Progamming in R - Functions

map

Functions

↑

an object in R.

advantages

- replicable

Example: Back to HTRU2

Example

```
# Read in the CSV
HTRU2 <-
  read_csv(here("Data_Analyses_MATH_208/Datasets/HTRU2/HTRU_2.csv"),
           col_names=FALSE)
# Name the variables
names(HTRU2) = c("Mean_IP", "SD_IP", "EK_IP", "SKW_IP",
                 "Mean_DMSNR", "SD_DMSNR", "EK_DMSNR", "SKW_DMSNR",
                 "Class")
```

```
HTRU2 <- HTRU2 %>%
  mutate(Class=factor(ifelse(Class==0,"Negative","Positive")))
```

Example

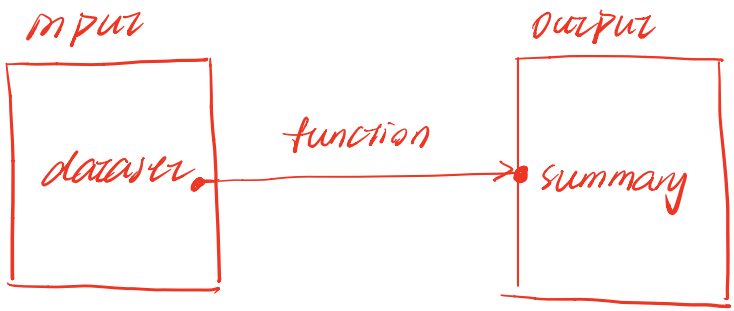
Measurements
1. Mean of the integrated profile
2. Standard deviation of the integrated profile
3. Excess kurtosis of the integrated profile
4. Skewness of the integrated profile
5. Mean of the DM-SNR curve
6. Standard deviation of the DM-SNR curve
7. Excess kurtosis of the DM-SNR curve
8. Skewness of the DM-SNR curve
9. True or false pulsar (human-verified)

Example

```
glimpse(HTRU2)
```

```
Observations: 17,898
Variables: 9
$ Mean_IP    <dbl> 140.562, 102.508, 103.016, 136.750, 88.727, 93.570, 1
$ SD_IP      <dbl> 55.684, 58.882, 39.342, 57.178, 40.672, 46.698, 48.76
$ EK_IP      <dbl> -0.234571, 0.465318, 0.323328, -0.068415, 0.600866, 0
$ SKW_IP     <dbl> -0.699648, -0.515088, 1.051164, -0.636238, 1.123492,
$ Mean_DMSNR <dbl> 3.19983, 1.67726, 3.12124, 3.64298, 1.17893, 1.63629,
$ SD_DMSNR   <dbl> 19.1104, 14.8601, 21.7447, 20.9593, 11.4687, 14.5451,
$ EK_DMSNR   <dbl> 7.9755, 10.5765, 7.7358, 6.8965, 14.2696, 10.6217, 19
$ SKW_DMSNR  <dbl> 74.2422, 127.3936, 63.1719, 53.5937, 252.5673, 131.39
$ Class      <fct> Negative, Negative, Negative, Negative, Negative, Neg
```

The goal



HTRU2 %>% groupby(class) %>% summarise_at(var, funclist)

Assigning a function

name of function object

```
htru2_sum <- function(){
  c("success")
}

class(htru2_sum)
```

the function to create function

```
[1] "function"
```

```
body(htru2_sum)
```

```
{
    c("success")
}
```

```
htru2_sum()
```

```
[1] "success"
```

Adding arguments

argument

```
htru2_sum <- function(htru2_df){
  # This function returns the class of the data frame
  class(htru2_df)
}

htru2_sum(HTRU2)
```

```
[1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

weakly-typed

Returning the result

```
htru2_sum <- function(htru2_df){
  # This function groups by Class and summarises the selected variables
  # by the mean
  htru2_df %>% group_by(Class) %>%
    summarise_at(vars(Mean_IP:SKW_IP),mean)
}
htru2_sum(HTRU2)  %>% kable(.)
```

nice format of result

Class	Mean_IP	SD_IP	EK_IP	SKW_IP
Negative	116.563	47.340	0.21044	0.38084
Positive	56.691	38.711	3.13066	15.55358

```
mean_results_IP<-htru2_sum(HTRU2)
mean_results_IP %>% kable(.)
```

Class	Mean_IP	SD_IP	EK_IP	SKW_IP
Negative	116.563	47.340	0.21044	0.38084
Positive	56.691	38.711	3.13066	15.55358

Returning the result

```
baby_fun<-function(x){
  sum(x)
  print("Success!")
}
baby_fun(c(1:4))
```

```
[1] "Success!"
```

Returning the result

```
baby_fun<-function(x){
  print(x[3])
  sum(x)
}
a<-baby_fun(c(1:4))
```

```
[1] 3
```

```
a
```

```
[1] 10
```

Returning the result

```
baby_fun<-function(x){
  return(sum(x))
  print("Success!")
}
baby_fun(c(1:4))
```

```
[1] 10
```

Anything can be an argument

```
htru2_sum <- function(htru2_df,which_vars,which_fun){
  htru2_df %>% group_by(Class) %>%
    summarise_at(which_vars,which_fun)
}

htru2_sum(htru2_df=HTRU2, which_vars=vars(Mean_IP:SKW_IP),
          which_fun=mean) %>% kable(.)
```

Class	Mean_IP	SD_IP	EK_IP	SKW_IP
Negative	116.563	47.340	0.21044	0.38084
Positive	56.691	38.711	3.13066	15.55358

```
htru2_sum(HTRU2, vars(Mean_IP:SKW_IP),mean) %>% kable(.)
```

Class	Mean_IP	SD_IP	EK_IP	SKW_IP
Negative	116.563	47.340	0.21044	0.38084
Positive	56.691	38.711	3.13066	15.55358

How to set default arguments

```
htru2_sum <- function(htru2_df,which_vars,which_fun){
  htru2_df %>% group_by(Class) %>%
    summarise_at(which_vars,which_fun)
}
htru2_sum(HTRU2, which_vars=vars(Mean_IP:SKW_IP)) %>% kable(.)
```

```
Error in is_fun_list(.funs): argument "which_fun" is missing, with no de
```

```
htru2_sum <- function(htru2_df,which_vars,which_fun=mean){
  htru2_df %>% group_by(Class) %>%
    summarise_at(which_vars,which_fun)
}

htru2_sum(HTRU2, which_vars=vars(Mean_IP:SKW_IP)) %>% kable(.)
```

Class	Mean_IP	SD_IP	EK_IP	SKW_IP
Negative	116.563	47.340	0.21044	0.38084
Positive	56.691	38.711	3.13066	15.55358

Conditional control statements

```
htru2_sum <- function(htru2_df,grouped=TRUE,
                      which_vars,which_fun=mean){
  if(grouped){
    htru2_df %>% group_by(Class) %>%
    summarise_at(which_vars,which_fun)}
  else{
    htru2_df %>% summarise_at(which_vars,which_fun)
  }
}
```

Conditional control statements

```
htru2_sum(HTRU2, grouped=FALSE,
          which_vars=vars(Mean_IP:SKW_IP)) %>% kable(.)
```

Mean_IP	SD_IP	EK_IP	SKW_IP
111.08	46.55	0.47786	1.7703

```
htru2_sum(HTRU2, which_vars=vars(Mean_IP:SKW_IP)) %>% kable(.)
```

Class	Mean_IP	SD_IP	EK_IP	SKW_IP
Negative	116.563	47.340	0.21044	0.38084
Positive	56.691	38.711	3.13066	15.55358

Conditional control statements

```
htru2_sum <- function(htru2_df,one_or_both  = "Both",
                      which_vars,which_fun=mean){
  if(one_or_both=="Both"){
    print("Summaries for both classes")
    htru2_df %>% group_by(Class) %>%
    summarise_at(which_vars,which_fun)}
  else if(one_or_both=="Neither"){
    print("Marginal summaries")
    htru2_df %>% summarise_at(which_vars,which_fun)
  } else{
    print(str_c("Summaries for group",one_or_both,sep=" "))
    htru2_df %>% filter(Class==one_or_both) %>%
    summarise_at(which_vars,which_fun)
    }
}
```

concatenate string variable

if not valid -> empty.

Conditional control statements

```
htru2_sum(HTRU2, one_or_both="Negative",
          which_vars=vars(Mean_IP:SKW_IP)) %>% kable(.)
```

```
[1] "Summaries for group Negative"
```

Mean_IP	SD_IP	EK_IP	SKW_IP
116.56	47.34	0.21044	0.38084

```
htru2_sum(HTRU2, one_or_both="Neither",
          which_vars=vars(Mean_IP:SKW_IP)) %>% kable(.)
```

```
[1] "Marginal summaries"
```

Mean_IP	SD_IP	EK_IP	SKW_IP
111.08	46.55	0.47786	1.7703

Conditional control statements: if/else vs. ifelse

```
a<-c(1,2,3)
ifelse(a%%2 ==0, "Even", "Odd")
```

```
[1] "Odd"  "Even" "Odd"
```

```
if(a%%2 ==0){
  print("Even")
} else{
  print("odd")
}
```

```
Warning in if (a%%2 == 0) {: the condition has length > 1 and only the
first element will be used
```

```
[1] "odd"
```

Multiple conditions: switch

```
htru2_sum_switch <- function(htru2_df,one_or_both  = "Both",
                        which_vars,which_fun=mean){
  print(str_c("Printing summaries for: ", one_or_both))
  switch(one_or_both,
  1 Both = htru2_df %>% group_by(Class) %>%
    summarise_at(which_vars,which_fun),
  2 Neither = htru2_df %>% summarise_at(which_vars,which_fun),
  3 Negative = htru2_df %>% filter(Class=="Negative") %>%
      summarise_at(which_vars,which_fun),
  4 Positive = htru2_df %>% filter(Class=="Positive") %>%
      summarise_at(which_vars,which_fun)
  )

}
```

? NOT valid

Argument is missing, with no default

Multiple conditions: switch

```
htru2_sum_switch(HTRU2, one_or_both="Negative",
          which_vars=vars(Mean_IP:SKW_IP)) %>% kable(.)
```

```
[1] "Printing summaries for: Negative"
```

Mean_IP	SD_IP	EK_IP	SKW_IP
116.56	47.34	0.21044	0.38084

Environments and scope

Environment as a collection of things

```
environment()
```

```
<environment: R_GlobalEnv>
```

```
ls()
```

list of all objects in the environment

```
[1] "a"                 "baby_fun"          "HTRU2"
[4] "htru2_sum"         "htru2_sum_switch"  "mean_results_IP"
[7] "meas_table"
```

Environment as a collection of things

```
print(ls.str(),max.level=0)
```

```
a :  num [1:3] 1 2 3
baby_fun : function (x)
HTRU2 : Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame':    1789
htru2_sum : function (htru2_df, one_or_both = "Both", which_vars, which_
htru2_sum_switch : function (htru2_df, one_or_both = "Both", which_vars,
mean_results_IP : Classes 'tbl_df', 'tbl' and 'data.frame': 2 obs. of  5
meas_table : Classes 'tbl_df', 'tbl' and 'data.frame':  9 obs. of  1 var
```

Environment as a collection of things

```
my_global_fun <- function(x){
  len_x <- length(x)
  y<-seq(-len_x/2,len_x/2, length=len_x)
  euclid_dist <- function(x,y){
    sum((x-y)^2)
  }
  euclid_dist(x,y)
  }
```

Environment as a collection of things

```
environment(my_global_fun)
```

```
<environment: R_GlobalEnv>
```

```
environment(euclid_dist)
```

inside the function

```
Error in environment(euclid_dist): object 'euclid_dist' not found
```

Environment as a collection of things

```
my_global_fun <- function(x){
  len_x <- length(x)
  y<-seq(-len_x/2,len_x/2, length=len_x)
  euclid_dist <- function(x,y){
     sum((x-y)^2)
   }
   print("Euclid_dist environ:")
   print(environment(euclid_dist))
   print(ls.str())
   euclid_dist(x,y)
   }
```

Environment as a collection of things

```
my_global_fun(c(1,2,3))
```

pass by value.

```
[1] "Euclid_dist environ:"
<environment: 0x7fb4a680e0b0>
euclid_dist : function (x, y)
len_x :  int 3
x :  num [1:3] 1 2 3
y :  num [1:3] -1.5 0 1.5
```

memory address

copy

```
[1] 12.5
```

```
environment()
```

```
<environment: R_GlobalEnv>
```

Objects passed as arguments are copies, not original

```
w <- c(0.1,0.2,0.3,0.4)
z <- c(10,15,20,30)
sum(z*w)
```

```
[1] 22
```

```
my_boring_fun <- function(z,w){
  z <- c(0,0,0,0)
  sum(z*w)
}
```

Objects passed as arguments are copies, not original

```
body(my_boring_fun)
```

```
{
    z <- c(0, 0, 0, 0)
    sum(z * w)
}
```

```
my_boring_fun(z,w)
```

```
[1] 0
```

```
z
```

```
[1] 10 15 20 30
```

What if I call something that only exists globally?

```r
this_is_global <- c(1,2,3)

not_a_good_idea <- function(){
  this_is_global[2]<-10
  this_is_global
}
```

copy

If a variable not found,
go to higher level, find it and put a copy in
the local environment.

What if I call something that only exists globally?

```
temp<-not_a_good_idea()
temp
```

```
[1]  1 10  3
```

```
this_is_global
```

```
[1] 1 2 3
```

Checking what objects are out of function scope

```
are_you_kidding <- c(1,2,3)

terrible_idea <- function(){
  2*are_you_kidding
}

terrible_idea()
```

```
[1] 2 4 6
```

use function without loading the package.

```
codetools::findGlobals(terrible_idea)
```

```
[1] "{"                "*"                "are_you_kidding"
```

Using `with()` (as a replacement for dollar signs)

```
ls()
```

```
 [1] "a"                  "are_you_kidding"   "baby_fun"
 [4] "HTRU2"              "htru2_sum"         "htru2_sum_switch"
 [7] "mean_results_IP"    "meas_table"        "my_boring_fun"
[10] "my_global_fun"      "not_a_good_idea"   "temp"
[13] "terrible_idea"      "this_is_global"    "w"
[16] "z"
```

environment (function's scope)

```
with(HTRU2,ls())
```

↑ func.

```
[1] "Class"       "EK_DMSNR"    "EK_IP"       "Mean_DMSNR"  "Mean_IP"
[6] "SD_DMSNR"    "SD_IP"       "SKW_DMSNR"   "SKW_IP"
```

↗ not in global environment

```
mean(EK_IP)
```

```
Error in mean(EK_IP): object 'EK_IP' not found
```

```
with(HTRU2,mean(Mean_IP))
```

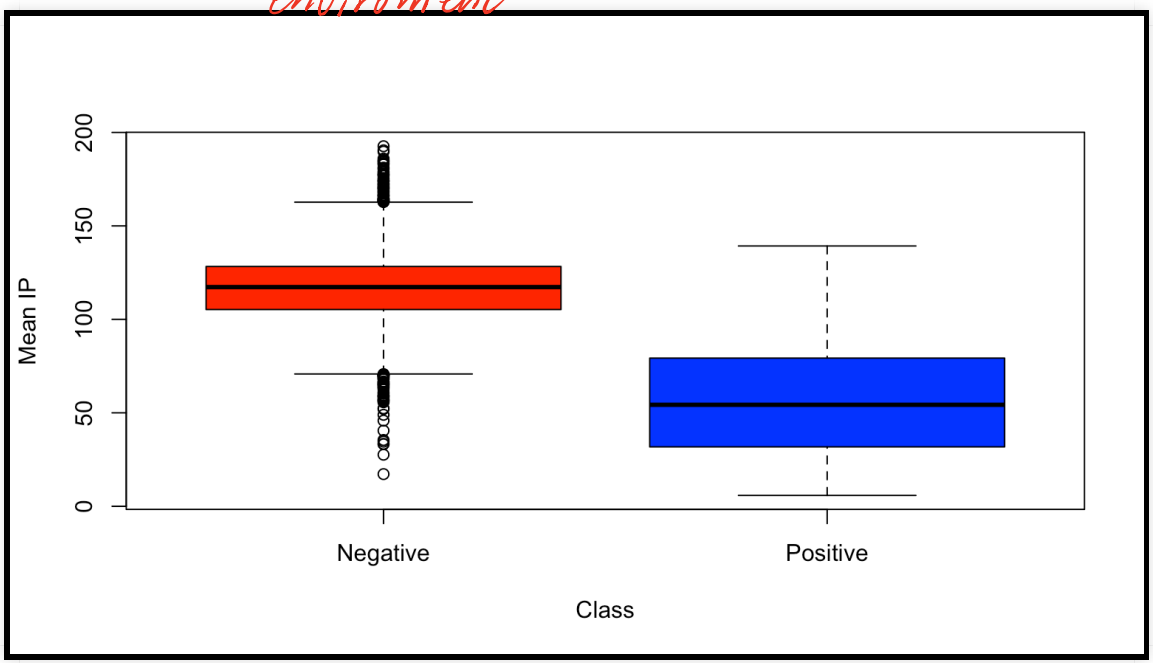
```
[1] 111.08
```

Using `with()` and baseR plotting

```
boxplot(Mean_IP~Class,ylab="Mean IP",
        xlab="Class", data=HTRU2,col=c(2,4))
```

environment

color



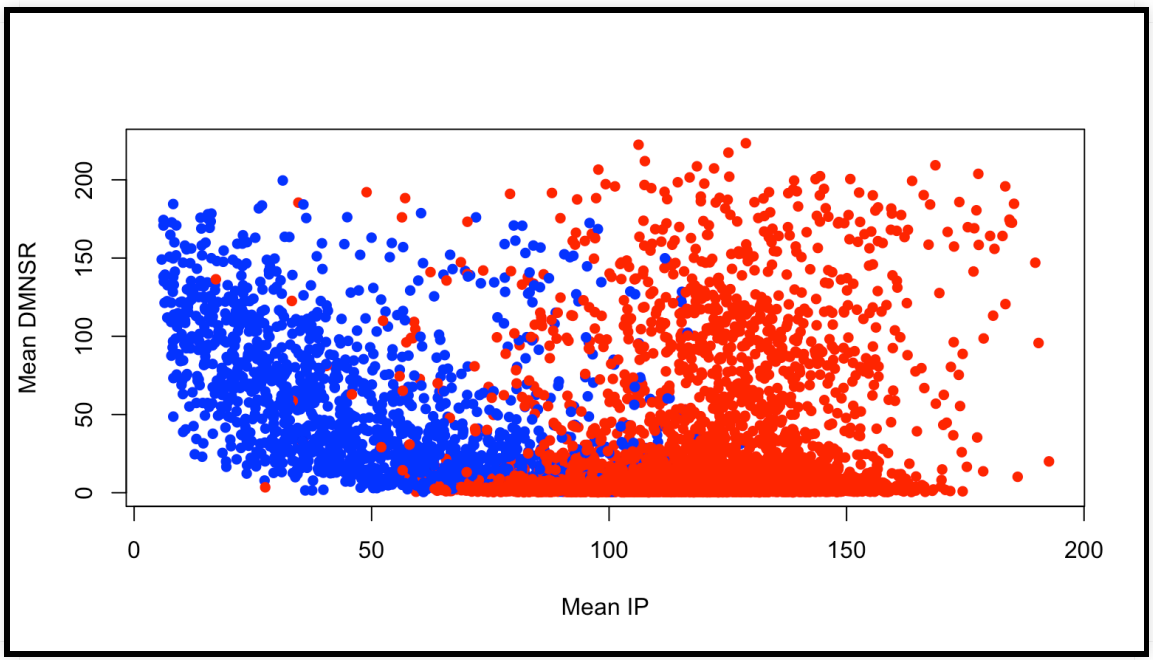
Using `with()` and baseR plotting

don't have enviroment arg.

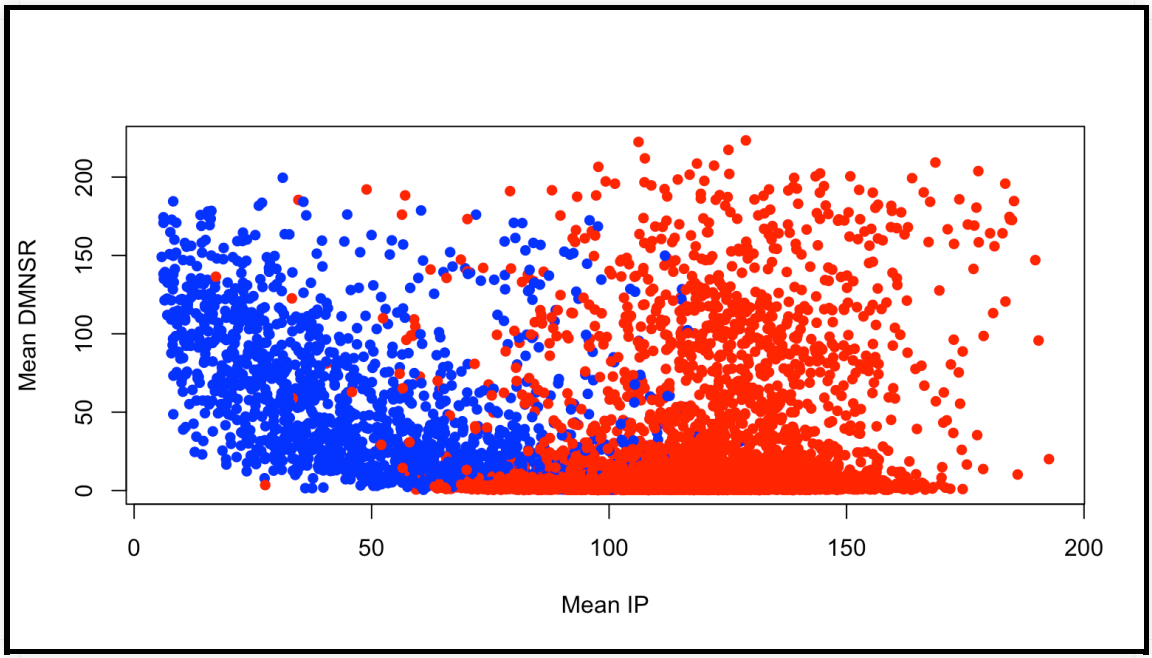
```
plot(HTRU2$Mean_IP,HTRU2$Mean_DMSNR,
     col=ifelse(HTRU2$Class=="Negative",2,4),
     pch=16,xlab="Mean IP", ylab="Mean DMNSR")
```

never attach.

solid point



```
with(HTRU2,
     plot(Mean_IP,Mean_DMSNR,
      col=ifelse(Class=="Negative",2,4),
      pch=16,xlab="Mean IP", ylab="Mean DMNSR"))
```



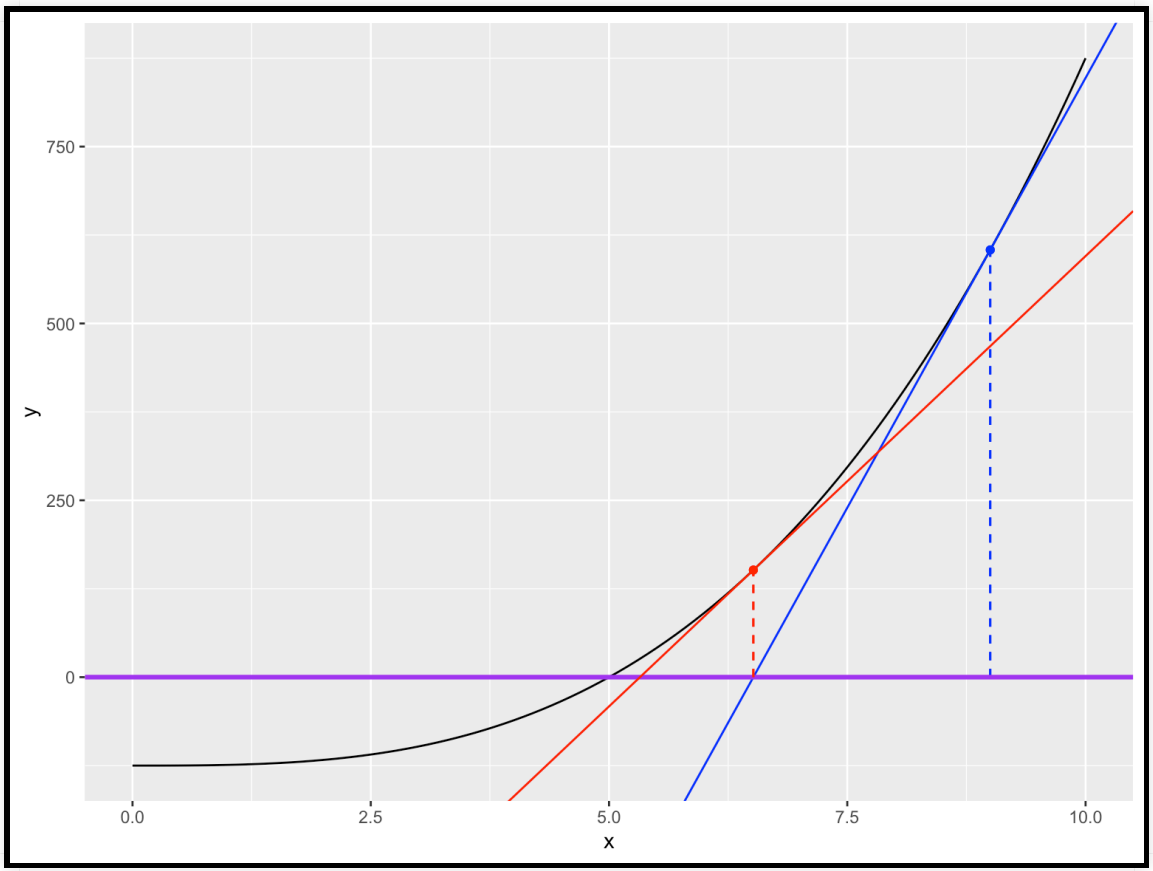
Basic Iteration

An example: Newton's method

```r
f_x <- function(x,target){
  x^3 - target
}
df_x <- function(x,target){
  3*x^2
}
##OR
library(Deriv)
df_x <- Deriv(f_x,"x")
df_x
```

```
function (x, target)
3 * x^2
```

An example: Newton's method



`for` loops

```
for(index in Some list){
    ... Do All This Stuff...
}
```

`for` loops: looping over elements

a vector of all capital letters.

```
for(each_letter in LETTERS){
    if(each_letter %in% c("A","E","I","O","U")){
      print(str_c(each_letter,"is a vowel",sep=" "))
    }
}
```

```
[1] "A is a vowel"
[1] "E is a vowel"
[1] "I is a vowel"
[1] "O is a vowel"
[1] "U is a vowel"
```

vec 1 %in% vec 2

return c(T, F, F, ... F)

`for` loops: looping over indices

```
for(i in 1:26){
    if(LETTERS[i] %in% c("A","E","I","O","U")){
      print(str_c(LETTERS[i],"is a vowel",sep=" "))
    }
}
```

```
[1] "A is a vowel"
[1] "E is a vowel"
[1] "I is a vowel"
[1] "O is a vowel"
[1] "U is a vowel"
```

`for` loops: looping over indices via seq_along

a set of index from 1 to length

```
for(i in seq_along(LETTERS)){
    if(LETTERS[i] %in% c("A","E","I","O","U")){
      print(str_c(LETTERS[i],"is a vowel",sep=" "))
    }
}
```

```
[1] "A is a vowel"
[1] "E is a vowel"
[1] "I is a vowel"
[1] "O is a vowel"
[1] "U is a vowel"
```

Back to Newton's

put everything outside inside function

```
newtons_method<-function(init,g_x,dg_x,nsteps=10, ...){
  # Set current to initial value
  current<-init
  # Print current status
  print(str_c("Step:",0,"x=:",current,"f(x):",g_x(current,...),sep="
  "))
  # Loop over number of steps
  for(i in 1:nsteps){
    # Gradient update
    current<-current - g_x(current,...)/dg_x(current,...)
    print(str_c("Step:",i,"x=:",current,"f(x):",g_x(current,...),sep="
  "))
  }
  # Return current value
  current
}
```

Back to Newton's

```
newtons_method(9,f_x,df_x,nsteps=10,target=125)
```

```
[1] "Step: 0 x=: 9 f(x): 604"
[1] "Step: 1 x=: 6.51440329218107 f(x): 151.454665641083"
[1] "Step: 2 x=: 5.32477271244931 f(x): 25.9743693085871"
[1] "Step: 3 x=: 5.01940606080981 f(x): 1.46111079690725"
[1] "Step: 4 x=: 5.00007493114922 f(x): 0.00561992041218673"
[1] "Step: 5 x=: 5.00000000112291 f(x): 8.42184988414374e-08"
[1] "Step: 6 x=: 5 f(x): 0"
[1] "Step: 7 x=: 5 f(x): 0"
[1] "Step: 8 x=: 5 f(x): 0"
[1] "Step: 9 x=: 5 f(x): 0"
[1] "Step: 10 x=: 5 f(x): 0"
```

```
[1] 5
```

Different function

```r
h_x <- function(x,target){
  cos(x) - target
}
dh_x <- Deriv(h_x,"x")
newtons_method(0.9,h_x,dh_x,nsteps=10,target= (-1))
```

```
[1] "Step: 0 x=: 0.9 f(x): 1.62160996827066"
[1] "Step: 1 x=: 2.97015736130121 f(x): 0.0146590743116977"
[1] "Step: 2 x=: 3.056085563519 f(x): 0.00365350437383427"
[1] "Step: 3 x=: 3.09886517685924 f(x): 0.000912679769354274"
[1] "Step: 4 x=: 3.12023216602107 f(x): 0.000228126540439244"
[1] "Step: 5 x=: 3.13091281591391 f(x): 5.70289243336797e-05"
[1] "Step: 6 x=: 3.13625278550789 f(x): 1.42570616886717e-05"
[1] "Step: 7 x=: 3.13892272589311 f(x): 3.56425483538647e-06"
[1] "Step: 8 x=: 3.14025769053447 f(x): 8.91063047236962e-07"
[1] "Step: 9 x=: 3.14092517216128 f(x): 2.22765720425677e-07"
[1] "Step: 10 x=: 3.14125891288791 f(x): 5.56914275806619e-08"
```

```
[1] 3.1413
```

`while` loops

```
while(this_condition_is_true){

  ... Do All This Stuff...

   ### Definitely don't forget if not above
  ...Update Your Condition...

   ### And for your sanity
    if(this_is_taking_too_long){
      break;
    }
}
```

Using `while` loops for Newton

```
newtons_method_while<-function(init,g_x,dg_x,
                               eps_f=1e-8,
                               eps_df=1e-8,
                               eps_x=1e-8, maxsteps=50,...){

  # Set current to initial value
  current<-init
  # Set number of steps to zero
  nsteps<-0
  # Print current status

  print(str_c("Step:",nsteps,"x=:",current,"f(x):",g_x(current,...),sep=
   "))
  flag<-TRUE
```

Using `while` loops for Newton

```
while(flag){
   previous<-current
   nsteps<-nsteps+1
   # Gradient update
   current<-current - g_x(current,...)/dg_x(current,...)
   print(str_c("Step:",nsteps,"x=:",current,"f(x):",
               g_x(current,...),"df(x):",
               dg_x(current,...),sep=" "))
   # Update condition
   flag<-all(abs(previous-current)>eps_x,abs(g_x(previous,...)-
 g_x(current,...))>eps_f,

             abs(dg_x(previous,...)-dg_x(current,...))>eps_df)
   # If taking too long
   if(nsteps>=maxsteps){ break;}
 }
```

any
union ∪

intersection ∩

Using `while` loops for Newton

```
newtons_method_while(0.9,f_x,df_x,target=125)
```

```
[1] "Step: 0 x=: 0.9 f(x): -124.271"
[1] "Step: 1 x=: 52.040329218107 f(x): 140810.404408427 df(x): 8124.5875
[1] "Step: 2 x=: 34.7089382083801 f(x): 41689.2185394114 df(x): 3614.131
[1] "Step: 3 x=: 23.173878598005 f(x): 12320.0367219512 df(x): 1611.0859
[1] "Step: 4 x=: 15.5268398173727 f(x): 3618.25331518069 df(x): 723.2482
[1] "Step: 5 x=: 10.5240579310341 f(x): 1040.60040631133 df(x): 332.2673
[1] "Step: 6 x=: 7.39224165855622 f(x): 278.950795459962 df(x): 163.9357
[1] "Step: 7 x=: 5.69065513361116 f(x): 59.2836483406509 df(x): 97.15066
[1] "Step: 8 x=: 5.08043134579521 f(x): 6.12990928201091 df(x): 77.43234
[1] "Step: 9 x=: 5.00126663696123 f(x): 0.0950218396623939 df(x): 75.038
[1] "Step: 10 x=: 5.00000032076549 f(x): 2.40574134124927e-05 df(x): 75.
[1] "Step: 11 x=: 5.00000000000002 f(x): 1.53477230924182e-12 df(x): 75.
[1] "Step: 12 x=: 5 f(x): 0 df(x): 75"
```

```
[1] 5
```

Iterating over objects

`gapminder` dataset

```
library(gapminder)
glimpse(gapminder)
```

```
Observations: 1,704
Variables: 6
$ country   <fct> Afghanistan, Afghanistan, Afghanistan, Afghanistan, Af
$ continent <fct> Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia, Asia,
$ year      <int> 1952, 1957, 1962, 1967, 1972, 1977, 1982, 1987, 1992,
$ lifeExp   <dbl> 28.801, 30.332, 31.997, 34.020, 36.088, 38.438, 39.854
$ pop       <int> 8425333, 9240934, 10267083, 11537966, 13079460, 148803
$ gdpPercap <dbl> 779.45, 820.85, 853.10, 836.20, 739.98, 786.11, 978.01
```

gapminder dataset

don't need to use gapminder$country.

dataframe

```
gap_list_country<-with(gapminder,split(gapminder,country))
length(gap_list_country)
```

a list with one country in a row

factor variable vector

```
[1] 142
```

small dataframe (each has the same country)

Note: "country" still exist in the small dataframes.

```
names(gap_list_country)[1:5]
```

```
[1] "Afghanistan" "Albania"     "Algeria"     "Angola"      "Argentina"
```

Finding correlation between GDPpercap and life expectancy per country via `for` loop

[Add elements to a vector]

create empty vector

```
num_countries <- length(gap_list_country)
correlations<-vector("numeric",num_countries)
names(correlations)<-names(gap_list_country)
for(i in seq_along(gap_list_country)){
 correlations[i]<-with(gap_list_country[[i]], cor(gdpPercap,lifeExp))
}
head(correlations)
```

1: length(list)

```
Afghanistan      Albania      Algeria       Angola    Argentina    Australia
  -0.047547     0.837116     0.904471    -0.301079     0.831648     0.986446
```

Finding correlation between GDPpercap and life expectancy per country via `lapply`

```
cor_for_lapply<-function(x){
  with(x,cor(gdpPercap,lifeExp))
}
correlations_lapply<-lapply(gap_list_country,
                            cor_for_lapply)
```

list

for each element in the list, apply the function

Finding correlation between GDPpercap and life expectancy per country via `lapply`

```
head(correlations_lapply)
```

```
$Afghanistan
[1] -0.047547

$Albania
[1] 0.83712

$Algeria
[1] 0.90447

$Angola
[1] -0.30108

$Argentina
[1] 0.83165

$Australia
[1] 0.98645
```

Finding correlation between GDPpercap and life expectancy per country via `lapply`

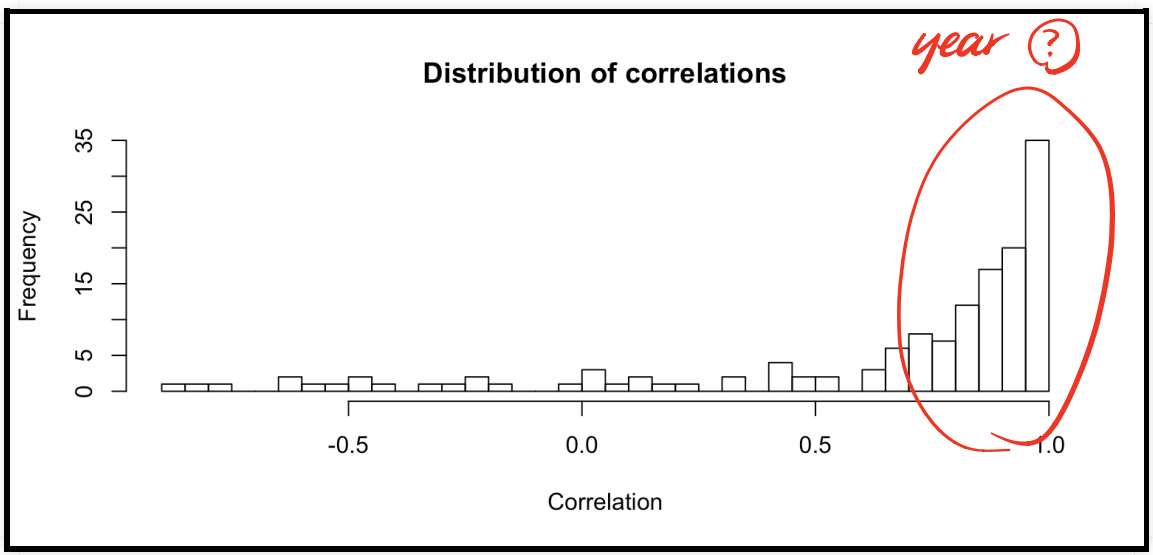
```
head(unlist(correlations_lapply))
```

turn to a vector (same mode in a vector)

```
Afghanistan     Albania     Algeria      Angola   Argentina   Australia
  -0.047547    0.837116    0.904471   -0.301079    0.831648    0.986446
```

Finding correlation between GDPpercap and life expectancy per country via `lapply`

```
hist(unlist(correlations_lapply), breaks=30,
     main=c("Distribution of correlations"),
     xlab=c("Correlation"))
```



Finding correlation between GDPpercap and life expectancy per country via `sapply`

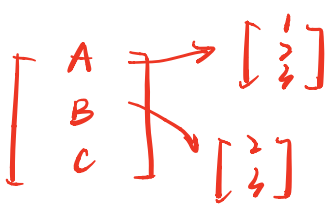
lapply + unlist

```
correlations_sapply<-sapply(gap_list_country,
                            cor_for_lapply)
head(correlations_sapply)
```

```
Afghanistan     Albania     Algeria      Angola   Argentina   Australia
  -0.047547    0.837116    0.904471   -0.301079    0.831648    0.986446
```

unexpected consequence

use sapply on a list containing subsets with different lengths.



Finding correlation between GDPpercap and life expectancy per country via `map`

tidyverse

```
using_cor_map<-gap_list_country %>% map(cor_for_lapply)
head(using_cor_map)
```

```
$Afghanistan
[1] -0.047547

$Albania
[1] 0.83712

$Algeria
[1] 0.90447

$Angola
[1] -0.30108

$Argentina
[1] 0.83165

$Australia
[1] 0.98645
```

Finding correlation between GDPpercap and life expectancy per country via `map`

unlist to double

```
using_cor_map_dbl<- gap_list_country %>% map_dbl(cor_for_lapply)
head(using_cor_map_dbl)
```

```
Afghanistan     Albania     Algeria      Angola   Argentina   Australia
  -0.047547    0.837116    0.904471   -0.301079    0.831648    0.986446
```

map_ int

map_ dfr data.frame + tibble

map_ chr string

Using `map` anonymous functions

```
using_cor_map_anon<-gap_list_country %>% map_dbl(function(x){
        with(x,cor(lifeExp,gdpPercap))})
head(using_cor_map_anon)
```

works for lapply too

```
Afghanistan     Albania     Algeria      Angola   Argentina   Australia
  -0.047547    0.837116    0.904471   -0.301079    0.831648    0.986446
```

```
using_cor_map_anon2<-gap_list_country %>%
        map_dbl(~with(.x,cor(lifeExp,gdpPercap)))
head(using_cor_map_anon2)
```

only for map

pipe in.

function(x) { }

```
Afghanistan     Albania     Algeria      Angola   Argentina   Australia
  -0.047547    0.837116    0.904471   -0.301079    0.831648    0.986446
```

`tapply` as a baseR `group_by`

```
### Want to find mean lifeExp per continent by year
my_lexp_summaries<-with(gapminder,
        tapply(lifeExp,data.frame(year,continent),mean))
class(my_lexp_summaries)
```

vector groupby apply this function

```
[1] "matrix"
```

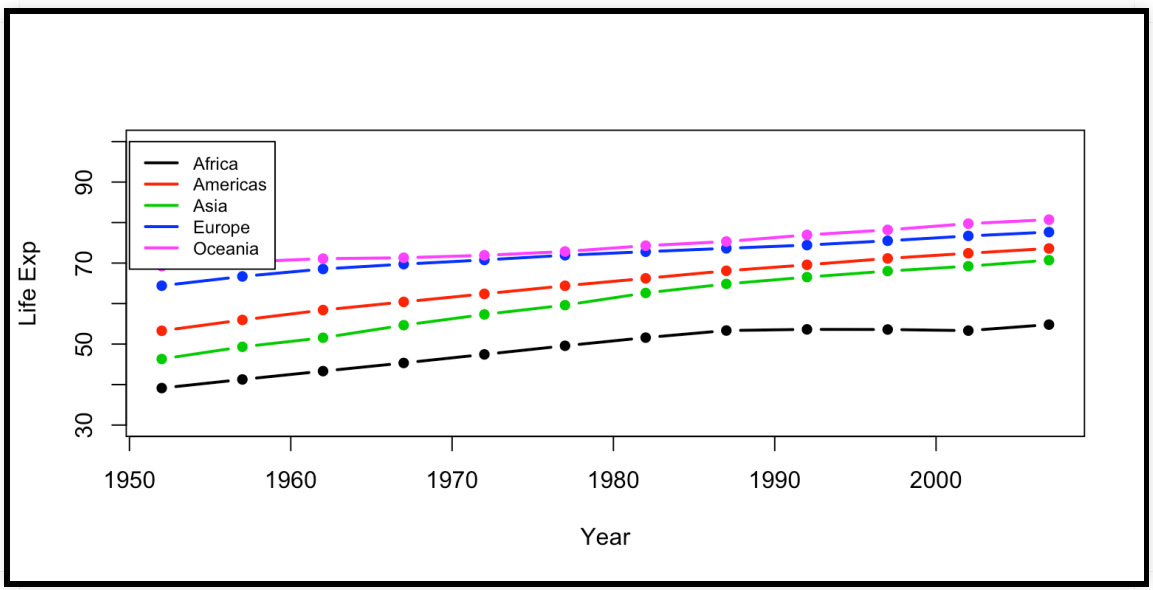
2d.

```
head(my_lexp_summaries)
```

```
      continent
year   Africa Americas   Asia Europe Oceania
  1952 39.136   53.280 46.314 64.409  69.255
  1957 41.266   55.960 49.319 66.703  70.295
  1962 43.319   58.399 51.563 68.539  71.085
  1967 45.335   60.411 54.664 69.738  71.310
  1972 47.451   62.395 57.319 70.775  71.910
  1977 49.580   64.392 59.611 71.938  72.855
```

tapply as a baseR group_by

```
matplot(as.numeric(rownames(my_lexp_summaries)),
        my_lexp_summaries,type="b",pch=16,lwd=2,lty=1,xlab="Year",
        ylab="Life Exp",ylim=c(30,100),col=c(1:4,6))
legend(x=1950,y=100,legend=colnames(my_lexp_summaries),
       lwd=2,col=c(1:4,6),cex=0.75)
```



tapply as a baseR group_by

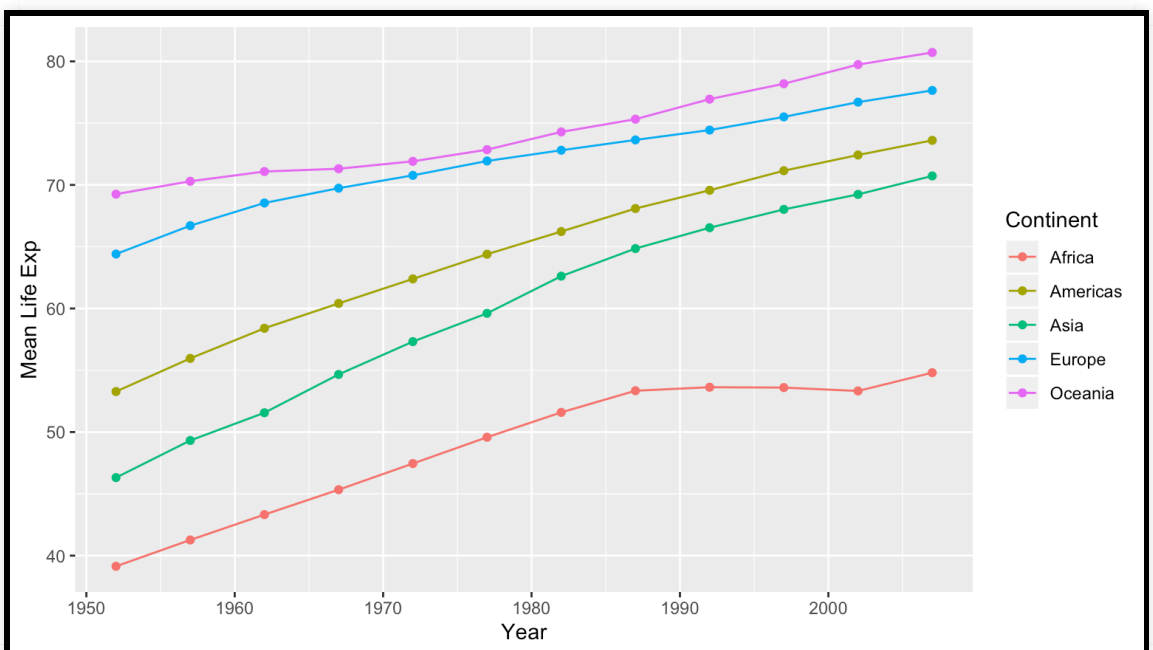
```
## melt is baseR pivot_long, cast is baseR pivot_wide
my_lexp_summaries_plot<-cbind(Year=as.numeric(rownames(my_lexp_summaries
                        as.data.frame(my_lexp_summaries))

library(reshape2)
my_lexp_summaries_plot<-melt(my_lexp_summaries_plot,
                             variable.name="Continent",
                             value.name="Mean_Life_Exp",
                             id.vars="Year")
head(my_lexp_summaries_plot)
```

```
  Year Continent Mean_Life_Exp
1 1952    Africa        39.136
2 1957    Africa        41.266
3 1962    Africa        43.319
4 1967    Africa        45.335
5 1972    Africa        47.451
6 1977    Africa        49.580
```

`tapply` as a baseR `group_by`

```
ggplot(my_lexp_summaries_plot,aes(x=as.numeric(Year),y=Mean_Life_Exp,col
          +
  geom_point() + geom_line() +
  labs(x="Year",y="Mean Life Exp", legend="Continent")
```



Women's mobility dataset

- A rural subsample of 8445 women from the Bangladesh Fertility Survey of 1989.
- Women were asked whether they could engage in the following activities alone (1 = yes, 0 = no):

Women's mobility dataset items

```
Item 1: Go to any part of the village/town/city.
Item 2: Go outside the village/town/city.
Item 3: Talk to a man you do not know.
Item 4: Go to a cinema/cultural show.
Item 5: Go shopping.
Item 6: Go to a cooperative/mothers' club/other club.
Item 7: Attend a political meeting.
Item 8: Go to a health centre/hospital.
```

Using `apply` to compute total sum score

```
library(ltm)
data(Mobility) # Data containing response to 8 questions on women's
        mobility
head(Mobility) %>% kable()
```

Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8
1	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Using `apply` to compute total sum score

new dataset

```
Mobility_new<-Mobility
Mobility_new$TotalScore<-apply(Mobility,1,sum)
head(Mobility_new) %>% kable()
```

row → function

2 column

Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Item 7	Item 8	TotalScore
1	1	1	1	0	0	0	0	4
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Using `apply` to compute column summaries

```
my_summaries<-function(x){
  y<-c(mean(x),sd(x))
  names(y)<-c("Mean","Sd")
  y  }
apply(Mobility_new,2,my_summaries) %>% t(.) %>% kable()
```

return

take original format

transpose

	Mean	Sd
Item 1	0.79870	0.40100
Item 2	0.31391	0.46411
Item 3	0.75181	0.43199
Item 4	0.36471	0.48138
Item 5	0.06939	0.25413
Item 6	0.11119	0.31439
Item 7	0.05305	0.22414
Item 8	0.08668	0.28138
TotalScore	2.54944	1.80270

Note: Everything in a column should be in the same mode.

t(.) doesn't work all the time

Arrays: contingency tables

```
syph_counts_array
```

```
, , Age = <=19

        Race
Sex      Black Other White
  Female  2422   169   267
  Male    1443   217    90

, , Age = 20-29

        Race
Sex      Black Other White
  Female  8093   590   908
  Male    8180  1287   957

, , Age = 30-44

        Race
Sex      Black Other White
```

Arrays: contingency tables

```
apply(syph_counts_array, c(2),sum)
```

```
Black Other White
35508  3956  4617
```

```
apply(syph_counts_array, c("Race"),sum)
```

```
Black Other White
35508  3956  4617
```

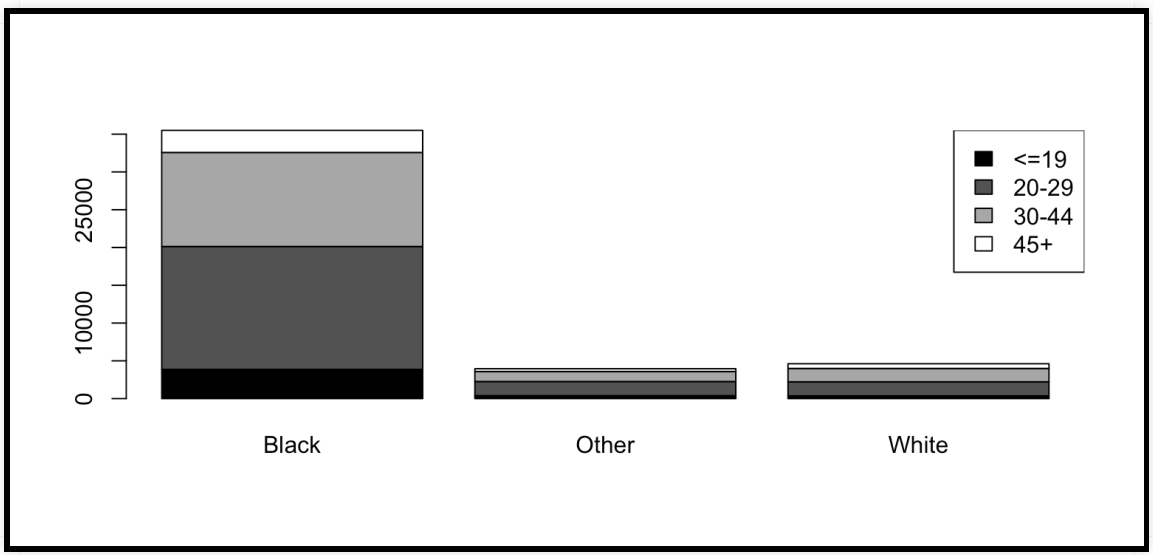
Arrays: contingency tables

```
apply(syph_counts_array, c("Race","Age"),sum)
```

```
       Age
Race    <=19 20-29 30-44  45+
  Black 3865 16273 12444 2926
  Other  386  1877  1328  365
  White  357  1865  1777  618
```

Arrays: contingency tables

```
myplot<-barplot(apply(syph_counts_array, c("Age","Race"),sum),
                col=grey(seq(0,1,length=4)))
legend("topright",fill=grey(seq(0,1,length=4)),
       legend=levels(factor(syph_data$Age)))
```



Arrays: contingency tables

```
column_props <-apply(syph_counts_array, c("Age","Race"),sum) %>%
                  prop.table(.,c(2))
column_props
```

```
       Race
Age         Black    Other    White
  <=19  0.108849 0.097573 0.077323
  20-29 0.458291 0.474469 0.403942
  30-44 0.350456 0.335693 0.384882
  45+   0.082404 0.092265 0.133853
```

Arrays: contingency tables

```
layout(matrix(c(1,2),nrow=1), c(2,1)) ## For multiple plots
myplot<-barplot(column_props,col=grey(seq(0,1,length=4))) # First plot
# Need empty plot for legend
plot(c(0,0),type="n",axes=FALSE,ylab="",xlab="")
legend("center",fill=grey(seq(0,1,length=4)),
       legend=levels(factor(syph_data$Age)))
```