DSCI353-353m-453: Class 11a Tidyverse Review

Profs: R. H. French, L. S. Bruckman, P. Leu, K. Davis, S. Cirlos

TAs: W. Oltjen, K. Hernandez, M. Li, M. Li, D. Colvin

03 November, 2022

Contents

7.2.4 Tidyverse Review . . . 1
7.2.4.1 CREATING TIBBLES WITH tibble() —- . . . 2
7.2.4.2 CONVERTING DATA FRAMES TO TIBBLES WITH as_tibble() —- . . . 2
7.2.4.3 TIBBLES DON'T CONVERT STRINGS TO FACTORS BY DEFAULT —- 3
7.2.4.4 IF YOU WANT TO CREATE A FACTOR, WRAP THE c() FUNCTION INSIDE factor() —- . . . 3
7.2.5 Tidyverse of the `starwars` dataset of the dplyr package . . . 3
7.2.5.1 PRINTING A TIBBLE KEEPS THE OUTPUT CONCISE —- . . . 3
7.2.5.2 SUBSETTING WITH [ALWAYS RETURNS ANOTHER TIBBLE —- . . . 4
7.2.5.3 VARIABLE CREATION IN tibble() IS SEQUENTIAL —- . . . 5
7.2.6 EXPLORING THE CO2 DATASET —- . . . 5
7.2.6.1 SELECTING COLUMNS WITH select() —- . . . 5
7.2.6.2 FILTERING DATA WITH filter() —- . . . 6
7.2.6.3 GROUPING DATA WITH group_by() —- . . . 6
7.2.6.4 SUMMARIZING DATA WITH summarize() —- . . . 6
7.2.6.5 CREATING NEW VARIABLES WITH mutate() —- . . . 6
7.2.6.6 ARRANGING DATA WITH arrange() —- . . . 7
7.2.6.7 USING THE %>% ("PIPE") OPERATOR —- . . . 7
7.2.6.8 COMBINING DPLYR VERBS WITH THE %>% OPERATOR —- . . . 7
7.2.6.9 PLOTTING THE IRIS DATASET WITH ggplot() —- . . . 8
7.2.6.10 ADDING ADDITIONAL GEOMETRIC OBJECTS ("GEOMS") AS PLOT LAYERS —- . . . 8
7.2.6.11 MAPPING SPECIES TO THE SHAPE AND COLOR AESTHETICS —- . 9
7.2.6.12 FACETING BY SPECIES —- . . . 10
7.2.6.13 CREATING AN UNTIDY TIBBLE —- . . . 11
7.2.6.14 CONVERTING UNTIDY DATA TO TIDY FORMAT USING gather() . . . 11
7.2.6.14.1 or the same can be achieved with: . . . 12
7.2.6.14.2 or: . . . 12
7.2.6.15 CONVERTING DATA INTO WIDE FORMAT WITH spread() —- . . . 12
7.2.6.16 EXAMPLE OF PURE FUNCTION VS ONE WITH SIDE EFFECTS —- . 13
7.2.6.17 USING purrr FUNCTIONS FOR VECTORIZATION —- . . . 13
7.2.7 Links . . . 17

7.2.4 Tidyverse Review

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6      v purrr   0.3.5
## v tibble  3.1.7      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts --------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
myTib <- tibble(x =  1:4,
                y = c("london", "beijing", "las vegas", "berlin"))

myTib
```

7.2.4.1 CREATING TIBBLES WITH tibble() —-

```
## # A tibble: 4 x 2
##       x y
##   <int> <chr>
## 1     1 london
## 2     2 beijing
## 3     3 las vegas
## 4     4 berlin
```

```
myDf <- data.frame(x =  1:4,
                   y = c("london", "beijing", "las vegas", "berlin"))

dfToTib <- as_tibble(myDf)

dfToTib
```

7.2.4.2 CONVERTING DATA FRAMES TO TIBBLES WITH as_tibble() —-

```
## # A tibble: 4 x 2
##       x y
##   <int> <chr>
## 1     1 london
## 2     2 beijing
## 3     3 las vegas
## 4     4 berlin
```

```
myDf <- data.frame(x =  1:4,
                   y = c("london", "beijing", "las vegas", "berlin"))

myDfNotFactor <- data.frame(x =  1:4,
                            y = c("london", "beijing", "las vegas", "berlin"),
                            stringsAsFactors = FALSE)

myTib <- tibble(x =  1:4,
                y = c("london", "beijing", "las vegas", "berlin"))

class(myDf$y)
```

7.2.4.3 TIBBLES DON'T CONVERT STRINGS TO FACTORS BY DEFAULT —-

```
## [1] "character"
```

```
class(myDfNotFactor$y)
```

```
## [1] "character"
```

```
class(myTib$y)
```

```
## [1] "character"
```

```
myTib <- tibble(x =  1:4,
                y = factor(c("london", "beijing", "las vegas", "berlin")))
myTib
```

7.2.4.4 IF YOU WANT TO CREATE A FACTOR, WRAP THE c() FUNCTION INSIDE factor() —-

```
## # A tibble: 4 x 2
##       x y
##   <int> <fct>
## 1     1 london
## 2     2 beijing
## 3     3 las vegas
## 4     4 berlin
```

7.2.5 Tidyverse of the starwars dataset of the dplyr package

Starwars characters

- Description
  - This data comes from SWAPI, the Star Wars API, https://swapi.dev/
- Usage
  - `starwars`
- Format
  - A tibble with 87 rows and 13 variables:

```
?starwars
```

```
data(starwars)

glimpse(starwars)
```

7.2.5.1 PRINTING A TIBBLE KEEPS THE OUTPUT CONCISE —-

```
## Rows: 87
## Columns: 14
## $ name       <chr> "Luke Skywalker", "C-3PO", "R2-D2", "Darth Vader", "Leia Or~
## $ height     <int> 172, 167, 96, 202, 150, 178, 165, 97, 183, 182, 188, 180, 2~
## $ mass       <dbl> 77.0, 75.0, 32.0, 136.0, 49.0, 120.0, 75.0, 32.0, 84.0, 77.~
## $ hair_color <chr> "blond", NA, NA, "none", "brown", "brown, grey", "brown", N~
## $ skin_color <chr> "fair", "gold", "white, blue", "white", "light", "light", "~
## $ eye_color  <chr> "blue", "yellow", "red", "yellow", "brown", "blue", "blue",~
## $ birth_year <dbl> 19.0, 112.0, 33.0, 41.9, 19.0, 52.0, 47.0, NA, 24.0, 57.0, ~
## $ sex        <chr> "male", "none", "none", "male", "female", "male", "female",~
```

```
## $ gender     <chr> "masculine", "masculine", "masculine", "masculine", "femini~
## $ homeworld  <chr> "Tatooine", "Tatooine", "Naboo", "Tatooine", "Alderaan", "T~
## $ species    <chr> "Human", "Droid", "Droid", "Human", "Human", "Human", "Huma~
## $ films      <list> <"The Empire Strikes Back", "Revenge of the Sith", "Return~
## $ vehicles   <list> <"Snowspeeder", "Imperial Speeder Bike">, <>, <>, <>, "Imp~
## $ starships  <list> <"X-wing", "Imperial shuttle">, <>, <>, "TIE Advanced x1",~
```

```
head(as.data.frame(starwars))
```

```
##             name height mass  hair_color  skin_color eye_color birth_year
## 1 Luke Skywalker    172   77       blond        fair      blue       19.0
## 2          C-3PO    167   75        <NA>        gold    yellow      112.0
## 3          R2-D2     96   32        <NA> white, blue       red       33.0
## 4    Darth Vader    202  136        none       white    yellow       41.9
## 5    Leia Organa    150   49       brown       light     brown       19.0
## 6      Owen Lars    178  120 brown, grey       light      blue       52.0
##      sex    gender homeworld species
## 1   male masculine  Tatooine   Human
## 2   none masculine  Tatooine   Droid
## 3   none masculine     Naboo   Droid
## 4   male masculine  Tatooine   Human
## 5 female  feminine  Alderaan   Human
## 6   male masculine  Tatooine   Human
##
## 1                                          The Empire Strikes Back, Revenge of the Sith, Return of
## 2                   The Empire Strikes Back, Attack of the Clones, The Phantom Menace, Revenge of th
## 3 The Empire Strikes Back, Attack of the Clones, The Phantom Menace, Revenge of the Sith, Return of
## 4                                                                 The Empire Strikes Back, Revenge of th
## 5                                          The Empire Strikes Back, Revenge of the Sith, Return of
## 6                                                                                              Attack of the
##                            vehicles               starships
## 1 Snowspeeder, Imperial Speeder Bike X-wing, Imperial shuttle
## 2
## 3
## 4                                          TIE Advanced x1
## 5              Imperial Speeder Bike
## 6
```

```
myDf[, 1]
```

7.2.5.2 SUBSETTING WITH [ALWAYS RETURNS ANOTHER TIBBLE —-

```
## [1] 1 2 3 4
```

```
myTib[, 1]
```

```
## # A tibble: 4 x 1
##       x
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
```

```
myTib[[1]]
```

```
## [1] 1 2 3 4
```

```
myTib$x
```

```
## [1] 1 2 3 4
```

```
sequentialTib <- tibble(nItems = c(12, 45, 107),
                        cost = c(0.5, 1.2, 1.8),
                        totalWorth = nItems * cost)

sequentialTib
```

7.2.5.3 VARIABLE CREATION IN tibble() IS SEQUENTIAL —-

```
## # A tibble: 3 x 3
##   nItems  cost totalWorth
##    <dbl> <dbl>      <dbl>
## 1     12   0.5          6
## 2     45   1.2         54
## 3    107   1.8        193.
```

7.2.6 EXPLORING THE CO2 DATASET —-

- Carbon Dioxide Uptake in Grass Plants
  - Description
    - The CO2 data frame has 84 rows and 5 columns of data
      - from an experiment on the cold tolerance
      - of the grass species Echinochloa crus-galli.

```
data(CO2)

CO2tib <- as_tibble(CO2)

glimpse(CO2tib)
```

```
## Rows: 84
## Columns: 5
## $ Plant     <ord> Qn1, Qn1, Qn1, Qn1, Qn1, Qn1, Qn1, Qn2, Qn2, Qn2, Qn2, Qn2, ~
## $ Type      <fct> Quebec, Quebec, Quebec, Quebec, Quebec, Quebec, Quebec, Queb~
## $ Treatment <fct> nonchilled, nonchilled, nonchilled, nonchilled, nonchilled, ~
## $ conc      <dbl> 95, 175, 250, 350, 500, 675, 1000, 95, 175, 250, 350, 500, 6~
## $ uptake    <dbl> 16.0, 30.4, 34.8, 37.2, 35.3, 39.2, 39.7, 13.6, 27.3, 37.1, ~
```

```
selectedData <- select(CO2tib, 1, 2, 3, 5)

glimpse(selectedData)
```

7.2.6.1 SELECTING COLUMNS WITH select() —-

```
## Rows: 84
## Columns: 4
## $ Plant     <ord> Qn1, Qn1, Qn1, Qn1, Qn1, Qn1, Qn1, Qn2, Qn2, Qn2, Qn2, Qn2, ~
## $ Type      <fct> Quebec, Quebec, Quebec, Quebec, Quebec, Quebec, Quebec, Queb~
## $ Treatment <fct> nonchilled, nonchilled, nonchilled, nonchilled, nonchilled, ~
## $ uptake    <dbl> 16.0, 30.4, 34.8, 37.2, 35.3, 39.2, 39.7, 13.6, 27.3, 37.1, ~
```

```
filteredData <- filter(selectedData, uptake > 16)

glimpse(filteredData)
```

7.2.6.2 FILTERING DATA WITH filter() —-

```
## Rows: 66
## Columns: 4
## $ Plant     <ord> Qn1, Qn1, Qn1, Qn1, Qn1, Qn1, Qn2, Qn2, Qn2, Qn2, Qn2, Qn2, ~
## $ Type      <fct> Quebec, Quebec, Quebec, Quebec, Quebec, Quebec, Quebec, Queb~
## $ Treatment <fct> nonchilled, nonchilled, nonchilled, nonchilled, nonchilled, ~
## $ uptake    <dbl> 30.4, 34.8, 37.2, 35.3, 39.2, 39.7, 27.3, 37.1, 41.8, 40.6, ~
```

```
groupedData <- group_by(filteredData, Plant)

glimpse(groupedData)
```

7.2.6.3 GROUPING DATA WITH group_by() —-

```
## Rows: 66
## Columns: 4
## Groups: Plant [11]
## $ Plant     <ord> Qn1, Qn1, Qn1, Qn1, Qn1, Qn1, Qn2, Qn2, Qn2, Qn2, Qn2, Qn2, ~
## $ Type      <fct> Quebec, Quebec, Quebec, Quebec, Quebec, Quebec, Quebec, Queb~
## $ Treatment <fct> nonchilled, nonchilled, nonchilled, nonchilled, nonchilled, ~
## $ uptake    <dbl> 30.4, 34.8, 37.2, 35.3, 39.2, 39.7, 27.3, 37.1, 41.8, 40.6, ~
```

```
summarizedData <- summarize(groupedData, meanUp = mean(uptake),
                            sdUp = sd(uptake))

glimpse(summarizedData)
```

7.2.6.4 SUMMARIZING DATA WITH summarize() —-

```
## Rows: 11
## Columns: 3
## $ Plant  <ord> Qn1, Qn2, Qn3, Qc1, Qc3, Qc2, Mn3, Mn2, Mn1, Mc3, Mc1
## $ meanUp <dbl> 36.10000, 38.75000, 37.61429, 32.60000, 35.50000, 36.60000, 26.~
## $ sdUp   <dbl> 3.4234486, 6.0724789, 10.3499482, 5.0318983, 7.5158499, 5.14470~
```

```
mutatedData <- mutate(summarizedData,  CV = (sdUp / meanUp) * 100)

mutatedData
```

7.2.6.5 CREATING NEW VARIABLES WITH mutate() —-

```
## # A tibble: 11 x 4
##    Plant meanUp   sdUp    CV
##    <ord>  <dbl>  <dbl> <dbl>
##  1 Qn1     36.1  3.42   9.48
##  2 Qn2     38.8  6.07  15.7
##  3 Qn3     37.6 10.3   27.5
```

```
##  4 Qc1     32.6  5.03  15.4
##  5 Qc3     35.5  7.52  21.2
##  6 Qc2     36.6  5.14  14.1
##  7 Mn3     26.2  3.49  13.3
##  8 Mn2     29.9  3.92  13.1
##  9 Mn1     29.0  5.70  19.6
## 10 Mc3     18.4  0.826  4.48
## 11 Mc1     20.1  1.83   9.11
```

```
arrangedData <- arrange(mutatedData, CV)

glimpse(arrangedData)
```

7.2.6.6 ARRANGING DATA WITH arrange() —-

```
## Rows: 11
## Columns: 4
## $ Plant  <ord> Mc3, Mc1, Qn1, Mn2, Mn3, Qc2, Qc1, Qn2, Mn1, Qc3, Qn3
## $ meanUp <dbl> 18.41667, 20.12000, 36.10000, 29.90000, 26.25000, 36.60000, 32.~
## $ sdUp   <dbl> 0.8256311, 1.8335757, 3.4234486, 3.9181628, 3.4852547, 5.144706~
## $ CV     <dbl> 4.483065, 9.113200, 9.483237, 13.104224, 13.277161, 14.056574, ~
```

```
c(1, 4, 7, 3, 5) %>% mean()
```

7.2.6.7 USING THE %>% (“PIPE”) OPERATOR —-

```
## [1] 4
```

```
arrangedData <- CO2tib %>%
  select(c(1:3, 5)) %>%
  filter(uptake > 16) %>%
  group_by(Plant) %>%
  summarize(meanUp = mean(uptake), sdUp = sd(uptake)) %>%
  mutate(CV = (sdUp / meanUp) * 100) %>%
  arrange(CV)

glimpse(arrangedData)
```

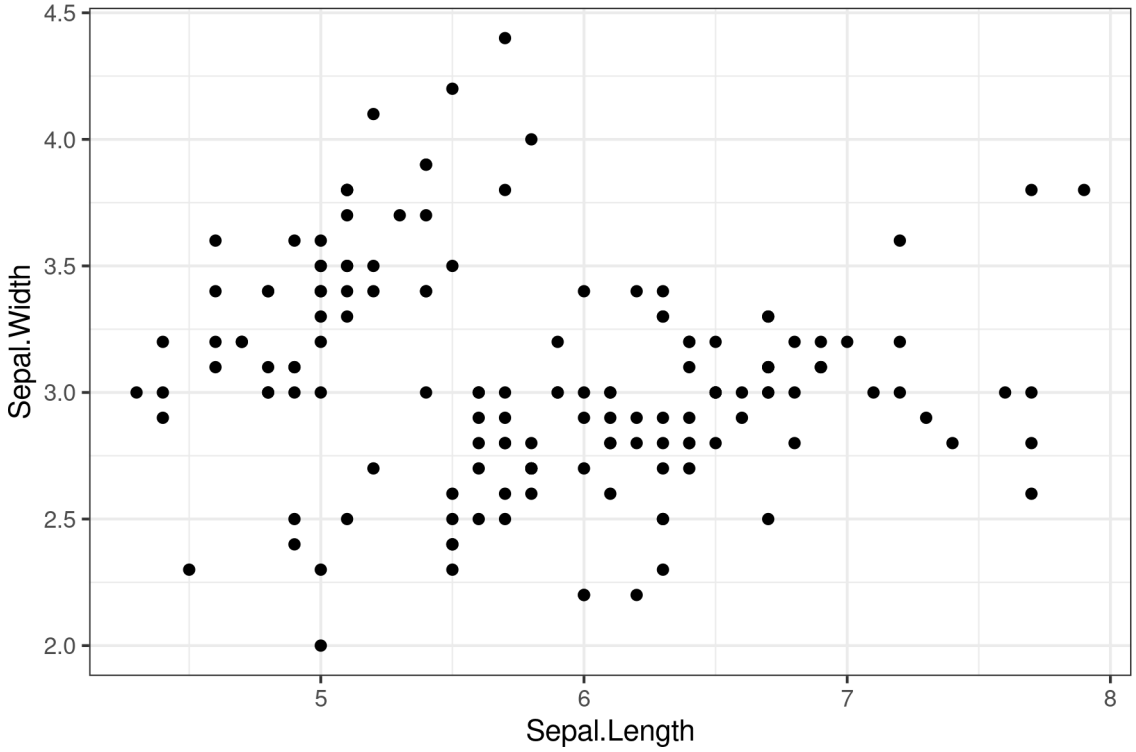
7.2.6.8 COMBINING DPLYR VERBS WITH THE %>% OPERATOR —-

```
## Rows: 11
## Columns: 4
## $ Plant  <ord> Mc3, Mc1, Qn1, Mn2, Mn3, Qc2, Qc1, Qn2, Mn1, Qc3, Qn3
## $ meanUp <dbl> 18.41667, 20.12000, 36.10000, 29.90000, 26.25000, 36.60000, 32.~
## $ sdUp   <dbl> 0.8256311, 1.8335757, 3.4234486, 3.9181628, 3.4852547, 5.144706~
## $ CV     <dbl> 4.483065, 9.113200, 9.483237, 13.104224, 13.277161, 14.056574, ~
```

```
data(iris)
myPlot <- ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point() +
  theme_bw()
```

```
myPlot
```

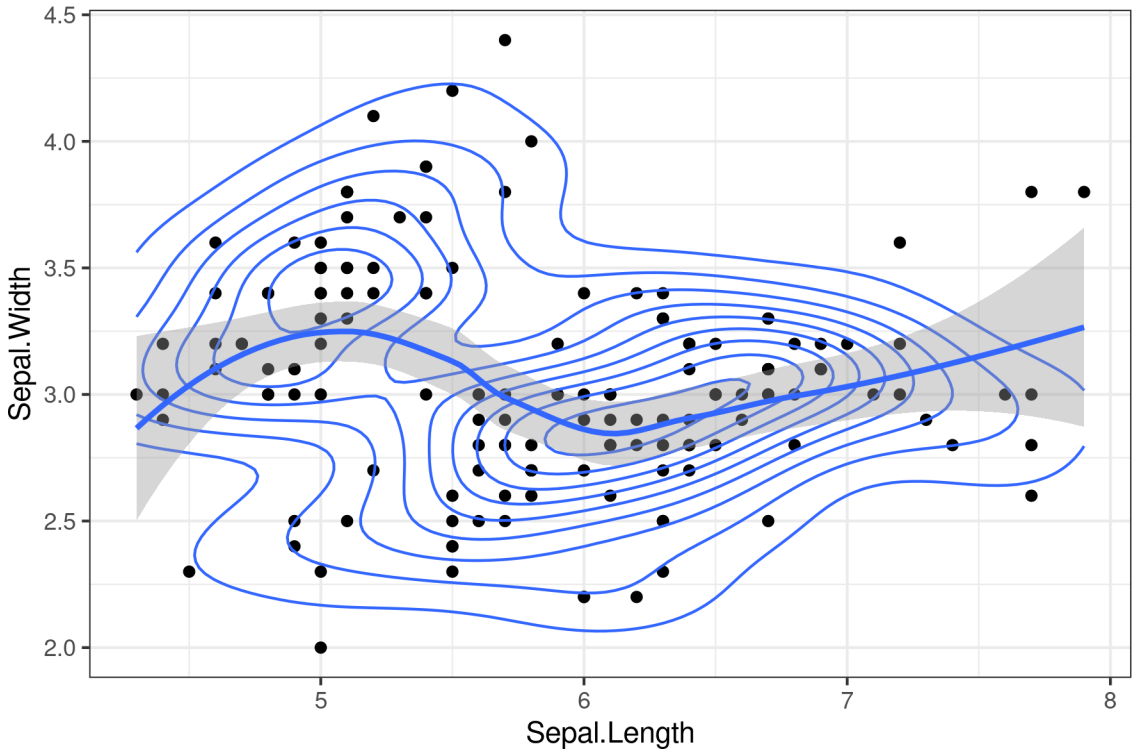
7.2.6.9 PLOTTING THE IRIS DATASET WITH ggplot() —-




```
myPlot +
  geom_density_2d() +
  geom_smooth()
```

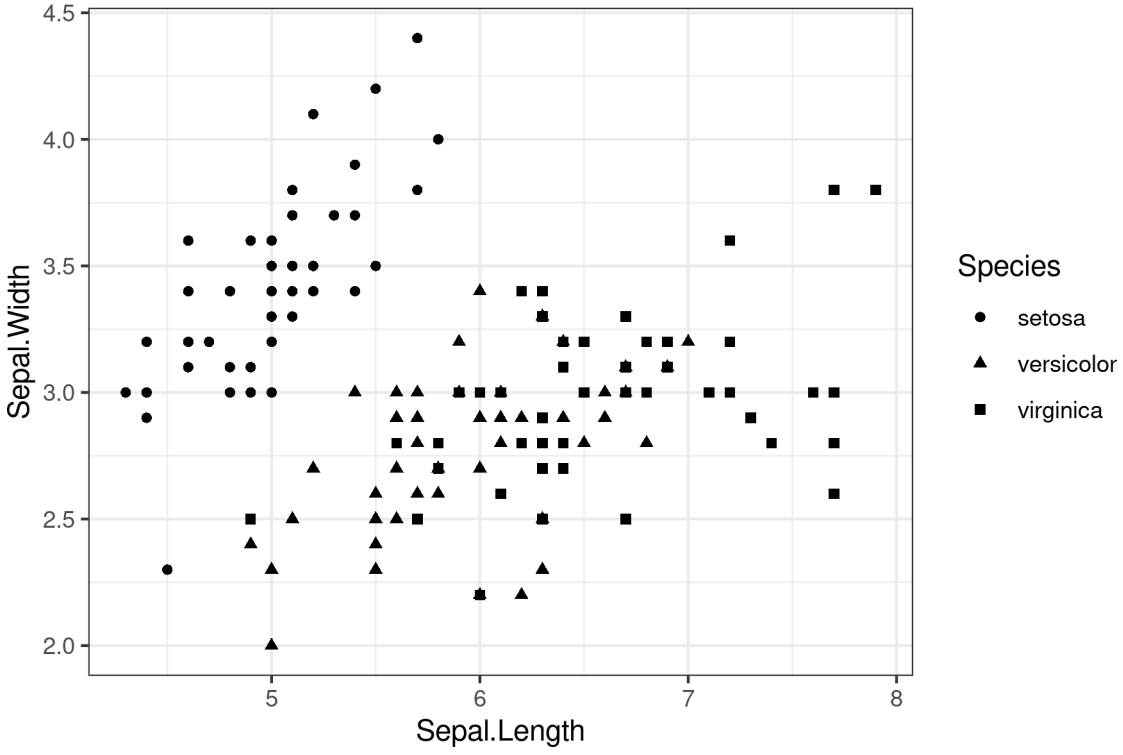
7.2.6.10 ADDING ADDITIONAL GEOMETRIC OBJECTS (“GEOMS”) AS PLOT LAYERS —-

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

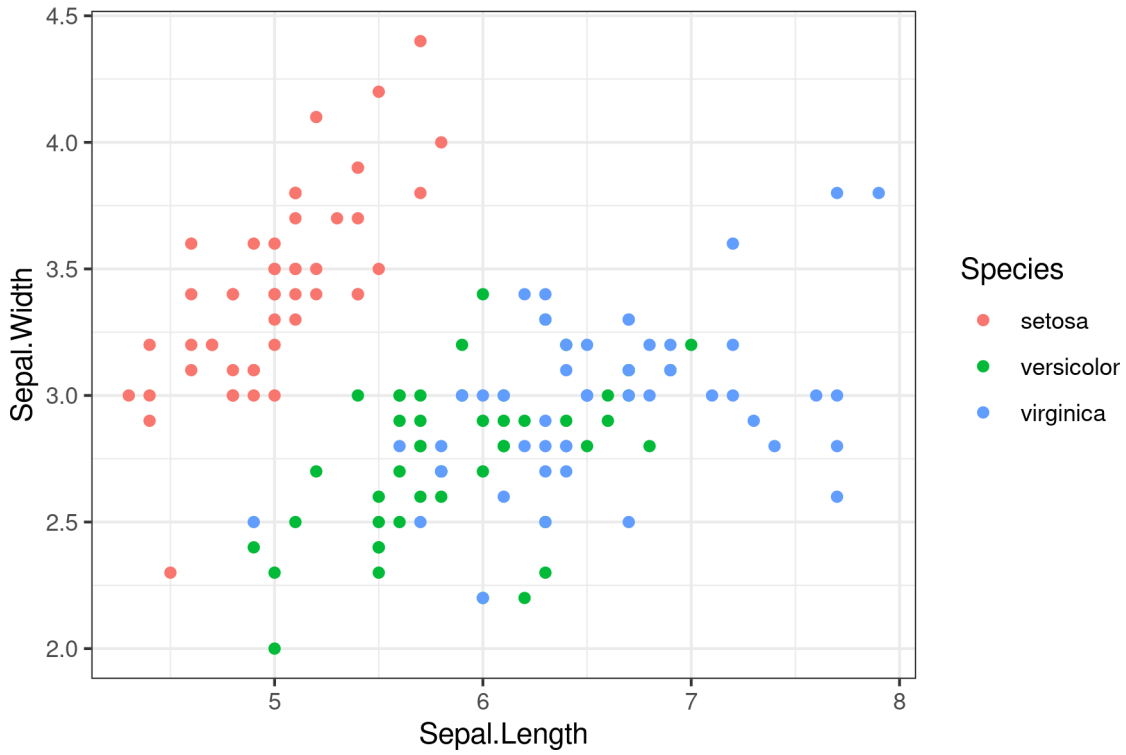



```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, shape = Species)) +
  geom_point()  +
  theme_bw()
```

7.2.6.11 MAPPING SPECIES TO THE SHAPE AND COLOR AESTHETICS —-

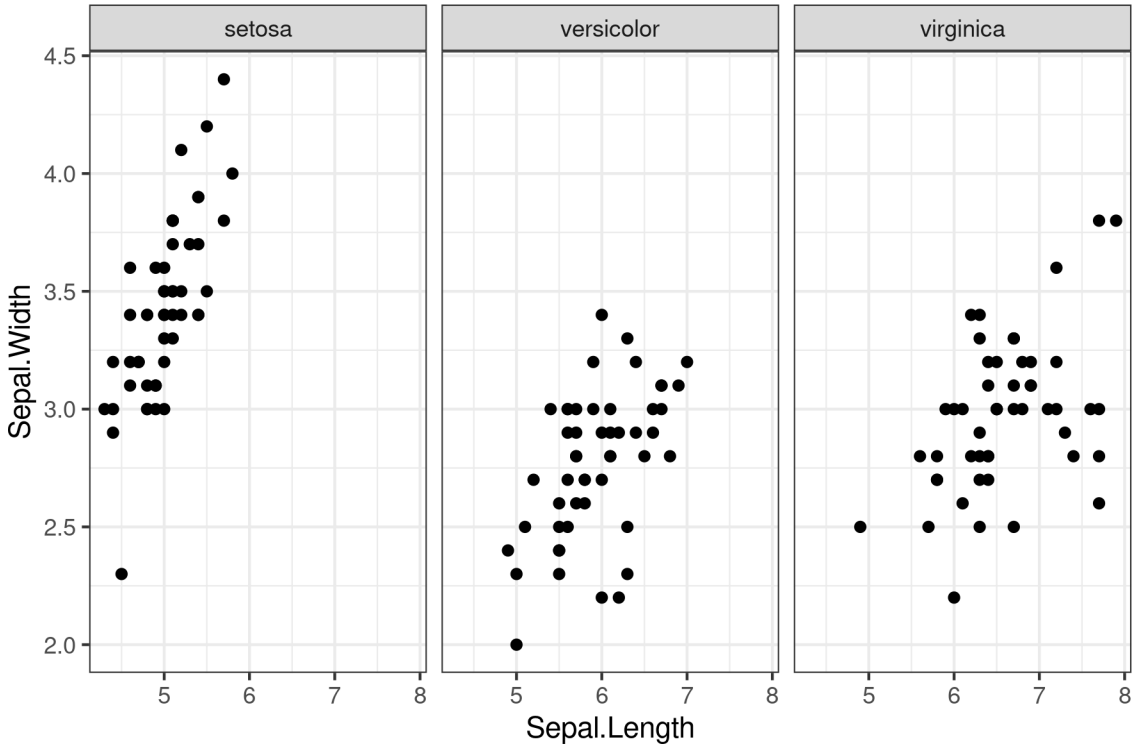


```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_point()  +
  theme_bw()
```




```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  facet_wrap(~ Species) +
  geom_point()  +
  theme_bw()
```

7.2.6.12 FACETING BY SPECIES —-




```
patientData <- tibble(Patient = c("A", "B", "C"),
                      Month0 = c(21, 17, 29),
                      Month3 = c(20, 21, 27),
                      Month6 = c(21, 22, 23))

glimpse(patientData)
```

7.2.6.13 CREATING AN UNTIDY TIBBLE —-

```
## Rows: 3
## Columns: 4
## $ Patient <chr> "A", "B", "C"
## $ Month0  <dbl> 21, 17, 29
## $ Month3  <dbl> 20, 21, 27
## $ Month6  <dbl> 21, 22, 23
```

```
tidyPatientData <- gather(patientData, key = Month,
                          value = BMI, -Patient)
glimpse(tidyPatientData)
```

7.2.6.14 CONVERTING UNTIDY DATA TO TIDY FORMAT USING gather()

```
## Rows: 9
## Columns: 3
## $ Patient <chr> "A", "B", "C", "A", "B", "C", "A", "B", "C"
## $ Month   <chr> "Month0", "Month0", "Month0", "Month3", "Month3", "Month3", "M~
## $ BMI     <dbl> 21, 17, 29, 20, 21, 27, 21, 22, 23
```

```
gather(patientData, key = Month, value = BMI, Month0:Month6)
```

7.2.6.14.1 or the same can be achieved with:

```
## # A tibble: 9 x 3
##   Patient Month    BMI
##   <chr>   <chr>  <dbl>
## 1 A       Month0    21
## 2 B       Month0    17
## 3 C       Month0    29
## 4 A       Month3    20
## 5 B       Month3    21
## 6 C       Month3    27
## 7 A       Month6    21
## 8 B       Month6    22
## 9 C       Month6    23
```

```
gather(patientData, key = Month, value = BMI, c(Month0, Month3, Month6))
```

7.2.6.14.2 or:

```
## # A tibble: 9 x 3
##   Patient Month    BMI
##   <chr>   <chr>  <dbl>
## 1 A       Month0    21
## 2 B       Month0    17
## 3 C       Month0    29
## 4 A       Month3    20
## 5 B       Month3    21
## 6 C       Month3    27
## 7 A       Month6    21
## 8 B       Month6    22
## 9 C       Month6    23
```

```
spread(tidyPatientData, key = Month, value = BMI)
```

7.2.6.15 CONVERTING DATA INTO WIDE FORMAT WITH spread() —-

```
## # A tibble: 3 x 4
##   Patient Month0 Month3 Month6
##   <chr>    <dbl>  <dbl>  <dbl>
## 1 A           21     20     21
## 2 B           17     21     22
## 3 C           29     27     23
```

```
a <- 20

pure <- function() {
  a <- a + 1
  a
}
```

```
side_effect <- function() {
  a <<- a + 1
  a
}

c(pure(), pure())
```

7.2.6.16 EXAMPLE OF PURE FUNCTION VS ONE WITH SIDE EFFECTS —-

```
## [1] 21 21
```

```
c(side_effect(), side_effect())
```

```
## [1] 21 22
```

```
listOfNumerics <- list(a = rnorm(5),
                       b = rnorm(9),
                       c = rnorm(10))

listOfNumerics
```

7.2.6.17 USING purrr FUNCTIONS FOR VECTORIZATION —-

```
## $a
## [1]  0.2336365  0.6387299 -2.4522889  0.7026484  0.8055726
##
## $b
## [1]  1.3834358  2.2914127 -0.5568244  1.1799996 -0.1562491 -1.2957353  1.5716689
## [8] -0.1459925 -0.6674849
##
## $c
##  [1] -1.04686734 -0.99372054 -1.31635997 -1.12880287  0.11694195  0.08030374
##  [7]  0.81075242  0.40770716  0.34221514  1.41335929
```

```
elementLengths <- vector("list", length = 3)

for (i in seq_along(listOfNumerics)) {
  elementLengths[[i]] <- length(listOfNumerics[[i]])
}

elementLengths
```

```
## [[1]]
## [1] 5
##
## [[2]]
## [1] 9
##
## [[3]]
## [1] 10
```

This code is difficult to read,

- requires us to predefine an empty vector
  - to prevent the loop from being slow,
- and has a side effect:

  - if we run the loop again,
  - it will overwrite the elementLengths list.

Instead, we can replace the for loop with the map() function.

The first argument of all the functions in the map family

- is the data we're iterating over.

The second argument is the function

- we're applying to each list element.

Take a look at figure 2.5, which illustrates how the map() function

- applies a function to every element of a list/vector
- and returns a list containing the outputs.

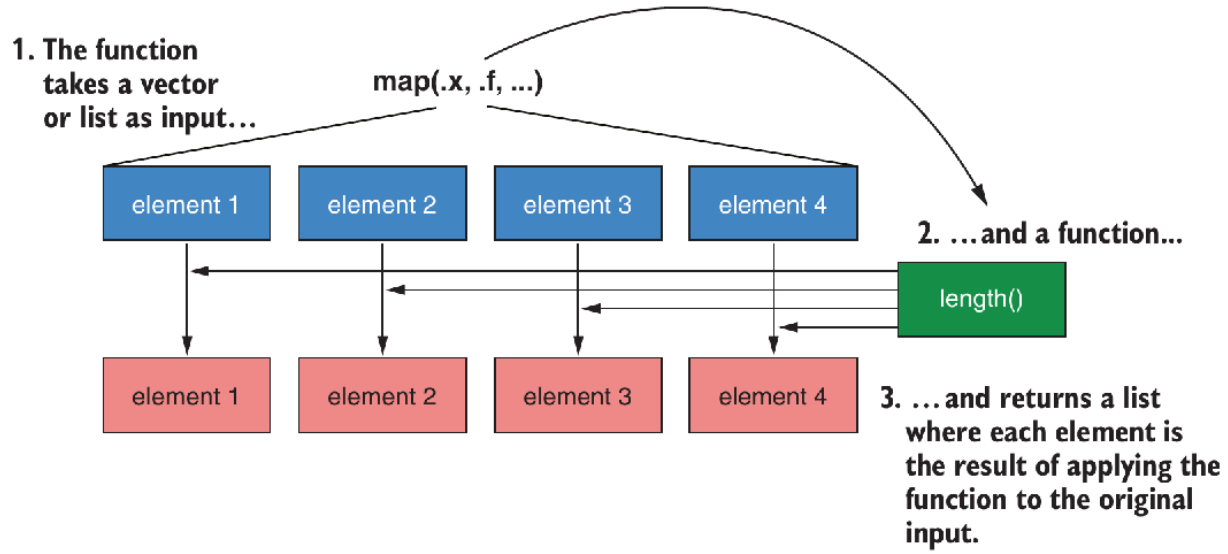


Figure 2.5 The `map()` function takes a vector or list as input, applies a function to each element individually, and returns a list of the returned values.

Figure 1: map() function

```
map(listOfNumerics, length)
```

```
## $a
## [1] 5
##
## $b
## [1] 9
##
## $c
## [1] 10
```

```
map_int(listOfNumerics, length)
```

```
##  a  b  c
##  5  9 10
```

```
map_chr(listOfNumerics, length)
```

```
##    a    b    c
##  "5"  "9" "10"
```

```
# map_lgl(listOfNumerics, length) # this throws an error
# Error: Can't coerce element 1 from a integer to a logical
```

We can do the same thing using the map_chr() function,

- which coerces the output into a character vector,
- but the map_lgl() function throws an error
  - because it can't coerce the output into a logical vector.

NOTE

- Forcing us to explicitly state the type of output we want to return
  - prevents bugs from unexpected types of output.

```
map_df(listOfNumerics, length)
```

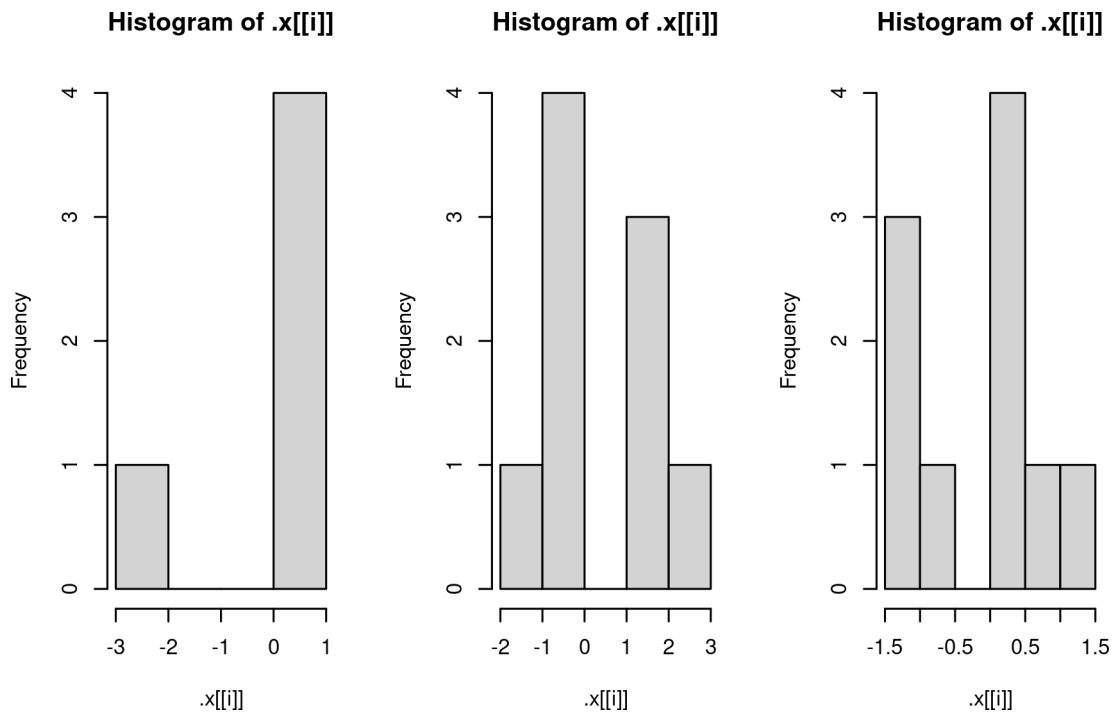
```
## # A tibble: 1 x 3
##       a     b     c
##   <int> <int> <int>
## 1     5     9    10
```

```
map(listOfNumerics, ~. + 2)
```

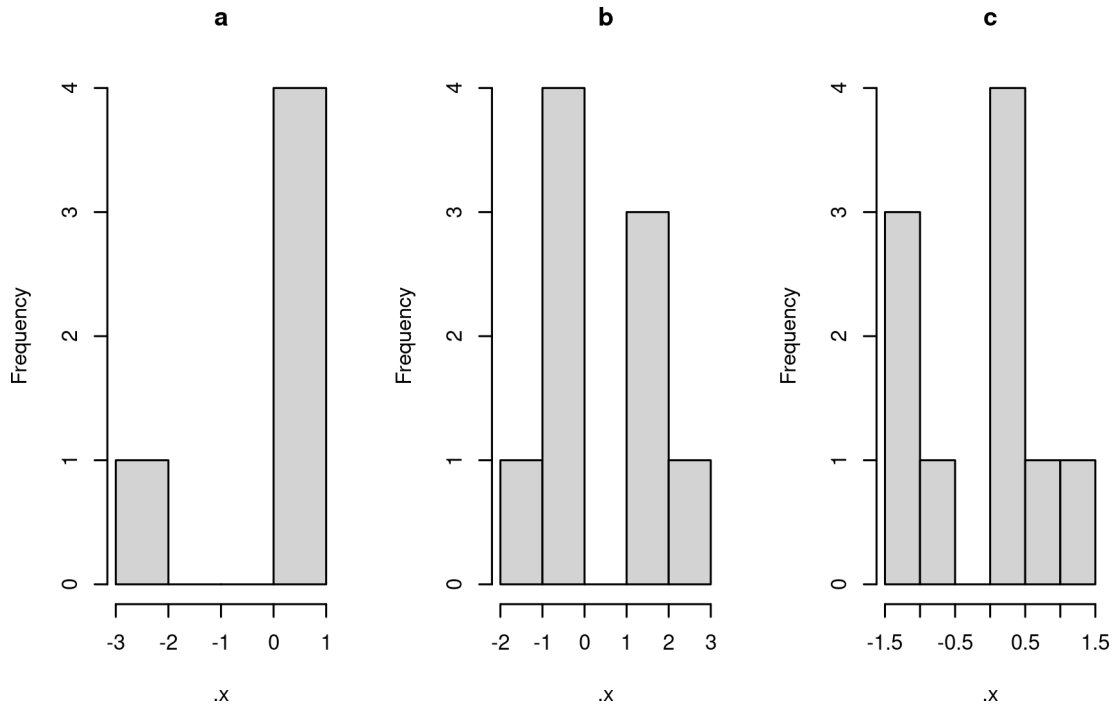
```
## $a
## [1]  2.2336365  2.6387299 -0.4522889  2.7026484  2.8055726
##
## $b
## [1] 3.3834358 4.2914127 1.4431756 3.1799996 1.8437509 0.7042647 3.5716689
## [8] 1.8540075 1.3325151
##
## $c
##  [1] 0.9531327 1.0062795 0.6836400 0.8711971 2.1169419 2.0803037 2.8107524
##  [8] 2.4077072 2.3422151 3.4133593
```

```
par(mfrow = c(1, 3))

walk(listOfNumerics, hist)
```




```
iwalk(listOfNumerics, ~hist(.x, main = .y))
```




```
multipliers <- list(0.5, 10, 3)

map2(.x = listOfNumerics, .y = multipliers, ~.x * .y)
```

```
## $a
## [1]  0.1168182  0.3193650 -1.2261444  0.3513242  0.4027863
##
```

```
## $b
## [1]  13.834358  22.914127  -5.568244  11.799996  -1.562491 -12.957353  15.716689
## [8]  -1.459925  -6.674849
##
## $c
##  [1] -3.1406020 -2.9811616 -3.9490799 -3.3864086  0.3508258  0.2409112
##  [7]  2.4322573  1.2231215  1.0266454  4.2400779
```

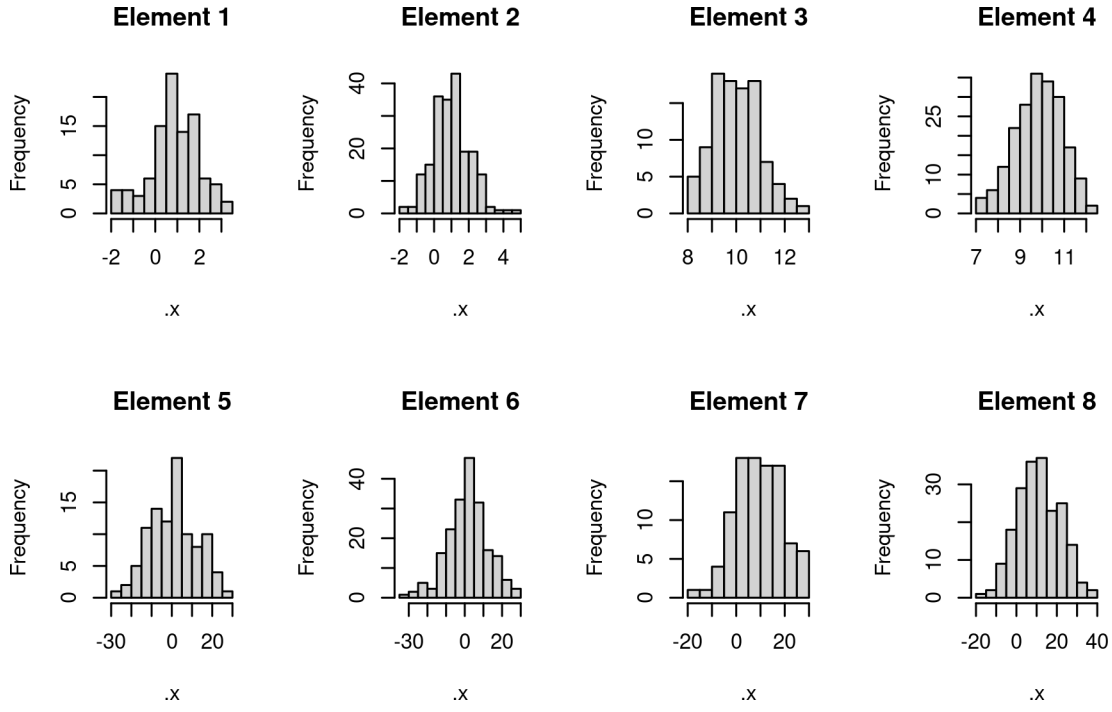
```
arguments <- expand.grid(n = c(100, 200),
                         mean = c(1, 10),
                         sd = c(1, 10))

arguments
```

```
##     n mean sd
## 1 100    1  1
## 2 200    1  1
## 3 100   10  1
## 4 200   10  1
## 5 100    1 10
## 6 200    1 10
## 7 100   10 10
## 8 200   10 10
```

```
par(mfrow = c(2, 4))

pmap(arguments, rnorm) %>%
  iwalk(~hist(.x, main = paste("Element", .y)))
```



7.2.7 Links

[1] Hefin I. Rhys, Machine Learning with R, the tidyverse, and mlr. Shelter Island, New York: Manning Publications, 2020. Available: https://www.manning.com/books/machine-learning-with-r-the-tidyverse-

and-mlr. [Accessed: 20-Oct-2020]