10: Lab - Data Scraping

Environmental Data Analytics | John Fay

spring 2025

Objectives

1. Answer questions on M9
2. Web services: URLs, "REST", and APIs
3. Scraping Exercises

Set up

```
#Install familiar packages
library(tidyverse);library(lubridate);library(viridis);library(here)
here()
```

```
## [1] "C:/Workspace/Teaching_S25/EDE_base"
```

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

#install.packages("dataRetrieval")
library(dataRetrieval)

#install.packages("tidycensus")
library(tidycensus)

# Set theme
mytheme <- theme_gray() +
  theme(axis.text = element_text(color = "black"),
        legend.position = "top")
theme_set(mytheme)
```

Web Services

An Exercise

- Open your browser to Google's home page
- Search for "Duke University"
- Check the URL of the search result
- Within the url is `q=Duke+University`
- Try the URL: "https://www.google.com/search?q=NSOE"

An Explainer

- Interacting with the web boils down to **requests** and **responses**
- Requests are often sent as browser URLs. They usually include:
- The address of the server that will handle the request
- The name of the service on that server that will process the request
- The parameters of the service > Can you identify these components in the Google search request?
- Responses are sent as text files, which are interpreted by a browser
- Written in HTML: Tags (w/attributes) & Values, which browsers can interpret
- Often include JavaScript, which browsers can run
- "REST" is the framework of sending requests and handling responses - all in text.
- "REpresentational State Transfer"
- REST is a form of an API..

Why do we care?

- Navigate to: https://waterdata.usgs.gov/nwis
- Click on "Current Conditions"
- Click on "Build Time Series"
- Check the box next to "Site Number" and click "Submit"
- Enter the site number `02087500` and scroll to the bottom of the page
- Select the very last radio button: "Tab-separated data", but change "Save to file" to "Display in browser"
- Click "Submit"

You should be presented with a page of the latest water flow data for the Neuse River near Clayton, NC.If you had issues, a shortened link is here: https://on.doi.gov/3ewCMPP.

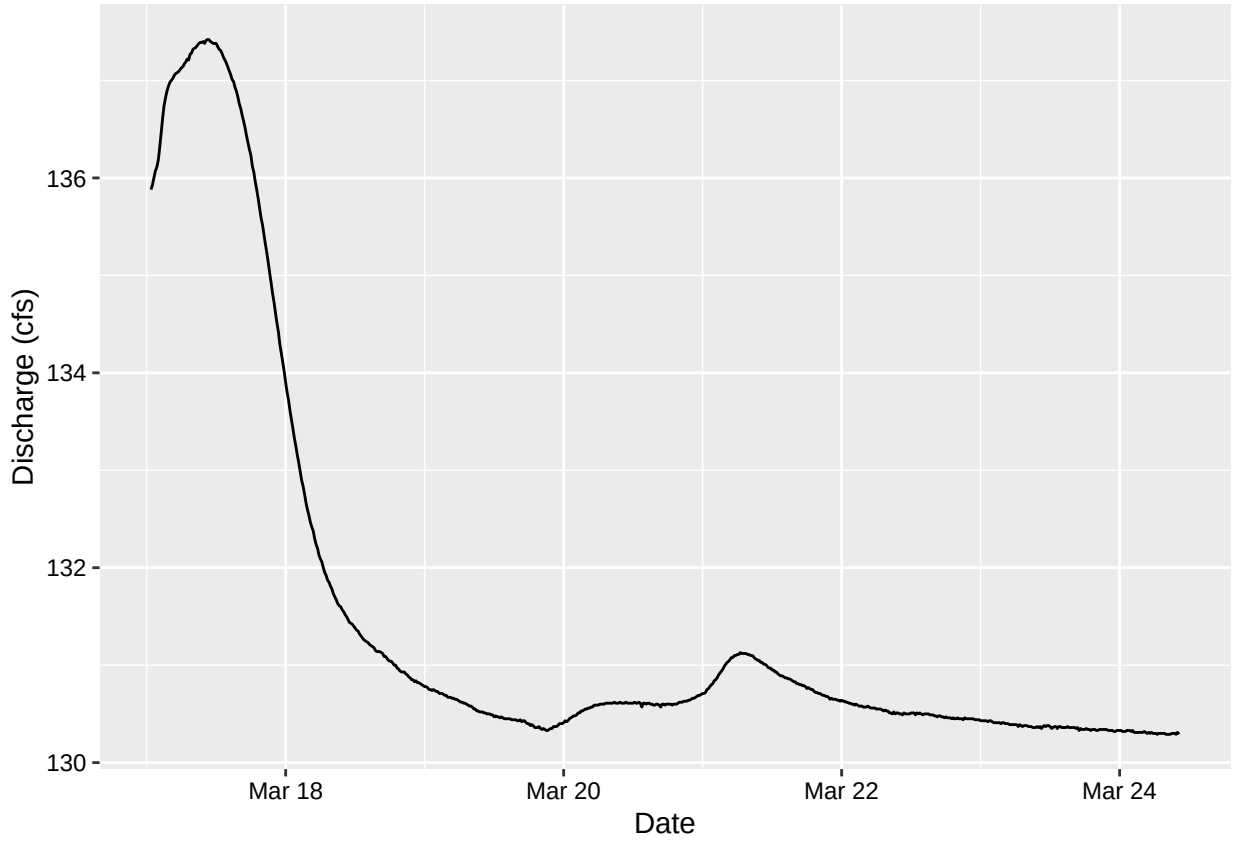
Have a look at the link (not the shortened one) and you'll see that it has the RESTful format. We can easily edit the URL to fetch data for another site: * In the URL replace `02087500` with `02085070` and re-submit. What site's data do we have now? Saved a lot of clicks in getting to this dataset, didn't we?

The power in this is that we aren't limited to sending requests and handling responses in a browser; we can do this via scripts in R. If the data we are retrieving is a simple text file, as it is here, we don't even need any special packages, though we do need to do a bit of wrangling to skip header lines...

Pull and plot discharge data

```
#Set the URL
theURL <-
  'https://waterdata.usgs.gov/nwis/uv?search_site_no=02087500&period=7&format=rdb'

#Get the data, which starts on line 30
gage_data <- read.table(
  theURL,
  skip = 29,
```

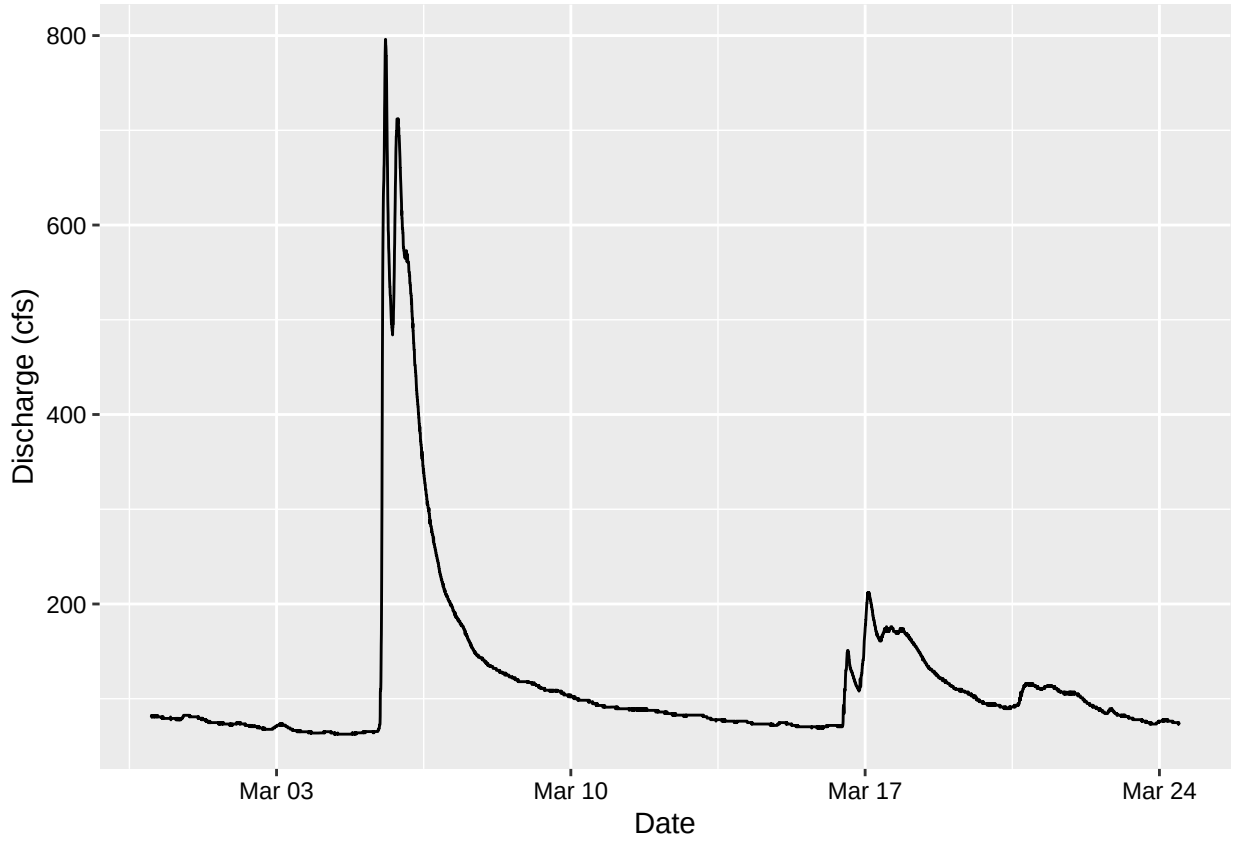
```
  header=TRUE,
  sep='\t',
  stringsAsFactors = T)

#Update the column headers
colnames(gage_data) = c(
  "agency_cd", "site_no", "datetime", "tz_cd",
  "discharge_cfs", "89192_00060_cd", "gage_ht_ft", "89193_00065_cd")

#Tidy the data
gage_data <- gage_data %>%
  select(datetime, discharge_cfs, gage_ht_ft) %>%
  mutate(datetime = ymd_hm(datetime))

#Plot
ggplot(gage_data,aes(x=datetime,y=discharge_cfs)) +
  geom_line() +
  labs(x = "Date",y = "Discharge (cfs)")
```



Exercise:

Alter the above code so that it plots the last *24* days of flow for gage *02085070*

```r
#Set the URL
theURL <-
  'https://waterdata.usgs.gov/nwis/uv?search_site_no=02085070&period=24&format=rdb'

#Get the data, which starts on line 30
gage_data <- read.table(
  theURL,
  skip = 29,
  header=TRUE,
  sep='\t',
  stringsAsFactors = T)

#Update the column headers
colnames(gage_data) = c(
  "agency_cd", "site_no", "datetime", "tz_cd",
  "discharge_cfs", "89192_00060_cd", "gage_ht_ft", "89193_00065_cd")

#Tidy the data
gage_data <- gage_data %>%
  select(datetime, discharge_cfs, gage_ht_ft) %>%
  mutate(datetime = ymd_hm(datetime))

#Plot
ggplot(gage_data,aes(x=datetime,y=discharge_cfs)) +
  geom_line() +
  labs(x = "Date",y = "Discharge (cfs)")
```



APIs

Application programming interfaces, or APIs, are a formalization of the REST request/response process. More precisely, they are web sites that have been designed to allow a client to query specific bits of data from an on-line resource without just "hacking" a URL.

More and more organizations are providing API access to their data. Have a look at the US Census catalog of APIs to grab demographic data: https://www.census.gov/data/developers/data-sets.html.

And with the formalization of APIs, more R packages are being written to streamline access to data via these APIs. Let's look at some examples.

Grabing discharge data via the `dataRetrieval` package

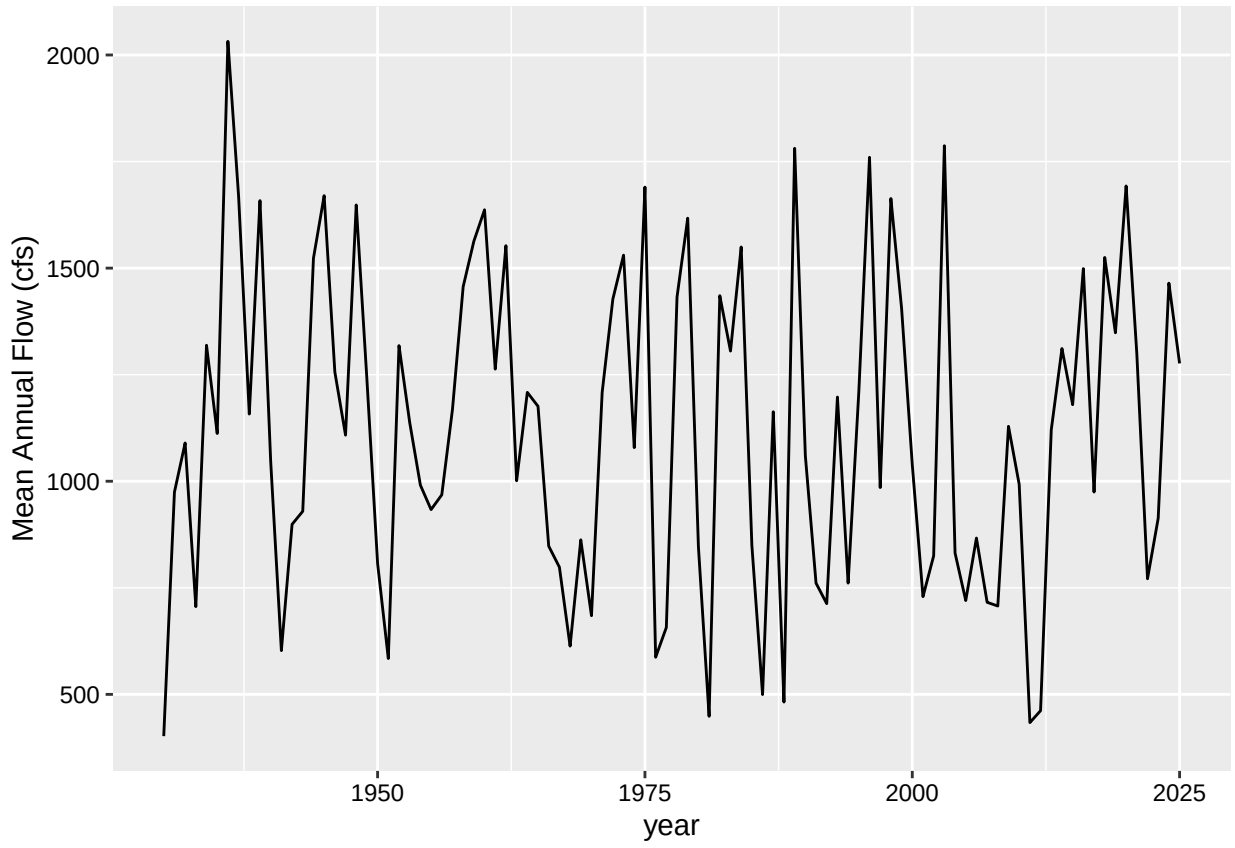
More info: https://cran.r-project.org/web/packages/dataRetrieval/vignettes/dataRetrieval.html

```
#Identify gauge to download
siteNo <- '02087500'

#Identify parameter of interest:
pcode = '00060' #discharge (cfs)

#Identify statistic code for daily values:
scode = "00003"  #mean
```

```r
#Identify start and end dates
start.date = "1930-10-01"
end.date = Sys.Date() #Gets today's date

#Load in data using the package's "readNWISdv" function
neuse <- readNWISdv(siteNumbers = siteNo,
                    parameterCd = pcode,
                    statCd = scode,
                    startDate=start.date,
                    endDate=end.date)

#Rename columns to something more useful
neuse <- renameNWISColumns(neuse);
colnames(neuse)
```

```
## [1] "agency_cd" "site_no"   "Date"      "Flow"      "Flow_cd"
```

```r
#Compute annual means
neuse_by_year <- neuse %>%
  group_by(year = year(Date)) %>%
  summarise(mean_annual_flow = mean(Flow))

#Plot
ggplot(neuse_by_year, aes(x=year,y=mean_annual_flow)) +
  geom_line() +
  labs(y="Mean Annual Flow (cfs)")
```



Using `tidycensus` to extract census data

The `tidycensus` package uses the Census' API to pull data into our coding environment. To use it, however, you first need to get a free Census API key at https://api.census.gov/data/key_signup.html. After that, it's pretty easy to fetch data into your coding environment.

We'll use it to extract total population of counties in NC.

You'll also need to look up the IDs of the variables you want to extract. Those are found here: https://walker-data.com/tidycensus/articles/basic-usage.html#searching-for-variables

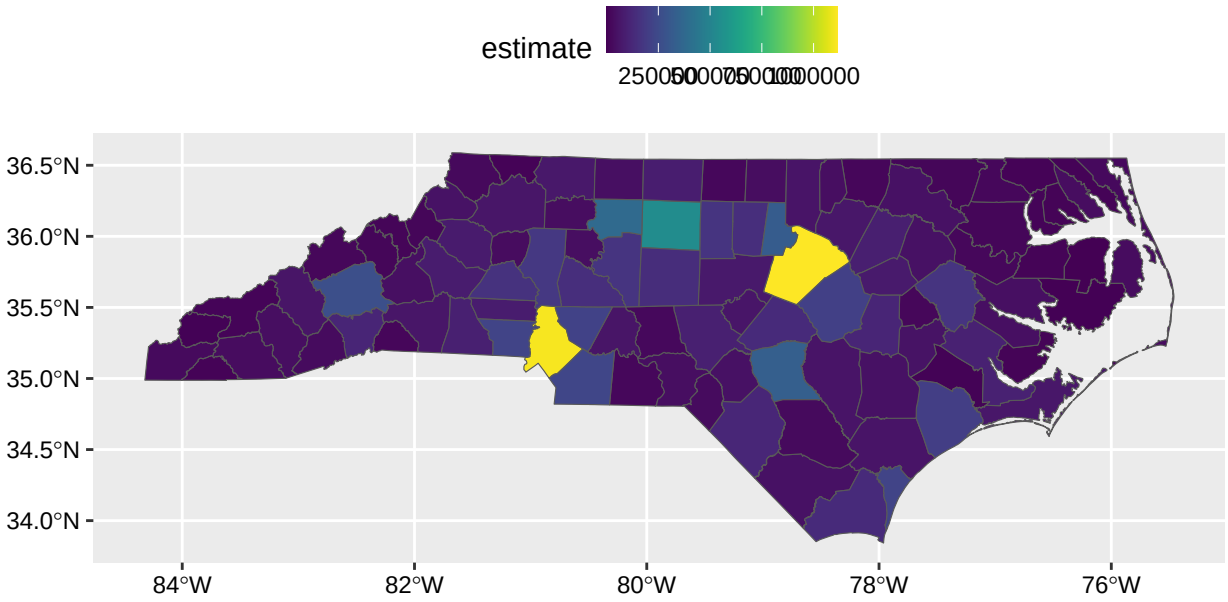
```
#Apply the API key
#census_api_key("d4ad7874c996c7a83fec0a81466309ddc07d8e22a", install = TRUE)

nc_pop <-
  get_acs(geography = "county",
          variables = "B01003_001",
          state = "NC",
          geometry = TRUE)
```

```
## Getting data from the 2017-2021 5-year ACS
```

```
## Warning: * You have not set a Census API key. Users without a key are limited to 500
## queries per day and may experience performance limitations.
## i For best results, get a Census API key at
```

```
## http://api.census.gov/data/key_signup.html and then supply the key to the
## `census_api_key()` function to use it throughout your tidycensus session.
## This warning is displayed once per session.

## Downloading feature geometry from the Census website.  To cache shapefiles for use in future sessions

##   |                                                                              |
```

```
ggplot(nc_pop, aes(fill = estimate)) +
  geom_sf() +
  coord_sf(crs = 4326) +
  scale_fill_viridis(option = "viridis")
```



Scraping Exercises

APIs are great, when they are available. But if not, we can still scrape data from a web site.

Let's scrape data the US Energy Administration's State Electricity Archive: https://www.eia.gov/electricity/state/

1. Navigate to the page. Note the year for which these data are provided.
2. Use the Selector Gadget tool to scrape the following values into lists:

- State name

- Average Retail price
- Net summer capacity
- Net generation
- Total retail sales

3. Compile the data into a table
4. Repeat for 2021

The RVest scraping workflow is as follows: 1. Connect to the website using the `read_html` function. 2. Locate specific elements in the web site via the node IDs found using Selector Gadget, reading them in using `html_nodes` 3. Read the text value(s) associated with those nodes into the coding environment via `html_text` 4. Wrangle values into a dataframe, being sure to convert numeric values to numbers

```
#1 Link to the web site using read_html
the_website <- read_html('https://www.eia.gov/electricity/state/')

#2&3 Locate elements and read their text attributes into variables
the_states <- the_website %>% html_nodes('td:nth-child(1)') %>% html_text()
the_price <- the_website %>% html_nodes('td:nth-child(2)') %>% html_text()
the_capacity <- the_website %>% html_nodes('td:nth-child(3)') %>% html_text()
net_generation <- the_website %>% html_nodes('td:nth-child(4)') %>% html_text()
total_retail <- the_website %>% html_nodes('td:nth-child(5)') %>% html_text()

#3 Construct a dataframe from the values
energy_data <- data.frame(
  "State" = the_states,
  "Price" = as.numeric(the_price),
  "Capacity" = as.numeric(gsub(',','',the_capacity)),
  "Net Generation" = as.numeric(gsub(',','',the_capacity)),
  "Total Retail" = as.numeric(gsub(',','',total_retail))
) %>%
  filter(State != 'U.S. Total')
```

```
scrape.it <- function(the_year){
  #Get the proper url
  the_url <- ifelse(
    the_year>=2023,
    'https://www.eia.gov/electricity/state/',
    paste0('https://www.eia.gov/electricity/state/archive/',the_year,'/')
    )

  #Fetch the website
  the_website <- read_html(the_url)
  print(the_url)

  #Scrape the data
  the_states <- the_website %>% html_nodes('td:nth-child(1)') %>% html_text()
  the_price <- the_website %>% html_nodes('td:nth-child(2)') %>% html_text()
  the_capacity <- the_website %>% html_nodes('td:nth-child(3)') %>% html_text()
  net_generation <- the_website %>% html_nodes('td:nth-child(4)') %>% html_text()
  total_retail <- the_website %>% html_nodes('td:nth-child(5)') %>% html_text()

  #Convert to dataframe
```

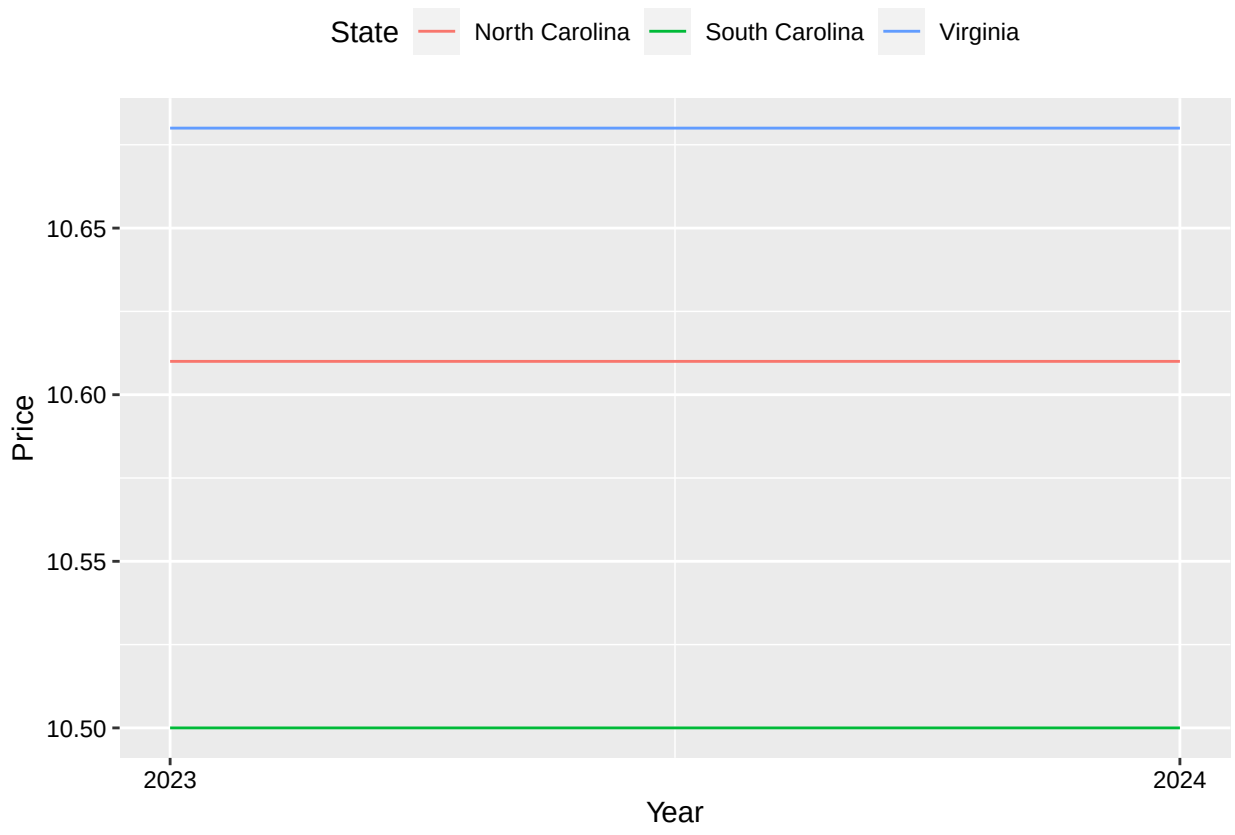
```
  energy_data <- data.frame(
    "State" = the_states,
    "Price" = as.numeric(the_price),
    "Capacity" = as.numeric(gsub(',','',the_capacity)),
    "Net Generation" = as.numeric(gsub(',','',the_capacity)),
    "Total Retail" = as.numeric(gsub(',','',total_retail))
  ) %>%
    filter(State != 'U.S. Total') %>%
    mutate(Year = my(paste('1,',the_year)))

  #Return the dataframe
  return(energy_data)
}

# Use the above function to scrape data for 2017
energy_2017 <- scrape.it(2017)
```

```
## [1] "https://www.eia.gov/electricity/state/archive/2017/"
```

```
# Map the function to scrape data from 2017 to 2023
energy_17_24 <- seq(2023,2024) %>% map(scrape.it) %>% bind_rows()
```

```
## [1] "https://www.eia.gov/electricity/state/"
## [1] "https://www.eia.gov/electricity/state/"
```

```
# Example plot
energy_17_24 %>%
  filter(State %in% c('North Carolina','South Carolina', 'Virginia')) %>%
  ggplot(aes(x=Year, y=Price, color=State)) +
  geom_line() +
  #geom_smooth(method='loess',se=FALSE) +
  scale_x_date(date_breaks = '1 year', date_labels = '%Y')
```


State
North Carolina
South Carolina
Virginia
10.65
10.60
10.55
10.50
Price
2023
2024
Year