Using functions with Data

Naomi Tague

January, 2022

Questions

- sampling_example.Rmd
- error_checking.Rmd

Steps for running your function over multiple inputs

1. design a data structure to store results: sometimes this is automatic but not always
2. generate the input data
3. apply to the function

Consider our function that estimates the power required to keep a car moving at a given speed

Code

```
#' Power Required by Speed
#'
#' This function determines the power required to keep a vehicle moving at
#' a given speed
#' @param cdrag coefficient due to drag default=0.3
#' @param crolling coefficient due to rolling/friction default=0.015
#' @param v vehicle speed (m/2)
#' @param m vehicle mass (kg)
#' @param A area of front of vehicle (m2)
#' @param g acceleration due to gravity (m/s) default=9.8
#' @param pair (kg/m3) default =1.2
#' @return power (W)

autopower = function(V, m, A, cdrag=0.3, crolling=0.015,pair=1.2,g=9.8) {
  P = crolling*m*g*V + 1/2*A*pair*cdrag*V**3
  return(P)
}
```

I've also stored the autopower.R function as a file = using the recommended naming convention

autopower.R

Generating data for the function and iterating over that data

Two parts

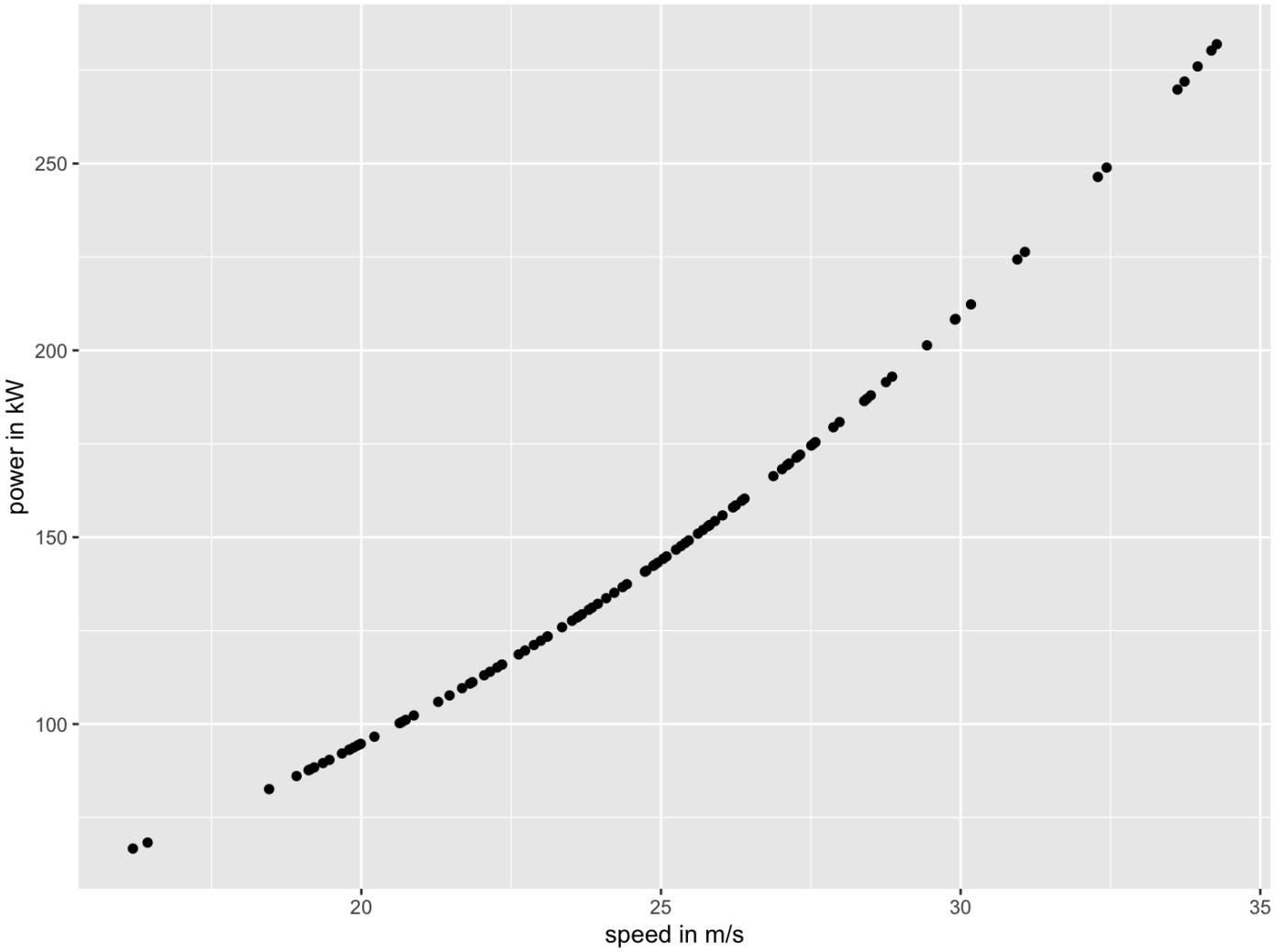
- Generate a data structure to store your results
- Repeat application of your function over the data

Example use: Imagine we want to see how much power is generated given a scenario where we know the mean and standard deviation of vehicle speeds

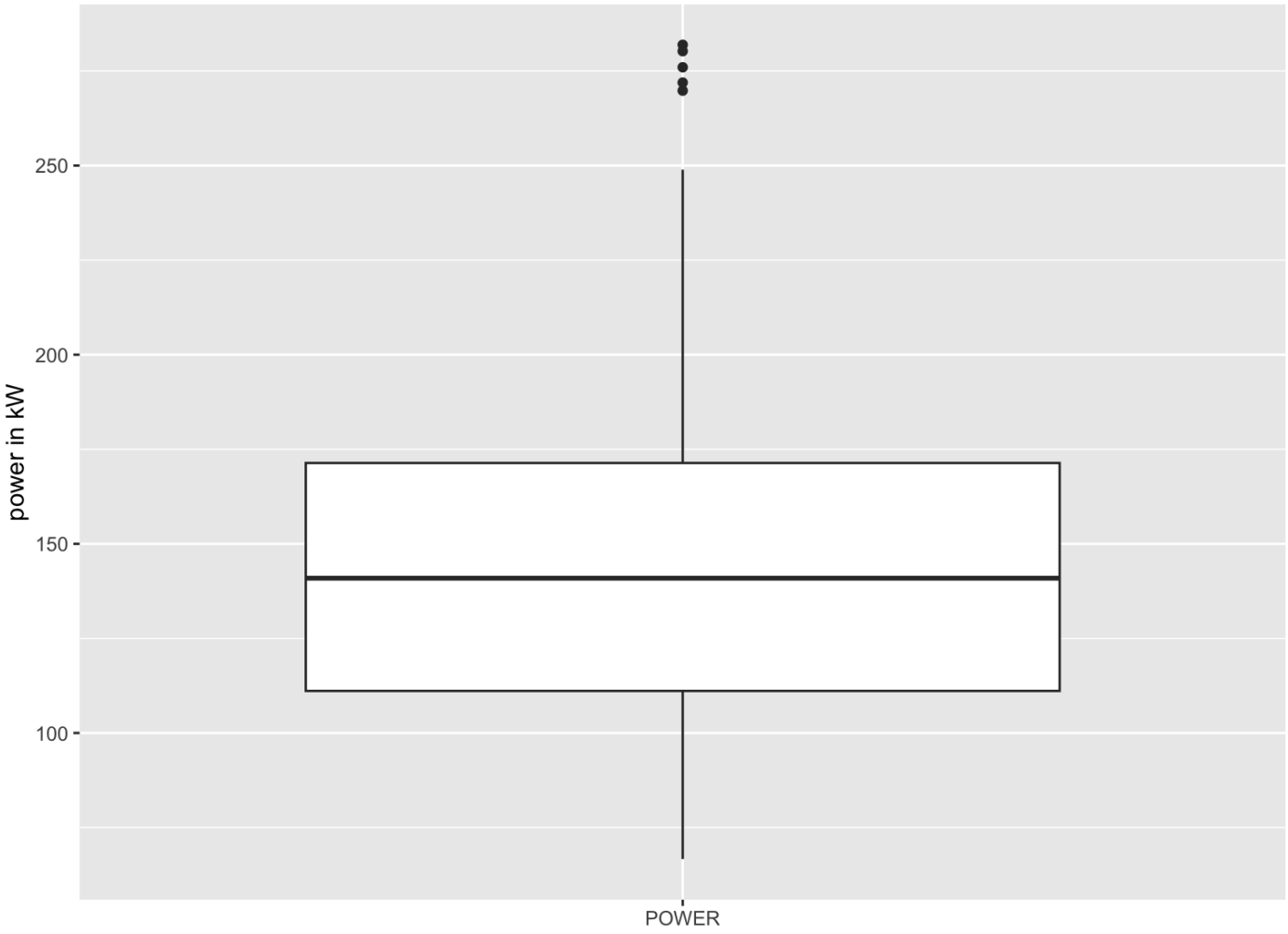
```
source("../R/autopower.R")

# generate sample speeds from a distribution
nsample = 100
speeds = rnorm(mean=25, sd=4, n=nsample)

# Step 1  create data frame to store results
# how many simulations, what do you want to keep

#create a dataframe that has rows for each model run
# columns for height, flowrate and power estimate
results = data.frame(speed=speeds, power=NA)

head(results)
```

```
##      speed power
## 1 20.63874    NA
## 2 22.26732    NA
## 3 19.79542    NA
## 4 19.14711    NA
## 5 24.73116    NA
## 6 22.73167    NA
```

```
# if you only have one input and everything else is the same, R is smart enough
# to create a set of outputs
results$power = autopower(V=speeds, A=25, m=20000)

# ALWAYS look at your results to make sure it make sense
ggplot(results, aes(speed, power/1000))+geom_point()+labs(x="speed in m/s", y="power in kW")
```




```
ggplot(results, aes(x="POWER", y=power/1000))+geom_boxplot()+labs(y="power in kW", x="")
```


250
200
150
100
power in kW
POWER

Looping (or repetition)

Looping - or repeating applying your function over multiple inputs is not always automatic

- more complex scenarios - number of different values is not the same for all inputs (e.g - you have 3 different speeds for 5 different cars (mass/Area))
- other programming languages

Lets start with the basics of how you do repetition in programming in general often called *looping*

For loops

for statement

- defines a loop using a counter that is incremented each time you go through the loop
- the counter - could be any variable (we often use *i*) but you could use what ever variable you want
- the loop is the commands between the { and } after the *for* keyword
- these commands are repeated each time you go through the loop (following the definition)

```
# simple example
# a = 0+1+2+3+4+5
a=0
for (i in 1:5) {
 a = a+i
}
a
```

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

```
#if you want to keep track of a for each iteration
# start with a data structure to hold the results - there will be 5 iterations

a = rep(x=0, times=5)
for (i in 1:5) {
 a[i] = a[i]+i
}
a
```

```
## [1] 1 2 3 4 5
```

```
# lets say we wanted to look income minus expenses for net income
# create some data for an example

income = runif(min=1000, max=5000, n=10)
income
```

```
##  [1] 3669.918 3533.862 2015.554 1266.658 3132.240 2406.359 3406.016 1472.564
##  [9] 2936.154 1093.203
```

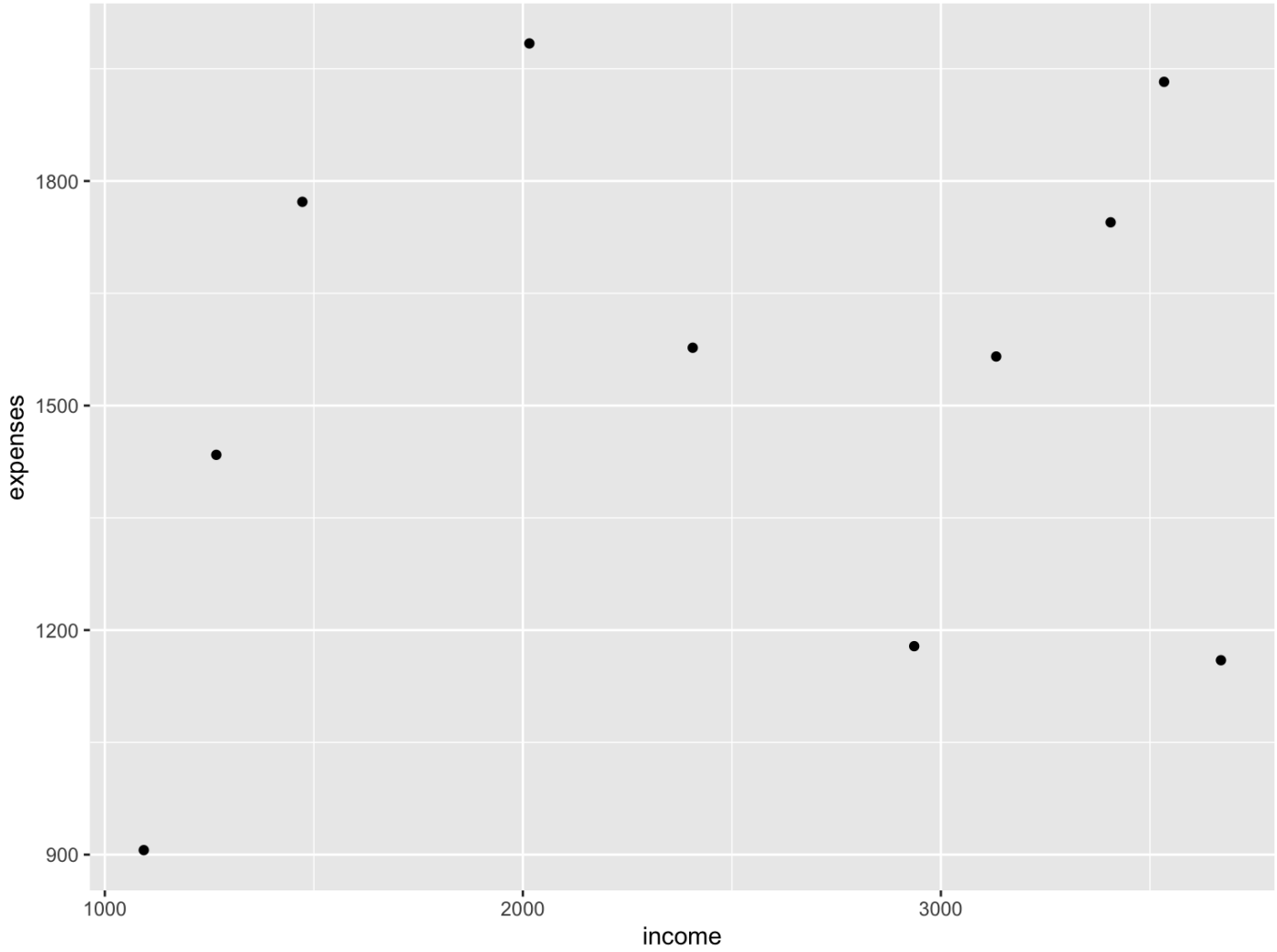
```
expenses = rnorm(mean=1500, sd=500, n=10)
expenses
```

```
##  [1] 1159.7311 1932.5389 1983.8530 1434.1941 1565.5637 1577.2585 1744.8377
##  [8] 1772.2391 1178.5152  906.0263
```

```
# it is possible that this approach could generate negative expenses
expenses = ifelse(expenses < 0, 0, expenses)
# normally we can just do
net = income-expenses
net
```

```
##  [1] 2510.18681 1601.32278   31.70102 -167.53644 1566.67582  829.10059
##  [7] 1661.17811 -299.67466 1757.63891  187.17639
```

```
# lets make this a data frame for nice plotting
account = data.frame(income=income, expenses=expenses, net = net)
ggplot(account, aes(income, expenses))+geom_point()
```




```
# lets add a year
account$year = seq(from=2000, length.out=nrow(account))


# we could have computed net income with a for loop
# sometimes we use NA to show we haven't computed it yet, so "initialize" as NA
netloop = rep(NA, times=10)
# alternatively we could add to account
```

```
account$netloop=NA
account
```

```
##       income  expenses        net year netloop
## 1  3669.918 1159.7311 2510.18681 2000      NA
## 2  3533.862 1932.5389 1601.32278 2001      NA
## 3  2015.554 1983.8530   31.70102 2002      NA
## 4  1266.658 1434.1941 -167.53644 2003      NA
## 5  3132.240 1565.5637 1566.67582 2004      NA
## 6  2406.359 1577.2585  829.10059 2005      NA
## 7  3406.016 1744.8377 1661.17811 2006      NA
## 8  1472.564 1772.2391 -299.67466 2007      NA
## 9  2936.154 1178.5152 1757.63891 2008      NA
## 10 1093.203  906.0263  187.17639 2009      NA
```

```
# note I can use any variable I want to as a counter -
# notice how I'm now using columns of the data frame
for (n in 1:10) {
  account$netloop[n] = account$income[n]-account$expenses[n]
}

# as expected net and netloop are the same :)
head(account)
```

```
##      income expenses        net year    netloop
## 1 3669.918 1159.731 2510.18681 2000 2510.18681
## 2 3533.862 1932.539 1601.32278 2001 1601.32278
## 3 2015.554 1983.853   31.70102 2002   31.70102
## 4 1266.658 1434.194 -167.53644 2003 -167.53644
## 5 3132.240 1565.564 1566.67582 2004 1566.67582
## 6 2406.359 1577.259  829.10059 2005  829.10059
```

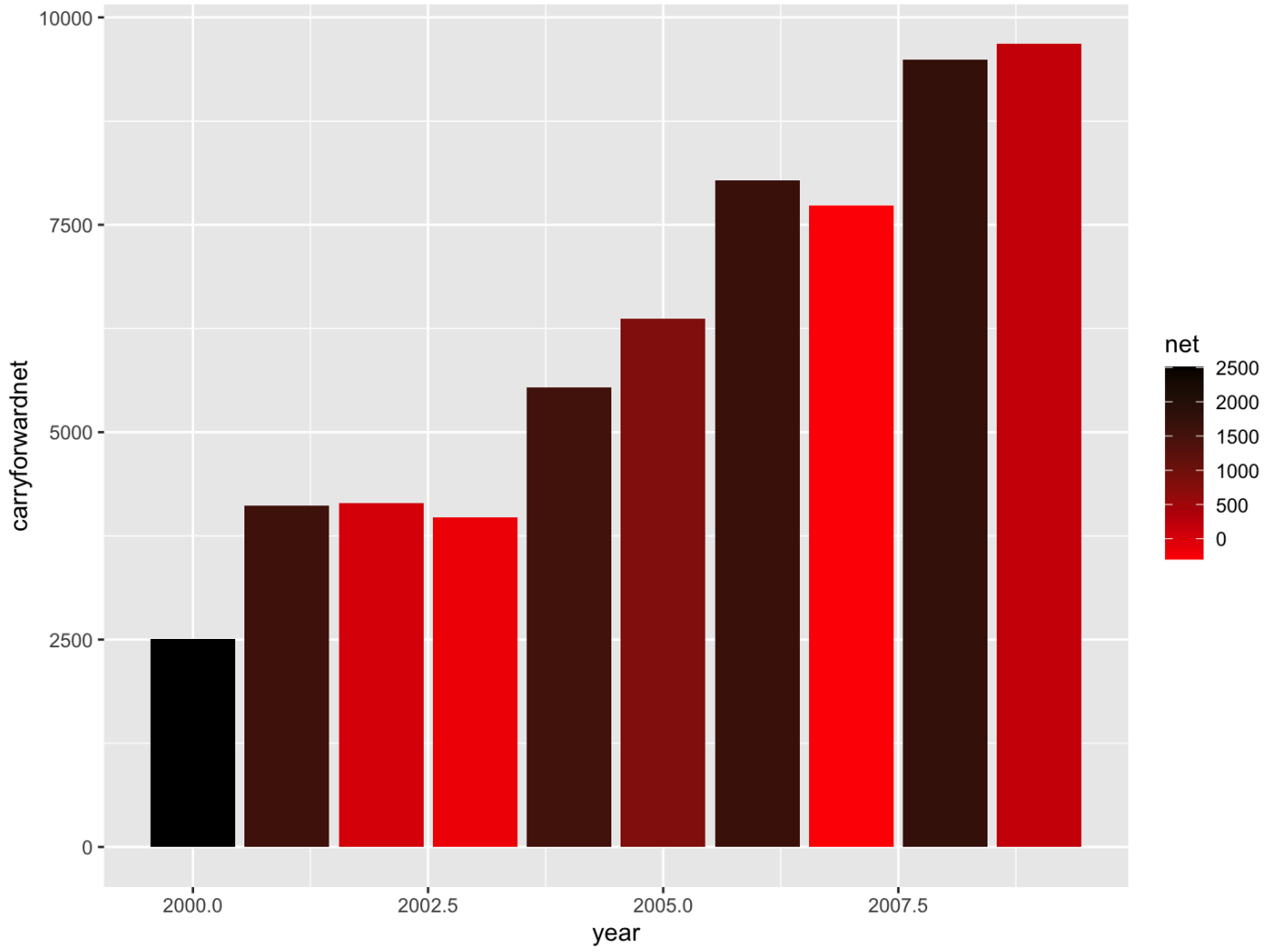
Why loop

what if we wanted to combine different rows and different columns?

for example, we wanted to carry forward from the previous year (info from a different row)

so carry forward net income is current years net income plus last years net

```
#View(account)

# note: we treat first year differently because we don't have carry forward
# initialize our column to zero, start counter (n) at 2 instead of 1
# make the first value equal to the net of first year

account$carryforwardnet = 0
account$carryforwardnet[1]=account$net[1]
# now loop - looking backward one year to get previous years carry forward
for (n in 2:10) {
  account$carryforwardnet[n] = account$net[n]+account$carryforwardnet[n-1]
}

# plot - and use color to show current years contribution as positive or negative
ggplot(account, aes(year, carryforwardnet, fill=net, group=year))+geom_col()+scale_fill_gradient(low="red", high="black")
```




```
# another example - find the maximum speed from a set of speeds - lets make up an example
# by sampling from a random uniform distribution
speeds = runif(min=0, max=100, n=300)

maxspeed=0
for ( i  in 1:length(speeds)) {
  maxspeed = ifelse(speeds[i] > maxspeed, speeds[i], maxspeed)
}

maxspeed
```

```
## [1] 99.97145
```

```
max(speeds)
```

```
## [1] 99.97145
```

Another Example

Try this - make a random sample of fertilizer application, 10 values with mean of *5* and standard deviation of *0.5*

(user *rnorm* function in R to do this)

Function to computed crop yield

Lets imagine that annual yield of a crop can be estimated follows:

$$yield = 1.8 * fertilizer^2 - 0.5 * fertilizer + 0.1 * TP$$

and TP is mean precipitation in cm

Task 1: Use a *for* loop to compute the total yield after 10 years

- fertilizer is normally distributed with mean of 5 and standard deviation of 0.5
- TP is 20cm

Task 2: create a function to that takes as input a single value for annual fertilizer and annual total precipitation and returns yield

Task 3: modify your function so that it returns total yield if the user inputs a vector of fertilizer and a vector of precipitation

Check that function returns the same result as your original *for* loop

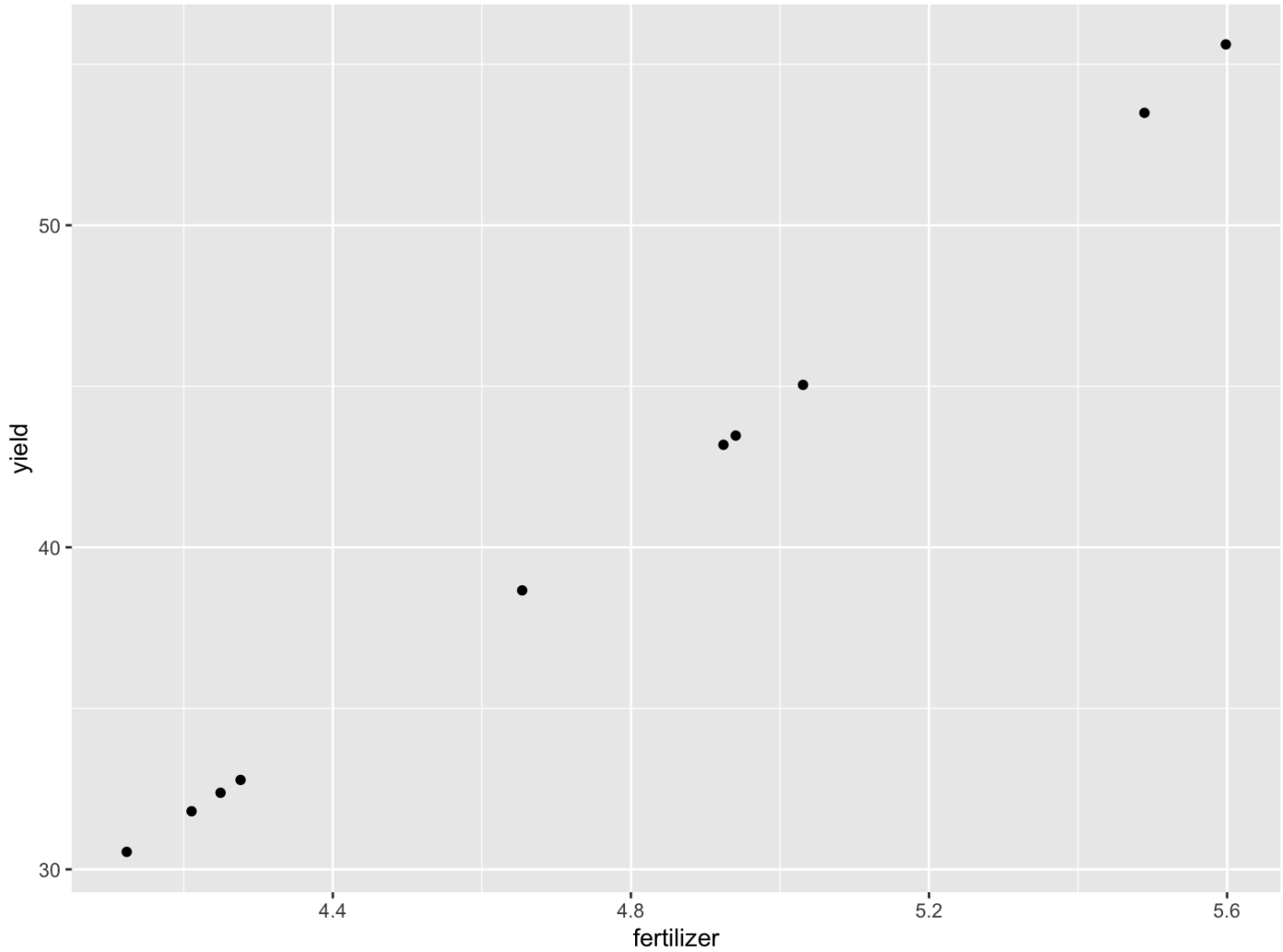
Add some error checking and test

Do on your own first before you check with this

```
fert = rnorm(n=10, mean=5, sd=0.5)
TP = 20
crop = data.frame(fert=fert)
crop$yield = NA
for (yr in 1:10) {
  crop$yield[yr] = 1.8*fert[yr]^2 - 0.5*fert[yr]  + 0.1*TP
}

sum(crop$yield)
```

```
## [1] 406.9616
```

```
ggplot(crop, aes(fert, yield))+geom_point()+labs(y="yield", x="fertilizer")
```




```
# as a function

compute_yield = function(fert, TP) {
  yield = 1.8*fert^2-0.5* fert + 0.1*TP
```

```
  return(yield)
}

# apply function
compute_yield(fert, TP)
```

```
##  [1] 45.04344 31.80435 32.77617 32.37720 38.66181 43.18296 30.54264 53.48983
##  [9] 43.46797 55.61525
```

```
sum(compute_yield(fert, TP))
```

```
## [1] 406.9616
```

```
# a different function that includes annual sum
compute_total_yield = function(fert, TP) {
  yield = 1.8*fert^2-0.5* fert + 0.1*TP
  ty = sum(yield)
  return(ty)
}

compute_total_yield(fert, TP)
```

```
## [1] 406.9616
```

```
# useful error checking
compute_total_yield = function(fert, TP) {

  fert = ifelse(fert < 0, return("fertilizer cannot be negative"), fert)


  yield = 1.8*fert^2-0.5* fert + 0.1*TP
  ty = sum(yield)
  return(ty)
}
```

More formal error checking

R also has specified totols for returning error messages

You can also use *stop()* or *warning()* inside your function to alter the user

stop() stops the remainder of the functions from executing *warning()* continues but lets the user know

```
source("../R/compute_total_yield.R")
compute_total_yield
```

```
## function (fert, TP, coeff1 = 1.8, coeff2 = -0.5, pcoeff = 0.1)
## {
##     fert = ifelse(fert < 0, stop("fertilizer cannot be negative"),
##         fert)
##     if (TP < 0)
##         stop("Total Precipitation (TP) cannot be negative")
##     if (TP == 0)
##         warning("Total Precipitation is zero")
##     yield = coeff1 * fert^2 - coeff2 * fert + pcoeff * TP
##     ty = sum(yield)
##     return(ty)
## }
```

```
compute_total_yield(fert, TP)
```

```
## [1] 454.4582
```

```
compute_total_yield(fert=c(-2, 2,2), TP=3)
```

```
## Error in ifelse(fert < 0, stop("fertilizer cannot be negative"), fert): fertilizer cannot be
negative
```

```
compute_total_yield(fert=c(2,2,2), TP=0)
```

```
## Warning in compute_total_yield(fert = c(2, 2, 2), TP = 0): Total Precipitation
## is zero
```

```
## [1] 24.6
```

- Review looping.Rmd - its a long one