



purrr Package

Luka Sandmayr, Adalbert Istvan Tiabiassy

SE Statistics, Visualization and More Using “R”
Mai 7th, 2024

Installation

Install the **tidyverse** package

```
install.packages("tidyverse")
```

Install just **purrr**

```
install.packages("purrr")
```

Install development version directly from Github

```
devtools::install_github("tidyverse/purrr")
```

Introduction

- Subpackage of tidyverse
- Emphasis on iteration and mapping.
- Functions like `map()`, `map2()`, and `pmap()` for applying functions to data structures
- Functions for filtering, reducing, and modifying data

Map

map(.x, .f, ...):

- Applies a function (.f) to all elements of the input vector or list
- Returns with a list

```
fst_list <- as.list(1:10)  
fst_list_mapped <- map(fst_list, sqrt)
```

Map

map(.x, .f, ...):

- Applies a function (.f) to all elements of the input vector or list
- Returns with a list

```
fst_list <- as.list(1:10)  
fst_list_mapped <- map(fst_list, sqrt)
```

| fst_list | list [10] | List of length 10 |
|----------|-------------|-------------------|
| [[1]] | integer [1] | 1 |
| [[2]] | integer [1] | 2 |
| [[3]] | integer [1] | 3 |
| [[4]] | integer [1] | 4 |
| [[5]] | integer [1] | 5 |
| [[6]] | integer [1] | 6 |
| [[7]] | integer [1] | 7 |
| [[8]] | integer [1] | 8 |
| [[9]] | integer [1] | 9 |
| [[10]] | integer [1] | 10 |

Map

map(.x, .f, ...):

- Applies a function (.f) to all elements of the input vector or list
- Returns with a list

```
fst_list <- as.list(1:10)  
fst_list_mapped <- map(fst_list, sqrt)
```

| | | | | | | |
|----------|-------------|-------------------|--|-----------------|------------|-------------------|
| fst_list | list [10] | List of length 10 | | fst_list_mapped | list [10] | List of length 10 |
| [[1]] | integer [1] | 1 | | [[1]] | double [1] | 1 |
| [[2]] | integer [1] | 2 | | [[2]] | double [1] | 1.414214 |
| [[3]] | integer [1] | 3 | | [[3]] | double [1] | 1.732051 |
| [[4]] | integer [1] | 4 | | [[4]] | double [1] | 2 |
| [[5]] | integer [1] | 5 | | [[5]] | double [1] | 2.236068 |
| [[6]] | integer [1] | 6 | | [[6]] | double [1] | 2.44949 |
| [[7]] | integer [1] | 7 | | [[7]] | double [1] | 2.645751 |
| [[8]] | integer [1] | 8 | | [[8]] | double [1] | 2.828427 |
| [[9]] | integer [1] | 9 | | [[9]] | double [1] | 3 |
| [[10]] | integer [1] | 10 | | [[10]] | double [1] | 3.162278 |

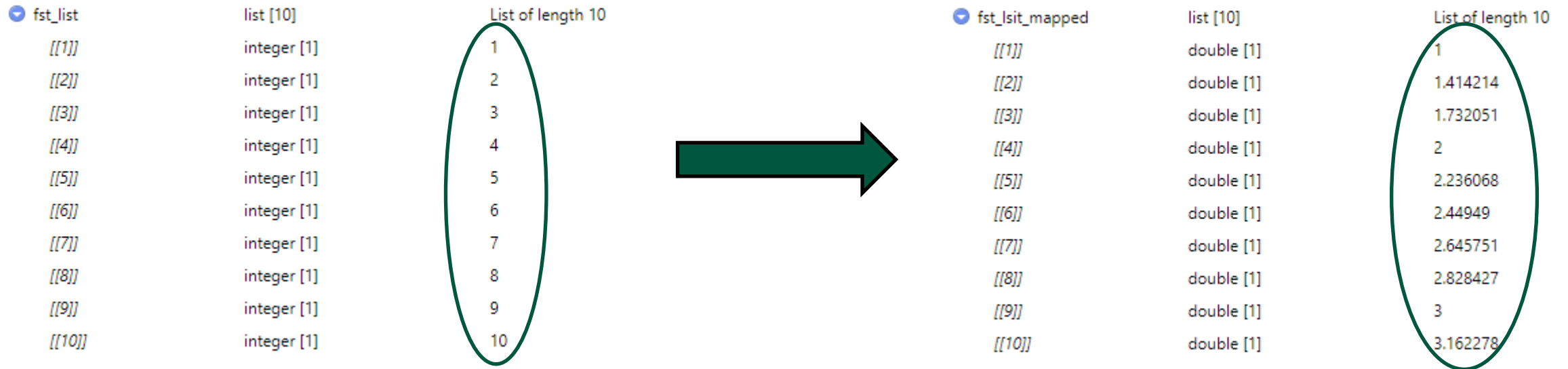


Map

map(.x, .f, ...):

- Applies a function (.f) to all elements of the input vector or list
- Returns with a list

```
fst_list <- as.list(1:10)  
fst_list_mapped <- map(fst_list, sqrt)
```



Map

map_...(.x, .f, ...) - returns with the given datatype

- **map_dbl(.x, .f, ...)** - double
- **map_int(.x, .f, ...)** - integer
- **map_chr(.x, .f, ...)** - character
- **map_lgl(.x, .f, ...)** - boolean/logical
- **map_vec(.x, .f, ...)** - vector
- **map_df(.x, .f, ...)** - dataframe

Map

Example:

```
data("morley")
scnd_list <- list(1:5,
                 FALSE,
                 1,
                 3.14159263,
                 "example",
                 morley
                )
```

```
class(scnd_list)
```

```
[1] "list"
```

Map

Example:

```
data("morley")
scnd_list <- list(1:5,
                 FALSE,
                 1,
                 3.14159263,
                 "example",
                 morley
                )
```

```
class(scnd_list)
```

```
[1] "list"
```



```
map(scnd_list, class)
```

```
[[1]]
[1] "integer"

[[2]]
[1] "logical"

[[3]]
[1] "numeric"

[[4]]
[1] "numeric"

[[5]]
[1] "character"

[[6]]
[1] "data.frame"
```

Map

Example:

```
data("morley")
scnd_list <- list(1:5,
                 FALSE,
                 1,
                 3.14159263,
                 "example",
                 morley
                )
```

```
class(scnd_list)
```

```
[1] "list"
```



```
map(scnd_list, class)
```

```
[[1]]
[1] "integer"

[[2]]
[1] "logical"

[[3]]
[1] "numeric"

[[4]]
[1] "numeric"

[[5]]
[1] "character"

[[6]]
[1] "data.frame"
```

```
map_chr(scnd_list, class)
```

```
[1] "integer" "logical" "numeric" "numeric" "character" "data.frame"
```

Map

Example:

```
data("morley")
glimpse(morley)
```

Rows: 100
Columns: 3
\$ Expt <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3,
3, ...
\$ Run <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 1, 2,
3, 4...
\$ Speed <int> 850, 740, 900, 1070, 930, 850, 950, 980, 980, 880, 1000, 980, 930,
650, 760, 810, 1000, 1000, 960, 960, 960, 940, 960, 940, 880, 800, 850, 880, 900,
840...

```
morley_dbl <- morley %>%  
  map_df(as.character)
```

```
glimpse(morley_dbl)
```

Rows: 100
Columns: 3
\$ Expt <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1",
"1", "1", "1", "1", "1", "1", "1", "1", "2", "2", "2", "2", "2", "2", "2", "2",
"2", "2...
\$ Run <chr> "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13",
"14", "15", "16", "17", "18", "19", "20", "1", "2", "3", "4", "5", "6", "7", "8",
"...
\$ Speed <chr> "850", "740", "900", "1070", "930", "850", "950", "980", "980",
"880", "1000", "980", "930", "650", "760", "810", "1000", "1000", "960", "960",
"960", "...

Anonymous functions

Function for "single-use":

- Self-written functions
- Arguments for other functions (e.g. for map...)

```
newFunc <- function(x){  
  (x + 1)^3  
}  
map(X, newFunc)
```

Anonymous functions

Function for "single-use":

- Self-written functions
- Arguments for other functions (e.g. for map...)

```
newFunc <- function(x){  
  (x + 1)^3  
}  
map(X, newFunc)  
  
map(X, function(x) (x + 1)^3)  
map(X, \(x) (x + 1)^3)
```

Map – multiple args

The needed function has 2+ arguments

Example:

besselJ(x, nu) have 2 arguments

```
map(x, besselJ)
```

```
Error in `map()`:
```

```
i In index: 1.
```

```
caused by error in `.f()`:
```

```
! argument "nu" is missing, with no default
```

```
Backtrace:
```

1. `purrr::map(x, besselJ)`
2. `purrr::map_("list", .x, .f, ..., .progress = .progress)`
6. `base (local) .f(.x[[i]], ...)`

```
Error in map(x, besselJ) :
```

```
Caused by error in `.f()`:
```

```
! argument "nu" is missing, with no default
```

Show Traceback

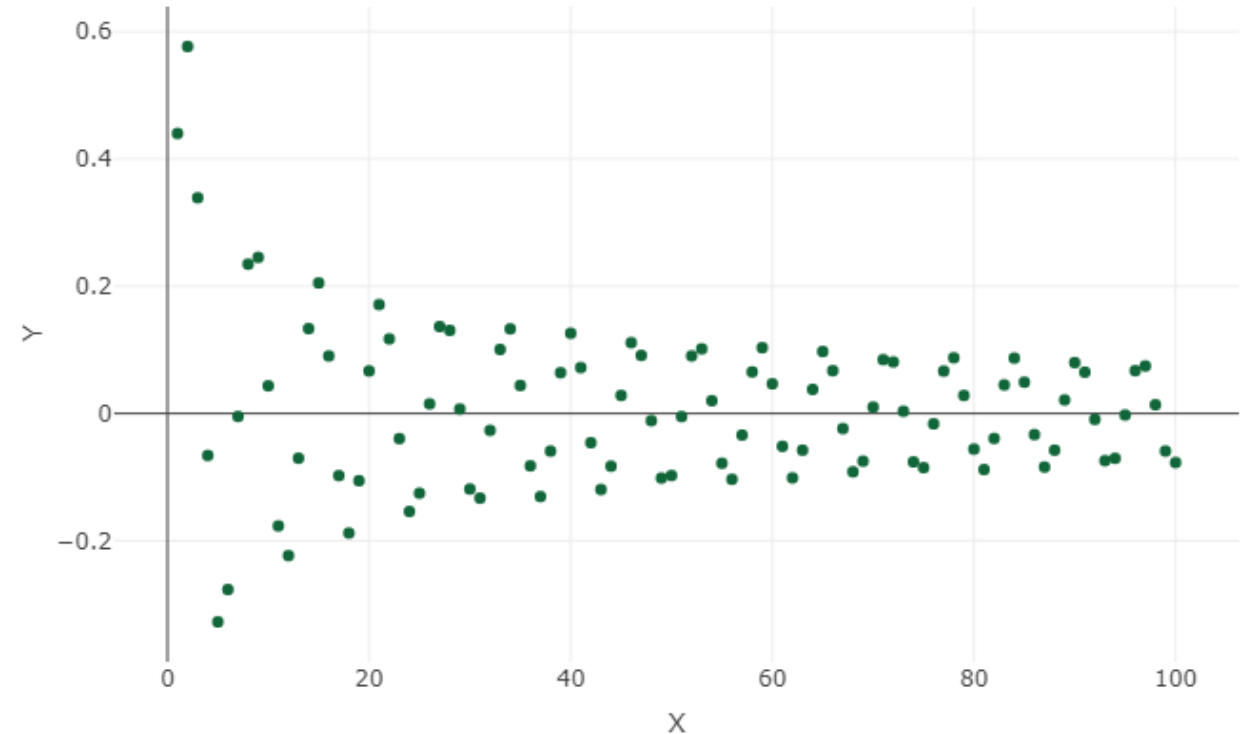
Map – multiple args

The needed function has 2+ arguments

Example:

besselJ(x, nu) have 2 arguments

```
BesselJ <- map(X, besselJ, nu = 1)
```



Map – multiple args

The needed function has 2+ arguments

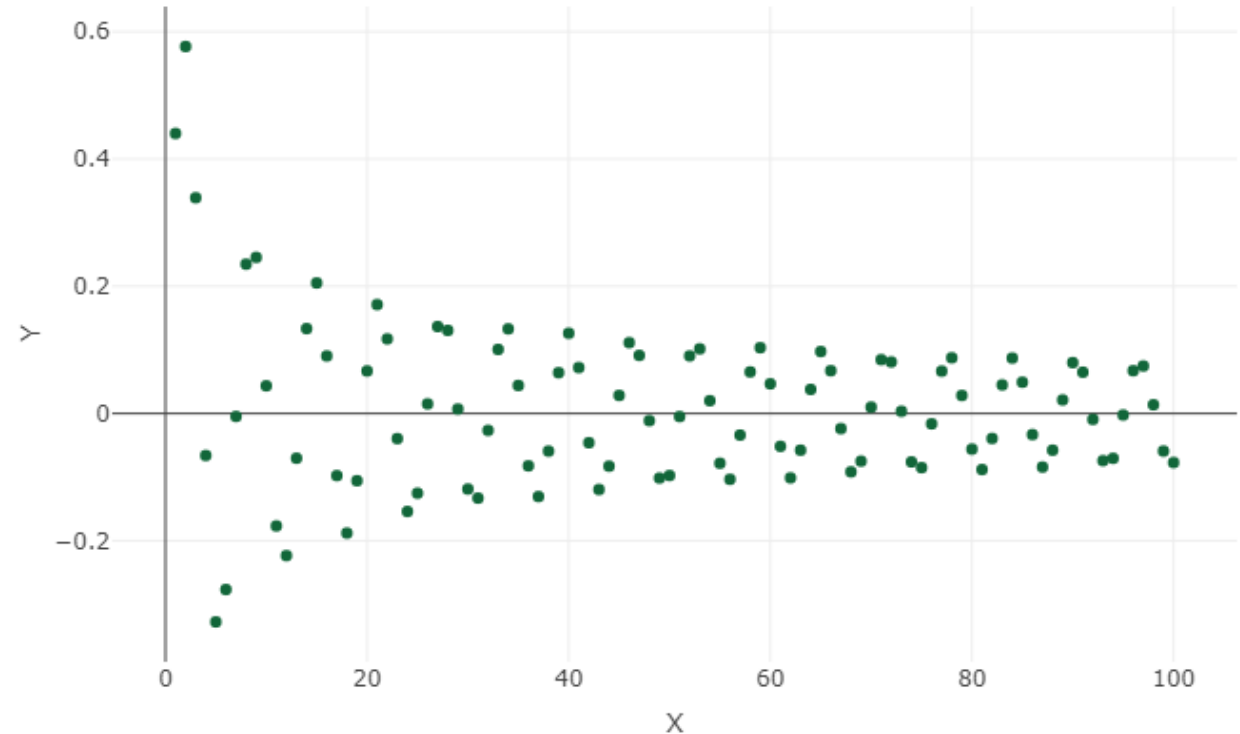
Example:

besselJ(x, nu) have 2 arguments

```
BesselJ <- map(X, besselJ, nu = 1)
```



```
BesselJ_2 <- map(X, \(x) besselJ(x, 1))
```



Map – map_if()

map_if(.x, .p, .f ...):

- Returns with a list
- .p refers to a predicate function – which elements of .x need to be converted using .f

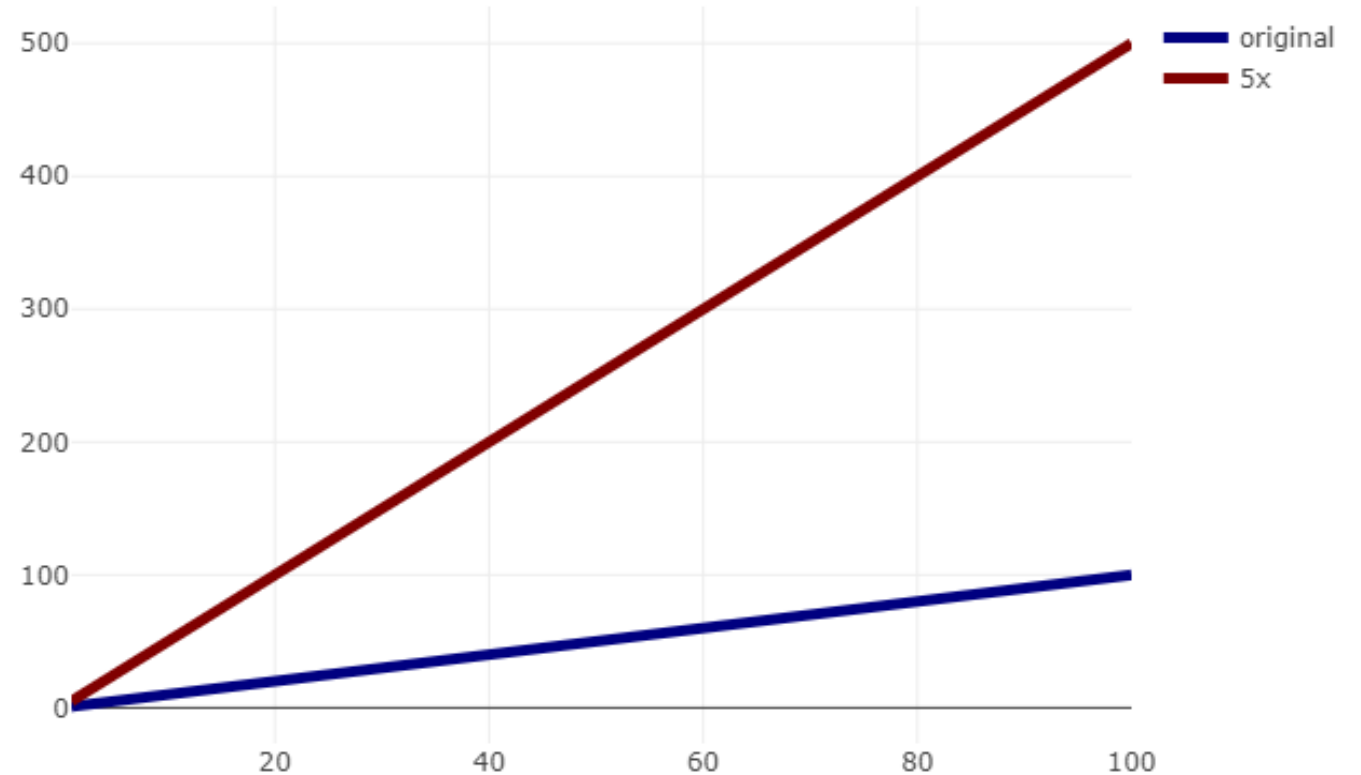
Map – map_if()

Example:

```
X <- 1:100
Y <- 1:100

p <- plot_ly() %>%
  add_lines(X,Y,
            name = "original",
            line = list(width = 5, color = "navy")) %>%
  add_lines(X, map(Y, \ (y) y*5),
            name = "5x",
            line = list(width = 5, color = "maroon"))

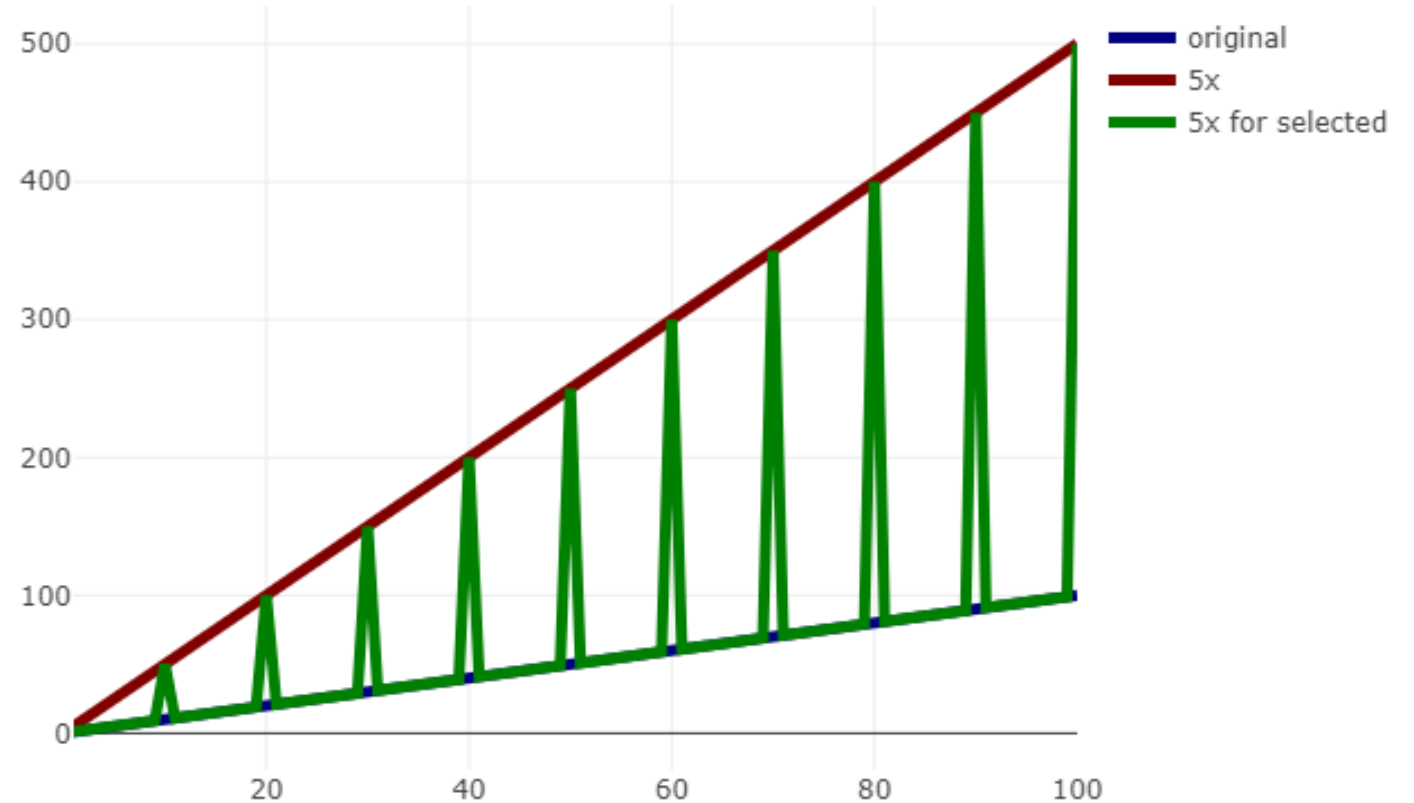
p
```



Map – map_if()

Example:

```
p2 <- plot_ly() %>%  
  add_lines(x, y,  
            name = "original",  
            line = list(width = 5, color = "navy")) %>%  
  add_lines(x, map(Y, \ (y) y*5),  
            name = "5x",  
            line = list(width = 5, color = "maroon")) %>%  
  add_lines(x, map_if(Y, ~.x %% 10 == 0, \ (y) y*5 ),  
            name = "5x for selected",  
            line = list(width = 5, color = "green"))  
p2
```



Map – map_at()

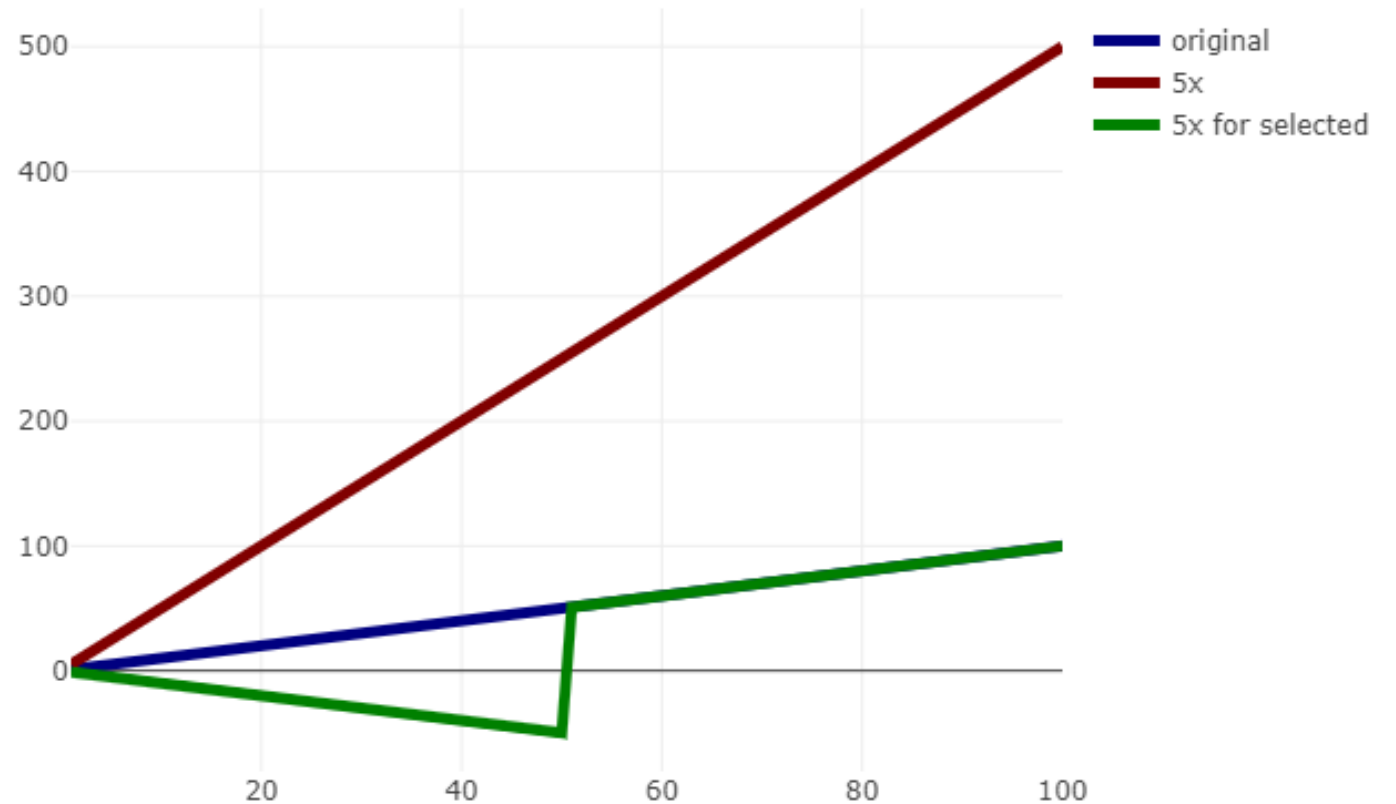
map_at(.x, .at, .f ...):

- Returns with a list
- .at refers to the positions of the elements of .x which need to be converted using .f

Map – map_at()

Example:

```
p2 <- plot_ly() %>%  
  add_lines(x,Y,  
    name = "original",  
    line = list(width = 5, color = "navy")) %>%  
  add_lines(x, map(Y,\(y) y*5),  
    name = "5x",  
    line = list(width = 5, color = "maroon")) %>%  
  add_lines(x, map_at(Y, c(1:50), \(y) -y ),  
    name = "5x for selected",  
    line = list(width = 5, color = "green"))  
p2
```



Map – map_depth()

map_depth(.x, .depth, .f, ...):

- Returns with a list
- .depth specifies the depth of the called map(.x, .f, ...) function
- **map_depth**(x, 0, f) = f(x)
- **map_depth**(x, 1, f) = map(x, f)
- **map_depth**(x, 2, f) = map(x, \ (y) map(y, f))

2+ lists for map

map2(.x, .y, .f, ...) and **pmap**(.l, .f,...)

- Both of them return with a list
- Apply .f to the pair or the lists of listelements
- All the variants are available (map2_int, pmap_dbl, etc.)

2+ lists for map

Example:

```
vec1 <- 1:3  
vec2 <- 5:7  
  
map2(vec1, vec2, sum)
```

```
[[1]]  
[1] 6  
  
[[2]]  
[1] 8  
  
[[3]]  
[1] 10
```

```
vec1 <- list(1:3)  
vec2 <- list(5:7)  
vec3 <- list(8:11)  
  
pmap(list(vec1,vec2,vec3) , \ (x,y,z) paste(as.character(x),  
                                           as.character(y),  
                                           as.character(z)))
```

```
[[1]]  
[1] "1 5 8" "2 6 9" "3 7 10" "1 5 11"
```

Exercises

1. Create a sequence from 1 to 4π , and calculate the sinus of all the elements. Plot them! What happens if you take the cosine of the x, and plot it in that way? And further, if you take the tangent of the x, and after that the cosine? (Use anonymous functions...)
2. Create a list which includes the previous numbers, but they need to be characters!
3. Create a list with 100 random variables with mean=0.5! Multiply the numbers with 5, which are below 0.5! After take the square root of every odd element!

Transform Lists - Modify

map(): returns a fixed object type

- map(): list
- map_int(): integer
- Etc.

modify(): returns same output type as the input type

Iris flower data set

Three Species:

- setosa, virginica, versicolor

50 samples from each species

Four features

- Length and width of the sepals
- Length and width of the petals



| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|--------------|-------------|--------------|-------------|---------|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |

Modify

`modify(.x, .f, ...)`: Apply a function to each element

| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|--------------|-------------|--------------|-------------|---------|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4 | 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5 | 5.0 | 3.6 | 1.4 | 0.2 | setosa |

`modify(iris, ~.+2)`

| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|--------------|-------------|--------------|-------------|---------|
| 1 | 7.1 | 5.5 | 3.4 | 2.2 | NA |
| 2 | 6.9 | 5.0 | 3.4 | 2.2 | NA |
| 3 | 6.7 | 5.2 | 3.3 | 2.2 | NA |
| 4 | 6.6 | 5.1 | 3.5 | 2.2 | NA |
| 5 | 7.0 | 5.6 | 3.4 | 2.2 | NA |

```
> class(modify(iris, ~.+2))
```

```
[1] "data.frame"
```

```
Warnmeldung:
```

```
In Ops.factor(., 2) : '+' ist nicht sinnvoll
```

```
> class(map(iris, ~.+2))
```

```
[1] "list"
```

```
Warnmeldung:
```

```
In Ops.factor(., 2) : '+' ist nicht sinnvoll
```

Modify

`modify_at(.x, .at, .f, ...)`: Apply a function to selected elements
`map_at(.x, .at, .f, ...)`

```
modify_at(iris, "Sepal.Width", ~.+2)
```

| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|--------------|-------------|--------------|-------------|---------|
| 1 | 5.1 | 5.5 | 1.4 | 0.2 | setosa |
| 2 | 4.9 | 5.0 | 1.4 | 0.2 | setosa |
| 3 | 4.7 | 5.2 | 1.3 | 0.2 | setosa |
| 4 | 4.6 | 5.1 | 1.5 | 0.2 | setosa |
| 5 | 5.0 | 5.6 | 1.4 | 0.2 | setosa |

`modify_if(.x, .p, .f, ...)`: Apply a function to elements that satisfy a predicate
`map_if(.x, .p, .f, ...)`

```
modify_if(iris, is.numeric, ~.+2)
```

| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|--------------|-------------|--------------|-------------|---------|
| 1 | 7.1 | 5.5 | 3.4 | 2.2 | setosa |
| 2 | 6.9 | 5.0 | 3.4 | 2.2 | setosa |
| 3 | 6.7 | 5.2 | 3.3 | 2.2 | setosa |
| 4 | 6.6 | 5.1 | 3.5 | 2.2 | setosa |
| 5 | 7.0 | 5.6 | 3.4 | 2.2 | setosa |

Modify

`modify_depth(.x, .depth, .f, ...)`: Apply a function to each element at a given level of a list (`map_depth()`)

```
modify_depth(iris, 0, ~.+2)
```

`modify2(.x, .y, .f, ...)`: modifies the elements of `.x` but also passes the elements of `.y` to `.f` (`map2()`)

```
modify2(x, c(TRUE, FALSE), \(x, cond) if (cond) x else 0)
```

| | x | |
|---|---|---|
| a | | 1 |
| b | | 2 |

| | x | |
|---|---|---|
| a | | 1 |
| b | | 0 |

Reduce Lists

`reduce(.x, .f, ..., .init, .dir = c("forward", "backward"))`: Apply a function recursively to each element of a list or a vector

```
x <- 1:10
[1] 1 2 3 4 5 6 7 8 9 10

reduce(x, sum)
[1] 55
```

`accumulate(.x, .f, ..., .init)`: Reduce a list, but also return intermediate results

```
accumulate(x, sum)
[1] 1 3 6 10 15 21 28 36 45 55
```

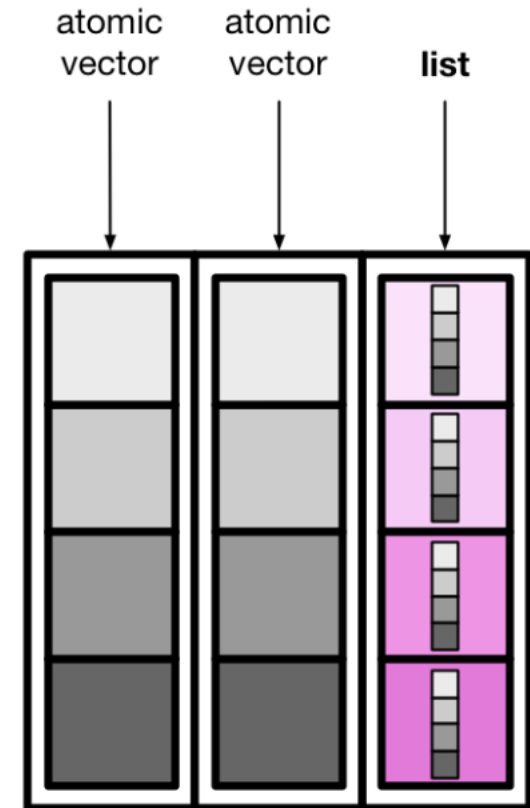
```
> accumulate(x, \(a,b) a*3/b)
[1] 1.000000000 1.500000000 1.500000000 1.125000000 0.675000000 0.337500000 0.144642857
[8] 0.054241071 0.018080357 0.005424107
```

`reduce2(.x, .y, .f, ..., .init)`, `accumulate2(.x, .y, .f, ..., .init)`

List-Columns

Columns of a data frame where each element is a list or a vector instead of an atomic value.

Manipulate list-columns using dplyr functions.
Use map functions within a column function to manipulate each element



List-Columns

```
starwars_1 <- starwars %>% select(name,vehicles,starships)
```

| | name | vehicles | starships |
|---|----------------|-------------------------------------------|---------------------------------|
| 1 | Luke Skywalker | c("Snowspeeder", "Imperial Speeder Bike") | c("X-wing", "Imperial shuttle") |
| 2 | C-3PO | character(0) | character(0) |
| 3 | R2-D2 | character(0) | character(0) |
| 4 | Darth Vader | character(0) | TIE Advanced x1 |
| 5 | Leia Organa | Imperial Speeder Bike | character(0) |

```
starwars_1 %>% mutate(ships=map2(vehicles,starships,rbind), .keep = "none")
```

| | ships |
|----|-----------------------------------------------------------|
| 1 | c("Snowspeeder", "X-wing", "Imperial Speeder Bike" [...]) |
| 2 | character(0) |
| 3 | character(0) |
| 4 | TIE Advanced x1 |
| 5 | Imperial Speeder Bike |
| 6 | character(0) |
| 7 | character(0) |
| 8 | character(0) |
| 9 | X-wing |
| 10 | c("Tribubble bongo", "Jedi starfighter", "Tribubbl [...]) |

Exercises – data set

mtcars: Motor Trend Car Road Tests

- 32 observations
- 11 variables

| | mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|--------------------------|------|-----|-------|-----|------|-------|-------|----|----|------|------|
| Mazda RX4 | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.620 | 16.46 | 0 | 1 | 4 | 4 |
| Mazda RX4 Wag | 21.0 | 6 | 160.0 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| Datsun 710 | 22.8 | 4 | 108.0 | 93 | 3.85 | 2.320 | 18.61 | 1 | 1 | 4 | 1 |
| Hornet 4 Drive | 21.4 | 6 | 258.0 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 |
| Hornet Sportabout | 18.7 | 8 | 360.0 | 175 | 3.15 | 3.440 | 17.02 | 0 | 0 | 3 | 2 |
| Valiant | 18.1 | 6 | 225.0 | 105 | 2.76 | 3.460 | 20.22 | 1 | 0 | 3 | 1 |
| Duster 360 | 14.3 | 8 | 360.0 | 245 | 3.21 | 3.570 | 15.84 | 0 | 0 | 3 | 4 |
| Merc 240D | 24.4 | 4 | 146.7 | 62 | 3.69 | 3.190 | 20.00 | 1 | 0 | 4 | 2 |

| | |
|------|------------------------------------------|
| mpg | Miles/(US) gallon |
| cyl | Number of cylinders |
| disp | Displacement (cu.in.) |
| hp | Gross horsepower |
| drat | Rear axle ratio |
| wt | Weight (1000 lbs) |
| qsec | 1/4 mile time |
| vs | Engine (0 = V-shaped, 1 = straight) |
| am | Transmission (0 = automatic, 1 = manual) |
| gear | Number of forward gears |

Exercises

1. Compare the values mpg, disp and hp to the means of these values:
 - I. Create a list column by binding (rbind) the values mpg, disp and hp into a list/vector
 - II. Create a list column by binding the mean values of mpg, disp and hp into a list/vector
 - III. Create a new column and compare the two created list columns by checking if the values are greater than the means (Result: Example: `c(TRUE, FALSE, FALSE)`)
2. Create a new column that displays all values in each row using a string. The values should be separated by a comma.

Filtering Lists - Pluck

`pluck(.x, ..., default=NULL)`: Select an element by using its name or index.

```
> pluck(iris, 1,2)
[1] 4.9
```

```
> pluck(iris,1,151)
NULL
```

```
> chuck(iris,1,151)
Error in `chuck()`:
! Index 2 exceeds the length of plucked object (151 > 150).
```

```
Run `rlang::last_trace()` to see where the error occurred.
```

`assign_in(x, where, value)`: Assign a value to a location using pluck selection

```
assign_in(iris, list(1,2), 50)
```

`modify_in(.x, .where, .f)`: Apply a function to a value at a selected location

```
modify_in(iris, list(1,2), ~.+10)
```

Filtering Lists - Pluck

```
> (x <- list(list("a", list(1,elt="foo")),  
+          list("b", list(2,elt="bar"))))
```

```
[[1]]  
[[1]][[1]]  
[1] "a"
```

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

```
[[1]][[2]]$elt  
[1] "foo"
```

```
[[2]]  
[[2]][[1]]  
[1] "b"
```

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

```
[[2]][[2]]$elt  
[1] "bar"
```

```
> pluck(x,1) #x[[1]]  
[[1]]  
[1] "a"
```

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

```
[[2]]$elt  
[1] "foo"
```

```
> pluck(x,1,2) #x[[1]][[2]]  
[[1]]  
[1] 1
```

```
$elt  
[1] "foo"
```

```
> pluck(x,1,2,2) #x[[1]][[2]][[2]]  
[1] "foo"
```

Filtering Lists

`keep(.x, .p, ...)`: Keep elements that pass a logical test

```
> iris[,1] %>% keep(~.x > 7)
[1] 7.1 7.6 7.3 7.2 7.7 7.7 7.7 7.2 7.2 7.4 7.9 7.7
```

`discard(.x, .p, ...)`: Discard elements that pass a logical test

```
> iris[,1] %>% discard(~.x > 4.5)
[1] 4.4 4.3 4.4 4.5 4.4
```

`keep_at(x, at)`: Keep elements based by name or position

```
> iris[,1] %>% keep_at(1:5)
[1] 5.1 4.9 4.7 4.6 5.0
```

`discard_at(x, at)`: Discard elements based by name or position

```
> iris[,1] %>% discard_at(2:(length(iris[,1])-1))
[1] 5.1 5.9
```

Filtering Lists

`head_while(.x, .p, ...)`: Return head elements until one does not pass

```
> iris[,1] %>% head_while(~.x>4.5)
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0
```

`tail_while(.x, .p, ...)`: Return tail elements until one does not pass

```
> iris[,1] %>% tail_while(~.x>5.8)
[1] 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

`compact(.x, .p = identity)`: Discard empty elements (NULL values)

```
x <- list(1,2,NULL)  [[1]]  > compact(x)
                    [1] 1  [[1]]
                    [[2]]  [1] 1
                    [1] 2  [[2]]
                    [[3]]  [1] 2
                    NULL
```

Nested data frame

```
iris_nest <- iris %>%  
  group_by(Species) %>%  
  nest()
```

| | Species | data |
|---|------------|------------|
| 1 | setosa | 1 variable |
| 2 | versicolor | 1 variable |
| 3 | virginica | 1 variable |



| | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
|---|--------------|-------------|--------------|-------------|
| 1 | 5.1 | 3.5 | 1.4 | 0.2 |
| 2 | 4.9 | 3.0 | 1.4 | 0.2 |
| 3 | 4.7 | 3.2 | 1.3 | 0.2 |
| 4 | 4.6 | 3.1 | 1.5 | 0.2 |
| 5 | 5.0 | 3.6 | 1.4 | 0.2 |

```
iris_nest %>% mutate(summary = map(data, ~ summary(.)))
```




| | Species | data | summary |
|---|------------|------------|------------------------------------------------------|
| 1 | setosa | 1 variable | c("Min. :4.300 ", "1st Qu.:4.800 ", "Median :5 [...] |
| 2 | versicolor | 1 variable | c("Min. :4.900 ", "1st Qu.:5.600 ", "Median :5 [...] |
| 3 | virginica | 1 variable | c("Min. :4.900 ", "1st Qu.:6.225 ", "Median :6 [...] |

```
> iris_nest %>% mutate(summary = map(data, ~ summary(.))) %>% pluck(3,1)
```

```
  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width  
Min. :4.300    Min. :2.300    Min. :1.000    Min. :0.100  
1st Qu.:4.800  1st Qu.:3.200    1st Qu.:1.400  1st Qu.:0.200  
Median :5.000  Median :3.400    Median :1.500  Median :0.200  
Mean :5.006    Mean :3.428     Mean :1.462    Mean :0.246  
3rd Qu.:5.200  3rd Qu.:3.675    3rd Qu.:1.575  3rd Qu.:0.300  
Max. :5.800    Max. :4.400     Max. :1.900    Max. :0.600
```

Nested data frame

```
iris_nest %>%  
  mutate(mean_Petal_L = map_dbl(data, ~mean(.$Petal.Length))) %>%  
  mutate(sd_Petal_L = map_dbl(data, ~sd(.$Petal.Length)))
```

| | Species | data | mean_Petal_L | sd_Petal_L |
|---|------------|----------------------------------------------------------------------------------------------|--------------|------------|
| 1 | setosa | 1 variable  | 1.462 | 0.1736640 |
| 2 | versicolor | 1 variable  | 4.260 | 0.4699110 |
| 3 | virginica | 1 variable  | 5.552 | 0.5518947 |

Summarise Lists

`every(.x, .p, ...)`: Do all elements pass a test?

```
> iris_10 %>% every(~.x>5)
[1] FALSE
```

`some(.x, .p, ...)`: Do some elements pass a test?

```
> iris_10 %>% some(~.x>5)
[1] TRUE
```

`none(.x, .p, ...)`: Do no elements pass a test?

```
> iris_10 %>% none(~.x<3)
[1] TRUE
```

`has_element(.x, .y)`: Does a list contain an element?

```
> iris_10 %>% has_element(4.5)
[1] FALSE
```

```
iris_10 <- iris[1:10,1]
```

| | x |
|----|-----|
| 1 | 5.1 |
| 2 | 4.9 |
| 3 | 4.7 |
| 4 | 4.6 |
| 5 | 5.0 |
| 6 | 5.4 |
| 7 | 4.6 |
| 8 | 5.0 |
| 9 | 4.4 |
| 10 | 4.9 |

Summarise Lists

`detect(.x, .f, ..., dir=c("forward", "backward"), right= NULL, default = NULL):`

Find first element to pass

```
> iris_10 %>% detect(~.x < 4.5)
[1] 4.4
```

`detect_index(.x, .f, ..., dir=c("forward", "backward"), right= NULL):`

Find index of first element to pass

```
> iris_10 %>% detect_index(~.x < 4.5)
[1] 9
```

`set_names(x, nm=x):` Set the names of a vector/list directly or within a function

```
> iris_10 %>% set_names(letters[1:10])
  a   b   c   d   e   f   g   h   i   j
5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

Reshaping Lists - Flatten

`list_flatten(.x)`: Removes a level of indexes from a list

```
x <- list(1,list(2,3),list(4,list(5)))
```

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

```
> x %>% list_flatten()
```

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

```
> x %>% list_flatten() %>% list_flatten()
```

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

Reshaping Lists

`list_transpose(.l, .names=NULL)`: Transposes the index order in a multi-level list

`list_c(x)`: Combines elements into a vector by concatenating them together

```
x <- list(a=1,b=2,c=3)
```

```
> x
$a
[1] 1

$b
[1] 2

$c
[1] 3
```

```
> list_c(x)
[1] 1 2 3
```

Reshaping Lists

```
x2 <- list(a = data.frame(x=1:2), b=data.frame(y="a"))
```

```
$a  
  x  
1 1  
2 2
```

```
$b  
  y  
1 a
```

`list_rbind(x)`: Combines elements into a data frame by row-binding them together

```
> list_rbind(x2)
```

```
  x    y  
1 1 <NA>  
2 2 <NA>  
3 NA  a
```

`list_cbind(x)`: Combines elements into a data frame by column-binding them together

```
> list_cbind(x2)
```

```
  x y  
1 1 a  
2 2 a
```

Reshaping Lists - Flatten

`flatten(.x)`: superseded by `list_flatten()`

Following functions have been superseded by `list_c()`:

- `flatten_lgl()`: returns a logical vector
- `flatten_int()`: returns an integer vector
- `flatten_dbl()`: returns a double vector
- `flatten_chr()`: returns a character vector

Following functions have been superseded by `list_rbind()` & `list_cbind()`

- `flatten_dfr()`: returns a data frames created by row-binding
- `flatten_dfc()`: returns a data frames created by column-binding

Exercises-Nested data frame

```
mtcars_nested <- mtcars %>%  
  group_by(cyl) %>%  
  nest()
```

| | cyl | data |
|---|-----|------------|
| 1 | 6 | 1 variable |
| 2 | 4 | 1 variable |
| 3 | 8 | 1 variable |



| | mpg | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|---|------|-------|-----|------|-------|-------|----|----|------|------|
| 1 | 21.0 | 160.0 | 110 | 3.90 | 2.620 | 16.46 | 0 | 1 | 4 | 4 |
| 2 | 21.0 | 160.0 | 110 | 3.90 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| 3 | 21.4 | 258.0 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 |
| 4 | 18.1 | 225.0 | 105 | 2.76 | 3.460 | 20.22 | 1 | 0 | 3 | 1 |
| 5 | 19.2 | 167.6 | 123 | 3.92 | 3.440 | 18.30 | 1 | 0 | 4 | 4 |
| 6 | 17.8 | 167.6 | 123 | 3.92 | 3.440 | 18.90 | 1 | 0 | 4 | 4 |
| 7 | 19.7 | 145.0 | 175 | 3.62 | 2.770 | 15.50 | 0 | 1 | 5 | 6 |

Exercises

1. Determine the maximum and minimum weight for each group using the `mtcars_nested` data frame by creating two new columns
2. Are all `mpg` values greater than 15 for each group of cylinders?
3. For which cylinder does the value 258 exist for the variable `disp`?
4. Keep the values for `hp` that are greater than 100 and write them into a vector
5. Determine the value of the second row for displacements (`disp`) for 6 cylinders by using the `pluck` function

Sources

- <https://purrr.tidyverse.org/>
- <https://cran.r-project.org/web/packages/purrr/purrr.pdf>
- <https://rstudio.github.io/cheatsheets/html/purrr.html>
- <https://medium.com/@ozturkfemre/purrr-in-r-a-powerful-tool-for-iteration-d4eb6c0b2d20>
- <https://dcl-prog.stanford.edu/purrr-extras.html>

Questions

Thank you

Additional slides

Handling errors

possibly(.f, otherwise = NULL, quiet = TRUE): Wrap a function to return a value instead of an error

```
> list("a",10) %>% map(possibly(log))
[[1]]
NULL

[[2]]
[1] 2.302585
```

safely(.f, , otherwise = NULL, quiet = TRUE): Wrap a function to capture errors

```
> list("a",10) %>% map(safely(log))
[[1]]
[[1]]$result
NULL

[[1]]$error
<simpleError in .Primitive("log")(x, base): Nicht-
numerisches Argument für mathematische Funktion>

[[2]]
[[2]]$result
[1] 2.302585

[[2]]$error
NULL
```

Handling errors

quietly(.f): Wrap a function to capture side-effects (warnings, messages)

```
f <- function() {  
  print("Quietly")  
  message("Hello")  
  warning("Warning Text")  
  100  
}
```

```
> f()  
[1] "Quietly"  
Hello  
[1] 100  
Warnmeldung:  
In f() : Warning Text
```

```
f_quiet <- quietly(f)
```

```
> f_quiet()  
$result  
[1] 100  
  
$output  
[1] "[1] \"quietly\""   
  
$warnings  
[1] "Warning Text"  
  
$messages  
[1] "Hello\n"
```

Transforming Functions

`slowly(f, rate = rate_backoff(), quiet = TRUE)`:
takes a function and modifies it to wait a given
amount of time between each call

```
rate_ind <- rate_delay(0.1)
slow_runif <- slowly(~ runif(1), rate_ind, quiet=FALSE)
map(1:5,slow_runif)
```

```
Retrying in 0.1 seconds.
Retrying in 0.1 seconds.
Retrying in 0.1 seconds.
Retrying in 0.1 seconds.
[[1]]
[1] 0.6778499

[[2]]
[1] 0.1472278

[[3]]
[1] 0.700526

[[4]]
[1] 0.9579925

[[5]]
[1] 0.8286251
```

Transforming Functions

`insistently(f, rate = rate_backoff(), quiet = TRUE)`: takes a function and modifies it to retry a given amount of time on error

```
risky_runif <- function() {  
  y <- runif(1, 0, 1)  
  if(y < 0.9) {  
    stop(y, " is too small")  
  }  
  y  
}
```

```
insistent_risky_runif <- insistently(risky_runif, rate_ind, quiet = FALSE)  
Error: 0.274401541100815 is too small  
Retrying in 0.1 seconds.  
Error: 0.433338072383776 is too small  
Retrying in 0.1 seconds.  
Error: 0.521773493150249 is too small  
Retrying in 0.1 seconds.  
Error: 0.126169137191027 is too small  
Retrying in 0.1 seconds.  
[1] 0.9561519
```

Compose

`compose(f, g)` is equivalent to `function(...) f(g(...))`

```
max_absolute <- compose(max, abs)
> max_absolute(rnorm(100))
[1] 2.901802
```

`negate(.p)`: Negate a predicate function so it selects what it previously rejected

Modify a list

`list_modify(.x, ...)`: modifies the elements of a list recursively

```
x <- list(x=1:5, y=4, z=list(a=1,b=2))
```

```
$x  
[1] 1 2 3 4 5
```

```
$y  
[1] 4
```

```
$z  
$z$a  
[1] 1
```

```
$z$b  
[1] 2
```

```
> list_modify(x, a=1)
```

```
$x  
[1] 1 2 3 4 5
```

```
$y  
[1] 4
```

```
$z  
$z$a  
[1] 1
```

```
$z$b  
[1] 2
```

```
$a  
[1] 1
```

```
> list_modify(x, z=list(a=1:5))
```

```
$x  
[1] 1 2 3 4 5
```

```
$y  
[1] 4
```

```
$z  
$z$a  
[1] 1 2 3 4 5
```

```
$z$b  
[1] 2
```

Modify a list

`list_assign(.x, ...)`: modifies the elements of a list by name or position

```
$x  
[1] 1 2 3 4 5
```

```
$y  
[1] 4
```

```
$z  
$z$a  
[1] 1
```

```
$z$b  
[1] 2
```

```
> list_assign(x, z=list(a=1:5))
```

```
$x  
[1] 1 2 3 4 5
```

```
$y  
[1] 4
```

```
$z  
$z$a  
[1] 1 2 3 4 5
```

Working with lists

`list_simplify(x, ..., strict = TRUE, ptype = NULL)`: simplify a list into an atomic vector

```
> list_simplify(list(1, 2, 3))  
[1] 1 2 3
```

```
> list_simplify(list(1, 2, 1:3))  
Error in `list_simplify()`:  
! `x[[3]]` must have size 1, not size 3.  
Run rlang::last_trace() to see where the error occurred.
```

```
> list_simplify(list(1, 2, 1:3), strict = FALSE)  
[[1]]  
[1] 1  
  
[[2]]  
[1] 2  
  
[[3]]  
[1] 1 2 3
```

Progress bars

`map(.x, .f, .progress = FALSE)`: progress argument

- TRUE: create unnamed progress bar
- String: create progress bar with the given name

```
> map(1:10000000, ~.+5, .progress="Progress")  
Progress ██████████ 14% | ETA: 17s
```