Data transformation: dplyr package

# Introduction

Data transformation is an important step before starting to visualize and analyze your data. Transformation may involve:

* creating new variables and summaries
* renaming some variables
* reordering observations
* format variables
* select variables
* perform analyses by group
* filter/ subset observations
* etc …

The *dplyr* package (Wickham et al. 2023) is the workhorse for data transformation in R.

# Installing dplyr

*Dplyr* can be installed in two ways:

* as a standalone package:

*install.packages(“dplyr”)*

* which is then called in R by:
* *library(dplyr)*
* as part of the *tidyverse* package (Wickham et al. 2019)
* *install.packages(‘tidyverse’)*
  + Note: *tidyverse* is a collection of packages for data science:
    - dplyr
    - ggplot2
    - tidyr
    - …
  + In this case, loading *tidyverse* automatically loads *dplyr* plus the rest of the packages
* *library(tidyverse)*

We will load *dplyr* as part of *tidyverse* package and show some of its uses on the iris data

#install.packages('tidyverse') # uncomment to install tidyverse  
  
library(tidyverse)   
  
iris = iris

A subset of the iris dataset is shown in the table below:

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
| --- | --- | --- | --- | --- |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |

In the next subsections, we will show some key dplyr functions and their examples on the iris data.

# Dplyr functions

## Pipes (|> or %>%)

Pipes are used to combine operations. Pipes are very common in R codes that use functions from *dplyr* and *tidyverse*.

* say we want to calculate the means for *Sepal.Length* and *Sepal.Width* columns by species for the iris dataset:
* I. One way to do this could be
  + first, group the observations by *Species*
  + second, calculate the means for the columns

by\_group = group\_by(iris, Species)  
  
meanSepals <- summarise(by\_group,   
 mean\_Sepal.Length = mean(Sepal.Length, na.rm = TRUE),  
 mean\_Sepal.Width = mean(Sepal.Width, na.rm = TRUE))

| Species | mean\_Sepal.Length | mean\_Sepal.Width |
| --- | --- | --- |
| setosa | 5.006 | 3.428 |
| versicolor | 5.936 | 2.770 |
| virginica | 6.588 | 2.974 |

1. A second way is to combine the operations through a *pipe*, annotated by *%>%* or *|>* as shown in the code below

meanSepals2 <- iris %>% group\_by(Species) %>%   
 summarise(mean\_Sepal.Length = mean(Sepal.Length, na.rm = TRUE),   
 mean\_Sepal.Width = mean(Sepal.Width, na.rm = TRUE))

| Species | mean\_Sepal.Length | mean\_Sepal.Width |
| --- | --- | --- |
| setosa | 5.006 | 3.428 |
| versicolor | 5.936 | 2.770 |
| virginica | 6.588 | 2.974 |

* Note:
  + the *group\_by()* and *summarise()* functions used here are discussed in later slides

## Select()

Selects columns by name or position

* Select all sepal columns by name

# Notes:  
## pipes: %>%  
## select()  
  
irissubset = iris %>% select(Sepal.Length, Sepal.Width)

| Sepal.Length | Sepal.Width |
| --- | --- |
| 5.1 | 3.5 |
| 4.9 | 3.0 |
| 4.7 | 3.2 |
| 4.6 | 3.1 |
| 5.0 | 3.6 |
| 5.4 | 3.9 |

Alternative ways of specifying columns

* Using *contains* option in *select* function
  + *contains* picks out all column names having a *Sepal* pattern

# Notes:  
## select(contains())  
  
irissubset = iris %>% select(contains("Sepal"))

* By position

irissubset = iris %>% select(1:2)

| Sepal.Length | Sepal.Width |
| --- | --- |
| 5.1 | 3.5 |
| 4.9 | 3.0 |
| 4.7 | 3.2 |
| 4.6 | 3.1 |
| 5.0 | 3.6 |
| 5.4 | 3.9 |

Other helper functions within *select()*:

* *starts\_with(“Sep”)*
* *ends\_with(““)*
* *num\_range(“x”, 1:3)*
* Use *select()* together with *everything()* to reorder columns in a data frame as shown below

Initial order of columns

irissubset = head(iris)

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
| --- | --- | --- | --- | --- |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |

After moving Species and Petal columns to start

arranged\_iris = irissubset %>% select(Species, contains("Petal"), everything())

| Species | Petal.Length | Petal.Width | Sepal.Length | Sepal.Width |
| --- | --- | --- | --- | --- |
| setosa | 1.4 | 0.2 | 5.1 | 3.5 |
| setosa | 1.4 | 0.2 | 4.9 | 3.0 |
| setosa | 1.3 | 0.2 | 4.7 | 3.2 |
| setosa | 1.5 | 0.2 | 4.6 | 3.1 |
| setosa | 1.4 | 0.2 | 5.0 | 3.6 |
| setosa | 1.7 | 0.4 | 5.4 | 3.9 |

## Arrange()

Sorts rows according to one or more columns

* by default, *arrange()* sorts in an ascending order
* to change to descending order, place a **minus sign (i.e. -)** before the ordering column
  + alternative 2: use *desc(column)* function
  + missing values are always sorted to the end

### Arrange: Ascending

* Before *arrange()* is applied
  + For illustration, only the sepal columns are used

# Notes  
## pipes: %>%  
## select()   
  
irissubset = iris %>% select(c(Sepal.Length, Sepal.Width))

| Sepal.Length | Sepal.Width |
| --- | --- |
| 5.1 | 3.5 |
| 4.9 | 3.0 |
| 4.7 | 3.2 |
| 4.6 | 3.1 |
| 5.0 | 3.6 |
| 5.4 | 3.9 |

* After *arrange()* is applied on *Sepal.Length* column

# Notes:  
## pipes: %>%  
## arrange()  
  
arranged\_iris = irissubset %>% arrange(Sepal.Length)

| Sepal.Length | Sepal.Width |
| --- | --- |
| 4.3 | 3.0 |
| 4.4 | 2.9 |
| 4.4 | 3.0 |
| 4.4 | 3.2 |
| 4.5 | 2.3 |
| 4.6 | 3.1 |

### Arrange: Descending

* Before *arrange()*
  + For illustration, only the sepal columns are used

# Notes  
## pipes: %>%  
## select()   
  
irissubset = iris %>% select(c(Sepal.Length, Sepal.Width))

| Sepal.Length | Sepal.Width |
| --- | --- |
| 5.1 | 3.5 |
| 4.9 | 3.0 |
| 4.7 | 3.2 |
| 4.6 | 3.1 |
| 5.0 | 3.6 |
| 5.4 | 3.9 |

* After applying *arrange()* on *Sepal.Length* column

# Notes:  
## pipes: %>%  
## arrange()  
  
arranged\_iris2 = irissubset %>% arrange(-Sepal.Length)

| Sepal.Length | Sepal.Width |
| --- | --- |
| 7.9 | 3.8 |
| 7.7 | 3.8 |
| 7.7 | 2.6 |
| 7.7 | 2.8 |
| 7.7 | 3.0 |
| 7.6 | 3.0 |

### Arrange on two columns

* Before *arrange()*
  + For illustration, only the sepal columns are used

# Notes  
## pipes: %>%  
## select()   
  
irissubset = iris %>% select(c(Sepal.Length, Sepal.Width))

| Sepal.Length | Sepal.Width |
| --- | --- |
| 5.1 | 3.5 |
| 4.9 | 3.0 |
| 4.7 | 3.2 |
| 4.6 | 3.1 |
| 5.0 | 3.6 |
| 5.4 | 3.9 |

* After *arrange()*
  + decreasing *Sepal.Length*, increasing *Sepal.Width*

# Notes:  
## pipes: %>%  
## arrange()  
  
arranged\_iris3 = irissubset %>% arrange(-Sepal.Length, Sepal.Width)

| Sepal.Length | Sepal.Width |
| --- | --- |
| 7.9 | 3.8 |
| 7.7 | 2.6 |
| 7.7 | 2.8 |
| 7.7 | 3.0 |
| 7.7 | 3.8 |
| 7.6 | 3.0 |

## Mutate()

For creating new variables to a dataset and/or transforming existing variables

* the columns are always added at the end of the dataset
* as an example, if we want to calculate differences between Sepal & Petal lengths and widths

irissubset = head(iris %>% select(Species, everything()))  
irissubset = irissubset %>%   
 mutate(SP\_length\_diff = Sepal.Length-Petal.Length, SP\_width\_diff = Sepal.Width - Petal.Width)

| Species | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | SP\_length\_diff | SP\_width\_diff |
| --- | --- | --- | --- | --- | --- | --- |
| setosa | 5.1 | 3.5 | 1.4 | 0.2 | 3.7 | 3.3 |
| setosa | 4.9 | 3.0 | 1.4 | 0.2 | 3.5 | 2.8 |
| setosa | 4.7 | 3.2 | 1.3 | 0.2 | 3.4 | 3.0 |
| setosa | 4.6 | 3.1 | 1.5 | 0.2 | 3.1 | 2.9 |
| setosa | 5.0 | 3.6 | 1.4 | 0.2 | 3.6 | 3.4 |
| setosa | 5.4 | 3.9 | 1.7 | 0.4 | 3.7 | 3.5 |

Similar to *select()*, *mutate()* can use many functions to create new variables:

* arithmetic operators: +, -, \*, /
* logs
* offsets e.g. *lead()*, *lag()*
* cumulative and rolling aggregates: *cumprod()*, *cumsum()*, *cummean()* etc

### mutate\_if()

Checks if a condition is met before applying a transformation to a column

* say we want to format all character variables to be factors in the iris dataset

irissubset = head(iris %>% select(Species, everything()))  
  
# calculate the ratios and then format the rel  
irissubset = irissubset %>%   
 mutate(SP\_len\_ratio = Sepal.Length/Petal.Length, SP\_wid\_ratio = Sepal.Width/Petal.Width) %>%  
 mutate\_if(is.character, as.factor)

### mutate\_at()

Allows to specify specific columns on which an operation should be performed

* say we want to format all ratios between corresponding Sepal and Petal variables to appear with 2 decimal places

irissubset = head(iris %>% select(Species, everything()))  
  
# function to format variables to 2 decimal places  
format\_fn = function(x) formatC(x, format = 'f', digits = 2)  
  
# columns to format  
cols\_to\_format = c("SP\_len\_ratio", "SP\_wid\_ratio")  
  
# calculate the ratios and then format the retios to 2 decimal places using format\_fn()  
  
irissubset = irissubset %>%   
 mutate(SP\_len\_ratio = Sepal.Length/Petal.Length, SP\_wid\_ratio = Sepal.Width/Petal.Width) %>%  
 mutate\_at(.vars = all\_of(cols\_to\_format), .funs = format\_fn)

| Species | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | SP\_len\_ratio | SP\_wid\_ratio |
| --- | --- | --- | --- | --- | --- | --- |
| setosa | 5.1 | 3.5 | 1.4 | 0.2 | 3.64 | 17.50 |
| setosa | 4.9 | 3.0 | 1.4 | 0.2 | 3.50 | 15.00 |
| setosa | 4.7 | 3.2 | 1.3 | 0.2 | 3.62 | 16.00 |

## Filter()

Used to extract rows based on a specified condition

* for instance, to extract all rows of the **Setosa** species

irissetosa = iris %>% filter(Species == 'setosa') # keep Setosa Species

flextable(table(irissetosa %>% pull(Species)) %>% as.data.frame())

| Var1 | Freq |
| --- | --- |
| setosa | 50 |
| versicolor | 0 |
| virginica | 0 |

* filter Setosa Species with Sepal length above the median Sepal Length

irissetosa2 = iris %>% filter(Species == 'setosa' & Sepal.Width > median(Sepal.Width))

## Summarise()

Provides summary statistics from a dataset. For instance, if we want to find the median values for Petal columns by species:

medianPetals <- iris %>% group\_by(Species) %>% select(contains('Petal')) %>%   
 summarise(median\_Pelal.Length = mean(Petal.Length, na.rm = TRUE),  
 median\_Petal.Width = mean(Petal.Width, na.rm = TRUE))

| Species | median\_Pelal.Length | median\_Petal.Width |
| --- | --- | --- |
| setosa | 1.462 | 0.246 |
| versicolor | 4.260 | 1.326 |
| virginica | 5.552 | 2.026 |

## Group\_by()

It helps other functions perform their operations by groups in a dataset. It does not change the data in itself, but changes how the data is interpreted by other functions.

## Other important functions in dplyr

The functions mentioned are not exhaustive but are among the most commonly used. The package has many other functions that are useful for specific operations.

# Resources/ References

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D’Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. “Welcome to the Tidyverse” 4: 1686. <https://doi.org/10.21105/joss.01686>.

Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2023. “Dplyr: A Grammar of Data Manipulation.” <https://CRAN.R-project.org/package=dplyr>.