Assignment 3

Due at 11:59pm on October 15.

You may work in pairs or individually for this assignment. Make sure you join a group in Canvas if you are working in pairs. Turn in this assignment as an HTML or PDF file to ELMS. Make sure to include the R Markdown or Quarto file that was used to generate it. Include the GitHub link for the repository containing these files.

```r
library(xml2)
library(rvest)
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter()         masks stats::filter()
x readr::guess_encoding() masks rvest::guess_encoding()
x dplyr::lag()            masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becon
```

```r
library(rvest)
library(jsonlite)
```

```
Attaching package: 'jsonlite'

The following object is masked from 'package:purrr':

    flatten
```

```r
library(robotstxt)
```

```
Warning: package 'robotstxt' was built under R version 4.4.1
```

```r
library(RSocrata)
```

Web Scraping

In this assignment, your task is to scrape some information from Wikipedia. We start with the following page about Grand Boulevard, a Chicago Community Area.

https://en.wikipedia.org/wiki/Grand_Boulevard,_Chicago

The ultimate goal is to gather the table "Historical population" and convert it to a `data.frame`.

As a first step, read in the html page as an R object. Extract the tables from this object (using the `rvest` package) and save the result as a new object. Follow the instructions if there is an error. Use `str()` on this new object – it should be a list. Try to find the position of the "Historical population" in this list since we need it in the next step.

Extract the "Historical population" table from the list and save it as another object. You can use subsetting via `[[…]]` to extract pieces from a list. Print the result.

You will see that the table needs some additional formatting. We only want rows and columns with actual values (I called the table object `pop`).

```r
url_grand_boulevard <- read_html("https://en.wikipedia.org/wiki/Grand_Boulevard,_Chicago")

tables_grand_boulevard <- html_table(url_grand_boulevard, fill = TRUE)
# str(tables_grand_boulevard)

pop <- tables_grand_boulevard[2] %>%
  as.data.frame() %>%
  transmute(
    Census = factor(Census),
    Population = as.numeric(gsub(",", "", Pop.)),
    Percentage_Change = parse_number(gsub("−", "-", X..)) # Handle "−" and convert to nume
  ) %>%
  # Filter out rows with NA or placeholder values more succinctly
  filter(!is.na(Census) & !is.na(Population))
```

```
Warning: There were 2 warnings in `transmute()`.
The first warning was:
i In argument: `Population = as.numeric(gsub(",", "", Pop.))`.
Caused by warning:
! NAs introduced by coercion
i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
head(pop, 6)
```

```
  Census Population Percentage_Change
1   1930      87005                NA
2   1940     103256              18.7
3   1950     114557              10.9
4   1960      80036             -30.1
5   1970      80166               0.2
6   1980      53741             -33.0
```

Expanding to More Pages

That's it for this page. However, we may want to repeat this process for other community areas. The Wikipedia page https://en.wikipedia.org/wiki/Grand_Boulevard,_Chicago has a section on "Places adjacent to Grand Boulevard, Chicago" at the bottom. Can you find the corresponding table in the list of tables that you created earlier? Extract this table as a new object.

Then, grab the community areas east of Grand Boulevard and save them as a character vector. Print the result.

We want to use this list to create a loop that extracts the population tables from the Wikipedia pages of these places. To make this work and build valid urls, we need to replace empty spaces in the character vector with underscores. This can be done with `gsub()`, or by hand. The resulting vector should look like this: "Oakland,_Chicago" "Kenwood,_Chicago" "Hyde_Park,_Chicago"

To prepare the loop, we also want to copy our `pop` table and rename it as `pops`. In the loop, we append this table by adding columns from the other community areas.

```
# pops <- pop

comm_areas <- tables_grand_boulevard[[4]] %>%
  as.data.frame() %>%
  rename(
    West = X1, NorthSouth = X2,  East = X3
  ) %>%
  filter(
    East != "" & NorthSouth != "" & West != "",
    !is.na(East) & !is.na(NorthSouth) & !is.na(West)
  )

east_comms <- comm_areas$East
east_comms
```

```
[1] "Oakland, Chicago"   "Kenwood, Chicago"   "Hyde Park, Chicago"
```

```
east_comms_valid <- gsub(" ", "_", east_comms)
east_comms_valid
```

```
[1] "Oakland,_Chicago"   "Kenwood,_Chicago"   "Hyde_Park,_Chicago"
```

Build a small loop to test whether you can build valid urls using the vector of places and pasting each element of it after `https://en.wikipedia.org/wiki/` in a for loop. Calling `url` shows the last url of this loop, which should be `https://en.wikipedia.org/wiki/Hyde_Park,_Chicago`.

```
base_url <- "https://en.wikipedia.org/wiki/"
urls <- c()

for (area in east_comms_valid) {
  url <- paste0(base_url, area)
  urls <- c(urls, url)


}
url
```

```
[1] "https://en.wikipedia.org/wiki/Hyde_Park,_Chicago"
```

Finally, extend the loop and add the code that is needed to grab the population tables from each page. Add columns to the original table `pops` using `cbind()`.

```r
# pops <- pop %>%
#   mutate(Community = "PPKK")
pops <- pop
# i = 1
# Loop over each URL to extract population tables and bind to pops
for (i in 1:length(urls)) {
  # Access each URL using the index i
  url <- urls[i]

  # Read the HTML from the current URL
  url_data <- read_html(url)

  # Extract the tables from the HTML
  tables <- html_table(url_data, fill = TRUE)

  # Check if there are enough tables to avoid indexing errors
  if (length(tables) >= 2) {
    # Extract the population table and transform it
    new_pop <- tables[2] %>%
      as.data.frame() %>%
      transmute(
        Census = factor(Census),
        Population = as.numeric(gsub(",", "", Pop.)),
        Percentage_Change = parse_number(gsub("-", "-", X..))  # Handle "-" and convert to
        # Community = east_comms[i]
      ) %>%
      # Filter out rows with NA or placeholder values more succinctly
      filter(!is.na(Census) & !is.na(Population))

    # Bind the new population data to the existing pops table
    pops <- rbind(pops, new_pop)
  } else {
    # Print a message if the table is not available
    cat("Population table not found for:", url, "\n")
  }
}
```

```
Warning: There were 2 warnings in `transmute()`.
The first warning was:
i In argument: `Population = as.numeric(gsub(",", "", Pop.))`.
Caused by warning:
! NAs introduced by coercion
```

```
i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
There were 2 warnings in `transmute()`.
The first warning was:
i In argument: `Population = as.numeric(gsub(",", "", Pop.))`.
Caused by warning:
! NAs introduced by coercion
i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
There were 2 warnings in `transmute()`.
The first warning was:
i In argument: `Population = as.numeric(gsub(",", "", Pop.))`.
Caused by warning:
! NAs introduced by coercion
i Run `dplyr::last_dplyr_warnings()` to see the 1 remaining warning.
```

```
# View the final pops table with the added columns
pops
```

```
   Census Population Percentage_Change
1    1930      87005                NA
2    1940     103256              18.7
3    1950     114557              10.9
4    1960      80036             -30.1
5    1970      80166               0.2
6    1980      53741             -33.0
7    1990      35897             -33.2
8    2000      28006             -22.0
9    2010      21929             -21.7
10   2020      24589              12.1
11   1910      13763                NA
12   1920      16540              20.2
13   1930      14962              -9.5
14   1940      14500              -3.1
15   1950      24464              68.7
16   1960      24378              -0.4
17   1970      18291             -25.0
18   1980      16748              -8.4
19   1990       8197             -51.1
20   2000       6110             -25.5
21   2010       5918              -3.1
22   2020       6799              14.9
23   1930      26942                NA
```

```
24   1940      29611               9.9
25   1950      35705              20.6
26   1960      41533              16.3
27   1970      26890             -35.3
28   1980      21974             -18.3
29   1990      18178             -17.3
30   2000      18363               1.0
31   2010      17841              -2.8
32   2020      19116               7.1
33   1930      48017                NA
34   1940      50550               5.3
35   1950      55206               9.2
36   1960      45577             -17.4
37   1970      33531             -26.4
38   1980      31198              -7.0
39   1990      28630              -8.2
40   2000      29920               4.5
41   2010      25681             -14.2
42   2020      29456              14.7
```

Scraping and Analyzing Text Data

Suppose we wanted to take the actual text from the Wikipedia pages instead of just the information in the table. Our goal in this section is to extract the text from the body of the pages, then do some basic text cleaning and analysis.

First, scrape just the text without any of the information in the margins or headers. For example, for "Grand Boulevard", the text should start with, "**Grand Boulevard** on the South Side of Chicago, Illinois, is one of the …". Make sure all of the text is in one block by using something like the code below (I called my object `description`).

```
# Function to extract and clean text from a Wikipedia page
extract_wiki_text <- function(url) {
  # Read the HTML from the specified URL
  page <- read_html(url)

  # Extract the main content (the body text) from the page
  # The main content is usually within the <p> tags
  text_nodes <- page %>%
    html_nodes("p") %>%  # Select all <p> tags
    html_text()          # Extract text from the <p> tags
  # Collapse all text into a single string, separating by a space
  description <- paste(text_nodes, collapse = ' ')

  # Clean up any extra whitespace
  description <- gsub("\\s+", " ", description)  # Replace multiple spaces with a single s

  return(description)
}

# Base URL for Wikipedia
base_url <- "https://en.wikipedia.org/wiki/"

# Example URL for Grand Boulevard
grand_boulevard_url <- paste0(base_url, "Grand_Boulevard,_Chicago")

# Extract the description for Grand Boulevard
description <- extract_wiki_text(grand_boulevard_url)

description <- description %>% paste(collapse = ' ')

# View the cleaned text
cat(description)
```

```
 Grand Boulevard on the South Side of Chicago, Illinois, is one of the city's Community Areas
King College in Englewood. A high school diploma had been earned by 85.5% of Grand Boulevard
```

Using a similar loop as in the last section, grab the descriptions of the various communities areas. Make a tibble with two columns: the name of the location and the text describing the location.

Let's clean the data using `tidytext`. If you have trouble with this section, see the example shown in https://www.tidytextmining.com/tidytext.html

```
library(tidytext)

# Function to extract and clean text from a Wikipedia page
extract_wiki_text <- function(url) {
  page <- read_html(url)
  text_nodes <- page %>%
    html_nodes("p") %>%
    html_text()
  description <- paste(text_nodes, collapse = ' ')
  description <- gsub("\\s+", " ", description)  # Clean up whitespace
  return(description)
}

# Base URL for Wikipedia
base_url <- "https://en.wikipedia.org/wiki/"

# Community areas vector
community_areas <- c("Armour_Square,_Chicago", "Douglas,_Chicago",
                     "Oakland,_Chicago", "Fuller_Park,_Chicago",
                     "Grand_Boulevard,_Chicago", "Kenwood,_Chicago",
                     "New_City,_Chicago", "Washington_Park,_Chicago",
                     "Hyde_Park,_Chicago")

# Initialize an empty tibble to store the results
descriptions_df <- tibble(Location = character(), Description = character())

# Loop over each community area to extract descriptions
for (area in community_areas) {
  # Build the URL for the current community area
  url <- paste0(base_url, area)

  # Extract the description
  description <- extract_wiki_text(url)

  # Append to the tibble
  descriptions_df <- bind_rows(descriptions_df, tibble(Location = area, Description = desc
}

# View the resulting tibble
print(descriptions_df)
```

```
# A tibble: 9 x 2
  Location                 Description
  <chr>                    <chr>
1 Armour_Square,_Chicago   " Armour Square is a Chicago neighborhood on the cit~
2 Douglas,_Chicago         " Douglas, on the South Side of Chicago, Illinois, i~
3 Oakland,_Chicago         "Oakland, located on the South Side of Chicago, Illi~
4 Fuller_Park,_Chicago     "Fuller Park is the 37th of Chicago's 77 community a~
5 Grand_Boulevard,_Chicago " Grand Boulevard on the South Side of Chicago, Illi~
6 Kenwood,_Chicago         " Kenwood, one of Chicago's 77 community areas, is o~
```

```
7 New_City,_Chicago        " New City is one of Chicago's 77 official community~
8 Washington_Park,_Chicago "Washington Park, Chicago may refer to: "
9 Hyde_Park,_Chicago       " Hyde Park is a neighborhood on the South Side of C~
```

Create tokens using `unnest_tokens`. Make sure the data is in one-token-per-row format. Remove any stop words within the data. What are the most common words used overall?

Plot the most common words within each location. What are some of the similarities between the locations? What are some of the differences?

```r
# Create tokens and remove stop words
tidy_descriptions <- descriptions_df %>%
  unnest_tokens(word, Description) %>%     # Tokenize the text into words
  anti_join(stop_words)                     # Remove stop words
```

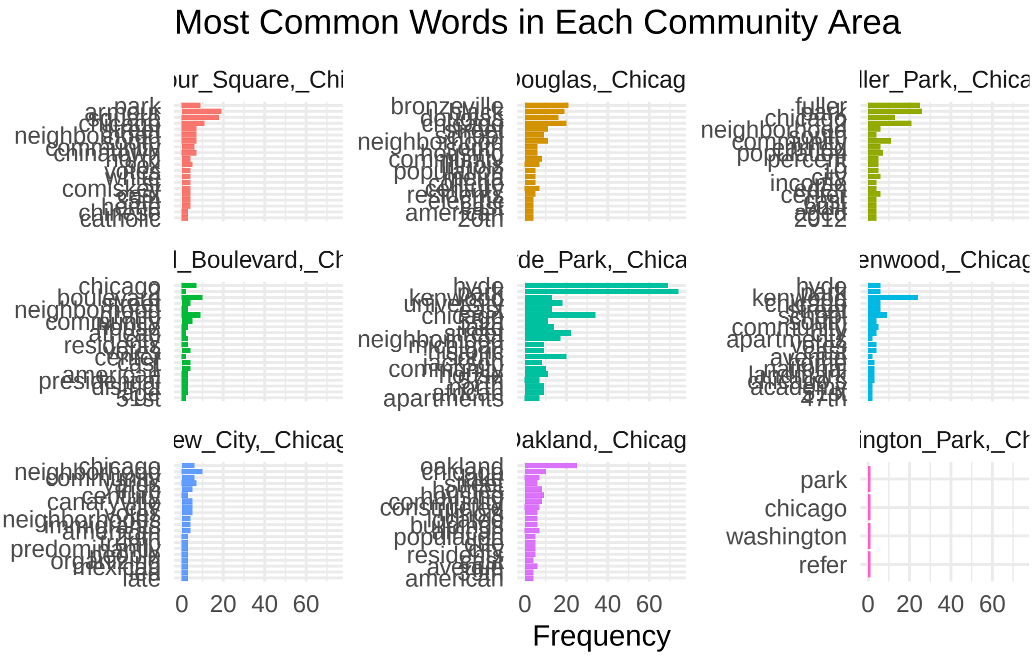
```
Joining with `by = join_by(word)`
```

```r
# View the tidy text data after removing stop words
print(tidy_descriptions)
```

```
# A tibble: 5,005 x 2
   Location                 word
   <chr>                    <chr>
 1 Armour_Square,_Chicago armour
 2 Armour_Square,_Chicago square
 3 Armour_Square,_Chicago chicago
 4 Armour_Square,_Chicago neighborhood
 5 Armour_Square,_Chicago city's
 6 Armour_Square,_Chicago south
 7 Armour_Square,_Chicago larger
 8 Armour_Square,_Chicago officially
 9 Armour_Square,_Chicago defined
10 Armour_Square,_Chicago community
# i 4,995 more rows
```

```r
# Count the frequency of each word per location
location_word_counts <- tidy_descriptions %>%
  count(Location, word, sort = TRUE) %>%
  group_by(Location) %>%
  slice_head(n = 20) %>%  # Get the top 20 words for each location
  ungroup()

# Plot the most common words within each location
ggplot(location_word_counts, aes(x = reorder(word, n), y = n, fill = Location)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ Location, scales = "free_y") +
  labs(x = NULL, y = "Frequency", title = "Most Common Words in Each Community Area") +
  coord_flip() +
  theme_minimal()
```


Most Common Words in Each Community Area
ur_Square,_Chi
park
neighborhood
community
chinatown
catholic
ouglas,_Chicag
bronzeville
neighborhood
population
american
20th
ller_Park,_Chica
fuller
chicago
neighborhood
income
l_Boulevard,_Ch
chicago
boulevard
neighborhood
community
residents
american
31st
de_Park,_Chica
hyde
kenwood
university
chicago
neighborhood
community
apartments
enwood,_Chicag
hyde
kenwood
chicago
community
apartments
academy
47th
ew_City,_Chicag
chicago
neighborhood
community
neighborhoods
american
mexican
late
akland,_Chicag
oakland
chicago
housing
community
population
residents
american
ington_Park,_Ch
park
chicago
washington
refer
Frequency