Rbasics

PhD toolbox - 41th PhD cycle

Part III - Data Manipulation with tidyverse

- How to import and export data in R
- How to check & inspect imported data
- aggregating and analyzing data with dplyr
- a few function for manipulating strings
- Practice!

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/PhDToolbox2026/main/Datasets/flowers/flowers.tsv", "flowers.tsv")
```

import dataset

```
>flowers<-read.table("flowers.tsv", header=T)
```

View table
```
>View(flowers)
```

Check data structure
```
>str(flowers)
```

dataframe content inspection

Other ways to check your imported object

```
# show the object class and structure

>str(your_dataframe)

# 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
```

The flowers dataset

```
>head(your_dataframe)
```

These dataset is from the study “Plant scientists’ research attention is skewed towards colourful, conspicuous and broadly distributed flowers” (Adamo, Chialva et al., 2021) about the preference of part of the scientific community to study “beautiful plants”. The dataset is composed of several species-specific traits:

```
"SP"= species name
"IUCN" = extinction risk
"LUM", "UMD", "TEM", "CON", "REA", "HUM", "TEX", "NIT" = Landolt Indexes
(plant’s ecology)
"SSIZE", "FSIZE" = Body size
"FLEN" = Flowering duration
"QMIN", "QRANGE", "QMAX" = Altitudinal range descriptors
"AREA", "RANGE_DISPERSION" = distribution
"CONGEN"  = number of congeneric species
"PAPER" = number of published studies
"CIT" = citations per study
"H" = plant’s H index
```

dataframe content inspection

display descriptive statistics for each column in the dataframe

> **summary**(your_dataframe)

Review Exercise!

1) How many columns and rows are in the Flowers dataset?
2) Select species with QRANGE different from 1400 and stem size (SSIZE) higher than 10 and save in the new object "flowers_filtered".
3) How many species?

Review Exercise

1) How many columns and rows are in the Flowers dataset?

```
> dim(flowers)
[1] 113  22
```

2) Select species with QRANGE different from 1400 and stem size (SSIZE) higher than 10 and save in the new object "flowers_filtered".

```
>flowers_filtered<-subset(flowers, QRANGE !=1400 & SSIZE>10)
```

```
>flowers_filtered<-flowers[flowers$QRANGE!=1400 & flowers$SSIZE>10,]
```

3) How many species?

```
> length(unique(flowers_filtered$SP))
```

```
[1] 75
```

data export

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

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

Exercise!

Export flowers_filtered object into a tsv table

data export

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

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

Exercise!

Export flowers_filtered object into a tsv table

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

tidyverse

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

8 core packages

- intuitive and consistent functions through different packages
- require less code and allow complex data manipulation
- reproducible
- highly documented
- more efficient computation (for larger tables)
- functions work “in pipe”

dplyr: now

stringr: next topic

ggplot2: Stream 2

dplyr library: A Grammar of Data Manipulation

What is **dplyr**?

- **dplyr** is a R package that simplifies data manipulation

- 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()

dplyr: shortcuts functions to subset columns

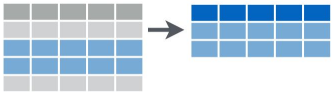
Subset Variables (Columns)

select(dataframe, columns to keep)

```
library(dplyr)
> select(flowers, SSIZE, AREA, CIT)        # select non contiguous columns
> 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

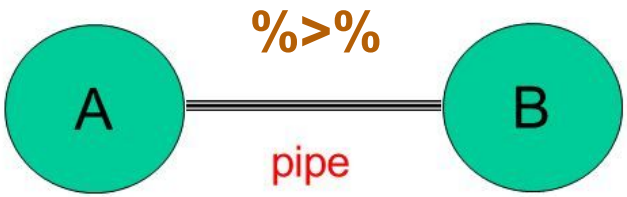
> **filter**(flowers, CIT==20) # select observations with 20 citations

> **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

Pipes!



1) Derives from UNIX language
2) Takes the previous object (or the output of the previous functions) and pushes it as an input for the following function

Example:

```
> flowers %>% dim()

[1] 113  22
```

dplyr: work in pipe

Notes:

- pipes are made available through the package **magrittr** (installed and loaded automatically with **tidyverse**)
- 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

Pipes!

1) Select species with blue flowers (COL) and stem size (SSIZE) higher than 10 and save in the new object "Flowers_blue10".

```
> flowers_filtered_dplyr<-flowers %>% filter(QRANGE !=1400 & SSIZE>10)
```

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 (base R) **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")) **%>%** **View**()

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

```
> flowers %>%
        filter(SSIZE>10 & QRANGE %in% 0:600 ) %>%
        select(SP, NIT, QMIN:last_col()) %>%
        dim()
[1]  8 22
```

dplyr: mutate

df **%>%** **mutate**(new_column_name = content)

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

> flowers<- flowers **%>%** **mutate**(FSR=FSIZE/SSIZE)

dplyr: mutate

df **%>%** **mutate**(new_column_name = content)

add column using conditional rules with **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<- flowers **%>%** **mutate**(QRANGE1000=**if_else**(QRANGE>1000, "A", "B"))

dplyr: mutate

df **%>%** **mutate**(new_column_name = content)

Exercise!

- Add a column called “QRANGE_log” to the flowers dataset with the logarithm of QRANGE column (tip: use the **log**() function) and save into the same object.
- do it with PIPES!

dplyr: mutate

df **%>%** **mutate**(new_column_name = content)

Exercise!

- Add a column called “QRANGE_log” to the flowers dataset with the logarithm of QRANGE column (tip: use the **log**() function) and save into the same object.
- do it with PIPES!

> flowers <- flowers **%>%** **mutate**(QRANGE_log=log(QRANGE))

dplyr: left_join()

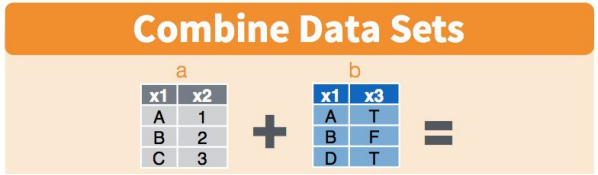
Combine two dataframes containing common key column/s

Key Variable	Variable A	Variable B	Variable C	Variable D
1	3.1	7.3	1	23
2	4.5	9.9	0	21
3	5.0	8.5	0	44
4	1.0	8.4	1	50

Key Variable	Variable E	Variable F	Variable G	Variable H
1	86	Red	4.9	19
2	95	Green	5.0	20
3	78	Red	5.0	14
4	91	Blue	4.1	13

Key Variable	Variable A	Variable B	Variable C	Variable D	Variable E	Variable F	Variable G	Variable H
1	3.1	7.3	1	23	86	Red	4.9	19
2	5.0	8.5	0	44	95	Green	5.0	20
3	5.0	8.5	0	44	78	Red	5.0	14
4	1.0	8.4	1	50	91	Blue	4.1	13

dplyr: left_join()



df **%>%** **left_join**(df2, index)

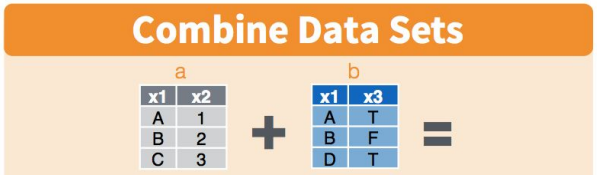
We need to join the flowers dataset with flowers_details.tsv table which contains additional attributes

>flowers_details<-**read.table**("https://raw.githubusercontent.com/mchialva/PhDToolbox2026/main/Datasets/flowers/flowers_details.tsv", header=T)

join (or merge) and save into a new object

> flowers_all<- flowers **%>%** **left_join**(flowers_details, by="SP")

dplyr: left_join()



Some tips on joining tables

- Most of the time you need to merge tables using row names as key (index) column.

  but….

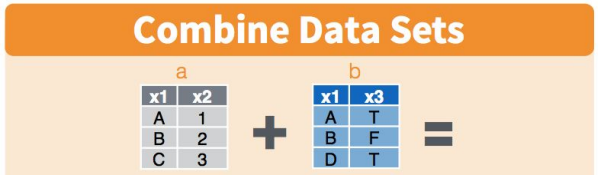
- tidyverse packages hardly manage rownames

  the solution comes again from tidyverse (package **tibble**):

```
df %>% rownames_to_column(var="index_column") %>%
       left_join(df2, index) %>%
       column_to_rownames(var="index_column")
```

Note: some functions necessarily require rownames!

dplyr: left_join()



Some tips on joining tables

- You can use an index column with different names in the two dataset!

first we need to change the index column name in flowers detail (for demonstration purposes) using rename() function

using rename() function

```
flowers_details<- flowers_details %>% rename("Species"="SP")
```

merge the datasets

```
> flowers_all<- flowers %>% left_join(flowers_details, by=c("SP"="Species"))
```

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()
```

dplyr: summarize() and group_by()

Most of the time 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 factors

df **%>%** **group_by**() **%>%** **summarize**(new_column_name = content, …)

dplyr: summarize() and group_by()

```
# Summarize min, mean and maximum SSIZE values for each family (FAM)


> flowers_all %>% group_by(FAM) %>%
                        summarize(      min_FSIZE=min(FSIZE),
                                        mean_SSIZE=mean(SSIZE),
                                        max_FSIZE=max(FSIZE)) %>% View()
```

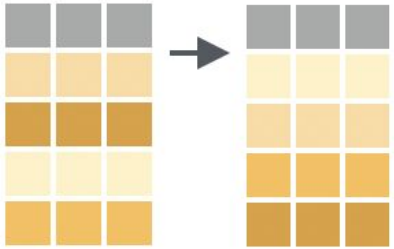
dplyr: summarize() and group_by()

Summarize min, mean and maximum SSIZE values for each family (FAM) distinguishing between flower colours (COL):

```
> flowers_all %>% group_by(FAM, COL) %>%
                  summarize(      min_FSIZE=min(FSIZE),
                                  mean_SSIZE=mean(SSIZE),
                                  max_FSIZE=max(FSIZE)) %>% View()
```

dplyr: arrange

Reorder dataframes row by column values (increasing/decreasing or alphabetical)



df %>% **arrange**(columns to order by)

Order the data frame *flowers_all* by increasing FSIZE value

> flowers_all %>% **arrange**(FSIZE) %>% **View**()

Order the dataframe flowers_all by decreasing FSIZE value

> flowers_all %>% **arrange**(-FSIZE) %>% **View**()

if applied to character vectors the function orders in alphabetic order

> flowers_all %>% **arrange**(COL) %>% **View**()

dplyr: exercise 4

Exercise!

- Take as input the *flowers_all* dataset
- Summarize (minimum, mean and maximum values) flower dimensions (FLEN) for each color (COL), habitat (HBT) and biology (FBIO) excluding Asteraceae and Crassulaceae families
- sort observations by decreasing mean value
- do it with PIPES!
- store the filtered dataset into the *flowers_subset* object
- 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: exercise 1

Solution:

```
> flowers_subset <-  flowers_all %>%
                     filter(!FAM %in% c("Asteraceae", "Crassulaceae")) %>%
                     group_by(COL,HBT,FBIO) %>%
                     summarize(      min=min(FLEN),
                                     mean=mean(FLEN),
                                     max=max(FLEN)      ) %>%
                     arrange(-mean)
# How many observations are there?
> nrow(flowers_subset)
[1] 62
```

dplyr: Reshape data

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

Required when using ggplot2 library for graphics with multiple variables!

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)

dplyr: Reshape data

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

df **%>%** **pivot_longer**(column/s_to_pivot, …)

melt columns CIT and H of the *flowers_all* dataset

dplyr: Reshape data

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

df %>% **pivot_longer**(column/s_to_pivot, …)

melt columns CIT and H of the *flowers_all* dataset

```
> 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")
```

dplyr exercise 5: The survey dataset

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

>download.file("https://raw.githubusercontent.com/mchialva/PhDToolbox2026/main/Datasets/surveys/plots.csv", "plots.csv")

>download.file("https://raw.githubusercontent.com/mchialva/PhDToolbox2026/main/Datasets/surveys/species.csv", "species.csv")

>download.file("https://raw.githubusercontent.com/mchialva/PhDToolbox2026/main/Datasets/surveys/surveys.csv", "surveys.csv")

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?*
- *Is there any table header information?*

dplyr exercise 5: The survey dataset

Capture data for every desert rodent caught on the 20 ha study area in the Chihuahuan Desert near Portal, Arizona, USA. A simplified version from Ernest et al., 2017

plots

Variable	type	Notes
plot_id	Int.	
plot_type	Int.	

species

Variable	type	Notes
species_id	Char.	
genus	Char.	
species	Char.	
taxa	Char.	

surveys

Variable	type	Notes
record_id	Int.	
month	Int.	
day	Int.	
year	Int.	
plot_id	Int.	1-24
species_id	Char.	See species table
sex	Char.	M/F
hindfoot_length	Int.	mm
weight	Int.	g

dplyr exercise 5 - The survey dataset

```
# 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 5 - The survey dataset

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

2) Filter observation of Rodents surveyed between 2001 and 2002 and put them in a new object called rodents

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

- group by year, species_id, sex and variable then summarize the mean and maximum parameter values

dplyr exercise 5 - The survey dataset

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) Filter observation of Rodents surveyed between 1998 and 2002 and put them in a new object called ***rodents***

```
>rodents<-surveys_complete %>% filter(year %in% 1998:2002 & taxa=="Rodent" )
```

dplyr exercise 5 - The survey dataset

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
- group by year, species_id, sex and variable then summarize the mean and maximum parameter values

dplyr: survey exercise 5 - The survey dataset

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))

A quick note on object classes in dplyr

Some of dplyr functions coerce your output into a tibble object (*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: %>% **as.data.frame**() function
- you may need to visualize more rows: %>% **print**(n=rows_to_show)

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**!

Manipulate text strings with tidyverse

Some of the most useful stringr() functions

str_sub()	# substring text
str_length()	# extract lenghts
str_split_fixed()	# split vectors by patterns
separate_wider_delim()	# split dataframe columns by pattern
str_locate_all()	# match patterns
str_replace()	# replace patterns
str_c() and str_flatten()	# paste and combine strings

create a simple vector

Let’s write a simple vector

```
> animals<-c("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"

strings exercise

Parse *flowers_all* dataset according to the following rules

- 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 rule

- create a new column called COL_EN with color names in COL translated in English

```
> unique(flowers_all$COL)

> flowers_all %>% mutate(
        COL_EN=str_replace_all(COL,
        c("blanc"="white", "rose"="red", "jaune"="yellow",
          "vert"="green", "bleu"="blue", marron="brown") ) )
        %>% View()
```

tidyr: separate_wider_delim()

split vectors according to a pattern:

separate_wider_delim(dataframe, column_to _split, separator,
vector_of_new_colnames, remove=T/F)

Example: split the SP column of the flower dataset into Genus and Species

```
> flowers<-flowers %>% separate_wider_delim(SP, delim=" ", names = c("Genus", "Species"), cols_remove = F)
```

cols_remove tells to the functions if the selected column should be keeped or removed

Exercise 6 The Tree-of-Life dataset

A real study case from Mammola et al., 2023 (https://doi.org/10.7554/eLife.88251.3).The dataset has been built to demonstrate which are organisms features that most attract people and scientists biasing our knowledge of biodiversity and species conservation. The dataset contain species specific features of >3000 species randomly selected across the Tree of Life.

Two main dataset:

1) Species traits
2) IUCN categories

Notes:

- some variables are binary-coded boolean values (0=absence/true; 1=presence/false or yes/no)

Check it on the Course github repository here!