Advanced R

Unit 4

Sereina Herzog

Institute for Medical Informatics, Statistics and Documentation
Medical University of Graz

07.03.2025

Course Content - Advanced R (Unit 4)

- Data import
- Data set format - long vs wide
- Data manipulation - *group_by*
- Data manipulation - join data sets
- Functions in R

Data import

R packages for import

- **readr**
  - to read rectangular data (like csv, tsv, and fwf)
  - is a core package
  - https://readr.tidyverse.org/

R packages for import

- **readr**
  - to read rectangular data (like csv, tsv, and fwf)
  - is a core package
  - https://readr.tidyverse.org/
- **readxl**
  - to read data from Excel (xls, xlsx)
  - is not a core package
  - https://readxl.tidyverse.org/

R packages for import

- **readr**
  - to read rectangular data (like csv, tsv, and fwf)
  - is a core package
  - https://readr.tidyverse.org/
- **readxl**
  - to read data from Excel (xls, xlsx)
  - is not a core package
  - https://readxl.tidyverse.org/
- ... just search, e.g., import .xls to R

Data import

- file type $\rightarrow$ R package and function

Data import

- file type $\rightarrow$ R package and function
- work with arguments in functions
  - e.g., first row contains column headers
  - e.g., type of parameters
  - e.g., which strings to interpret as missing values
  - ...

Useful functions for data preparation

- **select()** extracts columns and returns a tibble

Useful functions for data preparation

- **select()** extracts columns and returns a tibble
- **arrange()** changes the ordering of the rows

Useful functions for data preparation

- **select()** extracts columns and returns a tibble
- **arrange()** changes the ordering of the rows
- **filter()** picks cases based on their values

Useful functions for data preparation

- **select()** extracts columns and returns a tibble
- **arrange()** changes the ordering of the rows
- **filter()** picks cases based on their values
- **mutate()** adds new variables that are functions of existing variables

Examples

In folder *R/Rfiles*

- 0_struma_import_vYYYYMMDD.R
- 0_supraclavicular_import_vYYYYMMDD.R
- 0_supraclavicular_import_SOLUTION_vYYYYMMDD.R

Wide and long

Long/Wide format

- A data set can be written in two different formats: wide and long

Long/Wide format

- A data set can be written in two different formats: wide and long
- A **wide format** has one line for each patient/animal
  - i.e., column with unique identifier has only unique entries

Long/Wide format

- A data set can be written in two different formats: wide and long
- A **wide format** has one line for each patient/animal
  - i.e., column with unique identifier has only unique entries
- A **long format** can have more more than one line for each patient/animal -i.e., column with unique identifier has recurring entries

Long/Wide format - Example 1

id	sex	weight_v1	weight_v2	date_v1	date_v2
P1	m	67	66	2024-01-03	2025-02-03
P2	NA	63	71	2024-04-03	2025-03-03
P3	f	81	69	2024-01-01	2025-01-02
P4	x	88	88	2024-02-06	2025-02-10

Long/Wide format - Example 1

id	sex	visit	weight	date
P1	m	v1	67	2024-01-03
P1	m	v2	66	2025-02-03
P2	NA	v1	63	2024-04-03
P2	NA	v2	71	2025-03-03
P3	f	v1	81	2024-01-01
P3	f	v2	69	2025-01-02
P4	x	v1	88	2024-02-06
P4	x	v2	88	2025-02-10

Long/Wide format - why?

- each format has advantages and is more useful for certain tasks

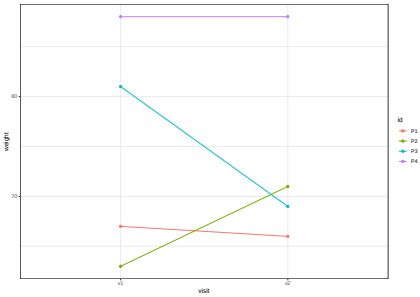
Long/Wide format - why?

- each format has advantages and is more useful for certain tasks
- long format
  - repeated measurements
    - especially if number of repetitions differ per patient/animal
    - long format needed for plots over repeated measurements
  - can be more efficient in terms of storage space

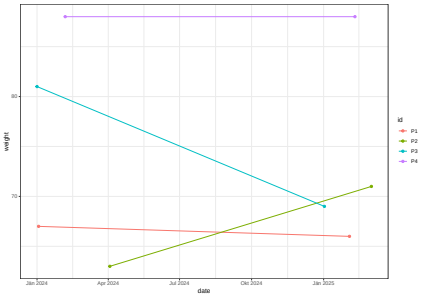
Long/Wide format - why?

- each format has advantages and is more useful for certain tasks
- long format
  - repeated measurements
    - especially if number of repetitions differ per patient/animal
    - long format needed for plots over repeated measurements
  - can be more efficient in terms of storage space
- wide format
  - easier to read and look up a patient/animal
  - often easier to make calculations (e.g., BMI)

Long/Wide format - Example 1 plot with visit



Long/Wide format - Example 1 plot with dates



Long/Wide format - reshape

in R package *tidyr*

- **pivot_wider()** "widens" data, increasing the number of columns and decreasing the number of rows

Long/Wide format - reshape

in R package *tidyr*

- **pivot_wider()** "widens" data, increasing the number of columns and decreasing the number of rows
- **pivot_longer()** "lengthens" data, increasing the number of rows and decreasing the number of columns

Example - pivot_wider()

```
dt_example <-
  tibble(id = rep(paste0("P", 1:3), 2),
         sys = c(120,118,NA, 125,125,110),
         measurement = c(1, 1, 1, 2, 2, 2))

dt_example %>%
    pivot_wider(data = ., names_prefix = "sys",
                names_from = measurement,
                values_from = sys)
```

Example - pivot_wider() - output

```
## # A tibble: 6 x 3
##   id      sys measurement
##   <chr> <dbl>       <dbl>
## 1 P1      120           1
## 2 P2      118           1
## 3 P3       NA           1
## 4 P1      125           2
## 5 P2      125           2
## 6 P3      110           2
```

```
## # A tibble: 3 x 3
##   id     sys1  sys2
##   <chr> <dbl> <dbl>
## 1 P1      120   125
## 2 P2      118   125
## 3 P3       NA   110
```

Example - pivot_longer()

```
dt_example <-
  tibble(id = paste0("P", 1:4),
         age = c(224, 31,50,26),
         weight = c(67, 63, 81, 88))

dt_example %>%
  pivot_longer(data = ., cols = c(age, weight))
```

Example - pivot_longer() - output

```
## # A tibble: 4 x 3
##   id      age weight
##   <chr> <dbl>  <dbl>
## 1 P1      224     67
## 2 P2       31     63
## 3 P3       50     81
## 4 P4       26     88
```

```
## # A tibble: 8 x 3
##   id    name   value
##   <chr> <chr>  <dbl>
## 1 P1    age      224
## 2 P1    weight    67
## 3 P2    age       31
## 4 P2    weight    63
## 5 P3    age       50
## 6 P3    weight    81
## 7 P4    age       26
## 8 P4    weight    88
```

Exercise pivot

- Work through ‘Unit 4 - Exercise 1’ (no pdf)
  - *UNIT4_ex1_pivot_vYYYYMMDD.Rmd*

Data manipulation - *group_by*

Data manipulation - *group_by*

- **group_by**: allows you to group by a one or more variables

```
iris %>%  group_by(Species)
```

```
## # A tibble: 150 x 5
## # Groups:   Species [3]
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            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           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # i 140 more rows
```

Data manipulation - *group_by* with *tally()*

```
iris %>%  group_by(Species) %>%  tally()
```

```
## # A tibble: 3 x 2
##   Species        n
##   <fct>      <int>
## 1 setosa        50
## 2 versicolor    50
## 3 virginica     50
```

Data manipulation - *group_by* with *mutate()*

```
iris %>%  group_by(Species) %>%  mutate(n = n(), mean_SL = mean(Sepal.Length))
```

```
## # A tibble: 150 x 7
## # Groups:   Species [3]
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species     n mean_SL
##           <dbl>       <dbl>        <dbl>       <dbl> <fct>   <int>   <dbl>
##  1          5.1         3.5          1.4         0.2 setosa     50    5.01
##  2          4.9         3            1.4         0.2 setosa     50    5.01
##  3          4.7         3.2          1.3         0.2 setosa     50    5.01
##  4          4.6         3.1          1.5         0.2 setosa     50    5.01
##  5          5           3.6          1.4         0.2 setosa     50    5.01
##  6          5.4         3.9          1.7         0.4 setosa     50    5.01
##  7          4.6         3.4          1.4         0.3 setosa     50    5.01
##  8          5           3.4          1.5         0.2 setosa     50    5.01
##  9          4.4         2.9          1.4         0.2 setosa     50    5.01
## 10          4.9         3.1          1.5         0.1 setosa     50    5.01
## # i 140 more rows
```

Data manipulation - *group_by* with *summarize()*

- **summarize**: creates a new data.frame containing calculated summary information about a grouped variable

```
iris %>%  group_by(Species) %>%  summarize(n = n(), mean_SL = mean(Sepal.Length), mean_SW = mean(Sepal.Width))
```

```
## # A tibble: 3 x 4
##   Species        n mean_SL mean_SW
##   <fct>      <int>   <dbl>   <dbl>
## 1 setosa        50    5.01    3.43
## 2 versicolor    50    5.94    2.77
## 3 virginica     50    6.59    2.97
```

Data manipulation - *group_by* with *mutate()* and long format data

```
## # A tibble: 11 x 4
##    id    sex   visit weight
##    <chr> <chr> <chr>  <dbl>
##  1 P1    m     v1        67
##  2 P1    m     v2        66
##  3 P1    m     v3        68
##  4 P2    <NA>  v1        63
##  5 P2    <NA>  v2        71
##  6 P3    f     v1        81
##  7 P3    f     v2        69
##  8 P3    f     v3        77
##  9 P4    x     v1        88
## 10 P4    x     v2        88
## 11 P4    x     v3        90
```

Data manipulation - *group_by* with *mutate()* and long format data

```
dt_example_long %>%  group_by(id) %>%  mutate(n = n(), diff_w_v1 = weight - weight[1])
```

```
## # A tibble: 11 x 6
## # Groups:   id [4]
##    id    sex   visit weight     n diff_w_v1
##    <chr> <chr> <chr>  <dbl> <int>     <dbl>
##  1 P1    m     v1        67     3         0
##  2 P1    m     v2        66     3        -1
##  3 P1    m     v3        68     3         1
##  4 P2    <NA>  v1        63     2         0
##  5 P2    <NA>  v2        71     2         8
##  6 P3    f     v1        81     3         0
##  7 P3    f     v2        69     3       -12
##  8 P3    f     v3        77     3        -4
##  9 P4    x     v1        88     3         0
## 10 P4    x     v2        88     3         0
## 11 P4    x     v3        90     3         2
```

Exercise data manipulation with group_by

- Work through 'Unit 4 - Exercise 2' (no pdf)
  - *UNIT4_ex2_dm_groupbby_vYYYYMMDD.Rmd*

Data manipulation - join data sets

Data manipulation - join data sets

- data analysis involves rarely only one single data frame
- having several data frames means that we must join them together
  - using keys, i.e., which observations belong together

Data manipulation - join data sets

- data analysis involves rarely only one single data frame
- having several data frames means that we must join them together
  - using keys, i.e., which observations belong together

- two important types of joins:
  - **mutating joins**: add new variables to one data frame from matching observations in another
  - filtering joins: filter observations from one data frame based on whether or not they match an observation in another

Data manipulation - join data sets (dplyr package)

How to join (merge) data frames (inner, outer, left, right):

- assume we have two datasets: **x** and **y**

- **inner_join()**: only keeps observations from x that have a matching key in y
- **full_join()**: keeps all observations in x and y
- **left_join()**: keeps all observations in x
- **right_join()**: keeps all observations in y

Data manipulation - join data sets - Example

Dataset *A*:

```
## # A tibble: 4 x 2
##   key1   par2
##   <chr> <dbl>
## 1 A         1
## 2 B         2
## 3 C         3
## 4 D         4
```

Dataset *B*:

```
## # A tibble: 3 x 2
##   key1  par3
##   <chr> <chr>
## 1 B     no
## 2 C     no
## 3 A     yes
```

Data manipulation - join data sets - Example

```
inner_join(x = A, y = B, by = c("key1" = "key1"))
```

```
## # A tibble: 3 x 3
##   key1   par2 par3
##   <chr> <dbl> <chr>
## 1 A         1 yes
## 2 B         2 no
## 3 C         3 no
```

```
full_join(x = A, y = B, by = c("key1" = "key1"))
```

```
## # A tibble: 4 x 3
##   key1   par2 par3
##   <chr> <dbl> <chr>
## 1 A         1 yes
## 2 B         2 no
## 3 C         3 no
## 4 D         4 <NA>
```

Data manipulation - join data sets - Example

```
left_join(x = A, y = B, by = c("key1" = "key1"))
```

```
## # A tibble: 4 x 3
##   key1   par2 par3
##   <chr> <dbl> <chr>
## 1 A         1 yes
## 2 B         2 no
## 3 C         3 no
## 4 D         4 <NA>
```

```
right_join(x = A, y = B, by = c("key1" = "key1"))
```

```
## # A tibble: 3 x 3
##   key1   par2 par3
##   <chr> <dbl> <chr>
## 1 A         1 yes
## 2 B         2 no
## 3 C         3 no
```

Exercise data manipulation joining data sets

- Work through ‘Unit 4 - Exercise 3’ (no pdf)
  - *UNIT4_ex3_dm_joindatasets_vYYYYMMDD.Rmd*

Functions in R

Functions in R

We are constantly working with functions

Functions in R

We are constantly working with functions

- a function takes argument(s)
- some argument(s) are mandatory
- some argument(s) have default values

Functions in R

We are constantly working with functions

- a function takes argument(s)
- some argument(s) are mandatory
- some argument(s) have default values

```
mean(x = c(1, 2, 3))
```

Functions in R

We are constantly working with functions

- a function takes argument(s)
- some argument(s) are mandatory
- some argument(s) have default values

```
mean(x = c(1, 2, 3))
```

- a function returns a ‘value’ (e.g., value, ggplot, . . .)

Example mean()

mean() - calculate the mean value of a vector

Example mean()

mean() - calculate the mean value of a vector

- mean() has several arguments

Example mean()

mean() - calculate the mean value of a vector

- mean() has several arguments
- x: numeric/logical vectors (others also possible, e.g. time intervals)

Example mean()

mean() - calculate the mean value of a vector

- mean() has several arguments
- x: numeric/logical vectors (others also possible, e.g. time intervals)
- trim: the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed

Example mean()

mean() - calculate the mean value of a vector

- mean() has several arguments
- x: numeric/logical vectors (others also possible, e.g. time intervals)
- trim: the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed
- na.rm: a logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before the computation proceeds

Example mean()

mean() - calculate the mean value of a vector

- mean() has several arguments
- x: numeric/logical vectors (others also possible, e.g. time intervals)
- trim: the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed
- na.rm: a logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before the computation proceeds
- ... : further arguments passed to or from other methods

Example mean()

mean() - calculate the mean value of a vector

- mean() has several arguments
- x: numeric/logical vectors (others also possible, e.g. time intervals)
- trim: the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed
- na.rm: a logical evaluating to TRUE or FALSE indicating whether NA values should be stripped before the computation proceeds
- ... : further arguments passed to or from other methods

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

Write your own function

```
myFunction <- function() {

}
```

- object *myFunction* is now a function

Write your own function

```
myFunction <- function() {

}
```

- object *myFunction* is now a function
- () → currently we have no arguments

Write your own function

```
myFunction <- function() {

}
```

- object *myFunction* is now a function
- () → currently we have no arguments
- {} is the place for all commands we want

Write your own function

```r
myFunction <- function(name) {
   greetings <- paste0("Hallo, ", name, "!")
   cat(greetings)
}
```

- *name* is our argument
- {} creates the *greetings* and returns the last line

Write your own function

```
myFunction <- function(name) {
   greetings <- paste0("Hallo, ", name, "!")
   cat(greetings)
}
```

- *name* is our argument
- {} creates the *greetings* and returns the last line

```
myFunction(name = "Susi")
```

```
## Hallo, Susi!
```

Write your own function - Example

```r
CreateSampleDataset <- function(nrow = 100) {
    Condition <- rbinom(n = nrow, size = 1, prob = 0.5)
    IQ <- rnorm(n = nrow, mean = 100, sd = 15)
    Age <- rnorm(n = nrow, mean = 40, sd = 7.5)
    Motivation <- runif(n = nrow, min = 1, max = 10)
    dfSampleData <- tibble(Condition, IQ, Age, Motivation)
    return(dfSampleData)
}
```

Write your own function - Example

```r
myFunction2 <- function(x = 0) {
   x-3
   return(c(answer = 42))
}
```

Write your own function - Example

```
myFunction2 <- function(x = 0) {
   x-3
   return(c(answer = 42))
}
```

```
myFunction2()
```

```
## answer
##     42
```

Write your own function - Example

```
myFunction2 <- function(x = 0) {
   x-3
   return(c(answer = 42))
}
```

```
myFunction2()
```

```
## answer
##     42
```

- *return()*
  - variable can be any R object
  - a function in R can only ever return one object (use list() for more)

Remarks

- Functions are used when the same or similar program code is used in several places in the script

Remarks

- Functions are used when the same or similar program code is used in several places in the script
- Variables declared inside a function are local to that function

Remarks

- Functions are used when the same or similar program code is used in several places in the script
- Variables declared inside a function are local to that function
- Function name should make it clear what the function is good for
- Inputs and outputs should be clear

Exercise function writting

- Work through ‘Unit 4 - Exercise 4’ (no pdf)
  - *UNIT4_ex5_functions_vYYYYMMDD.Rmd*

Links

Links (I)

- Introduction to R
  - R for Data Science (https://r4ds.hadley.nz/)
- Plots using ggplot
  - Overview with further links to course material: https://ggplot2.tidyverse.org/
- Display tables using flextable
  - flextable bool https://ardata-fr.github.io/flextable-book/
  - Function references https://davidgohel.github.io/flextable/reference/index.html
- knit_child()
  - link (https://bookdown.org/yihui/rmarkdown-cookbook/child-document.html)

Links (II)

- Download R
  - CRAN (https://cran.r-project.org/)
- Download RStudio
  - RStudio Desktop (https://posit.co/download/rstudio-desktop/)