Data Literate with R

Nicolas Meseth

Table of contents

Preface

Download materials

You can download the ZIP-archive with all material here. This archive includes:

Folder	Content
`book`	The compiled book in PDF format
`data`	All data from the chapters
`docs`	All chapters as single PDF files
`exercises`	All exercises as PDF files (sometimes with solutions)
`scripts`	All code from the chapters as plain R-Scripts (`.R`)
`slides`	A collection of slide decks in PDF format

Part I

Data Loading

This part deals with loading data from various sources.

1 From CSV files

Loading data from a CSV file is simple with the `{readr}` package:

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

#>       order_id name  order~1 app_id created_at          updated_at          test  curre
#>          <dbl> <chr>   <dbl>  <dbl> <dttm>              <dttm>              <lgl>   <db
# > 1      1.13e12 B1014    1014 580111 2019-05-24 12:59:16 2019-06-19 13:23:26 FALSE    9
#> 2      1.13e12 B1015    1015 580111 2019-05-24 13:09:08 2019-06-21 14:40:07 FALSE    32
#> 3      1.13e12 B1016    1016 580111 2019-05-24 13:22:41 2019-06-21 12:35:23 FALSE    30
#> ...
```

2 From Excel files

Coming soon.

3 From RDS files

With the `readRDS()` function, we can load data from R's proprietary data format:

```
orders <- readRDS(file = "data/orders.rds")
```

If the original data was a tibble, as in this case, the loaded data will be, too:

```
orders
```

```
# A tibble: 2,874 x 68
     order_id name  order~1 app_id created_at          updated_at          test
        <dbl> <chr>   <dbl>  <dbl> <dttm>              <dttm>              <lgl>
 1    1.13e12 B1014    1014 580111 2019-05-24 12:59:16 2019-06-19 13:23:26 FALSE
 2    1.13e12 B1015    1015 580111 2019-05-24 13:09:08 2019-06-21 14:40:07 FALSE
 3    1.13e12 B1016    1016 580111 2019-05-24 13:22:41 2019-06-21 12:35:23 FALSE
 4    1.13e12 B1017    1017 580111 2019-05-24 13:27:43 2019-06-21 14:27:18 FALSE
 5    1.13e12 B1018    1018 580111 2019-05-24 13:36:46 2019-06-21 12:11:57 FALSE
 6    1.13e12 B1019    1019 580111 2019-05-24 13:44:41 2019-06-21 14:37:21 FALSE
 7    1.13e12 B1020    1020 580111 2019-05-24 13:49:21 2019-06-21 12:25:16 FALSE
 8    1.13e12 B1021    1021 580111 2019-05-24 13:59:57 2019-06-21 11:49:47 FALSE
 9    1.13e12 B1022    1022 580111 2019-05-24 14:43:53 2019-06-19 14:12:38 FALSE
10    1.13e12 B1023    1023 580111 2019-05-24 14:48:16 2019-06-21 15:54:24 FALSE
# ... with 2,864 more rows, 61 more variables: current_subtotal_price <dbl>,
#   current_total_price <dbl>, current_total_discounts <dbl>,
#   current_total_duties_set <dbl>, total_discounts <dbl>,
#   total_line_items_price <dbl>, total_outstanding <dbl>, total_price <dbl>,
#   total_tax <dbl>, total_tip_received <dbl>, taxes_included <lgl>,
#   discount_codes <chr>, financial_status <chr>, fulfillment_status <chr>,
#   source_name <chr>, landing_site <chr>, landing_site_ref <chr>, ...
```

3.1 Saving data to `.rds` format

We can save any data frame to an `.rds` file using the `saveRDS()` function:

```r
saveRDS(orders, file = "data/orders.rds")
```

3.2 Read more

Find more information in the R file format under the following links:

- Hands-On Programming with R - Appendix D.4 - R Files

4 From Google Spreadsheets

Coming soon.

5 From JSON files

Coming soon.

Part II

Data Transformation

This part introduces the basic tools for data transformation with R.

6 Operations

Data is the new oil, according to the mathematician Clive Humby:

> "Data is the new oil. Like oil, data is valuable, but if unrefined, it cannot really be used. It has to be changed into gas, plastic, chemicals, etc. to create a valuable entity that drives profitable activity. So, must data be broken down, analysed for it to have value."

If we take this analogy seriously, the data, like oil, needs to be refined to turn it into something of value. Two important tools for refining data into a valuable output are *data transformation* and *data visualization*, both of which are the main focus of this book. In this part of the book, we first need to learn how to transform data so that we can apply visualization later on.

To learn how to transform data, we need to learn how to to the following operations:

- Remove any variables we don't currently need (or specify those we **do** need)
- Remove any records we don't currently need (or specify those we **do** need)
- Add new variables that don't exist yet
- Summarize many records into one or a few numbers
- Change the order of the records

The goal of the following chapters is to introduce means to perform theses five operations with R.

7 Select columns

This chapter introduces tools to remove unnecessary columns from the data set. Or, positively stated, we learn how to specify the columns we need for our analysis. As with most data transformation operations, we mostly introduce functions from the `{dplyr}` package.

The function `select()` is the designated tool to select columns with `{dplyr}`. By passing different things to the function, we can efficiently define the set of columns in the resulting data frame.

7.1 By column names

The easiest and intuitive way to specify the columns we want is by listing their names. We can pass one or more column names to the `select()` function. In case of two or more, we use commas to separate the names:

```
# Just one column name
orders %>%
  select(order_id)

#> # A tibble: 2,874 x 1
#>        order_id
#>           <dbl>
#> 1 1130007101519
#> 2 1130014965839
#> 3 1130026958927
#> ...

# A list of column names
orders %>%
  select(order_id, total_price)

#> # A tibble: 2,874 x 2
#>        order_id total_price
#>           <dbl>       <dbl>
#> 1 1130007101519        94.7
```

```
#> 2 1130014965839        32.2
#> 3 1130026958927        30.2
#> ...
```

When we only want a few columns, this approach works fine and is usually a good choice. I expect you apply this method in more than 90% of all cases. However, there are cases when you'd wish there was something more flexible. Luckily, there is.

7.2 By name patterns

7.2.1 Names starting with a string

Sometimes we want to select columns based on a pattern of their names. Take the orders data set as an example. Here, all variables that contain information about the shipping address have the prefix `shipping`. We leverage this with the helper function `starts_with()`:

```
orders %>%
  select(starts_with("shipping")) %>%
  colnames()

#> [1] "shipping_address_city"      "shipping_address_zip"       "shipping_address_country
#> [4] "shipping_address_latitude"  "shipping_address_longitude"
```

7.2.2 Names ending with a string

7.2.3 Names with a string anywhere

7.2.4 Using regular expressions

7.3 By data type

```
orders %>%
  select(where(is.numeric))

orders %>%
  select(where(is.logical))

orders %>%
```

```r
  select(where(is.character))

orders %>%
  select(where(is.factor))

orders %>%
  select(where(is.list))

# The package lubridate provides a function to check for date (without time)
orders %>%
  select(where(lubridate::is.Date))

# Select all date/time columns
orders %>%
  select(where(lubridate::is.POSIXct))
```

8 Filter rows

9 Add columns

10 Summarize rows

11 Sort rows

Part III

Data Visualization

This part introduces the basic tools for data visualization with R.

12 Overview

13 Pleas for data visualization

The R code for the following sections is also available as plain `.R` scripts. If you downloaded the ZIP-file and you view this as a PDF-document, you find the `.R` files in the same folder as this document.

To illustrate why data visualization is useful, let's look at two examples. Below we read some data from a CSV-file.

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

#> # A tibble: 142 x 2
#>        x     y
#>    <dbl> <dbl>
#> 1   55.4  97.2
#> 2   51.5  96.0
#> 3   46.2  94.5
#> ...
```

As you can see, the data contains two variables `x` and `y` with 142.

If we didn't have visualization as a tool in our data analytics toolkit, we could try to get some insight into the data with descriptive statistics. For example, we could calculate the mean for both variables:

```
some_data %>%
  summarise(across(everything(), mean, .names = "{.col}_mean"))
```

```
# A tibble: 1 x 2
  x_mean y_mean
   <dbl>  <dbl>
1   54.3   47.8
```

Similarly, we could calculate a measure of spread, such as the standard deviation:

```
some_data %>%
  summarise(across(everything(), sd, .names = "{.col}_sd"))
```

```
# A tibble: 1 x 2
   x_sd  y_sd
  <dbl> <dbl>
1  16.8  26.9
```

Or other measures:

```
some_data %>%
  summarise(
    across(everything(),
           list(mean = mean, sd = sd, median = median),
           .names = "{.col}_{.fn}"
           )
    )
```

```
# A tibble: 1 x 6
  x_mean  x_sd x_median y_mean  y_sd y_median
   <dbl> <dbl>    <dbl>  <dbl> <dbl>    <dbl>
1   54.3  16.8     53.3   47.8  26.9     46.0
```

We could also calculate Pearson's correlation coefficient:

```
tibble(
  pearson = cor(some_data$x, some_data$y)
  )
```

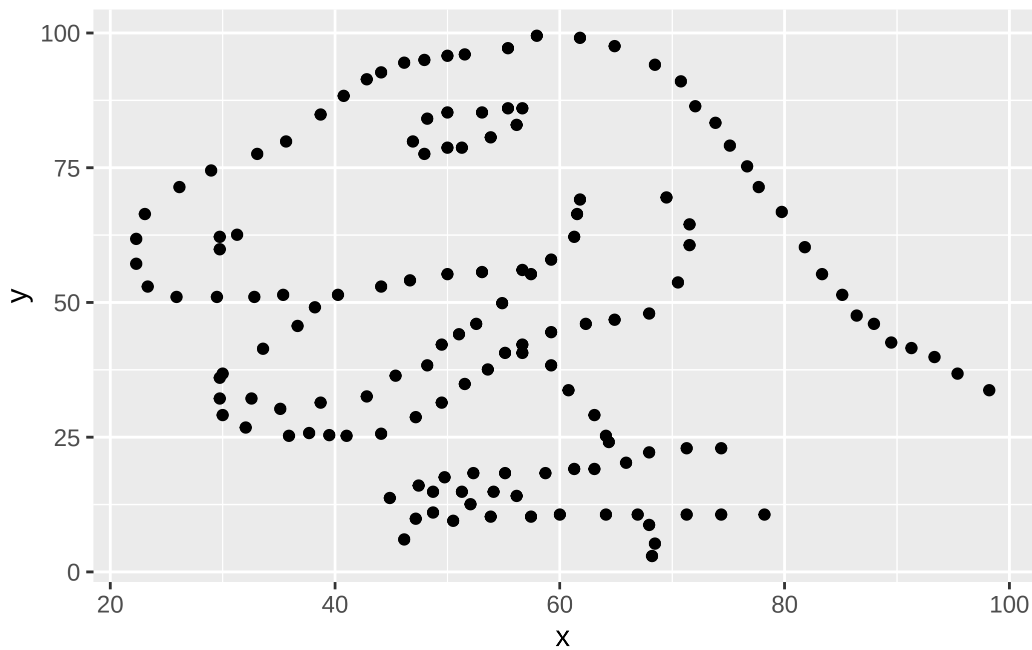
```
# A tibble: 1 x 1
  pearson
    <dbl>
1 -0.0645
```

From the rather small value, we could hypothesize that the variables are unrelated. But are they?

13.1 Visualization can reveal hidden patterns

Let's add visualization to our toolkit and find out:

```
some_data %>%
  ggplot() +
  aes(x, y) +
  geom_point()
```



The data certainly does not look unrelated to me. Of course, this an exaggerated example, but it makes the point: Only when we visualize data can we identify patterns that would otherwise stay hidden in the numbers. No statistical method could have told us there is dinosaur in the data. Well, actually it is called a datasaurus, and there is even a whole R-package with the name `{datasauRus}` dedicated to it. This packages contains the same data set, but adds more that share the same statistical measures. We could not distinguish between the data by just looking at measures such as mean, standard deviation or correlation coefficient. We would have to visualize the data:

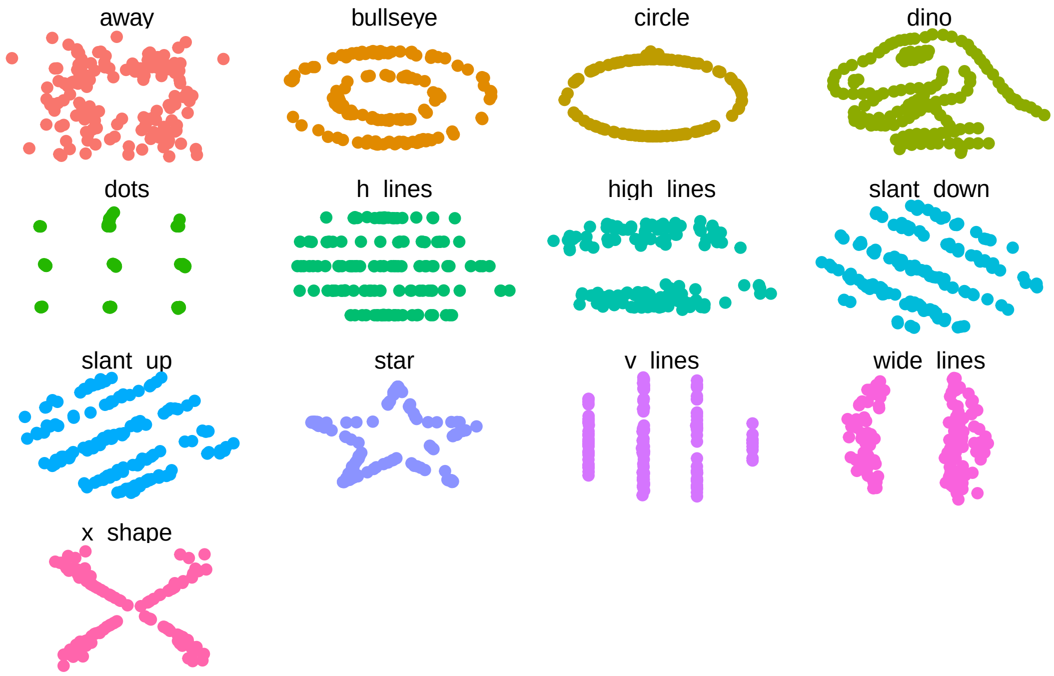
```
#install.packages("datasauRus")
library(datasauRus)

datasaurus_dozen %>%
  group_by(dataset) %>%
  summarize(
    mean_x     = mean(x),
```

```r
    mean_y     = mean(y),
    std_dev_x = sd(x),
    std_dev_y = sd(y),
    corr_x_y   = cor(x, y)
    )
```

```
# A tibble: 13 x 6
   dataset    mean_x mean_y std_dev_x std_dev_y corr_x_y
   <chr>       <dbl>  <dbl>     <dbl>     <dbl>    <dbl>
 1 away         54.3   47.8      16.8      26.9  -0.0641
 2 bullseye     54.3   47.8      16.8      26.9  -0.0686
 3 circle       54.3   47.8      16.8      26.9  -0.0683
 4 dino         54.3   47.8      16.8      26.9  -0.0645
 5 dots         54.3   47.8      16.8      26.9  -0.0603
 6 h_lines      54.3   47.8      16.8      26.9  -0.0617
 7 high_lines   54.3   47.8      16.8      26.9  -0.0685
 8 slant_down   54.3   47.8      16.8      26.9  -0.0690
 9 slant_up     54.3   47.8      16.8      26.9  -0.0686
10 star         54.3   47.8      16.8      26.9  -0.0630
11 v_lines      54.3   47.8      16.8      26.9  -0.0694
12 wide_lines   54.3   47.8      16.8      26.9  -0.0666
13 x_shape      54.3   47.8      16.8      26.9  -0.0656
```

The table shows the mean, standard deviation and correlation coefficient for all 13 data sets included in the `{datasauRus}` package. As you can see, the values are the same across all data sets. Only when we visualize do we see the different patterns in the data:

```r
datasaurus_dozen %>%
  ggplot() +
  aes(x = x, y = y, colour = dataset) +
  geom_point() +
  theme_void() +
  theme(legend.position = "none") +
  facet_wrap(~dataset, ncol = 4)
```



13.2 Anscombe's Quartet

Another and even older plea for the visualization of data can be found in Francis Anscombe's publication *Graphs in Statistical Analysis* from the year 1973. In this paper, Anscombe presented four data sets that looked very much the same when viewing the common descriptive statistical measures. Again, only by visualizing the data can we see the real patterns.

Let's load the data and see for ourselves:

```
anscombe1 <- read_csv("data/anscombe1.csv") %>%
  mutate(dataset = "1")

anscombe2 <- read_csv("data/anscombe2.csv") %>%
  mutate(dataset = "2")

anscombe3 <- read_csv("data/anscombe3.csv") %>%
  mutate(dataset = "3")

anscombe4 <- read_csv("data/anscombe4.csv") %>%
  mutate(dataset = "4")
```

We now want all four in one data frame. We can achieve this with the `union_all()` function:

```r
anscombe <-
  anscombe1 %>%
  union_all(anscombe2) %>%
  union_all(anscombe3) %>%
  union_all(anscombe4)

#> # A tibble: 44 x 3
#>       x     y dataset
#>   <dbl> <dbl> <chr>
#> 1    10  8.04 1
#> 2     8  6.95 1
#> 3    13  7.58 1
#> ...
```

We now have all four of Anscombe's Quartet in one data frame and we can distinguish the original data set by the column `dataset`. First, let's look at the descriptive statistics:

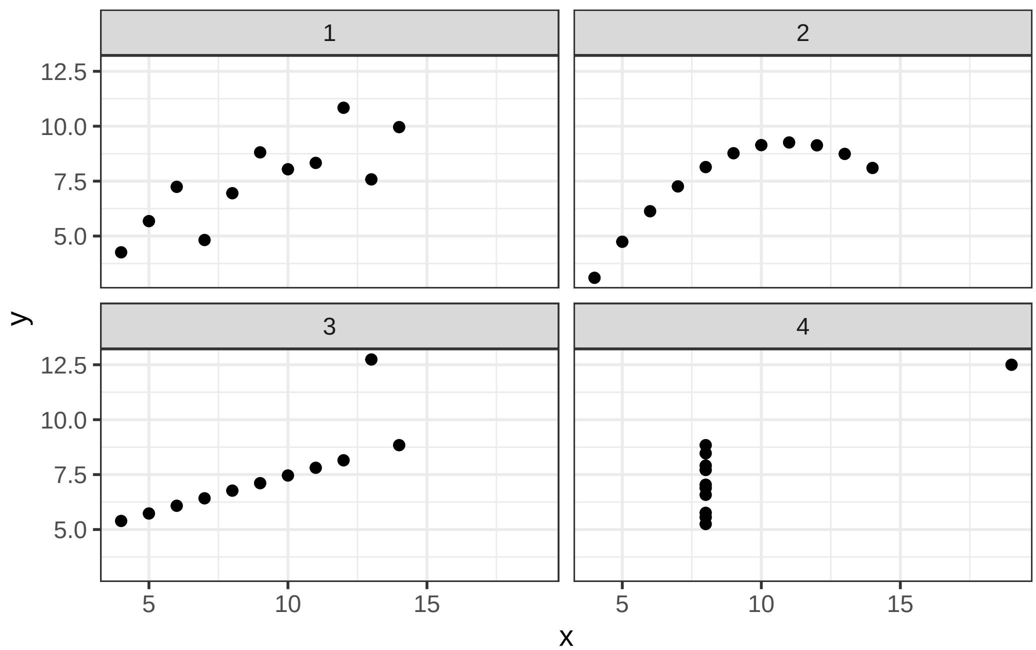
```r
anscombe %>%
  group_by(dataset) %>%
  summarize(
    mean_x    = mean(x),
    mean_y    = mean(y),
    std_dev_x = sd(x),
    std_dev_y = sd(y),
    corr_x_y  = cor(x, y)
    )
```

```
# A tibble: 4 x 6
  dataset mean_x mean_y std_dev_x std_dev_y corr_x_y
  <chr>    <dbl>  <dbl>     <dbl>     <dbl>    <dbl>
1 1            9   7.50      3.32      2.03    0.816
2 2            9   7.50      3.32      2.03    0.816
3 3            9   7.5       3.32      2.03    0.816
4 4            9   7.50      3.32      2.03    0.817
```

As expected, all measures look the same for all 4 data sets. But again, a plot reveals the truth:

```r
anscombe %>%
  ggplot() +
  aes(x, y) +
```

```
  geom_point() +
  theme_bw() +
  theme(legend.position = "none") +
  facet_wrap(~dataset, ncol = 2)
```



The first plot shows a linear trend with some noise, as we might already have suspected from a correlation coefficient of roughly 0.81. The second plot, although having the same correlation coefficient, displays a obviously non-linear trajectory. The third plot would have had a perfect correlation if it wasn't for the single outlier. In contrast, the last plot would have had no correlation between `x` and `y` if the point on the very top-right didn't exist. Again, we could not have gotten this insight from any statistical measure we can calculate.

I hope the examples convinced you of the importance of data visualization in data analytics. There are even more good reasons why we should visualize data, besides the fact that otherwise couldn't reveal hidden patterns. We know from psychological research about the way humans process information that the visualizations are a much faster way into our brains. We can not only grasp what we see in a good data visualization faster, but also comprehend it better and create a better memory of it. If that doesn't convince you, nothing will.

13.3 References

- The official website of the `{datasauRus}` package
- YouTube video on Anscombe's Quartet
- Original Paper *Graphs in Statistical Analysis* by Francis Anscombe