

A panoramic photograph of the Magdeburg skyline across the Elbe River. In the background, the Gothic Magdeburg Cathedral with its two towers stands prominently. To the right, the Thomaskirche (Church of Saint Thomas) with its red-tiled roof and tall spire is visible. The foreground shows the river with several boats, including a large cargo ship labeled "NIKE PRAHA" and a white passenger ship. A bridge spans the river in the middle ground.

3 - The *Tidyverse*

Data Science with R • Summer 2021

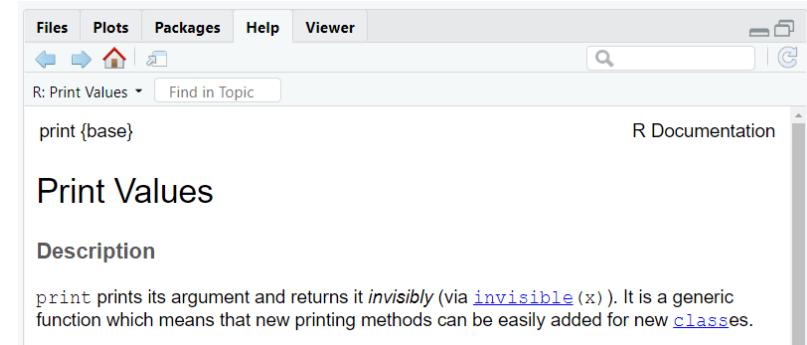
Uli Niemann · Knowledge Management & Discovery Lab

<https://brain.cs.uni-magdeburg.de/kmd/DataSciR/>

Why is (base) R hard to learn?

R has some idiosyncrasies that make it hard for learners who are used to other programming languages, e.g.:

- unhelpful help `?print`
- too many functions `colnames()`, `names()`
- inconsistent names `read.csv()`, `load()`, `readRDS()`
- clumsy console output `print(iris)`
- high flexibility (*is this bad?*)
- too many ways to select variables: `df$x`, `df$"x"`, `df[, "x"]`, or `df[[1]]`



⌚ "invisibly"?

⌚ "generic function"?

⌚ "class"?

See a more comprehensive list:

Robert A. Muenchen. "[Why R is Hard to Learn](#)". r4stats.com. Accessed 19.07.2018.

Base R vs. tidyverse code

⌚ "What does this base R code?"

```
aggregate(iris[, "Sepal.Length"],
          list(Species = iris[, "Species"]),
          mean)
```

```
##      Species     x
## 1      setosa 5.006
## 2 versicolor 5.936
## 3 virginica 6.588
```

⌚ "What are the square brackets [for?" → They are used for subsetting a data frame.

⌚ "What is list()?" → It's a type of R object.

⌚ "Why is the mean() function seemingly applied without argument?" → It is being passed to the subsets of the data frame.

⌚ "In the second line, why does the first "Species" does not need to be quoted, but the second "Species" does?" ...

⌚ tidyverse equivalent:

```
library(dplyr)
group_by(iris, Species) %>%
  summarize(avg_sl = mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species     avg_sl
##   <fct>       <dbl>
## 1 setosa      5.01
## 2 versicolor  5.94
## 3 virginica   6.59
```

The Tidyverse



Quote from the [Tidyverse website](#):

"R packages for data science. The tidyverse is an **opinionated collection of R packages designed for data science**. All packages share an underlying **design philosophy, grammar, and data structures**."

- collection of open-source `R` packages mainly for data wrangling and visualization
- shared conventions and common APIs across all Tidyverse packages

Installation

```
# Install all Tidyverse packages
install.packages("tidyverse")

# Attach core packages
library(tidyverse)
```

The meta-package `tidyverse` contains 26 packages. When running `library(tidyverse)`, only the **core** tidyverse packages become available in your current `R` session. The core packages are:

Core-Packages:

- `ggplot2`: creation of graphics
- `dplyr`: data wrangling
- `tidyr`: data reshaping
- `readr`: import of flat data files, e.g. csv
- `tibble`: enhanced data frames
- `stringr`: string manipulation
- `forcats`: factor manipulation
- `purrr`: functions for working with list columns

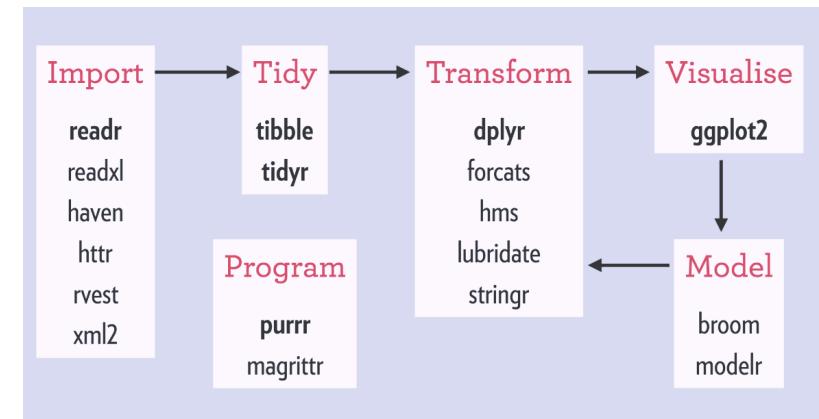


Figure source: Joseph Rickert. ["What is the tidyverse?"](#). R Views. Accessed 13.07.2019.



Data wrangling with `dplyr`

⌚ "What can I do with `dplyr`?"

- get an overview of a tibble with `glimpse()`
- select a subset of columns with `select()`
- filter a subset of rows with `filter()`
- add new or change existing columns with `mutate()`
- pick a subset of rows with `slice()`
- reorder rows with `arrange()`
- group rows by a grouping column with `group_by()`
- calculate a summary (per group) with `summarize()`
- join two distinct tibbles by a common column with `*join()`
- ... (and more)



Consistent API design:

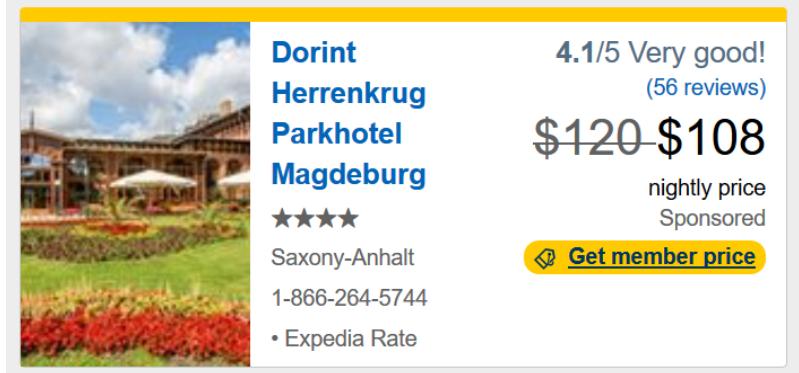
- first argument of each of these **verbs** is a data frame
- the output is (usually) also a data frame

Case study: customer bookings data

The company behind a travel price aggregator website wants to analyze its booking data to optimize the website's usability and thus improve their customers' travel experience.

The data is organized into two files:

- `bookings.csv`: hotel bookings
- `properties.csv`: hotel facilities



A screenshot of a travel price aggregator website showing a hotel listing. The listing includes a photo of the hotel, the name "Dorint Herrenkrug Parkhotel Magdeburg", a 4-star rating, location "Saxony-Anhalt", phone number "1-866-264-5744", and a price range from "\$120" to "\$108". A "Get member price" button is also visible.

Dorint
Herrenkrug
Parkhotel
Magdeburg

★★★★

Saxony-Anhalt

1-866-264-5744

\$120-\$108

nightly price
Sponsored

[Get member price](#)

```
library(tidyverse)
```

```
bookings <- read_csv("../data/bookings.csv")
```

```
##  
## -- Column specification -----  
## cols(  
##   booker_id = col_character(),  
##   property_id = col_double(),  
##   room_nights = col_double(),  
##   price_per_night = col_double(),  
##   checkin_day = col_character(),  
##   for_business = col_logical(),  
##   status = col_character(),  
##   review_score = col_double()  
## )
```

Get an overview with `glimpse()`

```
glimpse(bookings)
```

```
## Rows: 10,000
## Columns: 8
## $ booker_id      <chr> "215934017ba98c09f30dedd29237b43dad5c7b5f", "7f590fd6d318248a486~
## $ property_id    <dbl> 2668, 4656, 4563, 4088, 2188, 4171, 2907, 5141, 1696, 1901, 2188~
## $ room_nights    <dbl> 4, 5, 6, 7, 4, 2, 4, 4, 1, 7, 1, 9, 6, 4, 5, 5, 2, 2, 2, 2, 1, 3~
## $ price_per_night <dbl> 91.46696, 106.50500, 86.99137, 92.36562, 104.83894, 109.98188, 1~
## $ checkin_day     <chr> "mon", "tue", "wed", "fri", "tue", "fri", "fri", "wed", "wed", "~"
## $ for_business    <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, F~
## $ status           <chr> "cancelled", "cancelled", "stayed", "stayed", "stayed", "cancell~
## $ review_score     <dbl> NA, NA, 6.258123, 5.953598, 6.434745, NA, 7.599461, NA, 6.972698~
```

Select columns with `select()`



Select one column

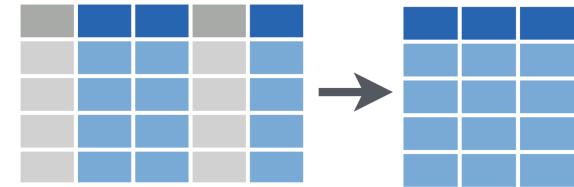
Select multiple columns

Exclude columns

Select the column `review_score`.

```
select(bookings, review_score)
```

```
## # A tibble: 10,000 x 1
##   review_score
##   <dbl>
## 1 NA
## 2 NA
## 3 6.26
## 4 5.95
## 5 6.43
## 6 NA
## 7 7.6
## 8 NA
## 9 6.97
## 10 NA
## # ... with 9,990 more rows
```



The output is **always** a `data.frame` or `tibble`, regardless of whether a single or multiple columns are selected.

Select columns with `select()`



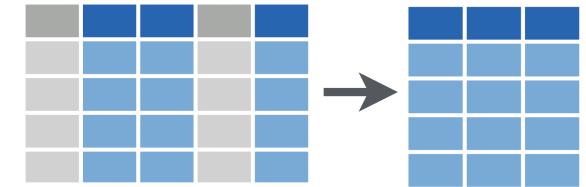
Select one column

Select multiple columns

Exclude columns

Select multiple columns by specifying column names as additional arguments.

```
select(bookings, review_score, status)
```



```
## # A tibble: 10,000 x 2
##       review_score status
##   <dbl> <chr>
## 1      6.26 stayed
## 2      5.95 stayed
## 3      6.43 stayed
## 4      7.60 stayed
## 5      6.97 stayed
## 6      NA  cancelled
## 7      NA  cancelled
## 8      NA  cancelled
## 9      NA  cancelled
## 10     NA  cancelled
## # ... with 9,990 more rows
```

Select columns with `select()`

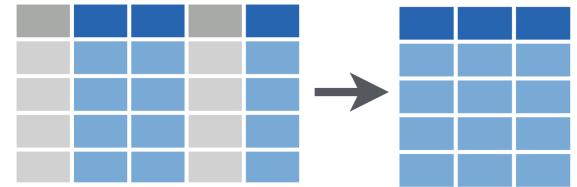
Select one column

Select multiple columns

Exclude columns

Negative selection: select all but specific columns using `-`.

```
select(bookings, -booker_id)
```



```
## # A tibble: 10,000 x 7
##   property_id room_nights price_per_night checkin_day for_business status   review_score
##       <dbl>        <dbl>            <dbl>      <chr>        <lgl>    <chr>           <dbl>
## 1       2668          4            91.5     mon        FALSE  cancelled        NA
## 2       4656          5            107.     tue        FALSE  cancelled        NA
## 3       4563          6            87.0     wed        FALSE  stayed           6.26
## 4       4088          7            92.4     fri        FALSE  stayed           5.95
## 5       2188          4            105.     tue        FALSE  stayed           6.43
## 6       4171          2            110.     fri        FALSE  cancelled        NA
## 7       2907          4            116.     fri        FALSE  stayed           7.60
## 8       5141          4            111.     wed        FALSE  cancelled        NA
## 9       1696          1            106.     wed        FALSE  stayed           6.97
## 10      1901          7            82.3     sat        FALSE  cancelled        NA
## # ... with 9,990 more rows
```

Helper functions for `select()`

Helper functions facilitate selecting multiple columns whose names satisfy a specific criterion.

Helper function	Description
<code>contains("abc")</code>	Names containing "abc"
<code>starts_with("abc")</code>	Names starting with "abc"
<code>ends_with("abc")</code>	Names ending with "abc"
<code>num_range("a", 1:3)</code>	Names matching the numerical range <code>a1, a2, a3</code>
<code>any_of(c("ab", "c"))</code>	Any of the names within a character vector
<code>all_of(c("ab", "c"))</code>	All of the names within a character vector
<code>matches(".id")</code>	Names matching the regular expr. ".id"
<code>everything()</code>	All (remaining) names
<code>last_col()</code>	Last column

Examples of `select()` with helper functions

Combined helpers

`any_of()`

`all_of()`

"Select all columns whose names either contain 'id' or end with 'night'!"

```
select(bookings, contains("id"), ends_with("night"))
```

```
## # A tibble: 10,000 x 3
##   booker_id          property_id price_per_night
##   <chr>              <dbl>            <dbl>
## 1 215934017ba98c09f30dedd29237b43dad5c7b5f  2668        91.5
## 2 7f590fd6d318248a48665f7f7db529aca40c84f5  4656       107.
## 3 10f0f138e8bb1015d3928f2b7d828ccb50cd0804  4563        87.0
## 4 7b55021a4160dde65e31963fa55a096535bcad17  4088        92.4
## 5 6694a79d158c7818cd63831b71bac91286db5aff  2188       105.
## 6 d0358740d5f15e85523f94ab8219f25d8c017347  4171        110.
## 7 944e568a0b511b9140bcc7c7c80c68c3edd3a86f  2907        116.
## 8 95476c2ef6bb9e3c227b46f1283310737fa13b7a  5141        111.
## 9 df235631a4c281c01e007b6a4c364e07f5843994  1696        106.
## 10 ff610140227d40d27daa01f0dc5d64ea82e833d  1901        82.3
## # ... with 9,990 more rows
```

Examples of `select()` with helper functions

Combined helpers

any_of()

all_of()

"Select any of the two columns `room_nights`, `this_column_is_not_there`."

```
select(bookings, any_of(c("room_nights", "this_column_is_not_there")))

## # A tibble: 10,000 x 1
##       room_nights
##   <dbl>
## 1      4
## 2      5
## 3      6
## 4      7
## 5      4
## 6      2
## 7      4
## 8      4
## 9      1
## 10     7
## # ... with 9,990 more rows
```

Examples of `select()` with helper functions

Combined helpers

`any_of()`

`all_of()`

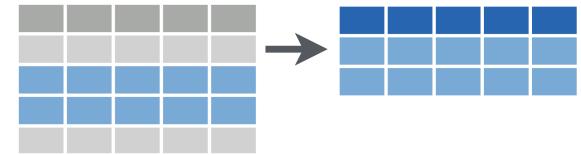
"Select both columns `room_nights`, `this_column_is_not_there`."

```
select(bookings, all_of(c("room_nights", "this_column_is_not_there")))
```

```
## Error: Can't subset columns that don't exist.  
## x Column `this_column_is_not_there` doesn't exist.
```

Filter rows with `filter()`

Filter bookings which stayed, i.e., didn't cancel.



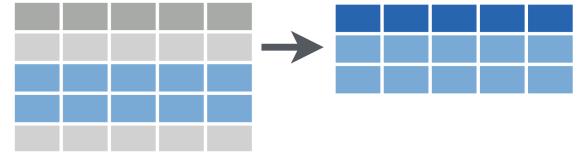
```
filter(bookings, status == "stayed")
```

```
## # A tibble: 7,775 x 8
##   booker_id      property_id room_nights price_per_night checkin_day for_business status
##   <chr>          <dbl>        <dbl>            <dbl> <chr>       <lgl>      <chr>
## 1 10f0f138e8bb10~     4563         6            87.0 wed        FALSE      stayed
## 2 7b55021a4160dd~     4088         7            92.4 fri        FALSE      stayed
## 3 6694a79d158c78~     2188         4           105. tue        FALSE      stayed
## 4 944e568a0b511b~     2907         4           116. fri        FALSE      stayed
## 5 df235631a4c281~     1696         1           106. wed        FALSE      stayed
## 6 aeef2e2d3d23b1~     2188         1           109. wed        FALSE      stayed
## 7 5a1442f4c7237e~     2307         9            84.2 sat        FALSE      stayed
## 8 39804a2e3fb2e4~     2907         6           112. sun        FALSE      stayed
## 9 e150e559405ef2~     2870         4           127. sat        FALSE      stayed
## 10 4e9c7c21dfcf27~    1674         5           102. sun        FALSE     stayed
## # ... with 7,765 more rows, and 1 more variable: review_score <dbl>
```

```
# the same as:
filter(bookings, status != "cancelled")
```

Filter rows with `filter()`

Combine multiple conditions with `&`.



```
filter(  
  bookings,  
  status == "stayed" &  
  !is.na(review_score) &  
  between(price_per_night, 90, 120)  
)
```

```
## # A tibble: 3,032 x 8  
##   booker_id      property_id room_nights price_per_night checkin_day for_business status  
##   <chr>          <dbl>        <dbl>            <dbl> <chr>       <lgl>       <chr>  
## 1 7b55021a4160dd~     4088         7            92.4 fri        FALSE      stayed  
## 2 6694a79d158c78~     2188         4           105. tue        FALSE      stayed  
## 3 944e568a0b511b~     2907         4           116. fri        FALSE      stayed  
## 4 df235631a4c281~     1696         1           106. wed        FALSE      stayed  
## 5 39804a2e3fb2e4~     2907         6           112. sun        FALSE      stayed  
## 6 4e9c7c21dfcf27~     1674         5           102. sun        FALSE      stayed  
## 7 8537874da3fa74~     4420         2           106. sat        FALSE      stayed  
## 8 0fcce2f5be8e34~     1951         3           110. wed        FALSE      stayed  
## 9 9864af3aba5bcb~     1981         3           96.9 sun        FALSE      stayed  
## 10 ccd24f993579b4~    2307         3           115. fri       FALSE      stayed  
## # ... with 3,022 more rows, and 1 more variable: review_score <dbl>
```

Sort rows with `arrange()`

```
arrange(bookings, price_per_night) # default: sort in ascending order
```

```
## # A tibble: 10,000 x 8
##   booker_id      property_id room_nights price_per_night checkin_day for_business status
##   <chr>          <dbl>        <dbl>           <dbl> <chr>       <lgl>      <chr>
## 1 028c395e745c41~     3096         1            39.4  tue      FALSE      stayed
## 2 037bec01b4f11d~     3619         1            45.8  wed      TRUE      stayed
## 3 7341b69d8de545~     3983         1            47.7  thu      TRUE      stayed
## 4 cbedb290616cb7~    1172         2            53.2  thu      TRUE      stayed
## 5 dff926426fdf09~    4354         6            54.2  fri      FALSE      stayed
## 6 b28ca9bd608e46~    3118         2            54.8  sun      TRUE      stayed
## 7 e2aff0e6df1abe~    4116         5            55.7  tue      FALSE      stayed
## 8 28421366c10d89~    4391         4            56.6  fri      FALSE      stayed
## 9 a07a0a8a9d8924~    4150         2            57.7  sat      FALSE      cance~
## 10 97a4a1dcfce1ab~   1172         2            61.0  tue     TRUE      stayed
## # ... with 9,990 more rows, and 1 more variable: review_score <dbl>
```

```
arrange(bookings, desc(price_per_night))
```

```
## # A tibble: 10,000 x 8
##   booker_id      property_id room_nights price_per_night checkin_day for_business status
##   <chr>          <dbl>        <dbl>           <dbl> <chr>       <lgl>      <chr>
## 1 3d62487f31468b~     4931         1            273.  tue      TRUE      cance~
## 2 dc6add0d9538bf~    2305         1            272.  fri      TRUE      stayed
## 3 c57c799c4fa86c~    5034         1            263.  tue      TRUE      stayed
## 4 0dbc70d94f448b~    1819         1            259.  fri      TRUE      cance~
## 5 1d0d33ccf02e4e~    2306         1            257.  fri      FALSE      cance~
## 6 6b70130a6cd8bb~    3916         1            256.  tue      TRUE      stayed
## 7 c478924be4be9c~    2464         1            255.  sun      FALSE      stayed
```

Select rows by position with `slice()`

```
x <- arrange(bookings, desc(review_score))  
slice(x, 1:10)
```

```
## # A tibble: 10 x 8  
##   booker_id      property_id room_nights price_per_night checkin_day for_business status  
##   <chr>          <dbl>        <dbl>           <dbl> <chr>       <lgl>       <chr>  
## 1 51c3e2e855240f~     1970         3            126. fri      FALSE      stayed  
## 2 99f91abc431bad~    2175         1            189. wed      FALSE      stayed  
## 3 1d7930a998f002~    3089         6            77.4 thu      FALSE      stayed  
## 4 58c94e9418fe51~    3089         5            91.5 sat      FALSE      stayed  
## 5 164c4797e41a32~    3890         1            232. sat      TRUE       stayed  
## 6 3119d16c994efc~    3089         6            93.4 fri      FALSE      stayed  
## 7 10f0f138e8bb10~    3089         1            175. sun      FALSE      stayed  
## 8 714d7f95f1766d~    1855         2            129. thu      FALSE      stayed  
## 9 b4d77fe4ea2472~    3089         2            83.3 sat      FALSE      stayed  
## 10 68686391598302~   3089         5            106. fri     FALSE      stayed  
## # ... with 1 more variable: review_score <dbl>
```

Chaining multiple operations

`dplyr` code is very intuitive and expressive, but also quite verbose.

 "How to organize the code so that it remains readable even for a high number of operations?"

Example: For all bookers who didn't cancel their trip and checked in on a friday, we are interested in the 5 highest review scores. Only the columns `price_per_night` and `review_score` should be included in a new tibble which is sorted by `review_score` in descending order.

Chaining multiple operations

Nesting

Save & overwrite intermediate results

Save result of each operation as new object

Solution 1 (nesting):

```
select(  
  slice(  
    arrange(  
      filter(  
        bookings, !status == "cancelled" & checkin_day == "fri"),  
        desc(review_score)  
      ),  
      1:5  
    ),  
    price_per_night, review_score  
)
```

```
## # A tibble: 5 x 2  
##   price_per_night review_score  
##       <dbl>         <dbl>  
## 1       126.          10  
## 2       93.4          9.80  
## 3       106.          9.77  
## 4       148.          9.64  
## 5       225.          9.49
```



Chaining multiple operations



Nesting

Save & overwrite intermediate results

Save result of each operation as new object

Solution 2 (save and overwrite intermediate results):

```
x <- filter(bookings, !status == "cancelled" & checkin_day == "fri")
x <- arrange(x, review_score)
x <- slice(x, 1:5)
x <- select(x, price_per_night, review_score)
x
```

```
## # A tibble: 5 x 2
##   price_per_night review_score
##       <dbl>        <dbl>
## 1      123.        2.79
## 2      109.        3.31
## 3      127.        3.75
## 4      111.        4.09
## 5      79.1        4.15
```

Chaining multiple operations

Nesting

Save & overwrite intermediate results

Save result of each operation as new object

Solution 3 (save each intermediate results as new object):

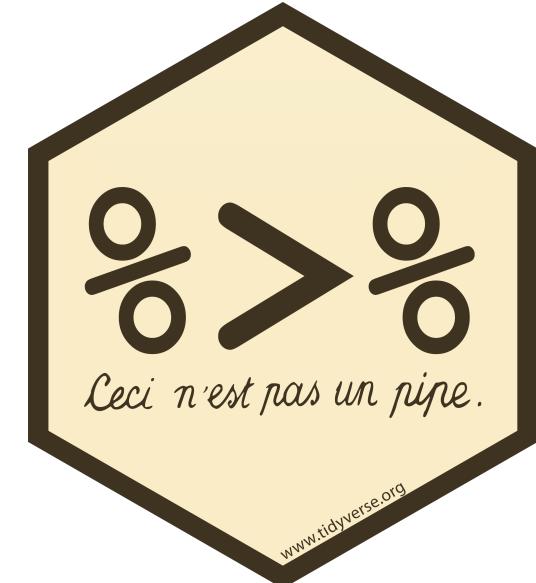
```
x1 <- filter(bookings, !status == "cancelled" & checkin_day == "fri")
x2 <- arrange(x1, desc(review_score))
x3 <- slice(x2, 1:5)
x4 <- select(x3, price_per_night, review_score)
x4
```

There is a mistake in the above code. Can you spot it?

The pipe operator %>%

- The **pipe operator** %>% is used to pass information from one operation to the next.
- Its main purpose is to **express a sequence of operations**.
- Using the pipe, a function's output becomes the first argument of the subsequent function.
- Thus, the pipe operator helps to write **code that is easy to read and understand**.

```
bookings %>%
  filter(!status == "cancelled" & checkin_day == "fri") %>%
  arrange(desc(review_score)) %>%
  slice(1:5) %>%
  select(price_per_night, review_score)
```



"From table `bookings`, filter all non-canceled bookings with check-in on a Friday
THEN sort by review score in descending order
THEN take the top-5 bookings
THEN return price per night and review score."

- In RStudio, the shortcut for %>% is **Ctrl+Shift+M**.
- Although the pipe operator is implemented in the package `magrittr`, we do not need to load this package explicitly when we have loaded `tidyverse`.

Adding new columns with `mutate()` +

Create a new column for the **total price** of a booking, which is the product of `price_per_night` and `room_nights`.

`mutate()` `mutate() + select()` `mutate(..., .before = ...)` `mutate(..., .after = ...)`

```
bookings %>%
  mutate(total_price = price_per_night * room_nights)
```

```
## # A tibble: 10,000 x 9
##   booker_id      property_id room_nights price_per_night checkin_day for_business status
##   <chr>          <dbl>        <dbl>           <dbl> <chr>       <lgl>      <chr>
## 1 215934017ba98c~     2668         4            91.5 mon        FALSE      cance~
## 2 7f590fd6d31824~     4656         5            107. tue       FALSE      cance~
## 3 10f0f138e8bb10~     4563         6            87.0 wed       FALSE      stayed
## 4 7b55021a4160dd~     4088         7            92.4 fri       FALSE      stayed
## 5 6694a79d158c78~     2188         4            105. tue      FALSE      stayed
## 6 d0358740d5f15e~     4171         2            110. fri      FALSE      cance~
## 7 944e568a0b511b~     2907         4            116. fri      FALSE      stayed
## 8 95476c2ef6bb9e~     5141         4            111. wed      FALSE      cance~
## 9 df235631a4c281~     1696         1            106. wed      FALSE      stayed
## 10 ff610140227d40~    1901         7            82.3 sat      FALSE      cance~
## # ... with 9,990 more rows, and 2 more variables: review_score <dbl>, total_price <dbl>
```

Note that a new variable will be appended as the last column of the data frame.

Adding new columns with `mutate()` +

Create a new column for the **total price** of a booking, which is the product of `price_per_night` and `room_nights`.

mutate() **mutate() + select()** mutate(..., .before = ...) mutate(..., .after = ...)

(Only show `price_per_night`, `room_nights`, `total_price`)

```
bookings %>%
  mutate(total_price = price_per_night * room_nights) %>%
  select(price_per_night, room_nights, total_price)
```

```
## # A tibble: 10,000 x 3
##   price_per_night room_nights total_price
##       <dbl>        <dbl>      <dbl>
## 1         91.5          4      366.
## 2        107.           5      533.
## 3        87.0           6      522.
## 4        92.4           7      647.
## 5        105.           4      419.
## 6        110.           2      220.
## 7        116.           4      465.
## 8        111.           4      446.
## 9        106.           1      106.
## 10       82.3           7      576.
## # ... with 9,990 more rows
```

Adding new columns with `mutate()` +

Create a new column for the **total price** of a booking, which is the product of `price_per_night` and `room_nights`.

mutate() mutate() + select() **mutate(..., .before = ...)** mutate(..., .after = ...)

(Insert the new column before `booker_id`.)

```
bookings %>%
  mutate(total_price = price_per_night * room_nights, .before = booker_id)
```

```
## # A tibble: 10,000 x 9
##   total_price booker_id  property_id room_nights price_per_night checkin_day for_business
##       <dbl>     <chr>      <dbl>        <dbl>          <dbl>     <chr>      <lgl>
## 1       366.  215934017~     2668         4            91.5    mon      FALSE
## 2       533.  7f590fd6d~     4656         5           107.    tue      FALSE
## 3       522.  10f0f138e~     4563         6           87.0    wed      FALSE
## 4       647.  7b55021a4~     4088         7           92.4    fri      FALSE
## 5       419.  6694a79d1~     2188         4           105.    tue      FALSE
## 6       220.  d0358740d~     4171         2           110.    fri      FALSE
## 7       465.  944e568a0~     2907         4           116.    fri      FALSE
## 8       446.  95476c2ef~     5141         4           111.    wed      FALSE
## 9       106.  df235631a~     1696         1           106.    wed      FALSE
## 10      576.  ff6101402~     1901         7           82.3    sat      FALSE
## # ... with 9,990 more rows, and 2 more variables: status <chr>, review_score <dbl>
```

Adding new columns with `mutate()` +

Create a new column for the **total price** of a booking, which is the product of `price_per_night` and `room_nights`.

`mutate()` `mutate() + select()` `mutate(..., .before = ...)` **`mutate(..., .after = ...)`**

(Insert the new column after the second column (`property_id`)).

```
bookings %>%  
  mutate(total_price = price_per_night * room_nights, .after = 2)
```

```
## # A tibble: 10,000 x 9  
##   booker_id  property_id total_price room_nights price_per_night checkin_day for_business  
##   <chr>        <dbl>      <dbl>        <dbl>            <dbl> <chr>       <lgl>  
## 1 215934017~     2668      366.         4            91.5 mon        FALSE  
## 2 7f590fd6d~     4656      533.         5            107. tue        FALSE  
## 3 10f0f138e~     4563      522.         6            87.0 wed        FALSE  
## 4 7b55021a4~     4088      647.         7            92.4 fri        FALSE  
## 5 6694a79d1~     2188      419.         4            105. tue        FALSE  
## 6 d0358740d~     4171      220.         2            110. fri        FALSE  
## 7 944e568a0~     2907      465.         4            116. fri        FALSE  
## 8 95476c2ef~     5141      446.         4            111. wed        FALSE  
## 9 df235631a~     1696      106.         1            106. wed        FALSE  
## 10 ff6101402~    1901      576.         7            82.3 sat        FALSE  
## # ... with 9,990 more rows, and 2 more variables: status <chr>, review_score <dbl>
```

Replacing existing columns with `mutate()` +

Convert the column `property_id` from `character` into `factor`.

```
bookings %>%
  mutate(property_id = as.factor(property_id))

## # A tibble: 10,000 x 8
##   booker_id      property_id room_nights price_per_night checkin_day for_business status
##   <chr>          <fct>        <dbl>           <dbl> <chr>       <lgl>      <chr>
## 1 215934017ba98c~ 2668            4             91.5 mon        FALSE     cance~
## 2 7f590fd6d31824~ 4656            5            107. tue        FALSE     cance~
## 3 10f0f138e8bb10~ 4563            6             87.0 wed        FALSE     stayed
## 4 7b55021a4160dd~ 4088            7             92.4 fri        FALSE     stayed
## 5 6694a79d158c78~ 2188            4             105. tue        FALSE     stayed
## 6 d0358740d5f15e~ 4171            2             110. fri        FALSE     cance~
## 7 944e568a0b511b~ 2907            4             116. fri        FALSE     stayed
## 8 95476c2ef6bb9e~ 5141            4             111. wed        FALSE     cance~
## 9 df235631a4c281~ 1696            1             106. wed        FALSE     stayed
## 10 ff610140227d40~ 1901           7             82.3 sat        FALSE    cance~
## # ... with 9,990 more rows, and 1 more variable: review_score <dbl>
```

mutate(across(...))

Apply a transformation to **multiple columns**.

Select columns by name

Select columns by logical condition

Transform columns whose names end with "id" to factor variables.

```
bookings %>%
  mutate(across(ends_with("id"), as.factor))

## # A tibble: 10,000 x 8
##   booker_id     property_id room_nights price_per_night checkin_day for_business status
##   <fct>       <fct>           <dbl>            <dbl> <chr>      <lgl>      <chr>
## 1 215934017ba98c~ 2668             4            91.5 mon        FALSE    cance~
## 2 7f590fd6d31824~ 4656             5            107. tue        FALSE    cance~
## 3 10f0f138e8bb10~ 4563             6            87.0 wed        FALSE    stayed
## 4 7b55021a4160dd~ 4088             7            92.4 fri        FALSE    stayed
## 5 6694a79d158c78~ 2188             4            105. tue        FALSE    stayed
## 6 d0358740d5f15e~ 4171             2            110. fri        FALSE    cance~
## 7 944e568a0b511b~ 2907             4            116. fri        FALSE    stayed
## 8 95476c2ef6bb9e~ 5141             4            111. wed        FALSE    cance~
## 9 df235631a4c281~ 1696             1            106. wed        FALSE    stayed
## 10 ff610140227d40~ 1901            7            82.3 sat        FALSE    cance~
## # ... with 9,990 more rows, and 1 more variable: review_score <dbl>
```

mutate(across(...))

Apply a transformation to **multiple columns**.

Select columns by name

Select columns by logical condition

Transform all columns of type `character` to factor variables.

```
bookings %>%
  mutate(across(where(is.character), as.factor))
```

```
## # A tibble: 10,000 x 8
##   booker_id      property_id room_nights price_per_night checkin_day for_business status
##   <fct>          <dbl>        <dbl>           <dbl> <fct>       <lgl>       <fct>
## 1 215934017ba98c~     2668         4            91.5 mon        FALSE      cance~
## 2 7f590fd6d31824~     4656         5            107. tue        FALSE      cance~
## 3 10f0f138e8bb10~     4563         6            87.0 wed        FALSE      stayed
## 4 7b55021a4160dd~     4088         7            92.4 fri        FALSE      stayed
## 5 6694a79d158c78~     2188         4            105. tue        FALSE      stayed
## 6 d0358740d5f15e~     4171         2            110. fri        FALSE      cance~
## 7 944e568a0b511b~     2907         4            116. fri        FALSE      stayed
## 8 95476c2ef6bb9e~     5141         4            111. wed        FALSE      cance~
## 9 df235631a4c281~     1696         1            106. wed        FALSE      stayed
## 10 ff610140227d40~    1901         7            82.3 sat        FALSE      cance~
## # ... with 9,990 more rows, and 1 more variable: review_score <dbl>
```

Summarize many rows with `summarize()`



`summarize()` (or `summarise()`) performs some kind of aggregation and returns a summary table with fewer rows and removes all columns that are irrelevant to the calculation.



A single one-valued stat

Multiple n-valued stats

Multiple n-valued stats

What is the average review score over all bookings?

```
bookings %>%  
  summarize(review_score = mean(review_score, na.rm = TRUE))
```

```
## # A tibble: 1 x 1  
##   review_score  
##       <dbl>  
## 1         7.22
```

(one row, one column)

Summarize many rows with `summarize()`



`summarize()` (or `summarise()`) performs some kind of aggregation and returns a summary table with fewer rows and removes all columns that are irrelevant to the calculation.



A single one-valued stat

Multiple n-valued stats

Multiple n-valued stats

What is the total number of bookings, the number of bookings without review score, and the average review score over all bookings?

```
bookings %>%
  summarize(n = n(), # Total no. of bookings
            n_miss = sum(is.na(review_score)), # No. of bookings w/o review score
            review_score = mean(review_score, na.rm = TRUE)) # Avg. review score

## # A tibble: 1 x 3
##       n   n_miss review_score
##   <int>   <int>      <dbl>
## 1 10000     3817      7.22
```

(one row, multiple columns)

Summarize many rows with `summarize()`



`summarize()` (or `summarise()`) performs some kind of aggregation and returns a summary table with fewer rows and removes all columns that are irrelevant to the calculation.



A single one-valued stat

Multiple n-valued stats

Multiple n-valued stats

What is the price range over all bookings?

```
bookings %>%
  summarize(statistic = c("min", "max"), value = range(price_per_night))

## # A tibble: 2 x 2
##   statistic value
##   <chr>     <dbl>
## 1 min       39.4
## 2 max      273.
```

(multiple rows, multiple columns)

Grouping with `group_by()`

`group_by()` lets us perform operations for each *group* separately.

```
bookings %>%
  group_by(for_business) %>%
  summarize(n = n(), review_avg = mean(review_score, na.rm = TRUE))
```

```
## # A tibble: 2 x 3
##   for_business     n  review_avg
##   <lgl>        <int>     <dbl>
## 1 FALSE          6285      7.50
## 2 TRUE           3715      6.85
```

```
class(bookings)
```

```
## [1] "spec_tbl_df" "tbl_df"       "tbl"          "data.frame"
```

```
bookings %>% group_by(for_business) %>% class()
```

```
## [1] "grouped_df" "tbl_df"       "tbl"          "data.frame"
```

Use `ungroup()` to undo the grouping.

Group by multiple columns

```
# Average score by travel type and check-in day of the week
bookings %>%
  group_by(for_business, checkin_day) %>%
  summarize(mean_review = mean(review_score, na.rm = TRUE))
```

```
## # A tibble: 14 x 3
## # Groups:   for_business [2]
##   for_business checkin_day mean_review
##   <lgl>        <chr>          <dbl>
## 1 FALSE        fri            7.60
## 2 FALSE        mon            7.34
## 3 FALSE        sat            7.57
## 4 FALSE        sun            7.57
## 5 FALSE        thu            7.42
## 6 FALSE        tue            7.32
## 7 FALSE        wed            7.43
## 8 TRUE         fri            7.04
## 9 TRUE         mon            6.87
## 10 TRUE        sat            7.05
## 11 TRUE        sun            6.93
## 12 TRUE        thu            6.79
## 13 TRUE        tue            6.75
## 14 TRUE        wed            6.84
```

Count the number of rows per group with `count()`

Syntax:

```
bookings %>%
  count(x, y, ...)
```

...is a shortcut for...

```
bookings %>%
  group_by(x, y, ...) %>%
  summarize(n = n())
```

```
bookings %>%
  count(for_business, status)
```

```
## # A tibble: 4 x 3
##   for_business status     n
##   <lgl>        <chr>    <int>
## 1 FALSE        cancelled 1762
## 2 FALSE        stayed    4523
## 3 TRUE         cancelled  463
## 4 TRUE         stayed    3252
```

```
bookings %>%
  group_by(for_business, status) %>%
  summarize(n = n())
```

```
## # A tibble: 4 x 3
## # Groups:   for_business [2]
##   for_business status     n
##   <lgl>        <chr>    <int>
## 1 FALSE        cancelled 1762
## 2 FALSE        stayed    4523
## 3 TRUE         cancelled  463
## 4 TRUE         stayed    3252
```

Combining multiple data frames

Customer bookings data

What is the number of bookings per destination (city)?

To answer this question, we have to combine the two data frames `bookings` and `properties`.

`bookings` `properties`

```
bookings <- read_csv("../data/bookings.csv")  
  
## # A tibble: 10,000 x 8  
##   booker_id      property_id room_nights price_per_night checkin_day for_business status  
##   <chr>          <dbl>        <dbl>            <dbl> <chr>       <lgl>      <chr>  
## 1 215934017ba98c~     2668         4            91.5 mon      FALSE    cance~  
## 2 7f590fd6d31824~     4656         5            107. tue      FALSE    cance~  
## 3 10f0f138e8bb10~     4563         6            87.0 wed      FALSE    stayed  
## 4 7b55021a4160dd~     4088         7            92.4 fri      FALSE    stayed  
## 5 6694a79d158c78~     2188         4            105. tue      FALSE    stayed  
## 6 d0358740d5f15e~     4171         2            110. fri      FALSE    cance~  
## 7 944e568a0b511b~     2907         4            116. fri      FALSE    stayed  
## 8 95476c2ef6bb9e~     5141         4            111. wed      FALSE    cance~  
## 9 df235631a4c281~     1696         1            106. wed      FALSE    stayed  
## 10 ff610140227d40~    1901         7            82.3 sat      FALSE    cance~  
## # ... with 9,990 more rows, and 1 more variable: review_score <dbl>
```

Customer bookings data

What is the number of bookings per destination (city)?

To answer this question, we have to combine the two data frames `bookings` and `properties`.

`bookings` `properties`

```
properties <- read_csv("../data/properties.csv")
properties

## # A tibble: 4,178 x 5
##   property_id destination property_type nr_rooms facilities
##       <dbl> <chr>        <chr>          <dbl> <chr>
## 1       2668 Brisbane     Hotel           32 airport shuttle,free wifi,garden,breakf~
## 2       4656 Brisbane     Hotel           39 on-site restaurant,pool,airport shuttle~
## 3       4563 Brisbane     Apartment        9 laundry
## 4       4088 Brisbane     Apartment        9 kitchen,laundry,free wifi
## 5       2188 Brisbane     Apartment        4 parking,kitchen,bbq,free wifi,game cons~
## 6       4171 Brisbane     Apartment        5 kitchen,pool,laundry,parking,free wifi,~
## 7       2907 Brisbane     Hotel            22 airport shuttle,on-site restaurant,brea~
## 8       5141 Brisbane     Hotel            20 breakfast,free wifi,on-site restaurant,~
## 9       1696 Brisbane     Apartment        5 free wifi,laundry,pool,game console,par~
## 10      1901 Brisbane     Apartment        11 free wifi,bbq,laundry,breakfast,pool,pa~
## # ... with 4,168 more rows
```

Joining data frames

Data

Basic syntax

Inner

Left

Left 2

Right

Full

Semi

Anti

Consider the following two toy data frames `x` and `y`:

```
(x <- tibble(id = c(1L, 2L, 3L), x = c("x1", "x2", "x3")))
```

```
## # A tibble: 3 x 2
##       id     x
##   <int> <chr>
## 1     1     x1
## 2     2     x2
## 3     3     x3
```

```
(y <- tibble(id = c(1L, 2L, 4L), y = c("y1", "y2", "y4")))
```

```
## # A tibble: 3 x 2
##       id     y
##   <int> <chr>
## 1     1     y1
## 2     2     y2
## 3     4     y4
```

X y

1	x1
2	x2
3	x3

1	y1
2	y2
4	y4

Source of the figures on this and the following slides: <https://github.com/gadenbuie/tidyexplain>

Joining data frames

Data Basic syntax Inner Left Left 2 Right Full Semi Anti

Basic syntax:

```
**_join(df1, df2, by = "<ID>")
```

Joining data frames

[Data](#)[Basic syntax](#)[Inner](#)[Left](#)[Left 2](#)[Right](#)[Full](#)[Semi](#)[Anti](#)

Join two data frames `x` and `y`. The result is a data frame containing all rows from `x` with matching values in `y` for column `id` and all columns from `x` and `y`.

```
inner_join(x, y, by = "id")
```

```
## # A tibble: 2 x 3
##       id   x     y
##   <int> <chr> <chr>
## 1     1   x1    y1
## 2     2   x2    y2
```

x	y
1	x1
2	x2
3	x3

x	y
1	y1
2	y2
4	y4

Joining data frames

Data Basic syntax Inner **Left** Left 2 Right Full Semi Anti

Join two data frames `x` and `y`. The result is a data frame containing all rows from `x` and all columns from `x` and `y`. In rows where there are no matching values in `y` for column `id`, the values of the columns that are present only in `y` are set to `NA`.

```
left_join(x, y, by = "id")  
  
## # A tibble: 3 x 3  
##       id   x     y  
##   <int> <chr> <chr>  
## 1     1   x1    y1  
## 2     2   x2    y2  
## 3     3   x3    <NA>
```

x	y
1	x1
2	x2
3	x3

x	y
1	y1
2	y2
4	y4

Joining data frames

Data Basic syntax Inner Left **Left 2** Right Full Semi Anti

In case of multiple matches, **all** combinations of the matches are returned:

```
(y_extra <- bind_rows(y, tibble(id = 2L, y = "y5")))
```

```
## # A tibble: 4 x 2
##       id     y
##   <int> <chr>
## 1     1     y1
## 2     2     y2
## 3     4     y4
## 4     2     y5
```

x y

1	x1	1	y1
2	x2	2	y2
3	x3	4	y4

```
left_join(x, y_extra, by = "id")
```

```
## # A tibble: 4 x 3
##       id     x     y
##   <int> <chr> <chr>
## 1     1     x1    y1
## 2     2     x2    y2
## 3     2     x2    y5
## 4     3     x3    <NA>
```

Joining data frames

Data Basic syntax Inner Left Left 2 **Right** Full Semi Anti

Join two data frames `x` and `y`. The result is a data frame containing all rows from `y` and all columns from `x` and `y`. In rows where there are no matching values in `x` for column `id`, the values of the columns that are present only in `x` are set to `NA`.

```
right_join(x, y, by = "id")
```

```
## # A tibble: 3 x 3
##       id   x     y
##   <int> <chr> <chr>
## 1     1   x1    y1
## 2     2   x2    y2
## 3     4 <NA>    y4
```

x	y
1	x1
2	x2
3	x3

x	y
1	y1
2	y2
4	y4

Joining data frames

Data Basic syntax Inner Left Left 2 Right **Full** Semi Anti

Join two data frames `x` and `y`. The result is a data frame containing all rows and all columns from `x` and `y`. In rows where there are no matching values for column `id`, the values of the columns from the other data frame are set to `NA`.

```
full_join(x, y, by = "id")  
  
## # A tibble: 4 x 3  
##       id   x     y  
##   <int> <chr> <chr>  
## 1     1   x1    y1  
## 2     2   x2    y2  
## 3     3   x3    <NA>  
## 4     4   <NA>  y4
```

x	y
1	x1
2	x2
3	x3
4	y4
1	y1
2	y2

Joining data frames

[Data](#)[Basic syntax](#)[Inner](#)[Left](#)[Left 2](#)[Right](#)[Full](#)[Semi](#)[Anti](#)

Return all rows and columns from `x` with matching rows in `y`.

```
semi_join(x, y, by = "id")  
  
## # A tibble: 2 x 2  
##       id   x  
##   <int> <chr>  
## 1     1   x1  
## 2     2   x2
```

x	y
1	x1
2	x2
3	x3

x	y
1	y1
2	y2

Joining data frames

Data

Basic syntax

Inner

Left

Left 2

Right

Full

Semi

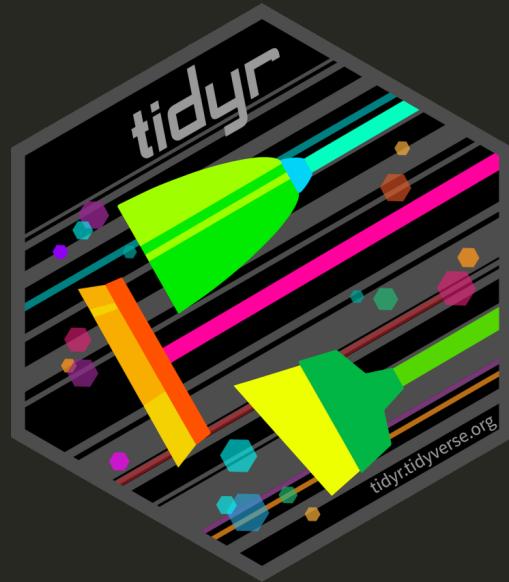
Anti

Return all rows and columns from `x` **without** matching rows in `y`.

```
anti_join(x, y, by = "id")  
  
## # A tibble: 1 x 2  
##       id   x  
##   <int> <chr>  
## 1     3  x3
```

x **y**

1	x1	1	y1
2	x2	2	y2
3	x3	4	y4



A

B

C

D

E

F

G

H

I

J

Gesamtübersicht: Daten zum Kriminalitätsatlas (PKS 2019)

Fallzahlen der Bezirke und Bezirksregionen

Hinweis: Aufgrund einer zum 1. Januar 2019 durchgeführten Anpassung beim Gebietszuschnitt im Bereich Allende-Viertel (Bezirk Treptow-Köpenick) sind bei den gelb markier

LOR-Schlüssel (Bezirksregion)	Bezeichnung (Bezirksregion)	Straftaten -insgesamt-	Raub	Straßenraub, Handtaschen- raub	Körper- verletzungen -insgesamt-	Gefährl. und schwere Körper- verletzung	Freiheits- beraubung,	Nötigung, Bedrohung, Nachstellung	Diebstahl -insgesamt-	Diebstahl von Kraftwagen
010000	Mitte	84.357	707	407	7.595	1.951	2.157	35.601	401	
010111	Tiergarten Süd	5.009	60	35	365	92	128	2.271	15	
010112	Regierungsviertel	7.950	42	20	554	136	152	3.692	13	
010113	Alexanderplatz	22.974	173	102	1.966	500	420	11.233	63	
010114	Brunnenstraße Süd	4.252	40	29	268	64	79	1.859	39	
010221	Moabit West	7.197	66	29	685	210	202	2.107	47	
010222	Moabit Ost	9.626	48	29	652	150	231	3.672	27	
010331	Osloer Straße	6.011	61	30	727	168	214	2.474	46	
010332	Brunnenstraße Nord	6.600	62	42	692	201	209	2.537	58	
010441	Parkviertel	6.413	64	45	699	159	218	2.779	50	
010442	Wedding Zentrum	8.090	86	42	968	264	299	2.883	43	
019900	Bezirk (Mi), nicht zuzuordnen	235	5	4	19	7	5	94	0	
020000	Friedrichshain-Kreuzberg	60.290	820	579	5.006	1.752	1.237	25.650	387	
020101	Südliche Friedrichstadt	8.923	125	70	716	200	255	3.973	43	
020202	Tempelhofer Vorstadt	12.527	119	82	872	283	271	5.797	113	
020303	nördliche Luisenstadt	5.484	94	67	524	215	99	2.509	27	
020304	südliche Luisenstadt	6.653	182	150	530	251	125	2.343	37	
020405	Karl-Marx-Allee-Nord	2.551	38	26	225	58	61	1.260	50	
020407	Karl-Marx-Allee-Süd	8.741	98							
020506	Frankfurter Allee Nord	4.229	30							
020508	Frankfurter Allee Süd FK	11.092	132							
029900	Bezirk (Fh-Kb), nicht zuzuordnen	90	2							
030000	Pankow	36.710	284							
030101	Buch	1.829	9							
030202	Blankenfelde/Niederschönhausen	1.633	7							
030203	Dahme	240	2							

Tidy and untidy data

Data source

Data come in different shapes...

table1 table2 table3 table4a & table4b

```
## # A tibble: 6 x 4
##   country     year   cases population
##   <chr>     <int>   <int>      <int>
## 1 Afghanistan 1999     745 19987071
## 2 Afghanistan 2000    2666 20595360
## 3 Brazil      1999  37737 172006362
## 4 Brazil      2000  80488 174504898
## 5 China       1999 212258 1272915272
## 6 China       2000 213766 1280428583
```

Data come in different shapes...

table1

table2

table3

table4a & table4b

```
## # A tibble: 12 × 4
##   country     year type     count
##   <chr>       <int> <chr>     <int>
## 1 Afghanistan 1999 cases      745
## 2 Afghanistan 1999 population 19987071
## 3 Afghanistan 2000 cases      2666
## 4 Afghanistan 2000 population 20595360
## 5 Brazil      1999 cases      37737
## 6 Brazil      1999 population 172006362
## 7 Brazil      2000 cases      80488
## 8 Brazil      2000 population 174504898
## 9 China       1999 cases      212258
## 10 China      1999 population 1272915272
## 11 China      2000 cases      213766
## 12 China      2000 population 1280428583
```

Data come in different shapes...

table1

table2

table3

table4a & table4b

```
## # A tibble: 6 x 3
##   country     year    rate
## * <chr>       <int> <chr>
## 1 Afghanistan 1999  745/19987071
## 2 Afghanistan 2000  2666/20595360
## 3 Brazil      1999  37737/172006362
## 4 Brazil      2000  80488/174504898
## 5 China       1999  212258/1272915272
## 6 China       2000  213766/1280428583
```

Data come in different shapes...

table1 table2 table3 **table4a & table4b**

```
## # A tibble: 3 x 3
##   country    `1999` `2000`
## * <chr>        <int>  <int>
## 1 Afghanistan     745    2666
## 2 Brazil          37737   80488
## 3 China           212258  213766

## # A tibble: 3 x 3
##   country    `1999`    `2000`
## * <chr>        <int>     <int>
## 1 Afghanistan 19987071 20595360
## 2 Brazil      172006362 174504898
## 3 China       1272915272 1280428583
```

Reshape data frames with `tidyverse`

The `tidyverse` inherits its name from the term **tidy data**. Tidy data refers to a specific standardized dataset structure.

Characteristics of tidy data:

1. Each **variable** must have its own column.
2. Each **observation** must have its own row.
3. Each **value** must have its own cell.

Many of the tidyverse functions require a *tidy* data frame input. The `tidyverse` package contains functions to reshape "messy" into tidy data frames.

Further reading:

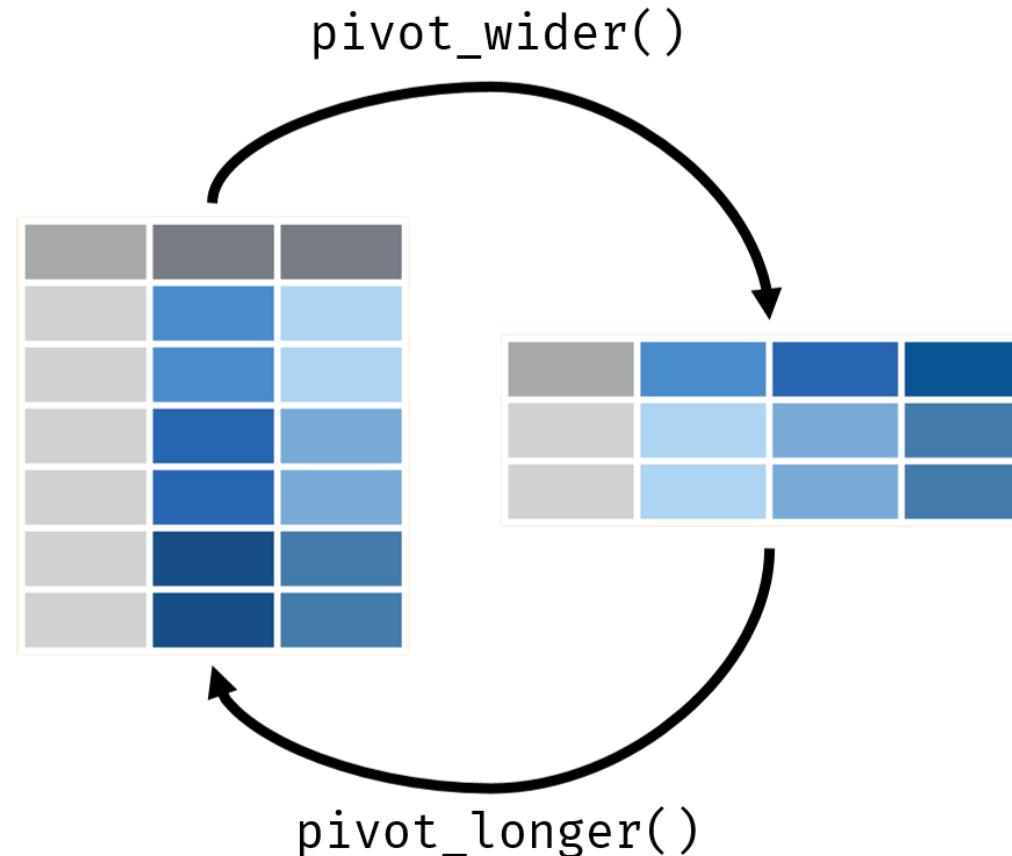
- [R for Data Science: Tidy data](#)
- Function overview: RStudio's [Data import cheat sheet](#)

wide				long		
id	x	y	z	id	key	val
1	a	c	e	1	x	a
2	b	d	f	2	x	b
				1	y	c
				2	y	d
				1	z	e
				2	z	f

`pivot_longer()` and `pivot_wider()`

`tidyrr`'s two main functions are `pivot_longer()` and `pivot_wider()`.

- `pivot_longer()` takes multiple columns and collapses them into **key-value pairs**.
- `pivot_wider()` takes two (or more) columns (a key-value pair) and spreads them into **multiple columns**.



pivot_longer() and pivot_wider()

```
(wide <- tibble(name = c("Alex", "Ben", "Cedric"), DataSciR = c(2.0, 1.7, 1.0), VisAnalytics = c(3.3, 1.3, 2.0)))  
  
## # A tibble: 3 x 3  
##   name    DataSciR VisAnalytics  
##   <chr>     <dbl>      <dbl>  
## 1 Alex        2          3.3  
## 2 Ben        1.7         1.3  
## 3 Cedric     1          2
```

This data frame is in **wide** format: there are 3 variables (name of student, course name, grade), but only the student's name has its own column.

Use `pivot_longer()` to collapse the two course columns into key-value pairs `course` and `grade`.

```
tidy <- wide %>%  
  pivot_longer(cols = -name,  
               names_to = "course", values_to = "grade")  
tidy  
  
## # A tibble: 6 x 3  
##   name    course      grade  
##   <chr>   <chr>      <dbl>  
## 1 Alex    DataSciR     2  
## 2 Alex    VisAnalytics 3.3  
## 3 Ben     DataSciR     1.7  
## 4 Ben     VisAnalytics 1.3  
## 5 Cedric  DataSciR     1  
## 6 Cedric  VisAnalytics 2
```

Use `pivot_wider()` to "unpivot" a data frame. It is useful when there are variables that form rows instead of columns.

```
tidy %>%  
  pivot_wider(names_from = course,  
             values_from = grade)  
  
## # A tibble: 3 x 3  
##   name    DataSciR VisAnalytics  
##   <chr>     <dbl>      <dbl>  
## 1 Alex        2          3.3  
## 2 Ben        1.7         1.3  
## 3 Cedric     1          2
```

Reshaping from "long" to "wide"

[Long data](#)[Wide data](#)[Code](#)[Wide data with correct order](#)[Undo](#)

Compute the number of bookings per city and day:

```
day_order <- c("mon", "tue", "wed", "thu", "fri", "sat", "sun")
df <- bookings %>% inner_join(properties, by = "property_id")
checkin_count <- df %>%
  count(destination, checkin_day) %>%
  mutate(checkin_day = factor(checkin_day, levels = day_order))
checkin_count
```

```
## # A tibble: 21 x 3
##   destination checkin_day     n
##   <chr>        <fct>      <int>
## 1 Amsterdam    fri        1074
## 2 Amsterdam    mon         517
## 3 Amsterdam    sat         889
## 4 Amsterdam    sun         813
## 5 Amsterdam    thu         667
## 6 Amsterdam    tue         498
## 7 Amsterdam    wed         542
## 8 Brisbane     fri        162
## 9 Brisbane     mon         133
## 10 Brisbane    sat         114
## # ... with 11 more rows
```

Reshaping from "long" to "wide"

Long data

Wide data

Code

Wide data with correct order

Undo

⌚ "How can we create the following table from `checkin_count`?"

```
## # A tibble: 3 x 8
##   destination   mon   tue   wed   thu   fri   sat   sun
##   <chr>       <int> <int> <int> <int> <int> <int> <int>
## 1 Amsterdam     517    498    542    667   1074    889    813
## 2 Brisbane      133    148    128    162    162    114    153
## 3 Tokyo         718    655    560    718    451    322    576
```

Reshaping from "long" to "wide"

Long data

Wide data

Code

Wide data with correct order

Undo

```
checkin_count %>%  
  pivot_wider(names_from = checkin_day, values_from = n)
```

```
## # A tibble: 3 x 8  
##   destination   fri   mon   sat   sun   thu   tue   wed  
##   <chr>       <int> <int> <int> <int> <int> <int>  
## 1 Amsterdam     1074    517    889    813    667    498    542  
## 2 Brisbane      162     133    114    153    162    148    128  
## 3 Tokyo         451     718    322    576    718    655    560
```

`pivot_wider()` creates the new columns in the order the keys appear in the data. Hence, we can sort the rows by `checkin_day` to obtain a data frame with the day-of-week columns in the correct order.

Reshaping from "long" to "wide"

Long data

Wide data

Code

Wide data with correct order

Undo

Arrange by `checkin_day` before reshaping the data frame into wide format:

```
checkin_count %>%
  arrange(checkin_day) %>%
  pivot_wider(names_from = checkin_day, values_from = n)
```

```
## # A tibble: 3 x 8
##   destination   mon   tue   wed   thu   fri   sat   sun
##   <chr>       <int> <int> <int> <int> <int> <int>
## 1 Amsterdam     517    498    542    667   1074    889    813
## 2 Brisbane      133    148    128    162    162    114    153
## 3 Tokyo         718    655    560    718    451    322    576
```

Reshaping from "long" to "wide"

Long data

Wide data

Code

Wide data with correct order

Undo

Bring the data back into long format:

```
checkin_count %>%
  pivot_wider(names_from = checkin_day,
              values_from = n) %>%
  pivot_longer(cols = c(mon, tue, wed, thu,
                       fri, sat, sun),
                names_to = "checkin_day",
                values_to = "n")
```

```
## # A tibble: 21 x 3
##   destination checkin_day     n
##   <chr>        <chr>     <int>
## 1 Amsterdam    mon       517
## 2 Amsterdam    tue       498
## 3 Amsterdam    wed       542
## 4 Amsterdam    thu       667
## 5 Amsterdam    fri      1074
## 6 Amsterdam    sat       889
## 7 Amsterdam    sun       813
## 8 Brisbane     mon      133
## 9 Brisbane     tue      148
## 10 Brisbane    wed      128
## # ... with 11 more rows
```

Alternative variables selection:

```
checkin_count %>%
  pivot_wider(names_from = checkin_day,
              values_from = n) %>%
  pivot_longer(cols = -destination,
                names_to = "checkin_day",
                values_to = "n")
```

```
## # A tibble: 21 x 3
##   destination checkin_day     n
##   <chr>        <chr>     <int>
## 1 Amsterdam    fri       1074
## 2 Amsterdam    mon        517
## 3 Amsterdam    sat        889
## 4 Amsterdam    sun        813
## 5 Amsterdam    thu        667
## 6 Amsterdam    tue        498
## 7 Amsterdam    wed        542
## 8 Brisbane     fri       162
## 9 Brisbane     mon      133
## 10 Brisbane    sat       114
## # ... with 11 more rows
```

separate()

```
properties %>% head()
```

```
## # A tibble: 6 x 5
##   property_id destination property_type nr_rooms facilities
##       <dbl>      <chr>        <chr>          <dbl>    <chr>
## 1       2668 Brisbane     Hotel           32 airport shuttle,free wifi,garden,breakfa~
## 2       4656 Brisbane     Hotel           39 on-site restaurant,pool,airport shuttle,~
## 3       4563 Brisbane   Apartment         9 laundry
## 4       4088 Brisbane   Apartment         9 kitchen,laundry,free wifi
## 5       2188 Brisbane   Apartment         4 parking,kitchen,bbq,free wifi,game conso~
## 6       4171 Brisbane   Apartment         5 kitchen,pool,laundry,parking,free wifi,g~
```

Have a look at the `facilities` column. It indicates the availability of various facilities in the accommodation.

separate()

`separate()` splits up two or more variables that are clumped together in one column.

```
properties %>%
  # Split `facilities` by `,` into multiple columns.
  separate(facilities, into = paste0("facility_", 1:9), sep = ",") %>%
  head(5)

## Warning: Expected 9 pieces. Missing pieces filled with `NA` in 4091 rows [1, 2, 3, 4, 5,
## 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].

## # A tibble: 5 x 13
##   property_id destination property_type nr_rooms facility_1      facility_2 facility_3
##       <dbl> <chr>        <chr>          <dbl> <chr>        <chr>        <chr>
## 1       2668 Brisbane     Hotel           32  airport shuttle free wifi  garden
## 2       4656 Brisbane     Hotel           39  on-site restaur~ pool      airport shut~
## 3       4563 Brisbane     Apartment       9   laundry      <NA>      <NA>
## 4       4088 Brisbane     Apartment       9   kitchen      laundry    free wifi
## 5       2188 Brisbane     Apartment       4   parking      kitchen   bbq
## # ... with 6 more variables: facility_4 <chr>, facility_5 <chr>, facility_6 <chr>,
## #   facility_7 <chr>, facility_8 <chr>, facility_9 <chr>
```

We get a warning because the number of facilities differ across properties. The maximum number of facilities is 9. If a property has less than 9 facilities, the remaining columns are filled with `NA`.

✓ `unite()` is the inverse function of `separate()` and combines multiple variables into one. For example, this operation could be useful for combining day, month and year columns into one date column.

Nesting tables

Common data structures are **hierarchical**, e.g. patient-centric data with repeat observations.

Nesting allows to store collapsed data frames and simplifies data management.

First, reshape `properties` into a tidy format:

```
tp <- properties %>%
  separate(facilities, into = paste0("facility_", 1:9), sep = ",") %>%
  pivot_longer(cols = starts_with("facility_"), names_to = "facility_nr", values_to = "facility")
```

```
## Warning: Expected 9 pieces. Missing pieces filled with `NA` in 4091 rows [1, 2, 3, 4, 5,
## 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].
```

```
head(tp)
```

```
## # A tibble: 6 x 6
##   property_id destination property_type nr_rooms facility_nr facility
##       <dbl> <chr>      <chr>           <dbl> <chr>      <chr>
## 1       2668 Brisbane Hotel            32 facility_1 airport shuttle
## 2       2668 Brisbane Hotel            32 facility_2 free wifi
## 3       2668 Brisbane Hotel            32 facility_3 garden
## 4       2668 Brisbane Hotel            32 facility_4 breakfast
## 5       2668 Brisbane Hotel            32 facility_5 pool
## 6       2668 Brisbane Hotel            32 facility_6 on-site restaurant
```

Now, the data frame contains a lot of duplicate information: 9x destination, property type and number of rooms for each property id.

Nesting tables

Use `nest()` to create a nested data frame. The nested data frame contains a column `facilities` which is a **list of data frames**:

```
nested_tp <- tp %>%
  nest(facilities = c(facility_nr, facility))
nested_tp

## # A tibble: 4,178 x 5
##   property_id destination property_type nr_rooms facilities
##       <dbl>      <chr>        <chr>          <dbl>     <list>
## 1       2668 Brisbane    Hotel           32 <tibble [9 x 2]>
## 2       4656 Brisbane    Hotel           39 <tibble [9 x 2]>
## 3       4563 Brisbane Apartment        9 <tibble [9 x 2]>
## 4       4088 Brisbane Apartment        9 <tibble [9 x 2]>
## 5       2188 Brisbane Apartment        4 <tibble [9 x 2]>
## 6       4171 Brisbane Apartment        5 <tibble [9 x 2]>
## 7       2907 Brisbane    Hotel           22 <tibble [9 x 2]>
## 8       5141 Brisbane    Hotel           20 <tibble [9 x 2]>
## 9       1696 Brisbane Apartment        5 <tibble [9 x 2]>
## 10      1901 Brisbane Apartment        11 <tibble [9 x 2]>
## # ... with 4,168 more rows
```

Use `unnest()` to expand a list column, such that each element of the list becomes a row.

drop_na()

Some of the rows in the nested `facilities` column contain `NA`'s. How can we remove them?

Access nested data frame

drop_na() on list columns

drop_na() before nesting

```
nested_tp$facilities[1]
```

```
## [[1]]
## # A tibble: 9 x 2
##   facility_nr facility
##   <chr>        <chr>
## 1 facility_1  airport shuttle
## 2 facility_2  free wifi
## 3 facility_3  garden
## 4 facility_4  breakfast
## 5 facility_5  pool
## 6 facility_6  on-site restaurant
## 7 facility_7  <NA>
## 8 facility_8  <NA>
## 9 facility_9  <NA>
```

drop_na()

Some of the rows in the nested `facilities` column contain `NA`'s. How can we remove them?

Access nested data frame

drop_na() on list columns

drop_na() before nesting

The function `drop_na()` removes all rows with at least one missing value.

```
nested_tp_wo_na <- nested_tp %>%
  drop_na(facilities)
# Did it work?
nested_tp_wo_na$facilities[1]
```

```
## [[1]]
## # A tibble: 9 x 2
##   facility_nr facility
##   <chr>        <chr>
## 1 facility_1  airport shuttle
## 2 facility_2  free wifi
## 3 facility_3  garden
## 4 facility_4  breakfast
## 5 facility_5  pool
## 6 facility_6  on-site restaurant
## 7 facility_7  <NA>
## 8 facility_8  <NA>
## 9 facility_9  <NA>
```

drop_na()

Some of the rows in the nested `facilities` column contain `NA`'s. How can we remove them?

Access nested data frame

drop_na() on list columns

drop_na() before nesting

`dplyr/tidyr` functions don't work recursive. Thus, we apply `drop_na()` before nesting.

```
nested_without_missing <- tp %>%  
  drop_na() %>%  
  nest(facilities = c(facility_nr, facility))  
nested_without_missing$facilities[[1]]
```

```
## # A tibble: 6 x 2  
##   facility_nr facility  
##   <chr>        <chr>  
## 1 facility_1  airport shuttle  
## 2 facility_2  free wifi  
## 3 facility_3  garden  
## 4 facility_4  breakfast  
## 5 facility_5  pool  
## 6 facility_6  on-site restaurant
```



Data import

The first step of every data analysis project is to import one or more datasets. The tidyverse provides seven packages for the import of various data formats:

- `readr`: flat files (.csv, .tsv, ...)
- `DBI`: databases (SQLite, MySQL, PostgreSQL, MonetDB, ...)
- `haven`: foreign statistical formats (.sas, .sav, .dta)
- `jsonlite`: json files
- `readxl`: Excel files (.xls, .xlsx)
- `rvest`: websites (.html)
- `xml2`: xml files



Function reference: RStudio's [Data import cheat sheet](#)

Data import: bookings.csv and properties.csv

bookings.csv

properties.csv

```
# library(readr) # readr is a core tidyverse package and hence doesn't need to be loaded separately.  
bookings <- read_csv("../data/bookings.csv")
```

```
##  
## -- Column specification -----  
## cols(  
##   booker_id = col_character(),  
##   property_id = col_double(),  
##   room_nights = col_double(),  
##   price_per_night = col_double(),  
##   checkin_day = col_character(),  
##   for_business = col_logical(),  
##   status = col_character(),  
##   review_score = col_double()  
## )
```

```
head(bookings, 3)
```

```
## # A tibble: 3 x 8  
##   booker_id      property_id room_nights price_per_night checkin_day for_business status  
##   <chr>          <dbl>        <dbl>           <dbl> <chr>       <lgl>        <chr>  
## 1 215934017ba98c0~     2668         4            91.5 mon        FALSE       cance~  
## 2 7f590fd6d318248~     4656         5            107. tue        FALSE       cance~  
## 3 10f0f138e8bb101~     4563         6            87.0 wed        FALSE       stayed  
## # ... with 1 more variable: review_score <dbl>
```

Data import: bookings.csv and properties.csv

bookings.csv

properties.csv

```
properties <- read_csv("../data/properties.csv")
properties

## # A tibble: 4,178 x 5
##   property_id destination property_type nr_rooms facilities
##       <dbl>     <chr>      <chr>          <dbl>    <chr>
## 1        2668 Brisbane   Hotel            32 airport shuttle,free wifi,garden,breakf~
## 2        4656 Brisbane   Hotel            39 on-site restaurant,pool,airport shuttle~
## 3        4563 Brisbane   Apartment         9 laundry
## 4        4088 Brisbane   Apartment         9 kitchen,laundry,free wifi
## 5        2188 Brisbane   Apartment         4 parking,kitchen,bbq,free wifi,game cons~
## 6        4171 Brisbane   Apartment         5 kitchen,pool,laundry,parking,free wifi,~
## 7        2907 Brisbane   Hotel             22 airport shuttle,on-site restaurant,brea~
## 8        5141 Brisbane   Hotel             20 breakfast,free wifi,on-site restaurant,~
## 9        1696 Brisbane   Apartment         5 free wifi,laundry,pool,game console,par~
## 10       1901 Brisbane   Apartment         11 free wifi,bbq,laundry,breakfast,pool,pa~
## # ... with 4,168 more rows
```

Comparison with Base R import functions

Base `R` already has functions for loading flat files, e.g. `read.csv()`, `read.delim()`.

Advantages of the Tidyverse implementations include:

- higher speed
- ~~characters are not coerced to factors by default (see `stringsAsFactors` argument)~~ (This is not the case anymore since `R` version 4.0)
- generates tibbles instead of data frames



Tibbles

A tibble (class `tbl_df`) is "a modern reimagining" of the data frame.

Advantages over traditional data frames include:

- improved print method that shows...
 - ...only the first 10 rows
 - ...all the columns that fit on screen + names of the remaining ones
 - ...column types
- more consistent subsetting
- less type coercion
- prohibits partial matching



Use `as_tibble()` to convert a data frame to a tibble.

Tibble vs. data frame

Print data.frame

Print tbl_df

Line width

Build function

Rowwise construction: tribble()

```
class(iris)
```

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

```
iris
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
## 1	5.1	3.5	1.4	0.2	setosa
## 2	4.9	3.0	1.4	0.2	setosa
## 3	4.7	3.2	1.3	0.2	setosa
## 4	4.6	3.1	1.5	0.2	setosa
## 5	5.0	3.6	1.4	0.2	setosa
## 6	5.4	3.9	1.7	0.4	setosa
## 7	4.6	3.4	1.4	0.3	setosa
## 8	5.0	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
## 11	5.4	3.7	1.5	0.2	setosa
## 12	4.8	3.4	1.6	0.2	setosa
## 13	4.8	3.0	1.4	0.1	setosa
## 14	4.3	3.0	1.1	0.1	setosa
## 15	5.8	4.0	1.2	0.2	setosa
## 16	5.7	4.4	1.5	0.4	setosa
## 17	5.4	3.9	1.3	0.4	setosa
## 18	5.1	3.5	1.4	0.3	setosa

Tibble vs. data frame

Print data.frame

Print `tbl_df`

Line width

Build function

Rowwise construction: `tribble()`

```
as_tibble(iris)
```

```
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##       <dbl>      <dbl>      <dbl>      <dbl> <fct>
## 1         5.1        3.5       1.4       0.2  setosa
## 2         4.9        3.0       1.4       0.2  setosa
## 3         4.7        3.2       1.3       0.2  setosa
## 4         4.6        3.1       1.5       0.2  setosa
## 5         5.0        3.6       1.4       0.2  setosa
## 6         5.4        3.9       1.7       0.4  setosa
## 7         4.6        3.4       1.4       0.3  setosa
## 8         5.0        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
```

Tibble vs. data frame

[Print data.frame](#)[Print tbl_df](#)[Line width](#)[Build function](#)[Rowwise construction: tribble\(\)](#)

Tibbles adjust to the available line width!

```
as_tibble(iris)

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

Tibble vs. data frame

Print data.frame

Print tbl_df

Line width

Build function

Rowwise construction: tribble()

```
df <- data.frame(  
  `bad name` = 1:3, # syntactically invalid name because of SPACE  
  x = rep(letters[1:2], length.out = 3)  
)  
str(df)
```

```
## 'data.frame':   3 obs. of  2 variables:  
##   $ bad.name: int  1 2 3  
##   $ x        : chr  "a" "b" "a"
```

Tibbles don't coerce character vectors to factors.

```
ti <- tibble(  
  `bad name` = 1:3, # no "auto repair" of invalid column name  
  x = rep(letters[1:2], length.out = 3)  
)  
str(ti)
```

```
## tibble [3 x 2] (S3:tbl_df/tbl/data.frame)  
##   $ bad.name: int [1:3] 1 2 3  
##   $ x       : chr [1:3] "a" "b" "a"
```

Tibble vs. data frame

[Print data.frame](#)[Print tbl_df](#)[Line width](#)[Build function](#)[Rowwise construction: tribble\(\)](#)

Tibbles can also be created in a spreadsheet-like rowwise fashion:

```
tribble(  
  ~x, ~y,  
  1, "b",  
  2, "a"  
)
```

```
## # A tibble: 2 x 2  
##       x     y  
##   <dbl> <chr>  
## 1     1     b  
## 2     2     a
```

Use the tilde operator (~) to signify column names.

Very useful tidyverse packages we did not cover today



For an introduction, see the following chapters from Hadley Wickham and Garrett Grolemund. ["R for Data Science"](#). O'Reilly, 2017.

- [Strings with stringr](#)
- [Factors withforcats](#)
- [Dates and Times with lubridate](#)

Session info

```
## setting value
## version R version 4.0.4 (2021-02-15)
## os       Windows 10 x64
## system  x86_64, mingw32
## ui       RTerm
## language (EN)
## collate English_United States.1252
## ctype   English_United States.1252
## tz      Europe/Berlin
## date   2021-03-22
```

package	version	date	source
dplyr	1.0.5	2021-03-05	CRAN (R 4.0.4)
forcats	0.5.1	2021-01-27	CRAN (R 4.0.3)
ggplot2	3.3.3	2020-12-30	CRAN (R 4.0.3)
purrr	0.3.4	2020-04-17	CRAN (R 4.0.2)
readr	1.4.0	2020-10-05	CRAN (R 4.0.3)
stringr	1.4.0	2019-02-10	CRAN (R 4.0.2)
tibble	3.1.0	2021-02-25	CRAN (R 4.0.3)
tidyverse	1.1.3	2021-03-03	CRAN (R 4.0.4)
tidyverse	1.3.0	2019-11-21	CRAN (R 4.0.2)



Thank you! Questions?