THE BIG-R BOOK

FROM DATA SCIENCE TO LEARNING MACHINES AND BIG DATA

— PART 04—

Dr. Philippe J.S. De Brouwer

last compiled: September 1, 2021
Version 0.1.1

THE BIG R-BOOK: From Data Science to Big Data and Learning Machines

♡— PART 04: Data Wrangling —♡

These slides are to be used in with the book – for best experience, teachers will read the book *before* using the slides and students have access to the book and the code.

part 04: Data Wrangling

↓

chapter 16:

Anonymous Data

Encryption in SQL

```sql
-- Using AES 256 for example:
MariaDB [(none)]> SELECT AES_ENCRYPT("Hello World", "secret_key_string");
+-------------------------------------------------+
| AES_ENCRYPT("Hello World", "secret_key_string") |
+-------------------------------------------------+
| ï£¡Ewï£¡*0ï£¡ï£¡Wï£¡ï£¡ï£¡5%ï£¡                                 |
+-------------------------------------------------+
1 row in set (0.00 sec)

-- Example:
SELECT AES_ENCRYPT(name, "secret_key_string"), AES_ENCRYPT(phone number, "secret_key_string"),
     number_purchases, satifaction_rating, sustomer_since, etc.
       FROM tbl_customers;

-- To decrypt, use AES_DECRYPT(crypt_str, key_string)
```

Listing 1: *SQL code for MySQL (or MariaDB) to encrypt using AES256. Note that those relational database systems (RDBMSs) provide much more methods for encryption. It is worth to go through the documentation of your particular system for more support.*

Anonymising data in R i

There is also the package `sodium`, created by Jeroen Ooms. It is a wrapper around `libsodium`, which is a standard library. So you will need to install this first on your operating system (OS).

- deb: libsodium-dev (Debian, Ubuntu, etc.)
- rpm: libsodium-devel (Fedora, EPEL)
- csw: libsodium_dev (Solaris)
- brew: libsodium (OSX)

This means – most probably – your will first need to open a terminal and run the following commands in the CLI (command line interface of your OS):

```
sudo apt-get install libsodium-dev
```

Then we can open R and install the `sodium` library for R.

Anonymising data in R ii

```r
# install.packages('sodium') # do only once
                           # fails if you do not have libsodium-dev
library(sodium)

# Create the SHA256 key based on a secret password:
key <- sha256(charToRaw("My sweet secret"))

# Serialize the data to be encrypted:
msg <- serialize("Philippe J.S. De Brouwer", NULL)

# Encrypt:
msg_encr <- data_encrypt(msg, key)


orig <- data_decrypt(msg_encr, key)
stopifnot(identical(msg, orig))

# Tag the message with your key (HMAC):
tag <- data_tag(msg, key)
```

part 04: Data Wrangling

↓

chapter 17:

Data Wrangling in the tidyverse

part 04: Data Wrangling

↓

chapter 17: Data Wrangling in the tidyverse

↓

section 1:

Importing the Data

Reminder of the functions to connect to the database i

```r
# --
library(RMySQL)

# -- The functions as mentioned earlier:
# db_get_data
# Get data from a MySQL database
# Arguments:
#    con_info -- MySQLConnection object -- containing the connection
#                                          info to the MySQL database
#    sSQL     -- character string       -- the SQL statement that selects
#                                          the records
# Returns
#    data.frame, containing the selected records
db_get_data <- function(con_info, sSQL){
  con <- dbConnect(MySQL(),
                   user     = con_info$user,
                   password = con_info$password,
                   dbname   = con_info$dbname,
                   host     = con_info$host
                   )
  df <- dbGetQuery(con, sSQL)
  dbDisconnect(con)
  df
}
```

Reminder of the functions to connect to the database ii

```r
# db_run_sql
# Run a query that returns no data in an MySQL database
# Arguments:
#    con_info -- MySQLConnection object -- containing the connection
#                                          info to the MySQL database
#    sSQL     -- character string       -- the SQL statement to be run
db_run_sql <-function(con_info, sSQL)
{
  con <- dbConnect(MySQL(),
               user     = con_info$user,
               password = con_info$password,
               dbname   = con_info$dbname,
               host     = con_info$host
               )
  rs <- dbSendQuery(con,sSQL)
  dbDisconnect(con)
}
```

Connecting to the library database

```
# Load dplyr via tidyverse:
library(tidyverse)

# Define the wrapper functions:

# Step 1: define the connection info.
my_con_info <- list()
my_con_info$user     <- "librarian"
my_con_info$password <- "librarianPWD"
my_con_info$dbname   <- "library"
my_con_info$host     <- "localhost"


# -- The data import was similar to what we had done previously.
# -- However, now we import all tables separately
# Step 2: get the data
my_tables <- c("tbl_authors", "tbl_author_book",
               "tbl_books", "tbl_genres")
my_db_names <- c("authors", "author_book",
               "books", "genres")

# Loop over the four tables and download their data:
for (n in 1:length(my_tables)) {
  my_sql <- paste("SELECT * FROM `",my_tables[n],"`;", sep="")
  df <- db_get_data(my_con_info, my_sql)
  # the next line uses tibbles are from the tidyverse
  as_tibble(assign(my_db_names[n],df))
 }
```

Importing CSV files via the tidyverse

```
library(tidyverse)
s_csv = "'a','b','c'\n001,2.34,.\n2,3.14,55\n3,.,43"
read_csv(s_csv)
## # A tibble: 3 x 3
##   `'a'` `'b'` `'c'`
##   <chr> <chr> <chr>
## 1 001   2.34  .
## 2 2     3.14  55
## 3 3     .     43

read_csv(s_csv, na = '.')  # Tell R how to understand the '.'
## # A tibble: 3 x 3
##   `'a'` `'b'` `'c'`
##   <chr> <dbl> <dbl>
## 1 001    2.34    NA
## 2 2      3.14    55
## 3 3     NA       43

read_csv(s_csv, na = '.',  quote = "'") # Tell how a string is quoted
## # A tibble: 3 x 3
##   a         b     c
##   <chr> <dbl> <dbl>
## 1 001    2.34    NA
## 2 2      3.14    55
## 3 3     NA       43
```

Importing fixed width files i

```
# Make a string that looks like a fixed-width table (shortened):
txt <- "book_id  year  title                                              genre
       1  1896  Les plaisirs et les jour                       LITmod
       2  1927  Albertine disparue                             LITmod
       3  1954  Contre Sainte-Beuve                            LITmod
       8  1687  PhilosophiÃę Naturalis Principia Mathematica      SCIphy
       9  -300  Elements (translated )                         SCImat
      10  2014  Big Data World                                 SCIdat
      11  2016  Key Business Analytics                         SCIdat
      12  2011  Maslowian Portfolio Theory                     FINinv
      13  2016  R for Data Science                             SCIdat"
```

Starting from this string variable, we will create a text file that has data in the fixed-width format.

```
fileConn <- file("books.txt")
writeLines(txt, fileConn)
close(fileConn)

my_headers <- c("book_id","year","title","genre")
```

The previous code chunk has created the text file `book.txt` in the working path of R. Now, we can read it back in to illustrate how the `read_fwf()` function works.

Importing fixed width files ii

```
# Reading the fixed-width file
# -- > by indicating the widths of the columns
t <- read_fwf(
  file = "./books.txt",
  skip = 1,                # skip one line with headers
  fwf_widths(c(8, 6, 48, 8), my_headers)
  )

# Inspect the input:
print(t)
## # A tibble: 9 x 4
##   book_id  year title                                        genre
##     <dbl> <dbl> <chr>                                        <chr>
## 1       1  1896 Les plaisirs et les jour                     LITmod
## 2       2  1927 Albertine disparue                           LITmod
## 3       3  1954 Contre Sainte-Beuve                          LITmod
## 4       8  1687 PhilosophiÃę Naturalis Principia Mathematica SCIp
## 5       9  -300 Elements (translated )                       SCImat
## 6      10  2014 Big Data World                               SCIdat
## 7      11  2016 Key Business Analytics                       SCIdat
## 8      12  2011 Maslowian Portfolio Theory                   FINinv
## 9      13  2016 R for Data Science                           SCIdat
```

Importing fixed width files iii

```r
# -- > same but naming directly
t <- read_fwf(
  file="./books.txt",
  skip=1,               # skip one line with headers
  fwf_cols(book_id = 8, year = 6,
           title = 48, genre = 8)
  )
```

```r
# -- > by selecting columns (by indicating begin and end):
t2 <- read_fwf(
   file = "books.txt",
   skip = 1,
   fwf_cols(year = c(11, 15),
            title = c(17, 63))
   )
```

```r
# -- > by guessing the columns
# The function fwf_empty can help to guess where the columns start
# based on white space
t3 <- read_fwf(
  file = "books.txt",
  skip = 1,
  fwf_empty("books.txt")
  )
```

Importing fixed width files iv

Note that this last method fails: it identifies a separate column for the word “Mathematica”, while this is actually part of the column “title”:

```
print(t3)
## # A tibble: 9 x 5
##      X1    X2 X3                              X4          X5
##   <dbl> <dbl> <chr>                           <chr>       <chr>
## 1     1  1896 Les plaisirs et les jour        <NA>        LITmod
## 2     2  1927 Albertine disparue              <NA>        LITmod
## 3     3  1954 Contre Sainte-Beuve             <NA>        LITmod
## 4     8  1687 PhilosophiÃę Naturalis Principia Mathematica SCIphy
## 5     9  -300 Elements (translated )          <NA>        SCImat
## 6    10  2014 Big Data World                  <NA>        SCIdat
## 7    11  2016 Key Business Analytics          <NA>        SCIdat
## 8    12  2011 Maslowian Portfolio Theory      <NA>        FINinv
## 9    13  2016 R for Data Science              <NA>        SCIdat
```

PART 04: DATA WRANGLING

↓

CHAPTER 17: DATA WRANGLING IN THE TIDYVERSE

↓

SECTION 2:

Tidy Data

Tidy Data has

1. a tibble/data-frame for each dataset,
2. a column for each variable,
3. a row for each observation,
4. a value (or NA) in each cell – the intersection between row and column.

PART 04: DATA WRANGLING

↓

CHAPTER 17: DATA WRANGLING IN THE TIDYVERSE

↓

SECTION 3:

Tidying Up Data with tidyr

Importing one table that contains the whole database

```
# use the wrapper functions to get data.

# step 1: define the connection info
my_con_info <- list()
my_con_info$user     <- "librarian"
my_con_info$password <- "librarianPWD"
my_con_info$dbname   <- "library"
my_con_info$host     <- "localhost"


## -- Import 2 tables combined
# step 2: get the data
my_sql <- "SELECT * FROM tbl_authors
   JOIN tbl_author_book ON author_id = author
   JOIN tbl_books       ON book      = book_id
   JOIN tbl_genres      ON genre     = genre_id;"
t_mix <- db_get_data(my_con_info, my_sql)
t_mix <- as.tibble(t_mix)

# Show the result:
head(t_mix)
## # A tibble: 6 x 16
##   author_id pen_name   full_name    birth_date death_date ab_id author
##       <dbl> <chr>      <chr>        <chr>      <chr>      <dbl>  <dbl>
## 1         1 Marcel Pr~ Valentin Lo~ 1871-07-10 1922-11-18     1      1
## 2         1 Marcel Pr~ Valentin Lo~ 1871-07-10 1922-11-18     2      1
## 3         1 Marcel Pr~ Valentin Lo~ 1871-07-10 1922-11-18     3      1
## 4         1 Marcel Pr~ Valentin Lo~ 1871-07-10 1922-11-18     4      1
## 5         2 Miguel de  Miguel de C~ 1547-09-29 1616-04-22     5      2
```

Workflow to split tables in R i

1. Understand the data structure, eventually talk to the data owners and understand what is the job at hand. In this case, it is a mix of four tables: authors, a link-table to books, books, and genres.

```
# Make a table of how much each author_id occurs:
nbr_auth <- t_mix  %>% count(author_id)

# Do the same and include all fields that are assumed to
# be part of the table authors.
nbr_auth2 <- t_mix    %>%
  count(author_id, pen_name, full_name, birth_date, death_date, book)

nbr_auth$n - nbr_auth2$n
##  [1] 3 1 0 0 0 0 0 0 0 0 3 1 0 0
```

2. Learn from experiments till we find the right structure. In our case “book” is not unique for an “author,” so we try again.

```
# Try without book:
nbr_auth2 <- t_mix    %>%
  count(author_id, pen_name, full_name, birth_date, death_date)

# Now these occurrences are the same:
nbr_auth$n - nbr_auth2$n
##  [1] 0 0 0 0 0 0 0 0 0 0
```

Workflow to split tables in R ii

3. This looks better. But note that this exact match is only possible because our data is clean (because we took care and/or because we asked MySQL to help us to guard referential integrity). We still have to determine now which table takes which fields.
4. Now, the heavy lifting is done and we can simply extract all data.

Workflow to split tables in R iii

```
my_authors <- tibble(author_id = t_mix$author_id,
                     pen_name   = t_mix$pen_name,
                     full_name  = t_mix$full_name,
                     birth_date = t_mix$birth_date,
                     death_date = t_mix$death_date
                     )      %>%
              unique        %>%
              print
## # A tibble: 10 x 5
##    author_id pen_name        full_name            birth_date death_date
##        <dbl> <chr>           <chr>                <chr>      <chr>
##  1         1 Marcel Proust   Valentin Louis G. ~ 1871-07-10 1922-11-18
##  2         2 Miguel de Cerv~ Miguel de Cervante~ 1547-09-29 1616-04-22
##  3         4 E. L. James     Erika Leonard       1963-03-07 <NA>
##  4         5 Isaac Newton    Isaac Newton        1642-12-25 1726-03-20
##  5         7 Euclid          Euclid of Alexandr~ <NA>       <NA>
##  6        11 Bernard Marr    Bernard Marr        <NA>       <NA>
##  7        13 Bart Baesens    Bart Baesens        1975-02-27 <NA>
##  8        14 Philippe J.S. ~ Philippe J.S. De B~ 1969-02-21 <NA>
##  9        15 Hadley Wickham  Hadley Wickham      <NA>       <NA>
## 10        16 Garrett Grolem~ Garrett Grolemund   <NA>       <NA>
```

5. Repeat this process for all other tables.

Workflow to split tables in R iv

6. Check the data and see once more if it all makes sense. In our case we will want to correct some of the data that has been imported and coerce them to the right type.

```
auth <- tibble(
            author_id = as.integer(my_authors$author_id),
            pen_name   = my_authors$pen_name,
            full_name  = my_authors$full_name,
            birth_date = as.Date(my_authors$birth_date),
            death_date = as.Date(my_authors$death_date)
               )       %>%
        unique         %>%
        print
## # A tibble: 10 x 5
##    author_id pen_name        full_name           birth_date death_date
##        <int> <chr>           <chr>               <date>     <date>
##  1         1 Marcel Proust   Valentin Louis G. ~ 1871-07-10 1922-11-18
##  2         2 Miguel de Cerv~ Miguel de Cervante~ 1547-09-29 1616-04-22
##  3         4 E. L. James     Erika Leonard       1963-03-07 NA
##  4         5 Isaac Newton    Isaac Newton        1642-12-25 1726-03-20
##  5         7 Euclid          Euclid of Alexandr~ NA         NA
##  6        11 Bernard Marr    Bernard Marr        NA         NA
##  7        13 Bart Baesens    Bart Baesens        1975-02-27 NA
##  8        14 Philippe J.S. ~ Philippe J.S. De B~ 1969-02-21 NA
##  9        15 Hadley Wickham  Hadley Wickham      NA         NA
## 10        16 Garrett Grolem~ Garrett Grolemund   NA         NA
```

Convert headers to data: the example

```
# First read in some data (using a flat file to remind
# how this works):
 x <- " January   100       102       108
 February  106       105       105
 March     104       104       106
 April     120       122       118
 May       130       100       133
 June      141       139       135
 July      175       176       180
 August    170       188       187
 September 142       148       155
 October   133       137       145
 November  122       128       131
 December  102       108       110"

# Read in the flat file via read_fwf from readr:
t <- read_fwf(x,  fwf_empty(x, col_names = my_headers))

# Set the column names:
colnames(t) <-  c("month", "Sales2017", "Sales2018", "Sales2019")

# Finally, we can show the data as it appeared in the spreadsheet
# from the sales department:
print(t)
## # A tibble: 12 x 4
##    month     Sales2017 Sales2018 Sales2019
##    <chr>         <dbl>     <dbl>     <dbl>
##  1 January         100       102       108
##  2 February        106       105       105
```

Converting headers to data: the workflow

```
t2 <- gather(t, "year", "sales", 2:4)
t2$year <- str_sub(t2$year,6,9)  # delete the sales word
t2$year <- as.integer(t2$year)   # convert to integer

# Show the result:
t2
## # A tibble: 36 x 3
##    month      year sales
##    <chr>     <int> <dbl>
##  1 January    2017   100
##  2 February   2017   106
##  3 March      2017   104
##  4 April      2017   120
##  5 May        2017   130
##  6 June       2017   141
##  7 July       2017   175
##  8 August     2017   170
##  9 September  2017   142
## 10 October    2017   133
## # ... with 26 more rows
```

Spreading one column over many

```
library(dplyr)
sales_info <- data.frame(
       time = as.Date('2016-01-01') + 0:9 + rep(c(0,-1), times=5),
       type  = rep(c("bought","sold"),5),
       value = round(runif(10, min = 0, max = 10001))
       )

# Show the data frame:
sales_info
##          time   type value
## 1  2016-01-01 bought  9949
## 2  2016-01-01   sold  3717
## 3  2016-01-03 bought  1936
## 4  2016-01-03   sold  1319
## 5  2016-01-05 bought  2131
## 6  2016-01-05   sold  9032
## 7  2016-01-07 bought  5954
## 8  2016-01-07   sold  9344
## 9  2016-01-09 bought  3999
## 10 2016-01-09   sold  6871

# Use the function spread():
spread(sales_info, type, value)
##         time bought sold
## 1 2016-01-01   9949 3717
## 2 2016-01-03   1936 1319
## 3 2016-01-05   2131 9032
## 4 2016-01-07   5954 9344
```

Split one column into many

```r
library(tidyr)
# The original data frame:
turnover <- data.frame(
       what = paste(as.Date('2016-01-01') + 0:9 + rep(c(0,-1), times=5),
                    rep(c("HSBC","JPM"),5), sep="/"),
       value = round(runif(10, min = 0, max = 50))
       )
turnover
##                what value
## 1  2016-01-01/HSBC    29
## 2   2016-01-01/JPM     5
## 3  2016-01-03/HSBC    32
## 4   2016-01-03/JPM    13
## 5  2016-01-05/HSBC    26
## 6   2016-01-05/JPM     5
## 7  2016-01-07/HSBC    11
## 8   2016-01-07/JPM    36
## 9  2016-01-09/HSBC    41
## 10  2016-01-09/JPM    14

# Use the function separate():
separate(turnover, what, into=c("date","counterpart"), sep="/")
##          date counterpart value
## 1  2016-01-01        HSBC    29
## 2  2016-01-01         JPM     5
## 3  2016-01-03        HSBC    32
## 4  2016-01-03         JPM    13
## 5  2016-01-05        HSBC    26
```

Merging multiple columns into one

```
library(tidyr)

# Define a data frame:
df <- data.frame(year = 2018, month = 0 + 1:12, day = 5)
print(df)
##    year month day
## 1  2018     1   5
## 2  2018     2   5
## 3  2018     3   5
## 4  2018     4   5
## 5  2018     5   5
## 6  2018     6   5
## 7  2018     7   5
## 8  2018     8   5
## 9  2018     9   5
## 10 2018    10   5
## 11 2018    11   5
## 12 2018    12   5

# Merge the columns to one variable:
unite(df, 'date', 'year', 'month', 'day', sep = '-')
##          date
## 1   2018-1-5
## 2   2018-2-5
## 3   2018-3-5
## 4   2018-4-5
## 5   2018-5-5
## 6   2018-6-5
```

PART 04: DATA WRANGLING

↓

CHAPTER 17: DATA WRANGLING IN THE TIDYVERSE

↓

SECTION 4:

SQL-like Functionality via dplyr

Loading `dplyr`

This functionality is provided by the library `dplyr` of the `tidyverse`. So, we will load it here and not repeat this in every sub-section.

```
library(dplyr)
```

Selecting columns

```
# Using the example of the library:
dplyr::select(genres,      # the first argument is the tibble
       genre_id, location) # then a list of column names
##   genre_id location
## 1   FINinv   405.08
## 2   LITero   001.67
## 3   LITmod   001.45
## 4   SCIbio   300.10
## 5   SCIdat   205.13
## 6   SCImat   100.53
## 7   SCIphy   200.43
```

Selecting certain rows only (filtering)

```
a1 <- filter(authors, birth_date > as.Date("1900-01-01"))
paste(a1$pen_name,"--",a1$birth_date)
## [1] "E. L. James -- 1963-03-07"
## [2] "Bart Baesens -- 1975-02-27"
## [3] "Philippe J.S. De Brouwer -- 1969-02-21"
```

dplyr and SQL

Hint – Equivalence between dplyr and SQL

Note that

```
filter(count(author_book, author), n  > 1}
```

is equivalent with the following in SQL

```
SELECT COUNT(author) FROM tbl_author_book
   HAVING COUNT(author) > 1;
```

Joining tables i

In the tidyverse , `dplyr` provides a series of join-functions, that all share a similar synthax:

```
*_join(x, y, by = NULL, copy = FALSE, ...)
```

We distinguish the following join functions.

1. **mutating joins**: Contrary to what the name suggests, they do not mutate the tibbles on which they opearate. These joins output fields of both data frames. `dplyr` provides the following “mutating joins”.
   - *inner_join()* returns all the columns for `x` and `y`, but only those rows that have matching values in their respective field/columns mentioned in the `by`-clause, and all columns from “x” and “y.” Note that it is possible that some of the join-fields are not unique and hence there can be multiple matches for the same record, then all combinations are all returned.
   - *left_join()* returns all the columns for `x` and `y`, so that all rows of `x` will be returned at least once (with a match of `y` if it exists, otherwise with a match to `NA` (or `NULL` in SQL vocabulary) (matches are defined by the `by`-clause). Note that it is possible that some of the join-fields are not unique and hence there can be multiple matches for the same record or `x`, then all combinations are all returned.
   - *right_join()* is similar to the previous but roles of `x` and `y` are inverted. Hence, it returns all rows from `y`, and all columns from `x` and `y`. Rows in `y` with no match in `x` will still be returned but have `NA` values in those rows of the `y` data frame.
   - *full_join()* returns all rows and all columns from both data frames `x` and `y`. Where there are not matching values, returns “NA” for the one missing.
2. **filtering joins** that only output the fields (columns) of the left data frame.

Joining tables ii

- *semi_join()* returns all rows from x but only if there is a matching values in the field of y, while only keeping the columns of x. Note that unlike an inner join, the semi join will never duplicate rows of x
- *anti_join()* returns all rows from x that do not have a matching value in y, while keeping only the columns of x.

sqldf: leverage you SQL knowledge

```r
library(sqldf)
# Because we have RMySQL loaded (and we don't want to unload it) sqldf will
# default to using that engine to run the queries. If we want it to use the
# R environment and data frames, then use the following line:
options(sqldf.driver = "SQLite")
```

```r
# Now you can use SQL syntax on R-data-frames. Imagine that we need to find the
# titles of books of the authors with name ending in 'Brouwer':
sqldf("SELECT B.title FROM authors AS A, author_book as AB, books AS B
              WHERE A.author_id = AB.author AND AB.book = B.book_id
                    AND full_name LIKE '%Brouwer';")
##                        title
## 1 Maslowian Portfolio Theory
```

Mutating data

```
ab <- authors                                              %>%
  inner_join(author_book, by = c("author_id" = "author"))  %>%
  inner_join(books,       by = c("book"      = "book_id")) %>%
  add_count(author_id)
ab$n
##  [1] 4 4 4 4 2 2 1 1 1 1 1 1 1 1
```

Mutating data

```
t <- authors                                                   %>%
     mutate(short_name = str_sub(pen_name,1,7))                %>%
     mutate(x_name = if_else(str_length(pen_name) > 15,
                            paste(str_sub(pen_name,1,8),
                                  "...",
                                  str_sub(pen_name,
                                         start = -3),
                                  sep=''),
                            pen_name,
                            "pen_name is NA"
                           )
           )                                                   %>%
     mutate(is_alive =
       if_else(!is.na(birth_date) & is.na(death_date),
            "YES",
            if_else(death_date < Sys.Date(),
                "no",
                "maybe"),
             "NA")
            )                                                  %>%
    dplyr::select(c(x_name, birth_date, death_date, is_alive)) %>%
    print()
##             x_name birth_date death_date is_alive
## 1   Marcel Proust 1871-07-10 1922-11-18       no
## 2  Miguel d...tes 1547-09-29 1616-04-22       no
## 3     James Joyce 1882-02-02 1941-01-13       no
## 4     E. L. James 1963-03-07       <NA>      YES
## 5    Isaac Newton 1642-12-25 1726-03-20       no
## 6          Euclid       <NA>       <NA>     <NA>
```

Set operations

These functions are:

- `intersect(x, y)`: $A \cap B$ but with duplicates removed,
- `union(x, y)`: $A \cup B$ but with duplicates removed,
- `union_all(x, y)`: $A \cup B$,
- `setdiff(x, y)`: $A - B$ but with duplicates removed,
- `setequal(x, y)`: $A \cap B$.

Examples for set operators

```
# Define two sets (with one column):
A <- tibble(col1 = c(1L:4L))
B <- tibble(col1 = c(4L,4L,5L))

# Study some of the set-operations:
dplyr::intersect(A,B)
## # A tibble: 1 x 1
##    col1
##   <int>
## 1     4

union(A,B)
## # A tibble: 5 x 1
##    col1
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
## 5     5

union_all(A,B)
## # A tibble: 7 x 1
##    col1
##   <int>
## 1     1
## 2     2
## 3     3
```

PART 04: DATA WRANGLING

↓

CHAPTER 17: DATA WRANGLING IN THE TIDYVERSE

↓

SECTION 5:

String Manipulation in the tidyverse

Strings in the tidyverse

```r
library(tidyverse)
library(stringr)

# define strings
s1 <- "Hello"  # double quotes are fine
s2 <- 'world.' # single quotes are also fine

# Return the length of a string:
str_length(s1)
## [1] 5

# Concatenate strings:
str_c(s1, ", ", s2)       # str_c accepts many strings
## [1] "Hello, world."

str_c(s1, s2, sep = ", ") # str_c also has a
## [1] "Hello, world."
```

The str_* family of functions of stringr i

```
library(stringr)                    # or library(tidyverse)
sVector <- c("Hello", ", ", "world", "Philippe")

str_sub (sVector,1,3)               # the first 3 characters
## [1] "Hel" ", "  "wor" "Phi"

str_sub (sVector,-3,-1)             # the last 3 characters
## [1] "llo" ", "  "rld" "ppe"

str_to_lower(sVector[4])            # convert to lowercase
## [1] "philippe"

str_to_upper(sVector[4])            # convert to uppercase
## [1] "PHILIPPE"

str_c(sVector, collapse=" ")        # collapse into one string
## [1] "Hello ,  world Philippe"

str_flatten(sVector, collapse=" ")  # flatten string
## [1] "Hello ,  world Philippe"

str_length(sVector)                 # length of a string
## [1] 5 2 5 8
```

The str_* family of functions of stringr ii

```
# Nest the functions:
str_c(str_to_upper(str_sub(sVector[4],1,4)),
      str_to_lower(str_sub(sVector[4],5,-1))
     )
## [1] "PHILippe"

# Use pipes:
sVector[4]        %>%
   str_sub(1,4)   %>%
   str_to_upper()
## [1] "PHIL"
```

Duplicating strings with `stringr`

One of the most simple string manipulations is duplicating them to form a longer string. Here we ask `stringr` to produce a dark shade of grey.

```
str <- "F0"
str_dup(str, c(2,3))  # duplicate a string
## [1] "F0F0"   "F0F0F0"
```

Manage White Space

```
str <- c(" 1 ", "  abc", "Philippe De Brouwer   ")
str_pad(str, 5)  # fills with white-space to x characters
## [1] "   1 "                 "  abc"
## [3] "Philippe De Brouwer   "

# str_pad never makes a string shorter!
# So to make all strings the same length we first truncate:
str      %>%
  str_trunc(10) %>%
  str_pad(10,"right")   %>%
  print
## [1] " 1        " "  abc     " "Philipp..."

# Remove trailing and leading white space:
str_trim(str)
## [1] "1"                   "abc"                 "Philippe De Brouwer"

str_trim(str,"left")
## [1] "1 "                     "abc"
## [3] "Philippe De Brouwer   "

# Modify an existing string to fit a line length:
"The quick brown fox jumps over the lazy dog. "  %>%
   str_dup(5)    %>%
   str_c         %>%  # str_flatten also removes existing \n
   str_wrap(50)  %>%  # Make lines of 50 characters long.
   cat                # or writeLines (print shows "\n")
## The quick brown fox jumps over the lazy dog. The
## quick brown fox jumps over the lazy dog. The quick
## brown fox jumps over the lazy dog. The quick brown
## fox jumps over the lazy dog. The quick brown fox
## jumps over the lazy dog.
```

Determining Order and Sorting Strings

```r
str <- c("a", "z", "b", "c")

# str_order informs about the order of strings (rank number):
str_order(str)
## [1] 1 3 4 2

# Sorting is done with str_sort:
str_sort(str)
## [1] "a" "b" "c" "z"
```

Matching Regex Patterns with `stringr`

```
library(stringr)   # or library(tidyverse)
sV <- c("philosophy", "physiography", "phis",
        "Hello world", "Philippe", "Philosophy",
        "physics", "philology")

# Extracting substrings that match a regex pattern:
str_extract(sV, regex("Phi"))
## [1] NA    NA    NA    NA    "Phi" "Phi" NA    NA

str_extract(sV, "Phi")          # the same, regex assumed
## [1] NA    NA    NA    NA    "Phi" "Phi" NA    NA
```

Further Regex Matching stringr

```
str_extract(sV, "(p|P)hi")
## [1] "phi" NA    "phi" NA    "Phi" "Phi" NA    "phi"

# Or do it this way:
str_extract(sV, "(phi|Phi)")
## [1] "phi" NA    "phi" NA    "Phi" "Phi" NA    "phi"
```

This logic is easy to extend:

```
# Match also i and y:
str_extract(sV, "(p|P)h(i|y)")
## [1] "phi" "phy" "phi" NA    "Phi" "Phi" "phy" "phi"

# This is equivalent to:
str_extract(sV, "(phi|Phi|phy|Phy)")
## [1] "phi" "phy" "phi" NA    "Phi" "Phi" "phy" "phi"
```

Regex Syntax i

Anchors

`^`	begin of string or line
`$`	endof string (or line)
`\<`	beginning of a word
`\>`	end of a word

Special characters

`\n`	newline
`\r`	carriage return)
`\t`	tab
`\v`	vertical tab
`\f`	form feed

Character groups

`.`	any character, but `\n`
`[abc]`	accepted characters
`[a-z]`	character range
`(...)`	characters group

Quantifiers

`?`	0 or 1 times
`*`	0 or more
`+`	1 or more
`{n}`	n times
`{n,m}`	between n and m times
`{n,}`	n or more times
`{,m}`	m or less times

Regex Syntax ii

Logic

`|`	"OR," e.g. `(a|b)` matches a or b
`\1`	content of group one, e.g. `r(\w)g(\1)x` matches "regex"
`\2`	group two, e.g. `r(\w)g(\1)x(\2)xpr` matches "regexexpr"
`(?:..)`	non capturing group = ignore that match in the string to return
`[^a-d]`	"not": no character in range a to d

Lookaround – requires `PERL = TRUE`

`a(?!b)`	a not followed by b
`a(?=b)`	a if followed by b
`(?<=b)a`	a if preceded by b
`(?<!b)a`	a if not preceded by b

Other

`\Qa\E`	treat a verbatim, e.g. `\QC++?\E` matches "C++?"
`\K`	drop match so far, e.g. `x\K\d`returns from `x1` only `1`

Line modifiers

`(?i)`	makes all matches case insensitive
`(?s)`	single line mode: `.` also matches `\n`
`(?m)`	multi line mode: `^` and `$` become begin and end of line

Regex Syntax iii

POSIX Character classes

`[[:digit:]] or \d`	digit: `[0-9]`
`\\D`	not a digit: `[^0-9]`
`[[:xdigit:]] or \x`	hexadec. digits: `[0-9A-Fa-f]`
`[[:lower:]]`	lower-case: `[a-z]`
`[[:upper:]]`	upper-case: `[A-Z]`
`[[:alnum:]]`	alphanumeric: `[A-z0-9]`
`\\w`	word characters: `[A-z0-9_]`
`\\W`	not word characters: `[^A-z0-9_]`
`[[:blank:]]`	blank: `[\\s\\t]`
`[[:space:]] or \s`	space : `\\s`
`\\S`	not space: `[^\\s]`
`[[:punct:]]`	punctuation character : `!"#$%&'()*+,-./:;<=>?@[]^_`{}~|`
`[[:graph:]]`	graphical character : `[[:alnum:]] [[:punct:]]`
`[[:print:]]`	printable character : `[[:graph:]] [[:space:]]`
`[[:cntrl:]] or \c`	control characters: e.g. `[\\n\\rt]`

Lazy and Greedy Quantifiers i

The basic rule is that a quantifier applies to whatever is immediately left of it. For example:

- `abcd+` matches "abcdddd" but not "abcdabcd" (the + applies only to the last letter);
- this behaviour can be modified with grouping characters: `x(F1)+` will match "xF1F1F1," but also note that
- `\QC++\E+` matches "C+++++' but not "C+C+C+"

However, there are more nuances that need to be understood.

- The default quantifiers are **greedy:** `\d+` will match `123` (as many digits as possible, not necessarily all the same). In other words, a greedy quantifier gives you the longest possible match (eg, `^\.*` will match always the whole line). However, quantifiers are actually **greedy, but with good manners**. We mean with that the engine will swallow as many matches as possible, but if that would hinder the rest of the pattern to be matched, it will back-track to allow for a match. That is why `^\.*ippe` will still match `Philippe`.
- A quantifier can be made **reluctant or lazy** by adding `?` to it. For example, `^P\.*?` will match as little as possible within the possibilities of `*` (which is "zero or more" and hence defaults to "zero").
- Actually, quantifiers might be **reluctant or lazy but still benevolent**. Meaning if the match was so small that this would hinder the rest of the match to be made, then they will start matching more in order for the further match to be made possible.

Lazy and Greedy Quantifiers ii

Below we illustrate these concepts with greedy and lazy pattern matching:

```
str_extract("Philippe", "Ph\\w*")  # is greedy
## [1] "Philippe"

str_extract("Philippe", "Ph\\w*?") # is lazy
## [1] "Ph"
```

Regex for Humans with `rex` i

Regex expressions easily get hard to read. To solve that, there is a library `rex` that provides a function `rex()` to make the process of creating a regular expression a lot easier and a lot more readable.

```
# Load the library rex:
library(rex)

# In this example we construct the regex to match a valid URL, and will
# define the valid characters first:
valid_chars <- rex(one_of(regex('a-z0-9\u00a1-\uffff')))
```

Regex for Humans with rex ii

```
# Then build the regex:
expr <- rex(
  start,      # start of the string: ^

  # Protocol identifier (optional) + //
  group(list('http', maybe('s')) %or% 'ftp', '://'),

  # User: pass authentication (optional)
  maybe(non_spaces,
    maybe(':', zero_or_more(non_space)),
    '@'),

  # Host name:
  group(zero_or_more(valid_chars,
        zero_or_more('-')),
        one_or_more(valid_chars)),

  # Domain name:
  zero_or_more('.',
              zero_or_more(valid_chars,
              zero_or_more('-')),
              one_or_more(valid_chars)),

  # Top Level Domain (TLD) identifier
  group('.', valid_chars %>% at_least(2)),

  # Server port number (optional)
  maybe(':', digit %>% between(2, 5)),
```

Use the Regex created by `rex`

Now we have the regex stored in the variable `expr` and can use it:

```
# Print the result elegantly:
substring(expr, seq(1,  nchar(expr)-1, 40),
                seq(41, nchar(expr),   40))      %>%
  str_c(sep="\n")
## [1] "^(?:(?:http(?:s)?|ftp)://)(?:[^[:space:]]"
## [2] "]+(?::(?:[^[:space:]])*)?@)?(?:(?:[a-z0-9"
## [3] "9Âą-\uffff](?:-)*)*(?:[a-z0-9Âą-\uffff])+)(?:\\.(?:[a-"
## [4] "-z0-9Âą-\uffff](?:-)*)*(?:[a-z0-9Âą-\uffff])+)*(?:\\.("
## [5] "(?:[a-z0-9Âą-\uffff]){2,})(?::(?:[[:digit:]]){2"
## [6] ""
```

We can check if an URL is valid as follows:

```
# for example:
str_extract("www.de-brouwer.com", expr)
## [1] NA

str_extract("http://www.de-brouwer.com", expr)
## [1] "http://www.de-brouwer.com"

str_extract("error=www.de-brouwer.com", expr)
## [1] NA
```

Detect a Match

These functions will only report if a match is found, no information about starting positions of he match is given.

```
# grep() returns the whole string if a match is found:
grep(pattern, string, value = TRUE)
## [1] "one:1"  "c5c5c5" "d123d"  "123"    "6"

# The default for value is FALSE -> only returns indexes:
grep(pattern, string)
## [1] 1 3 4 5 6

# L for returning a logical variable:
grepl(pattern, string)
## [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE

# --- stringr ---
# similar to grepl (note order of arguments!)
str_detect(string, pattern)
## [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

Locate a Match

In many cases, it is not enough to know if there is a match, but also where the match occurs in the string; that is what we call "locating" a match in a string.

```
# Locate the first match (the numbers are the position in the string):
regexpr (pattern, string)
## [1]  5 -1  2  2  1  1
## attr(,"match.length")
## [1]  1 -1  1  1  1  1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE

# grepexpr() finds all matches and returns a list:
gregexpr(pattern, string)
## [[1]]
## [1] 5
## attr(,"match.length")
## [1] 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
##
## [[2]]
## [1] -1
## attr(,"match.length")
## [1] -1
## attr(,"index.type")
```

Replace a Match

Often we want to do more than just finding where a match occurs, but we want to change it with something else. This process is called “replacing” matches with strings.

```
# First, we need additionally a replacement (repl)
repl <- "___"

# sub() replaces the first match:
sub(pattern, repl, string)
## [1] "one:___"  "NO digit" "c___c5c5" "d___23d"  "___23"    "___"

# gsub() replaces all matches:
gsub(pattern, repl, string)
## [1] "one:___"      "NO digit"     "c___c___c___" "d_________d"
## [5] "_________"    "___"

# --- stringr ---
# str_replace() replaces the first match:
str_replace(string, pattern, repl)
## [1] "one:___"  "NO digit" "c___c5c5" "d___23d"  "___23"    "___"

# str_replace_all() replaces all mathches:
str_replace_all(string, pattern, repl)
## [1] "one:___"      "NO digit"     "c___c___c___" "d_________d"
## [5] "_________"    "___"
```

Extract

If it is not our aim to replace the match, then it might be the case that we want to extract it for further use and manipulation in other sections or functions. The following functions allow to extract matches to regular expressions. from strings. The output of these functions can be quite verbose such as the functions to locate matches.

```
# regmatches() with regexpr() will extract only the first match:
regmatches(string, regexpr(pattern, string))
## [1] "1" "5" "1" "1" "6"

# regmatches() with gregexpr() will extract all matches:
regmatches(string, gregexpr(pattern, string)) # all matches
## [[1]]
## [1] "1"
##
## [[2]]
## character(0)
##
## [[3]]
## [1] "5" "5" "5"
##
## [[4]]
## [1] "1" "2" "3"
##
## [[5]]
## [1] "1" "2" "3"
##
## [[6]]
## [1] "6"
```

Split Strings Using the Match as Separator

Finally, it might be useful to split strings based on a separator (for example file-names and file-extensions, dates, etc.). This can be done with the function `strsplit()`

```
# --- base-R ---
strsplit(string, pattern)
## [[1]]
## [1] "one:"
##
## [[2]]
## [1] "NO digit"
##
## [[3]]
## [1] "c" "c" "c"
##
## [[4]]
## [1] "d" ""  ""  "d"
##
## [[5]]
## [1] "" "" ""
##
## [[6]]
## [1] ""

# --- stringr ---
str_split(string, pattern)
## [[1]]
## [1] "one:" ""
##
```

PART 04: DATA WRANGLING

↓

CHAPTER 17: DATA WRANGLING IN THE TIDYVERSE

↓

SECTION 6:

Dates with lubridate

Manipulating dates with `lubridate`

We will load the package here and show this part of the code only once. All sub-sections that follow will use this package.

```r
# Load the tidyverse for its functionality such as pipes:
library(tidyverse)

# Lubridate is not part of the core-tidyverse, so we need
# to load it separately:
library(lubridate)
```

Date Format Conventions and Internal Storage

The first key concept is that of a date and a date-time. For most practical purposes, a date is something that can be stored as `yyyy-mmm-dd`.

It can be noted that R follows the ISO 8601 Notation and so will we do. Any person who believes in inclusion and not in imposing historical nation bound standards to the rest of the world will embrace the ISO standards, a fortiori any programmer or modeller with an inclusive world-view will also use the ISO 8601 standards. But there are many people who will not do this, and it is not uncommon to get dates in other formats or have to report dates in those formats. So, the format that we will use is the ISO format: `yyyy-mm-dd`, but we will also show how to convert to other systems.

Digression – R's internal date-format

Internally, R will store date-times as a Unix timestamp or POSIXct format:

```
as.numeric(Sys.time())   # the number of seconds passed since 1 January 1970
## [1] 1630480658

as.numeric(Sys.time()) / (60 * 60 * 24 * 365.2422)
## [1] 51.66792
```

Converting Strings and Numbers to Dates i

```
# There is a list of functions that convert to a date
mdy("04052018")
## [1] "2018-04-05"

mdy("4/5/2018")
## [1] "2018-04-05"

mdy("04052018")
## [1] "2018-04-05"

mdy("4052018")  # ambiguous formats are refused!
## [1] NA

dmy("04052018") # same string, different date
## [1] "2018-05-04"
```

Converting Strings and Numbers to Dates ii

Warning – Dates as numbers can be confusing

The functions of the family `ymd()` do not only take strings as input, they can also can take a numerical input. This might lead to confusion as it is not what one would expect: the internal representation of a date.

```
dt <- ymd(20180505)  %>% print
## [1] "2018-05-05"

as.numeric(dt)
## [1] 17656

ymd(17656)
## [1] NA
```

Timezones

```
# Note it converts the system time-zone to UTC:
as_datetime("2006-07-22T14:00")
## [1] "2020-06-07 22:14:00 UTC"

# Force time-zone:
as_datetime("2006-07-22T14:00 UTC")
## [1] "2020-06-07 22:14:00 UTC"

as_datetime("2006-07-22 14:00 Europe/Warsaw") #Fails silently!
## [1] "2020-06-07 22:14:00 UTC"

dt <- as_datetime("2006-07-22 14:00", tz = "Europe/Warsaw") %>%
      print
## [1] "2020-06-07 22:14:00 CEST"

# Get the same date-time numerals in a different time-zone:
force_tz(dt, "Pacific/Tahiti")
## [1] "2020-06-07 22:14:00 -10"

# Get the same cosmic moment in a new time-zone
with_tz(dt,  "Pacific/Tahiti")
## [1] "2020-06-07 10:14:00 -10"
```

Extract Date and Time Components

```
# We will use the date from previous hint:
dt1
## [1] "1890-12-29 08:00:00 MST"

year(dt)       # extract the year
## [1] 2020

month(dt)      # extract the month
## [1] 6

week(dt)       # extract the week
## [1] 23

day(dt)        # extract the day
## [1] 7

wday(dt)       # extract the day of the week as number
## [1] 1

qday(dt)       # extract the day of the quarter as number
## [1] 68

yday(dt)       # extract the day of the year as number
## [1] 159

hour(dt)       # extract the hour
```

Calculating with Date-Times: the Problems

```
moment1 <- as_datetime("2018-10-28 01:59:00", tz = "Europe/Warsaw")
moment2 <- as_datetime("2018-10-28 02:01:00", tz = "Europe/Warsaw")

moment2 - moment1  # Is it 2 minutes or 1 hour and 3 minutes?
## Time difference of 1.033333 hours

moment3 <- as_datetime("2018-10-28 03:01:00", tz = "Europe/Warsaw")

# The clocks were put back in this tz from 3 to 2am.
# So, there is 2 hours difference between 2am and 3am!
moment3 - moment1
## Time difference of 2.033333 hours
```

Calculating with Date-Times: the Concepts

1. **Duration:** A duration is the physical amount of time that has been elapsed between two events.
2. **Periods:** Track changes in clock times (so pretend that DST, leap seconds, and leap years do not exist).
3. **Intervals:** Periods of time defined by start and end date-time (duration or period can be extracted)

Calculating with Date-Times: Durations i

```
# Calculate the duration in seconds:
dyears(x = 1/365)
## [1] "86459.1780821918s (~1 days)"

dweeks(x = 1)
## [1] "604800s (~1 weeks)"

ddays(x = 1)
## [1] "86400s (~1 days)"

dhours(x = 1)
## [1] "3600s (~1 hours)"

dminutes(x = 1)
## [1] "60s (~1 minutes)"

dseconds(x = 1)
## [1] "1s"

dmilliseconds(x = 1)
## [1] "0.001s"

dmicroseconds(x = 1)
## [1] "1e-06s"

dnanoseconds(x = 1)
```

Calculating with Date-Times: Durations ii

```r
# Note that a duration object times a number is again a Duration object
# and it allows arithmetic:
dpicoseconds(x = 1) * 10^12
## [1] "1s"
```

```r
# Investigate the object type:
dur <- dnanoseconds(x = 1)
class(dur)
## [1] "Duration"
## attr(,"package")
## [1] "lubridate"

str(dur)
## Formal class 'Duration' [package "lubridate"] with 1 slot
##   ..@ .Data: num 1e-09

print(dur)
## [1] "1e-09s"
```

Calculating with Date-Times: Durations iii

If the duration is not given in one number, but for example in with units expressed as a string, we can use the function `duration()`. There is also a series of functions that can coerce to a duration, check if something is a duration:

```
# Useful for automation:
duration(5, unit = "years")
## [1] "157788000s (~5 years)"

# Coerce and logical:
dur <- dyears(x = 10)
as.duration(60 * 60 * 24)
## [1] "86400s (~1 days)"

as.duration(dur)
## [1] "315576000s (~10 years)"

is.duration(dur)
## [1] TRUE

is.difftime(dur)
## [1] FALSE

as.duration(dur)
## [1] "315576000s (~10 years)"

make_difftime(60, units="minutes")
## Time difference of 1 mins
```

Calculating with Date-Times: Periods

```
years(x = 1)
## [1] "1y 0m 0d 0H 0M 0S"

months(x = 1)
## [1] "1m 0d 0H 0M 0S"

weeks(x = 1)
## [1] "7d 0H 0M 0S"

days(x = 1)
## [1] "1d 0H 0M 0S"

hours(x = 1)
## [1] "1H 0M 0S"

minutes(x = 1)
## [1] "1M 0S"

seconds(x = 1)
## [1] "1S"

milliseconds(x = 1)
## [1] "0.001S"

microseconds(x = 1)
## [1] "1e-06S"
```

Calculating with Date-Times: the Intervals i

```
d1 <- ymd_hm("1939-09-01 09:00", tz = "Europe/Warsaw")
d2 <- ymd_hm("1945-08-15 12:00", tz = "Asia/Tokyo")

interval(d1, d2)  # defines the interval
## [1] 1939-09-01 09:00:00 CET--1945-08-15 05:00:00 CEST

# Or use the operator %--%:
ww2 <- d1 %--% d2 # defines the same interval

ww2 / days(1)   # the period expressed in days
## [1] 2174.833

ww2 / ddays(1)  # duration in terms of days
## [1] 2174.792

# The small difference is due to DST and equals one hour:
(ww2 / ddays(1) - ww2 / days(1)) * 24
## [1] -1

# Allow the interval to report on its length:
int_length(ww2) / 60 / 60 / 24
## [1] 2174.792
```

Calculating with Date-Times: the Intervals ii

The package `lubridate` provides a set of functions that allow to check if a date is in an interval, move the interval forward, etc.

Calculating with Date-Times: the Intervals iii

```
d_date <- ymd("19450430")

# Is a date or interval in another:
d_date %within% ww2
## [1] TRUE


ph <- interval(ymd_hm("1941-12-07 07:48", tz = "US/Hawaii"),
                     ymd_hm("1941-12-07 09:50", tz = "US/Hawaii")
                     )
ph %within% ww2    # is ph in ww2?
## [1] TRUE


int_aligns(ph, ww2) # do ww2 and ph share start or end?
## [1] FALSE


# Shift forward or backward:
int_shift(ww2, years(1))
## [1] 1940-09-01 09:00:00 CEST--1946-08-15 05:00:00 CEST


int_shift(ww2, years(-1))
## [1] 1938-09-01 09:00:00 CET--1944-08-15 05:00:00 CEST


# Swap start and end moment
flww2 <- int_flip(ww2)


# Coerce all to "positive" (start-date before end-date)
int_standardize(flww2)
```

PART 04: DATA WRANGLING

↓

CHAPTER 17: DATA WRANGLING IN THE TIDYVERSE

↓

SECTION 7:

Factors with forcats

Building Data for an Example: Survey Data

```r
set.seed(1911)
s <- tibble(reply = runif(n = 1000, min = 0, max = 13))
hml <- function (x = 0) {
  if (x < 0)  return(NA)
  if (x <= 4) return("L")
  if (x <= 8) return("M")
  if (x <= 12) return("H")
  return(NA)
  }
surv <- apply(s, 1, FUN = hml)  # output is a vector
surv <- tibble(reply = surv)  # coerce back to tibble
surv
## # A tibble: 1,000 x 1
##    reply
##    <chr>
##  1 H
##  2 M
##  3 L
##  4 M
##  5 L
##  6 H
##  7 L
##  8 H
##  9 <NA>
## 10 L
## # ... with 990 more rows
```

Building Data for an Example: Ordening the Labels

To put the labels in the right orders, we have to make clear to R that they are factors and that we have a specific order for our factors. This can be done with the argument `levels` in the function `parse_factor()`.

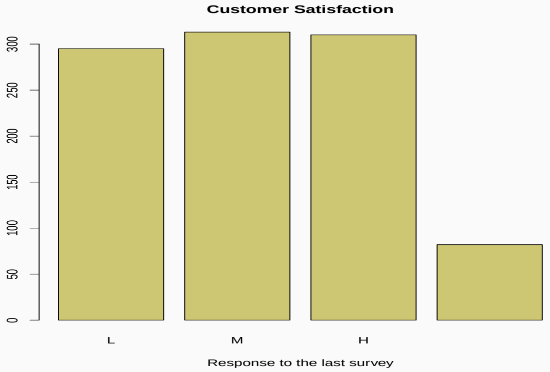
```r
# 1. Define the factor-levels in the right order:
f_levels <- c("L", "M", "H")

# 2. Define our data as factors:
survey <- parse_factor(surv$reply, levels = f_levels)
```

Building Data for an Example: Visualizing the Data

```
summary(survey)
##    L    M    H <NA>
##  295  313  310   82

plot(survey, col="khaki3",
     main = "Customer Satisfaction",
     xlab = "Response to the last survey"
     )
```



The Functions of `forcats`: Counting Labels

```
# Count the labels:
fct_count(survey)

## Error:  Tibble columns must have compatible sizes.
## * Size 4:  Existing data.
## * Size 3:  Column ‘n‘.
## i Only values of size one are recycled.
```

The Functions of `forcats`: Changing Labels i

```
# Relabel factors with fct_relabel:
HML <- function (x = NULL) {
  x[x == "L"] <- "Low"
  x[x == "M"] <- "Medium/High"
  x[x == "H"] <- "Medium/High"
  x[!(x %in% c("High", "Medium/High", "Low"))] <- NA
  return(x)
  }
f <- fct_relabel(survey, HML)
summary(f)
##         Low Medium/High        <NA>
##         295         623          82

plot(f, col="khaki3",
     main = "Only one third of customers is not happy",
     xlab = "Response to the expensive survey"
     )
```

The Functions of `forcats`: Changing Labels ii

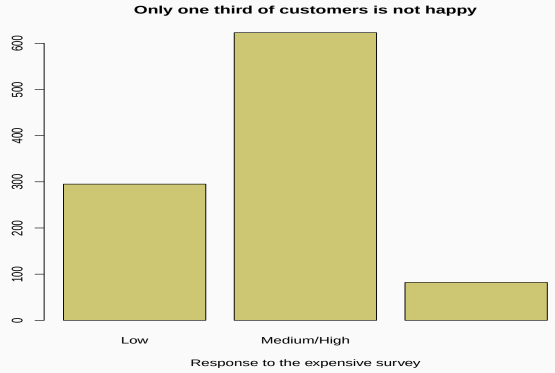


Figure 2: Maybe you would prefer to show this plot to the board meeting? This plot takes the two best categories together and creates the impression that more people are happy. Compare this to previous plot.

The Functions of `forcats`: More Examples i

```r
num_obs <- 1000  # the number of observations in the survey
# Start from a new survey: srv
srv <- tibble(reply = 1:num_obs)
srv$age <- rnorm(num_obs, mean=50,sd=20)
srv$age[srv$age < 15] <- NA
srv$age[srv$age > 85] <- NA

hml <- function (x = 0) {
  if (x < 0)  return(NA)
  if (x <= 4) return("L")
  if (x <= 8) return("M")
  if (x <= 12) return("H")
  return(NA)
  }

for (n in 1:num_obs) {
  if (!is.na(srv$age[n])) {
     srv$reply[n] <- hml(rnorm(n = 1, mean = srv$age[n] / 7, sd = 2))
   }
   else {
     srv$reply[n] <- hml(runif(n = 1, min = 1, max = 12))
   }
}
f_levels <- c("L", "M", "H")
srv$fct <- parse_factor(srv$reply, levels = f_levels)
```

The Functions of `forcats`: More Examples ii

Now that we have the data of the survey, we can showcase `forcats` and use it to visualize and manipulate the data.

```
# From most frequent to least frequent:
srv$fct                        %>%
fct_infreq(ordered = TRUE) %>%
  levels()
## [1] "M" "H" "L" NA
```

```
# From least frequent to more frequent:
srv$fct      %>%
 fct_infreq  %>%
  fct_rev    %>%
  levels
## [1] NA  "L" "H" "M"
```

```
# Reorder the reply variable in function of median age:
fct_reorder(srv$reply, srv$age) %>%
   levels
## [1] "H" "L" "M"
```

The Functions of `forcats`: More Examples iii

```
# Add the function min() to order based on the minimum
# age in each group (instead of default median):
fct_reorder(srv$reply, srv$age, min) %>%
   levels
## [1] "H" "L" "M"
```

The Functions of `forcats`: More Examples iv

```
# Show the means per class of satisfaction in base-R style:
by(srv$age, srv$fct, mean, na.rm = TRUE)
## srv$fct: L
## [1] 30.65112
## ----------------------------------------------------
## srv$fct: M
## [1] 44.41898
## ----------------------------------------------------
## srv$fct: H
## [1] 60.01358
## ----------------------------------------------------
## srv$fct: NA
## [1] 62.67211

# Much more accessible result with the dplyr:
satisf <- srv            %>%
          group_by(fct)  %>%
          summarize(
             age = median(age, na.rm = TRUE),
             n = n()
             )           %>%
         print
## # A tibble: 4 x 3
##   fct     age     n
##   <fct> <dbl> <int>
## 1 L      29.9   173
## 2 M      43.6   432
## 3 H      61.0   328
```

part 04: Data Wrangling

↓

chapter 18:

Dealing with Missing Data

part 04: Data Wrangling

↓

chapter 18: Dealing with Missing Data

↓

section 1:

Reasons for Data to be Missing

Missing Data: reasons

Typically data is missing for one of the following reasons:

1. input and pre-processing (e.g. conversion of units: some dates were in American format, others in UK format, some dates got misunderstood and others rejected; a decimal comma is not understood, data is copied to a system that does not recognize large number, etc.);
2. unclear or incomplete formulated questions (e.g. asking “are you male or female?”, while the possible answers in the questionnaire are “yes” and “no”);
3. fraud or other intend (if we know that young males will pay higher for a car insurance we might omit the box where the gender is put);
4. random reason (e.g. randomly skipped a question, interruption of financial markets due to external reason, mistake, typing a number too much, etc.)

Examples of Unclear questions

Example (Unclear questions)

Often, data is missing because questionnaires are written carelessly and formulated ambiguously. What to think about questions such as these:

1. rate the quality of the printed materials (1 . . . 5);
2. parent name, student name, study level;
3. it is never so that the teacher is too late: yes/no;
4. is the DQP sufficient to monitor the IMRDP and in line with GADQP? yes/no
5. I belong to a minority group (when applying for a job).

These example questions have – at least – the following issues.

1. The reader will assume that this question is there because the teacher will be assessed on the quality of printed materials and it also assumes that the student cares. Which is an assumption, that should be asked first. What would you do with this question if you did not want printed materials in the first place? Another common mistake is asking “how important are printed materials?” – also this question does not help to find whethere people assess the printed materials as positive or negative.
2. Whose study level are we asking here? That of the student or that of the parent?
3. Assume that the teacher is often late, then you can answer both “yes” or “no,” assume the opposite and the same holds. So what to answer?
4. This question has two common problems. First, it uses acronyms that might not be clear to everyone;

PART 04: DATA WRANGLING

↓

CHAPTER 18: DEALING WITH MISSING DATA

↓

SECTION 2:

Methods to Handle Missing Data

Alternative solutions to missing data i

1. Leave out the rows with missing data. If there is no underlying reason why data is missing, then leaving out the missing data is most probably harmless. This will work fine if the dataset is sufficiently large and of sufficient quality (even a dataset with hundred thousand lines but with 500 columns can lead to problems when we leave out all rows that miss one data point).
2. Carefully leave out rows with missing data. Same as above, but first make up our mind which variable will be in the model and then only leave out those rows that miss data on those rows.
3. Somehow fill in the missing data based on some rules or models. For example, we can replace the missing value by:
    1. the mean value for that column;
    2. median value for that column;
    3. conditional mean or median (e.g. fill in missing value for height with gender based mean) where some pre-existing logic or clear and well accepted rules hold;
    4. an educated guess (e.g. someone who scored all requested dimensions as “4/5” probably intended the missing value for “quality of printed materials” to be a “4/5” as well);
    5. the mid-value (if that makes sense for example the “3/5” could be a mid-value for “rate on a scale from 1 to 5”), this means choosing the middle of the possible values regardless which values occur more;
    6. replace the missing value by a more complex model, eventually based on machine learning, such as:
        - regression substitution, which tries to guess the missing value based on a multiple linear regression on other variables,
        - multiple imputation, which uses statistical methods to guess plausible values based on the data that is not missing (linear regression or machine learning) and then reset averages of the variables back by adding random errors in the predictions.

The Example Used in the Remainder of this Section

Example

For the purpose of this example, we will use this database and introduce some missing values.

```
set.seed(1890)
# Get the data:
d1       <- d0  <- iris

# Introduce the missing values:
i        <- sample(1:nrow(d0), round(0.20 * nrow(d0)))
d1[i,1]  <- NA
i        <- sample(1:nrow(d0), round(0.30 * nrow(d0)))
d1[i,2]  <- NA

# Show a part of the resulting dataset:
head(d1, n=10L)
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3            NA          NA          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6          NA          1.4         0.3  setosa
## 8            NA          NA          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
```

Predictive mean matching — PMM

For each observation that has a variable with a missing value, the method finds an observation (that has no missing value on this variable) with the closest predictive mean to that variable. The observed value from this observation is used as imputed value. This means that it preserves automatically many important characteristics such as skew, boundness (e.g. only positive data), base type (e.g. integer values only), etc.

The PMM Process

The PMM process is as follows:

1. Take all observations that have no missing values and fit a linear regression of variable *x* – that has the missing values – to one or more variables *y*, and produce a set of coefficients *b*.
2. Draw random coefficients b^* from the posterior predictive distribution of *b*. Typically, this would be a random draw from a multivariate normal distribution with mean *b* and the estimated covariance matrix of *b* (with an additional random draw for the residual variance). This step will ensure that there is sufficient variability in the imputed values.
3. Using b^*, generate predicted values for *x* for all cases (as well for those that have missing values in *x* as those that do not.
4. For each observation with missing *x*, identify a set of cases with observed *x* whose predicted values are close to the predicted value for the observation with missing data.
5. From those observations, randomly choose one and assign its observed value as value to be imputed to the missing *x*.
6. Repeat steps 2 – 5 till all the missing variables for *x* have an impute candidate.
7. Repeat steps 1 – 6 for all variables that have missing values.

part 04: Data Wrangling

↓

chapter 18: Dealing with Missing Data

↓

section 3:

R Packages to Deal with Missing Data

mice i

```r
#install.packages('mice')  # uncomment if necessary
library(mice)              # load the package

# mice provides the improved visualization function md.pattern():
md.pattern(d1)  # function provided by mice
```

mice ii

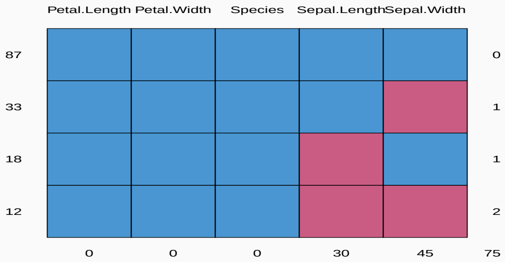


Figure 4: The visualization of missing data with the function `md.pattern()` of the package `mice`. This simple visualisation shows the contingents of data with missing values.

mice iii

```
##    Petal.Length Petal.Width Species Sepal.Length Sepal.Width
## 87            1           1       1            1           1  0
## 33            1           1       1            1           0  1
## 18            1           1       1            0           1  1
## 12            1           1       1            0           0  2
##               0           0       0           30          45 75
```

The table shows that the dataset *d*1 has 87 complete cases, 33 missing observations in Sepal.Width, 18 observations, where Sepal.Length is missing, and 12 cases where both are missing.

Imputing data with mice

```
d2_imp <- mice(d1, m = 5, maxit = 25, method = 'pmm', seed = 1500)
```

This created five possible datasets and we can select one completed set as follows.

```
# Choose set number 3:
d3_complete <- complete(d2_imp, 3)
```

missForest i

```
# install.packages('missForest') # only first time
library(missForest)                # load the library
d_mf <- missForest(d1)             # using the same data as before
##   missForest iteration 1 in progress...done!
##   missForest iteration 2 in progress...done!
##   missForest iteration 3 in progress...done!
##   missForest iteration 4 in progress...done!
##   missForest iteration 5 in progress...done!
##   missForest iteration 6 in progress...done!
##   missForest iteration 7 in progress...done!

# access the imputed data in the ximp attribute:
head(d_mf$ximp)
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1     5.100000    3.500000          1.4         0.2  setosa
## 2     4.900000    3.000000          1.4         0.2  setosa
## 3     4.516867    3.205947          1.3         0.2  setosa
## 4     4.600000    3.100000          1.5         0.2  setosa
## 5     5.000000    3.600000          1.4         0.2  setosa
## 6     5.400000    3.900000          1.7         0.4  setosa

# normalized MSE of imputation:
d_mf$OOBerror
##     NRMSE       PFC
## 0.1080024 0.0000000
```

missForest ii

Hmisc i

```
# Install the package first via:
# install.packages('Hmisc')
library(Hmisc)

# impute using mean:
SepLImp_mean <- with(d1, impute(Sepal.Length, mean))

# impute a randomly chosen value:
SepLImp_rand <- with(d1, impute(Sepal.Length, 'random'))

# impute the maximum value:
SepLImp_max <- with(d1, impute(Sepal.Length, max))

# impute the minimum value:
SepLImp_min <- with(d1, impute(Sepal.Length, min))

# note the '*' next to the imputed values"
head(SepLImp_min, n = 10L)
##    1    2    3    4    5    6    7    8    9   10
##  5.1  4.9 4.3*  4.6  5.0  5.4  4.6 4.3*  4.4  4.9
```

part 04: Data Wrangling

↓

chapter 19:

Data Binning

PART 04: DATA WRANGLING

↓

CHAPTER 19: DATA BINNING

↓

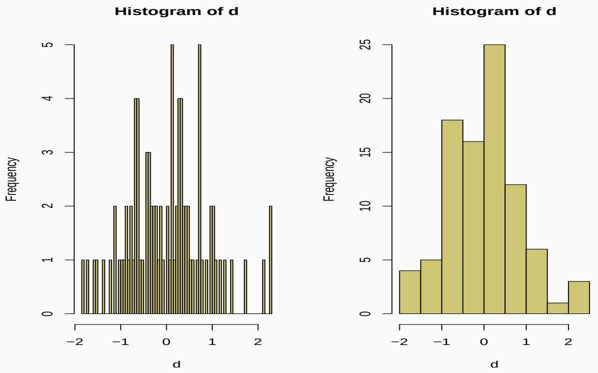
SECTION 1:

What is Binning and Why Use It

Histograms are a Binning Method

Consider the following simple example where we start with data drawn from a known distribution and plot the histogram (the output of this code is in Figure 5):

```
set.seed(1890)
d <- rnorm(90)
par(mfrow=c(1,2))
hist(d, breaks=70, col="khaki3")
hist(d, breaks=12, col="khaki3")
```



Binning with the Function `cut()`

```
# Try a possible cut
c <- cut(d, breaks = c(-3, -1, 0, 1, 2, 3))
table(c)
## c
## (-3,-1]  (-1,0]   (0,1]   (1,2]   (2,3]
##       9      34      37       7       3

# This is not good, it will not make solid predictions for the last bin.
# So, we neet to use other bins:
c <- cut(d, breaks = c(-3, -0.5, 0.5, 3))
table(c)
## c
##  (-3,-0.5] (-0.5,0.5]    (0.5,3]
##         27         41         22

# We have now a similar number of observations in each bin.
# Is that the only thing to think about?
```

PART 04: DATA WRANGLING

↓

CHAPTER 19: DATA BINNING

↓

SECTION 2:

Tuning the Binning Procedure

Generating the Data for this Section i

```r
set.seed(1890)
age <- rlnorm(1000, meanlog = log(40), sdlog = log(1.3))
y <- rep(NA, length(age))
for(n in 1:length(age)) {
  y[n] <- max(0,
              dnorm(age[n], mean= 40, sd=10)
                 + rnorm(1, mean = 0, sd = 10 * dnorm(age[n],
                   mean= 40, sd=15)) * 0.075)
}
y <- y / max(y)
plot(age, y,
     pch = 21, col = "blue", bg = "red",
     xlab = "age",
     ylab = "spending ratio"
     )
```

Generating the Data for this Section ii

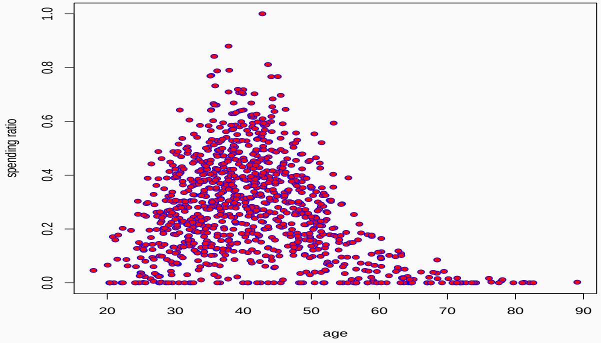


Figure 6: A plot of the fabricated dataset with the spending ratio in function of the age of the customers. The spending ratio is defined as $\frac{S_n}{S_{n-1}+S_n}$, where S_n is the spending in period n. If both spends are 0, then the spending ratio is defined as 0.

Generating the Data for this Section iii

```
# Assume this data is:
#   age           = age of customer
#   spending_ratio = R : = S_n/ (S_{n-1} + S_n)
#                         (zero if both are zero)
#     with S_n the spending in month n
dt <- tibble (age = age, spending_ratio = y)
```

Selection of bin borders i

```r
# Leave out NAs (in this example redundant):
d1 <- dt[complete.cases(dt),]

# order() returns sorted indices, so this orders the vector:
d1 <- d1[order(d1$age),]

# Fit a loess:
d1_loess <- loess(spending_ratio ~ age, d1)

# Add predictions:
d1_pred_loess <- predict(d1_loess)

# Plot the results:
par(mfrow=c(1,2))
plot(d1$age, d1$spending_ratio, pch=16,
     xlab = 'age', ylab = 'spending ratio')
lines(d1$age, d1_pred_loess, lwd = 7, col = 'dodgerblue4')
hist(d1$age, col = 'dodgerblue4', xlab = 'age')
```

Selection of bin borders ii

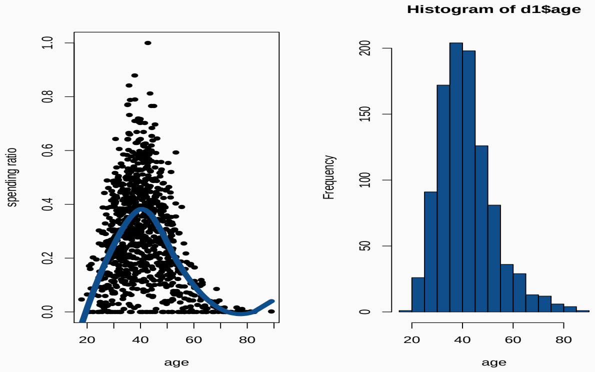


Figure 7: A simple aid to select binning borders is plotting a non-parametric fit (left) and the histogram (right). The information from both plots combined can be used to decide on binning.

```
par(mfrow=c(1,1))
```

Further Observations in this Example

From the histogram and loess estimate in Figure 7 on slide 117, we can see that:

- the spending ratio does not simply increase or decrease with age – the relation is non-linear;
- the local volatility is not constant (the dataset is “heteroscedastic”);
- we have little young customers and little older ones (it even looks as if some of those have a definite reason to be inactive on our Internet-shop).

A Model Without Binning i

To illustrate the effect of binning, we will use a logistic regression.[1] First, without binning and then with binning. Fitting the logistic regression worsk as follows:

```
# Fit the logistic regression directly on the data without binning:
lReg1 <- glm(formula = spending_ratio ~ age,
             family = quasibinomial,
             data = dt)
```

A Model Without Binning ii

```
# Investigate the model:
summary(lReg1)
##
## Call:
## glm(formula = spending_ratio ~ age, family = quasibinomial, data = dt)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.96344  -0.35725  -0.03202   0.25994   1.62106
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.09892    0.11733  -0.843    0.399
## age         -0.02107    0.00282  -7.473 1.71e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 0.1652355)
##
##     Null deviance: 195.63  on 999  degrees of freedom
## Residual deviance: 185.93  on 998  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 4

# Calculate predictions and means square error:
pred1 <- 1 / (1 + exp( -(coef(lReg1)[1] + dt$age * coef(lReg1)[2])))
SE1   <-  (pred1 - dt$spending_ratio)^2
```

A Model with Binning i

Inspired by Figure 7 on slide 117, we can make an educated guess of what bins would make sense. We choose bins that capture the dynamics of our data and make sure to have a bin for values with high spending rations and bins that have low spending ratios.

Now, we will introduce a simple data binning, calculate the logistic mode and show the results:

```
# Bin the variable age:
c <- cut(dt$age, breaks = c(15, 30, 55, 90))

# Check the binning:
table(c)
## c
## (15,30] (30,55] (55,90]
##     118     781     101


# We have one big bucket and two smaller (with the smallest
# more than 10% of our dataset.

lvls <- unique(c)      # find levels
lvls                   # check levels order
## [1] (30,55] (15,30] (55,90]
## Levels: (15,30] (30,55] (55,90]


# Create the tibble (a data-frame also works):
dt <- as_tibble(dt)                             %>%
      mutate(is_L = if_else(age <= 30, 1, 0))  %>%
      mutate(is_H = if_else(age > 55 , 1, 0))
```

A Model with Binning ii

```
# Investigate the logistic model:
summary(lReg2)
##
## Call:
## glm(formula = spending_ratio ~ is_L + is_H, family = quasibinomial,
##     data = dt)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.88247  -0.31393  -0.03812   0.22173   1.50439
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.74222    0.02791 -26.595   <2e-16 ***
## is_L        -0.85871    0.09404  -9.132   <2e-16 ***
## is_H        -2.20235    0.16876 -13.050   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 0.132909)
##
##     Null deviance: 195.63  on 999  degrees of freedom
## Residual deviance: 144.92  on 997  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 5

# Calculate predictions for our model and calculate MSE:
```

A Model with Binning iii

Finally, we can compare the *MSE* of both models:

```
# Compare the MSE of the two models:
MSE1
## [1] 0.03294673

MSE2
## [1] 0.02603179
```

We see that indeed the mean square error (MSE) is improved.[2] That is great: our model will make better predictions. However, what is even more important: the significance of our coefficients is up: we have now 3 stars for each *and* the significance of the intercept is up from 0 to 3 stars. That means that the model 2 is much more significant and hence robust to predict the future.

PART 04: DATA WRANGLING

↓

CHAPTER 19: DATA BINNING

↓

SECTION 3:

More Complex Cases: Matrix Binning

Generating the Data for Matrix Binning i

Generating the Data for Matrix Binning ii

```r
# Load libraries and define parameters:
library(tidyverse) # provides tibble (only used in next block)
set.seed(1880)     # to make results reproducible
N <- 500           # number of rows

# Ladies first:
# age will function as our x-value:
age_f   <- rlnorm(N, meanlog = log(40), sdlog = log(1.3))
# x is a temporary variable that will become the propensity to buy:
x_f <- abs(age_f + rnorm(N, 0, 20))    # Add noise & keep positive
x_f <- 1 - (x_f - min(x_f)) / max(x_f) # Scale between 0 and 1
x_f <- 0.5 * x_f / mean(x_f)           # Coerce mean to 0.5
# This last step will produce some outliers above 1
x_f[x_f > 1] <- 1   # Coerce those few that are too big to 1

# Then the gentlemen:
age_m   <- rlnorm(N, meanlog = log(40), sdlog = log(1.3))
x_m <- abs(age_m + rnorm(N, 0, 20))    # Add noise & keep positive
x_m <- 1 - (x_m - min(x_m)) / max(x_m) # Scale between 0 and 1
x_m <- 0.5 * x_m / mean(x_m)           # Coerce mean to 0.5
# This last step will produce some outliers above 1
x_m[x_m > 1] <- 1   # Coerce those few that are too big to 1
x_m <- 1 - x_m                         # relation to be increasing

# Rename (p_x is not the gendered propensity to buy)
p_f <- x_f
p_m <- x_m
```

Step 2 to Create the Data: Merge the Sets i

This first step was only to prepare the data and show what is exactly inside. In the next step, we will merge the data, and assume that this merged data set is what we got to work with. The following block of code will do this and then plot the histogram for the all observations (combined males and females) in Figure 9 on slide 128:

```
tf <- tibble("age" = age_f, "sex" = "F", "is_good" = p_f)
tm <- tibble("age" = age_m, "sex" = "M", "is_good" = p_m)
t  <- full_join(tf, tm, by = c("age", "sex", "is_good"))

# Change plot parameters and capture old values:
oldparams <- par(mfrow=c(1,2))
plot(t$age, t$is_good,
     pch  = 21, col = "black", bg = "khaki3",
     xlab = "Age",
     ylab = "Spending probability",
     main = "Dependence on age"
     )
fct_sex <- factor(t$sex, levels=c("F","M"), labels=c(0,1))
t$sexM  <- as.numeric(fct_sex)    # store for later use
plot(fct_sex, t$is_good,
     col="khaki3",
     main="Dependence on sex",
     xlab="Female       Male")
```

Step 2 to Create the Data: Merge the Sets ii

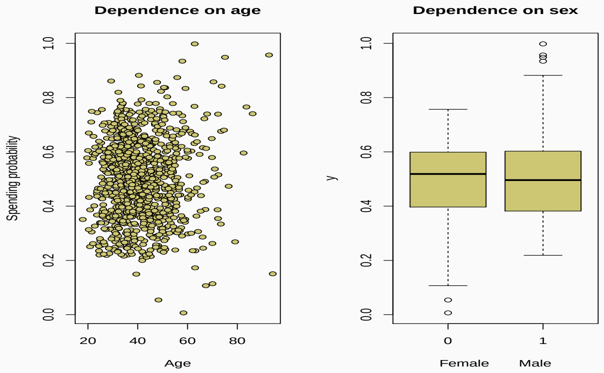


Figure 9: The dataset “as received from the customer service department” does not show any clear relationship between Age or Sex and the variable that we want to explain: the spending ratio.

```r
par(oldparams)   # Reset the plot parameters
```

Visualize the Fabricated Data i

```r
d1 <- t[complete.cases(t),]

d1 <- d1[order(d1$age),]
d1_age_loess <- loess(is_good ~ age, d1)
d1_age_pred_loess <- predict(d1_age_loess)

d1 <- d1[order(d1$sexM),]
d1_sex_loess <- loess(is_good ~ sexM, d1)
d1_sex_pred_loess <- predict(d1_sex_loess)

# Plot the results:
par(mfrow=c(2,2))
d1 <- d1[order(d1$age),]
plot(d1$age, d1$is_good, pch=16,
     xlab = 'Age', ylab = 'Spending probability')
lines(d1$age, d1_age_pred_loess, lwd = 7, col = 'dodgerblue4')
hist(d1$age, col = 'khaki3', xlab = 'age')

d1 <- d1[order(d1$sexM),]
plot(d1$sexM, d1$is_good, pch=16,
     xlab = 'Gender', ylab = 'Spending probability')
lines(d1$sexM, d1_sex_pred_loess, lwd = 7, col = 'dodgerblue4')
hist(d1$sexM, col = 'khaki3', xlab = 'gender')
```

Visualize the Fabricated Data ii

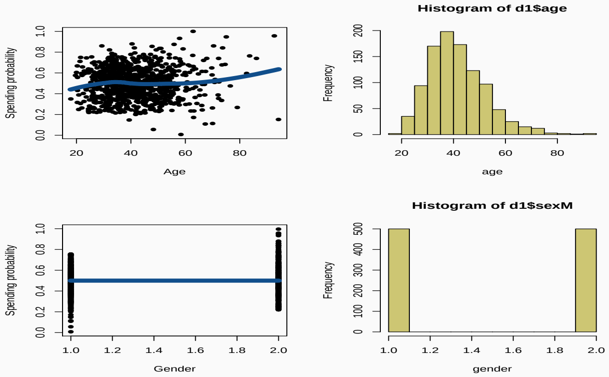


Figure 10: The data does not reveal much patterns for any of the variables (Gender and Age).

```
par(mfrow=c(1,1))
```

Model 1: a Logistic Regression i

```r
# Note that we can feed "sex" into the model and it will create
# for us a variable "sexM" (meaning the same as ours)
# To avoid this confusion, we put in our own variable.
regr1 <- glm(formula = is_good ~ age + sexM,
             family = quasibinomial,
             data = t)
```

Model 1: a Logistic Regression ii

```
# assess the model:
summary(regr1)
##
## Call:
## glm(formula = is_good ~ age + sexM, family = quasibinomial, data = t)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.15510  -0.21981   0.00556   0.20597   1.14979
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.990e-02  9.133e-02  -0.765    0.444
## age          1.684e-03  1.680e-03   1.002    0.316
## sexM         6.015e-05  3.730e-02   0.002    0.999
##
## (Dispersion parameter for quasibinomial family taken to be 0.08694359)
##
##     Null deviance: 91.316  on 999  degrees of freedom
## Residual deviance: 91.229  on 997  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 3

pred1 <- 1 / (1+ exp(-(coef(regr1)[1] + t$age * coef(regr1)[2]
                     + t$sexM * coef(regr1)[3])))
SE1   <- (pred1 - t$is_good)^2
MSE1  <- sum(SE1) / length(SE1)
```

Model 2: a Binned Variable that Combines Sex and Age i

```
# 1. Check the potential cut:
c <- cut(t$age, breaks = c(min(t$age), 35, 55, max(t$age)))
table(c)
## c
## (17.9,35]   (35,55] (55,94.2]
##       300       591       108
```

Model 2: a Binned Variable that Combines Sex and Age ii

```
# 2. Create the matrix variables:
t <- as_tibble(t)                                                %>%
    mutate(is_LF = if_else((age <= 35) & (sex == "F"), 1L, 0L)) %>%
    mutate(is_HF = if_else((age >  50) & (sex == "F"), 1L, 0L)) %>%
    mutate(is_LM = if_else((age <= 35) & (sex == "M"), 1L, 0L)) %>%
    mutate(is_HM = if_else((age >  50) & (sex == "M"), 1L, 0L)) %>%
    print
## # A tibble: 1,000 x 8
##      age sex   is_good  sexM is_LF is_HF is_LM is_HM
##    <dbl> <chr>   <dbl> <dbl> <int> <int> <int> <int>
##  1  44.3 F       0.564     1     0     0     0     0
##  2  38.3 F       0.636     1     0     0     0     0
##  3  38.9 F       0.552     1     0     0     0     0
##  4  58.5 F       0.351     1     0     1     0     0
##  5  31.5 F       0.623     1     1     0     0     0
##  6  48.4 F       0.487     1     0     0     0     0
##  7  28.9 F       0.552     1     1     0     0     0
##  8  29.9 F       0.493     1     1     0     0     0
##  9  30.1 F       0.549     1     1     0     0     0
## 10  51.1 F       0.241     1     0     1     0     0
## # ... with 990 more rows
```

Model 2: a Binned Variable that Combines Sex and Age iii

```
# 3. Check if the final bins aren't too small:
t[,5:8] %>% map_int(sum)
## is_LF is_HF is_LM is_HM
##   154   104   147   101
```

Model 2: a Logistic Regression with a Variable that Combines Sex and Age i

```r
regr2 <- glm(formula = is_good ~ is_LF + is_HF + is_LM + is_HM,
             family = quasibinomial,
             data = t)
```

Model 2: a Logistic Regression with a Variable that Combines Sex and Age ii

```
# Assess the model:
summary(regr2)
##
## Call:
## glm(formula = is_good ~ is_LF + is_HF + is_LM + is_HM, family = quasibinomial,
##     data = t)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.98606  -0.18858  -0.00424   0.18159   0.98651
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.01768    0.02467  -0.716    0.474
## is_LF        0.27945    0.05094   5.485 5.23e-08 ***
## is_HF       -0.35564    0.06002  -5.925 4.29e-09 ***
## is_LM       -0.22844    0.05183  -4.408 1.16e-05 ***
## is_HM        0.45028    0.06106   7.375 3.46e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for quasibinomial family taken to be 0.07518569)
##
##     Null deviance: 91.316  on 999  degrees of freedom
## Residual deviance: 78.274  on 995  degrees of freedom
## AIC: NA
##
## Number of Fisher Scoring iterations: 3
```

Compare the Performance of Model 1 and Model 2

Finally, we also note that the MSE has improved too:

```
MSE1
## [1] 0.02166756

MSE2
## [1] 0.01844601
```

Questions

Question #1 Binary dependent variables

In many cases, the dependent variable will be binary (0 or 1, “yes” or “no”). That means that we are trying to model a yes/no decision. For example, 1 can be a customer that defaulted on a loan, a customer to receive a special offer, etc.

```r
t <- mutate(t, "is_good" = if_else(is_good >= 0.5, 1L, 0L))
```

Remake model 1 (logistic regression in function of age and sexM) and model 2 (logistic regression for the variables is_LF, is_HF, is_LM, and is_HM). What does this change? Are the conclusions different?

Question #2 Think outside the box

In this particular case, what other approach would you suggest?

PART 04: DATA WRANGLING

↓

CHAPTER 19: DATA BINNING

↓

SECTION 4:

Weight of Evidence and Information Value

Weight of Evidence (WOE)

For each bin *i* of variable *j* (or for each binary variable *j*) is defined as

$$WOE_{ij} = \log\left\{\frac{\frac{\#G_{ij}}{\#G}}{\frac{\#B_{ij}}{\#B}}\right\}$$

Where $\#G_{ij}$ is the number of “good observations” (binary variable is 1) in bin *i* for variable *j*. $\#G$is the number of good observations for the whole dataset, and “B” refers to the “bad observations” (i.e. where the dependent variable is 0).

This makes WOE a measure of predicting power of a binned variable.

Information Value (IV)

The information value of bin *i* for variable *j* is defined as

$$IV_{ij} = \left(\frac{\#G_{ij}}{\#G} - \frac{\#B_{ij}}{\#B} \right) .WOE_{ij}$$

Rule of thumb:

IV	Predictability
< 0.02	Not predictive
$0.02 - 0.3$	Weak
$0.1 - 0.3$	Medium
$0.3 - 0.5$	Strong
> 0.5	Suspicious

Table 1: Different levels of information value and their commonly accepted interpretation – which works good in the environment of credit data for example.

WOE and IV in R i

```
# We start from this dataset used in previous section:
print(t)
## # A tibble: 1,000 x 8
##      age sex   is_good  sexM is_LF is_HF is_LM is_HM
##    <dbl> <chr>   <int> <dbl> <int> <int> <int> <int>
##  1  44.3 F           1     1     0     0     0     0
##  2  38.3 F           1     1     0     0     0     0
##  3  38.9 F           1     1     0     0     0     0
##  4  58.5 F           0     1     0     1     0     0
##  5  31.5 F           1     1     1     0     0     0
##  6  48.4 F           0     1     0     0     0     0
##  7  28.9 F           1     1     1     0     0     0
##  8  29.9 F           0     1     1     0     0     0
##  9  30.1 F           1     1     1     0     0     0
## 10  51.1 F           0     1     0     1     0     0
## # ... with 990 more rows
```

This dataset contains a specific property where males and females have a similar propensity to spend as an average population. However, this propensity is decreasing for females and increasing for males. This situation is particularly difficult, since at first glance the variables Age and Sex will not be predictive at all. We need to look at the interactions between the variables in order to find the underlying relations.

WOE and IV in R ii

Now, that we have data, we can load the package `InformationValue`, create a weight of evidence table and calculate the information value for a given variable:

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

WOETable(X = factor(t$sexM), Y = t$is_good, valueOfGood=1)    %>%
   knitr::kable(format.args = list(big.mark = " ", digits=2))
```

CAT	GOODS	BADS	TOTAL	PCT_G	PCT_B	WOE	IV
1	267	233	500	0.52	0.48	0.088	0.0039
2	245	255	500	0.48	0.52	-0.088	0.0039

```
## also functions WOE() and IV(), e.g.
# IV of a categorical variable is the sum of IV of its categories
IV(X = factor(t$sexM), Y = t$is_good, valueOfGood=1)
## [1] 0.007757952
## attr(,"howgood")
## [1] "Not Predictive"
```

Using Matrix Variables and the Function IV()

Dividing the data based on gender is not sufficient, and it does not work. Using our variables, such as is_LF (female from the lower age group), which combine the information of age and gender should work better.

```
WOETable(X = factor(t$is_LF), Y = t$is_good, valueOfGood=1) %>%
   knitr::kable(digits=2)
```

CAT	GOODS	BADS	TOTAL	PCT_G	PCT_B	WOE	IV
0	396	450	846	0.77	0.92	-0.18	0.03
1	116	38	154	0.23	0.08	1.07	0.16

```
# The package porvides also functions WOE() and IV().
# The IV of a categorical variable is the sum of IV of its categories.
IV(X = factor(t$is_LF), Y = t$is_good, valueOfGood=1)
## [1] 0.1849507
## attr(,"howgood")
## [1] "Highly Predictive"
```

Question

Question #3

Consider the dataset `mtcars` and investigate if the gearbox type (the variable `am` is a good predictor for the layout of the motor (the variable `vs`, V-motor or not). Do this by using WOE and IV.

part 04: Data Wrangling

$\downarrow$

chapter 20:

Factoring Analysis and Principle Components

PART 04: DATA WRANGLING

↓

CHAPTER 20: FACTORING ANALYSIS AND PRINCIPLE COMPONENTS

↓

SECTION 1:

Principle Components Analysis (PCA)

PCA in R and Visualising Results i

```
fit <- princomp(mtcars, cor=TRUE)

summary(fit)      # print the variance explained by PC
## Importance of components:
##                           Comp.1    Comp.2     Comp.3     Comp.4
## Standard deviation     2.5706809 1.6280258 0.79195787 0.51922773
## Proportion of Variance 0.6007637 0.2409516 0.05701793 0.02450886
## Cumulative Proportion  0.6007637 0.8417153 0.89873322 0.92324208
##                            Comp.5     Comp.6     Comp.7     Comp.8
## Standard deviation     0.47270615 0.45999578 0.36777981 0.35057301
## Proportion of Variance 0.02031374 0.01923601 0.01229654 0.01117286
## Cumulative Proportion  0.94355581 0.96279183 0.97508837 0.98626123
##                             Comp.9     Comp.10     Comp.11
## Standard deviation     0.277572792 0.228112781 0.148473587
## Proportion of Variance 0.007004241 0.004730495 0.002004037
## Cumulative Proportion  0.993265468 0.997995963 1.000000000
```

PCA in R and Visualising Results ii

```
loadings(fit)     # show PC loadings
##
## Loadings:
##      Comp.1 Comp.2 Comp.3 Comp.4 Comp.5 Comp.6 Comp.7 Comp.8 Comp.9
## mpg   0.363         0.226         0.103  0.109  0.368  0.754  0.236
## cyl  -0.374         0.175               -0.169         0.231
## disp -0.368                -0.257  0.394  0.336  0.214         0.198
## hp   -0.330  0.249 -0.140         0.540                0.222 -0.576
## drat  0.294  0.275 -0.161 -0.855        -0.244
## wt   -0.346 -0.143 -0.342 -0.246         0.465                0.359
## qsec  0.200 -0.463 -0.403        -0.165  0.330         0.232 -0.528
## vs    0.307 -0.232 -0.429  0.215  0.600 -0.194 -0.266         0.359
## am    0.235  0.429  0.206                0.571 -0.587
## gear  0.207  0.462 -0.290  0.265         0.244  0.605 -0.336
## carb -0.214  0.414 -0.529  0.127 -0.361 -0.184 -0.175  0.396  0.171
##      Comp.10 Comp.11
## mpg   0.139   0.125
## cyl  -0.846   0.141
## disp         -0.661
## hp    0.248   0.256
## drat -0.101
## wt            0.567
## qsec -0.271  -0.181
## vs   -0.159
## am   -0.178
## gear -0.214
## carb         -0.320
##
```

PCA before the model

Hint – Executing PCA before fitting a model

Since a lot of variance is explained in the first PCs, it is a good idea to fit any model (such as a logistic regression or any other model) not directly on `mtcars`, but rather on its principal components.
This will make the model more stable and one can expect the model to perform better out of sample, the only cost is the loss of transparency of the model.[a]

[a] If the first principal components can be summarizes as a certain concept, then there is little to no loss of transparency. However, usually, the PCs are composed of too many variables and do not summarize as one concept.

part 04: Data Wrangling

↓

chapter 20: Factoring Analysis and Principle Components

↓

section 2:

Factor Analysis

Factor Analysis in R i

Factor Analysis in R ii

```
# Maximum Likelihood Factor Analysis

# Extracting 3 factors with varimax rotation:
fit <- factanal(mtcars, 3, rotation = "varimax")
print(fit, digits = 2, cutoff = .3, sort = TRUE)
##
## Call:
## factanal(x = mtcars, factors = 3, rotation = "varimax")
##
## Uniquenesses:
##  mpg  cyl disp   hp drat   wt qsec   vs   am gear carb
## 0.13 0.06 0.09 0.13 0.29 0.06 0.05 0.22 0.21 0.12 0.16
##
## Loadings:
##      Factor1 Factor2 Factor3
## mpg   0.64   -0.48   -0.47
## disp -0.72    0.54    0.32
## drat  0.80
## wt   -0.78            0.52
## am    0.88
## gear  0.91
## cyl  -0.62    0.70
## hp            0.72    0.51
## qsec         -0.95
## vs           -0.80
## carb          0.56    0.72
##
##                Factor1 Factor2 Factor3
## SS loadings       4.38    3.52    1.58
```

Factor Analysis: Deciding How Many Factors to Use i

```
# load the library nFactors:
library(nFactors)
```

Then we can perform the analysis, get the optimal number of factors, and plot a visualisation:

```
# Get the eigenvectors:
eigV <- eigen(cor(mtcars))


# Get mean and selected quantile of the distribution of eigen-
# values of correlation or a covariance matrices of standardized
# normally distributed variables:
aPar <- parallel(subject = nrow(mtcars),var = ncol(mtcars),
                 rep = 100, cent = 0.05)

# Get the optimal number of factors analysis:
nScr <- nScree(x = eigV$values, aparallel = aPar$eigen$qevpea)

# See the result
nScr
##   noc naf nparallel nkaiser
## 1   2   1         2       2

# and plot it.
plotnScree(nScr)
```

Factor Analysis: Deciding How Many Factors to Use ii

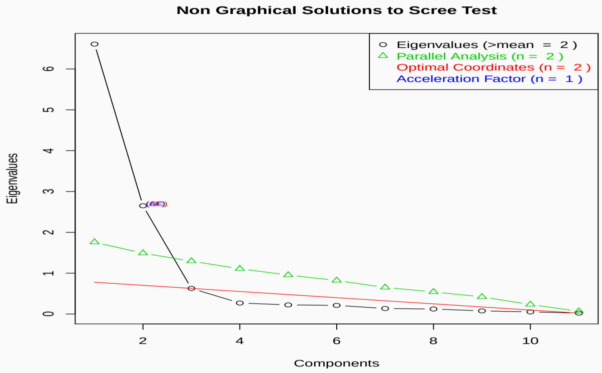


Figure 14: Visual aids to select the optimal number of factors.