>, region <chr>, capital <chr>,
#   longitude <chr>, latitude <chr>, income <chr>, lending <chr>
```

5.7.5 anti_join

`anti_join` returns all the rows from the left dataset that do not have a match in the right dataset. Let's join the `trade_df` and `wdi_df` datasets using the `iso3c` and `year` columns.

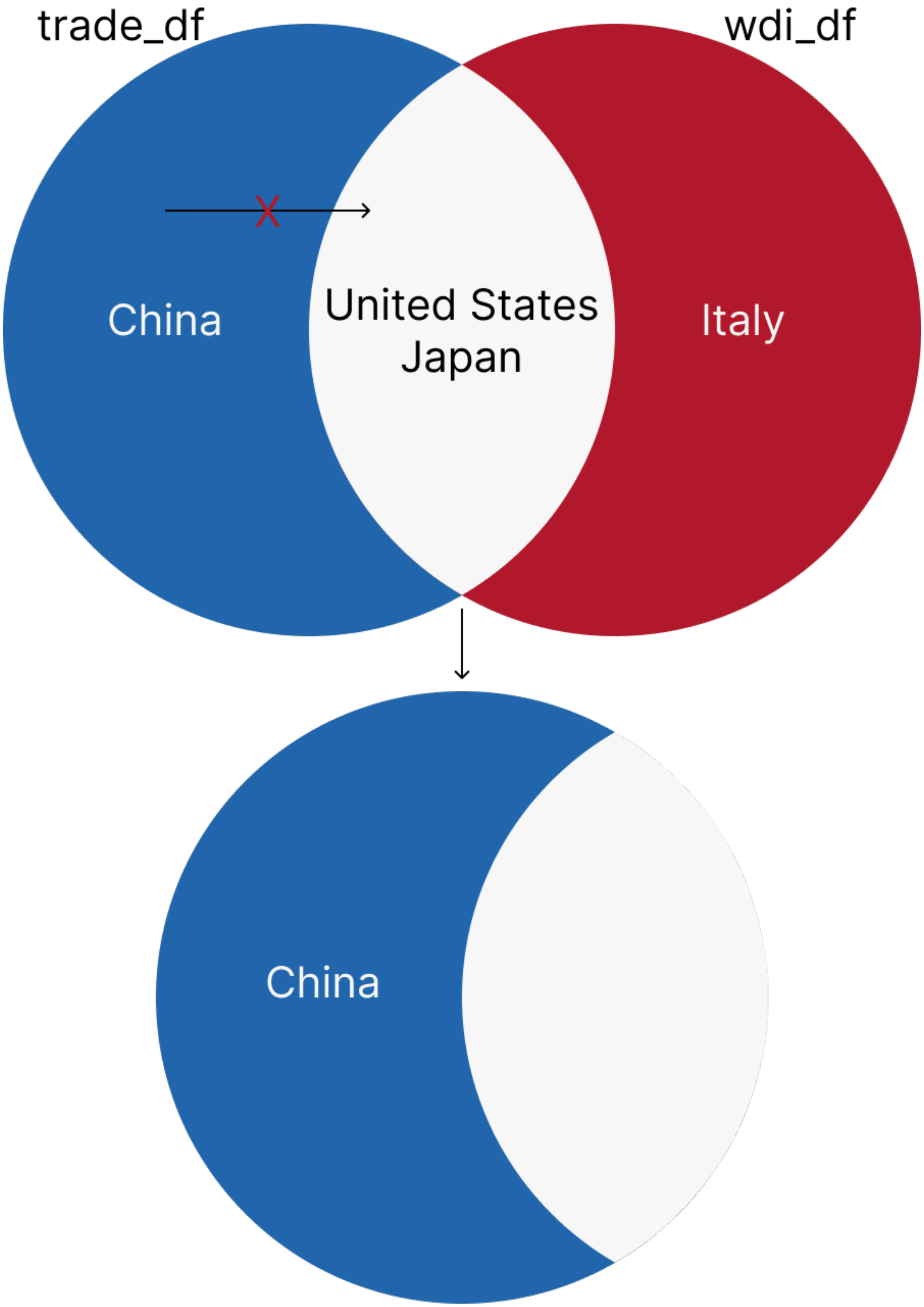


Figure 5.6: anti_join

```
anti_df <- anti_join(trade_df, wdi_df, by = c("iso3c", "year"))

head(anti_df)
```

```
# A tibble: 6 x 7
  iso3c country  year export_kosis import_kosis  trade_kosis trade_status
  <chr> <chr>   <dbl>        <dbl>        <dbl>        <dbl> <chr>
1 CHN   China    2023 124817682000 142857338000 267675020000 deficit
2 CHN   China    2022 155789389000 154576314000 310365703000 surplus
3 CHN   China    2021 162912974000 138628127000 301541101000 surplus
4 CHN   China    2020 132565445000 108884645000 241450090000 surplus
5 CHN   China    2019 136202533000 107228736000 243431269000 surplus
6 CHN   China    2018 162125055000 106488592000 268613647000 surplus
```

5.8 A Note on Country Codes

It is often easier to work with standard country codes than country names when we work with multiple datasets. There are a few widely used standard country codes. Above, we used the ISO 3166-1 alpha-3 country codes. There are other commonly used country codes such as Correlates of War (COW) country codes, Varieties of Democracy (V-Dem) country codes, and more.

We can convert country names to country codes using the `countrycode` package. Let's install the package if you do not have it yet.

```
# install.packages("countrycode") # if you haven't installed the countrycode package yet, rem

library(countrycode) # load the countrycode package
```

Let's convert the country names in the `trade_df` dataset to Correlates of War country codes. You can find the `countrycode` documentantion on its Github page or by using the `?` function.

```
# ?countrycode

trade_df <- trade_df |>
  mutate(cown = countrycode(country, origin = "country.name", destination = "cown")) # conve
```

5.9 A Note on Working with Korean Country Names[1]

In my research, I often work with country-year data from Korean sources, including data on diplomatic visits, trade, aid and so on. One of the fundamental difficulties I have had is the lack of universal country codes across different datasets. Further complicating matters is the inconsistency of country names in these datasets. For example, Democratic Republic of the Congo has five different spellings across different official sources that I could find: 콩고민주공화국, 자이르, 콩고민주공화국, 콩고 민주 공화국, 콩고민주공화국(DR콩고).

To address this issue, I have created a function in my `kdiplo` package that converts Korean country names into ISO 3166-1 alpha-3 (iso3c) country codes. This function, `iso3c_kr`, is designed to assign universal iso3c country codes to Korean-language country names that will make it easier to join different kinds of data.

One still needs to check if the output is correct, especially for countries that have gone through political transitions such as Germany, Yugoslavia, Russia, Vietnam, Yemen and so on.

Sometimes the Korean government sources have overlapping data for Yugoslavia and Serbia, for example. In such cases, one needs to check the data and make sure that the data is correct.

For example, the following is sample Korean trade data from Korean Statistical Information Service (KOSIS):

```
# install.packages("readxl") # if you haven't installed the readxl package yet, remove the #

library(readxl) # load the readxl package

# let's read the xlsx data

kosis_trade <- read_xlsx("data/kosis_trade_240330.xlsx")

# let's take a look at the data

# install.packages("gt") # if you haven't installed the gt package yet, remove the # sign.

# let's take a look at some of the data

# remember, [row, column] format can be used in R for subsetting dataframes. So, we can look

kosis_trade[533:538,c(1,57:62)] |> gt::gt()
```

[1] This subsection is adapted from the vignette of the `iso3c_kr` function in the `kdiplo` package.

국가별	2018 년	2019 년	2020 년	2021 년	2022 년	2023 년
잠비아	26241	16087	17619	28356	14068	15459
잠비아	108344	54542	15164	100606	82198	53867
자이르	NA	NA	NA	NA	NA	NA
자이르	618	8	113	4	NA	NA
짐바브웨	25964	14088	15514	20404	16083	19563
짐바브웨	4909	13098	11377	9627	10415	20862

```
# you can use the gt package to create a table.
# you can use "::" to access the functions in the package without loading the package.
```

And, the following is sample Korean aid data from Korea's ODA portal:

```
aid <- read_xlsx("data/korea_total_aid_2019_230709.xlsx")


aid <- aid |> select(1:5) # we only need the first five columns

aid <- aid |> set_names(c("country_kr", "sector", "no_of_projects", "aid_usd", "aid_krw"))

# This sample data is only 2019; so we will add the year column, and assign 2019 to all rows

aid$year <- 2019

# let's take a look at some of the data
aid[c(50, 150, 250, 350, 450),] |> gt::gt()
```

country_kr	sector	no_of_projects	aid_usd	aid_krw	year
베트남	통신정책, 계획 및 행정(voluntary code)	2	232334	270736486	2019
캄보디아	11321	1	85815	99999361	2019
미얀마	사회보호/보장	1	103460	120560903	2019
라오스	비정규 농업훈련	1	107958	125802378	2019
몽골	의료서비스	5	511824	596423389	2019

5.9.1 Converting wide data to long format

Wide format is quite common in official Korean data sources. Trade data is in wide format. Before using the `iso3c_kr` function, let's first transform the trade data into a long (country-year) format to make it in the same format as the aid data. This will make joining the two datasets more feasible.

To convert the trade data into a long format, we will use the `pivot_longer()` function from the `tidyr` package.

```
# we will divide the trade data into export and import data

export <- kosis_trade

import <- kosis_trade
```

In `pivot_longer()`, we need to specify the columns that we want to pivot. In this case, we want to pivot columns 4 to 62, which are years. We also need to specify the names of the columns that will be created. In this case, we will create a column called `year` and a column called `export_kosis` for the export data. We will create a column called `year` and a column called `import_kosis` for the import data.

```
export_long <- export |>
  pivot_longer(4:62, names_to = "year", values_to = "export_kosis") # we will pivot the data
```

We can rename the columns using `set_names` function in `rlang` package, which is also a member of the `tidyverse` family, to make them more informative.

```
export_long <- export_long |>
  set_names(c("country_kr", "type", "unit", "year", "export_kosis"))
```

We can filter the data for only export data using the `filter()` function. We can also convert the export data from thousands of dollars to dollars by multiplying the `export_kosis` column by 1000. We can also convert the `year` column to numeric using the `parse_number()` function from the `readr` package, which is also a member of the `tidyverse` family.

```
export_long <- export_long |>
  filter(type == "  [  ]") |> # we only need the export data which has the column name in Kor
  mutate(export_kosis = parse_number(export_kosis) * 1000, # we convert the export data from
         year = parse_number(year)) |> # we convert the year column to numeric using parse_nu
  select(-type, -unit) # we do not need the type and unit columns
```

We repeat the same steps for the import data.

```
import_long <- import |>
  pivot_longer(4:62, names_to = "year", values_to = "import_kosis")

import_long <- import_long |>
  set_names(c("country_kr", "type", "unit", "year", "import_kosis"))


import_long <- import_long |>
  filter(type == "  [ ]") |>
  mutate(import_kosis = parse_number(import_kosis) * 1000,
         year = parse_number(year)) |>
  select(-type, -unit)
```

Now, we can join the export and import data using the `left_join()` function.

```
trade_long <- export_long |>
  left_join(import_long, by = c("country_kr", "year"))
```

Here, we get a warning message that there are rows that have the same country name and year in both the export and import data. It is because, KOSIS reported trade with Palestine in two separate entries (probably, West Bank and Gaza are recorded separately), but assigning both the same name “팔레스타인 해방기구”. We will ignore this warning for now.

5.9.2 `iso3c_kr` function to convert Korean country names to iso3c country codes

Using the `iso3c_kr` function, we can simply convert Korean country names into iso3c country codes. For example, the following is the output of the `iso3c_kr` function for the Korean trade data:

```
trade_long <- iso3c_kr(trade_long, "country_kr") #you copy paste the column name that has the

trade_long[c(50, 150, 250, 350, 450, 550), c(1,5, 2:4)] |> gt::gt()
```

country_kr	iso3c	year	export_kosis	import_kosis
계	NA	2014	572664607000	525514506000
아랍에미리트 연합	ARE	1996	1377933000	2259205000
앤티가바부다	ATG	1978	NA	NA
앵귈라	AIA	2019	817000	1000
아르메니아	ARM	2001	1255000	43000

앙골라	AGO	1983	235000	NA

We see that in this example, “계” (gyae) did not get any iso3c country code. This is because the `iso3c_kr` function could not find the iso3c country code for this entry. This is because, it is not a country name. “계” means total. It is best to check the data to see which entries did not get an iso3c code.

```
missing_iso3c <- trade_long |>
  filter(is.na(iso3c)) |> # we only need the rows that do not have iso3c country codes
  pull(country_kr) |> # pull() function is used to extract a column as a vector
  unique() # we need each Korean country name only once to see which ones are missing rather

missing_iso3c
```

```
[1] " "              "    " " "           "  "
```

They mean “total”, “IMF”, “other”, and “other countries” in Korean. In other words, we are not missing any countries, which is good.

Now let’s convert the Korean country names in the aid data into iso3c country codes:

```
aid <- iso3c_kr(aid, "country_kr") #you copy paste the column name that has the Korean count

aid[c(50, 150, 250, 350, 450, 550),c(1, 6, 2:5)] |> gt::gt()
```

country_kr	year	sector	no_of_projects	aid_usd	aid_krw
베트남	2019	통신정책, 계획 및 행정(voluntary code)	2	232334	270736486
캄보디아	2019	11321	1	85815	99999361
미얀마	2019	사회보호/보장	1	103460	120560903
라오스	2019	비정규 농업훈련	1	107958	125802378
몽골	2019	의료서비스	5	511824	596423389
필리핀	2019	농업용수자원	2	0	0

Once you know the iso3c country codes, you can get the English country names, or other country codes (such as Correlates of War country codes) using the `countrycode` package, for example.

```
trade_long <- trade_long |>
  mutate(country_name = countrycode::countrycode(iso3c, origin = "iso3c", destination = "coun

trade_long[c(50, 150, 250, 350, 450, 550),c(1, 5, 6, 2:4)] |> gt::gt()
```

country_kr	iso3c	country_name	year	export_kosis	import_kosis
계	NA	NA	2014	572664607000	525514506000
아랍에미리트 연합	ARE	United Arab Emirates	1996	1377933000	2259205000
앤티가바부다	ATG	Antigua & Barbuda	1978	NA	NA
앵귈라	AIA	Anguilla	2019	817000	1000
아르메니아	ARM	Armenia	2001	1255000	43000
앙골라	AGO	Angola	1983	235000	NA

More importantly, `iso3c_kr` function allows users to be able to join different datasets that have Korean country names. For example, one can join the trade data with the aid data using the iso3c country codes. In this example, I will join the trade data with the aid data using the iso3c country codes.

```
trade_aid <- trade_long |>
  left_join(aid, by = c("iso3c", "year"), suffix = c("_trade", "_aid"))

trade_aid |>
  filter(year == 2019 & !is.na(iso3c)) |> # just as a sample, we only need the data for 2019
  slice(c(30, 130, 230, 330, 430, 530)) |> # just as a sample, let's only look at the rows 30
  select(c(iso3c, country_kr_trade, country_kr_aid, year, export_kosis, import_kosis, aid_usd
  gt::gt()
```

iso3c	country_kr_trade	country_kr_aid	year	export_kosis	import_kosis	aid_usd
AFG	아프가니스탄	아프가니스탄	2019	49930000	38000	6081
BGD	방글라데시	방글라데시	2019	1282342000	404703000	746593
BOL	볼리비아	볼리비아	2019	30434000	450576000	535262
COD	콩고 민주공화국	콩고민주공화국(DR콩고)	2019	37083000	411274000	0
CHN	중국	중국	2019	136202533000	107228736000	0
DOM	도미니카 공화국	도미니카공화국	2019	252420000	88516000	25792

Voilà! Now we have a dataset that has both trade and aid data, both of which originally did not have consistent country names or country codes. If we only used `country_kr` column to join the two datasets, we would have failed to merge all the data, such as “콩고 민주공화국” and

“콩고민주공화국(DR콩고)”, both of which are Democratic Republic of the Congo; or “도미니카 공화국” and “도미니카공화국” (Dominican Republic) which merelt have a space difference between the words. But with the `iso3c_kr` function, we were able to merge the two datasets successfully.

5.10 Working with dates

To be added

6 Data Visualization: Tables

7 Data Visualization: Plots

[A Very Early Draft]

In this chapter, we will learn data visualization using the `ggplot2` package, which is part of the `tidyverse` family. We will learn how to create scatter plots, line plots, bar plots, and histograms. We will also learn how to customize the appearance of these plots. Since this is a beginner's guide, we will only cover the very basics of data visualization.

For more advanced topics, please refer to the official documentation of the `ggplot2` package. You can also check out the cheat sheet for `ggplot2` here. You can find other R cheat sheets here.

Before we begin, I would like to emphasize that learning how to create plots is important, but learning which plot to use when is equally important. I am not covering the latter here, but there are some great books and resources available that can help you with that, including

ggplot2 (3e)

Data Visualization: A Practical Introduction

Modern Data Visualization with R

Visual Vocabulary

The R Graph Gallery

Dataviz Project

Friends Don't Let Friends Make Bad Graphs

Let's get started by loading the `tidyverse` package, which automatically loads the `ggplot2` package.

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

# install.packages("devtools") # if you haven't installed the devtools package yet, remove th

# devtools::install_github("kjayhan/kdiplo") # if you haven't installed the kdiplo package ye

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

We will work with the Korean trade data, and some additional data, which we have already cleaned and prepared in the previous chapter. I copy paste the same code here for your convenience.

```
trade <- kdiplo::trade_data |> # making sure that we get trade_data from the kdiplo package,
  select(iso3c, country, year, export_kosis, import_kosis) |> # select columns iso3c, countr
  filter(year > 1964) |> # filter rows where year is greater than 1964
  mutate(trade_kosis = export_kosis + import_kosis) # create a new column trade_kosis for Ko

# install.packages("WDI") # if you haven't installed the WDI package yet, remove the # sign.

library(WDI) # load the WDI package

wdi <- WDI(country = "all", # all countries
           indicator = c("gdp" = "NY.GDP.MKTP.KD", # GDP at constant 2015 US dollars
                         "gdp_pc" = "NY.GDP.PCAP.KD"), # GDP per capita at constant 2015 US
           start = 1965, # start year
           end = 2024, # end year
           extra = TRUE, # include extra columns included in the WDI package defaults
           language = "en" # language is English
           ) |>
    select(-iso2c, -status, -lastupdated, -capital, -lending, -longitude, -latitude, -countr
  filter(iso3c != "" | is.na(iso3c)) # exclude the rows that are missing country codes

# let's join these two datasets.

df <- trade |>
  inner_join(wdi, by = c("iso3c", "year")) # join the trade and wdi datasets by iso3c and yea

head(df)
```

```
# A tibble: 6 x 10
  iso3c country  year export_kosis import_kosis trade_kosis   gdp gdp_pc region
  <chr> <chr>   <dbl>        <dbl>        <dbl>       <dbl> <dbl>  <dbl> <chr>
1 ABW   Aruba    1965           NA           NA          NA    NA     NA Latin ~
2 ABW   Aruba    1966           NA           NA          NA    NA     NA Latin ~
3 ABW   Aruba    1967           NA           NA          NA    NA     NA Latin ~
4 ABW   Aruba    1968           NA           NA          NA    NA     NA Latin ~
5 ABW   Aruba    1969           NA           NA          NA    NA     NA Latin ~
6 ABW   Aruba    1970           NA           NA          NA    NA     NA Latin ~
# i 1 more variable: income <chr>
```

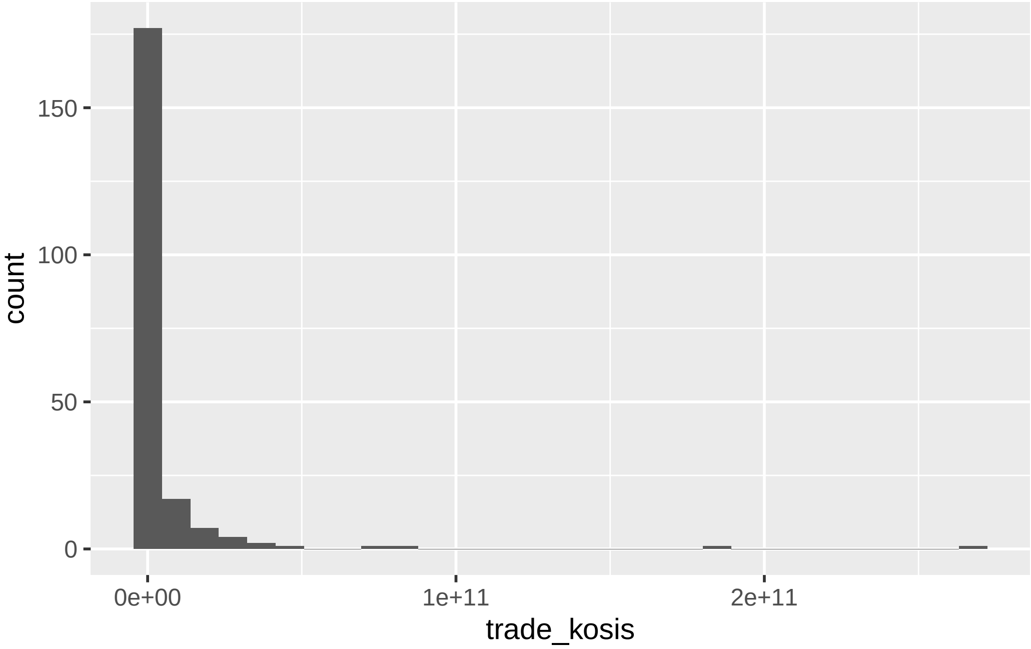
7.1 Histograms

A histogram is a graphical representation of the distribution of a continuous variable. It is a type of bar plot that shows the frequency of values in a dataset. The bars in a histogram are adjacent to each other, and the height of each bar represents the frequency of values in a specific range.

To create a histogram, we use the `geom_histogram()` function in `ggplot2`. Let's create a histogram of Korea's trade volume.

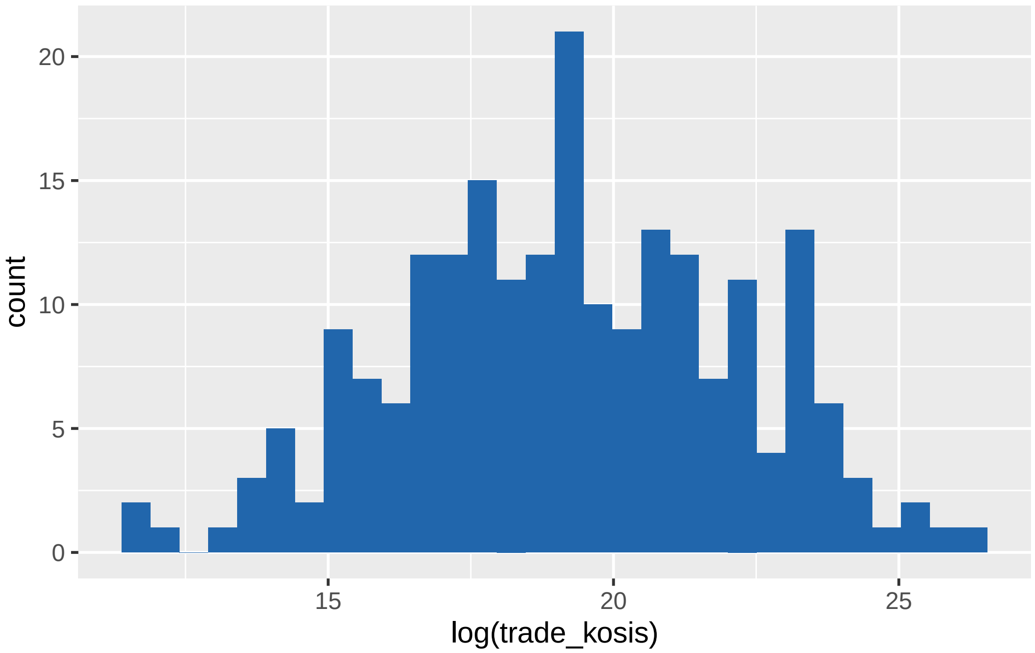
In ggplot2, we first specify the dataset and the aesthetic mappings using the `ggplot()` function. In `geom_histogram()`, we specify only the x-axis variable using the `aes()` function. We then add layers to the plot using the + operator. We should be careful not to use a pipe (|> or %>%) instead of + operator within `ggplot()` calls. In this case, we specify the x-axis variable using the `aes()` function and add the `geom_histogram()` layer to create the histogram.

```
# Create a histogram of Korea's trade volume
df |>
  filter(year == 2023) |> # filter rows to include only the year 2023
  ggplot(aes(x = trade_kosis)) + # specify the x-axis variable
  geom_histogram() # create a histogram
```



This is not very useful as a histogram because of the skewed distribution of the trade volume. We can improve the appearance of the histogram by using log transformation.

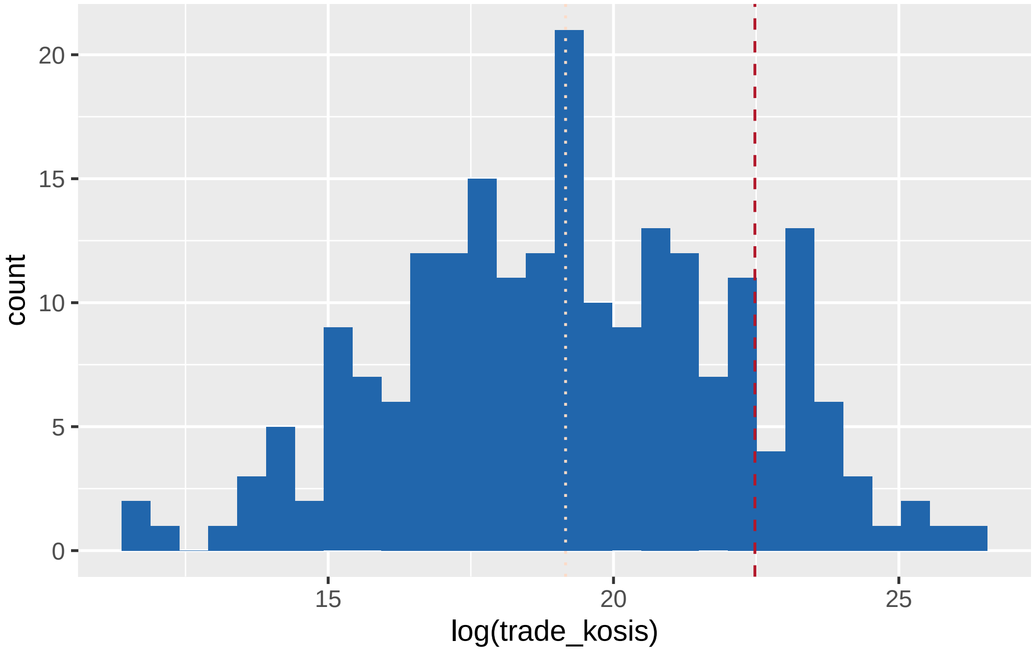
```
# Create a histogram of Korea's trade volume with log transformation and assign it to an obj

hist1 <- df |>
  filter(year == 2023) |> # filter rows to include only the year 2023
  ggplot(aes(x = log(trade_kosis))) + # specify the x-axis variable with log transformation
  geom_histogram(# you can specify the number of bins using the bins argument, e.g., bins =
    fill = "#2166ac") # create a histogram with blue bars; you can change the color by speci


hist1 # print the histogram
```



This is more informative than the previous histogram. This time let's add a vertical line to the histogram to show the mean and median of trade volume. We need to remember to use `na.rm = TRUE` in the `mean()` and `median()` functions to remove missing values from the calculation. Otherwise, the functions will return `NA`.

We can draw these vertical lines by adding the `geom_vline()` layer to the plot. We can also change the color of the line using the `color` arguments in the `geom_vline()` function.

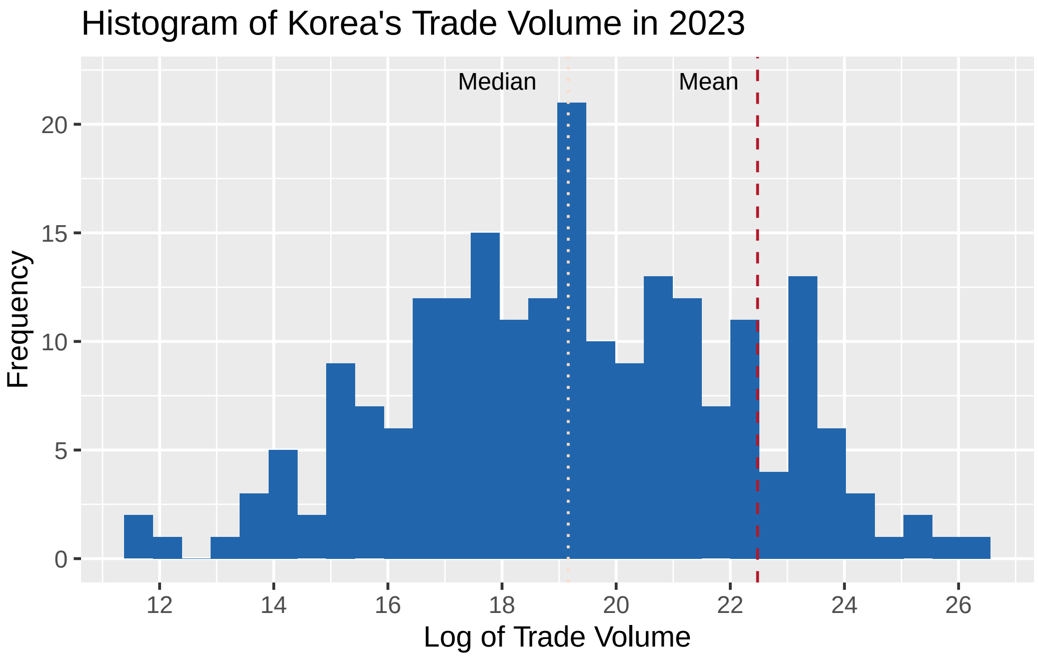
```
# since we are going to use the same histogram object, we can add layers to it. Let's call i

hist2 <- hist1 +
  geom_vline(aes(xintercept = log(mean(trade_kosis, na.rm = TRUE))), color = "#b2182b", line
  geom_vline(aes(xintercept = log(median(trade_kosis, na.rm = TRUE))), color = "#fddbc7", li
hist2 # print the histogram
```



The dashed line represents the mean of the trade volume, and the dotted line represents the median of the trade volume. The mean is greater than the median, which indicates that the distribution is right-skewed, with most values concentrated around the median, which is 17.9179225 (formula = `log(median(df$trade_kosis, na.rm = TRUE))`), which corresponds to 6.0486×10^7 (formula = `exp(log(median(df$trade_kosis, na.rm = TRUE)))`) in the original scale. `exp()` is the inverse of `log()`.

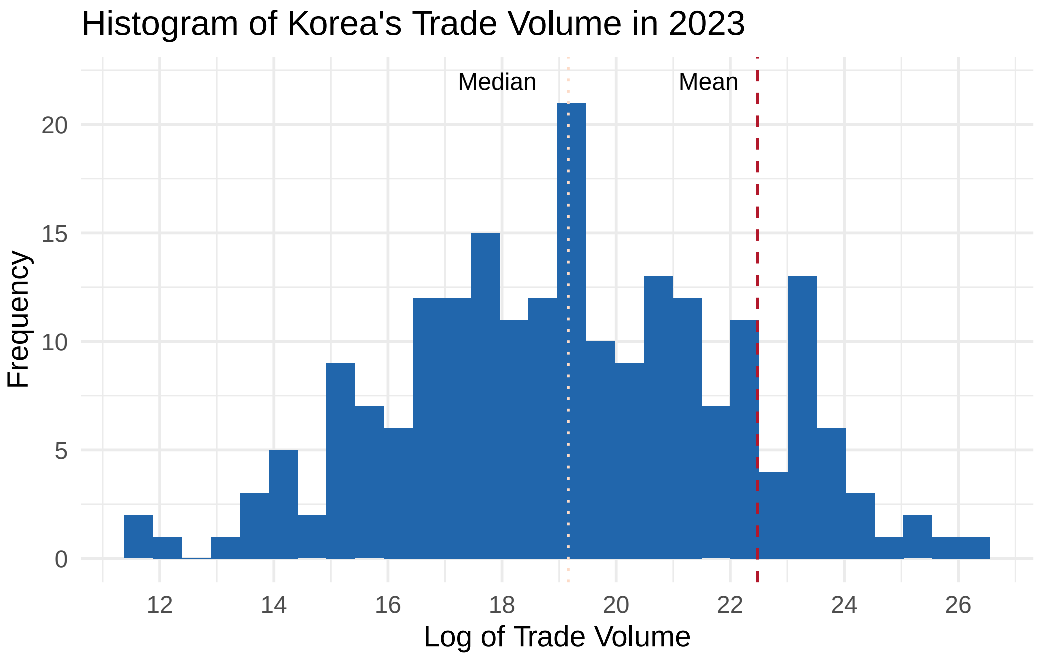
We can add a title, x-axis label, and y-axis label to the histogram using the `labs()` function. We can change the breaks and labels of the x-axis using the `scale_x_continuous()` function. We can also add annotations to the plot using the `annotate()` function.

```
# Add a title, x-axis label, and y-axis label to the histogram

hist3 <- hist2 +
  labs(title = "Histogram of Korea's Trade Volume in 2023", x = "Log of Trade Volume", y = "F
  scale_x_continuous(breaks = seq(0, 30, 2)) + # change the breaks of the x-axis
  annotate("text", x = log(median(df$trade_kosis, na.rm = TRUE)), y = 22, label = "Median", c
  annotate("text", x = log(mean(df$trade_kosis, na.rm = TRUE)), y = 22, label = "Mean", colo

hist3 # print the histogram
```



This histogram is now more informative and visually appealing. We can also change the theme of the plot using the `theme_minimal()` function. For other themes, you can check out `ggtthemes` package as well as many others out there.

```
# Change the theme of the histogram

hist3 +
  theme_minimal() # change the theme of the plot to minimal; you can also use other themes su
```



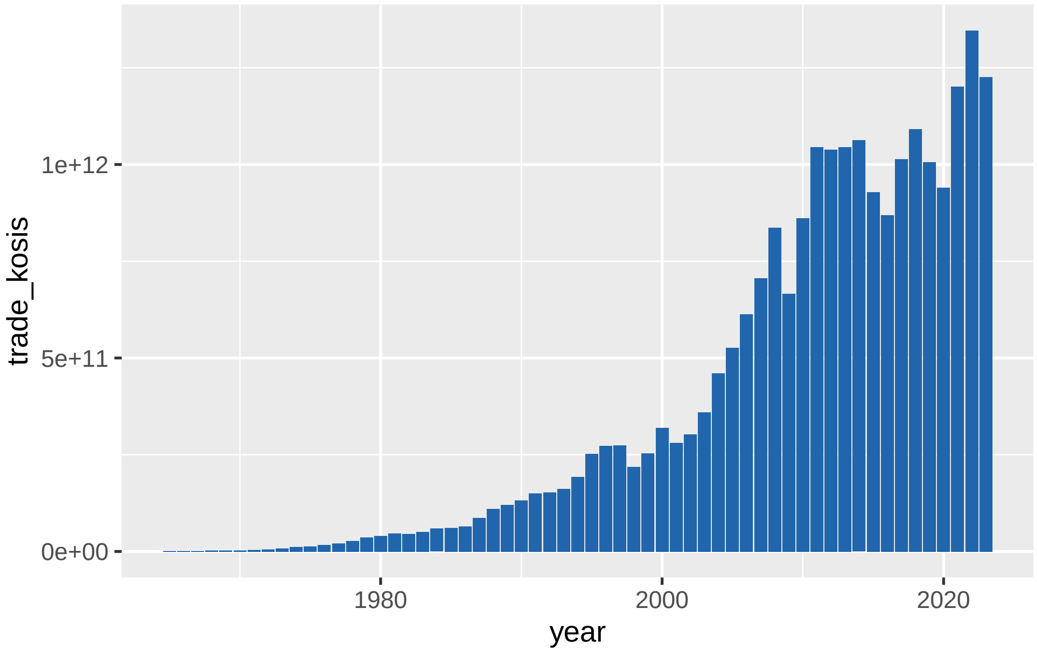
```
  # theme_light() # change the theme of the plot to light
  # theme_dark() # change the theme of the plot to dark
  # theme_bw() # change the theme of the plot to black and white
```

7.2 Bar Plots

A bar plot is a graphical representation of the distribution of a categorical variable. It is a type of plot that shows the frequency of categories in a dataset. The bars in a bar plot are separated from each other, and the height of each bar represents the frequency of categories.

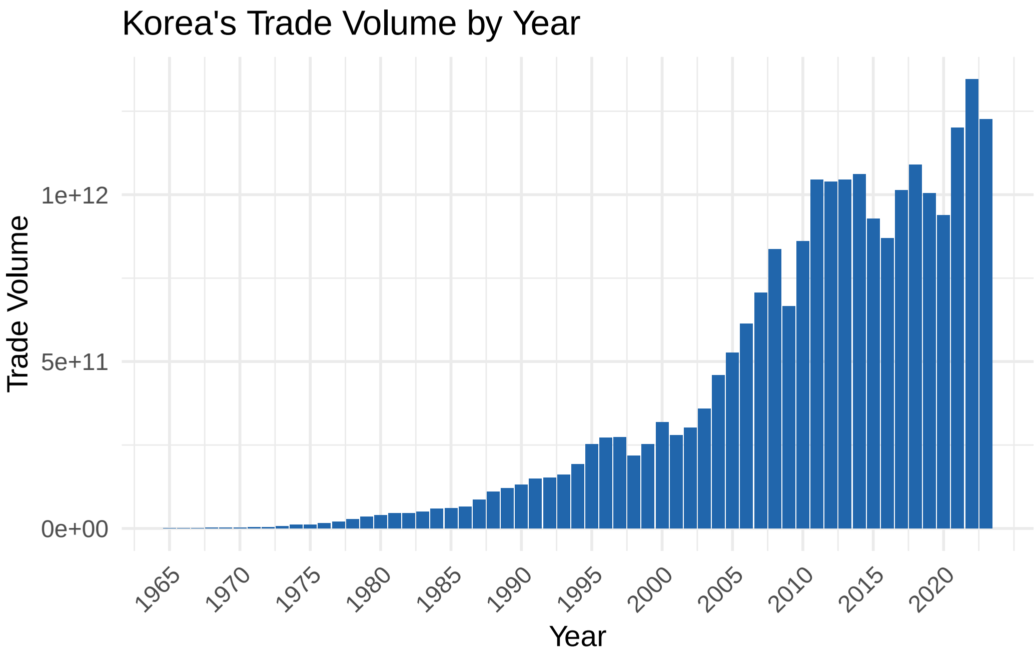
To create a bar plot, we use the `geom_bar()` function in `ggplot2`. Let's create a bar plot of Korea's trade volume by year.

```
# Create a bar plot of Korea's trade volume by year

bar1 <- df |>
  ggplot(aes(x = year, y = trade_kosis)) + # specify the x-axis and y-axis variables
  geom_col(fill = "#2166ac") # create a bar plot with the actual values of trade volume

# geom_col() is an alias for geom_bar(stat = "identity"), which is used when the y-axis vari

bar1 # print the bar plot
```



Let’s customize the appearance by adding a title, x-axis label, and y-axis label, and changing the theme of the plot, and changing the scale of the x-axis.

```
# Customize the appearance of the bar plot

bar2 <- bar1 +
  labs(title = "Korea's Trade Volume by Year", x = "Year", y = "Trade Volume") + # add a titl
  scale_x_continuous(breaks = seq(1965, 2023, 5)) + # change the breaks of the x-axis
  theme_minimal() + # change the theme of the plot to minimal. Be careful here, if this theme
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) # change the angle and justificati

bar2 # print the bar plot
```



This bar plot is now more informative and visually appealing.

7.3 Line Plots

A line plot is a graphical representation of the relationship between two continuous variables. It is a type of plot that shows the trend of a variable over time or another continuous variable. The lines in a line plot are connected, and the slope of each line represents the relationship between the variables.

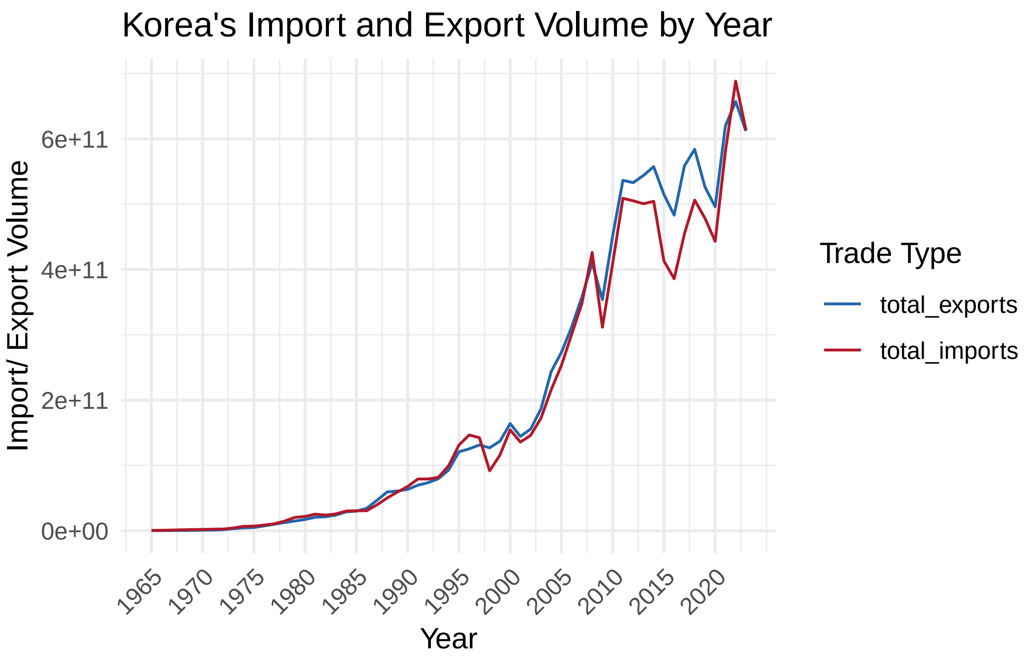
To create a line plot, we use the `geom_line()` function in `ggplot2`. Let's create a line plot of Korea's total exports and total imports by year. Let's first group the data by year and calculate the total exports and total imports for each year.

```
# Group the data by year and calculate the total exports and total imports for each year

df_year <- df |>
  group_by(year) |> # group the data by year
  summarise(total_exports = sum(export_kosis, na.rm = TRUE),
            total_imports = sum(import_kosis, na.rm = TRUE)) # calculate the total exports a
```

Now let's put imports and exports volume amounts into the same column, using the `pivot_longer()` function, which we learned in the previous chapter.

```
# Put imports and exports volume amounts into the same column

df_year <- df_year |>
  pivot_longer(cols = c(total_exports, total_imports), names_to = "type", values_to = "volume
```

```
# Create a line plot of Korea's exports and imports by year

line1 <- df_year |>
  ggplot(aes(x = year, y = volume, color = type)) + # specify the x-axis variable
  geom_line() + # create a line plot for exports
  scale_color_manual(values = c("#2166ac", "#b2182b")) + # change the colors of the lines
  scale_x_continuous(breaks = seq(1965, 2023, 5)) + # change the breaks of the x-axis
  labs(title = "Korea's Import and Export Volume by Year", x = "Year", y = "Import/ Export Vo
  theme_minimal() + # change the theme of the plot to minimal. Be careful here, if this theme
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) # change the angle and justificati

line1 # print the line plot
```



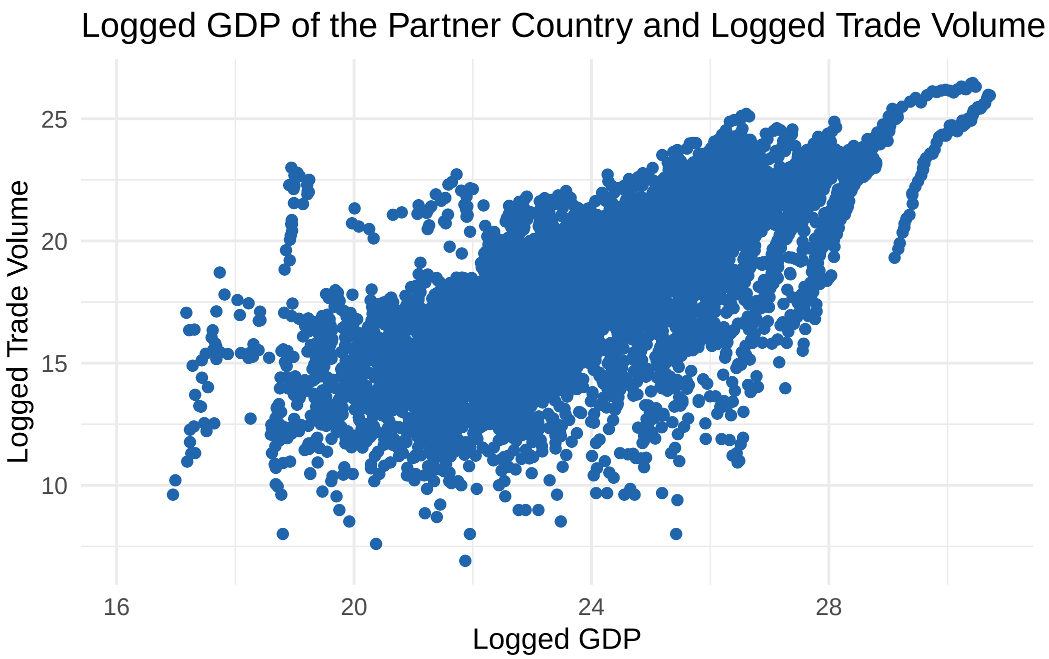
This line plot shows the trend of Korea’s total exports and total imports over time. The blue line represents total exports, and the red line represents total imports.

7.4 Scatter Plots

A scatter plot is a graphical representation of the relationship between two continuous variables. It is a type of plot that shows the correlation between two variables. The points in a scatter plot are not connected, and the position of each point represents the values of the variables.

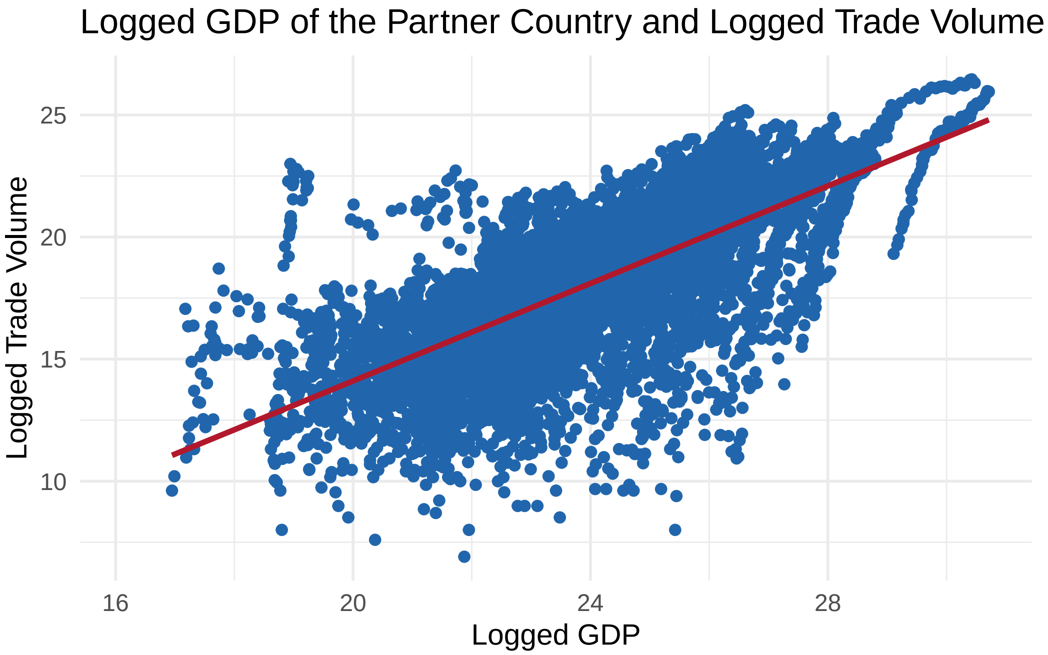
To create a scatter plot, we use the `geom_point()` function in `ggplot2`. Let's create a scatter plot of logged GDP of Korea's trade partner and the bilateral logged trade volume. We take logs because of skewed distributions of both data.

```
# Create a scatter plot of partner country's logged GDP and bilateral logged trade volume

scatter1 <- df |>
  ggplot(aes(x = log(gdp), y = log(trade_kosis))) + # specify the x-axis and y-axis variables
  geom_point(color = "#2166ac") + # create a scatter plot with blue points
  labs(title = "Logged GDP of the Partner Country and Logged Trade Volume", x = "Logged GDP"
  theme_minimal() # change the theme of the plot to minimal

scatter1 # print the scatter plot
```



To this, we can add a linear regression line to show the relationship between the variables. We can do this by adding the `geom_smooth()` layer to the plot. We can also change the color of the line using the `color` argument in the `geom_smooth()` function.

```
# Add a linear regression line to the scatter plot

scatter2 <- scatter1 +
  geom_smooth(method = "lm", se = FALSE, color = "#b2182b") # add a linear regression line wi

scatter2 # print the scatter plot
```



This scatter plot shows the relationship between the logged GDP of Korea's trade partner and the bilateral logged trade volume. The linear regression line indicates a positive relationship between the variables, as expected.

Now let's differentiate the points by the income level of the partner country. We can do this by adding the `color` argument in the `aes()` function and specifying the `income` variable.

First, let's see the categories and frequency of each category in the income variable.

```
# Check the categories in the income variable

table(df$income) # count the frequency of each category in the income variable
```

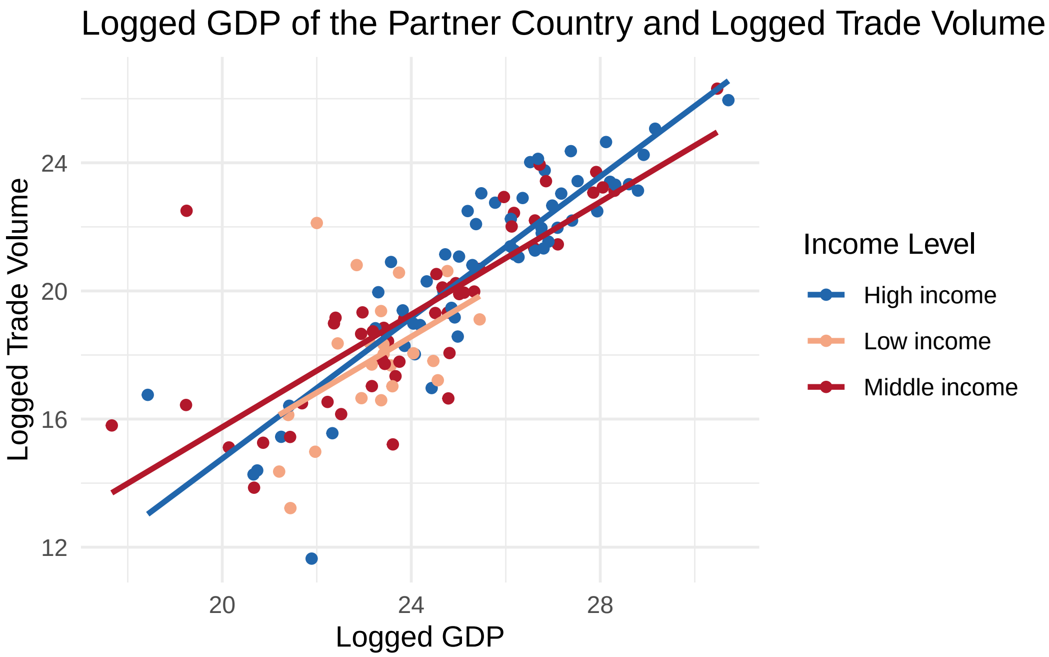
```

        High income          Low income Lower middle income      Not classified 
               4543                1652                3127                  59 
Upper middle income 
```

```
3134
```

To make the visual neat, we can group the income levels into three categories: low, middle, and high. We can do this by using the `case_when()` function in the `mutate()` function.

```
# Group the income levels into three categories: low, high, and not classified

df <- df |>
  mutate(income2 = case_when(
    income %in% c("Low income") ~ "Low income", # if the income level is "Low income", assig
    income %in% c("Lower middle income ", "Upper middle income") ~ "Middle income", # if the
    income %in% c("High income") ~ "High income", # if the income level is "High income", as
    TRUE ~ "Not Classified" # otherwise, assign it to the Not Classified category
  ))
```

```
# Differentiate the points by the income level of the partner country

scatter3 <- df |>
  filter(income2 != "Not Classified" & year == 2023) |> # filter rows to exclude the Not Cla
  ggplot(aes(x = log(gdp), y = log(trade_kosis), color = income2)) + # specify the x-axis an
  geom_point() + # create a scatter plot
  labs(title = "Logged GDP of the Partner Country and Logged Trade Volume by Income Level",
  geom_smooth(method = "lm", se = FALSE) + # add a linear regression line without confidence
  scale_color_manual(values = c("#2166ac", "#f4a582", "#b2182b")) + # change the colors of t
  theme_minimal() # change the theme of the plot to minimal

scatter3 # print the scatter plot
```


Logged GDP of the Partner Country and Logged Trade Volume
Logged Trade Volume
24
20
16
12
20
24
28
Logged GDP
Income Level
High income
Low income
Middle income

8 Data Visualization: Maps

9 Korean Text Analysis

In this chapter, we will learn how to analyze Korean text data using R. We will use the tidyverse, pdftools, and bitNLP packages to extract text from a pdf file and analyze it. We will use Korea's 2022 Diplomatic White Paper (외교백서, waegyo baekseo) as an example text.

We will learn the following things in order:

- Extracting text and tables from a PDF file.
- Extracting text and tables from the internet.
- Ensuring accurate spacing between words in Korean text.
- Analyzing morphemes in Korean text.
- Analyzing word frequency in Korean text.
- Analyzing the noun word network in Korean text.
- Analyzing the sentiment of Korean text.
- Topic modeling of Korean text.

9.1 Libraries

First, we need to install bitNLP which requires us to install the MeCab library for Korean text analysis. Uncomment the following lines in your first usage. After the first usage, you can comment out the installation lines.

```
# install.packages("remotes")
# remotes::install_github("bit2r/bitNLP")
library(bitNLP)
# install_mecab_ko()
# install.packages("RcppMeCab")
```

Now let's load the necessary libraries. If you are missing any of the following packages, you can install them by uncommenting the install.packages lines.

```r
# install.packages("tidyverse")
# install.packages("pdftools")
# install.packages("rvest")
# install.packages("tidytext")
# install.packages("igraph")
# install.packages("ggraph")
# install.packages("extrafont")
library(tidyverse)
library(pdftools)
library(rvest)
library(tidytext)
library(igraph)
library(ggraph)
library(extrafont)
```

9.2 Loading pdf Data

Let's analyze the text from Korea's 2024 Public Diplomacy Comprehensive Implementation Plan (2024년 공공외교 종합시행계획 개요) which is available as a pdf file on the Ministry of Foreign Affairs' (MOFA) website[1].

If the pdf file is in your local directory, you can load it using the following code.

```r
# Load PDF
pdf_path <- "data/2024     .pdf"
```

Alternatively, you can download the pdf file from the MOFA's website using the `download.file` function. You can then load the pdf file using the `pdf_path` variable. Working with the online pdf file and the local pdf file is the same. We can do either. For now, I will use the local pdf file since the MOFA might change the url for the pdf later. That is why I commented the download code. You can comment the earlier code for the local pdf file and uncomment the following code for the online pdf file.

```r
# Download PDF
#file <- tempfile()

# This url works for now. But MOFA might change it later. You can replace the link with any
```

[1] Please bear in mind that MOFA website's url might change later, making this hyperlink broken. In that case, you can download the pdf file on the MOFA's website by searching for "2024년 공공외교 종합시행계획 개요".

```
#url <- "https://www.mofa.go.kr/cntntsDown.do?path=www&physic=2024%EB%85%84%EB%8F%84_%EA%B3%

# download.file(url, pdf_path, headers = c("User-Agent" = "My Custom User Agent"))
```

Now let's extract the text from the pdf file using the `pdf_text` function from the `pdftools` package.

```
# Extract text
pdf_text_all <- pdf_text(pdf_path)
```

Now, `pdf_text_all` is a list of character vectors, where each element corresponds to a page in the pdf file. For example, we can look at the 4th page of the pdf file in the following way.

```
# Let's look at the 4th page
pdf_text_all[4]
```

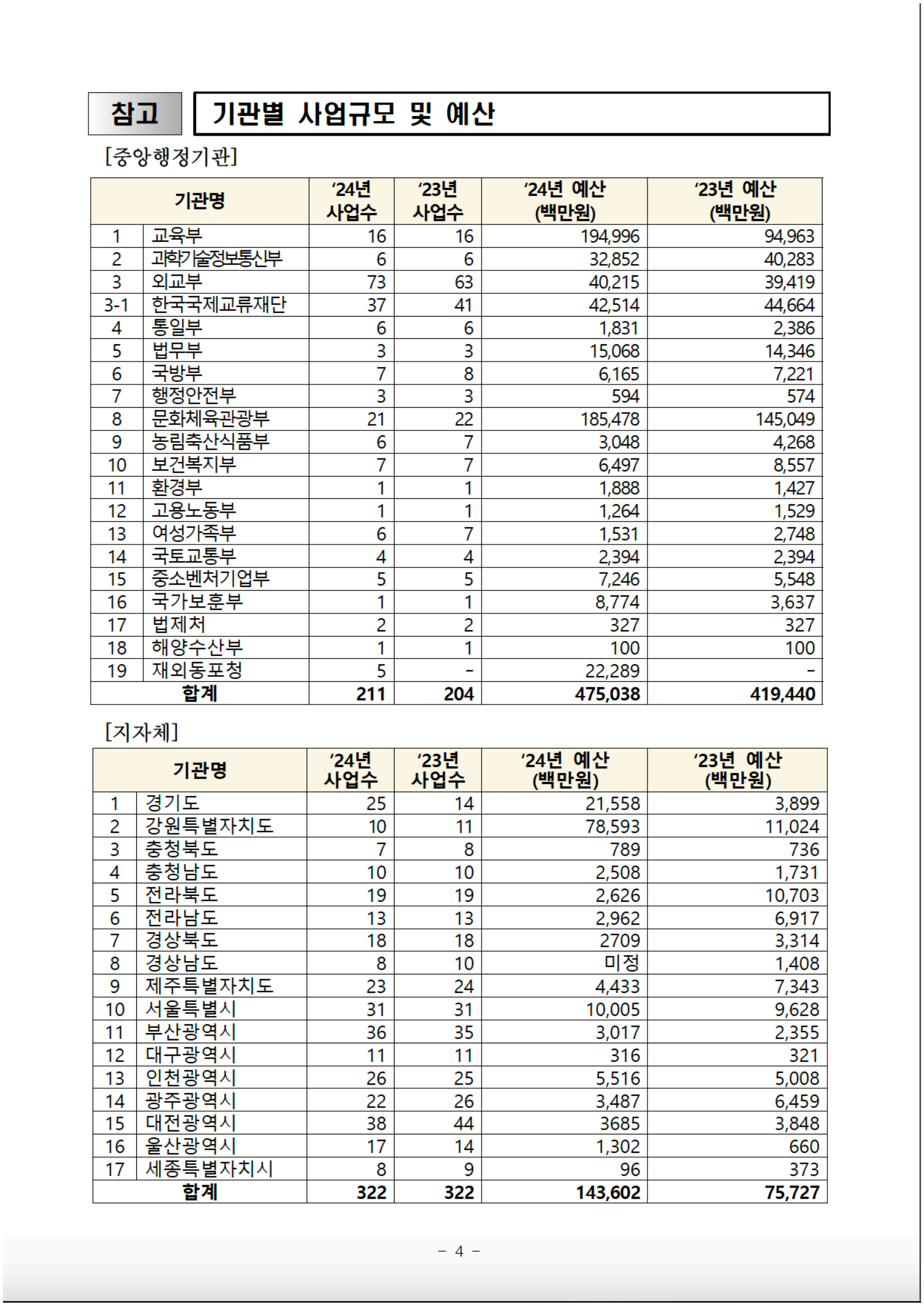
```
[1] "                 \n[   ]\n                  '24      '23       '24              '23   \n
```

Oh, this is too long even for an example. But you can realize that there are many `\n` characters in the text. Let's split the text by the newline character and look at the first 10 lines of the 4th page. `\n` refers to a new line in the text. We can split the text into lines by using the `str_split` function from the `stringr` package, which is part of `tidyverse`. So, we don't need to load it separately. Let's look at the first six lines of the 4th page.

```
# Look at the first 10 lines of the 4th page
pdf_text_all[4] |>
  # Split by newline character.
  str_split("\n") |>
  # Unlist
  unlist() |>
  # Take the first 10 lines
  head(10)
```

```
 [1] "                 "
 [2] "[   ]"
 [3] "                  '24      '23       '24              '23   "
 [4] "         "
 [5] "                                          ( )             ( )"
 [6] " 1                   16        16           194,996           94,963"
 [7] " 2                 6         6           32,852           40,283"
 [8] " 3                  73       63           40,215          39,419"
 [9] "3-1               37       41           42,514          44,664"
[10] " 4                   6        6            1,831           2,386"
```

The 4th page in the pdf file looks like this:



참고 기관별 사업규모 및 예산

[중앙행정기관]

기관명		'24년 사업수	'23년 사업수	'24년 예산 (백만원)	'23년 예산 (백만원)
1	교육부	16	16	194,996	94,963
2	과학기술정보통신부	6	6	32,852	40,283
3	외교부	73	63	40,215	39,419
3-1	한국국제교류재단	37	41	42,514	44,664
4	통일부	6	6	1,831	2,386
5	법무부	3	3	15,068	14,346
6	국방부	7	8	6,165	7,221
7	행정안전부	3	3	594	574
8	문화체육관광부	21	22	185,478	145,049
9	농림축산식품부	6	7	3,048	4,268
10	보건복지부	7	7	6,497	8,557
11	환경부	1	1	1,888	1,427
12	고용노동부	1	1	1,264	1,529
13	여성가족부	6	7	1,531	2,748
14	국토교통부	4	4	2,394	2,394
15	중소벤처기업부	5	5	7,246	5,548
16	국가보훈부	1	1	8,774	3,637
17	법제처	2	2	327	327
18	해양수산부	1	1	100	100
19	재외동포청	5	-	22,289	-
합계		**211**	**204**	**475,038**	**419,440**

[지자체]

기관명		'24년 사업수	'23년 사업수	'24년 예산 (백만원)	'23년 예산 (백만원)
1	경기도	25	14	21,558	3,899
2	강원특별자치도	10	11	78,593	11,024
3	충청북도	7	8	789	736
4	충청남도	10	10	2,508	1,731
5	전라북도	19	19	2,626	10,703
6	전라남도	13	13	2,962	6,917
7	경상북도	18	18	2709	3,314
8	경상남도	8	10	미정	1,408
9	제주특별자치도	23	24	4,433	7,343
10	서울특별시	31	31	10,005	9,628
11	부산광역시	36	35	3,017	2,355
12	대구광역시	11	11	316	321
13	인천광역시	26	25	5,516	5,008
14	광주광역시	22	26	3,487	6,459
15	대전광역시	38	44	3685	3,848
16	울산광역시	17	14	1,302	660
17	세종특별자치시	8	9	96	373
합계		**322**	**322**	**143,602**	**75,727**

- 4 -

Figure 9.1: 2024 Public Diplomacy Comprehensive Implementation Plan, p. 4

9.3 pdf Table Extraction

Let's try to extract the second table on page 4 of the pdf file. The table has the number of public diplomacy projects and budgets for first-tier local administration unit (hereafter, province_city for short) in Korea. We will unlist each line as we did earlier so that we can see the table in a more readable way.

```
# Look at the first 10 lines of the 4th page
lines_pdf_4 <- pdf_text_all[4] |>
  # Split by newline character.
  str_split("\n") |>
  # Unlist
  unlist()
```

First, let's look at the 29th and 30th lines for the column names in the pdf file.

```
lines_pdf_4[29:30]
```

```
[1] "                  ‘24      ‘23      ‘24             ‘23   "
[2] "         "
```

The column names are the line number, province or city's name, project numbers for 2024 and 2023 respectively, and the budget for 2024 and 2023 in million Korean Won respectively. Let's use the following English column names that correspond to the Korean column names in the pdf file.

```
# Column names
col_names <- c("no", "province_city", "project_no_2024", "project_no_2023", "budget_2024", "b
```

By observing the lines_pdf_4 object using view(lines_pdf_4), we can see that the second table starts from the 32nd line and ends on the 48th. We will extract only those lines. We will use str_trim "removes whitespace from start and end of string". We will also use str_replace_all to remove commas from each line to convert entries into numbers. We will then split each line based on two or more consecutive spaces (our string is "\s{2,}") using str_split and simplify the result into a matrix. We will convert this matrix into a data frame with non-factor columns using data.frame(stringsAsFactors = FALSE). We will set the column names of the data frame using the col_names vector that we created above. These explanations are also available in each step in the following code chunk.

```r
# Select lines 32 to 48 from the lines_pdf_4 data frame
province_city_pd <- lines_pdf_4[32:48] |>
  # Trim whitespace from both ends of each element in the selected rows
  str_trim() |>
  # Replace all commas with an empty string in each element
  str_replace_all(",", "") |>
  # Split each element based on 2 or more consecutive spaces and simplify into a matrix
  str_split("\\s{2,}", simplify = TRUE) |>
  # Convert the matrix into a data frame with non-factor columns
  data.frame(stringsAsFactors = FALSE) |>
  # Set column names for the data frame using the provided 'col_names' vector
  setNames(col_names)
```

Let's rearrange the table (which is originally in alphabetical order) by descending order based on public diplomacy budgets in 2024.

```r
province_city_pd |>
  arrange(desc(budget_2024))
```

```
   no  province_city project_no_2024 project_no_2023 budget_2024 budget_2023
1   8                        8              10                 1408
2  17                    8               9          96         373
3   3                        7               8         789         736
4   2                   10              11       78593       11024
5  13                      26              25        5516        5008
6   9                   23              24        4433        7343
7  15                      38              44        3685        3848
8  14                      22              26        3487        6459
9  12                      11              11         316         321
10 11                      36              35        3017        2355
11  6                       13              13        2962        6917
12  7                       18              18        2709        3314
13  5                       19              19        2626       10703
14  4                       10              10        2508        1731
15  1                         25              14       21558        3899
16 16                      17              14        1302         660
17 10                      31              31       10005        9628
```

But these province_city names are in Korean since the document was in Korean. Let's practice extracting a table from internet then to find English names for these Korean provinces or cities. As of May 6, 2024, Wikipedia's list of South Korea's administrative divisions seems to be correct. Let's extract the table there.

9.4 html Table Extraction

We will use the rvest package to extract the table from the Wikipedia page. We will use the read_html function to read the html content of the Wikipedia page. We will then use the html_node function to select the table we want to extract. You can refer to rvest package for more information on how to extract what you want. We can use the xpath of the table we want to extract. You can find the xpath of the table by right-clicking on the table on the Wikipedia page and selecting "Inspect" or "Inspect Element" depending on your browser. You can then right-click on the highlighted html element in the "Elements" tab of the "Developer Tools" and select "Copy" -> "Copy XPath". The xpath of the table we want to extract is //*[@id="mw-content-text"]/div[1]/table[5]. We will use the html_table function to extract the table as a data frame. We will use the fill = TRUE argument to fill in the missing values in the table.

```
html <- read_html("https://en.wikipedia.org/wiki/Administrative_divisions_of_South_Korea")

table <- html |>
  html_node(xpath = '//*[@id="mw-content-text"]/div[1]/table[5]') |>
  html_table(fill = TRUE)
```

Let's look at the first 10 rows of the table.

```
head(table)
```

```
# A tibble: 6 x 9
  Code  Emblem Name   Official English nam~1 Hangul Hanja Population 2020 Cens~2
  <chr> <lgl>  <chr>  <chr>                  <chr>  <chr> <chr>
1 KR-11 NA     Seoul~ Seoul                   ~ .mw--~ 9,586,195
2 KR-26 NA     Busan~ Busan                   ~   ~ 3,349,016
3 KR-27 NA     Daegu~ Daegu                   ~   ~ 2,410,700
4 KR-28 NA     Inche~ Incheon                 ~   ~ 2,945,454
5 KR-29 NA     Gwang~ Gwangju                 ~   ~ 1,477,573
6 KR-30 NA     Daeje~ Daejeon                 ~   ~ 1,488,435
# i abbreviated names: 1: `Official English name[5]`,
#   2: `Population 2020 Census`
# i 2 more variables: `Area (km2)` <chr>,
#   `Population density  2022 (per km2)` <chr>
```

Perfect! Now, let's keep only the columns that we will need.

```r
# Select columns 4 and 5 from the table
table <- table |>
  select(4:5)

# Let's change the English province_city column name.

table <- table |>
  rename(province_city_eng = `Official English name[5]`)
```

Let's hope that the Korean names in the Wikipedia table and the MOFA's pdf file are the same. Let's merge the two tables based on the Korean names.

```r
# Merge the two tables based on the Korean names
province_city_pd_joined <- province_city_pd |>
  left_join(table, by = c("province_city" = "Hangul"))
```

Let's see if we have any missing values in the English names.

```r
# Check for missing values in the English names
province_city_pd_joined |>
  filter(is.na(province_city_eng))
```

```
  no province_city project_no_2024 project_no_2023 budget_2024 budget_2023
1  5                       19              19        2626       10703
  province_city_eng
1              <NA>
```

We almost got it! The only difference is 전라북도 (North Jeolla Province) in the MOFA's pdf file which is written as 전북특별자치도 (Jeonbuk State) in the Wikipedia table. Let's fix this.

```r
# Move the English name column next to the Korean name column, and remove the 'no' column

province_city_pd_joined <- province_city_pd_joined |>
  select(province_city, province_city_eng, everything(), -no)

# Fix the English name of

province_city_pd_joined <- province_city_pd_joined |>
  mutate(province_city_eng = ifelse(province_city == "  ", "North Jeolla province_city", pro
```

9.5 Text Analysis

9.5.1 Word Frequency

This time let's look at all of the text in the 2024 Public Diplomacy Comprehensive Implementation Plan. We will combine all the text into a single character vector.

```
# Combine text
pdf_text <- str_c(pdf_text_all, collapse = " ")
```

We will now split the text into words using the `str_split` function from the `stringr` package. We will then convert the result into a data frame with non-factor columns using the `data.frame(stringsAsFactors = FALSE)` function. We will set the column name of the data frame as `word`.

```
# Split the text into words
words <- pdf_text |>
  # Split the text into words
  str_split("\\s+") |>
  # Convert the result into a data frame with non-factor columns
  data.frame(stringsAsFactors = FALSE) |>
  # Set the column name of the data frame as "word"
  setNames("word")
```

Let's look at the first 10 rows of the data frame.

```
head(words, 10)
```

```
           word
1
2         2024
3
4
5
6
7
8
9
10
```

Now, let's count the frequency of each word in the text using the `count` function from the `dplyr` package package. We will then arrange the result in descending order based on the frequency of the words.

```r
# Count the frequency of each word
word_freq <- words |>
  count(word, sort = TRUE)
```

Let's look at the first 10 rows of the data frame

```r
head(word_freq, 10)
```

```
       word  n
1          72
2         - 55
3     40
4          33
5          28
6       , 22
7         22
8         18
9         18
10        17
```

This is not very useful. There are two main issues with Korean text. First, Korean text does not have consistent spacing between words. Second, Korean text has particles and other morphemes that are not words. We will address these issues now.

9.5.2 Spacing in Korean Text

Let's get the spacing right in Korean text using the `bitNLP` package's `get_spacing` function, which will add spaces between words in the Korean text. So, for example "한국공공외교" will become "한국 공공 외교".

```r
# Get the spacing right in Korean text
pdf_text_ko <- get_spacing(pdf_text)
```

Now, let's split the text into words again using the `str_split` function from the `stringr` package.

```
# Split the text into words
words_ko <- pdf_text_ko |>
  # Split the text into words
  str_split("\\s+") |>
  # Convert the result into a data frame with non-factor columns
  data.frame(stringsAsFactors = FALSE) |>
  # Set the column name of the data frame as "word"
  setNames("word")
```

Let's analyze the word frequency in the text again.

```
# Count the frequency of each word
word_freq_ko <- words_ko |>
  count(word, sort = TRUE)

head(word_freq_ko, 10)
```

```
   word   n
1      175
2     (  97
3     -  80
4       73
5     67
6     62
7     36
8     35
9       33
10    30
```

We have many special characters in the text. Let's remove all characters except for Korean characters, spaces, English letters, and numbers using the `str_replace_all` function from the `stringr` package.

```
# Remove all characters except for Korean characters, spaces, English letters, and numbers
word_freq_ko <- pdf_text_ko |>
  # Remove all characters except Korean characters, English letters, numbers, and spaces
  str_replace_all("[^ - a-zA-Z0-9\\s]", "") |>
  # Split the cleaned text into words based on one or more spaces
  str_split("\\s+") |>
  # Convert the list result into a data frame with non-factor columns
  data.frame(stringsAsFactors = FALSE) |>
  # Set the column name of the data frame as "word"
  setNames("word")
```

Let’s analyze the word frequency in the text again.

```
# Count the frequency of each word
word_freq_ko <- word_freq_ko |>
  count(word, sort = TRUE)

head(word_freq_ko, 10)
```

```
   word  n
1      73
2     67
3     62
4     44
5     37
6     36
7      35
8     30
9     29
10    28
```

This is much better! We have removed the special characters and have more meaningful words in the text. Let's move on to morpheme analysis which makes more sense in Korean text analysis context.

9.5.3 Morpheme Analysis in Korean Text

Let’s analyze the morphemes in the Korean text using the `morpho_mecab` function from the `bitNLP` package, which will extract morphemes from the Korean text.

```
# Analyze the morphemes in the Korean text
morphemes <- morpho_mecab(pdf_text_ko)
```

This creates a list of character vectors, where each element corresponds to a morpheme in the text. We can also combine all of the morphemes and tokenize them into a single character vector.

```r
# Combine all the morphemes into a single character vector

morphemes_single <- morpho_mecab(pdf_text_ko, indiv = FALSE)
```

Now, let’s split the text into words again this time by converting `morphemes_single` into a data frame using the `as.data.frame` function. We will set the column name of the data frame as “word”.

```r
# Split the text into words
words_morphemes <- morphemes_single |>
  as.data.frame() |>
  # Set the column name of the data frame as "word"
  setNames("word")
```

We will now count the frequency of each morpheme in the text using the `count` function from the `dplyr` package package. We will then arrange the result in descending order based on the frequency of the morphemes.

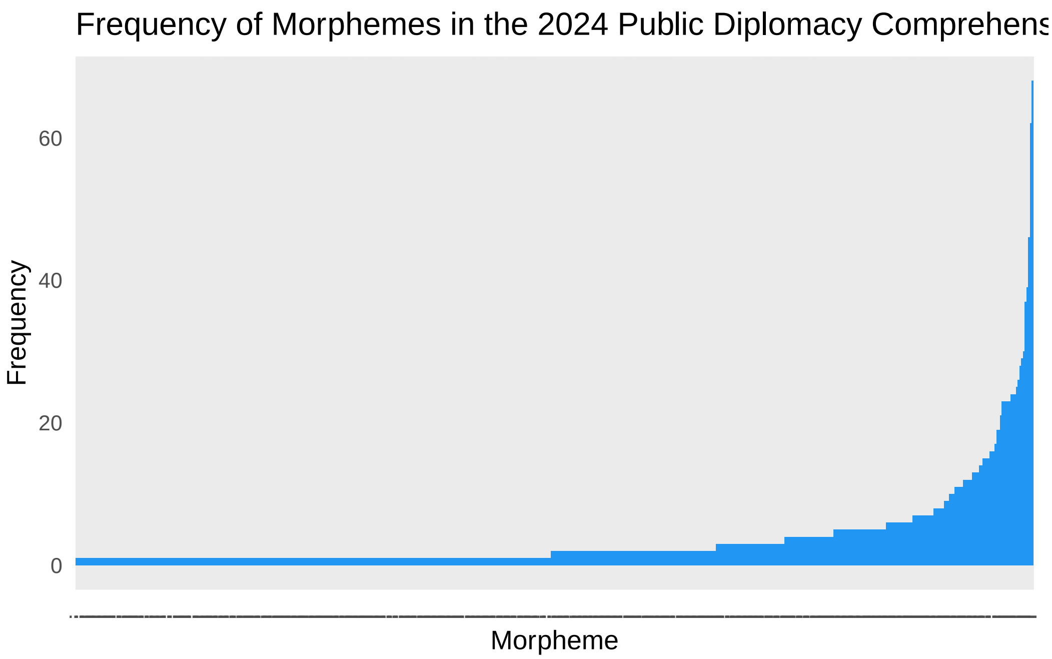
```r
# Count the frequency of each morpheme

morpheme_freq <- words_morphemes |>
  count(word, sort = TRUE)

head(morpheme_freq, 10)
```

```
   word  n
1     68
2     62
3     46
4     39
5     37
6     30
7     29
8     28
9     26
10    25
```

Now, this is more like it!

Let’s visualize the frequency of the morphemes in the text using a bar plot. We will use the `ggplot` function from the `ggplot2` package to create the plot. We will use the `geom_col` function to add the bars to the plot. We will use the theme_minimal function to set the theme of the plot to minimal. We will use the theme function to adjust the font size in the plot. We will set the font size to 10. We will use the labs function to add the title and labels to the plot.

```
# Visualize the frequency of the morphemes in the text

morpheme_freq |>
  # Create a bar plot
  ggplot(aes(x = reorder(word, n), y = n)) +
  geom_col(fill = "#2196f3") +
  theme_minimal() +
  theme(text = element_text(size = 10)) +
  labs(title = "Frequency of Morphemes in the 2024 Public Diplomacy Comprehensive Implementa
       x = "Morpheme",
       y = "Frequency")
```



9.5.4 Word Network in Korean Text

Let's analyze the word network in the Korean text using the tokenize_noun_ngrams function from the bitNLP package which builds on tidytext package. We will use the tokenize_noun_grams function to extract the noun word network from the Korean text.

```
# We can use a user-defined dictionary to improve the accuracy of the tokenization. We will

dic_path <- system.file("dic", package = "bitNLP")
dic_file <- glue::glue("{dic_path}/buzz_dic.dic")

word_network <- tokenize_noun_ngrams(pdf_text_ko, simplify = TRUE, user_dic = dic_file, n = 2
  as_tibble() |>
  setNames("paired_words")
```

Now, let's separate the paired words into two columns using the `separate` function from the `tidyr` package which is loaded as part of the `tidyverse` package. This will allow us to create bigrams from the paired words.

```
word_network_separated <- word_network |>
  separate(paired_words, c("word1", "word2"), sep = " ")
```

We will now count the frequency of each bigram in the text using the `count` function from the `dplyr` package package, which is also party of the `tidyverse`. We will then arrange the result in descending order based on the frequency of the bigrams.

```
# new bigram counts:
word_network_counts <- word_network_separated |>
  count(word1, word2, sort = TRUE)
```

Korean text sometimes is not visible in the graph due to the font issue. This was the case in my Macbook. Let's set the font to one that supports Korean characters. We will use the `extrafont` package to set the font to one that supports Korean characters. We will use the `font_import` function to import the fonts from the system. This may take some time. You only need to do it once. That's why I commented it. You can uncomment it in first usage.

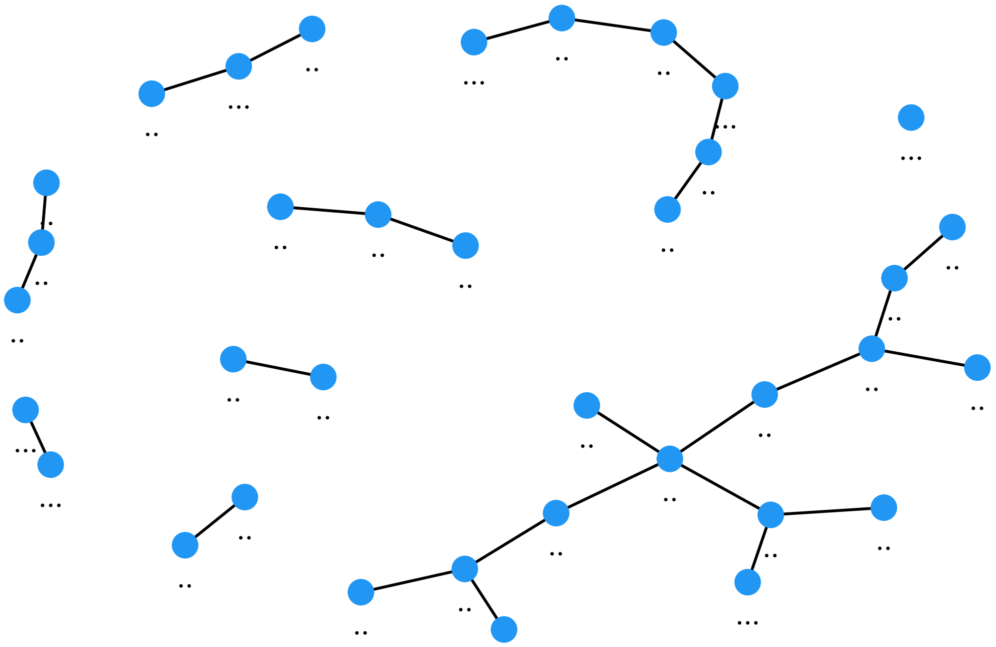
```
# Load extrafont and register fonts

#font_import()  # This might take a while if it's the first time you're running it
```

We will then use the `loadfonts` function to load the fonts. We will use the `fonts` function to display the available fonts and find one that supports Korean characters. We will set the font to one that supports Korean characters. For now, I have chosen "Arial Unicode MS" as the Korean font. You can replace it with a font from your system that supports Korean characters if necessary.

```
#loadfonts(device = "all")

# Display available fonts, find one that supports Korean
#fonts()

# Set the font to one that supports Korean characters
korean_font <- "Arial Unicode MS"  # Replace with a font from your system that supports Korea
```

We will now create a graph from the bigram counts using the `graph_from_data_frame` function from the `igraph` package. We will use the `ggraph` function from the `ggraph` package to create the graph. We will use the `geom_edge_link` function to add the edges to the graph. We will use the `geom_node_point` function to add the nodes to the graph. We will use the `geom_node_text` function to add the labels to the nodes in the graph. We will set the font to the Korean font that we set earlier. We will then adjust the font in the graph. Here, `n >= 6` is used to filter out bigrams that appear less than 6 times. You can adjust this number as needed. You can check out `ggraph` layout options here.

```
word_network_select <- word_network_counts |>
  filter(n >= 6) |>
  graph_from_data_frame() |>
  ggraph(layout = "fr") +
  geom_edge_link(aes()) +
  geom_node_point(color = "#2196f3", size = 4) +
  geom_node_text(aes(label = name), family = korean_font, vjust = 2, size = 4) +  # Set famil
  theme_void()


word_network_select
```



9.5.5 Sentiment Analysis

9.5.6 Topic Modeling

9.6 Korean Tweet Analysis

9.7 Further Readings

9.8 References

9.9 Session Info

```
sessionInfo()
```

```
R version 4.4.0 (2024-04-24)
Platform: aarch64-apple-darwin20
Running under: macOS Sonoma 14.4.1
```

```
Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: Asia/Seoul
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] extrafont_0.19    ggraph_2.2.1      igraph_2.0.3      tidytext_0.4.2
 [5] rvest_1.0.4       pdftools_3.4.0    lubridate_1.9.3   forcats_1.0.0
 [9] stringr_1.5.1     dplyr_1.1.4       purrr_1.0.2       readr_2.1.5
[13] tidyr_1.3.1       tibble_3.2.1      ggplot2_3.5.1     tidyverse_2.0.0
[17] bitNLP_1.4.3.9000

loaded via a namespace (and not attached):
 [1] tidyselect_1.2.1    viridisLite_0.4.2   farver_2.1.1
 [4] viridis_0.6.5       fastmap_1.1.1       tweenr_2.0.3
 [7] janeaustenr_1.0.0   promises_1.3.0      shinyjs_2.1.0
[10] digest_0.6.35       timechange_0.3.0    mime_0.12
[13] lifecycle_1.0.4     qpdf_1.3.3          tokenizers_0.3.0
[16] magrittr_2.0.3      compiler_4.4.0      rlang_1.1.3
[19] sass_0.4.9          tools_4.4.0         utf8_1.2.4
[22] knitr_1.46          labeling_0.4.3      askpass_1.2.0
[25] graphlayouts_1.1.1  htmlwidgets_1.6.4   curl_5.2.1
[28] xml2_1.3.6          miniUI_0.1.1.1      ngram_3.2.3
[31] withr_3.0.0         grid_4.4.0          polyclip_1.10-6
[34] fansi_1.0.6         xtable_1.8-4        colorspace_2.1-0
[37] extrafontdb_1.0     scales_1.3.0        MASS_7.3-60.2
[40] tinytex_0.50        cli_3.6.2           rmarkdown_2.26
[43] generics_0.1.3      RcppParallel_5.1.7  rstudioapi_0.16.0
[46] httr_1.4.7          tzdb_0.4.0          cachem_1.0.8
[49] ggforce_0.4.2       RcppMeCab_0.0.1.2   parallel_4.4.0
[52] rhandsontable_0.3.8 vctrs_0.6.5         Matrix_1.7-0
[55] jsonlite_1.8.8      hms_1.1.3           ggrepel_0.9.5
[58] jquerylib_0.1.4     shinyBS_0.61.1      glue_1.7.0
[61] stringi_1.8.3       gtable_0.3.5        later_1.3.2
[64] munsell_0.5.1       pillar_1.9.0        htmltools_0.5.8.1
```

```
[67] R6_2.5.1            tidygraph_1.3.1     evaluate_0.23
[70] shiny_1.8.1.1       lattice_0.22-6      SnowballC_0.7.1
[73] memoise_2.0.1       DataEditR_0.1.5     httpuv_1.6.15
[76] bslib_0.7.0         Rcpp_1.0.12         Rttf2pt1_1.3.12
[79] gridExtra_2.3       xfun_0.43           pkgconfig_2.0.3
```

10 Statistical Analysis

11 Storytelling with Quarto

12 Productivity Tools

Setting up Github.

Creating a new Github project.

Copilot etc.

13 Working with API to get Korean Data

WDI etc. readily available packages

Creating your own API

https://httr2.r-lib.org/articles/wrapping-apis.html

https://www.andrewheiss.com/blog/2024/01/12/diy-api-plumber-quarto-ojs/_book/

14 Making Korean Data Visualization Social

14.1 #kdiplo #kdiploviz

I love Korea, and I love data.

Combining my enthusiasm for Korean Studies and data, I am initiating an exciting project to make engaging and valuable Korean datasets publicly accessible… in an enjoyable manner!

I invite you to explore and interact with the data I will be sharing. Let's craft stories together using these datasets and connect through the hashtags `#kdiplo`, #kdiploviz, `#kdata`, and #kdataviz.

Recently, I have created several novel datasets on Korean diplomacy for my research[1], mainly focusing on high-level diplomatic visits (both outgoing and incoming), their formats (bilateral, multilateral, informal), nature (such as state visits), purposes (economic, security, etc.), timelines, and the conveners in multilateral contexts among others.

I will make these datasets available via a new R package, `#kdiplo`. Although this is a work in progress, the first version is already shaping up.

The current development version features a pivotal function (along with an accompanying dataset) designed to assist researchers in merging various Korean datasets by country names. Due to inconsistent naming conventions across Korean government datasets (for instance, Thailand might appear as 태국 [Taeguk] or 타이 [Tai]), the kdiplo::iso3c function creates iso3c country codes for Korean country names, simplifying the joining process (similar to countrycode::countrycode).

Next on the agenda is adding comprehensive Korean trade data spanning from 1948 to 2023, inclusive of multiple sources and estimations/ imputations for missing data.

More datasets are on the way, and I am open to data requests.

Stay tuned (follow hashtags #kdiplo, #kdiploviz #kdata, and #kdataviz) for more updates on (https://github.com/kjayhan/kdiplo) - a one-stop public repository for data insights on Korean diplomacy and foreign policy!

For now check this website out, which I will soon update as well.

[1] See these blog posts for now.

14.2 #kdata #kdataviz

While my main interests in Korean Studies lie in foreign policy and (public) diplomacy, I am also interested in everything related to Korea, from business to education to culture.

Indeed, I was trained as an economist, with a double major in international trade, wrote my master's thesis on Korean popular culture (from an international relations angle), and have published at least 8 peer-reviewed articles on international student mobility programs (from a public diplomacy angle).

So... in addition to the `#kdiplo` package, I am happy to announce that, I am also building another package, `#kdata`, dedicated to datasets on Korean business, culture, and education. Although this is a work-in-progress, I have already uploaded multiple datasets to the `#kdiplo` repository. I will upload documentation and vignettes for these datasets soon.

To kick things off with the vibrant Spring season in Korea, I present our first challenge: the Korean Festivals dataset! □□□

Explore and interact with the data available at `#kdiplo` `kdiplo::korean_festivals_data`.

Check out my blog post where I've used this dataset.

I encourage you to dive into this dataset and share your insights. Remember to use hashtags `#kdiplo`, #kdiploviz `#kdata`, and #kdataviz in your posts across various social media platforms!

15 R for Korean Studies Bootcamps

I plan to organize bootcamps to help Korean Studies scholars and students to jumpstart their R learning with Korean Studies-based examples.

You can sign up for my newsletter to get updates on the workshops.

You can find more information about the bootcamps here.

References

Ayhan, Kadir Jun. 2024. R for Korean Studies: A Gentle Introduction to Computational Social Science. Draft Version 0.0.1. https://r4ks.com.