STAT 508 – Introduction to Statistical Learning and Exploratory Data Analysis (Lesson 1)

Lesson 1 Objectives:

1.1. Explain the difference between supervised and unsupervised learning.
1.2. Explain the difference between regression and classification.
1.3. Explain the difference between inference and prediction.
1.4. Gain proficiency in R programming by understanding its distinction from RStudio, using built-in functions, installing packages and loading libraries, and loading datasets from various sources into R.
1.5. Perform an exploratory data analysis, including the calculation of summary statistics and data visualization, to gain insights from the data.
1.6. Perform data wrangling tasks such as subsetting a dataset and creating new variables.

Lesson 1 Outline:

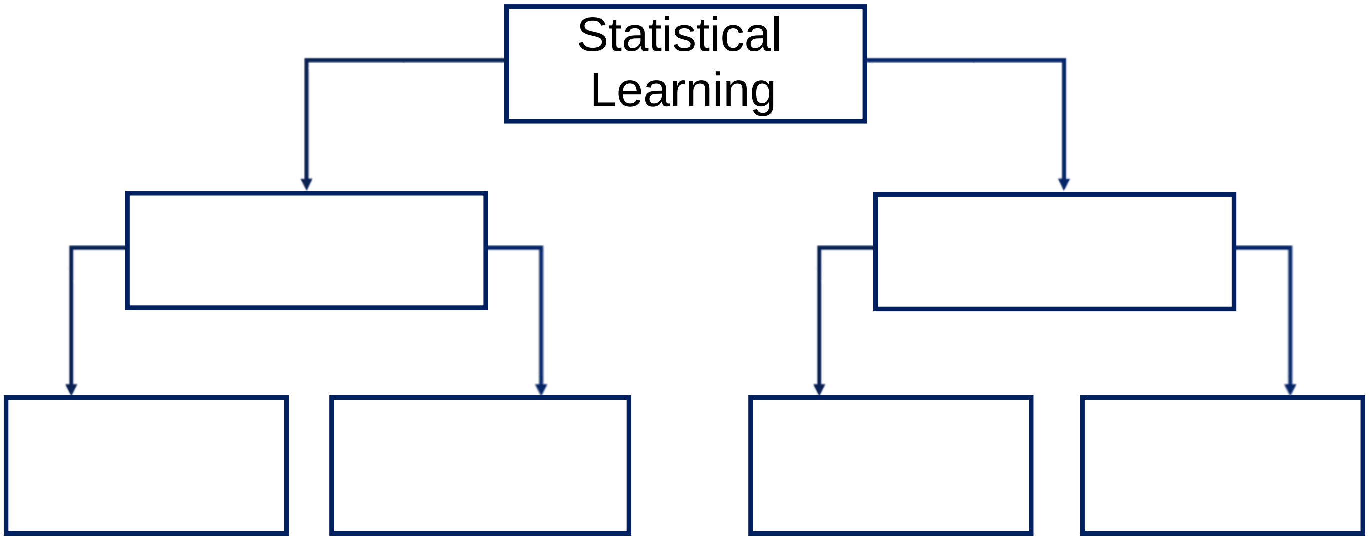
1. Introduction to Statistical Learning
    a. Statistical Learning Terminology
    b. Common Goals
2. Toolkit
    a. Overview of Tools
    b. RStudio Tour/Basic Functionality
        i. R as a calculator and using functions
        ii. Storing and using objects
        iii. Accessing help files
        iv. Common mistakes (typos, case, incomplete command)
        v. Reading and using external data
        vi. Installing Packages and loading libraries
    c. R Markdown
        i. Purpose
        ii. Structure
        iii. Examples
3. Exploratory Data Analysis (EDA)
    a. Overview
    b. Data Visualization and Numerical Summaries
        i. Types of plots
        ii. Layers of ggplot and aesthetics
        iii. Examples of visualization
        iv. Numerical summaries
    c. Data Wrangling
        i. dplyr verbs
        ii. Command chains
        iii. Examples

Introduction to Statistical Learning

Statistical Learning Terminology

Definition: Statistical learning refers to a vast set of tools for understanding and discovering structures in data. These tools can be classified as supervised or unsupervised.

EXAMPLE: (Statistical Learning Overview) The following diagram shows the types of statistical learning problems covered in STAT 508:



Terminology:

- **Supervised learning** involves building a statistical model for predicting, or estimating, an output based on one or more inputs. We can view supervised learning as using a function that maps input variable(s) to an output.
  - **Input variables** go by many names, such as: x, explanatory, predictor, regressor, independent, and feature.
  - The **output variable** also has many names, such as: y, response, dependent, target, outcome, and label.
  - Supervised learning problems fall into two broad categories.
    - **Regression** techniques are used when the output variable is quantitative.
    - **Classification** techniques are used when the output variable is categorical or qualitative.

- In **unsupervised learning**, there are inputs but no supervising output; nevertheless, we can learn relationships and structure from such data. The unsupervised learning problems we cover fall into two broad categories
  - **Dimension reduction** is a technique for transforming high dimensional data into a low dimensional space, while preserving useful properties of the data.
  - **Clustering** is a technique that involves finding subgroups, or clusters, in a dataset so that the observations within the same group are quite "similar" to each other, while observations in different groups are quite "different" from each other.

Common Goals

Our primary focus will be on supervised learning problems. Suppose that we observe a quantitative response, Y, and p predictors, $X_1, X_2, \dots, X_p$. Assuming there is a relationship between Y and $X = (X_1, X_2, \dots, X_p)$, we describe the general form of the relationship as:

$$\text{General Form of Regression Problem: } Y = f(X) + \epsilon$$

EXAMPLE: (General Regression Form): For each, identify the systematic components.

a. (Multiple linear regression) $y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + \epsilon_i$

b. (Exponential regression) $y_i = \beta_0 + \beta_1 e^{\beta_2 x_i} + \epsilon_i$

NOTE: There are two main reasons for estimating f: Prediction and Inference.

Prediction:

- Setting: We want to predict Y as accurately as possible
- Implementation: Predict Y using $\hat{Y} = \hat{f}(X)$ where $\hat{f}$ is an estimate of f

Inference:

- Setting: We want to understand/explain the association between X and Y
- Implementation: Understand the nature of $\hat{f}$
- Types of Questions:
  - Which predictors are associated with the response?
  - Is a linear relationship appropriate?
  - How does each predictor affect the response?

While many problems involve a combination of prediction and inference, the approach that we take for estimating f often depends on whether the goal is inference or prediction.

EXAMPLE: (Prediction vs. Inference) Suppose we will use a model for predicting whether a patient will develop heart disease based on demographic information (age, race, sex) and health metrics (body mass index (bmi), blood pressure, waist-to-hip ratio). Based on the stated goal, identify the problem as a prediction problem or an inference problem.

a. Goal: help patients understand how their health metrics affect the risk of developing heart disease. For instance, the doctor may want to explain how reducing bmi by 1 point affects the odds of developing heart disease.

b. Goal: diagnose patients as accurately as possible.

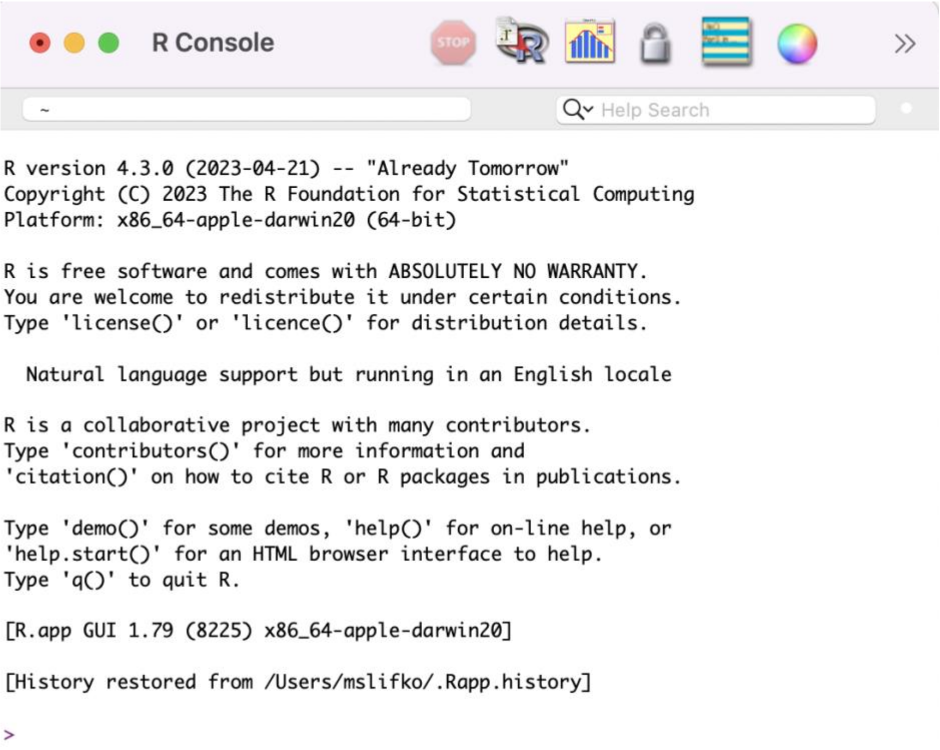
Toolkit

Overview of Tools

Throughout the course, we will use 3 primary tools for analyzing data.

1. R – a programming language and environment for statistical computing and graphics.

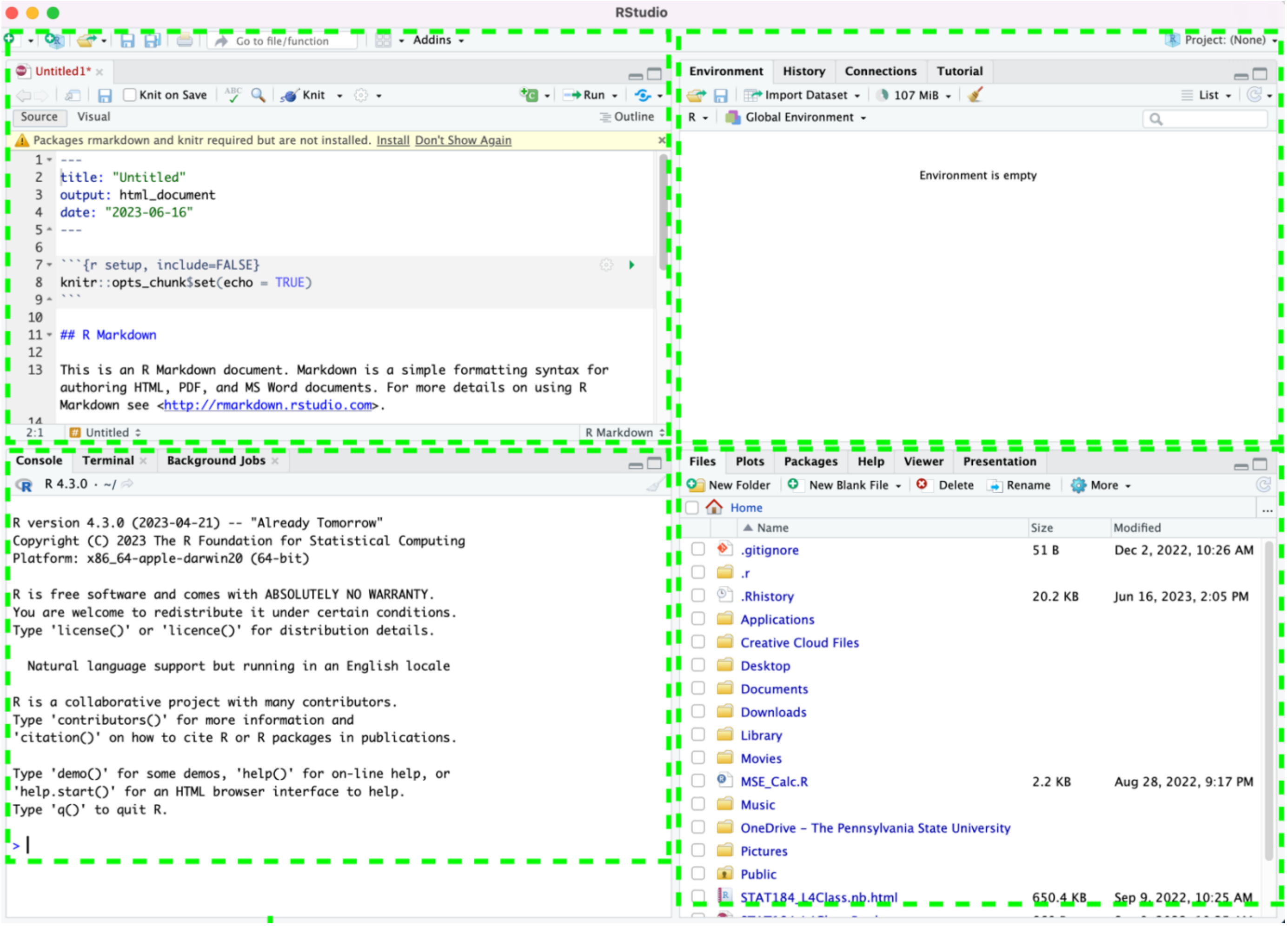
    a. While we must install R before installing RStudio, we will **not directly use** the R application.
    b. If you are seeing a window named “R Console” as shown below, you are using the wrong application. Close the R app and open the RStudio app instead.



```
R version 4.3.0 (2023-04-21) -- "Already Tomorrow"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin20 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.79 (8225) x86_64-apple-darwin20]

[History restored from /Users/mslifko/.Rapp.history]

>
```

2. RStudio – an integrated development environment (IDE) for R.

    a. RStudio includes an interface that simplifies the process of interacting with R.
    b. Upon opening the RStudio app, you should see a window named “RStudio”, as shown in the figure on the next page. The four green boxes, with a dashed line frame, were added to emphasize the four panes of RStudio. The green boxes will not be visible when you open the RStudio app.



Packages rmarkdown and knitr required but are not installed. Install Don't Show Again

~~~
---
title: "Untitled"
output: html_document
date: "2023-06-16"
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for
authoring HTML, PDF, and MS Word documents. For more details on using R
Markdown see <http://rmarkdown.rstudio.com>.
~~~

Environment is empty

```
R version 4.3.0 (2023-04-21) -- "Already Tomorrow"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin20 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

▲ Name	Size	Modified
.gitignore	51 B	Dec 2, 2022, 10:26 AM
.r		
.Rhistory	20.2 KB	Jun 16, 2023, 2:05 PM
Applications		
Creative Cloud Files		
Desktop		
Documents		
Downloads		
Library		
Movies		
MSE_Calc.R	2.2 KB	Aug 28, 2022, 9:17 PM
Music		
OneDrive – The Pennsylvania State University		
Pictures		
Public		
STAT184_L4Class.nb.html	650.4 KB	Sep 9, 2022, 10:25 AM

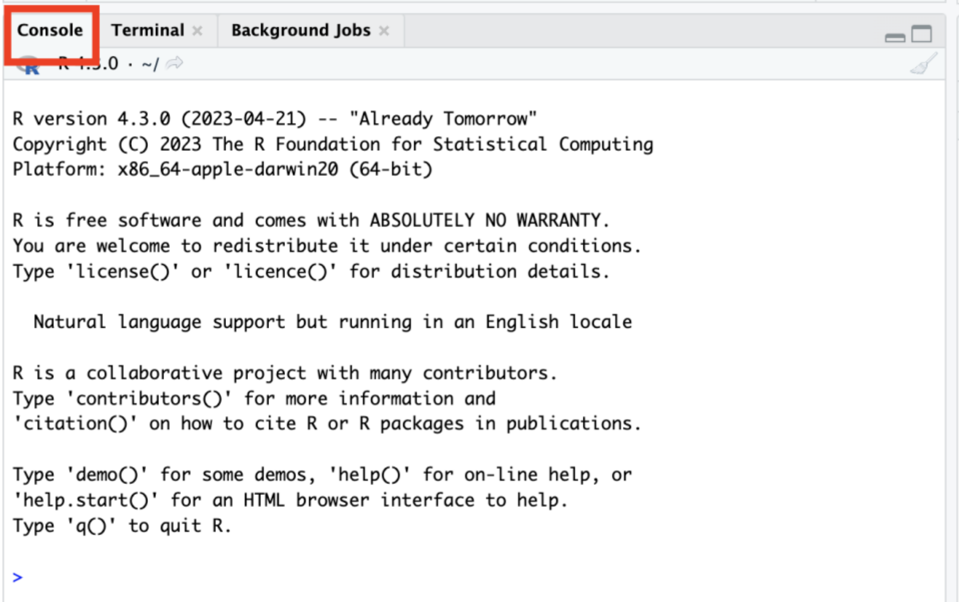
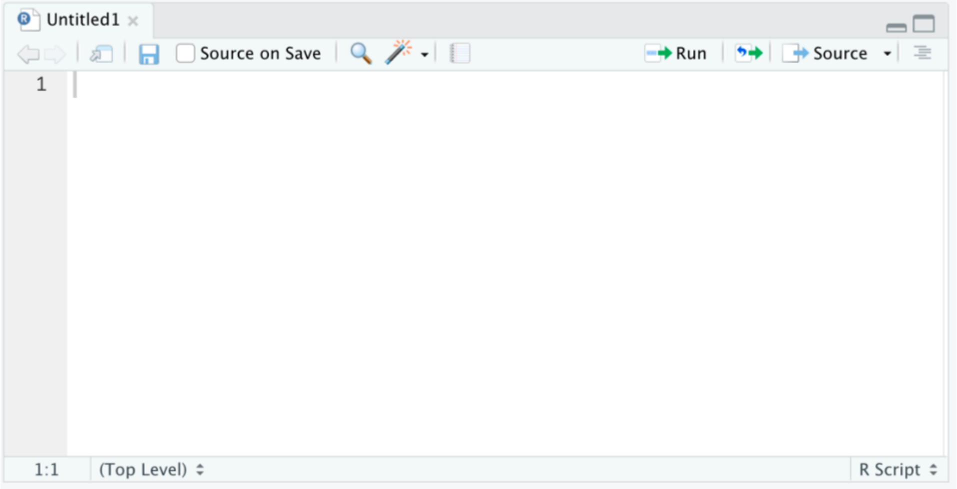
c. Upon using RStudio for the first time, you may only see 3 panes. If this is the case, choose File -> New File -> R Script. This will display the 4th window.

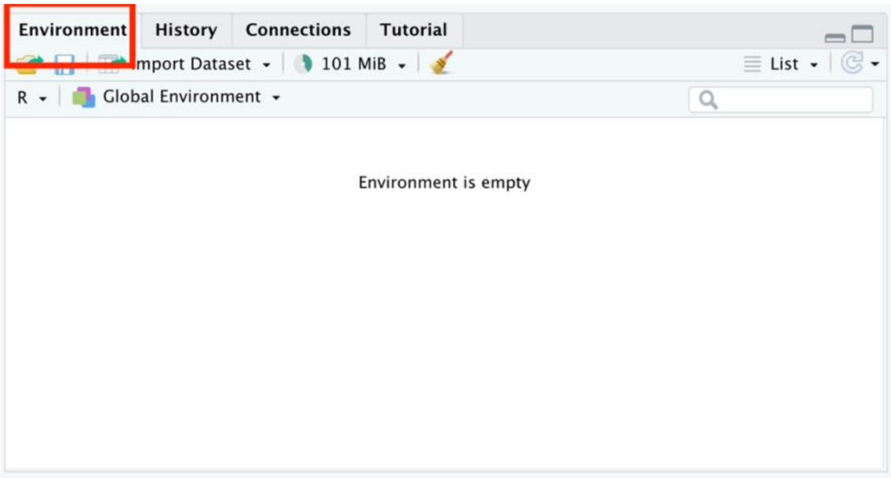
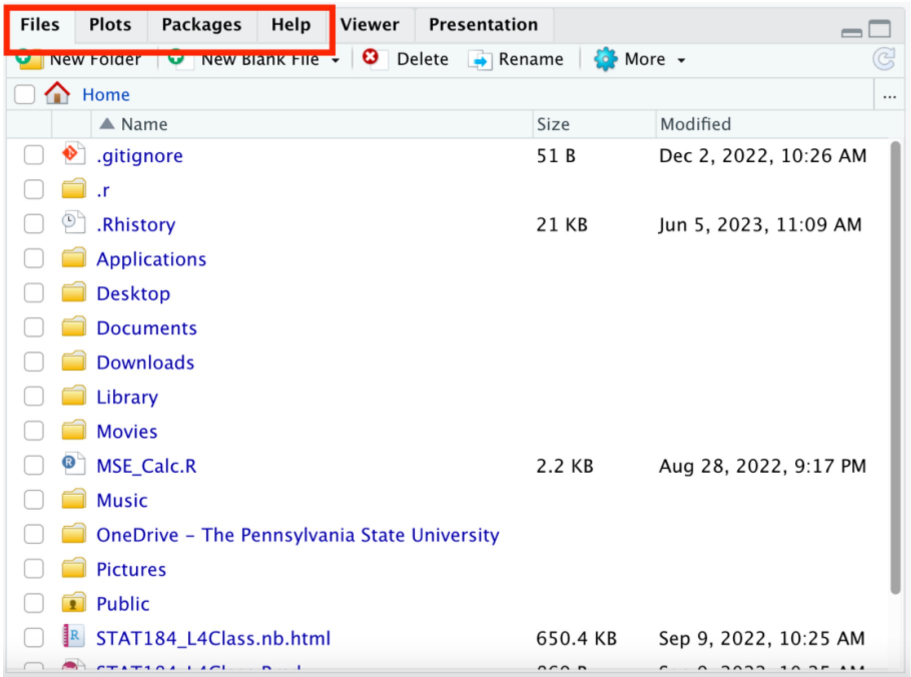
3. RMarkdown – a file format for creating dynamic, reproducible documents which blend code, output, and discussion.

RStudio Tour/Basic Functionality

Before working through the examples that follow, you are encouraged to create a folder for this class. Using either Windows Explorer (Windows) or Finder (Mac), navigate to where you wish to store the folder on your computer. Create a new folder called STAT508. Within the STAT508 folder, you may also want to create folders for Lessons, Data Sets, Code, Data Analysis Activities, etc.

NOTE: As mentioned previously, RStudio consists of four panes. Each pane serves a different purpose. In the following table, we describe the four panes.

Name and Use of Pane	Screenshot of Pane
Bottom Left: **Console** Pane Use: We can enter and run small segments of code in the Console pane. We will also see results, errors, and warnings in this pane.	 R version 4.3.0 (2023-04-21) -- "Already Tomorrow" Copyright (C) 2023 The R Foundation for Statistical Computing Platform: x86_64-apple-darwin20 (64-bit) R is free software and comes with ABSOLUTELY NO WARRANTY. You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details. Natural language support but running in an English locale R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications. Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R. >
Upper Left: **Scripting** Pane Use: The Scripting pane is where we create R Markdown documents and write code that we wish to save. NOTE: If you are not seeing the Scripting pane, choose File -> New File -> R Script.	 Untitled1 Source on Save Run Source 1 1:1 (Top Level) R Script

Upper Right: **Environment** Pane Use: The Environment pane shows the names of objects that you have stored.	 Environment History Connections Tutorial mport Dataset 101 MiB List R Global Environment Environment is empty
Bottom Right: Multi-use Pane Use: This pane serves many purposes including navigating files in the working directory, exploring packages, and searching for the help file documentation. NOTE: We will use the name of the tab (e.g., Help) that we are discussing to refer to this pane.	 Files Plots Packages Help Viewer Presentation New Folder New Blank File Delete Rename More Home

▲ Name	Size	Modified
.gitignore	51 B	Dec 2, 2022, 10:26 AM
.r		
.Rhistory	21 KB	Jun 5, 2023, 11:09 AM
Applications		
Desktop		
Documents		
Downloads		
Library		
Movies		
MSE_Calc.R	2.2 KB	Aug 28, 2022, 9:17 PM
Music		
OneDrive – The Pennsylvania State University		
Pictures		
Public		
STAT184_L4Class.nb.html	650.4 KB	Sep 9, 2022, 10:25 AM

EXAMPLE: (R as a calculator.) R may be used as a calculator by entering calculations directly into the Console pane. In the Console pane, look for the > symbol. This serves as the prompt telling you that R is ready for you to enter a command. In the Console pane, type the R command from each row and press "Enter" or "Return" on your keyboard to obtain the numerical answer.

R Command	**Mathematical Expression**	**Numerical Answer**
`9^2`	9^2	81
`sqrt(36)`	$\sqrt{36}$	6
`exp(3)`	e^3	20.08554

NOTE: `sqrt()` and `exp()` are examples of **functions**. For example, "sqrt" is the name of the square root function and the information inside the parentheses (i.e, 36) is called an argument of the function. With a few exceptions, R functions are almost always followed by a set of parentheses and the corresponding arguments.

NOTE: We often wish to associate values or results of calculations with objects in R. We can accomplish this using assignment operator (`<-`) and statements of the form:

```
object_name <- value
```

There are a few simple rules that apply when creating a name for an object.

1. The object name cannot start with a number, but numbers may be used elsewhere
2. The object name cannot contain punctuation symbols, but there are two exceptions: you can use a period (.) or an underscore (_) in object names.
3. The case of the letters matters (ABC is different than Abc which is also different than abc)

EXAMPLE: (Storing objects.) Create 2 vectors named `Pretest_score` and `Posttest_score` by running each line of code shown below in the Console pane. When using an assignment statement, the object will be stored in the Environment and is listed in the Environment pane.

```
Pretest_score <- c(5,3,10)

Posttest_score <- c(14,17,19)
```

EXAMPLE: (Using stored objects). Using the objects created in the previous example, verify the following:

Task	Numerical Answer
Calculate the improvement in test scores by calculating: `Posttest_score – Pretest_score`	9 14 9
Calculate: `sd(Posttest_score)`.	2.516611

EXAMPLE: What does the function “sd” do? You can access the help file/documentation by running `?sd` OR `help(“sd”)` in the Console pane. This opens the “Help” tab in the bottom right pane.

EXAMPLE: (Common mistakes - typos and case in R). As you begin to type your own commands, you will encounter typos. Run each R command in the Console pane. For each case, the error message is shown. Identify the cause of the error.

R Command	Error Message	Cause of Error
`srqt(36)`	Error in srqt(36)∶ could not find function "srqt"	
`Sqrt(36)`	Error in Sqrt(36) : could not find function “Sqrt”	
`sqrt[36]`	Error in sqrt[36] : object of type 'builtin' is not subsettable	

NOTE: There’s an implied contract between you and R: it will do the tedious computation for you, but in return, you must be completely precise in your instructions. **Typos matter. Case matters.**

NOTE: Quotation marks and parentheses (or other grouping symbols) must always come in pairs. RStudio does its best to help you, but it's still possible to end up with a mismatch. If this happens, R will show you the continuation character "+" in the console.

EXAMPLE: (Common mistake - incomplete command.) Enter the line exactly as shown below in the Console pane and run it:

```
y <- “Hello World!
```

NOTE: The + symbol in the Console window tells you that R is waiting for more input; it does not think you are done yet. This usually means you have forgotten either a " or a close parenthesis (i.e.,)). You can add the missing piece or press ESCAPE (ESC) on your keyboard to start over.

EXAMPLE: (Reading an external dataset into R.) R contains a number of built-in datasets, but we also often analyze external (outside of R) data. There are many ways to read external data into R. One approach is to take the following steps:

a) Download the dataset L01_BirthWt.csv from the U1:L1 module in Canvas and locate the file. (Mac users should avoid using Safari for downloading the dataset from Canvas. Use Chrome or Firefox. Also, avoid saving the file in Numbers.)
b) In the Environment pane (upper right) click on Import Dataset.
c) Select From Text (base).
d) Browse to where you stored the file L01_BirthWt.csv and select Open.
e) Edit the name (upper left) if desired. (Name it Births for this and future examples.)
f) You will see a preview in the Data Frame portion. If it looks reasonable, select IMPORT.
g) PRO TIP: This sequence of actions will cause R to create some code in the Console pane. I highly recommend copying that code into your Scripting pane.

EXAMPLE: (More functions and accessing variables in a data frame.) Enter the following commands in the Console pane. Verify the result of each and explain what each function is doing.

R Command	Output	Explanation
`names(Births)`	[1] "bwt" "gestation" "parity" "age" [4] "height" "weight" "smoke"	
`mean(Births$weight)`	128.4787	
`table(Births$smoke)`	No Yes 715 459	
`table(Births[ , 7])`	No Yes 715 459	

NOTATION: `datasetName$variableName` references the variableName column within a dataset.

NOTATION: `datasetName[a, b]` references the a^{th} row and b^{th} column in a dataset. Since no rows are specified in `Births[ , 7]` all rows in the 7th column are used.

Definition: An R package is a collection of functions, data, and documentation that extends the capabilities of base R.

NOTE: Each time you start a new RStudio session, a small subset of packages is automatically loaded. Since these packages are always available, we call them the Base-R packages.

NOTE: We will often extend the capabilities of the Base-R packages by downloading additional packages. This is a two step process:

1. Download the package to your computer. (This is done ONCE by using the `install.packages()` function OR by using the Install button on the Packages tab in the bottom right pane.)
2. Tell RStudio that you wish to load the capabilities of the desired package. (This is done EACH TIME you restart your RStudio session by using the `library()` function OR by using the checkbox on the Packages tab in the bottom right pane. Only load packages that are necessary for your analysis).

NOTE: The `install.packages()` code is only run one time to download the files to your computer; however, you must use the `library()` command in each R session in which you wish to use the functionality of an add-on package.

NOTE: We can view the packages that are currently installed by selecting the Package tab near the Plots tab (bottom right pane). Packages installed on your computer will be listed and packages that have a check mark are ready for use in your R session.

EXAMPLE: (Installing packages.) Install the `palmerpenguins` and `tidyverse` packages suing the `install.packages()` function and load their functionality using the `library()` function.

NOTE: To check if this worked properly, check the Packages tab in the bottom right pane. If both palmerpenguins and tidyverse have checked boxes, you have successfully downloaded the packages and loaded their capabilities.

NOTE: The `palmerpenguins` package contains a dataset with size measurements (such as bill length, bill depth, flipper length, and body mass), sex, and island for three penguin species observed on three islands in the Palmer Archipelago, Antarctica over a study period of three years. You can read more at: https://education.rstudio.com/blog/2020/07/palmerpenguins-cran/.

NOTE: The `tidyverse` is a collection of R packages that are commonly used for data wrangling and data visualization. Installing the `tidyverse` package installs a number of important packages including `dplyr()` and `ggplot2()`. It is common for students, especially those working on a Mac, to encounter some problems when initially installing the `tidyverse`. If you encounter problems, please contact your instructor for help. (Most importantly, do not let the initial stresses of getting the software set up cause you to panic. We will get through it together.)

R Markdown

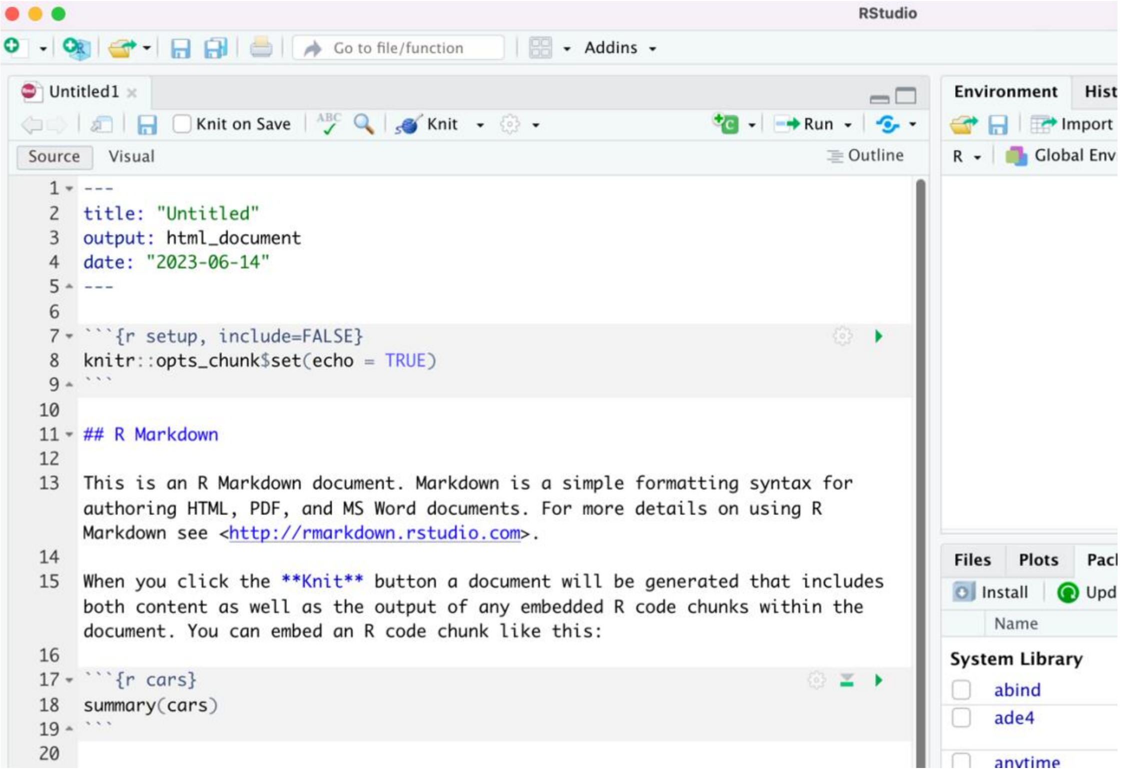
The purpose of data wrangling and visualization is communication: condensing and presenting data in a form that conveys information. An important part of communication is documentation and reporting.

PURPOSE: An important concept in data-driven reporting is reproducibility. The idea is to be able to reproduce your entire document without any manual intervention, and, *more importantly, to be easily able to generate a new report in response to changes in data or revisions in computer commands*. In other words, reproducible reports contain all the information needed to generate a new report. Common document formats such as .pdf, .docx, or .html do not offer support for reproducibility.

Definition: We will use a Markdown tool that will loosely be called RMarkdown. RMarkdown files have a .Rmd file extension, which is a file format for creating dynamic, reproducible documents which blend code, output, and discussion. We can start a new RMarkdown document by going to:

File >> New File >> R Markdown

You may be prompted to install or update one (or more) packages. If so, agree to install/update the packages. (Selecting "Yes" should automatically download the files and no further action is necessary.) A "New R Markdown" window will open. Accept the defaults by selecting the OK button. This produces a new tab, titled Untitled1 in the scripting pane as shown below:



~~~
---
title: "Untitled"
output: html_document
date: "2023-06-14"
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for
authoring HTML, PDF, and MS Word documents. For more details on using R
Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes
both content as well as the output of any embedded R code chunks within the
document. You can embed an R code chunk like this:

```{r cars}
summary(cars)
```

~~~

NOTE: (Structure) .Rmd documents generally require two parts:

1. "YAML" header at the top (designated by --- before/after) includes some document controls:
    a. title
    b. author name
    c. date
    d. output type

2. body of the document is made of various combinations of components such as:
    a. Markdown syntax (like hashtag headers)
    b. Narrative text
    c. Lists (bullets or numbers)
    d. R Code "chunks" (begin with ```{r} and end with ```)
    e. URLs
    f. Tables

EXAMPLE: (R markdown basics) In order to explore some basics of R Markdown, perform the following steps:

a. In your new R markdown script, click the Knit button. This should trigger a prompt to save the file to a location on your computer. Name the file STAT508_L1_RmdDemo and select Save. This will open a new window showing you the document that you just created!!!!

b. If you navigate (using Windows Explorer or Finder) to the folder where you save the file, you should see two files:

    1. STAT508_L1_RmdDemo.Rmd (this is the R code/script in Markdown form)
    2. STAT508_L1_RmdDemo.html (this is the HTML document) – which you can think of as your final document.

c. Delete all code and text from Line 11 to the end of the document.

d. Add a level 2 header called "Front Matter". Add an R chuck that includes a library command for `tidyverse` and the R code for reading in the L01_BirthWt.csv file. Run the R chunk.

e. Add a level 2 called "Regression Example". Using `Births` create a scatterplot showing the birth weight (`bwt`) as a function of the gestation period. Then comment on the nature of the relationship.

f. Build a simple linear regression model for predicting `bwt` as a function of `gestation`. Then, write the estimated equation.

g. Edit the YAML header to include your name, change the title to "Rmd Demo", and Knit the document.

h. (Extra markdown resources) Go to Help >> Markdown Quick Reference. This will show you some useful tips for formatting text (using bold/italics), adding headers, creating lists, inserting a link to a website, inserting a picture that is not created by R, etc.

In many instances, such as for the lecture notes, I will provide with your .Rmd files that include some code covered in Lectures. Be sure to download the files from Canvas and open them in RStudio.

EXAMPLE: (Opening an R markdown file.) At this time, download and open the R markdown file STAT508_U1L1.Rmd.

Exploratory Data Analysis

Overview of Exploratory Data Analysis

Definition: Exploratory Data Analysis (or EDA) is a process that uses a combination of transformations, data visualizations, and numerical summaries to gain insights about a dataset. EDA is often the first step when exploring a dataset.

EDA is a creative process that involves asking and answering questions. The questions often involve exploring:

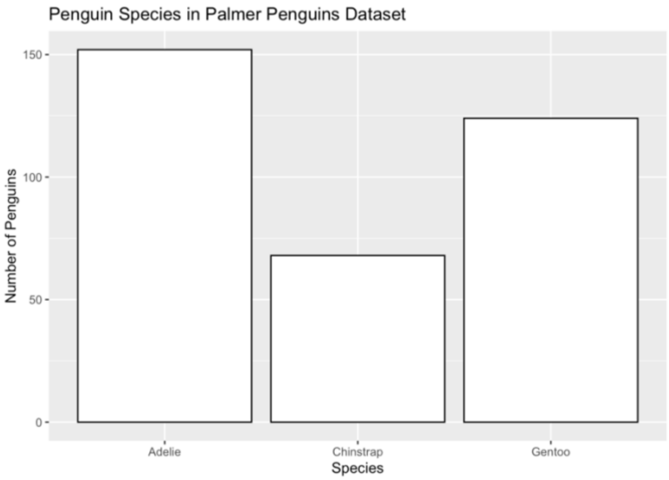
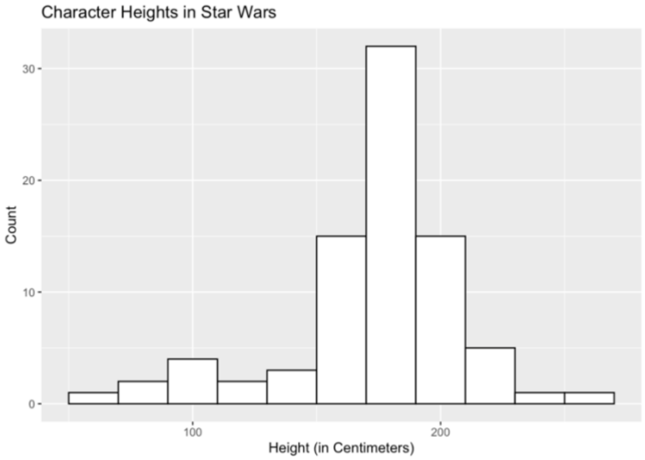
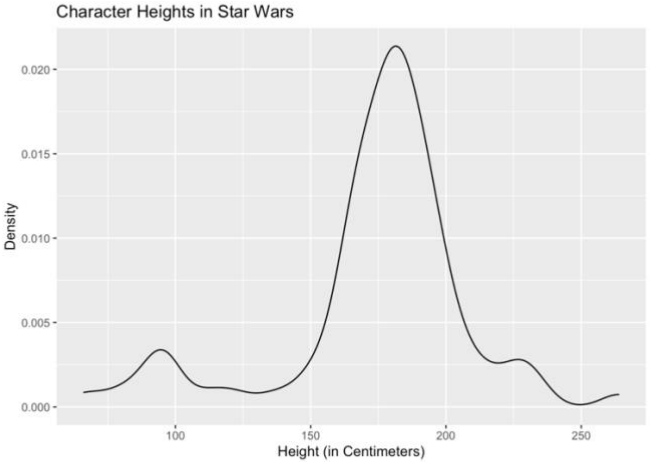
- the distribution of a single variable (univariate analysis)
- the relationship between two or more variables (bivariate or multivariate analysis)

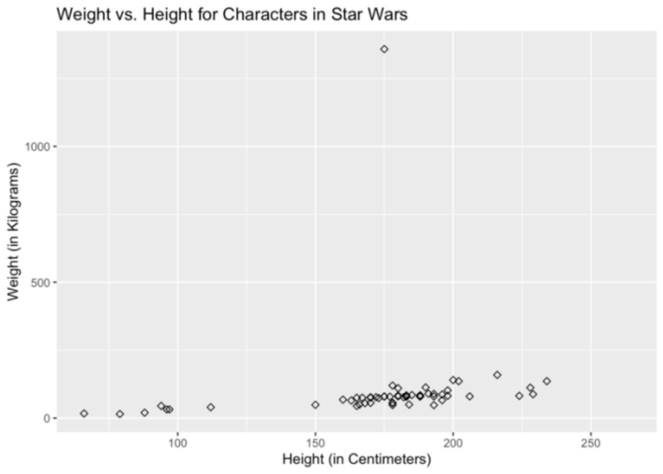
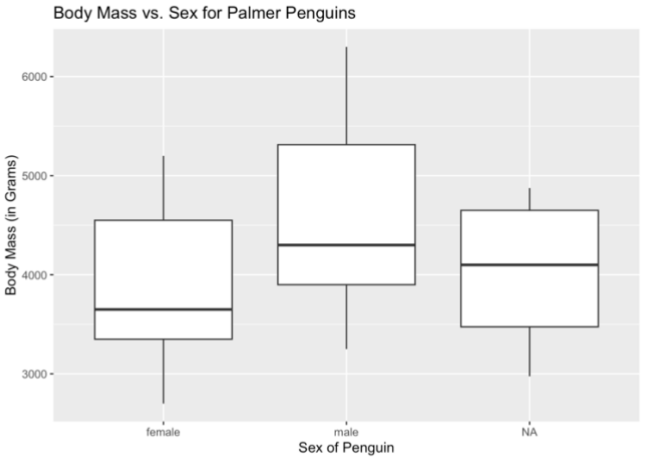
Resource: *R for Data Science* (Wickham and Grolemund) provide a nice overview of EDA: https://r4ds.had.co.nz/exploratory-data-analysis.html#exploratory-data-analysis

Data Visualization and Numerical Summaries

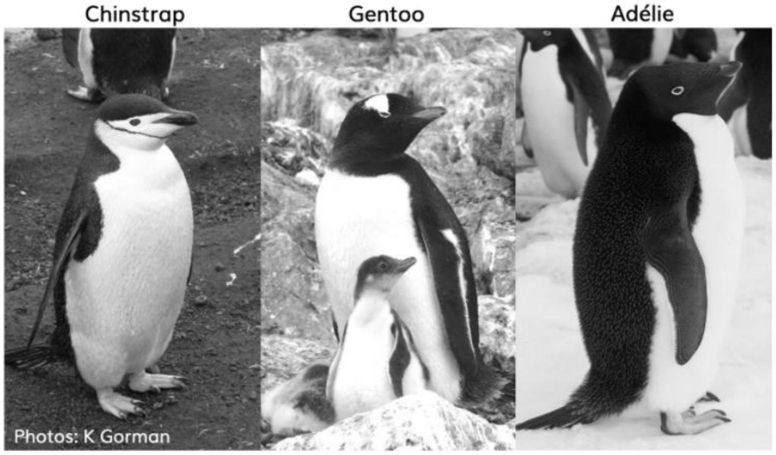
MAJOR IDEA: Throughout this course, you will learn a variety of tools for various purposes (data wrangling, visualization, modeling, etc.). A recurring theme is that you must be able to choose the appropriate tool for a given task. **The choice often depends on the type of data** being used.

EXAMPLE: (Common visualizations.) This example shows a few common data visualizations produced using the `ggplot()` function from the `ggplot2` package within the `tidyverse`. The “Plot Information” column contains the name of the visualization and information about the plot, such as when it is appropriate use the given visualization. The code for creating these plots, along with some customization commands not covered in the notes, are shown in the STAT508_U1L1.Rmd file. Be sure to read through the R code in the markdown file.

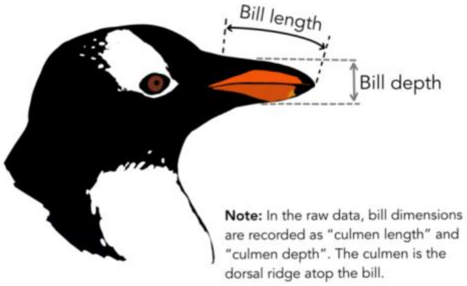
Plot Example	Plot Information
	Name: **Bar Plot/Bar Chart** Command: `geom_bar()` Shows: the number of observations (or relative proportions) of each category of a categorical variable. Use when: visualizing the distribution of a **single categorical variable** (univariate plot)
	Name: **Histogram** Command: `geom_histogram()` Shows: the number of observations that fall into each interval of the variable Use when: visualizing the distribution of a **single quantitative variable** (univariate plot)
	Name: **Density Plot** Command: `geom_density()` Shows: the kernel density estimate, which is like a smoothed histogram. The y-axis is the density (total area under the density curve is 1 like a Probability Density Function). Use when: visualizing the distribution of a **single quantitative variable** (univariate plot)

Plot Example	Plot Information
 Weight vs. Height for Characters in Star Wars 	Name: **Scatterplot** Command: `geom_point()` Shows: the relationship between two quantitative variables measured on the same observations. Use when: visualizing the relationship between **two quantitative variables** (bivariate plot)
 Body Mass vs. Sex for Palmer Penguins 	Name: **Side-by-side boxplots** Command: `geom_boxplot()` Shows: the five-number summary (minimum value, 1st quartile (Q1), median, 3rd quartile (Q3), and maximum) of a quantitative variable for each level of a categorical variable Use when: visualizing the relationship between **one quantitative variable and one categorical variable** (bivariate plot)

NOTE: Several upcoming examples will use the dataset penguins. This dataset, found in the palmerpenguins package, contains size measurements (such as bill length, bill depth, flipper length, and body mass), sex, and island for three penguin species observed on three islands in the Palmer Archipelago, Antarctica over a study period of three years. You can read more at: https://education.rstudio.com/blog/2020/07/palmerpenguins-cran/.



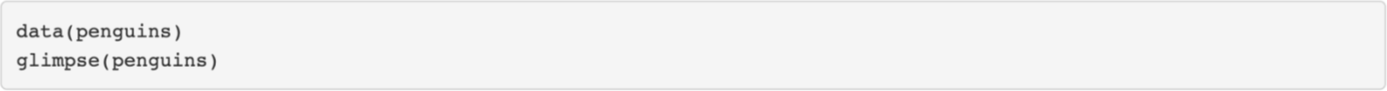
Photos: K Gorman



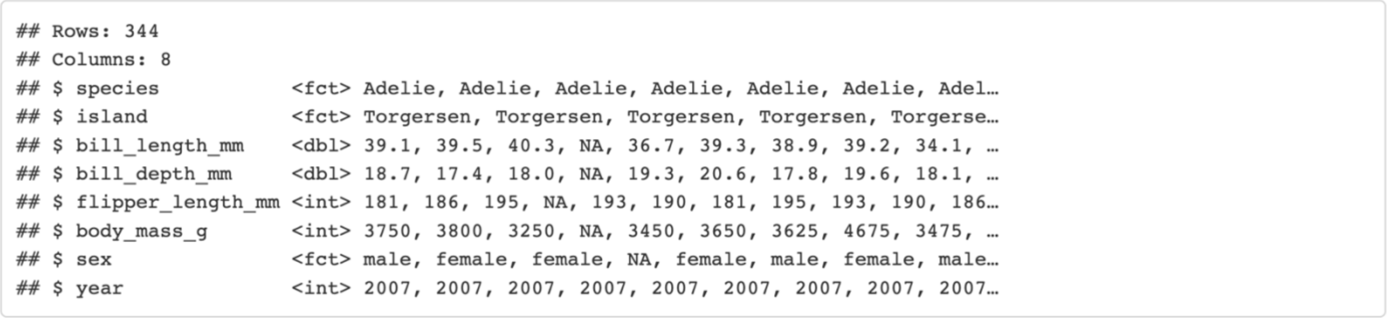
Note: In the raw data, bill dimensions are recorded as "culmen length" and "culmen depth". The culmen is the dorsal ridge atop the bill.

Artwork by @allison_horst

EXAMPLE: (Loading and previewing a dataset from a package.) After loading the palmerpenguins library, create an object named penguins in the Environment using the data() function shown below. Then, preview the dataset by using the glimpse() function from the dplyr library.



```
data(penguins)
glimpse(penguins)
```



```
## Rows: 344
## Columns: 8
## $ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel…
## $ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse…
## $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, …
## $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, …
## $ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186…
## $ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, …
## $ sex               <fct> male, female, female, NA, female, male, female, male…
## $ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007…
```

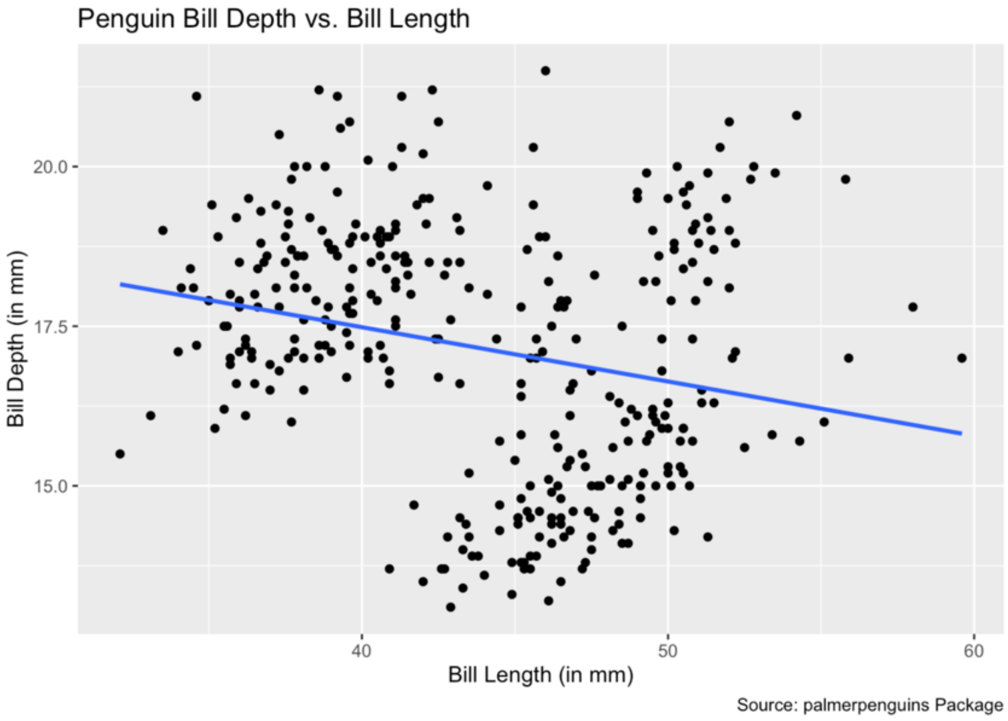
As shown in the output, the glimpse() provides some useful information about the dataset including:

- the number of rows/observations and columns/variables
- the names of the variables
- the type of data contained in each variable
  - <fct> represents a factor (i.e., a categorical variable)
  - <dbl> represents a double (i.e., a number including decimal places)
  - <int> represents an integer
  - <chr> (not pictured above) represents a character data
- a preview of the first few values in each variable

While R includes many tools for data visualization, the `ggplot()` function from the `ggplot2` package is one of the most versatile methods for visualizing data.

EXAMPLE: (Layers of ggplot demonstration.) To fully understand how `ggplot()` uses layers to build a plot, enter and run one line at a time. Then, comment on you have learned from the plot.

```
ggplot(data = penguins,
    mapping = aes(x = bill_length_mm,
                               y = bill_depth_mm)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
     labs(x = "Bill Length (in mm)",
     y = "Bill Depth (in mm)",
     title = "Penguin Bill Depth vs. Bill Length",
     caption = "Source: palmerpenguins Package")
```



NOTE: More advanced examples will follow, but `ggplot()` function calls often start with the following foundation:

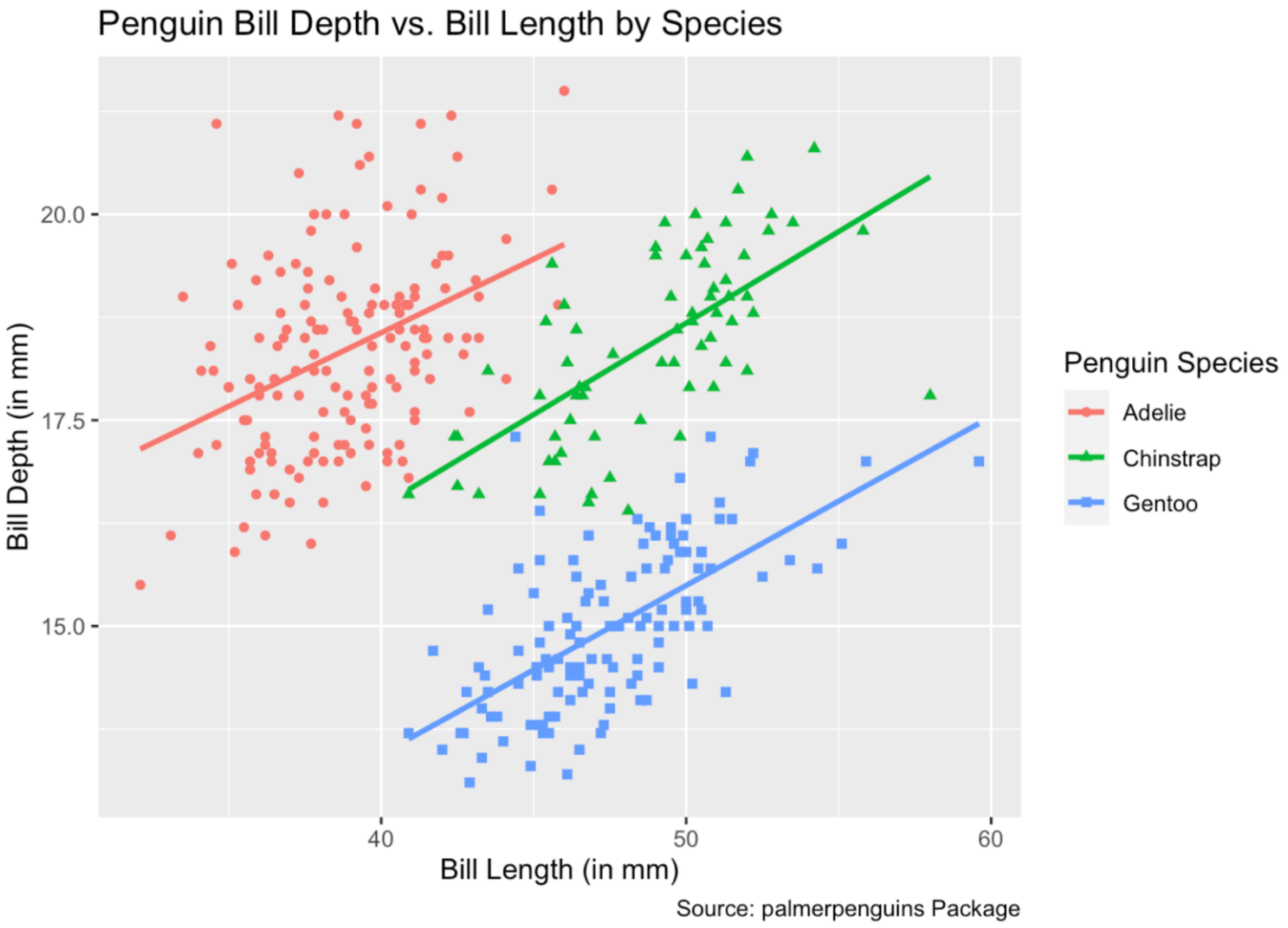
```
ggplot(data = [dataset],
     mapping = aes(x = [x-variable], y = [y-variable])) +
  geom_xxx() +
  other options
```

Observations:

- `ggplot()` creates a coordinate system that you can add layers to.
- The first argument of `ggplot()` is the dataset to use in the graph. So `ggplot(data = penguins)` creates an empty graph, but it's not very interesting
- The `mapping` argument defines how variables in your dataset are mapped to visual properties.
- The `mapping` argument is always paired with the `aes()` (short for aesthetic) function, and the `x` and `y` arguments of `aes()` specify which variables to map to the x and y axes.
- Aesthetics include details like the x/y axis variables and the size, shape, or color of your points.
- We complete the graph by adding one or more layers to `ggplot()`. For example, the `geom_point()` adds a layer of points to your plot, which creates a scatterplot.
- `ggplot2` comes with many geom functions (such as `geom_point()`, `geom_boxplot()`, `geom_histogram()`, etc.) that each add a different type of layer to a plot.
- Resource: A `ggplot2` cheat sheet with many details, including various geom functions and their options, may be found at: https://ggplot2.tidyverse.org/

NOTE: Aesthetics are an important part of the `ggplot()` functionality. Beyond mapping variables to the axes, aesthetics allow characteristics of the symbols on the plot to be mapped to a specific variable in the data. Examples of aesthetics include color (or colour), shape, size, alpha (controls the transparency). Using aesthetics is one method for representing more than 2 variables in a given visualization.

EXAMPLE: (Using aesthetics.) Create an appropriate plot for visualizing the bill depth as a function of the bill length. Add an aesthetic (here, use color and shape) to incorporate the categorical variable species. Then comment on what you have learned about the relationship between bill depth, bill length, and species.

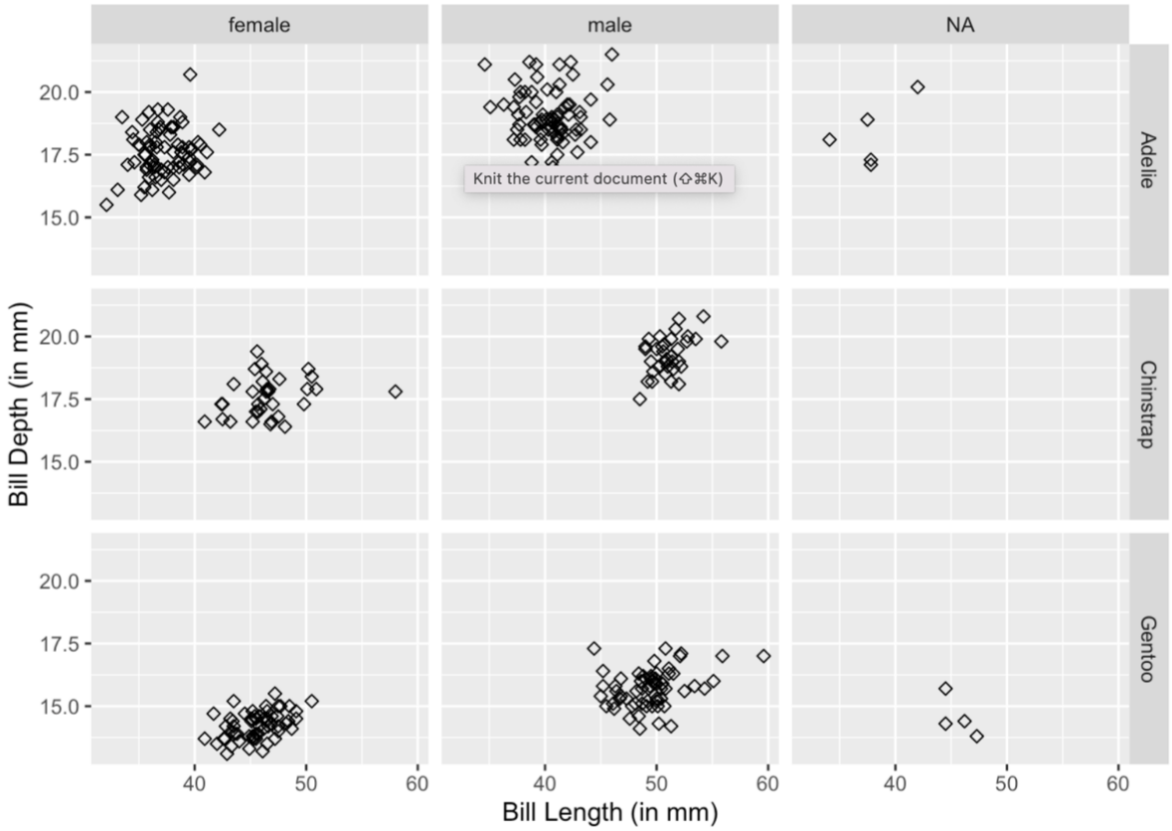


CAUTION: Using multiple aesthetics such as shape, color, and size to display multiple variables can produce a confusing, hard-to-read graph.

SOLUTION: Instead, facets—multiple side-by-side graphs used to display levels of a categorical variable—provide a simple and effective alternative.

EXAMPLE: (Faceting using two categorical variables.) The following code allows us to investigate the relationship between bill depth, bill length, sex, and species. To do this, we will facet on species and sex.

```
ggplot(data = penguins, mapping = aes(x = bill_length_mm, y = bill_depth_mm)) +
  geom_point(shape = 5) +
  facet_grid(species ~ sex) +
  labs(x = "Bill Length (in mm)",
       y = "Bill Depth (in mm)")
```

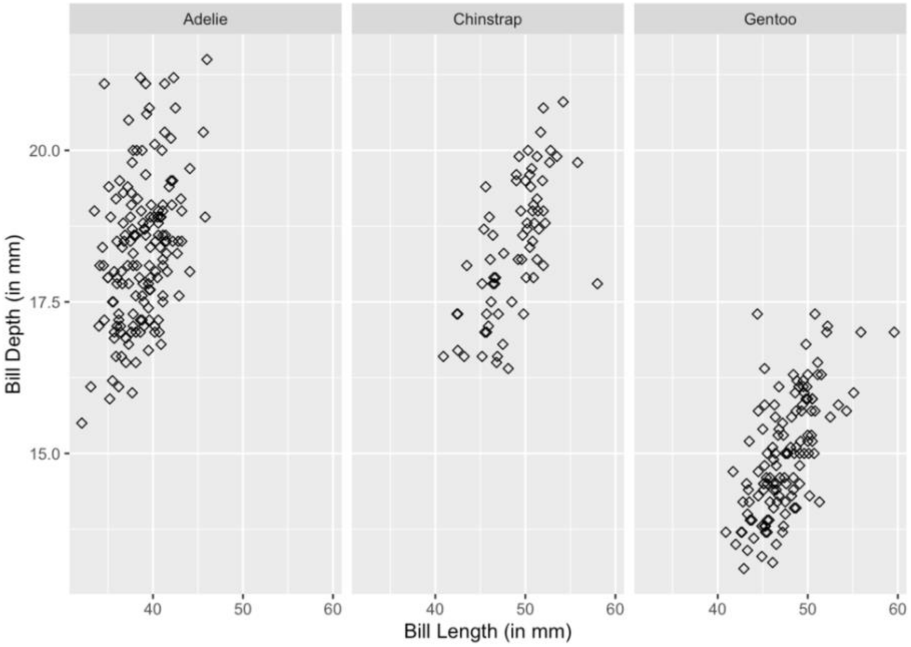


Observations:

- In the `facet_grid(species ~ sex)` command, the variable to the left of the tilde (`~`) controls the rows of the grid, while the variable to the right of the tilde controls the columns.
- The `shape = 5` command controls the shape of the plotting symbol. This is a visual feature of the symbols on the plot; however, since we are fixing a value for the shape instead of controlling it by the values of a variable, the command does not go inside the `aes()` function. Enter it in the `geom_point()` function directly.
- Resource: For a complete list of shapes, please see: http://www.sthda.com/english/wiki/ggplot2-point-shapes

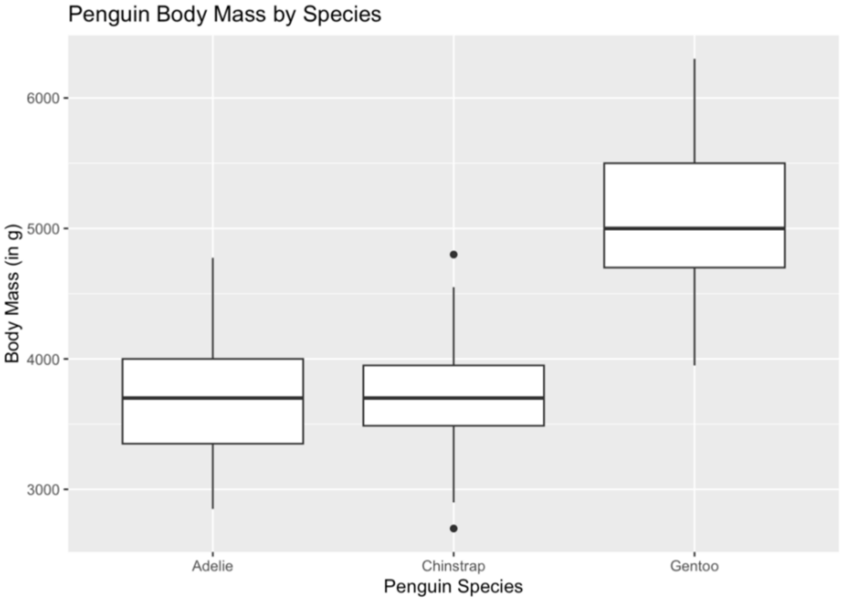
EXAMPLE: (Faceting using one categorical variable.) Repeat the previous example but only facet on species to produce the plot shown below. Since species is the column variable, but no column variable is needed, replace the row variable name by a period (.) in the `facet_grid()` command.

```
ggplot(penguins, aes(x = bill_length_mm, y = bill_depth_mm)) +
  geom_point(shape = 5) +
  facet_grid( . ~ species) +
  labs(x = "Bill Length (in mm)",
       y = "Bill Depth (in mm)")
```

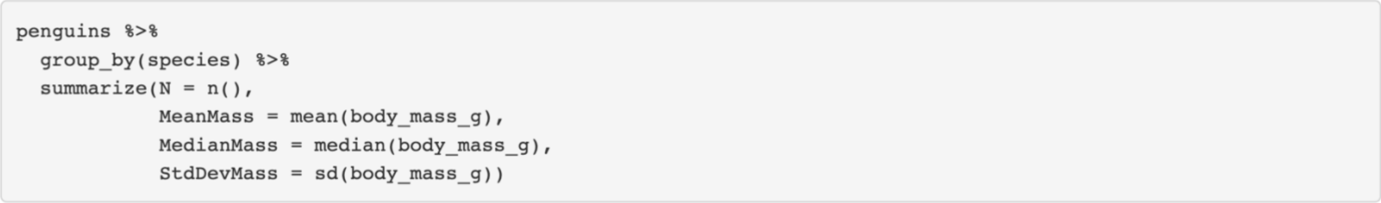


EXAMPLE: (Supporting data visualization with numerical summaries.) Suppose we wish to explore whether a penguin's species is related to its body mass.

a. Explain why a scatterplot is not appropriate for this problem and recreate the plot below.



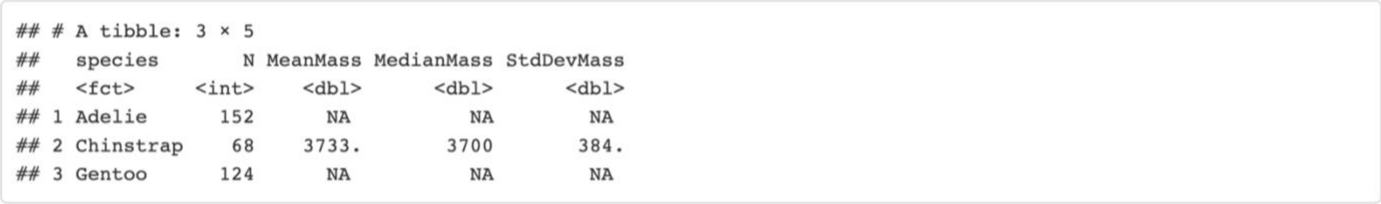
b. In addition to the side-by-side boxplots generated Part a., we would support the visualization with summary statistics. For each species, find the number of penguins and the mean, median, and standard deviation of the body masses. Modify the given code given below to eliminate the NA's.



```
penguins %>%
  group_by(species) %>%
  summarize(N = n(),
            MeanMass = mean(body_mass_g),
            MedianMass = median(body_mass_g),
            StdDevMass = sd(body_mass_g))
```

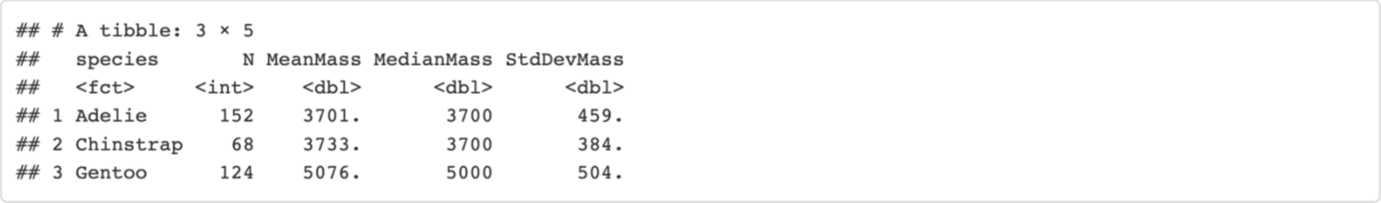
NOTE: `group_by()` paired with `summarize()` finds summary statistics for each level of the categorical variable included in the `group_by()` statement (or for each combination of categorical variables if more than one is listed).

Unfortunately, some of the values are missing and reported as NA's.



```
## # A tibble: 3 × 5
##   species       N MeanMass MedianMass StdDevMass
##   <fct>     <int>    <dbl>      <dbl>      <dbl>
## 1 Adelie      152       NA         NA         NA
## 2 Chinstrap    68    3733.       3700       384.
## 3 Gentoo      124       NA         NA         NA
```

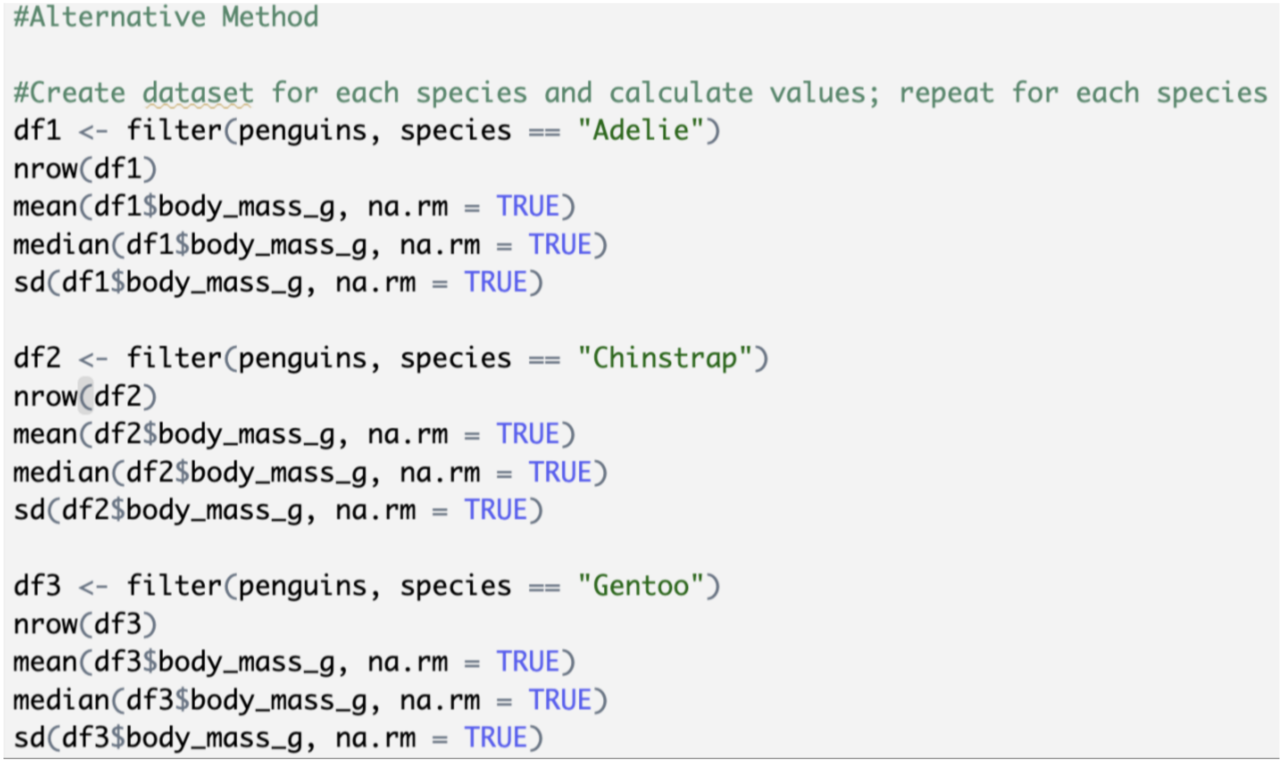
The corrected output is shown below.



```
## # A tibble: 3 × 5
##   species       N MeanMass MedianMass StdDevMass
##   <fct>     <int>    <dbl>      <dbl>      <dbl>
## 1 Adelie      152    3701.       3700       459.
## 2 Chinstrap    68    3733.       3700       384.
## 3 Gentoo      124    5076.       5000       504.
```

c. Based on the boxplots, what have you learned about the relationship between body mass and the species of the penguins?

EXAMPLE: (Alternative methods for finding the summary statistics.) An alternative method is presented below for finding the summary statistics that were calculated in the previous example. Explain why this alternative method is less efficient than using `group_by()` paired with `summarize()`.



```
#Alternative Method

#Create dataset for each species and calculate values; repeat for each species
df1 <- filter(penguins, species == "Adelie")
nrow(df1)
mean(df1$body_mass_g, na.rm = TRUE)
median(df1$body_mass_g, na.rm = TRUE)
sd(df1$body_mass_g, na.rm = TRUE)

df2 <- filter(penguins, species == "Chinstrap")
nrow(df2)
mean(df2$body_mass_g, na.rm = TRUE)
median(df2$body_mass_g, na.rm = TRUE)
sd(df2$body_mass_g, na.rm = TRUE)

df3 <- filter(penguins, species == "Gentoo")
nrow(df3)
mean(df3$body_mass_g, na.rm = TRUE)
median(df3$body_mass_g, na.rm = TRUE)
sd(df3$body_mass_g, na.rm = TRUE)
```

Explanation of code and inefficiency:

- The `filter()` command selects the observations/rows in `penguins` for which the value of `species` matches the species indicated inside the quotation marks. The resulting dataset is stored as `df1`, `df2`, etc.
- For each dataset, four distinct commands are used to calculate the count, mean, median, and standard deviation, respectively.
- This code requires 5 lines of code (4 summary statistics + 1 filter) for each of the 3 species.
- Using `group_by()` paired with `summarize()` is more efficient because the code is not dependent on the number of levels in species. Whether there are 3 species or 30 species, the `group_by()` paired with `summarize()` code would be the same; however, the “Alternative Method” would require 5 x 30 or 150 lines of code if there were 30 species.

Data Wrangling

Definition: Data wrangling is the process of reforming, summarizing, and combining data to make it more suitable for a given purpose.

NOTE: The `tidyverse` is a collection of R packages that are commonly used for data wrangling and data visualization. The `dplyr` package, which is part of the `tidyverse`, contains a number of important data wrangling functions. In particular, we regularly use the following "**dplyr verbs**":

- `filter()` – select a subset of rows (observations), often according to their values
- `arrange()` – reorder/sort the rows
- `select()` – select a subset of columns (variables)
- `mutate()` – add (e.g., create new variables) or modify existing columns
- `rename()` – change the names of columns (variables)
- `summarize()` – aggregate data across rows
    - often paired with `group_by()` to find the summary statistics for each level of a categorical variable

Resource: The *Data transformation with dplyr cheatsheet* at https://posit.co/resources/cheatsheets/ is a great resource.

EXAMPLE: (dplyr verbs) Consider the following information for consultants working on a project. The variables include the employee ID number, the worker level (entry-level, mid-level, or senior-level), the employee's hourly wage in dollars, and the hours worked on a given project. You may assume the data frame containing this information is named `df` in R. For each R command listed below, indicate the resulting dataset.

EmployeeID	Level	HourlyWage	Hours
5073	Entry	20	10
4059	Senior	80	3
7941	Senior	60	5
4909	Senior	100	2

a. R command: `select(df, EmployeeID, HourlyWage) OR`

```
df %>%
  select(EmployeeID, HourlyWage)
```

EmployeeID	Level	HourlyWage	Hours
5073	Entry	20	10
4059	Senior	80	3
7941	Senior	60	5
4909	Senior	100	2

b. R command: `filter(df, Level == "Entry") OR`

```
df %>%
  filter(Level == “Entry”)
```

EmployeeID	Level	HourlyWage	Hours
5073	Entry	20	10
4059	Senior	80	3
7941	Senior	60	5
4909	Senior	100	2

c. R command:

```
df %>%
  filter(Level != “senior”) %>%
  rename(ID = EmployeeID)
```

EmployeeID	Level	HourlyWage	Hours
5073	Entry	20	10
4059	Senior	80	3
7941	Senior	60	5
4909	Senior	100	2

d. R command:

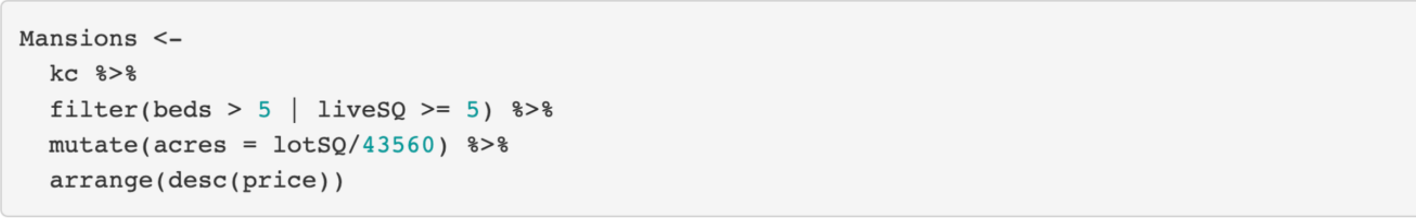
```
df %>%
  mutate(Pay = HourlyWage * Hours)
```

EmployeeID	Level	HourlyWage	Hours
5073	Entry	20	10
4059	Senior	80	3
7941	Senior	60	5
4909	Senior	100	2

e. R command:

```
df %>%
  summarize(MeanHours = mean(Hours)
```

EXAMPLE: (Command chains) It is common to combine multiple dplyr verbs into a single command chain. The chain starts with `kc`, which represents 1000 randomly selected properties for in King County (Seattle, Washington). The chain creates a new data frame called Mansions by selecting properties that have more than 5 bedrooms or at least 5 thousand square feet of living space, creates a new variable called acres that represents the lot size in acres instead of square feet, and sorts the properties in descending order of price.



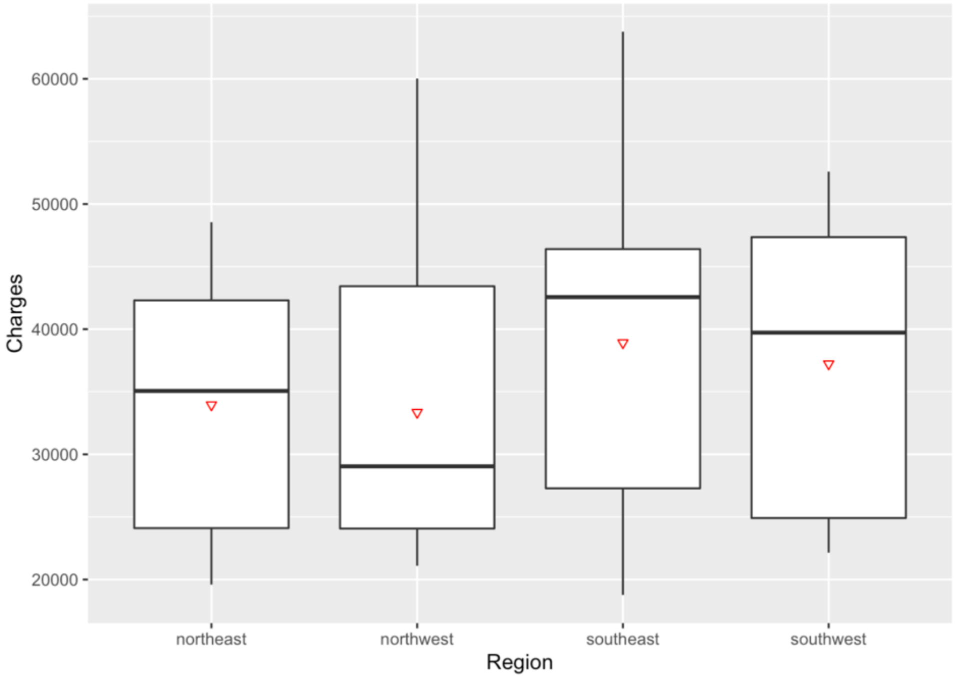
```
Mansions <-
  kc %>%
  filter(beds > 5 | liveSQ >= 5) %>%
  mutate(acres = lotSQ/43560) %>%
  arrange(desc(price))
```

- Each link in the chain is a “data verb” or “data move” with its arguments
  - The very first link is typically a data table/data frame. Here, it starts with `kc`.
  - Links are connected by the pipe: `%>%`
- Often, but not always, you will store the result of the chain in a named object
  - This is done with the assignment operator, `<-`
- Use a new line for each link
- Note that `%>%` is at the end of each line. **Except**
  - `Mansions <-` is an assignment statement
  - Last line has no `%>%` (otherwise R would expect more commands.)
- In the `filter()` command, the vertical bar (`|`) is interpreted as OR. This command is selecting properties that have more than 5 bedrooms OR at least 5 thousand square feet of living space. If you wanted to use AND, you would use the ampersand symbol (`&`).

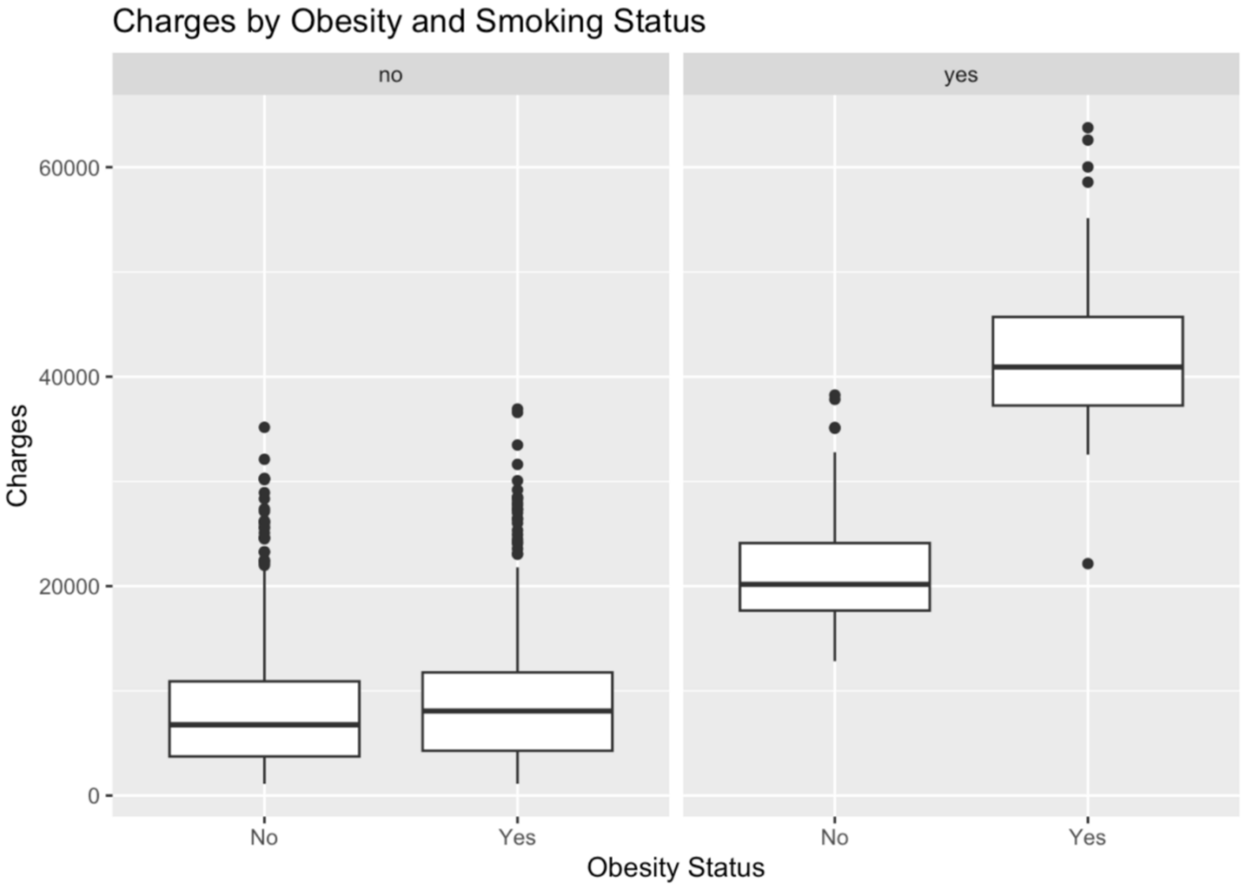
EXAMPLE: (dplyr practice in R) Download L01_Insurance_m.csv from Canvas and the read dataset into R. The dataset contains information about a number of health insurance policies. In particular, the data set contains some attributes of the policy holder (such as age, sex, etc.) and the total charges billed by the health care provider. The variables are:

- age: age of primary beneficiary
- sex: sex of primary beneficiary
- bmi: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height, objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9
- children: Number of children covered by the health insurance policy (i.e., the number of dependents)
- smoker: Status indicating whether the person is a smoker (options include ‘yes’ and ‘no’)
- region: the beneficiary's residential area in the US (options include northeast, southeast, southwest, northwest).
- charges: Individual medical costs as billed by health insurance

a. Suppose we wish to determine whether the medical charges depend on the region for smokers over 40. To answer this, calculate appropriate summary statistics and create a supporting data visualization. (One possible visualization is shown below.)



b. Suppose we are interested in whether smoking status and a person’s obesity classification affect medical charges. To answer this, recreate the plot shown below. What have you learned about the relationship between the variables? NOTE: Obese takes on a value of “Yes” if the person’s BMI is 30.0 or higher.



NOTE: The `ifelse()` function is useful when creating new variables. The command form is:

```
ifelse(check a condition, value if true, value if false)
```