DATA MANIPULATION

`tidyverse`, grouping and formulas

Gerko Vink
g.vink@uu.nl
Methodology & Statistics @ Utrecht University

Statistical programming with R team
no email
Summerschool @ Utrecht University

5 Jun 2025

DISCLAIMER

I owe a debt of gratitude to many people as the thoughts and code in these slides are the process of years-long development cycles and discussions with my team, friends, colleagues and peers. When someone has contributed to the content of the slides, I have credited their authorship.

Images are either directly linked, or generated with StableDiffusion or DALL-E. That said, there is no information in this presentation that exceeds legal use of copyright materials in academic settings, or that should not be part of the public domain.

> ⚠ **Warning**
>
> You **may use** any and all content in this presentation - including my name - and submit it as input to generative AI tools, with the following **exception**:
>
> - You must ensure that the content is not used for further training of the model

SLIDE MATERIALS AND SOURCE CODE

Materials

- lecture slides on Moodle
- course page: www.gerkovink.com/sur
- source: github.com/gerkovink/sur

RECAP

Gisteren hebben we deze onderwerpen behandeld:

- Importeren en bestuderen van datasets
- Begrijpen en toepassen van verschillende datatypes en database formats
- Variabelen labelen en (her)coderen
- De blauwdruk van R: frames en environments
- Pipes
- Formules gebruiken in functies

TODAY

Vandaag leren we:

- Het combineren van datasets
- Groeperen en aggregeren
- Nieuwe variabelen creëren
- Filteren en sorteren van gegevens
- Het maken en aanpassen van datagroepen
- Clustering van gegevens

PACKAGES WE USE

```
library(tibble)   # tibbles variation on data frames
library(dplyr)    # data manipulation
library(magrittr) # pipes
library(tidyr)    # data tidying
library(stringr)  # string manipulation
library(psych)    # descriptive statistics
```


MAKE SOME DATA

```r
planet <- c("Mercury", "Venus", "Earth", "Mars",
            "Jupiter", "Saturn", "Uranus", "Neptune")
planet_type <- c("Terrestrial planet", "Terrestrial planet",
                 "Terrestrial planet", "Terrestrial planet", "Gas giant",
                 "Gas giant", "Gas giant", "Gas giant")
diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
rings    <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)
planets <- data.frame(planet_type = factor(planet_type),
                      diameter, rotation, rings,
                      row.names = planet)
planets
```

```
              planet_type diameter rotation rings
Mercury Terrestrial planet    0.382    58.64 FALSE
Venus   Terrestrial planet    0.949  -243.02 FALSE
Earth   Terrestrial planet    1.000     1.00 FALSE
Mars    Terrestrial planet    0.532     1.03 FALSE
Jupiter          Gas giant   11.209     0.41  TRUE
Saturn           Gas giant    9.449     0.43  TRUE
Uranus           Gas giant    4.007    -0.72  TRUE
Neptune          Gas giant    3.883     0.67  TRUE
```


DESCRIPTIVE STATISTICS

psych::describe()

```
planets %>%
  describe(check = TRUE) # converts non-numerical variables
```

```
             vars n   mean    sd median trimmed  mad     min   max  range  skew
planet_type*    1 8   1.50  0.53   1.50    1.50 0.74    1.00  2.00   1.00  0.00
diameter        2 8   3.93  4.23   2.44    3.93 2.58    0.38 11.21  10.83  0.69
rotation        3 8 -22.70 91.32   0.55  -22.70 0.69 -243.02 58.64 301.66 -1.65
rings           4 8    NaN    NA     NA     NaN   NA     Inf  -Inf   -Inf    NA
             kurtosis    se
planet_type*    -2.23  0.19
diameter        -1.36  1.49
rotation         1.32 32.29
rings              NA    NA
```


psych::describe()

```
planets %>%
  describe(omit = TRUE) # omits non-numerical variables
```

```
         vars n   mean    sd median trimmed  mad     min   max  range  skew
diameter    2 8   3.93  4.23   2.44    3.93 2.58    0.38 11.21  10.83  0.69
rotation    3 8 -22.70 91.32   0.55  -22.70 0.69 -243.02 58.64 301.66 -1.65
         kurtosis    se
diameter    -1.36  1.49
rotation     1.32 32.29
```


DATASETS COMBINEREN

JOINING DATA

```
band_members
```

```
# A tibble: 3 × 2
  name  band
  <chr> <chr>
1 Mick  Stones
2 John  Beatles
3 Paul  Beatles
```

```
band_instruments
```

```
# A tibble: 3 × 2
  name  plays
  <chr> <chr>
1 John  guitar
2 Paul  bass
3 Keith guitar
```


INNER JOIN

With an inner join, we combine two data frames based on a common key. Only the rows with matching keys in both data frames are kept.

```
1 band_members %>% inner_join(band_instruments)
```

```
# A tibble: 2 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
```


LEFT JOIN

With a left join, we keep all rows from the left data frame and only the matching rows from the right data frame. If there is no match, the result will contain NA for the columns from the right data frame.

```
1 band_members %>% left_join(band_instruments)
```

```
# A tibble: 3 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
```


RIGHT JOIN

With a right join, we keep all rows from the right data frame and only the matching rows from the left data frame. If there is no match, the result will contain NA for the columns from the left data frame.

```
band_members %>% right_join(band_instruments)
```

```
# A tibble: 3 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 John  Beatles guitar
2 Paul  Beatles bass
3 Keith <NA>    guitar
```


FULL JOIN

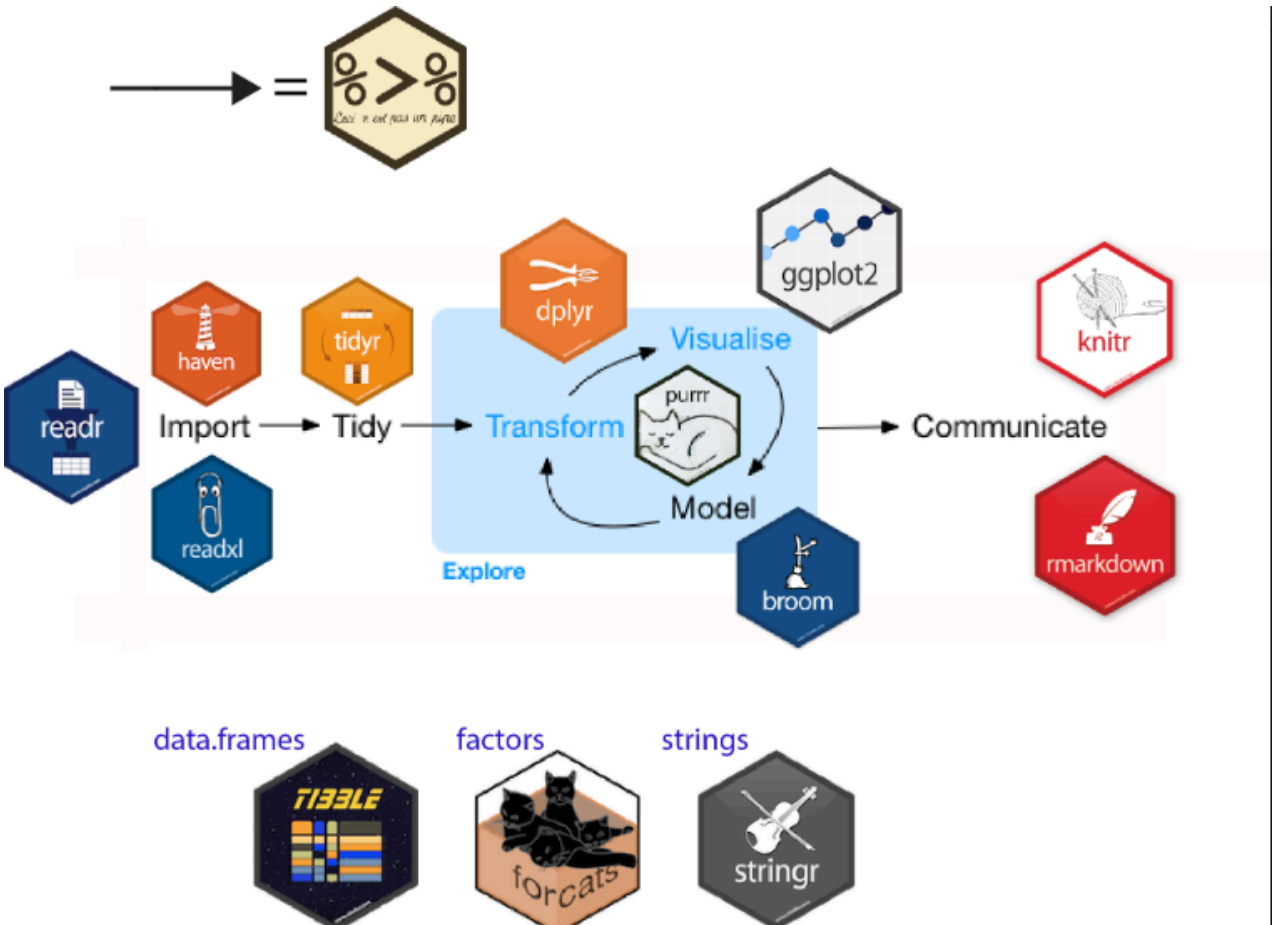
With a full join, we keep all rows from both data frames. If there is no match, the result will contain NA for the columns from the other data frame.

```
1 band_members %>% full_join(band_instruments)
```

```
# A tibble: 4 × 3
  name  band    plays
  <chr> <chr>   <chr>
1 Mick  Stones  <NA>
2 John  Beatles guitar
3 Paul  Beatles bass
4 Keith <NA>    guitar
```


THE TIDYVERSE PACKAGES

`tidyverse` AND THE DATA ANALYSIS CYCLE



TIDYVERSE AND THE VERBS OF DATA MANIPULATION

Leading principle: language of programming should really behave like a language, tidy**verse**.

tidyverse: a few key **verb** that perform common types of data manipulation.

TIDY DATA

The `tidyverse` packages operate on *tidy* data:

1. Each column is a **variable**
2. Each row is an **observation**
3. Each cell is a single **value**

Filter

	id	obs1	obs2
1	1	4	2
2	2	9	6
3	3	5	3

Filter

	id	obs	score
1	1	1	4
2	2	1	9
3	3	1	5
4	1	2	2
5	2	2	6
6	3	2	3

Untidy versus tidy data

THE `dplyr` PACKAGE

DATA MANIPULATION WITH `dplyr`

The *dplyr* package is a specialized package for working with `data.frames` (and the related `tibble`) to transform and summarize tabular data:

- summary statistics for grouped data
- selecting variables
- filtering cases
- (re)arranging cases
- computing new variables
- recoding variables

dplyr CHEATSHEET

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).

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.

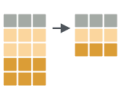
mtcars |>
group_by(cyl) |>

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()	!	&	

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() |>

COMMON `dplyr` FUNCTIONS

There are many functions available in dplyr, but we will focus on just the following `dplyr` functions (verbs):

dplyr verbs	Description
`glimpse()`	a transposed print of the data that shows all variables
`select()`	selects variables (columns) based on their names
`filter()`	subsets the rows of a data frame based on their values
`arrange()`	re-order or arrange rows
`mutate()`	adds new variables, or new variables that are functions of existing variables
`summarise()`	creates a new data frame with statistics of the variables (optional grouped by another variables)
`group_by()`	allows for group operations in the “split-apply-combine” concept

Check the `dplyr` cheat sheet for examples.

dplyr::glimpse()

- Prints a transposed version of the data: variables are the rows, observations are the columns.
- Makes it possible to see every column in a data frame.
- It is similar to `str()`, but shows more data.
- `str()` shows more detailed information about data structure.

```
glimpse(planets)
Rows: 8
Columns: 4
$ planet_type <fct> Terrestrial planet, Terrestrial planet, Terrestrial planet…
$ diameter    <dbl> 0.382, 0.949, 1.000, 0.532, 11.209, 9.449, 4.007, 3.883
$ rotation    <dbl> 58.64, -243.02, 1.00, 1.03, 0.41, 0.43, -0.72, 0.67
$ rings       <lgl> FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE
str(planets)
'data.frame':   8 obs. of  4 variables:
 $ planet_type: Factor w/ 2 levels "Gas giant","Terrestrial planet": 2 2 2 2 1 1 1 1
 $ diameter   : num  0.382 0.949 1 0.532 11.209 ...
 $ rotation   : num  58.64 -243.02 1 1.03 0.41 ...
 $ rings      : logi  FALSE FALSE FALSE FALSE TRUE TRUE ...
```

COMPUTE NEW VARIABLES

COMPUTE NEW VARIABLES WITH `dplyr::mutate()`

```
data %>%
  dplyr::mutate(..., .keep = c("all", ...), .before = NULL, .after = NULL)
```

```
planets %>%
  mutate(rotation_diameter = rotation/diameter, .keep = "all") %>%
  glimpse()
```

```
Rows: 8
Columns: 5
$ planet_type       <fct> Terrestrial planet, Terrestrial planet, Terrestrial …
$ diameter          <dbl> 0.382, 0.949, 1.000, 0.532, 11.209, 9.449, 4.007, 3.…
$ rotation          <dbl> 58.64, -243.02, 1.00, 1.03, 0.41, 0.43, -0.72, 0.67
$ rings             <lgl> FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE
$ rotation_diameter <dbl> 153.50785340, -256.08008430, 1.00000000, 1.93609023,…
```


TEMPORARY / PERMANENT CHANGES

The pipe operations do not make changes to the original data set, unless you save the results:

Temporary:

```
planets %>%
  mutate(rotation_diameter = rotation/diameter)
```

```
names(planets)
```

```
[1] "planet_type" "diameter"    "rotation"    "rings"
```

Changes saved in new data frame:

```
new_data_set <- planets %>%
  mutate(rotation_diameter = rotation/diameter)
names(new_data_set)
[1] "planet_type"       "diameter"          "rotation"
[4] "rings"             "rotation_diameter"
```


REMEMBER THE %<>% PIPE?

We could have used the %<>% pipe to save the changes to the original data frame directly:

Changes saved in new data frame:

```
planets %<>%
  mutate(rotation_diameter = rotation/diameter)
names(new_data_set)
```

You should only do so if the intend is to overwrite the original set.

FILTERING AND SORTING

SELECT COLUMNS WITH `dplyr::select()`

Select variables `type` and `diameter` from the planets data frame:

```
planets %>%
  select(planet_type, diameter)
```

```
              planet_type diameter
Mercury Terrestrial planet    0.382
Venus   Terrestrial planet    0.949
Earth   Terrestrial planet    1.000
Mars    Terrestrial planet    0.532
Jupiter          Gas giant   11.209
Saturn           Gas giant    9.449
Uranus           Gas giant    4.007
Neptune          Gas giant    3.883
```


SELECT NUMERIC COLUMNS WITH `dplyr::select()`

Select numerical variables with `where(is.numeric)`:

```
planets %>%
  select(where(is.numeric))
```

```
        diameter rotation
Mercury    0.382    58.64
Venus      0.949  -243.02
Earth      1.000     1.00
Mars       0.532     1.03
Jupiter   11.209     0.41
Saturn     9.449     0.43
Uranus     4.007    -0.72
Neptune    3.883     0.67
```


SELECT FACTOR COLUMNS WITH `dplyr::select()`

Select numerical variables with `where(is.factor)`:

```
planets %>%
  select(where(is.factor))
```

```
              planet_type
Mercury Terrestrial planet
Venus   Terrestrial planet
Earth   Terrestrial planet
Mars    Terrestrial planet
Jupiter          Gas giant
Saturn           Gas giant
Uranus           Gas giant
Neptune          Gas giant
```


SELECT ROWS WITH `dplyr::filter()`

Selects subsets of the rows of a data frame based on their values.

Filter the data based on the planets that have a ring and that are gas giants:

```
planets %>%
  filter(rings == TRUE,
         planet_type == "Gas giant")
```

```
        planet_type diameter rotation rings
Jupiter   Gas giant   11.209     0.41  TRUE
Saturn    Gas giant    9.449     0.43  TRUE
Uranus    Gas giant    4.007    -0.72  TRUE
Neptune   Gas giant    3.883     0.67  TRUE
```


SELECT ROWS AND COLUMNS

Select diameter only for the planets that have a ring and that are gas giants:

```
planets %>%
  select(diameter) %>%
  filter(rings == TRUE,
         planet_type == "Gas giant")
```

```
        diameter
Jupiter   11.209
Saturn     9.449
Uranus     4.007
Neptune    3.883
```


SELECTING SPECIFIC COLUMNS

Select diameter only for the planets that have a ring and that are gas giants:

```
planets %>%
  select(starts_with("r")) # selects all columns that start with "r"
```

```
        rotation rings
Mercury    58.64 FALSE
Venus    -243.02 FALSE
Earth       1.00 FALSE
Mars        1.03 FALSE
Jupiter     0.41  TRUE
Saturn      0.43  TRUE
Uranus     -0.72  TRUE
Neptune     0.67  TRUE
```

MOVING ROWNAMES TO COLUMNS

To move the row names to a column, you can use `rownames_to_column()` from the `tibble` package:

```
planets %>%
  rownames_to_column(var = "planet_name") %>%
  select(planet_name, diameter, rings)
```

```
  planet_name diameter rings
1     Mercury    0.382 FALSE
2       Venus    0.949 FALSE
3       Earth    1.000 FALSE
4        Mars    0.532 FALSE
5     Jupiter   11.209  TRUE
6      Saturn    9.449  TRUE
7      Uranus    4.007  TRUE
8     Neptune    3.883  TRUE
```


SELECTING SPECIFIC ROWS

In this case, we want to select the planets that start with the letter “M”. We first use the `rownames_to_column()` function to move the row names to a column, and then we use `filter()` with `stringr::str_starts()` to select the rows:

```
planets %>%
  rownames_to_column(var = "planet_name") %>%
  filter(stringr::str_starts(planet_name, "M"))
```

```
  planet_name        planet_type diameter rotation rings
1     Mercury Terrestrial planet    0.382    58.64 FALSE
2        Mars Terrestrial planet    0.532     1.03 FALSE
```

We cannot use `starts_with()` here, because it only works for column names, not for values in a column.

SORTING DATA

RE-ORDER ROWS WITH `dplyr::arrange()`

Order the rows of the `planets` data set on ascending values of `diameter`:

Original data set:

```
1 # just the planets data
2 planets
```

	planet_type	diameter	rotation	rings
Mercury	Terrestrial planet	0.382	58.64	FALSE
Venus	Terrestrial planet	0.949	-243.02	FALSE
Earth	Terrestrial planet	1.000	1.00	FALSE
Mars	Terrestrial planet	0.532	1.03	FALSE
Jupiter	Gas giant	11.209	0.41	TRUE
Saturn	Gas giant	9.449	0.43	TRUE
Uranus	Gas giant	4.007	-0.72	TRUE
Neptune	Gas giant	3.883	0.67	TRUE

Ordered data set, based on diameter:

```
1 planets %>%
2   dplyr::arrange(diameter)
```

	planet_type	diameter	rotation	rings
Mercury	Terrestrial planet	0.382	58.64	FALSE
Mars	Terrestrial planet	0.532	1.03	FALSE
Venus	Terrestrial planet	0.949	-243.02	FALSE
Earth	Terrestrial planet	1.000	1.00	FALSE
Neptune	Gas giant	3.883	0.67	TRUE
Uranus	Gas giant	4.007	-0.72	TRUE
Saturn	Gas giant	9.449	0.43	TRUE
Jupiter	Gas giant	11.209	0.41	TRUE

MULTIPLE TRANSFORMATIONS: BASE R AND `dplyr`

Suppose we want to perform the following transformations:

1. Sort the rows of `planets` on ascending values of `rotation`
2. Select only planets with diameter `> 1`
3. Display the variables `planet_type`, `diameter` and `rotation`

With base R code:

```r
subset(planets[order(planets$rotation), ],
       subset = diameter > 1,
       select = c(planet_type, diameter,
                  rotation))
```

```
        planet_type diameter rotation
Uranus    Gas giant    4.007    -0.72
Jupiter   Gas giant   11.209     0.41
Saturn    Gas giant    9.449     0.43
Neptune   Gas giant    3.883     0.67
```

With `dplyr` and the pipe `%>%` operator

```r
planets %>%
  filter(diameter > 1) %>%
  arrange(rotation) %>%
  select(planet_type, diameter, rotation)
```

```
        planet_type diameter rotation
Uranus    Gas giant    4.007    -0.72
Jupiter   Gas giant   11.209     0.41
Saturn    Gas giant    9.449     0.43
Neptune   Gas giant    3.883     0.67
```


SUMMARY STATISTICS WITH `summarise()`

The `dplyr` function for summarizing data:

```
planets %>%
  summarise(
    mean_diameter = mean(diameter),
    sd_diameter = sd(diameter)
  )
```

```
  mean_diameter sd_diameter
1      3.926375    4.226738
```

- Various summary function(s):
  - `mean()`, `median()`, `sd()`, `var()`, `sum()`, for numeric variables
  - `n()`, `n_distinct()` for counts
  - many others, see: `?dplyr::select` and cheat sheet)

SUMMARIES FOR GROUPS WITH `group_by()`

The `dplyr` function for grouping rows of a data frame is very useful in combination with `summarise()`

Example: group the planets based on having rings (or not) and compute the mean and the standard deviation for each group.

```
planets %>%
  group_by(rings) %>%
  summarise(
    mean_diameter = mean(diameter),
    sd_diameter = sd(diameter)
  )
```

```
# A tibble: 2 × 3
  rings mean_diameter sd_diameter
  <lgl>         <dbl>       <dbl>
1 FALSE         0.716       0.306
2 TRUE          7.14        3.76
```


STANDARD SOLVES FOR MISSING VALUES

DEALING WITH MISSING VALUES IN R

Calculations based on missing values (NA's) are not possible in R:

```
variable <- c(1, 2, NA, 4, 5)
mean(variable)
[1] NA
```

There are two easy ways to perform "listwise deletion":

```
mean(variable, na.rm = TRUE)
[1] 3
mean(na.omit(variable))
[1] 3
```

DEALING WITH MISSING VALUES WITH `dplyr`

```
df$score
[1]  1  2 NA  4  5
```

No solution for missing values:

```
df %>%
  summarise(
    mean_variable = mean(score),
    sd_variable = sd(score)
  )
```

```
  mean_variable sd_variable
1            NA          NA
```

Use `na.rm = TRUE`:

```
df %>%
  summarise(
    mean_variable = mean(score, na.rm = TRUE),
    sd_variable = sd(score, na.rm = TRUE)
  )
```

```
  mean_variable sd_variable
1             3    1.825742
```


STYLE GUIDE FOR CODING PIPES

Code with a single pipe operator on one line and spaces around %>%:

```
data %>% select(X)
```

Code with multiple pipe operators on multiple lines:

```
data %>%
  group_by(X) %>%
  filter(Y > 4) %>%
  summarise(mean(Y))
```

but definitely NOT:

```
data%>%group_by(X)%>%filter(Y>4)%>%summarise(mean(Y))
```

MORE ABOUT CODING STYLE: `tidyverse` STYLE GUIDE

https://style.tidyverse.org/index.html

PRACTICAL