Chapter 2 Online Text as Data

In this section, we'll start to use the information around us – the books we might read, the news that we consume, the tweets and webpages that we view – as data.

2.1 Working with dataframes in R

Before jumping into online text analyses, it is necessary that we understand a bit more about how to work with data in R. Most of the time, we'll be working with *dataframes*, which are $m \cdot n$ objects with m rows (typically observations) and n columns (typically variables). We'll start by creating a simple dataframe. We'll call our variables "year" and "temp," although of course these are just fake temperatures.

```
# create a dataframe (note that : returns a sequence between the numbers, by 1)
df <- data.frame(year = 2000:2020, temp = 40:60)

# look at the first 6 rows
head(df)
```

```
##   year temp
## 1 2000   40
## 2 2001   41
## 3 2002   42
## 4 2003   43
## 5 2004   44
## 6 2005   45
```

Great! Let's say, however, that our temperatures are in Celsius, and we want them in Fahrenheit. We can write a function to do this, and implement it in our dataframe.

```r
library(dplyr)

# function to get from C to F
c_to_f <- function(c){
  f <- 9/5*c+32
  return(f)
}

# run function on temperature variable to create new variable
df %>%
  mutate(temp_f = c_to_f(temp)) %>%
  head()
```

```
##   year temp temp_f
## 1 2000   40  104.0
## 2 2001   41  105.8
## 3 2002   42  107.6
## 4 2003   43  109.4
## 5 2004   44  111.2
## 6 2005   45  113.0
```

There's a lot going in there. First, `mutate()` is how one would normally create or modify variables within the `dplyr` framework in R. This is generally what we will use to do so.

Second, **what was going on with the %>%?** The code that we just ran is equivalent to `head(mutate(temp_f = c_to_f(temp)))` . You can try running them both yourself. So why make it more complicated? For this small example, it doesn't really matter. But, as we run more and more complicated code, we will probably want to run multiple functions on a dataframe (or vector, or something else) at once. It can get confusing to nest these within the function commands. For example:

```r
fourth_function(third_function(second_function(first_function(x))))
```

So instead, we opt for the `dplyr` method, which is written as follows:

```
x %>%
  first_function() %>%
  second_function() %>%
  third_function() %>%
  fourth_function()
```

This is usually a bit easier for the reader to understand. That being said, dplyr is specific to R, so if you use other languages it may be important to learn other standards of communicating code. As always, remember that our code is language, and that there are multiple ways to communicate most statements, but that we want to make our code as easily interpretable as possible.

Why `%>%` ? This is what is called a **pipe**. It pipes whatever is before the `%>%` into the function that follows `%>%` . This concept is vital to our coding, so let's make sure that we understand how this is working.

```
# first, let's write a simple function
add_2 <- function(x){
  return(x+2)
}

# check that it is working
add_2(5)
```

```
## [1] 7
```

```
# now try the pipe method
5 %>% add_2()
```

```
## [1] 7
```

```
# try adding 2, twice
5 %>%
  add_2() %>%
  add_2()
```

```
## [1] 9
```

Most of the time, we will be applying functions to vectors and dataframes, rather than individual numbers, as we did in the first example. So let's go take another look at our dataframe.

```
# take a look at the first 6 rows, again
head(df)
```

```
##   year temp
## 1 2000   40
## 2 2001   41
## 3 2002   42
## 4 2003   43
## 5 2004   44
## 6 2005   45
```

Oh no! The column we created, temp_f, has disappeared … what happened? When we ran the earlier function, we *temporarily* created a new variable within df, but we did not permanently change df. So how would we permanently change df? There are a couple ways.

```
library(magrittr)

# first, we can assign the mutated dataframe to itself
df <- df %>%
  mutate(temp_f = c_to_f(temp))

# second, we can use the magrittr pipe to do it all at once
df %<>%
  mutate(temp_f = c_to_f(temp))

# check that either of these methods worked (we didn't really need to run them both)
head(df)
```

```
##   year temp temp_f
## 1 2000   40  104.0
## 2 2001   41  105.8
## 3 2002   42  107.6
## 4 2003   43  109.4
## 5 2004   44  111.2
## 6 2005   45  113.0
```

Notice the **magrittr pipe**, `%<>%` . This is generally what I will use when performing operations on a dataframe, and I recommend that you do so as well! As a side note, there is another pipe, `|>` , which is another version of `%>%` , and is mostly functionally equivalent. However, sometimes the magrittr pipe only works with `%>%` ! For this reason, we will mostly use `%>%` and `%<>%` in this class, but you are free to try other things in your code. New versions of these operations are common, so there may be other ways to do this that I am unaware of! The goal is to stay flexible and understand the rationale behind these commands. I recommend trying all these out now to make sure you are comfortable with them.

2.2 Newspapers & Books

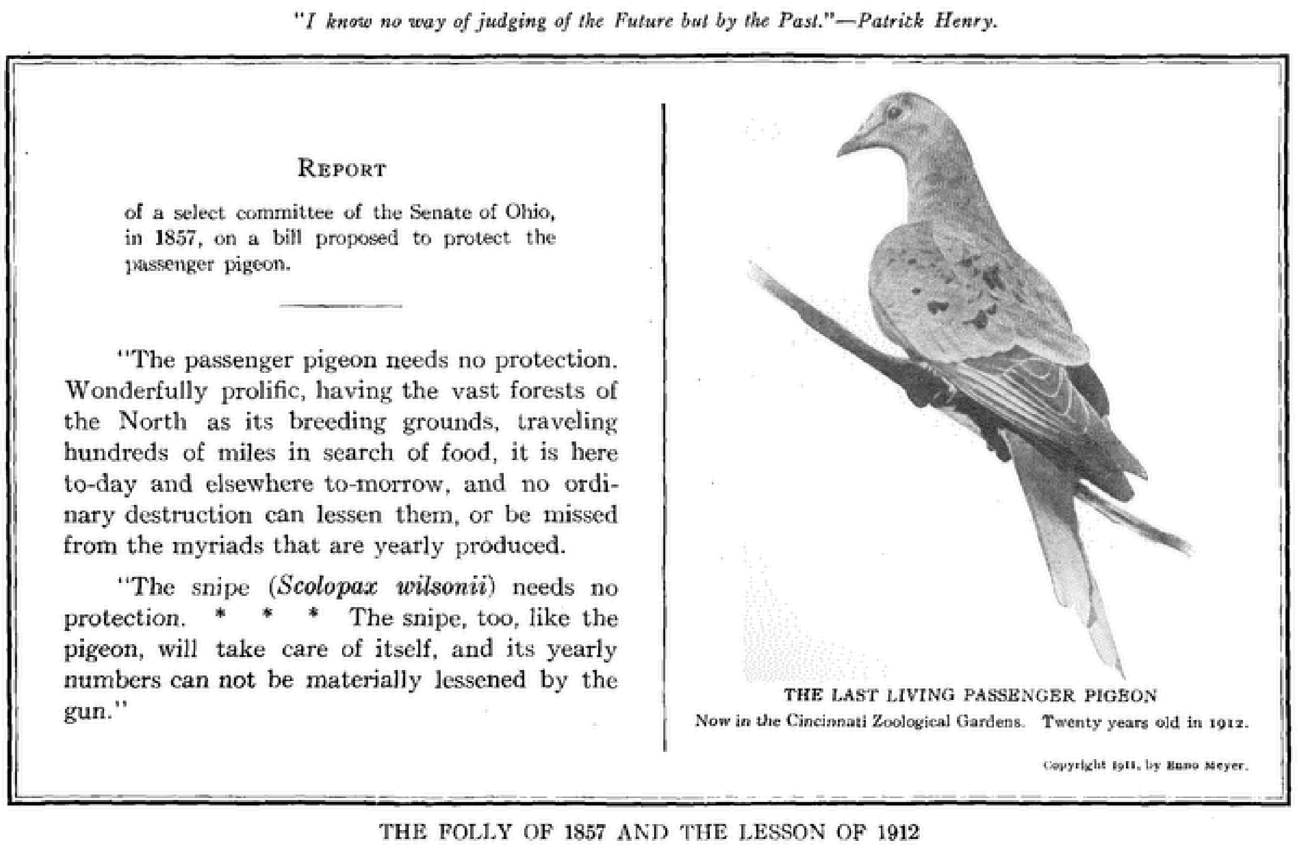
We will start our journey to analyzing text data with some classic sources: books and newspapers. Luckily for us, there are massive databases of these types of text.

We can start with the `gutenbergr` R package. We can load this package and read in a book from this list, most of which are in the public domain. We'll try a 1913 book called "Our Vanishing Wild Life: Its Extermination and Preservation."

```
#install.packages("gutenbergr")
library(gutenbergr)

# download the book - notice that the number is taken from the gutenberg website
vanishing_wl <- gutenberg_download(c(13249), meta_fields = "title")
```

Great! The book contains a lot of information on birds and other animals that have gone extinct, like this passenger pigeon:



"I know no way of judging of the Future but by the Past."—*Patrick Henry.*

REPORT

of a select committee of the Senate of Ohio, in 1857, on a bill proposed to protect the passenger pigeon.

"The passenger pigeon needs no protection. Wonderfully prolific, having the vast forests of the North as its breeding grounds, traveling hundreds of miles in search of food, it is here to-day and elsewhere to-morrow, and no ordinary destruction can lessen them, or be missed from the myriads that are yearly produced.

"The snipe (*Scolopax wilsonii*) needs no protection. * * * The snipe, too, like the pigeon, will take care of itself, and its yearly numbers can not be materially lessened by the gun."

THE LAST LIVING PASSENGER PIGEON

Now in the Cincinnati Zoological Gardens. Twenty years old in 1912.

Copyright 1911, by Bruno Meyer.

THE FOLLY OF 1857 AND THE LESSON OF 1912

But what can we actually do with the text? First, let’s take a look at the first 50 rows.

gutenberg_id	text	title
13249	“*I know no way of judging of the Future but by the Past*.”	Our Vanishing Wild Life: Its Extermination and Preservation
13249	–*Patrick Henry*.	Our Vanishing Wild Life: Its Extermination and Preservation
13249		Our Vanishing Wild Life: Its Extermination and Preservation
13249	REPORT	Our Vanishing Wild Life: Its Extermination and Preservation
13249		Our Vanishing Wild Life: Its Extermination and Preservation
13249	of a select committee of the Senate of Ohio, in 1857, on a bill proposed	Our Vanishing Wild Life: Its Extermination and Preservation
13249	to protect the passenger pigeon.	Our Vanishing Wild Life: Its Extermination and Preservation

It's a bit of a mess! But that's alright, we can clean it up. We'll start by using the `unnest_tokens()` function from the `tidytext` package (remember that you need to install and library this package before using it!). We will also use the `dplyr` package, which I'll elaborate on below. The first argument in `unnest_tokens()` , `word` , means we want to create a new variable called "word." The second argument, `text` , means we want to use the old variable called "text."

```
library(tidytext)
library(dplyr)

# try to tokenize into single words
vanishing_wl %>%
  unnest_tokens(word, text)
```

```
## # A tibble: 181,050 × 3
##    gutenberg_id title                                                  word
##           <int> <chr>                                                  <chr>
##  1        13249 Our Vanishing Wild Life: Its Extermination and Preservati… _i
##  2        13249 Our Vanishing Wild Life: Its Extermination and Preservati… know
##  3        13249 Our Vanishing Wild Life: Its Extermination and Preservati… no
##  4        13249 Our Vanishing Wild Life: Its Extermination and Preservati… way
##  5        13249 Our Vanishing Wild Life: Its Extermination and Preservati… of
##  6        13249 Our Vanishing Wild Life: Its Extermination and Preservati… judg…
##  7        13249 Our Vanishing Wild Life: Its Extermination and Preservati… of
##  8        13249 Our Vanishing Wild Life: Its Extermination and Preservati… the
##  9        13249 Our Vanishing Wild Life: Its Extermination and Preservati… futu…
## 10        13249 Our Vanishing Wild Life: Its Extermination and Preservati… but
## # ℹ 181,040 more rows
```

It's still a bit of a mess! Maybe even more so. But we are on our way to organizing this dataframe. Notice that in the "text" column, each row contains just one word now, rather than chunks of sentences. We will run this code again with the magrittr pipe to permanently change vanishing_wl.

```
# try to tokenize into single words
vanishing_wl %<>%
  unnest_tokens(word, text)
```

Great! Now we can do things like look at the frequency of word lengths:

```
library(stringr)

# table of wordlengths in vanishing_wl
vanishing_wl$word %>%
  str_length() %>%
  table()
```

```
## .
##     1     2     3     4     5     6     7     8     9    10    11    12    13
##  4677 31412 37517 31772 20981 15954 13749  9316  7103  4148  2225  1387   641
##    14    15    16    17    18    20
##   105    52     7     2     1     1
```

So we see that the majority of words have between 2 and 5 letters, which is probably what we would expect. This is not super interesting, but it's always good to do data checks like this to catch if anything is seriously wrong! Notice that the `$` sign tells the dataframe that we want to select a particular variable, or vector. We could also have used the `select()` command from `dplyr`.

Next, we can search for the frequency of particular words or parts of words. For example, maybe we want to know how times the book mentions "law" or "policy." The "|" symbol below denotes "or."

```
# count of words that contain "law"
vanishing_wl %>%
  filter(word == "law" | word == "policy") %>%
  count(word, sort = T)
```

```
## # A tibble: 2 × 2
##   word        n
##   <chr>   <int>
## 1 law       331
## 2 policy     14
```

We notice that the book mentions “law” 331 times, but only mentions “policy” 14 times. Try running similar code on your own to look at other words! You can use `str_starts()` and `str_ends()` to capture words that start or end with certain strings (groups of letters).

2.3 APIs

Application Program Interfaces, or APIs, are tools that allow researchers to pull data from websites. I won’t go into too much about what an API is doing or how it works, but you can learn more about this in one of the problem set videos.

Lists of APIs that may be useful for social science research can be found here and here. Many of these would be great sources of data for a final project.

Let’s start with pulling data from the Google Trends API. You might have used the Google Trends Interface directly before. Lucky for us, there’s an R package that can pull this information directly into our environment.

```
library(gtrendsR)
```

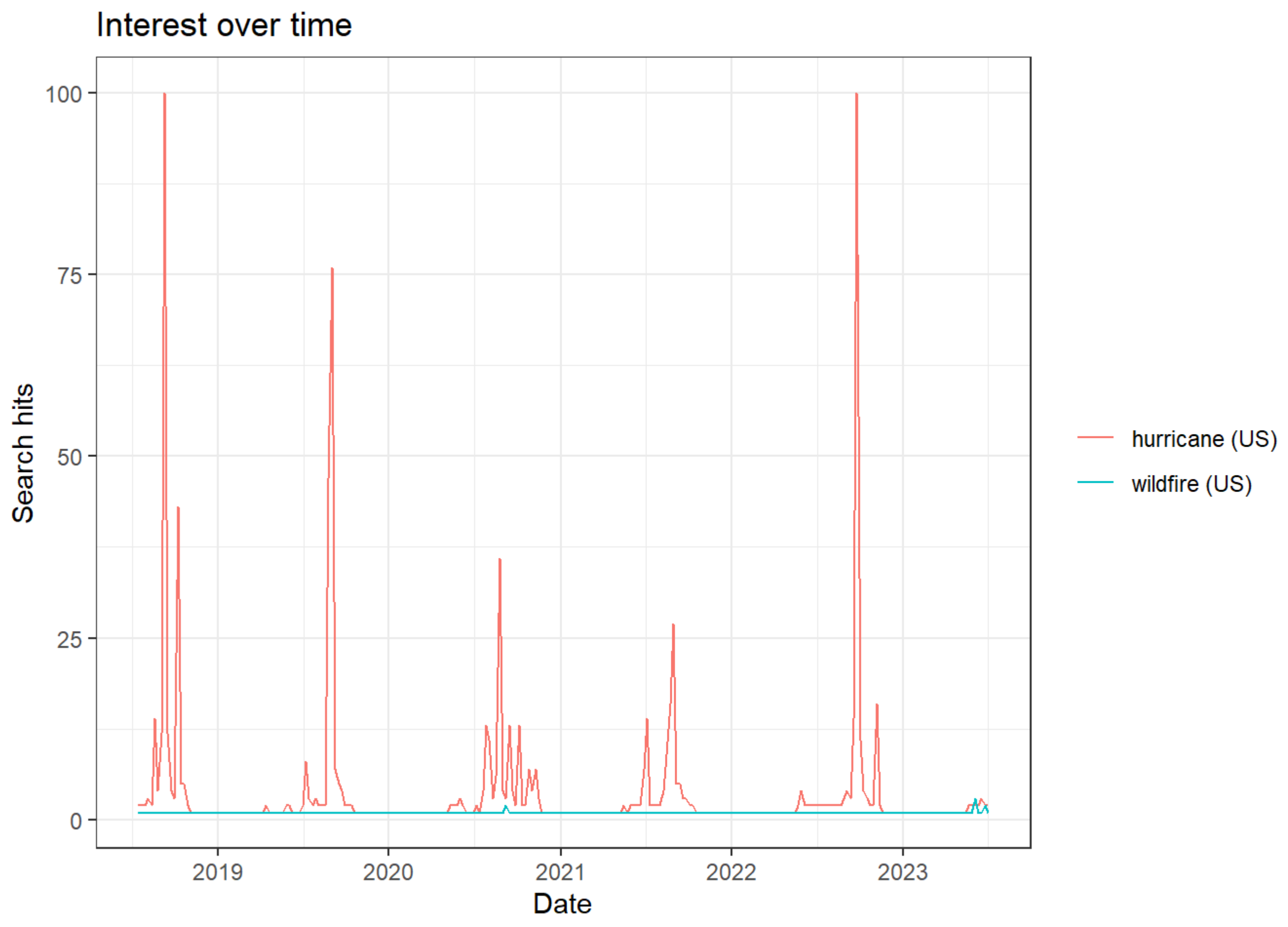
Great, now let’s look at a couple trends. We’ll start with the frequency that people are searching for “hurricane” and “wildfire.”

```
hur_wf <- gtrends(c("wildfire", "hurricane"),
                  geo = c("US"))
```

Notice that this returns a list of multiple dataframes and terms. We can explore these by typing `hur_wf$`. Also note the different parameters that we can set. When we specify `c()`, this is a simple way to define a one-dimensional vector of numerical or text entries. There are other options that we didn't specify here, which are evaluated at their default levels. For example, the default time period for these trends is "today+5-y", which means the last 5 years (you can see this by running `?gtrendsR`).

The package comes with a built-in capability for plotting time trends. We can look at these trends with the following code:

```
plot(hur_wf)
```

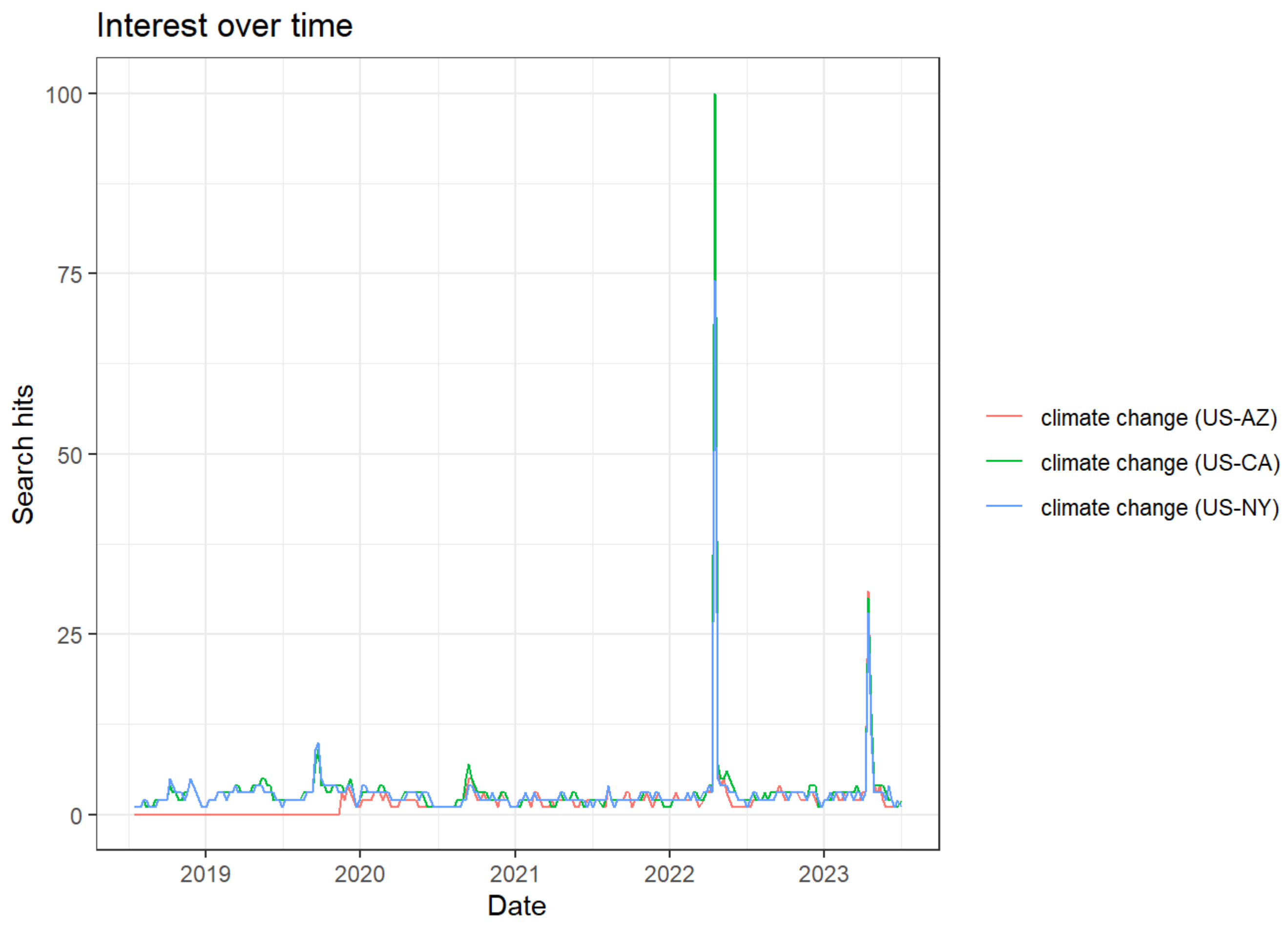


Woah, people search for hurricanes way more than wildfires! Is this surprising? We can also look at the timing of searches: do spikes roughly correlate to major hurricanes and wildfires in the U.S.? I encourage you to run this on your own, and investigate!

Next, we might want to use Google Trends to evaluate searches for ideas over time. For example, we can examine searches for "climate change" in different states.

```
climchange <- gtrends(c("climate change"),
                      geo = c("US-CA", "US-NY", "US-AZ"))
```

Notice that we can specify states within the U.S. We can also specify Metropolitan Statistical Areas (MSAs) if we want to. When we plot these time trends, the different states appear in different colors.



In all three states (Arizona, California, and New York), searches for climate change spike around September 2019, April 2022, and April 2023. Any ideas for why these spikes occur? On your own, I encourage you to play around a bit more with the other parameters in the `gtrends()` function and see if you can find any interesting phenomena.

We've seen one example of an API that runs straight out of an R package, no registration or setup required. However, most take a bit more work on the front end. To get a sense of what this can look like, we'll try using the API for the newspaper The Guardian.

First, we'll need a key to access the API. You can obtain one here by clicking "Register for a Developer Key."

Next, we can install and library the `guardianapi` R package. There is another R package, “guardian” which also works with this API, but it is not on CRAN, so it is a little trickier to set up. It is often the case that multiple R packages can perform a single task, so we may want to try the other one if we are running into difficulties.

```
library(guardianapi)
```

You will also need to run the `gu_api_key()` command and enter your API key. The Guardian API will let us pull the full newspaper text from Guardian articles within topics and date ranges that we specify. For example, we can pull articles on the recent Canadian wildfires and smoke in New York City with the following code:

```
ca_wf <- gu_content('"Canada" AND "wildfire" AND  "smoke" AND  "air quality" AND "New York
                         from_date = "2023-06-01")
```

Let’s take a look at what’s in our dataframe.

```
## # A tibble: 6 × 41
##   id         type  section_id section_name web_publication_date web_title web_url
##   <chr>      <chr> <chr>      <chr>        <dttm>               <chr>     <chr>  
## 1 world/20… arti… world      World news   2023-06-12 02:25:58  Poor air… https:…
## 2 us-news/… arti… us-news    US news      2023-06-08 17:08:23  Tens of … https:…
## 3 sport/20… arti… sport      Sport        2023-06-08 03:36:13  US sport… https:…
## 4 world/20… arti… world      World news   2023-06-10 00:06:29  Millions… https:…
## 5 us-news/… arti… us-news    US news      2023-06-09 01:00:37  New York… https:…
## 6 world/20… arti… world      World news   2023-06-07 22:32:29  ‘Out of … https:…
## # ℹ 34 more variables: api_url <chr>, tags <lgl>, is_hosted <lgl>,
## #   pillar_id <chr>, pillar_name <chr>, headline <chr>, standfirst <chr>,
## #   trail_text <chr>, byline <chr>, main <chr>, body <chr>, wordcount <dbl>,
## #   first_publication_date <dttm>, is_inappropriate_for_sponsorship <lgl>,
## #   is_premoderated <lgl>, last_modified <dttm>, production_office <chr>,
## #   publication <chr>, short_url <chr>, should_hide_adverts <lgl>,
## #   show_in_related_content <lgl>, thumbnail <chr>, legally_sensitive <lgl>, …
```

We notice there are lots of variables! We can take a closer look at a few by using the `select()` function. For example, you could run the following code to view what we see in the table below:

```
ca_wf %>%
  select(headline, byline, web_publication_date, body_text) %>%
  View()
```

headline	byline	web_publication_date	body_text

Fantastic, this illustrates one way that we can download a lot of text data from newspapers. How we analyze this is a separate question, which we will get to in later weeks!

Lastly, I'd like to note that one of the most popular and interesting sources for online text analysis is Twitter. Twitter has an API for academic researchers to use this information, and would be a great resource for final projects. In this week's problem set, we'll apply for Twitter APIs!

2.4 Web Scraping

Another method for gathering data from the internet is called web scraping. In the "wild west" days of the internet, sites were fairly disconnected and scraping was an acceptable way to gather information from sites. In the modern day, many sites disallow web scraping, and offer APIs as alternatives. Therefore, it is important to check that you are not violating terms of agreement in scraping websites. You can check a site's regulations by typing the website and adding "robots.txt" to the end of the url. For example, if we go to https://en.wikipedia.org/robots.txt we see that some crawlers and bots have been banned for going to fast or for other actions, but they do not list any total restrictions on scraping.

To try scraping a Wikipedia page, we will first download and library the `rvest` package.

```
library(rvest)
```

Great! Now we can try reading an HTML file (the language behind a website, which tells your browser the meaning and structure of the site) into R. It will work best for us to read in a Wikipedia page with a table, so let's try the costliest disasters in history.

```
# read in html
us_disasters <- read_html("https://en.wikipedia.org/wiki/List_of_natural_disasters_in_the_U

# take a look
us_disasters
```

```
## {html_document}
## <html class="client-nojs vector-feature-language-in-header-enabled vector-feature-langua
## [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
## [2] <body class="skin-vector skin-vector-search-vue mediawiki ltr sitedir-ltr ...
```

Hmmmm, it's not really in a readable format. But that's ok! In order to extract our table of interest, we just need to gather a little more information:

1. Go to the page of interest and find the table that we want to extract
2. Right click the table and click inspect
3. Find the <table… element of interest in the Element pane on the right-hand side. When you hold your mouse over this text, it should highlight the entire table on the left-hand side
4. Right-click the table element, Copy -> Copy Xpath

Great, now that we have the Xpath copied we can use `rvest` 's `html_nodes()` function to extract just the html of interest.

```
# extract nodes of interest
us_disasters_table <- html_nodes(us_disasters,
                                 xpath = '//*[@id="mw-content-text"]/div[1]/table[3]')

# take another look
```

It is still in html format, but there is another function, `html_table()` , which will convert this to a dataframe. Let's try it!

```
# convert object to table
us_disasters_table %<>%
  html_table()

#take another look
us_disasters_table %>%
  head()
```

```
## [[1]]
## # A tibble: 173 × 7
##    Year  Disaster    `Death toll` `Damage costUS$` `Main article` Location Notes
##    <chr> <chr>       <chr>        <chr>            <chr>          <chr>    <chr>
##  1 2023  Tornado ou… 33           "$4.3 billion"   Tornado outbr… Souther… ""
##  2 2023  Tornado ou… 25           "$1.9 billion"   Tornado outbr… Souther… "Inc…
##  3 2023  Flooding a… 13           "$4.5 billion"   Early-March 2… Southwe… ""
##  4 2023  Derecho, T… 14           ""               February 2023… Western… ""
##  5 2022  Winter sto… 106          "$5.4 billion"   December 2022… Western… ""
##  6 2022  Earthquake  2            ""               2022 Ferndale… North C… ""
##  7 2022  Winter sto… 4            ""               November 2022… Great L… ""
##  8 2022  Hurricane   11           "≥ $1 billion"   Hurricane Nic… Dominic… "Nic…
##  9 2022  Hurricane   157+         "≥ $113.1 billi… Hurricane Ian  Trinida… "Hur…
## 10 2022  Hurricane   25           "≥$5.88 billion" Hurricane Fio… Puerto … ""
## # ℹ 163 more rows
```

This looks better. But notice the `[[1]]`? This means that our table is still technically in list format, and that this is the first element of a list. We will change that below with the `as.data.frame()` function.

Also, you might be wondering, what is a tibble? According to CRAN, "Tibbles are a modern take on data frames. They keep the features that have stood the test of time, and drop the features that used to be convenient but are now frustrating." For our purposes, we can treat them just like dataframes.

```
# convert list to dataframe
us_disasters_table %<>%
  as.data.frame()

# take another look
us_disasters_table %>%
  head()
```

```
##   Year                                      Disaster Death.toll Damage.costUS.
## 1 2023                              Tornado outbreak         33   $4.3 billion
## 2 2023                              Tornado outbreak         25   $1.9 billion
## 3 2023                 Flooding and Tornado outbreak         13   $4.5 billion
## 4 2023 Derecho, Tornado outbreak and Winter storm         14
## 5 2022                                  Winter storm        106   $5.4 billion
## 6 2022                                    Earthquake          2
##                                         Main.article
## 1  Tornado outbreak of March 31 – April 1, 2023
## 2         Tornado outbreak of March 24–27, 2023
## 3 Early-March 2023 North American storm complex
## 4    February 2023 North American storm complex
## 5     December 2022 North American winter storm
## 6                        2022 Ferndale earthquake
##
## 1                                                                              So
## 2
## 3                                                                        Southwes
## 4                                                    Western United States, South
## 5 Western United States, Midwestern United States, Great Lakes region (especially the Bu
## 6
##                                                     Notes
## 1
## 2 Includes the 2023 Rolling Fork–Silver City tornado.
## 3
## 4
## 5
## 6
```

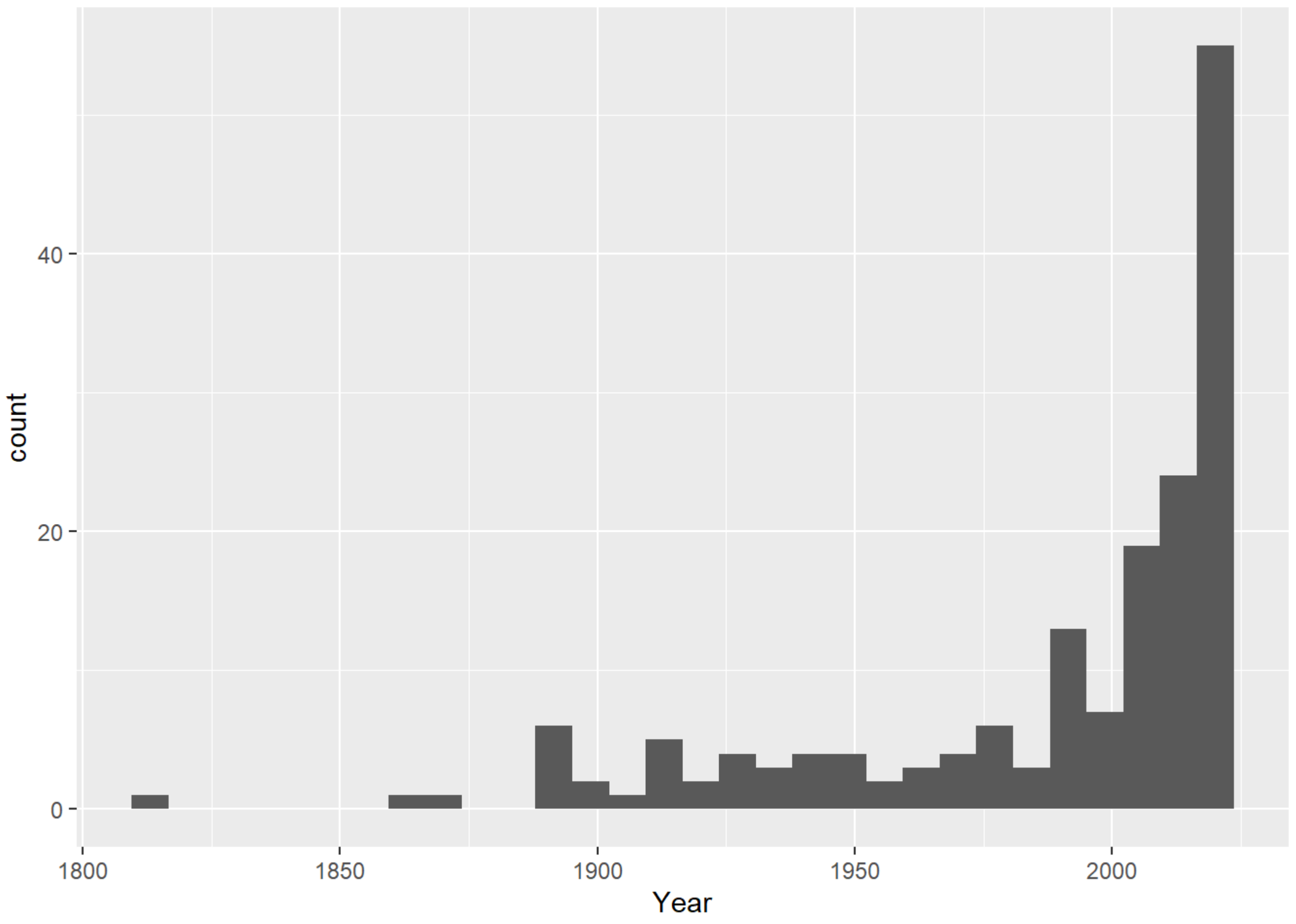
Ok perfect, now we have a true dataframe. In order to look at our data, we should clean it up a bit first. We'll start by creating new variables which convert `Year` and `Death.toll` to numeric values (which are easier to work with).

```
# create new variables
us_disasters_table %<>%
  mutate(year = as.numeric(Year),
         death_toll = as.numeric(Death.toll))
```

You will get some warning messages like “NAs introduced by coercion.” What do you think this means?

For now we will move on. Now that we have some numeric variables, we can make some simple plots. First, let’s look at the counts of disasters by year.

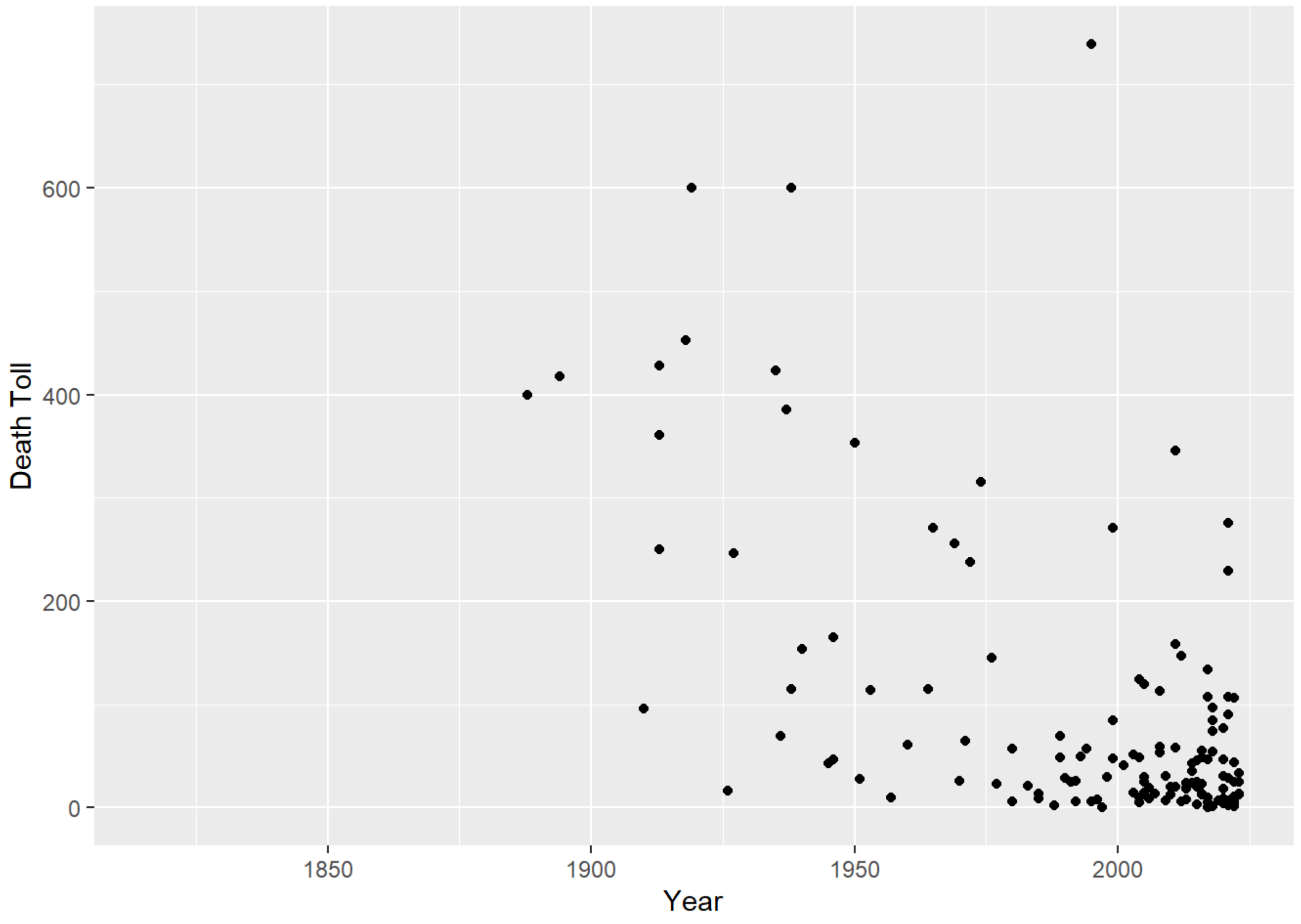
```
library(ggplot2)
# first, we can plot a histogram of disaster years
ggplot(us_disasters_table, aes(x = year))+
  geom_histogram()+
  labs(x = "Year")
```



It seems that disasters are **really** increasing in frequency in the 2000's! This might make sense to us given what we know about climate change. But should we also be skeptical of this result?

Let's look at the disaster death tolls over the years.

```
# now, we can look at death tolls across years
ggplot(us_disasters_table, aes(x = year, y = death_toll))+
  geom_point()+
  labs(x = "Year", y = "Death Toll")
```



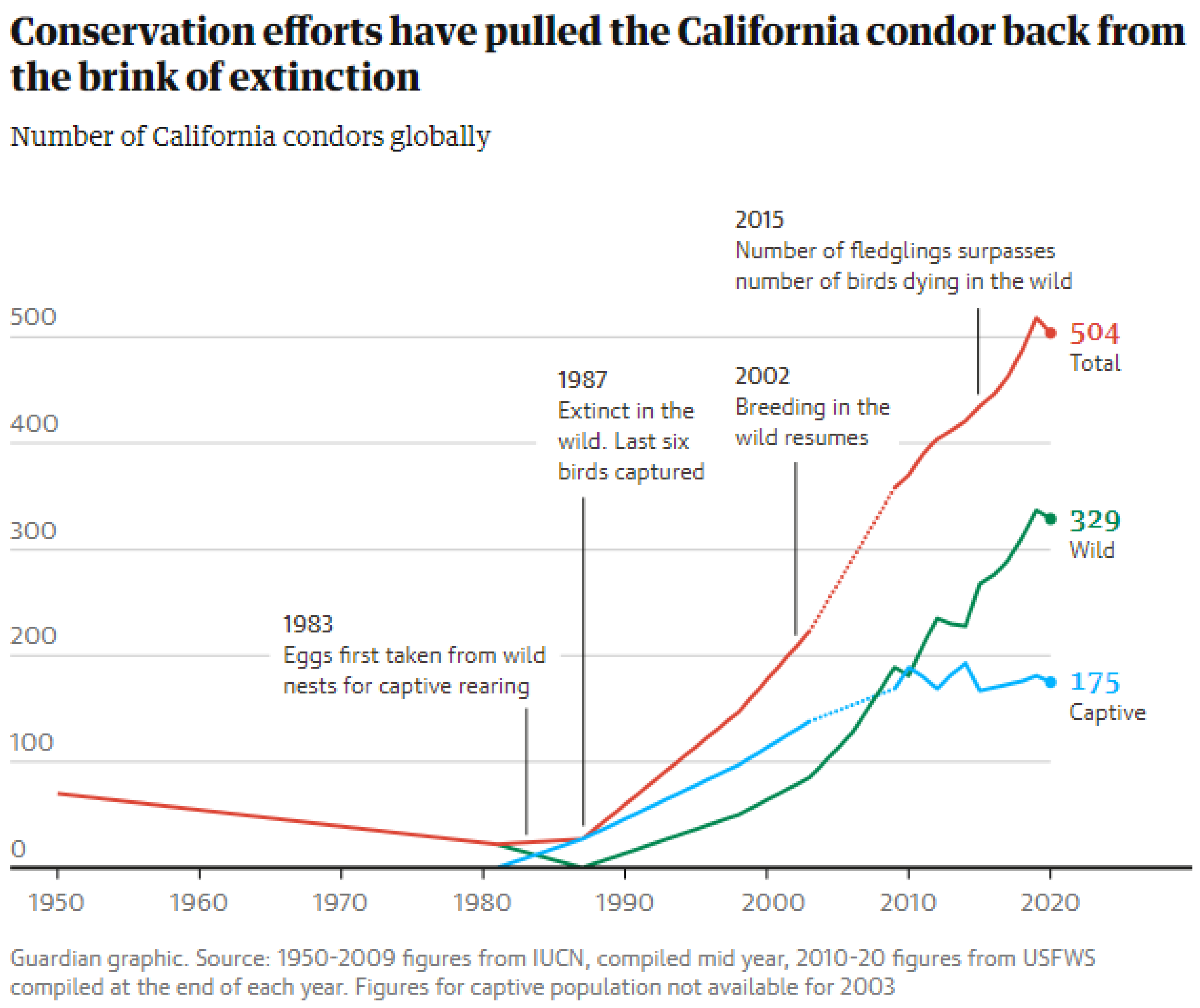
Interesting! It seems as though most of the recent disasters have *lower* death tolls than historic events. It could be the case that our modern infrastructure protects people today better than it used to when disasters hit. But a more likely explanation is that we don't have records of all the small-scale events of previous decades and centuries, at least on a place like Wikipedia. That is, we likely have *non-random missingness* in our data. Wherever we gather data, we should be wary of this, but especially when we gather data from websites and organizations where the data was not created for researchers.

2.5 Disasters, Risk, and Resillience

In this section, we've focused on information around extinction and disasters. We'll cover these more in future weeks, but for now, let's define some key terms and frameworks for understanding these.

Biodiversity loss is the reduction of genes, species, and traits in an ecosystem. We are currently in the middle of earth's sixth mass extinction event, in which most vertabrates are experiencing decline in population sizes and ranges. Mounting evidence shows that ecosystems function more efficiently and are more stable with greater biodiversity, and that change in an ecosystem can accelerate, and can become more unpredictable, with biodiversity loss. In other words, changes in earth's ecology could have wide-ranging impacts. Researchers have also noted that "the impacts of diversity loss on ecological processes might be sufficiently large to rival the impacts of many other global drivers of environmental change."

The news on biodiversity can be dismal … but conservation efforts have paid off in some cases! Below is a chart documenting the return of the California Condor. (Source)



Risks are probabilities that people or places will be affected by events such as natural disasters. In a famous sociological work called Risk Society, Ulrich Beck defines “risk positions” as individuals’ social locations in relation to various risks. Beck argues that in modernity, society will be organized largely according to risk positions, rather than class positions. That is - people’s proximity to dangers such as flooding or wildfires may determine their life outcomes more so than their wealth. As Paprocki, Cohen, Koslov and Elliott noted in one of our first readings, the consequences of climate change sometimes correspond to social class (i.e. more advantaged countries or people are able to leverage resources to avoid climate dangers), but sometimes are distributed across nations and people. Wildfire smoke might be an example of the latter case, particularly if it is unexpected and people are unable to prepare. Beck captures this idea by stating that while “poverty is hierarchic, smog is democratic.”

We can also think about risks and how they are connected with places. Floodplains are a great example of this - state and local governments are tasked with creating risk distribution maps at various levels (not in any floodplain, 100-year floodplain and 500-year floodplain, as well as more detailed assessments). These estimates are then used by FEMA to produce insurance rates for homeowners in the floodplain. However, many of these estimates may be incorrect (see this article) or the insurance rates may be adjusted in ways that do not reflect the real risks (we will read about this a little bit in the coming week). A critical sociology of risk interrogates the power structures that produce risks, including governments and other institutions, and how these relate to inequalities in society.

Resilience is the ability to withstand or avoid the negative consequences of disasters. Resilience might vary across age, gender, race, health, and other social characteristics. As sociologists, we note that differences in resilience are often not due to individual actions, but the social structures they are embedded in. For example, in his book Heat Wave, Eric Klinenberg investigates differences in death rates between two Chicago neighborhoods with relatively similar levels of economic disadvantage, aging populations, and location in the urban space. He finds social networks in one neighborhood - which resulted from a vibrant commercial district, public spaces, and reduced fears of crime - drove neighbors to check on each other, preventing many deaths that otherwise would have occurred. While age put some individuals at higher risk of heat stroke, their social ties enabled or prevented resilience.

Across populations, sociologists sometimes discuss resilience as segmented. This means that groups experience different levels of resilience, and can lead to changes in the overall social structure of the place. For example, certain economic or ethnoracial groups may experience displacement at higher rates than others, or some groups may choose to leave a place while others remain. In the readings for the coming week, we'll learn a bit more about Hurricane Katrina and the social changes that took place in New Orleans after this catastrophic event.

2.6 Problem Set 2

Due: July 10th, 2023

Recommended Resources:

R Basics

APIs

How to Get Access to the Twitter API

Optional: Academic Research with Twitter API V2

A note: this assignment and future assignments should be submitted as PDF or HTML documents generated using R Markdown or a Jupyter Notebook. If you experience issues creating these files, come to office hours or reach out!

1. Load the temps.csv dataframe into your environment. Write a function that transforms the Fahrenheit temperatures to Celsius, and create a new variable for the Celsius temperatures.
2. Try out the `gtrendsR` package! Load it into your machine, library it, and try searching for one or more environmental keywords of your choosing (e.g. “carbon sequestration”, “hurricane harvey”, “fossil fuels”, “climate justice”, etc.). Plot these keywords over time, and see if you can explain the spikes and dips in searches.
3. ~~Apply for a Twitter Developer Account. You should get an email that your application has been sent. Take a screenshot and include it in your R Markdown or Jupyter Notebook.~~ Nevermind! Apparently the main research functions are no longer free :(
4. Download us_disasters_table.csv from Canvas. Create a new variable (you may want to use the `mutate` function) called `new_cost` , which takes `Damage.cost.US` and converts it to numeric values. You can use the `money_numeric` function below to do so.

```
library(stringr)
# function to conver "X billion" or "Y million" into numeric values
money_numeric <- function(money){
  # get numeric amount (we will use the first cost provided, although this may cause issue
  num <- str_extract(money, "\\$([0-9,.]+)")
  # remove the dollar sign
  num %<>%
    str_remove_all("\\$")

  # get the million/billion value
  order <- str_extract(money, "million|billion")
  # replace these with appropriate number of zeros
  order %<>%
    str_replace("million", "1000000") %>%
    str_replace("billion", "1000000000")

  # create value
  val = as.numeric(num)*as.numeric(order)

  return(val)
}
```

Examine the new cost variable using the `View()` function. Do you feel that the function did a reasonable job of converting the damage costs into numeric values? Why or why not?

5. Using the same data, with the new cost variable, plot damage costs alongside one of the other variables. What type of relationship do you see? Are there reasons we should trust, or not trust, this apparent relationship?

6. Choose an API from one of the lists above, such as The Guardian API, or one that you find on your own. Try to download some data! Show a snippet of this data (for example, using the `head()` function), and comment on the types of questions that you might answer with it. You don't need to do any data cleaning or analysis yourself for this question.