R Crashcourse

Fishy Data and Aquatic Insights

Anıl Axel Tellbüscher

2024-08-26

The perfect working directory

```
|__README.Rmd
|__data
|__code
|__output
   |__interm
   |__plots
   |__tables
```

- a well-structured repository keeps your mind clear!
- RStudio Project files make code more portable
- GitHub helps with tracking changes and collaborative work!

Some Markdown magic

- Markdown or Quarto helps you to document your work

Windows	macOS	Function
	`CMD` + `OPT` + `I`	create new code chunk
	`CMD` + `SHIFT` + `M`	create pipe operator `%>%`

Getting data into R

Clipboard

The option for “quick and dirty” work

```r
# macOS
data <- read.delim(pipe("pbpaste"))

# Windows
data <- read.delim("clipboard")
```

Dedicated reading functions

Reading almost everything

```
1 library(readr) # for almost everything
```

Reading Excel

```
1 library(readxl) # for Excel files
```

Data needs a place to live

Where does your data live?

Let’s find out!

Check the working directory (WD)

```
1  getwd()
```

```
[1] "/Users/anilaxeltellbuescher/Projekte/EAS R-Course/presentations"
```

Working with absolute file paths

```
1 data <- read_csv("/Users/anilaxeltellbuescher/Projekte/Rintro/myfil
```

- inflexible
- code tends to break
- **when using Project files, RStudio sets the working directory automatically to the location where the file is!**

Better: relative file paths!

```
1 # alternative way of calling a function from an installed package
2 readxl::read_excel("../data/example_data.xlsx", sheet = "tidy")
```

```
# A tibble: 6 × 5
  treatment length unit_length weight unit_weight
  <chr>      <dbl> <chr>         <dbl> <chr>
1 A             34 cm              234 g
2 A             31 cm              311 g
3 A             30 cm              276 g
4 B             25 cm              299 g
5 B             28 cm              304 g
6 B             29 cm              332 g
```

MUCH better: here()

- troubles with portability of code are common because of the way how file paths in different operating systems are built.
- the `here` package offers a solution by adjusting the file path according to your operating system!

```
1 library(here)
2 readxl::read_excel(here("data", "example_data.xlsx"), sheet = "tidy
```

```
# A tibble: 6 × 5
  treatment length unit_length weight unit_weight
  <chr>      <dbl> <chr>         <dbl> <chr>
1 A             34 cm              234 g
2 A             31 cm              311 g
3 A             30 cm              276 g
4 B             25 cm              299 g
5 B             28 cm              304 g
6 B             29 cm              332 g
```

Working with data

In a typical data analysis, most (~80%) of our time is consumed by data cleaning.

The time demand can be reduced by doing the following things:

- think about your data BEFORE you start collecting it
  - type? (numeric, character,…)
  - how many tables?
  - common identifier?
- follow the key principles of tidy data

Tidy data 1

Hadley Wickham - Tidy Data

What is tidy data?

- Each variable forms a column.
- Each observation forms a row.
- Each type of observational unit forms a table.

Tidy data 2

What is messy data?

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.
- Multiple types of observational units are stored in the same table.
- A single observational unit is stored in multiple tables.

Joining 1

Imagine you have two tables with a common identifier, e.g., fish tank.

- table 1: water quality data

```r
table1 <- data.frame(
  tank = rep(c(1,2), each = 3),
  date = rep(c(1,2,3), times = 2),
  pH = c(7.7,6.8,7.5,7.6,7.5,7.5)
)
```

Joining 2

- table 2: feeding data

```r
table2 <- data.frame(
  tank = rep(c(1,2), each = 3),
  date = rep(c(1,2,3), times = 2),
  feed_g = c(200,120,120,120,120,120)
  )
```

Joining 3

Now let’s combine the data!

```
1  library(tidyverse)
2
3  table1 %>%
4    left_join(table2)
```

```
  tank date  pH feed_g
1    1    1 7.7    200
2    1    2 6.8    120
3    1    3 7.5    120
4    2    1 7.6    120
5    2    2 7.5    120
6    2    3 7.5    120
```

Example data 1

R comes with many example datasets which we can use to play with.

```
data() # list all datasets provided by default
```

Example data 2

Let's check the `diamonds` dataset.

```
          1  diamonds
```

```
# A tibble: 53,940 × 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
 7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
 8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
 9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
# ℹ 53,930 more rows
```

Example data 3

```
1 glimpse(diamonds)
```

```
Rows: 53,940
Columns: 10
$ carat   <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, 0.23,
0.…
$ cut     <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very Good,
Ver…
$ color   <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, J, J,
I,…
$ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI1, VS1,
…
$ depth   <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, 59.4,
64…
$ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54, 62,
58…
$ price   <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339, 340,
```

Example data 4

```
1 summary(diamonds)
```

```
     carat               cut        color        clarity          depth      
 Min.   :0.2000   Fair      : 1610   D: 6775   SI1     :13065   Min.   :43.00  
 1st Qu.:0.4000   Good      : 4906   E: 9797   VS2     :12258   1st Qu.:61.00  
 Median :0.7000   Very Good:12082   F: 9542   SI2     : 9194   Median :61.80  
 Mean   :0.7979   Premium   :13791   G:11292   VS1     : 8171   Mean   :61.75  
 3rd Qu.:1.0400   Ideal     :21551   H: 8304   VVS2    : 5066   3rd Qu.:62.50  
 Max.   :5.0100                     I: 5422   VVS1    : 3655   Max.   :79.00  
                                    J: 2808   (Other): 2531                  
     table           price             x                y        
 Min.   :43.00   Min.   :  326   Min.   : 0.000   Min.   : 0.000  
 1st Qu.:56.00   1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720  
 Median :57.00   Median : 2401   Median : 5.700   Median : 5.710  
 Mean   :57.46   Mean   : 3933   Mean   : 5.731   Mean   : 5.735  
 3rd Qu.:59.00   3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540  
 Max.   :95.00   Max.   :18823   Max.   :10.740   Max.   :58.900  
```

Pivoting 1

We can also transform data from wide to long format. This is important for plotting and statistical analyses.

```
my_diamonds <- diamonds %>%
  pivot_longer(x:z, names_to = "random_letters", values_to = "value
  print()
```

```
# A tibble: 161,820 × 9
   carat cut       color clarity depth table price random_letters values
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <chr>            <dbl>
 1  0.23 Ideal     E     SI2      61.5    55   326 x                 3.95
 2  0.23 Ideal     E     SI2      61.5    55   326 y                 3.98
 3  0.23 Ideal     E     SI2      61.5    55   326 z                 2.43
 4  0.21 Premium   E     SI1      59.8    61   326 x                 3.89
 5  0.21 Premium   E     SI1      59.8    61   326 y                 3.84
 6  0.21 Premium   E     SI1      59.8    61   326 z                 2.31
 7  0.23 Good      E     VS1      56.9    65   327 x                 4.05
 8  0.23 Good      E     VS1      56.9    65   327 y                 4.07
 9  0.23 Good      E     VS1      56.9    65   327 z                 2.31
10  0.29 Premium   I     VS2      62.4    58   334 x                 4.2
# ℹ 161,810 more rows
```

Pivoting 2

We transformed data into the long format, but how to transform long data into wide data?

```
my_diamonds %>%
  group_by(cut, color) %>%
  summarise(mean = mean(carat)) %>%
  pivot_wider(names_from = "cut", values_from = "mean")
```

```
# A tibble: 7 × 6
  color  Fair  Good `Very Good` Premium Ideal
  <ord> <dbl> <dbl>       <dbl>   <dbl> <dbl>
1 D     0.920 0.745       0.696   0.722 0.566
2 E     0.857 0.745       0.676   0.718 0.578
3 F     0.905 0.776       0.741   0.827 0.656
4 G     1.02  0.851       0.767   0.841 0.701
5 H     1.22  0.915       0.916   1.02  0.800
6 I     1.20  1.06        1.05    1.14  0.913
7 J     1.34  1.10        1.13    1.29  1.06
```

Filtering and Selecting

Let’s select diamonds with more than 1 carat and keep only columns we are interested in.

```
diamonds %>%
  filter(carat >= 1) %>%
  select(carat, cut, color, price)
```

```
# A tibble: 19,060 × 4
   carat cut       color price
   <dbl> <ord>     <ord> <int>
 1  1.17 Very Good J      2774
 2  1.01 Premium   F      2781
 3  1.01 Fair      E      2788
 4  1.01 Premium   H      2788
 5  1.05 Very Good J      2789
 6  1.05 Fair      J      2789
 7  1    Premium   I      2795
 8  1.01 Fair      E      2797
 9  1.04 Premium   G      2801
10  1    Premium   J      2801
# ℹ 19,050 more rows
```

Task

Load the mtcars dataset. Extract all rows with mpg greater than 20.

Task

Load the mtcars dataset. Select the gear and carb columns.

Mutating

Let us now calculate the price per carat.

```
1 diamonds %>%
2   mutate(price_per_carat = price/carat)
```

```
# A tibble: 53,940 × 11
   carat cut   color clarity depth table price     x     y     z
price_per_carat
   <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
<dbl>
 1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
1417.
 2  0.21 Prem… E     SI1      59.8    61   326  3.89  3.84  2.31
1552.
 3  0.23 Good  E     VS1      56.9    65   327  4.05  4.07  2.31
1422.
 4  0.29 Prem… I     VS2      62.4    58   334  4.2   4.23  2.63
1152.
 5  0.31 Good  J     SI2      63.3    58   335  4.34  4.35  2.75
1081.
```

Mutating multiple columns

We can also change multiple columns at a time!

```
diamonds %>%
  mutate(across(x:z, ~ . + x)) %>%
  mutate(across(is.numeric, ~ . + x)) # . represents 'all selected
```

```
# A tibble: 53,940 × 10
   carat cut       color clarity depth table price     x     y     z
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  8.13 Ideal     E     SI2      69.4  62.9  334.  15.8  15.8  14.3
 2  7.99 Premium   E     SI1      67.6  68.8  334.  15.6  15.5  14.0
 3  8.33 Good      E     VS1      65    73.1  335.  16.2  16.2  14.5
 4  8.69 Premium   I     VS2      70.8  66.4  342.  16.8  16.8  15.2
 5  8.99 Good      J     SI2      72.0  66.7  344.  17.4  17.4  15.8
 6  8.12 Very Good J     VVS2     70.7  64.9  344.  15.8  15.8  14.3
 7  8.14 Very Good I     VVS1     70.2  64.9  344.  15.8  15.8  14.3
 8  8.4  Very Good H     SI1      70.0  63.1  345.  16.3  16.3  14.7
 9  7.96 Fair      E     VS2      72.8  68.7  345.  15.5  15.4  14.1
10  8.23 Very Good H     VS1      67.4  69    346   16    16.0  14.4
# ℹ 53,930 more rows
```

Control statements

What if…?

```
diamonds %>%
  mutate(origin = rep(c("Botswana", "Kiribati", "Netherlands"),
                      each = nrow(.)/3)) %>%
  mutate(price_plus_danger = if_else(origin == "Netherlands",
                                     price + 10,
                                     price -4))
```

```
# A tibble: 53,940 × 12
   carat cut       color clarity depth table price     x     y     z origin
   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>
 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43 Botswana
 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31 Botswana
 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31 Botswana
 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63 Botswana
 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75 Botswana
 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48 Botswana
 7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47 Botswana
 8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53 Botswana
 9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49 Botswana
10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39 Botswana
# ℹ 53,930 more rows
# ℹ 1 more variable: price_plus_danger <dbl>
```

Groupwise mutating 1

We can also analyse our data groupwise! Let’s do the following:

1. group the diamonds by quality of their cuts
2. rank them by their price
3. sort them according to their cut and rank

Groupwise mutating 2

```
1  diamonds %>%
2    group_by(cut) %>%
3    mutate(rank = min_rank(price)) %>%
4    arrange(cut,rank)
```

```
# A tibble: 53,940 × 11
# Groups:   cut [5]
   carat cut   color clarity depth table price     x     y     z  rank
   <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl> <int>
 1  0.22 Fair  E     VS2      65.1    61   337  3.87  3.78  2.49     1
 2  0.25 Fair  E     VS1      55.2    64   361  4.21  4.23  2.33     2
 3  0.23 Fair  G     VVS2     61.4    66   369  3.87  3.91  2.39     3
 4  0.27 Fair  E     VS1      66.4    58   371  3.99  4.02  2.66     4
 5  0.3  Fair  J     VS2      64.8    58   416  4.24  4.16  2.72     5
 6  0.3  Fair  F     SI1      63.1    58   496  4.3   4.22  2.69     6
 7  0.34 Fair  J     SI1      64.5    57   497  4.38  4.36  2.82     7
 8  0.37 Fair  F     SI1      65.3    56   527  4.53  4.47  2.94     8
 9  0.3  Fair  D     SI2      64.6    54   536  4.29  4.25  2.76     9
10  0.25 Fair  D     VS1      61.2    55   563  4.09  4.11  2.51    10
# ℹ 53,930 more rows
```

Grouping and summarising

Instead of mutating, we can also summarise our data after grouping!

```
diamonds %>%
  group_by(cut) %>%
  summarise(mean_price = mean(price)) %>%
  arrange(desc(mean_price))
```

```
# A tibble: 5 × 2
  cut       mean_price
  <ord>          <dbl>
1 Premium        4584.
2 Fair           4359.
3 Very Good      3982.
4 Good           3929.
5 Ideal          3458.
```

Some plotting 1

A good data analysis usually starts with some visualisation

```
diamonds %>%
  ggplot(aes(x = carat, y = price)) +
  geom_point(alpha = .5) +
  facet_grid(cut ~ clarity) +
  geom_smooth(method = "lm", se = F) +
  theme_classic()
```

Some plotting 2