Notes 01 - An Introduction to R

STS 2300 (Spring 2025)

Updated: 2025-01-25

Table of Contents

Reading for Notes 01

- Read the Introduction for Students section of the Modern Dive textbook to acclimate yourself to our primary textbook and some topics we will cover.
- Then read Chapter 1 of the Modern Dive textbook to learn more about the programming language R.

Learning Goals for Notes 01

- Understand the difference between R and RStudio
- Gain a basic understanding of the setup of RStudio and the purpose of R packages

- Understand the basic elements of R code (objects / functions / operators)
- Be able to use functions to explore vectors and data frames

R, RStudio, and R Packages

R vs. SAS

R is a computer programming language like **SAS**, which you learned (or will learn) in STS 2120. However, there are some important differences between the two.

SAS is a *procedure-based* language. Think about how hard it is to do simple tasks (like calculating 2 + 2 or finding the mean of three numbers) in SAS. This is because SAS is organized around performing pre-set procedures on data sets that often accomplish multiple tasks at once.

R is considered a *functional* programming language. In R, we interact with objects via functions. Those objects may be data sets, but they may also be a single number or a single categorical variable, for example. This makes some simple calculations (like `2 + 2`) much simpler in R.

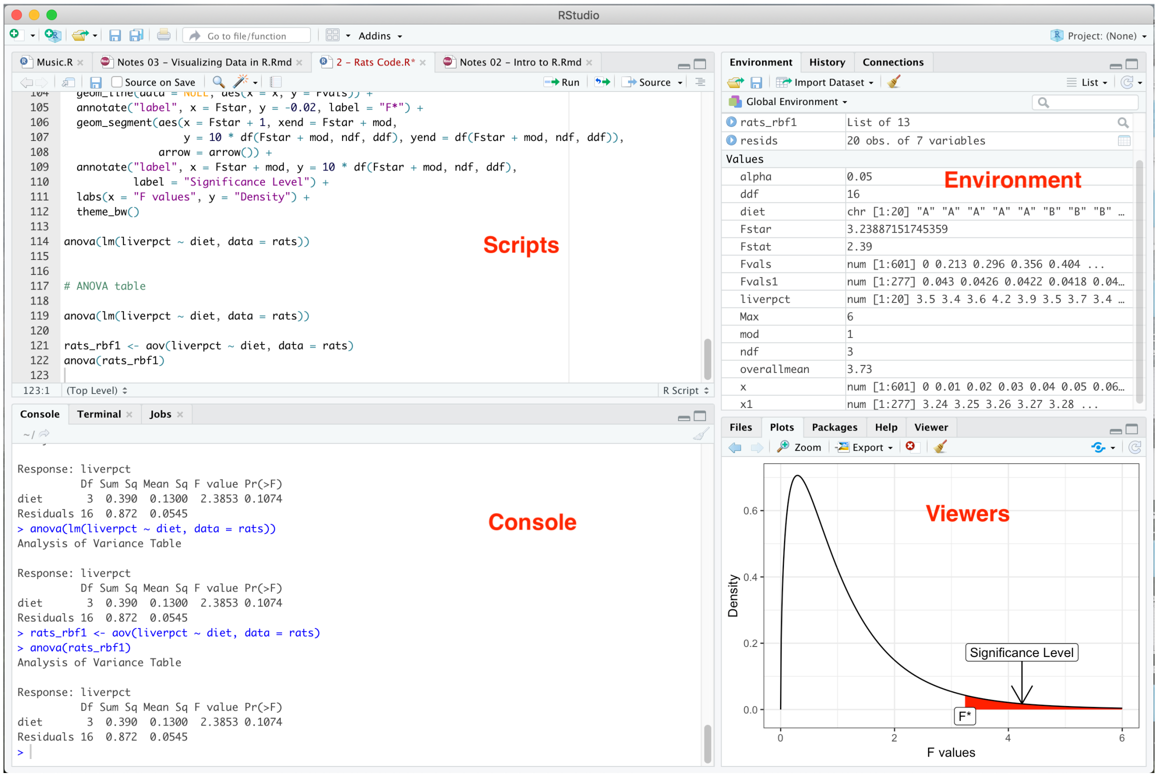
It may take some time to get used to differences between these two languages (and any other programming languages you may be familiar with), but you'll discover pros and cons of each throughout the semester.

R is also open source and has a rich community of people who are constantly creating new functionality we can utilize.

What is RStudio?

RStudio is an interface to help us use R more efficiently. Our textbook uses the analogy of R as a car's engine and RStudio as the car's dashboard. We use the dashboard to more effectively interact with the engine when we drive.

If you haven't done so yet, you should first install R and *then* install RStudio on your personal computer if you plan to use one. See section 1.1.1 in the textbook to do this. If you run into any issues, please let me know as soon as possible so that we can troubleshoot before you are under a time crunch for an assignment. If you don't have a computer on which you can install R, that's OK too. R and RStudio can be accessed from anywhere using your Elon credentials via AppsAnywhere. If you plan to use AppsAnywhere, you will find that you will need to repeat some steps that those using their own computers will not need to do.



```
  geom_line(data = NULL, aes(x = x, y = Fvals)) +
  annotate("label", x = Fstar, y = -0.02, label = "F*") +
  geom_segment(aes(x = Fstar + 1, xend = Fstar + mod,
                   y = 10 * df(Fstar + mod, ndf, ddf), yend = df(Fstar + mod, ndf, ddf)),
               arrow = arrow()) +
  annotate("label", x = Fstar + mod, y = 10 * df(Fstar + mod, ndf, ddf),
           label = "Significance Level") +
  labs(x = "F values", y = "Density") +
  theme_bw()

anova(lm(liverpct ~ diet, data = rats))


# ANOVA table

anova(lm(liverpct ~ diet, data = rats))

rats_rbf1 <- aov(liverpct ~ diet, data = rats)
anova(rats_rbf1)
```

```
Response: liverpct
          Df Sum Sq Mean Sq F value Pr(>F)
diet       3  0.390  0.1300  2.3853 0.1074
Residuals 16  0.872  0.0545
> anova(lm(liverpct ~ diet, data = rats))
Analysis of Variance Table

Response: liverpct
          Df Sum Sq Mean Sq F value Pr(>F)
diet       3  0.390  0.1300  2.3853 0.1074
Residuals 16  0.872  0.0545
> rats_rbf1 <- aov(liverpct ~ diet, data = rats)
> anova(rats_rbf1)
Analysis of Variance Table

Response: liverpct
          Df Sum Sq Mean Sq F value Pr(>F)
diet       3  0.390  0.1300  2.3853 0.1074
Residuals 16  0.872  0.0545
> 
```

rats_rbf1	List of 13
resids	20 obs. of 7 variables
Values	
alpha	0.05
ddf	16
diet	chr [1:20] "A" "A" "A" "A" "A" "B" "B" "B" …
Fstar	3.23887151745359
Fstat	2.39
Fvals	num [1:601] 0 0.213 0.296 0.356 0.404 ...
Fvals1	num [1:277] 0.043 0.0426 0.0422 0.0418 0.04…
liverpct	num [1:20] 3.5 3.4 3.6 4.2 3.9 3.5 3.7 3.4 …
Max	6
mod	1
ndf	3
overallmean	3.73
x	num [1:601] 0 0.01 0.02 0.03 0.04 0.05 0.06…
x1	num [1:277] 3.24 3.25 3.26 3.27 3.28 ...

Sample picture of RStudio

The RStudio interface is broken into four main sections.

- **Console** - In this section you can directly type and run code. Code and output will be mixed together in this section.
- **Scripts** - In this section you can type/edit code so that it can be saved and reused later. We will primarily type our code as R scripts for this reason. Code up here can be sent down to the console to run.
- **Environment** - In this section you can see what objects are loaded into R and available for you to use in your functions.
- **Viewers** - In this section you can view graphs you've made, help files, available R packages, what files are available on your computer, and more.

What are R packages?

We will learn more about using R packages in upcoming sections of notes. R is an open source software that contains a ton of things you can do right off the bat. You are also able to install what we call packages to add more functionality to your R code. Think of this like downloading apps on a phone. When someone gets a new phone there are a bunch of apps already installed on it (these are like packages built into R), but you can also download new ones that give you additional functionality (like accessing social media, playing a game, or getting more detailed weather reports). Throughout the semester, we will download packages that let us make nicer graphs (`ggplot2`), help us organize and clean our data (`dplyr`), and that let us easily do simulations based on our data (`infer`).

Just like apps on your phone, you will need to download/install the package only *once* to give you access to it. After that, you will need to open/load the package each time you want to use it though.

Writing Code in R

Looking at some sample code

The two main ways to write code in R involve typing code in the (i) console pane or (ii) in an R script. For this class, we will largely write our code in R scripts. This is because code in a script can be easily saved and rerun. This is an important aspect of **reproducibility**. It lets us update our code, re-run it with new data, share it with others, and more. Code written in an R script is still “sent” to the console to run, but we maintain a record of it for later. At this point, you may want to start an R script called **notes01.R** to save your code from Notes 01 so you can come back to it later.

Below is an example of some R code. This code consists of three lines (with the dark gray background). Lines that start with ## and have a white background represent output generated by those lines of code.

```
mynumbers <- c(4, 8, 15, 16, 23, 42)
mynumbers * 2

## [1]  8 16 30 32 46 84

max(mynumbers)

## [1] 42
```

Let’s run each of the three lines and then answer some questions based on what happens.

1) **Objects** are created in R when we store something in our environment for later use. What object(s) did we create and how did we do this?

Answer: We type a name followed by <- and then some code to determine what the object should look like.

2) **Functions** can be used on objects to carry out some sort of task. Which function(s) did we use, and what did they do? (Note: Functions are most easily identified as a word or letters with parentheses after them).

Answer: We used max() and c() functions. The max() function found the maximum of mynumbers. The c() function is used to combine multiple objects into on object.

3) The symbol `<-` is called the **assignment operator** because it assigns what's on the right into the name we choose on the left. What is different about lines that used the assignment operator versus those that didn't?

Answer: When we use the assignment operator, we usually don't have any output associated with it in the console. Instead, we create something in the environment section.

4) Why did the third line return 42 as the maximum instead of 84?

Answer: Even though we multiplied mynumbers by 2, we didn't save the result of that in the mynumbers object in the console. The mynumbers object still had 42 as the highest number.

5) Try writing code to create your own object called `mynums` that includes a few numbers of your choosing. Then write code that calculates and stores the maximum in an object called `my_max`. (Bonus: Could you find the minimum of `mynums`? What about the mean or median?)

Answer: See notes01.R for code.

6) Unlike SAS, **R is case sensitive**. Try typing `Max(mynumbers)`. How does R try to communicate that it doesn't understand what you're asking?

Answer: Error in Max(mynumbers) : could not find function "Max"

This error is letting us know that we should check if we had a type or misspelling in our function that it mentions.

7) Object and function names cannot have spaces in them (or start with a number). Try typing `my numbers <- c(1, 2, 3)`. How does R communicate this error to you?

Answer: Error: unexpected symbol in "my numbers"

Object types in R

Objects in R can take many different forms.

- A **scalar** is an object made up of a single element (e.g. `a <- 3` or `b <- "frog"`)
- A **vector** is a one-dimensional series of elements that are all the same type (e.g. `a <- c(1, 4, 2)` or `b <- c("frog", "pizza", "elon university")`)
- A **matrix** is a two-dimensional object made of elements that are all the same type

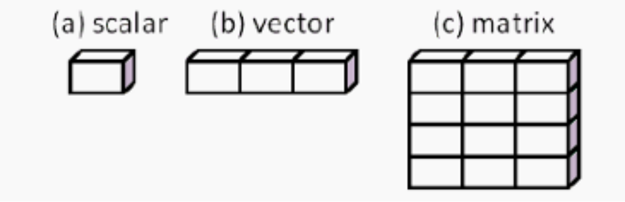


Illustration of three different object types

- A **data frame** is a two-dimensional object where each column is made up of elements of a single type, but different columns may have different element types.

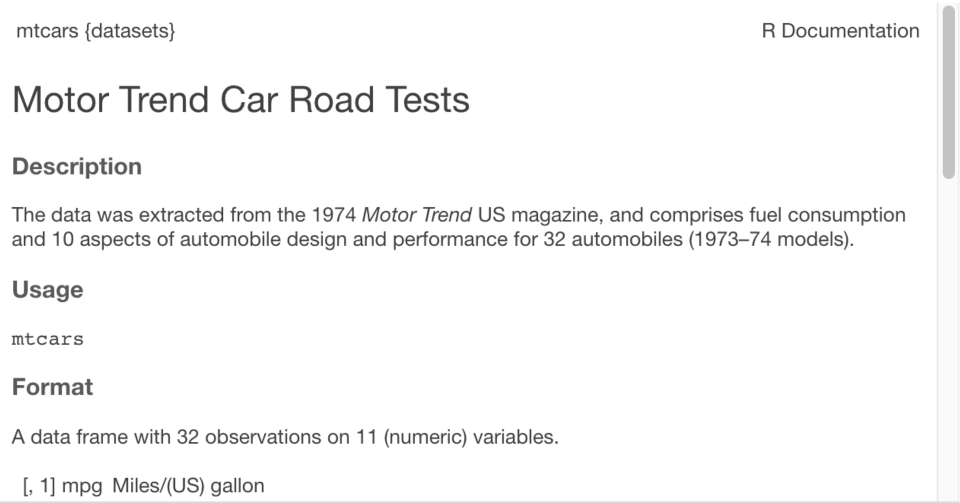
Data Frames

Data frames are what you typically think of as a data set. Usually each row will represent an observation and each column will represent a variable. There are many data sets already stored in R or available in packages we can install. We can load our own data through things like Microsoft Excel files too (which we will discuss in Notes 2). For now, let's look at one of the data sets already installed in R, called `mtcars`.

For data sets stored in R, we can use the `help()` function (or type `?` followed by the data set name) to get information about the data set.

```
?mtcars
```

When you do this, you should see the below image in the help pane of the Viewers section of RStudio



mtcars {datasets} R Documentation

Motor Trend Car Road Tests

Description

The data was extracted from the 1974 *Motor Trend* US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

Usage

```
mtcars
```

Format

A data frame with 32 observations on 11 (numeric) variables.

[, 1] mpg Miles/(US) gallon

Help file for mtcars dataset

Exploring a data set

Let's learn a few common functions for exploring data sets. Functions have **arguments** that are options we can change to alter what the function does.

Below, I'm using the `head()` function to look at the first 5 observations of the `mtcars` data set.

```
head(mtcars, n = 5)

##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
```

Question How do you think I could look at the first 10 observations? What about the last ten observations? Try it out...

Answer: We can get the first 10 observations by changing n=5 to n=10. We can use the tail() function instead of head() to look at the last 10 observations.

Note: You can also use the help files to learn more about functions. Some of these are easier to read than others, and it will likely take practice to feel comfortable with these files. The Usage, Arguments, and Examples sections are often most useful.

Below are some other options for other functions to explore our data set.

- `summary()` to get a summary of each variable
- `View()` to get a pop up window of the full data set
- `glimpse()` to see the first several values in each variable
- `skim()` to get a different summary of each variable (and of the types of variables)

(Note: The `glimpse()` function is part of the `dplyr` package and the `skim()` function is part of the `skimr` package. If you haven't installed those packages, you will need to do that before you can run the code below. We can install a package using the `install.pacakges()` function. Remember that we only install a package *once*, so you don't want to use this function in your R script that may get run multiple times. Instead you can run it in the console. For example, `install.packages("skimr")` would install the `skimr` package.)

```r
# First we will load the two packages we need

library(dplyr)
library(skimr)

# Then we will compare and contrast our four functions

summary(mtcars)
View(mtcars)
glimpse(mtcars)
skim(mtcars)
```

Reflection: Can you think of examples why someone may prefer one or the other in certain situations? Which one(s) do you think you might want to use to explore data in the future?

Answer: This will likely vary from person to person. Experiment with each function and use what's useful to you to get a better sense of the data structure and contents when you encounter new data sets.

The `$` operator to look at specific variables

Each of these functions looked at the entire `mtcars` data set, which was stored as a data frame. If we want to just look at a single variable, we can use the $ operator to tell R we are interested in a specific *variable* within a data frame. This will give us a vector of just that column. For example, I can run the code below to have R print a list of all of the horsepower values.

mtcars is a data frame object.

mtcars$hp is a vector (it pulls out just the hp column from the mtcars data frame).

```
mtcars$hp

##  [1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180 205 215 230
66  52
## [20]  65  97 150 150 245 175  66  91 113 264 175 335 109
```

Practice Questions

- Try writing code to get R to print all of the car weights instead. (Hint: Use the help file to figure out what the weight variable is called)

- Try writing code to calculate the minimum car miles per gallon. (Note: This is already printed when you use the `summary()` function above, but try to do it so that the only number produced is the minimum mpg.)

- (Bonus): Look up the `Loblolly` data set and find the oldest tree and the shortest tree in the data.

Adding comments to code with `#`

Notice that in some of the code above you see lines that start with #. Any line that starts with this symbol will be ignored by R when you run the code. This allows us to leave notes to ourselves (or to others that read our code) telling them what we're doing. It is a good habit (especially as you are learning R or working with collaborators) to frequently add comments to your code. This will help when you go back to review code you previously wrote.

Practice: At this point, you should have an R script called **notes01.R** that contains all your code from Notes 01. Go back and add comments to each section reminding you what those sections did. Because this code is specifically for *you*, you can write the notes to yourself. In the future, you will write code that is shared with others and will need to think about how to write comments to help others understand your code.

Revisiting the Learning Goals for Notes 01

- Understand the difference between R and RStudio
  - When you want to write code in this class, which program will you open? Why?

- Gain a basic understanding of the setup of RStudio and the purpose of R packages
  - Which two sections can we type code into (console, environment, scripts, viewer)?
  - Where can we look to see which objects we've created?
  - Where do we look to learn more about functions we want to use?
  - How is an R package like an app on a cell phone?

- Understand the basic elements of R code (objects / functions / operators)
  - How do we create objects in R?
  - What is the difference between a vector and a data frame?
  - How can functions be identified in code? What are some examples we've seen?
  - What do we mean when we refer to an argument?
  - Which operators have we learned about so far?

- Be able to use functions to explore vectors and data frames
  - There is a data frame built into R called `USJudgeRatings`. Use a function to learn how many observations and variables are in the data. Then find the average rating for judge diligence (variable called `DILG`).

- Be able to run pre-written code and make small changes to it
  - Take some of the code we've written and see if you can apply it in a new context (e.g. to a new data frame / object or to accomplish a slightly different task)
  - Thinking about activity 01, could you create an object called `die` that contains the numbers 1 through 6 and then write code that simulates rolling the die three times?