# Entering the tidyverse

*Martin Boutroux (inspired by Massimo Bourquin)*

*October 2025*

*Multivariate statistics in R*

**1975**
The S language is developed at the Bell Labs

**1988**
"The New S Language" book by John Chambers defines the features of the mature S

**1993**
Ross Ihaka and Robert Gentleman first mention R, a new implementation of S

**1995**
R is released under the GNU license

**1996**
The first official R paper "R: A Language for Data Analysis and Graphics" is published on the Journal of Computational and Graphical Statistics

**1997**
Founding of CRAN, the R mailing lists and the R core team

**2000**
**R 1.0.0** first stable release

**2001**
Creation of Bioconductor

**2004**
**R 2.0.0** released, introducing lazy loading and big data support

**2007**
Hadley Wickham releases ggplot2

**2011**
Rstudio beta 0.92 is first released

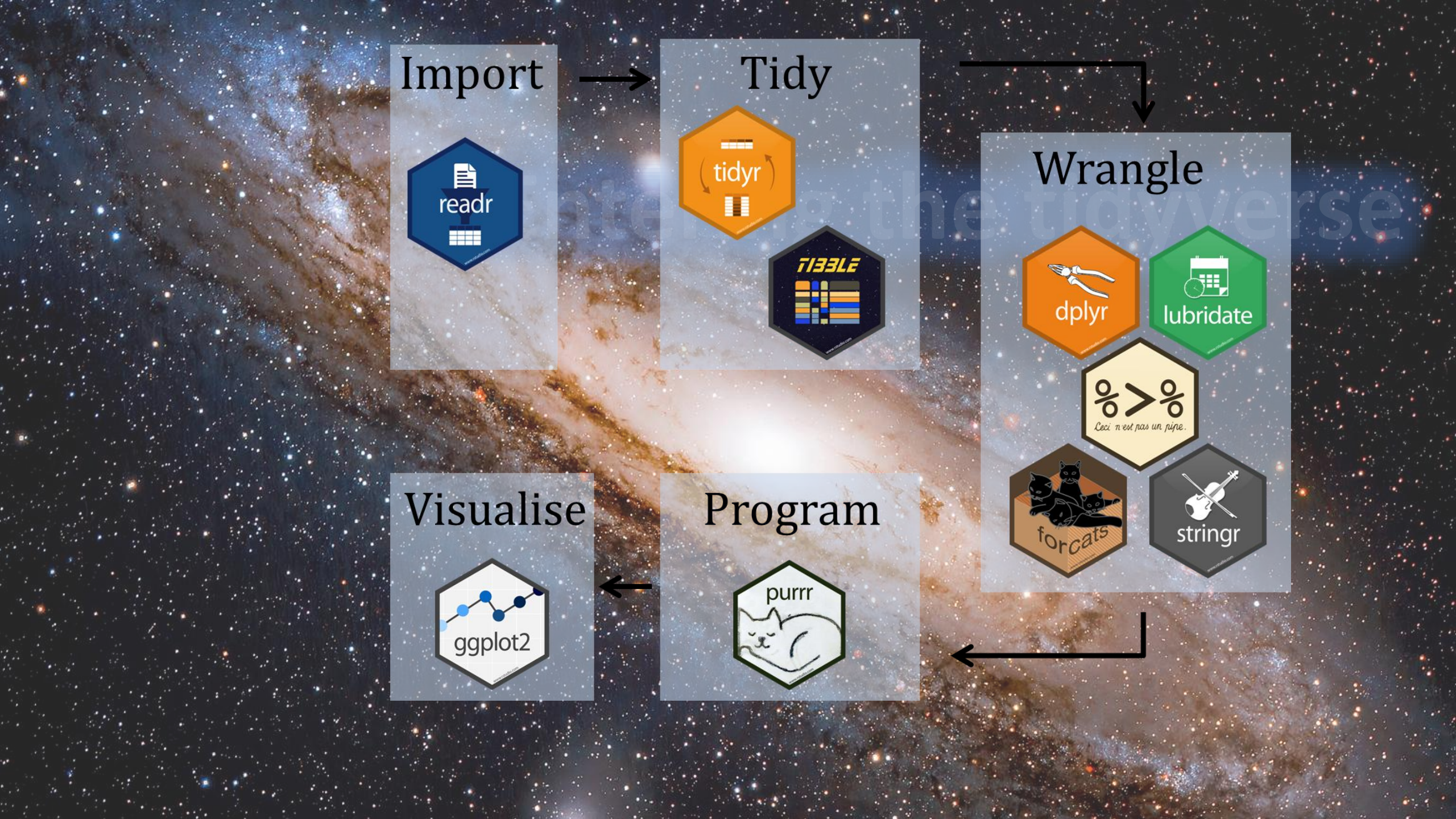
**2012**
Release of the *shiny* package

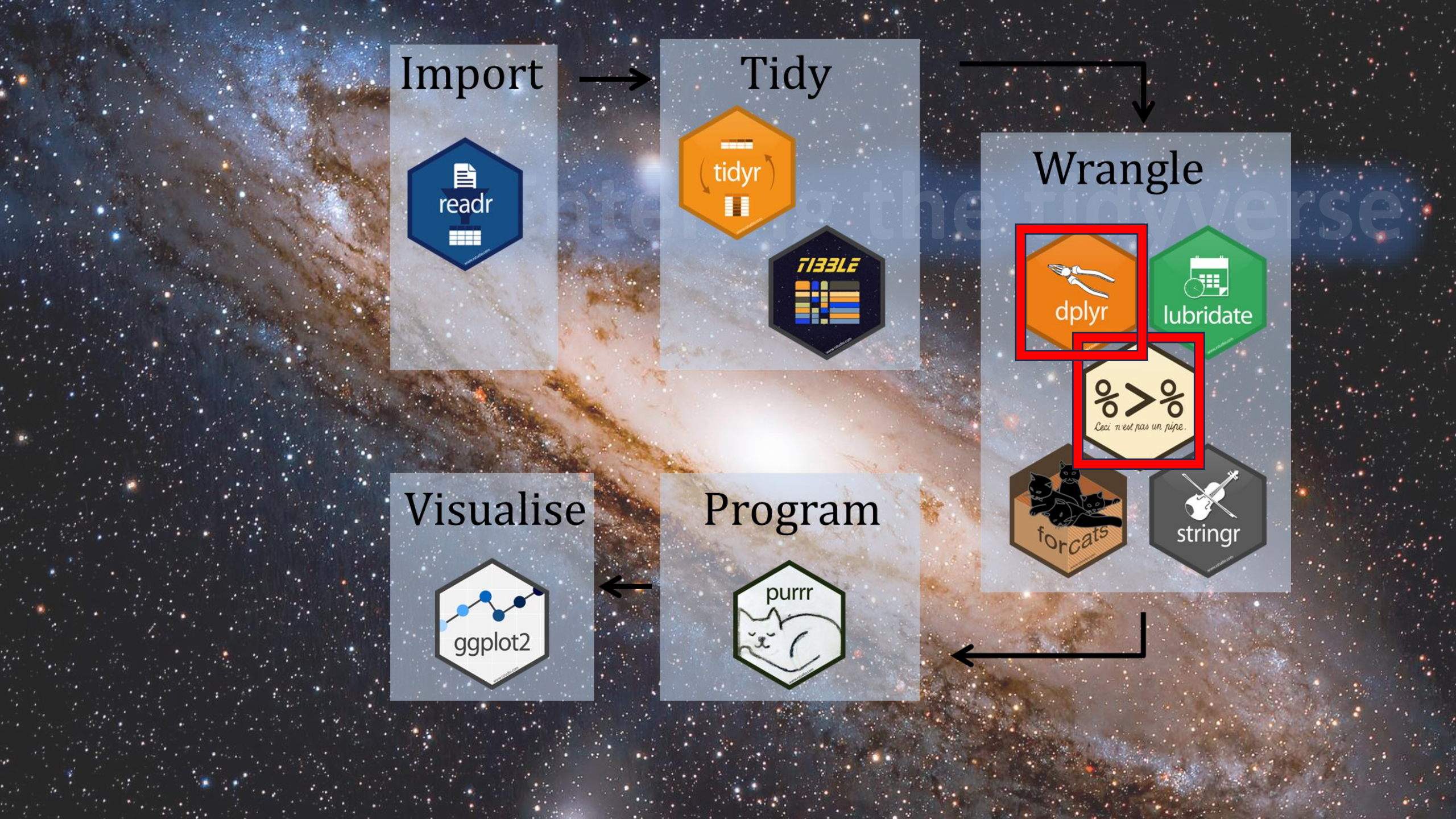
**2013**
**R 3.0.0** released with full support for 64bit architectures

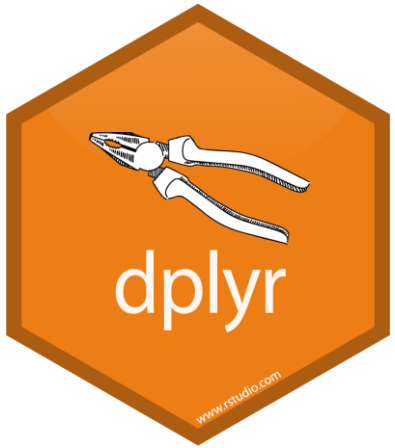
**2014**
Rmarkdown stable release

**2016**
Tidyverse package set version 1.0.0

**2020**
**R.4.0.0** is released


Import
readr
Tidy
tidyr
tibble
Wrangle
dplyr
lubridate
%>%
Ceci n'est pas un pipe.
forcats
stringr
Program
purrr
Visualise
ggplot2


Import
readr
Tidy
tidyr
tibble
Wrangle
dplyr
lubridate
%>%
Leci n'est pas un pipe.
forcats
stringr
Program
purrr
Visualise
ggplot2




# Package #1: dplyr

*Data wrangling*

# *A Grammar of Data Manipulation*

- A fast, consistent tool for working with data frames
- Widely used in ecology and data science in R
- Easy syntax, simplifies complicated tasks

# *Filtering rows in a data frame*

- **filter(data, mask1, mask2, etc.)**

```
(dec25 <- filter(flights, month == 12, day == 25))
#> # A tibble: 719 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_a
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1  2013    12    25      456            500        -4      649
#> 2  2013    12    25      524            515         9      805
#> 3  2013    12    25      542            540         2      832
#> 4  2013    12    25      546            550        -4     1022
#> 5  2013    12    25      556            600        -4      730
#> 6  2013    12    25      557            600        -3      743
#> # … with 713 more rows, and 11 more variables: arr_delay <dbl>, carrie
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dl
#> #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Copy

# *Reordering rows in a data frame*

- **arrange(data, col1, col2, etc.)**

```
arrange(flights, year, month, day)
#> # A tibble: 336,776 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_a
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1  2013     1     1      517            515         2      830
#> 2  2013     1     1      533            529         4      850
#> 3  2013     1     1      542            540         2      923
#> 4  2013     1     1      544            545        -1     1004
#> 5  2013     1     1      554            600        -6      812
#> 6  2013     1     1      554            558        -4      740
#> # … with 336,770 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <ch
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hou
```

Copy

# *Selecting columns*

- **select(data, col1, col2, etc.)**

```
# Select columns by name
select(flights, year, month, day)
#> # A tibble: 336,776 x 3
#>    year month   day
#>   <int> <int> <int>
#> 1  2013     1     1
#> 2  2013     1     1
#> 3  2013     1     1
#> 4  2013     1     1
#> 5  2013     1     1
#> 6  2013     1     1
#> # … with 336,770 more rows
# Select all columns between year and day (inclusive)
```

Copy

# *Creating new columns based on existing ones*

- **mutate(data, new_col = calculation)**

```
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
#> # A tibble: 336,776 x 9
#>    year month   day dep_delay arr_delay distance air_time  gain speed
#>   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
#> 1  2013     1     1         2        11     1400      227    -9  370.
#> 2  2013     1     1         4        20     1416      227   -16  374.
#> 3  2013     1     1         2        33     1089      160   -31  408.
#> 4  2013     1     1        -1       -18     1576      183    17  517.
#> 5  2013     1     1        -6       -25      762      116    19  394.
#> 6  2013     1     1        -4        12      719      150   -16  288.
#> # … with 336,770 more rows
```

# *Grouping data and summarising values*

- **group_by(data, col1, col2,  etc.)**
- **summarise(data, new_summary = calculation)**

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
#> `summarise()` regrouping output by 'year', 'month' (override with `.gr
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>    year month   day delay
#>   <int> <int> <int> <dbl>
#> 1  2013     1     1 11.5
#> 2  2013     1     2 13.9
#> 3  2013     1     3 11.0
#> 4  2013     1     4  8.95
#> 5  2013     1     5  5.73
#> 6  2013     1     6  7.15
#> # … with 359 more rows
```

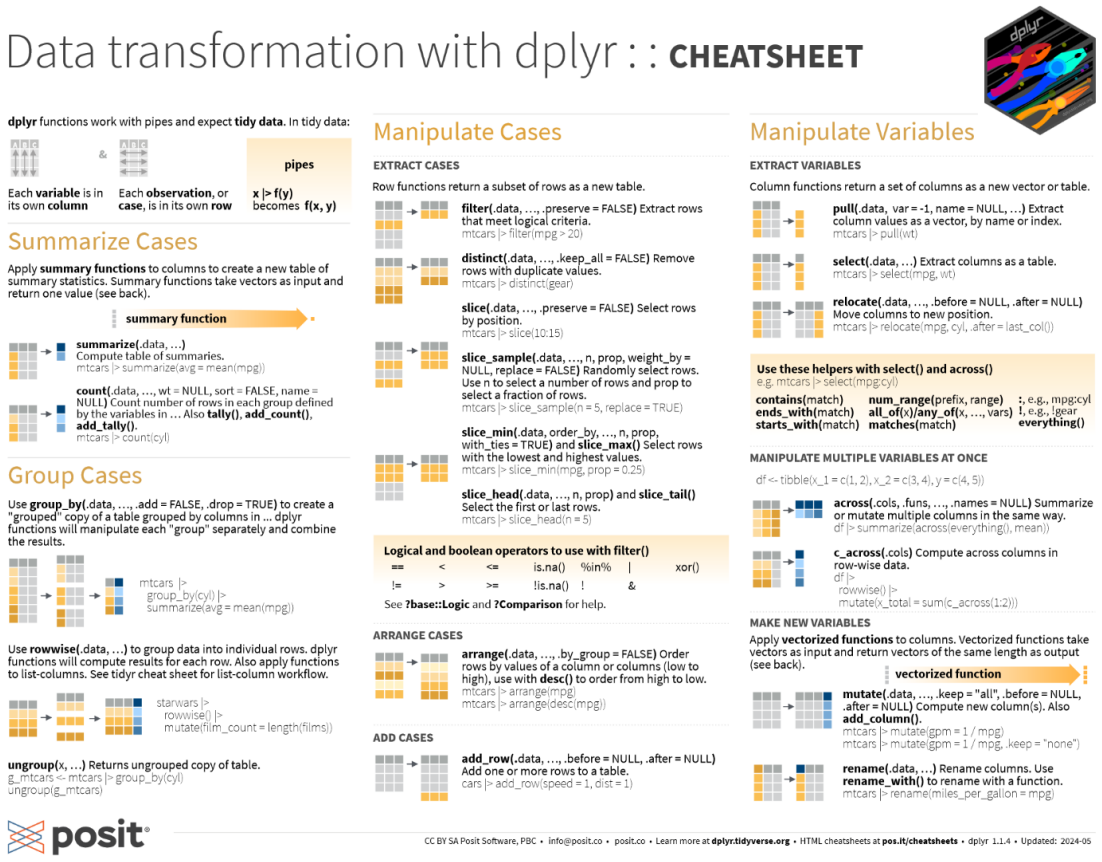
Copy

# *Summary*

- **filter(data, mask1, mask2, etc.)** > filter rows
- **arrange(data, col1, col2, etc.)** > order rows
- **select(data, col1, col2, etc.)** > select columns
- **mutate(data, new_col = calculation)** > create new columns based on existing ones
- **group_by(data, col1, col2, etc.)** > group a dataframe based on categorical columns
- **summarise(data, new_summary = calculation)** > summarise data e.g. groups means

# Cheatsheet

https://github.com/rstudio/cheatsheets/blob/main/data-transformation.pdf



Data transformation with dplyr : : **CHEATSHEET**

**dplyr** functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

**pipes**

**x |> f(y)** becomes **f(x, y)**

## Summarize Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

**summarize(**.data, …**)** Compute table of summaries.
mtcars |> summarize(avg = mean(mpg))

**count(**.data, …, wt = NULL, sort = FALSE, name = NULL**)** Count number of rows in each group defined by the variables in … Also **tally()**, **add_count()**, **add_tally()**.
mtcars |> count(cyl)

## Group Cases

Use **group_by(**.data, …, .add = FALSE, .drop = TRUE**)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars |>
  group_by(cyl) |>
  summarize(avg = mean(mpg))
```

Use **rowwise(**.data, …**)** to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

```
starwars |>
  rowwise() |>
  mutate(film_count = length(films))
```

**ungroup(**x, …**)** Returns ungrouped copy of table.
g_mtcars <- mtcars |> group_by(cyl)
ungroup(g_mtcars)

## Manipulate Cases

### EXTRACT CASES

Row functions return a subset of rows as a new table.

**filter(**.data, …, .preserve = FALSE**)** Extract rows that meet logical criteria.
mtcars |> filter(mpg > 20)

**distinct(**.data, …, .keep_all = FALSE**)** Remove rows with duplicate values.
mtcars |> distinct(gear)

**slice(**.data, …, .preserve = FALSE**)** Select rows by position.
mtcars |> slice(10:15)

**slice_sample(**.data, …, n, prop, weight_by = NULL, replace = FALSE**)** Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
mtcars |> slice_sample(n = 5, replace = TRUE)

**slice_min(**.data, order_by, …, n, prop, with_ties = TRUE**)** and **slice_max()** Select rows with the lowest and highest values.
mtcars |> slice_min(mpg, prop = 0.25)

**slice_head(**.data, …, n, prop**)** and **slice_tail()** Select the first or last rows.
mtcars |> slice_head(n = 5)

**Logical and boolean operators to use with filter()**

| | | | | | | |
|---|---|---|---|---|---|---|
| == | < | <= | is.na() | %in% | \| | xor() |
| != | > | >= | !is.na() | ! | & | |

See **?base::Logic** and **?Comparison** for help.

### ARRANGE CASES

**arrange(**.data, …, .by_group = FALSE**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
mtcars |> arrange(mpg)
mtcars |> arrange(desc(mpg))

### ADD CASES

**add_row(**.data, …, .before = NULL, .after = NULL**)** Add one or more rows to a table.
cars |> add_row(speed = 1, dist = 1)

## Manipulate Variables

### EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

**pull(**.data, var = -1, name = NULL, …**)** Extract column values as a vector, by name or index.
mtcars |> pull(wt)

**select(**.data, …**)** Extract columns as a table.
mtcars |> select(mpg, wt)

**relocate(**.data, …, .before = NULL, .after = NULL**)** Move columns to new position.
mtcars |> relocate(mpg, cyl, .after = last_col())

**Use these helpers with select() and across()**
e.g. mtcars |> select(mpg:cyl)

| | | |
|---|---|---|
| **contains**(match) | **num_range**(prefix, range) | **:**, e.g., mpg:cyl |
| **ends_with**(match) | **all_of**(x)/**any_of**(x, …, vars) | **!**, e.g., !gear |
| **starts_with**(match) | **matches**(match) | **everything()** |

### MANIPULATE MULTIPLE VARIABLES AT ONCE

```
df <- tibble(x_1 = c(1, 2), x_2 = c(3, 4), y = c(4, 5))
```

**across(**.cols, .funs, …, .names = NULL**)** Summarize or mutate multiple columns in the same way.
df |> summarize(across(everything(), mean))

**c_across(**.cols**)** Compute across columns in row-wise data.
df |>
rowwise() |>
mutate(x_total = sum(c_across(1:2)))

### MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

**mutate(**.data, …, .keep = "all", .before = NULL, .after = NULL**)** Compute new column(s). Also **add_column()**.
mtcars |> mutate(gpm = 1 / mpg)
mtcars |> mutate(gpm = 1 / mpg, .keep = "none")

**rename(**.data, …**)** Rename columns. Use **rename_with()** to rename with a function.
mtcars |> rename(miles_per_gallon = mpg)

## Vectorized Functions

### TO USE WITH MUTATE ()

**mutate()** applies vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

### OFFSET

dplyr::**lag()** - offset elements by 1
dplyr::**lead()** - offset elements by -1

### CUMULATIVE AGGREGATE

dplyr::**cumall()** - cumulative all()
dplyr::**cumany()** - cumulative any()
**cummax()** - cumulative max()
dplyr::**cummean()** - cumulative mean()
**cummin()** - cumulative min()
**cumprod()** - cumulative prod()
**cumsum()** - cumulative sum()

### RANKING

dplyr::**cume_dist()** - proportion of all values <=
dplyr::**dense_rank()** - rank w ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

### MATH

**+, - , * , /, ^, %/%, %%** - arithmetic ops
**log(), log2(), log10()** - logs
**<, <=, >, >=, !=, ==** - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

### MISCELLANEOUS

dplyr::**case_when()** - multi-case if_else()

```
starwars |>
  mutate(type = case_when(
    height > 200 | mass > 200 ~ "large",
    species == "Droid"        ~ "robot",
    TRUE                      ~ "other"
  ))
```

dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
**pmax()** - element-wise max()
**pmin()** - element-wise min()

## Summary Functions

### TO USE WITH SUMMARIZE ()

**summarize()** applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function

### COUNT

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
**sum(!is.na())** - # of non-NAs

### POSITION

**mean()** - mean, also **mean(!is.na())**
**median()** - median

### LOGICAL

**mean()** - proportion of TRUEs
**sum()** - # of TRUEs

### ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

### RANK

**quantile()** - nth quantile
**min()** - minimum value
**max()** - maximum value

### SPREAD

**IQR()** - Inter-Quartile Range
**mad()** - median absolute deviation
**sd()** - standard deviation
**var()** - variance

## Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

tibble::**rownames_to_column()** Move row names into col.
a <- mtcars |>
rownames_to_column(var = "C")

tibble::**column_to_rownames()** Move col into row names.
a |> column_to_rownames(var = "C")

Also tibble::**has_rownames()** and tibble::**remove_rownames()**.

## Combine Tables

### COMBINE VARIABLES

**bind_cols(**…, .name_repair**)** Returns tables placed side by side as a single table. Column lengths must be equal. Columns will NOT be matched by id (to do that look at Relational Data below), so be sure to check that both tables are ordered the way you want before binding.

### RELATIONAL DATA

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

**left_join(**x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na"**)** Join matching values from y to x.

**right_join(**x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na"**)** Join matching values from x to y.

**inner_join(**x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na"**)** Join data. Retain only rows with matches.

**full_join(**x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), …, keep = FALSE, na_matches = "na"**)** Join data. Retain all values, all rows.

### COLUMN MATCHING FOR JOINS

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

### COMBINE CASES

**bind_rows(**…, .id = NULL**)** Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured).

### RELATIONAL DATA

Use a "**Filtering Join**" to filter one table against the rows of another.

**semi_join(**x, y, by = NULL, copy = FALSE, …, na_matches = "na"**)** Return rows of x that have a match in y. Use to see what will be included in a join.

**anti_join(**x, y, by = NULL, copy = FALSE, …, na_matches = "na"**)** Return rows of x that do not have a match in y. Use to see what will not be included in a join.

Use a "**Nest Join**" to inner join one table to another into a nested data frame.

**nest_join(**x, y, by = NULL, copy = FALSE, keep = FALSE, name = NULL, …**)** Join data, nesting matches from y in a single new data frame column.

### SET OPERATIONS

**intersect(x, y, …)** Rows that appear in both x and y.

**setdiff(x, y, …)** Rows that appear in x but not y.

**union(x, y, …)** Rows that appear in x or y, duplicates removed). **union_all()** retains duplicates.

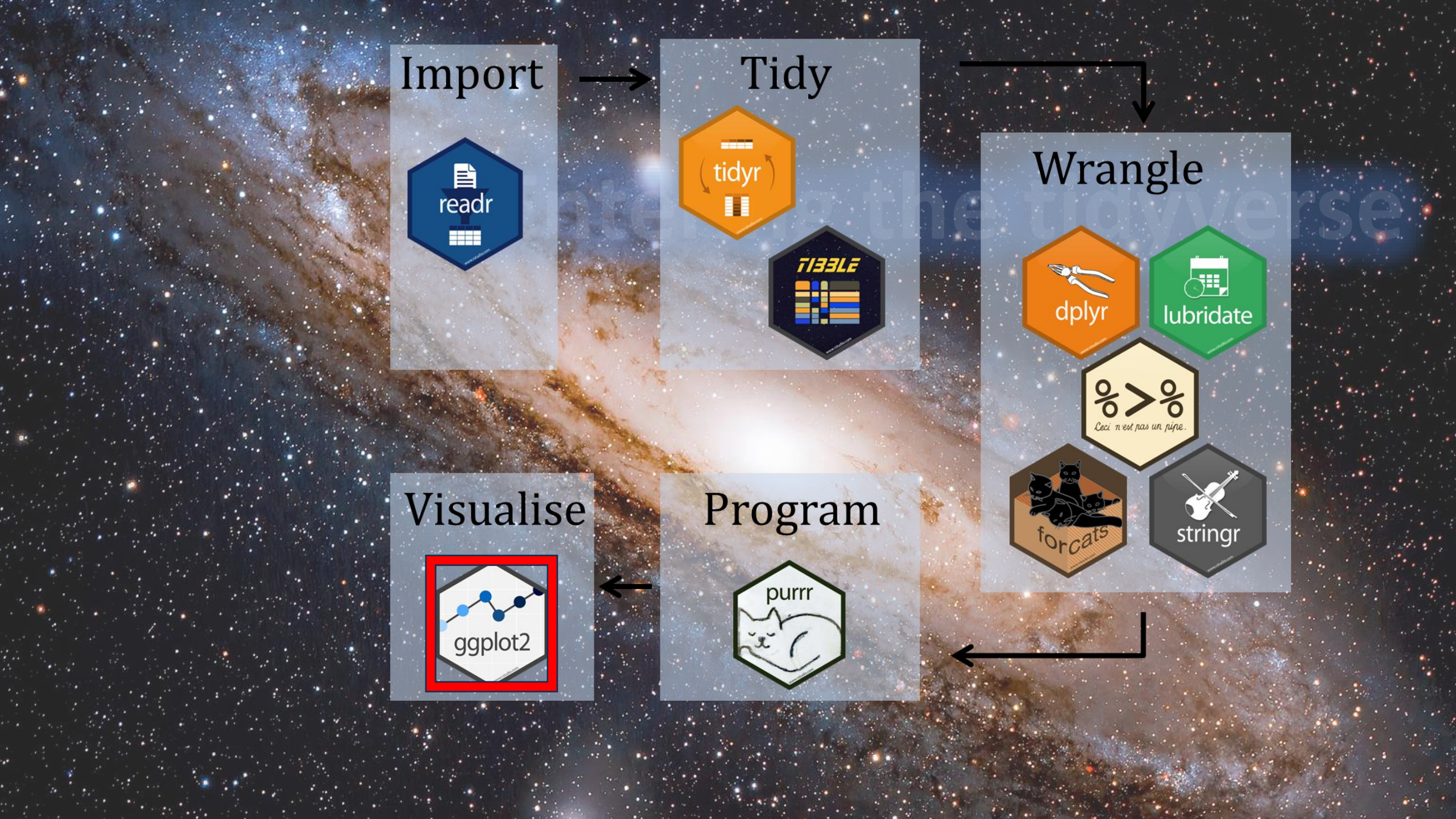
Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

# *Also includes magrittr: pipelines*


- **%>%** (called pipe) to perform operations sequentially

- Improves the **readability** of the code:

```
data %>% filter(blood_oxygen < 0.9) %>%
  group_by(status, sex) %>%
  summarise(mean_ox = mean(blood_oxygen))
```


Import
readr
Tidy
tidyr
tibble
Wrangle
dplyr
lubridate
%>%
Ceci n'est pas un pipe.
forcats
stringr
Program
purrr
Visualise
ggplot2


# Package #2: ggplot2

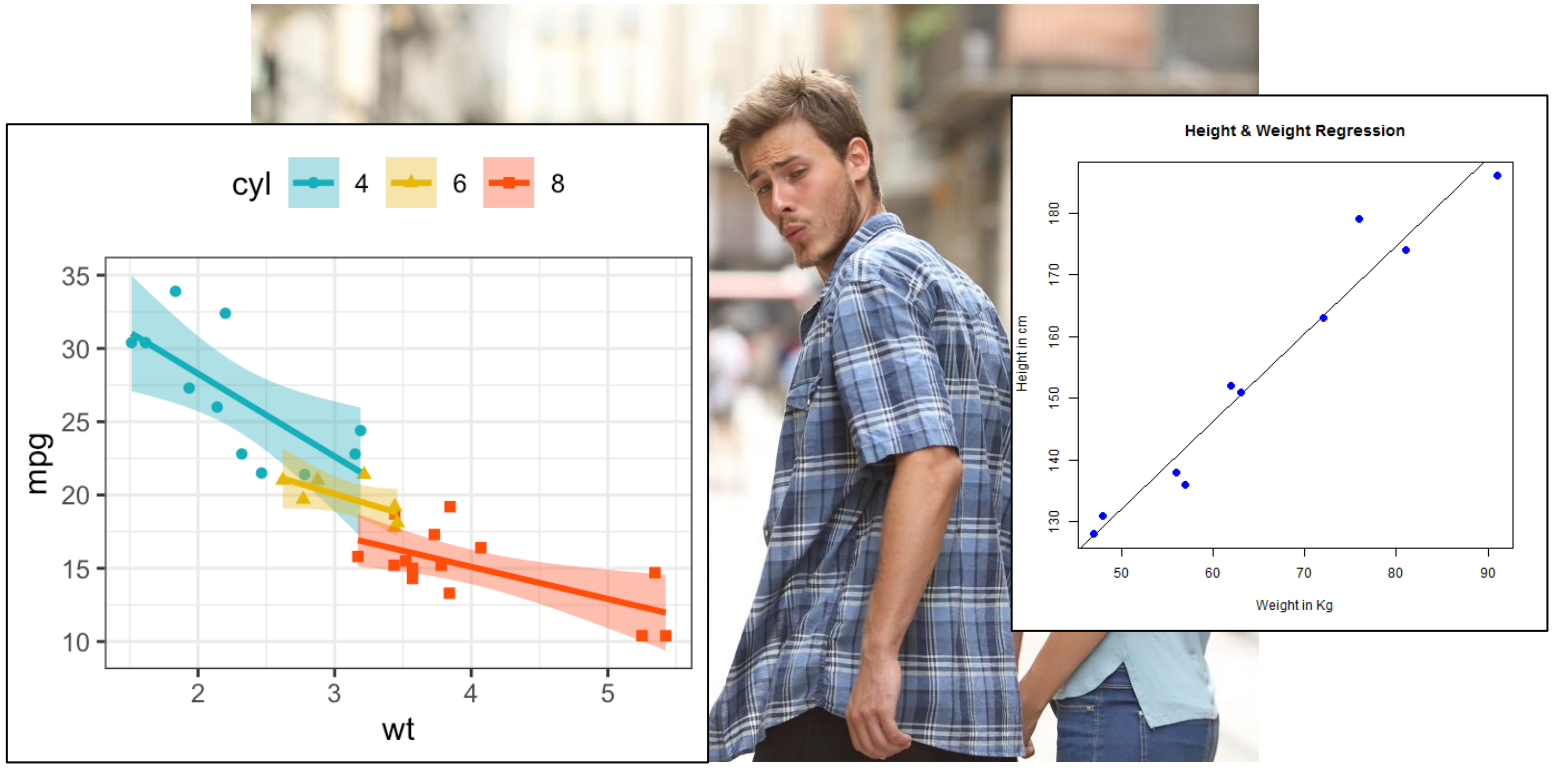
*Plotting library*

**Cheatsheet:**

https://github.com/rstudio/cheatsheets/blob/main/data-visualization.pdf

# *Better visualisations / flexibility*

# *Less code / easier to understand*

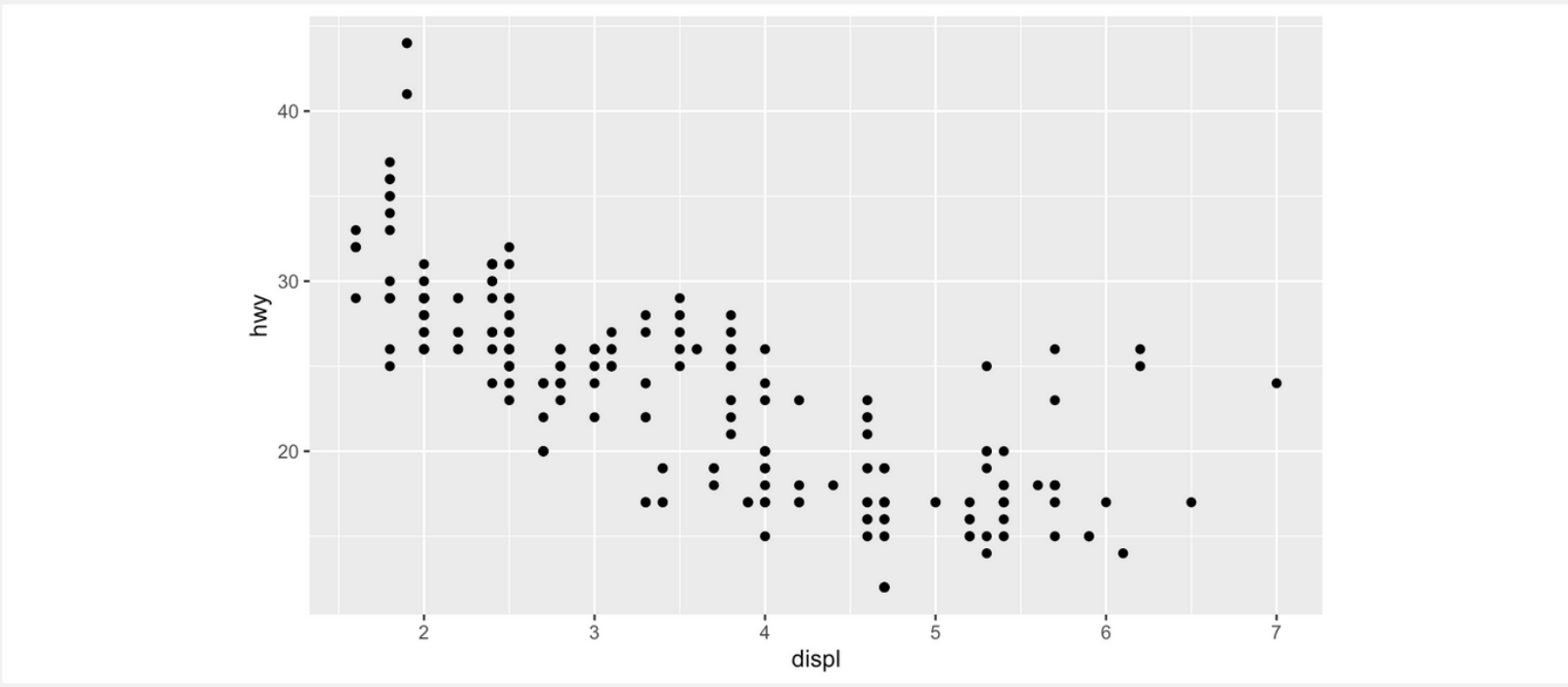


# Basic example

| | manufacturer | model | displ | year | cyl | trans | drv | cty | hwy | fl | class |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | audi | a4 | 1.8 | 1999 | 4 | auto(l5) | f | 18 | 29 | p | compact |
| 2 | audi | a4 | 1.8 | 1999 | 4 | manual(m5) | f | 21 | 29 | p | compact |
| 3 | audi | a4 | 2.0 | 2008 | 4 | manual(m6) | f | 20 | 31 | p | compact |
| 4 | audi | a4 | 2.0 | 2008 | 4 | auto(av) | f | 21 | 30 | p | compact |
| 5 | audi | a4 | 2.8 | 1999 | 6 | auto(l5) | f | 16 | 26 | p | compact |
| 6 | audi | a4 | 2.8 | 1999 | 6 | manual(m5) | f | 18 | 26 | p | compact |
| 7 | audi | a4 | 3.1 | 2008 | 6 | auto(av) | f | 18 | 27 | p | compact |
| 8 | audi | a4 quattro | 1.8 | 1999 | 4 | manual(m5) | 4 | 18 | 26 | p | compact |
| 9 | audi | a4 quattro | 1.8 | 1999 | 4 | auto(l5) | 4 | 16 | 25 | p | compact |
| 10 | audi | a4 quattro | 2.0 | 2008 | 4 | manual(m6) | 4 | 20 | 28 | p | compact |

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```

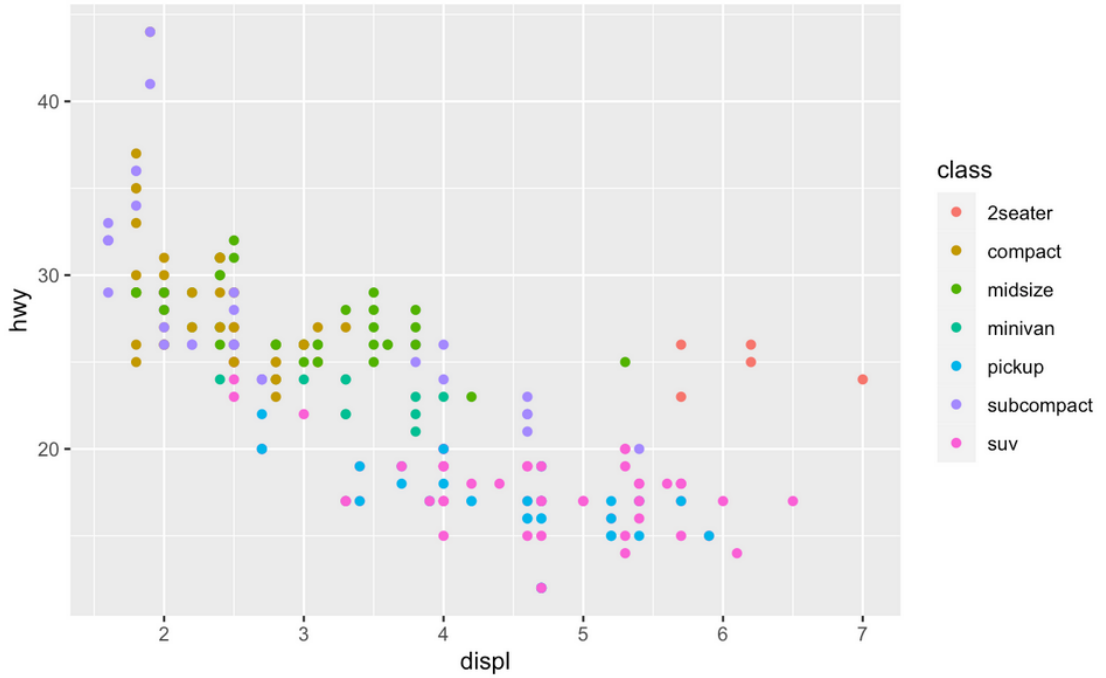
Copy



# *Customisation*

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```

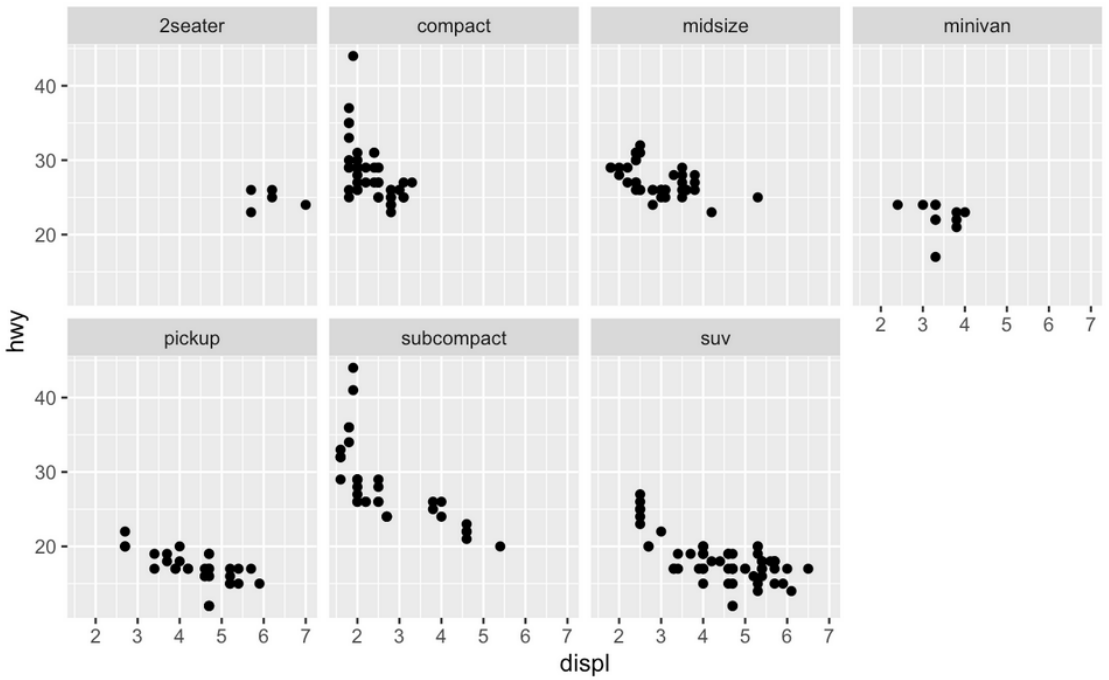
Copy



# *Facet_grid / wrap*

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```

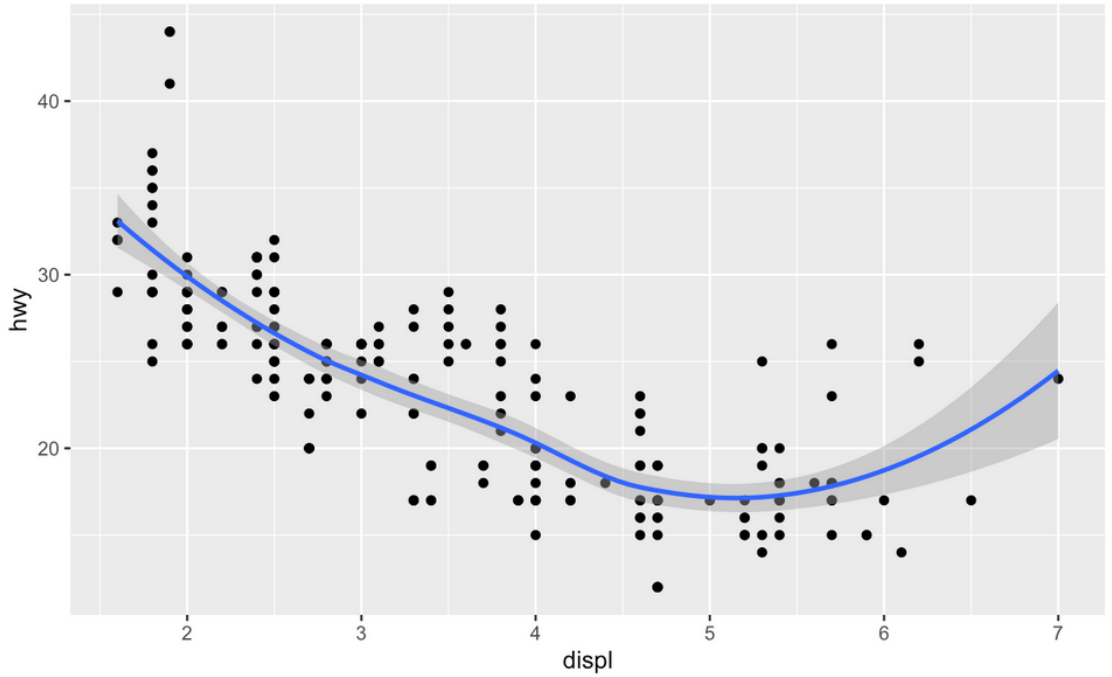
Copy



# *Plotting models*

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

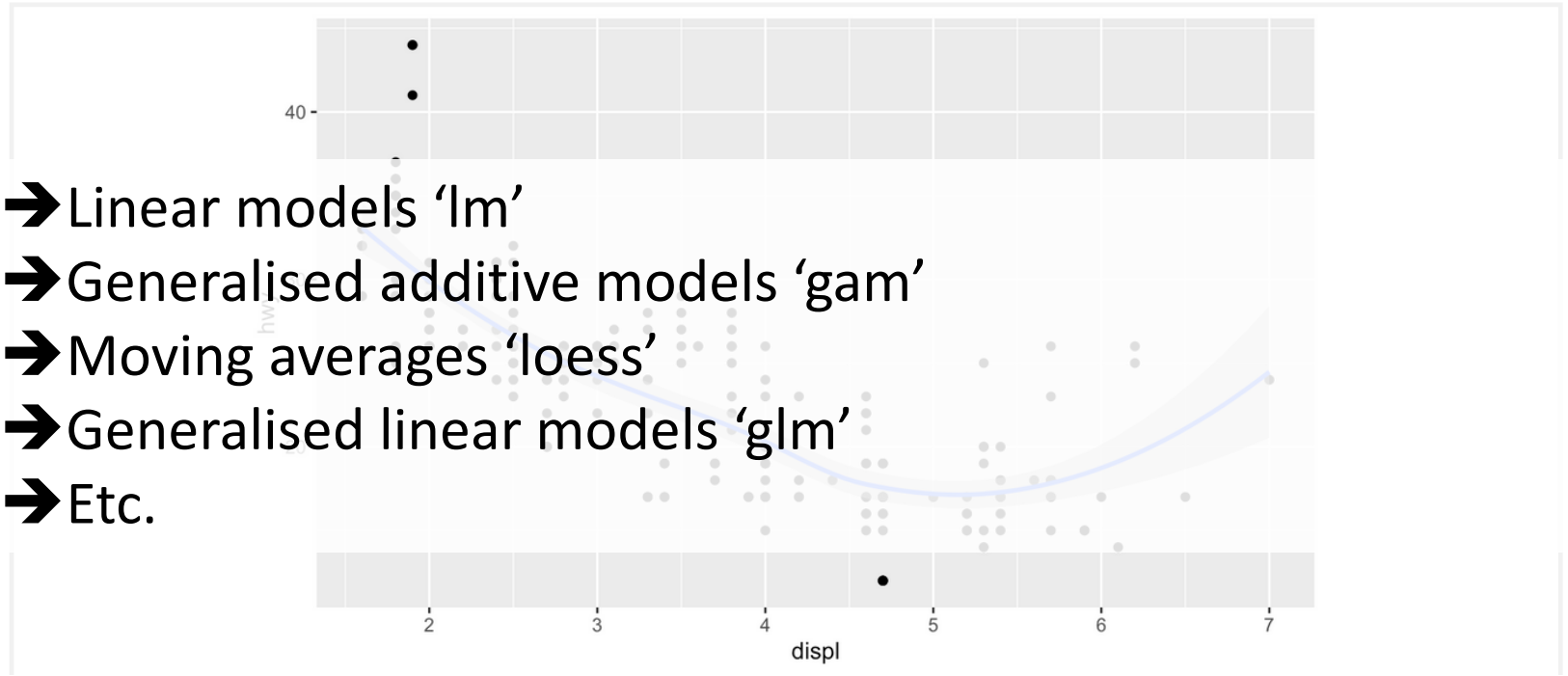
Copy



# *Plotting models*

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy))
```

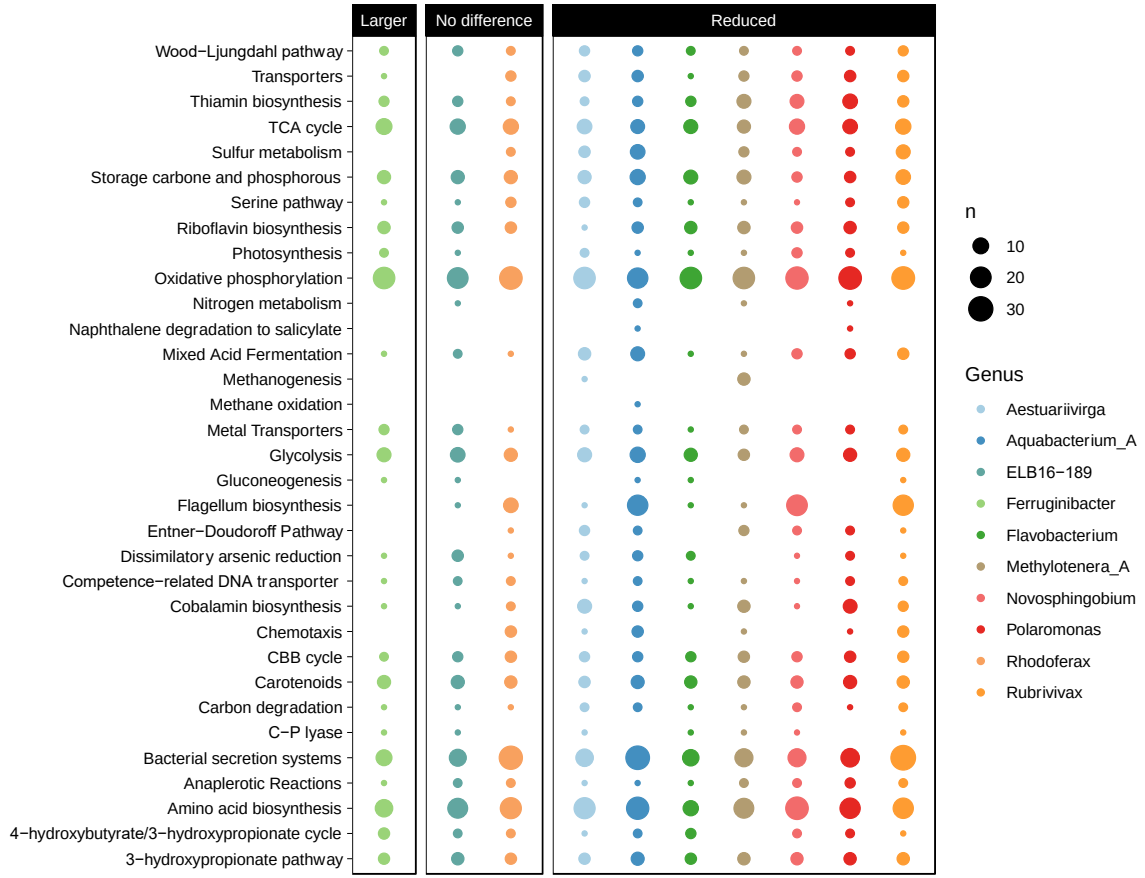
Copy

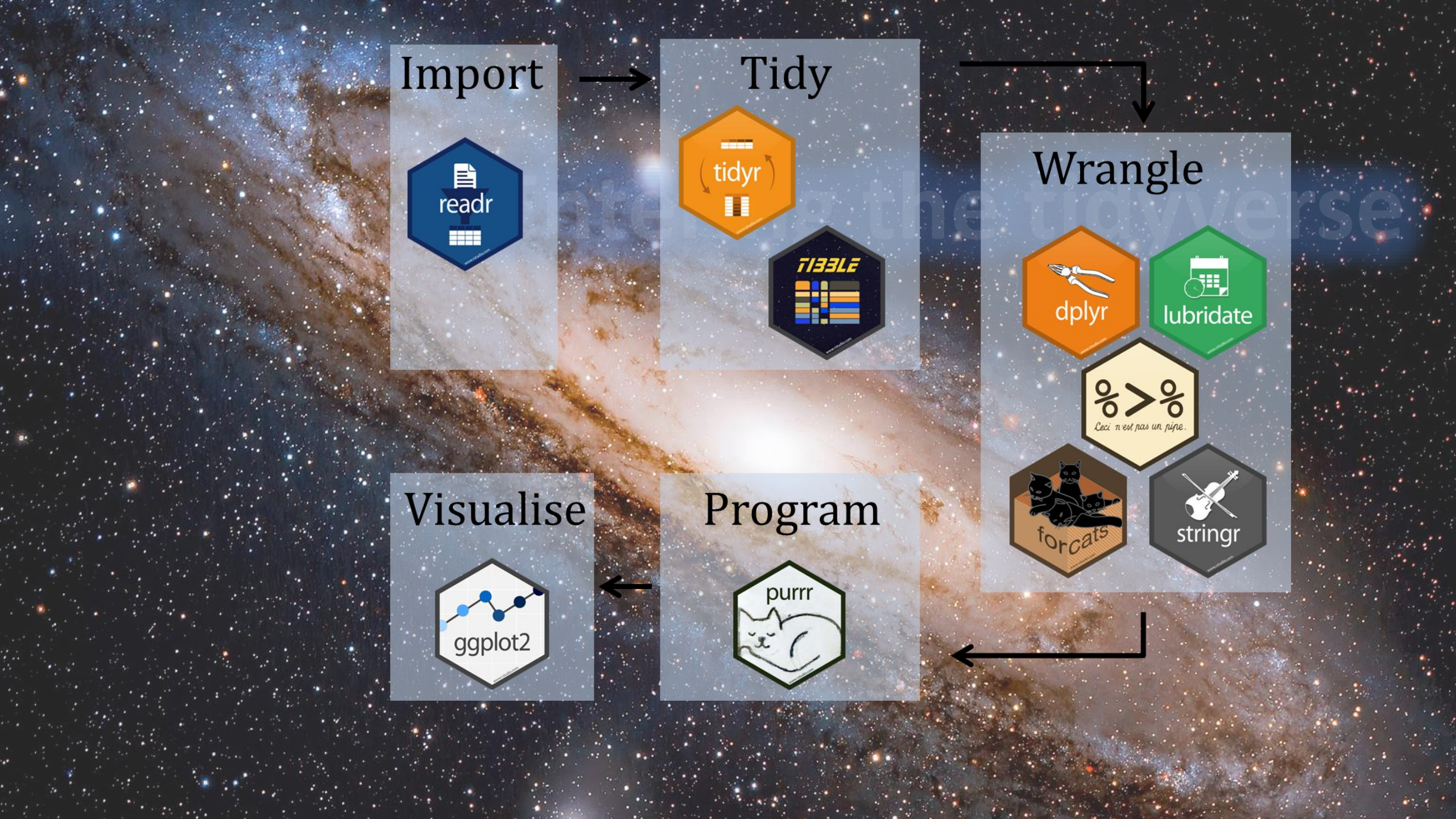


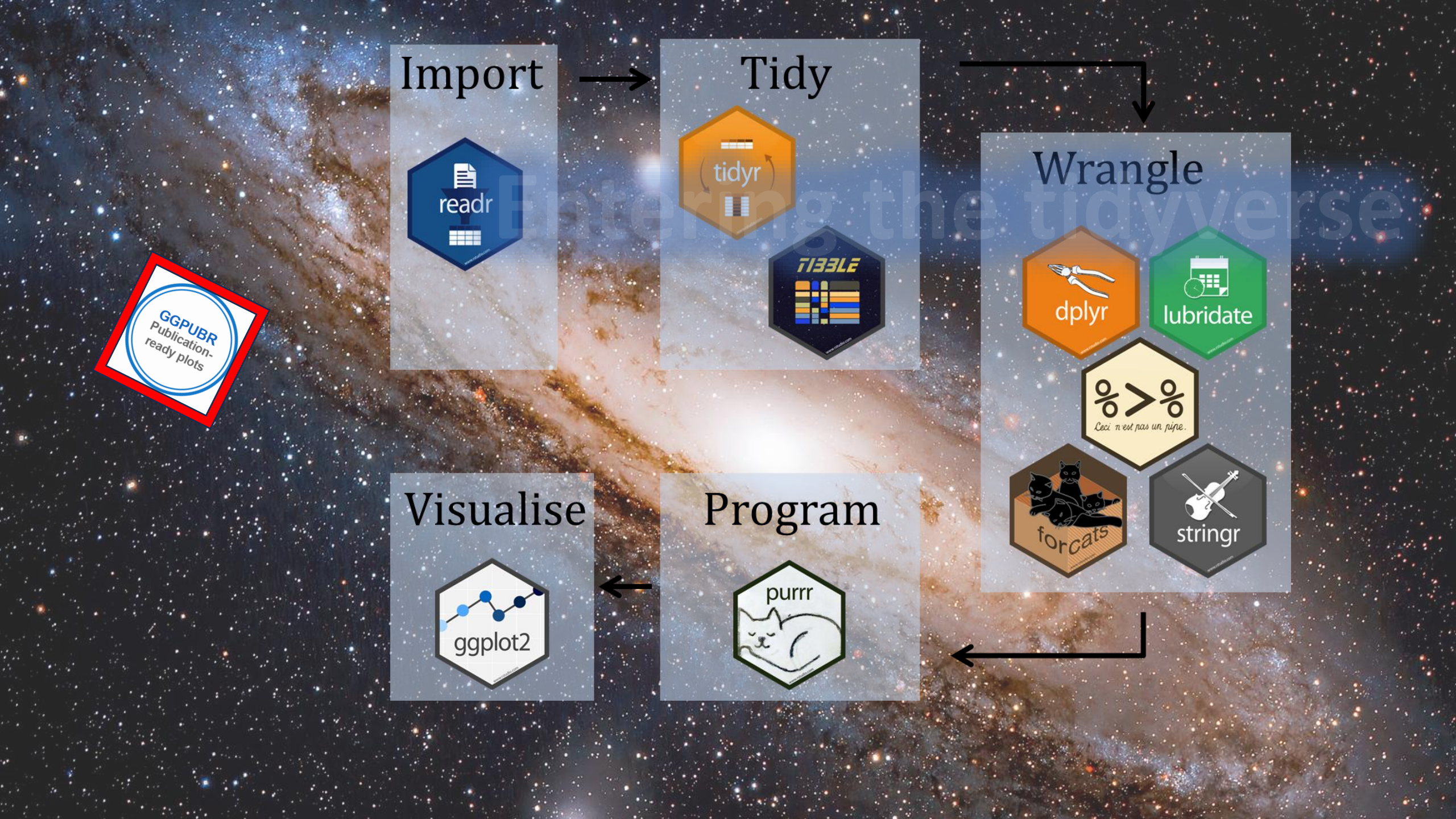

➔Linear models ‘lm’
➔Generalised additive models ‘gam’
➔Moving averages ‘loess’
➔Generalised linear models ‘glm’
➔Etc.

# A more complicated example

```
ggplot(core_funcs, aes(x=Genus, y=Category)) + geom_count(aes(colour=Genus)) + xlab('') + ylab('') +
  scale_colour_manual(values = cols) + theme_linedraw() +
  theme(axis.text.x = element_blank(), axis.ticks.x = element_blank(), panel.grid.major = element_blank(), panel.grid.minor = element_blank()) +
  facet_grid(.~Genus_category, scales = 'free', space = 'free')
```




Import
readr
Tidy
tidyr
tibble
Wrangle
dplyr
lubridate
%>%
Ceci n'est pas un pipe.
forcats
stringr
Program
purrr
Visualise
ggplot2


Import
readr
Tidy
tidyr
tibble
Wrangle
dplyr
lubridate
%>%
Leci n'est pas un pipe.
forcats
stringr
Program
purrr
Visualise
ggplot2
GGPUBR
Publication-ready plots


# Package #3: ggpubr

*Publication ready plots*

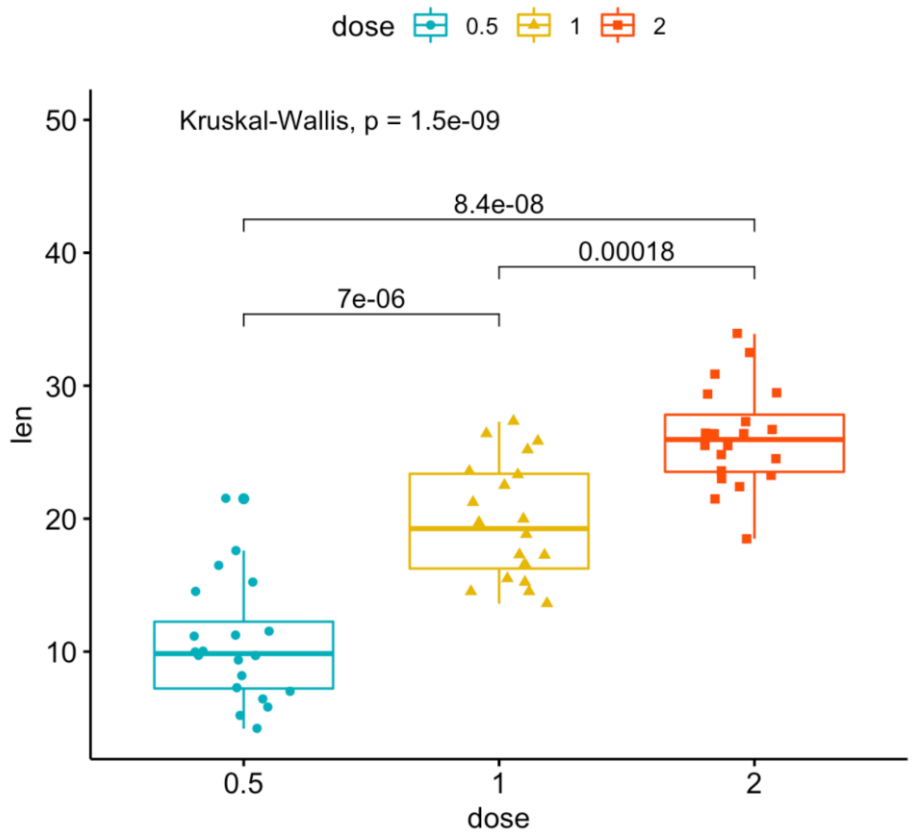
* Not in tidyverse

# *ggplot2 but easier*

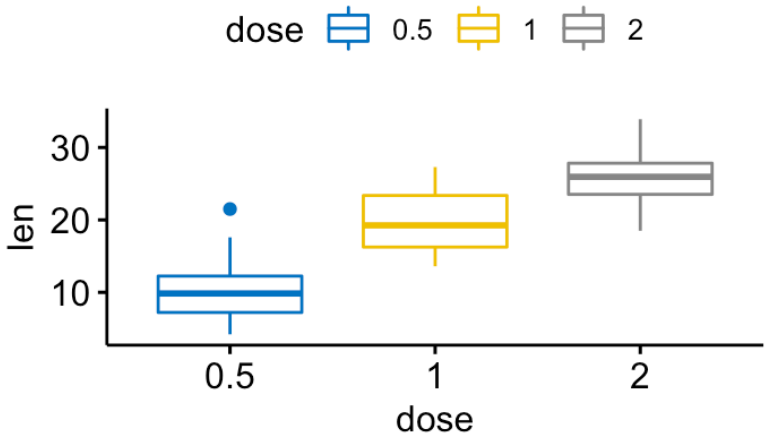
- ggplot2 needs some formatting before publication
- Syntax can be quite opaque and difficult to assimilate
- ggpubr provides easy-to-use equivalents of ggplot2 functions

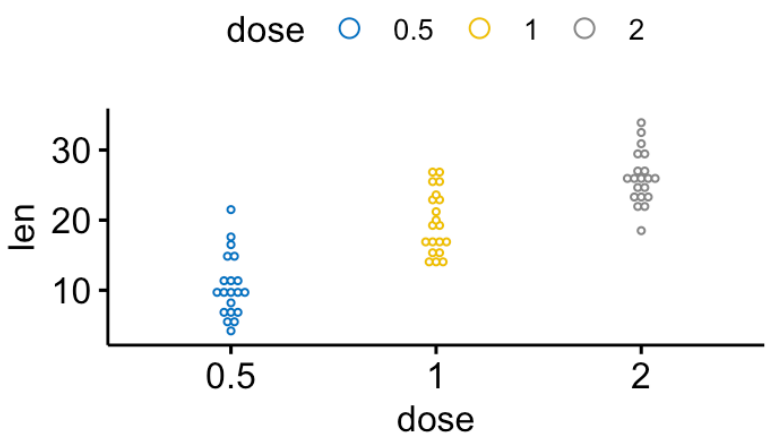
# *Adding statistics on plots*

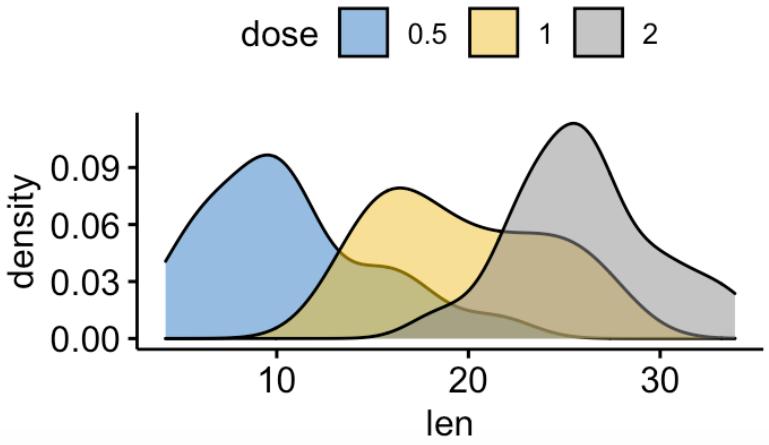
```
 # Add p-values comparing groups
 # Specify the comparisons you want
my_comparisons <- list( c("0.5", "1"), c("1", "2"), c("0.5", "2") )
p + stat_compare_means(comparisons = my_comparisons)+ # Add pairwise comparisons p-value
  stat_compare_means(label.y = 50)                    # Add global p-value
```



# *Arranging figures panels*



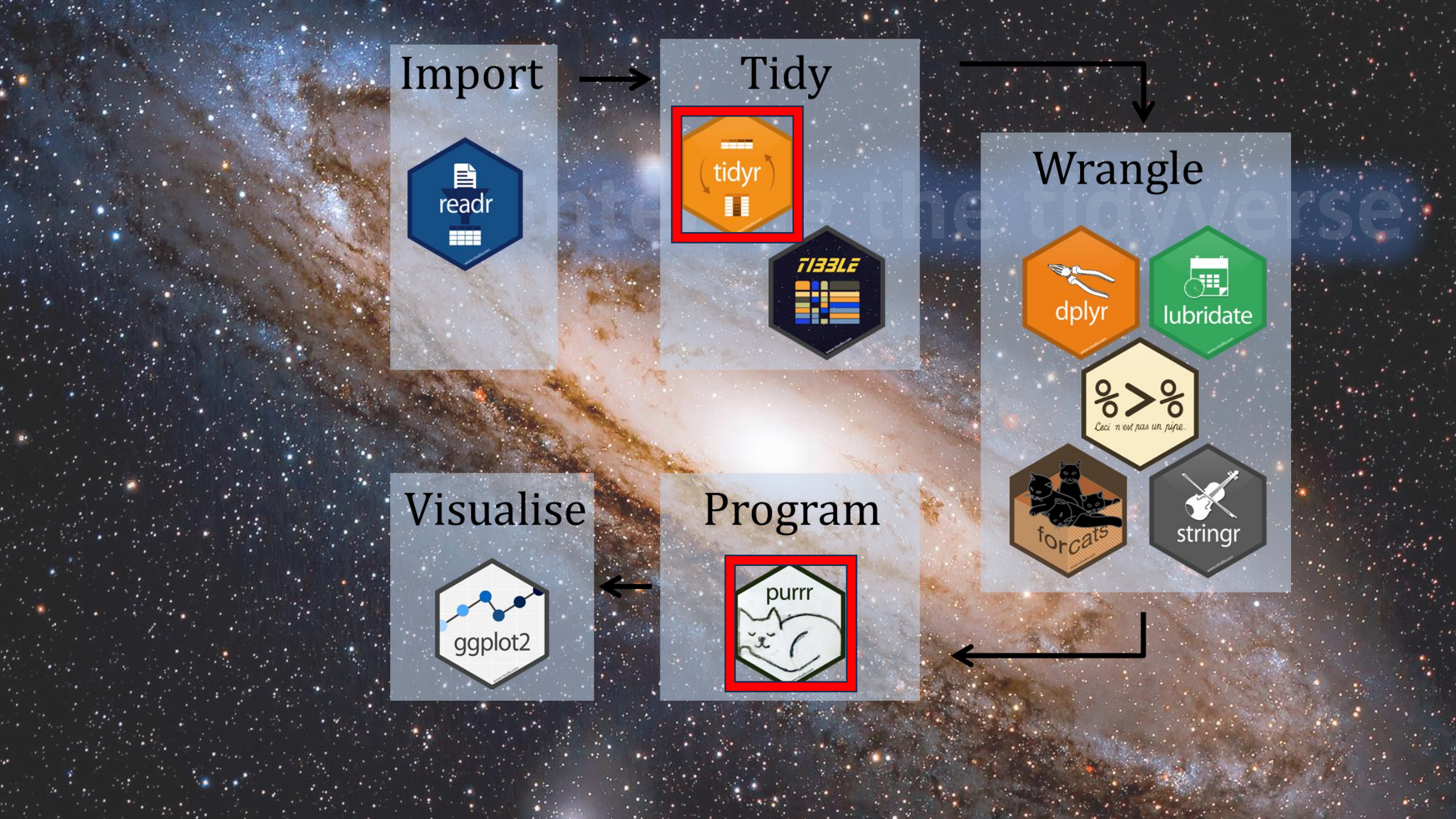




```r
# Box plot
bxp <- ggboxplot(df, x = "dose", y = "len",
    color = "dose", palette = "jco")
# Dot plot
dp <- ggdotplot(df, x = "dose", y = "len",
    color = "dose", palette = "jco")
# Density plot
dens <- ggdensity(df, x = "len", fill = "dose", palette = "jco")

# Arrange
# :::::::::::::::::::::::::::::::::::::::::::::::::
ggarrange(bxp, dp, dens, ncol = 2, nrow = 2)
#> `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```

# Other useful packages


Import
readr
Tidy
tidyr
tibble
Wrangle
dplyr
lubridate
%>%
Ceci n'est pas un pipe.
forcats
stringr
Program
purrr
Visualise
ggplot2

# *Tidyr*

**Cheatsheet:** https://github.com/rstudio/cheatsheets/blob/main/tidyr.pdf

## Reshape Data - Pivot data to reorganize values into a new layout.

table4a

| 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 |

**pivot_longer(**data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE**)**

"Lengthen" data by collapsing several columns into two. Column names move to a new names_to column and values to a new values_to column.

```
pivot_longer(table4a, cols = 2:3, names_to ="year",
  values_to = "cases")
```

table2

| country | year | type | count |
|---|---|---|---|
| A | 1999 | cases | 0.7K |
| A | 1999 | pop | 19M |
| A | 2000 | cases | 2K |
| A | 2000 | pop | 20M |
| B | 1999 | cases | 37K |
| B | 1999 | pop | 172M |
| B | 2000 | cases | 80K |
| B | 2000 | pop | 174M |
| C | 1999 | cases | 212K |
| C | 1999 | pop | 1T |
| C | 2000 | cases | 213K |
| C | 2000 | pop | 1T |

→

| country | year | cases | pop |
|---|---|---|---|
| A | 1999 | 0.7K | 19M |
| A | 2000 | 2K | 20M |
| B | 1999 | 37K | 172M |
| B | 2000 | 80K | 174M |
| C | 1999 | 212K | 1T |
| C | 2000 | 213K | 1T |

**pivot_wider(**data, names_from = "name", values_from = "value"**)**

The inverse of pivot_longer(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

```
pivot_wider(table2, names_from = type,
  values_from = count)
```

# *Purrr*


**Cheatsheet:** https://github.com/rstudio/cheatsheets/blob/main/purrr.pdf

- Never copy and paste more than twice!
- You can use functions or loops
- Purr does the same but with clarity with the map family (equivalent of apply family in base R: lapply, sapply, vapply)
- Super fast (coded in C)!
- Check https://r4ds.had.co.nz/iteration.html for a nice tutorial

# *Purrr*

**Cheatsheet:** https://github.com/rstudio/cheatsheets/blob/main/purrr.pdf


```
df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)
```

```
median(df$a)
#> [1] -0.2457625
median(df$b)
#> [1] -0.2873072
median(df$c)
#> [1] -0.05669771
median(df$d)
#> [1] 0.1442633
```

```
output <- vector("double", ncol(df))  # 1. output
for (i in seq_along(df)) {            # 2. sequence
  output[[i]] <- median(df[[i]])      # 3. body
}
output
#> [1] -0.24576245 -0.28730721 -0.05669771  0.14426335
```

```
map_dbl(df, mean)
#>          a          b          c          d
#> -0.3260369  0.1356639  0.4291403 -0.2498034
map_dbl(df, median)
#>           a           b           c           d
#> -0.51850298  0.02779864  0.17295591 -0.61163819
```

# *Purrr*

**Cheatsheet:** https://github.com/rstudio/cheatsheets/blob/main/purrr.pdf


- Anonymous functions can be be written directly in map

```
models <- mtcars %>%
  split(.$cyl) %>%
  map(function(df) lm(mpg ~ wt, data = df))
```

```
models <- mtcars %>%
  split(.$cyl) %>%
  map(~lm(mpg ~ wt, data = .))
```

- Shortcuts available

```
models %>%
  map(summary) %>%
  map_dbl(~.$r.squared)
#>         4         6         8
#> 0.5086326 0.4645102 0.4229655
```

```
models %>%
  map(summary) %>%
  map_dbl("r.squared")
#>         4         6         8
#> 0.5086326 0.4645102 0.4229655
```

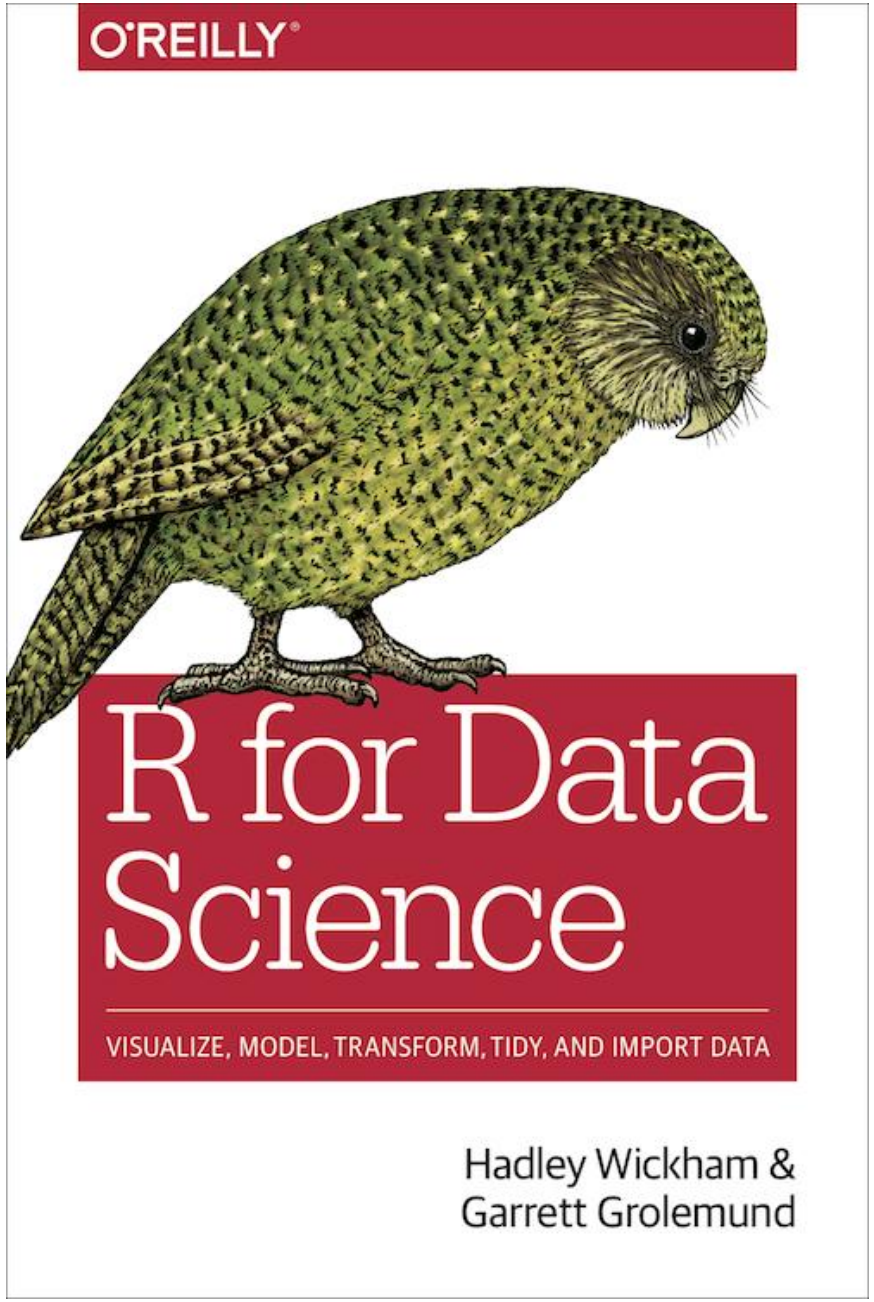
# *Ask the internet*


- Coding is all about **searching** on the internet, don’t be shy to search and **copy some code** (always copy the link too, for credits and reference) !!!

- The more you work with a package/language, the better you will know the associated **keywords** to search

# *Resources*

*https://r4ds.hadley.nz/*



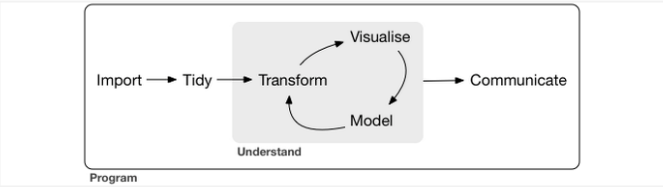
# 1 Introduction

You're reading the first edition of R4DS; for the latest on this topic see the **Introduction chapter** in the second edition.

Data science is an exciting discipline that allows you to turn raw data into understanding, insight, and knowledge. The goal of "R for Data Science" is to help you learn the most important tools in R that will allow you to do data science. After reading this book, you'll have the tools to tackle a wide variety of data science challenges, using the best parts of R.

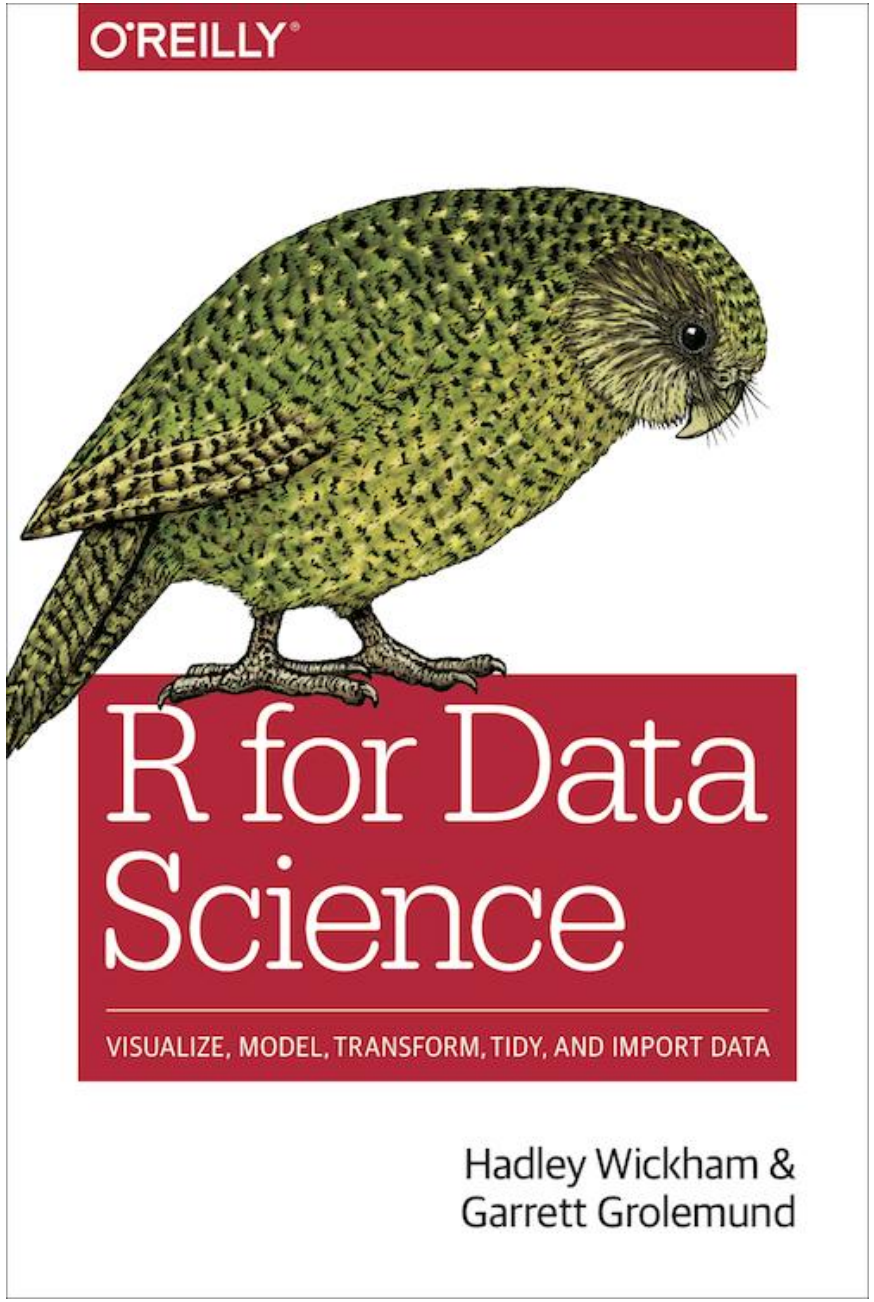
## 1.1 What you will learn

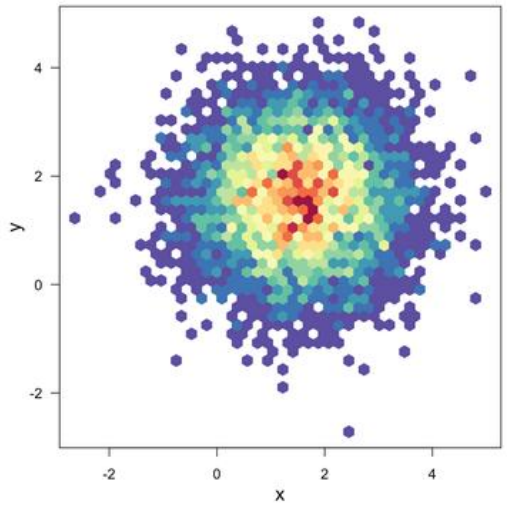
Data science is a huge field, and there's no way you can master it by reading a single book. The goal of this book is to give you a solid foundation in the most important tools. Our model of the tools needed in a typical data science project looks something like this:

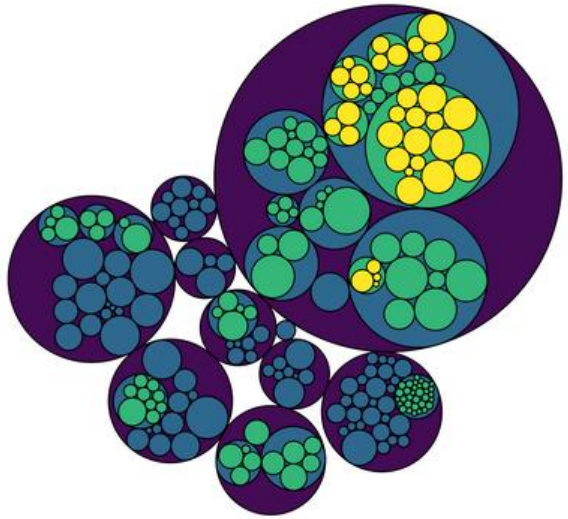


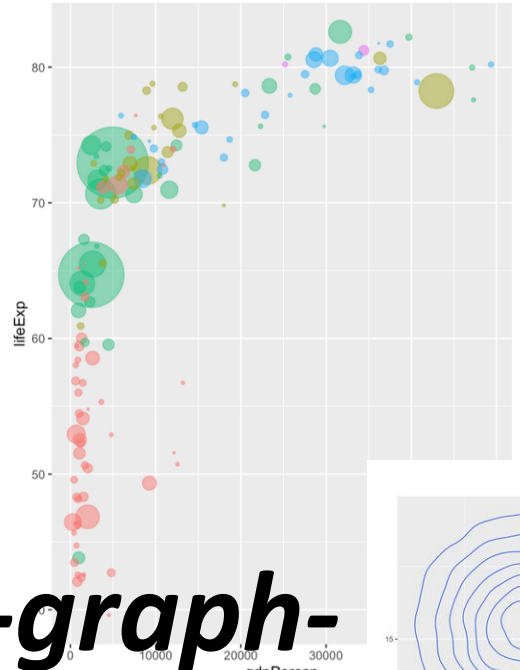

First you must **import** your data into R. This typically means that you take data stored in a file, database, or web application programming interface (API), and load it into a data frame in R. If you can't get your data into R, you can't do data science on it!

Once you've imported your data, it is a good idea to **tidy** it. Tidying your data means storing it in a consistent form that matches the semantics of the dataset with the way it is stored. In brief, when your data is tidy, each column is a variable, and each row is an observation. Tidy data is important because the consistent structure lets you focus your struggle on questions about the data, not fighting to get the data into the right form for different functions.

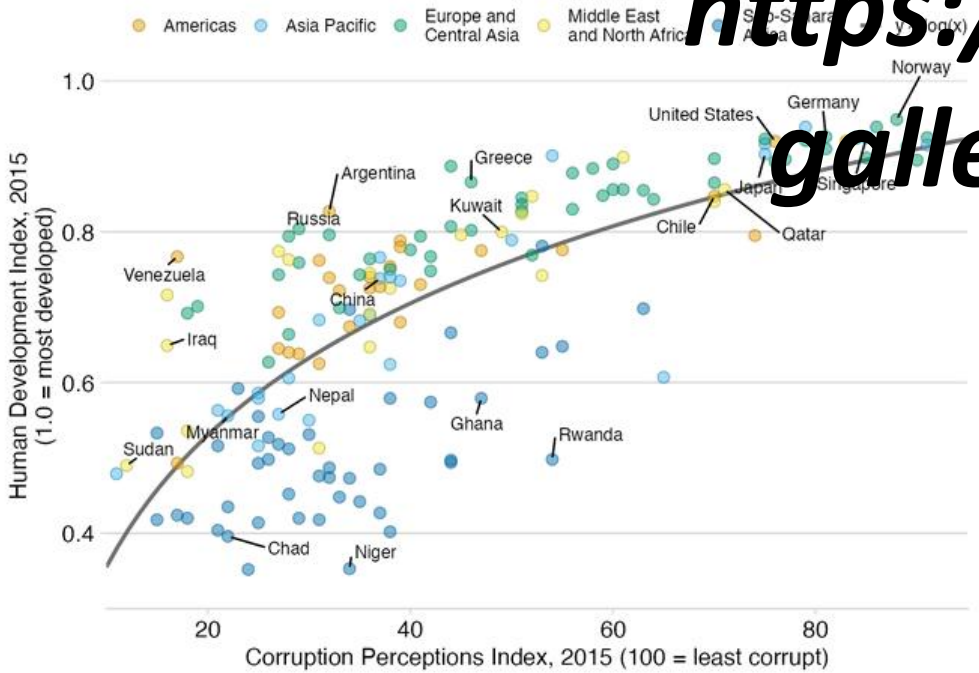


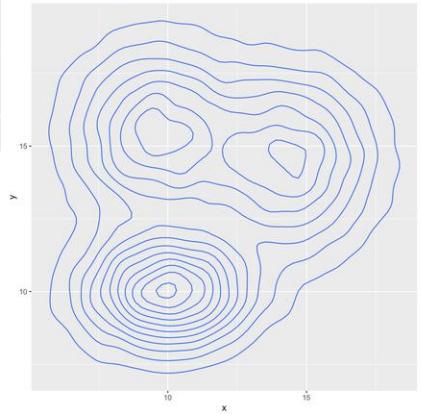


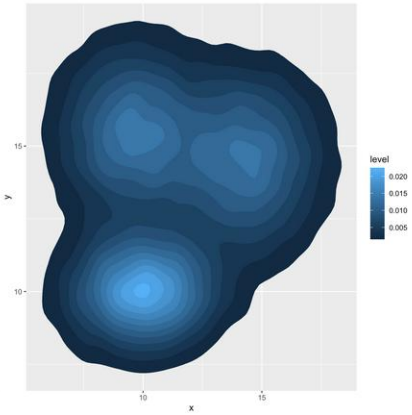



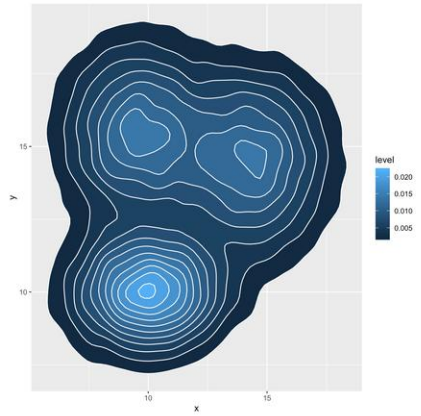


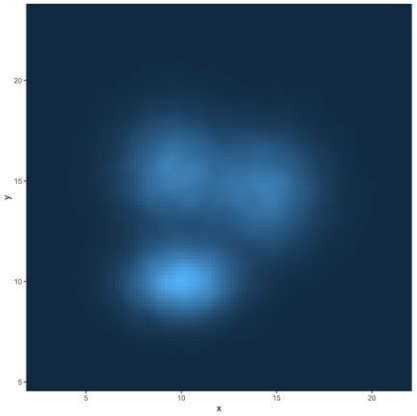

# https://r-graph-gallery.com/



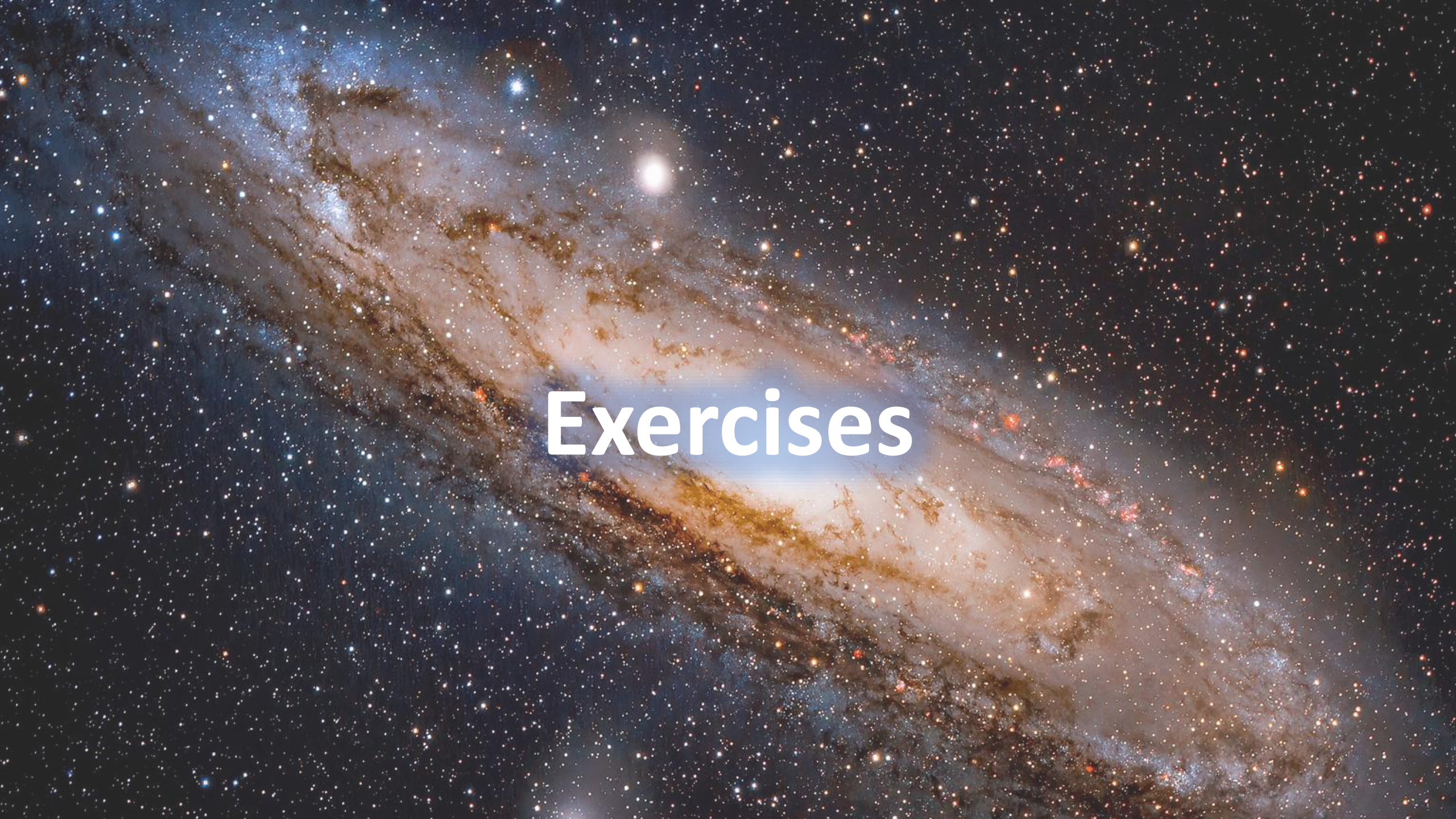



# *Resources*

Other resources:

- https://www.tidyverse.org/
- https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html
- https://rpkgs.datanovia.com/ggpubr/


Exercises

1. ***Load the iris data*** *and do a* ***scatter plot*** *of the sepal.width against the sepal.length with a* ***different colour for each species****.*

2. *Add a* ***linear model*** *for each species.*

3. ***Compare this plot with another one*** *showing the sepal.length/sepal.width ratio to the petal.length/petal.width ratio.* ***Place them side by side.***

4. ***Summarise*** *the sepal.width (mean, sd) for each species, and plot a* ***bar*** *chart with* ***error bars (=sd)****.*

5. *Try to* ***reproduce the figure on the right****.*

***Purr bonus!*** *Compute the mean and the number of unique values in each column of iris.*