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)

Data Science for Operational Researchers using R

08 – purrr

https://github.com/JimDuggan/explore_or

Overview

- We earlier used the functional lapply() to iterate over an atomic vector, list, or data frame.
- The tidyverse package purrr provides a comprehensive set of functions that can be used to iterate over data structures
- It also integrates with other elements of the tidyverse, for example, the package dplyr.

- Topics
  - map() family of functions
  - Integtating with dplyr to process tibble
  - Mini-case with linear models.

(1) The map() family of functions

- The idea of the map() function is to provide 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

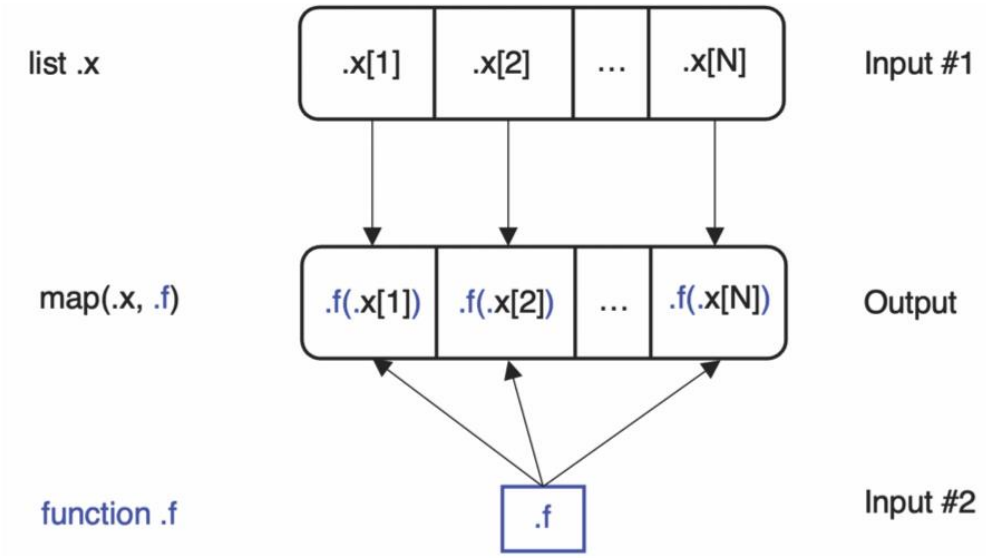


FIGURE 10.1 The map function in purrr

General format of map(.x, .f)

- .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.

```
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
```

Using a formula

- A formula (which is defined by ~) can be used as a function argument, and it is essentially a function shortcut in purrr, and commonly used in map()
- ~ marks the start of the function, and .x as the function parameter, where . can also be used to represent the function parameter.

```
o2 <- purrr::map(c(1,2,3,2),~.x^2)
str(o2)
#> List of 4
#>  $ : num 1
#>  $ : num 4
#>  $ : num 9
#>  $ : num 4
o3 <- purrr::map(c(1,2,3,2),~.^2)
str(o3)
#> List of 4
#>  $ : num 1
#>  $ : num 4
#>  $ : num 9
#>  $ : num 4
```

Using map() to select a list element

map() functions can be used to select a list element by name, and these are beneficial when working with lists that are deeply nested (Wickham, 2019).

```
library(repurrrsive)
dirs <- sw_films %>% purrr::map("director") %>% unique()
str(dirs)
#> List of 4
#>  $ : chr "George Lucas"
#>  $ : chr "Richard Marquand"
#>  $ : chr "Irvin Kershner"
#>  $ : chr "J. J. Abrams"
```

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. These include:

- `map_dbl()`, which returns an atomic vector of type `double`.
- `map_chr()`, which 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()`, which returns a data frame or tibble.

map_dbl()

Here, we can process a number of columns from the data frame `mtcars` and return the average of each column as an atomic vector. Note that because a data frame is also a list, we can use it as an input to the `map` family of functions.

```
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()

If we wanted to extract the director names from `sw_films` as an atomic vector, we can do this with `map_chr()`. Note that in this example, we use a formula to specify the function shortcut, and the argument is accessed using `.x`.

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

map_lgl()

Here, we process a number of the columns in `mpg` to test whether the columns are numeric. Here, we use the anonymous function option.

```
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()

In this example, we select a number of numeric columns from `mpg`, and then use `map_int()` to count the number of observations that are greater than the mean in each of the three columns. An atomic vector of integers is returned.

```
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()

The function `map_df()` creates a new data frame or tibble based on the input list. A tibble is specified within the function, and as `map_df()` iterates through the input list, rows will be added to the tibble with the specified values. In

```
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)
```

```
#> # 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
```

Iterating over 2 inputs – map2()

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
```

Iterating over multiple inputs with pmap()

pmap() can take a list containing any number of arguments, and process these elements within the function using the symbols ..1, ..2 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 ...
```

pmap() example

```
set.seed(100)
grades <- tibble(ID=paste0("S-",10:15),
                 Subject1=rnorm(6,70,10),
                 Subject2=rnorm(6,60,20),
                 Subject3=rnorm(6,50,15))
grades
#> # A tibble: 6 x 4
#>   ID    Subject1 Subject2 Subject3
#>   <chr>    <dbl>    <dbl>    <dbl>
#> 1 S-10      65.0     48.4     47.0
#> 2 S-11      71.3     74.3     61.1
#> 3 S-12      69.2     43.5     51.9
#> 4 S-13      78.9     52.8     49.6
#> 5 S-14      71.2     61.8     44.2
#> 6 S-15      73.2     61.9     57.7
```

pmap() example 2

```
grades1 <- grades %>%
            dplyr::mutate(Summary=pmap_chr(grades,
                                           ~paste0("ID=",
                                                   ..1,
                                                   " Max=",
                                                   round(max(..2,..3,..4),2))))
grades1
```

```
#> # A tibble: 6 x 5
#>   ID    Subject1 Subject2 Subject3 Summary
#>   <chr>    <dbl>    <dbl>    <dbl> <chr>
#> 1 S-10      65.0     48.4     47.0 ID=S-10 Max=64.98
#> 2 S-11      71.3     74.3     61.1 ID=S-11 Max=74.29
#> 3 S-12      69.2     43.5     51.9 ID=S-12 Max=69.21
#> 4 S-13      78.9     52.8     49.6 ID=S-13 Max=78.87
#> 5 S-14      71.2     61.8     44.2 ID=S-14 Max=71.17
#> 6 S-15      73.2     61.9     57.7 ID=S-15 Max=73.19
```

2. Integrating purrr with dplyr and tidyr

- A benefit of using the tidyverse is having the facility to combine tools from different packages, and switching between the use of lists and tibbles where appropriate.
- A common task is to divide a tibble into sub-groups, and perform operations on these
- We first define a test data set

```
set.seed(100)
test <- mpg %>%
        dplyr::select(manufacturer:displ,cty,class) %>%
        dplyr::filter(class %in% c("compact","midsize")) %>%
        dplyr::sample_n(5)
test
#> # A tibble: 5 x 5
#>   manufacturer model  displ   cty class
#>   <chr>        <chr>  <dbl> <int> <chr>
#> 1 volkswagen   jetta    2      21 compact
#> 2 volkswagen   jetta    2.5    21 compact
#> 3 chevrolet    malibu   3.6    17 midsize
#> 4 volkswagen   gti      2      21 compact
#> 5 audi         a4       2      21 compact
```

group_split()

- Takes a tibble
- And grouping variable(s)
- Generates a list of tibbles

```
test_s %>% purrr::map_int(~nrow(.x))
#> [1] 4 1
```

```
test_s <- test %>%
          dplyr::group_by(class) %>%
          dplyr::group_split()
test_s
#> <list_of<
#>   tbl_df<
#>     manufacturer: character
#>     model       : character
#>     displ       : double
#>     cty         : integer
#>     class       : character
#>   >
#> >[2]>
#> [[1]]
#> # A tibble: 4 x 5
#>   manufacturer model displ   cty class
#>   <chr>        <chr> <dbl> <int> <chr>
#> 1 volkswagen   jetta   2      21 compact
#> 2 volkswagen   jetta   2.5    21 compact
#> 3 volkswagen   gti     2      21 compact
#> 4 audi         a4      2      21 compact
#>
#> [[2]]
#> # A tibble: 1 x 5
#>   manufacturer model  displ   cty class
#>   <chr>        <chr>  <dbl> <int> <chr>
#> 1 chevrolet    malibu   3.6    17 midsize
```

Exploring data

- Our goal is to calculate the correlation coefficient between two variables at each station: mean sea level pressure and average wind speed.
- We simplify the dataset to daily values, where we take (1) the maximum wind speed (wdsp) recorded and (2) the average mean sea level pressure (msl).
- Our first task is to use dplyr to generate a summary tibble, and we also exclude any cases that have missing values, by combining complete.cases() within filter

```
d_data <- observations %>%
            dplyr::filter(complete.cases(observations)) %>%
            dplyr::group_by(station,month,day) %>%
            dplyr::summarize(MaxWdsp=max(wdsp,na.rm=TRUE),
                             DailyAverageMSL=mean(msl,na.rm=TRUE)) %>%
            dplyr::ungroup()
d_data
#> # A tibble: 8,394 x 5
#>    station month   day MaxWdsp DailyAverageMSL
#>    <chr>   <dbl> <int>   <dbl>           <dbl>
#>  1 ATHENRY     1     1      12            1027.
#>  2 ATHENRY     1     2       8            1035.
#>  3 ATHENRY     1     3       6            1032.
#>  4 ATHENRY     1     4       4            1030.
#>  5 ATHENRY     1     5       9            1029.
#>  6 ATHENRY     1     6       9            1028.
#>  7 ATHENRY     1     7       6            1032.
#>  8 ATHENRY     1     8       9            1029.
#>  9 ATHENRY     1     9      16            1015.
#> 10 ATHENRY     1    10      13            1013.
#> # ... with 8,384 more rows
```

Process the results

```
cor7 <- d_data %>%
        dplyr::group_by(station) %>%
        dplyr::group_split() %>%
        purrr::map_df(~{
                corr <- cor(.x$MaxWdsp,.x$DailyAverageMSL)
                tibble(Station=first(.x$station),
                       CorrCoeff=corr)
        }) %>%
        dplyr::arrange(CorrCoeff) %>%
        dplyr::slice(1:7)
```

```
cor7
#> # A tibble: 7 x 2
#>   Station               CorrCoeff
#>   <chr>                     <dbl>
#> 1 SherkinIsland            -0.589
#> 2 VALENTIA OBSERVATORY     -0.579
#> 3 ROCHES POINT             -0.540
#> 4 MACE HEAD                -0.539
#> 5 MOORE PARK               -0.528
#> 6 SHANNON AIRPORT          -0.524
#> 7 CORK AIRPORT             -0.522
```

Alternative solution…

```
cor7_b <- d_data %>%
            dplyr::group_by(station) %>%
            dplyr::summarize(CorrCoeff=cor(MaxWdsp,DailyAverageMSL)) %>%
            dplyr::arrange(CorrCoeff) %>%
            dplyr::slice(1:7)
```

```
cor7_b
#> # A tibble: 7 x 2
#>   station             CorrCoeff
#>   <chr>                   <dbl>
#> 1 SherkinIsland          -0.589
#> 2 VALENTIA OBSERVATORY   -0.579
#> 3 ROCHES POINT           -0.540
#> 4 MACE HEAD              -0.539
#> 5 MOORE PARK             -0.528
#> 6 SHANNON AIRPORT        -0.524
#> 7 CORK AIRPORT           -0.522
```

nest()

- nest(), which is part of the package tidyr, can be used to create a list column within a tibble that contains a data frame.
- Nesting generates one row for each defined group, which is identified using the function group_by().
- The second column is named data, and is a list, and each list element contains all of the tibble's data for a particular group.

```
d_data
#> # A tibble: 8,394 x 5
#>    station month   day MaxWdsp DailyAverageMSL
#>    <chr>   <dbl> <int>   <dbl>           <dbl>
#>  1 ATHENRY     1     1      12            1027.
#>  2 ATHENRY     1     2       8            1035.
```

```
data_n <- d_data %>%
          dplyr::group_by(station) %>%
          tidyr::nest()

data_n %>% head()
#> # A tibble: 6 x 2
#> # Groups:   station [6]
#>   station      data
#>   <chr>        <list>
#> 1 ATHENRY      <tibble [365 x 4]>
#> 2 BALLYHAISE   <tibble [365 x 4]>
#> 3 BELMULLET    <tibble [365 x 4]>
#> 4 CASEMENT     <tibble [365 x 4]>
#> 5 CLAREMORRIS  <tibble [365 x 4]>
#> 6 CORK AIRPORT <tibble [365 x 4]>
```

The data column

- Here, the tibble data_n contains two columns, with a row for each weather station (the first six rows are shown here).
- All of the data for each weather station is stored in the respective cell in the column data.
- first() is a wrapper around the list operator [[that returns the first value in a list.
- Note the group name does not appear in data (it is already in the first column)

```
data_n %>% head()
#> # A tibble: 6 x 2
#> # Groups:   station [6]
#>   station      data
#>   <chr>        <list>
#> 1 ATHENRY      <tibble [365 x 4]>
```

```
data_n %>%
  dplyr::pull(data) %>%
  dplyr::first()
#> # A tibble: 365 x 4
#>    month   day MaxWdsp DailyAverageMSL
#>    <dbl> <int>   <dbl>           <dbl>
#>  1     1     1      12           1027.
#>  2     1     2       8           1035.
#>  3     1     3       6           1032.
#>  4     1     4       4           1030.
#>  5     1     5       9           1029.
#>  6     1     6       9           1028.
#>  7     1     7       6           1032.
#>  8     1     8       9           1029.
#>  9     1     9      16           1015.
#> 10     1    10      13           1013.
#> # ... with 355 more rows
```

Generating a linear model for each row using map()

```
data_n <- data_n    %>%
            dplyr::mutate(LM=map(data,
                                 ~lm(MaxWdsp~DailyAverageMSL,
                                     data=.)))

data_n %>%
  head()
#> # A tibble: 6 x 3
#> # Groups:   station [6]
#>   station      data               LM
#>   <chr>        <list>             <list>
#> 1 ATHENRY      <tibble [365 x 4]> <lm>
#> 2 BALLYHAISE   <tibble [365 x 4]> <lm>
#> 3 BELMULLET    <tibble [365 x 4]> <lm>
#> 4 CASEMENT     <tibble [365 x 4]> <lm>
#> 5 CLAREMORRIS  <tibble [365 x 4]> <lm>
#> 6 CORK AIRPORT <tibble [365 x 4]> <lm>
```

Exploring the result for “BELMULLET”

Note, probably not needed…

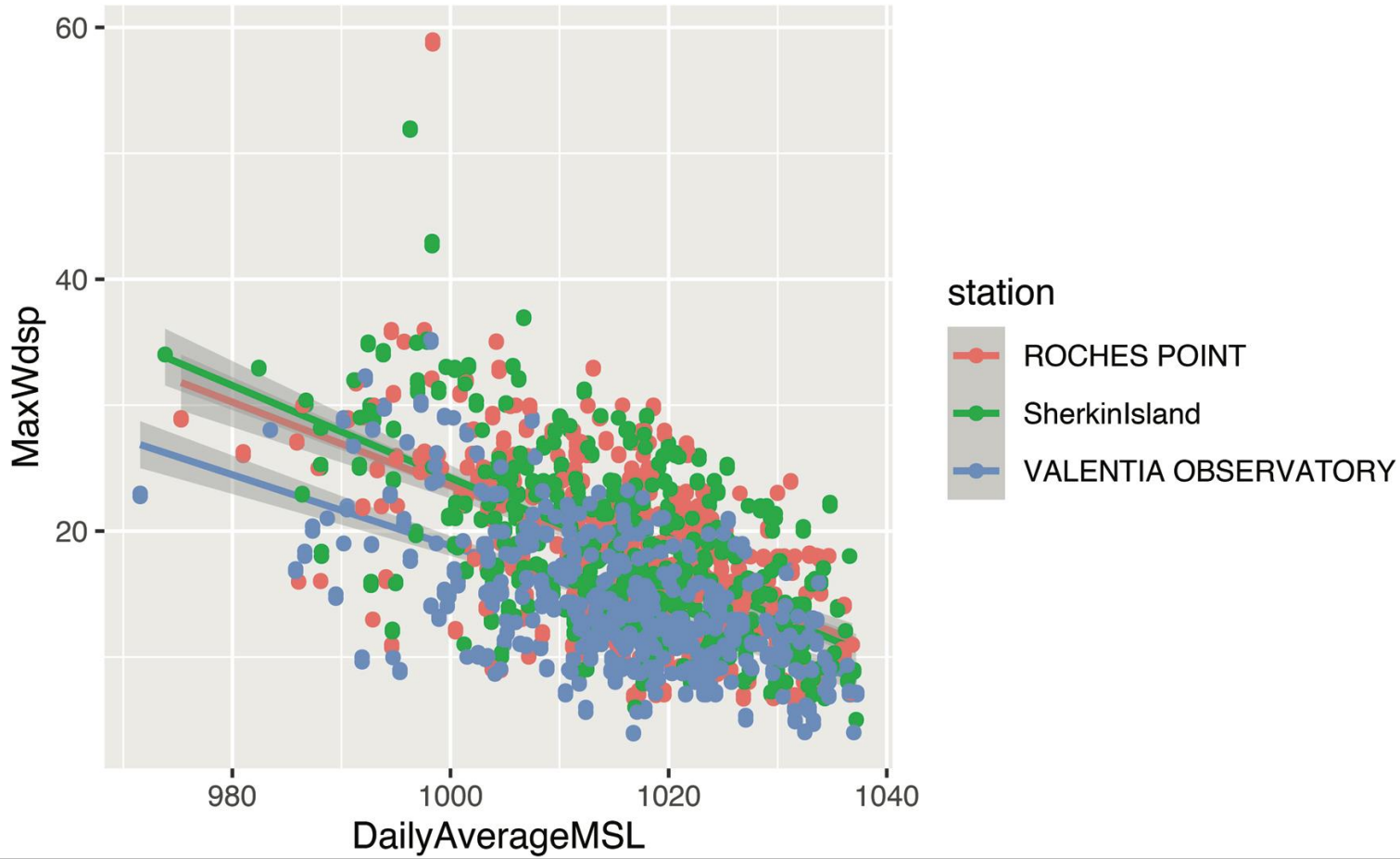
```
data_n %>%
  dplyr::filter(station=="BELMULLET") %>%
  dplyr::pull(LM) %>%
  dplyr::first() %>%
  summary()
#>
#> Call:
#> lm(formula = MaxWdsp ~ DailyAverageMSL, data = .)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -14.021  -4.069  -0.516   3.958  17.962
#>
#> Coefficients:
#>                 Estimate Std. Error t value Pr(>|t|)
#> (Intercept)      242.786     26.365    9.21   <2e-16 ***
#> DailyAverageMSL   -0.222      0.026   -8.53    4e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 5.8 on 363 degrees of freedom
#> Multiple R-squared:  0.167,  Adjusted R-squared:  0.165
#> F-statistic: 72.8 on 1 and 363 DF,  p-value: 4.03e-16
```

Add R^2 value as a column

```
data_n <- data_n %>%
            dplyr::mutate(RSq=map_dbl(LM,~summary(.x)$r.squared)) %>%
            dplyr::arrange(desc(RSq))
data_n  <- data_n %>% head(n=3)
data_n
#> # A tibble: 3 x 4
#> # Groups:   station [3]
#>   station             data               LM        RSq
#>   <chr>               <list>             <list> <dbl>
#> 1 SherkinIsland       <tibble [365 x 4]> <lm>   0.347
#> 2 VALENTIA OBSERVATORY <tibble [365 x 4]> <lm>   0.335
#> 3 ROCHES POINT        <tibble [365 x 4]> <lm>   0.291
```

```r
data1 <- d_data %>%
         dplyr::filter(station %in% dplyr::pull(data_n,station))

ggplot(data1,aes(x=DailyAverageMSL,y=MaxWdsp,color=station))+
       geom_point()+geom_smooth(method="lm")+geom_jitter()
```



pluck()

- The function pluck() provides a generalized form of the [[operator and provides the means to index data structures in a flexible way.
- The arguments include .x, which is a vector, and a list of accessors for indexing into the object, which can include an integer position or a string name.

```
library(ggplot2)
library(repurrrsive)

# Use pluck() to access the second element of an atomic vector
mpg %>% dplyr::pull(class) %>% unique() %>% purrr::pluck(2)
#> [1] "midsize"
# Use pluck() to access the director in the first list location
sw_films %>% purrr::pluck(1,"director")
#> [1] "George Lucas"
```

walk()

- walk(.x,.f) is similar to map, except that it returns the input .x and calls the function .f to generate a side effect.
- The side effect, for example, could be displaying information onto the screen.

```
l <- list(el1=20,el2=30,el3=40)
o <- purrr::walk(l,~cat("Creating a side effect...\n"))
#> Creating a side effect...
#> Creating a side effect...
#> Creating a side effect...
str(o)
#> List of 3
#>  $ el1: num 20
#>  $ el2: num 30
#>  $ el3: num 40
```

keep()

- The function keep(.x,.f) takes in a list .x and, based on the evaluation of a predicate function, will either keep or discard list element.
- In effect, it provides a way to filter a list.

```
o <- sw_films %>% keep(~.x$director=="George Lucas")
purrr::walk(o,~cat(.x$director," ==> Title =",.x$title,"\n"))
#> George Lucas  ==> Title = A New Hope
#> George Lucas  ==> Title = Attack of the Clones
#> George Lucas  ==> Title = The Phantom Menace
#> George Lucas  ==> Title = Revenge of the Sith
```

invoke_map()

- Provides a feature to call a list of functions with a list of parameters.
- It is a wrapper around the R function do.call().
- The function takes a list of functions (where the function names are contained in a character vector), and an argument list, where the names of the function arguments are contained in the list.
- Function is now deprecated (see advice)

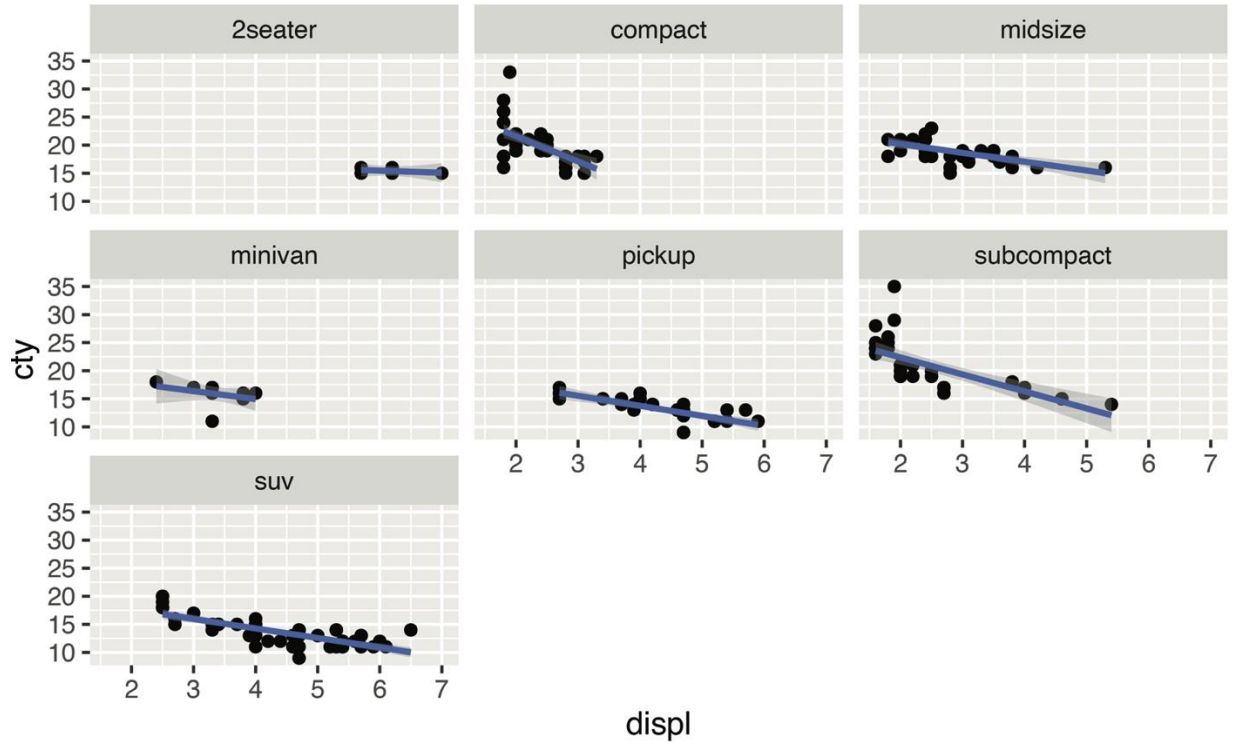
```
f <- c("min","max","sum")
l <- list(
  arg1=list(x=1:3),
  arg2=list(x=10:12),
  arg3=list(x=1:4)
)

str(purrr::invoke_map(f,l))
#> Warning: `invoke_map()` was deprecated in purrr 1.0.0.
#> i Please use map() + exec() instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where this
#> warning was generated.
#> List of 3
#>  $ : int 1
#>  $ : int 12
#>  $ : int 10
```

Mini-case - Linear models

- Here, we generate a collection of linear models from the mpg tibble, and explore the relationship between a dependent variable (cty) and a dependent variable (displ).
- In this example, we will partition the data by vehicle class, and so build seven different linear models.
- Before starting the modelling workflow, we can visualize the different linear models using ggplot().

```
ggplot(mpg,aes(x=displ,y=cty))+
  geom_point()+geom_smooth(method="lm")+
  facet_wrap(~class)
```



Exercise 1

1. Create the following tibble with the three columns shown, using the functions `keep()` and `map_df()`, in order to provide a tabular view of the list `repurrrsive::sw_vehicles`. Note that possible invalid values of `length` in `sw_vehicles` include "unknown", and any of these should be removed prior to creating the data frame.

```
#> # A tibble: 6 x 3
#>   Name                  Model                          Length
#>   <chr>                 <chr>                           <dbl>
#> 1 C-9979 landing craft  C-9979 landing craft             210
#> 2 SPHA                  Self-Propelled Heavy Artillery   140
#> 3 Clone turbo tank      HAVw A6 Juggernaut                49.4
#> 4 Sand Crawler          Digger Crawler                    36.8
#> 5 Multi-Troop Transport Multi-Troop Transport             31
#> 6 Sail barge            Modified Luxury Sail Barge        30
```

Overall approach

- Create a nested tibble (grouped by class), and this will have two columns, class and data, where the data column is created by the call to nest().
- Using mutate(), add a new column that will store the results of each linear model.
- Process the linear model results to extract a measure of model performance (r-squared) to see how well each regression model explains observed data.
- Arrange the tibble so that the best model scores are shown first.
- Add a column that will store the plots for each model.

Use nest() to reconfigure tibble

```
mpg1 <- mpg %>%
        dplyr::group_by(class) %>%
        tidyr::nest()
mpg1
#> # A tibble: 7 x 2
#> # Groups:   class [7]
#>   class      data
#>   <chr>      <list>
#> 1 compact    <tibble [47 x 10]>
#> 2 midsize    <tibble [41 x 10]>
#> 3 suv        <tibble [62 x 10]>
#> 4 2seater    <tibble [5 x 10]>
#> 5 minivan    <tibble [11 x 10]>
#> 6 pickup     <tibble [33 x 10]>
#> 7 subcompact <tibble [35 x 10]>
```

Perform regression on each vehicle class

```
mpg1 <- mpg1 %>%
        dplyr::mutate(LM=map(data,~lm(cty~displ,data=.x)))
mpg1
#> # A tibble: 7 x 3
#> # Groups:   class [7]
#>   class      data               LM
#>   <chr>      <list>             <list>
#> 1 compact    <tibble [47 x 10]> <lm>
#> 2 midsize    <tibble [41 x 10]> <lm>
#> 3 suv        <tibble [62 x 10]> <lm>
#> 4 2seater    <tibble [5 x 10]>  <lm>
#> 5 minivan    <tibble [11 x 10]> <lm>
#> 6 pickup     <tibble [33 x 10]> <lm>
#> 7 subcompact <tibble [35 x 10]> <lm>
```

Show one of the model results

```
mpg1 %>%
  dplyr::filter(class=="suv") %>%  # Select row where class == suv"
  dplyr::pull(LM) %>%              # Get the column "LM"
  dplyr::first() %>%               # Extract the lm object
  summary()                        # Call the summary function
#>
#> Call:
#> lm(formula = cty ~ displ, data = .x)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -4.087 -1.027 -0.087  1.096  3.967
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   21.060      0.893    23.6  < 2e-16 ***
#> displ         -1.696      0.195    -8.7  3.2e-12 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.62 on 60 degrees of freedom
#> Multiple R-squared:  0.558,  Adjusted R-squared:  0.55
#> F-statistic: 75.7 on 1 and 60 DF,  p-value: 3.17e-12
```

Extract R^2 value into new column

```
mpg1 <- mpg1 %>%
        dplyr::mutate(RSquared=map_dbl(LM,~summary(.x)$r.squared)) %>%
        dplyr::arrange(desc(RSquared))
mpg1
#> # A tibble: 7 x 4
#> # Groups:   class [7]
#>   class      data               LM     RSquared
#>   <chr>      <list>             <list>    <dbl>
#> 1 suv        <tibble [62 x 10]> <lm>      0.558
#> 2 subcompact <tibble [35 x 10]> <lm>      0.527
#> 3 pickup     <tibble [33 x 10]> <lm>      0.525
#> 4 compact    <tibble [47 x 10]> <lm>      0.358
#> 5 midsize    <tibble [41 x 10]> <lm>      0.339
#> 6 2seater    <tibble [5 x 10]>  <lm>      0.130
#> 7 minivan    <tibble [11 x 10]> <lm>      0.124
```

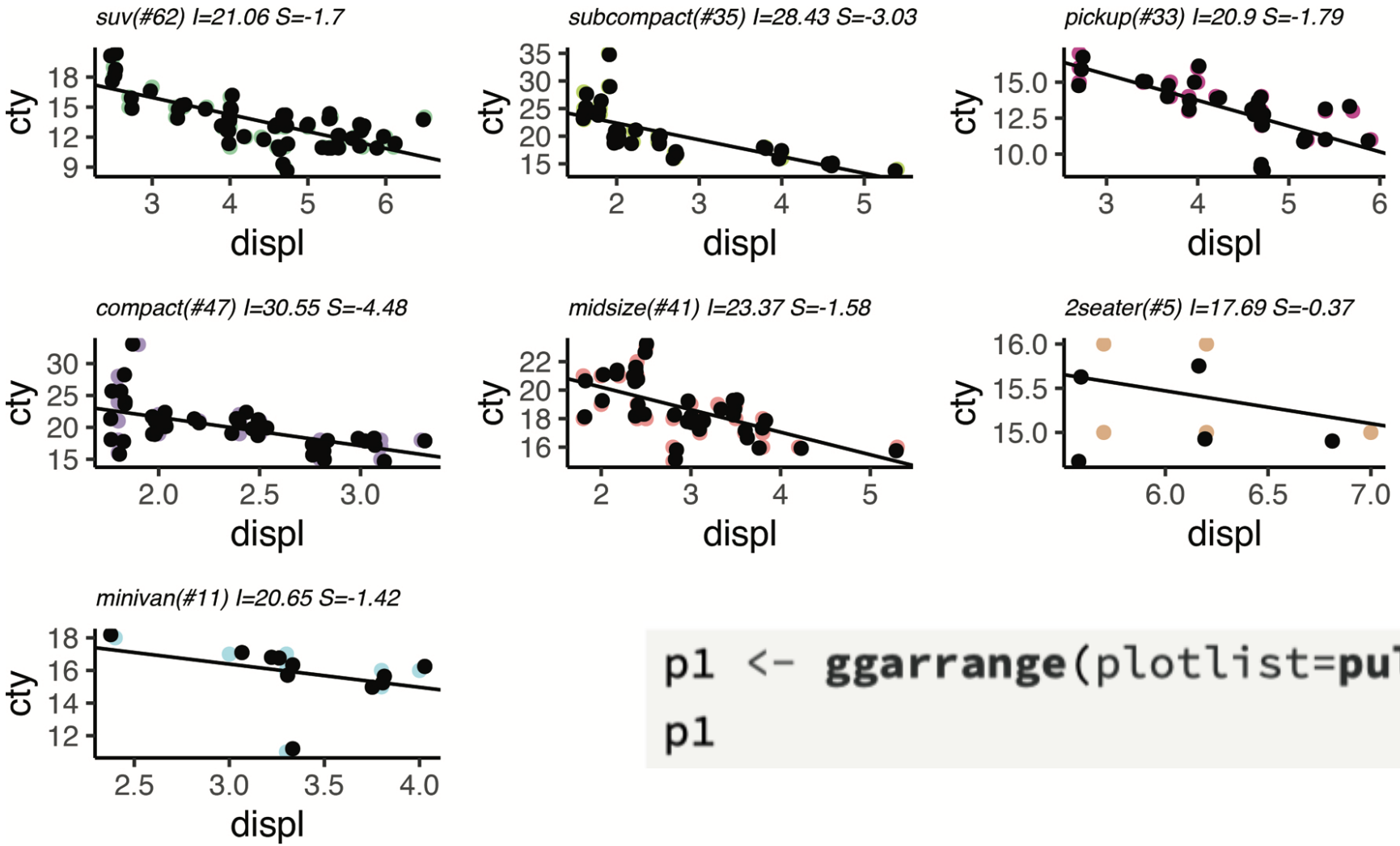
Create custom plots via map2()

```
library(randomcoloR) # Functions to generate different colors
set.seed(100) # For random color generation

mpg1 <- mpg1 %>%
       dplyr::mutate(Plots=map2(class,data, ~{
                # (1) run linear model
                m <- lm(cty~displ,data=.y)
                # (2) Extract coefficients
                intercept <- round(coef(m)[1],2)
                slope     <- round(coef(m)[2],2)
                # (3) return the ggplot,
                #     include the #obs and
                #     coefficients in title
               ggplot(.y,aes(x=displ,y=cty))+
                 ggtitle(paste0(.x,"(#",nrow(.y),") I=",
                                intercept,
                                " S=",slope))+
                 geom_point(color=randomColor(),size=1.5)+
                 geom_abline(slope=slope,intercept = intercept)+
                 theme_classic()+
                 geom_jitter()+
                 theme(plot.title = element_text(size = 7,
                                                 face = "italic"))
            }))
```

```
mpg1
#> # A tibble: 7 x 5
#> # Groups:   class [7]
#>   class      data               LM     RSquared Plots
#>   <chr>      <list>             <list>    <dbl> <list>
#> 1 suv        <tibble [62 x 10]> <lm>      0.558 <gg>
#> 2 subcompact <tibble [35 x 10]> <lm>      0.527 <gg>
#> 3 pickup     <tibble [33 x 10]> <lm>      0.525 <gg>
#> 4 compact    <tibble [47 x 10]> <lm>      0.358 <gg>
#> 5 midsize    <tibble [41 x 10]> <lm>      0.339 <gg>
#> 6 2seater    <tibble [5 x 10]>  <lm>      0.130 <gg>
#> 7 minivan    <tibble [11 x 10]> <lm>      0.124 <gg>
```

Visualise results (package ggpubr)




```
p1 <- ggarrange(plotlist=pull(mpg1,Plots))
p1
```