R for Korean Studies

A Gentle Introduction to Computational Social Science

Kadir Jun Ayhan, Ph.D.

2024-05-05

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.

1 Introduction

2 Setting Up

Downloading R, and Rstudio.

3 Korean Studies Data Sources

3.1 Statistical Data

3.2 Text Data

4 Basics of R

5 Data Wrangling

6 Data Visualization: Figures

7 Data Visualization: Plots

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 (　, *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.

```
# install.packages("tidyverse")
# install.packages("pdftools")
# install.packages("rvest")
# install.packages("tidytext")
```

```r
# 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 (2024   ) 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 an

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

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

---

[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 "2024   ".

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

```r
# 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.

```r
# 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.

```r
# 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:

[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

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 29$^{\text{th}}$ and 30$^{\text{th}}$ 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",
```

By observing the `lines_pdf_4` object using `view(lines_pdf_4)`, we can see that the second table starts from the 32$^{\text{nd}}$ line and ends on the 48$^{\text{th}}$. 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.

```
# Select lines 32 to 48 from the lines_pdf_4 data frame
province_city_pd <- lines_pdf_4[32:48] |>
```

```r
    # 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.

```
# 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.

```
# 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.

```
# 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 　(North Jeolla Province) in the MOFA's pdf file which is written as 　(Jeonbuk State) in the Wikipedia table. Let's fix this.

```
# 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", p
```

21

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.

```
# 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

```
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 "   " will become "   ".

```
# 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.

```r
# 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.

```r
# 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".

```
# 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.

```
# 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.

```r
# 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 Implemen
       x = "Morpheme",
       y = "Frequency")
```


Frequency of Morphemes in the 2024 Public Diplomacy Comprehens
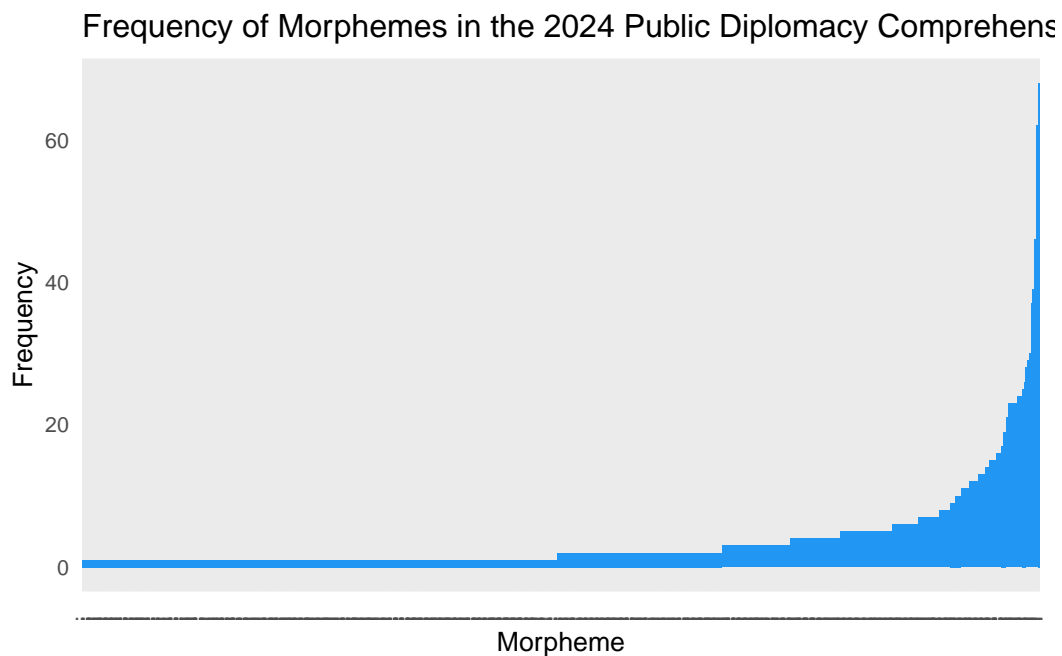
9.5.4 Word Network in Korean Text

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.

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

dic_path <- system.file("dic", package = "bitNLP")
```

27

```r
dic_file <- glue::glue("{dic_path}/buzz_dic.dic")

word_network <- tokenize_noun_ngrams(pdf_text_ko, simplify = TRUE, user_dic = dic_file, n
  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.

```r
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.

```r
# 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.

```r
# 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.

```r
#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 Ko
```

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 fa
  theme_void()


word_network_select
```
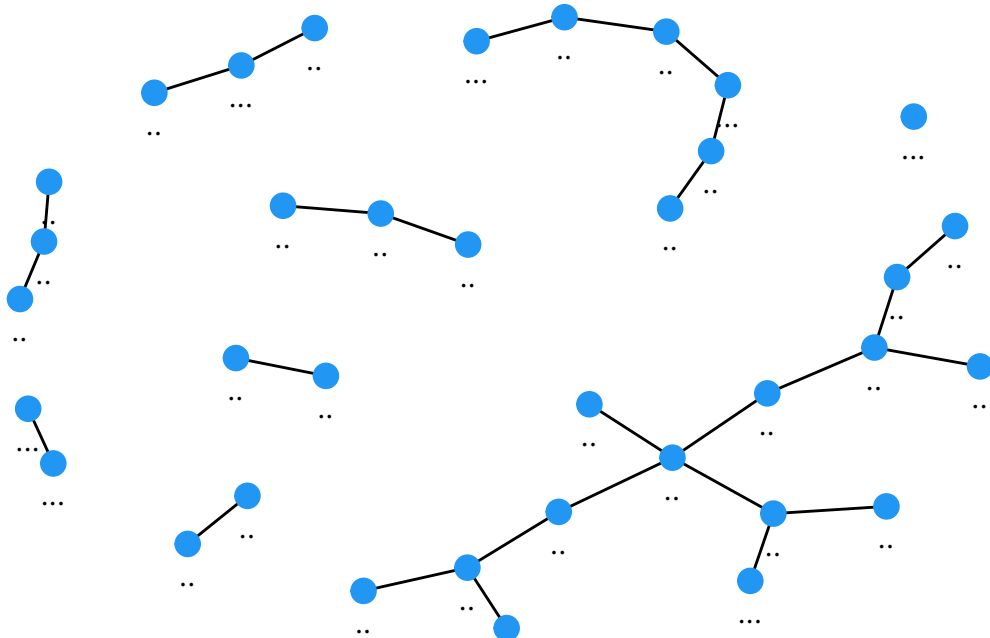
9.5.5 Sentiment Analysis

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 [Taeguk] or [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 1 to 3-days 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.

References