ResBaz Arizona 2020 Intro to R

Adriana Picoral

2020-05-17

Contents

Chapter 1

Before we start

This is a two part intro to R workshop. Part I introduces the basics of coding in R, including how to manipulate objects, use functions, and write if statements, for loops, and simple functions. Part II is based on the tidyverse package, and it covers how to load, inspect, and explore data in R. While learners at different expertise levels are welcome to attend, these workshops were designed for participants with no or little programming experience.

This lesson plan was created based on many other resources that are already available online, mainly Claudia A Engel's "Introduction to R" bookdown available at https://cengel.github.io/R-intro and "Programming with R" from Software Carpentry available at http://swcarpentry.github.io/r-novice-inflammation.

1.1 Installing R and R Studio

If you are running your R code in your computer, you need to install both R and RStudio. Alternatively, you can create a free account at http://rstudio.cloud and run your R code in the cloud. Either way, we will be using the same IDE (i.e., RStudio).

What's an **IDE**? IDE stands for **i**ntegrated **d**evelopment **e**nvironment, and its goal is to facilitate coding by integrating a **text editor**, a **console** and other tools into one window.

1.1.1 I've never installed R and RStudio in my computer OR I'm not sure I have R and RStudio installed in my computer

1. Download and install R from https://cran.r-project.org
2. Download and install RStudio from https://rstudio.com/products/rstudio/download/#download

1.1.2 I already have R and RStudio installed

1. Open RStudio
2. Check your R version by entering `sessionInfo()` on your console.
3. The latest release for R was April 24, 2020 (R version 4.0.0, 2020-04-24, Arbor Day). If your R version is older than the most recent version, please follow step 1 in the previous section to update R.
4. Check your RStudio version, if your version is older than Version 1.2.5042, please follow step 2 in the previous section to update RStudio.

How often should I update R and RStudio? Always make sure that you have the latest version of R, RStudio, and the packages you're using in your code to ensure you are not running into bugs that are caused by having older versions installed in your computer.

When asked, Jenny Bryan summarizes the importance of keeping your system up-to-date saying that "You will always eventually have a reason that you must update. So you can either do that very infrequently, suffer with old versions in the middle, and experience great pain at update. Or admit that maintaining your system is a normal ongoing activity, and do it more often."

You can ensure your packages are also up-to-date by clicking on "Tools" on your RStudio top menu bar, and selecting "Check for Packages Updates…"

Chapter 2

Intro to R Part I

2.1 Getting to know your IDE

What's an **IDE**? IDE stands for **i**ntegrated **d**evelopment **e**nvironment, and its goal is to facilitate coding by integrating a **text editor**, a **console** and other tools into one window.

We are using RStudio as our IDE for this workshop. You can either download and install R and RStudio on your computer (for instructions on how to do so, see the "Before we start" section) or create a free account at http://rstudio.cloud and run your R code in the cloud.

In this part of the workshop we will start an R project and situating ourselves around our IDE.

Why create a RStudio project? RStudio projects make it easier to keep your projects organized, since each project has their own working directory, workspace, history, and source documents.

1. Start a new R project
2. Create a new R script
3. Save that R script as 01-intro_to_r_part_one

Take a moment to look around your IDE. What are the main panes on the RStudio interface. What are the 4 main areas of the interface? Can you guess what each area is for?

2.2 Operations and Objects

Let's start by using R as a calculator. On your **console** type `3 + 3` and hit enter.

```
3 + 3
```

```
## [1] 6
```

What symbols do we use for all basic operations (addition, subtraction, multiplication, and division)? What happens if you type `3 +`?

Let's save our calculation into an object, by using the assignment symbol `<-`.

```
sum_result <- 3 + 3
```

Take a moment to look around your IDE once again. What has changed?

Now, let's use this new object in our calculation

```
sum_result + 3
```

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

Take a moment to look around your IDE once again. Has anything changed?

What else can we do with an object?

```
class(sum_result)
```

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

R is primarily a functional programming language. That means that there pre-programmed functions in base R such as `class()` and that you can also write your own functions (more on that later).

Type `?class` in your console and hit enter to get more information about this function.

CHALLENGE

Create an object called `daisys_age` that holds the number 8. Multiply `daisys_age` by 4 and save the results in another object called `daisys_human_age`

Imagine I had multiple pets (unfortunately, that is not true, Daisy is my only pet). I can create a **vector** to hold multiple numbers representing the age of each of my pets.

```
my_pets_ages <- c(8, 2, 6, 3, 1)
```

Take a moment to look around your IDE once again. What has changed?

What is the class of the object `my_pets_ages`?

Now let's multiply this vector by 4.

```
my_pets_ages * 4
```

```
## [1] 32  8 24 12  4
```

Errors are pretty common when writing code in any programming language, so be ready to read error messages and debug your code. Let's insert a typing error in our previous code:

```
my_pets_ages <- c(8, 2, 6, '3', 1)
```

CHALLENGE

Try to multiply `my_pets_ages` by 4. What happens? How can we debug our code to find out what is causing the error?

2.3 Dataframes

You will rarely work with individual numeric values, or even individual numeric vectors. Often, we have information organized in dataframes, which is R's version of a spreadsheet.

Let's go back to my imaginary pet's ages (make sure you have the correct vector in your global environment).

```
my_pets_ages <- as.numeric(my_pets_ages)
```

We will now create a vector of strings or characters that holds my imaginary pets' names (we have to be careful to keep the same order then the `my_pets_ages` vector).

```
my_pets_names <- c('Daisy', 'Violet', 'Lily', 'Iris', 'Poppy')
```

Let's now create a dataframe that contains info about my pets.

```
# create dataframe
my_pets <- data.frame(name = my_pets_names, age = my_pets_ages)

# print out dataframe
my_pets
```

```
##     name age
## 1  Daisy   8
## 2 Violet   2
## 3   Lily   6
## 4   Iris   3
## 5  Poppy   1
```

CHALLENGE

There's a number of functions you can run on dataframes. Try running the following functions on `my_pets`:

- summary()
- nrow()
- ncol()
- dim()

What other functions can/do you think/know of?

2.4 Slicing you dataframe

There are different ways you can slice or subset your dataframe.

You can use indices for rows and columns.

```
my_pets[1,]
```

```
##    name age
## 1 Daisy   8
```

```
my_pets[, 1]
```

```
## [1] "Daisy"  "Violet" "Lily"   "Iris"   "Poppy"
```

```
my_pets[1, 1]
```

```
## [1] "Daisy"
```

You can use a column name or a row name instead of an index.

```
my_pets[, 'age']
```

```
## [1] 8 2 6 3 1
```

```
my_pets['1', ]
```

```
##    name age
## 1 Daisy   8
```

```
my_pets['1', 'age']
```

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

Or you can use `$` to retrieve values from a column.

```
my_pets$age
```

```
## [1] 8 2 6 3 1
```

```
my_pets$age[1]
```

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

You can also use comparisons to filter your dataframe

```
# get index with which() function
which(my_pets$age == 8)
```

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

```
# use which() inside dataframe indexing my_pets[row_number, column_number]
my_pets[which(my_pets$age == 8),]
```

```
##    name age
## 1 Daisy   8
```

```
my_pets[which(my_pets$age == 8), 1]
```

```
## [1] "Daisy"
```

```
my_pets[which(my_pets$age == 8), 'name']
```

```
## [1] "Daisy"
```

```
my_pets[which(my_pets$age == 8),]$name
```

```
## [1] "Daisy"
```

CHALLENGE

Print out a list of pet names that are older than 3.

2.5 Adding new variables (i.e., columns) to your dataframe

So far the `my_pets` dataframe has two columns: name and age.

Let's add a third column with the pets' ages in human years. For that, we are going to use `$` on with a variable (or column) name that does not exist in our dataframe yet. We will then assign to this variable the value in the `age` column multiplied by 4.

```
# create new column called human_years
my_pets$human_years <- my_pets$age * 4

# print dataframe
my_pets
```

```
##      name age human_years
## 1  Daisy   8          32
## 2 Violet   2           8
## 3   Lily   6          24
## 4   Iris   3          12
## 5  Poppy   1           4
```

Inspect the new `my_pets` dataframe. What dimensions does it have now? How could you get a list of just the human years values in the data frame?

2.6 Descriptive stats on dataframes

Let's explore some functions for descriptive statistics.

CHALLENGE

Try running the following functions on `my_pets$age` and `my_pets$human_years`:

- mean()
- sd()
- median()
- max()
- min()
- range()

What other functions can/do you think/know of?

2.7 For loops

Besides implementing operations on an entire column (e.g., `my_pets$age * 4` multiplies each value in the `age` column of `my_pets` dataframe by 4), you can loop through each element in your dataframe column using a for loop.

There are two ways of writing a for loop in R.

```
for (pet in my_pets$name) {
  print(pet)
}
```

```
## [1] "Daisy"
## [1] "Violet"
## [1] "Lily"
## [1] "Iris"
## [1] "Poppy"
```

```
for (i in c(1:5)) {
  print(my_pets$name[i])
}
```

```
## [1] "Daisy"
## [1] "Violet"
## [1] "Lily"
## [1] "Iris"
## [1] "Poppy"
```

CHALLENGE

Write a `for` loop to print each pets' name and age. You can use the function `paste()` to combined the two variables into one line.

Remember you can enter `?paste` in your console to get information on how to use this function.

Take a moment to look around your IDE once again. What objects do you have in your environment?

2.8 If blocks

Maybe calculating a pet's age in human years is more complex than just multiplying it by 4 (or 7?). The American Kennel club has the following on how to calculate dog years to human years:

- 15 human years equals the first year of a medium-sized dog's life.
- Year two for a dog equals about nine years for a human.
- And after that, each human year would be approximately five years for a dog.

Let's first figure out all the if statements we need for this

```
this_dogs_age <- my_pets$age[1]

this_dogs_human_age <- 15

if (this_dogs_age >= 2) {
  this_dogs_human_age <- this_dogs_human_age + 9
}

if (this_dogs_age >=3) {
  this_dogs_human_age <- this_dogs_human_age + ((this_dogs_age - 2) * 5)
}
```

Now, let's create a for loop to calculate human age for every pet.

```r
for (i in c(1:5)) {
  # store dog i age in an object
  this_dogs_age <- my_pets$age[i]

  # 15 human years equals the first year of a medium-sized dog's life.
  this_dogs_human_years <- 15

  # if the pet is two years or older
  if (this_dogs_age >= 2) {
    # Year two for a dog equals about nine years for a human.
    this_dogs_human_years <- this_dogs_human_years + 9

    # And after that, each human year would be approximately five years for a dog.
    partial_dog_age <- (this_dogs_age - 2) * 5

    # sum up both parts
    this_dogs_human_years <- this_dogs_human_years + partial_dog_age
  }
  # add the final calculation to the dataframe
  my_pets$human_years2[i] <- this_dogs_human_years
}

# print dataframe
my_pets
```

```
##     name age human_years human_years2
## 1  Daisy   8          32           54
## 2 Violet   2           8           24
## 3   Lily   6          24           44
## 4   Iris   3          12           29
## 5  Poppy   1           4           15
```

2.9 Functions

Functions are extremely useful to make your R code more organized and reusable.

The main structure of a function is `object_name <- function() code_here`,Here's an example of a simple function.

```r
human_years <- function(pets_age) {

  # 15 human years equals the first year of a medium-sized dog's life.
  human_years <- 15
  # if the pet is two years or older
  if (pets_age >= 2) {
    # Year two for a dog equals about nine years for a human.
    # And after that, each human year would be approximately five years for a dog.
    human_years <- human_years + 9 + (pets_age - 2) * 5

  }
  return(human_years)
}

human_years(3)
```

```
## [1] 29
```

```
my_pets$human_years2 <- sapply(my_pets$age, human_years)
```

Read more on writing your own functions: Nice R Code - Functions

2.10 Putting it all together

Read and run the code below that provides some info on our `my_pets` dataframe.

```
# get number of rows for the for loop
how_many_pets <- nrow(my_pets)

# a for loop to print info on each pet
for (i in c(1:how_many_pets)) {
  # paste info with some prose
  info_to_print <- paste(my_pets$name[i], 'is',
                         my_pets$age[i],
                         'years old in pet years, which is equivalent to',
                         my_pets$human_years[i], 'human years.')
  # print out the info for pet i
  print(info_to_print)
} # end of for loop to print info on each pet
```

```
## [1] "Daisy is 8 years old in pet years, which is equivalent to 32 human years."
## [1] "Violet is 2 years old in pet years, which is equivalent to 8 human years."
## [1] "Lily is 6 years old in pet years, which is equivalent to 24 human years."
## [1] "Iris is 3 years old in pet years, which is equivalent to 12 human years."
## [1] "Poppy is 1 years old in pet years, which is equivalent to 4 human years."
```

```
# print name of oldest pet
## get max age
max_age <- max(my_pets$age)
```

```
max_age
```

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

```
## get index of max_age
row_max_age <- which(my_pets$age == max_age)
row_max_age
```

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

```
## print info for oldest pet
oldest_pet <- my_pets$name[row_max_age]
oldest_pet
```

```
## [1] "Daisy"
```

```
print(paste(oldest_pet, 'is the oldest pet'))
```

```
## [1] "Daisy is the oldest pet"
```

CHALLENGE

Add to the code above, to print the following information:

1. the name of the youngest pet
2. the mean pet age
3. any other info you find relevant

2.11 Note on coding style

Coding style refers to how you name your objects and functions, how you comment your code, how you use spacing throughout your code, etc. If your coding style is consistent, your code is easier to read and easier to debug as a result. Here's some guides, so you can develop your own coding style:

- The tidyverse style guide
- Hadley Wickham's Advance R coding style
- Google's R Style Guide

Chapter 3

Intro to R Part II

3.1 Installing and using packages in R

There are a lot of R packages out here (check the Comprehensive R Archive Network, i.e., CRAN, for a full list). That is one of the beautiful things about R, anyone can create an R package to share their code (check out the workshop on how to create your own R package later this week).

Open your RStudio (if you haven't attended the first part of this workshop, please check the "Before we start" section for instructions on how to download and/or update R and RStudio).

The function to install packages in R is `install.packages()`. We will be working with TidyVerse today, which is a collection of R packages carefully designed for data science.

Let's install tidyverse (this may take a while).

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

You need to install any package only once (remember to check for new package versions and to keep your packages updated). However, with every new R session, you need to load the packages you are going to use by using the `library()` function.

```
library(tidyverse)
```

Note that when calling the `install.packages()` function you need to enter the package name between quotation marks (e.g., "tidyverse"). When you call the `library()` function, you don't use quotation marks (e.g., tidyverse).

3.2 Acquire data

First, we will download the data we are going to be using today.

1. Go to https://www.kaggle.com/neuromusic/avocado-prices/data and click on the "Download (2 MB)" button
2. Find the zip file (avocado-prices.zip) you downloaded and unzip it
3. On the same level as your project folder, add a "new folder" called "data"
4. Move, copy, or upload the data file (avocado.csv) to the "data" folder

3.3 Load data in R

For this workshop, we will be using data from kaggle. In the previous section, you created a *data* folder in your project folder, which should contain the *avocado.csv* data file.

Although we are working within an R project, which sets the working directory automatically for you, it's good practice to check what folder you are working from by calling the `getwd()` function.

```
getwd()
```

```
## [1] "/Users/adriana/Desktop/workshops/resbaz_2020/intro_to_R"
```

You can also list the contents of your `data` folder by using the `dir()` function.

```
dir("data")
```

```
## [1] "avocado.csv"
```

We will use the `read_csv()` function from the `readr` package (which is part of `tidyverse`) to read data in.

```
avocado_data <- read_csv("data/avocado.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   Date = col_date(format = ""),
##   AveragePrice = col_double(),
##   `Total Volume` = col_double(),
##   `4046` = col_double(),
##   `4225` = col_double(),
##   `4770` = col_double(),
##   `Total Bags` = col_double(),
##   `Small Bags` = col_double(),
##   `Large Bags` = col_double(),
##   `XLarge Bags` = col_double(),
##   type = col_character(),
##   year = col_double(),
##   region = col_character()
## )
```

CHALLENGE

Reading warnings - R often prints out warnings in red (these are not always errors). What information did you get when loading your data?

3.4 Inspect your data

Now, let's inspect our dataframe. As usual, there are multiple ways of inspecting your data.

Here's one of my favorites:

```
# get an overview of the data frame
glimpse(avocado_data)
```

```
## Rows: 18,249
## Columns: 14
## $ X1             <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15...
## $ Date           <date> 2015-12-27, 2015-12-20, 2015-12-13, 2015-12-06, 201...
## $ AveragePrice   <dbl> 1.33, 1.35, 0.93, 1.08, 1.28, 1.26, 0.99, 0.98, 1.02...
## $ `Total Volume` <dbl> 64236.62, 54876.98, 118220.22, 78992.15, 51039.60, 5...
## $ `4046`         <dbl> 1036.74, 674.28, 794.70, 1132.00, 941.48, 1184.27, 1...
## $ `4225`         <dbl> 54454.85, 44638.81, 109149.67, 71976.41, 43838.39, 4...
## $ `4770`         <dbl> 48.16, 58.33, 130.50, 72.58, 75.78, 43.61, 93.26, 80...
## $ `Total Bags`   <dbl> 8696.87, 9505.56, 8145.35, 5811.16, 6183.95, 6683.91...
## $ `Small Bags`   <dbl> 8603.62, 9408.07, 8042.21, 5677.40, 5986.26, 6556.47...
## $ `Large Bags`   <dbl> 93.25, 97.49, 103.14, 133.76, 197.69, 127.44, 122.05...
## $ `XLarge Bags`  <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ type           <chr> "conventional", "conventional", "conventional", "con...
## $ year           <dbl> 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015, 2015...
## $ region         <chr> "Albany", "Albany", "Albany", "Albany", "Albany", "A...
```

```
summary(avocado_data)
```

```
##        X1             Date             AveragePrice    Total Volume
##  Min.   : 0.00   Min.   :2015-01-04   Min.   :0.440   Min.   :      85
##  1st Qu.:10.00   1st Qu.:2015-10-25   1st Qu.:1.100   1st Qu.:   10839
##  Median :24.00   Median :2016-08-14   Median :1.370   Median :  107377
##  Mean   :24.23   Mean   :2016-08-13   Mean   :1.406   Mean   :  850644
##  3rd Qu.:38.00   3rd Qu.:2017-06-04   3rd Qu.:1.660   3rd Qu.:  432962
##  Max.   :52.00   Max.   :2018-03-25   Max.   :3.250   Max.   :62505647
##       4046              4225              4770           Total Bags
##  Min.   :       0   Min.   :       0   Min.   :      0   Min.   :       0
##  1st Qu.:     854   1st Qu.:    3009   1st Qu.:      0   1st Qu.:    5089
##  Median :    8645   Median :   29061   Median :    185   Median :   39744
##  Mean   :  293008   Mean   :  295155   Mean   :  22840   Mean   :  239639
##  3rd Qu.:  111020   3rd Qu.:  150207   3rd Qu.:   6243   3rd Qu.:  110783
##  Max.   :22743616   Max.   :20470573   Max.   :2546439   Max.   :19373134
##    Small Bags         Large Bags       XLarge Bags           type
##  Min.   :       0   Min.   :      0   Min.   :     0.0   Length:18249
##  1st Qu.:    2849   1st Qu.:    127   1st Qu.:     0.0   Class :character
##  Median :   26363   Median :   2648   Median :     0.0   Mode  :character
##  Mean   :  182195   Mean   :  54338   Mean   :  3106.4
##  3rd Qu.:   83338   3rd Qu.:  22029   3rd Qu.:   132.5
##  Max.   :13384587   Max.   :5719097   Max.   :551693.7
##       year         region
##  Min.   :2015   Length:18249
##  1st Qu.:2015   Class :character
##  Median :2016   Mode  :character
##  Mean   :2016
##  3rd Qu.:2017
##  Max.   :2018
```

```
# check some of the categorical variables
unique(avocado_data$type)
```

```
## [1] "conventional" "organic"
```

```
unique(avocado_data$region)
```

```
##  [1] "Albany"              "Atlanta"             "BaltimoreWashington"
##  [4] "Boise"               "Boston"              "BuffaloRochester"
##  [7] "California"          "Charlotte"           "Chicago"
## [10] "CincinnatiDayton"    "Columbus"            "DallasFtWorth"
## [13] "Denver"              "Detroit"             "GrandRapids"
## [16] "GreatLakes"          "HarrisburgScranton"  "HartfordSpringfield"
## [19] "Houston"             "Indianapolis"        "Jacksonville"
## [22] "LasVegas"            "LosAngeles"          "Louisville"
## [25] "MiamiFtLauderdale"   "Midsouth"            "Nashville"
## [28] "NewOrleansMobile"    "NewYork"             "Northeast"
## [31] "NorthernNewEngland"  "Orlando"             "Philadelphia"
## [34] "PhoenixTucson"       "Pittsburgh"          "Plains"
## [37] "Portland"            "RaleighGreensboro"   "RichmondNorfolk"
## [40] "Roanoke"             "Sacramento"          "SanDiego"
## [43] "SanFrancisco"        "Seattle"             "SouthCarolina"
## [46] "SouthCentral"        "Southeast"           "Spokane"
## [49] "StLouis"             "Syracuse"            "Tampa"
## [52] "TotalUS"             "West"                "WestTexNewMexico"
```

Do you know any other ways of checking your data?

CHALLENGE

Which variables are numeric? Which are categorical?

What functions do you remember from Part I? Run these functions (e.g., `mean()`) on your numeric variables.

3.5 Explore your data

We will be using the package `dplyr` (which is also part of `tidyverse`) to do an exploratory analysis of our data.

The package `dplyr` most used function is `%>%` (called the pipe). The pipe allows you to "pipe" (or redirect) objects into functions. (hint: use ctrl+shift+m or cmd+shift+m as a shortcut for typing `%>%`).

Here's how to pipe the `avocado_data` object into the `summary()` function

```
# get an overview of the data frame
avocado_data %>%
  summary()
```

```
##        X1             Date             AveragePrice    Total Volume
##  Min.   : 0.00   Min.   :2015-01-04   Min.   :0.440   Min.   :      85
##  1st Qu.:10.00   1st Qu.:2015-10-25   1st Qu.:1.100   1st Qu.:   10839
##  Median :24.00   Median :2016-08-14   Median :1.370   Median :  107377
##  Mean   :24.23   Mean   :2016-08-13   Mean   :1.406   Mean   :  850644
##  3rd Qu.:38.00   3rd Qu.:2017-06-04   3rd Qu.:1.660   3rd Qu.:  432962
##  Max.   :52.00   Max.   :2018-03-25   Max.   :3.250   Max.   :62505647
##       4046               4225               4770             Total Bags
##  Min.   :       0   Min.   :       0   Min.   :      0   Min.   :       0
##  1st Qu.:     854   1st Qu.:    3009   1st Qu.:      0   1st Qu.:    5089
##  Median :    8645   Median :   29061   Median :    185   Median :   39744
##  Mean   :  293008   Mean   :  295155   Mean   :  22840   Mean   :  239639
##  3rd Qu.:  111020   3rd Qu.:  150207   3rd Qu.:   6243   3rd Qu.:  110783
##  Max.   :22743616   Max.   :20470573   Max.   :2546439   Max.   :19373134
##    Small Bags         Large Bags       XLarge Bags           type
##  Min.   :       0   Min.   :      0   Min.   :     0.0   Length:18249
##  1st Qu.:    2849   1st Qu.:    127   1st Qu.:     0.0   Class :character
##  Median :   26363   Median :   2648   Median :     0.0   Mode  :character
##  Mean   :  182195   Mean   :  54338   Mean   :  3106.4
##  3rd Qu.:   83338   3rd Qu.:  22029   3rd Qu.:   132.5
##  Max.   :13384587   Max.   :5719097   Max.   :551693.7  
##       year         region         
##  Min.   :2015   Length:18249      
##  1st Qu.:2015   Class :character  
##  Median :2016   Mode  :character  
##  Mean   :2016                     
##  3rd Qu.:2017                     
##  Max.   :2018                     
```

The pipe allows us to apply multiple functions to the same object.

Let's start by selecting one column in our data.

```
avocado_data %>%
  select(type)
```

```
## # A tibble: 18,249 x 1
##    type        
##    <chr>       
##  1 conventional
##  2 conventional
##  3 conventional
##  4 conventional
##  5 conventional
##  6 conventional
##  7 conventional
##  8 conventional
##  9 conventional
## 10 conventional
## # ... with 18,239 more rows
```

Now let's add another pipe to get unique values in this column.

```
avocado_data %>%
  select(type) %>%
  unique()
```

```
## # A tibble: 2 x 1
##   type        
##   <chr>       
## 1 conventional
## 2 organic
```

3.5.1 Group by + count

One of the most useful pipe combinations is `group_by()` and `count()`.

```
avocado_data %>%
  group_by(type) %>%
  count()
```

```
## # A tibble: 2 x 2
## # Groups:   type [2]
##   type            n
##   <chr>       <int>
## 1 conventional  9126
## 2 organic       9123
```

We can add more variables to the `group_by()` function.

```
avocado_data %>%
  group_by(region, type) %>%
  count()
```

```
## # A tibble: 108 x 3
## # Groups:   region, type [108]
##    region              type             n
##    <chr>               <chr>        <int>
##  1 Albany              conventional   169
##  2 Albany              organic        169
##  3 Atlanta             conventional   169
##  4 Atlanta             organic        169
##  5 BaltimoreWashington conventional   169
##  6 BaltimoreWashington organic        169
##  7 Boise               conventional   169
##  8 Boise               organic        169
##  9 Boston              conventional   169
## 10 Boston              organic        169
## # ... with 98 more rows
```

3.5.2 Group by + summarise

We can also use the `summarise()` function after `group_by()`. Inside `summarise()` you can use other functions such as `sum()`.

```
avocado_data %>%
  group_by(region, type) %>%
  summarise(total_volume = sum(`Total Volume`))
```

```
## # A tibble: 108 x 3
## # Groups:   region [54]
##    region              type         total_volume
##    <chr>               <chr>               <dbl>
##  1 Albany              conventional    15700611.
##  2 Albany              organic          367188.
##  3 Atlanta             conventional    86661392.
##  4 Atlanta             organic          1943727.
##  5 BaltimoreWashington conventional   130745575.
##  6 BaltimoreWashington organic          3968344.
##  7 Boise               conventional    14000540.
##  8 Boise               organic           412647.
##  9 Boston              conventional    94900438.
## 10 Boston              organic          2373547.
## # ... with 98 more rows
```

Let's add another pipe and arrange the results by `total_volume`.

```
avocado_data %>%
  group_by(region) %>%
  summarise(total_volume = sum(`Total Volume`)) %>%
  arrange(total_volume)
```

```
## # A tibble: 54 x 2
##    region           total_volume
##    <chr>                   <dbl>
##  1 Syracuse            10942668.
##  2 Boise               14413188.
##  3 Spokane             15565275.
##  4 Albany              16067800.
##  5 Louisville          16097002.
##  6 Pittsburgh          18806346.
##  7 BuffaloRochester    22962470.
##  8 Roanoke             25042011.
##  9 Jacksonville        28790005.
## 10 Columbus            29993361.
## # ... with 44 more rows
```

3.6 Mutate

You can use `mutate()` to add a new column to your data.

```
avocado_data %>%
  group_by(region, type) %>%
  summarise(total_type_volume = sum(`Total Volume`)) %>%
  mutate(total_volume = sum(total_type_volume))
```

```
## # A tibble: 108 x 4
## # Groups:   region [54]
##    region              type         total_type_volume total_volume
##    <chr>               <chr>                    <dbl>        <dbl>
##  1 Albany             conventional        15700611.    16067800.
##  2 Albany             organic               367188.    16067800.
##  3 Atlanta            conventional        86661392.    88605119.
##  4 Atlanta            organic              1943727.    88605119.
##  5 BaltimoreWashington conventional       130745575.   134713919.
##  6 BaltimoreWashington organic              3968344.   134713919.
##  7 Boise              conventional        14000540.    14413188.
##  8 Boise              organic               412647.    14413188.
##  9 Boston             conventional        94900438.    97273985.
## 10 Boston             organic              2373547.    97273985.
## # ... with 98 more rows
```

CHALLENGE

Use `group_by()`, `summarise()`, and `mutate()` to print out the volume percentage of *conventional* and *organic* avocado types per region. (hint: add a new variable inside `mutate()` that is the result of total_type_volume divided by total_volume)

We've been just printing our results to our console. Let's save the results are a new data frame. When assigning your `group_by()` results to a new object, make sure to add `ungroup()` as the last pipe (this will save you headaches in the future).

```
volume_type_region <- avocado_data %>%
  group_by(region, type) %>%
  summarise(total_type_volume = sum(`Total Volume`)) %>%
  mutate(total_volume = sum(total_type_volume),
         type_percentage = total_type_volume/total_volume) %>%
  ungroup()
```

CHALLENGE #1

1. Inspect your new data frame.
2. Base on `volume_type_region` calculate the mean and standard deviation of type percentage per type (i.e., conventional vs. organic). **Question**: What is the percentage of conventional vs. organic avocados sold in each region?

CHALLENGE #2

Calculate the average avocado price for each type in each region.

3.7 Filter your data

One of the "regions" in our data is `TotalUS` which is not really a specific region, but the sum of all the other regions. To calculate averages per year, for example, we need to filter our the `TotalUS` region.

```
avocado_data %>%
  filter(region != 'TotalUS')
```

```
## # A tibble: 17,911 x 14
##       X1 Date       AveragePrice `Total Volume` `4046` `4225` `4770`
##    <dbl> <date>            <dbl>          <dbl>  <dbl>  <dbl>  <dbl>
##  1     0 2015-12-27         1.33         64237.  1037. 5.45e4   48.2
##  2     1 2015-12-20         1.35         54877.   674. 4.46e4   58.3
##  3     2 2015-12-13         0.93        118220.   795. 1.09e5  130.
##  4     3 2015-12-06         1.08         78992.  1132  7.20e4   72.6
##  5     4 2015-11-29         1.28         51040.   941. 4.38e4   75.8
##  6     5 2015-11-22         1.26         55980.  1184. 4.81e4   43.6
##  7     6 2015-11-15         0.99         83454.  1369. 7.37e4   93.3
##  8     7 2015-11-08         0.98        109428.   704. 1.02e5   80
##  9     8 2015-11-01         1.02         99811.  1022. 8.73e4   85.3
## 10     9 2015-10-25         1.07         74339.   842. 6.48e4  113
## # ... with 17,901 more rows, and 7 more variables: `Total Bags` <dbl>, `Small
## #   Bags` <dbl>, `Large Bags` <dbl>, `XLarge Bags` <dbl>, type <chr>,
## #   year <dbl>, region <chr>
```

CHALLENGE

Calculate the mean type average per year. Remember to filter out the `TotalUS` region. **Question**: What is the percentage of conventional vs. organic avocados sold each year?

3.8 Conditionally mutate your data

You can use the `if_else()` function inside `mutate()` to create a new variable that is conditional on an existing variable in your data frame.

Let's create a new column in our data frame, indicating whether the average price per avocado is higher than $1.50.

```
avocado_data <- avocado_data %>%
  mutate(expensive = if_else(AveragePrice > 1.50, 1, 0))
```

Now we can calculate the percentage of expensive avocados by using the `mean()` function on our new `expensive` variable.

```
mean(avocado_data$expensive)
```

```
## [1] 0.3787605
```

CHALLENGE

Calculate the percentage of expensive avocados per region.

3.9 Pivot your data

Numerical column names refer to price lookup codes.

4046: small Hass

4225: large Hass

4770: extra large Hass

First, let's slice our data, to remove `Total Volume`.

```
avocado_data_v2 <- avocado_data %>%
  select(Date, region, `4046`, `4225`, `4770`)
```

Inspect your data.

Now, let's pivot our data frame.

```
avocado_data_longer <- avocado_data_v2 %>%
  pivot_longer(cols = c(`4046`, `4225`, `4770`),
               names_to = "lookup_code")
```

CHALLENGE

Summarize volume of each avocado type per region. Which region buys a larger portion of large avocados?