Data Wrangling

re-organising data for analysis

Vector Refresher and Shortcuts

Do it yourself: vector shortcuts:

```
> c(3,4,5,6,7,8) ↵
# the c() function concatenates anything (including other vectors)
> 3:8 ↵
# the `:` is a shortcut to give you all the integers in between
> 1:5*2↵
# multiplying each number by 2 gives you different spacing
> seq(2,10, by=2) ↵
# the equivalent using seq() function
> rep(3:8, times=2)↵
# the rep() function repeats a vector `times` times
> rep(3:8, each=2)↵
# or you can repeat the elements (try both if you want)
```

Refresher: Types of Object

Do it yourself: Let's make some objects:

```
> NumV <- 1:20 ↵
> FacV <- factor(rep(letters[1:5], each=4)) ↵
> SecondV <- rbinom(n=16, size=100, prob=.5) ↵

> M <- matrix(SecondV, nrow=4, ncol=4) ↵
> L <- list(a=1, b=1:3, c=10:100) ↵
```

- Now take a look at these objects. What types of objects are they? Use `str(object)` to help you.
- Not sure what `letters()` and `rbinom()` are doing? Take a look at those parts of the code.
- Factors are important in analysis.

Subsetting Objects

Do it yourself: Take subsets of letters

```
> letters[c(2:4,26)] ↲
# can you make sense of this?
```

Now take subsets of the matrix we just made:

```
> M[2,3] ↲
> M[2,] ↲
> M[,3] ↲
> M[2,2:4] ↲
# can you make sense of these?
```

This also works with data frames and their tidyverse cousins: `tibbles`.

Base Aggregation Functions (*apply)

Function	Mnenomic	What it does	Example (Try them!)
`apply()`	--	apply "function" to rows OR columns of a matrix	`apply(M, 1, min)` `apply(M, 2, max)`
`lapply()`	list	apply function to each element in a list and emit list	`lapply(L, FUN = length)` `lapply(L, FUN = sum)`
`sapply()`	simplifying	apply function to each element in list and output list	`sapply(L, FUN = length)` `sapply(L, FUN = sum)`
`tapply()`	per type	apply function to subsets of a vector (defined by other vector)	`tapply(NumV, FacV, sum)`
`by()`	by column	apply to subsets of columns in a data frame (defined by vector)	`by(warpbreaks[,1:2], tension, summary)`
`aggregate()`	--	equivalent to by() but outputs to a new data frame	`aggregate(breaks, list(wool, tension), summary)`

Attaching a Dataset

Do it yourself: Many datasets are available:

```
> library(help = "datasets") ↵
```

Let’s get one of them and find out about it:

```
> data(warpbreaks)↵
> ?warpbreaks ↵
> View(warpbreaks)
```

and then attach the data to memory:

```
> attach(warpbreaks)↵
```

This means that variable names will be remembered.

Advanced Aggregation

Function	Mnenomic	What it does	Example (Try them!)
`apply()`	--	apply "function" to rows OR columns of a matrix	`apply(M, 1, min)` `apply(M, 2, max)`
`lapply()`	list	apply function to each element in a list and emit list	`lapply(L, FUN = length)` `lapply(L, FUN = sum)`
`sapply()`	simplifying	apply function to each element in list and output list	`sapply(L, FUN = length)` `sapply(L, FUN = sum)`
`tapply()`	per type	apply function to subsets of a vector (defined by other vector)	`tapply(NumV, FacV, sum)`
`by()`	by column	apply to subsets of columns in a data frame (defined by vector)	**`by(warpbreaks[,1:2], tension, summary)`**
`aggregate()`	--	equivalent to by() but outputs to a new data frame	**`aggregate(breaks, list(wool, tension), summary)`**

Detaching a Dataset

Always do it !

```
> detach(warpbreaks)↵
```

This means that variable names will be forgotten.

How do we refer to columns is detached objects?

The old-fashioned way:

```
> warpbreaks[,3] ↵
```

By column header using $ notation (very useful!):

```
> warpbreaks$tension ↵
```