DATA (PRE-)PROCESSING WITH `TIDYVERSE`
LECTURE: UNSUPERVISED LEARNING AND EVOLUTIONARY COMPUTATION USING R

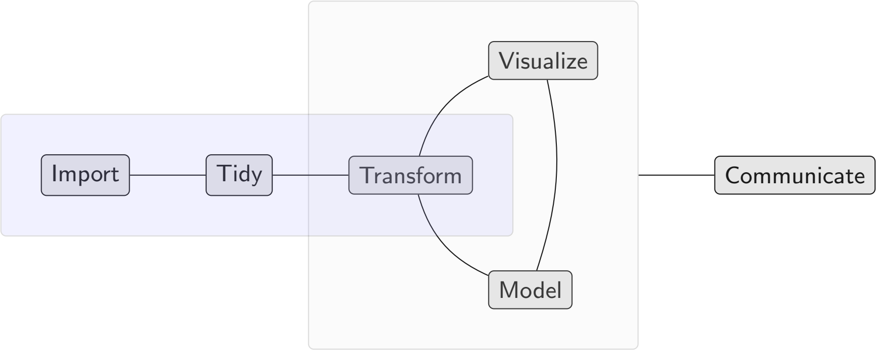
Jakob Bossek

MALEO Group, Department of Computer Science, Paderborn University, Germany

28th Oct, 2024

Model of data science

Model of data science according to (Wickham and Grolemund 2017):



Today's motivation

- We learned already how to work with rectangular data in R: dataframes, basic import, subsetting, aggregating etc.
- These are powerful tools, yet do not follow a common interface.
- Large data analysis project may become "hard to read".[1]
- Core R functions cannot be changed easily.[2]
- **Solution:** provide new language concepts through R packages.

[1] Readability of code is in fact a key requirement in programming. Readable code helps to avoid mistakes, to fix issues, to dive into code written some time ago, to dive into extraneous code etc.

[2] This would break a lot of existing code.

Core tidyverse

tidyverse is a collection aka "universe" of packages[3] that are nowadays indispensible in modern data analysis.[4]

Core packages

- **tibble** Essentially a nicer dataframe (today)
- **readr** Fast import of rectangular data (today)
- **dplyr** Data manipulation (today)
- **tidyr** Collection of utilities to tidy[5] data (today)
- **ggplot2** Excellent visualization framework (next week)
- **purrr** Functional programming toolkit
- **stringr** Cohesive utilities for working with strings (i.e. characters)
- **forcats** Utilities to work with factors

3 Actually, ≥ 83 packages in total and 8 core packages.

4 Once you start using tidyverse base R will become increasingly cumbersome.

5 I.e. prepare and clean-up (one of the tedious tasks in data analysis).

`tidyverse` core developer: Hadley Wickham

- Born on October 14, 1979 in Hamilton, New Zealand
- Chief Scientist at RStudio
- Adjunct Professor of Statistics at University of Auckland, Stanford University, Rice University
- The brain behind numerous packages for data science, data import and R software engineering
- Author of many data-science books (Wickham 2009; Wickham 2014a; Wickham 2015; Wickham and Grolemund 2017)
- Visit Hadley's website for more information

`tidyverse`: design principles

High level: "... language for solving data science challenges with R code ..."[6]

- Tools for most common problems data scientists usually struggle with in everyday life.
- Human centered (with respect to readability, effectiveness etc.)
- Common "grammer" such that being familiar with package *A* makes it easier to learn another package *B* from the collection.
- Read the thoughts of the (many) authors online.[7]
- `tidyverse` is also a incredibly active community of people.[8]

---

6 `https://tidyverse.tidyverse.org/articles/paper.html`

7 `https://design.tidyverse.org/`

8 Vast majority of R-related questions on Stack Overflow deal with `tidyverse` packages.

Installation

`tidyverse` is just a wrapper package that contains many others:

```
> install.packages("tidyverse", dependencies = TRUE)
> library(tidyverse)
```

Dataframe vs. tibble

Printing a dataframe in R sucks:[9]

```
> data(mtcars)
> cbind(mtcars[1:3, ], mtcars[1:3, ])
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb  mpg cyl disp
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4 21.0   6  160
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4 21.0   6  160
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1 22.8   4  108
##                hp drat    wt  qsec vs am gear carb
## Mazda RX4     110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710     93 3.85 2.320 18.61  1  1    4    1
```

9 If the dataframe has many rows and columns the output is a mess!

Dataframe vs. tibble

- A *tibble* is a modern dataframe with way nicer output (in particular for large tables).
- Tweaks: avoids bad properties of data frames and adds some nice ones.

```
> mtcars = as_tibble(mtcars)
> mtcars
## # A tibble: 32 x 11
##      mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
##  2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
##  3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
##  4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
##  5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
##  6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
##  7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
##  8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
##  9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
## 10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
## # i 22 more rows
```

Dataframe vs. tibble

A *tibble* indeed IS a data frame:

```
> mtcars = as_tibble(mtcars)
>
> class(mtcars) # actually a tibble IS a data frame
## [1] "tbl_df"     "tbl"        "data.frame"

> class(mtcars) = "data.frame" # drop additional classes
> head(mtcars)
##    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## 4 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## 5 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## 6 18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Data import with `readr`

We already know base R's `read.table(...)`, `read.csv(...)` for import. `readr` offers a collection of very fast reimplementations of `read_*` functions:[10]

```
> data(mtcars)
> write.table(mtcars, "mtcars.csv", row.names = FALSE, sep = ";", dec = ",")
>
> # most general version
> mtcars = readr::read_delim("mtcars.csv", delim = ";")
>
> # by default , as decimal separator and ; as field separator
> mtcars = readr::read_csv2("mtcars.csv")
```

```
> mtcars[1:3, ]
##    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## 3 22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
```

10 Up to 2-10 times faster.

Exercises

1. Download the CIFAR 10 dataset[11] (220 MB).

2. Import the data set using (a) base R's functions and (b) `readr`'s alternatives and measure the time it takes.
   Hint:

```
> system.time({ # measure time it takes to evaluate an expression
+   # my code goes here, e.g.
+   eigen(matrix(runif(1000000), ncol = 1000))
+ })
```

3. Convince yourself that a tibble is actually a data frame by toying around with subsetting, selecting rows/columns using base R commands etc.

[11] `https://www.openml.org/data/get_csv/16797612/cifar-10-small.csv`

Sample solutions

Ad 2) Import the data set using (a) base R's functions and (b) `readr`'s alternatives and measure the time it takes.

```
>   file = "data/cifar-10-small.csv"
>   system.time({readr::read_delim(file, delim = ",", show_col_types = FALSE)})

## Error:  'data/cifar-10-small.csv' does not exist in current working directory
('/Users/jboss/science/teaching/UPB/WT2024/Unsupervised Learning and Evolutionary Computation
Using R/slides/presentation slides/ULEOR-04-tidyverse').
## Timing stopped at:  0.055 0.012 0.22

>   system.time({read.table(file, header = TRUE, sep = ",")})

## Warning in file(file, "rt"):  cannot open file 'data/cifar-10-small.csv':  No such file or
directory
## Error in file(file, "rt"):  cannot open the connection
## Timing stopped at:  0 0 0.001
```

Data manipulation with `dplyr`

- Data is rarely in a format that is suitable for visualization or modelling.
- Transformation of data in most cases necessary:
  - Join multiple data frames from multiple sources
  - Aggregate data (e.g., calculate summary statistics)
  - Rename, subset/filter, re-order, sort, transform variable types etc.
- We already know some of R's data frame manipulation function, e.g., `within`, `split`, `aggregate`, ...
- Now we will learn about the data manipulation package `dplyr`

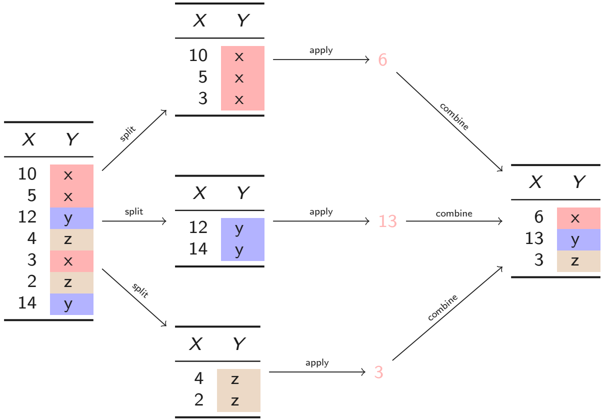
Data manipulation with `dplyr`

- `dplyr` defines a grammar/language of data manipulation which can essentially be broken down to six "verbs" and their variations:
  - Subset observations with `filter()`
  - Split in sub-datasets with `group_by()`
  - Pick variables (columns) by names with `select()`
  - Create new variables with `mutate()`
  - Aggregate data with `summarize()`
  - Reorder rows with `arrange()`
- All `dplyr` function share a similar interface:
  1. first argument is the input dataframe/tibble,
  2. subsequent arguments give control over details and
  3. the output is a tibble.
- Chaining is perfectly possible![12]

[12] Chaining is the process of passing the result of one function directly to another.

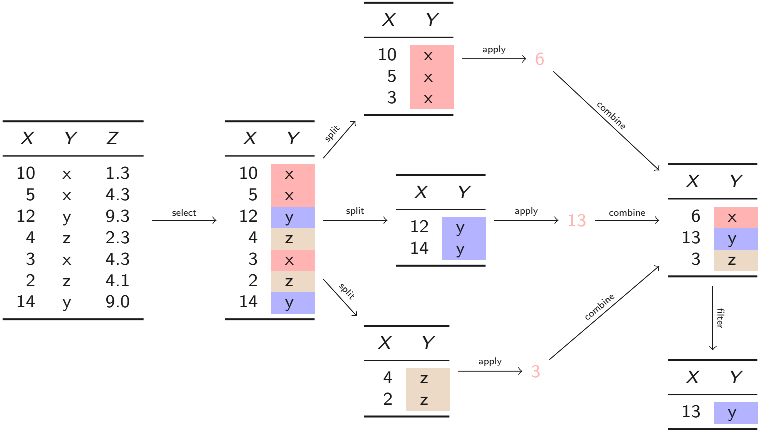
Split-apply-combine paradigm

Schema of split-apply-combine workflow:



Split-apply-combine paradigm

Can be more complex (more layers) and contain pre- and post-processing



The `diamonds` data set

In the following we will work with a subset of `diamonds` dataset which ships with the `ggplot2` package:

```
> library(ggplot2) # install.packages("ggplot2") if not installed
> data(diamonds)
> diamonds
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # i 53,930 more rows
```

Subset with `filter()`

Extract useful information $\rightsquigarrow$ drop observations and keep the interesting ones.
E.g. get all observations for which $Y \neq z$:

X	Y
10	x
5	x
12	y
4	z
3	x
2	z
14	y

$\xrightarrow{\text{filter}}$

X	Y
10	x
5	x
12	y
3	x
14	y

Subset with `filter()`

Replaces base R subsetting:

```
> # get 5% most expensive diamonds with "Premium" cut
> filter(diamonds, cut == "Premium", price >= quantile(price, probs = 0.95))
## # A tibble: 985 x 10
##    carat cut     color clarity depth table price     x     y     z
##    <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  1.5  Premium G     VS2      60.5  59   13112  7.39  7.35  4.46
##  2  2.02 Premium I     SI2      58    58   13117  8.34  8.25  4.81
##  3  2.02 Premium I     SI2      61.2  58   13117  8.11  8.04  4.94
##  4  2.09 Premium H     SI2      60.9  58   13119  8.23  8.2   5
##  5  1.53 Premium G     VS1      60.2  59   13119  7.5   7.45  4.5
##  6  1.54 Premium G     VS2      61.8  58   13120  7.43  7.39  4.58
##  7  1.72 Premium H     VS2      61.9  56   13122  7.74  7.67  4.77
##  8  1.7  Premium I     VVS2     61.7  57.4 13127  7.62  7.67  4.71
##  9  1.58 Premium G     VS2      62.6  59   13132  7.47  7.44  4.67
## 10  2.01 Premium J     SI2      60.8  57.2 13133  8.13  8.15  4.95
## # i 975 more rows
```

Subset with `filter()`

Use of logical operators:

```
> # get best and worse quality diamonds
> # alternative: filter(diamonds, cut %in% c("Premium", "Fair"))
> filter(diamonds, cut == "Premium" | cut == "Fair")
## # A tibble: 15,401 x 10
##    carat cut     color clarity depth table price     x     y     z
##    <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.21 Premium E     SI1      59.8    61   326  3.89  3.84  2.31
##  2  0.29 Premium I     VS2      62.4    58   334  4.2   4.23  2.63
##  3  0.22 Fair    E     VS2      65.1    61   337  3.87  3.78  2.49
##  4  0.22 Premium F     SI1      60.4    61   342  3.88  3.84  2.33
##  5  0.2  Premium E     SI2      60.2    62   345  3.79  3.75  2.27
##  6  0.32 Premium E     I1       60.9    58   345  4.38  4.42  2.68
##  7  0.24 Premium I     VS1      62.5    57   355  3.97  3.94  2.47
##  8  0.29 Premium F     SI1      62.4    58   403  4.24  4.26  2.65
##  9  0.22 Premium E     VS2      61.6    58   404  3.93  3.89  2.41
## 10  0.22 Premium D     VS2      59.3    62   404  3.91  3.88  2.31
## # i 15,391 more rows
```

Select with `select()`

Extract useful variables. Say we need only variables X and Y:

X	Y	Z
10	x	1.3
5	x	4.3
12	y	9.3
4	z	2.3
3	x	4.3
2	z	4.1
14	y	9.0

$\xrightarrow{\text{select}}$

X	Y
10	x
5	x
12	y
4	z
3	x
2	z
14	y

Select with `select()`

Replaces `[, c(...)]` in base R:

```
> select(diamonds, carat, color, price)
## # A tibble: 53,940 x 3
##    carat color price
##    <dbl> <ord> <int>
##  1  0.23 E       326
##  2  0.21 E       326
##  3  0.23 E       327
##  4  0.29 I       334
##  5  0.31 J       335
##  6  0.24 J       336
##  7  0.24 I       336
##  8  0.26 H       337
##  9  0.22 E       337
## 10  0.23 H       338
## # i 53,930 more rows
```

Select with `select()`

However, way more flexible:

```
> select(diamonds, carat:color, price)
## # A tibble: 53,940 x 4
##    carat cut       color price
##    <dbl> <ord>     <ord> <int>
##  1  0.23 Ideal     E       326
##  2  0.21 Premium   E       326
##  3  0.23 Good      E       327
##  4  0.29 Premium   I       334
##  5  0.31 Good      J       335
##  6  0.24 Very Good J       336
##  7  0.24 Very Good I       336
##  8  0.26 Very Good H       337
##  9  0.22 Fair      E       337
## 10  0.23 Very Good H       338
## # i 53,930 more rows
```

Select with `select()`

However, way more flexible:

```
> select(diamonds, x, y, ends_with("t"))
## # A tibble: 53,940 x 4
##         x     y carat cut
##     <dbl> <dbl> <dbl> <ord>
##  1  3.95  3.98  0.23 Ideal
##  2  3.89  3.84  0.21 Premium
##  3  4.05  4.07  0.23 Good
##  4  4.2   4.23  0.29 Premium
##  5  4.34  4.35  0.31 Good
##  6  3.94  3.96  0.24 Very Good
##  7  3.95  3.98  0.24 Very Good
##  8  4.07  4.11  0.26 Very Good
##  9  3.87  3.78  0.22 Fair
## 10  4     4.05  0.23 Very Good
## # i 53,930 more rows
```

Add variables with `mutate()`

Add new variables/features. Say we want X^2:

X	Y
10	x
5	x
12	y
4	z
3	x
2	z
14	y

$\xrightarrow{\text{mutate}}$

X	Y	X^2
10	x	100
5	x	25
12	y	144
4	z	16
3	x	9
2	z	4
14	y	196

Add variables with `mutate()`

Replaces basic assignment:

```
> mutate(diamonds,
+   ratio = (price * carat) / depth,
+   excellent = (cut >= "Premium") & (color == "E")
+ )
## # A tibble: 53,940 x 12
##    carat cut   color clarity depth table price     x     y     z ratio excellent
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl> <dbl> <lgl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43  1.22 TRUE
##  2  0.21 Prem~ E     SI1      59.8    61   326  3.89  3.84  2.31  1.14 TRUE
##  3  0.23 Good  E     VS1      56.9    65   327  4.05  4.07  2.31  1.32 FALSE
##  4  0.29 Prem~ I     VS2      62.4    58   334  4.2   4.23  2.63  1.55 FALSE
##  5  0.31 Good  J     SI2      63.3    58   335  4.34  4.35  2.75  1.64 FALSE
##  6  0.24 Very~ J     VVS2     62.8    57   336  3.94  3.96  2.48  1.28 FALSE
##  7  0.24 Very~ I     VVS1     62.3    57   336  3.95  3.98  2.47  1.29 FALSE
##  8  0.26 Very~ H     SI1      61.9    55   337  4.07  4.11  2.53  1.42 FALSE
##  9  0.22 Fair  E     VS2      65.1    61   337  3.87  3.78  2.49  1.14 FALSE
## 10  0.23 Very~ H     VS1      59.4    61   338  4     4.05  2.39  1.31 FALSE
## # i 53,930 more rows
```

Aggregate data with `summarize()`

Reduce data to one single observation. Say we want the average of the X:

X	Y
10	x
5	x
12	y
4	z
3	x
2	z
14	y

$\xrightarrow{\text{summarize}}$

X
7.143

Aggregate data with summarize()

Reduces data frame to single observation (replaces aggregate()):

```
> summarize(diamonds,
+   no_excellent = sum(cut == "Ideal"),
+   avg_price = mean(price, na.rm = TRUE),
+   max_depth = max(depth, na.rm = TRUE)
+ )
## # A tibble: 1 x 3
##   no_excellent avg_price max_depth
##          <int>     <dbl>     <dbl>
## 1        21551     3933.        79
```

Split/group with `group_by()`

Decomposition of dataset into multiple with respect to some variable(s). Say we want to split by the cetegorical variable Y:

X	Y
10	x
5	x
12	y
4	z
3	x
2	z
14	y

¡groupby¿

X	Y
10	x
5	x
3	x

¡groupby¿

X	Y
12	y
14	y

¡groupby¿

X	Y
4	z
2	z

Split/group with `group_by()`

Example with multiple split-variables variables Y and Z:

X	Y	Z
10	x	a
5	x	a
12	y	a
4	z	b
3	x	b
2	z	b
14	y	b

¡groupby¿

X	Y	
12	y	a

¡groupby¿

X	Y	
14	y	b

¡groupby¿

X	Y	
4	z	b
2	z	b

¡groupby¿

X	Y	Z
10	x	a
5	x	a

¡groupby¿

X	Y	Z
3	x	b

Split/group with group_by()

In particularly shines in combination with summarize()):

```
> tmp = group_by(diamonds, color) # split into two tibbles
> print(tmp, n = 2) # show just 2 top lines
## # A tibble: 53,940 x 10
## # Groups:   color [7]
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal   E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium E     SI1      59.8    61   326  3.89  3.84  2.31
## # i 53,938 more rows

> print(ungroup(tmp), n = 2) # combine again (i.e., rbind)
## # A tibble: 53,940 x 10
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal   E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium E     SI1      59.8    61   326  3.89  3.84  2.31
## # i 53,938 more rows
```

Bring it all together

Let us chain multiple commands (see split-apply-combine):

```
> data(diamonds)
>
> tmp = filter(diamonds, clarity %in% c("I1", "SI2", "IF"))
> tmp = group_by(tmp, cut) # split data by cat. variable 'cut'
> tmp = summarize(tmp, # reduce to interesting measures per group
+   no = n(), # number of diamonds
+   mean.depth = mean(depth),
+   max.price = max(price)
+ )
> tmp = ungroup(tmp) # union results
> tmp
## # A tibble: 5 x 4
##   cut          no mean.depth max.price
##   <ord>     <int>      <dbl>     <int>
## 1 Fair        685       64.8     18531
## 2 Good       1248       62.2     18788
## 3 Very Good  2452       61.8     18692
## 4 Premium    3384       61.2     18784
## 5 Ideal      3956       61.7     18806
```

Excursus: pipes

- In programming and in particular in data analysis tasks we often need to pass down the result of on function down to another and another etc. $\leadsto$ many parentheses or intermediate variables necessary ☹
- Solution by package `magrittr`:
  `f(x)` becomes `x %>% f()`
- For multiple arguments:
  `f(x, a, b, c)` becomes `x %>% f(a, b, c)`
- ... and in consequence:
  `h(g(f(x, a, b, c), i, j, k), p, q, r)` simplifies to
  `x %>% f(a, b, c) %>% g(i, j, k) %>% h(p, q, r)`
- Recall that `tidyverse` functions are designed in such way that the first argument is a tibble and so is the result:
  $\leadsto$ `magrittr` pipes are the perfect addition to `tidyverse`!

Bring it all together (with pipes)

Let us chain multiple commands (see split-apply-combine) with with `magrittr` pipes: much more readable!

```
> data(diamonds)
>
> diamonds %>%
+   filter(clarity %in% c("I1", "SI2", "IF")) %>%
+   group_by(cut) %>% # split data by cat. variable 'cut'
+   summarize( # reduce to interesting measures per group
+     no = n(), # number of diamonds
+     mean.depth = mean(depth),
+     max.price = max(price)
+   ) %>%
+   ungroup() # union results
## # A tibble: 5 x 4
##   cut          no mean.depth max.price
##   <ord>     <int>      <dbl>     <int>
## 1 Fair        685       64.8     18531
## 2 Good       1248       62.2     18788
## 3 Very Good  2452       61.8     18692
```

Base R now looks really ugly

Let's reproduce the previous slide with base R:

```
> data(diamonds)
> x = diamonds[diamonds$clarity %in% c("I1", "SI2", "IF"), ]
> res1 = aggregate(x[, "cut"], by = list(cut = x$cut), FUN = length)
> res2 = aggregate(x[, "depth"], by = list(cut = x$cut), FUN = mean)
> res3 = aggregate(x[, "price"], by = list(cut = x$cut), FUN = max)
> res = cbind(res1, res2[, "depth"], res3[, "price"])
> colnames(res) = c("cut", "no", "mean.depth", "max.price")
> res
##          cut   no mean.depth max.price
## 1       Fair  685   64.75620     18531
## 2       Good 1248   62.18373     18788
## 3 Very Good 2452   61.80473     18692
## 4    Premium 3384   61.17033     18784
## 5      Ideal 3956   61.68213     18806
```

Exercises

1. Load the `diamonds` data set and use `tidyverse`.
2. Find the number of observations and the median values of `price` and `carat` for all combinations of `clarity` and `cut` considering all diamonds with `color` in $\{I, J, H\}$.
3. Split the data by `color` and `cut`, normalize the `price` in each group (divide by the maximum price). Next, calulate the mean and standard deviation of the normalized price per group. Next, group by `cut` and subset all rows where the normalized price is larger than its average value. Finally, sort the results with respect to `cut` and in decreasing order of the mean normalized price (Hint: check `arrange()`).
4. Split the data by `carat` and calculate the mean `depth`. Does this split make sense? Why or why not?

Sample solutions i

Ad 2) Find the number of observations and the median values of `price` and `carat` for all combinations of `clarity` and `cut` considering all diamonds with `color` in $\{I, J, H\}$.

```
>   diamonds %>%
+     filter(color %in% c("I", "J", "H")) %>%
+     group_by(clarity, cut) %>%
+     summarize(
+       n_obs = n(),
+       median_price = median(price)
+     ) %>%
+     ungroup() %>%
+     print(n = 5)
```

Sample solutions ii

Ad 3) Split the data by `color` and `cut`, normalize the `price` in each group (divide by the maximum price). Next, calulate the mean and standard deviation of the normalized price per group. Next, group by `cut` and subset all rows where the normalized price is larger than its average value. Finally, sort the results with respect to `cut` and in decreasing order of the mean normalized price (Hint: check `arrange()`).

```
>   diamonds %>%
+     group_by(color, cut) %>%
+     mutate(price_norm = price / max(price)) %>%
+     summarize(
+       price_mean = mean(price_norm),
+       price_sd = sd(price_norm)
+     ) %>%
+     group_by(cut) %>%
+     filter(price_mean >= mean(price_mean)) %>%
+     ungroup() %>%
+     arrange(cut, desc(price_mean)) %>%
+     print(n = 5)
```

Sample solutions iii

Ad 4) Split the data by `carat` and calculate the mean `depth`. Does this split make sense? Why or why not?

```
>   aggr = diamonds %>%
+     group_by(carat) %>% # 273 groups! carat is numeric
+     summarize(depth_mean = mean(depth)) %>%
+     ungroup()
>
>   dim(aggr)
```

Data wrangling

Tidy data

So far we dealt with *clean data*, so-called *tidy data* (Wickham 2014b)

- Each column is a variable. In particular in R all variables can be accessed with `dataset$variable_name` (or `dataset[["variable_name"]]`.
- Each line is an observation.
- Each value has its own cell.
- Each type of observational unit is a table (i.e., a dataframe/tibble).
- Such a format is a very good starting point for analysis (easy to manipulate, visualize, query, understand etc.).

Tidy data is rare

- Usually considerable effort is spent on cleaning/tidying; up to 80% (Dasu and Johnson 2003)!
- Data preparation involves:
  - Collecting data from different sources.
  - Dealing with missing values (`NA`s) via imputaton, i.e., replacing missing values with some reasonable values.[13]
  - Parsing dates in different formats.
  - Recoding factors/categories.
  - Renaming variables.
  - etc.
- Process has to be repeated if new questions come up.

[13] Part of DA1. However, I decided to do this later.

Messy data

We speak about *messy data* if one of the following conditions is met:

- Multiple variables are stored in one column (separated by some delimiter).
- Column headers are variables, not variable names.
- Multiple types of observational units stored in one table.
- A single observational unit is stored in multiple tables.
- etc.

Messy data: multiple variables in one column

Schema of multiple variables stored in one column (separated by some delimiter; here the semi-colon):

X	Y
10	x;a;l
5	x;a;k
12	y;b;l
4	y;a;m
3	x;c;k
2	z;c;l
14	y;c;l

$\xrightarrow{\text{tidy}}$

X	Y_1	Y_2	Y_3
10	x	a	l
5	x	a	k
12	y	b	l
4	y	a	m
3	x	c	k
2	z	c	l
14	y	c	l

Messy data: multiple variables in one column

Explode the column with `separate()`:

```
> print(diamonds, n = 2)
## # A tibble: 53,940 x 8
##   carat cut     color clarity depth table price xyz
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <chr>
## 1  0.23 Ideal   E     SI2      61.5    55   326 3.95/3.98/2.43
## 2  0.21 Premium E     SI1      59.8    61   326 3.89/3.84/2.31
## # i 53,938 more rows

> diamonds %>% separate(xyz, into = c("x", "y", "z"), sep = "/") %>% print(n = 2)
## # A tibble: 53,940 x 10
##   carat cut     color clarity depth table price x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <chr> <chr> <chr>
## 1  0.23 Ideal   E     SI2      61.5    55   326 3.95  3.98  2.43
## 2  0.21 Premium E     SI1      59.8    61   326 3.89  3.84  2.31
## # i 53,938 more rows
```

Note: the split columns are characters!

Messy data: multiple variables in one column

Explode the column with `separate()` and let the function guess the data types:[14]

```
> print(diamonds, n = 2)
## # A tibble: 53,940 x 8
##   carat cut     color clarity depth table price xyz
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <chr>
## 1  0.23 Ideal   E     SI2      61.5    55   326 3.95/3.98/2.43
## 2  0.21 Premium E     SI1      59.8    61   326 3.89/3.84/2.31
## # i 53,938 more rows

> diamonds %>%
+   separate(xyz, into = c("x", "y", "z"), sep = "/", convert = TRUE) %>%
+   print(n = 2)
## # A tibble: 53,940 x 10
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal   E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium E     SI1      59.8    61   326  3.89  3.84  2.31
## # i 53,938 more rows
```

[14] We could also do it explicitely.

Messy data: column headers are values

Schema of column headers being variables $\rightsquigarrow$ pivot the offending columns into a new pair (Y, Z) of variables:

X	a	b	c
10	1	2	3
5	4	5	6
12	7	8	9

$\xrightarrow{\text{tidy}}$

X	Y	Z
10	a	1
10	b	2
10	c	3
5	a	4
5	b	5
5	c	6
12	a	7
12	b	8
12	c	9

Messy data: column headers are values

Pivot the offending columns into a new pair of variables with `pivot_longer()`:

```
> # number of diamonds per color
> diamaggr = tibble(
+  D = 2834, E = 3903, F = 3826, G = 4884,
+  H = 3115, I = 2093, J = 896)
>
> diamaggr
## # A tibble: 1 x 7
##       D     E     F     G     H     I     J
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  2834  3903  3826  4884  3115  2093   896

> diamaggr %>%
+   pivot_longer(colnames(diamaggr), names_to = "color", values_to = "count")
## # A tibble: 7 x 2
##   color count
##   <chr> <dbl>
## 1 D      2834
## 2 E      3903
## 3 F      3826
```

Messy data: observations spreads multiple rows

Observations span multiple rows. Here, X and Y in column K are actually variables with values given in column V:

X	K	V
A	Y	2
A	Z	5
B	Y	10
B	Z	8
C	Y	14
C	Z	2

$\xrightarrow{\text{tidy}}$

X	Y	Z
A	2	5
B	10	8
C	14	2

Messy data: observations spreads multiple rows

This is besically `pivot_longer()` the other way around. Hence, the function is called `pivot_wider()`:

```
> # number of diamonds per color
> diamaggr = tibble(
+   cut = c("Fair", "Fair", "Good", "Good", "Very Good", "Very Good"),
+   key = c("n", "n_ideal", "n", "n_ideal", "n", "n_ideal"),
+   count = c(1610, 23, 4906, 200, 12082, 92))
> diamaggr
## # A tibble: 6 x 3
##   cut       key     count
##   <chr>     <chr>   <dbl>
## 1 Fair      n        1610
## 2 Fair      n_ideal    23
## 3 Good      n        4906
## 4 Good      n_ideal   200
## 5 Very Good n       12082
## 6 Very Good n_ideal    92

> diamaggr %>% pivot_wider(names_from = key, values_from = count)
```

Exercises

1. Think of situations where we would actually like to merge two (or more) variables into one variable (violating tidy data according to our definition).

2. Search for the function `unite()`. Use it to merge columns `cut` and `color`. Split again with `separate()`.

Sample solutions i

Ad 1) Think of situations where we would actually like to merge two (or more) variables into one variable (violating tidy data according to our definition).
R: Imagine you have columns `day`, `month` and `year`. It makes sense to combine them in order to format dates in a certain way, say, *Oct. 15, 2022*.[15]

[15] Nevertheless it is a very good idea to start with tidy data.

Sample solutions ii

Ad 2) Search for the function `unite()`. Use it to merge columns `cut` and `color`. Split again with `separate()`.

```
> x = diamonds %>% unite(index, cut, color, sep = "-")
> print(x, n = 1)
## # A tibble: 53,940 x 7
##   carat index   clarity depth table price xyz
##   <dbl> <chr>   <ord>   <dbl> <dbl> <int> <chr>
## 1  0.23 Ideal-E SI2      61.5    55   326 3.95/3.98/2.43
## # i 53,939 more rows

> # Alternative: diamonds %>% mutate(index = sprintf("(%s, %s)", cut, color))
> x %>% separate(index, into = c("cut", "color"), sep = "-") %>% print(n = 1)
## # A tibble: 53,940 x 8
##   carat cut   color clarity depth table price xyz
##   <dbl> <chr> <chr> <ord>   <dbl> <dbl> <int> <chr>
## 1  0.23 Ideal E     SI2      61.5    55   326 3.95/3.98/2.43
## # i 53,939 more rows
```

Tidy data is the holy grail, isn't it?

The answer is **NO**!

- Tidy data is a good foundation if your data is in fact rectangular.
- "Non-tidy" data sets may be beneficial when it comes to required disk space.
- Some analysis methods require different input (e.g., matrices to perform fast linear algebra operations).
- Data may simply not be rectangular.
- **Takeaway:** non-tidy (in the sense of Wickham 2014b) data can also be well organized and clean.

We just saw the tip of the iceberg

Topics not covered

- Many `dyplr` function (we discussed the most essential functions only).
- Relational data and join-operations to "merge" observations from two tables.
- Working with dates and times (essential for time series data).
- Visualizing tidy data with `ggplot2` (next week!)

Wrap-Up

Wrap-Up

Todays content

How to use `tidyverse` grammer of data manipulation or "how to make working with data even more fun" ☺

Your task(s)

- We just scratched the surface here! Work through the data transformation chapter in (Wickham and Grolemund 2017).
- Work on next exercise sheet
- Optional: try to reproduce (some of) todays `tidyverse` examples with base R.

References I

Wickham, Hadley and Garrett Grolemund (Jan. 2017). *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data.* 1st ed. O'Reilly Media. ISBN: 1491910399.

Wickham, Hadley (2009). *ggplot2: elegant graphics for data analysis.* Springer New York. ISBN: 978-0-387-98140-6.

— (2014a). *Advanced R.* Chapman & Hall/CRC The R Series. Taylor & Francis. ISBN: 9781466586963. URL: https://books.google.de/books?id=PFHFNAEACAAJ.

— (2015). *R Packages.* 1st. O'Reilly Media, Inc. ISBN: 1491910593.

— (2014b). "Tidy Data". In: *Journal of Statistical Software* 59.10, pp. 1–23. DOI: 10.18637/jss.v059.i10.

Dasu, Tamraparni and Theodore Johnson (2003). *Exploratory Data Mining and Data Cleaning.* 1st ed. USA: John Wiley & Sons, Inc. ISBN: 0471268518.