Datatransformation

Zeilen und Spalten ändern

Daniela Palleschi

2023-05-16

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

Heute werden wir...

- lernen, wie man Daten mit dem Paket `dplyr` aus dem `tidyverse` verarbeitet
- lernen, wie man die `pipe` (`%>%`) benutzt, um das Ergebnis einer Funktion in eine andere Funktion zu übertragen
- Funktionen kennenlernen, die auf Zeilen operieren
- Funktionen kennenlernen, die mit Spalten arbeiten
- lernen, wie man `dplyr`-Funktionen mit Plots von `ggplot2` kombiniert

Lust auf mehr?

- Ch. 4 in Wickham et al. (o. J.)
- Ch. 9 in Nordmann & DeBruine (2022)

1 Pre-requisites

1. Frisches Quarto-Dokument
    - ein neues Quarto-Dokument für den heutigen Unterricht erstellen
        - Datei > Neues Dokument > Quarto Dokument, mit dem Namen `04-wrangling`
    - YAML einrichten: Titel, Ihr Name, ein `toc` hinzufügen

```yaml
title: "Datatransformation"
subtitle: "Woche 5"
author: "Your name here"
lang: de
format:
  html:
    toc: true
```

2. Pakete
    - Die heutigen Pakete sind:
        - `tidyverse`: zum Verarbeiten (`dplyr`) und Plotten (`ggplot2`)

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

3. Datensatz
    - Speichern Sie den Datensatz aus Moodle in Ihrem Ordner `daten`:
        - `flights.csv`

2 Data Wrangling (hadern, rangeln, zanken)

- Im Englischen bezieht sich "wrangling" auf einen langen, schwierigen Prozess, oder auf einen Streif
- Es gibt zwei Hauptbestandteile des Wrangling
    - *Transformieren*: Sortieren oder Erstellen neuer Variablen (was wir heute tun werden)
    - *Aufräumen*: Umformung oder Strukturierung Ihrer Daten (dies werden wir in einigen Wochen tun)
- Sowohl das Aufräumen als auch das Transformieren von Daten erfordern das Paket `dplyr` aus dem `tidyverse`.
    - `dplyr` Funktionen werden oft als Verben bezeichnet, weil sie etwas *tun*

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

- die Daten enthalten Informationen über Flüge, die im Jahr 2013 von New York City aus gestartet sind
  - sie stammen vom Bureau of Transportation Statistics
  - um mehr darüber zu erfahren, rufen Sie die Hilfeseite mit `?df_flights` auf

Aufgabe 2.1: `nycflights13`

Beispiel 2.1.

1. Ladet den Datensatz `flights.csv` und speichert ihn als `df_flights`.
   - Wie viele Beobachtungen sind vorhanden?
   - Wie viele Variablen gibt es?
2. den Datensatz untersuchen (z. B. `summary()`, `glimpse()`, usw.)

2.2 `dplyr` Grundlagen

- heute lernen wir einige der wichtigsten `dplyr`-Verben (Funktionen) kennen, mit denen wir die meisten unserer Datenmanipulationsprobleme lösen können
  - Ich verwende diese Verben mehrfach in wahrscheinlich jedem Analyseskript
- Die `dplyr`-Verben haben einige Dinge gemeinsam:
  1. das erste Argument ist immer ein Datenrahmen
  2. die folgenden Argumente beschreiben in der Regel die zu bearbeitenden Spalten, wobei der Variablenname (ohne Anführungszeichen) verwendet wird
  3. die Ausgabe ist immer ein neuer Datenrahmen

2.3 `dplyr` Grundlagen: Pipe

- Die Verben sind alle für eine Sache gut geeignet, so dass wir oft mehrere Verben auf einmal verwenden wollen.
  - Wir benutzen die Pipe, um dies zu tun (`%>%` oder `|>`)
  - Denkt daran, dass wir die Pipe als `(und) dann` lesen können
- im folgenden Code identifizieren
  - den Datenrahmen
  - "dplyr"-Verben
  - die Variablennamen

Abbildung 1: Image source: magrittr documentation (all rights reserved)

- Kannst du versuchen, herauszulesen (zu erraten), was der folgende Code macht?

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

3 Zeilen

- In aufgeräumten Daten stellen die Zeilen Beobachtungen dar.
- die wichtigsten Verben für Zeilen sind:
  - `filter()`: ändert, welche Zeilen vorhanden sind
  - `arrange()`: ändert die Reihenfolge der Zeilen
- Wir besprechen auch
  - `distinct()`: findet Zeilen mit unterschiedlichen Werten auf der Grundlage einer Variablen (Spalte)

3.1 `filter()`

- ändert, welche Zeilen vorhanden sind, ohne ihre Reihenfolge zu ändern
- nimmt den Datenrahmen als erstes Argument

  - Die folgenden Argumente sind Bedingungen, die `TRUE` sein müssen, damit die Zeile erhalten bleibt

- alle Flüge zu finden, die mit mehr als 120 Minuten Verspätung abfliegen:

```
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>
```

- wenn man die gefilterten Daten speichern will, ist es in der Regel ratsam, sie unter einem *neuen* Objektnamen zu speichern
  - wenn man nicht die vorgefilterte Version überschreiben will, ist ein neuer Name notwendig

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

Logical operators

- Symbole, die zur Beschreibung einer logischen Bedingung verwendet werden
  - `==` *ist identisch* (`1 == 1`)
  - `!=` *ist nicht identisch* (`1 != 2`)
  - `>` *ist größer als* (`2 > 1`)
  - `<` *ist kleiner als* (`1 < 2`)

- um Bedingungen zu kombinieren
  - `&` oder `,` *und auch* (für mehrere Bedingungen)
  - `|` *oder* (für mehrere Bedingungen)
- es gibt eine nette Abkürzung für die Kombination von `==` und `|`: `%in%`
  - behält Zeilen, in denen die Variable gleich einem der Werte auf der rechten Seite ist

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. Filtert die Daten so, dass sie Zeilen von Ihrem Geburtstag enthalten.
2. Wie viele Flüge sind 2013 an deinem Geburtstag von NYC abgeflogen?

3.2 `arrange()`

- ändert die Reihenfolge der Zeilen auf der Grundlage eines Wertes in einer oder mehreren Spalten

```
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>
```

- wir können `desc()` innerhalb von `arrange()` hinzufügen, um eine absteigende Reihenfolge (von groß nach klein) anstelle der standardmäßigen aufsteigenden Reihenfolge zu verwenden
  - `desc` ist die Abkürzung für *descending* im Englischen (= absteigend)

```
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. Ordne die Daten nach Jahr, Monat, Tag und Abfahrtszeit (`dep_time`)
2. Filtere die Daten so, dass sie Beobachtungen aus deinem Geburtsmonat und dem Geburtsmonat, der 6 Monate nach deinem Geburtsmonat liegt, enthalten, *dann*
   - ordne die Daten nach Tag und absteigender Ankunftszeit (`arr_time`)

4 Spalten

- In Tidy Data stellen die Spalten Variablen dar
- die wichtigsten Verben für Spalten sind:
  - `rename()`: ändert die Namen der Spalten
  - `mutate()`: erzeugt neue Spalten, die von den vorhandenen Spalten abgeleitet werden
  - `select()`: ändert, welche Spalten vorhanden sind
  - `relocate()`: ändert die Position der Spalten

4.1 `rename()`

- Mit `rename()` können wir den Namen von Spalten ändern
  - die Reihenfolge der Argumente ist `neuer_name` = `alter_name`
- Versuchen wir, einige der Variablennamen auf Deutsch zu ändern
  - Ich behalte die Variablennamen in Kleinbuchstaben, als Kodierungskonvention

```
# 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()` erzeugt neue Spalten aus vorhandenen Spalten
  - z.B. können wir einfache Algebra mit den Werten in jeder Spalte durchführen

```
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>
```

- mit "mutate()" werden diese neuen Spalten auf der rechten Seite des Datensatzes hinzugefügt
  - Das macht es schwierig zu sehen, was passiert.
- um zu kontrollieren, wo die neue Spalte hinzugefügt wird, können wir `.before` oder `.after` verwenden

```
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. Erstelle eine neue Variable namens `Geschwindigkeit`, die gleich ist:
    - Entfernung" geteilt durch "air_time", multipliziert mit 60
    - erscheint vor `dep_time`
2. Rendert euer Dokument

4.4 `select()`

- `select()` fasst die Daten so zusammen, dass sie nur die gewünschten Spalten enthalten
- Spalten nach Namen auswählen

```
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
```

- alle Spalten zwischen `year` und `day` auswählen

```r
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
```

- alle Spalten außer denen von Jahr bis Tag auswählen (`!` wird als "nicht" gelesen)

```r
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

- einige Hilfsfunktionen, die das Leben bei der Arbeit mit `select()` erleichtern:
  - `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. Drucke die Spalten in `df_flights`, die mit "d" beginnen
2. Drucke die Spalten in `df_flights`, die "dep" enthalten
3. Drucke die Spalten in `df_flights`, die
   - mit mit "a" beginnen, und
   - mit "e" enden

4.6 `relocate()`

- `relocate()` verschiebt Variablen
  - standardmäßig werden sie nach vorne verschoben

```
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>
```

- aber wir können auch `.before` oder `.after` verwenden, um eine Variable zu platzieren

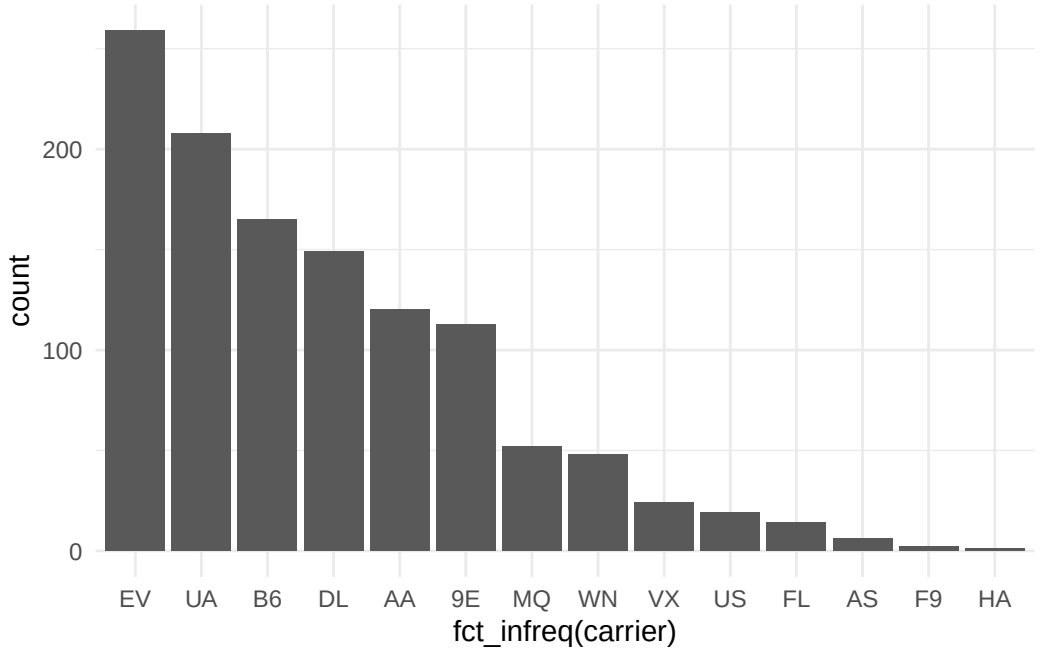
```
df_flights %>% relocate(speed, .after = day)
```

```
# 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

- wir können einen Datensatz mit Hilfe der Verben "dplyr" ändern und diese Änderungen dann in "ggplot2" einspeisen
- Was würde der folgende Code ergeben?

```
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()
```



- wichtig: wir können Pipes (`%>%`) für neue Verben/Funktionen verwenden
  - aber die Funktion `ggplot()` verwendet `+`, um neue *Ebenen* zur Darstellung hinzuzufügen

5.1 Exercises

- damit Ihr Quarto-Skript für diese Woche als abgeschlossen gilt, muss es zumindest die Übungen aus `nettle_1999_climate.csv` enthalten (siehe unten)

5.1.1 `flights.csv`

1. Drucke in einer einzigen Pipeline alle Flüge aus, die jede der folgenden Bedingungen erfüllen:
   - Kam mit mehr als zwei Stunden Verspätung an, ist aber nicht zu spät abgeflogen
   - Sie flogen nach Houston (`IAH` oder `HOU`)
   - Wurden von United Airlines (`UA`), American Airlines (`AA`) oder Delta (`DL`) durchgeführt
   - Sie sind im Sommer abgeflogen (Juli, August oder September)
   - kamen mehr als zwei Stunden zu spät an, flogen aber nicht zu spät ab
2. Sortiert `df_flights`, um die Flüge mit den größten Abflugverspätungen zu finden

3. Welche Flüge haben die weiteste Strecke zurückgelegt? Welche Flüge haben die geringste Entfernung zurückgelegt?

4. Speichert in einer einzigen Pipeline ein neues Objekt namens `df_fluege` und:

   - den Datensatz `flights.csv` erneut laden
   - Auswahl der Variablen `Jahr`, `Monat`, `Tag`, `dep_delay`, `arr_delay`, `carrier`
   - eine neue Variable `gain` erstellen, die `dep_delay` subtrahiert von `arr_delay` ist
     - und vor `dep_delay` eingefügt wird
   - benennen Sie diese Variablen um, so dass sie deutsch sind

```
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. Speichere den Datensatz `nettle_1999_climate.csv` als ein Objekt namens `df_nettle`

   - einen kurzen Blick auf den Datensatz werfen (z.B. `summary()`)

6. In einer einzigen Pipeline:

   - Erstellen eines neuen Objekts namens `fig_nettle`, das die folgenden Schritte enthält:

- nehmt `df_nettle`, *und dann* (d.h., benutzt eine `Pipeline`)
  - verwende die Funktion `clean_names` aus dem `janitor`-Paket, um die Namen zu bereinigen (siehe die Anmerkungen von letzter Woche), *und dann*
  - Umbenennen von `mgs` in `grow_seasons`, *und dann*
  - ein Streudiagramm erstellen, das `grow_seasons` auf der `x` Achse und `langs` auf der `y` Achse hat

Heutige Ziele

Heute haben wir gelernt…

- gelernt, wie man mit dem Paket `dplyr` aus dem `tidyverse` Daten verarbeiten kann

- gelernt, wie man die `pipe` (`%>%`) verwendet, um das Ergebnis einer Funktion in eine andere Funktion einzuspeisen
- Funktionen kennengelernt, die auf Zeilen operieren
- Funktionen kennengelernt, die mit Spalten arbeiten
- gelernt, wie man `dplyr`-Funktionen mit Plots von `ggplot2` kombiniert

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
```

Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills* (Version 2.0). Zenodo. https://doi.org/10.5281/zenodo.6365078

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (o. J.). *R for Data Science* (2. Aufl.). https://r4ds.hadley.nz/