Looping

Naomi Tague

January, 2023

Comments/Questions

looping.RMD

Steps for running function over multiple inputs

1. design a data structure to store results: sometimes this is automatic but not always
2. generate the data (or read it in)
3. find ways to repeat things efficiently

- *purr* family of function in R
- more generally looping (*for*, *while*, …others)

A bit more on Looping

Loops can be "nested" with one loop inside the other

"Nesting" means loops inside loops

useful for "doing" something to each row and each column

FOR (i in 1:nrow) {

FOR (j in 1: ncol)

…

}}

In nested loops, the inner loop (last **FOR** in the sequences, gets done first)

So the loop above will all columns for the first row and then go to the next row

Example

Calculate NPV for

- a range of different interest rates and
- a range of damages
- that may be incurred 10 years in the future

Steps

- define inputs (interest rates, damages)
- define output (NPV)
- write the function
- create a data structure to store results where we vary both interest rates and damages
- use nested for loops to fill in the data structure

Try it first…

Rcode

```
# write a function to compute npv
source("../R/compute_NPV.R")
compute_NPV
```

```
## function (value, time, discount)
## {
##     result = value/(1 + discount)^time
##     return(result)
## }
```

```
compute_NPV(20, discount=0.01, time=20)
```

```
## [1] 16.39089
```

```
#generate some input
damages = c(25,33,91,24)
damages
```

```
## [1] 25 33 91 24
```

```
# sensitivity to discount rate
discount_rates = seq(from=0.01, to=0.04, by=0.005)
discount_rates
```

```
## [1] 0.010 0.015 0.020 0.025 0.030 0.035 0.040
```

```
yr=10

# note that the "simple" application doesn't work - why?
compute_NPV(20, discount=discount_rates, value=damages)
```

```
## Warning in value/(1 + discount)^time: longer object length is not a multiple of
## shorter object length
```

```
## [1] 20.48861 24.50152 61.24039 14.64650 13.84189 16.58467 41.53121
```

```
# compute some npv's for different discount rates for each damage
# first generate a dataframe to store results - we have both damages and discount rates that are changing - so we need rows and
        columns
npvs = data.frame(matrix(nrow=length(damages), ncol=length(discount_rates)))

# here we have a 2-dimensional array - rows and columns
View(npvs)

# now use a nested for loop to populate
 for (i in 1:length(damages)) {
        for (j in 1:length(discount_rates)) {
      npvs[i,j]= compute_NPV(value=damages[i],      discount=discount_rates[j],time=yr )

        }
 }
 npvs
```

```
##          X1       X2       X3       X4       X5       X6       X7
## 1 22.63217 21.54168 20.50871 19.52996 18.60235 17.72297 16.88910
## 2 29.87447 28.43502 27.07149 25.77955 24.55510 23.39432 22.29362
## 3 82.38111 78.41172 74.65170 71.08905 67.71255 64.51161 61.47634
## 4 21.72689 20.68001 19.68836 18.74876 17.85825 17.01405 16.21354
```

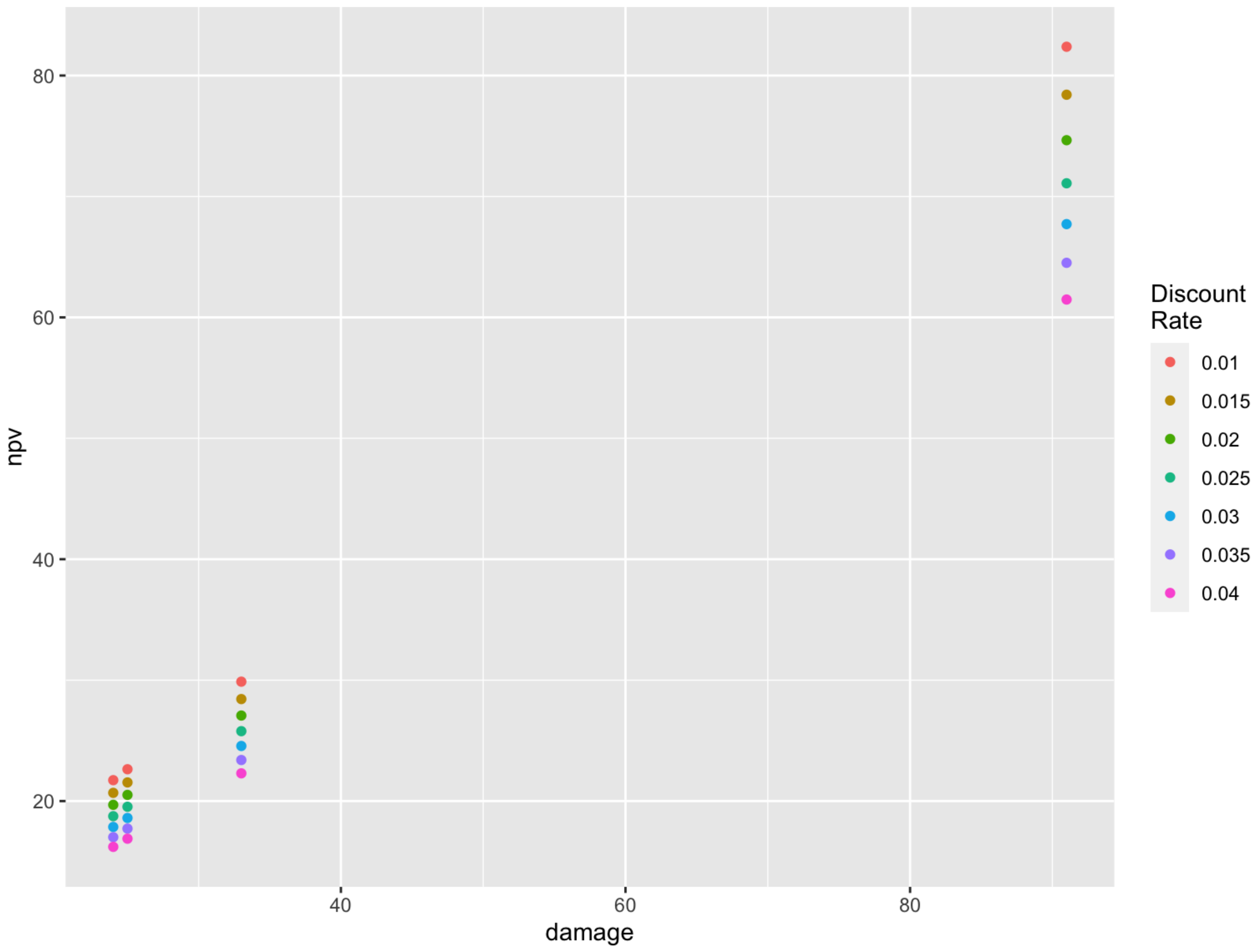
```
 #some data wrangling to make it pretty
colnames(npvs)=discount_rates
rownames(npvs)=damages
 npvs
```

```
##        0.01    0.015     0.02    0.025     0.03    0.035     0.04
## 25 22.63217 21.54168 20.50871 19.52996 18.60235 17.72297 16.88910
## 33 29.87447 28.43502 27.07149 25.77955 24.55510 23.39432 22.29362
## 91 82.38111 78.41172 74.65170 71.08905 67.71255 64.51161 61.47634
## 24 21.72689 20.68001 19.68836 18.74876 17.85825 17.01405 16.21354
```

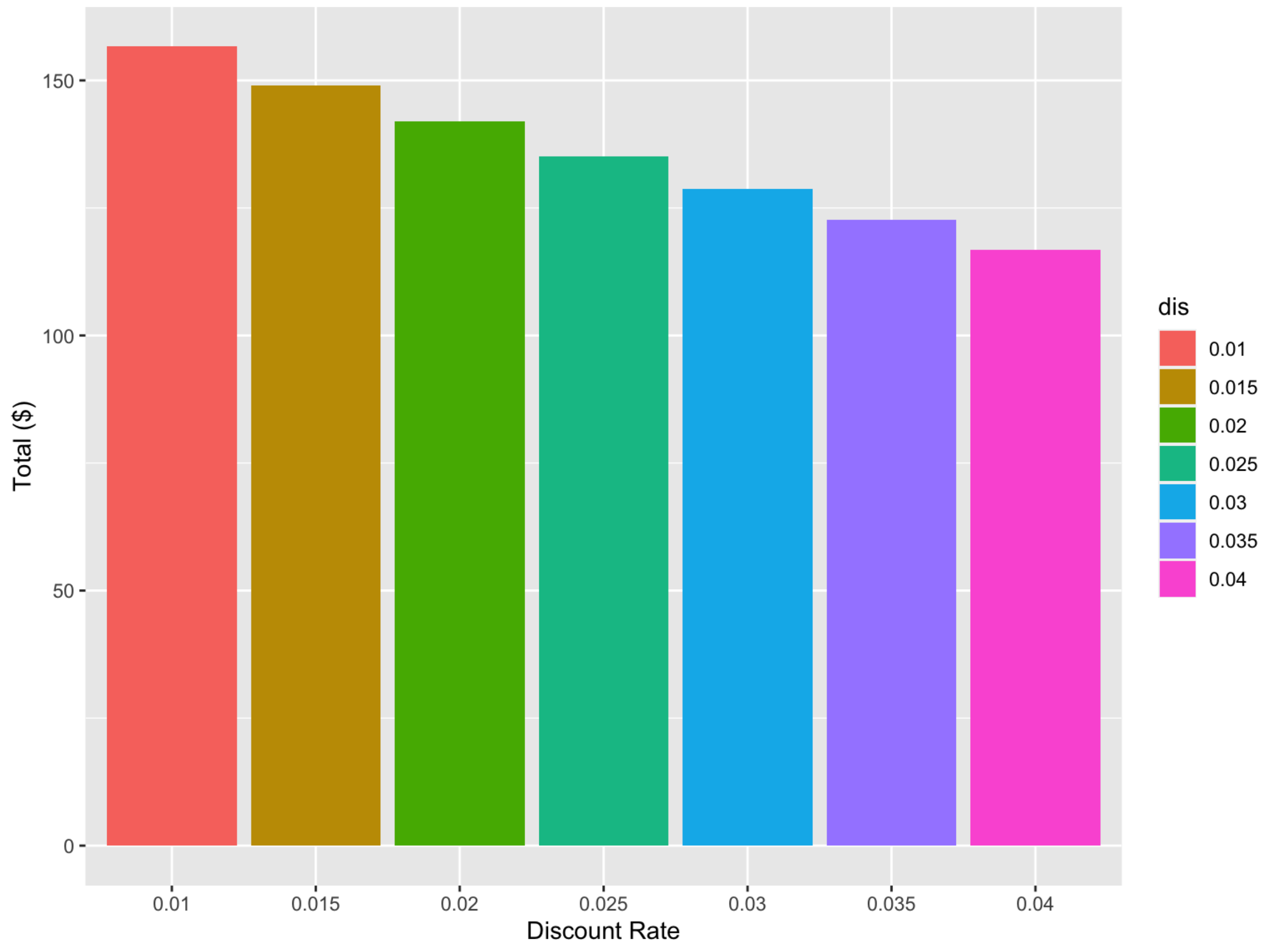
```
# how do I plot this with ggplot - add a column for original value and then rearrange
 npvs$damage = damages
 npvsg =npvs %>% pivot_longer(!damage,names_to="dis", values_to="npv")
 head(npvsg)
```

```
## # A tibble: 6 × 3
##   damage dis     npv
##    <dbl> <chr> <dbl>
## 1     25 0.01   22.6
## 2     25 0.015  21.5
## 3     25 0.02   20.5
## 4     25 0.025  19.5
## 5     25 0.03   18.6
## 6     25 0.035  17.7
```

```
 ggplot(npvsg, aes(x=damage, y=npv, col=dis))+geom_point()+labs(col="Discount\nRate")
```



```
# how about summing all the damages
npv.total =npvsg %>% group_by(dis) %>% summarize(t=sum(npv))
ggplot(npv.total, aes(dis,t, fill=dis))+geom_col() + labs(x="Discount Rate", y="Total ($)")
```



150
100
50
0
Total ($)
0.01
0.015
0.02
0.025
0.03
0.035
0.04
Discount Rate
dis
0.01
0.015
0.02
0.025
0.03
0.035
0.04

An alternative method

An alternative (R specific) is to use functions from the **purr** package

Specifically *pmap* - which applies a function over multiple inputs

```
 # we could also do the repetition with map_dfc from the purr family of functions

# use damages and discount rates from above

discount_rates
```

```
## [1] 0.010 0.015 0.020 0.025 0.030 0.035 0.040
```

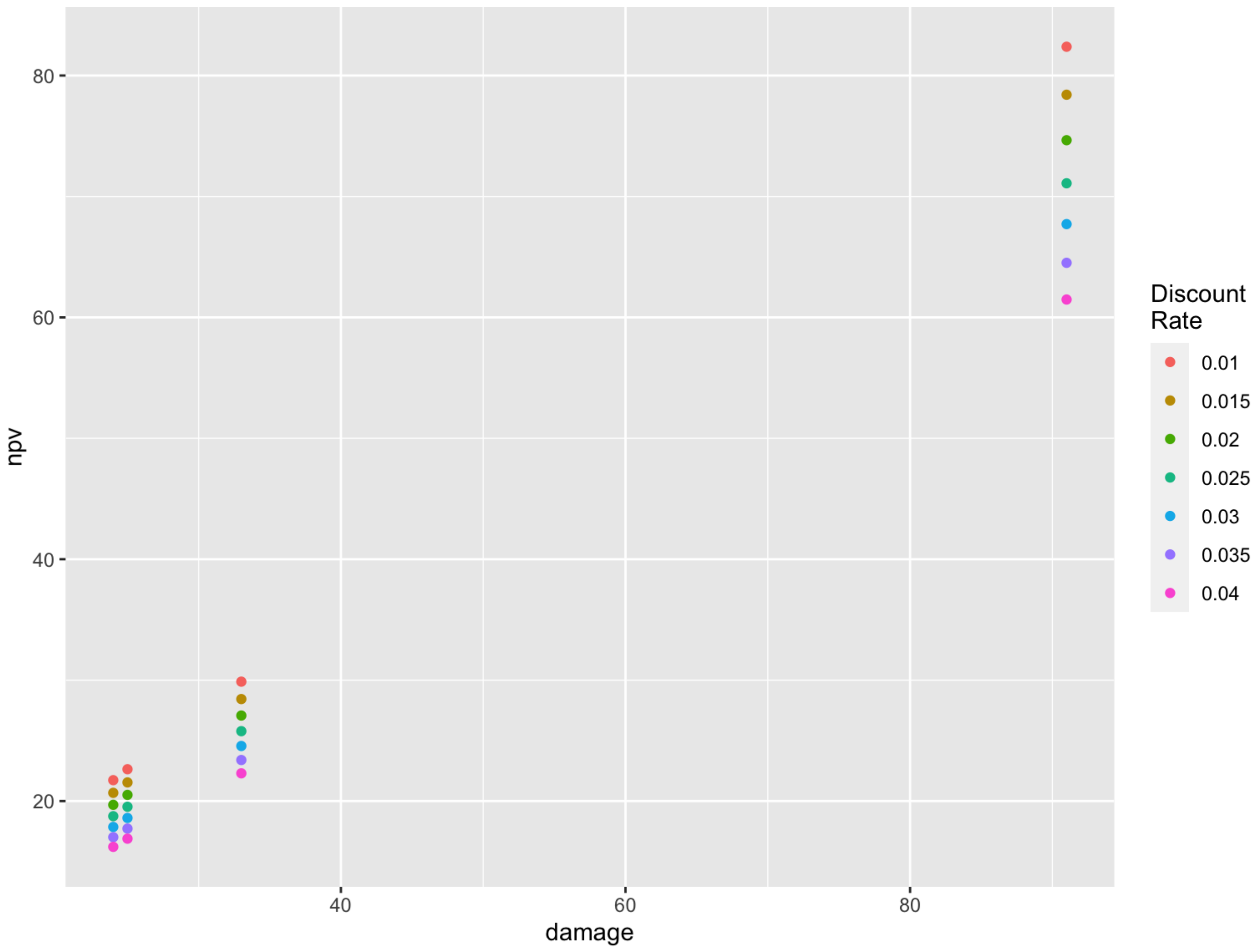
```
damages
```

```
## [1] 25 33 91 24
```

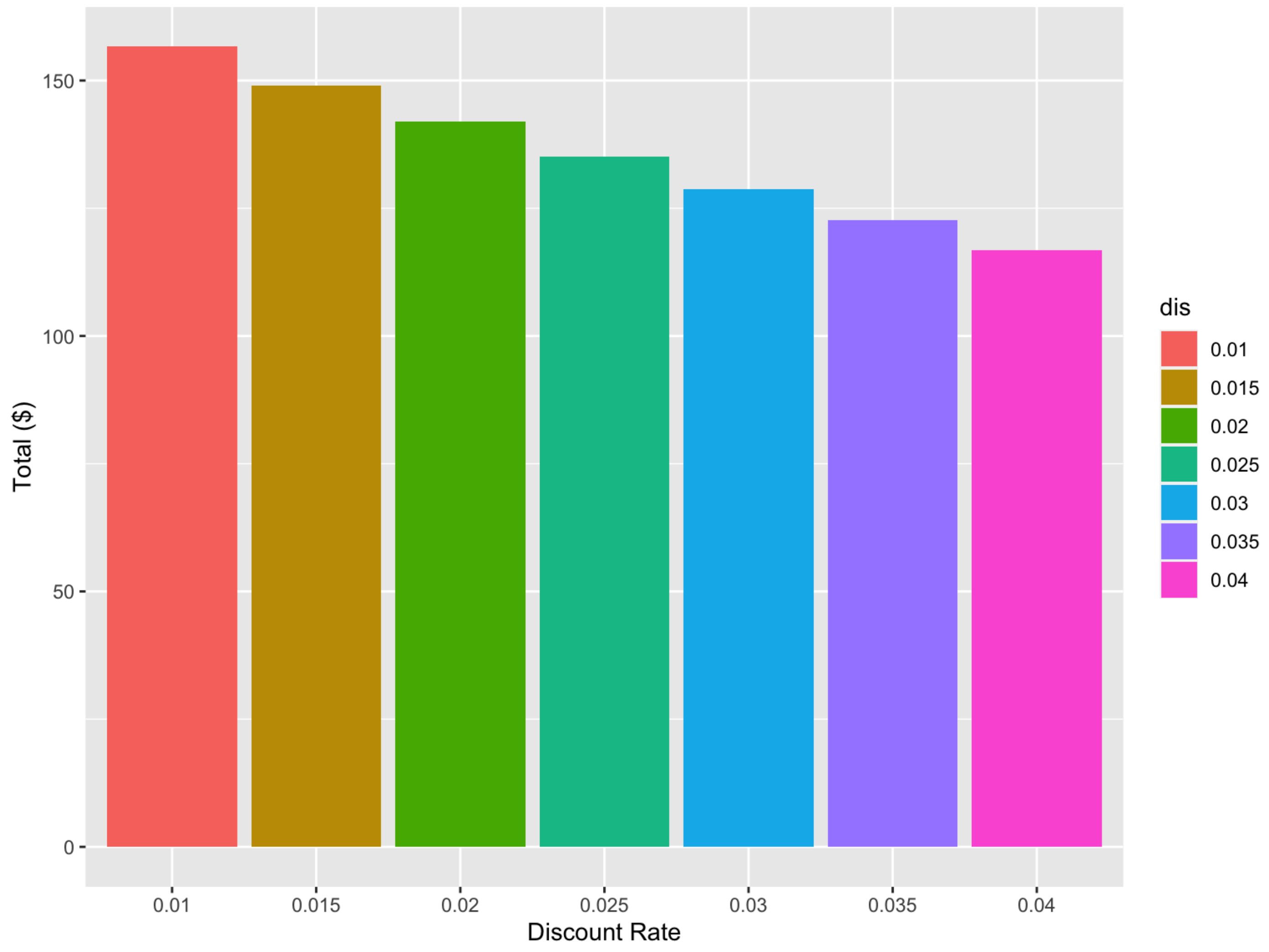
```
# with purr family, pmap runs the function for each value and returns as a data frame

npvs = pmap(list(discount_rates),compute_NPV, time=10, value=damages)
# turn into a data frame
npvs = as.data.frame(npvs, col.names=discount_rates)
colnames(npvs) = discount_rates
npvs$damage = damages
# clean up and plot
 npvsg =npvs %>% pivot_longer(!damage,names_to="dis", values_to="npv")
 head(npvsg)
```

```
## # A tibble: 6 × 3
##   damage dis     npv
##    <dbl> <chr> <dbl>
## 1     25 0.01   22.6
## 2     25 0.015  21.5
## 3     25 0.02   20.5
## 4     25 0.025  19.5
## 5     25 0.03   18.6
## 6     25 0.035  17.7
```

```
 ggplot(npvsg, aes(x=damage, y=npv, col=dis))+geom_point()+labs(col="Discount\nRate")
```



```
# how about summing all the damages
npv.total =npvsg %>% group_by(dis) %>% summarize(t=sum(npv))
ggplot(npv.total, aes(dis,t, fill=dis))+geom_col() + labs(x="Discount Rate", y="Total ($)")
```



# Some other types of loops

- while useful for repeating until a condition is met

Example if a metal toxin in a lake increases by 1% per year, how many years will it take for the metal level to be greater than 30 units, if toxin is current at 5 units

```
# accumulate pollutant until a threshold - how many years does it take

# initial conditions
yr=1
pollutant_level = 5

# loop
while ((pollutant_level < 30) &&(yr < 1000))  {
  # increase pollutant
pollutant_level = pollutant_level + 0.01* pollutant_level
# keep track of time
yr = yr + 1
}

pollutant_level
```

```
## [1] 30.2788
```

```
yr
```

```
## [1] 182
```

```
# while loop dangers
# what if it doesn't end....

# see below - I left it commented out -because it will run for ever :)
# you can use controlC to stop

#while ((pollutant_level < 30))  {
  # increase pollutant
#pollutant_level = pollutant_level - 0.01* pollutant_level
#}
```

A bit more on nesting loops

we could even have multiple nesting (think about doings something to a 3 dimensional matrix time and space (x,y)

useful for "doings" something to each row and each column

```
**FOR** (x in 1:nrow) {

    **FOR** (y in 1: ncol)

      **FOR**  (t in 1: ntime)


            ...

    }
  }
}
```

To illustrate

```
# lets ocean be a 3 dimensional array - space (x,y) and time
# lets say is a 2 by 3 spatial maps (2 units across, 3 units down), and 2 time steps
ocean = array(dim=c(2,3,2))

ocean
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
```

```
# lets make up data for our ocean

ocean[1,1,1] = runif(n=1, min=0, max=1)

# second time step
ocean[1,1,2] = runif(n=1, min=0, max=1)

# next column, first time step
ocean[2,1,1] = runif(n=1, min=0, max=1)

# for loops let us do this quickly
for (x in 1:nrow(ocean)) {
  for (y in 1:ncol(ocean)) {
    for (t in 1:dim(ocean)[3]) {
      ocean[x,y, t] = runif(n=1, min=0, max=1)
    }}}
```

Assignment 3

For this assignment you will write a new Rmarkdown document based on the analysis of power required for different *possible_cars* from looping.RMD; You will

- Add an additional super light car with mass 5,000 kg and a surface area 10m2
- Compute the mean and maximum power for Cars A, B, C and for your new car, assuming that average highway speed is 80 km/hr with a standard deviation of 10km/hr; Use 2 different methods to compute the mean and maximum power for each car type; a) **FOR** loop and b) the **pmap** function from **purr**
- Create two boxplots (one for each method (**FOR** and **pmap**)) that show the range of power consumption (across sampled speeds) for each car type.

Put the Rmarkdown in your assignment github and submit a link on Gauchospace for Assignment 3 when its ready to be graded

Grading Rubric

1. Correct R code for adding the super light car (5pts)
2. Values for mean and maximum power for all 4 cars are in expected ranges (remember you are sampling so values will differ slightly) (5pts)
3. Used both methods (**FOR** and **pmap**) correctly (4pts)
4. Boxplots (6pts)
1. correctly show range of power consumption for each car type for the two methdos
2. good presentation (e.g axis labels, colors)
5. Good coding practice (e.g Adequate documentation throughout) (5pts)

Total out of 25pts

To Review for next class

- DataTypes.Rmd