R for Data Analytics Part 1, Lecture 4

Michèle Fille

Table of contents

Lecture 4 – Packages for Data Wrangling: dplyr and tidyr

4.1 Manipulating data with dplyr

Exercise 4.1. dyplyr

Exercise 4.1: Practicing with dplyr Verbs and Pipes

Install the nycflights13 package and load it. Also load dplyr.

```
#install.packages("nycflights13")

library(nycflights13)
```

```
Warning: Paket 'nycflights13' wurde unter R Version 4.3.3 erstellt
```

```r
library(dplyr)
```

```
Attache Paket: 'dplyr'

Die folgenden Objekte sind maskiert von 'package:stats':

    filter, lag

Die folgenden Objekte sind maskiert von 'package:base':

    intersect, setdiff, setequal, union
```

a) The data frame flights is now accessible to you. Use appropriate functions to inspect it:

- How many rows and columns does it have?
- What are the names of the columns?
- Use ?flights to search for documentation on the data set (for what the columns represent).

```r
nrow(flights)
```

```
[1] 336776
```

```r
ncol(flights)
```

```
[1] 19
```

```r
colnames(flights)
```

```
 [1] "year"           "month"          "day"            "dep_time"      
 [5] "sched_dep_time" "dep_delay"      "arr_time"       "sched_arr_time"
 [9] "arr_delay"      "carrier"        "flight"         "tailnum"       
[13] "origin"         "dest"           "air_time"       "distance"      
[17] "hour"           "minute"         "time_hour"     
```

```r
?flights
```

```
starte den http Server für die Hilfe fertig
```

b) Use dplyr to give the data frame a new column that is the amount of time gained or lost while flying (that is: how much of the delay arriving occurred during flight, as opposed to before departing).

- Hint: If your new column doesn't show up with print(), look at the bottom of the output written in grey: Maybe there was not enough space to print it in your console window! In this case you use print(flights, width = Inf) to show all columns.

```r
flights <- mutate(flights, gain_in_air = arr_delay - dep_delay)
print(flights, width = Inf)
```

```
# A tibble: 336,776 x 20
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 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
   arr_delay carrier flight tailnum origin dest  air_time distance  hour minute
       <dbl> <chr>    <int> <chr>   <chr>  <chr>    <dbl>    <dbl> <dbl>  <dbl>
 1        11 UA        1545 N14228  EWR    IAH        227     1400     5     15
 2        20 UA        1714 N24211  LGA    IAH        227     1416     5     29
 3        33 AA        1141 N619AA  JFK    MIA        160     1089     5     40
 4       -18 B6         725 N804JB  JFK    BQN        183     1576     5     45
 5       -25 DL         461 N668DN  LGA    ATL        116      762     6      0
 6        12 UA        1696 N39463  EWR    ORD        150      719     5     58
 7        19 B6         507 N516JB  EWR    FLL        158     1065     6      0
 8       -14 EV        5708 N829AS  LGA    IAD         53      229     6      0
 9        -8 B6          79 N593JB  JFK    MCO        140      944     6      0
10         8 AA         301 N3ALAA  LGA    ORD        138      733     6      0
   time_hour           gain_in_air
```

```
   <dttm>                          <dbl>
 1 2013-01-01 05:00:00                 9
 2 2013-01-01 05:00:00                16
 3 2013-01-01 05:00:00                31
 4 2013-01-01 05:00:00               -17
 5 2013-01-01 06:00:00               -19
 6 2013-01-01 05:00:00                16
 7 2013-01-01 06:00:00                24
 8 2013-01-01 06:00:00               -11
 9 2013-01-01 06:00:00                -5
10 2013-01-01 06:00:00                10
# i 336,766 more rows
```

c) Use dplyr to sort your data frame in descending order by the column you just created. Save it as a variable (or in the same one!)

```
flights1 <- arrange(flights, desc(gain_in_air))
View(head(flights1))
```

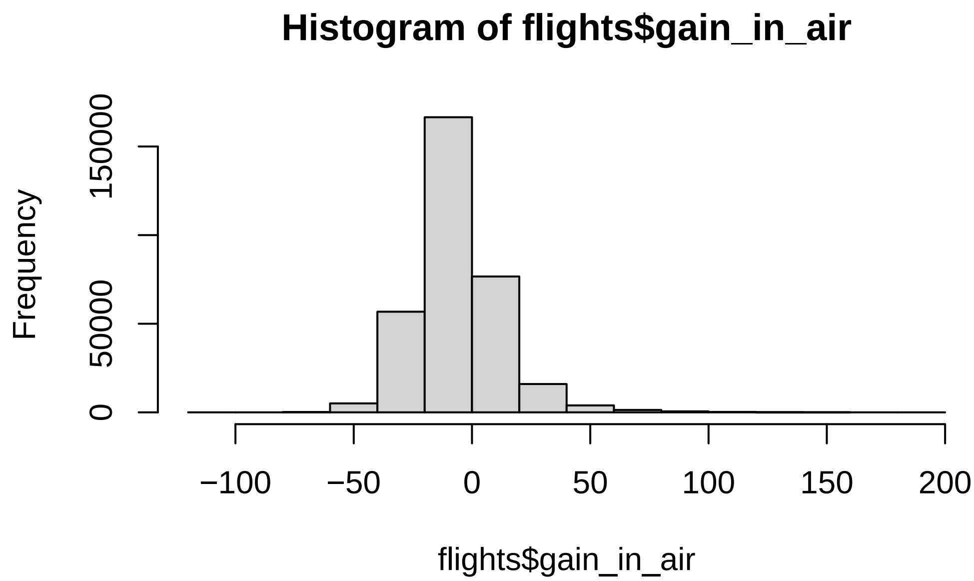
d) For practice, repeat the last 2 steps in a single statement using the pipe operator. You can clear your environmental variables to "reset" the data frame.

```
flights2 <- flights %>%
  mutate(gain_in_air = arr_delay - dep_delay) %>% # if atribute not already created
  arrange(desc(gain_in_air))
```

e) Make a histogram of the amount of time gained using the hist() function from base R. Alternatively, you can use ggplot2 to create a histogram.

- Hint: Use geom_histogram() to make a histogram with ggplot.

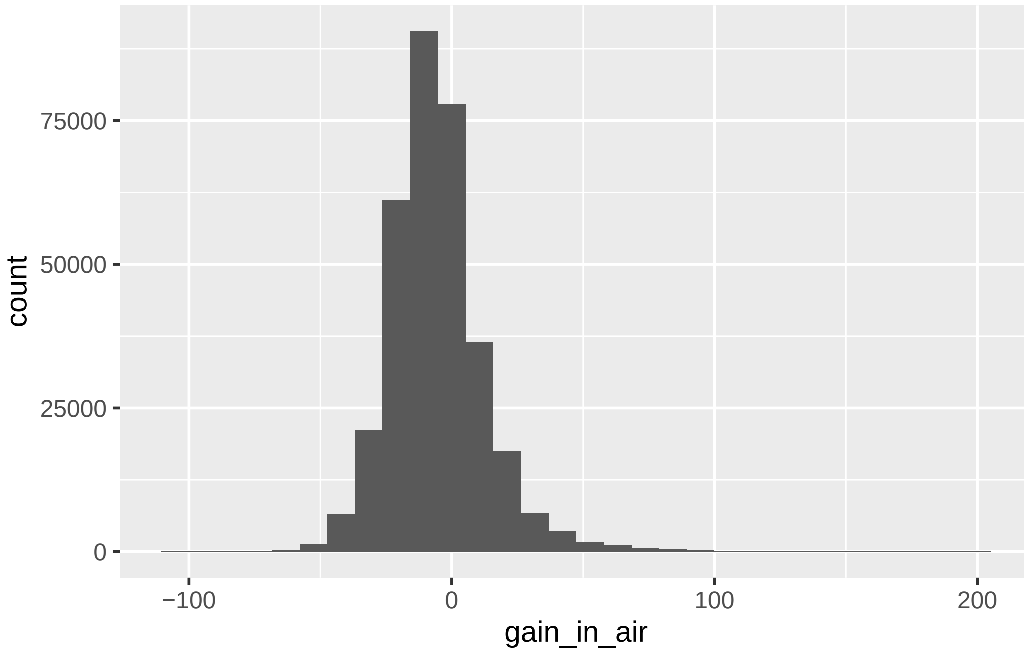
  ```
  library(ggplot2)

  # histogram with base R:
  hist(flights$gain_in_air)
  ```




```
# histogram with ggplot2:
ggplot(flights) +
  geom_histogram(mapping = aes(x = gain_in_air))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
Warning: Removed 9430 rows containing non-finite outside the scale range
(`stat_bin()`).
```



- Bonus: Compare the two visualizations: what is different and why are they different?

  In the first plot, the mode (the most frequent gain) has a count of over 150'000. In contrast, in the second plot, it has a count of over a bit over 100'000. The difference results from different binning: The bins ("intervals") used for counting the frequencies have different widths. The larger the intervals, the more occurances per interval!

f) On average, did flights gain or lose time?

- Note: Use the na.rm = TRUE argument to remove NA values from your aggregation. Otherwise the result will be NA.

```
mean(flights$gain_in_air, na.rm = TRUE)
```

```
[1] -5.659779
```

g) Create a data.frame of flights headed to SeaTac ('SEA'), only including the origin, destination, and the gain_in_air column you created.

```
to_sea <- flights %>%
  select(origin, dest, gain_in_air) %>%
  filter(dest == "SEA")
```

h) On average, did flights to SeaTac gain or lose time?

```
mean(to_sea$gain_in_air, na.rm = TRUE)
```

```
[1] -11.6991
```

i) Consider flights from JFK to SEA. What was the average, min, and max air time of those flights?

- Hint: Don't forget to use the argument na.rm = TRUE in your aggregations.
- Bonus: Try to use pipes so that you can answer the last question in one single statement!

```
flights %>%
  filter(origin == "JFK",
         dest == "SEA") %>%
  summarize(avg_air_time = mean(air_time, na.rm = TRUE),
            min_air_time = min(air_time, na.rm = TRUE),
            max_air_time = max(air_time, na.rm = TRUE))
```

```
# A tibble: 1 x 3
  avg_air_time min_air_time max_air_time
         <dbl>        <dbl>        <dbl>
1         329.          275          389
```

Self-Study 4.1. dyplyr

Self-Study 4.1 - Task 1: Using dplyr for Grouping

Install the nycflights13 package (if needed) and load it. Also load dplyr. View the data set flights .

a) What was the average departure delay in each month? Save this as a data frame dep_delay_by_month.

- Hint: you'll have to perform a grouping operation then summarizing your data.

```
dep_delay_by_month <- flights %>%
  group_by(month) %>% # creates a tibble that groups by month
  summarize(delay_avg = mean(dep_delay, na.rm = TRUE)) # calculates the mean departure del

print(dep_delay_by_month)
```

```
# A tibble: 12 x 2
   month delay_avg
   <int>     <dbl>
 1     1     10.0 
 2     2     10.8 
 3     3     13.2 
 4     4     13.9 
 5     5     13.0 
 6     6     20.8 
 7     7     21.7 
 8     8     12.6 
 9     9      6.72
10    10      6.24
11    11      5.44
12    12     16.6 
```

b) Which month had the greatest average departure delay?

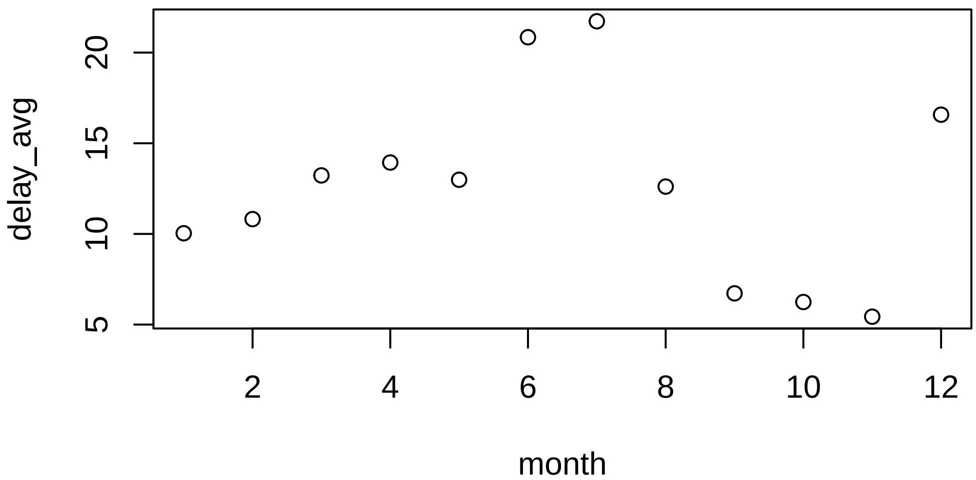
```
filter(dep_delay_by_month, delay_avg == max(delay_avg)) %>%
  select(month)
```

```
# A tibble: 1 x 1
  month
  <int>
1     7
```

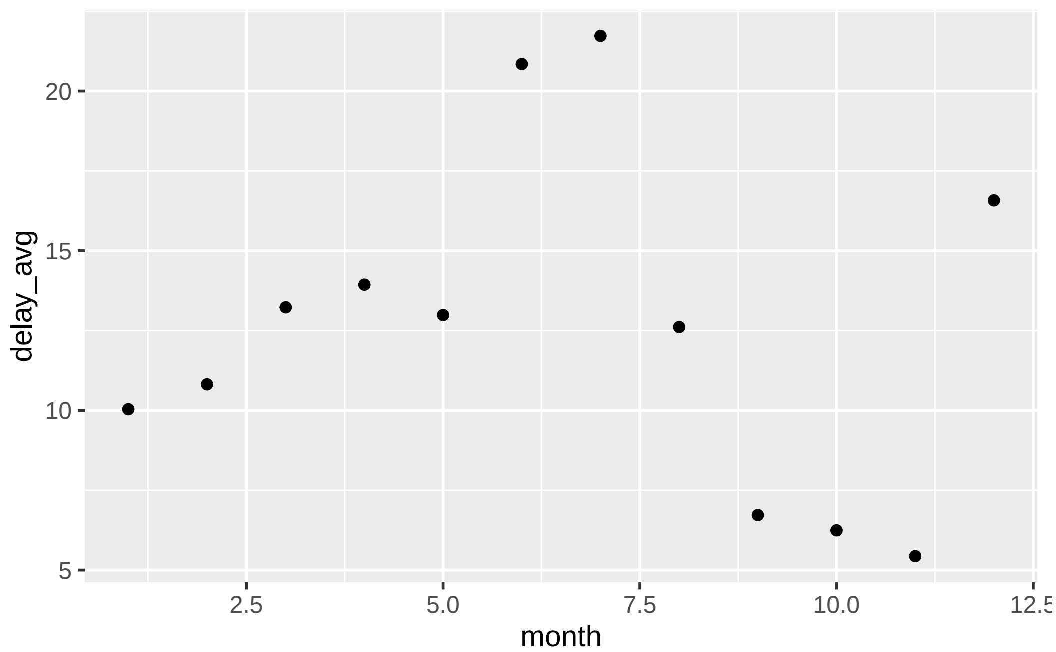
c) If your data frame dep_delay_by_month contains only two columns (e.g., "month", and "delay" in that order), you can create a scatterplot by passing that data frame directly to the base R function plot(). It is a generic function, that automatically makes a scatterplot when passed a data frame with 2 columns.

- Alternatively, you can of course also use ggplot2 to create the scatterplot.

```
# With base R:
plot(dep_delay_by_month) # notice that we only need to pass the data frame as is!
```




```
# With ggplot2:
# In this case, ggplot is more effort!! (BUT it's easier to pimp your plot so that it look
library(ggplot2)
ggplot(dep_delay_by_month) +
  geom_point(mapping = aes(x = month, y = delay_avg))
```



d) To which destinations were the average arrival delays the highest?

- Hint: you'll have to perform a grouping operation then summarize your data. You can use the head() function to view just the first few rows for checking.

```
arr_delay_by_month <- flights %>%
  group_by(dest) %>%
  summarise(delay_avg = mean(arr_delay, na.rm = TRUE)) %>%
  arrange(-delay_avg)  # = arrange(desc(delay_avg))

head(arr_delay_by_month)
```

```
# A tibble: 6 x 2
  dest  delay_avg
  <chr>     <dbl>
1 CAE        41.8
2 TUL        33.7
3 OKC        30.6
4 JAC        28.1
5 TYS        24.1
6 MSN        20.2
```

e) The package nycflights13 also includes a data frame called airports. You can look up the above destinations in the airports data frame!

```
head(airports)
```

```
# A tibble: 6 x 8
  faa   name                             lat   lon   alt    tz dst   tzone
  <chr> <chr>                          <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1 04G   Lansdowne Airport               41.1 -80.6  1044    -5 A     America/Ne~
2 06A   Moton Field Municipal Airport   32.5 -85.7   264    -6 A     America/Ch~
3 06C   Schaumburg Regional             42.0 -88.1   801    -6 A     America/Ch~
4 06N   Randall Airport                 41.4 -74.4   523    -5 A     America/Ne~
5 09J   Jekyll Island Airport           31.1 -81.4    11    -5 A     America/Ne~
6 0A9   Elizabethton Municipal Airport  36.4 -82.2  1593    -5 A     America/Ne~
```

```
filter(airports, faa == arr_delay_by_month$dest[1]) # for example we can look up teh first
```

```
# A tibble: 1 x 8
  faa   name                    lat   lon   alt    tz dst   tzone
  <chr> <chr>                 <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1 CAE   Columbia Metropolitan  33.9 -81.1   236    -5 A     America/New_York
```

```
# see all destinations from above (would blow up the file size)
# airports %>%
#   filter(faa %in% arr_delay_by_month$dest)
```

f) Which city was flown to with the highest average speed?

```
city_fasted_speed <- flights %>%
  mutate(speed = distance / air_time * 60) %>%
  group_by(dest) %>%
  summarise(avg_speed = mean(speed, na.rm = TRUE)) %>%
  filter(avg_speed == max(avg_speed, na.rm = TRUE))

city_fasted_speed
```

```
# A tibble: 1 x 2
  dest  avg_speed
  <chr>     <dbl>
1 ANC        490.
```

Self-Study 4.1 - Task 2: Using the dplyr Join Operations

Install the nycflights13 package (if needed) and load it. Also load dplyr. View the data set flights .

a) Create a dataframe of the average arrival delays for each destination from the flights data frame. Then use left_join() to join on the airports dataframe.

- Remark: The airports dataframe is also part of the nycflights13 package and holds information about the airports.

```
avg_delay <- flights %>%
  group_by(dest) %>%    # creates it as tibble that groups rows by destination
  summarise(avg_delay = mean(arr_delay, na.rm = TRUE))  # calculates the mean arrival dela

avg_delay_dest <- avg_delay %>%
  mutate(faa = dest) %>% # create a new column faa, so we can use it as join condition
  left_join(airports, by = "faa")

head(avg_delay)
```

```
# A tibble: 6 x 2
  dest  avg_delay
  <chr>     <dbl>
1 ABQ        4.38
2 ACK        4.85
3 ALB       14.4
4 ANC       -2.5
5 ATL       11.3
6 AUS        6.02
```

```
head(avg_delay_dest)
```

```
# A tibble: 6 x 10
  dest  avg_delay faa   name                  lat    lon   alt    tz dst   tzone
  <chr>     <dbl> <chr> <chr>               <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
1 ABQ        4.38 ABQ   Albuquerque Intern~  35.0 -107.   5355    -7 A     Amer~
2 ACK        4.85 ACK   Nantucket Mem        41.3  -70.1    48    -5 A     Amer~
3 ALB       14.4  ALB   Albany Intl          42.7  -73.8   285    -5 A     Amer~
4 ANC       -2.5  ANC   Ted Stevens Anchor~  61.2 -150.    152    -9 A     Amer~
5 ATL       11.3  ATL   Hartsfield Jackson~  33.6  -84.4  1026    -5 A     Amer~
6 AUS        6.02 AUS   Austin Bergstrom I~  30.2  -97.7   542    -6 A     Amer~
```

b) Which airport had the largest average arrival delay?

```r
largest_arrival_delay <- avg_delay_dest %>%
  filter(avg_delay == max(avg_delay, na.rm = TRUE))

print(largest_arrival_delay)
```

```
# A tibble: 1 x 10
  dest  avg_delay faa   name                    lat   lon   alt    tz dst   tzone
  <chr>     <dbl> <chr> <chr>                 <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1 CAE        41.8 CAE   Columbia Metropolit~  33.9 -81.1   236    -5 A     Amer~
```

```r
# Notice that we could have done all the above in one single statement using pipes!
largest_arrival_delay <- flights %>%
  group_by(dest) %>%
  summarise(avg_delay = mean(arr_delay, na.rm = TRUE)) %>%
  mutate(faa = dest) %>%
  left_join(airports, by = "faa") %>%
  filter(avg_delay == max(avg_delay, na.rm = TRUE))

print(largest_arrival_delay)
```

```
# A tibble: 1 x 10
  dest  avg_delay faa   name                    lat   lon   alt    tz dst   tzone
  <chr>     <dbl> <chr> <chr>                 <dbl> <dbl> <dbl> <dbl> <chr> <chr>
1 CAE        41.8 CAE   Columbia Metropolit~  33.9 -81.1   236    -5 A     Amer~
```

c) Create a dataframe of the average arrival delay for each airline, then use left_join() to join on the airlines dataframe (which is also part of the nycflights13 package).

```r
head(airlines)
```

```
# A tibble: 6 x 2
  carrier name
  <chr>   <chr>
1 9E      Endeavor Air Inc.
2 AA      American Airlines Inc.
3 AS      Alaska Airlines Inc.
4 B6      JetBlue Airways
5 DL      Delta Air Lines Inc.
6 EV      ExpressJet Airlines Inc.
```

```r
avg_delay_airline <- flights %>%
  group_by(carrier) %>%
  summarise(avg_delay = mean(arr_delay, na.rm = TRUE)) %>%
  left_join(airlines, by = "carrier")

avg_delay_airline
```

```
# A tibble: 16 x 3
   carrier avg_delay name
   <chr>       <dbl> <chr>
 1 9E          7.38  Endeavor Air Inc.
 2 AA          0.364 American Airlines Inc.
 3 AS         -9.93  Alaska Airlines Inc.
 4 B6          9.46  JetBlue Airways
 5 DL          1.64  Delta Air Lines Inc.
 6 EV         15.8   ExpressJet Airlines Inc.
 7 F9         21.9   Frontier Airlines Inc.
 8 FL         20.1   AirTran Airways Corporation
 9 HA         -6.92  Hawaiian Airlines Inc.
10 MQ         10.8   Envoy Air
11 OO         11.9   SkyWest Airlines Inc.
12 UA          3.56  United Air Lines Inc.
13 US          2.13  US Airways Inc.
14 VX          1.76  Virgin America
15 WN          9.65  Southwest Airlines Co.
16 YV         15.6   Mesa Airlines Inc.
```

d) Which airline had the smallest average arrival delay?

```r
smallest_airline_delay <- avg_delay_airline %>%
  filter(avg_delay == max(avg_delay, na.rm = TRUE))

smallest_airline_delay
```

```
# A tibble: 1 x 3
  carrier avg_delay name
  <chr>       <dbl> <chr>
1 F9           21.9 Frontier Airlines Inc.
```

OR all in one

```r
smallest_airline_delay <- flights %>%
  group_by(carrier) %>%
  summarise(avg_delay = mean(arr_delay, na.rm = TRUE)) %>%
  left_join(airlines, by = "carrier") %>%
  filter(avg_delay == max(avg_delay, na.rm = TRUE))

smallest_airline_delay
```

```
# A tibble: 1 x 3
  carrier avg_delay name
  <chr>       <dbl> <chr>
1 F9           21.9 Frontier Airlines Inc.
```

Self-Study 4.1 – Task 3: Comparing base R and dplyr

a) Install and load dplyr if needed.

Already done above

b) Install and load the fueleconomy package from GitHub as follows:

- Install the devtools package (as usual).
- The devtool package allows us to make installations from GitHub. Use the following command to install the fueleconomy package from GitHub: devtools::install_github("hadley/fueleconomy")
- Load the fueleconomy package as usual.

```r
# install.packages("devtools")
# devtools::install_github("hadley/fueleconomy")
library(fueleconomy)
```

c) Now you have access to the vehicles data frame. Use View() to get a first impression. Select from this data frame the column makes, which holds the different car manufacturers. Save it in the variable makes.

- Hint: Since you made a selection on a data frame, the result is a vector.

```r
View(vehicles)

makes <- vehicles$make
```

d) Use the function unique() to list and count the different car manufacturers. Alternatively, use the dplyr function distinct()to do the same. What is the difference?

```r
unique(makes) # returns a vector
```

```
 [1] "Acura"                          "Alfa Romeo"
 [3] "AM General"                     "American Motors Corporation"
 [5] "ASC Incorporated"               "Aston Martin"
 [7] "Audi"                           "Aurora Cars Ltd"
 [9] "Autokraft Limited"              "Azure Dynamics"
[11] "Bentley"                        "Bertone"
[13] "Bill Dovell Motor Car Company"  "Bitter Gmbh and Co. Kg"
[15] "BMW"                            "BMW Alpina"
[17] "Bugatti"                        "Buick"
[19] "BYD"                            "Cadillac"
[21] "CCC Engineering"                "Chevrolet"
[23] "Chrysler"                       "CODA Automotive"
[25] "Consulier Industries Inc"       "CX Automotive"
[27] "Dabryan Coach Builders Inc"     "Dacia"
[29] "Daewoo"                         "Daihatsu"
[31] "Dodge"                          "E. P. Dutton, Inc."
[33] "Eagle"                          "Environmental Rsch and Devp Corp"
[35] "Evans Automobiles"              "Excalibur Autos"
[37] "Federal Coach"                  "Ferrari"
[39] "Fiat"                           "Fisker"
[41] "Ford"                           "General Motors"
[43] "Geo"                            "GMC"
[45] "Goldacre"                       "Grumman Allied Industries"
[47] "Grumman Olson"                  "Honda"
[49] "Hummer"                         "Hyundai"
[51] "Import Foreign Auto Sales Inc"  "Import Trade Services"
[53] "Infiniti"                       "Isis Imports Ltd"
[55] "Isuzu"                          "J.K. Motors"
[57] "Jaguar"                         "JBA Motorcars, Inc."
[59] "Jeep"                           "Kia"
[61] "Laforza Automobile Inc"         "Lambda Control Systems"
[63] "Lamborghini"                    "Land Rover"
[65] "Lexus"                          "Lincoln"
[67] "London Coach Co Inc"            "London Taxi"
[69] "Lotus"                          "Mahindra"
[71] "Maserati"                       "Maybach"
[73] "Mazda"                          "Mcevoy Motors"
[75] "McLaren Automotive"             "Mercedes-Benz"
[77] "Mercury"                        "Merkur"
[79] "MINI"                           "Mitsubishi"
```

```
 [81] "Morgan"                           "Nissan"
 [83] "Oldsmobile"                       "Panos"
 [85] "Panoz Auto-Development"           "Panther Car Company Limited"
 [87] "PAS Inc - GMC"                    "PAS, Inc"
 [89] "Peugeot"                          "Pininfarina"
 [91] "Plymouth"                         "Pontiac"
 [93] "Porsche"                          "Quantum Technologies"
 [95] "Qvale"                            "Ram"
 [97] "Red Shift Ltd."                   "Renault"
 [99] "Rolls-Royce"                      "Roush Performance"
[101] "Ruf Automobile Gmbh"              "S and S Coach Company  E.p. Dutton"
[103] "Saab"                             "Saleen"
[105] "Saleen Performance"               "Saturn"
[107] "Scion"                            "Shelby"
[109] "smart"                            "Spyker"
[111] "SRT"                              "Sterling"
[113] "Subaru"                           "Superior Coaches Div E.p. Dutton"
[115] "Suzuki"                           "Tecstar, LP"
[117] "Tesla"                            "Texas Coach Company"
[119] "Toyota"                           "TVR Engineering Ltd"
[121] "Vector"                           "Vixen Motor Company"
[123] "Volga Associated Automobile"      "Volkswagen"
[125] "Volvo"                            "VPG"
[127] "Wallace Environmental"            "Yugo"
```

```r
length(unique(makes))
```

```
[1] 128
```

```r
distinct(vehicles, make) # returns a tibble
```

```
# A tibble: 128 x 1
   make
   <chr>
 1 Acura
 2 Alfa Romeo
 3 AM General
 4 American Motors Corporation
 5 ASC Incorporated
 6 Aston Martin
```

```
 7 Audi
 8 Aurora Cars Ltd
 9 Autokraft Limited
10 Azure Dynamics
# i 118 more rows
```

```r
nrow(distinct(vehicles, make))
```

```
[1] 128
```

e) Filter the data set for vehicles manufactured in 1997. Do it first with base R, then with dplyr alone, then with dplyr and piping.

```r
# With base R:
cars_1997 <- vehicles[vehicles$year == 1997, ]
```

```r
# With dplyr:
cars_1997 <- filter(vehicles, year == 1997)
```

```r
# With dplyr and piping:
cars_1997 <- vehicles %>%
  filter(year == 1997)
```

f) Arrange (sort, order) the 1997 cars by highway (hwy) gas milage (in increasing order). Do it first with base R, then with dplyr alone, then with dplyr and piping.

- Hint: In base R, use the order() function to get a vector of indices in order by value.

```r
# With base R:
cars_1997_byhwy <- cars_1997[order(cars_1997$hwy), ]
```

```r
# With dplyr:
cars_1997_byhwy <- arrange(cars_1997, hwy)
```

```r
# With dplyr and piping:
cars_1997_byhwy <- cars_1997 %>%
  arrange(hwy)
```

g) Mutate the ordered 1997 cars data frame to add a column average that holds the average gas milage (between city and highway mpg) for each car. Do it first with base R, then with dplyr alone, then with dplyr and piping.

```
# With base R:
cars_1997_byhwy_av <- cars_1997_byhwy
cars_1997_byhwy_av$average <- (cars_1997_byhwy_av$hwy + cars_1997_byhwy_av$cty) / 2
```

```
# With dplyr:
cars_1997_byhwy_av <- mutate(cars_1997_byhwy, average = (hwy + cty) / 2)
```

```
# With dplyr and piping:
cars_1997_byhwy_av <- cars_1997_byhwy %>%
  mutate(average = (hwy + cty) / 2)
```

h) Filter the whole vehicles data set for 2-Wheel Drive vehicles that get more than 20 miles/gallon in the city. Save this new data frame in a variable. Do it first with base R, then with dplyr alone, then with dplyr and piping.

```
# With base R:
two_wheel_20_mpg <- vehicles[vehicles$drive == "2-Wheel Drive" & vehicles$cty > 20, ]
```

```
# With dplyr:
two_wheel_20_mpg <- filter(vehicles,
                           drive == "2-Wheel Drive",
                           cty > 20
)
```

```
# With dplyr and piping:
two_wheel_20_mpg <- vehicles %>%
  filter(drive == "2-Wheel Drive") %>%
  filter(cty > 20)
```

i) Of the above vehicles, what is the vehicle ID of the vehicle with the worst (i.e., smallest) hwy mpg? Do it first with base R, then with dplyr alone, then with dplyr and piping.

- Hint: filter for the worst vehicle, then select its ID.

```
# With base R:
worst_hwy <- two_wheel_20_mpg$id[two_wheel_20_mpg$hwy == min(two_wheel_20_mpg$hwy)] # Noti
```

```r
# With dplyr:
filtered <- filter(two_wheel_20_mpg, hwy == min(hwy))
worst_hwy <- select(filtered, id)
```

```r
# With dplyr and piping:
worst_hwy <- two_wheel_20_mpg %>%
  filter(hwy == min(hwy)) %>%
  select(id)
```

j) Write a function that takes a year_choice and a make_choice as parameters, and returns the vehicle model that has the best (i.e., highest) hwy miles/gallon of vehicles of that make in that year. You'll need to filter more (and do some selecting)! Do it first with base R, then with dplyr alone, then with dplyr and piping.

```r
# With base R:
make_year_filter <- function(make_choice, year_choice) {
  filtered <- vehicles[vehicles$make == make_choice & vehicles$year == year_choice, ]
  filtered[filtered$hwy == max(filtered$hwy), "model"]
}
```

```r
# With dplyr:
make_year_filter1 <- function(make_choice, year_choice) {
  filtered <- filter(vehicles,
                     make == make_choice,
                     year == year_choice)
  filtered <- filter(filtered, hwy == max(hwy))
  select(filtered, model)
}
```

```r
# With dplyr and piping:
make_year_filter2 <- function(make_choice, year_choice) {
  vehicles %>%
    filter(make == make_choice,
           year == year_choice) %>%
    filter(hwy == max(hwy)) %>%
    select(model)
}
```

k) What was the most efficient Honda model of 1995 ? (Use your function!)

```r
make_year_filter("Honda", 1995)
```

```
# A tibble: 1 x 1
  model
  <chr>
1 Civic HB VX
```

```r
make_year_filter1("Honda", 1995)
```

```
# A tibble: 1 x 1
  model
  <chr>
1 Civic HB VX
```

```r
make_year_filter2("Honda", 1995)
```

```
# A tibble: 1 x 1
  model
  <chr>
1 Civic HB VX
```

4.2 Reshaping data with tidyr

Exercise 4.2. tidyr

Exercise 4.2 – Task 1: Plotting Time Series of Weights

Consider the following toy data set of weight time series per person:

```r
#install.packages("tidyr")
library(tidyr)

name <- c('ann', 'bob', 'charlie')
jan <- c(102, 155, 211)
feb <- c(112, 150, 211)
mar <- c(123, 147, 213)
apr <- c(130, 140, 210)
wts <- tibble(name=name, jan=jan, feb=feb, mar=mar, apr=apr)
```

a) Copy / paste it in your R-script, view it and answer the following questions:

- What is the observed event?

  A person has a weight in a specific month. Example: Ann weighs 102 pounds in January. (Note: Probably it's always measured on the same day of each month!)

- What are the recorded aspects of the event?

  (1) Name, (2) Month, (3) Weight

- Is this data set tidy or messy?

  messy

- If messy, describe in words how a tidy version of the data would look.

  (1) Keep name as a column

  (2) Create a column month. Its values are the current column names of columns 2-5.

  (3) Create a column weight. Its values are the current values of these current columns.

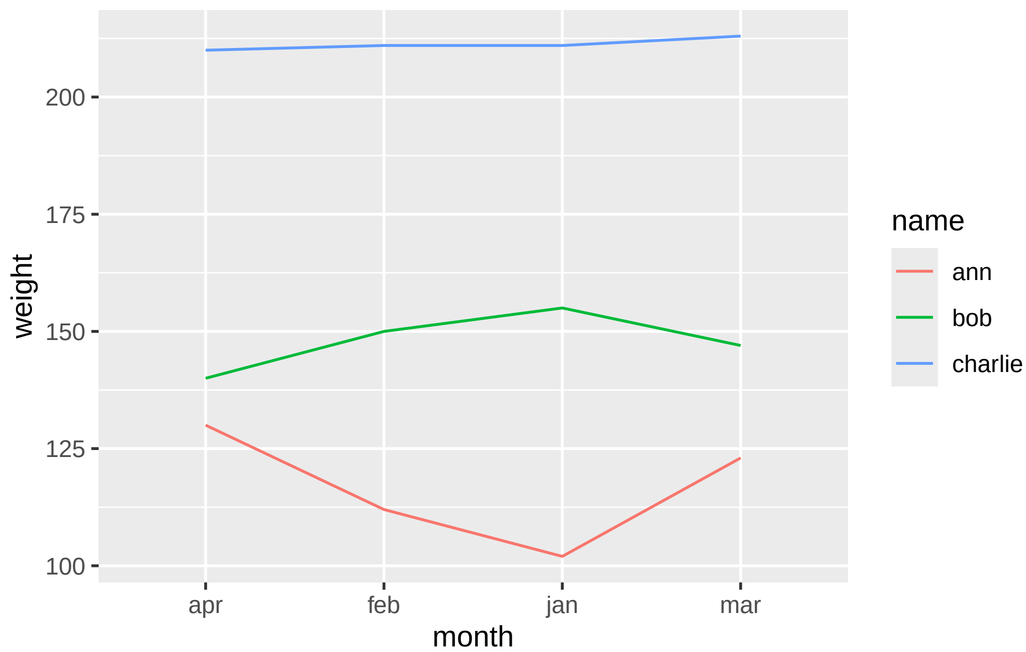
b) Tidy up the data set using pivot_longer(). Store the result in a new data frame called wts_tidy.

```
wts_tidy <- wts %>% pivot_longer(cols = jan:apr,  # messy part
                                 names_to = "month", # headers as values for month
                                 values_to = "weight" # values as vales for weight
                                 )
```

c) Use geom_line() to plot the time series of weights per person. Hints:

- Map month to the x-axes.
- Map weight to the y-axes.
- Map name to the color scale.
- Additionally, use the argument group = name within the aesthetic of geom_line() to group observations by person. Otherwise, geom_line() tries to connect all obersvations with a single line, which does not work.

```
wts_tidy %>%
  ggplot() +
  geom_line(mapping = aes(x = month,
                          y = weight,
                          col = name,
                          group = name))
```



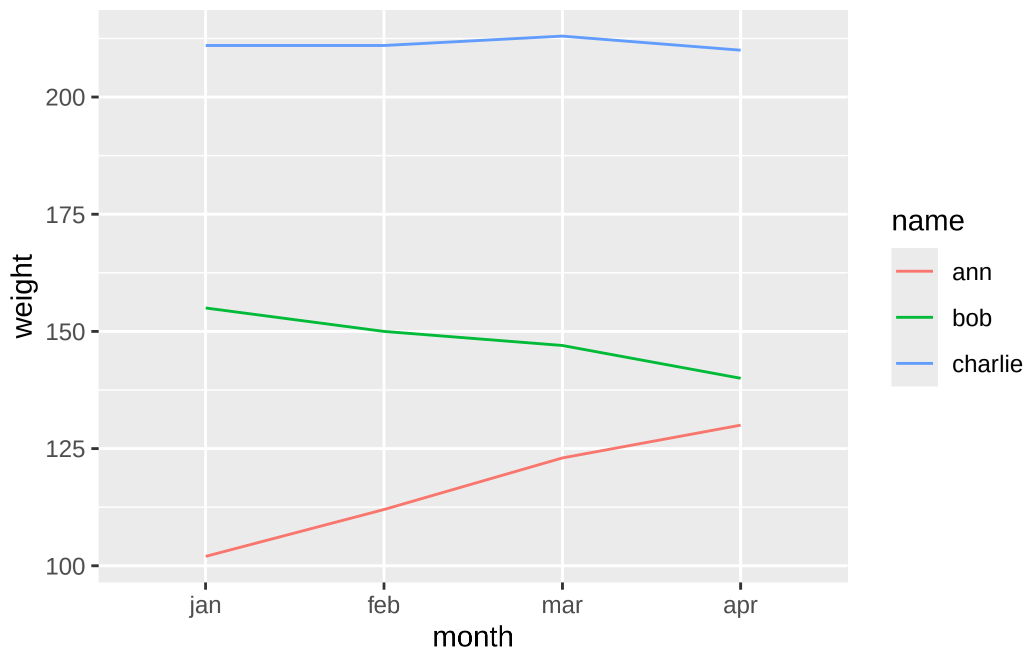
d) Notice that the months in your x-axes are ordered alphabetically. That's not the order we want! To change that, use mutate() to change the column month from integer to "ordered factor". Hint:

- An "ordered factor" is a normal factor, but with an order that we define manually.
- do that, use the arguments ordered and level as follows: factor(month, ordered = TRUE, levels = c('jan', 'feb', 'mar', 'apr'))

```
wts_tidy <- wts_tidy %>%
  mutate(month = factor(month,
                        ordered = TRUE,
                        levels = c('jan', 'feb', 'mar', 'apr')))
```

e) Now redo the plot. The months will now appear in the order you specified above.

```
wts_tidy %>%
  ggplot() +
  geom_line(mapping = aes(x = month,
                          y = weight,
                          col = name,
                          group = name))
```



f) Now calculate the minimal, maximal and average weight per person.

- Hint: Use group_by() and summarize() from dplyr.

```
wts_tidy_agg <- wts_tidy %>%
  group_by(name) %>%
  summarize(min = min(weight),
            max = max(weight),
            avg = mean(weight)
            )
```

Exercise 4.2 – Task 2: German Car Manufacturers

The following (made-up) data set lists different German car manufacturers. It reports how many models with a specified number of cylinders have been built per manufacturer.

```
set.seed(3)
cars <- tibble( manufacturer = c("Audi", "BMW",
                  "Mercedes", "Opel", "VW"),
                              `3 cyl` = sample(20, 5, replace = TRUE),
                              `4 cyl` = sample(50:100, 5, replace = TRUE),
                              `5 cyl` = sample(10, 5, replace = TRUE),
```

```
                                                  `6 cyl` = sample(30:50, 5, replace = TRUE),
                                                  `8 cyl` = sample(20:40, 5, replace = TRUE),
                                                  `10 cyl` = sample(10, 5, replace = TRUE),
                                                  `12 cyl` = sample(20, 5, replace = TRUE),
                                                  `16 cyl` = rep(0, 5)
)
```

a) Copy / paste the above code in your R-script, view the data set, and answer the following questions:

- What is the observed event?

  A manufacturer produces a certain number of cars with a certains number of cylinders. (Example: Audi builds 5 cars with 3 cylinders.)

- What are the recorded aspects of the event?

  (1) manufacturer, (2) number of cylinders, (3) number of cars

- Is this data set tidy or messy?

  messy

- If messy, describe in words how a tidy version of the data would look.

  (1) Keep 'manufacturer' as a column.

  (2) Create a column 'cyl'. Its values are the current column names of columns 2-5.

  (3) Create a column 'freq'. Its values are the current values of columns 2-5.

b) Tidy up the data set using pivot_longer(). Store the result in a new data frame cars_tidy.

```
cars_tidy <- cars %>% pivot_longer(cols = -manufacturer,
                                   names_to = "cyl",
                                   values_to = "freq")
```

c) Use geom_col() to create a bar plot that shows the frequency per cylinder. Use facet_wrap() to create one such plot per manufacturer. Use ggplotly() to make it interactive.

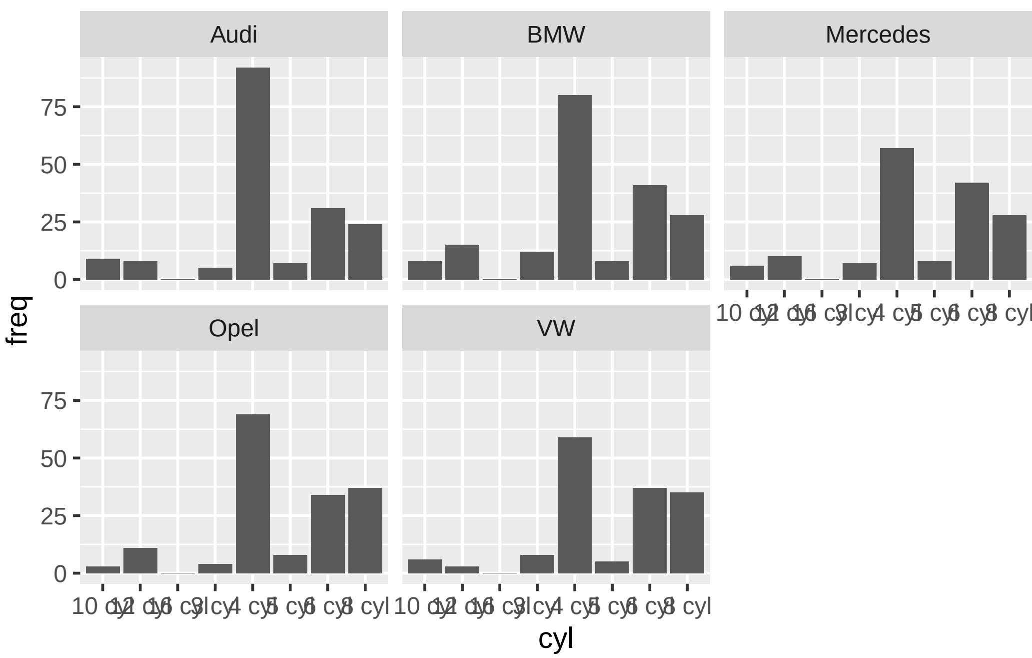
- Hint: Don't forget to load the library plotly.

```
library(plotly)
```

```
Attache Paket: 'plotly'
```

```
Das folgende Objekt ist maskiert 'package:ggplot2':

    last_plot


Das folgende Objekt ist maskiert 'package:stats':

    filter


Das folgende Objekt ist maskiert 'package:graphics':

    layout
```

```
p_tidy_cars <- cars_tidy %>% ggplot() +
  geom_col(mapping = aes(x = cyl,
                         y = freq)) +
  facet_wrap(~ manufacturer)

p_tidy_cars # not interactive
```




```
#ggplotly(p_tidy_cars) # does not work as pdf output
```

d) Notice that the number of cylinders is not in a natural order. To change that, use mutate() to change the data type of the variable cyl. To do that, you have 2 options:

1. You can either convert the variable cyl in an ordered factor,
2. or you can use gsub("\D", " ", cyl) and as.numeric() to extract the numbers from the strings.

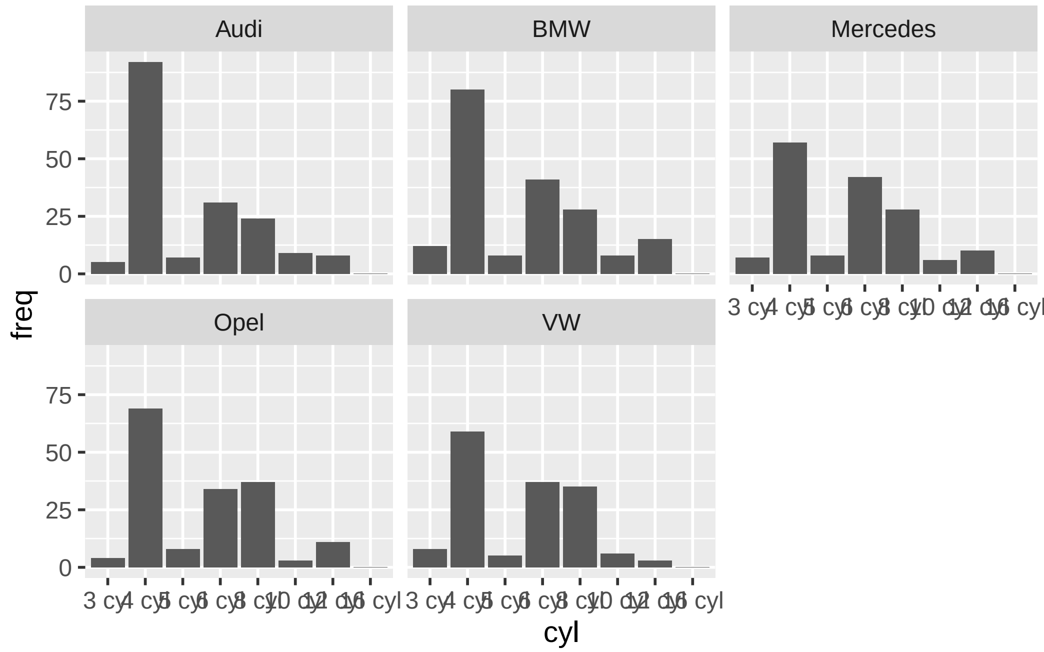
Try out both options!

```
# Option 1: converting it 'cyl' in an ordered factor:
cars_tidy1 <- cars_tidy %>%
  mutate(cyl = factor(cyl,
                      ordered = TRUE,
                      levels = c("3 cyl", "4 cyl", "5 cyl", "6 cyl", "8 cyl", "10 cyl", "1
                     )
        )
```

```
# Option 2: using gsub() to extract the numbers from the strings
cars_tidy2 <- cars_tidy %>%
  mutate(cyl = as.numeric(gsub("\\D", "", cyl))) # All non-numbers (\\D) are replaced by t
```

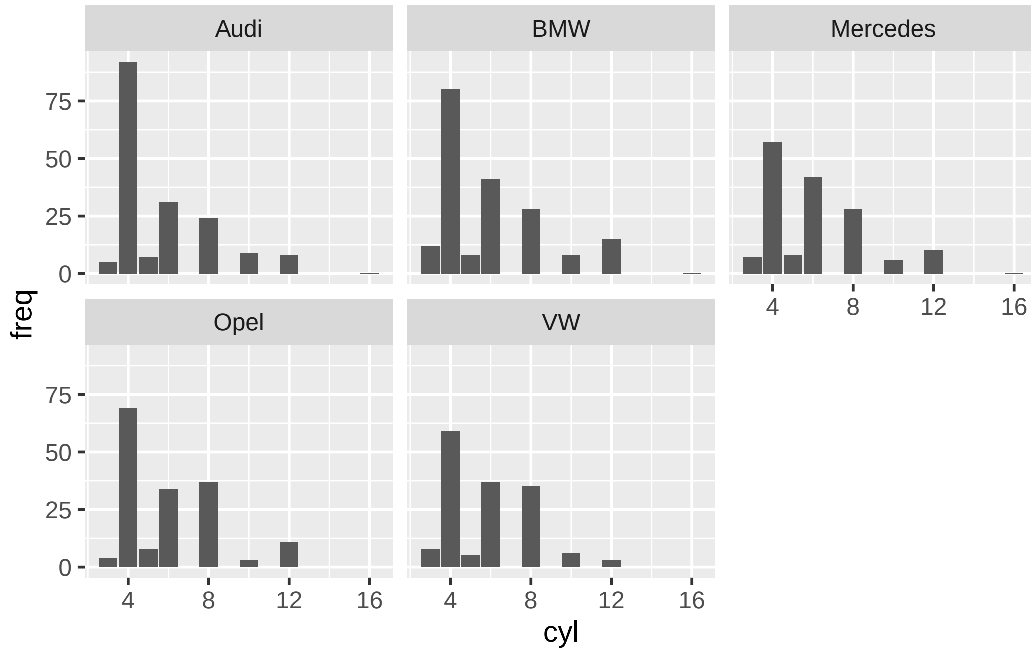
e) Redo the plot for both options.

```
p_tidy_cars1 <- cars_tidy1 %>% ggplot() +
  geom_col(mapping = aes(x = cyl,
                         y = freq)) +
  facet_wrap(~ manufacturer)

p_tidy_cars1  # not interactive
```




```
#ggplotly(p_tidy_cars1) # does not work as pdf output
```

```
p_tidy_cars2 <- cars_tidy2 %>% ggplot() +
  geom_col(mapping = aes(x = cyl,
                         y = freq)) +
  facet_wrap(~ manufacturer)

p_tidy_cars2  # not interactive
```




```
#ggplotly(p_tidy_cars2) # does not work as pdf output
```

- Do you notice a difference?

  \- *When using gsub(), cyl is converted to numbers, and thus, ggplot puts a number scale on the x-axes.*

  Thus, there is a slot reserved for, e.g., 7 cylinders, even though cars with 7 cylinders do not exist!

  On the other hand, cars with 16 cylinders exist, but zero are produced.

  When using this option we cannot distinguish between 'non-existing' and 'zero'!

  \- *This does not happen when we use ordered factors.*

- Which option is better for visualization?

  \- *Thus, ordered factors are the better option for visualization!*

Self-Study 4.2. tidyr

Self-Study 4.2: Analyzing Avocado Sales with tidyr and dplyr

a) Load the packages tidyr, dplyr, and ggplot2. Download the avocado.csv file from GitHub and load it into a variable avocados. Get a first impression of the data using View() and str().

```r
avocados <- read.csv("avocado.csv")

str(avocados)
```

```
'data.frame':   18249 obs. of  14 variables:
 $ X           : int  0 1 2 3 4 5 6 7 8 9 ...
 $ Date        : chr  "2015-12-27" "2015-12-20" "2015-12-13" "2015-12-06" ...
 $ AveragePrice: num  1.33 1.35 0.93 1.08 1.28 1.26 0.99 0.98 1.02 1.07 ...
 $ Total.Volume: num  64237 54877 118220 78992 51040 ...
 $ X4046       : num  1037 674 795 1132 941 ...
 $ X4225       : num  54455 44639 109150 71976 43838 ...
 $ X4770       : num  48.2 58.3 130.5 72.6 75.8 ...
 $ Total.Bags  : num  8697 9506 8145 5811 6184 ...
 $ Small.Bags  : num  8604 9408 8042 5677 5986 ...
 $ Large.Bags  : num  93.2 97.5 103.1 133.8 197.7 ...
 $ XLarge.Bags : num  0 0 0 0 0 0 0 0 0 0 ...
 $ type        : chr  "conventional" "conventional" "conventional" "conventional" ...
 $ year        : int  2015 2015 2015 2015 2015 2015 2015 2015 2015 2015 ...
 $ region      : chr  "Albany" "Albany" "Albany" "Albany" ...
```

```r
View(avocados)
```

b) From str(), you can see that the Date column is of type char. To tell R to treat the Date column as a date and not as a string, transform that column using the as.Date() function.

- Hint: You can use mutate().

```r
avocados <- avocados %>%
  mutate(Date = as.Date(Date))

str(avocados)
```

```
'data.frame':   18249 obs. of  14 variables:
 $ X           : int  0 1 2 3 4 5 6 7 8 9 ...
 $ Date        : Date, format: "2015-12-27" "2015-12-20" ...
 $ AveragePrice: num  1.33 1.35 0.93 1.08 1.28 1.26 0.99 0.98 1.02 1.07 ...
 $ Total.Volume: num  64237 54877 118220 78992 51040 ...
```

```
 $ X4046       : num  1037 674 795 1132 941 ...
 $ X4225       : num  54455 44639 109150 71976 43838 ...
 $ X4770       : num  48.2 58.3 130.5 72.6 75.8 ...
 $ Total.Bags  : num  8697 9506 8145 5811 6184 ...
 $ Small.Bags  : num  8604 9408 8042 5677 5986 ...
 $ Large.Bags  : num  93.2 97.5 103.1 133.8 197.7 ...
 $ XLarge.Bags : num  0 0 0 0 0 0 0 0 0 0 ...
 $ type        : chr  "conventional" "conventional" "conventional" "conventional" ...
 $ year        : int  2015 2015 2015 2015 2015 2015 2015 2015 2015 2015 ...
 $ region      : chr  "Albany" "Albany" "Albany" "Albany" ...
```

c) The file has some uninformative column names. Rename these columns:

- X4046 to small_haas
- X4225 to large_haas
- X4770 to xlarge_haas

These are the sales volumes of haas avocados.

```r
avocados <- avocados %>%
  rename(small_haas = X4046,
         large_haas = X4225,
         xlarge_haas = X4770)

str(avocados)
```

```
'data.frame':	18249 obs. of  14 variables:
 $ X           : int  0 1 2 3 4 5 6 7 8 9 ...
 $ Date        : Date, format: "2015-12-27" "2015-12-20" ...
 $ AveragePrice: num  1.33 1.35 0.93 1.08 1.28 1.26 0.99 0.98 1.02 1.07 ...
 $ Total.Volume: num  64237 54877 118220 78992 51040 ...
 $ small_haas  : num  1037 674 795 1132 941 ...
 $ large_haas  : num  54455 44639 109150 71976 43838 ...
 $ xlarge_haas : num  48.2 58.3 130.5 72.6 75.8 ...
 $ Total.Bags  : num  8697 9506 8145 5811 6184 ...
 $ Small.Bags  : num  8604 9408 8042 5677 5986 ...
 $ Large.Bags  : num  93.2 97.5 103.1 133.8 197.7 ...
 $ XLarge.Bags : num  0 0 0 0 0 0 0 0 0 0 ...
 $ type        : chr  "conventional" "conventional" "conventional" "conventional" ...
 $ year        : int  2015 2015 2015 2015 2015 2015 2015 2015 2015 2015 ...
 $ region      : chr  "Albany" "Albany" "Albany" "Albany" ...
```

d) The data only holds total sales volumes (Total.Volume) and the sales volumes for haas avocados (small_haas, large_haas, xlarge_haas), but there are also other avocados included in Total.Volume. Double-check this by summing up haas avocado sales and comparing the sum with the total sales value.

```
sum(avocados$Total.Volume - (avocados$small_haas + avocados$large_haas + avocados$xlarge_h
```

```
[1] 18237
```

e) Create a new column other_avos that is the Total.Volume minus all haas avocados (small, large, xlarge).

```
avocados <- avocados %>%
  mutate(other_avos = Total.Volume - small_haas - large_haas - xlarge_haas)
```

f) To perform analysis by avocado size, create a dataframe by_size that has only Date, other_avos, small_haas, large_haas, xlarge_haas.

- Note: other_avos is not strictly a size, but we ignore this. We may view it as the bin that holds the sales volumes of avocados of size "unknown".

```
by_size <- avocados %>%
  select(Date,
         other_avos,
         small_haas,
         large_haas,
         xlarge_haas)
```

g) Use head() to view the first few lines of your dataframe by_size.

- Is it tidy or messy?

  messy

- What is the observed event?

  At a certain date, a number of avocados of a certain size are sold.

- What are the recorded aspects?

  (1) date, (2) size, (3) number sold.

- How would a tidy version of the data look?

  (1) keep 'date' column.

  (2) create a 'size' column that holds the column names `other_avos`, `small_haas`, `large_haas`, `xlarge_haas` as values.

  (3) create a 'volume' column that holds the sales volumes.

```r
head(by_size)
```

```
        Date other_avos small_haas large_haas xlarge_haas
1 2015-12-27    8696.87    1036.74   54454.85       48.16
2 2015-12-20    9505.56     674.28   44638.81       58.33
3 2015-12-13    8145.35     794.70  109149.67      130.50
4 2015-12-06    5811.16    1132.00   71976.41       72.58
5 2015-11-29    6183.95     941.48   43838.39       75.78
6 2015-11-22    6683.91    1184.27   48067.99       43.61
```

h) Tidy it up using pivot_longer(). Store the result in a new data frame by_size_tidy. Hints:

- The four column names other_avos, small_haas, large_haas, xlarge_haas go into a new column called size.
- The volumes of sales (currently stored in each of the above columns) go to a new column called volume.

```r
by_size_tidy <- by_size %>%
  pivot_longer(cols = -Date,
               names_to = "size",
               values_to = "volume"
               )

head(by_size_tidy)
```

```
# A tibble: 6 x 3
  Date       size        volume
  <date>     <chr>        <dbl>
1 2015-12-27 other_avos   8697.
2 2015-12-27 small_haas   1037.
3 2015-12-27 large_haas  54455.
4 2015-12-27 xlarge_haas    48.2
5 2015-12-20 other_avos   9506.
6 2015-12-20 small_haas    674.
```

i) The shape of by_size_tidy is not only tidier, but it also facilitates the visualization of sales over time by size: Use ggplot2 with geom_smooth() to plot a smoothed trendline of sales volumes over time – make one trendline for each size. Hints:

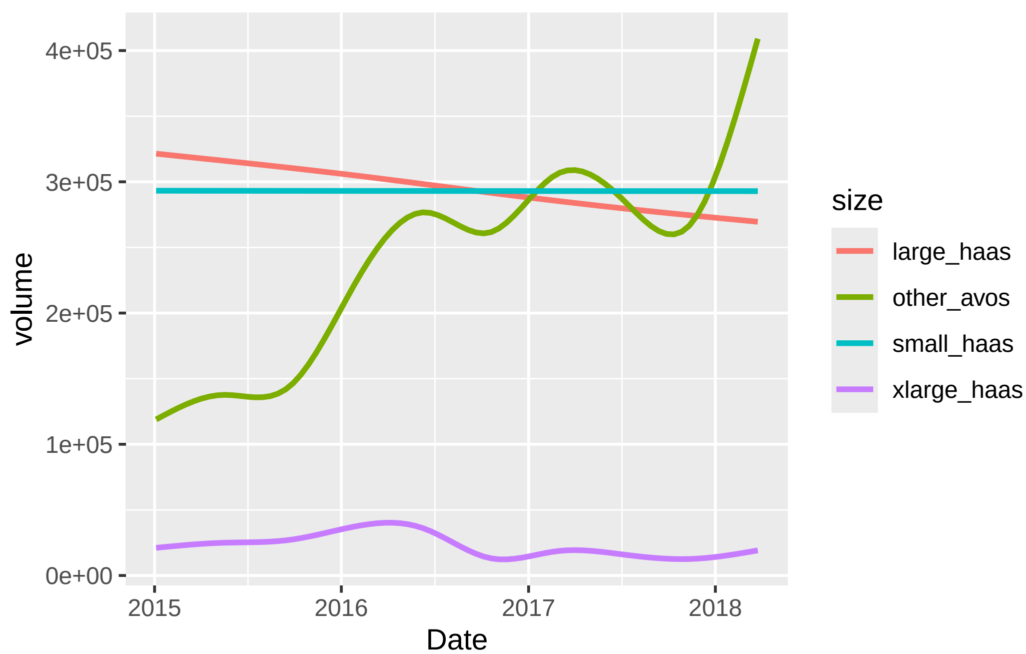
- Map the Date to the x-axes, map the volume to the y-axes, map the size to the colour scale.
- Inside of geom_smooth(), you can set the argument se = F to hide the confidence bands around the trendlines.)

Bonus:

- To see the advantage of this shape for plotting sales over time by size, try to produce the same plot using the data frame by_size instead of the data frame by_size_tidy.

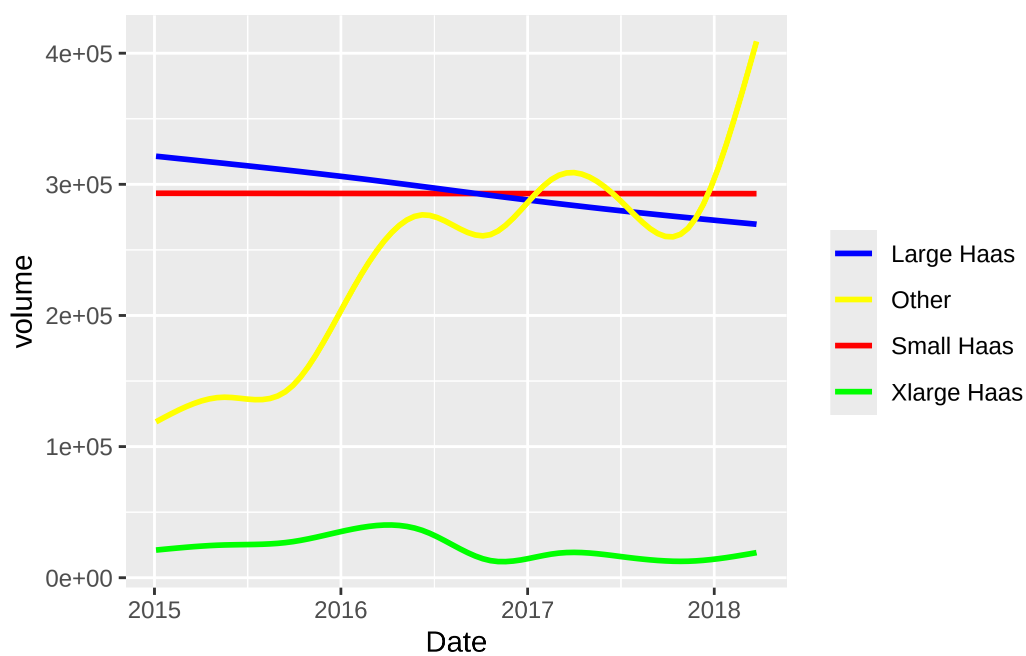
```
ggplot(by_size_tidy) +
  geom_smooth(mapping = aes(x = Date,
                            y = volume,
                            col = size),
              se = F) # Don't display the confidence intervals around the smoothed conditi
```

```
`geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



```r
by_size %>% ggplot() +
  geom_smooth(mapping = aes(x = Date,
                            y = small_haas, color = "Small Haas"), se = F) +    # specify
  geom_smooth(mapping = aes(x = Date,
                            y = large_haas, color = "Large Haas"), se = F) +
  geom_smooth(mapping = aes(x = Date,
                            y = xlarge_haas, color = "Xlarge Haas"), se = F) +
  geom_smooth(mapping = aes(x = Date,
                            y = other_avos, color = "Other"), se = F) +
  labs(x = "Date", y = "volume") + # Specify the title for the axes
  scale_color_manual(name = "", values = c("Small Haas" = "red", "Large Haas" = "blue", "X
```

```
`geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
`geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
`geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
`geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```



j) Now use by_size_tidy to compute the average sales volume per size.

- Hint: First group by size using group_by(), then compute the average using summarize().

```r
average_sales <- by_size_tidy %>%
  group_by(size) %>%
  summarize(avg_volume = mean(volume))

print(average_sales)
```

```
# A tibble: 4 x 2
  size        avg_volume
  <chr>            <dbl>
1 large_haas     295155.
2 other_avos     239641.
3 small_haas     293008.
4 xlarge_haas     22840.
```

k) We can also investigate sales by avocado type (conventional, organic).

- To do this, consider again the original avocados data frame.
- Group it by Date and type, and
- calculate the sum of the column Total.Volume for each group.
- Store the result in a new data frame called by_type.

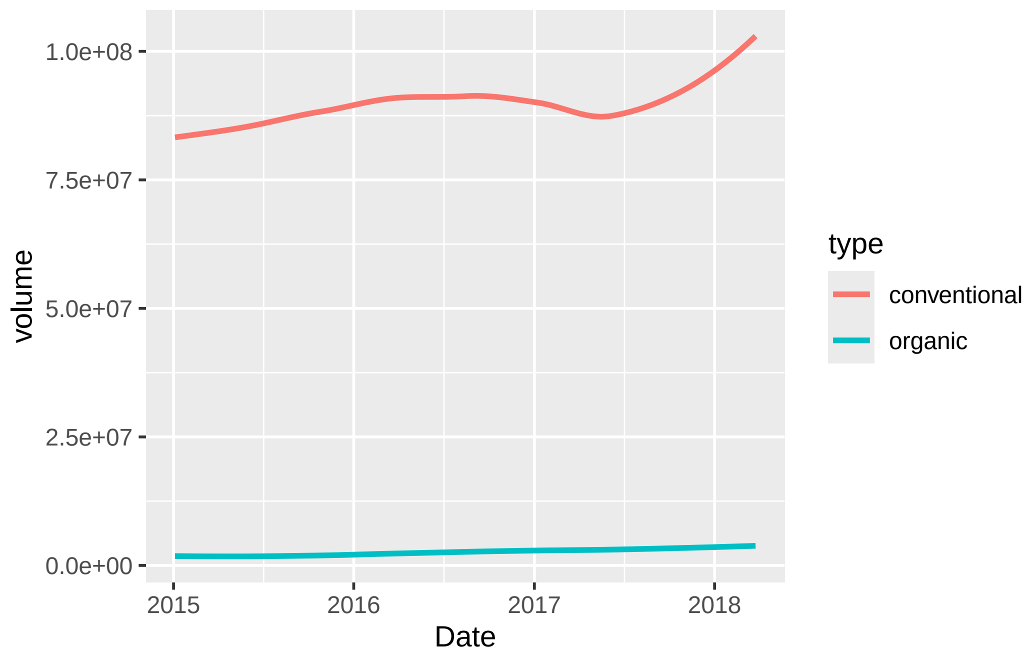
```r
by_type <- avocados %>%
  group_by(Date, type) %>%
  summarise(volume = sum(Total.Volume), .groups = 'drop') # .groups = 'drops' needed, with
```

l) This data set is already tidy. Visualize the avocado sales over time by type using geom_smooth().

- Note: This is completely analogous to the plot you did before!

```r
ggplot(by_type) +
  geom_smooth(mapping = aes(x = Date,
                            y = volume,
                            col = type),
              se = F)
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

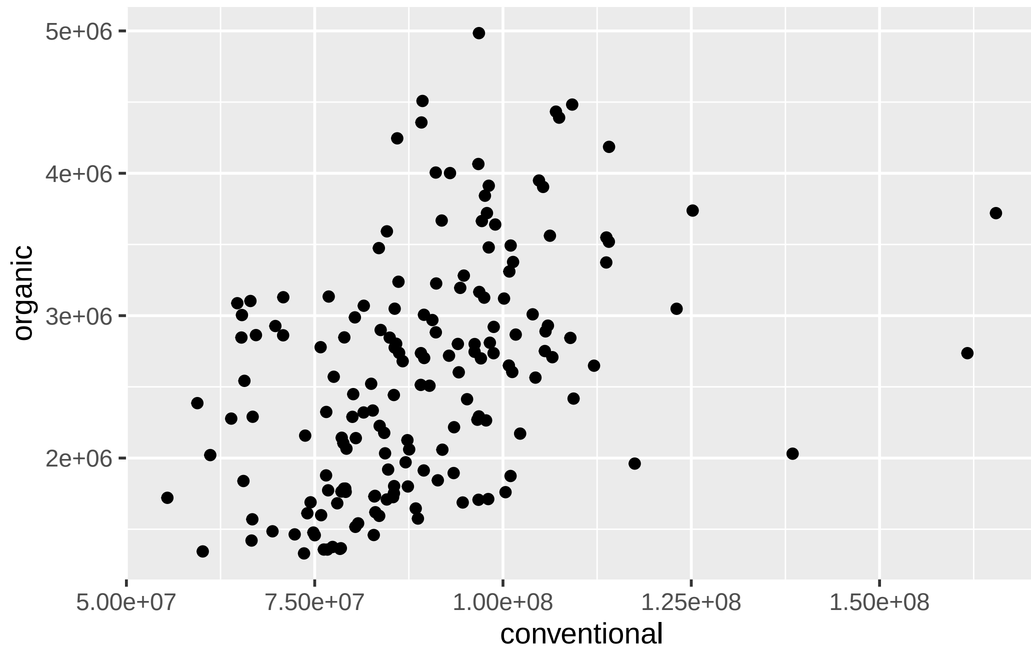


m) From the above plot we see that the sales volumes of both avocado types seem to increase over the years. Now let's see if we can (visually) confirm this correlation in a scatterplot: if our assumption is correct, we should see a linear correlation between conventional and organic sales. Create this scatterplot using ggplot2. Hints:

- In order to check for a linear correlation between the types, we must map conventional sales to the x-axes and organic sales to the y-axes.
- Yet, in the data frame by_type the sales numbers for both avocado types are mingled in one column, namely volume.
- To facilitate the plotting, it would be good to have one column per type, each holding the respective sales numbers. Then we could simply map each column to an axes.
- To achieve this, reformat the data frame by_type using pivot_wider(). Store the result in a new data frame called by_type_wide.
- Now use ggplot2 with geom_point() to generate the scatterplot. Does it confirm our assumption?

```
by_type_wide <- by_type %>%
  pivot_wider(names_from = type,
              values_from = volume)
```

```
ggplot(by_type_wide) +
  geom_point(mapping = aes(x = conventional,
                           y = organic))
```



As expected, the scatter plot shows some linear correlation between the sales numbers, but it is not too strong. This was expected as well: We could already see in the temporal plot that conventional sales vary much stronger than organic sales, which is reflected in the relatively wide spread of points.