Intro to R

L.G.Carlson, M.Barajas, A.Hart

6/1/2020

Learning R: What/Why/How at GMRI

What is R/Rstudio (now posit?)

R:

- Programming language
- Write in text what you want the computer to do, it does that stuff

Rstudio:

- Free software program for code editing and testing built for the R user
- *Think:* MS word or google sheets, but for writing code
- Compare to: Visual studio code, pycharm, sublime text, notepad

Posit:

- Public benefit corporation, previously named "Rstudio" (confusing right?)
- Responsible for Rstudio software, Rshiny development, R user groups (regional coding clubs)

Why do people use it?

> R is an open-source, domain-specific language, explicitly designed for data science. Very popular in finance and academia, R is a perfect language for data manipulation, processing and visualization, as well as statistical computing and machine learning.
>
> ~DataCamp: 12 Top Data Science Languages of 2023, #2: R

- **Scientists/analysts use it for:**
  - Reproducibility
  - Efficiency
  - Performance gains (data too big to look at)
  - Access to different tools & packages (GIS, ML, etc.)
  - GUI interfaces are running it under the hood anyways

- **People stick around for:**
  - Open-source community
  - Skill transferability
  - Self-empowerment (I can solve, or teach myself X, Y, & Z)

Learning to code is a knowledge investment; it *is* learning a new language, after all. But like learning any language, coding will open new doors.

How do we at GMRI use R?

- **We use R (and sometimes also python) to:**
  - **Wrangle messy data**
  - Run statistical tests
  - Document our research analysis
  - Create reports to communicate our findings
  - Build portfolio websites from scratch
  - Create maps of findings
  - Automate simple tasks
  - Design web-applications

Workshop expectations:

Workshop starting point

1. We are starting at ground zero.
2. We assume no prior knowledge of R.
3. We only assume a general familiarity with computers.
4. Please ask for help whenever needed!

How this workshop is planned

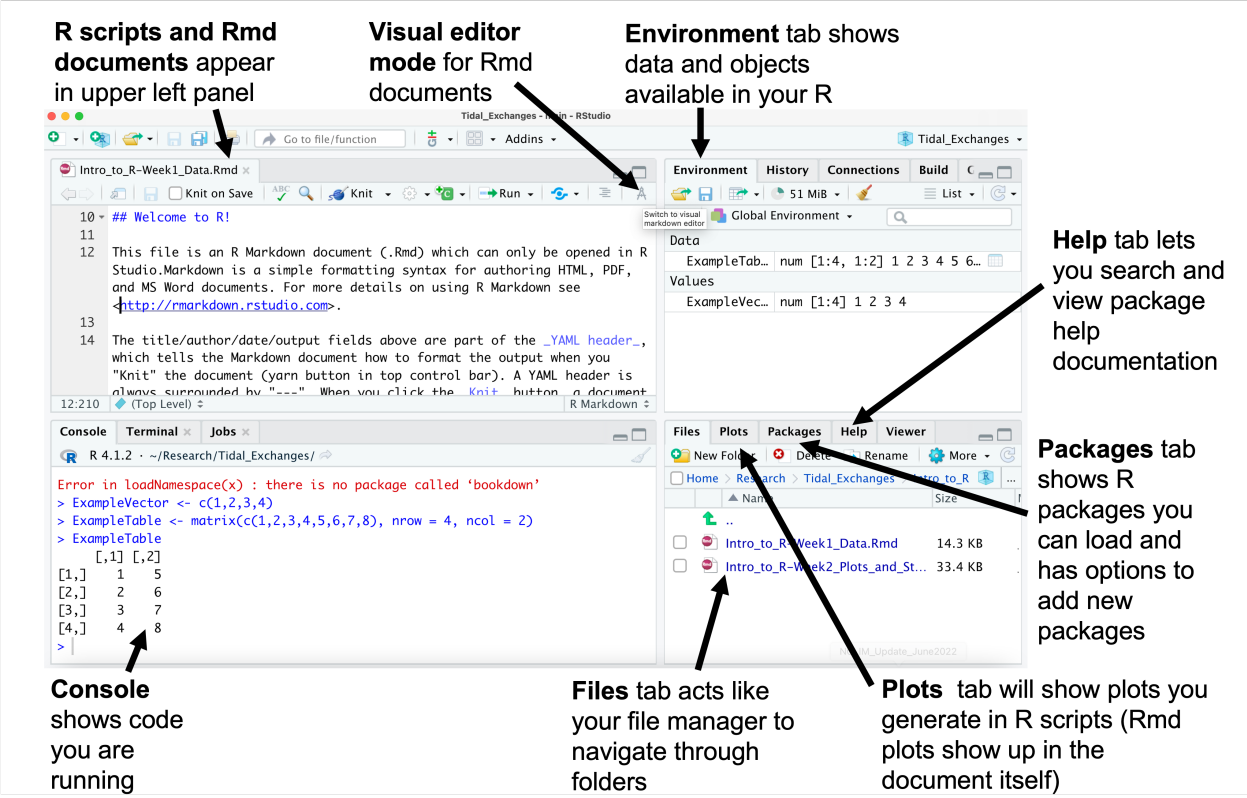
This workshop is intended to expose people to the R language and its core functionality.

Beyond that we are happy to explore interests of the group and detour for any and all questions.

This workshop is for y'all!

Welcome to R & RStudio!

RStudio is the user interface for R and will look a bit like the following when you first get started:



There are three types of files that we typically work on using RStudio:

1. R scripts end with a ".R" extension and contain exclusively code, hashtags can be inserted to denote text comments that shouldn't be executed as code. Plots generated by the R script will appear in the plots tab in the bottom right of the RStudio interface.

2. RMarkdown files end with ".Rmd" extensions and can include a mix of text, code chunks, and plots. RMarkdown files can only be opened in R Studio.Markdown and have simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

3. Quarto files:

All RMarkdown documents will have a YAML header, which tells Markdown how to format the output when you "Knit" the document (yarn button in top control bar) to the final format and contains information on the title, author, date, and output type at the top of the file. When you click the *Knit* button, a document will be generated that includes both written content as well as the output of any embedded R code chunks within the document. All other information in the file can be edited in one of two ways: 1) edit in the source editor (default view) or 2) switch to the visual editor mode (compass icon in upper right of RMarkdown panel) to edit more like a typical text editor.

1. Use source editor view

    - You can treat the *white space* in this document almost like a Word document. Use it to add metadata or document your process.

- You can use html codes to format the text in the white space. For example, you can *italicize* or **bold** regular text, add hyperlinks and create various level headers:

  - # Header 1

    ## Header 2

    ### Header 3

    #### Header 4

    ##### Header 5

  - Header 6

- You can also create bulleted or numbered lists:
  1. Item
  2. Item
     - Sub-item
     - Sub-item
  3. Item
- The *gray* areas are called "embedded code chunks." You can add a code chunk using the green + "Insert" button on the top control bar, or by typing three backticks (‘). You must also close to chunk with three backticks to run the chunk. The "r" in {brackets} at the top of the code chunks tells R Studio that you're writing in the R programming language (you can also write in other languages, but we won't worry about that for now).
- You can run the chunk by highlighting the code and pressing the "Run" button on the top control bar. You can also press the green arrow at the far right of the code chunk to run it.
- The code will run in the console (bottom left panel) and the code's output will be displayed inline or embedded in the active script (top left panel... and probably where you're reading this).
- Objects created by the code will be saved into the Environment (top right panel).
- The bottom right panel has a few useful functionalities including allowing you to install packages (non-programmatically) and view which packages you already have installed (Packages Tab), search the help menu/documentation (Help Tab), and view plots you created in the code (Viewer Tab).
- Base R has >2000 pre-loaded functions, so without doing anything else, you can actually do a lot in R! But there are thousands of packages that can also be installed and use in R, which gives you endless possibilities to explore.

2. Use visual editor view

   - Use control panel to format document as you would a word document or GoogleDoc
   - Code chunks can be added via the "Insert" button and will be displayed in the same way as the source editor once the document has been knitted to its final format.

What is a function?

Functions are "self contained" modules of code that accomplish a specific task. Functions usually take in some sort of data structure (value, vector, dataframe etc.), process it, and return a result. Functions are often called either from base R or a package you've loaded into your library. You can also write your own functions. We'll get to that later.

The general usage for a function is the name of the function followed by parentheses:

```
function_name(input)
```

The input(s) are called arguments, which can include:

- The data object (any data structure) on which the function carries out a task
- Specifications that alter the way the function operates (e.g. options)

Some functions can be called without arguments provided, in which case default options are used. For example `getwd()` tells you what folder RStudio is working within and requires no argument to run:

```
getwd()
```

```
## [1] "/Users/akemberling/Documents/Repositories/Tidal_Exchanges/Intro_to_R"
```

Most functions can take several arguments. If you don't specify a required argument when calling the function, you will either receive an error (you are required to provide the argument) or the function will fall back on using a default setting.

The defaults represent standard values that the author of the function specified as being "good enough in standard cases". An example would be what symbol or color to use in a plot.

To get more information about a function and its arguments use the `help()` function to pull up documentation in the 'Help' tab (bottom right). The argument is any function from base R or a package you have installed.

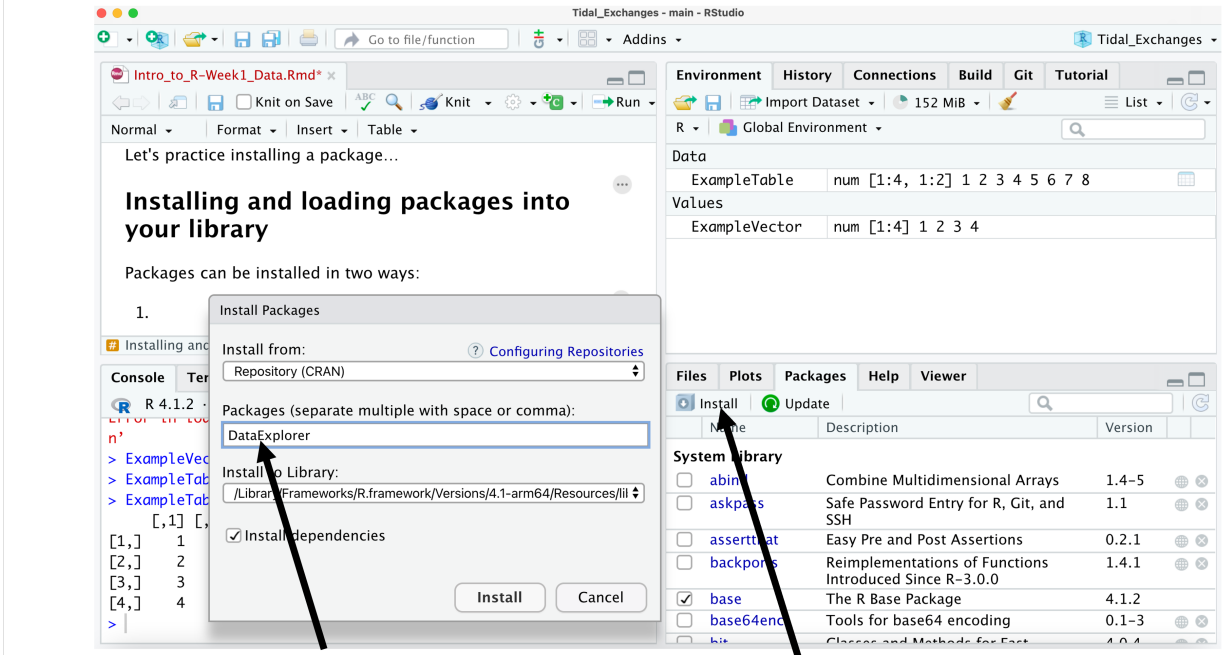
```
help(getwd)
```

All analysis in R uses functions, lets start by looking at a few that can help add new tools to R...

Installing and loading packages into your library

Packages contain groups of related functions and come pre-installed (i.e. functions and packages that are part of base R) or can be installed in two ways:

1. Use the 'Packages' tab and select the 'Install' option to install a specific package.



2. Install using code:

```
# You can also write documentation (non-code) in the code chunk, but you must # it out (will turn green

# This is the setup chunk (chunk names go after the language in the brackets)

knitr::opts_chunk$set(echo = TRUE)

# First, we will install 2 packages using code. You only need to do this step once. After tidyverse and

#install.packages("tidyverse")
#install.packages("DataExplorer")

install.packages("tidyverse")
install.packages("DataExplorer")
```

```
# Second, you must load the installed packages into your library (working RStudio session) to be able t

library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.2     v purrr   1.0.1
## v tibble  3.2.1     v dplyr   1.1.2
## v tidyr   1.3.0     v stringr 1.5.0
```

```
## v readr   2.1.3     v forcats 0.5.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(DataExplorer)
```

`tidyverse` is actually a collection of packages. The packages installed by tidyverse (tidyr, dplyr, readr, purrr, stringr, forcats, ggplot, and more!) contain the most useful (and most widely used) functions in R, outside of those functions which come as part of the base R package (you don't need to install this). We will cover many of these functions later on.

`DataExplorer` is a package that provides useful summaries for data exploration which we will use later on.

Reading in data

How data is stored determines how we can read it into R for use in our analysis. Often public data sets are available through R packages or as Rdata (files with ".rds" extension), while our own data sets are saved as Excel or CSV files. For one-time analyses and quick testing it is sometimes possible to copy and paste data into R. Here we outline paths to load some of the more common data formats that we use, but know that R has lots of options to load other types of data (e.g. large data files are often pulled from a SQL databases and climate data is often stored as a NetCDF file).

- Use the `<-` to assign something to an object that will be saved in your environment, this is equivalent to `=` in R but people get defensive about this so it is best to do what feels right for you and try not to debate it with others.
- You will need to provide the full file path for CSV, Rdata, and Excel files whose data you want to load (e.g. "/Users/yourusername/folderwithdata/data.csv" instead of "data.csv"). The exception is when data you want to load is in your current working directory (i.e. the folder that RStudio is working in).

Before you start loading data check your current working directory using `getwd()`:

```
getwd()
```

```
## [1] "/Users/akemberling/Documents/Repositories/Tidal_Exchanges/Intro_to_R"
```

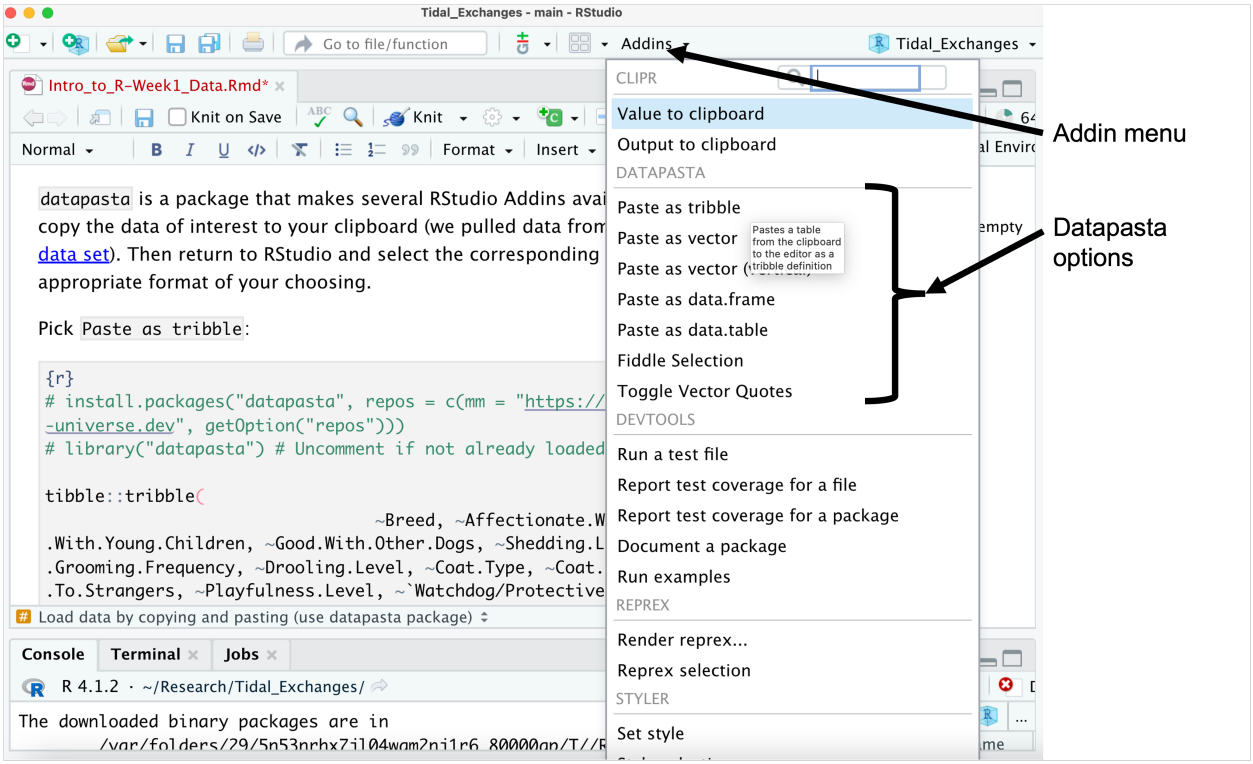
Use `setwd()` to change your working directory to the location where your data is stored:

```
#setwd("/Users/yourusername/folderwithdata")
```

```
setwd("/Users/yourusername/folderwithdata")
```

Load data by copying and pasting (use `datapasta` package)

`datapasta` is a package that makes several RStudio Addins available. To use, first copy the data of interest to your clipboard (we pulled data from this TidyTuesday data set). Then return to RStudio and select the corresponding Addin to paste in an appropriate format of your choosing. These addin paste options may also be linked to keyboard shortcuts.



Pick `Paste as tribble`:

```
#install.packages("datapasta", repos = c(mm = "https://milesmcbain.r-universe.dev", getOption("repos"))
library("datapasta") # Uncomment if not already loaded

tibble::tribble(
                                ~Breed, ~Affectionate.With.Family, ~Good.With.Young.Children, ~Good.Wi
               "Retrievers (Labrador)",                        5L,                         5L,
                    "French Bulldogs",                        5L,                         5L,
              "German Shepherd Dogs",                        5L,                         5L,
               "Retrievers (Golden)",                        5L,                         5L,
                           "Bulldogs",                        4L,                         3L,
                            "Poodles",                        5L,                         5L,
                            "Beagles",                        3L,                         5L,
                         "Rottweilers",                        5L,                         3L,
      "Pointers (German Shorthaired)",                        5L,                         5L,
                         "Dachshunds",                        5L,                         3L,
       )
```

```
## # A tibble: 10 x 17
##    Breed      Affectionate.With.Fa~1 Good.With.Young.Chil~2 Good.With.Other.Dogs
##    <chr>                       <int>                  <int>                 <int>
##  1 Retriever~                      5                      5                     5
##  2 French Bu~                      5                      5                     4
##  3 German Sh~                      5                      5                     3
##  4 Retriever~                      5                      5                     5
##  5 Bulldogs                        4                      3                     3
##  6 Poodles                         5                      5                     3
##  7 Beagles                         3                      5                     5
```

```
##  8 Rottweile~                     5                     3                     3
##  9 Pointers ~                     5                     5                     4
## 10 Dachshunds                     5                     3                     4
## # i abbreviated names: 1: Affectionate.With.Family, 2: Good.With.Young.Children
## # i 13 more variables: Shedding.Level <int>, Coat.Grooming.Frequency <int>,
## #   Drooling.Level <int>, Coat.Type <chr>, Coat.Length <chr>,
## #   Openness.To.Strangers <int>, Playfulness.Level <int>,
## #   `Watchdog/Protective.Nature` <int>, Adaptability.Level <int>,
## #   Trainability.Level <int>, Energy.Level <int>, Barking.Level <int>,
## #   Mental.Stimulation.Needs <int>
```

Pick `Paste as data.frame`:

```
data.frame(
              stringsAsFactors = FALSE,
                   check.names = FALSE,
                         Breed = c("Retrievers (Labrador)","French Bulldogs",
                                   "German Shepherd Dogs","Retrievers (Golden)",
                                   "Bulldogs","Poodles","Beagles","Rottweilers",
                                   "Pointers (German Shorthaired)","Dachshunds"),
      Affectionate.With.Family = c(5L, 5L, 5L, 5L, 4L, 5L, 3L, 5L, 5L, 5L),
      Good.With.Young.Children = c(5L, 5L, 5L, 5L, 3L, 5L, 5L, 3L, 5L, 3L),
          Good.With.Other.Dogs = c(5L, 4L, 3L, 5L, 3L, 3L, 5L, 3L, 4L, 4L),
                Shedding.Level = c(4L, 3L, 4L, 4L, 3L, 1L, 3L, 3L, 3L, 2L),
       Coat.Grooming.Frequency = c(2L, 1L, 2L, 2L, 3L, 4L, 2L, 1L, 2L, 2L),
                 Drooling.Level = c(2L, 3L, 2L, 2L, 3L, 1L, 1L, 3L, 2L, 2L),
                      Coat.Type = c("Double","Smooth","Double","Double","Smooth",
                                   "Curly","Smooth","Smooth","Smooth","Smooth"),
                   Coat.Length = c("Short",
                                   "Short","Medium","Medium","Short",
                                   "Long","Short","Short","Short","Short"),
        Openness.To.Strangers = c(5L, 5L, 3L, 5L, 4L, 5L, 3L, 3L, 4L, 4L),
            Playfulness.Level = c(5L, 5L, 4L, 4L, 4L, 5L, 4L, 4L, 4L, 4L),
  `Watchdog/Protective.Nature` = c(3L, 3L, 5L, 3L, 3L, 5L, 2L, 5L, 4L, 4L),
           Adaptability.Level = c(5L, 5L, 5L, 5L, 3L, 4L, 4L, 4L, 4L, 4L),
           Trainability.Level = c(5L, 4L, 5L, 5L, 4L, 5L, 3L, 5L, 5L, 4L),
                 Energy.Level = c(5L, 3L, 5L, 3L, 3L, 4L, 4L, 3L, 5L, 3L),
                Barking.Level = c(3L, 1L, 3L, 1L, 2L, 4L, 4L, 1L, 3L, 5L),
      Mental.Stimulation.Needs = c(4L, 3L, 5L, 4L, 3L, 5L, 4L, 5L, 5L, 3L)
)
```

```
##                          Breed Affectionate.With.Family
## 1          Retrievers (Labrador)                       5
## 2                French Bulldogs                       5
## 3           German Shepherd Dogs                       5
## 4            Retrievers (Golden)                       5
## 5                        Bulldogs                       4
## 6                         Poodles                       5
## 7                         Beagles                       3
## 8                      Rottweilers                       5
## 9  Pointers (German Shorthaired)                       5
## 10                      Dachshunds                       5
##    Good.With.Young.Children Good.With.Other.Dogs Shedding.Level
```

```
## 1                          5                   5              4
## 2                          5                   4              3
## 3                          5                   3              4
## 4                          5                   5              4
## 5                          3                   3              3
## 6                          5                   3              1
## 7                          5                   5              3
## 8                          3                   3              3
## 9                          5                   4              3
## 10                         3                   4              2
##    Coat.Grooming.Frequency Drooling.Level Coat.Type Coat.Length
## 1                        2              2    Double       Short
## 2                        1              3    Smooth       Short
## 3                        2              2    Double      Medium
## 4                        2              2    Double      Medium
## 5                        3              3    Smooth       Short
## 6                        4              1     Curly        Long
## 7                        2              1    Smooth       Short
## 8                        1              3    Smooth       Short
## 9                        2              2    Smooth       Short
## 10                       2              2    Smooth       Short
##    Openness.To.Strangers Playfulness.Level Watchdog/Protective.Nature
## 1                      5                 5                         3
## 2                      5                 5                         3
## 3                      3                 4                         5
## 4                      5                 4                         3
## 5                      4                 4                         3
## 6                      5                 5                         5
## 7                      3                 4                         2
## 8                      3                 4                         5
## 9                      4                 4                         4
## 10                     4                 4                         4
##    Adaptability.Level Trainability.Level Energy.Level Barking.Level
## 1                   5                  5            5             3
## 2                   5                  4            3             1
## 3                   5                  5            5             3
## 4                   5                  5            3             1
## 5                   3                  4            3             2
## 6                   4                  5            4             4
## 7                   4                  3            4             4
## 8                   4                  5            3             1
## 9                   4                  5            5             3
## 10                  4                  4            3             5
##    Mental.Stimulation.Needs
## 1                         4
## 2                         3
## 3                         5
## 4                         4
## 5                         3
## 6                         5
## 7                         4
## 8                         5
## 9                         5
## 10                        3
```

Pick `Paste as data.table` :

```
data.table::data.table(
                  check.names = FALSE,
                        Breed = c("Retrievers (Labrador)","French Bulldogs",
                                  "German Shepherd Dogs","Retrievers (Golden)",
                                  "Bulldogs","Poodles","Beagles","Rottweilers",
                                  "Pointers (German Shorthaired)","Dachshunds"),
      Affectionate.With.Family = c(5L, 5L, 5L, 5L, 4L, 5L, 3L, 5L, 5L, 5L),
      Good.With.Young.Children = c(5L, 5L, 5L, 5L, 3L, 5L, 5L, 3L, 5L, 3L),
          Good.With.Other.Dogs = c(5L, 4L, 3L, 5L, 3L, 3L, 5L, 3L, 4L, 4L),
                Shedding.Level = c(4L, 3L, 4L, 4L, 3L, 1L, 3L, 3L, 3L, 2L),
       Coat.Grooming.Frequency = c(2L, 1L, 2L, 2L, 3L, 4L, 2L, 1L, 2L, 2L),
                Drooling.Level = c(2L, 3L, 2L, 2L, 3L, 1L, 1L, 3L, 2L, 2L),
                     Coat.Type = c("Double","Smooth","Double","Double","Smooth",
                                  "Curly","Smooth","Smooth","Smooth","Smooth"),
                   Coat.Length = c("Short",
                                  "Short","Medium","Medium","Short",
                                  "Long","Short","Short","Short","Short"),
        Openness.To.Strangers = c(5L, 5L, 3L, 5L, 4L, 5L, 3L, 3L, 4L, 4L),
           Playfulness.Level = c(5L, 5L, 4L, 4L, 4L, 5L, 4L, 4L, 4L, 4L),
  `Watchdog/Protective.Nature` = c(3L, 3L, 5L, 3L, 3L, 5L, 2L, 5L, 4L, 4L),
           Adaptability.Level = c(5L, 5L, 5L, 5L, 3L, 4L, 4L, 4L, 4L, 4L),
           Trainability.Level = c(5L, 4L, 5L, 5L, 4L, 5L, 3L, 5L, 5L, 4L),
                 Energy.Level = c(5L, 3L, 5L, 3L, 3L, 4L, 4L, 3L, 5L, 3L),
                Barking.Level = c(3L, 1L, 3L, 1L, 2L, 4L, 4L, 1L, 3L, 5L),
      Mental.Stimulation.Needs = c(4L, 3L, 5L, 4L, 3L, 5L, 4L, 5L, 5L, 3L)
)
```

```
##                            Breed Affectionate.With.Family
##  1:         Retrievers (Labrador)                       5
##  2:               French Bulldogs                       5
##  3:          German Shepherd Dogs                       5
##  4:           Retrievers (Golden)                       5
##  5:                      Bulldogs                       4
##  6:                       Poodles                       5
##  7:                       Beagles                       3
##  8:                    Rottweilers                      5
##  9: Pointers (German Shorthaired)                       5
## 10:                     Dachshunds                      5
##     Good.With.Young.Children Good.With.Other.Dogs Shedding.Level
##  1:                        5                    5              4
##  2:                        5                    4              3
##  3:                        5                    3              4
##  4:                        5                    5              4
##  5:                        3                    3              3
##  6:                        5                    3              1
##  7:                        5                    5              3
##  8:                        3                    3              3
##  9:                        5                    4              3
## 10:                        3                    4              2
##     Coat.Grooming.Frequency Drooling.Level Coat.Type Coat.Length
##  1:                       2              2    Double       Short
##  2:                       1              3    Smooth       Short
```

```
##  3:                        2              2    Double      Medium
##  4:                        2              2    Double      Medium
##  5:                        3              3    Smooth       Short
##  6:                        4              1     Curly        Long
##  7:                        2              1    Smooth       Short
##  8:                        1              3    Smooth       Short
##  9:                        2              2    Smooth       Short
## 10:                        2              2    Smooth       Short
##     Openness.To.Strangers Playfulness.Level Watchdog/Protective.Nature
##  1:                     5                 5                         3
##  2:                     5                 5                         3
##  3:                     3                 4                         5
##  4:                     5                 4                         3
##  5:                     4                 4                         3
##  6:                     5                 5                         5
##  7:                     3                 4                         2
##  8:                     3                 4                         5
##  9:                     4                 4                         4
## 10:                     4                 4                         4
##     Adaptability.Level Trainability.Level Energy.Level Barking.Level
##  1:                  5                  5            5             3
##  2:                  5                  4            3             1
##  3:                  5                  5            5             3
##  4:                  5                  5            3             1
##  5:                  3                  4            3             2
##  6:                  4                  5            4             4
##  7:                  4                  3            4             4
##  8:                  4                  5            3             1
##  9:                  4                  5            5             3
## 10:                  4                  4            3             5
##     Mental.Stimulation.Needs
##  1:                        4
##  2:                        3
##  3:                        5
##  4:                        4
##  5:                        3
##  6:                        5
##  7:                        4
##  8:                        5
##  9:                        5
## 10:                        3
```

Load data from a CSV (.csv) file

```
library(palmerpenguins)
write.csv(penguins, here::here("Intro_to_R/Data","penguins_data.R"))
```

```
library(readr) # If the readr package is not loaded, uncomment and run this line of code
penguinsCSV <- read_csv("Data/penguins_data.csv")
```

```
## Rows: 344 Columns: 7
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): species, island, sex
## dbl (4): bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# library(readr) # If the readr package is not loaded, uncomment and run this line of code
penguinsCSV <- read_csv(here::here("Intro_to_R", "Data/penguins_data.csv"))
```

```
## Rows: 344 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): species, island, sex
## dbl (4): bill_length_mm, bill_depth_mm, flipper_length_mm, body_mass_g
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

The text below the code gives you information about what happened when you ran the code. Sometimes you'll get an error or warning message, but in this case, the output is telling you what variable types it assigned to each column.

Now that you've run this code chunk, you'll be able to see "penguinsCSV" in your environment (top right). Let's see what the data looks like inline.

```
# To look at the entire dataset, just rerun the "penguins" object or type it into the console and hit e
penguinsCSV
```

```
## # A tibble: 344 x 7
##    species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <chr>   <chr>              <dbl>         <dbl>             <dbl>       <dbl>
##  1 Adelie  Torgersen           39.1          18.7               181        3750
##  2 Adelie  Torgersen           39.5          17.4               186        3800
##  3 Adelie  Torgersen           40.3          18                 195        3250
##  4 Adelie  Torgersen           NA            NA                  NA          NA
##  5 Adelie  Torgersen           36.7          19.3               193        3450
##  6 Adelie  Torgersen           39.3          20.6               190        3650
##  7 Adelie  Torgersen           38.9          17.8               181        3625
##  8 Adelie  Torgersen           39.2          19.6               195        4675
##  9 Adelie  Torgersen           34.1          18.1               193        3475
## 10 Adelie  Torgersen           42            20.2               190        4250
## # i 334 more rows
## # i 1 more variable: sex <chr>
```

R Markdown gives you a preview of the dataset and allows you to look through the whole dataset using the 'Next' button at the bottom of the table. Markdown also shows you what type of data is contained in each column below the column name (i.e., or).

Load data from an Excel (.xls or .xlsx) file

```
# install.packages("readxl")
# library("readxl") # Uncomment if package not already loaded

sheet1 <- readxl::read_excel(here::here("Intro_to_R/Data/testExcel.xlsx"), sheet = 1) # Pick sheet by n
sheet1
```

```
## # A tibble: 5 x 3
##   frog  common_name     size_mm
##   <lgl> <chr>             <dbl>
## 1 TRUE  tree_frog           100
## 2 FALSE horntoad            144
## 3 TRUE  glass_frog           32
## 4 FALSE frog_eyed_gecko     200
## 5 FALSE axolotl             450
```

```
sheet2 <- readxl::read_excel(here::here("Intro_to_R/Data/testExcel.xlsx"), sheet = "Is-it-a-fish") # Pi
sheet2
```

```
## # A tibble: 5 x 3
##   fish  common_name   largest_size_cm
##   <lgl> <chr>                  <dbl>
## 1 FALSE cuddlefish              1
## 2 TRUE  flying_fish             0.5
## 3 TRUE  basking_shark          12.3
## 4 TRUE  mola_mola               2.72
## 5 FALSE jellyfish              36.5
```

Load data from an RData (.rds) file

```
practice <- readRDS(file = here::here("Intro_to_R/Data/testRData.rds"))
practice # Print the data you loaded to screen
```

```
## $happy
## [1]  TRUE FALSE  TRUE  TRUE FALSE
##
## $weather
## [1] "Sun"    "Rain"   "Fog"    "Breezy" "Snow"
##
## $temperature
## [1] 80 54 70 73 35
```

Load data from an R package

Typically all you need to do to load data from an R package is to load the package and call the named data set.

```
# install.packages("palmerpenguins")
# Load the palmerpenguins package
library(palmerpenguins)
```

```r
# Take a look at the data set from this R package
penguins
```

```
## # A tibble: 344 x 8
##    species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
##  1 Adelie  Torgersen           39.1          18.7               181        3750
##  2 Adelie  Torgersen           39.5          17.4               186        3800
##  3 Adelie  Torgersen           40.3          18                 195        3250
##  4 Adelie  Torgersen           NA            NA                  NA          NA
##  5 Adelie  Torgersen           36.7          19.3               193        3450
##  6 Adelie  Torgersen           39.3          20.6               190        3650
##  7 Adelie  Torgersen           38.9          17.8               181        3625
##  8 Adelie  Torgersen           39.2          19.6               195        4675
##  9 Adelie  Torgersen           34.1          18.1               193        3475
## 10 Adelie  Torgersen           42            20.2               190        4250
## # i 334 more rows
## # i 2 more variables: sex <fct>, year <int>
```

```r
# palmerpenguins::penguins is equivalent to the above line of code but explicitly lists the package the
```

Types of objects in R

Scalars

Scalars hold a single element or atomic value – numeric, character, etc

*Note: Character elements must be enclosed by quotation mark: either " or '

```r
my_col<-'red'
my_num<-13

my_col; my_num
```

```
## [1] "red"
```

```
## [1] 13
```

```r
class(my_col)
```

```
## [1] "character"
```

```r
class(my_num)
```

```
## [1] "numeric"
```

Vectors

Vectors hold multiple elements all of the same class (can't mix numbers and character values in same vector)

```r
year_vec<-c(1968, 1981, 1996)
org_vec<-c('GMRI', 'NMFS', 'DFO')

year_vec;org_vec
```

```
## [1] 1968 1981 1996
```

```
## [1] "GMRI" "NMFS" "DFO"
```

Vectorized operations work on numeric vectors (i.e., you can do math on vectors)

```r
year_vec+1
```

```
## [1] 1969 1982 1997
```

Dataframes/tibbles

2-dimensional data storage type, similar to an Excel spreadsheet

Different columns (which are each vectors) can have different classes

To create a dataframe or tibble, you can use the function data.frame() or tibble() where the arguments are the column names = a vector. You can either refer to a previously created vector (i.e., org_vec) below, or create on within the function using c().

```r
my_df<-data.frame(Org = org_vec, Year = c(2000,2000,2000))
my_df
```

```
##    Org Year
## 1 GMRI 2000
## 2 NMFS 2000
## 3  DFO 2000
```

Note: There are other types of objects you can create in R such as matrices, lists, and arrays, but for now, you can get away with using mostly vectors and dataframes.

Exploratory data analysis

Getting started

Most data exploration tries to answer the following questions:

- What type of data do you have (i.e. in what format is it stored)?
- Are there missing values or unusual features that may cause problems with your analysis?
- Do you need to restructure your data or do preliminary calculations before you can start your analysis?

Here are a few functions to help you take a first look at your data quickly:

`DataExplorer::create_report()` is a function that will automatically generate an html report that summarizes the input data using plots and tables. Full tutorial available here.

Data Profiling Report

- Basic Statistics
  - Raw Counts
  - Percentages
- Data Structure
- Missing Data Profile
- Univariate Distribution
  - Histogram
  - Bar Chart (with frequency)
  - QQ Plot
- Correlation Analysis
- Principal Component Analysis

Basic Statistics

Raw Counts

Name	Value
Rows	344
Columns	8
Discrete columns	3
Continuous columns	5
All missing columns	0
Missing observations	19
Complete Rows	333
Total observations	2,752
Memory allocation	17.6 Kb

```
# install.package("DataExplorer")
 library(DataExplorer) # Uncomment if package isn't loaded in R
DataExplorer::create_report(penguins)
```

```
##
##
## processing file: report.rmd

## output file: /Users/akemberling/Documents/Repositories/Tidal_Exchanges/Intro_to_R/report.knit.md

##
## Output created: report.html
```

`summary()` is a function that will automatically count the number of occurrences for qualitative data and will provide the range, median, mean and innerquartile range for quantitative data.

```
summary(penguins)
```

```
##       species          island    bill_length_mm  bill_depth_mm
```

```
##  Adelie   :152   Biscoe   :168   Min.   :32.10   Min.   :13.10
##  Chinstrap: 68   Dream    :124   1st Qu.:39.23   1st Qu.:15.60
##  Gentoo   :124   Torgersen: 52   Median :44.45   Median :17.30
##                                  Mean   :43.92   Mean   :17.15
##                                  3rd Qu.:48.50   3rd Qu.:18.70
##                                  Max.   :59.60   Max.   :21.50
##                                  NA's   :2       NA's   :2
##  flipper_length_mm  body_mass_g       sex           year
##  Min.   :172.0     Min.   :2700   female:165   Min.   :2007
##  1st Qu.:190.0     1st Qu.:3550   male  :168   1st Qu.:2007
##  Median :197.0     Median :4050   NA's  : 11   Median :2008
##  Mean   :200.9     Mean   :4202                Mean   :2008
##  3rd Qu.:213.0     3rd Qu.:4750                3rd Qu.:2009
##  Max.   :231.0     Max.   :6300                Max.   :2009
##  NA's   :2         NA's   :2
```

`head()` and `tail()` are functions that will respectively show you the first and last 6 rows of a table. A larger or smaller number of rows can be displayed by specifying the argument `n=#rows to display` .

```
head(penguins) # default number of rows at top of table
```

```
## # A tibble: 6 x 8
##   species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
## 1 Adelie  Torgersen           39.1          18.7               181        3750
## 2 Adelie  Torgersen           39.5          17.4               186        3800
## 3 Adelie  Torgersen           40.3          18                 195        3250
## 4 Adelie  Torgersen           NA            NA                  NA          NA
## 5 Adelie  Torgersen           36.7          19.3               193        3450
## 6 Adelie  Torgersen           39.3          20.6               190        3650
## # i 2 more variables: sex <fct>, year <int>
```

```
tail(penguins, n=2) # Display only last 2 rows of a table
```

```
## # A tibble: 2 x 8
##   species   island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##   <fct>     <fct>           <dbl>         <dbl>             <int>       <int>
## 1 Chinstrap Dream            50.8          19                 210        4100
## 2 Chinstrap Dream            50.2          18.7               198        3775
## # i 2 more variables: sex <fct>, year <int>
```

`dim()` Will tell you the dimensions of a table (matrix, data.table, data.frame, tibble, ect.), `nrow()` will return the number of rows, `ncol()` will return the number of columns, and `length()` will provide the length (number of data points) in a vector or list.

```
dim(penguins)
```

```
## [1] 344   8
```

```
nrow(penguins)
```

```
## [1] 344
```

```
ncol(penguins)
```

```
## [1] 8
```

```
length(penguins) # Tibbles are also lists so here the return from length() is the same as ncol()
```

```
## [1] 8
```

`names()` will return the names of data points in a vector or list, while `colnames()` and `rownames()` return the column and row names respectively for tables.

```
names(penguins)
```

```
## [1] "species"           "island"            "bill_length_mm"
## [4] "bill_depth_mm"     "flipper_length_mm" "body_mass_g"
## [7] "sex"               "year"
```

```
colnames(penguins) # Same return as names() because tibbles are lists
```

```
## [1] "species"           "island"            "bill_length_mm"
## [4] "bill_depth_mm"     "flipper_length_mm" "body_mass_g"
## [7] "sex"               "year"
```

```
rownames(penguins) # Returns a vector of numbers because row names are just an index for this data set
```

```
##   [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"  "11"  "12"
##  [13] "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"  "23"  "24"
##  [25] "25"  "26"  "27"  "28"  "29"  "30"  "31"  "32"  "33"  "34"  "35"  "36"
##  [37] "37"  "38"  "39"  "40"  "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"
##  [49] "49"  "50"  "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"
##  [61] "61"  "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"  "71"  "72"
##  [73] "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"  "81"  "82"  "83"  "84"
##  [85] "85"  "86"  "87"  "88"  "89"  "90"  "91"  "92"  "93"  "94"  "95"  "96"
##  [97] "97"  "98"  "99"  "100" "101" "102" "103" "104" "105" "106" "107" "108"
## [109] "109" "110" "111" "112" "113" "114" "115" "116" "117" "118" "119" "120"
## [121] "121" "122" "123" "124" "125" "126" "127" "128" "129" "130" "131" "132"
## [133] "133" "134" "135" "136" "137" "138" "139" "140" "141" "142" "143" "144"
## [145] "145" "146" "147" "148" "149" "150" "151" "152" "153" "154" "155" "156"
## [157] "157" "158" "159" "160" "161" "162" "163" "164" "165" "166" "167" "168"
## [169] "169" "170" "171" "172" "173" "174" "175" "176" "177" "178" "179" "180"
## [181] "181" "182" "183" "184" "185" "186" "187" "188" "189" "190" "191" "192"
## [193] "193" "194" "195" "196" "197" "198" "199" "200" "201" "202" "203" "204"
## [205] "205" "206" "207" "208" "209" "210" "211" "212" "213" "214" "215" "216"
## [217] "217" "218" "219" "220" "221" "222" "223" "224" "225" "226" "227" "228"
## [229] "229" "230" "231" "232" "233" "234" "235" "236" "237" "238" "239" "240"
```

```
## [241] "241" "242" "243" "244" "245" "246" "247" "248" "249" "250" "251" "252"
## [253] "253" "254" "255" "256" "257" "258" "259" "260" "261" "262" "263" "264"
## [265] "265" "266" "267" "268" "269" "270" "271" "272" "273" "274" "275" "276"
## [277] "277" "278" "279" "280" "281" "282" "283" "284" "285" "286" "287" "288"
## [289] "289" "290" "291" "292" "293" "294" "295" "296" "297" "298" "299" "300"
## [301] "301" "302" "303" "304" "305" "306" "307" "308" "309" "310" "311" "312"
## [313] "313" "314" "315" "316" "317" "318" "319" "320" "321" "322" "323" "324"
## [325] "325" "326" "327" "328" "329" "330" "331" "332" "333" "334" "335" "336"
## [337] "337" "338" "339" "340" "341" "342" "343" "344"
```

`class()` is a function that allows you to see what type of object something is. In this case, penguins is a "tibble" or "data frame" (same thing).

```
class(penguins)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

What is the argument (or input) for class()?

We used "penguins" here, but you could use any object in your R environment.

`str()` is a function that allows you to look at the structure of a dataframe (or other type of data object in R). It's always a good idea to use str() to get an idea of what types of values are included in your data.

```
str(penguins)
```

```
## tibble [344 x 8] (S3: tbl_df/tbl/data.frame)
##  $ species          : Factor w/ 3 levels "Adelie","Chinstrap",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ island           : Factor w/ 3 levels "Biscoe","Dream",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ bill_length_mm   : num [1:344] 39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 42 ...
##  $ bill_depth_mm    : num [1:344] 18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.2 ...
##  $ flipper_length_mm: int [1:344] 181 186 195 NA 193 190 181 195 193 190 ...
##  $ body_mass_g      : int [1:344] 3750 3800 3250 NA 3450 3650 3625 4675 3475 4250 ...
##  $ sex              : Factor w/ 2 levels "female","male": 2 1 1 NA 1 2 1 2 NA NA ...
##  $ year             : int [1:344] 2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

Notice that some of the categorical variables in our penguins data set are listed as factors (e.g. 'species: Factor w/ 3 levels "Adelie", "Chinstrap",..: 1111111111...'). The factor levels have 2 pieces: the string (penguin names) that shows up in the tibble, and an integer that corresponds with this label that R can use in subsequent analysis of the categorical variable. The factor suggests that the first 10 rows should all have "Adelie" listed as the species, and if we look at the tibble this is indeed the case.

```
head(penguins, n=10)
```

```
## # A tibble: 10 x 8
##    species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
##  1 Adelie  Torgersen           39.1          18.7               181        3750
##  2 Adelie  Torgersen           39.5          17.4               186        3800
##  3 Adelie  Torgersen           40.3          18                 195        3250
##  4 Adelie  Torgersen           NA            NA                  NA          NA
##  5 Adelie  Torgersen           36.7          19.3               193        3450
```

```
##  6 Adelie  Torgersen           39.3          20.6               190        3650
##  7 Adelie  Torgersen           38.9          17.8               181        3625
##  8 Adelie  Torgersen           39.2          19.6               195        4675
##  9 Adelie  Torgersen           34.1          18.1               193        3475
## 10 Adelie  Torgersen           42            20.2               190        4250
## # i 2 more variables: sex <fct>, year <int>
```

Indexing

An important aspect of working with R objects is knowing how to "index" them. Indexing means selecting a subset of the data elements in order to use them in further analysis or possibly change them. Here we focus just on three kinds of vector indexing: positional, named reference, and logical Any of these indexing techniques works the same for all classes of vectors

Here are the most common types of index you'll use:

- `[n]`: Use brackets to pick the nth item in a data object
  - For vectors provide a single number `[n]` or name of the item
  - For tibbles, dataframes, and matrices provide a row and column numbers or names `[nrow, ncol]`
- `$`: access columns of a dataframe, or items in a list using `$` notation

Use brackets to index a vector or dataframe

```
# access value 2 in a vector
year_vec[2]
```

```
## [1] 1981
```

```
# access the value in row 1, column 1 of a dataframe
my_df[1,1]
```

```
## [1] "GMRI"
```

Rows using brackets

```
# access row 1 in a dataframe (number goes after the comma)
my_df[1,]
```

```
##    Org Year
## 1 GMRI 2000
```

Columns using brackets:

```
# access column 1 in a dataframe (number goes after the comma)
my_df[,1]
```

```
## [1] "GMRI" "NMFS" "DFO"
```

```
# Or index column by name
my_df[,"Org"]
```

```
## [1] "GMRI" "NMFS" "DFO"
```

You can also access columns by name rather than number by using the dollar sign indexing. If you type a dataframe's name followed by a $, R will suggest the column names. Because rows aren't named, you can do the same thing for them.

```
my_df$Org
```

```
## [1] "GMRI" "NMFS" "DFO"
```

You may want to use indexing to create new objects (vector in this case). For example:

```
bill_lengths<-penguins$bill_length_mm

class(bill_lengths)
```

```
## [1] "numeric"
```

```
head(bill_lengths,20)
```

```
##  [1] 39.1 39.5 40.3   NA 36.7 39.3 38.9 39.2 34.1 42.0 37.8 37.8 41.1 38.6 34.6
## [16] 36.6 38.7 42.5 34.4 46.0
```

`head` is a new function that returns the first "n" values in an object. The primary argument is any R object. The default for n=6, but here I specified 20.

Intro to the tidyverse

For this section, we will begin using functions from the tidyverse packages. To connect these functions, we will use an operator called the pipe that looks like this `%>%`

This operator will forward a value, or the result of an expression, into the next function call/expression. For instance a function to check the structure of data can be written as:

```
str(my_df)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ Org : chr  "GMRI" "NMFS" "DFO"
##  $ Year: num  2000 2000 2000
```

```
#or

my_df %>% str()
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ Org : chr  "GMRI" "NMFS" "DFO"
##  $ Year: num  2000 2000 2000
```

Lets just remind ourselves what the penguins dataset looks like:

```
penguins
```

```
## # A tibble: 344 x 8
##    species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
##  1 Adelie  Torgersen           39.1          18.7               181        3750
##  2 Adelie  Torgersen           39.5          17.4               186        3800
##  3 Adelie  Torgersen           40.3          18                 195        3250
##  4 Adelie  Torgersen           NA            NA                  NA          NA
##  5 Adelie  Torgersen           36.7          19.3               193        3450
##  6 Adelie  Torgersen           39.3          20.6               190        3650
##  7 Adelie  Torgersen           38.9          17.8               181        3625
##  8 Adelie  Torgersen           39.2          19.6               195        4675
##  9 Adelie  Torgersen           34.1          18.1               193        3475
## 10 Adelie  Torgersen           42            20.2               190        4250
## # i 334 more rows
## # i 2 more variables: sex <fct>, year <int>
```

Data wrangling

`filter()` - Often we want to work with particular parts of data sets. The `filter` function from package `dplyr` allows us to filter (or subset) based on a specific set of criteria

```
# filter by an exact condition, in this case, keep only species == "Gentoo"    *this is more often used

penguins %>% filter(species == "Gentoo")
```

```
## # A tibble: 124 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>             <int>       <int>
##  1 Gentoo  Biscoe           46.1          13.2               211        4500
##  2 Gentoo  Biscoe           50            16.3               230        5700
##  3 Gentoo  Biscoe           48.7          14.1               210        4450
##  4 Gentoo  Biscoe           50            15.2               218        5700
##  5 Gentoo  Biscoe           47.6          14.5               215        5400
##  6 Gentoo  Biscoe           46.5          13.5               210        4550
##  7 Gentoo  Biscoe           45.4          14.6               211        4800
##  8 Gentoo  Biscoe           46.7          15.3               219        5200
##  9 Gentoo  Biscoe           43.3          13.4               209        4400
## 10 Gentoo  Biscoe           46.8          15.4               215        5150
## # i 114 more rows
## # i 2 more variables: sex <fct>, year <int>
```

*Note, you can also write the function this way:

`filter(penguins,species == "Gentoo")`

But the pipe operator becomes useful when running multiple functions in a row

```r
# filter by value (can be greater than, lesser than, or equal to)

penguins %>% filter(body_mass_g > 5000)
```

```
## # A tibble: 61 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>             <int>       <int>
##  1 Gentoo  Biscoe           50            16.3               230        5700
##  2 Gentoo  Biscoe           50            15.2               218        5700
##  3 Gentoo  Biscoe           47.6          14.5               215        5400
##  4 Gentoo  Biscoe           46.7          15.3               219        5200
##  5 Gentoo  Biscoe           46.8          15.4               215        5150
##  6 Gentoo  Biscoe           49            16.1               216        5550
##  7 Gentoo  Biscoe           48.4          14.6               213        5850
##  8 Gentoo  Biscoe           49.3          15.7               217        5850
##  9 Gentoo  Biscoe           49.2          15.2               221        6300
## 10 Gentoo  Biscoe           48.7          15.1               222        5350
## # i 51 more rows
## # i 2 more variables: sex <fct>, year <int>
```

```r
# filter to remove by an exact condition, in this case, island != "Torgersen" Which means remove any pe

penguins %>% filter(island != "Torgersen")
```

```
## # A tibble: 292 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>             <int>       <int>
##  1 Adelie  Biscoe           37.8          18.3               174        3400
##  2 Adelie  Biscoe           37.7          18.7               180        3600
##  3 Adelie  Biscoe           35.9          19.2               189        3800
##  4 Adelie  Biscoe           38.2          18.1               185        3950
##  5 Adelie  Biscoe           38.8          17.2               180        3800
##  6 Adelie  Biscoe           35.3          18.9               187        3800
##  7 Adelie  Biscoe           40.6          18.6               183        3550
##  8 Adelie  Biscoe           40.5          17.9               187        3200
##  9 Adelie  Biscoe           37.9          18.6               172        3150
## 10 Adelie  Biscoe           40.5          18.9               180        3950
## # i 282 more rows
## # i 2 more variables: sex <fct>, year <int>
```

```r
# filter out NA values (works for both characters and numbers)

penguins %>% filter(!is.na(sex))
```

```
## # A tibble: 333 x 8
##    species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
##  1 Adelie  Torgersen           39.1          18.7               181        3750
##  2 Adelie  Torgersen           39.5          17.4               186        3800
##  3 Adelie  Torgersen           40.3          18                 195        3250
##  4 Adelie  Torgersen           36.7          19.3               193        3450
##  5 Adelie  Torgersen           39.3          20.6               190        3650
```

```
##  6 Adelie  Torgersen           38.9          17.8               181        3625
##  7 Adelie  Torgersen           39.2          19.6               195        4675
##  8 Adelie  Torgersen           41.1          17.6               182        3200
##  9 Adelie  Torgersen           38.6          21.2               191        3800
## 10 Adelie  Torgersen           34.6          21.1               198        4400
## # i 323 more rows
## # i 2 more variables: sex <fct>, year <int>
```

`mutate()` - adds new variables and preserves existing ones (unless you overwrite their name)

```
# Perhaps we want to flipper length (in mm) to cm, we could add a new column to do this while doing mat

penguins %>% mutate(flipper_length_cm = flipper_length_mm*0.1)
```

```
## # A tibble: 344 x 9
##    species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
##    <fct>   <fct>              <dbl>         <dbl>             <int>       <int>
##  1 Adelie  Torgersen           39.1          18.7               181        3750
##  2 Adelie  Torgersen           39.5          17.4               186        3800
##  3 Adelie  Torgersen           40.3          18                 195        3250
##  4 Adelie  Torgersen           NA            NA                  NA          NA
##  5 Adelie  Torgersen           36.7          19.3               193        3450
##  6 Adelie  Torgersen           39.3          20.6               190        3650
##  7 Adelie  Torgersen           38.9          17.8               181        3625
##  8 Adelie  Torgersen           39.2          19.6               195        4675
##  9 Adelie  Torgersen           34.1          18.1               193        3475
## 10 Adelie  Torgersen           42            20.2               190        4250
## # i 334 more rows
## # i 3 more variables: sex <fct>, year <int>, flipper_length_cm <dbl>
```

Calculating summary statistics

`group_by()` - group by one or more variables

`summarize()` - reduce multiple values down to a single value

Calculate the mean bill length by species

```
penguins %>%
  group_by(species) %>%
  summarise(mean_bill_length = mean(bill_length_mm, na.rm = T))
```

```
## # A tibble: 3 x 2
##   species   mean_bill_length
##   <fct>                <dbl>
## 1 Adelie                38.8
## 2 Chinstrap             48.8
## 3 Gentoo                47.5
```

Calculate the mean and standard deviation of body mass by sex

```
penguins %>%
  group_by(sex) %>%
  summarise(mean_mass = mean(body_mass_g, na.rm = T),
            sd_mass = sd(body_mass_g, na.rm = T))
```

```
## # A tibble: 3 x 3
##   sex    mean_mass sd_mass
##   <fct>      <dbl>   <dbl>
## 1 female     3862.    666.
## 2 male       4546.    788.
## 3 <NA>       4006.    679.
```

Other functions to know

Let's look at a few common base functions.

`c()` is a function that combines values into a vector or list. We will use it here to create a numeric vector and test out some other functions

Again, use the <- symbols to assign a name to some object you're creating. Oftentimes, you'll use functions to create a new object.

```
my_vector <- c(5,2,8,1,9,5)
```

Now, let's run some basic mathematical functions since we have a numeric vector.

- `mean()` - calculates mean (average) of a vector
- `max()` - calculates maximum value of a vector
- `sum()` - calculates sum of all values in a vector
- `sort()` - sorts values in a vector numerically

```
mean(my_vector)
```

```
## [1] 5
```

```
max(my_vector)
```

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

```
sum(my_vector)
```

```
## [1] 30
```

```
sort(my_vector)
```

```
## [1] 1 2 5 5 8 9
```

Do you agree with the outputs?

Specifying additional arguments or options within functions

Sometimes functions have additional arguments that change their behavior. If you don't specify additional arguments or options, R will use the default settings.

For the `sort` function we just learned, there are actually two arguments. Let's see what they are by opening the documentation using help().

```
help(sort)
```

Now we know that in addition to the vector, sort also includes an argument called "decreasing" for which the default setting is "FALSE". Thus, items are sorted in ascending order by default. If we want to sort in decreasing order, we can set this argument to "TRUE".

```
sort(my_vector, decreasing = TRUE)
```

```
## [1] 9 8 5 5 2 1
```

You don't have to specify the argument name (i.e., x= or decreasing=), so long as you put the arguments in the correct order. For example, this function works just as well

```
sort(my_vector, TRUE)
```

```
## [1] 9 8 5 5 2 1
```

Next week:

- More data wrangling
- Statistical tests & models
- Plots & maps