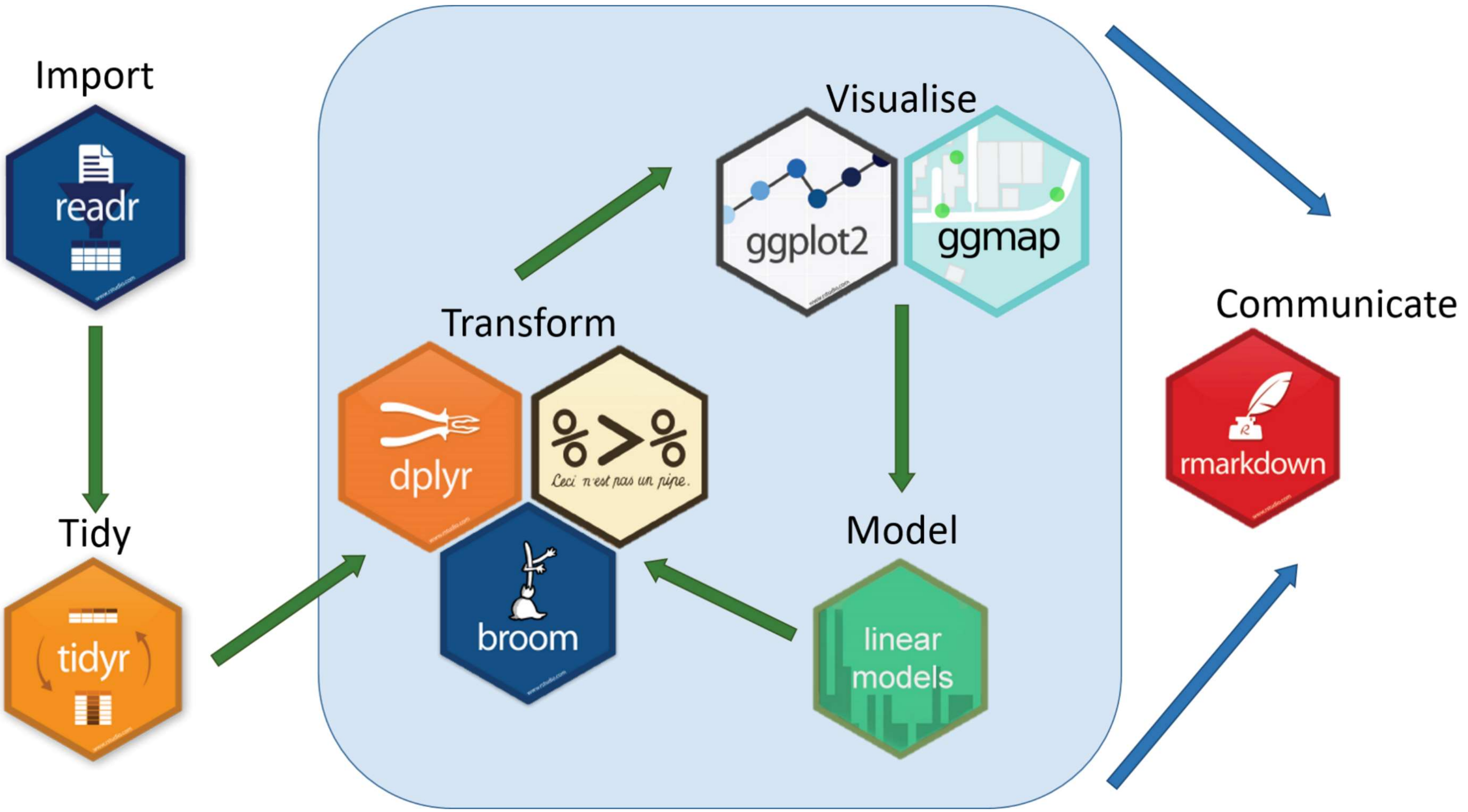
Manipulating data with dplyr

Paloma Rojas S.

First, what is tidyverse??



The **%>%** (Control + shift + m)

- You'll structure the sequence of your data operations from left to right, instead of inside and out
- You'll avoid nested function calls
- You'll minimize the need for local variables and function definitions
- You'll make it easy to add steps anywhere in the sequence of operations

The **dplyr** package

- The format is the following:
  - The fist argument is a dataframe
  - The subsequent argument describes what to do with that dataframe
  - Each call returns a new dataframe
- Works with tidy data (each column is a variable, each row is an observation)
- The functions are very efficient in when it comes to arrange your dataset
- The name of the functions are really intuitive
- Combine functions with the `%>%` and make your code easy to understand

Getting started

Install and open the **dplyr** package

```
install.packages(dplyr)
```

```
library("dplyr")
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Open the example dataset

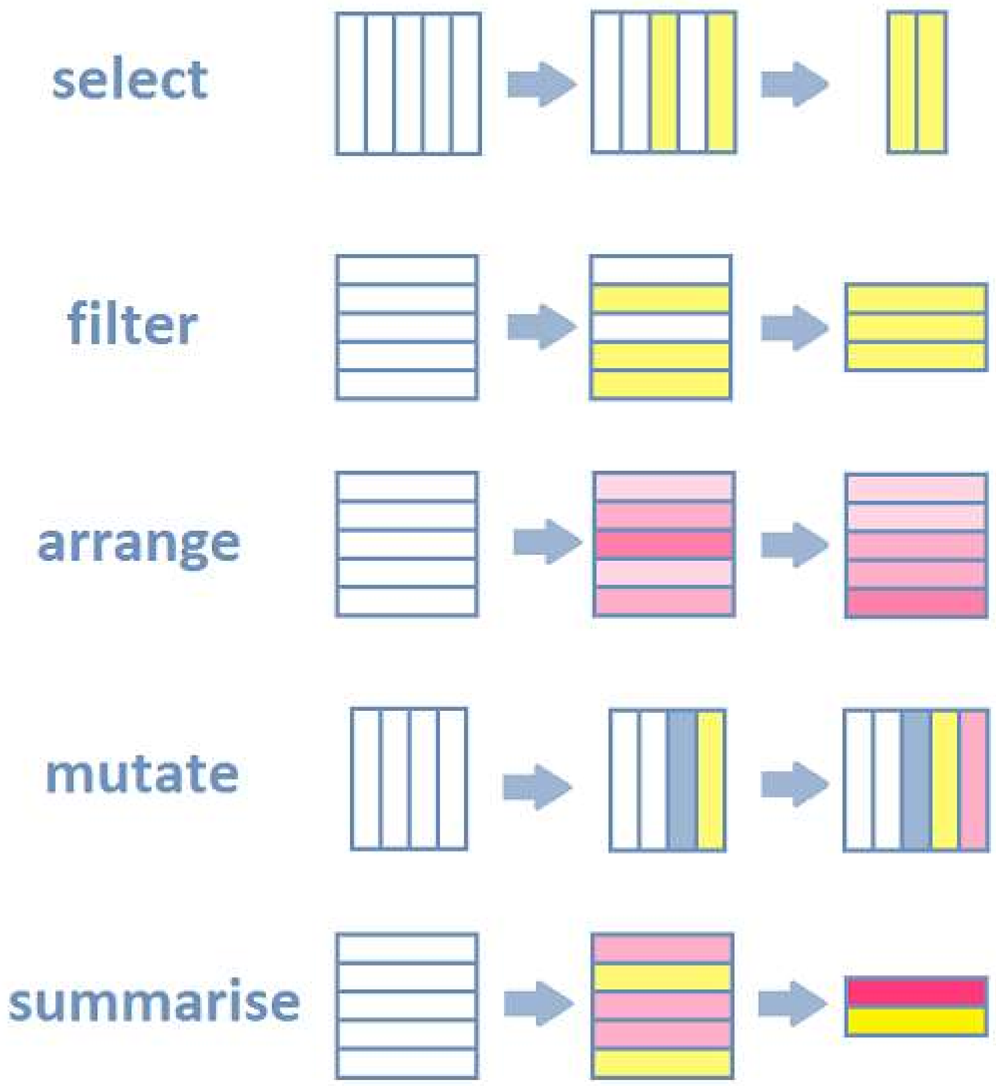
```
db <- iris

head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          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          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

The main functions

- `select()` picks columns based on their names.
- `arrange()` changes the ordering of rows.
- `filter()` selects rows based on given criteria.
- `mutate()` creates new variables(columns) based on existing (or replaces).
- `summarise()` reduces multiple values down to a single summary.
- `group_by()` performes everything above on a group-by-group basis.



https://itsalocke.com/blog/data-manipulation-in-r/

Select()

```
db %>%
  select(Petal.Length, starts_with("Sepal"), Species) %>%
  head()
```

```
##   Petal.Length Sepal.Length Sepal.Width Species
## 1          1.4          5.1         3.5  setosa
## 2          1.4          4.9         3.0  setosa
## 3          1.3          4.7         3.2  setosa
## 4          1.5          4.6         3.1  setosa
## 5          1.4          5.0         3.6  setosa
## 6          1.7          5.4         3.9  setosa
```

Select()

```
db %>%
  select(Species, ends_with("Width"),everything()) %>%
    head()
```

```
##   Species Sepal.Width Petal.Width Sepal.Length Petal.Length
## 1  setosa         3.5         0.2          5.1          1.4
## 2  setosa         3.0         0.2          4.9          1.4
## 3  setosa         3.2         0.2          4.7          1.3
## 4  setosa         3.1         0.2          4.6          1.5
## 5  setosa         3.6         0.2          5.0          1.4
## 6  setosa         3.9         0.4          5.4          1.7
```

Select()

```
db %>%
  select(-c(ends_with("Width"))) %>%
    head()
```

```
##   Sepal.Length Petal.Length Species
## 1          5.1          1.4  setosa
## 2          4.9          1.4  setosa
## 3          4.7          1.3  setosa
## 4          4.6          1.5  setosa
## 5          5.0          1.4  setosa
## 6          5.4          1.7  setosa
```

Helper functions

- `starts_with("a")`
- `ends_with("z")`
- `contains("letters")`
- `matches(<regex>)`
- `-variable`
- `-c(var1, var2)`
- `everything()`

arrange()

- Note that default is ascending order, and NA's are at the end

```
db %>%
  arrange(Sepal.Width) %>%
  head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1          5.0         2.0          3.5         1.0 versicolor
## 2          6.0         2.2          4.0         1.0 versicolor
## 3          6.2         2.2          4.5         1.5 versicolor
## 4          6.0         2.2          5.0         1.5  virginica
## 5          4.5         2.3          1.3         0.3     setosa
## 6          5.5         2.3          4.0         1.3 versicolor
```

filter()

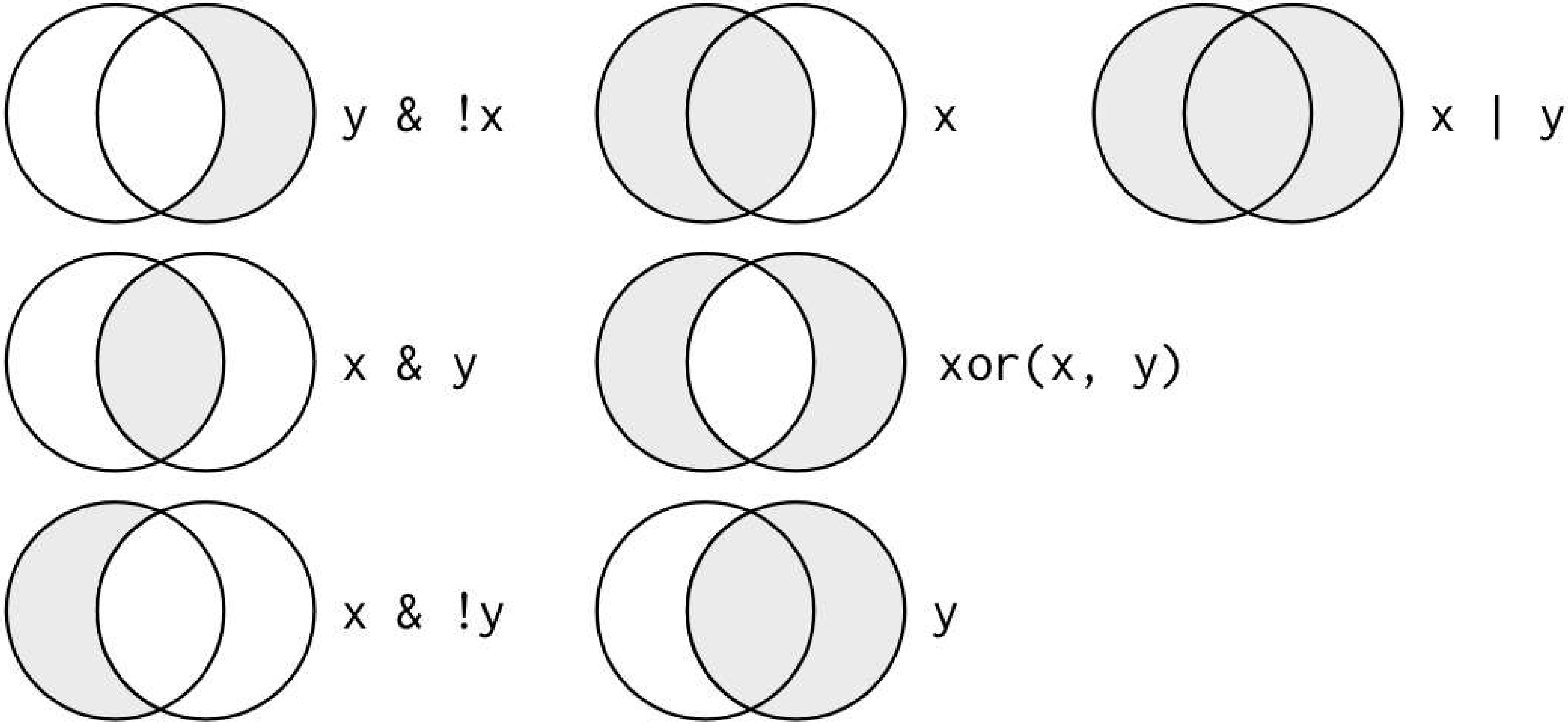
```
db %>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   150
```

```
db %>%
filter(Species == "virginica") %>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    50
```

filter() + *booleans*



You can also filter NA's is.na(var) | !is.na(var)

Check: R4DS book

filter() + *booleans*

```
db %>%
filter(Species != "virginica" & Sepal.Length > 4.5) %>%
count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    95
```

```
db %>%
filter(Species != "virginica" & Sepal.Length > 4.5) %>%
  arrange(Sepal.Length) %>%
  head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          4.6         3.1          1.5         0.2  setosa
## 2          4.6         3.4          1.4         0.3  setosa
## 3          4.6         3.6          1.0         0.2  setosa
## 4          4.6         3.2          1.4         0.2  setosa
## 5          4.7         3.2          1.3         0.2  setosa
## 6          4.7         3.2          1.6         0.2  setosa
```

mutate()

mutate:
add column(s),
keep existing.
A
1
5
10
0
B
2
-3
4
9
6
C
3
2
14
9
8
THE PLAN:
A
B
C
Horst '18

Follow @allison_horst for amazing illustration <3

mutate()

```
db %>%
mutate(petal_area = Petal.Length*Petal.Width) %>%
head()
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species petal_area
## 1          5.1         3.5          1.4         0.2  setosa       0.28
## 2          4.9         3.0          1.4         0.2  setosa       0.28
## 3          4.7         3.2          1.3         0.2  setosa       0.26
## 4          4.6         3.1          1.5         0.2  setosa       0.30
## 5          5.0         3.6          1.4         0.2  setosa       0.28
## 6          5.4         3.9          1.7         0.4  setosa       0.68
```

mutate() useful functions

- Arithmetic operators (+, - , *, / , ^)
- Log functions (like `log10()`)
- Offsets like `lead()` and `lag()`
- Logical comparisons (<, <=, >, >=, !=)
- `ifelse` statement (if this, then this, else this)
- Or when more than 1 logical split use `case_when()`
- Cumulative and rolling aggregates
- Accepts functions from different packages like `na.locf`
- Check `mutate_at`, `mutate_if`

mutate()

```
db %>%
  select(-starts_with("Sepal")) %>%
  mutate(id = row_number(),
         petal_l_bin = ifelse(Petal.Length >1.5, "Above 1.5", "Below 1.5")) %>%
  head()
```

```
##   Petal.Length Petal.Width Species id petal_l_bin
## 1          1.4         0.2  setosa  1   Below 1.5
## 2          1.4         0.2  setosa  2   Below 1.5
## 3          1.3         0.2  setosa  3   Below 1.5
## 4          1.5         0.2  setosa  4   Below 1.5
## 5          1.4         0.2  setosa  5   Below 1.5
## 6          1.7         0.4  setosa  6   Above 1.5
```

summarise()

```
db %>%
  summarise(avg_sepal_l = mean(Sepal.Length),
            median_petal_l = median(Petal.Length))
```

```
##   avg_sepal_l median_petal_l
## 1    5.843333           4.35
```

```
#More options
db %>%
  summarise_if(is.numeric, mean)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.843333    3.057333        3.758    1.199333
```

group_by()

```
db %>%
  select(Species, Sepal.Length) %>%
  filter(Sepal.Length > 6.7) %>%
  arrange(Species, Sepal.Length) %>%
  group_by(Species) %>%
  mutate(id = row_number(),
         avg_petal_l = mean(Sepal.Length)) %>%
  head()
```

```
## # A tibble: 6 x 4
## # Groups:   Species [2]
##   Species    Sepal.Length    id avg_petal_l
##   <fct>             <dbl> <int>       <dbl>
## 1 versicolor          6.8     1        6.9 
## 2 versicolor          6.9     2        6.9 
## 3 versicolor          7       3        6.9 
## 4 virginica           6.8     1        7.29
## 5 virginica           6.8     2        7.29
## 6 virginica           6.9     3        7.29
```

My most used

- `select(contains)`
- `mutate + ifelse/case_when`
- `count(var1,var2,var3)` To see number of combinations of variables
- `df %>% group_by %>% mutate` Specially if you work with clustered data
- `df %>% mutate_if(is.character,as.factor)`
- `df %>% filter() %>% mutate %>% ggplot()` Combine your functions and graph them
- `joins()` To merge datasets

Enjoy **dplyr**

Useful resources

Learn tidyverse:

https://r4ds.had.co.nz/

Check our Rladies-neighbour @SuzanBaert blog on dplyr:

https://suzan.rbind.io/

Get the dplyr cheatsheet

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

Slides created via the R package **Xaringan** , using **Rladies** template

https://github.com/yihui/xaringan

Thank you!
Pusheen.com