Week 1 Lab

2024-04-05

R Markdown

What is R Markdown?

R Markdown is a format for writing reports and presentations that include R Code. It can create files in html, pdf, Word, slideshows, and other formats.

Writing in R Markdown

Unlike R script files, writing on a plain line in R Markdown will appear as wording in the final file. Here are a few options for word stylization:

- *Italics*: Single asterisks on either side of the word/phrase: $*$word$*$
- **Bold**: Double asterisks on either side of the word/phrase: $**$ word $**$
- Headings: Start a line with a hashtag/pound sign. More hashtags = smaller heading: #heading

Heading

Heading

Heading

Heading

- Lists: Single asterisk, number, or other sign: $*, +, -, 1$

LaTeX in R Markdown

If you want even more options for writing, such as writing math equations, in R Markdown, it uses LaTeX

Many of the writing formatting operations can be performed with LaTeX

- Such as writing lists
- Or *italicizing* which can be done with \emphword $\Rightarrow$ *word*
- Or **bolding** which can be done with \textbfword $\Rightarrow$ **word**

Most importantly, it can be used for writing math equations. In line with \$e = m c ^ 2 \$$\Rightarrow e = mc^2$

Or as its own centered body of text: \$\sum_t=0^\infty \beta^t u(c_t) \$$\Rightarrow$

$$\sum_{t=0}^{\infty} \beta^t u(c_t)$$

Greek letters can also be displayed through LaTeX's math mode. Lowercase first letter yields lowercase greek letter, uppercase first letter yields upper case greek letter: \omega $\Rightarrow \omega$, \Omega $\Rightarrow \Omega$

Coding Terminology

When creating **chunks** of code, you have the following options, which are set to **T** by default.

- eval: (T/F) run/evaluate the code and display results
- echo: (T/F) display code and results (runs code either way)
- warning: (T/F) display warnings
- error: (T/F) display errors

Knitting

The **Knit** button generates the document (in the selected format) with the writing and embedded code chunks and their results.

Knitting to html is automatically built into R Markdown. To knit to pdf, you will need to download LaTeXonto your computer (which is a large file and takes time)

Example

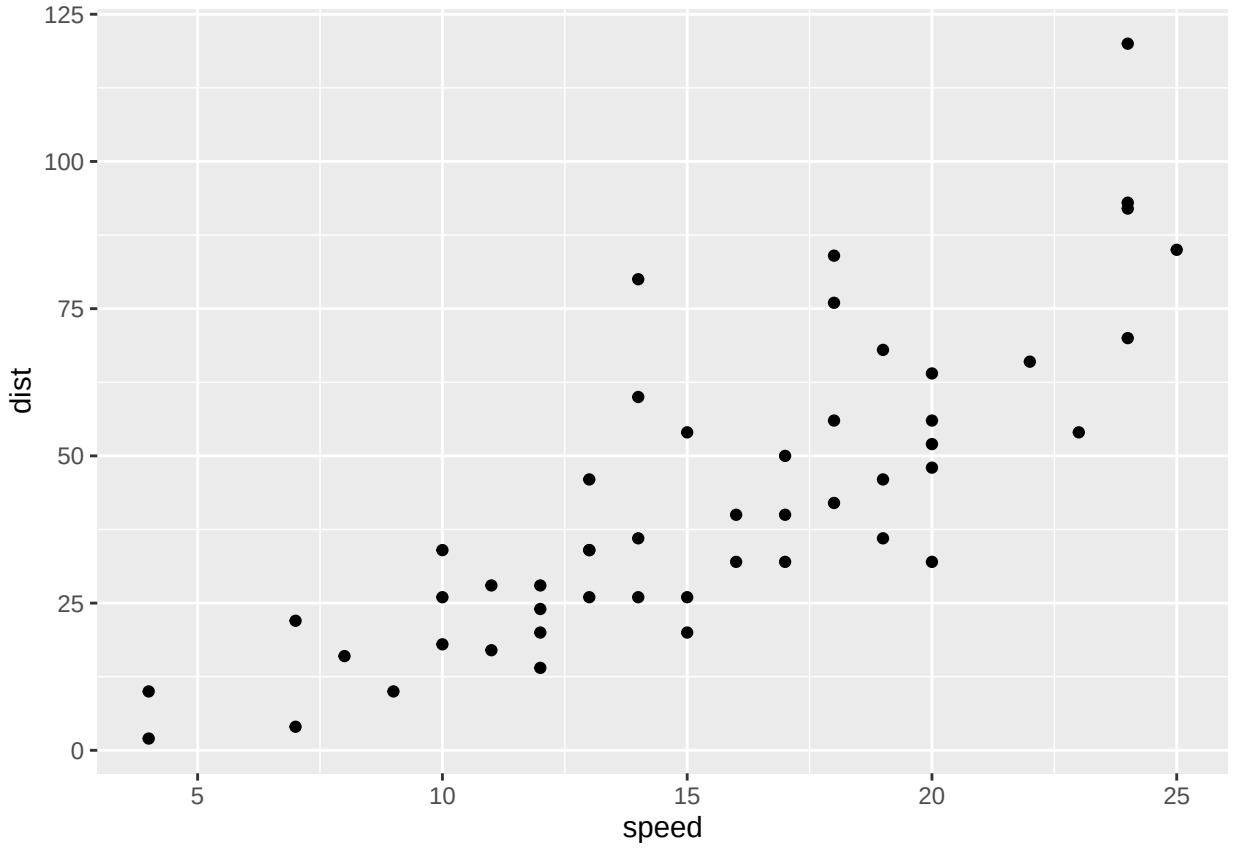
Let's do an example file and practice:

- formatting the file
- creating code chunks
- running code in R Markdown
- displaying plots
- knitting

```
# Load car data
car_data = cars

# And ggplot2
library(ggplot2)
```

```
ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()
```



Now plot with eval = F

```
ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()
```

No plot?

Review Functions

We will briefly review creating functions in R, since they will come in handy during this term, and some problem sets will specifically ask for functions that perform a certain task.

```
# function name = function(input) {
#                 operations on input = output
#                 return(output)
# }
```

Basics:

Example: Create a function of this function:

$$f(x) = \frac{x-5}{4}$$

```
# Create the function
f_x = function(x) {
```

```
  ans = (1/4)*(x - 5)

  return(ans)
}


# Try it on x = 10
f_x(10)
```

```
## [1] 1.25
```

Now suppose we want this new function:

$$g(x, y) = f(x) - 3y$$

You could do this by plugging in $f(x) = \frac{x-5}{4}$ is, OR a faster way is to put our previous function f_x into this new function!

```
# Create function
g_xy = function(x, y) { # note: two inputs

  ans = f_x(x) - 3*y

  return(ans)
}

# Try with x = 10, y = 2
g_xy(10, 2)
```

```
## [1] -4.75
```

For Loops

We briefly discussed *for* loops last term, but didn't use them. You will use them this term, so let's review.

```
# for (i in start:end) {
#   operation performed on each value of i
# }
```

Basics That's rather abstract so it's easier to start with an example. Apply our function f_x to every integer from 1 to 20 and print the answer.

```
for (i in 1:20){
  print(f_x(i))
}
```

```
## [1] -1
## [1] -0.75
## [1] -0.5
## [1] -0.25
## [1] 0
## [1] 0.25
## [1] 0.5
## [1] 0.75
## [1] 1
```

```
## [1] 1.25
## [1] 1.5
## [1] 1.75
## [1] 2
## [1] 2.25
## [1] 2.5
## [1] 2.75
## [1] 3
## [1] 3.25
## [1] 3.5
## [1] 3.75
```

Handy functions for *for* loops We typically don't want to just perform operations on integers. We can get any linear sequence of numbers we want using the *seq*() function

Suppose we want every hundredth place from 0 to 1: 0, 0.01, 0.02, . . . , 0.98, 0.99, 1

```
# Basically
##  seq(from = start, to = end, by = step)

seq(from = 0, to = 1, by = 0.01)
```

```
##   [1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12 0.13 0.14
##  [16] 0.15 0.16 0.17 0.18 0.19 0.20 0.21 0.22 0.23 0.24 0.25 0.26 0.27 0.28 0.29
##  [31] 0.30 0.31 0.32 0.33 0.34 0.35 0.36 0.37 0.38 0.39 0.40 0.41 0.42 0.43 0.44
##  [46] 0.45 0.46 0.47 0.48 0.49 0.50 0.51 0.52 0.53 0.54 0.55 0.56 0.57 0.58 0.59
##  [61] 0.60 0.61 0.62 0.63 0.64 0.65 0.66 0.67 0.68 0.69 0.70 0.71 0.72 0.73 0.74
##  [76] 0.75 0.76 0.77 0.78 0.79 0.80 0.81 0.82 0.83 0.84 0.85 0.86 0.87 0.88 0.89
##  [91] 0.90 0.91 0.92 0.93 0.94 0.95 0.96 0.97 0.98 0.99 1.00
```

We also often want a vector to put our output into. The function $rep(value, n)$ repeats whatever value you give it, n times

```
# "empty" vector
rep(NA, 10)
```

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

In a *for* loop, we can find the i^{th} value of a vector by using square brackets

For example: apply f_x to each value in the sequence above using a *for* loop, and store the answers into a new vector

```
seq_1 = seq(from = 0, to = 1, by = 0.01)

seq_2 = rep(NA, 101)

for (i in 1:101){
  seq_2[i] = f_x(seq_1[i])
}

seq_2
```

```
##   [1] -1.2500 -1.2475 -1.2450 -1.2425 -1.2400 -1.2375 -1.2350 -1.2325 -1.2300
##  [10] -1.2275 -1.2250 -1.2225 -1.2200 -1.2175 -1.2150 -1.2125 -1.2100 -1.2075
##  [19] -1.2050 -1.2025 -1.2000 -1.1975 -1.1950 -1.1925 -1.1900 -1.1875 -1.1850
##  [28] -1.1825 -1.1800 -1.1775 -1.1750 -1.1725 -1.1700 -1.1675 -1.1650 -1.1625
##  [37] -1.1600 -1.1575 -1.1550 -1.1525 -1.1500 -1.1475 -1.1450 -1.1425 -1.1400
```

```
##  [46] -1.1375 -1.1350 -1.1325 -1.1300 -1.1275 -1.1250 -1.1225 -1.1200 -1.1175
##  [55] -1.1150 -1.1125 -1.1100 -1.1075 -1.1050 -1.1025 -1.1000 -1.0975 -1.0950
##  [64] -1.0925 -1.0900 -1.0875 -1.0850 -1.0825 -1.0800 -1.0775 -1.0750 -1.0725
##  [73] -1.0700 -1.0675 -1.0650 -1.0625 -1.0600 -1.0575 -1.0550 -1.0525 -1.0500
##  [82] -1.0475 -1.0450 -1.0425 -1.0400 -1.0375 -1.0350 -1.0325 -1.0300 -1.0275
##  [91] -1.0250 -1.0225 -1.0200 -1.0175 -1.0150 -1.0125 -1.0100 -1.0075 -1.0050
## [100] -1.0025 -1.0000
```

Combine functions and for loops Let's create a function that creates the first N numbers of the Fibonacci sequence, defined as:

$$F_1 = 0,\ F_2 = 1,\ F_n = F_{n-1} + F_{n-2}$$

```
# function
fibonacci = function(N) {

  data = c(0, 1, rep(NA, N - 2))

  for (i in 3:N) {
    data[i] = data[i - 1] + data[i - 2]
  }

  return(data)
}

# suppose we want the first 20 values (including 0 and 1)
fibonacci(20)
```

```
##  [1]    0    1    1    2    3    5    8   13   21   34   55   89  144  233  377
## [16]  610  987 1597 2584 4181
```