Day 1 session notes

Tim Riffe

7/26/2021

Getting started

You can use R as a little basic calculator:

```
# This is R!!!!!
1 + 1
```

```
## [1] 2
```

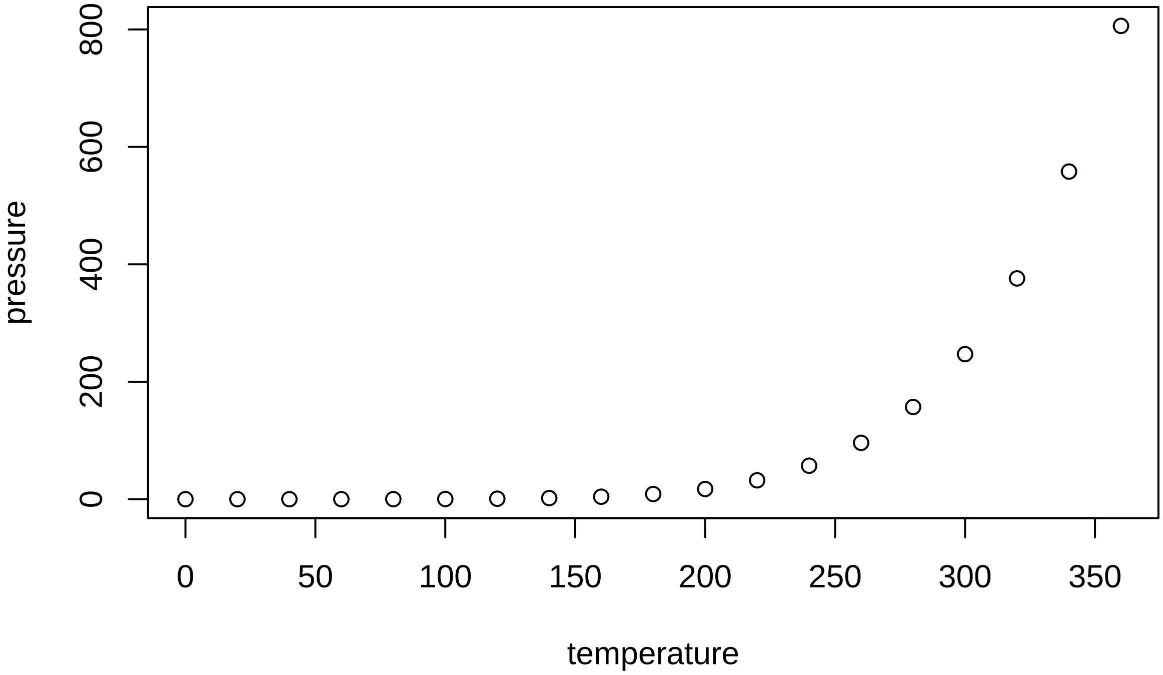
The short cut to create a code chunk is:

press these three keys on your keyboard: Ctrl + Alt + i

That created a blank code chunk! So we can stick code in it that gets executed. Which is the only thing we're going to do in this course!

Including Plots

You can also embed plots, for example:



This space here is just for notes. What you write here is your business. Your note-taking space is for your own benefit, and you should be writing here ideally in your work language or native language.

```
# install.packages("DemoDecomp")
install.packages("readr")
install.packages("readxl")
install.packages("lubridate")
install.packages("scales")
install.packages("colorspace")
install.packages("tidyverse")
```

Installing packages gets them into your R libraries on your computer, but it does not make them directly available to you in your R session. To load a package, use `library()` and type the package name inside it with no quotes.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.3     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Back to basics. Here are the operators, there are more!

```
1 + 1
```

```
## [1] 2
```

```
1 / 2
```

```
## [1] 0.5
```

```
1 * 2
```

```
## [1] 2
```

```
1 - 2
```

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

```
5^2
```

```
## [1] 25
```

```
# matrix multiplication is different
# %*%
```

Create an object. Here we created a vector using `c()`. It has three elements, so we call it a vector of length 3. We use the `length()` function to ask how many elements it has.

```
x <- c(2, 5, 7)
length(x)
```

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

How can we look at data? (`View()`), or we can query metadata about an object using `str()` or `glimpse()`.

```
View(x)
str(x)   # metadata
```

```
##  num [1:3] 2 5 7
```

```r
glimpse(x)
```

```
##  num [1:3] 2 5 7
```

Using arithmetic functions on vectors.

```r
2 + 5 + 7
```

```
## [1] 14
```

```r
sum(x)
```

```
## [1] 14
```

```r
cumsum(x)
```

```
## [1]  2  7 14
```

```r
mean(x)
```

```
## [1] 4.666667
```

```r
sd(x)
```

```
## [1] 2.516611
```

```r
var(x)
```

```
## [1] 6.333333
```

```r
exp(x)
```

```
## [1]    7.389056  148.413159 1096.633158
```

```r
log(x)
```

```
## [1] 0.6931472 1.6094379 1.9459101
```

How to learn how functions work and what they do?

```r
?sum
y <- c(2,5,7,NA)
sum(y, na.rm = FALSE)
```

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

Here are the examples from `?sum` (the help file) pasted directly into a new R chunk.

```r
## Pass a vector to sum, and it will add the elements together.
sum(1:5)
```

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

```r
## Pass several numbers to sum, and it also adds the elements.
sum(1, 2, 3, 4, 5)
```

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

```r
## In fact, you can pass vectors into several arguments, and everything gets added.
sum(1:2, 3:5)
```

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

```r
## If there are missing values, the sum is unknown, i.e., also missing, ....
sum(1:5, NA)
```

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

```r
## ... unless  we exclude missing values explicitly:
sum(1:5, NA, na.rm = TRUE)
```

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

Lesson is: getting a help file is as easy as typing `?` and the function name. You can also search in the help tab in R Studio.

```r
x
```

```
## [1] 2 5 7
```

```r
is.vector(x)
```

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

```r
class(x)
```

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

```r
is.integer(x)
```

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

```r
x <- as.integer(x)
is.integer(x)
```

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

```r
length(x)
```

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

```r
dim(x)
```

```
## NULL
```

Cleaning up, use `rm()` to remove objects from your *workspace*

```r
rm(y)
```

There are also character vectors (strings):

```r
b <- c("A","1","c","b")
d <- c(TRUE, TRUE, FALSE, FALSE)
```

What about tabular data: rectangular data, or data organized in rows and columns, like in a spreadsheet:

```r
A <- data.frame(b, d)
A
```

```
##   b     d
## 1 A  TRUE
## 2 1  TRUE
## 3 c FALSE
## 4 b FALSE
```

A `data.frame()` has a dimension! `dim()` tells us rows and then columns.

```r
dim(A)
```

```
## [1] 4 2
```

```
nrow(A) # ncol()
```

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

You can add rows to a `data.frame` using `rbind()`, `bind_rows()`. Rules for adding rows are that you need the same number and structure of columns in both pieces of data.

```
B <- data.frame(b = c("adth","Tim"),
                d = c(FALSE, FALSE))
# This is identical. No need to create intermediate objects.
# becuase what we're doing is small
# b2 <- c("adth", "Tim")
# d2 <- c(FALSE, FALSE)
# data.frame(b = b2, d = d2)

D <- rbind(A, B)
D <- bind_rows(A, B)
```

You can add a column to a `data.frame`:

```
# old fashioned way:
D$z <- runif(6)

# Delete the column:
D$z <- NULL

# the tidy way:
z <- runif(6)
E <- bind_cols(D,z = z)

# tidy column deletion
select(E, -z)
```

```
##      b     d
## 1    A  TRUE
## 2    1  TRUE
## 3    c FALSE
## 4    b FALSE
## 5 adth FALSE
## 6  Tim FALSE
```