Data Transformation

Altering rows and columns

Daniela Palleschi

2023-05-15

Inhaltsverzeichnis

Wiederholung

Letze Woche haben wir...

- gelernt, wie man einen neuen Datensatz in Augenschein nimmt
- gelernt, wie man verschiedene Datentypen importiert
- gelernt, wie man Daten von Hand eingibt
- einen neuen Datensatz visualisiert

Heutige Ziele

Today we will...

- learn how to wrangle data using the `dplyr` package from the `tidyverse`
- learn to use uses the `pipe` (`%>%`) to feed the result of one function into another function
- learn about functions that operate on rows
- learn about functions that operate on columns
- learn how to combine `dplyr` functions with plots from `ggplot2`

Lust auf mehr?

- Ch. 4 in (**wickham_r_nodate?**)
- Ch. 9 in (**nordmann_applied_2022?**)

1 Pre-requisites

1. Fresh Quarto document
    - create a new Quarto document for today's class
        - File > New Document > Quarto Document, named something like `04-wrangling`
    - set up the YAML: title, your name, add a `toc`

```
title: "Data wrangling"
subtitle: "Transforming data"
author: "Your name here"
```

```
lang: de
date: "`r Sys.Date()`"
format:
  html:
    toc: true
```

2. Packages
    - today's packages are:
        - `tidyverse`: for wrangling (`dplyr`) and plotting (`ggplot2`)

```
pacman::p_load("tidyverse")
```

3. Dataset
    - save the dataset from Moodle in your `daten` folder:
        - `flights.csv`

2 Data Wrangling

- in English, wrangling refers to a long, difficult process
    - e.g., cowboys wrangle their cattle or herd (gather, collect their animals)
- there are two major parts of wrangling
    - transforming: sorting or creating new variables (what we'll do today)
    - tidying: reshaping or structuring your data (we'll do this in a few weeks)
- both data tidying and transforming require the `dplyr` package from the `tidyverse`
    - `dplyr` functions are often referred to as verbs, because they *do* something

2.1 `nycflights13`

- we will use the `flights.csv` dataset to explore the basic `dplyr` verbs
    - this dataset is originally from the `nycflights13` package, but I've saved it as a CSV
- the data contains information about df_flights departing from New York City in 2013
    - it comes from the Bureau of Transportation Statistics
    - to find out more about it, call the help page with `?df_flights`

Aufgabe 2.1: `nycflights13`

Beispiel 2.1.

1. load in the dataset `flights.csv` and store it as `df_flights`
    - how many observations are there?
    - how many variables are there?
2. explore the dataset (e.g., `summary()`, `glimpse()`, etc.)

2.2 `dplyr` basics

- today we'll learn some of the primary `dplyr` verbs (functions) that allow us to solve the majority of our data manipulation challenges
    - I use these verbs multiple times in probably every analysis script
- `dplyr` verbs have some things in common:
    1. the first argument is always a data frame
    2. the following arguments typically describe which columns to be operated on, using the variable name (without quotation marks)
    3. the output is always a new dataframe
- the verbs all do one thing well, so we often want to use multiple verbs at once
    - we use the pipe to do this (`%>%` or `|>`)
    - remember, we can read the pipe as `then`
- in the following code, identify
    - the data frame
    - `dplyr` verbs
    - variable names
- can you try to read out (guess) what the following code does?

```
df_flights %>%
  filter(dest == "IAH") %>%
  select(year, month, day) %>%
  relocate(year, .after = day)
```

3 Rows

- in tidy data, rows represent observations
- the most important verbs for rows are:
  - `filter()`: changes which rows are present
  - `arrange()`: changes the order of rows
- we'll also discuss
  - `distinct()`: finds rows with distinct values based on a variable (column)

3.1 `filter()`

- changes which rows are present without changing their order
- takes the dataframe as first argument
  - following arguments are conditions that must be `TRUE` to keep the row
- find all flights that depart more than 120 minutes late:

```
df_flights %>%
  filter(dep_delay > 120)
```

```
# A tibble: 9,723 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>          <dbl>
 1  2013     1     1      848           1835       853     1001           1950
 2  2013     1     1      957            733       144     1056            853
 3  2013     1     1     1114            900       134     1447           1222
 4  2013     1     1     1540           1338       122     2020           1825
 5  2013     1     1     1815           1325       290     2120           1542
 6  2013     1     1     1842           1422       260     1958           1535
 7  2013     1     1     1856           1645       131     2212           2005
 8  2013     1     1     1934           1725       129     2126           1855
 9  2013     1     1     1938           1703       155     2109           1823
10  2013     1     1     1942           1705       157     2124           1830
# i 9,713 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <dbl>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- if you want to save the filtered data, it's usually wise to save it with a *new* object name
  - unless you want to overwrite the pre-filtered version, a new name is necessary

```
df_delay_120 <- df_flights %>%
  filter(dep_delay > 120)
```

Logical operators

- symbols used to describe a logical condition
- `==` *is idential* (`1 == 1`)
- `!=` *is not identical* (`1 != 2`)
- `>` *is greater than* (`2 > 1`)
- `<` *is less than* (`1 < 2`)
- to combine conditions
  - `&` or `,` *and also* (for multiple conditions)
  - `|` *or* (for multiple conditions)
- there's a nice shortcut for cominbing `==` and `|`: `%in%`
  - keeps rows where the variable equals one of the values on the right

3.1.1 == and |

```
df_flights %>%
  filter(month == 1 | month == 2)
```

```
# A tibble: 51,955 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>          <dbl>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 51,945 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <dbl>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
```

```
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

3.1.2 %in%

```
df_flights %>%
  filter(month %in% c(1, 2))
```

```
# A tibble: 51,955 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>          <dbl>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 51,945 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <dbl>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Aufgabe 3.1: `filter()`

Beispiel 3.1.

1. Filter the data to include rows from your birthday
2. How many flights departed NYC on your birthday in 2013?

3.2 `arrange()`

- changes the order of the rows based on a value in a column(s)

```
df_flights %>%
  arrange(arr_time)
```

```
# A tibble: 336,776 x 19
```

```
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>          <dbl>
 1  2013     1     2     2130           2130         0        1             18
 2  2013     1    11     2157           2000       117        1           2208
 3  2013     1    11     2253           2249         4        1           2357
 4  2013     1    14     2122           2130        -8        1              2
 5  2013     1    14     2246           2250        -4        1              7
 6  2013     1    15     2304           2245        19        1           2357
 7  2013     1    16     2018           2025        -7        1           2329
 8  2013     1    16     2303           2245        18        1           2357
 9  2013     1    19     2107           2110        -3        1           2355
10  2013     1    22     2246           2249        -3        1           2357
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <dbl>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- if you use more than one column name, each additional column will be used to break ties between values of the preceding columns

```
df_flights %>%
  arrange(arr_time,dep_delay)
```

```
# A tibble: 336,776 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
  <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>          <dbl>
 1  2013     7    26     2117           2130       -13        1             14
 2  2013     8    12     2103           2115       -12        1           2349
 3  2013     2    19     2247           2258       -11        1             19
 4  2013     6     6     2244           2255       -11        1             19
 5  2013     5     3     2120           2130       -10        1             17
 6  2013     9    19     2120           2130       -10        1           2359
 7  2013    10    19     2246           2255        -9        1             12
 8  2013    11    26     2246           2255        -9        1           2356
 9  2013     4    30     2121           2130        -9        1             16
10  2013     8    24     2256           2305        -9        1             13
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <dbl>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- we can add `desc()` inside `arrange()` to use descending order (big-to-small) instead of the default ascending order

```r
df_flights %>%
  arrange(desc(dep_delay))
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>          <dbl>
 1  2013     1     9      641            900      1301     1242           1530
 2  2013     6    15     1432           1935      1137     1607           2120
 3  2013     1    10     1121           1635      1126     1239           1810
 4  2013     9    20     1139           1845      1014     1457           2210
 5  2013     7    22      845           1600      1005     1044           1815
 6  2013     4    10     1100           1900       960     1342           2211
 7  2013     3    17     2321            810       911      135           1020
 8  2013     6    27      959           1900       899     1236           2226
 9  2013     7    22     2257            759       898      121           1026
10  2013    12     5      756           1700       896     1058           2020
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <dbl>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Aufgabe 3.2: `arrange()`

Beispiel 3.2.

1. Arrange the data by year, month, day, and departure time (`dep_time`)
2. Filter the data to include observations from your birth month and the birth month that is 6 months after your birth month, *then*
    - arrange the data by day and descending arrival time (`arr_time`)

4 Columns

- in tidy data, columns represent variables
- the most important verbs for columns are:
    - `rename()`: changes the names of the columns
    - `mutate()`: creates new columns that are derived from the existing columns
    - `select()`: changes which columns are present
    - `relocate()`: changes the positions of the columns

4.1 `rename()`

- `rename()` lets us change the name of columns
  - the order of the arguments is `new_name` = `old_name`
- let's try changing some of the variable names to German
  - I keep the variable names with lower case, as a coding convention

```
# single variable
fluege <- df_flights %>% rename(jahr = year)

# or multiple variables at once
fluege <- df_flights %>%
  rename(jahr = year,
         monat = month,
         tag = day)
```

4.2 `mutate()`

- `mutate()` creates new columns from existing columns
  - e.g., we can perform basic algebra on the values in each column

```
df_flights %>%
  mutate(
    gain = dep_delay - arr_delay,
  )
```

```
# A tibble: 336,776 x 20
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>          <dbl>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
```

```
# i 336,766 more rows
# i 12 more variables: arr_delay <dbl>, carrier <chr>, flight <dbl>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>, gain <dbl>
```

- `mutate()` adds these new columns to the right of your dataset
  - this makes it difficult to see what's happening
- to control where the new column is added, we can use `.before` or `.after`

```
df_flights |>
  mutate(
    gain = dep_delay - arr_delay,
    .after = day
  )
```

```
# A tibble: 336,776 x 20
    year month   day  gain dep_time sched_dep_time dep_delay arr_time
   <dbl> <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>
 1  2013     1     1    -9      517            515         2      830
 2  2013     1     1   -16      533            529         4      850
 3  2013     1     1   -31      542            540         2      923
 4  2013     1     1    17      544            545        -1     1004
 5  2013     1     1    19      554            600        -6      812
 6  2013     1     1   -16      554            558        -4      740
 7  2013     1     1   -24      555            600        -5      913
 8  2013     1     1    11      557            600        -3      709
 9  2013     1     1     5      557            600        -3      838
10  2013     1     1   -10      558            600        -2      753
# i 336,766 more rows
# i 12 more variables: sched_arr_time <dbl>, arr_delay <dbl>, carrier <chr>,
#   flight <dbl>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

4.3 Exercise

1. Create a new variable called `speed` that equals:
   - `distance` divided by `air_time`, multiplied by 60
   - appears before `dep_time`
2. Render your document

4.4 `select()`

- `select()` subsets the data to include only the columns you want
- select columns by name

```
df_flights %>%
  select(year, month, day)
```

```
# A tibble: 336,776 x 3
    year month   day
   <dbl> <dbl> <dbl>
 1  2013     1     1
 2  2013     1     1
 3  2013     1     1
 4  2013     1     1
 5  2013     1     1
 6  2013     1     1
 7  2013     1     1
 8  2013     1     1
 9  2013     1     1
10  2013     1     1
# i 336,766 more rows
```

- select all columns between `year` and `day`

```
df_flights %>%
  select(year:day)
```

```
# A tibble: 336,776 x 3
    year month   day
   <dbl> <dbl> <dbl>
 1  2013     1     1
 2  2013     1     1
 3  2013     1     1
 4  2013     1     1
 5  2013     1     1
 6  2013     1     1
 7  2013     1     1
 8  2013     1     1
 9  2013     1     1
10  2013     1     1
```

```
# i 336,766 more rows
```

- select all columns except those from year to day (`!` is read as "not")

```
df_flights %>%
  select(!year:day)
```

```
# A tibble: 336,776 x 17
   speed dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay
   <dbl>    <dbl>          <dbl>     <dbl>    <dbl>          <dbl>     <dbl>
 1  370.      517            515         2      830            819        11
 2  374.      533            529         4      850            830        20
 3  408.      542            540         2      923            850        33
 4  517.      544            545        -1     1004           1022       -18
 5  394.      554            600        -6      812            837       -25
 6  288.      554            558        -4      740            728        12
 7  404.      555            600        -5      913            854        19
 8  259.      557            600        -3      709            723       -14
 9  405.      557            600        -3      838            846        -8
10  319.      558            600        -2      753            745         8
# i 336,766 more rows
# i 10 more variables: carrier <chr>, flight <dbl>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#   minute <dbl>, time_hour <dttm>
```

4.5 `select()` helper functions

- some helper functions that make life easier when working with `select()`:
  - `starts_with("abc")`
  - `ends_with("xyz")`
  - `contains("ijk")`
  - `where(is.character)`

```
df_flights %>%
  select(starts_with("d"))
```

```
# A tibble: 336,776 x 5
    day dep_time dep_delay dest  distance
  <dbl>    <dbl>     <dbl> <chr>    <dbl>
1     1      517         2 IAH       1400
```

```
 2     1      533         4 IAH       1416
 3     1      542         2 MIA       1089
 4     1      544        -1 BQN       1576
 5     1      554        -6 ATL        762
 6     1      554        -4 ORD        719
 7     1      555        -5 FLL       1065
 8     1      557        -3 IAD        229
 9     1      557        -3 MCO        944
10     1      558        -2 ORD        733
# i 336,766 more rows
```

```
df_flights %>%
  select(ends_with("ay"))
```

```
# A tibble: 336,776 x 3
     day dep_delay arr_delay
   <dbl>     <dbl>     <dbl>
 1     1         2        11
 2     1         4        20
 3     1         2        33
 4     1        -1       -18
 5     1        -6       -25
 6     1        -4        12
 7     1        -5        19
 8     1        -3       -14
 9     1        -3        -8
10     1        -2         8
# i 336,766 more rows
```

Aufgabe 4.1: `select()`

Beispiel 4.1.

1. Print the columns in `df_flights` that begin with "d"
2. Print the columns in `df_flights` that contain "dep"
3. Print the columns in `df_flights` that
   - begin with begin with "a", and
   - end with "e"

4.6 `relocate()`

- `relocate()` moves variables around
  - by default, it moves them to the front

```
df_flights %>% relocate(speed)
```

```
# A tibble: 336,776 x 20
   speed  year month   day dep_time sched_dep_time dep_delay arr_time
   <dbl> <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>
 1  370.  2013     1     1      517            515         2      830
 2  374.  2013     1     1      533            529         4      850
 3  408.  2013     1     1      542            540         2      923
 4  517.  2013     1     1      544            545        -1     1004
 5  394.  2013     1     1      554            600        -6      812
 6  288.  2013     1     1      554            558        -4      740
 7  404.  2013     1     1      555            600        -5      913
 8  259.  2013     1     1      557            600        -3      709
 9  405.  2013     1     1      557            600        -3      838
10  319.  2013     1     1      558            600        -2      753
# i 336,766 more rows
# i 12 more variables: sched_arr_time <dbl>, arr_delay <dbl>, carrier <chr>,
#   flight <dbl>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

- but we can also use `.before` or `.after` to place a variable

```
df_flights %>% relocate(speed, .after = day)
```

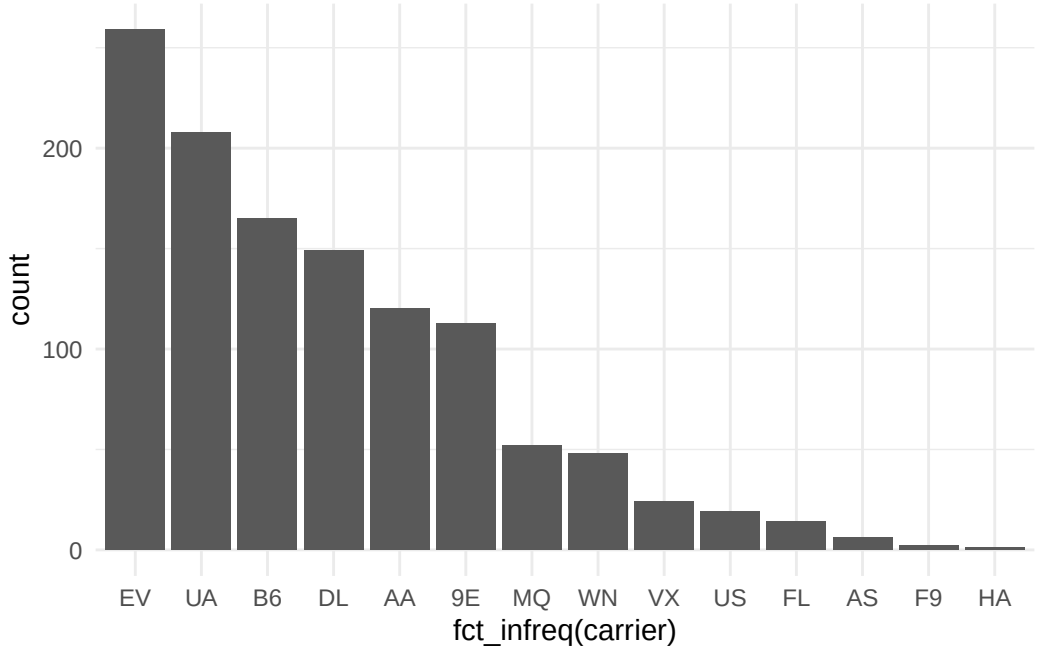
```
# A tibble: 336,776 x 20
    year month   day speed dep_time sched_dep_time dep_delay arr_time
   <dbl> <dbl> <dbl> <dbl>    <dbl>          <dbl>     <dbl>    <dbl>
 1  2013     1     1  370.      517            515         2      830
 2  2013     1     1  374.      533            529         4      850
 3  2013     1     1  408.      542            540         2      923
 4  2013     1     1  517.      544            545        -1     1004
 5  2013     1     1  394.      554            600        -6      812
 6  2013     1     1  288.      554            558        -4      740
 7  2013     1     1  404.      555            600        -5      913
 8  2013     1     1  259.      557            600        -3      709
 9  2013     1     1  405.      557            600        -3      838
```

```
10  2013     1     1  319.      558            600        -2      753
# i 336,766 more rows
# i 12 more variables: sched_arr_time <dbl>, arr_delay <dbl>, carrier <chr>,
#   flight <dbl>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

5 `dplyr` and `ggplot2`

- we can change a dataset using the `dplyr` verbs, and then feed these changes into `ggplot2`
- what will the following code produce?

```
df_flights %>%
  # filter the data
  filter(dep_delay > 120,
         arr_delay < 120) %>%
  # plot the filtered data
  ggplot(aes(x = fct_infreq(carrier))) +
  geom_bar() +
  theme_minimal()
```



- important: we can use pipes (`%>%`) for new verbs/functions

  - but the `ggplot()` function uses `+` to add new *layers* to the plot

5.1 Exercises

- for your Quarto script to count as completed for this week, at the very least it needs to contain the exercises from `nettle_1999_climate.csv` (see below)

5.1.1 `flights.csv`

1. In a single pipeline, print all flights that meet each of the following conditions:
   - Arrived more than two hours late, but didn't leave late
   - Flew to Houston (`IAH` or `HOU`)
   - Were operated by United Airlines (`UA`), American Airlines (`AA`), or Delta (`DL`)
   - Departed in summer (July, August, and September)
   - Arrived more than two hours late, but didn't leave late
2. Sort `df_flights` to find the flights with longest departure delays.
3. Which flights traveled the farthest distance? Which traveled the least distance?
4. In a single pipeline, store a new object called `df_fluege` and:
   - re-load the `flights.csv` dataset
   - select the variables `year`, `month`, `day`, `dep_delay`, `arr_delay`, `carrier`
   - create a new variable `gain`, that is `dep_delay` subtract `arr_delay`
     - and is palced before `dep_delay`
   - rename these variables to be German

```
df_fluege <-
  read_csv(here::here("daten","flights.csv")) %>%
  select(year,month,day,dep_delay,arr_delay,carrier) %>%
  mutate(gewinn = dep_delay - arr_delay, .before = dep_delay) %>%
  rename(jahr = year,
         monat = month) # etcetera
```

5.1.2 `nettle_1999_climate.csv`

5. Store the dataset `nettle_1999_climate.csv` as an object called `df_nettle`
   - take a quick look at the dataset (e.g., `summary()`)
6. In a single pipeline:

- create a new object called `fig_nettle`, which contains the following steps:
- take `df_nettle`, *and then* (i.e., use a `pipe`)
- use the `clean_names` function from the `janitor` package to tidy the names (see last week's notes), *and then*
- rename `mgs` as `grow_seasons`, *and then*
- create a scatterplot that has `grow_seasons` on the `x` axis and `langs` on the `y` axis

Heutige Ziele

Today we learned...

- learn how to wrangle data using the `dplyr` package from the `tidyverse`
- learn to use uses the `pipe` (`%>%`) to feed the result of one function into another function

- learn about functions that operate on rows
- learn about functions that operate on columns
- learn how to combine `dplyr` functions with plots from `ggplot2`

Session Info

Hergestellt mit R version 4.2.3 (2023-03-15) (Shortstop Beagle) und RStudioversion 2023.3.0.386 (Cherry Blossom).

```
sessionInfo()
```

```
R version 4.2.3 (2023-03-15)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.2.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
```

```
 [1] lubridate_1.9.2 forcats_1.0.0   stringr_1.5.0   dplyr_1.1.1
 [5] purrr_1.0.1     readr_2.1.4     tidyr_1.3.0     tibble_3.2.1
 [9] ggplot2_3.4.2   tidyverse_2.0.0

loaded via a namespace (and not attached):
 [1] pillar_1.9.0     compiler_4.2.3   tools_4.2.3      bit_4.0.5
 [5] digest_0.6.31    timechange_0.2.0 jsonlite_1.8.4   evaluate_0.20
 [9] lifecycle_1.0.3  gtable_0.3.3     pkgconfig_2.0.3  rlang_1.1.0
[13] cli_3.6.1        rstudioapi_0.14  parallel_4.2.3   yaml_2.3.7
[17] xfun_0.38        fastmap_1.1.1    withr_2.5.0      knitr_1.42
[21] generics_0.1.3   vctrs_0.6.1      hms_1.1.3        rprojroot_2.0.3
[25] bit64_4.0.5      grid_4.2.3       tidyselect_1.2.0 here_1.0.1
[29] glue_1.6.2       R6_2.5.1         fansi_1.0.4      vroom_1.6.1
[33] rmarkdown_2.21   pacman_0.5.1     farver_2.1.1     tzdb_0.3.0
[37] magrittr_2.0.3   scales_1.2.1     htmltools_0.5.5  colorspace_2.1-0
[41] labeling_0.4.2   utf8_1.2.3       stringi_1.7.12   munsell_0.5.0
[45] crayon_1.5.2
```