INTERFACULTY CENTER
DATA PROCESSING & STATISTICS
CORE FACILITY

Data Wrangling with R's tidyverse

dplyr, tidyr and friends

w. cools

Compiled Jul 12, 2020

Current draft aims to introduce researchers to data manipulation in R with the `dplyr`, `tidyr`, and `stringr` packages of the `tidyverse` ecosystem.

Our target audience is primarily the research community at VUB / UZ Brussel, those who have some basic experience in R and want to know more.

We invite you to help improve this document by sending us feedback
wilfried.cools@vub.be or anonymously at icds.be/consulting (right side, bottom)

key message on data manipulation

data manipulation is inherent to data analysis, not just a precursor

- no -fit's all data representation-, dependent on analysis or visualization (note: raw data should be unaltered)
- flexible use of data manipulation elicits better data exploration and modeling

data manipulation is best done with coding (as opposed to manual changes), provides the best guarantee to

- efficiently and correctly process data and statistics
- maintain structure and transparency, to support reproducibility

data manipulation is easier and more intuitive when maintaining tidy data

- tidy data: meaning appropriately mapped into structure
  - each row an observation as research unit,
  - each column a variable as property,
  - each cell a particular value, linking row to column
  - note: data can be split into multiple tables (relational data).
- aim for tidy data registration (avoid tedious manipulations)

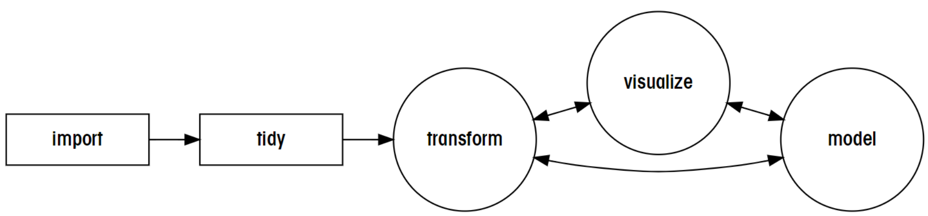

Figure 1: workflow: tidyverse lingo

R's tidyverse packages

focus in current draft is on R

- free, hugely flexible, large community online to help out
- offers general functionality (base R module)
- offers many packages with dedicated functionality

in particular on the `tidyverse` package (Hadley Wickham etal.), an ecosystem that includes

- `dplyr` for manipulating data frames [main focus]
- `tidyr` for tidying data [check Data Representation]
- `stringr` for dealing with texts
- `readr` for reading in data [highlighted]
- `tibble` for data representation [highlighted]
- `forcats` for dealing with factors
- `ggplot` for visualizing data [separate draft]
- `purrr` for functional programming (advanced)
- …

find convenient cheat sheets at https://rstudio.com/resources/cheatsheets/

`tidyverse` is a very good extension of base R, because it

- is much more consistent (functions and packages) → eco-system
- avoids poor historical choices, sets good defaults
- explicitly links to tidy data

set up tidyverse packages

Install (at least once) and load (once per R session) the `tidyverse` package.

```
install.packages('tidyverse')
```

The individual packages that are loaded are listed, as are their conflicts.

```
library(tidyverse)
```

Conflicts can arise when loading packages with the same function names. Resolve conflicts for example with explicitly referencing the package with `::`, eg., `stat::filter( )`.

Conflicts can be checked for tidyverse.

```
tidyverse_conflicts( )
```

```
| -- Conflicts -------------------------------------------------------------------------
| x dplyr::filter() masks stats::filter()
| x dplyr::lag()    masks stats::lag()
```

The `tidyverse` ecosystem includes
broom, cli, crayon, dbplyr, dplyr, forcats, ggplot2, haven, hms, httr, jsonlite, lubridate, magrittr, modelr, pillar, purrr, readr, readxl, reprex, rlang, rstudioapi, rvest, stringr, tibble, tidyr, xml2, tidyverse.

```
tidyverse_packages( )
```

Most of these packages should be loaded explicitly (not included in `library(tidyverse)`).

tibbles and pipes

data type: `tibble` package

- R data type for analysis is a `data.frame`, a list of equally sized vectors.
  - numeric vector (either double, integer, or complex)
  - factor (ordered, not ordered)
  - boolean vector
  - character
- a `tibble` is a `data.frame`, enhanced for convenience and consistency.
  - creating a `tibble` with the `tribble( )` function (`class( )` shows both `data.frame` and `tbl_df`)

```
(mytibble <- tribble(
  ~colA, ~colB,
  "a",   1,
  "b",   2,
  "c",   3
))
```

```
| # A tibble: 3 x 2
|   colA   colB
|   <chr> <dbl>
| 1 a         1
| 2 b         2
| 3 c         3
```

```
class(mytibble)
```

```
| [1] "tbl_df"     "tbl"        "data.frame"
```

creating a `data.frame`, it looks (not shown) slightly different.

```
mydf <- data.frame(colA=c('a','b','c'),colB=1:3)
class(mydf)
```

```
| [1] "data.frame"
```

process: `magrittr` package for pipes

- R processes data with functions, eg., `mean(mytibble$colB)`
  - reads inside-out, standard in base R and optional in tidyverse
- R data can be processed using pipes too, eg., `mytibble %>% summarize(mean(colB))`
  - reads left to right, standard in tidyverse
  - especially of interest with multiple steps, serves readability

wilfried.cools@vub.be

example: calculate the square rooted sum of squared differences between two variables.

```
x1 <- rnorm(10); x2 <- rnorm(10)
sqrt(sum((x1-x2)^2))
```

```
| [1] 2.676459
```

```
(x1-x2)^2 %>% sum( ) %>% sqrt( )
```

```
| [1] 2.676459
```

example, getting ahead of ourselves

exemplary data (part of base R): `mtcars`.

load data available in R packages with the `data( )` function, observe it's structure with the `str( )` function and observe the first 6 observations with the `head( )` function.

```
data(mtcars)
str(mtcars)
```

```
| 'data.frame': 32 obs. of  11 variables:
|  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
|  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
|  $ disp: num  160 160 108 258 360 ...
|  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
|  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
|  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
|  $ qsec: num  16.5 17 18.6 19.4 17 ...
|  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
|  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
|  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
|  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
head(mtcars)
```

```
|                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
| Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
| Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
| Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
| Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
| Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
| Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

note: calling `data( )` without arguments shows all the available data currently in reach.

a `tidyverse` look at the data, an alternative to `str( )` is `glimpse( )`.

```
glimpse(mtcars)
```

```
| Observations: 32
| Variables: 11
| $ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8, 16.4, 17.3, 15.2, 10.4
| $ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 8, 8, 8, 8, 4, 4, 4, 8, 6
| $ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 140.8, 167.6, 167.6, 275.8, 275.8
```

```
| $ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, 180, 180, 205, 215, 230, 66, 52
| $ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92, 3.07, 3.07, 3.07, 2.93
| $ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3.150, 3.440, 3.440, 4.070, 3.730
| $ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 22.90, 18.30, 18.90, 17.40, 17.60
| $ vs   <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0
| $ am   <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1
| $ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 3, 4, 5, 5, 5, 5
| $ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, 1, 1, 2, 2, 4, 2, 1, 2, 2, 4, 6
```

an example !

take the `mtcars` data,
select specific variables (mpg,cyl,hp,am) and rename one (hp turns to hpow),
select specific rows (mpg bigger than 15),
create a new variable based on existing variables (mpgr is the ratio mpg on hpow),
summarize that new variable per group formed by combining two variables (minimum of mpgr per cyl/am group), and reshape the result into a table with one row per cyl-value (4,6,8) and a column for each am value (0,1), with column variable names renamed to am0 and am1.

```
mtcars %>%
    select(mpg, cyl, hpow=hp, am) %>%
    filter(mpg > 15) %>%
    mutate(mpgr = mpg/hpow) %>%
    group_by(cyl, am) %>%
    summarize(min=min(mpgr)) %>%
    pivot_wider(names_from=am,values_from=min) %>%
    select(cyl,am0=`0`,am1=`1`)
```

```
| # A tibble: 3 x 3
| # Groups:   cyl [3]
|     cyl    am0    am1
|   <dbl>  <dbl>  <dbl>
| 1     4 0.222  0.196
| 2     6 0.145  0.113
| 3     8 0.0844 0.0598
```

dplyr to manipulate data

dplyr (package within tidyverse) is focused on

- manipulating dataframes (tibbles): subsetting, altering, summarizing, ordering, combining, reshaping
- to explore and transform (for visualization / modeling)
- applying functions to dataframes (tibbles)

main -verbs- (see example above)

- filter( ) : conditional selection of cases
- select( ) : conditional selection of variables, allows reordering and renaming
- mutate( ) : creation of new variables based on existing variables

- summarise( ) : reduce sets of values to single values

verb to structure data (see example above)

- group_by( ) : internal grouping, undo with ungroup( )
- works preceding main verbs

verbs enhanced with control on scope (advanced)

- across( ) : new way of scoping (instead of *_it, *_at, *_all)
- works for selection in mutate( ) and summarize( )

additional dplyr verbs:

- arrange( ) : ordering of cases
- sample_n( ) and sample_frac( ) : random sampling
- slice( ), transmute( ), rename( ), relocate( ), …

verbs to extend data

- bind_rows( ) and bind_cols( ) : append data of same structure
- left_, right_, inner_, full_, semi_ and anti_ join( ) : join data using indicator variable(s)

```
final comment:

only the core of dplyr is discussed, much more is possible and you will find on the Net.
```

grouping

grouping prepares data for group specific operations.

a glimpse of the data shows variables and..

- number of observations and variables
- number of groups and grouping variables

```r
tst <- mtcars %>% group_by(am,vs)
glimpse(tst)
```

```
| Observations: 32
| Variables: 11
| Groups: am, vs [4]
| $ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19.2, 17.8, 16.4, 17.3, 15.2, 10.4
| $ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, 4, 4, 8, 8, 8, 8, 4, 4, 4, 8, 6
| $ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 140.8, 167.6, 167.6, 275.8, 275.8
| $ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, 180, 180, 205, 215, 230, 66, 52
| $ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.92, 3.92, 3.07, 3.07, 3.07, 2.93
| $ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3.150, 3.440, 3.440, 4.070, 3.730
| $ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 22.90, 18.30, 18.90, 17.40, 17.60
| $ vs   <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0
```

```
| $ am   <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1
| $ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3, 3, 3, 3, 4, 5, 5, 5, 5
| $ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, 1, 1, 2, 2, 4, 2, 1, 2, 2, 4, 6
```

```
tst %>% summarize(n( ))
```

```
| # A tibble: 4 x 3
| # Groups:   am [2]
|      am    vs `n()`
|   <dbl> <dbl> <int>
| 1     0     0    12
| 2     0     1     7
| 3     1     0     6
| 4     1     1     7
```

actions on grouped data are grouped too, eg., count the number of observations (`n( )`).

remove grouping with `ungroup( )`, good practice to avoid side effects.

```
tst <- tst %>% ungroup( )
tst %>% summarize(n( ))
```

```
| # A tibble: 1 x 1
|   `n()`
|   <int>
| 1    32
```

help file shows ways for consecutive grouping with `.add` and `.drop` arguments.

transformed variable can be used for grouping, for example cutting the `mpg` in 3 groups.

```
tst <- mtcars %>% group_by(mpg3 = cut(mpg, 3))
tst %>% summarize(n( ))
```

```
| # A tibble: 3 x 2
|   mpg3        `n()`
|   <fct>       <int>
| 1 (10.4,18.2]    14
| 2 (18.2,26.1]    13
| 3 (26.1,33.9]     5
```

filter

Return rows using matching conditions.

example: the `mpg` could be set with a minimum of 30.

```
mtcars %>% filter(mpg > 30)
```

```
|                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
| Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
```

```
| Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
| Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
| Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
```

More than one condition can be considered jointly.

example: extract rows with `mpg` above 20 AND `qsec` below or equal to 18.

```
mtcars %>% filter(mpg > 20, qsec <= 18)
```

```
|               mpg cyl  disp  hp drat    wt  qsec vs am gear carb
| Mazda RX4     21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
| Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
| Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
| Lotus Europa  30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```

example: extract rows with `mpg` above 30 OR `qsec` below 20 AND `am` equal to 0.

```
mtcars %>% filter(mpg > 30 | qsec > 20, am==0)
```

```
|               mpg cyl  disp  hp drat    wt  qsec vs am gear carb
| Valiant       18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
| Merc 230      22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
| Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
```

Technically it is possible to use it on grouped data.

example: calculate group average per level of `vs` for `disp` and extract only rows with values above the 90th percentile.

```
mtcars %>% group_by(vs) %>% filter(disp >= quantile(disp,.9))
```

```
| # A tibble: 4 x 11
| # Groups:   vs [2]
|     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
|   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
| 1  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
| 2  18.1     6   225   105  2.76  3.46  20.2     1     0     3     1
| 3  10.4     8   472   205  2.93  5.25  18.0     0     0     3     4
| 4  10.4     8   460   215  3     5.42  17.8     0     0     3     4
```

example: get the first 2 rows per group of `vs`.

```
mtcars %>% group_by(vs) %>% top_n(2)
```

```
| # A tibble: 4 x 11
| # Groups:   vs [2]
|     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
|   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
| 1  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
| 2  17.8     6  168.   123  3.92  3.44  18.9     1     0     4     4
| 3  19.7     6  145    175  3.62  2.77  15.5     0     1     5     6
| 4  15       8  301    335  3.54  3.57  14.6     0     1     5     8
```

```
mtcars %>% group_by(gear) %>% distinct(cyl)
```

```
| # A tibble: 8 x 2
| # Groups:   gear [3]
|     cyl  gear
|   <dbl> <dbl>
| 1     6     4
```

```
| 2     4     4
| 3     6     3
| 4     8     3
| 5     4     3
| 6     4     5
| 7     8     5
| 8     6     5
```

exercises on filter

- the starwars dataset is part of tidyverse, load it in !
- have a glimpse at the data, what do you see ?
- filter the data to retain only characters with light skin and brown eye color.
- arrange the data according the the character's height, largest on top !
- who is smallest ?
- slice the data and keep only the 5th to 10th observation !
- slice the top 2 observations (check help on `slice_head( )`), for each gender (group your data) !
- what other functions are discussed at `?slice_head` ?
- use `slice_sample( )` to randomly select 5 observations !
- use `slice_max( )` to select 3 observations with highest values on `height` !
- repeat the above, but ignored characters with missing data for `mass` and get the top 3 for each species !

select

Extract columns (variables) by name, rename and/or reorder them.

example: the `mpg` could be selected.

```
mtcars %>% select(mpg)
```

```
|                      mpg
| Mazda RX4           21.0
| Mazda RX4 Wag       21.0
| Datsun 710          22.8
| Hornet 4 Drive      21.4
| Hornet Sportabout   18.7
| Valiant             18.1
| Duster 360          14.3
| Merc 240D           24.4
| Merc 230            22.8
| Merc 280            19.2
| Merc 280C           17.8
| Merc 450SE          16.4
| Merc 450SL          17.3
| Merc 450SLC         15.2
| Cadillac Fleetwood  10.4
| Lincoln Continental 10.4
| Chrysler Imperial   14.7
| Fiat 128            32.4
```

```
| Honda Civic         30.4
| Toyota Corolla      33.9
| Toyota Corona       21.5
| Dodge Challenger    15.5
| AMC Javelin         15.2
| Camaro Z28          13.3
| Pontiac Firebird    19.2
| Fiat X1-9           27.3
| Porsche 914-2       26.0
| Lotus Europa        30.4
| Ford Pantera L      15.8
| Ferrari Dino        19.7
| Maserati Bora       15.0
| Volvo 142E          21.4
```

```
# mtcars$mpg
```

Notice that even with one column, the result remains a dataframe (not a vector)

The dplyr way to get the base R result is by using pull( ).
example: pull out the mpg.

```
mtcars %>% pull(mpg)
```

```
|  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4 14.7 32.4 30.4 33
| [31] 15.0 21.4
```

More than one column can be considered jointly, their order is specified as such.

example: extract columns qsec and mpg (top 6 observations).

```
mtcars %>% select(qsec,mpg) %>% head( )
```

```
|                    qsec  mpg
| Mazda RX4         16.46 21.0
| Mazda RX4 Wag     17.02 21.0
| Datsun 710        18.61 22.8
| Hornet 4 Drive    19.44 21.4
| Hornet Sportabout 17.02 18.7
| Valiant           20.22 18.1
```

Columns can be extracted by their position.
example: extract third and first column.

```
mtcars %>% select(3,1)
```

```
|                      disp  mpg
| Mazda RX4           160.0 21.0
| Mazda RX4 Wag       160.0 21.0
| Datsun 710          108.0 22.8
| Hornet 4 Drive      258.0 21.4
| Hornet Sportabout   360.0 18.7
| Valiant             225.0 18.1
| Duster 360          360.0 14.3
| Merc 240D           146.7 24.4
| Merc 230            140.8 22.8
| Merc 280            167.6 19.2
| Merc 280C           167.6 17.8
| Merc 450SE          275.8 16.4
```

```
| Merc 450SL          275.8 17.3
| Merc 450SLC         275.8 15.2
| Cadillac Fleetwood  472.0 10.4
| Lincoln Continental 460.0 10.4
| Chrysler Imperial   440.0 14.7
| Fiat 128             78.7 32.4
| Honda Civic          75.7 30.4
| Toyota Corolla       71.1 33.9
| Toyota Corona       120.1 21.5
| Dodge Challenger    318.0 15.5
| AMC Javelin         304.0 15.2
| Camaro Z28          350.0 13.3
| Pontiac Firebird    400.0 19.2
| Fiat X1-9            79.0 27.3
| Porsche 914-2       120.3 26.0
| Lotus Europa         95.1 30.4
| Ford Pantera L      351.0 15.8
| Ferrari Dino        145.0 19.7
| Maserati Bora       301.0 15.0
| Volvo 142E          121.0 21.4
```

example: remove columns at third to sixth position.

```
mtcars %>% select(-c(3:6))
```

```
|                      mpg cyl  qsec vs am gear carb
| Mazda RX4           21.0   6 16.46  0  1    4    4
| Mazda RX4 Wag       21.0   6 17.02  0  1    4    4
| Datsun 710          22.8   4 18.61  1  1    4    1
| Hornet 4 Drive      21.4   6 19.44  1  0    3    1
| Hornet Sportabout   18.7   8 17.02  0  0    3    2
| Valiant             18.1   6 20.22  1  0    3    1
| Duster 360          14.3   8 15.84  0  0    3    4
| Merc 240D           24.4   4 20.00  1  0    4    2
| Merc 230            22.8   4 22.90  1  0    4    2
| Merc 280            19.2   6 18.30  1  0    4    4
| Merc 280C           17.8   6 18.90  1  0    4    4
| Merc 450SE          16.4   8 17.40  0  0    3    3
| Merc 450SL          17.3   8 17.60  0  0    3    3
| Merc 450SLC         15.2   8 18.00  0  0    3    3
| Cadillac Fleetwood  10.4   8 17.98  0  0    3    4
| Lincoln Continental 10.4   8 17.82  0  0    3    4
| Chrysler Imperial   14.7   8 17.42  0  0    3    4
| Fiat 128            32.4   4 19.47  1  1    4    1
| Honda Civic         30.4   4 18.52  1  1    4    2
| Toyota Corolla      33.9   4 19.90  1  1    4    1
| Toyota Corona       21.5   4 20.01  1  0    3    1
| Dodge Challenger    15.5   8 16.87  0  0    3    2
| AMC Javelin         15.2   8 17.30  0  0    3    2
| Camaro Z28          13.3   8 15.41  0  0    3    4
| Pontiac Firebird    19.2   8 17.05  0  0    3    2
| Fiat X1-9           27.3   4 18.90  1  1    4    1
| Porsche 914-2       26.0   4 16.70  0  1    5    2
| Lotus Europa        30.4   4 16.90  1  1    5    2
| Ford Pantera L      15.8   8 14.50  0  1    5    4
```

```
| Ferrari Dino        19.7   6 15.50  0  1    5    6
| Maserati Bora       15.0   8 14.60  0  1    5    8
| Volvo 142E          21.4   4 18.60  1  1    4    2
```

Making use of **helper functions**, selections can be more automated.

Selections can be based on partial string matching, directly with `contains( )` but also using regular expressions with `matches( )`.
example: extract columns with names that include the string `ar` (show 6).

```
mtcars %>% select(contains('ar')) %>% head( )
```

```
|                    gear carb
| Mazda RX4             4    4
| Mazda RX4 Wag         4    4
| Datsun 710            4    1
| Hornet 4 Drive        3    1
| Hornet Sportabout     3    2
| Valiant               3    1
```

example: extract columns with names that include the string `ar` but with at least one element before and after it (show 6).

```
mtcars %>% select(matches('.ar.')) %>% head( )
```

```
|                    carb
| Mazda RX4             4
| Mazda RX4 Wag         4
| Datsun 710            1
| Hornet 4 Drive        1
| Hornet Sportabout     2
| Valiant               1
```

Selection can be used to rename variables, but to avoid selection the `rename( )` function may be more interesting.

example: rename the `cyl` into `cyl468` to reflect its values, same for `vs` and `am`, and select it together with `mpg` (show 6).

```
mtcars %>% select(mpg,cyl468=cyl,vs01=vs,am01=am) %>% head( )
```

```
|                     mpg cyl468 vs01 am01
| Mazda RX4          21.0      6    0    1
| Mazda RX4 Wag      21.0      6    0    1
| Datsun 710         22.8      4    1    1
| Hornet 4 Drive     21.4      6    1    0
| Hornet Sportabout  18.7      8    0    0
| Valiant            18.1      6    1    0
```

example: rename the `cyl`, `vs` and `am` as before without selection (show 6).

```
mtcars %>% rename(cyl468=cyl,vs01=vs,am01=am) %>% head( )
```

```
|                     mpg cyl468 disp  hp drat    wt  qsec vs01 am01 gear carb
| Mazda RX4          21.0      6  160 110 3.90 2.620 16.46    0    1    4    4
| Mazda RX4 Wag      21.0      6  160 110 3.90 2.875 17.02    0    1    4    4
| Datsun 710         22.8      4  108  93 3.85 2.320 18.61    1    1    4    1
| Hornet 4 Drive     21.4      6  258 110 3.08 3.215 19.44    1    0    3    1
| Hornet Sportabout  18.7      8  360 175 3.15 3.440 17.02    0    0    3    2
| Valiant            18.1      6  225 105 2.76 3.460 20.22    1    0    3    1
```

note that a `select( )` will include the grouping variables by default.
These grouping variables can be isolated too with the `group_cols( )` function.

example: create a grouping by `vs` and `am`, and extract only those columns.

```
mtcars %>% group_by(vs,am) %>% select(group_cols( ))
```

```
| # A tibble: 32 x 2
| # Groups:   vs, am [4]
|       vs    am
|    <dbl> <dbl>
|  1     0     1
|  2     0     1
|  3     1     1
|  4     1     0
|  5     0     0
|  6     1     0
|  7     0     0
|  8     1     0
|  9     1     0
| 10     1     0
| # ... with 22 more rows
```

exercises on select

- the starwars dataset is probably still loaded into your workspace !
- select the columns hair, skin and eye color !
- use the `:` operator for consecutive columns !
- remove these columns instead of selecting them !
- select all columns with a name ending with `color` !
- use select to rename `homeworld` to `home_world` !
- do the same with the `rename( )` function !
- select only the numeric variables, use `where( )` and `is.numeric( )` !
- select only those variables with names `heigth`, `mass` and/or `size`, if present, use `any_of( )` !

mutate

Create new variables based on existing ones.

example: the new `mpg2` is the `mpg` value squared (show 6).

```
mtcars %>% mutate(mpg2=mpg^2) %>% head( )
```

```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb   mpg2
| 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4 441.00
| 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4 441.00
| 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1 519.84
| 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1 457.96
| 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2 349.69
| 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1 327.61
```

example: the `mpg` variable is overwritten with its value squared (show 6).

```
mtcars %>% mutate(mpg=mpg^2) %>% head( )
```

```
|      mpg cyl disp  hp drat    wt  qsec vs am gear carb
| 1 441.00   6  160 110 3.90 2.620 16.46  0  1    4    4
| 2 441.00   6  160 110 3.90 2.875 17.02  0  1    4    4
| 3 519.84   4  108  93 3.85 2.320 18.61  1  1    4    1
| 4 457.96   6  258 110 3.08 3.215 19.44  1  0    3    1
| 5 349.69   8  360 175 3.15 3.440 17.02  0  0    3    2
| 6 327.61   6  225 105 2.76 3.460 20.22  1  0    3    1
```

A new variable (column) can be created based on multiple existing variables.
example: the new `NEWVAR` is the `mpg` value multiplied by the vs value (show 6).

```
mtcars %>% mutate(NEWVAR=mpg*vs) %>% head( )
```

```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb NEWVAR
| 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    0.0
| 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    0.0
| 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1   22.8
| 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1   21.4
| 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    0.0
| 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1   18.1
```

A new variable can be created based on a newly created variable.
example: the new `NEWVAR` is the `mpg` value multiplied by the vs value and this new variable is divided by the `disp` variable (show 6).

```
mtcars %>% mutate(NEWVAR=mpg*vs,NEWVAR2=NEWVAR/disp) %>% head( )
```

```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb NEWVAR    NEWVAR2
| 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4    0.0 0.00000000
| 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4    0.0 0.00000000
| 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1   22.8 0.21111111
| 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1   21.4 0.08294574
| 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    0.0 0.00000000
| 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1   18.1 0.08044444
```

Newly created variables are added to the existing dataframe, but to isolate them, `transmute( )` can be helpful.

```
mtcars %>% transmute(NEWVAR=mpg*vs,NEWVAR2=NEWVAR/disp) %>% head( )
```

```
|   NEWVAR    NEWVAR2
| 1    0.0 0.00000000
| 2    0.0 0.00000000
| 3   22.8 0.21111111
| 4   21.4 0.08294574
| 5    0.0 0.00000000
| 6   18.1 0.08044444
```

Making use of **window functions**, mutations can be more automated.

example: add a column with the cumulative sum of `mpg` using `cumsum( )` (show 6).

```
mtcars %>% mutate(NEWVAR=cumsum(mpg)) %>% head( )
```

```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb NEWVAR
| 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4   21.0
| 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4   42.0
```

```
| 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1   64.8
| 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1   86.2
| 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2  104.9
| 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1  123.0
```

example: add a column with indicator whether the `mpg` is between 20 and 22 (show 6).

```
mtcars %>% mutate(NEWVAR=between(mpg,20,22)) %>% head( )
```

```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb NEWVAR
| 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4   TRUE
| 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4   TRUE
| 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1  FALSE
| 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1   TRUE
| 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2  FALSE
| 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1  FALSE
```

example: add a row number dependent on the rank of `mpg` values, with the `rownumber( )` function. When arranged by `mpg` this is more clear.

```
mtcars %>% mutate(id=row_number(mpg)) %>% head( )
```

```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb id
| 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4 19
| 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4 20
| 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1 24
| 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1 21
| 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2 15
| 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1 14
```

```
mtcars %>% mutate(id=row_number(mpg)) %>% arrange(mpg) %>% head( )
```

```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb id
| 1 10.4   8  472 205 2.93 5.250 17.98  0  0    3    4  1
| 2 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4  2
| 3 13.3   8  350 245 3.73 3.840 15.41  0  0    3    4  3
| 4 14.3   8  360 245 3.21 3.570 15.84  0  0    3    4  4
| 5 14.7   8  440 230 3.23 5.345 17.42  0  0    3    4  5
| 6 15.0   8  301 335 3.54 3.570 14.60  0  1    5    8  6
```

Grouping variables can isolate the operations.

example: create a grouping by `vs` and `am`, and mutate only those columns.

```
mtcars %>% group_by(vs,am) %>% mutate(id=row_number(mpg)) %>% head( )
```

```
| # A tibble: 6 x 12
| # Groups:   vs, am [4]
|     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb    id
|   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int>
| 1  21       6   160   110  3.9   2.62  16.5     0     1     4     4     4
| 2  21       6   160   110  3.9   2.88  17.0     0     1     4     4     5
| 3  22.8     4   108    93  3.85  2.32  18.6     1     1     4     1     2
| 4  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1     4
| 5  18.7     8   360   175  3.15  3.44  17.0     0     0     3     2    11
| 6  18.1     6   225   105  2.76  3.46  20.2     1     0     3     1     2
```

Now for each combination of `vs` and `am`, there will be a 1 (first), 2 (second)... for `id`.

exercises on mutate

- the starwars dataset is probably still loaded into your workspace !
- create a new variable `height_m` with `height` divided by 100 !
- create the same new variable, but also define BMI as `mass` / `height_m` to the power 2 !
- use transmute to repeat the above mutation but keep only `height_m` and `BMI` !
- create a new variable with the z-score of height for each species (zcore = (value-mean)/sd) !
- create that z-score per species !
- create a gender indicator that replaces the `male` and `female` labels with `m` and `f` (use `recode( )`) !
- create a gender indicator that, when sex is none changes it to the species and otherwise keeps the sex specification (use `ifelse( )`))!

summarize

Reduce sets of values into their summaries, based on grouped data.

A new variable (column) is created based on an existing one by summarizing, condensing the data.

example: the mean of all `mpg` values can be obtained.

```
mtcars %>% summarize(myAverage=mean(mpg))
```

```
|   myAverage
| 1  20.09062
```

Multiple new variables can be created.

example: the mean and standard deviation of all `mpg` values can be obtained, for multiple variables.

```
mtcars %>% summarize(myAvMpg=mean(mpg),mySdMpg=sd(mpg),myAvDisp=mean(disp),mySdDisp=sd(disp))
```

```
|    myAvMpg  mySdMpg myAvDisp mySdDisp
| 1 20.09062 6.026948 230.7219 123.9387
```

Grouping variables are very natural to `summarize( )`.
example: the mean of all `mpg` values can be obtained for each level of `vs`.

```
mtcars %>% group_by(vs) %>% summarize(myAverage=mean(mpg))
```

```
| # A tibble: 2 x 2
|      vs myAverage
|   <dbl>     <dbl>
| 1     0      16.6
| 2     1      24.6
```

example: the mean and standard deviation of all `mpg` values can be obtained, for multiple variables, for multiple combinations of grouping, `vs` and `am`.

```
mtcars %>% group_by(vs,am) %>% summarize(myAvMpg=mean(mpg),mySdMpg=sd(mpg),myAvDisp=mean(disp),mySdDisp=
```

```
| # A tibble: 4 x 6
| # Groups:   vs [2]
|      vs    am myAvMpg mySdMpg myAvDisp mySdDisp
|   <dbl> <dbl>   <dbl>   <dbl>    <dbl>    <dbl>
| 1     0     0    15.0    2.77     358.     71.8
| 2     0     1    19.8    4.01     206.     95.2
```

```
| 3     1     0    20.7    2.47    175.       49.1
| 4     1     1    28.4    4.76     89.8      18.8
```

example: the total number of observations within a group, eg., `vs`, can be obtained with `n( )`, or using the special verb `count( )`.

```
mtcars %>% group_by(vs) %>% count( )
```

```
| # A tibble: 2 x 2
| # Groups:   vs [2]
|       vs     n
|    <dbl> <int>
| 1     0    18
| 2     1    14
```

```
mtcars %>% group_by(vs) %>% summarize(mycount=n( ))
```

```
| # A tibble: 2 x 2
|       vs mycount
|    <dbl>   <int>
| 1     0      18
| 2     1      14
```

Making use of **summary functions**, summarizing can be more automated.

example: the number of distinct values in a vector for each combination `vs` and `am` can be obtained with `n_distinct( )`, and the third number of each group with `nth( )`.

```
mtcars %>% group_by(vs,am) %>% summarize(nrDist=n_distinct(mpg),`3th`=nth(mpg,3))
```

```
| # A tibble: 4 x 4
| # Groups:   vs [2]
|       vs    am nrDist `3th`
|    <dbl> <dbl>  <int> <dbl>
| 1     0     0     10  16.4
| 2     0     1      5  26
| 3     1     0      7  24.4
| 4     1     1      6  30.4
```

exercises on summarize

- the starwars dataset is probably still loaded into your workspace !
- summarize the `height` into the average height (some missing values need to be dealt with) !
- repeat to above, but group by `species` and `sex`, and include the average `mass` !

scoping a verb

The `across( )` function allows for selection of variables within the `summarize( )` or `mutate( )` function. It will replace the earlier functions *_at, *_if and *_all. To select the variables, they can be explicitly named or extracted with dedicated functions.

example, the structure is asked for after transforming a selected set of variables from numeric into factors.

example part 1, after turning the `am` and `vs` variable into a factor.

```
mtcars %>% select(mpg,cyl,am,vs) %>% mutate(across(c('am','vs'),factor)) %>% str( )
```

```
| 'data.frame': 32 obs. of  4 variables:
|  $ mpg: num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
|  $ cyl: num  6 6 4 6 8 6 8 4 4 6 ...
|  $ am : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
|  $ vs : Factor w/ 2 levels "0","1": 1 1 2 2 1 2 1 2 2 2 ...
```

example part 2, after turning the consecutive variables `cyl`, `am` and `vs` into a factor with a `:` operator.

```
mtcars %>% select(mpg,cyl,am,vs) %>% mutate(across(cyl:vs,factor)) %>% str( )
```

```
| 'data.frame': 32 obs. of  4 variables:
|  $ mpg: num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
|  $ cyl: Factor w/ 3 levels "4","6","8": 2 2 1 2 3 2 3 1 1 2 ...
|  $ am : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
|  $ vs : Factor w/ 2 levels "0","1": 1 1 2 2 1 2 1 2 2 2 ...
```

example part 3, after turning the variables with names that contain `ar` into a factor.

```
mtcars %>% select(mpg,cyl,gear,carb) %>% mutate(across(contains("ar"),factor)) %>% str( )
```

```
| 'data.frame': 32 obs. of  4 variables:
|  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
|  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
|  $ gear: Factor w/ 3 levels "3","4","5": 2 2 2 1 1 1 1 2 2 2 ...
|  $ carb: Factor w/ 6 levels "1","2","3","4",..: 4 4 1 1 2 1 4 2 2 4 ...
```

example, multiple values can be obtained by specifying a list of functions, for example a median, mean and sd for the first and third variable (show only first 6).

```
descr <- list(
  md = ~median(.x, na.rm = TRUE),
  av = ~mean(.x, na.rm = TRUE),
  sd = ~sd(.x, na.rm = TRUE)
)
mtcars %>% mutate(across(c(1,3), descr)) %>% head( )
```

```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb mpg_md   mpg_av   mpg_sd disp_md  disp_av  disp_
| 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4   19.2 20.09062 6.026948   196.3 230.7219 123.93
| 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4   19.2 20.09062 6.026948   196.3 230.7219 123.93
| 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1   19.2 20.09062 6.026948   196.3 230.7219 123.93
| 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1   19.2 20.09062 6.026948   196.3 230.7219 123.93
| 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2   19.2 20.09062 6.026948   196.3 230.7219 123.93
| 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1   19.2 20.09062 6.026948   196.3 230.7219 123.93
```

Various functions exist, as part of the `tidyselect` package, eg., `all_of( )`, `where( )`, `matches( )`, `starts_with( )`, and more.
While the examples use `mutate( )`, they are also possible with `summarize( )`.

exercises on across

- the starwars dataset is probably still loaded into your workspace !
- summarize the numeric variables into their minimum and maximum (some missing values need to be dealt with) !

join

Different datafiles can be combined into one datafile using common variables that serve as key (cfr. relational databases). Methods differ primarily in how they deal with mismatches in key variable values.

example: assume a cylinder specific datafile, `mtcyl`, with a 2 cylinder but no 8 cylinder unlike the `mtcars` (4,6,8).

```r
mtcyl <- tribble(
~cyl,~type,
2,'small',
4,'medium',
6,'large'
)
```

example: combine the `mtcars` and `mtcyl` but ignore the irrelevant `cyl` equal to 2 (not part of `mtcars`), with a `left_join( )`.

````
```r
mtcars %>% left_join(mtcyl) %>% head( )
```
````

````
```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb   type
| 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4  large
| 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4  large
| 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1 medium
| 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1  large
| 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2   <NA>
| 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1  large
```
````

Notice that `cyl` equal to 8 turns out missing, because it is not specified in the -right- datafile.

example: combine the `mtcars` and `mtcyl` but ignore the `cyl` equal to 8 because it lacks information on type, with a `right_join( )`.

````
```r
mtcars %>% right_join(mtcyl) %>% head( )
```
````

````
```
|    mpg cyl  disp  hp drat    wt  qsec vs am gear carb   type
| 1 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4  large
| 2 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4  large
| 3 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1 medium
| 4 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1  large
| 5 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1  large
| 6 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2 medium
```
````

Notice that `cyl` equal to 2 is included, but turns out missing for most variables because it is not specified in the -left- datafile.

example: combine the `mtcars` and `mtcyl` for only those observations with the linking variable `cyl` in both files, with an `right_join( )`.

```
mtcars %>% inner_join(mtcyl) %>% head( )
```

```
|    mpg cyl  disp  hp drat    wt  qsec vs am gear carb   type
| 1 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4  large
| 2 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4  large
| 3 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1 medium
| 4 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1  large
| 5 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1  large
| 6 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2 medium
```

Notice no missing values, but some data is not included.

example: combine the `mtcars` and `mtcyl` keeping all available information, with a `full_join( )` showing selected rows 1 to 3, 5, 7 and 33.

```
mtcars %>% full_join(mtcyl) %>% slice(c(1:3,5,7,33))
```

```
|    mpg cyl disp  hp drat    wt  qsec vs am gear carb   type
| 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4  large
| 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4  large
| 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1 medium
| 4 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2   <NA>
| 5 14.3   8  360 245 3.21 3.570 15.84  0  0    3    4   <NA>
| 6   NA   2   NA  NA   NA    NA    NA NA NA   NA   NA  small
```

Other types of join exist, like `semi_join( )`, `nest_join( )`, `anti_join()`, which are described in the helpfile.

exercises on join

- the two mini tibbles `band_members` and `band_instruments` are probably loaded into your workspace as part of the tidyverse !
- combined the two, left/right/inner/full !
- try out the same with `semi_join( )` and `anti_join( )` and interpret what happens.

dplyr exercises

Compare the structure of the mtcars data with a glimpse at that data.

Compare a select of `mpg` with a pull of `mpg`.

Check the help file and pull out the second before last column.

Select all columns except the `am`.

Select all columns except the `am` and `vs`.

Keep only columns `mpg`, `cyl` and `disp`, but rename `mpg` to `miles_gallon`.

Insist, keep only columns `mpg`, `cyl` and `disp`, but rename `mpg` to `miles per gallon`.

Keep only the consecutive columns in between `disp` and `wt`, in addition to `mpg` as a last column, use a `:`.

Create a variable for the row names.

Change the `mpg` (miles per gallon) into `kpl` (kilometers per liter) with 1 mpg is 0.425 km/l, using `mutate( )`.

Select about 10% of the observations, twice, check the help file on using `sample_frac( )`.

note that the matrix way could be:

Select the 10th to 15th row, check the help file on using `slice( )`.

Select the distinct combinations only, for variables `am` and `vs`.

Check the help files to determine how to keep all variables (for each first observation of that combination).

Filter the data to retain only cases with mpg > 20 and hp above or equal to 110.

Filter the data to retain only the Datsun 710.

example, getting ahead of ourselves again

read in data delim pivot and separate/unite

exemplary data read into an R-workspace: `repeated.txt`, a tab-delimited text file, by copy-pasting.

```
myrepeated <- read_delim(clipboard(),delim='\t')
```

```
| # A tibble: 3 x 7
|   id    `t1 score` `t1 posit` `t2 score` `t2 posit` `t3 score` `t3 posit`
|   <chr>      <dbl> <chr>           <dbl> <chr>           <dbl> <chr>
| 1 id1            1 x                  NA y                   4 x
| 2 id2            2 y                   3 x                  NA <NA>
| 3 id3            1 x                   2 y                   5 x
```

To get the data tidy, the different time points should not be at different columns. (See draft on Data Representation)

take the id and scores at time points and pivot it to have all scores at their own designated row, with times named `type` take the id and positions at time points and pivot it to have all positions at their own designated row too, with times named `type` separate the time part from the type part in the type variable for both scores and positions and name the time part `time` merge both datasets with a join like before, using id as key, but first eliminate at least one of the inconsistent `type` variables select only the relevant variables, id, time, score and posit, and remove all observations with a missing value for either score or posit.

```
| # A tibble: 9 x 3
|   id    type     score
|   <chr> <chr>    <dbl>
| 1 id1   t1 score     1
| 2 id1   t2 score    NA
| 3 id1   t3 score     4
| 4 id2   t1 score     2
| 5 id2   t2 score     3
| 6 id2   t3 score    NA
| 7 id3   t1 score     1
| 8 id3   t2 score     2
| 9 id3   t3 score     5
```

```
| # A tibble: 9 x 3
|   id    type     posit
|   <chr> <chr>    <chr>
| 1 id1   t1 posit x
| 2 id1   t2 posit y
```

```
| 3 id1   t3 posit x
| 4 id2   t1 posit y
| 5 id2   t2 posit x
| 6 id2   t3 posit <NA>
| 7 id3   t1 posit x
| 8 id3   t2 posit y
| 9 id3   t3 posit x
```

```
| # A tibble: 9 x 4
|   id    time  type  score
|   <chr> <chr> <chr> <dbl>
| 1 id1   t1    score     1
| 2 id1   t2    score    NA
| 3 id1   t3    score     4
| 4 id2   t1    score     2
| 5 id2   t2    score     3
| 6 id2   t3    score    NA
| 7 id3   t1    score     1
| 8 id3   t2    score     2
| 9 id3   t3    score     5
```

```
| # A tibble: 9 x 4
|   id    time  type  posit
|   <chr> <chr> <chr> <chr>
| 1 id1   t1    posit x
| 2 id1   t2    posit y
| 3 id1   t3    posit x
| 4 id2   t1    posit y
| 5 id2   t2    posit x
| 6 id2   t3    posit <NA>
| 7 id3   t1    posit x
| 8 id3   t2    posit y
| 9 id3   t3    posit x
```

```
| # A tibble: 9 x 5
|   id    time  score type  posit
|   <chr> <chr> <dbl> <chr> <chr>
| 1 id1   t1        1 posit x
| 2 id1   t2       NA posit y
| 3 id1   t3        4 posit x
| 4 id2   t1        2 posit y
| 5 id2   t2        3 posit x
| 6 id2   t3       NA posit <NA>
| 7 id3   t1        1 posit x
| 8 id3   t2        2 posit y
| 9 id3   t3        5 posit x
```

```
| # A tibble: 7 x 4
|   id    time  score posit
|   <chr> <chr> <dbl> <chr>
| 1 id1   t1        1 x
| 2 id1   t3        4 x
| 3 id2   t1        2 y
| 4 id2   t2        3 x
| 5 id3   t1        1 x
| 6 id3   t2        2 y
```

```
| 7 id3   t3        5 x
```

```
scores <- myrepeated %>% select(id,`t1 score`,`t2 score`,`t3 score`) %>%
    pivot_longer(-id,names_to='type',values_to='score') %>%
    separate(type,c('time','type')))
positions <- myrepeated %>% select(id,`t1 posit`,`t2 posit`,`t3 posit`) %>%
    pivot_longer(-id,names_to='type',values_to='posit') %>%
    separate(type,c('time','type')))
longform <- scores %>%
    select(-type) %>%
    full_join(positions)) %>%
    select(-type) %>%
    filter(!is.na(score),!is.na(posit)))
```

It is possible to switch back to a wider data representation, for example to calculate correlations. Maybe fill in the missing values NA as 0 values.

```
| # A tibble: 3 x 6
|   id     t1_x  t3_x  t1_y  t2_x  t2_y
|   <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
| 1 id1       1     4    NA    NA    NA
| 2 id2      NA    NA     2     3    NA
| 3 id3       1     5    NA    NA     2
```

```
longform %>% pivot_wider(values_from=score,names_from=c(time,posit))
# longform %>% pivot_wider(values_from=score,names_from=c(time,posit),values_fill=list(score=0))
```

friends of dplyr

dplyr is the main package for data transformations

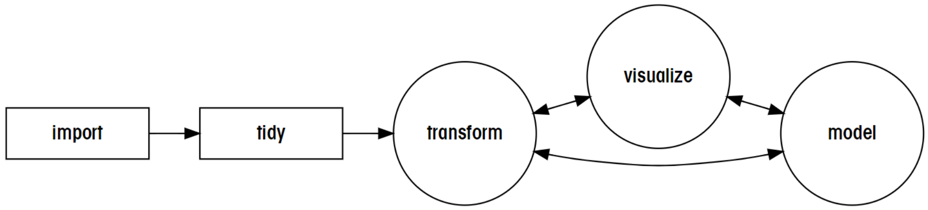

Figure 2: workflow: tidyverse lingo

in preparation of data manipulation

- the data has to be brought into the R workspace
- the data has to be tidy for efficient further processing

after the transformation, the data should be ready for

- modeling
- visualization [see ggplot in Data Visualization]

tidyr to tidy data

tidyr (package within tidyverse) is focused on

- tidying dataframes (tibbles): pivoting data into longer or wider form
- to explore and transform (for visualization / modeling)
- creating pure variables

main -verbs- (see example above)

- pivot_wider( ) and pivot_longer( ): turn multiple columns or rows into one, making datafiles longer or wider
- separate( ) and extract( ): create multiple columns from one column using delimiters or regular expressions

pivot

turning long form data into wide form and vise verse

- each research unit is assigned to a row, in a tidy dataframe (tibble)
- the research unit in focus can change throughout an analysis (eg., test score → student)
- both univariate and multivariate data representations can be required for data analysis
- multivariate data representation most intuitive, univariate most flexible

example: for the `iris` dataset, with 4 values for each unit within each species, showing only the first 6 observations.

```
long_iris <- iris %>% pivot_longer(-Species,names_to='type',values_to='score')
long_iris %>% head( )
```

```
| # A tibble: 6 x 3
|   Species type         score
|   <fct>   <chr>        <dbl>
| 1 setosa  Sepal.Length   5.1
| 2 setosa  Sepal.Width    3.5
| 3 setosa  Petal.Length   1.4
| 4 setosa  Petal.Width    0.2
| 5 setosa  Sepal.Length   4.9
| 6 setosa  Sepal.Width    3
```

example: for the long form iris dataset, the univariate case, it can not be switched into a wider form.

```
long_iris %>% pivot_wider(values_from=score,names_from=type)
```

there should be a unique combination for rows x columns, the new dataset does not link values from an individual unit (row).

example: redo the pivoting from wide to long, after adding an indicator variable for each unit.

```
long_iris <- iris %>% mutate(id=1:n()) %>% pivot_longer(-c(Species,id),names_to='type',values_to='score
long_iris %>% head( )
```

```
| # A tibble: 6 x 4
|   Species    id type         score
|   <fct>   <int> <chr>        <dbl>
| 1 setosa      1 Sepal.Length   5.1
| 2 setosa      1 Sepal.Width    3.5
| 3 setosa      1 Petal.Length   1.4
| 4 setosa      1 Petal.Width    0.2
| 5 setosa      2 Sepal.Length   4.9
| 6 setosa      2 Sepal.Width    3
```

Also in long form, rows are combined into clusters of scores related to the same unit.
example: the pivoting can now be done from long to wide (using id information to assign scores to the appropriate row)

```
wide_iris <- long_iris %>% pivot_wider(values_from=score,names_from=type)
wide_iris
```

```
| # A tibble: 150 x 6
|    Species    id Sepal.Length Sepal.Width Petal.Length Petal.Width
|    <fct>   <int>        <dbl>       <dbl>        <dbl>       <dbl>
|  1 setosa      1          5.1         3.5          1.4         0.2
|  2 setosa      2          4.9         3            1.4         0.2
|  3 setosa      3          4.7         3.2          1.3         0.2
|  4 setosa      4          4.6         3.1          1.5         0.2
|  5 setosa      5          5           3.6          1.4         0.2
|  6 setosa      6          5.4         3.9          1.7         0.4
|  7 setosa      7          4.6         3.4          1.4         0.3
|  8 setosa      8          5           3.4          1.5         0.2
|  9 setosa      9          4.4         2.9          1.4         0.2
| 10 setosa     10          4.9         3.1          1.5         0.1
| # ... with 140 more rows
```

Note in wider format, cluster information is implied by the row, in longer format it is made explicit with indicator variables. The original wide and long distinction is abolished because it is recognized that there are various shades in between, hence wide-r and long-er.

exercises on pivot

- the `world_bank_pop` dataset that is part of the tidyr package
- pivot the dataset to have univariate data for the scores over the different years
- the `us_rent_income` dataset is also part of the tidyr package
- pivot the dataset to have a multivariate version with variables for all estimate x moe combinations
- first verify what happens when constructing a multivariate version only for estimate

separate / unite

splitting up information within a variable, or combining over variables

- each variable should consist of one type of information, in a tidy dataframe (tibble)
- variables that combine information should often be split

- variables that provide no additional information should be removed, sometimes united

example: the long form iris data shows a type that consists of both Petal/Sepal and Length/Width, the can be separated.

```
long_iris_x <- long_iris %>% separate(type,c('PT','lw'))
long_iris_x %>% head( )
```

```
| # A tibble: 6 x 5
|   Species    id PT    lw     score
|   <fct>   <int> <chr> <chr>  <dbl>
| 1 setosa      1 Sepal Length   5.1
| 2 setosa      1 Sepal Width    3.5
| 3 setosa      1 Petal Length   1.4
| 4 setosa      1 Petal Width    0.2
| 5 setosa      2 Sepal Length   4.9
| 6 setosa      2 Sepal Width    3
```

example: the separated columns can be combined, using a separator dash in this case (default is underscore).

```
unite_iris <- long_iris_x %>% unite('myType',PT:lw,sep='-')
unite_iris
```

```
| # A tibble: 600 x 4
|    Species    id myType       score
|    <fct>   <int> <chr>        <dbl>
|  1 setosa      1 Sepal-Length   5.1
|  2 setosa      1 Sepal-Width    3.5
|  3 setosa      1 Petal-Length   1.4
|  4 setosa      1 Petal-Width    0.2
|  5 setosa      2 Sepal-Length   4.9
|  6 setosa      2 Sepal-Width    3
|  7 setosa      2 Petal-Length   1.4
|  8 setosa      2 Petal-Width    0.2
|  9 setosa      3 Sepal-Length   4.7
| 10 setosa      3 Sepal-Width    3.2
| # ... with 590 more rows
```

The tidyr package includes some other functions that can be of interest when getting more involved into programming and simulations studies, like `expand( )`, `crossover( )`, `nesting( )`. Best check the helpfile.

exercises on separate / unite

- the `mtcars` should still be loaded into your workspace
- turn the rownames to a variabla called type using the `rownames_to_column( )` function, it consists of car type information, car subtype and subtype specification.
- separate the type into three pieces (if there are less, make third and if necessary the second a missing value)
- unite the second and third part under the name subtype.

import data with readr, readxl or haven

when using your own data, they have to be imported into the workspace.

- data that are saved as R objects in a workspace (`*.RData`) can be loaded with the `load( )` function
- various packages and the R base package offer functions for various types of data (excel, spss, sas, …)
- in tidyverse the `readr` package deals with most common data, `readxl` is dedicated to notorious excel, and `haven` addresses the link with the main statistical software SAS, spss and Stata.

readr

A common function that is of interest is the `read_delim( )` function from the `readr` package. A file in the working directory can be specified jointly with the delimiter, it can be obtained from the clipboard of searched through.

```
myrepeated <- read_delim(file='repeated.txt',delim='\t') # if repeated.txt is in the working directory
myrepeated <- read_delim(clipboard(),delim='\t')
myrepeated <- read_delim(file.choose(),delim='\t')
```

The `file.choose( )` and `clipboard( )` can be used with other functions as well, like the ones discussed next.

Consult with `?read_delim( )` on how to set different parameters and gain flexibility to read in data.

readxl

Dedicated to Excel, the `read_excel( )` function facilitates reading Excel files. Alternatively, save Excel files to tab-delimited or comma separated value files to read with `read_delim( )`.

Example, read in the `RealData_clean.xlsx` file making use of the defaults.

```
read_excel('RealData_clean.xlsx')
```

```
| # A tibble: 280 x 21
|    Pte   Dx        P     A `Location opera~ `Size mm (FMT,V~ `FMT (mm)` `MSD (mm)` `RCOP (mm)` `Measu
|    <chr> <chr> <dbl> <dbl>            <dbl> <chr>                 <dbl>      <dbl>       <dbl> <chr>
|  1 1     TB        3     1               NA VZ 24 (vierling)         NA         24          NA Y
|  2 2     TB        1    NA               NA FMT 13                   13         NA          NA Y/N
|  3 3     FLIN      1    NA               NA VZ 23                    NA         23          NA Y
|  4 4     TB        1    NA               NA FMT 12                   12         NA          NA Y
|  5 5     TB        2    NA               NA FMT 8                     8         NA          NA N
|  6 6     TB        3     2               NA FMT 6                     6         NA          NA N
|  7 7     FLIN      1     1               NA FMT 2                     2         NA          NA Y
|  8 8     TB        2    NA                1 FMT 8                     8         NA          NA Y
|  9 9     QRT       1     1               NA QRT 10                   NA         NA          10 Y
| 10 10    TB        4    NA               NA VZ 45                    NA         45          NA Y
| # ... with 270 more rows, and 8 more variables: `Delay Dx-evac` <chr>, `evac <20d` <dbl>, `evac > 20d
| #   Pil <chr>, Age <dbl>
```

Consult with ?read_excel on alternative parameter values to extract specific columns, specifics Excel-tabs, …

haven

Haven is dedicated to the major statistical software packages, SPSS, SAS and STATA.

An SPSS version of the iris data is part of the tidyverse system, located in the examples folder of the haven package.
The data is simply read, using default parameters, as `read_sav`.

```
pathToIrisSpssData <- system.file("examples", "iris.sav", package = "haven")
read_sav(pathToIrisSpssData)
```

```
| # A tibble: 150 x 5
|    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
|           <dbl>       <dbl>        <dbl>       <dbl>  <dbl+lbl>
|  1          5.1         3.5          1.4         0.2 1 [setosa]
|  2          4.9         3            1.4         0.2 1 [setosa]
|  3          4.7         3.2          1.3         0.2 1 [setosa]
|  4          4.6         3.1          1.5         0.2 1 [setosa]
|  5          5           3.6          1.4         0.2 1 [setosa]
|  6          5.4         3.9          1.7         0.4 1 [setosa]
|  7          4.6         3.4          1.4         0.3 1 [setosa]
|  8          5           3.4          1.5         0.2 1 [setosa]
|  9          4.4         2.9          1.4         0.2 1 [setosa]
| 10          4.9         3.1          1.5         0.1 1 [setosa]
| # ... with 140 more rows
```

A SAS version of the iris data is simply read with `read_sas`. The dataset has a `sas7dat` extension.

```
path <- system.file("examples", "iris.sas7bdat", package = "haven")
read_sas(path)
```

```
| # A tibble: 150 x 5
|    Sepal_Length Sepal_Width Petal_Length Petal_Width Species
|           <dbl>       <dbl>        <dbl>       <dbl> <chr>
|  1          5.1         3.5          1.4         0.2 setosa
|  2          4.9         3            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           3.6          1.4         0.2 setosa
|  6          5.4         3.9          1.7         0.4 setosa
|  7          4.6         3.4          1.4         0.3 setosa
|  8          5           3.4          1.5         0.2 setosa
|  9          4.4         2.9          1.4         0.2 setosa
| 10          4.9         3.1          1.5         0.1 setosa
| # ... with 140 more rows
```

A Stata version of the iris data is read in with `read_dta( )` or `read_stata( )`. The dataset has a `dta` extension.

```
path <- system.file("examples", "iris.dta", package = "haven")
read_dta(path)
```

```
| # A tibble: 150 x 5
|    sepallength sepalwidth petallength petalwidth species
|          <dbl>      <dbl>       <dbl>      <dbl> <chr>
```

```
|  1         5.10        3.5          1.40       0.200 setosa
|  2         4.90        3            1.40       0.200 setosa
|  3         4.70        3.20         1.30       0.200 setosa
|  4         4.60        3.10         1.5        0.200 setosa
|  5         5           3.60         1.40       0.200 setosa
|  6         5.40        3.90         1.70       0.400 setosa
|  7         4.60        3.40         1.40       0.300 setosa
|  8         5           3.40         1.5        0.200 setosa
|  9         4.40        2.90         1.40       0.200 setosa
| 10         4.90        3.10         1.5        0.100 setosa
| # ... with 140 more rows
```

To write any of the files, use the `write_` prefix, for `dta`, `sas` and `sav`. example, write the mtcars into sas format.

```
write_sas(mtcars,'mytryinSAS.sas7bdat')
```

final summary on Data Manipulation

Current draft provides a primer on data manipulation, tidy data and the importing of data, which are the main steps in preparation of most real data analyses and visualizations. It is strongly advised to play with the techniques discussed above to get some proficiency in using it, as it would add significantly to the flexibility of whatever you want to further do with your data.

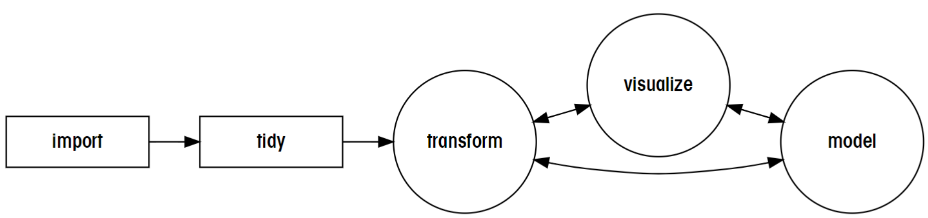

Figure 3: workflow: tidyverse lingo

A different draft addresses what is tidy data, with a focus on how data should be registered. A next draft will address how to visualize data, using the `ggplot( )` function.

Several tidyverse packages are not yet discussed, which does suggest they are not useful but they are more specific. The consistency within tidyverse should give you a push though, to study the other packages yourself when of interest.

Base R still is a proper alternative to the tidyverse package, so be aware that others may do things differently.