Descriptive statistics

Sara Souther

2024-09-20

Contents

Chapter 1

Ecological data analysis in ENV 226 laboratory

This course surveys the central concepts in ecology: evolution, population dynamics, community interactions, biogeochemical cycling, and limiting factors, as well as how those factors are measured, quantified, and interact with drivers of global environmental change. This course is required for the B.S. in Environmental Sciences degree program, and also the B.S. in Environmental and Sustainability Studies degree. This course acquaints students with foundational concepts and theories in ecology and provides a broad basis for more advanced courses in subdisciplines and applications of ecology.

A critical part of ecological research is developing practical and analytic skills. Most ecologists and data scientists use R statistical software. Learning how to manage, manipulate and analyze data in R will serve your undergraduate career and beyond! In ENV 226 lab, we will ease you into using R for all your data needs!

1.1 How to use this resource

Each chapter in this online book corresponds to a lesson that will help you in lab. When starting a new chapter, create a new R script with a name that allows you to easily connect the content in the R script to the chapter. Then, copy and paste sections of code in the chapter into your R script to practice!

```
knitr::opts_chunk$set(warning = FALSE, message = FALSE)
```

Chapter 2

Welcome to R!

R is an open source statistical software package commonly used by researchers and other folks, who crave a free way to manipulate, analyze, and visualize date. R uses it's own programming language, which is similar to S+ (the paid precursor to R). R employs an object-oriented programming (OOP) paradigm, specifically a type of OOP known as "class-based object-oriented programming," to manage and manipulate data. In R, objects are fundamental entities, and you work with data and functions through objects. Let's walk through the basics of installing and using R!

2.1 R and R studio installation

You will first want to download R statistical software and R studio, which is a powerful program that interfaces with R to make your coding experience more organized and enjoyable. Notice that you need to select a version of R depending on your operating system.

2.2 Download the R statistical software from the official R Project website.

Open your web browser and go to the official R Project website at https://www.r-project.org/.

Choose a CRAN Mirror: On the R Project website's main page, you'll see a section that says "Download and Install R." Click on the link that says "CRAN (Comprehensive R Archive Network)." This will take you to the CRAN website.

1. Select Your Mirror: On the CRAN website, you'll find a list of mirrors (servers) from which you can download R. Choose a mirror that is geographically close to your location, as this will generally provide faster download speeds. Click on the mirror's link.

2. Download R for Your Operating System: On the mirror's page, you'll see options to download R for various operating systems (e.g., Windows, macOS, Linux). Click on the appropriate link for your operating system.

3. Choose the Latest Version: You'll typically see multiple versions of R available for download. It's recommended to choose the latest stable version unless you have a specific reason to use an older version.

4. Download and Install: After clicking on the download link, the installation file for R will begin downloading. Once the download is complete, run the installer and follow the installation instructions for your operating system.

5. Start Using R: After the installation is complete, you can launch R from your computer. Depending on your operating system, you may also have an option to install RStudio, a popular integrated development environment (IDE) for R, to enhance your R programming experience.

2.3 Download R studio

Now download R studio!

1. Visit the RStudio Website: Open your web browser and go to the official RStudio website at https://www.rstudio.com/.

2. Download RStudio: On the RStudio website's main page, click on the "Products" menu at the top, and then select "RStudio" from the dropdown menu.

3. Choose the RStudio Edition: RStudio offers different editions, including RStudio Desktop (for use on your local machine), RStudio Server (for remote access), and RStudio Workbench (formerly known as RStudio Server Pro, designed for collaboration and sharing in enterprise environments). You will want to choose the free version, RStudio Desktop.

4. Download the Installer: After selecting the edition, you'll be directed to a page with download options. Click on the download link for your operating system (e.g., Windows, macOS, Linux).

5. Download and Install: The installation file for RStudio will begin downloading. Once the download is complete, run the installer and follow the installation instructions for your operating system.

Start Using RStudio: After the installation is complete, you can launch RStudio from your computer. You'll have access to a powerful IDE that provides a user-friendly interface for working with R, including code editing, interactive R console, data visualization, and more. RStudio greatly enhances your R programming and data analysis experience, and it's widely used by R users for its features and capabilities.

2.4 Set up R studio

Alright, now that you've downloaded R and R studio, open R studio. You can customize the panes that you are visualizing in R.

In RStudio, the four panels or panes are commonly referred to as:

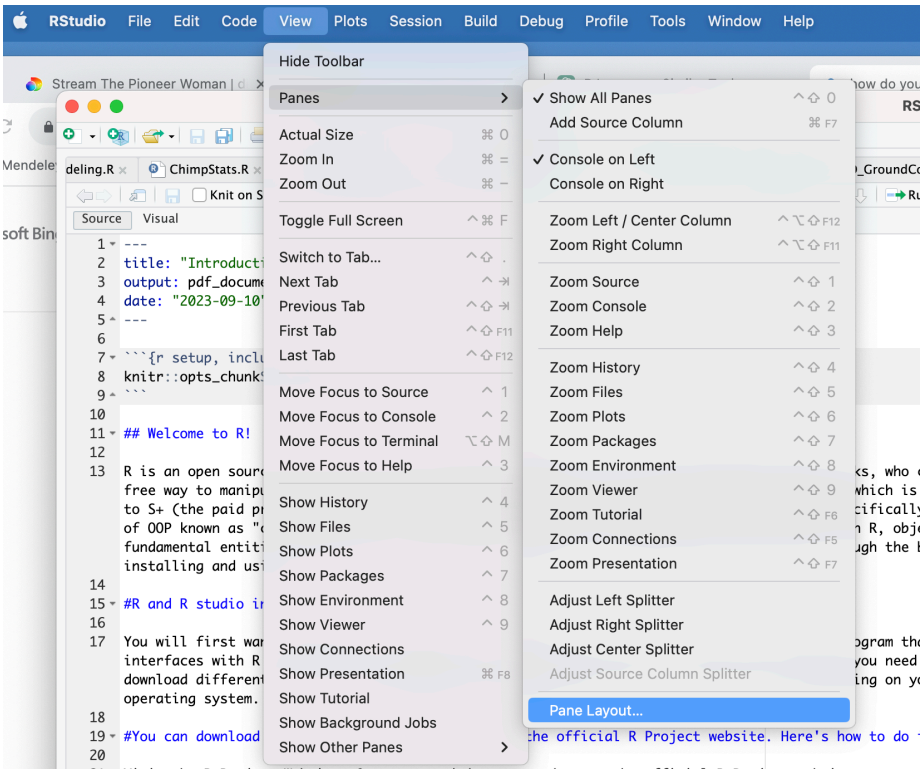
Source Pane: This is where you can write, edit, and save your R scripts and code files. It is typically used for script development and editing. You can open and create new R script files in this pane.

Console Pane: The console is where you interact with R directly. You can execute R commands and see their output here. It's an interactive environment where you can test and run R code line by line or in batches.

Environment Pane: The environment pane displays information about the objects, data frames, variables, and functions currently loaded in your R session. You can also use this pane to view data frames in a spreadsheet-like format and manage your workspace.

Files/Plots/Packages/Help Pane: This pane has multiple tabs and serves various purposes: *Files*: It shows the file system of your project, allowing you to navigate and manage files and directories. *Plots*: When you create plots in R, they will appear in this tab. You can interact with and export the plots from here. *Packages*: This tab displays information about installed packages, and you can use it to install, update, or load packages. *Help*: When you need documentation or help for R functions or packages, you can use the Help tab to search for and view documentation.

Typically, I select a structure in which I have my **Source pane** in the upper left, my **Console Pane** in the lower left position, my **Environment Pane** in the upper right corner, and the **Files/Plots/Packages/Help Pane** in the lower right position. You can select any position that you'd like, but if we create the same work environment, it will be easy for me to direct you when we are trouble-shooting code. To adjust the panels positions, use the pane layout function. Here's what that looks at for a Mac, but typically this arrangement is the default positioning for panels in R studio, so you likely won't have to adjust positioning!



2.5 Good housekeeping

When you are coding in R, you will want to save your R or R markdown scripts and any other data files (e.g., .csv or spatial files) that you are analyzing in a common file. With good housekeeping, you will be able to seamlessly rerun your analyses at in point in time, allowing you to pick back up on projects that may have been dormant!

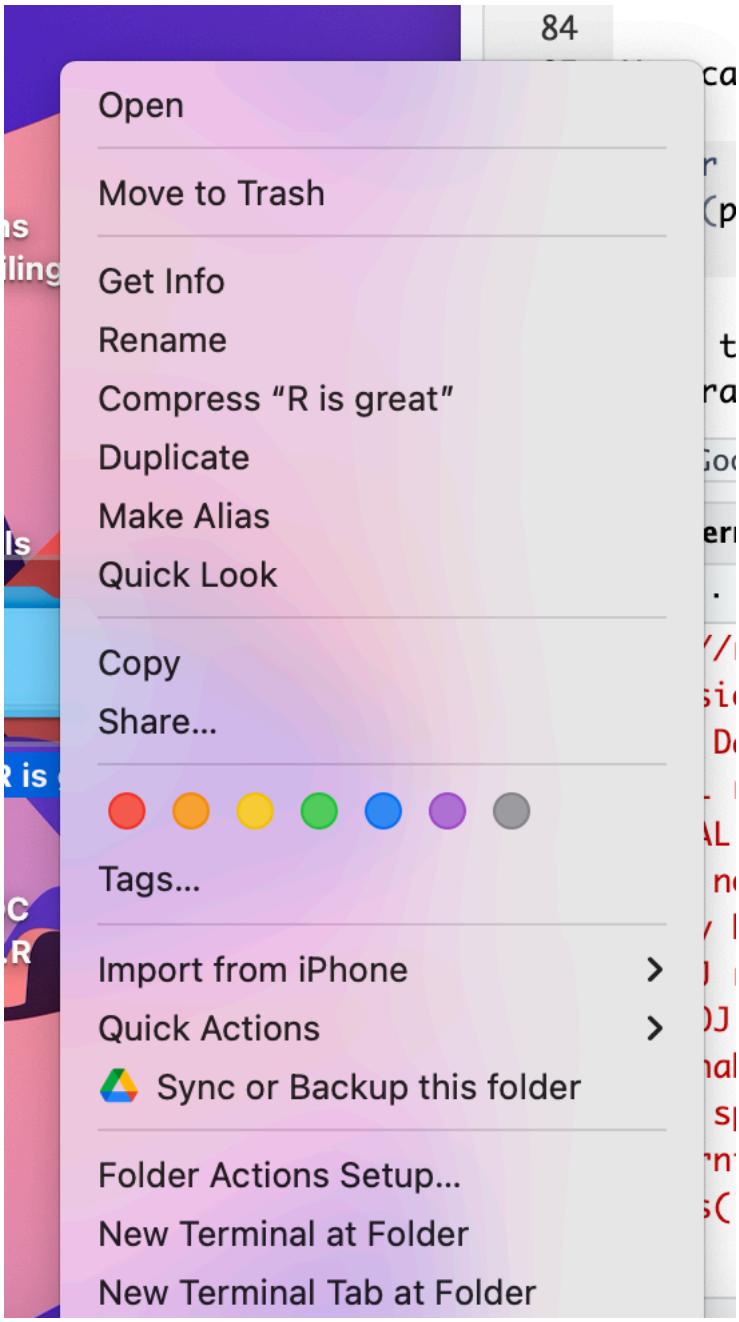
The first step in good R housekeeping is to set a **working directory**. A working directory tells R where to look for and save files. As an example, let's say I am planning to save everything associated with this tutorial to a file called Class1_IntroToR. To do this, create a new folder on your desktop called 'R is great'. You can change your working directory using the R studio interface by selecting a working directory at the top of the console panel in the "Files" tab. If you want to change your working directory to a different location, click on the "..." (ellipsis) button in RStudio's "Files" tab. This will allow you to browse your file system and select a new directory as your working directory. That said, you will be *far* better served by including code in your R script that directs R to your working directory. I prefer to set it within the code in order to allow you to instantaneously be able to pick up work where you left off rather

than searching through files and trying to remember how you set up the code. You can view your working directory by running a simple bit of code (run code below).

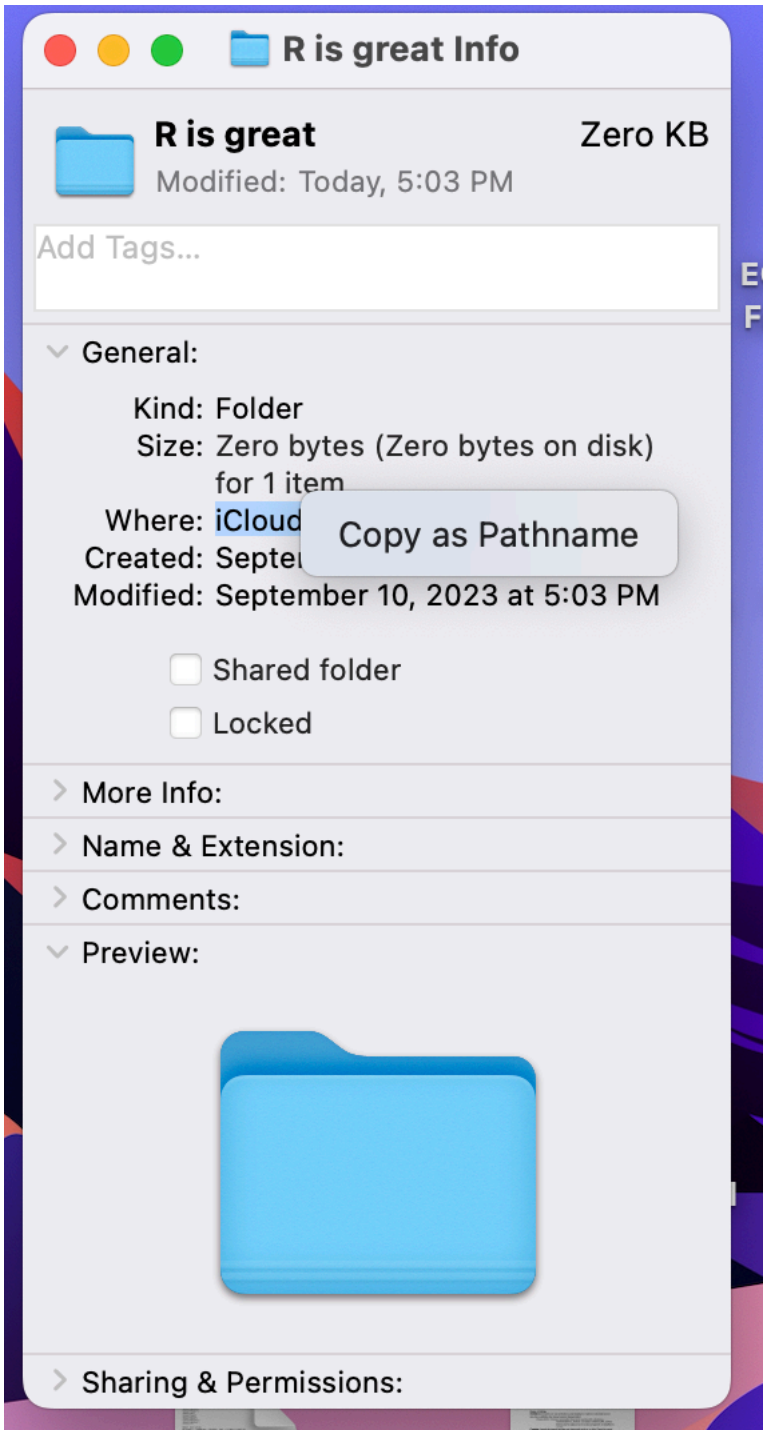
```
getwd() #display file path to R studio
```

```
## [1] "/Users/sks379/Desktop/ENV226LabRManual"
```

When you run, getwd() you will see where R is looking for files. Now, let's tell R where we want it to access files from! First, you will need to identify your file path. To find the file path to the Class1_IntroToR on a mac, double click on the file and should see several option, including 'Get info' (check out picture below).



Then, select 'Get info'. Then, highlight the information after 'where' and copy it as a pathname (see picture).


R is great Info
R is great
Zero KB
Modified: Today, 5:03 PM
Add Tags...
General:
Kind: Folder
Size: Zero bytes (Zero bytes on disk)
for 1 item
Where: iCloud
Copy as Pathname
Created: Septe
Modified: September 10, 2023 at 5:03 PM
Shared folder
Locked
More Info:
Name & Extension:
Comments:
Preview:
Sharing & Permissions:

For PCs, start by opening your File Explorer: Press the Windows key + E on your keyboard. Alternatively, you can click the "File Explorer" or "This PC" icon on your taskbar or Start menu. Navigate to the Folder: Use the File Explorer to navigate to the folder where the file is located. You can click on folders to open them and view their contents.

1. Find the File: Locate the file you are interested in within the folder.

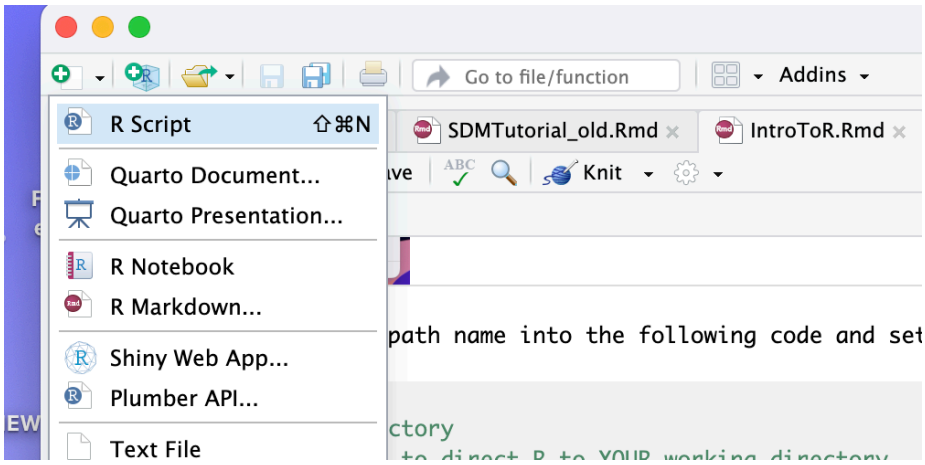
2. View the File Path: Once you've found the file, you can see its full file path in the address bar at the top of the File Explorer window. The file path will be displayed as a sequence of folder and file names separated by backslashes. You can click in the address bar and copy the file path to the clipboard by pressing Ctrl + C after selecting it.

3. Once you have copied your file path, paste that path name into the following code and set your working directory: setwd("/Users/sks379/Desktop/R is great/")

Now R studio is directed to upload and save work to this folder.

2.6 Annotating your code

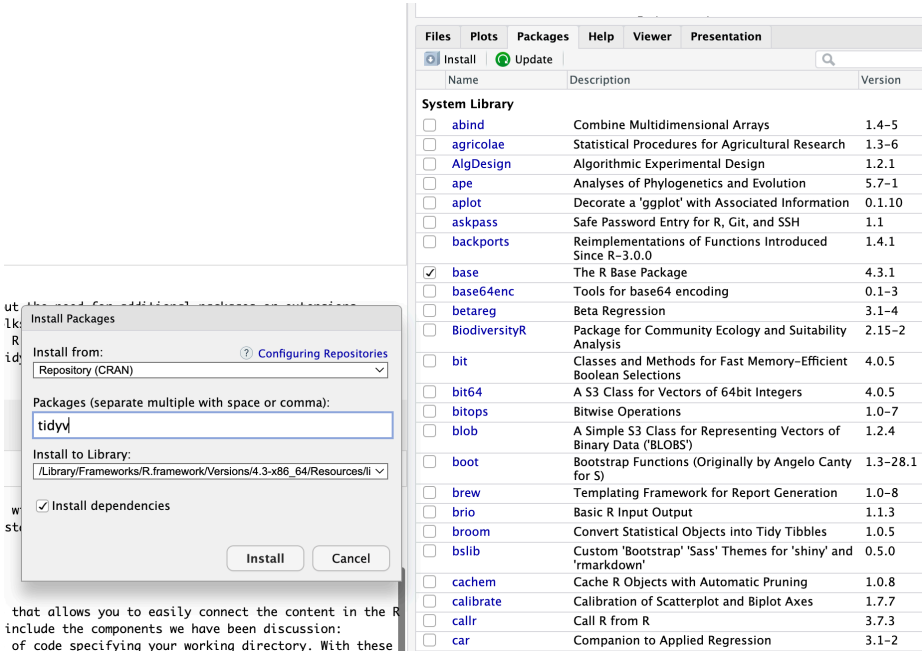
Notice anything about the code in the section you just ran? You can use the hast tag symbol to tell R not to run a section of code. Whenever I am generating code, I try to add lots of notes to myself, so that that future me knows what code I created and why. Annotating your code is just good practice for coding! Alternatively, you can create R markdown files (what this tutorial has been created in), but R markdown, while generating pretty PDFs and websites, adds an extra layer of complexity that you generally don't want or need while coding, so I typically recommend creating an R script and annotating your that file! One thing that you might want to include in your code description is the version of R that you are using (you may need to load older versions of R if your scripts stop working due to updates to the program). To check the version of R that you are using, paste this in the command line: R.version.string



2.7 Libraries

Base R, the fundamental, built-in set of functions, data structures, and libraries that come with the R programming language without the need for additional packages or extensions, including basic math functions, statistical analyses and visualization tools. However, one amazingly cool think about R is that folks are out there creating 'libraries', or a collection of R functions, data sets, and documentation bundled together into a single package, to do specialized analyses. For most tasks in R, you will need to install and load libraries. There are two methods to install libraries. Let's start by installing an important library (or package) for data manipulation, called 'tidyverse' (actually several packages - hence why the name references a universe). To do this, run this code: install.packages("tidyverse").

The weird thing about including install packages code is that you don't want to re-install packages every time that you use R (in fact, it caused my R markdown code to freak out, which is why I've included the install function in the text). You will need to load packages, but you update R and R studio, you won't need to install packages after you have done it once. You can either then install the packages and delete the install code OR you can install through the R studio interface by going to packages, selecting install and searching for and installing the packages that you are interested in. When prompted, be sure to install dependencies - this will make sure that you have any pieces of code that the library that you are installing needs to operate.



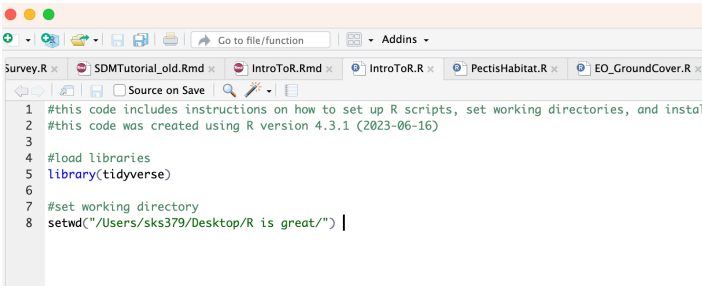
Excellent! You have installed a library! Now, we need to load it. AND you will need to load R packages everytime that you use R. Any functions associated with your package won't work, unless the package is loaded, so I suggest keeping the load library code in your R script. The code is simple (see below).

```
library(tidyverse)
```

Wonderful, you now have the essential knowledge base that you need to start working efficiently and effectively in R!

2.8 Organizing your code

A well-written R script will include the components we have been discussion: Annotated notes on what the script is doing and potentially even the version of R, loading commands for your libraries, and a line of code specifying your working directory. With these elements in place, you are ready to code your heart out!



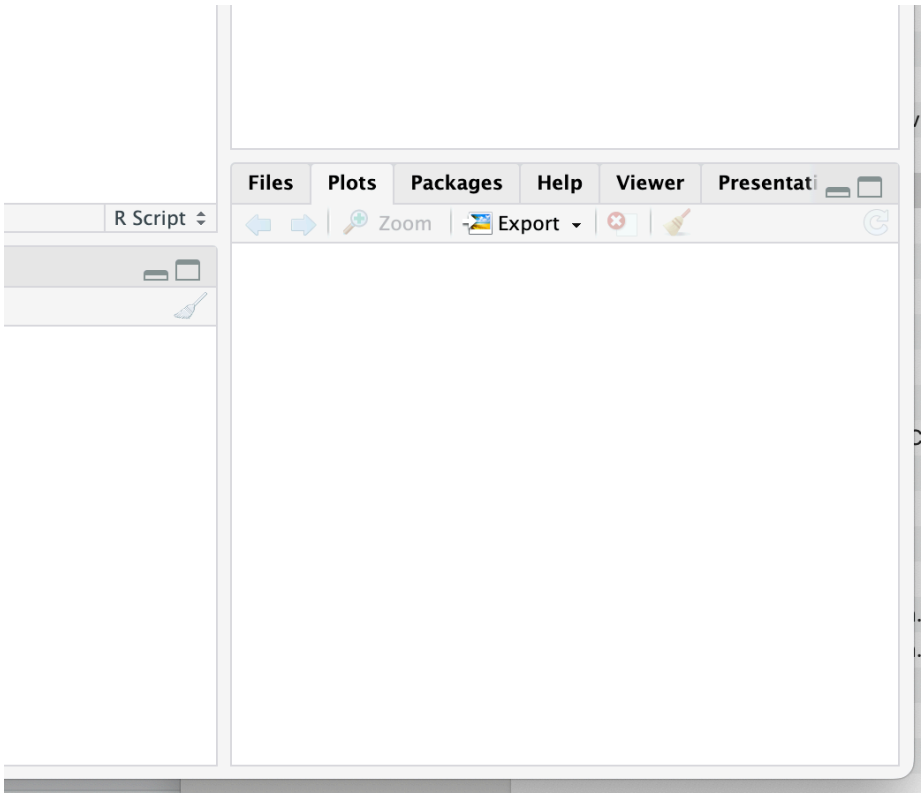
Here is a glimpse at what your R scripts should look like.
Importing and exporting .csv files

You may want to import data, and export a .csv file after analyses to include as a table in your results. The code to do this is simple:

```r
# Read in a file
read.csv("your_data.csv")

# Write a file
write.csv(your_object_to_export, "your_file_name.csv")
```

Finally, one easy way to export plots is to use the 'Plots' tab in your R studio window. Simply click on the 'Plots' tab and select export - you can choose different file types and sizes when you export.



Here is a nice example of a R script: Download the R file

And with that, welcome to the wonderful world of coding in R!

Chapter 3

Descriptive statistics

3.1 What can statistics tell us?

Welcome to our statistical exploration of the natural world! I want you to have an intuitive understanding of what we do when we conduct statistical analysis, understand how to select the appropriate statistical analysis and the assumptions of that analysis, and make a connection between running an analysis in a statistical software package and the statisticese of those unique individuals that we call statisticians. I think this will give you the confidence to tackle any analytical situation!

First off, we are **not** statisticians - whew! We are training to be ecologists. This means that we do not need to understand theoretical mathematical frameworks. We need to APPLY statistics appropriately. There will be limits to our mathematical understanding of statistics, and this is OKAY! Think of all of the knowledge about the natural world and conducting field work that we possess that statisticians don't!

Almost all statistical analysis boils down to answering 1 of 2 questions: *Do these groups differ?* Is there a relationship between these variables?

These seem like relatively simple questions to answer, perhaps just by looking at our data, so **Why do we need statistics?**

The short answer is: **error** and **sampling**! Whenever we collect data, we introduce **error**; our instruments are imprecise and do not capture an exact measure of whatever you are measuring (e.g., height, weight), and humans make mistakes during measurement collection. Secondly, we are **always** measuring a sub-sample of the true population (*true population* meaning all representatives of whatever you are trying to measure; this can be grass, marbles, or the tibia of humans). Not only is in intractable in most cases to measure all individuals of whatever you are interested in, even when it is possible to attempt to measure

all individuals (like in the case of rare plant work), statistics acknowledges that it is **still** unlikely that we are able to completely measure all individuals in your focal population, since individuals may be dormant or challenging to locate. If we could measure all individuals of our population of interest with perfect accurately, we could calculate population **parameters**, or quantities describing populations like averages and variation, rather than estimating these metrics, and just compare them. In this way, statistics is inherently practical, and asks: What can we say about whatever we are looking at, given our numerous flaws?

3.2 Sampling populations

After a few classes, we will explore sampling methodology in greater depth in order to design appropriate experiments that test a statistical **hypothesis**. Let's quickly talk about sampling now so that we have a shared understanding and vocabulary to build on - after all, statistics really centers around estimating characteristics of a true population from a sample. The **really, truly** amazing thing is that by properly applying statistics, we can learn practically anything about almost any population using samples!

In statistics, a **population** refers to the all units of the thing that you are interested (i.e., all suriname frogs, all grains of sand, all aspen leaves from a genotype found in southern Arizona). **Note**: Population in statistics differs from the term population in population ecology, where a population refers to a group of individuals in a particular area that interbreed.

A **sample** is a subset of the population that we measure to infer something about the population. Statistical analyses depend on a random sample or must account for non-randomness within the sample. Just imagine, for instance, that you were interested in whether coat color in cats differed between house cats and feral cats. To select the house cat sample, you randomly select house numbers, visit the house and record coat color, thus collecting a random sample. However, to survey feral cats, you go to several cat colonies at night and record the first cat that you see, which are always white or tan. The sample of feral cats introduces bias, and causes you to overestimate the number of light colored feral cats, and underestimate dark feral cats.

We can conduct statistical analysis until the cats come home (ha!), but if your sample is biased, our results will always be meaningless. In the cat example, it was pretty obvious that the researcher was introducing bias, BUT it is REALLY easy to introduce bias in ecological and social research on accident! Imagine that you looking at fire effects on vegetative communities in the Sonoran. In high severity burn areas, there are thickets of cat's claw (a pokey plant). Without proper field sampling protocols, it is very tempting to avoid establishing plots in the cat claw thickets, thus not capturing true differences in vegetation along burn severity gradients. We can go over field sampling methodology later, but let's talk about several types of appropriate sampling strategies. The key

is that we want our sample to be **representative** of the true population, so that estimates of values (i.e., means, variance) from the sample represent true population parameters.

Simple Random Sampling is when every sample has an equal likelihood of being selected.

We can quickly and easily generate such a sample in R, using the sample function.

```
sample(1:100, 10, replace=FALSE)
```

```
##  [1] 49 94 69 26 84 16 74  5 66 65
```

```
#1:10000 = numbers to chose among
#number of random numbers you wish to generate
#to replace or not (in other words do you wish for the same number to be selected multiple times)
```

Random Cluster Sampling randomly select groups (aka clusters) within a population. This sampling design is used commonly in ecology, when we select random locations for plots, then measure all individuals within those plots. If for instance, we are interested in Ponderosa Pine growth rates on the Coconino National Forest, we would randomly assign points across Pondo habitat on the Coconino. At each point, we would set up a plot in which we measure Ponderosa Pines within an 11.71m radius plot. **Why wouldn't we just go out to a point and measure 1 tree to create a totally random sample?** The plots are randomly assigned (yay!), but the trees within the plots are not **independent**. In other words, we might expect measures of trees within plot A to be more similar to each other than they are to trees within plot B, due to differences in microsite characteristics, genetic similarity among co-occurring trees, or site history (logging, fire). Luckily, we can account for this non-independence, as long as the plots are random!

Stratified Sampling draws samples using proportionality based on homogeneous groupings known as strata. This type of sampling is frequently used in ecology to account for landscape differences in key factors. For instance, say you asked to classify vegetation types for a Nature preserve in southern Arizona. The reserve has a large riparian area (25% of the property) with completely different vegetation from the upland area (75%). Random sampling might, by chance, under or over represent one of these two areas. To create a stratified sampling design, you would ensure proportional representation of both areas by randomly placing 25% of the sample points within the riparian area, and 75% of the sample points within the upland area.

Sample size More is better. However, practically we are often limited by time and money!

Statistical analysis is only one part of presenting your research results. Generally, a results section in a manuscript includes: statistical results, data description (e.g., describing means, ranges, maxima, minima of groups of interest), and data visualization (i.e., creating beautiful figures). For each analysis that we cover, we will talk about how to present statistical results, describe data, and create appropriate supporting figures.

3.3 Data types

Before we start learning to present research results (analysis, description, visualization), let's talk about data! Data comes in several varieties, and the variety dictates which statistical analysis we choose!

Categorical variables are non-numeric variables. **Examples**: Pet type (dog, cat, fish, bird), Size (small, medium, large), Car type (sedan, SUV), Present/Absent

Numerical variables are variables that are numbers, and occur in two forms: *Discrete = Counts of things (no decimal points/fractions) Data are discrete when it does not make sense to have a partial number of the variable. For instance, if counting the number of insects in a pond, it does not make sense to count a half a species. **Examples**: Number of people in a building, number of trees in a plot, number of bugs in a pond

*Continuous = Numerical data that can occur at any value. These are variables that can occur in any quantity. If you can have a fraction of this variable, it is continuous. **Examples** = Height, Weight, Length

Ordinal variables (sometimes referred to as ranked) can be categorical or numerical, but the order matters. **Examples** = Grades (A, B, C, D, E), Likert scale variables (Strongly disagree, Agree, Strongly Agree), Class rank (1, 2, 3, 4, 5)

3.4 Describing data

First, let's take a spin with data description. We are starting here to introduce a few concepts that will be important to understand, as we launch into statistical analysis. We will start by describing continuous data.

Let's use a simplified version of a dataset that I'm working with right now to look at the performance of several species of pollinator-friendly native species in agricultural gardens. Eventually, we'd like to develop seed to provide to restorationists for restoration of arid and semiarid grasslands. To do this, we need to understand how reliable these species are at establishing, producing seed, and attracting pollinators. Initially, we are conducting experiments with multiple populations of each species to determine how consistently plants grow,

reproduce, and perform. Here, We will take a look at the initial heights of 1 population of one species, *Asclepias subverticulata*.

Most of the time when writing up results, you present a mean (sum of numbers divided by the number of observations), and an estimate of variation (a measure of how different the observations are). Here, we calculated three estimates variation, variance, standard deviation, and standard error.

```
#create vector of heights (cm) of one population of A. subverticulata
sedonapopulation <- c(3, 3, 3, 3, 7, 8, 9)
#take the mean
mean(sedonapopulation)
```

```
## [1] 5.142857
```

```
#calculate variance
var(sedonapopulation)
```

```
## [1] 7.47619
```

```
#calculate standard deviation
sd(sedonapopulation)
```

```
## [1] 2.734262
```

```
#calculate standard error
#base r doesn't have this function
#so we have to write our own
std_error <- function(x) sd(x)/sqrt(length(x))
std_error(sedonapopulation)
```

```
## [1] 1.033454
```

Since you will occasionally need to include equations in your write-ups, let's get use to mathematical syntax, with these simple examples.

The formula for the sample mean is: $\mu = \frac{\Sigma x_i}{n}$; where μ indicates the sample mean (sample = group of numbers we are looking at); Σ means to add what ever follows; x_i is the value of one observation; (subscript i is often used to indicate that the action should be repeated for all values); n is the number of observations

Why didn't we just use $\bar{x}$ to indicate the mean? Because statisticians typically use $\bar{x}$ to indicate the true mean of the population, and μ to indicate the sample mean!

Just to show you, what the mean() function is doing, let's run:

```
sum = 3+3+3+3+7+8+9 #add all the numbers in the sample
n = length(sedonapopulation) #or you can just calculate the number of height measureme
mean = sum/n; mean #divide sum by number
```

```
## [1] 5.142857
```

This formula is simple, but sometimes with more complex formulas, I will solve the equations by hand, to make sure that I understand what is happening!

The formula for variance is: $S^2 = \frac{\Sigma(x_i - \mu)^2}{n-1}$ where S^2 is the sample variance; μ is the sample mean (remember from above); x_i is the value of one observation; n is the number of observations

In other words:

```
#We determine how much each observation varies from the mean.
diffobs1 = mean - 3
diffobs2 = mean - 3
diffobs3 = mean - 3
diffobs4 = mean - 3
diffobs5 = mean - 7
diffobs6 = mean - 8
diffobs7 = mean - 9

#Then we square each of these.
diffobj1_sq = diffobs1^2
diffobj2_sq = diffobs2^2
diffobj3_sq = diffobs3^2
diffobj4_sq = diffobs4^2
diffobj5_sq = diffobs5^2
diffobj6_sq = diffobs6^2
diffobj7_sq = diffobs7^2
```

Why do we square the differences rather than just adding them up? Because differences will be positive and negative. If we added them without squaring, sample differences would negate each other. We want an estimate of the absolute differences of samples from the mean.

```
#Then we add the differences up.
sumofsquares = sum(diffobj1_sq, diffobj2_sq, diffobj3_sq, diffobj4_sq, diffobj5_sq, di
#Divide the sum of squares by n - 1.
variance = sumofsquares/(n-1); variance
```

```
## [1] 7.47619
```

Why n - 1 instead of n? One reason is that, theoretically, because we are taking the mean of a sample, rather than all individuals, we underestimate the variance, so taking n-1 corrects that bias. Consider it a penalty for measuring a sample, not the entire population! Another practical reason is that dividing by n-1 makes the variance of a single sample undefined (unsolvable) rather than zero (solvable)

For standard deviation, we just take the square root of the variance, to remove the effect of squaring the differences when calculating the variance, and thus contextualizing our estimate of variation with regard to the mean. For example, the variance for the Sedona population is 7.48, larger than the sample mean of 5.12; while the standard deviation is 2.73, indicating that you would expect most observations to be 5.12 +/- 2.73 (we'll get to quantiles in a minute).

The formula for standard deviation is: $\sigma = \sqrt{\frac{\Sigma(x_i - \mu)^2}{n-1}}$ where σ is the sample variance; μ is the sample mean; x_i is the value of one observation; n is the number of observations.

Finally, standard error and confidence intervals (we'll get to confidence intervals later) are the most common metrics of variance presented in journals.

The formula for standard error is: $SE = \frac{\sigma}{\sqrt{n}}$ where SE is standard error of the sample; σ is the standard deviation; and n is the number of samples.

Why do we divide the standard deviation by the square root of the sample size to get standard error? While standard deviation measures the variation of the sample, standard error is meant to estimate the variation of the entire population of samples, if we could measure all individuals accurately. By dividing by the $\sqrt{n}$, the larger the sample size, the lower the error, because you have a more complete estimate of the true mean. In other words, standard deviation is just a measure of the variation of our sample, while standard error also incorporates information about our sampling process (how many individuals we have sampled). *Want to delve deep into standard error and deviation (me neither - ha)?: Google central limit theorem + standard error / standard deviation.*

Means and variance measures are the most common way to describe quantitative data. However, several other metrics are useful for understanding the nature of your data and making decisions about analyses. A comprehensive understanding of your dataset includes describing these four features: *Location (Mean, Median)* Spread (Variability) *Shape (Normal, skewed)* Outliers

We've talked about means. The median is just the central number in the dataset, and helps you identify skewness.

```
#an example of an unskewed population
sedona_unskewed <- c(1, 2, 3, 4, 5, 6, 7)
mean(sedona_unskewed)
```

```
## [1] 4
```

```
median(sedona_unskewed)
```

```
## [1] 4
```

```
#previous sedona population; skewed
sedonapopulation <- c(3, 3, 3, 3, 7, 8, 9)
mean(sedonapopulation)
```

```
## [1] 5.142857
```

```
median(sedonapopulation)
```

```
## [1] 3
```

In an unskewed population, the mean will equal the median. Skew may not seem important, but it has statistical ramifications, AND it tells us something meaningful about the data. For instance, what if I said that mean price of a home in Flagstaff is 350K, but the median price of a home is 300K? We would know the that average house prices are driven up by a smaller number of expensive homes.

We can quantify skew by comparing means and medians (mean > median = right-skewed; median > mean = left-skewed), but it is helpful to visualize the shape of data with a **histogram**. A **histogram** is a graph of the frequency of different measurements.

Let's add a few more observations to our Sedona populations (skewed and unskewed) and check out the look of the data!

```
sedona_unskewed <- c(7, 2, 2, 3, 3, 3, 3, 6, 6, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 0.5)
mean(sedona_unskewed)
```

```
## [1] 3.975
```

```
median(sedona_unskewed)
```

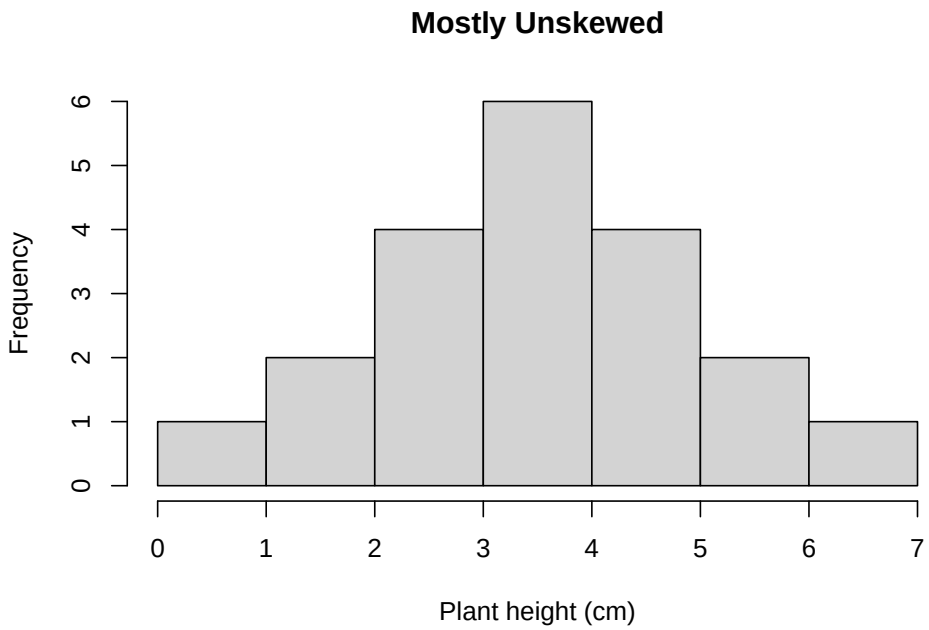
```
## [1] 4
```

```
#I'm renaming sedonapopulation, sedona_skewed for this example
sedona_skewed <- c(3, 3, 3, 3, 7, 3, 4, 5, 6, 3, 3, 3, 4, 4, 6, 7, 8, 9, 3, 4, 5, 2)
mean(sedona_skewed)
```

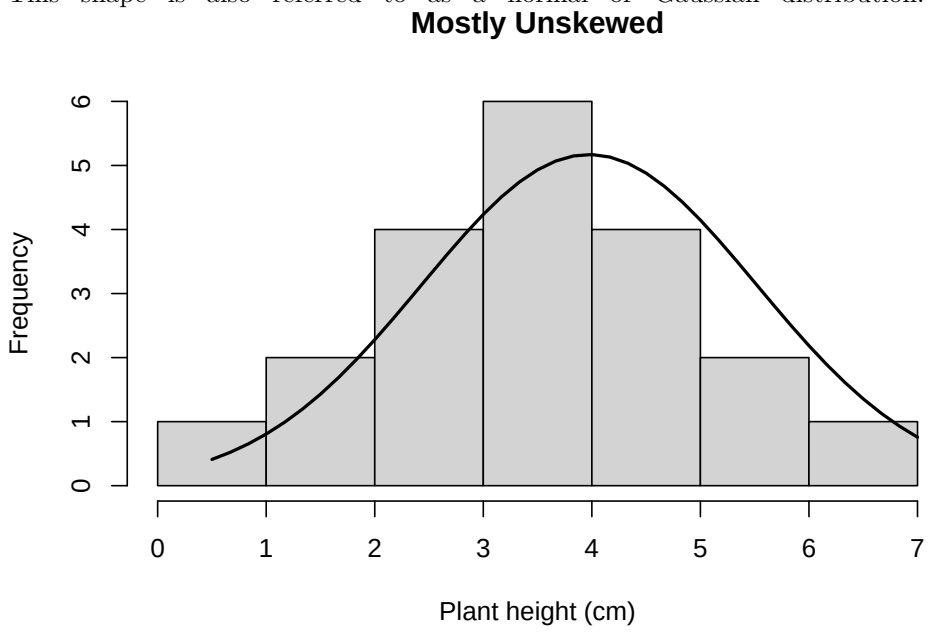
```
## [1] 4.454545
```

```
median(sedona_skewed)
```

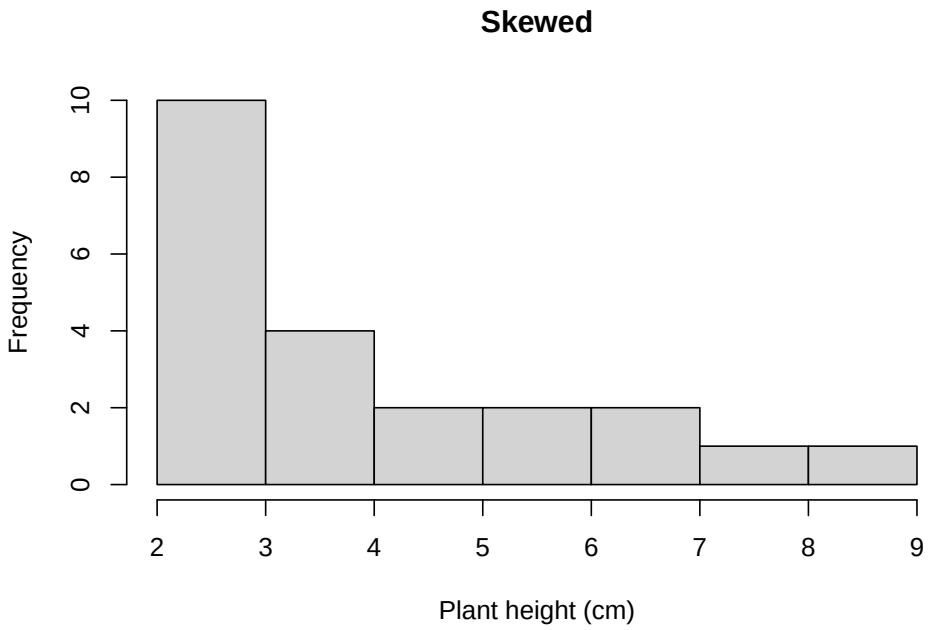
```
## [1] 4
```



In this relatively unskewed example, the tails are approximately even. This shape is also referred to as a normal or Gaussian distribution.



Here, we superimposed the bellshaped Normal or Gaussian distribution.

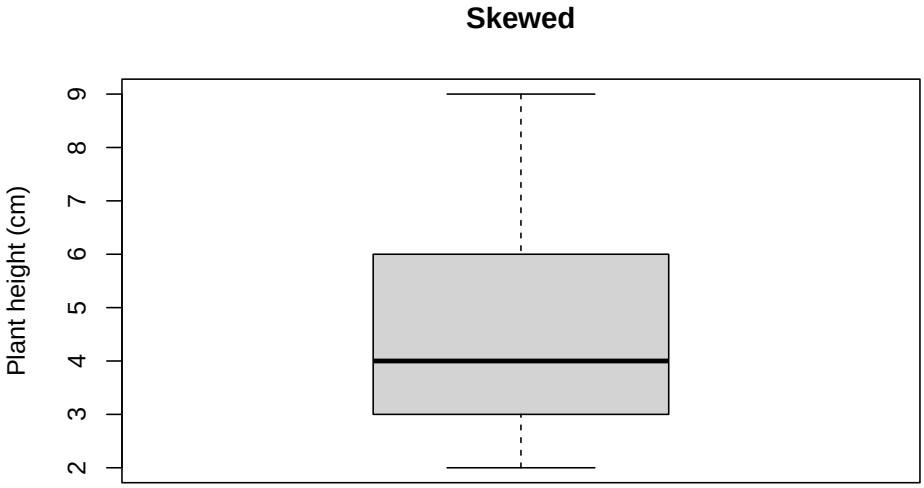


In this example of skewed data, the tail tapers to the right, indicated that the data is skewed to the right.

In order to explain outliers, we need to look at quantiles! Quantiles are proportions of your data, in other words a way to break your data into chunks to understand spread. You can break your data into as many quantiles as you would like, but it is most common to break your data into 4 parts, also called quartiles. (If you break data into 5 parts, the components are called quintiles, 10 parts = deciles, 100 parts = percentiles).

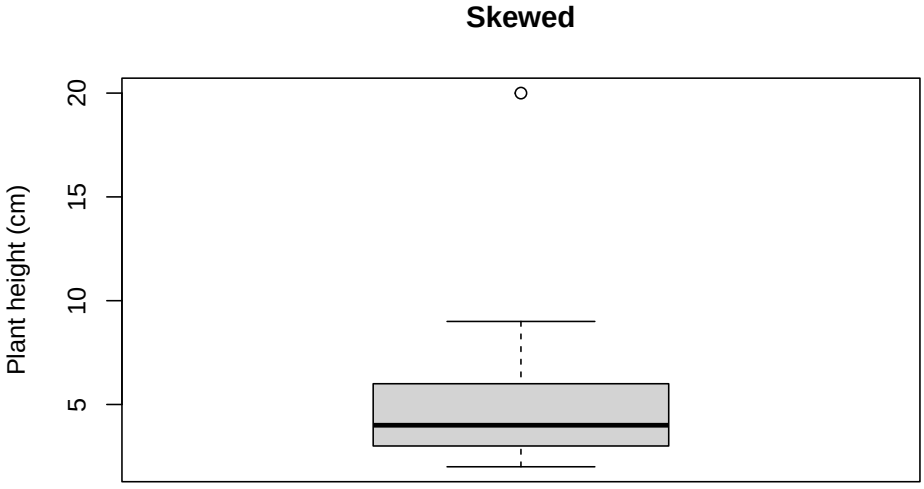
When you break data into quartiles, roughly 25 percent of the data occurs within each data chunk. The first chunk of the dataset contains 25% of the data (25th percentile; 25% of the data fall at or below this cut-off) is called the first quartile, the 50th percentile is called the sample median or the second quartile, the 75th percentile is called the third quartile.

Box and whisker plots are commonly used to quickly examine quartiles. Let's check out our plant height data again, using a box and whisker plot.



In the plot shown here, the box encapsulates the Interquartile Range (IQR); the center of the data ranging from the 25th percentile to the 75th. The black line in the middle of the box is the median (also called the 50th percentile, because it bisects the dataset; half of the data occur above the median and half below). The lines emerging from the box (whiskers) indicate the extent of the first and third quartiles, and usually corresponding with the minimum and maximum values of the dataset, unless there are **outliers**. An outlier is a datapoint that occurs outside of the 1st or 3rd quantile. Let's add one to our Sedona dataset, and see how it is represented on the box and whisker plot.

```
#Let's add a plant height of 20.
sedona_skewed <- c(3, 3, 3, 3, 7, 3, 4, 5, 6, 3, 3, 3, 4, 4, 6, 7, 8, 9, 3, 4, 5, 2, 20)
boxplot(sedona_skewed, main="Skewed", ylab="Plant height (cm)")
```



The outlier appears as a dot on the box and whisker plot, and is the maximum

value of the dataset.

One other thing to note: Standard deviation also breaks data into meaningful segments, but is only used when data conform to a normal distribution; the mean +/- 1 SD accounts for 68% of the data, +/-2 SDs contains 95% of data, and +/- 3SD includes 99% of data. That said, I've never presented standard deviation in a manuscript; it is much more common to include standard error or confidence intervals (discussed later).

We've played around a lot with data, but what do you actually need to take away from this? *Data types (Categorical, Numerical discrete, Numerical continuous, Ordinal)* ***Why?*** *We will select analyses based on data type.* The two basic questions that most statistical analyses answer. **Why?** This will help you define what statistics can and can't do and bound our learning space! *Ways to describe numerical continuous data (Location, Spread, Shape, Outliers).* ***Why?*** *You will describe your results using these concepts in write-up AND these concepts will be important for certain analyses.* Know how to calculate mean, median, and standard error. **Why?** These are typical ways to describe data in results sections. *Start to familiarize yourself with mathematical annotation.* ***Why?*** *You may need to include equations in your methods section.* Start to familiarize yourself with R code. **Why?** Most researchers now use R to analyze, describe, and visualize their data. *Be able to interpret a histogram and box-whisker plot. **Why?** These are commonly used ways to visualize data.

Now, let's play around a little more with R! For your reference, here is a guide that shows the basics formulas for calculations in R: Supplementary Material

Let's also try describing the petal lengths of 3 different plant species: Milkweed, Bluestar and Pectis using this dataset: Download the example dataset The code can be used to import the data, but in case you wanted to access the data yourself - provided above!

Build on this code, to generate your description of the petal lengths of your focal species: R Script for Chapter 1

Please put your results in a document and submit to your TA in canvas!