Data Science for Operational Researchers Using R Online

5. Introduction to Functionals using purrr

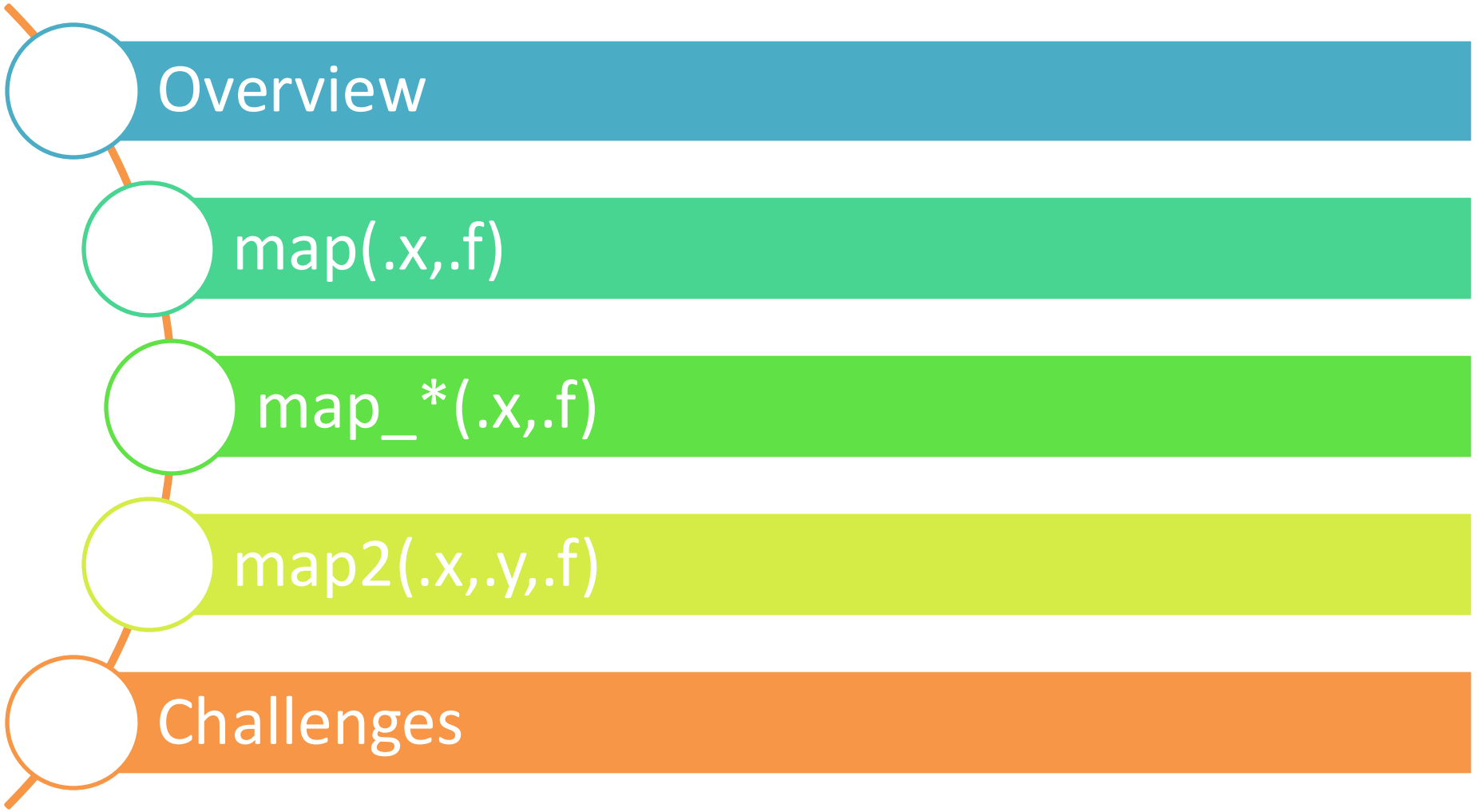
Prof. Jim Duggan,
School of Computer Science
University of Galway.

https://github.com/JimDuggan/explore_or

R has numerous ways to iterate over elements of a list (or vector), and Hadley Wickham aimed to improve on and standardise that experience with the `purrr` package.

— Jared P. Lander (Lander, 2017)

Overview



1. Overview

- The tidyverse package purrr provides a comprehensive set of functions that can be used to iterate over data structures (e.g. vectors, lists, data frames, tibbles)
- These functions are functionals, and typically take:
  - Input data (.x) to be iterated over
  - A function (usually anonymous) to process each data element
- The functions return a data object that is the same size as the input.

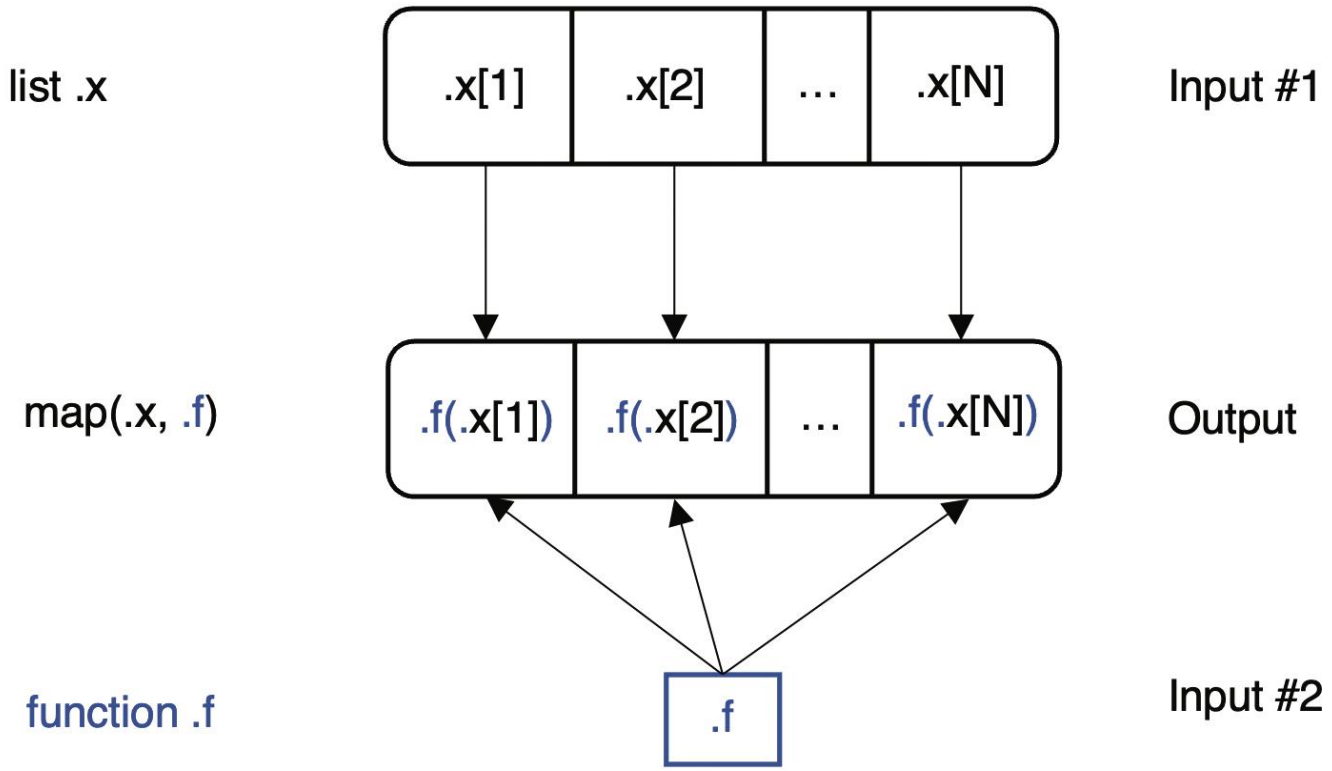
2. map(.x,.f)

- map() provides a mechanism to iterate over an input list or a vector.
- It applies a function to each input element, and returns the result within a list that is exactly the same length as the input.
- We call these functions functionals, as they accept a function as an argument, and use that function in order to generate the output

General format of map(.x,.f)

The general format of this function is map(.x, .f), where:

- .x is a list, or an atomic vector. If .x has named elements, the return value will preserve those names.
- f can be a function, formula, or vector.



Example 1 - function

```
library(purrr)

o1 <- purrr::map(c(1,2,3,2),function(x)x^2)
str(o1)
#> List of 4
#>  $ : num 1
#>  $ : num 4
#>  $ : num 9
#>  $ : num 4
```

Example 2 – formula (.x default argument)

```
o2 <- purrr::map(c(1,2,3,2),~.x^2)
str(o2)
#> List of 4
#>  $ : num 1
#>  $ : num 4
#>  $ : num 9
#>  $ : num 4
```

Example 3 – formula (. default argument)

```
o3 <- purrr::map(c(1,2,3,2),~.^2)
str(o3)
#> List of 4
#>  $ : num 1
#>  $ : num 4
#>  $ : num 9
#>  $ : num 4
```

3. Additional map_* functions

- While map() will always return a list, there may be circumstances where different formats are required, for example, an atomic vector.
- To address this purrr provides a set of additional functions that specify the result type.

map_* function	Description
map_dbl()	returns an atomic vector of type double
map_chr()	returns an atomic vector of type character
map_lgl()	which returns an atomic vector of type logical
map_int()	which returns an atomic vector of type integer
map_df()	returns a data frame or tibble

map_dbl()

```
library(dplyr)
library(purrr)
mtcars %>%
  dplyr::select(mpg,cyl,disp) %>%
  purrr::map_dbl(mean)
#>      mpg      cyl     disp
#>   20.091    6.188 230.722
```

map_chr()

```
library(repurrrsive)
library(purrr)
sw_films %>%
  purrr::map_chr(~.x$director) %>%
  unique()
#> [1] "George Lucas"     "Richard Marquand" "Irvin Kershner"
#> [4] "J. J. Abrams"
```

Note the tag operator ($) extracts the element of a list. It can also be used on data frames and tibble to extract a variable and all its values. On data frames, the pull() function has a similar purpose.

pull() and $

```
> mpg$cyl
  [1] 4 4 4 4 6 6 6 4 4 4 4 6 6 6 6 6 8 8 8 8 8 8 8 8 8 8 8 8 8 4 4 6 6 6 4 6 6 6 6 6 6 6 6 6 6 6
 [52] 6 8 8 8 8 8 6 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 6 6 6 6 8 8 6 6 8 8 8 8 8 6 6 6 6 8 8 8 8 4 4 4
[103] 4 4 4 4 4 4 4 4 4 4 6 6 6 4 4 4 4 6 6 6 6 6 8 8 8 8 8 8 8 8 8 8 8 8 6 6 8 8 4 4 4 4 6 6 6 6 6 6 6
[154] 8 6 6 6 6 8 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 6 6 6 8 4 4 4 4 6 6 6 4 4 4 4 6 6 6 4 4 4 4 4 8 8 4 4 4 6
[205] 6 6 6 4 4 4 4 6 4 4 4 4 4 5 5 6 6 4 4 4 4 5 5 4 4 4 4 6 6 6
>
> mpg %>% pull(cyl)
  [1] 4 4 4 4 6 6 6 4 4 4 4 6 6 6 6 6 8 8 8 8 8 8 8 8 8 8 8 8 8 4 4 6 6 6 4 6 6 6 6 6 6 6 6 6 6 6
 [52] 6 8 8 8 8 8 6 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 6 6 6 6 8 8 6 6 8 8 8 8 8 6 6 6 6 8 8 8 8 4 4 4
[103] 4 4 4 4 4 4 4 4 4 4 6 6 6 4 4 4 4 6 6 6 6 6 8 8 8 8 8 8 8 8 8 8 8 8 6 6 8 8 4 4 4 4 6 6 6 6 6 6 6
[154] 8 6 6 6 6 8 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 6 6 6 8 4 4 4 4 6 6 6 4 4 4 4 6 6 6 4 4 4 4 4 8 8 4 4 4 6
[205] 6 6 6 4 4 4 4 6 4 4 4 4 4 5 5 6 6 4 4 4 4 5 5 4 4 4 4 6 6 6
```

map_lgl()

```
library(ggplot2)
library(purrr)
library(dplyr)
mpg %>%
  dplyr::select(manufacturer:cyl) %>%
  purrr::map_lgl(function(x)is.numeric(x))
#> manufacturer        model        displ         year          cyl
#>        FALSE        FALSE         TRUE         TRUE         TRUE
```

map_int()

```
library(ggplot2)
library(dplyr)
library(purrr)


mpg %>%
  dplyr::select(displ,cty,hwy) %>%
  purrr::map_int(~sum(.x>mean(.x)))
#> displ   cty   hwy
#>   107   118   129
```

map_df()

```
library(repurrrsive)
library(purrr)
library(dplyr)

sw_films %>%
  purrr::map_df(~tibble(ID=.x$episode_id,
                        Title=.x$title,
                        Director=.x$director,
                        ReleaseDate=as.Date(.x$release_date))) %>%
  dplyr::arrange(ID)
```

Output from map_df()

```
#> # A tibble: 7 x 4
#>      ID Title                      Director         ReleaseDate
#>   <int> <chr>                      <chr>            <date>
#> 1     1 The Phantom Menace         George Lucas     1999-05-19
#> 2     2 Attack of the Clones       George Lucas     2002-05-16
#> 3     3 Revenge of the Sith        George Lucas     2005-05-19
#> 4     4 A New Hope                 George Lucas     1977-05-25
#> 5     5 The Empire Strikes Back    Irvin Kershner   1980-05-17
#> 6     6 Return of the Jedi         Richard Marquand 1983-05-25
#> 7     7 The Force Awakens          J. J. Abrams     2015-12-11
```

4. map2(.x,.y,.f)

The function map2() allows for two inputs, and these are then represented as arguments by .x and .y.

```
means <- c(10,20,30)
sds   <- c(2,4,7)

purrr::map2(means,sds,~rnorm(5,.x,.y)) %>% str()
#> List of 3
#>  $ : num [1:5] 10.42 6.78 10.54 12.05 7.1
#>  $ : num [1:5] 28 21.6 10.4 31.1 18.2
#>  $ : num [1:5] 32 35.3 30.6 25.5 28.3
```

pmap() is even more flexible.

pmap() can take a list containing any number of arguments, and process these elements within the function using the symbols ..1, ..2, etc, which represent the first, second, and additional arguments.

```
params <- list(means = c(10,20,30),
               sds   = c(2,4,7),
               n     = c(4,5,6))
```

```
purrr::pmap(params,
            ~rnorm(n    = ..3,
                   mean = ..1,
                   sd   = ..2)) %>%
        str()
#> List of 3
#>  $ : num [1:4] 12.68 10.27 5.06 11.31
#>  $ : num [1:5] 20.1 18.5 21 14.9 20.2
#>  $ : num [1:6] 28 25 34.4 35.6 36.7 ...
```

5. Challenges

- Use a purrr function to generate the following (mean)

```
# Create the list that will be processed by lapply
l1 <- list(a=1:5,b=100:200,c=1000:5000)
```

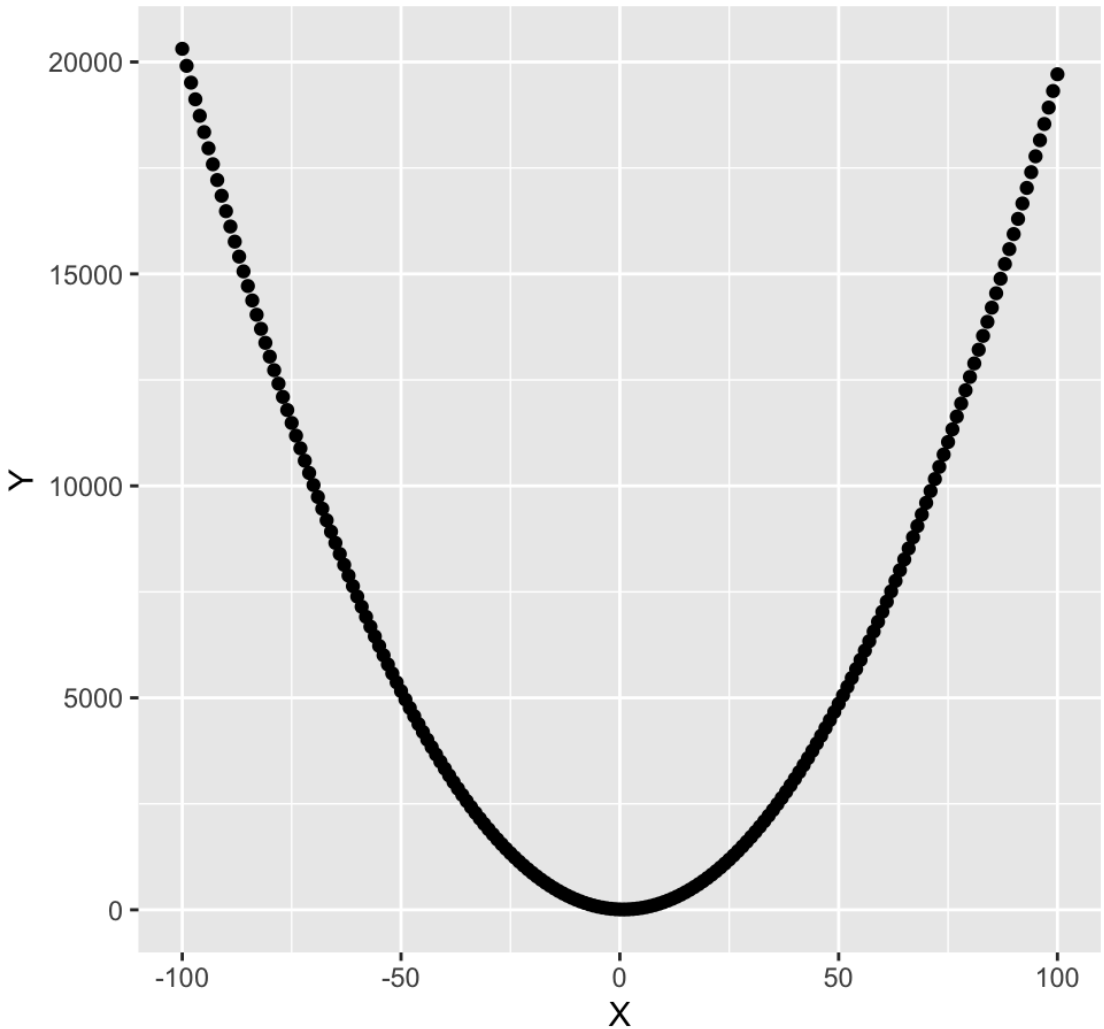
```
# The result is stored in ans
ans
#>    a    b    c
#>    3  150 3000
```

```r
library(purrr)
library(ggplot2)

a <- 2; b <- -3; c <- 10

res <- tibble(X=seq(-100,100),
              Y=map_dbl(X,~a*.x^2+b*.x+c))

ggplot(res,aes(x=X,y=Y))+geom_point()
```



```
res1 <- map_df(seq(-100,100),~tibble(X=.x,
                                     Y=a*.x^2+b*.x+c))
```

```
ggplot(res1,aes(x=X,y=Y))+
  geom_point(colour="blue")
```