3: Data Exploration

Environmental Data Analytics | John Fay and Luana Lima | Developed by Kateri Salk

Fall 2022

Lesson Objectives

1. Set up a data analysis session in RStudio
2. Import and explore datasets in R
3. Apply data exploration skills to a real-world example dataset

Best Practices in R

In many situations in data analytics, you may be expected to work from multiple computers or share projects among multiple users. A few general best practices will avoid common pitfalls related to collaborative work.

Set your working directory

A session in RStudio will always function by mapping to a specific folder in your computer, called the *working directory*. All navigation between folders and files will happen relative to this working directory. When you open an R project, your working directory will automatically set to the folder that holds the project file. If you open an R script or RMarkdown document directly by double-clicking the file, your working directory will automatically set to the folder that holds that file. It is a good idea to note with a comment at the top of your file which working directory you intend the user to designate.

In this course, we will always open the R project file for the course, and additional navigation of the working directory will happen from that folder. To check your working directory, use the following R command:

```r
# Working directory should be set to the parent folder for the Environmental Data Analytics Course, i.e

getwd()
```

```
## [1] "/home/guest/R/EDA-Fall2022/Lessons"
```

If your working directory is not set to the folder you want, you have several options. The first is to directly code your working directory. You may do this by defining an absolute file path (below). What are the pitfalls of using an absolute file path?

```r
# Absolute file path is commented out
#setwd("/home/guest/EDA-Fall2022")
```

You may change your working directory without coding by going to the Session menu in RStudio and navigating to the Set Working Directory tab. From there, you may select from a series of options to reset your working directory.

Another option is to use the R package `here`. We will not be using this option in class, but it is growing quite popular among R users. A more detailed description and rationale can be found here: https://github.com/jennybc/here_here.

Load your packages

At the top of your R scripts, you should load any packages that need to be used for that R script. A common issue that arises is that packages will be loaded in the middle of the code, making it difficult to run specific chunks of code without scrolling to make sure all necessary packages are loaded. For example, the tidyverse package is one that we will use regularly in class.

The Packages tab in the notebook stores the packages that you have saved in your system. A checkmark next to each package indicates whether the package has been loaded into your current R session. Given that R is an open source software, users can create packages that have specific functionalities, with complicated code "packaged" into a simple commands.

If you want to use a specific package that is not in your library already, you need to install it. You can do this in two ways:

1. Click the install button in the packages tab. Type the package name, which should autocomplete below (case matters). Make sure to check "install dependencies," which will also install packages that your new package uses.
2. Type `install.packages("packagename")` into your R chunk or console. It will then appear in your packages list. You only need to do this once.

If a package is already installed, you will need to load it every session. You can do this in two ways:

1. Click the box next to the package name in the Packages tab.
2. Type `library(packagename)` into your R chunk or console.

```r
# We will use the packages dplyr and ggplot2 regularly.
#install.packages("dplyr")
#install.packages("ggplot2")
# comment out install commands, use only when needed and re-comment

#library(dplyr)
#library(ggplot2)

# Some packages are umbrellas under which other packages are loaded
#install.packages("tidyverse")
#library(tidyverse)
```

Question: What happens in the console when you load a package?

> Answer:

Import your datasets

Datasets can be imported into R. Good data practices dictate that raw data (from yourself or others) should not be changed and re-saved within the spreadsheet, but rather the data should be changed with reproducible techniques and saved as a new file. Note: data should be saved in nonproprietary formats, namely .csv or .txt files rather than .xls or .xlsx files.

To read in a data file, you may specify a file path with an *absolute* or a *relative* file path. As above with your working directory, it is a better practice to use a relative directory. To navigate a relative file path, use `./` followed by the tab key to navigate forward in the folder structure, and use `../` followed by the tab key to navigate back out of the folder structure. For example, this lesson is located in the "Lessons" folder, and we need to navigate into the "Data" folder. After clicking the correct folder, use `/` and press tab again to continue the process.

You may also import datasets from the Files tab, but this is not recommended since this is not reproducible.

Commons functions to import datasets and store as data frames are *read.table()*, *read.csv()*, *read.xlsx()*. Useful inputs/arguments are described below.

- *file =* : use this input to point to your data file. If it's on the same folder as your .Rmd then you only need to write the file name. But if it's on another folder you need to point to the path were file is located;
- *header =* : if your file has a header you should set this to TRUE, o.w. FALSE;
- *skip =* : if your file has rows explaining the data or any other rows on the top that need to be skipped you should just set skip to be equal to the number of row that should be skipped before reading the data. Mote that if header=TRUE, you should not skip the row with the header. The defaul is *skip=0*;
- *dec =* : define *dec=".."* or *dec=","* depending on how it's defined on your set. The default is ".".

```
# Absolute file path (not recommended)
#read.csv("/home/guest/EDA-Fall2022/Data/Raw/USGS_Site02085000_Flow_Raw.csv", stringsAsFactors = TRUE)

#must include stringsasfactors = true when inputting a dataset

# Relative file path (friendly for users regardless of machine)
#USGS.flow.data <- read.csv("./Data/Raw/USGS_Site02085000_Flow_Raw.csv")

# What happens if we don't assign a name to our imported dataset?
#read.csv("./Data/Raw/USGS_Site02085000_Flow_Raw.csv")

# Another option is to choose with your browser
# read.csv(file.choose())

# To import .txt files, use read.table rather than read.csv
#read.table()
```

EXPLORE YOUR DATASET

Take a moment to read through the README file associated with the USGS dataset on discharge at the Eno River. Where can you find this file? How does the placement and information found in this file relate to the best practices for reproducible data analysis? > ANSWER: the file is under metadata. it is placed in a separate file to ensure that the data is safe.

```
#View(USGS.flow.data)
# Alternate option: click on data frame in Environment tab

#class(USGS.flow.data)
#colnames(USGS.flow.data)

# Rename columns
#colnames(USGS.flow.data) <- c("agency_cd", "site_no", "datetime",
                  #            "discharge.max", "discharge.max.approval",
                   #           "discharge.min", "discharge.min.approval",
                    #          "discharge.mean", "discharge.mean.approval",
                     #         "gage.height.max", "gage.height.max.approval",
                      #        "gage.height.min", "gage.height.min.approval",
                       #       "gage.height.mean", "gage.height.mean.approval")
#str(USGS.flow.data)
#just show data frame and all objects on it
#dim(USGS.flow.data)
#short for dimmension -- tells you how many rows and collumns
#length(USGS.flow.data)
```

```
#how many columns you have

#head(USGS.flow.data)
#shows the beginning of your data -- will show first six unless you specify
#head(USGS.flow.data, 10)
#tail(USGS.flow.data, 5)
#USGS.flow.data[30000:30005, c(3, 8, 14)]
#can specify which specific chunk you want to interogate
#class(USGS.flow.data$datetime)
#class(USGS.flow.data$discharge.mean)
#class(USGS.flow.data$gage.height.mean)

#summary(USGS.flow.data)  #could point to column only with $
#summary just gives statistical overview
```

What happened to blank cells in the spreadsheet when they were imported into R? > Answer: they are filled with N/A

Adjusting Datasets

Removing NAs

Notice in our dataset that our discharge and gage height observations have many NAs, meaning no measurement was recorded for a specific day. In some cases, it might be in our best interest to remove NAs from a dataset. Removing NAs or not will depend on your research question.

```
#summary(USGS.flow.data$discharge.mean)
#summary(USGS.flow.data$gage.height.mean)
```

Question: What types of research questions might make it favorable to remove NAs from a dataset, and what types of research questions might make it favorable to retain NAs in the dataset?

> Answer: If you want to average the data that you collected (just a list), the NAs should be removed so they do not draw the average down. If you want to know the span of time that values were collected for (time series analysis), you want to keep them to mark the times when no data was collected.

```
#USGS.flow.data.complete <- na.omit(USGS.flow.data)
#dim(USGS.flow.data)
#dim(USGS.flow.data.complete)

#mean(USGS.flow.data.complete$discharge.mean)
#sd(USGS.flow.data.complete$discharge.mean)
#summary(USGS.flow.data.complete$discharge.mean)
```

Formatting dates

R will often import dates as factors or characters rather than dates. To fix, this we need to tell R that it is looking at dates. We also need to specify the format the dates are in. By default, if you don't provide a format, R will attempt to use %Y-%m-%d or %Y/%m/%d as a default. Note: if you are working collaboratively in an international setting, using a year-month-day format in spreadsheets is the least ambiguous of date formats. Make sure to check whether month-day-year or day-month-year is used in an ambiguously formatted spreadsheet.

Formatting of dates in R:

%d day as number (0-31) %m month (00-12, can be e.g., 01 or 1) %y 2-digit year %Y 4-digit year %a abbreviated weekday %A unabbreviated weekday %b abbreviated month %B unabbreviated month

In some cases when dates are provided as integers, you may need to provide an origin for your dates. Beware: the "origin" date for Excel (Windows), Excel (Mac), R, and MATLAB all have different origin dates. Google this if it comes up.

```
#help(as.Date)

# Adjust date formatting [to be done in lab session]
# Write code for three differtent date formats.
# An example is provided to get you started.
# (code must be uncommented)
#today <- Sys.Date()
#format(today, format = "%B")
#format(today, format = "")
#format(today, format = "")
#format(today, format = "")

#USGS.flow.data$datetime <- as.Date(USGS.flow.data$datetime, format = "%m/%d/%y")
```

Note that for every date prior to 1969, R has assigned the date in the 2000s rather than the 1900s. This can be fixed with an `ifelse` statement inside a function. Run through the code below and write what is happening in the comment above each line.

```
#
#USGS.flow.data$datetime <- format(USGS.flow.data$datetime, "%y%m%d")

#
#create.early.dates <- (function(d) {
#       paste0(ifelse(d > 191226,"19","20"),d)
 #      })
#d is the input
#USGS.flow.data$datetime <- create.early.dates(USGS.flow.data$datetime)

#
#USGS.flow.data$datetime <- as.Date(USGS.flow.data$datetime, format = "%Y%m%d")
```

Saving datasets

We just edited our raw dataset into a processed form. We may want to return to this processed dataset later, which will be easier to do if we save it as a spreadsheet.

```
#write.csv(USGS.flow.data, file = "./Data/Processed/USGS_Site02085000_Flow_Processed.csv", row.names=FA
```

Tips and Tricks

###Packages

- The command `require(packagename)` will also load a package, but it will not give any error or warning messages if there is an issue.
- You may be asked to restart R when installing or updating packages. Feel free to say no, as this will obviously slow your progress. However, if the functionality of your new package isn't working properly, try restarting R as a first step.
- If asked "Do you want to install from sources the packages which needs compilation?", type `yes` into the console.

- You should only install packages once on your machine. If you store `install.packages` in your R chunks/scripts, comment these lines out.
- Update your packages regularly!

Knitting

- In the Knit menu in the Editor, you will need to specify whether your knit directory should be the document directory or the project directory. If your document is not knitting correctly, try switching between the document directory and project directory as a first troubleshooting option.

Spreadsheets

*Files should be saved as .csv or .txt for easy import into R. Note that complex formatting, including formulas in Excel, are not saved when spreadsheets are converted to comma separated or text formats (i.e., values alone are saved).

*The first row is reserved for column headers.

*A secondary row for column headers (e.g., units) should not be used if data are being imported into R. Incorporate units into the first row column headers if necessary.

*Short names are preferred for column headers, to the extent they are informative. Additional information can be stored in comments within R scripts and/or in README files.

*Spaces in column names will be replaced with a `.` when imported into R. When designing spreadsheets, avoid spaces in column headers.

*Avoid symbols in column headers. This can cause issues when importing into R.