R Part 2: Basic Data Tidying

Emma Garlock

June 12th, 2025

Table of Contents

# Introduction

In this session, we are going to learn some basics about cleaning data in R. The folder for this session is available at <https://tinyurl.com/45vxsawu>.



For this session you will need:

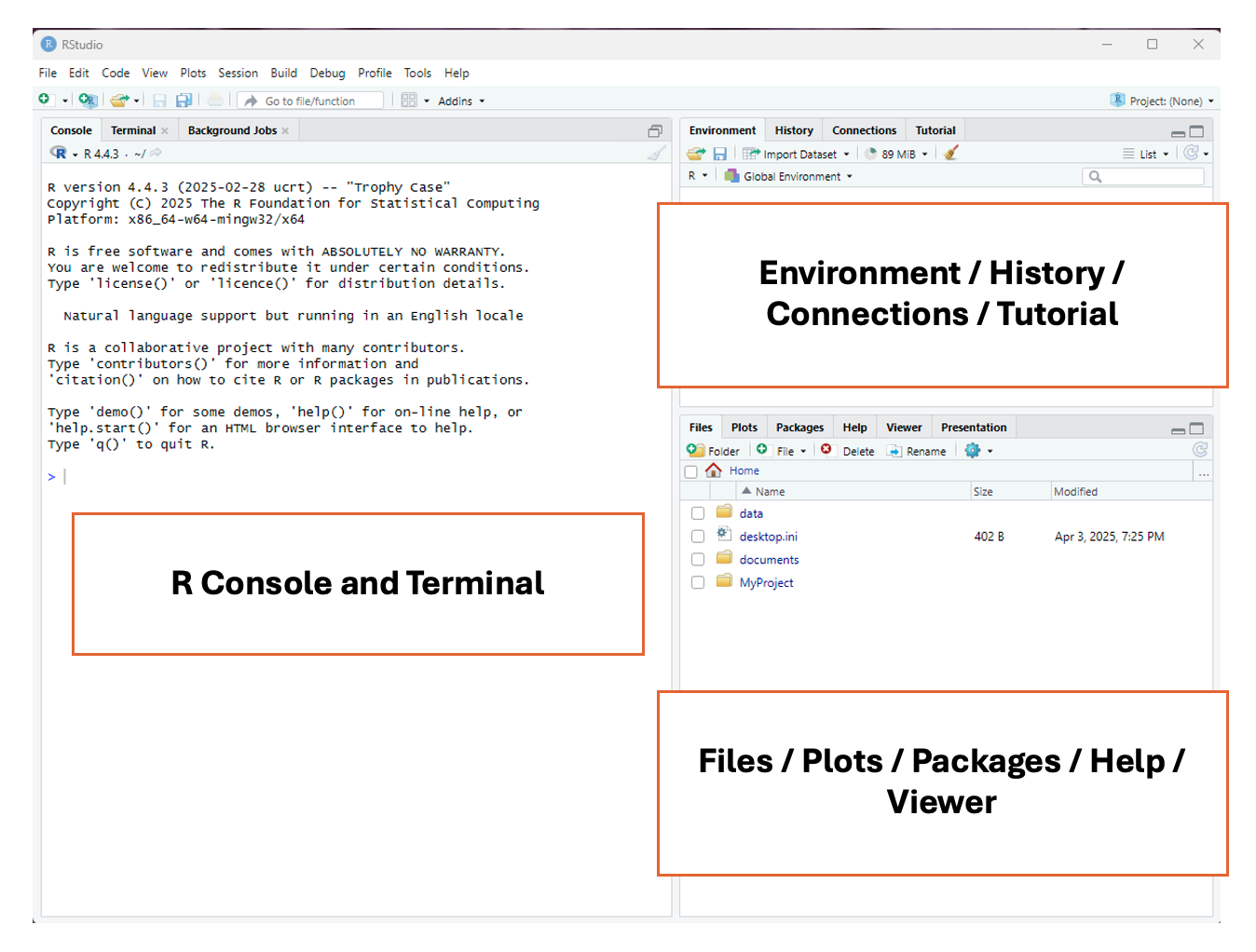
* FileA\_RMarkdown\_uOttawabiblio.rmd
  + *This is the same notebook that I will be showing with the code removed*
  + *It’s not necessary for you to use this file, you can also do it in a completely new notebook or R script*
* data/
  + SciHub\_SampleData.csv
  + SciHubDOI.csv

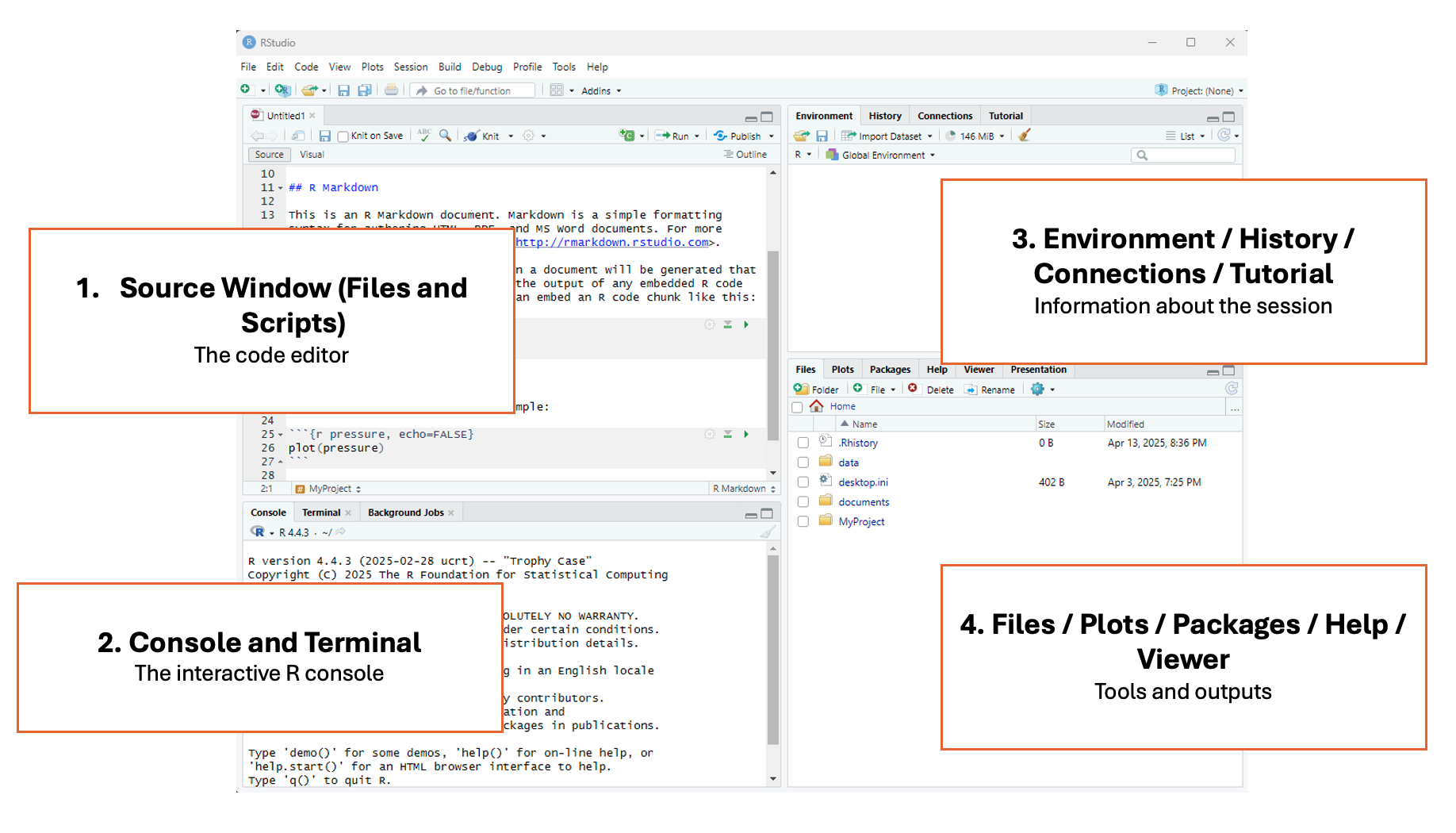
There are other files

* FileB\_MarkDown\_uOttawabiblio.rmd
  + *this is the same file as above, but with the code already there*
* FileB\_MarkDown\_uOttawabiblio.nb.html
  + *this is this the html file of the completed notebook*
* notebook\_images/
  + *this is just the images that are in the notebook*

But first we are going to have a general orientation about R Studio. If you are going through this at a later date, you can watch [this](https://www.youtube.com/watch?v=FIrsOBy5k58) video.

When you first open are you should see this:

 Once you open a file, you should see this.

 The above images are from the RDM Jumpstart Program. They also have introductory lessons on R, which are available [here](https://alliance-rdm-gdr.github.io/rdm-jumpstart/2-ACT-1-RSetup.html).

There’s 3 key features of R

1. R can do operations

125+65

## [1] 190

45\*76

## [1] 3420

8959/32

## [1] 279.9688

1. You can assign values to objects. Then do operations on the objects

x=3   
y=6  
x\*y

## [1] 18

* These values can be characters

test\_string="uOttawaBiblio"  
print(test\_string)

## [1] "uOttawaBiblio"

* It can also be multiple values, these are what we call lists

test\_number\_list=c(2,4,6,7,8,3)  
test\_character\_list=c("Spring","Summer","Fall","Winter")

* They can also be dataframes

df=read.csv("data/testfile.csv")

1. R has functions, and the functions are in packages.

We have seen a function already. print() and read.csv() are baseR functions (aka default). The function is the thing outside the brackets, and you perform the function on the argument, which is inside the bracket.

So, for the example above, the function was print(), and the argument was "test\_string".

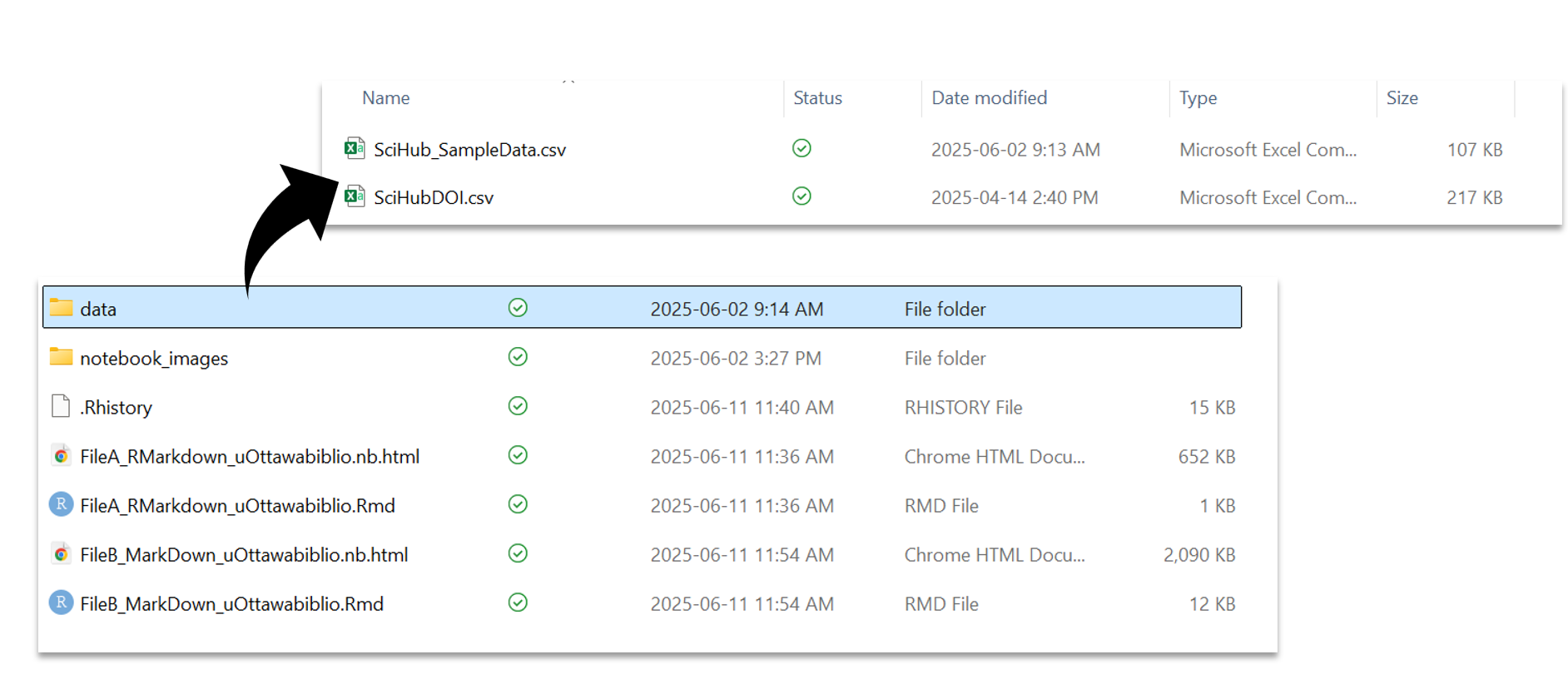
To get extra functions, you need to download packages. Read more about functions and packages [here](https://bookdown.org/nana/intror/install-and-load-packages.html).

# Set Up

## Working Directory

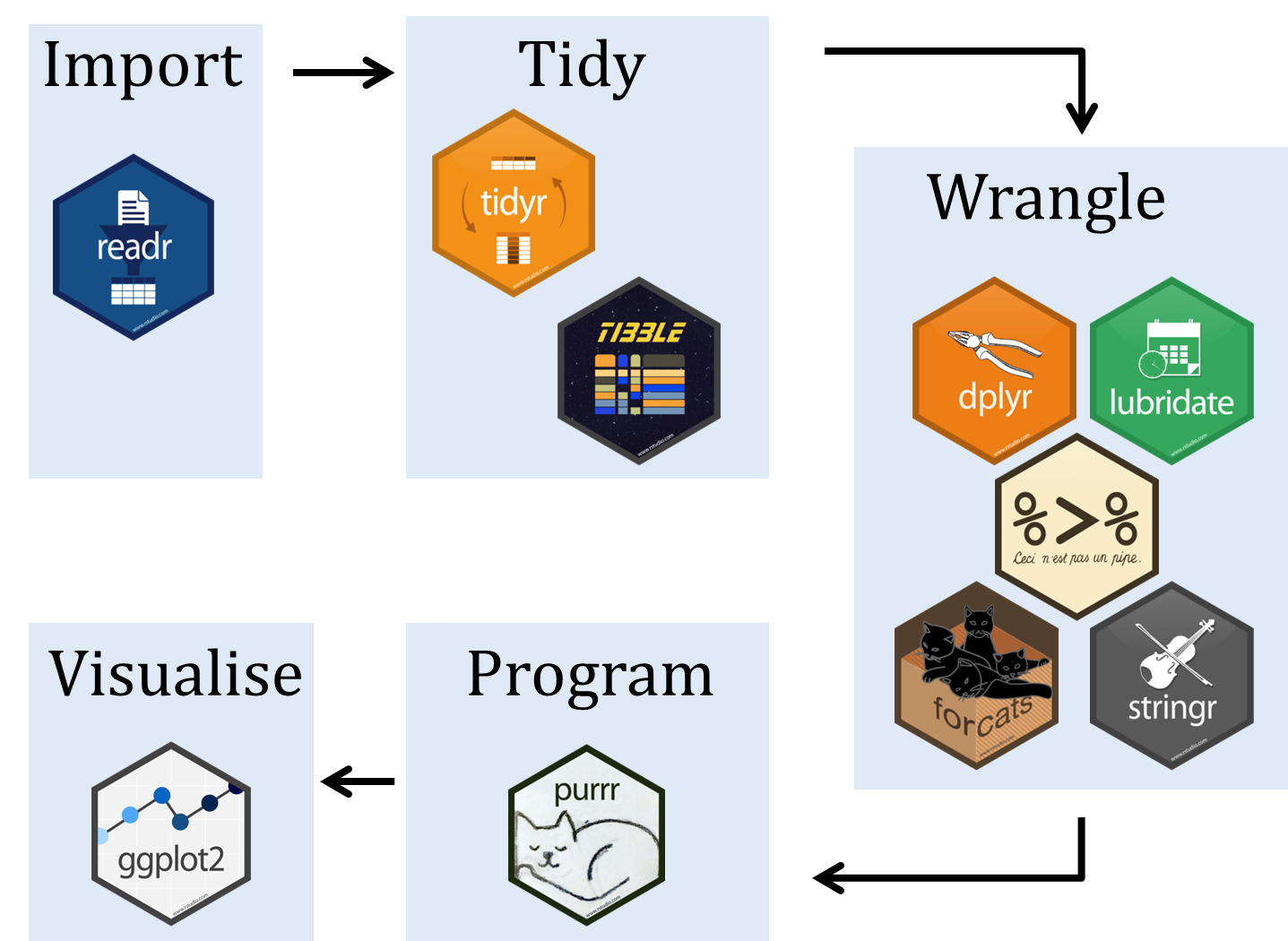
First, we are going to set ourselves up in a working directory.

*Note: if you downloaded the whole folder, and you opened one of the provided files, ignore the advice about where to save things. It should all be organized already*

1. Save the R notebook or R Script file to somewhere that makes sense, this should be the same location where you have the data stored for this session. See the example below. 
2. Select "Session" from the top menu bar, then "Set working directory" then "to Source file location". The directory should now be printed on the top of the console. See the example below.

## Installing Tidyverse

The following examples are going to be done using functions from tidyverse. tidyverse is a collection of packages that contain functions that are so commonly used for analyses, that people decided to just makes sure that you could download these all at once AND that they would be highly inter operable.You can learn more about tidyverse [here](https://r4ds.hadley.nz/data-transform.html)



There are two ways to get a package for the first time, the first is to run install.packages() with the package name in the brackets, the second is to go over to the panel on the lower right, hit the "Packages" tab, then install and type "tidyverse"

You do *not* have to install packages every time, but you *do* need to load them every time using library()

Lets load our package:

#this is how you install using code, this is equivalent to going through the Packages panel. I've commented it out since I don't actually need to install   
#install.packages("tidyverse")   
#Loading the package  
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.4 ✔ readr 2.1.5  
## ✔ forcats 1.0.0 ✔ stringr 1.5.1  
## ✔ ggplot2 3.5.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.3 ✔ tidyr 1.3.1  
## ✔ purrr 1.0.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

## Upload Data

Now, we can load our data, and assign it the name scihub\_df Then take a look at the first few rows using the head() function.There is also a tail() function to see the last rows. For more info on uploading data and the different formats you can use, check out [this](https://intro2r.com/importing-data.html)

I have elected to locate my data by specifying a file path. You could also do it like scihub\_df=read.csv(file.choose()) to open up a file explorer.

#upload the dataset, its located in the data file  
scihub\_df=read.csv("data/SciHub\_SampleData.csv")   
#show the first 6 rows  
head(scihub\_df)

## Timestamp DOI IP.identifier  
## 1 2017-06-26 21:46:59 10.1002/14651858.CD003392.pub2 7809386  
## 2 2017-09-07 22:58:24 10.1080/14786430601032386 1358764  
## 3 2017-05-02 09:59:00 10.1021/la501330j 6039317  
## 4 2017-07-09 09:07:20 10.1063/1.4913415 5997924  
## 5 2017-05-03 08:40:56 10.1021/jp809992g 858831  
## 6 2017-05-03 22:11:34 10.1021/ja025109g 858831  
## User.identifier Country.according.to.GeoIP City.according.to.GeoIP Latitude  
## 1 16866302 Canada Boucherville 45.59137  
## 2 33577860 Canada Toronto 43.65323  
## 3 9158745 Canada Toronto 43.65323  
## 4 19896736 Canada Toronto 43.65323  
## 5 9278539 Canada Toronto 43.65323  
## 6 9370108 Canada Toronto 43.65323  
## Longitude  
## 1 -73.43641  
## 2 -79.38318  
## 3 -79.38318  
## 4 -79.38318  
## 5 -79.38318  
## 6 -79.38318

## Rename Columns

Looks good, but from experience, those titles column names might make life difficult later, lets rename them to something without spaces. We can then check to make sure the names were changed properly and we didn’t mess anything up.

For more examples of how to rename columns check out [this](https://sparkbyexamples.com/r-programming/rename-column-in-r/) link.

We can then use the names() function to see what the names of the columns are.

#change the names of scihub\_df. The list needs to be the same length as the number of columns   
colnames(scihub\_df)=c("Timestamp",  
 "DOI",  
 "IP\_ID",  
 "User\_ID",  
 "Country\_GeoIP",  
 "City\_GeoIP",  
 "Latitude",  
 "Longitude")  
#just print the names of columns to confirm they are the new names   
names(scihub\_df)

## [1] "Timestamp" "DOI" "IP\_ID" "User\_ID"   
## [5] "Country\_GeoIP" "City\_GeoIP" "Latitude" "Longitude"

# Basic Tidying and Analyses

## Selecting Columns

tidyverse uses something called "pipes", which look like %>% or |>, which tells R to automatically use the last output as the input for the next function. Lets see an example.

Let’s say we only want a subset of the columns in "scihub\_df", not all 8. We can use the select() function to get those

#create new dataframe based on scihub\_df, just selecting the 3 columns we cant   
scihub\_df\_reduced=scihub\_df%>%  
 select(Timestamp,DOI,City\_GeoIP)#just selecting these three columns   
#preview the first 6 rows so we can see if it did what we think it did   
head(scihub\_df\_reduced)

## Timestamp DOI City\_GeoIP  
## 1 2017-06-26 21:46:59 10.1002/14651858.CD003392.pub2 Boucherville  
## 2 2017-09-07 22:58:24 10.1080/14786430601032386 Toronto  
## 3 2017-05-02 09:59:00 10.1021/la501330j Toronto  
## 4 2017-07-09 09:07:20 10.1063/1.4913415 Toronto  
## 5 2017-05-03 08:40:56 10.1021/jp809992g Toronto  
## 6 2017-05-03 22:11:34 10.1021/ja025109g Toronto

## Filtering Rows

We could also go the other way, and only take certain rows. Let’s say we only wanted rows where the city was “Ottawa”, we can use the filter() function to find those. We can then use the print() function so see our new dataframe in the console.

*Note: this is case sensitive*

#making a df that is just for Ottawa  
scihub\_df\_ottawa=scihub\_df%>% #using the same original dataset  
 filter(City\_GeoIP=="Ottawa") #select only the rows with "Ottawa" (case sensitive) int he City\_GeoIP column  
#print the whole dataset since it's small  
print(scihub\_df\_ottawa)

## Timestamp DOI IP\_ID User\_ID Country\_GeoIP  
## 1 2017-03-26 03:00:42 10.2307/1547968 4587502 6727298 Canada  
## 2 2017-07-21 16:20:54 10.1017/S1049096516001633 10172999 23057469 Canada  
## City\_GeoIP Latitude Longitude  
## 1 Ottawa 45.42153 -75.69719  
## 2 Ottawa 45.42153 -75.69719

## Summarizing Groups

There are a lot of basic things we can do. Lets just try getting a summary of how many time each city appears in the dataset. We’re going to use the "scihub\_df\_reduced" set (the one where we used select() to pick cetain columns).

We’re going to start by using the group\_by() function. The group\_by() functions creates groups based on a certain column, and then all subsequent operations (eg. summing, averaging, counting) are done on a *per group* basis. Learn more about group\_by() [here](https://r4ds.hadley.nz/data-transform.html).

city\_summary=scihub\_df\_reduced%>% #using the dataset with 3 columns   
 group\_by(City\_GeoIP)%>% #make the groups based on city   
 count() #count how many went into each group   
#see first 6 rows (they are automatically sorted alphabetically by grouping variable (aka City\_GeoIP))  
head(city\_summary)

## # A tibble: 6 × 2  
## # Groups: City\_GeoIP [6]  
## City\_GeoIP n  
## <chr> <int>  
## 1 Ajax 12  
## 2 Baddeck 2  
## 3 Baie-Comeau 2  
## 4 Beaconsfield 15  
## 5 Boucherville 10  
## 6 Bracebridge 1

If you want to do a little sanity check, the sum of everything in column n should be 1000.  
We can double check like this using the sum() function:

sum(city\_summary$n)

## [1] 1000

## Fixing Typos

Did anyone notice anything about the summarized data?

Yes, we have two different spellings for Montréal.

Lets fix it.

We’re not going to actually make a new dataset, we’re just going to edit what we already did. By adding a new line *before* the group\_by() where we use a function called mutate(). mutate() is a very versatile function and can be used for a lot of different applications. You can read more about that [here](https://bookdown.org/yih_huynh/Guide-to-R-Book/mutate.html).

One thing you can do with mutate() is called a "nested function" this is where you have a function *inside* another function. In this case we are going to use the replace() function.

The replace() function is formatted like this: replace("column that we need to edit","what values in the column need to be edited,"What we want the new value to be")

*Note: there are a lot of different ways to fix typos in data sets, this is just one of many.*

city\_summary=scihub\_df\_reduced%>% #3 column dataset   
 mutate(City\_GeoIP = replace(City\_GeoIP, City\_GeoIP == "Montréal", "Montreal"))%>% #fixing the error   
 group\_by(City\_GeoIP)%>%#set groups based on the city, same process as above :)   
 count()

If you remember, before we had 76 observations, now we have 75.

## Dates

Notice that we have a timestamp column, this has both date and the time. Could be useful, but maybe we just want the date. To do this, we are going to load a new package, called lubridate which is specifically used for working with date formats.

library(lubridate) #loading a package

We actually have a few ways we could do this.  
1. Use lubridate functions  
2. Separate using the space as a delimiter.  
3. Extract the first 10 characters of each row into it’s own column

Let’s do the 1st option. We are going to do another nested function with mutate() using the ymd\_hms() function from lubridate

scihub\_df\_reduced\_date=scihub\_df%>% #start with the original dataset  
 select(Timestamp,DOI,City\_GeoIP)%>% #select the columns we need   
 mutate(Timestamp=ymd\_hms(Timestamp))%>% #make sure the time is interpreted in the correct format   
 mutate(Date=date(Timestamp)) #extract the date   
  
head(scihub\_df\_reduced\_date) #preview the top 6

## Timestamp DOI City\_GeoIP Date  
## 1 2017-06-26 21:46:59 10.1002/14651858.CD003392.pub2 Boucherville 2017-06-26  
## 2 2017-09-07 22:58:24 10.1080/14786430601032386 Toronto 2017-09-07  
## 3 2017-05-02 09:59:00 10.1021/la501330j Toronto 2017-05-02  
## 4 2017-07-09 09:07:20 10.1063/1.4913415 Toronto 2017-07-09  
## 5 2017-05-03 08:40:56 10.1021/jp809992g Toronto 2017-05-03  
## 6 2017-05-03 22:11:34 10.1021/ja025109g Toronto 2017-05-03

## The Separate function

Lets try it using the separate() function to get the time (Option 2)

scihub\_df\_reduced\_time=scihub\_df%>% #same selection procedure as above   
 select(Timestamp,DOI,City\_GeoIP)%>%  
 separate(Timestamp, c("Date", "Time"), " ") #separate the date and time based on the space (the blank in between the quotes) and call the two new columns "Date" and "time"   
  
head(scihub\_df\_reduced\_time)

## Date Time DOI City\_GeoIP  
## 1 2017-06-26 21:46:59 10.1002/14651858.CD003392.pub2 Boucherville  
## 2 2017-09-07 22:58:24 10.1080/14786430601032386 Toronto  
## 3 2017-05-02 09:59:00 10.1021/la501330j Toronto  
## 4 2017-07-09 09:07:20 10.1063/1.4913415 Toronto  
## 5 2017-05-03 08:40:56 10.1021/jp809992g Toronto  
## 6 2017-05-03 22:11:34 10.1021/ja025109g Toronto

There is also a paste() function in R. It’s very similar to the concatenate in Excel, and you can learn more about it [here](https://www.digitalocean.com/community/tutorials/paste-in-r).

# Bonus Content if we get time

## Joins

So, we have this information about DOI, but what if we want more information? Luckily we have the title and other publication information available from Zotero, and we can export a csv from Zotero and “join” it to our existing dataset.

This csv is going to have a lot of columns. But maybe we only want DOI (Column 9), Title (Column 5) and Publication Year (Column 3). Before when we selected, we used the names of the columns, but we can also select based on the column number.

Notice that we were able to pipe the read.csv immediately into the select()

zotero=read.csv("data/SciHubDOI.csv")%>%  
 select(9,5,3) #selecting based on position rather than name   
head(zotero)

## DOI  
## 1 10.1021/jp809992g  
## 2 10.1093/beheco/arx008  
## 3 10.1149/1.2069301  
## 4 10.1002/dap.30253  
## 5 10.1126/science.aaa9092  
## 6 10.1002/anie.201605430  
## Title  
## 1 Spectroscopic Studies of Pristine and Fluorinated Nano-ZrO<sub>2</sub> in Photostimulated Heterogeneous Processes  
## 2 Why is the giant panda black and white?  
## 3 Solid‐State NMR Studies of Ions in Protective Coatings: II . Lithium and Cesium Ions in Polybutadiene Coatings  
## 4 How to learn and use your institution's student voting rates  
## 5 Boreal forest health and global change  
## 6 From Alkanes to Carboxylic Acids: Terminal Oxygenation by a Fungal Peroxygenase  
## Publication.Year  
## 1 2009  
## 2 2017  
## 3 1992  
## 4 2016  
## 5 2015  
## 6 2016

Now, lets join the datasets together. We are using left\_join() here, but there are lots of different types of joins that you can learn more about [here](https://r4ds.hadley.nz/joins.html#sec-mutating-joins).

scihub\_zotero=scihub\_df\_reduced%>%  
 left\_join(zotero,by="DOI") #telling it to join the dataset zotero by the values in column DOI   
  
head(scihub\_zotero)

## Timestamp DOI City\_GeoIP  
## 1 2017-06-26 21:46:59 10.1002/14651858.CD003392.pub2 Boucherville  
## 2 2017-09-07 22:58:24 10.1080/14786430601032386 Toronto  
## 3 2017-05-02 09:59:00 10.1021/la501330j Toronto  
## 4 2017-07-09 09:07:20 10.1063/1.4913415 Toronto  
## 5 2017-05-03 08:40:56 10.1021/jp809992g Toronto  
## 6 2017-05-03 22:11:34 10.1021/ja025109g Toronto  
## Title  
## 1 Breast stimulation for cervical ripening and induction of labour  
## 2 Adsorption characteristics of parent and copper-sputtered RD silica gels  
## 3 Micropatterned Ferrocenyl Monolayers Covalently Bound to Hydrogen-Terminated Silicon Surfaces: Effects of Pattern Size on the Cyclic Voltammetry and Capacitance Characteristics  
## 4 Conduction of molecular electronic devices: Qualitative insights through atom-atom polarizabilities  
## 5 Spectroscopic Studies of Pristine and Fluorinated Nano-ZrO<sub>2</sub> in Photostimulated Heterogeneous Processes  
## 6 Structural Basis for BABIM Inhibition of Botulinum Neurotoxin Type B Protease [ <i>J. Am. Chem. Soc.</i> <b>2000</b> , <i>122</i> , 11268−11269].  
## Publication.Year  
## 1 2005  
## 2 2007  
## 3 2014  
## 4 2015  
## 5 2009  
## 6 2002

## Pivots

We’re going to combine a few things we have seen so far. 1. making lists.  
2. group\_by(), but this time we will have TWO groupings.  
3. filter, but this time with a list of options and not just one.

We’re going to start with our reduced set. Let’s refresh on what it looks like.

head(scihub\_df\_reduced)

## Timestamp DOI City\_GeoIP  
## 1 2017-06-26 21:46:59 10.1002/14651858.CD003392.pub2 Boucherville  
## 2 2017-09-07 22:58:24 10.1080/14786430601032386 Toronto  
## 3 2017-05-02 09:59:00 10.1021/la501330j Toronto  
## 4 2017-07-09 09:07:20 10.1063/1.4913415 Toronto  
## 5 2017-05-03 08:40:56 10.1021/jp809992g Toronto  
## 6 2017-05-03 22:11:34 10.1021/ja025109g Toronto

We have 3 columns: Timestamp, DOI and City\_GeoIP. But maybe we want to see how often each DOI comes up in each city and the organize the information so we have 1 column for each city.

For the sake of not creating a huge dataset, we’re going to only include certain cities. Lets define those using a list.

cities\_list=c("Ottawa","Toronto","Montreal","Burnaby")

Now we know what we’re working with, we can string everything together. The final line is pivot\_wider, it will be easier to explain what it does after you have seen the final product.

scihub\_pivot=scihub\_df\_reduced%>%  
 group\_by(City\_GeoIP,DOI)%>% #group by city and DOI, so we'll get a summary of the doi count per city   
 count()%>%  
 filter(City\_GeoIP %in% cities\_list)%>% #filter, but only keep values that appear in cities\_list  
 pivot\_wider(id\_cols=DOI,names\_from=City\_GeoIP,values\_from=n)#here is the pivot, we say that the rows should be based on DOI, the new column names are going to be the city, and the values in the cells are the counts of that DOI in that city  
  
head(scihub\_pivot)

## # A tibble: 6 × 5  
## # Groups: DOI [6]  
## DOI Burnaby Montreal Ottawa Toronto  
## <chr> <int> <int> <int> <int>  
## 1 10.1021/jp011934s 1 NA NA 1  
## 2 10.1126/science.197.4307.967 1 NA NA 3  
## 3 10.1002/wcc.81 NA 2 NA 4  
## 4 10.1016/0006-8993(77)90423-1 NA 1 NA NA  
## 5 10.1016/S2214-109X(16)30188-7 NA 3 NA NA  
## 6 10.1037/a0017364 NA 1 NA NA