Part III - Manipulating data with tidyverse



- How to import and export data in R
- How to inspect imported data
- Manage files & folders
- Tips on Warning/errors
- R documentation and manuals
- aggregating and analyzing data with dplyr

dplyr library

What is **dplyr**?

- -dplyr is a R package that simplifies data manipulation through additional and intuitive functions
- Different functions are available for different tasks
- dplyr can work in "PIPE" simplifying complex workflows

most useful functions:

```
select()
filter()
arrange()
mutate()
group_by()
summarize()
left_join()
```

Let's learn first how to import data in R!

data import

Import your data in R! By the graphic interface or ... by the command line (far more quick and reproducible!)

• several functions: read.csv() read.delim() read.table()

```
> your dataframe <- read.table(x = "^{-}.txt", sep = "^{+}t", dec = ".", header = T)
```

- pay attention to your decimal separator (should be both "." or ",") and your field separator (must be different!)
- > str(your dataframe)
 - Check your imported table: numbers should be imported as numbers, texts as character
- > View(your_dataframe)
 - visually inspect and scroll your table!

data import

Exercise! Import the Flowers dataset (Flowers.txt) and inspect it

```
# download the flowers dataset from the course repository
>download.file("https://raw.githubusercontent.com/mchialva/PhDToolbox2023/main/Datasets/flowe
rs.tsv", "flowers.tsv")
# import dataset
> flowers<-read.table("flowers.txt")
# View table
> View(flowers)
# Check data structure
> str(Flowers)
```

dataframe content inspection

Other ways to check your imported object

```
# display only the first rows
> head(your dataframe)
# display only the final rows
> tail(your dataframe)
# display dimensions of your table (number of rows and columns)
> dim(your dataframe) # returns a vector of length 2 with rows
and columns number
> nrow(your dataframe) # returns the number of rows
> ncol(your dataframe) # returns the number of columns
```



dataframe content inspection

display descriptive statistics for each column in the dataframe

> **summary**(your_dataframe)

Exercise!

- 1) How many columns and rows are in the Flowers dataset?
- 2) Select species with blue flowers (COL) and stem size (SSIZE) higher than 10 and save in the new object "Flowers_blue10".
- 3) How many species?

Basic dataframe handling

We already explained cbind(), rbind() and how to remove entire columns using NULL, but we can also create new columns:

```
>df$colname_4 <- 1 # create a new column full of 1s
>df$colname_5 <- ifelse(df$colname_1 == "A", 50, 120) # create a new column with
dependent values
>table(df$colname_1) # very cool function to see frequencies of a factor
```

Exercise:

Create a new column in df called colname_4. We need that where in colname_3 values are higher or equal to 8 in colnames_4 there is the character "small", in all the other case there is "bigger".

Calculate frequencies of factors in colnames_4.

data export

• write a dataframe into a delimited text file using write.table() function

```
> write.table(x = df, file = "\sim/.txt", sep = "\setminust", row.name = F)
```

Exercise!

Export Flowers blue10 object into a tsv table

data export

• write a dataframe into a delimited text file using write.table() function

```
> write.table(x = df, file = "\sim/.txt", sep = "\setminust", row.name = F)
```

Exercise!

Export Flowers blue10 object into a tsv table

```
> write.table(x = Flowers_blue10, file = "Flowers_blue10.tsv", sep =
"\t", row.name = F)
```

tidyverse

tidyverse

>install.packages("tidyverse")



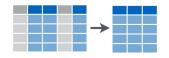
dplyr: now

stringr: next topic

ggplot2: Stream 2

dplyr: subset columns

Subset Variables (Columns)



select(dataframe, columns to keep)

library(dplyr)

- > select(flowers, -IUCN) # remove column
- > select(flowers, starts_with("Q") # select columns which start with "Q"
- > select(flowers, IUCN:NIT) # select contiguous columns
- > **select**(flowers, SSIZE, everything()) # reorder columns

dplyr: subset rows

Subset Observations (Rows)



filter(dataframe, subsetting_conditions)

library(dplyr)

> filter(flowers, SSIZE>10) # select observations with stem size higher than 10

> filter(flowers, CIT==20) # select observations with 20 citations

> filter(flowers, SSIZE<5 | (FSIZE>50 & FLEN<2)) # multiple logical criteria

logical operators are the same as seen before, except:

NOT in: ! column_names %in% vector

dplyr: work in pipe

Task: I want to filter and select columns at the same time

three ways to do this:

- create intermediate object
- nested functions

select(filter(flowers, SSIZE>10), SP, starts_with("Q"))

- PIPES: Take the output of a function and send it to the next one using the %>% operator

> my_selection<-flowers %>% filter(SSIZE>10) %>% select(SP, starts_with("Q")

TIP: to visualize on the fly the operation you are performing on your data frame, append the View()

function to your PIPE!



Notes:

- pipes are made available through the package magrittr (installed automatically with dplyr)
- input data needs to be given only at the very beginning of the PIPE or in the first function
- no intermediate objects are created
- to make permanent your operations you still need to write your pipe into an object!
- there are no limits in the number of functions you can pipe

dplyr: work in pipe

Exercise!

- Filter the flowers dataset by SSIZE>10 and QRANGE from 0 to 600
- select the following columns (SP, NIT and all columns after (and including) QMIN)
- count how many columns and rows are in the subsetted dataframe
- do it with PIPES!

dplyr: work in pipe

solution

[1] 8 22

```
> flowers %>%

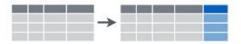
filter(SSIZE>10 & QRANGE %in% seq(0:600)) %>%

select(SP, NIT, QMIN, everything()) %>%

dim()
```

dplyr: mutate

Make New Variables



df %>% mutate(new_column_name = content)

Add a column containing the flower-to-stem ratio(FSR=FSIZE/SSIZE)

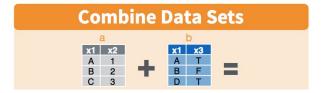
> flowers %>% mutate(FSR=FSIZE/SSIZE) %>% View()

add column using conditional rules with ifelse() or if_else(TEST, if TRUE, if FALSE)

add the column QRANGE1000 assigning to category "A" all the species with value higher than 1000 and the other to "B"

> flowers %>% mutate(QRANGE1000=if_else(QRANGE>1000, "A", "B")) %>% View()

dplyr: left_join()



df %>% left_join(df2, index)

We need to join the flowers dataset with flowers_details.tsv table which contains additional attributes

>download.file("https://raw.githubusercontent.com/mchialva/PhDToolbox2023/main/Datasets/flowers_det ails.tsv", "flowers_details.tsv")

- # import table
- > flowers_details<-read.table("Datasets/flowers_details.tsv", header=T)
- # join (or merge) and save into a new object
- > flowers_all<- flowers %>% left_join(flowers_details, by="SP")





df %>% left_join(df2, index)

different functions available for "merging":

left_join() keeps all records from the left table (df), and the matched records from the right table (df2)

right_join() keeps all records from the right table (df), and the matched records from the left table (df2)

full_join() keeps all observation in left and right table

dplyr: summarize()

```
df %>% summarize(new_column_name = content, ...)
```

Summarize min, mean and maximum SSIZE values

```
> flowers_all %>% summarize(min_SSIZE=min(SSIZE),
```

mean_SSIZE=**mean**(SSIZE),

max_SSIZE=max(SSIZE)) %>% View()

min_SSIZE mean_SSIZE max_SSIZE

1 1.5 29.40487 400

dplyr: summarize()

Task: I need to summarize values by categories.

E.g. which is the minimum and maximum flower size (FSIZE) for each plant family in the dataset?

group_by() groups by single or multiple factors

df %>% group_by() %>% summarize(new_column_name = content, ...)

Summarize min, mean and maximum SSIZE values

> flowers_all %>% group_by(FAM) %>% summarize(min_FSIZE=min(FSIZE),

max_FSIZE=max(FSIZE))

dplyr: group_by()

If called alone, the function **group_by()** does not produce any evident output (it is silently grouping your observation)

- you can summarize the number of observations in each of the grouped category by piping the function tally()

> flowers_all %>% group_by(FAM) %>% tally()

This command lists how many observations are for each Family in the dataset.

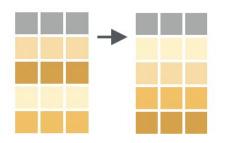


Tips: Some of dplyr functions coerce your output into a <u>tibble object</u> (*tbl* class object) which easily fits in your terminal when there are too many observations.

- you may need to convert your output into a data frame: pipe as.data.frame() function

you may need to visualize more rows: pipe print(n=rows_to_show)

dplyr: arrange



df %>% arrange(columns to order by)

- # Order the data frame flowers_all by increasing FSIZE value
- > flowers_all %>% arrange(FSIZE)
- # Order the dataframe flowers all by decreasing FSIZE value
- > flowers_all %>% arrange(-FSIZE)
- # if applied to character vectors the function orders in alphabetic order
- > flowers_all %>% arrange(COL)

dplyr: across

Iterate function across columns: to be used with **mutate**() or with **summarize**() much more complex and <u>variable syntax</u>

summarise

```
> flowers_all %>% group_by(FAM) %>%
    summarise(across(SSIZE:FLEN, list(mean = mean, sd = sd, max=max, min=min)))
    %>% View()
```

dplyr: across

Iterate function across columns: to be used with **mutate()** or with **summarize()**

much more complex and variable syntax

summarise with different arguments to set output column names

> flowers_all %>% group_by(FAM) %>%

summarise(across(starts_with("Q"), mean, .names = "mean_{.col}")) %>% View()

dplyr: across

```
Iterate function across columns:
                            to be used with mutate() or with summarize()
much more complex and variable syntax
# mutate (perform operation within columns
> flowers_all %>% group_by(FAM) %>%
 mutate(across(c(QRANGE, FLEN), ~ .x + 1)) %>% View()
# same but creates new columns and keep the original ones
> flowers_all %>% group_by(FAM) %>%
 mutate(across(c(QRANGE, FLEN), \sim .x + 1, .names = "plus1_{.col}")) %>% View()
```



Exercise!

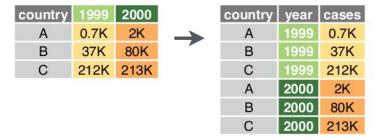
- Take as input the *flowers_all* dataset
- Summarize (minimum, mean and maximum values) flower dimensions (FLEN) by color (COL), habitat (HBT) and biology (FBIO) excluding Asteraceae and Crassulaceae families
- sort observations by decreasing mean value
- How many observations are there?

TIP: If you get stuck in the pipe run your code function by function and look at the output. This will guide you on how to modify or tune the next function.

dplyr: complex exercise 1

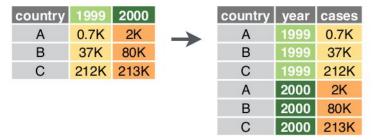
Solution:

```
> flowers_all %>% filter(!FAM %in% c("Asteraceae", "Crassulaceae")) %>%
                   group_by(COL,HBT) %>%
                   summarize(
                                 min=min(FLEN),
                                 mean=mean(FLEN),
                                 max=max(FLEN)
                                                    ) %>%
                   %>% arrange(-mean) %>%
                   print(n=100)
```



df %>% pivot_longer(column/s_to_pivot, ...)

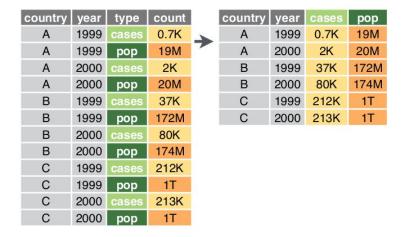
- increases the number of rows and decreases the number of columns (short to long data format).
- known as "melting": previous (but deprecated functions) are reshape::melt() and dplyr::gather()
- layout required by some data analysis (including ggplot2)



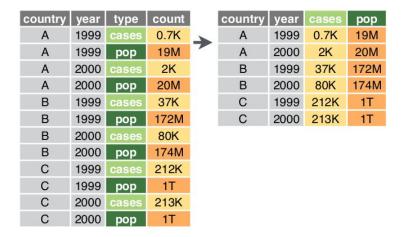
df %>% pivot_longer(column/s_to_pivot, ...)

```
# melt columns CIT and H
> flowers_all %>% pivot_longer(c(CIT, H)) %>% View()
# we can also control the names of created columns
```

> flowers_melted<-flowers_all %>% pivot_longer(c(CIT, H), names_to = "variable", values_to =
"value")



- increases the number of columns and decreases the rows (long to short data format).
- known as "casting": previous (but deprecated functions) are data.table::dcast() and dplyr::spread()



> flowers_melted %>% pivot_wider(names_from = "variable", values_from = "value") %>% View()

dplyr exercise 2

download the survey dataset (3 tables) from the course repository or from Moodle

```
>
download.file("https://raw.githubusercontent.com/mchialva/PhDToolbox2023/main/Datasets/flowers.tsv",
"flowers.tsv")
>
download.file("https://raw.githubusercontent.com/mchialva/PhDToolbox2023/main/Datasets/flowers.tsv",
"flowers.tsv")
>
download.file("https://raw.githubusercontent.com/mchialva/PhDToolbox2023/main/Datasets/flowers.tsv",
"flowers.tsv")
```

Note: You must be always aware of the format of the data you are importing specially if you do not know the dataset (not generated by you!). If you have doubts on how data has been formatted give it a look using a text editor before importing into R

- Which is the field separator?

dplyr exercise 2

- # import all the three datasets
- > surveys<-read.table("surveys.csv", sep=",", header=T)
- > plots<-read.table("plots.csv", sep=",", header=T)
- > species<-read.table("species.csv", sep=",", header=T)

Now you are almost ready to begin the exercise!

dplyr exercise 2

Tasks

1) Join the table *surveys* with *plots* and *species* ensuring that all observations from surveys are maintained and save the final merged table into an object called *surveys_complete*.

```
surveys_complete<-surveys %>% left_join(species, by="species_id") %>% left_join(plots, by="plot_id")
```

2) Add Rodents observed between 2001 and 2002, excluding in a new object called *rodents*

```
rodents<-surveys_complete %>% filter(year %in% seq(1998,2002, by=1) & taxa=="Rodent" & taxa=="Rodent" & species id != "DM")
```



Exercise!

- 3) Do the following in one step (using PIPE) and put the final output into an object called **survey_subset**
 - Melt the rodents dataset using columns hindfoot_length and weight
 - remove missing values in both the parameters (help: use !is.na() function) and in sex column
 - keep your information separate for each combination of year, species_id, sex and variable
 - summarize the mean and maximum parameter values

dplyr: complex exercise 2

Solution!

3) survey_subset<- rodents %>%

```
pivot_longer(c(hindfoot_length, weight), names_to =
"variable") %>%
filter(!is.na(value) & sex %in% c("M", "F")) %>%
group_by(year, species_id, sex, variable) %>%
summarize(mean=mean(value), max=max(value))
```

Text strings



Why strings are so important?

- Strings (character vectors) plays crucial role in annotating your data
- Data annotation can change or grow during your project
- Strings manipulation in an automated and reproducible way is highly advisable: **do never modify your raw data**!

How can I manage strings in R?

- In R you can easily manipulate strings with the stringr (as dplyr, it is within the tidyverse package collection)
- stringr functions works in PIPES!

Text strings with stringr

Some of the most useful stringr() functions

str_sub() # substring text

str_length() # extract lenghts

str_split_fixed() # split column by patterns

str_locate_all() # match patterns

str_replace() # replace patterns

str_c() and str_flatten() # paste and combine strings



Stringr: str_length()

Let's write (again) a simple vector

```
> animals<-c("cat", "dog", "rabbit", "duck", "monkey", "fish")
str_length(vector)</pre>
```

The functions outputs a vector of the same length of your input vector showing strings lengths (all characters are counted, spaces included)

> str_length(animals)

[1] 3 3 6 4 6 4

Stringr: str_sub()

str_sub(vector)

The function exports a piece of your strings. It's really useful to cut or subset strings.

```
> str_sub(animals, 1, 1)
```

```
[1] "c" "d" "r" "d" "m" "f"
```

extract from the second to the fourth character of each element

> str_sub(animals, 2, 4)

[1] "at" "og" "abb" "uck" "onk" "ish"

More complex subsetting patterns and usage are detailed in package vignette

Stringr: str_c()

collapse multiple variables (or vectors) in one column (or one vector): sum text vectors

str_c(vector_1, vector_2, vector_n, separator)

> str_c(animals, animals, sep = "_")

[1] "cat_cat" "dog_dog" "rabbit_rabbit" "duck_duck" "monkey_monkey" "fish_fish"

Note: columns are vectors, when working on data frames you work with vectors!

Stringr: str_flatten()

collapse one vector or multiple vectors in one string

str_flatten(c(vector_1, vector_2, vector_n), separator))

>str_flatten(animals, collapse = ", ")

[1] "cat, dog, rabbit, duck, monkey, fish"

Stringr: str_replace()

replace strings according to a pattern

str_replace(string, pattern to replace, replacement)

NB: finds and replace only the first match

This functions can be used in different ways

```
1) > str_replace(animals, "a", "*") # simple replacement

[1] "c*t" "dog" "r*bbit" "duck" "monkey" "fish"
```

Stringr: str_replace() & str_replace_all()

replace strings according to a pattern

str_replace(string, pattern to replace, replacement)

str_replace_all() finds and replace all the pattern matched

```
2) > str_replace(animals, c("a", "g", "b", "k", "m", "s"), "-") # multiple replacement along vector
```

[1] "c-t" "do-" "ra-bit" "duc-" "-onkey" "fi-h"

3) > str_replace_all(animals, c("cat"="first", "monkey"="fifth")) # multiple replacement on same vector

[1] "first" "dog" "rabbit" "duck" "fifth" "fish"

stringr exercise

Parse *flowers_all* dataset according to the following rules

- Add a column called SP1 with the first 10 character of column SP
- create a new column called COL_EN with color names in COL translated in English

TIP: use unique() to find out which are the unique strings that you'll need to replace with their English names.

stringr exercise

Solution

Parse *flowers_all* dataset according to the following rules

- Add a column called SP1 with the first 10 character of column SP
- create a new column called COL_EN with color names in COL translated in English

Stringr: str_split_fixed()

split vectors according to a pattern:

str_split_fixed(string, pattern to find, number of fields to split)

```
> str_split_fixed(c("A_B", "C_D"), "_", 2)
[,1] [,2]
[1,] "A" "B"
```

[2,] "C" "D"

The function outputs a matrix you can append it to your dataframe using cbind() function

Stringr: str_split_fixed() through separate()

However, a simpler dplyr() function wraps it

df %>% separate(column_to _split, vector_of_new_colnames, remove=T/F)

remove tells dplyr if the selected column should be keeped or removed

Stringr: str_locate()

locate position of a pattern in strings

str_locate(string, pattern to find)

Note: this function locates only the first occurrence of the pattern. More useful with unique patterns.

```
> str_locate(c("A_B", "C_D"), "_", 2)
[,1] [,2]
[1,] "A" "B"
[2,] "C" "D"
```

The function outputs a <u>matrix</u>

stringr exercise

Parse *flowers_all* dataset according to the following rules

- Divide the SP column into genus and species
- Add a **species_id** column coding each species with two capital letters plus the second letter of the species

e.g. Aquilegia alpina should be annotated as Aal



Solution

```
flowers_all %>% separate(SP, c("Genus", "Species"), sep=" ", remove = F) %>% mutate("species_id"=str_c(str_sub(Genus, 1, 1), str_sub(Species, 1, 2))) %>% View()
```