Lab 3: Web Scraping in R

2024-04-19

Web Scraping: the act of extracting data from websites.

Getting Data From Online

As economists, we often want data that is freely available online, but it is unclear how to get it from the online table into a dataframe we can analyze in R.

Financial and Macroeconomic Data

The Federal Reserve Economic Data (FRED) is home to hundreds of time series economic variables that are freely available.

You *can* freely download the data to a CSV or Excel file and then load that data into R using methods we have seen previously, but this is time consuming and memory consuming.

There is an R-package that allows us to directly download the data!

tidyquant

```
# Load packages
library(pacman)
p_load(dplyr, tidyquant) # new: tidyquant
```

The main function we will use from this package is **tq_get("SERIES ID", get = , from = , to =)**. For now, let's focus on FRED data

- **SERIES ID**: Each time series dataset has a unique ID such as: *GDP* (for GDP), *GDPC1* (real GDP), *UNRATE* (unemployment rate)
- **get = "economic.data"**: this tells the function that you want FRED data
- **from = " "** and **to = " "**: dates for the data you want. Excluding the **to** input gives you up until the most recent data

Let's download the unemployment rate data and take a look

```
un_data = tq_get("UNRATE", get = "economic.data", from="1948-01-01")

summary(un_data)
```

```
##     symbol               date                price       
##  Length:915         Min.   :1948-01-01   Min.   : 2.500  
##  Class :character   1st Qu.:1967-01-16   1st Qu.: 4.400  
##  Mode  :character   Median :1986-02-01   Median : 5.500  
##                     Mean   :1986-01-30   Mean   : 5.698  
##                     3rd Qu.:2005-02-15   3rd Qu.: 6.700  
##                     Max.   :2024-03-01   Max.   :14.800  
```

It gives us 3 variables: symbol ("UNRATE"), date (YEAR-MONTH-DAY), and price (the data)

```r
tq_get_options()
```

Other possibilities with tidyquant

```
##  [1] "stock.prices"       "stock.prices.japan" "dividends"
##  [4] "splits"             "economic.data"      "quandl"
##  [7] "quandl.datatable"   "tiingo"             "tiingo.iex"
## [10] "tiingo.crypto"      "alphavantager"      "alphavantage"
## [13] "rblpapi"
```

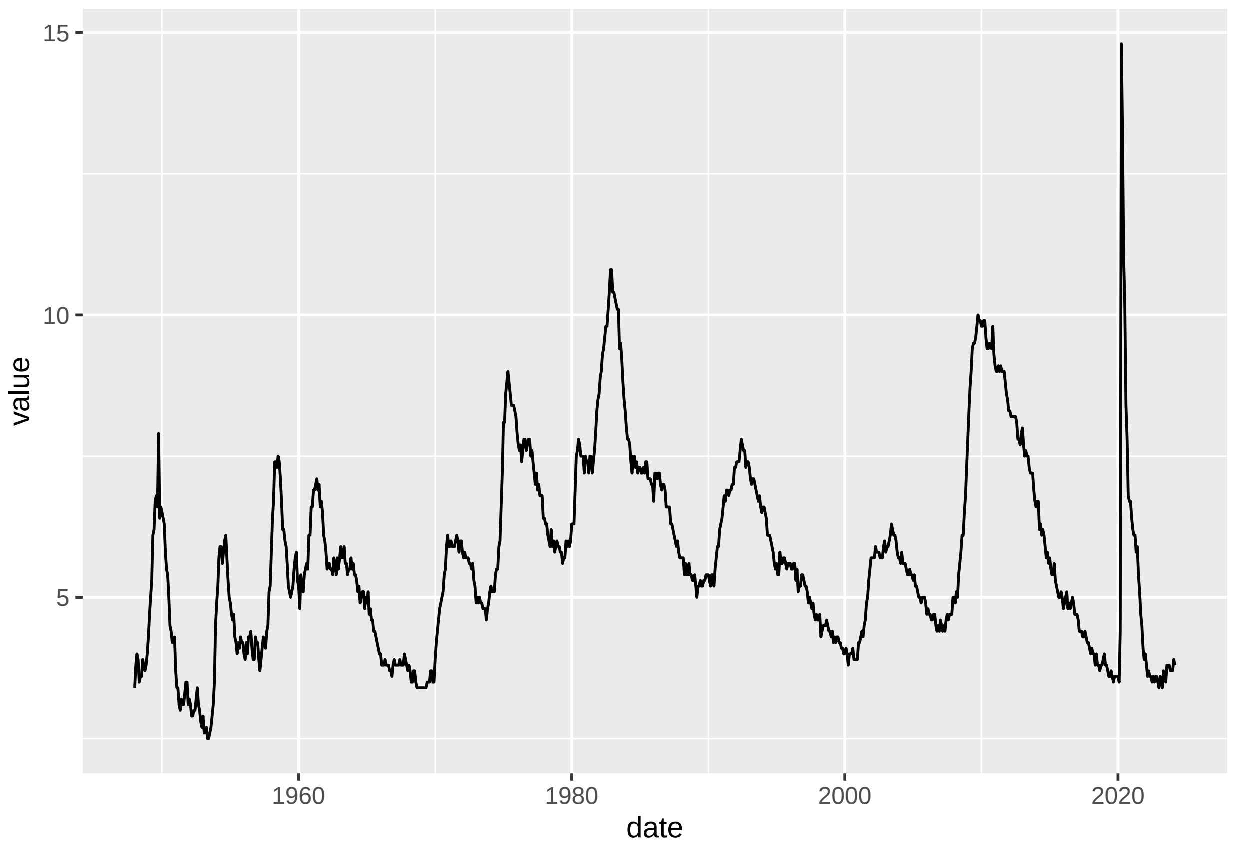
Shortcomings of tidyquant A main shortcoming is that you can only download the data that have series IDs. If you want the UNRATE data to actually be monthly

There is another package that allows us to change FRED data:

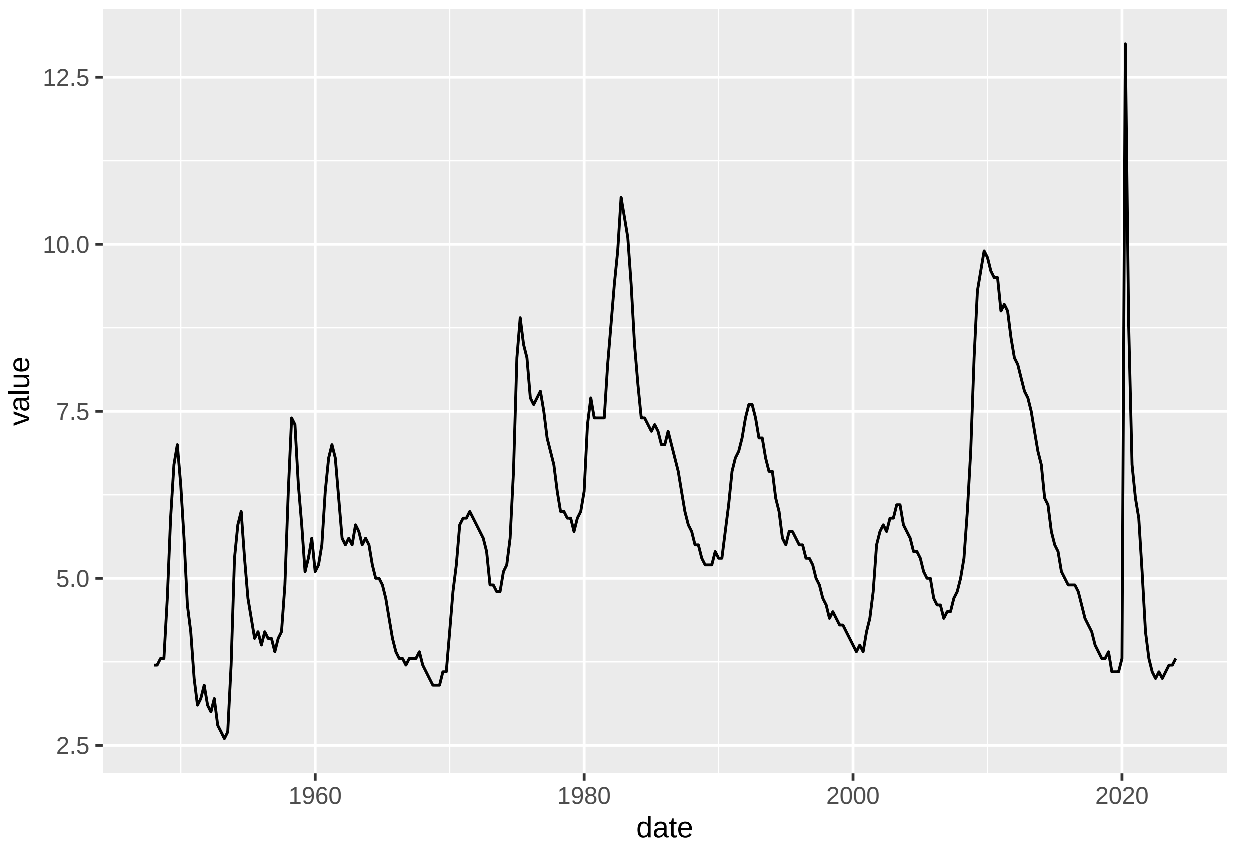
fredr

1. Create / log-in to a FRED account here
2. Get a personal API (Application Programming Interface) here
3. Load the API key:

```r
# Load package
p_load(fredr)

# Let's also do some plotting
p_load(ggplot2)



# Give personal API key
# fredr_set_key("personal")


# Download the UNRATE again
un_data_2 = fredr(
  series_id = "UNRATE"
)

ggplot(un_data_2, aes(x = date, y = value)) +
  geom_line()
```

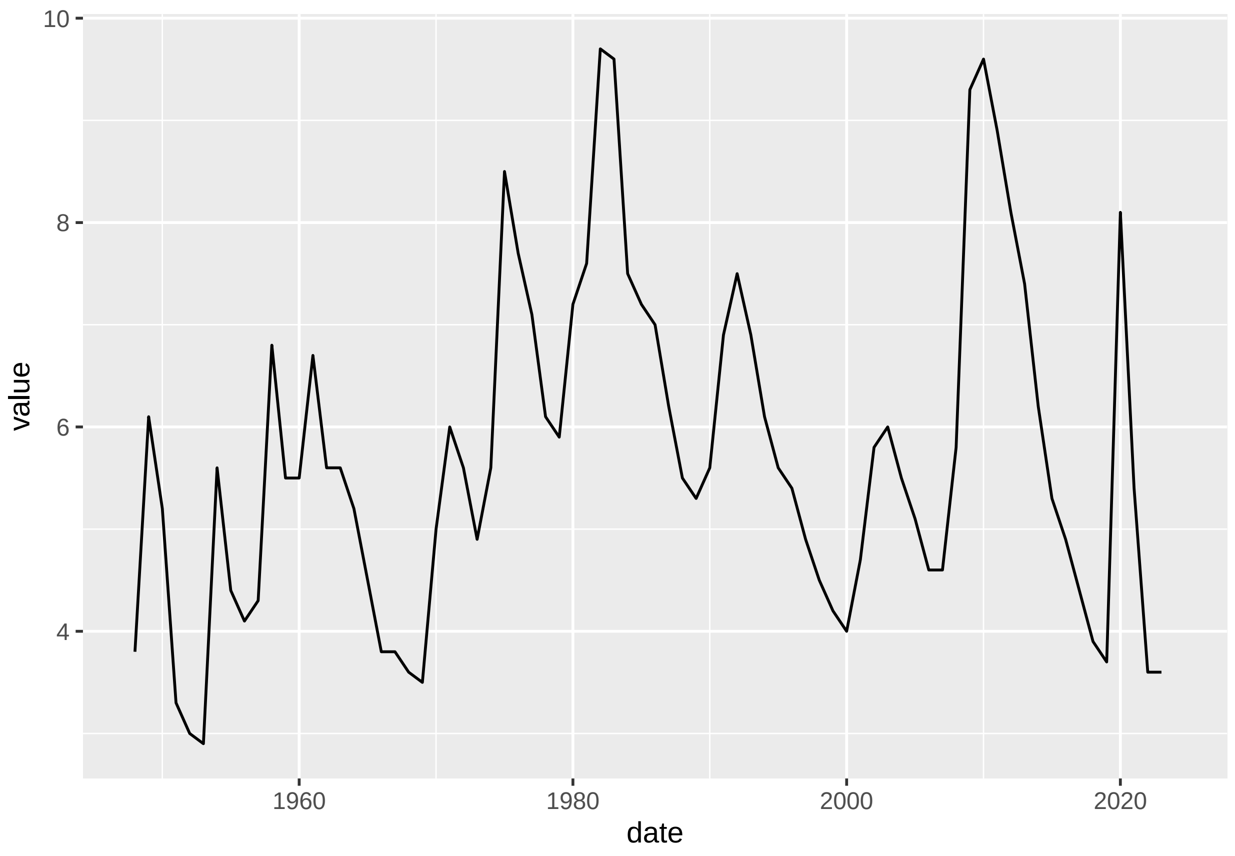


```r
# Now change the frequency to get quarterly or annual data
un_data_q = fredr(
  series_id = "UNRATE",
  frequency = "q" # quarterly
)

ggplot(un_data_q, aes(x = date, y = value)) +
  geom_line()
```



```r
un_data_a = fredr(
  series_id = "UNRATE",
  frequency = "a" # annual
)

ggplot(un_data_a, aes(x = date, y = value)) +
  geom_line()
```

```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```

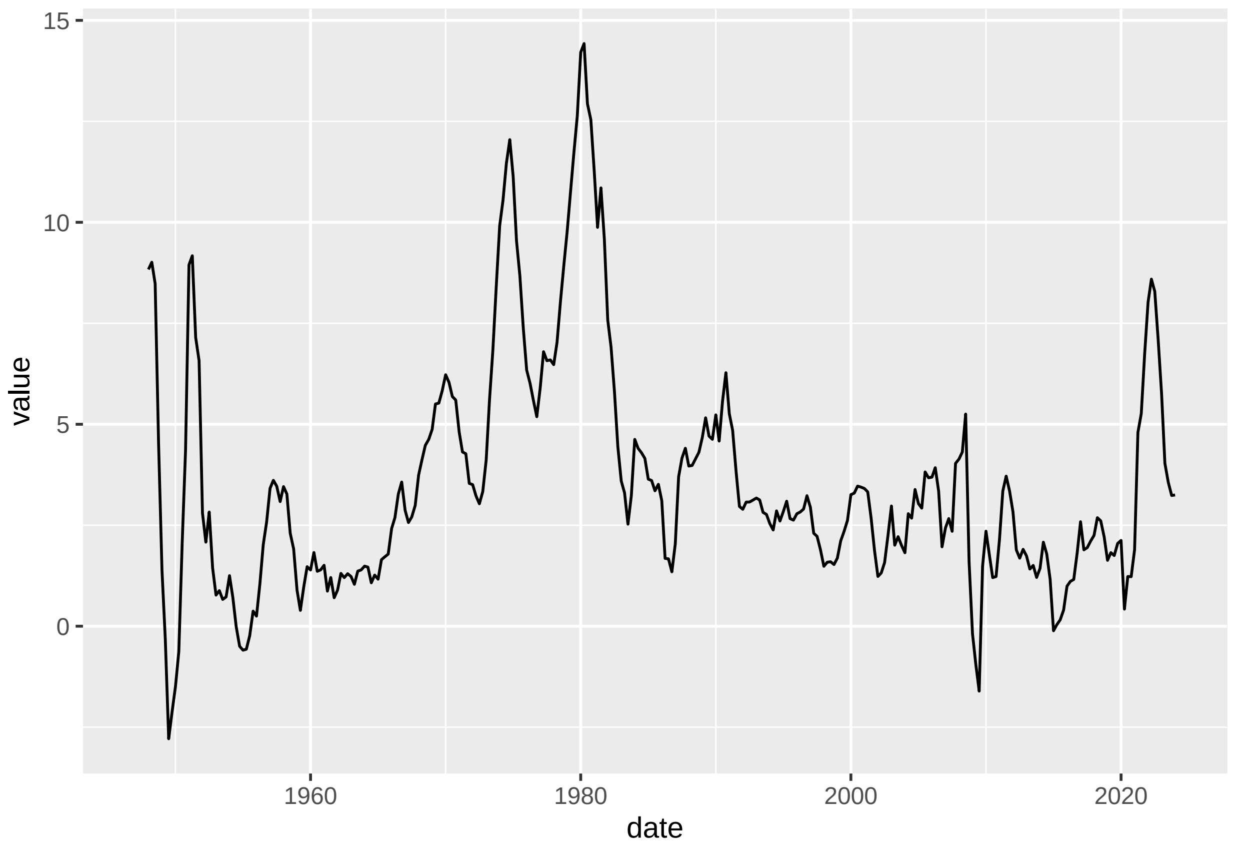


See page 32 of the fredr.pdf to see all of the options for frequency of data

You can also automatically change the *units* of the data. Suppose we want the annual inflation rate from the CPI in quarters:

```r
inflation_data = fredr(
  series_id = "CPIAUCSL",
  frequency = "q", # quarterly
  units = "pc1" # means "percent change from 1 year ago"
)


# Plot it
ggplot(inflation_data, aes(x = date, y = value)) +
  geom_line()
```

```
## Warning: Removed 4 rows containing missing values (`geom_line()`).
```



Data from Wikipedia

"Wikipedia is the best thing ever. Anyone in the world can write anything they want about any subject, so you know you are getting the best possible information." - Michael Scott

We are going to need a new package:

```
# New package: rvest
p_load(rvest)
```

We are going to be looking at data on the largest objects in the solar system from the Wikipedia Page here

```
# First, read the HTML code from the url ( from rvest package)
solar_html = read_html("https://en.wikipedia.org/wiki/List_of_Solar_System_objects_by_size")

# We can look at ALL of the tables on this page by:
# solar_html %>% html_nodes("table")
  # be careful knitting when running this code!


# We want the SECOND table on the page
# solar_html %>% html_nodes("table") %>% .[2]
  # the period allows us to select the item in the list within a pipe


# This is still a list, so make it an html table and grab
# solar_html %>% html_nodes("table") %>%  .[2] %>% html_table()

# Still a list, so grab the first element [[1]] and assign it an object:
```

```r
solar_table = solar_html %>%
  html_nodes("table") %>%
  .[2] %>%
  html_table() %>%
  .[[1]]
```

Looking at this data… it isn't that pretty… If we are to analyze any of this data, we are going to have to do some cleaning!

1. First: get rid of the first and last rows

```r
# gets rid of first
solar_table = solar_table[-1,]

# gets rid of last 37th row
solar_table = solar_table[-37,]
```

2. Let's focus on Volume for now. Let's just look at the Body names, volume in KM3, and type. Notice that we have repeat column names (UH OH!). Let's rename our columns using our technique from last week:

```r
p_load(dplyr)

# First, lets grab the first (body) and fifth (volume) columns:
solar_clean = solar_table %>% select(1, 5)


# Second, rename:
new_names = c("Body", "Volume")

solar_clean = solar_clean %>%
  rename_all(~new_names)
```

3. Get rid of the *numeric* and *letter* footnotes

```r
# remove numeric footnote
solar_clean$Volume = gsub("\\[\\d+\\]", "", solar_clean$Volume)


# The [c] footnote
solar_clean$Volume = gsub("\\[\\w+\\]", "", solar_clean$Volume)


summary(solar_clean)
```

```
##      Body              Volume
##  Length:36          Length:36
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

4. Almost done! Get rid of commas in the numbers and make it a numeric variable

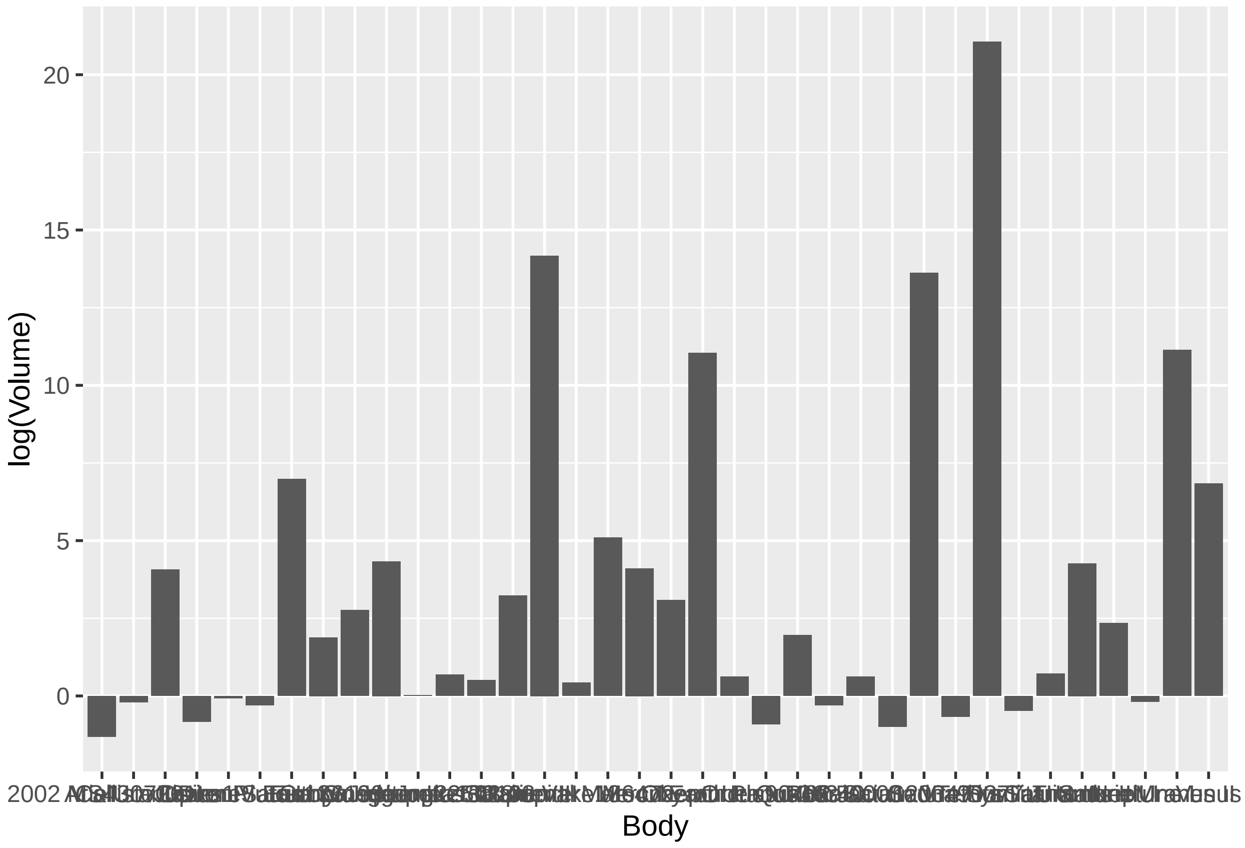
```r
# Make numberic
solar_clean$Volume = as.numeric(gsub(",", "", solar_clean$Volume))


summary(solar_clean)
```

```
##      Body               Volume
##  Length:36          Min.   :0.000e+00
##  Class :character   1st Qu.:1.000e+00
##  Mode  :character   Median :4.000e+00
##                     Mean   :3.921e+07
##                     3rd Qu.:7.300e+01
##                     Max.   :1.409e+09
```

5. Now let's graph it as a histogram: log(Volume)

(Recall that the volume is already in billions of cubic kilometers)

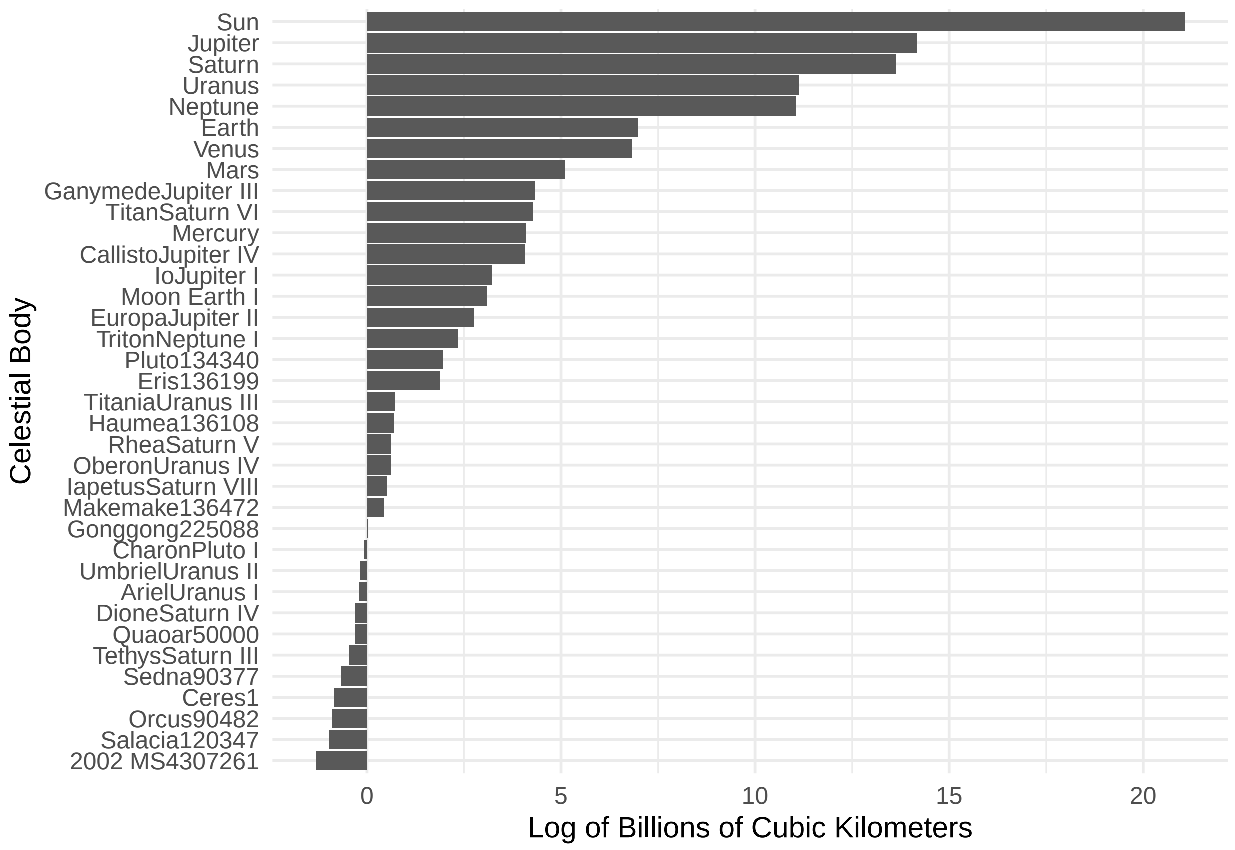
```r
# Graph
ggplot(solar_clean, aes(x = Body, y = log(Volume))) +
  geom_bar(stat="identity")
```



Hurray?

We will get into data visualization more next week, but for now, let's make this look slightly prettier

```r
# Graph (reorder(variable, by what we're ordering))
ggplot(solar_clean, aes(x = reorder(Body,Volume), y = log(Volume))) +
  geom_bar(stat="identity") +
  coord_flip() +
  labs(y = "Log of Billions of Cubic Kilometers", x = "Celestial Body") +
  theme_minimal()
```


Sun
Jupiter
Saturn
Uranus
Neptune
Earth
Venus
Mars
GanymedeJupiter III
TitanSaturn VI
Mercury
CallistoJupiter IV
IoJupiter I
Moon Earth I
EuropaJupiter II
TritonNeptune I
Pluto134340
Eris136199
TitaniaUranus III
Haumea136108
RheaSaturn V
OberonUranus IV
IapetusSaturn VIII
Makemake136472
Gonggong225088
CharonPluto I
UmbrielUranus II
ArielUranus I
DioneSaturn IV
Quaoar50000
TethysSaturn III
Sedna90377
Ceres1
Orcus90482
Salacia120347
2002 MS4307261
Celestial Body
0
5
10
15
20
Log of Billions of Cubic Kilometers