Introduction to R

BUGS Workshop - Fall 2023

Dominique Maucieri

11 September, 2023

Contents

Welcome to this Introduction to R workshop.

R etiquette and setting up your code

Set up

Lets start by making a new script for this tutorial. We will open a new script file and save it as BUGS_workshop.R

Now to set up this script, the first thing you always want to do in every new script, is put a description at the top of your script explaining what this script contains. Anything on a line following a # will be a comment, this code will not run, it will just be a note to anyone reading the code. For this workshop we can put a description kinda like this:

```
# This code is from the UVic BUGS Introduction to R Workshop
# Workshop led by Dominique Maucieri (dominiquemaucieri@gmail.com)
# 28-10-2023
```

The next thing you will put at the top of your code is **ALL** your packages. Packages are extensions to R what will contain things like code and data that we can use in our coding. They will often contain a bunch of different functions which will allow us to easily perform different tasks without coding them by hand. In the BEFORE_THE_WORKSHOP.pdf file, you would have been instructed to download a bunch of packages. We will be using these packages today to help us quickly perform task. It is proper etiquette to load all these packages at the start of your code so that someone else wanting to run your code will know what packages to download before starting. As you get better at coding you will know which packages you will want to use before you start, but as you learn, just keep scrolling to the top of your script and add packages as you need them.

For our workshop we will be using the dplyr, tidyr, ggplot2 and readxl packages, so lets load these and add them to the start of our script:

```
# Packages ----
library(dplyr)
library(ggplot2)
library(readxl)
library(tidyr)
```

Creating an outline

If you notice after the comment were I specified the following code was packages, there are 4 "-". This will create a new section in your code. If you open the outline, which is located at the top right of your script, you can see the outline of your code. This is super helpful when you start getting long code scripts, because you can easily jump around your code.

Importing data

Setting the working directory

In order to import data, we have to know where the data is located on your computer. When your import data, you will use code to tell R the location of the data on your computer to open and load into your working environment so you can use it in R. First you need to know the location on your computer where R is currently "looking", known as your working directory. Good protocol is to have your data in the same folder as where you saved your script file, or in a data folder within the folder that your script file is saved in.

To figure out where your working directory is we can use this code:

```
# Importing data ----
getwd() #where is my current working directory
```

```
## [1] "/Users/dom/Library/CloudStorage/Dropbox/Documents/GitHub/UVic_BUGS_RIntro"
```

For me, this folder "UVIC_BUGS_RIntro" is the folder where I have put the script we are using for today's workshop and it is also where I have the data we are going to import for today's workshop. For you however, this might not be the right spot. So if we need to change the working directory, there are a few ways:

1. Code. This is the toughest to work your head around but it works. You would use the setwd() and the file path to your script folder (this should also be where your data is, if not move your data into the same folder). You always put the filepath in quotations, and for my folder it would look like this:

```
setwd("~/Library/CloudStorage/Dropbox/Documents/GitHub/UVic_BUGS_RIntro")
```

Now often the issue with this method is your probably thinking well how do I get this file path? If you don't have a good understanding of file paths and how your computer works, this can seem very difficult, but there are a few short cuts to make this easier.

2. Using the File viewer pane. In the bottom right of your screen, there should be the files viewer. We can use this to move around on your computer and set your new working directory. Open files until you find the folder with your script and data. Then click the gear icon at the top of the file pane, and select "Set As Working Directory". This will run the line of code needed to set this folder as your working directory. Now important to remember to take the code outputted into your console, and paste it into your script so every time your come back to this code, you can set your working directory again.

3. Using the session options. At the top of your RStudio, go to Session > Set Working Directory > Choose Directory… Then you can move around on your computer until you find the folder with your script and data. This will run the line of code needed to set this folder as your working directory. Now important to remember to take the code outputted into your console, and paste it into your script so every time your come back to this code, you can set your working directory again.

TASK

Set your working directory to the folder where you have your script and data located.

Importing csv and excel data

There are many types of files you may want to import into R to use, and for this tutorial we will go through two formats of data, comma separated values (csv) and excel files. The biggest thing to figure out is where your data is in relation to where you working directory has been sent. The easiest is for the working directory to be set to the same location as your data.

.csv To import data that has been saved as a .csv file, we will use the read.csv() function. If this data is located in the same folder as your working directory has been set to (which it should be after the task above) we will import the file as follows:

```r
shark_data <- read.csv("LPD_Sharks_CPUE.csv")
```

But perhaps this data is in a folder called data, which is located in your working directory. This is how my data is always stored, so in order to import it, I have to set the file path as within a folder like this:

```r
shark_data <- read.csv("data/LPD_Sharks_CPUE.csv")
```

Troubleshooting:

- Always make sure the files are spelt the same way
- Make sure you include the extention (.csv)
- Make sure it is in quotes

Notice that we stored this data into an object called "shark_data". This should now show up in your environment window as data. This means that anywhere you use the word shark_data in your code after now, it will reference this data.

This data comes from the Living Planet Database (https://www.livingplanetindex.org/) and is a cleaned up version where I have subsetted out some shark abundance data for us to look at.

.excel Importing excel files are very similar, they will just use a different function, read.excel() and the .xls or .xlsx file extension:

```r
shark_data <- read_excel("data/LPD_Sharks_CPUE.xlsx")
```

Yay! Now we have data to work with in R. But what to do next? Well we need to look at this data and explore it a bit. There are many ways to do this. The first will be to just open the data frame and scroll around it. To open the data frame we can do a few things:

1. Over in the environment tab, we can click on the data frame we want to examine.
2. In the console or our script we can use the View() function to view our data, with the name of the data frame object we want to open and view within the brackets of the View() function.

TASK

Try to open the data frame to view it using a few different methods.

Types of Objects and Elements

There are a few types of objects you will come across in R. These will be data frames, matrices, vectors and lists.

- A vector is like a single string of variables, like a single row of data, one dimension.
- Data frames are what we have already encountered, it has rows and columns, two dimensions, and it can be a mix of numbers and characters
- Matrices like data frames with rows and columns but they only contain one type of data, so all numbers or all characters
- Lists are like a bunch of data frames all stacked on top of each other, three dimensions, but we won't be working with lists for this.

As eluded to above, there are different types of data as well. We can have numeric, integer, logical, character and factor type data.

- Numeric is numbers, they contain decimals and are continuous. Examples are length measures or heights
- Integers are whole numbers, no decimals but still numbers. Example would be the year for your data.
- Logical data is TRUE / FALSE data
- Characters are text, known as strings in R, which can be words or a jumble of various letters and symbols, but this data is not being treated as a number and has no order to it. Characters are shown with quotes around them so R knows to treat it as a character. Examples could be site names or species names. "Site1"
- Factors are also text, like characters but they have a set order to them. The default order will be alphabetical but you can change this in R so it makes more sense. Example would be treatments, like before and after, you would want before to be order before after or it won't make sense for something like a plot.

To create a vector, we can use the c() function to combine a bunch of elements together and assign them to an object with the arrow operator <-

```r
fruit <- c("apple", "banana", "pear")

#then calling the object name will output the vector values in your console
fruit
```

```
## [1] "apple"  "banana" "pear"
```

```
leaf_lengths <- c(4.5, 7.2, 4, 5.4)

leaf_lengths
```

```
## [1] 4.5 7.2 4.0 5.4
```

TASK

Try making your own object that contains both characters and numeric data. Then look at the structure of this object, what type of data does it contain?

It is easy to change the type of data to something else. Sometimes R will read in your numbers as characters or you might want to change your characters to a factor. So to this, first you need to know how R is currently classifying your data. To see this we can use class() or str(). I like str() because it gives a bit more information, but either works.

```
class(leaf_lengths)
```

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

```
str(fruit)
```

```
##  chr [1:3] "apple" "banana" "pear"
```

so leaf_length is a numeric vector and fruit is a character vector. To change to a different type of data you use as.******() with the asterics being the new data type. So as.numeric() or as.factor() etc. Lets turn fruit into a factor, and assign it to a new operator called fruit_fact

```
fruit_fact <- as.factor(fruit)

str(fruit_fact)
```

```
##  Factor w/ 3 levels "apple","banana",..: 1 2 3
```

Now when we want to change how a column in a data frame is being treated by R, we have to select that specific column. We can select a column within a data frame using the $

```
shark_data$Common_name
```

```
##   [1] "Porbeagle shark"           "Porbeagle shark"
##   [3] "Porbeagle shark"           "Porbeagle shark"
##   [5] "Porbeagle shark"           "Porbeagle shark"
##   [7] "Porbeagle shark"           "Porbeagle shark"
##   [9] "Porbeagle shark"           "Porbeagle shark"
##  [11] "Porbeagle shark"           "Porbeagle shark"
##  [13] "Thresher shark"            "Thresher shark"
##  [15] "Thresher shark"            "Thresher shark"
##  [17] "Thresher shark"            "Thresher shark"
##  [19] "Thresher shark"            "Thresher shark"
##  [21] "Thresher shark"            "Thresher shark"
##  [23] "Thresher shark"            "Thresher shark"
##  [25] "Thresher shark"            "Thresher shark"
##  [27] "Thresher shark"            "Thresher shark"
##  [29] "Thresher shark"            "Thresher shark"
##  [31] "Thresher shark"            "Thresher shark"
##  [33] "Thresher shark"            "Thresher shark"
##  [35] "Blue shark"                "Blue shark"
##  [37] "Blue shark"                "Blue shark"
##  [39] "Blue shark"                "Blue shark"
##  [41] "Blue shark"                "Blue shark"
##  [43] "Blue shark"                "Blue shark"
##  [45] "Blue shark"                "Blue shark"
##  [47] "Blue shark"                "Blue shark"
##  [49] "Blue shark"                "Blue shark"
##  [51] "Blue shark"                "Blue shark"
##  [53] "Blue shark"                "Blue shark"
##  [55] "Blue shark"                "Blue shark"
##  [57] "Blue shark"                "Shortfin mako"
##  [59] "Shortfin mako"             "Shortfin mako"
##  [61] "Shortfin mako"             "Shortfin mako"
##  [63] "Shortfin mako"             "Shortfin mako"
##  [65] "Shortfin mako"             "Shortfin mako"
##  [67] "Shortfin mako"             "Shortfin mako"
##  [69] "Shortfin mako"             "Shortfin mako"
##  [71] "Shortfin mako"             "Sand tiger shark"
##  [73] "Sand tiger shark"          "Sand tiger shark"
##  [75] "Sand tiger shark"          "Sand tiger shark"
##  [77] "Sand tiger shark"          "Sand tiger shark"
##  [79] "Sand tiger shark"          "Sand tiger shark"
##  [81] "Sand tiger shark"          "Sand tiger shark"
##  [83] "Sand tiger shark"          "Sand tiger shark"
##  [85] "Sand tiger shark"          "Sand tiger shark"
##  [87] "Sand tiger shark"          "Sand tiger shark"
##  [89] "Sand tiger shark"          "Sand tiger shark"
##  [91] "Sand tiger shark"          "Sand tiger shark"
##  [93] "Sand tiger shark"          "Sand tiger shark"
##  [95] "Sand tiger shark"          "Sand tiger shark"
##  [97] "Sand tiger shark"          "Sand tiger shark"
##  [99] "Sand tiger shark"          "Sand tiger shark"
## [101] "Sand tiger shark"          "Sand tiger shark"
## [103] "Sand tiger shark"          "Sand tiger shark"
## [105] "Great white shark"         "Great white shark"
## [107] "Great white shark"         "Great white shark"
## [109] "Great white shark"         "Great white shark"
## [111] "Great white shark"         "Great white shark"
## [113] "Great white shark"         "Great white shark"
## [115] "Great white shark"         "Great white shark"
## [117] "Great white shark"         "Great white shark"
## [119] "Great white shark"         "Great white shark"
## [121] "Great white shark"         "Great white shark"
## [123] "Great white shark"         "Great white shark"
## [125] "Great white shark"         "Great white shark"
## [127] "Great white shark"         "Great white shark"
## [129] "Great white shark"         "Great white shark"
## [131] "Great white shark"         "Great white shark"
## [133] "Great white shark"         "Great white shark"
## [135] "Great white shark"         "Great white shark"
## [137] "Great white shark"         "Blue shark"
## [139] "Blue shark"                "Blue shark"
## [141] "Blue shark"                "Blue shark"
## [143] "Blue shark"                "Blue shark"
## [145] "Blue shark"                "Blue shark"
## [147] "Blue shark"                "Blue shark"
## [149] "Atlantic sharpnose shark"  "Atlantic sharpnose shark"
## [151] "Atlantic sharpnose shark"  "Blacktip shark"
## [153] "Blacktip shark"            "Blacktip shark"
## [155] "Spinner shark"             "Spinner shark"
## [157] "Spinner shark"             "Blacknose shark"
## [159] "Blacknose shark"           "Blacknose shark"
## [161] "Bonnethead shark"          "Bonnethead shark"
## [163] "Bonnethead shark"          "Sandbar shark"
## [165] "Sandbar shark"             "Sandbar shark"
## [167] "Scalloped hammerhead"      "Scalloped hammerhead"
## [169] "Scalloped hammerhead"      "Finetooth shark"
## [171] "Finetooth shark"           "Finetooth shark"
## [173] "Shortfin mako"             "Shortfin mako"
## [175] "Shortfin mako"             "Shortfin mako"
## [177] "Tope shark"                "Tope shark"
## [179] "Broadnose sevengill shark" "Broadnose sevengill shark"
## [181] "Smooth hammerhead"         "Smooth hammerhead"
## [183] "Smooth hammerhead"         "Smooth hammerhead"
## [185] "Smooth hammerhead"         "Smooth hammerhead"
## [187] "Smooth hammerhead"         "Smooth hammerhead"
## [189] "Smooth hammerhead"         "Lemon shark"
## [191] "Lemon shark"               "Lemon shark"
## [193] "Grey reef shark"           "Grey reef shark"
## [195] "Grey reef shark"           "Grey reef shark"
## [197] "Whitetip reef shark"       "Whitetip reef shark"
## [199] "Whitetip reef shark"       "Whitetip reef shark"
## [201] "Tiger shark"               "Tiger shark"
## [203] "Blacktip reef shark"       "Blacktip reef shark"
## [205] "Scalloped hammerhead"      "Scalloped hammerhead"
## [207] "Scalloped hammerhead"      "Scalloped hammerhead"
## [209] "Tawny nurse shark"         "Tawny nurse shark"
## [211] "Silky shark"               "Silky shark"
## [213] "Whale shark"               "Whale shark"
## [215] "Silvertip shark"           "Silvertip shark"
## [217] "Round ribbontail ray"      "Round ribbontail ray"
## [219] "Ribbontail stingray"       "Ribbontail stingray"
```

This will select only one column within the data frame. So we could examine they type of data for the whole data frame or just a single column. However, class() of a data frame will tell you that it is a data frame, not the type of data in each column like str() will.

```r
class(shark_data)
```

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

```r
str(shark_data)
```

```
## tibble [220 x 15] (S3: tbl_df/tbl/data.frame)
##  $ ID         : num [1:220] 1934 1934 1934 1934 1934 ...
##  $ Binomial   : chr [1:220] "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" ...
##  $ Class      : chr [1:220] "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" ...
##  $ Order      : chr [1:220] "Lamniformes" "Lamniformes" "Lamniformes" "Lamniformes" ...
##  $ Family     : chr [1:220] "Lamnidae" "Lamnidae" "Lamnidae" "Lamnidae" ...
##  $ Genus      : chr [1:220] "Lamna" "Lamna" "Lamna" "Lamna" ...
##  $ Species    : chr [1:220] "nasus" "nasus" "nasus" "nasus" ...
##  $ Common_name: chr [1:220] "Porbeagle shark" "Porbeagle shark" "Porbeagle shark" "Porbeagle shark"
##  $ Location   : chr [1:220] "NAFO Subareas 3-6, Canada" "NAFO Subareas 3-6, Canada" "NAFO Subareas 3-
##  $ Country    : chr [1:220] "Canada" "Canada" "Canada" "Canada" ...
##  $ Region     : chr [1:220] "North America" "North America" "North America" "North America" ...
##  $ Latitude   : num [1:220] 43.3 43.3 43.3 43.3 43.3 ...
##  $ Longitude  : num [1:220] -54.8 -54.8 -54.8 -54.8 -54.8 ...
##  $ Year       : num [1:220] 1989 1990 1991 1992 1993 ...
##  $ Abundance  : num [1:220] 0.0035 0.0041 0.0052 0.0068 0.0034 0.0035 0.0036 0.0025 0.0005 0.0015 ..
```

```r
class(shark_data$Common_name)
```

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

```r
str(shark_data$Common_name)
```

```
##  chr [1:220] "Porbeagle shark" "Porbeagle shark" "Porbeagle shark" ...
```

Now did you see what Year was being treated as? It was treated as numeric, but maybe we want it as an integer. So to change it to an integer we have to use the as.integer() on only that column and assign it to a single column.

```r
shark_data$Year <- as.integer(shark_data$Year)
str(shark_data$Year)
```

```
##  int [1:220] 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 ...
```

Math in R

R can be used as a fancy calculator, and common math expressions will work within your code.

- a * will multiply
- a / will divide
- a - will subtract

- a + will add
- exp() is an exponent
- pi() is the complete unrounded pi
- brackets work using BEDMAS

TASK

complete some simple math expressions like 4 + 7

Summary Functions

With new data, it can be helpful to generate some summary statistics to explore your data and understand what the data looks like. Some helpful summary statistic functions can be:

nrow() and ncol()

These two functions will tell you the number of rows and number of columns in a data frame respectively.

```
ncol(shark_data)
```

```
## [1] 15
```

TASK

Determine the number of rows in the shark data frame.

---

unique()

This is one of my favorites and it will help to identify all the unique values within an object. However, it might at first seem unhelpful like this:

```
unique(shark_data)
```

```
## # A tibble: 220 x 15
##       ID Binomial  Class Order Family Genus Species Common_name Location Country
##    <dbl> <chr>     <chr> <chr> <chr>  <chr> <chr>   <chr>       <chr>    <chr>
##  1  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  2  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  3  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  4  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  5  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  6  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  7  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  8  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
##  9  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
## 10  1934 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada
## # i 210 more rows
## # i 5 more variables: Region <chr>, Latitude <dbl>, Longitude <dbl>,
## #   Year <int>, Abundance <dbl>
```

This will just output the whole data frame, not helpful. But if we select one column using a $ it will give us more information.

```
unique(shark_data$Common_name)
```

```
##  [1] "Porbeagle shark"            "Thresher shark"
##  [3] "Blue shark"                 "Shortfin mako"
##  [5] "Sand tiger shark"           "Great white shark"
##  [7] "Atlantic sharpnose shark"   "Blacktip shark"
##  [9] "Spinner shark"              "Blacknose shark"
## [11] "Bonnethead shark"           "Sandbar shark"
## [13] "Scalloped hammerhead"       "Finetooth shark"
## [15] "Tope shark"                 "Broadnose sevengill shark"
## [17] "Smooth hammerhead"          "Lemon shark"
## [19] "Grey reef shark"            "Whitetip reef shark"
## [21] "Tiger shark"                "Blacktip reef shark"
## [23] "Tawny nurse shark"          "Silky shark"
## [25] "Whale shark"                "Silvertip shark"
## [27] "Round ribbontail ray"       "Ribbontail stingray"
```

Now we can see all the different common names for sharks in this data set.

TASK

What are the different Countries in this data set?

____

names()

This function is useful for determining what the column names in your data set are.

```
names(shark_data)
```

```
##  [1] "ID"          "Binomial"    "Class"       "Order"       "Family"
##  [6] "Genus"       "Species"     "Common_name" "Location"    "Country"
## [11] "Region"      "Latitude"    "Longitude"   "Year"        "Abundance"
```

mean(), max(), min(), median()

These will compute these basic mathematical equations on the data given to the function to summarize your data. Note for these you will need to give the function ONLY numeric data (numbers only no words). So to use mean() we should select the Abundance column.

```r
mean(shark_data$Abundance)
```

```
## [1] 0.6136064
```

Now we know the mean abundance (catch per unit effort) for sharks in the whole data frame is 0.614

TASK

Determine the min, max and median for the abundance of sharks in the whole data frame.

---

length()

This function will give the length of elements within an object. However this will tell us different things based on whether we have a data frame object or a vector.

Lets start by looking at the length of the shark data

```r
length(shark_data)
```

```
## [1] 15
```

15? Does this look like a number we have seen before? It should because its the number of columns. So in a data frame, length() will tell us the number of columns. Now lets try selecting a single column in this function.

```r
length(shark_data$ID)
```

```
## [1] 220
```

This should also look familiar because its the number of rows, so there are 220 elements in this column, which would be 220 rows.

We can also get the length of vectors.

TASK

Determine the length of the vector fruit

---

sum()

This can be used to sum a column or a vector.

TASK

Try using sum() to sum our leaf_lengths vector.

Data Manipulation

One initial thing that can be very helpful is manipulating your data so you can get it into the format you want to calculate more summary statistics. Like what if you wanted to know the average abundance of sharks for each country? or in each year? To help with this we can use:

filter()

The filter() function can be used to filter your data set. So you can filter out certain rows so you only have attributes you want. We will take our shark data and filter out the "Whale shark" rows from the Common_name column and name this object whale_sharks. This is very useful for creating subsets of your original data. So we are looking for where Common_name is equal to "Whale shark"

```r
whale_sharks <- filter(shark_data, Common_name == "Whale shark")

View(whale_sharks)
```

We can also filter with multiple criteria, we could filter out all "Shortfin mako" data that also has an abundance over 1 CPUE.

```r
abund_short_mako <- filter(shark_data,
                           Common_name == "Shortfin mako" &
                             Abundance >= 1)
```

Some important symbols for the filter() are:

- "==" is equals exactly (a single = will not work in this usage)
- "!=" is does not equal
- ">" is greater than
- "<" is less than
- ">=" is greater than or equal to
- "<=" is less than or equal to
- "&" is and
- "|" is or

you can use any of these within the filter function to define what you are filterin out of a column.

TASK

Filter the shark data so there is only data from after 2010, and that the data set does not include the "International Waters" Region. Name this object Recent_Coastal_Sharks. Then check using functions to ensure this is correct

select()

The select() can be used to select columns from your data, or to remove columns from your data. We will select the common name and abundance column for one data frame and for another we will select all except the ID column.

```
simple_shark_data <- select(shark_data, Common_name, Abundance)
# you can list as many columns as you want, just make sure they are
# spelled correctly and separated by a comma

head(simple_shark_data) # this will show us the frist 6 rows of our data frame
```

```
## # A tibble: 6 x 2
##   Common_name     Abundance
##   <chr>               <dbl>
## 1 Porbeagle shark    0.0035
## 2 Porbeagle shark    0.0041
## 3 Porbeagle shark    0.0052
## 4 Porbeagle shark    0.0068
## 5 Porbeagle shark    0.0034
## 6 Porbeagle shark    0.0035
```

```
shark_data_noID <- select(shark_data, -ID)

head(shark_data_noID)
```

```
## # A tibble: 6 x 14
##   Binomial  Class Order Family Genus Species Common_name Location Country Region
##   <chr>     <chr> <chr> <chr>  <chr> <chr>   <chr>       <chr>    <chr>   <chr>
## 1 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 2 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 3 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 4 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 5 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## 6 Lamna_na~ Elas~ Lamn~ Lamni~ Lamna nasus   Porbeagle ~ NAFO Su~ Canada  North~
## # i 4 more variables: Latitude <dbl>, Longitude <dbl>, Year <int>,
## #   Abundance <dbl>
```

group_by()

The group_by() is useful when paired with a summarize function which we will learn next. This function will group your data based on the different values in a column. So for our data we could group our data by Common_name. This will mean we can then apply the summarize function to each different group in the Common_name column. So we could do something like average the abundance of each species of shark. Very useful. First lets group our data by the Common_name and call it sharks_grouped.

```
sharks_grouped <- group_by(shark_data, Common_name)
# you can group by as many columns as you would like, just make sure they
# spelled correctly and separated by a comma

#now to check that our data is grouped we can call the structure
str(sharks_grouped)
```

```
## gropd_df [220 x 15] (S3: grouped_df/tbl_df/tbl/data.frame)
##  $ ID         : num [1:220] 1934 1934 1934 1934 1934 ...
##  $ Binomial   : chr [1:220] "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" "Lamna_nasus" ...
##  $ Class      : chr [1:220] "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" "Elasmobranchii" ...
##  $ Order      : chr [1:220] "Lamniformes" "Lamniformes" "Lamniformes" "Lamniformes" ...
##  $ Family     : chr [1:220] "Lamnidae" "Lamnidae" "Lamnidae" "Lamnidae" ...
##  $ Genus      : chr [1:220] "Lamna" "Lamna" "Lamna" "Lamna" ...
##  $ Species    : chr [1:220] "nasus" "nasus" "nasus" "nasus" ...
##  $ Common_name: chr [1:220] "Porbeagle shark" "Porbeagle shark" "Porbeagle shark" "Porbeagle shark"
##  $ Location   : chr [1:220] "NAFO Subareas 3-6, Canada" "NAFO Subareas 3-6, Canada" "NAFO Subareas 3-
##  $ Country    : chr [1:220] "Canada" "Canada" "Canada" "Canada" ...
##  $ Region     : chr [1:220] "North America" "North America" "North America" "North America" ...
##  $ Latitude   : num [1:220] 43.3 43.3 43.3 43.3 43.3 ...
##  $ Longitude  : num [1:220] -54.8 -54.8 -54.8 -54.8 -54.8 ...
##  $ Year       : int [1:220] 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 ...
##  $ Abundance  : num [1:220] 0.0035 0.0041 0.0052 0.0068 0.0034 0.0035 0.0036 0.0025 0.0005 0.0015 ..
##  - attr(*, "groups")= tibble [28 x 2] (S3: tbl_df/tbl/data.frame)
##   ..$ Common_name: chr [1:28] "Atlantic sharpnose shark" "Blacknose shark" "Blacktip reef shark" "Bla
##   ..$ .rows      : list<int> [1:28]
##   .. ..$ : int [1:3] 149 150 151
##   .. ..$ : int [1:3] 158 159 160
##   .. ..$ : int [1:2] 203 204
##   .. ..$ : int [1:3] 152 153 154
##   .. ..$ : int [1:34] 35 36 37 38 39 40 41 42 43 44 ...
##   .. ..$ : int [1:3] 161 162 163
##   .. ..$ : int [1:2] 179 180
##   .. ..$ : int [1:3] 170 171 172
##   .. ..$ : int [1:33] 105 106 107 108 109 110 111 112 113 114 ...
##   .. ..$ : int [1:4] 193 194 195 196
##   .. ..$ : int [1:3] 190 191 192
##   .. ..$ : int [1:12] 1 2 3 4 5 6 7 8 9 10 ...
##   .. ..$ : int [1:2] 219 220
##   .. ..$ : int [1:2] 217 218
##   .. ..$ : int [1:33] 72 73 74 75 76 77 78 79 80 81 ...
##   .. ..$ : int [1:3] 164 165 166
##   .. ..$ : int [1:7] 167 168 169 205 206 207 208
##   .. ..$ : int [1:18] 58 59 60 61 62 63 64 65 66 67 ...
##   .. ..$ : int [1:2] 211 212
##   .. ..$ : int [1:2] 215 216
##   .. ..$ : int [1:9] 181 182 183 184 185 186 187 188 189
##   .. ..$ : int [1:3] 155 156 157
##   .. ..$ : int [1:2] 209 210
##   .. ..$ : int [1:22] 13 14 15 16 17 18 19 20 21 22 ...
##   .. ..$ : int [1:2] 201 202
##   .. ..$ : int [1:2] 177 178
##   .. ..$ : int [1:2] 213 214
##   .. ..$ : int [1:4] 197 198 199 200
##   .. ..@ ptype: int(0)
##   ..- attr(*, ".drop")= logi TRUE
```

Great, now we can see that they type of data set has changed to a grouped data set.

summarize()

Now lets use the summarize function to average the abundance of our sharks. We get to pick the column name we want to have R put these new averages in, I am going to choose Average_Abund

```
shark_averages <- summarize(sharks_grouped, Average_Abund = mean(Abundance))

shark_averages
```

```
## # A tibble: 28 x 2
##    Common_name               Average_Abund
##    <chr>                             <dbl>
##  1 Atlantic sharpnose shark         1.44
##  2 Blacknose shark                  0.162
##  3 Blacktip reef shark              0.115
##  4 Blacktip shark                   0.847
##  5 Blue shark                       1.57
##  6 Bonnethead shark                 0.810
##  7 Broadnose sevengill shark        0.0162
##  8 Finetooth shark                  0.641
##  9 Great white shark                0.551
## 10 Grey reef shark                  1.00
## # i 18 more rows
```

Our final data frame should look like this, if you only have one single value, you probably used the original shark_data data frame so it just took an average of the entire data set, make sure you use the grouped data frame.

mutate()

The last function we will talk about before discussing how to make this process quicker is the mutate(). This can be used to mutate columns and create new columns.

Combining with a Pipe

Plotting Data

Histogram

Scatter Plot

Box Plot

Stats Intro

Normality Test

Homocedasticity Test

T-test

ANOVA and Tukey Test

cor.test

regression

Now What?