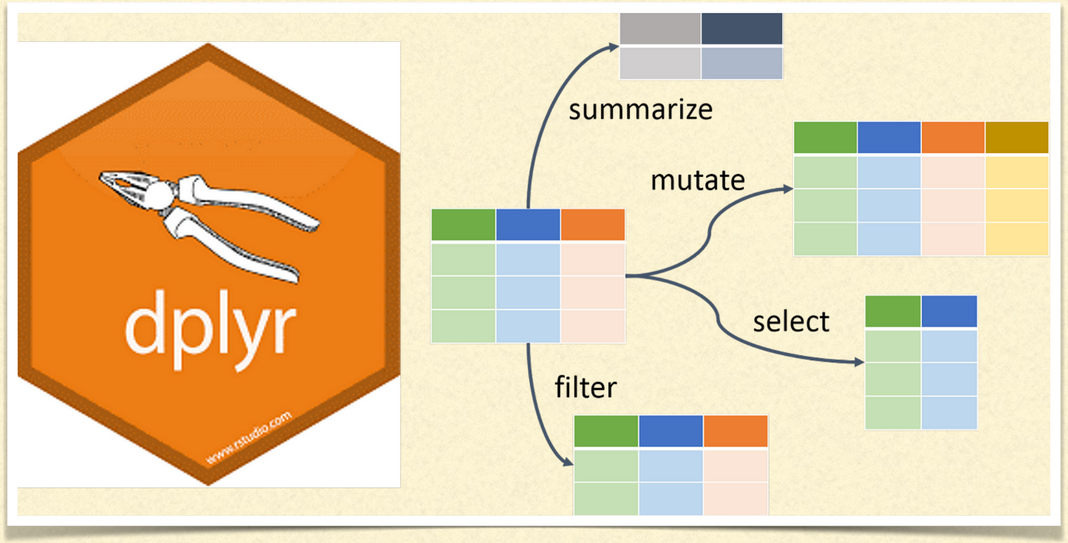
DATA "WRANGLING" IN R

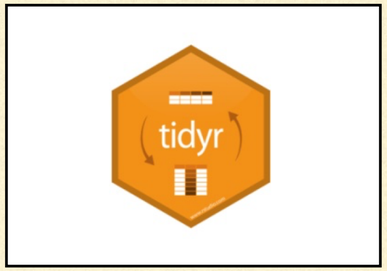
December 2023

DPLYR PACKAGE

https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

- Combine datasets
- Subset data
- Summarise data and checking
- Group data
- Make new variables
- …



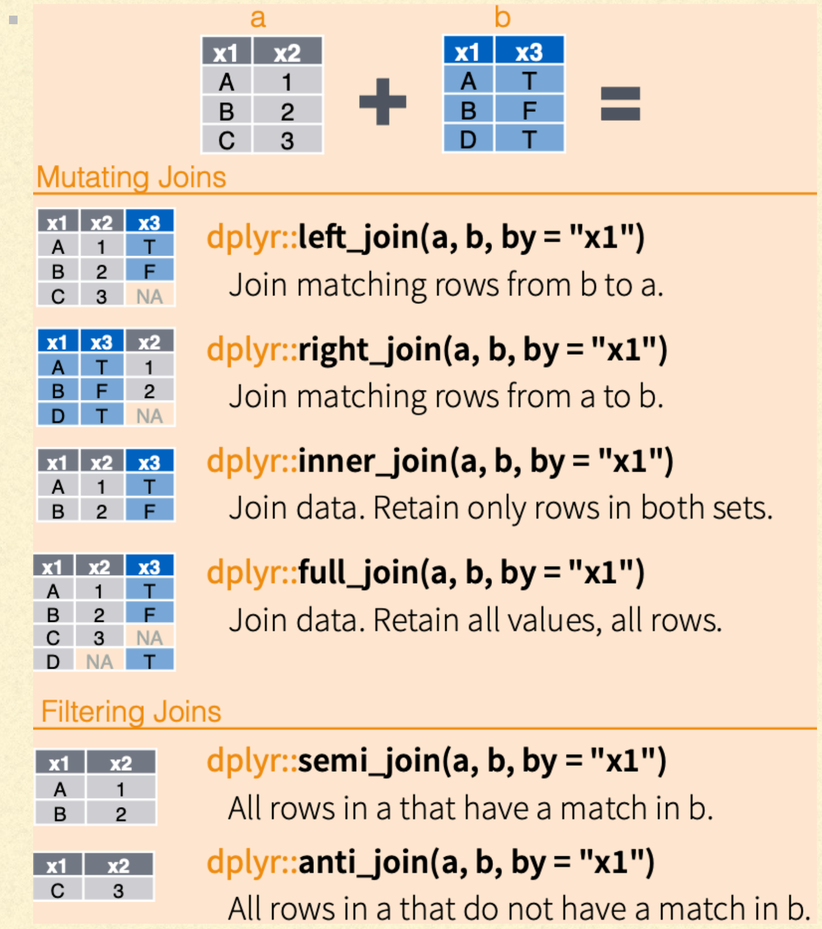


- The equivalent is **tidyr**. It's just a matter of preference:)
- Though combining both dplyr and tidyr can sometimes be extremely helpful

FIRST STOP, CHECK YOUR DF FIRST

- str(df) or summarise(df) # see what's in there
- head(df), tail(df)
- names(df) # just to see all column names
- dim(df) # how many data points (i.e. the number of rows and columns)
- View(df) # only view if your data is not so big, otherwise this can crash R…

1. COMBINING DATA



a

x1	x2
A	1
B	2
C	3

+

b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::**left_join(a, b, by = "x1")**

Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::**right_join(a, b, by = "x1")**

Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::**inner_join(a, b, by = "x1")**

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::**full_join(a, b, by = "x1")**

Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::**semi_join(a, b, by = "x1")**

All rows in a that have a match in b.

x1	x2
C	3

dplyr::**anti_join(a, b, by = "x1")**

All rows in a that do not have a match in b.

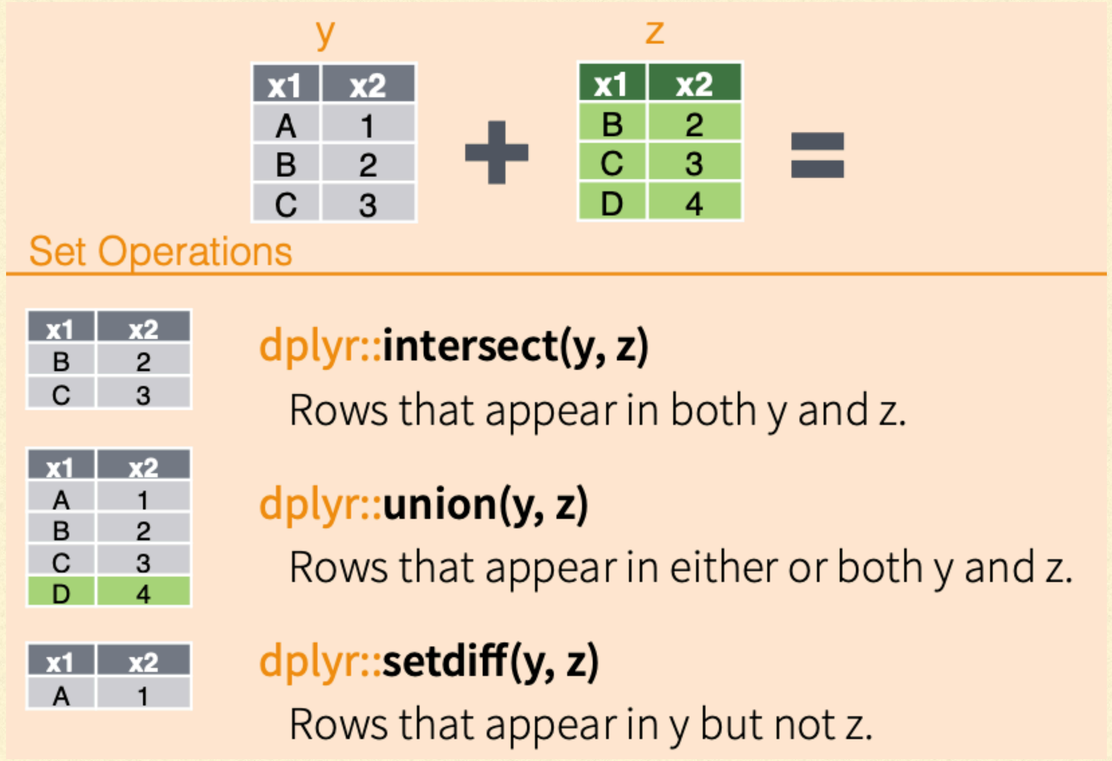
1. COMBINING DATA

ID	GERM	NON GERM	PERCENT
9036			
10005			
432			
…			

ID	LAT	LON	BIO1	BIO2
432				
532				
632				
…				
9036				
10005				
…				

ID	GERM	NON GERM	PERCENT	LAT	LON	BIO…
9036						
10005						
432						
…						

2. UNIFYINGDATA



2. UNIFYING DATA

- Secondary dormancy

ID	GERM	NON GERM	PERCENT
9036			
10005			
432			
530			
681			
1002			
…			

- Primary dormancy

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			
…			
…			

- Control

ID	GERM	NON GERM	PERCENT
9036			
10005			
432			
530			
681			
735			
1002			

2. UNIFYING DATA

- Secondary dormancy

ID	GERM	NON GERM	PERCENT
9036			
10005			
432			
530			
681			
1002			
…			

- Primary dormancy

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			
…			
…			

- Control

ID	GERM	NON GERM	PERCENT
9036			
10005			
432			
530			
681			
735			
1002			

2. UNIFYING DATA

- Secondary dormancy

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			

- Primary dormancy

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			

- Control

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			

NOTES ON COMBINING/SUBSETTING DATA

There are alternatives in base R, sometimes we can just do it in simple ways :)

- *merge(x, y, by = "column_name")*
- Using subsetting
  - *x_new = subset(x, x$ID %in% y$ID) —> only retain those that are present in y df*

Note: make sure the columns you use to merge or subset have **same class**. For example, when ID is a number like 432, 10005, choose to make it either *numeric* or *character* class

- *class(x$ID) # check class*
- *x$ID = as.character(x$ID) or as.numeric(x$ID), etc…*

NOTES ON COMBINING/SUBSETTING DATA

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

There are alternatives in base R, sometimes we can just do it in simple ways :)

—> same here, you can just use base R function:

df_new = df[,c(1,4)] # select column 1 and 4

Or *df_new = df[,c("ID","percent")]* # works the same

Remember the order in the syntax here, rows always come first, then columns
dataframe[(rows),(columns)]

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			

ID	PERCENT
9036	
10005	
530	
681	
1002	

3. ARRANGING DATA AND CHECKING

dplyr::arrange(mtcars, mpg)

Order rows by values of a column (low to high).

dplyr::arrange(mtcars, desc(mpg))

Order rows by values of a column (high to low).

dplyr::rename(tb, y = year)

Rename the columns of a data frame.

3. ARRANGING DATA AND CHECKING

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			

ID	GERM	NON GERM	PERCENT
530			
681			
1002			
9036			
10005			

Small tip: you can always make use of the "class". Let's say you want your ID to be "*character*" or "*factor*", for the sake of arranging it, just convert it to "*numeric*". After cleaning stuff is done, convert it back to whatever that fits your analyses.

3. ARRANGING DATA AND CHECKING

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			

ID	GERM	NON GERM	PERCENT
530			
681			
1002			
9036			
10005			

You did all this and want to check if the IDs of different data frames are all the same?

identical(df1$ID, df2$ID) # again, they should have the class, and also order. If yes, this should return *>TRUE*

4. MAKING A BIG DF

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			

ID	GERM	NON GERM	PERCENT
9036			
10005			
530			
681			
1002			

Secondary dormancy

Primary dormancy

Control

ID	GERM	NON GERM	PERCENT	GROUP
9036				SD
10005				SD
530				SD
681				SD
1002				SD

ID	GERM	NON GERM	PERCENT	GROUP
9036				PD
10005				PD
530				PD
681				PD
1002				PD

ID	GERM	NON GERM	PERCENT	GROUP
9036				CT
10005				CT
530				CT
681				CT
1002				CT

4. MAKING A BIG DF

Secondary dormancy

ID	GERM	NON GERM	PERCENT	GROUP
9036				SD
10005				SD
530				SD
681				SD
1002				SD

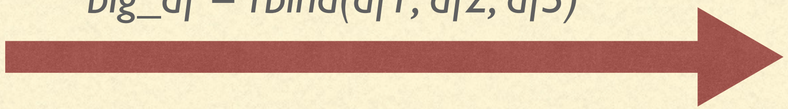
Primary dormancy

ID	GERM	NON GERM	PERCENT	GROUP
9036				PD
10005				PD
530				PD
681				PD
1002				PD

Control

ID	GERM	NON GERM	PERCENT	GROUP
9036				CT
10005				CT
530				CT
681				CT
1002				CT

big_df = rbind(df1, df2, df3)



Combined

ID	GERM	NON GERM	PERCENT	GROUP
9036				SD
10005				SD
...				SD
936				PD
10005				PD
...				PD
9036				CT
10005				CT
...				CT

Note: They must have same column names. If they have exactly same row order, then you can use cbind (“c” for column, “r" for row)

4. MAKING A BIG DF

Example for the time-series experiment

Timepoint 1

ID	GERM	NON GERM	PERCENT	TIME_ POINT
100				1
202				1
304				1
506				1
...				1

Timepoint 2

ID	GERM	NON GERM	PERCENT	TIME_ POINT
100				2
202				2
304				2
506				2
...				2

Timepoint 3

ID	GERM	NON GERM	PERCENT	TIME_ POINT
100				3
202				3
304				3
506				3
...				3

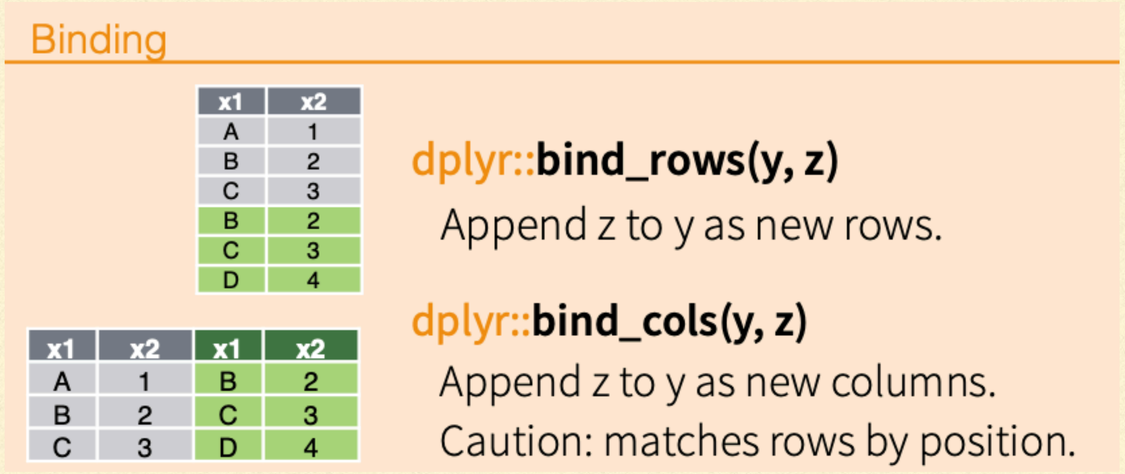
all_timepoint = rbind(timepoint1, timepoint2, timepoint3)

Combined

ID	GERM	NON GERM	PERCENT	TIME_POINT
100				1
202				1
...				1
100				2
202				2
...				2
100				3
202				3
...				3

4. MAKING A BIG DF

- Of course *dplyr* also has it. This is an example of things that can be done just as easy or possibly easier using base R



5. SUMMARISE AND MAKE NEW VARIABLES

Summarise Data

dplyr::**summarise(iris, avg = mean(Sepal.Length))**
Summarise data into single row of values.

dplyr::**summarise_each(iris, funs(mean))**
Apply summary function to each column.

dplyr::**count(iris, Species, wt = Sepal.Length)**
Count number of rows with each unique value of variable (with or without weights).

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

Make New Variables

dplyr::**mutate(iris, sepal = Sepal.Length + Sepal. Width)**
Compute and append one or more new columns.

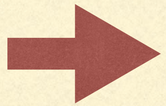
dplyr::**mutate_each(iris, funs(min_rank))**
Apply window function to each column.

dplyr::**transmute(iris, sepal = Sepal.Length + Sepal. Width)**
Compute one or more new columns. Drop original columns.

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

5. SUMMARISE AND MAKE NEW VARIABLES

ID	GERM	NON GERM
100		
202		
304		
506		
…		



ID	GERM	NON GERM	PERCENT
100			
202			
304			
506			
…			

- *new_df = dplyr::mutate(df, percent = germ/(germ+non_germ))* or directly modify the original by *df* **%>%** *mutate(percent = germ/(germ+non_germ))*
- *Or base R:*
  - *percent= df$germ/(df$germ+df$non_germ),* then *percent = as.data.frame(percent)* and *colnames(percent) = "percent"*
  - *new_df = cbind(df, percent)*

5. SUMMARISE AND MAKE NEW VARIABLES

Group Data

dplyr::**group_by(iris, Species)**

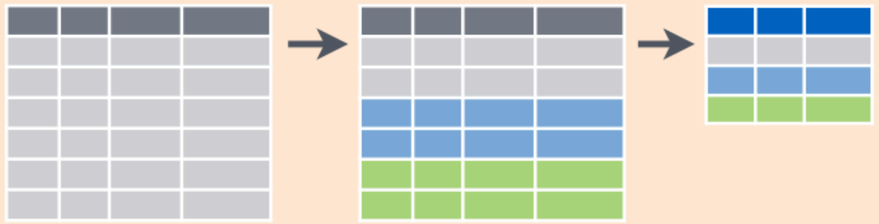
Group data into rows with the same value of Species.

dplyr::**ungroup(iris)**

Remove grouping information from data frame.

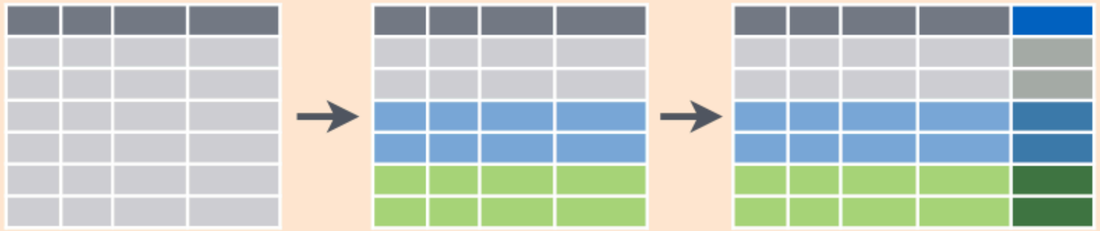
iris %>% group_by(Species) %>% summarise(…)

Compute separate summary row for each group.



iris %>% group_by(Species) %>% mutate(…)

Compute new variables by group.

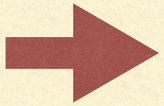


5. SUMMARISE AND MAKE NEW VARIABLES

ID	GERM	NON GERM	PERCENT	TIME_POINT
100				1
202				1
...				1
100				2
202				2
...				2
100				3
202				3
...				3

df %>% group_by(time_point)

%>% summarise(average_percent = mean(percent))



TIME_POINT	AVERAGE_PERCENT
1	0.5
2	
3	
4	
...	

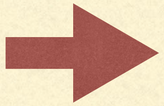
5. SUMMARISE AND MAKE NEW VARIABLES

ID	GERM	NON GERM	PERCENT	TIME_POINT	GROUP
100				1	SD
202				1	SD
…				1	SD
100				2	SD
202				2	SD
…				2	SD
100				3	SD
202				3	SD
…				3	SD
…				…	…

df %>% group_by(time_point, group)

%>% summarise(average_percent = mean(percent))

you can group multiple things



TIME_POINT	GROUP	AVERAGE_PERCENT
1	PD	
2	PD	
3	PD	
1	SD	
2	SD	

5. SUMMARISE AND MAKE NEW VARIABLES

ID	GERM	NON GERM	PERCENT	TIME_POINT	GROUP
100				1	SD
202				1	SD
…				1	SD
100				2	SD
202				2	SD
…				2	SD
100				3	SD
202				3	SD
…				3	SD
…				…	…

df %>% group_by(time_point)

%>% mutate(total_seed_sown = germ + non_germ)

TIME_POINT	TOTAL_SEED_SOWN
1	
2	
3	
4	
…	

5. SUMMARISE AND MAKE NEW VARIABLES

ID	GERM	NON GERM	PERCENT	TIME_POINT	GROUP
100				1	SD
202				1	SD
…				1	SD
100				2	SD
202				2	SD
…				2	SD
100				3	SD
202				3	SD
…				3	SD
…				…	…

df %>% group_by(time_point)

%>% tally()

Counting entries, number of samples in each group

TIME_POINT	COUNT(N)
1	19
2	18
3	18
4	17
…	

5. SUMMARISE AND MAKE NEW VARIABLES

ID	GERM	NON GERM	PERCENT	TIME_POINT	GROUP
100				1	SD
202				1	SD
…				1	SD
100				2	SD
202				2	SD
…				2	SD
100				3	SD
202				3	SD
…				3	SD
…				…	…

```
timepoint_2 = df  %>% filter(time_point == "2")

                  %>% filter(is.na(percent))
```

New data frame of only time point 2, if no data exists for "percent" column, the data point will be filtered out.

TO-DO

- Data entry to make life easy later:
  - All column names are consistent
  - No special character (f.e. % or & …)
  - Decimal number uses dot (.), not comma (,)
  - All IDs are consistent
  - Save as .csv, not .xlsx. (convert xlsx to csv)

TO-DO

- Next, bring it to R to prepare the data in this format. You should have:
  - ID preferably as *character* or *factor,* depending on what we do
  - germ, non_germ, percent: as *numeric*
  - time_point: as *factor* (5 levels)
  - group: as *factor* (3 levels (SD, PD, CT))
- As first look at the data:
  - average percent germination of different time points and different groups
  - ….

ID	GERM	NON GERM	PERCENT	TIME_POINT	GROUP
100				1	SD
202				1	SD
…				1	SD
100				2	SD
202				2	SD
…				2	SD
100				3	SD
202				3	SD
…				3	SD
…				…	…