Source Collection

Text as Data

Amber Boydstun & Mieke Miner

Spring 2025

Contents

Introduction

In today's digital world, so much of the information we need is online—whether it's news articles, social media posts, government reports, or public datasets. But manually copying and pasting data often isn't practical, especially when dealing with large amounts of text. That's where data collection techniques like web scraping and APIs come in. These tools let us automatically gather and structure text data from different sources, making it easier to analyze and uncover insights.

In this module, we'll dive into different ways to collect text data using R. We'll start by learning how to use APIs to access structured data from platforms like news sites and online databases. Then we will explore some web scraping techniques, where we pull information directly from websites.

To put these skills into practice, we'll work with real-world data, including online articles from NYT and BBC, as well as Wikipedia and a government report. Along the way, we'll clean and preprocess the data, getting it ready for text analysis—like sentiment analysis and keyword extraction. By the end of this module, you'll have a solid foundation in data collection methods that will help you automate and scale your research!

In this module, we'll need the following packages:

```r
# Load packages
library(httr)
library(jsonlite)
library(tesseract)
library(readtext)
library(rvest)
library(tidyverse)
library(purrr)
library(lubridate)
library(tidytext)
library(polite)
require(RCurl)
require(XML)
library(wordcloud)
library(syuzhet)
library(textdata)
library(stringr)
library(scales)
```

```
# Set working directory
setwd("~/Library/CloudStorage/Box-Box/A_teaching/POL290G_TextAsData/modules/data/")
#setwd("C:/Users/Windows/Box/POL290G_TextAsData/modules/data/")
```

What is an API? An **Application Programming Interface (API)** allows computers to request data from web services in a structured format. Many websites provide APIs to access their data programmatically.

Example: Using the New York Times Archive API To use the **NYT API**, follow these steps:

1. Sign up at the NYT Developer Network
2. Get an API key

Note: We are using Mieke's API key for this example but you should sign up for your own NYT developer account and create your own API key for your work!!

```
# Enter YOUR api key here
NYTAuth<-"Y92TLQV75VhFxPqKhtjceAGKRhRrFUdb"

# Or pop up to enter api key
#NYTAuth <- rstudioapi::askForPassword("Authorization Key")

# Set the date (year & month) that we want to access data for
year <- "2025"
month <- "1"

# Gives all the information for the pull request
baseurl <- paste0("https://api.nytimes.com/svc/archive/v1/", year, "/", month, ".json?api-key=", NYTAut
```

The query1$response contains a list of two elements: one for the metadata, and one for the docs (i.e., the archive of NYT articles).

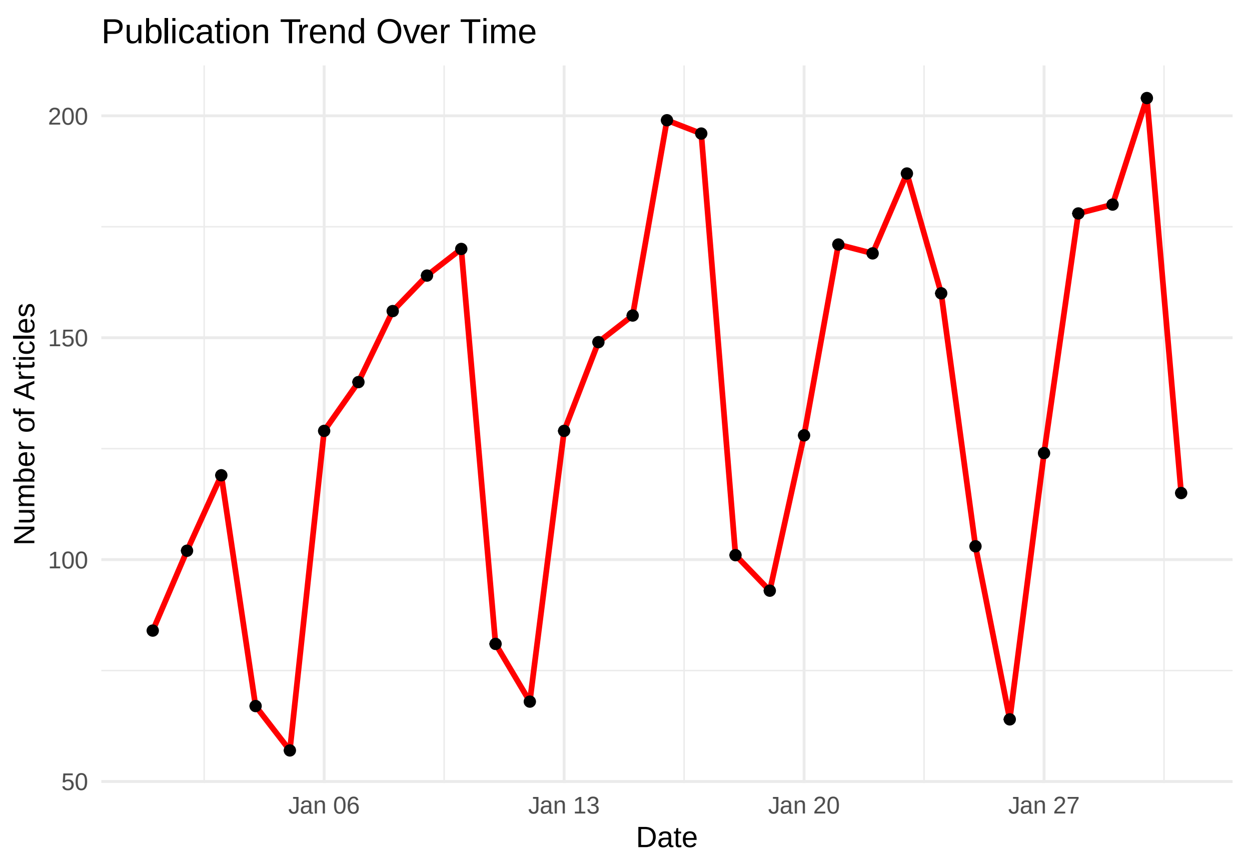
The dataframe contains 20 variables relevant to the archived article—things like the abstract, the headline, the publication date, word count, a snippet and lead paragraph from each article (but not the full body of the text).

```
# Actually pull all the data specified above into a list
query1 <- fromJSON(baseurl)

# Create empty dataframe with all the variable headers we want
df <- data.frame(year = integer(),
                 month = integer(),
                 abstract = character(),
                 web_url = character(),
                 snippet = character(),
                 lead_paragraph = character(),
                 print_section = character(),
                 print_page = character(),
                 source = character(),
                 pub_date = character(),
                 headline = character())

# Pull the information from inside our query
df_query <- query1$response$docs

# Turn this information into a dataframe
```

```r
df_temp <- data.frame(year = year,
                 month = month,
                 abstract = df_query$abstract,
                 web_url = df_query$web_url,
                 snippet = df_query$snippet,
                 lead_paragraph = df_query$lead_paragraph,
                 print_section = df_query$print_section,
                 print_page = df_query$print_page,
                 source = df_query$source,
                 pub_date = df_query$pub_date,
                 headline = df_query$headline)

# Link our queried dataframe with our pretty variable headings
df <- rbind(df, df_temp)

# Convert publication date to Date format
df$pub_date <- as.Date(df$pub_date)
```

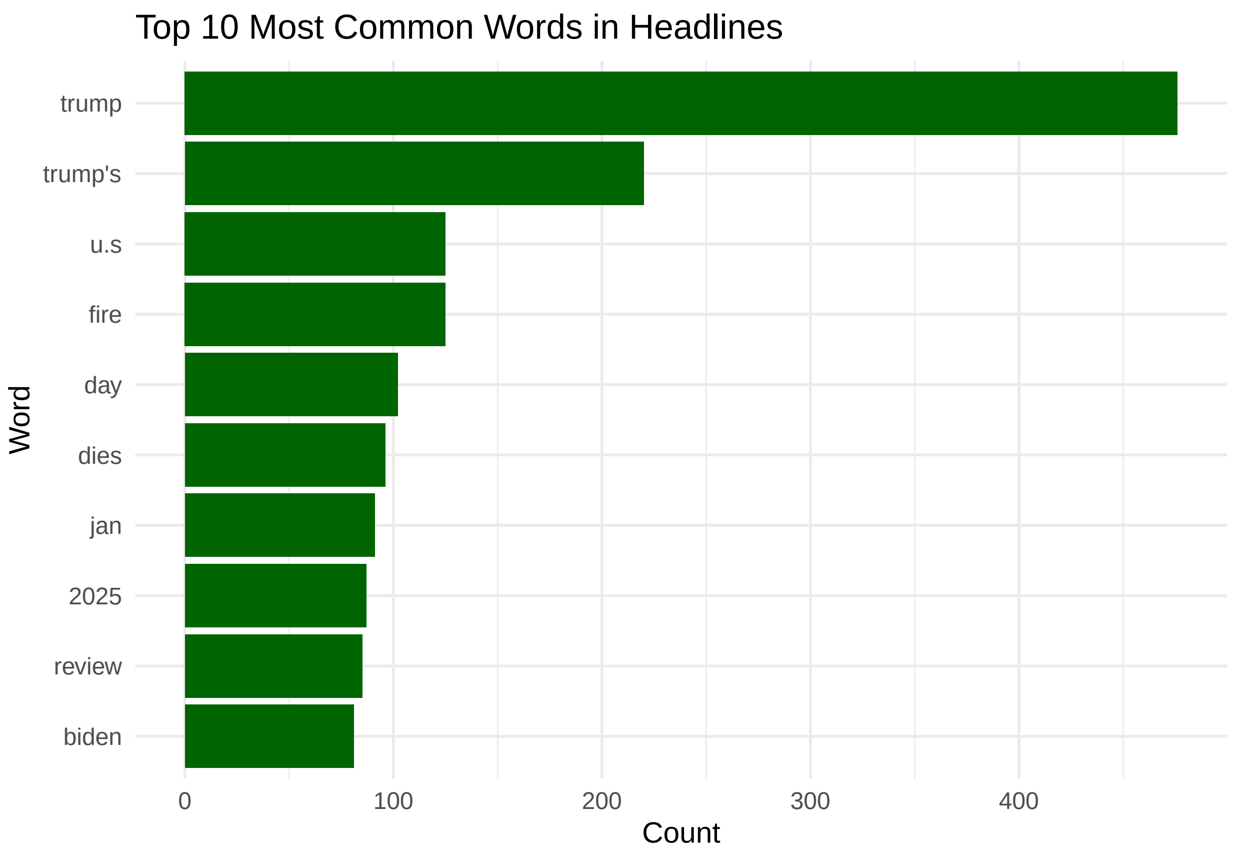
Let's explore our data!

```r
# Count articles per day
daily_counts <- df %>%
    group_by(pub_date) %>%
    summarise(count = n())

# Plot articles over time
ggplot(daily_counts, aes(x = pub_date, y = count)) +
    geom_line(color = "red", linewidth = 1) +
    geom_point(color = "black") +
    labs(title = "Publication Trend Over Time", x = "Date", y = "Number of Articles") +
    theme_minimal()
```



```r
# Most common words in headlines
word_counts <- df %>%
    unnest_tokens(word, headline.main) %>%
    anti_join(stop_words) %>%
    count(word, sort = TRUE)
```

```r
# Plot top 10 most common words
ggplot(head(word_counts, 10), aes(x = reorder(word, n), y = n)) +
    geom_bar(stat = "identity", fill = "darkgreen") +
    coord_flip() +
    labs(title = "Top 10 Most Common Words in Headlines", x = "Word", y = "Count") +
    theme_minimal()
```



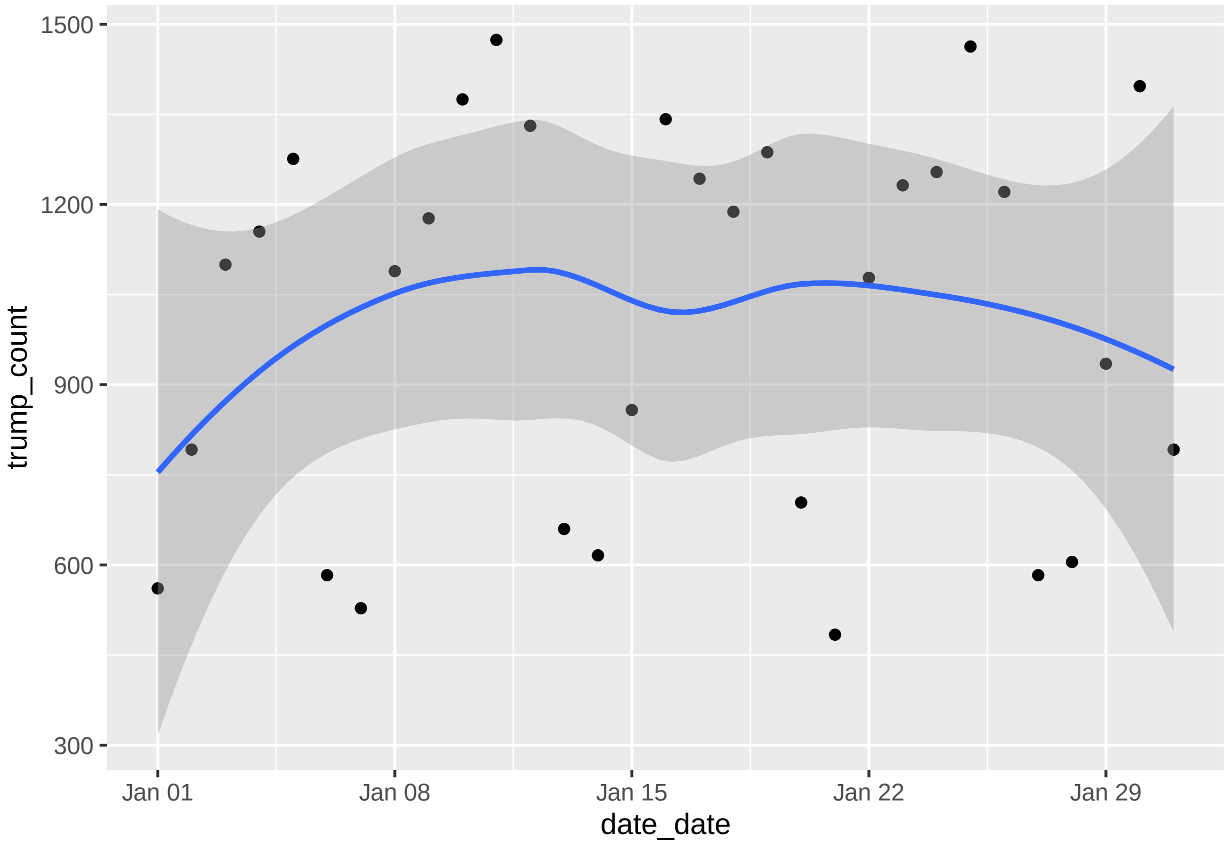
Example using the NYT Article Search API We can use the NYT API data to look at coverage of Donald Trump during the 2024 election.

```
NYTAuth<-"Y92TLQV75VhFxPqKhtjceAGKRhRrFUdb"


term <- "Trump"
begin_date <- "20241101"
end_date <- "20241109"

baseurl <- paste0("http://api.nytimes.com/svc/archive/v1/2024/1.json?q=",term,
                  "&begin_date=",begin_date,"&end_date=",end_date,
                  "&facet_filter=true&api-key=",NYTAuth,sep="")

initialQuery <- fromJSON(baseurl)
maxPages <- round((initialQuery$response$meta$hits[1] / 10)-1)
maxPages = ifelse(maxPages >= 10, 10, maxPages)

pages <- vector("list",length=maxPages)

for(i in 0:maxPages){
    nytSearch <- fromJSON(paste0(baseurl, "&page=", i), flatten = TRUE) %>% data.frame()
    pages[[i+1]] <- nytSearch
    Sys.sleep(10)
}

# Pull together all the results from the loop above
```

```
trump_articles <- rbind_pages(pages)

# Only keep news items (not videos, op-ed, etc)
trump_articles<-trump_articles%>%filter(response.docs.type_of_material=="News")
```

Let's plot Trump mentions over time (by day).

```
trump_articles<-trump_articles%>%mutate(date_string = str_extract(response.docs.pub_date, "[0-9]+\\-[0-9

trump_daily<-trump_articles%>%group_by(date_date)%>%summarise(trump_count=n())


trump_daily %>% ggplot(aes(x= date_date, y = trump_count)) +
                geom_point() +
                geom_smooth(method = 'loess', formula = 'y ~ x')
```



We can also do some initial text analysis of the data.

```
text_data <- trump_articles %>%
  unite("text", response.docs.headline.main, response.docs.abstract,response.docs.lead_paragraph, sep =

text_data <- tibble(text = text_data$text)

tidy_text <- text_data %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)
```

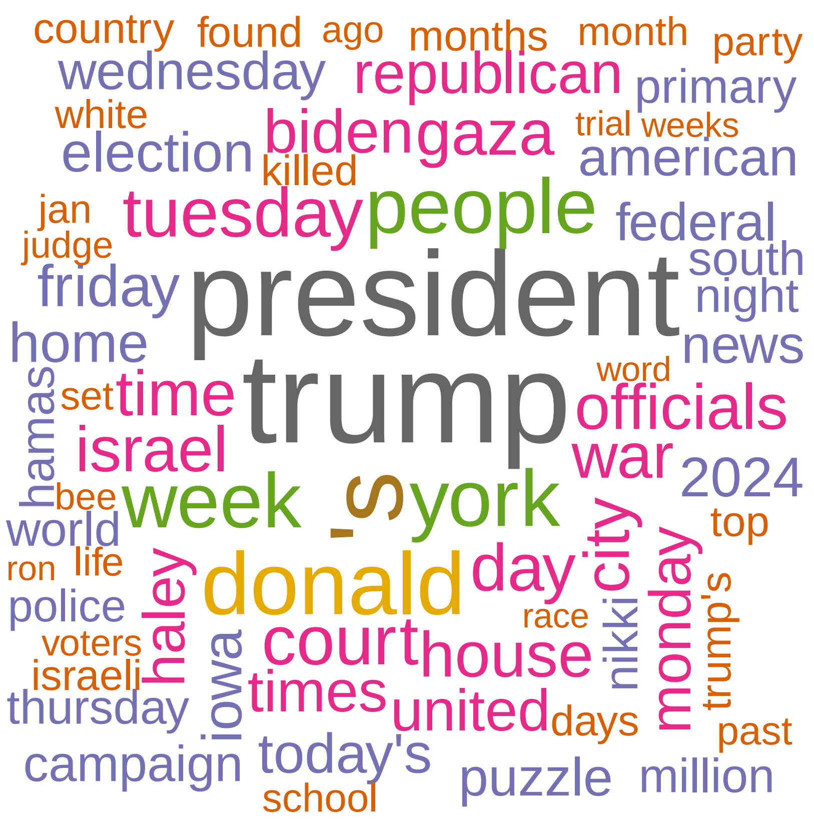
Find Common Words Associated with "Trump"

```r
# Count word co-occurrences
word_counts <- tidy_text %>%
  count(word, sort = TRUE)

# Display most common words
head(word_counts, 20)
```

```
## # A tibble: 20 x 2
##    word          n
##    <chr>     <int>
##  1 trump      5852
##  2 president  5478
##  3 donald     3795
##  4 york       3311
##  5 u.s        3289
##  6 people     3223
##  7 week       3135
##  8 court      2849
##  9 tuesday    2772
## 10 day        2607
## 11 house      2552
## 12 time       2552
## 13 israel     2519
## 14 war        2497
## 15 city       2464
## 16 gaza       2453
## 17 officials  2453
## 18 biden      2431
## 19 haley      2288
## 20 times      2288
```

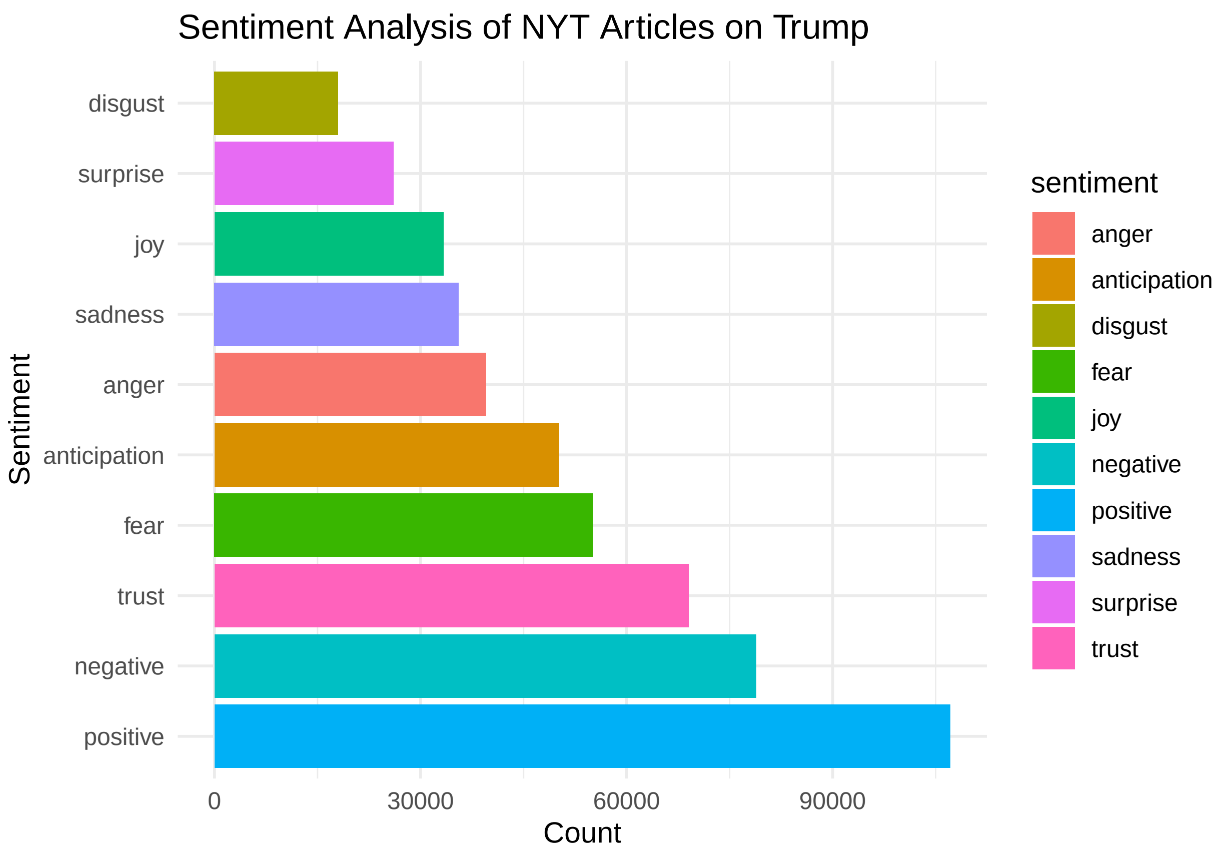
Create word cloud

```r
wordcloud(tidy_text$word, max.words = 100, random.order = FALSE, colors = brewer.pal(8, "Dark2"))
```



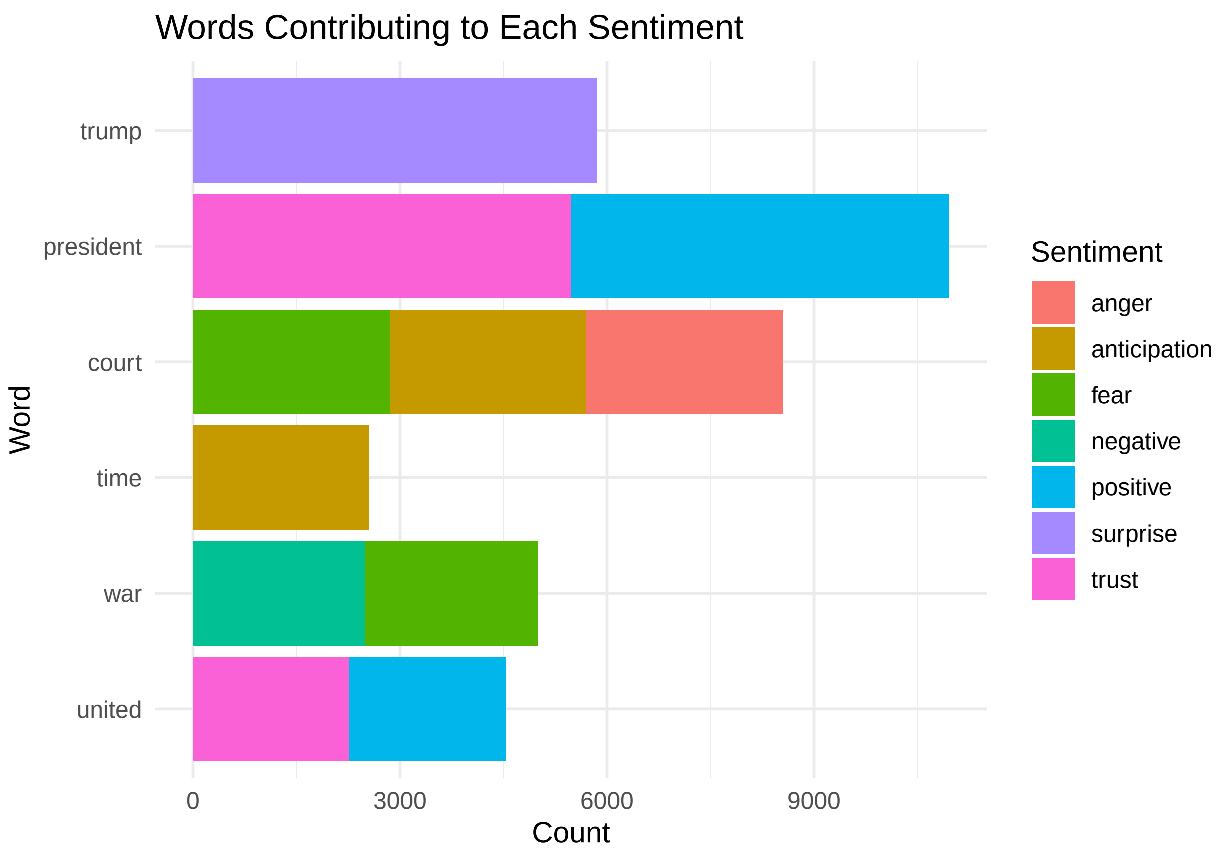
Calculate and plot sentiment analysis

```r
text_data$text <- iconv(text_data$text, from = "latin1", to = "UTF-8", sub = "")

# Get sentiment scores
sentiment_scores <- get_nrc_sentiment(text_data$text)

# Aggregate scores
sentiment_summary <- sentiment_scores %>%
  summarise_all(sum) %>%
  pivot_longer(cols = everything(), names_to = "sentiment", values_to = "count")

# Plot sentiment scores
ggplot(sentiment_summary, aes(x = reorder(sentiment, -count), y = count, fill = sentiment)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Sentiment Analysis of NYT Articles on Trump", x = "Sentiment", y = "Count")
```



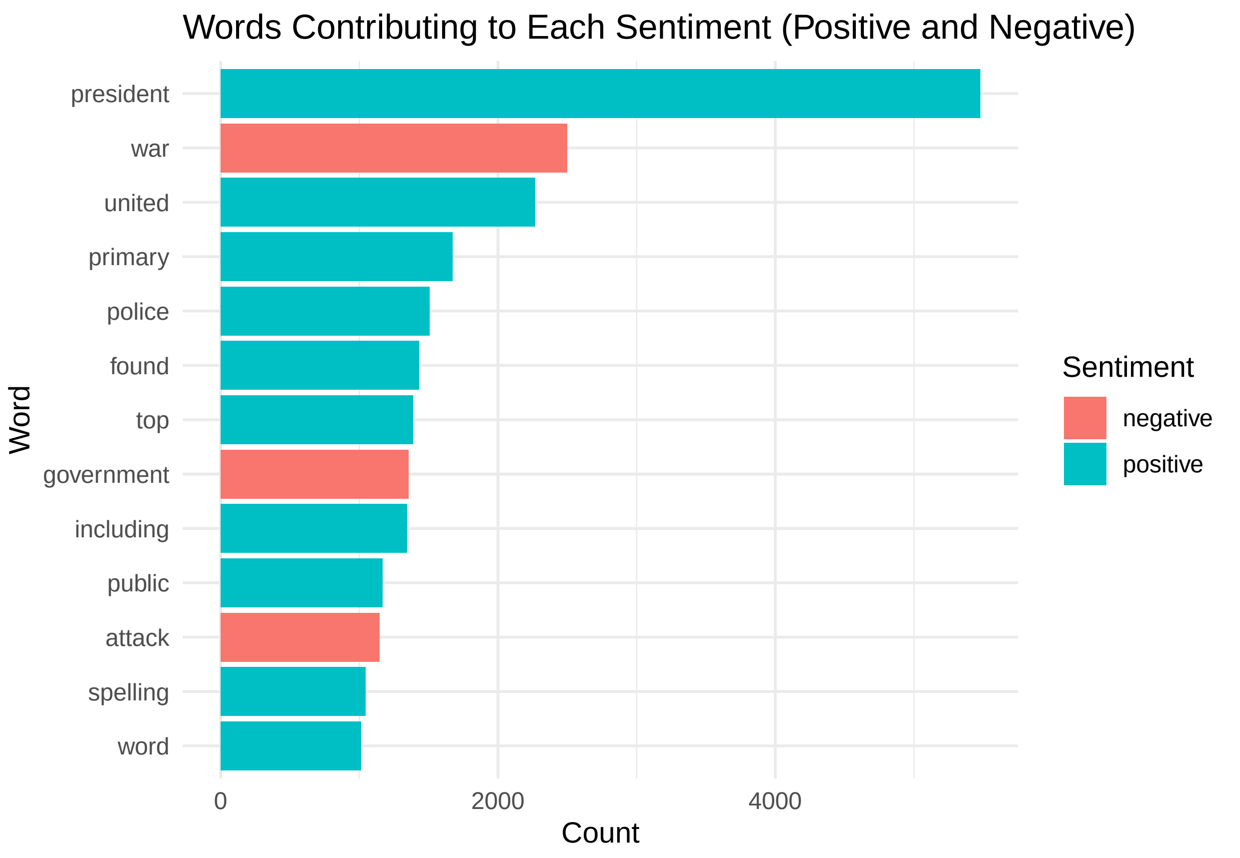
Let's look at the words contributing to each sentiment **Note:** this figure highlights some issues with using canned sentiment analysis: President **Trump** versus the verb to **trump**

```r
nrc_lexicon <- textdata::lexicon_nrc()

# Load NRC sentiment lexicon
nrc_lexicon <- get_sentiments("nrc")

# Join words with NRC sentiment categories
word_sentiments <- tidy_text %>%
  inner_join(nrc_lexicon, by = "word") %>%
  count(word, sentiment, sort = TRUE)

# Plot contributions of words to sentiment
emotions<-word_sentiments%>%filter(n>2000)

ggplot(emotions, aes(x = reorder(word, n), y = n, fill = sentiment)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Words Contributing to Each Sentiment",
       x = "Word",
       y = "Count",
       fill = "Sentiment")
```



Let's specifically look at the main words contributing to the negative and positive sentiments

```
posneg<-word_sentiments%>%filter(sentiment=="positive"|sentiment=="negative")
posneg<-posneg%>%filter(n>1000)

ggplot(posneg, aes(x = reorder(word, n), y = n, fill = sentiment)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Words Contributing to Each Sentiment (Positive and Negative)",
       x = "Word",
       y = "Count",
       fill = "Sentiment")
```



What is Web Scraping? Web scraping is the process of automatically extracting data from web pages.

Best Practices:

- Always check the website's `robots.txt` file for permissions.
- Use APIs when available.
- Scrape responsibly by limiting the frequency of requests.

For example, we can scrape the news page from the BBC.

```r
url <- "https://www.bbc.com/news/articles/cd7e38py4geo"
txt = getURL(url)

PARSED <- htmlParse(txt)
#title - h1 tag
xpathSApply(PARSED, "//h1", xmlValue)
```

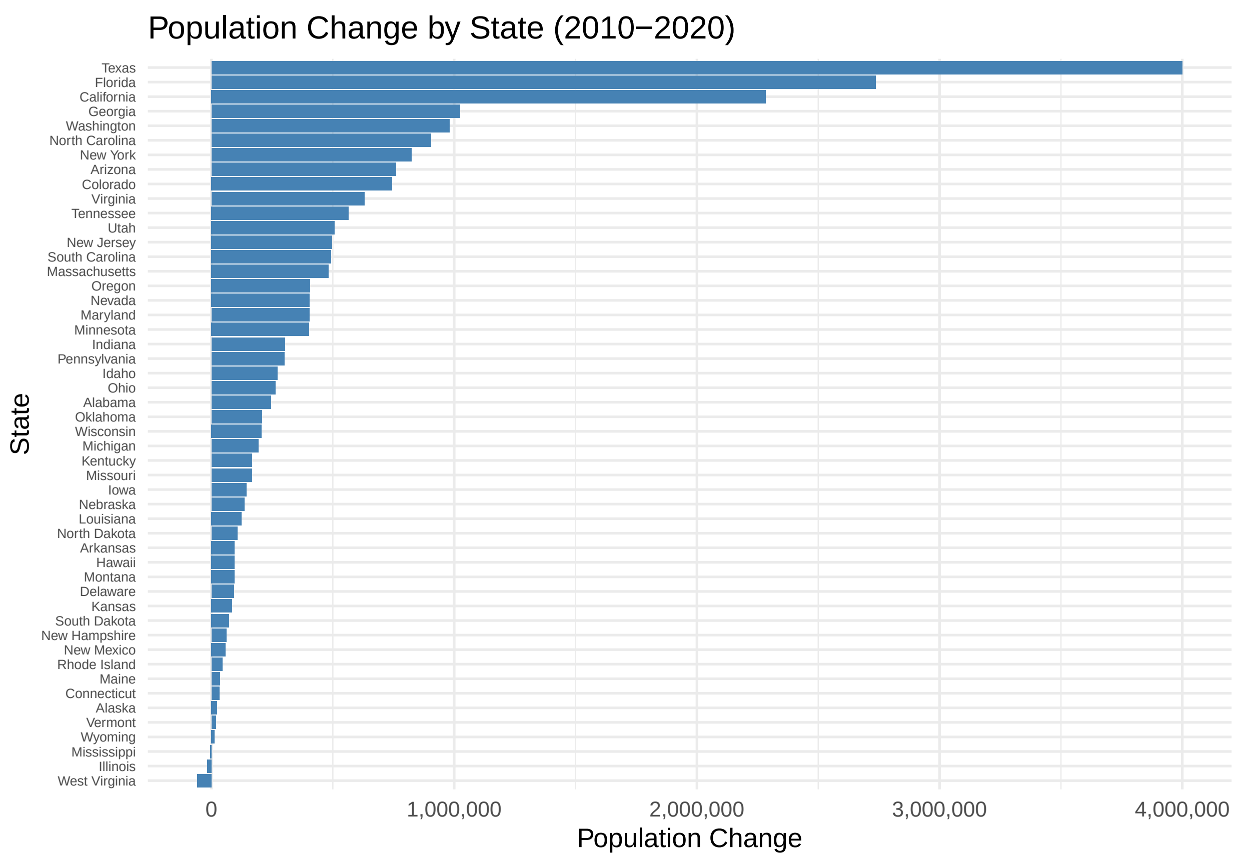
```
## [1] "Scores of whales to be euthanised after mass stranding in Australia"
```

```r
#body of article - p1 tag
xpathSApply(PARSED, "//p", xmlValue)
```

```
##  [1] "Australian authorities are euthanising about 90 false killer whales which survived a mass stran
##  [2] "A team of experts at the site said complex conditions have made it impossible to save them."
##  [3] "They are part of a pod of 157 whales that had beached near Arthur River, in the island's north
##  [4] "Tasmania has seen a series of mass whale strandings in recent years - including the country's w
##  [5] "False killer whales are technically one of world's largest dolphin species, like their orca nam
##  [6] "Authorities on Wednesday said the pod had been stranded at the site for 24 to 48 hours, and the
##  [7] "Local resident Jocelyn Flint told the Australian Broadcasting Corporation she had travelled to
##  [8] "\"There are babies... There's just families of them. Their eyes are open, they're looking at me
##  [9] "\"It's just absolutely horrific.\""
## [10] "The site - about 300km (186 miles) from the city of Launceston - is extremely difficult to acce
## [11] "\"This is possibly the trickiest location I've seen in 16 years of doing this role in Tasmania
## [12] "\"We're talking a very rough, steep, single lane road into the site. We can get four-wheel driv
## [13] "Rough conditions meant returning the animals to the sea at the location they stranded was impos
## [14] "\"The animals just can't get past the break to get out. They just keep turning around and comin
## [15] "With conditions for the next two days forecast to be similar, expert wildlife veterinarians mad
## [16] "\"The longer these animals are out stranded, the longer they are suffering. All alternative opt
## [17] "That grim task - which involves shooting the animals -  is expected to begin on Wednesday but c
## [18] "Authorities are still working out how to dispose of the carcasses. The site has important cultu
## [19] "Authorities have asked members of the public to avoid the site, with bushfires burning nearby a
## [20] "More than 80% of Australian whale strandings take place in Tasmania - often on its west coast.
## [21] "Around 470 pilot whales were stranded further south at Macquarie Harbour in 2020 and about 350
## [22] "Whales are highly social mammals and are well known for stranding in groups because they travel
## [23] "There are a range of theories for why beachings occur. Some experts say the animals can become
## [24] "Others believe that one individual can mistakenly lead whole groups to shore."
## [25] "Sign up for our Future Earth newsletter to get exclusive insight on the latest climate and envi
## [26] "The mistake at a Queensland fertility clinic has been blamed on human error."
## [27] "Martin French says about 100 butterfly chrysalises are brought to the house each week."
## [28] "Prime Minister Anthony Albanese said Australians would \"speak for ourselves\" on global trade
## [29] "Police are warning people to be responsible as whale sightings in the South West rise."
## [30] "The structure, which was funded by the Storm CiarÃ¡n recovery fund, aims to promote biodiversit
## [31] "Copyright 2025 BBC. All rights reserved.Â Â The BBC is not responsible for the content of exter
## [32] "Â "
```

Another example: We can scrape tables from Wikipedia.

```r
us_states <- read_html("https://en.wikipedia.org/wiki/List_of_U.S._states_and_territories_by_population
  #extract the first node with class of wikitable
  html_node(".wikitable") %>%
  #convert the HTML table into a data frame
  html_table(fill = TRUE)

# Rename columns
names(us_states)<- c('state', 'pop_2024', 'pop_2020', 'change_2010_2020_percent','change_2010_2020','hou

# Remove first row
us_states<-us_states[-1,]

#pull out only states
us_states$state_only <- ifelse(grepl("\\*", us_states$house_seats), 0, 1)

us_states<-us_states%>%filter(state_only==1)
us_states<-us_states%>%filter(state!="The 50 states")

# Clean and convert data
us_states_clean <- us_states %>%
  mutate(
    change_percent = change_2010_2020_percent %>%
      str_replace("\u2212", "-") %>%
      str_remove("%") %>%
      as.numeric(),
        change_absolute = change_2010_2020 %>%
      str_replace("\u2212", "-") %>%
      str_remove_all(",") %>%
      as.numeric()
  )

#plot population change by state
ggplot(us_states_clean, aes(x = reorder(state, change_absolute), y = change_absolute)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  scale_y_continuous(labels = label_comma()) +
  labs(
    title = "Population Change by State (2010-2020)",
    x = "State",
    y = "Population Change") +
  theme_minimal(base_size = 10) +
  theme(
    axis.text.y = element_text(size = 5)
  )
```



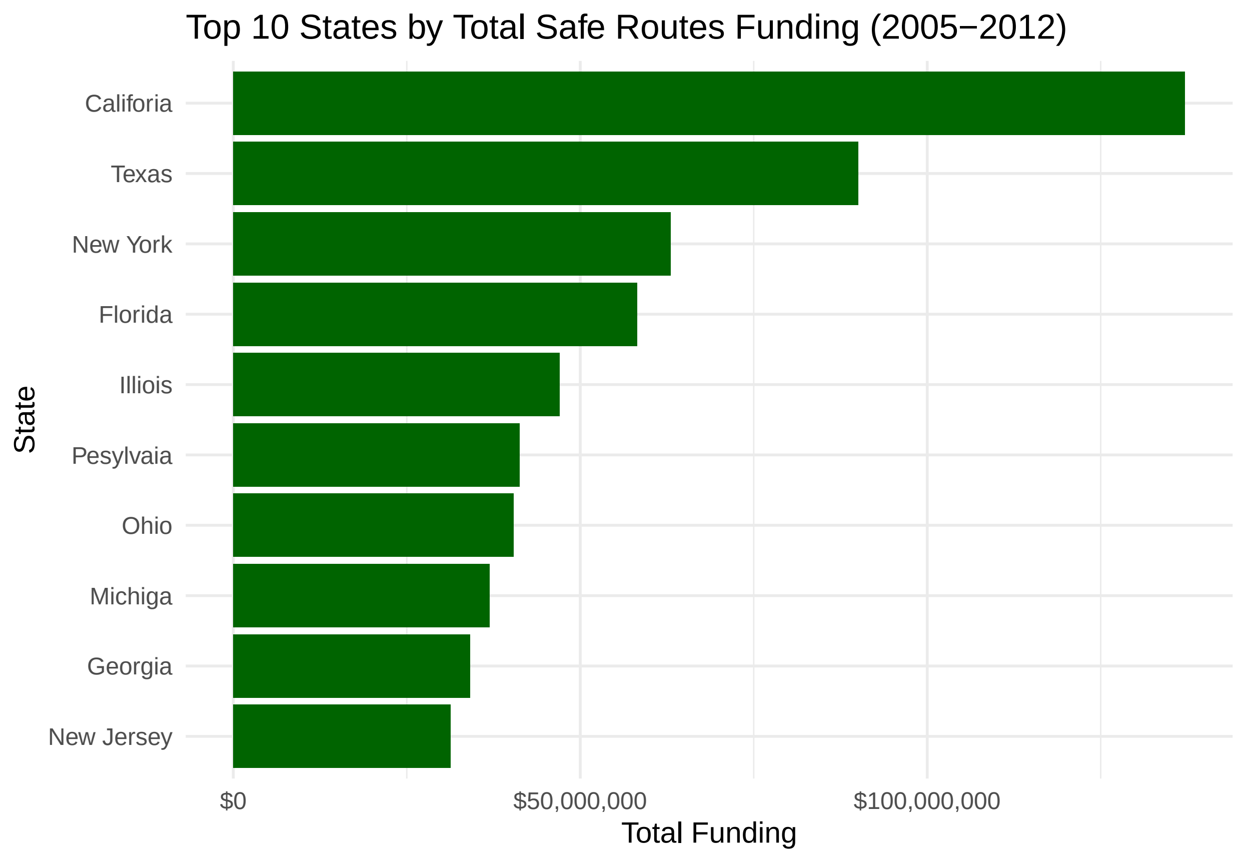
We can scrape other online tables, too.

```
url <- "http://apps.saferoutesinfo.org/legislation_funding/state_apportionment.cfm"

# Get the data
funding<-htmlParse(url)

# Find the table on the page and read it into a list object
funding<- readHTMLTable(funding,stringsAsFactors = FALSE)
```

```r
# Flatten data
funding.df <- do.call("rbind", funding)

# Fix column names
colnames(funding.df) <- colnames(funding.df) %>%
  str_replace_all("[\\n\\t]+", "") %>%
  str_trim() %>%
  str_replace_all("Actual ", "Actual_") %>%
  str_replace_all(" ", "_")

# Clean currency and convert to numeric
funding_clean <- funding.df %>%
  mutate(across(everything(), ~str_trim(gsub("[\\n\\t]+", "", .)))) %>%
  mutate(across(
    starts_with("Actual_") | starts_with("Total"),
    ~as.numeric(gsub("[$,]", "", .))
  ))

#remove Total
funding_clean<-funding_clean%>%filter(State!="Toal")


# Pivot for plotting by year
funding_long <- funding_clean %>%
  pivot_longer(cols = starts_with("Actual_"),
               names_to = "Year",
               names_prefix = "Actual_",
               values_to = "Funding")

# Convert year to numeric
funding_long$Year <- as.numeric(funding_long$Year)
```

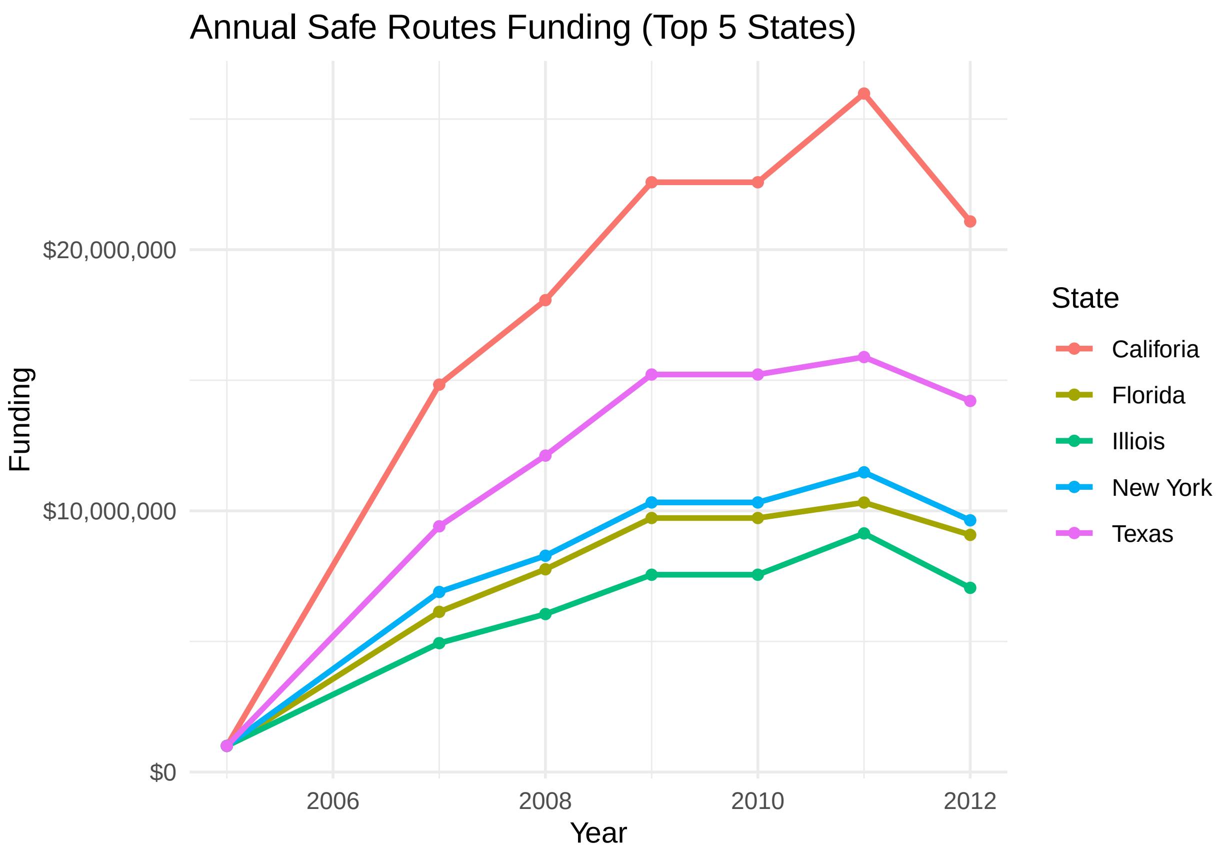
Plot top 10 states by total funding

```r
funding_clean %>%
  arrange(desc(Total)) %>%
  slice_head(n = 10) %>%
  ggplot(aes(x = reorder(State, Total), y = Total)) +
  geom_col(fill = "darkgreen") +
  coord_flip() +
  scale_y_continuous(labels = scales::label_dollar()) +
  labs(
    title = "Top 10 States by Total Safe Routes Funding (2005-2012)",
    x = "State",
    y = "Total Funding"
  ) +
  theme_minimal()
```



Plot funding over time for top 5 recipient states

```r
top_states <- funding_clean %>%
  top_n(5, Total) %>%
  pull(State)

funding_long %>%
  filter(State %in% top_states) %>%
  ggplot(aes(x = Year, y = Funding, color = State)) +
  geom_line(size = 1) +
  geom_point() +
  scale_y_continuous(labels = scales::label_dollar()) +
  labs(
    title = "Annual Safe Routes Funding (Top 5 States)",
    x = "Year",
    y = "Funding",
    color = "State"
  ) +
  theme_minimal()
```



Homework

Discussion Question:

1. Why is understanding the data generation process (DGP) crucial when working with text data? How can overlooking aspects of DGP lead to biased or misleading analysis outcomes? (You'll probably need a few paragraphs for this response.)

Think about things like:

- **Source bias and selection bias**

For example, the data you get from news sources carries the same biases present in news coverage like political slants, editorial policies, audience preferences, etc. Another example, if you are using social media data - you have to think about who (what population/segment of the population) is producing that data. Are Twitter users representative of the general population?

- **Contextual differences in language**

The use of language can differ by platform (think language used in news vs congressional speeches vs Twitter). Language can also evolve over time, making it difficult to correctly capture things like slang, sarcasm, or context-dependent terms (case in point is the word trump which in the above off-the-shelf dictionary approach to our sentiment analysis on news coverage of Trump treated the word trump as the verb rather than the Donald.

- **Missing data and ethics in data access**

If we can only conduct our research using the available data then what data we can actually access can have big implications for our findings and how correct/generalizable they are. Restrictions/limitations on data availability (eg API paywalls, platform controls over access) can distort our findings. Also if certain data is censored/removed by moderation algorithms, then this can create an artificial representation of actual discourse. Example: Twitter's API restriction imposed in 2023 had big consequences for researchers studying things like public opinion, misinformation, hate speech, etc. Moreover, the pricetag on access to the Twitter API is HUGE which has ethical concerns like reinforcing/exacerbating inequalities in knowledge production, restricting reproducibility in research, etc, etc.

Project Questions (show your work): Apply this approach to your own research/interests!

1. Find an API (other than NYT) that has data your are interested in.
2. Write R code to access the API, using one search term (e.g., "immigration").
3. Do some basic data exploration. Depending on your data, this could be things like word frequencies, temporal trends, etc.
4. Write a summary of (1) the data source you chose, (2) what you learned from your basic data exploration, and (3) any challenges you faced along the way - like finding/accessing a relevant API, getting permissions to use the API (RIP Twitter), and any issues with processing your text data.
5. Repeat steps 1-3 but this time with additional search terms (e.g., "immigration", "immigrant", "migrant").
6. Finally, discuss: what worked and what didn't, and why? Explain the similarities and differences between your two search term approaches and what you learned.

Optional: Do the same steps above but with webscraping rather than an API, using an online data source with interesting data.