Functions and purrr

Abhijit Dasgupta

March 25-27, 2019

What are functions?

- Functions are a set of instructions that have been packaged up for repeated use
- They can take single or multiple inputs
- They can have single or multiple outputs

Why functions?

- If you need to re-use code, you could copy and paste it every time
  - If you change the code, you will need to change every copy (yeah, right)
  - Chances are, you won't get everything, so you introduce error
- In a function, you change in one place and you make mistakes in one place
  - Easier to maintain
  - Easier to fix
  - Changes propagate directly

Rule of thumb

If you have to copy-and-paste more than twice, make a function

Creating a function

We can create a new function using the word "function" followed by the functions arguments and one or more R statements.

```
myDumbFunction <- function() 42
myDumbFunction()
```

```
[1] 42
```

- This is a function with **no** arguments. Usually functions have arguments, which we will see next. Here, `myDumbFunction` gives the same answer whenever it's called

Creating a multi-statement function

If there is more than one statement in a function, they should be enclosed in curly brackets:

```
doubleIt <- function(x) {
  myResult <- x * 2
  myResult # or, explicitly, return(myResult)
}
doubleIt(5)
```

```
[1] 10
```

The last statement within the curly brackets will be the value returned by the function.

- `x` is the function argument, in that it is a placeholder we can replace with an actual value when calling the function

Functions live in their own little world

Inside a function, variables that existed in your environment can be used and even changed. However, any changes made, including changing data stored in variables and creating new variables, happens solely within the function. Your environment stays the same.

```
exists("myResult")
```

```
[1] TRUE
```

```
myResult <- 1000
doubleItOutput <- doubleIt(2)
myResult
```

```
[1] 1000
```

```
my_sum <- function(x){
  s <- sum(x)
  n <- length(x)
  result <- s / n
  return(result)
}
```

> Functions are objects, so you can name them and store them

```
my_sum(1:10)
```

```
[1] 5.5
```

If you want to keep the results, name it

```
answer <- my_sum(1:10)
answer
```

```
[1] 5.5
```

```
my_sum <- function(x){
  s <- sum(x)
  n <- length(x)
  results<- list(sum = s, length = n, answer = s / n)
  return(results)
}
```

- You can only return a single object
    - Pack all your outputs into a list

```
my_sum(1:10)
```

```
$sum
[1] 55

$length
[1] 10

$answer
[1] 5.5
```

```r
my_sum <- function(x){
  s <- sum(x)
  n <- length(x)
  results<- list(sum = s, length = n, answer = s / n)
  return(results)
}
```

```r
answer <- my_sum(1:10)
answer$answer
```

```
[1] 5.5
```

```r
answer[['answer']]
```

```
[1] 5.5
```

```r
names(answer)
```

```
[1] "sum"    "length" "answer"
```

```
x <- 1:10
x[3] <- NA
my_sum(x)
```

```
$sum
[1] NA

$length
[1] 10

$answer
[1] NA
```

```
my_sum <- function(x){
  s <- sum(x, na.rm=T)
  n <- length(!is.na(x))
  results <- list("sum" = s, "length" = n, "answer" = s/n)
}
my_sum(x)
```

NO RESULTS!!!!

```r
my_sum <- function(x){
  s <- sum(x, na.rm = T)
  n <- length(!is.na(x))
  results <- list("sum" = s, "length" = n, "answer" = s/n)
  return(results)
}
my_sum(x)
```

```
$sum
[1] 52

$length
[1] 10

$answer
[1] 5.2
```

This is still not right

```r
my_sum <- function(x){
  s <- sum(x, na.rm = T)
   n <- length(!is.na(x))
  results <- list("sum" = s, "length" = n, "answer" = s/n)
  return(results)
}
```

```r
my_sum <- function(x){
  s <- sum(x, na.rm = T)
   n <- sum(!is.na(x))
  results <- list("sum" = s, "length" = n, "answer" = s/n)
  return(results)
}
my_sum(x)
```

```
$sum
[1] 52

$length
[1] 9

$answer
[1] 5.777778
```

```
my_sum <- function(x){
  s <- sum(x, na.rm = T)
  n <- sum(!is.na(x))
  results <- list("sum" = s, "length" = n, "answer" = s/n)
  return(results)
}
```

- This function will always remove missing values
- We don't have explicit control
    - Maybe not a good thing

```r
my_sum <- function(x, remove_missing = TRUE){
  s <- sum(x, na.rm = T)
  n <- sum(!is.na(x))
  results <- list("sum" = s, "length" = n, "answer" = s/n)
  return(results)
}
```

```
my_sum <- function(x, remove_missing = TRUE){
   if(remove_missing){
     x <- x[!is.na(x)]
   }
   s <- sum(x)
   n <- length(x)
  results <- list("sum" = s, "length" = n, "answer" = s/n)
  return(results)
}
my_sum(x)
```

```
$sum
[1] 52

$length
[1] 9

$answer
[1] 5.777778
```

```r
my_sum <- function(x, remove_missing = TRUE){
  if(remove_missing){
    x <- x[!is.na(x)]
  }
  s <- sum(x)
  n <- length(x)
  results <- list("sum" = s, "length" = n, "answer" = s/n, "nmiss" = sum(is.na(x)))
  return(results)
}
my_sum(x)
```

```
$sum
[1] 52

$length
[1] 9

$answer
[1] 5.777778

$nmiss
[1] 0
```

OOPS!!

```
my_sum <- function(x, remove_missing = TRUE){
  nmiss <- sum(is.na(x))
  if(remove_missing){
    x <- x[!is.na(x)]
  }
  s <- sum(x)
  n <- length(x)
  results <- list("sum" = s, "length" = n, "answer" = s/n, "nmiss" = sum(is.na(x)))
  return(results)
}
my_sum(x)
```

```
$sum
[1] 52

$length
[1] 9

$answer
[1] 5.777778

$nmiss
[1] 0
```

NOT QUITE!

```
my_sum <- function(x, remove_missing = TRUE){
  nmiss <- sum(is.na(x))
  if(remove_missing){
    x <- x[!is.na(x)]
  }
  s <- sum(x)
  n <- length(x)
  results <- list("sum" = s, "length" = n, "answer" = s/n, "nmiss" = nmiss)
  return(results)
}
my_sum(x)
```

```
$sum
[1] 52

$length
[1] 9

$answer
[1] 5.777778

$nmiss
[1] 1
```

```
my_sum(x, remove_missing = F)
```

```
$sum
[1] NA

$length
[1] 10

$answer
[1] NA

$nmiss
[1] 1
```

```
my_summary <- function(d){

}
```

```
my_summary <- function(d){
  require(tidyverse) #<
}
```

- You can be explicit in your package requirements
- `require` will try to activate the package if it's installed

```
my_summary <- function(d){
  require(tidyverse)
  summary_cts <- d %>%
    summarize_if(is.numeric, list("mean" = ~mean(x, na.rm=T),
                                  "median" = ~median(x, na.rm=T),
                                  'sd' = ~sd(x, na.rm=T),
                                  'nmiss' = ~sum(is.na(x))))
  return(list("cts" = summary_cts))
}
my_summary(iris)
```

```
$cts
  Sepal.Length_mean Sepal.Width_mean Petal.Length_mean Petal.Width_mean
1          5.777778         5.777778          5.777778         5.777778
  Sepal.Length_median Sepal.Width_median Petal.Length_median
1                   6                  6                   6
  Petal.Width_median Sepal.Length_sd Sepal.Width_sd Petal.Length_sd
1                  6        3.073181       3.073181        3.073181
  Petal.Width_sd Sepal.Length_nmiss Sepal.Width_nmiss Petal.Length_nmiss
1       3.073181                  1                 1                  1
  Petal.Width_nmiss
1                 1
```

```r
my_summary <- function(d){
  require(tidyverse)
  summary_cts <- d %>%
    summarize_if(is.numeric, list("mean" = ~mean(x, na.rm=T),
                                  "median" = ~median(x, na.rm=T),
                                  'sd' = ~sd(x, na.rm=T),
                                  'nmiss' = ~sum(is.na(x)))) %>%
    gather(variable, value) %>%
    separate(variable, c("variable","stat"), sep='_') %>%
    spread(stat, value)
  return(list("cts" = summary_cts))
}
my_summary(iris)
```

```
$cts
      variable     mean median nmiss       sd
1 Petal.Length 5.777778      6     1 3.073181
2  Petal.Width 5.777778      6     1 3.073181
3 Sepal.Length 5.777778      6     1 3.073181
4  Sepal.Width 5.777778      6     1 3.073181
```

HUH!!!

```
my_summary <- function(d){
  require(tidyverse)
  summary_cts <- d %>%
    summarize_if(is.numeric, list("mean" = ~mean(x, na.rm=T),
                                  "median" = ~median(x, na.rm=T),
                                  'sd' = ~sd(x, na.rm=T),
                                  'nmiss' = ~sum(is.na(x)))) %>%
    gather(variable, value) %>%
    separate(variable, c("variable","stat"), sep='_') %>%
    spread(stat, value)
  return(list("cts" = summary_cts))
}
my_summary(iris)
```

```
$cts
      variable     mean median nmiss       sd
1 Petal.Length 5.777778      6     1 3.073181
2  Petal.Width 5.777778      6     1 3.073181
3 Sepal.Length 5.777778      6     1 3.073181
4  Sepal.Width 5.777778      6     1 3.073181
```

- Function can't find x inside the function
- SO it pulls x from the environment

```
my_summary <- function(d){
  require(tidyverse)
  summary_cts <- d %>%
    summarize_if(is.numeric, list("mean" = ~mean(., na.rm=T),
                                  "median" = ~median(., na.rm=T),
                                  'sd' = ~sd(., na.rm=T),
                                  'nmiss' = ~sum(is.na(.)))) %>%
    gather(variable, value) %>%
    separate(variable, c("variable","stat"), sep='_') %>%
    spread(stat, value)
  return(list("cts" = summary_cts))
}
my_summary(iris)
```

```
$cts
      variable     mean median nmiss        sd
1 Petal.Length 3.758000   4.35     0 1.7652982
2  Petal.Width 1.199333   1.30     0 0.7622377
3 Sepal.Length 5.843333   5.80     0 0.8280661
4  Sepal.Width 3.057333   3.00     0 0.4358663
```

```
my_summary <- function(d){
  require(tidyverse)
  summary_cts <- d %>%
    summarize_if(is.numeric, list("mean" = ~mean(., na.rm=T),
                                  "median" = ~median(., na.rm=T),
                                  'sd' = ~sd(., na.rm=T),
                                  'nmiss' = ~sum(is.na(.)))) %>%
    gather(variable, value) %>%
    separate(variable, c("variable","stat"), sep='_') %>%
    spread(stat, value) %>%
    select(variable, nmiss, everything())
  return(list("cts" = summary_cts))
}
my_summary(iris)
```

```
$cts
      variable nmiss     mean median        sd
1 Petal.Length     0 3.758000   4.35 1.7652982
2  Petal.Width     0 1.199333   1.30 0.7622377
3 Sepal.Length     0 5.843333   5.80 0.8280661
4  Sepal.Width     0 3.057333   3.00 0.4358663
```

```
my_summary <- function(d){
  require(tidyverse)
  summary_cts <- d %>%
    summarize_if(is.numeric, list("mean" = ~mean(., na.rm=T),
                                  "median" = ~median(., na.rm=T),
                                  'sd' = ~sd(., na.rm=T),
                                  'nmiss' = ~sum(is.na(.)))) %>%
    gather(variable, value) %>%
    separate(variable, c("variable","stat"), sep='_') %>%
    spread(stat, value) %>%
    select(variable, nmiss, everything())
  summary_cat <- d %>%
    summarise_if(is.factor, list('nmiss' = ~sum(is.na(.)),
                                 'ncat' = ~length(unique(.)),
                                 'categories' = ~paste(sort(unique(levels(.))), collapse=', '))
                 )
  return(list("cts" = summary_cts,
              "cat" = summary_cat))
}
my_summary(iris)
```

```
$cts
      variable nmiss     mean median        sd
1 Petal.Length     0 3.758000   4.35 1.7652982
2  Petal.Width     0 1.199333   1.30 0.7622377
3 Sepal.Length     0 5.843333   5.80 0.8280661
4  Sepal.Width     0 3.057333   3.00 0.4358663

$cat
  nmiss ncat                   categories
1     0    3 setosa, versicolor, virginica
```

```
my_summary <- function(d){
  require(tidyverse)
  if(!is.data.frame(d)){
    stop("Input must be a data.frame")
  }
  summary_cts <- d %>%
    summarize_if(is.numeric, list("mean" = ~mean(., na.rm=T),
                                  "median" = ~median(., na.rm=T),
                                  'sd' = ~sd(., na.rm=T),
                                  'nmiss' = ~sum(is.na(.)))) %>%
    gather(variable, value) %>%
    separate(variable, c("variable","stat"), sep='_') %>%
    spread(stat, value) %>%
    select(variable, nmiss, everything())
  summary_cat <- d %>%
    summarise_if(is.factor, list('nmiss' = ~sum(is.na(.)),
                                 'ncat' = ~length(unique(.)),
                                 'categories' = ~paste(sort(unique(levels(.))), collapse=', '))
                 )
  return(list("cts" = summary_cts,
              "cat" = summary_cat))
}
my_summary(x)
```

purrr-ing

purrr

Cheatsheet: https://github.com/rstudio/cheatsheets/raw/master/purrr.pdf

Tutorial: https://jennybc.github.io/purrr-tutorial/

The package `purrr` gives us the `map` function and derivatives

- `map` applies a function iteratively to each element of a list or vector

```
datas <- list('cars' = mtcars, 'iris' = iris, 'diamonds'= diamonds)
map(datas, my_summary)
```

```
$cars
$cars$cts
# A tibble: 11 x 5
   variable nmiss    mean median      sd
   <chr>    <dbl>   <dbl>  <dbl>   <dbl>
 1 am           0   0.406   0      0.499
 2 carb         0   2.81    2      1.62
 3 cyl          0   6.19    6      1.79
 4 disp         0 231.    196.   124.
 5 drat         0   3.60    3.70   0.535
 6 gear         0   3.69    4      0.738
 7 hp           0 147.    123     68.6
 8 mpg          0  20.1    19.2    6.03
 9 qsec         0  17.8    17.7    1.79
10 vs           0   0.438   0      0.504
11 wt           0   3.22    3.32   0.978

$cars$cat
# A tibble: 1 x 0


$iris
$iris$cts
      variable nmiss     mean median        sd
1 Petal.Length     0 3.758000   4.35 1.7652982
2  Petal.Width     0 1.199333   1.30 0.7622377
3 Sepal.Length     0 5.843333   5.80 0.8280661
4  Sepal.Width     0 3.057333   3.00 0.4358663

$iris$cat
  nmiss ncat                  categories
```