An Introduction to Data Science for Sensory and Consumer Scientists

John Ennis, Julien Delarue, and Thierry Worch

2020-12-29

Contents

Preface

Welcome to the website for *Introduction to Data Science for Sensory and Consumer Scientists*. This book being written in the open and is currently under development.

About the Authors

John Ennis …

Julien Delarue …

Thierry Worch …

Introduction

Chapter 1

Introduction

Sensory and consumer science (SCS) is consider as a pillar of food science and technology and is useful to product development, quality control and market research. Most scientific and methodological advances in the field are applied to food. This book makes no exception as we chose a cookie formulation dataset as a main thread. However, SCS widely applies to many other consumer goods so are the content of this book and the principles set out below.

1.1 Core principles in Sensory and Consumer Science

1.1.1 Measuring and analyzing human responses

Sensory and consumer science aims at measuring and understanding consumers' sensory perceptions as well as the judgements, emotions and behaviors that may arise from these perceptions. SCS is thus primarily a science of measurement, although a very particular one that uses human beings and their senses as measuring instruments. In other words, sensory and consumer researchers measure and analyze human responses. To this end, SCS relies essentially on sensory evaluation which comprises a set of techniques that mostly derive from psychophysics and behavioral research. It uses psychological models to help separate signal from noise in collected data [ref O'Mahony, D.Ennis, others?]. Besides, sensory evaluation has developed its own methodological framework that includes most refined techniques for the accurate measurement of product sensory properties while minimizing the potentially biasing effects of brand identity and the influence of other external information on consumer perception [Lawless & Heymann, 2010]. A detailed description of sensory methods is beyond the scope of this book and many textbooks on sensory evaluation methods are available to readers seeking more information. However, just to give a brief overview, it is worth remembering that sensory methods can be roughly divided into three categories, each of them bearing many variants: - Discrimination tests that aim at detecting subtle differences between two products. - Descriptive analysis (DA), also referred to as 'sensory profiling', aims at providing both qualitative and quantitative information about product sensory properties. - Hedonic tests. This category gathers affective tests that aim at measuring consumers' liking for the tested products or their preferences among a product set. Each of these test categories generates its own type of data and related statistical questions in relation to the objectives of the study. Typically, data from difference tests consist in series of correct/failed binary answers depending on whether judges successfully picked the odd sample(s) among a set of three or more samples. These are used to determine whether the number of correct choices is above the level expected by chance. Conventional descriptive analysis data consist in intensity scores given by each panelist to evaluated samples on a series of sensory attributes, hence resulting in a product x attribute x panelist dataset (Figure 1). Note that depending on the DA method, quantifying means other than intensity ratings can be used (ranks, frequency, etc.). Most frequently, each panelist evaluates all the samples in the product set. However, the use of balanced incomplete design can also be found when the experimenters aim to limit the number of samples evaluated by each subject. Eventually, hedonic test datasets consist in hedonic scores (ratings for consumers' degree of liking or preference ranks) given by each interviewed consumer to a series of products. As for DA, each consumer usually evaluates all the samples in the product set, but balanced incomplete designs are sometimes used too. In addition, some companies favor pure monadic evaluation of product (i.e. between-subject design or independent groups design) which obviously result in unrelated sample datasets. Sensory and consumer researchers also borrow methods from other fields, in particular from sociology and experimental psychology. Definitely a multidisciplinary area, SCS develops in many directions and reaches disciplines that range from genetics and physiology to social marketing, behavioral economics and computational neuroscience. So have diversified the types of data sensory and consumer scientists must deal with.

1.2 How should sensory and consumer scientists learn data science?

1.3 Caution: Don't that everybody does

1.4 Example projects

Chapter 2

What is Data Science?

In this chapter we explain what is data science.

2.1 History and Definition

Data science has been called the "sexiest job of the 21st century" by Harvard Business Review [insert DJ Patil reference], but what is it? As with all rapidly growing fields, the definition depends on who you ask. Before we give our definition, however, we provide a brief history for context.

To begin, we note that there was a movement in early computer science to call their field "data science." Chief among the advocates for this viewpoint was Peter Naur, winner of the 2005 Turing award. This viewpoint is detailed in the preface to his 1974 book, "Concise Survey of Computer Methods," where he states that data science is "the science of dealing with data, once they have been established." From his perspective, this is the purpose of computer science. This viewpoint is echoed in the statement, often attributed to Edsger Dijkstr, that "Computer science is no more about computers than astronomy is about telescopes."

Interestingly, a similar movement arose in statistics, starting in 1962 with John Tukey's statements that "Data analysis, and the parts of statistics which adhere to it, must … take on the characteristics of science rather than those of mathematics" and that "data analysis is intrinsically an empirical science." This movement culminated in 1997 when Jeff Wu proposed during his inaugural lecture, upon becoming the chair of the University of Michigan's statistics department, that statistics should be called data science.

These two movements came together in 2001 in William S. Cleveland's paper "Data Science: An Action Plan for Expanding the Technical Areas in the Field of Statistics." In this highly influential monograph, Cleveland makes the key assertion that "The value of technical work is judged by the extent ot which it benefits the data analyst, either directly or indirectly."

[FOOTNOTE: It is worth noting that these two movements were connected by substantial work in the areas of statistical computing, knowledge discovery, and data mining, with important work contributed by Gregory Piatetsky-Shapiro, Usama Fayyad, and Padhraic Smyth among many others.]

Putting this history together, we provide our definition of **data science** as: The intersection of statistics, computer science, and industrial design. Accordingly, we use the following three definitions of these fields:

- **Statistics**: The branch of mathematics dealing with the collection, analysis, interpretation, and presentation of masses of numerical data.
- **Computer Science**: Computer science is the study of processes that interact with data and that can be represented as data in the form of programs.
- **Industrial Design**: The professional service of creating and developing concepts and specifications that optimize the function, value, and appearance of products and systems for the mutual benefit of both user and manufacturer.

Hence data science is the production of useful things through the collection, processing, analysis, and interpretation of data.

2.2 Workflow

A schematic of a data scientific workflow is shown in Figure 2.1. Each section is described in greater detail below.

2.2.1 Data Preparation

2.2.1.1 Inspect

Goal: Gain familiarity with the data Key Steps: Learn collection details Check data imported correctly Determine data types Ascertain consistency and validity Tabulate and compute other basic summary statistics Create basic plots of key variables of interest

2.2.1.2 Clean

Goal: Prepare data for analysis Key Steps: Remove/correct errors Make data formatting consistent Organize text data Create tidy data (one observation per row) Organize data into related tables Document all choices

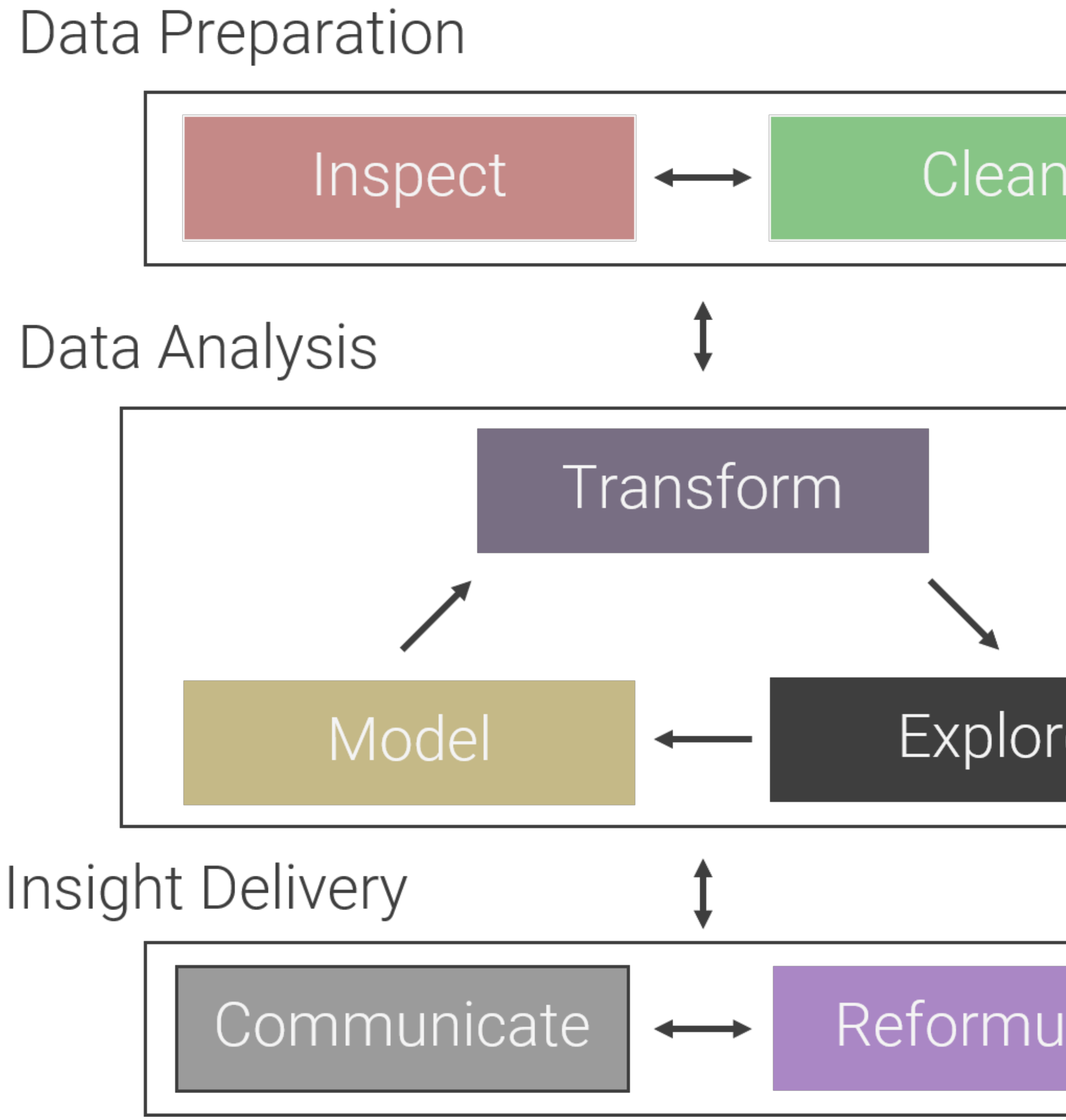

Figure 2.1: Data scientific workflow.

2.2.2 Data Analysis

2.2.2.1 Transform

Goal: Adjust data as needed for analysis Key Steps: Create secondary variables Decorrelate data Identify latent factors Engineer new features

2.2.2.2 Explore

Goal: Allow data to suggest hypotheses Key Steps: Graphical visualizations Exploratory analyses Note: Caution must be taken to avoid high false discovery rate when using automated tools

2.2.2.3 Model

Goal: Conduct formal statistical modeling Key Steps: Conduct traditional statistical modeling Build predictive models Note: This step may feed back into transform and explore

2.2.3 Insight Delivery

2.2.3.1 Communicate

Goal: Exchange research information Key Steps: Automate reporting as much as possible Share insights Receive feedback Note: Design principles essential to make information accessible

2.2.3.2 Reformulate

Goal: Incorporate feedback into workflow Key Steps: Investigate new questions Revise communications Note: Reformulation make take us back to data cleaning

2.3 Benefits of Data Science

2.3.1 Reproducible Research

- Time savings
- Collaboration
- Continuous improvement

2.3.2 Data-Driven Decision Making

2.3.3 Standardized Data Collection

2.3.4 Standardized Reporting

- Especially valuable when there are multiple sites globally

2.3.5 Improved Business Impact

2.4 How to Learn Data Science

Learning data science is much like learning a language or learning to play an instrument - you have to practice. Our advice based on mentoring many students and clients is to get started sooner rather than later, and to accept that the code you'll write in the future will always be better than the code you'll write today. Also, many of the small details that separate an proficient data scientist from a novice can only really be learned through practice as there are too many small details to learn them all in advice. So, starting today, do your best to write at least some code for all your projects. If a time deadline prevents you from completing the analysis in R, that's fine, but at least gain the experience of making an RStudio project and loading the data in R. Then, as time allows, try to duplicate your analyses in R, being quick to search for solutions when you run into errors. Often simply copying and pasting your error into a search engine will be enough to find the solution to your problem. Moreover, searching for solutions is its own skill that also requires practice. Finally, if you are really stuck, reach out to a colleague (or even the authors of this book) for help

2.5 How to Use This Book

We recommend following the instructions in Chapter 3 to get started.

2.6 Common Data Science Tools

2.7 Why R?

For sensory and consumer scientists, we recommend the R ecosystem of tools for three main reasons. The first reason is cultural - R has from its inception been oriented more towards statistics than to computer science, making the feeling of programming in R more natural (in our experience) for sensory and consumer scientists than programming in Python. This opinion of experience is not to say that a sensory and consumer scientist shouldn't learn Python if they are so inclined, or even that Python tools aren't sometimes superior to R tools (in fact, they sometimes are). This latter point leads to our second reason, which is that R tools are typically better suited to sensory and consumer science than are Python tools. Even when Python tools are superior, the R tools are still sufficient for sensory and consumer science purposes, plus there are many custom packages such as SensR, SensoMineR, and FactorMineR that have been specifically developed for sensory and consumer science. Finally, the recent work by the RStudio company, and especially the exceptional work of Hadley Wickham, has lead to a very low barrier to entry for programming within R together with exceptional tools for data manipulation.

We continue our discussion of getting started with R in the next chapter.

Chapter 3

Getting Started with R

3.1 R

R is an open-source programming language and software environment First released in 1995, R is an open-source implementation of S R was developed by Ross Ihaka and Robert Gentleman The name "R" is partly a play on Ihaka's and Gentleman's first names R is a scripting language (not a compiled language) Lines of R code run (mostly) in order R is currently the 7th most popular programming language in the world

3.1.1 Why Learn a Programming Language?

Control Speed Reduced errors Increased capability Continuous improvement Improved collaboration Reproducible results

3.1.2 Why R?

R originated as a statistical computing language It has a culture germane to sensory science R is well-supported with an active community Extensive online help is available Many books, courses, and other educational material exist The universe of available packages is vast R excels at data manipulation and results reporting R has more specialized tools for sensory analysis than other programming language

3.1.3 Steps to Install R

https://cran.r-project.org/bin/windows/base/ Download the latest version of R Install R with default options You will almost certainly be running 64-bit

R Note: If you are running R 4.0 or higher, you might need to install Rtools: https://cran.r-project.org/bin/windows/Rtools/

3.2 RStudio

3.2.1 Steps to Install RStudio

https://www.rstudio.com/products/rstudio/download/#download Download and install the latest (almost certainly 64-bit) version of RStudio with default options Adjustments: Uncheck "Restore .RData into workspace at startup Select "Never" for "Save workspace to .RData on exit" Change color scheme to dark (e.g. "Idle Fingers") Put console on right

3.2.2 Create a Local Project

Always work in an RStudio project Projects keep your files (and activity) organized Projects help manage your file path (so your computer can find things) Projects allow for more advanced capabilities later (like GitHub or renv) We cover the use of GitHub in a future webinar For now we create projects locally

3.2.3 Install and Load Packages

As you use R, you will want to make use of the many packages others (and perhaps you) have written Essential packages (or collections): tidyverse, readxl Custom Microsoft office document creation officer, flextable, rvg, openxlsx, extrafont, extrafontdb Sensory specific packages sensR , SensoMineR, FactoMineR, factoextra There are many more, for statistical tests of all varieties, to multivariate analysis, to machine learning, to text analysis, etc.

You only need to install each package once per R version To install a package, you can: Type install.packages("[package name]") Use the RStudio dropdown In addition, if a script loads package that are not installed, RStudio will prompt you to install the package Notes: If you do not have write access on your computer, you might need IT help to install packages You might need to safelist various R related tools and sites

3.2.4 Run Sample Code

Like any language, R is best learned first through example then through study We start with a series of examples to illustrate basic principles For this example, we analyze a series of Tetrad tests

Suppose you have 15 out of 44 correct in a Tetrad test Using sensR, it's easy to analyze these data:

```r
library(sensR)

num_correct <- 15
num_total <- 44

discrim_res <- discrim(num_correct, num_total, method = "tetrad")

print(discrim_res)
```

```
##
## Estimates for the tetrad discrimination protocol with 15 correct
## answers in 44 trials. One-sided p-value and 95 % two-sided confidence
## intervals are based on the 'exact' binomial test.
##
##          Estimate Std. Error  Lower  Upper
## pc       0.34091    0.07146 0.3333 0.4992
## pd       0.01136    0.10719 0.0000 0.2488
## d-prime  0.20363    0.96585 0.0000 1.0193
##
## Result of difference test:
## 'exact' binomial test:  p-value = 0.5141
## Alternative hypothesis: d-prime is greater than 0
```

3.3 Git and GitHub

MAKE COMMENTS BUT DON"T GO THROUGH SET UP. INSTEAD REFER READER TO JENNY BRYANT'S BOOK.

Data Scientific Workflow

Chapter 4

Example Project

4.1 Background

4.2 Other details

4.3 Conclusions?

Chapter 5

Data Preparation

To analyze data, one need *data*. If this data is already available in R, then the analysis can be performed directly. However, in much cases, the data is stored outside the R environment, and needs to be imported.

5.1 Data Importation

In practice, the data might be stored in as many format as one can imagine, whether it ends up being a fairly common solution (.txt file, .csv file, or .xls/.xlsx file), or software specific (e.g. Stata, SPSS, etc.). Since it is very common to store the data in Excel spreadsheets (.xlsx) due to its simplicity, the emphasis is on this solution. Fortunately, most generalities presented for Excel files also apply to other formats through `base::read.table()` for .txt files, `base::read.csv()` and `base::read.csv2()` for .csv files, or through the `{read}` package (which is part of the `{tidyverse}`).

For other (less common) formats, the reader can find packages that would allow importing their files into R. Particular interest can be given to the package `{rio}` (*rio* stands for *R I*nput and *O*utput) which provides an easy solution that 1. can handle a large variety of files, 2. can actually guess the type of file it is, and 3. provides tools to import, export, and convert almost any type of data format, including .csv, .xls and .xlsx, or data from other statistical software such as SAS (.sas7bdat and .xpt), SPSS (.sav and .por), or Stata (.dta). As an alternative, the package `{foreign}` provides functions that allow importing data stored from other statistical software (incl. Minitab, S, SAS, Stata, SPSS, etc.)

Although Excel is most likely one of the most popular way of storing data, there are no `{base}` functions that allow importing such files easily. Fortunately, many packages have been developed in that purpose, including `{XLConnect}`, `{xlsx}`, {gdata}, and {readxl}. Due to its convenience and speed of execution, we will be using {readxl} here.

5.1.1 Importing Structured Excel File

First, let's import the *Sensory Profile.xlsx* workbook using the `readxl::read_xlsx()` file, by informing as parameter the location of the file (informed in `file_path` using the package {here}) and the `sheet` we want to read from.

This file is called *structured* as all the relevant information is already stored in the same sheet in a structured way. In other words, no decoding is required here, and there are no 'unexpected' rows or columns (e.g. empty lines, or lines with additional information regarding the data but that is not data):

- The first row within the *Data* sheet of *Sensory Profile.xlsx* contains the headers,
- From the second row onwards, only data is being stored.

Since this data will be used for some analyses, it is assigned data to an R object called `sensory`.

```
library(here)
```

```
## here() starts at C:/Aigora/books/intro to data science/i2ds4scc_bookdown
```

```
file_path <- here("data","Sensory Profile.xlsx")

library(readxl)
sensory <- read_xlsx(file_path, sheet="Data")
```

To ensure that the importation went well, we print `sensory` to see how it looks like. Since {readxl} has been developed by Hadley Wickham and colleagues, its functions follow the {tidyverse} principles and the dataset thus imported is a `tibble`. Let's take advantage of the printing properties of a `tibble` to evaluate `sensory`:

```
sensory
```

```
## # A tibble: 99 x 35
##    Judge Code  Product Shiny `External color~ `Color evenness` `Qtt of inclusi~
##    <chr> <chr> <chr>   <dbl>            <dbl>            <dbl>            <dbl>
##  1 J01   B     12GP_f   48.6             30               13.2             10.8
```

```
##  2 J01   D     12GP16~  46.2             45.6             37.8              0
##  3 J01   C     12GP23~  48               45.6             17.4              7.8
##  4 J01   G     12SA_f    5.4              6.6             17.4              0
##  5 J01   I     12SA23~   0               42.6             18               21
##  6 J01   E     16PPK_p   0               23.4             49.2              0
##  7 J01   New   23pK_p    4.8             33.6             15.6             32.4
##  8 J01   H     23PLK_p   0               51.6             48.6             23.4
##  9 J01   F     29PPK_p   0               50.4             24               27.6
## 10 J01   A     ck1      52.8             30               22.8              9.6
## # ... with 89 more rows, and 28 more variables: `Surface defects` <dbl>, `Print
## #   quality` <dbl>, Thickness <dbl>, `Color constrast` <dbl>, `Overall odor
## #   intensity` <dbl>, `Fatty odor` <dbl>, `Roasted odor` <dbl>, `Cereal
## #   flavor` <dbl>, `Raw dough flavor` <dbl>, `Fatty flavor` <dbl>, `Dairy
## #   flavor` <dbl>, `Roasted flavor` <dbl>, `Overall flavor persistence` <dbl>,
## #   Salty <dbl>, Sweet <dbl>, Sour <dbl>, Bitter <dbl>, Astringent <dbl>,
## #   Warming <dbl>, `Initial hardness` <dbl>, Brittle <dbl>, Crunchy <dbl>,
## #   `Fatty in mouth` <dbl>, Light <dbl>, `Dry in mouth` <dbl>, `Qtt of
## #   inclusions in mouth` <dbl>, Sticky <dbl>, Melting <dbl>
```

`sensory` is a tibble with 99 rows and 35 columns that includes the `Judge` information (first column, defined as character), the `Product` information (second and third columns, defined as character), and the sensory attributes (fourth column onward, defined as numerical or `dbl`).

Additionally, we can also print a `summary()` of `sensory` to get some extra information regarding the data (such as the minimum, maximum, mean and median for each numerical variable)"

```
summary(sensory)
```

```
##     Judge               Code              Product              Shiny     
##  Length:99          Length:99          Length:99          Min.   : 0.0  
##  Class :character   Class :character   Class :character   1st Qu.: 9.3  
##  Mode  :character   Mode  :character   Mode  :character   Median :21.0  
##                                                           Mean   :23.9  
##                                                           3rd Qu.:38.4  
##                                                           Max.   :54.0  
##  External color intensity Color evenness  Qtt of inclusions Surface defects
##  Min.   : 6.60            Min.   : 6.60   Min.   : 0.00     Min.   : 4.80  
##  1st Qu.:27.00            1st Qu.:19.50   1st Qu.:13.80     1st Qu.:15.30  
##  Median :34.80            Median :26.40   Median :19.80     Median :21.00  
##  Mean   :33.68            Mean   :28.19   Mean   :20.63     Mean   :23.35  
##  3rd Qu.:42.60            3rd Qu.:37.20   3rd Qu.:29.10     3rd Qu.:30.60  
##  Max.   :55.20            Max.   :53.40   Max.   :40.80     Max.   :51.60  
##  Print quality     Thickness     Color constrast Overall odor intensity
```

```
##  Min.   :12.00   Min.   : 7.80   Min.   : 5.40   Min.   : 0.00
##  1st Qu.:36.30   1st Qu.:18.30   1st Qu.:21.00   1st Qu.:10.20
##  Median :40.80   Median :25.80   Median :32.40   Median :18.00
##  Mean   :40.72   Mean   :25.48   Mean   :30.02   Mean   :18.67
##  3rd Qu.:47.10   3rd Qu.:32.10   3rd Qu.:40.20   3rd Qu.:26.10
##  Max.   :60.00   Max.   :52.80   Max.   :51.60   Max.   :40.20
##    Fatty odor     Roasted odor   Cereal flavor   Raw dough flavor
##  Min.   : 0.00   Min.   : 0.00   Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 0.00   1st Qu.: 8.10   1st Qu.:18.00   1st Qu.: 3.00
##  Median : 5.40   Median :15.00   Median :25.20   Median :13.20
##  Mean   : 6.81   Mean   :15.07   Mean   :24.99   Mean   :14.23
##  3rd Qu.:10.50   3rd Qu.:20.70   3rd Qu.:31.20   3rd Qu.:24.60
##  Max.   :27.00   Max.   :42.00   Max.   :48.00   Max.   :43.20
##   Fatty flavor     Dairy flavor    Roasted flavor  Overall flavor persistence
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 9.00   1st Qu.:16.20
##  Median : 6.600   Median : 7.200   Median :17.40   Median :22.80
##  Mean   : 7.419   Mean   : 9.106   Mean   :17.68   Mean   :22.73
##  3rd Qu.:13.200   3rd Qu.:13.500   3rd Qu.:24.60   3rd Qu.:28.80
##  Max.   :24.000   Max.   :46.800   Max.   :51.60   Max.   :43.80
##      Salty            Sweet            Sour            Bitter
##  Min.   : 0.000   Min.   : 0.00   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 9.90   1st Qu.: 0.000   1st Qu.: 0.000
##  Median : 1.200   Median :18.00   Median : 0.000   Median : 7.800
##  Mean   : 5.027   Mean   :17.82   Mean   : 1.461   Mean   : 8.103
##  3rd Qu.:10.050   3rd Qu.:24.30   3rd Qu.: 0.000   3rd Qu.:14.100
##  Max.   :19.200   Max.   :43.20   Max.   :13.800   Max.   :27.600
##    Astringent       Warming      Initial hardness    Brittle
##  Min.   : 0.00   Min.   : 0.00   Min.   : 0.00    Min.   : 0.00
##  1st Qu.: 0.00   1st Qu.: 9.30   1st Qu.:19.50    1st Qu.:27.60
##  Median : 8.40   Median :16.80   Median :30.60    Median :34.80
##  Mean   :11.45   Mean   :16.76   Mean   :30.12    Mean   :31.77
##  3rd Qu.:19.50   3rd Qu.:25.20   3rd Qu.:39.30    3rd Qu.:39.14
##  Max.   :34.20   Max.   :47.40   Max.   :60.00    Max.   :57.00
##     Crunchy      Fatty in mouth      Light        Dry in mouth
##  Min.   : 8.40   Min.   : 0.00   Min.   : 5.40   Min.   :11.40
##  1st Qu.:23.25   1st Qu.: 0.00   1st Qu.:22.80   1st Qu.:39.00
##  Median :30.60   Median : 5.40   Median :31.80   Median :45.00
##  Mean   :29.62   Mean   : 7.32   Mean   :30.21   Mean   :42.99
##  3rd Qu.:36.30   3rd Qu.:12.90   3rd Qu.:37.20   3rd Qu.:49.80
##  Max.   :48.60   Max.   :27.00   Max.   :53.40   Max.   :58.80
##  Qtt of inclusions in mouth     Sticky         Melting
##  Min.   : 0.00              Min.   : 6.00   Min.   : 0.0
##  1st Qu.:15.90              1st Qu.:27.00   1st Qu.:13.2
##  Median :26.40              Median :33.60   Median :19.2
##  Mean   :24.92              Mean   :32.73   Mean   :20.5
```

```
##  3rd Qu.:35.40              3rd Qu.:39.60   3rd Qu.:27.3
##  Max.   :45.60              Max.   :52.80   Max.   :38.4
```

At last, we can list the structure of the dataset through the `str()` function.

```
str(sensory)
```

```
## tibble [99 x 35] (S3: tbl_df/tbl/data.frame)
##  $ Judge                      : chr [1:99] "J01" "J01" "J01" "J01" ...
##  $ Code                       : chr [1:99] "B" "D" "C" "G" ...
##  $ Product                    : chr [1:99] "12GP_f" "12GP16PSL_m" "12GP23P_m" "12SA_f" ...
##  $ Shiny                      : num [1:99] 48.6 46.2 48 5.4 0 0 4.8 0 0 52.8 ...
##  $ External color intensity   : num [1:99] 30 45.6 45.6 6.6 42.6 23.4 33.6 51.6 50.4 30 ...
##  $ Color evenness             : num [1:99] 13.2 37.8 17.4 17.4 18 49.2 15.6 48.6 24 22.8 ...
##  $ Qtt of inclusions          : num [1:99] 10.8 0 7.8 0 21 0 32.4 23.4 27.6 9.6 ...
##  $ Surface defects            : num [1:99] 13.2 48.6 14.4 36 36 12.6 13.8 18 39.6 22.8 ...
##  $ Print quality              : num [1:99] 54 45.6 49.2 42.6 51 47.4 43.8 45.6 53.4 48.6 ...
##  $ Thickness                  : num [1:99] 35.4 43.2 25.8 32.4 31.8 29.4 36 31.2 36 38.4 ...
##  $ Color constrast            : num [1:99] 40.2 45.6 17.4 41.4 41.4 12.6 36 5.4 21 37.8 ...
##  $ Overall odor intensity     : num [1:99] 24.6 7.2 21.6 13.8 26.4 18 16.2 13.8 0 16.8 ...
##  $ Fatty odor                 : num [1:99] 5.4 0 0 0 6.6 8.4 7.8 7.8 0 6.6 ...
##  $ Roasted odor               : num [1:99] 20.4 6 18.6 16.2 16.8 16.2 16.2 12.6 7.2 15.6 ...
##  $ Cereal flavor              : num [1:99] 25.8 16.2 30 18 28.8 21.6 23.4 23.4 28.8 24.6 ...
##  $ Raw dough flavor           : num [1:99] 28.8 28.2 26.4 21 26.4 27.6 31.2 27 27.6 28.2 ...
##  $ Fatty flavor               : num [1:99] 7.2 0 0 6 7.2 6.6 10.8 7.2 0 13.8 ...
##  $ Dairy flavor               : num [1:99] 0 0 0 0 0 0 4.8 0 0 0 ...
##  $ Roasted flavor             : num [1:99] 19.2 28.2 27 21.6 25.8 20.4 26.4 24.6 22.2 24.6 ...
##  $ Overall flavor persistence: num [1:99] 24.6 14.4 25.2 18 22.8 21 24 21.6 25.2 23.4 ...
##  $ Salty                      : num [1:99] 0 0 0 0 0 0 3.6 0 0 0 ...
##  $ Sweet                      : num [1:99] 19.2 11.4 9.6 10.8 21 20.4 21 10.2 21 13.8 ...
##  $ Sour                       : num [1:99] 0 0 0 0 0 0 0.6 0 0 0 ...
##  $ Bitter                     : num [1:99] 21.6 9 21 0 22.8 8.4 27.6 9.6 18.6 19.2 ...
##  $ Astringent                 : num [1:99] 0 18.6 25.8 21 26.4 16.2 31.8 23.4 25.2 0 ...
##  $ Warming                    : num [1:99] 0 0 13.8 10.8 15 0 27.6 19.8 14.4 0 ...
##  $ Initial hardness           : num [1:99] 17.4 19.8 33 10.2 29.4 16.2 18 34.2 28.2 11.4 ...
##  $ Brittle                    : num [1:99] 35.4 33.6 27.6 29.4 34.8 38.4 35.4 35.4 33 39.6 ...
##  $ Crunchy                    : num [1:99] 32.4 28.8 25.2 27 32.4 35.4 34.8 30.6 27.6 25.8 ...
##  $ Fatty in mouth             : num [1:99] 9.6 9.6 6 5.4 12 7.2 14.4 5.4 0 0 ...
##  $ Light                      : num [1:99] 21 40.5 20.4 25.8 21 ...
##  $ Dry in mouth               : num [1:99] 25.8 28.2 31.2 22.2 27.6 34.2 31.8 37.8 33.6 27 ...
##  $ Qtt of inclusions in mouth: num [1:99] 22.2 13.2 10.2 9 29.4 18 31.2 25.8 26.4 27.6 ...
##  $ Sticky                     : num [1:99] 35.4 21.6 37.2 22.8 37.2 39 34.2 36.6 36 37.2 ...
##  $ Melting                    : num [1:99] 36 34.2 8.4 34.8 19.8 34.8 36.6 19.2 21.6 33.6 ...
```

This function provides an overview of the first element of each variables (and the format they are in) as a list, and allows the user to have a first scan to eventually detect *errors* or *mishaps* during the importation.

Here, the data has been imported as expected.

5.1.2 Importing Unstructured Excel File

In some cases, the dataset is not so well organized/structured, and may need to be *decoded*. This is the case for the workbook entitled *TFEQ.xlsx*. For this file:

- The variables' name have been coded and their corresponding names (together with some other valuable information we will be using in the next chapter) are stored in a different sheet entitled *Variables*;
- The different levels of each variable (including their code and corresponding names) are stored in another sheet entitled *Levels*.

To import and decode this dataset, multiple steps are required:

- Import the variables' name only;
- Import the information regarding the levels;
- Import the dataset without the first line of header, but by providing the correct names obtained in the first step;
- Decode each question (when needed) by replacing the numerical code by their corresponding labels.

Let's start with importing the variables' names from *TFEQ.xlsx* (sheet *Variables*)

```
file_path <- here("data","TFEQ.xlsx")

var_names <- read_xlsx(file_path, sheet="Variables")
var_names
```

```
## # A tibble: 62 x 6
##    Code  `Original Questions (French)`  Name     Direction Value `Full Question`
##    <chr> <chr>                          <chr>    <chr>     <dbl> <chr>
##  1 Q1    Où habitez-vous?               Living ~ <NA>         NA <NA>
##  2 Q2    Comment habitez-vous?          Housing  <NA>         NA <NA>
##  3 Q3    N° Juge                        Judge    <NA>         NA <NA>
##  4 Q4    Quelle est votre taille (en m~ Height   <NA>         NA <NA>
##  5 Q5    Quel est votre poids?          Weight   <NA>         NA <NA>
##  6 Q6    IMC                            BMI      <NA>         NA <NA>
##  7 Q7    Quelle est votre situation ma~ Marital~ <NA>         NA <NA>
##  8 Q8    Nombre de personnes vivant au~ Househo~ <NA>         NA <NA>
```

```
##  9 Q9    Dans quelle tranche de revenu~ Income ~ <NA>         NA <NA>
## 10 Q10   Dans quelle catégorie sociopr~ Occupat~ <NA>         NA <NA>
## # ... with 52 more rows
```

In a similar way, let's import the information related to the levels of each variable, stored in the *Levels* sheet. A deeper look at the *Levels* sheet shows that only the coded names of the variables are available. In order to include the final names, `var_names` is joined (using `inner_join`).

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
var_labels <- read_xlsx(file_path, sheet="Levels") %>%
  inner_join(dplyr::select(var_names, Code, Name), by=c(Question="Code"))

var_labels
```

```
## # A tibble: 172 x 5
##    Question  Code `Original Levels (French)` Levels            Name
##    <chr>    <dbl> <chr>                      <chr>             <chr>
##  1 Q1           1 Zone urbaine               Urban Area        Living area
##  2 Q1           2 Zone rurbaine              Rurban Area       Living area
##  3 Q1           3 Zone rurale                Rural Area        Living area
##  4 Q2           1 Appartement                Apartment         Housing
##  5 Q2           2 Maison individuelle        House             Housing
##  6 Q7           1 Divorcé(e)                 Divorced          Marital status
##  7 Q7           2 Marié(e)                   Married           Marital status
##  8 Q7           3 Vivant maritalement        Conjugal          Marital status
##  9 Q7           4 Célibataire                Single            Marital status
## 10 Q7           5 Pacsé(e)                   Civil Partnership Marital status
## # ... with 162 more rows
```

Note: In some cases, this information is directly available in the dataset as sub-header: A solution is then to import the first rows of the dataset that contain this information using the parameter `n_max` from 'readxl::read_xlsx". For each variable (when information is available), store that information as a list of tables that contains the code and their corresponding label.

THIS SECTION BELOW MIGHT NEED TO GET PASSED ON TO THE EXERCISE

Since most likely this system of coding follow a fixed pattern, we strongly recommend the use of `{tidytext}` and its function `unnest_tokens()`. For example, let's imagine that the our information is structured as *code1=label1,code2=label2,...* (e.g. *0=No,1=Yes*). In such case, first use `unnest_tokens()` to split this string by ','. This creates a tibble with as many rows as there are *code=label* and one column. Next, split this column into two columns using `separate()` and `sep="="`.

(PREVIOUS PART) TO BE GIVEN AS AN EXAMPLE/EXERCISE

Finally, we import the dataset (*Data*) by substituting the coded names with their real names. To do so, we skip reading the first line (`skip=1`) that contains the coded names, and force the column names to be defined by `Name` from `var_names` (after ensuring that the names' sequence perfectly match across the two tables!).

```
TFEQ_data <- read_xlsx(file_path, sheet="Data", col_names=var_names$Name, skip=1)
summary(TFEQ_data)
```

```
##   Living area       Housing          Judge                Height     
##  Min.   :1.000   Min.   :1.000   Length:107         Min.   :1.450  
##  1st Qu.:1.000   1st Qu.:1.000   Class :character   1st Qu.:1.600  
##  Median :1.000   Median :2.000   Mode  :character   Median :1.630  
##  Mean   :1.542   Mean   :1.682                      Mean   :1.634  
##  3rd Qu.:2.000   3rd Qu.:2.000                      3rd Qu.:1.680  
##  Max.   :3.000   Max.   :2.000                      Max.   :1.800  
##      Weight           BMI        Marital status  Household size 
##  Min.   :43.00   Min.   :17.63   Min.   :1.000   Min.   :0.000  
##  1st Qu.:53.50   1st Qu.:20.30   1st Qu.:2.000   1st Qu.:2.000  
##  Median :58.00   Median :21.63   Median :3.000   Median :3.000  
##  Mean   :58.25   Mean   :21.79   Mean   :2.944   Mean   :2.944  
##  3rd Qu.:62.35   3rd Qu.:23.33   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :82.00   Max.   :26.47   Max.   :7.000   Max.   :6.000  
##   Income range     Occupation     Highest degree       D1       
##  Min.   :1.000   Min.   : 1.000   Min.   :1.00   Min.   :0.000  
##  1st Qu.:2.000   1st Qu.: 2.000   1st Qu.:2.00   1st Qu.:0.000  
##  Median :2.000   Median : 6.000   Median :3.00   Median :0.000  
##  Mean   :2.421   Mean   : 6.486   Mean   :2.71   Mean   :0.215  
##  3rd Qu.:3.000   3rd Qu.: 9.000   3rd Qu.:4.00   3rd Qu.:0.000  
##  Max.   :4.000   Max.   :17.000   Max.   :6.00   Max.   :1.000  
```

```
##       D2               H1              R1               H2
##  Min.   :0.0000   Min.   :0.000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :1.0000   Median :0.000   Median :0.0000   Median :1.0000
##  Mean   :0.5794   Mean   :0.243   Mean   :0.4579   Mean   :0.5514
##  3rd Qu.:1.0000   3rd Qu.:0.000   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.000   Max.   :1.0000   Max.   :1.0000
##       R2               D3               H3               D4
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.5000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.0000   Median :1.0000   Median :0.0000   Median :1.0000
##  Mean   :0.2991   Mean   :0.7477   Mean   :0.3738   Mean   :0.5607
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       R3               D5               H4              D6
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:0.0000
##  Median :1.0000   Median :0.0000   Median :0.000   Median :0.0000
##  Mean   :0.5327   Mean   :0.3551   Mean   :0.486   Mean   :0.3458
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.000   Max.   :1.0000
##       R4               D7               D8               H5
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.0000   Median :0.0000   Median :1.0000   Median :1.0000
##  Mean   :0.4486   Mean   :0.3178   Mean   :0.5047   Mean   :0.6168
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       R5               H6               D9               R6
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.0000   Median :1.0000   Median :1.0000   Median :1.0000
##  Mean   :0.4486   Mean   :0.5888   Mean   :0.5701   Mean   :0.5421
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       H7               R7               H8              D10
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.0000   Median :0.0000   Median :0.0000   Median :0.0000
##  Mean   :0.3271   Mean   :0.2523   Mean   :0.2243   Mean   :0.4206
##  3rd Qu.:1.0000   3rd Qu.:0.5000   3rd Qu.:0.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       H9              D11              R8              H10
##  Min.   :0.0000   Min.   :0.000   Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:0.000
##  Median :0.0000   Median :0.000   Median :0.0000   Median :0.000
```

```
##  Mean   :0.1963   Mean   :0.486   Mean   :0.1963   Mean   :0.243
##  3rd Qu.:0.0000   3rd Qu.:1.000   3rd Qu.:0.0000   3rd Qu.:0.000
##  Max.   :1.0000   Max.   :1.000   Max.   :1.0000   Max.   :1.000
##        R9              D12              R10              R11
##  Min.   :0.000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :1.000   Median :0.0000   Median :0.0000   Median :0.0000
##  Mean   :0.514   Mean   :0.2617   Mean   :0.1028   Mean   :0.4206
##  3rd Qu.:1.000   3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:1.0000
##  Max.   :1.000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       H11              R12              D13              R13
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :1.000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:1.000
##  Median :0.0000   Median :1.0000   Median :0.0000   Median :1.000
##  Mean   :0.1589   Mean   :0.7103   Mean   :0.1028   Mean   :1.533
##  3rd Qu.:0.0000   3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:2.000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :4.000
##       R14             H12             R15             H13
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:1.500   1st Qu.:2.000   1st Qu.:1.000
##  Median :3.000   Median :2.000   Median :2.000   Median :2.000
##  Mean   :2.589   Mean   :1.935   Mean   :2.121   Mean   :2.084
##  3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.:3.000
##  Max.   :4.000   Max.   :4.000   Max.   :4.000   Max.   :4.000
##       R16             R17             R18             D14
##  Min.   :2.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:3.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000
##  Median :3.000   Median :2.000   Median :2.000   Median :2.000
##  Mean   :3.252   Mean   :2.439   Mean   :1.869   Mean   :1.692
##  3rd Qu.:4.000   3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.:2.000
##  Max.   :4.000   Max.   :4.000   Max.   :4.000   Max.   :3.000
##       R19             H14             R20             D15
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000
##  Median :1.000   Median :3.000   Median :2.000   Median :2.000
##  Mean   :1.542   Mean   :2.579   Mean   :1.841   Mean   :1.991
##  3rd Qu.:2.000   3rd Qu.:4.000   3rd Qu.:2.000   3rd Qu.:3.000
##  Max.   :4.000   Max.   :4.000   Max.   :4.000   Max.   :4.000
##       R21             D16
##  Min.   :0.000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:1.000
##  Median :2.000   Median :2.000
##  Mean   :2.215   Mean   :1.785
##  3rd Qu.:3.000   3rd Qu.:2.000
##  Max.   :5.000   Max.   :4.000
```

The data has now the proper header, however each variable is still coded numerically.

5.2 Data Inspection

5.2.1 Data Type

In R, the variables can be of different types, going from numerical to nominal to binary etc. This section aims in presenting the most common types (and their properties) used in sensory and consumer studies, and in showing how to transform a variable from one type to another.

Remember that when your dataset is a tibble (as is the case here), the type of each variable is provided as sub-header when printed on screen. This eases the work of the analyst as the variables' type can be assessed at any moment.

In case the dataset is not in a tibble, the use of the `str()` function used previously becomes handy as it provides this information.

In sensory and consumer research, the four most common types are:

- Numerical (incl. integer or `int`, decimal or `dcl`, and double or `dbl`);
- Logical or `lgl`;
- Character or `char`;
- Factor or `fct`.

R still has plenty of other types, for more information please visit: https://tibble.tidyverse.org/articles/types.html

5.2.1.1 Numerical Data

Since a large proportion of the research done is quantitative, it is no surprise that our dataset are often dominated with numerical variables. In practice, numerical data includes integer (non-fractional number, e.g. 1, 2, -16, etc.), or decimal value (or double, e.g. 1.6, 2.333333, -3.2 etc.). By default, when reading data from an external file, R converts any numerical variables to integer unless decimal points are detected, in which case it is converted into double.

Do we want to show how to format R wrt the number of decimals? (e.g. options(digits=2))

5.2.1.2 Binary Data

Another common type that seem to be numerical in appearance, but that has additional properties is the binary type. Binary data is data that takes two possible values (`TRUE` or `FALSE`), and are often the results of a *test* (e.g. is `x>3`? Or is `MyVar` numerical?). A typical example of binary data in sensory and consumer research is data collected through Check-All-That-Apply (CATA) questionnaires.

Note: Intrinsically, binary data is *numerical*, TRUE being assimilated to 1, FALSE to 0. If multiple tests are being performed, it is possible to sum the number of tests that pass using the `sum()` function, as shown in the simple example below:

```
set.seed(123456)
# Generating 10 random values between 1 and 10 using the uniform distribution
x <- runif(10, 1, 10)
x
```

```
##  [1] 8.180059 7.782086 4.521301 4.074010 4.251647 2.785103 5.813722 1.868736
##  [9] 9.890622 2.508125
```

```
# Test whether the values generated are strictly larger than 5
test <- x>5
test
```

```
##  [1]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
```

```
# Counting the number of values strictly larger than 5
sum(test)
```

```
## [1] 4
```

5.2.1.3 Nominal Data

Nominal data is any data that is not numerical. In most cases, nominal data are defined through text, or strings. It can appear in some situations that nominal variables are still defined with numbers although they do not have a numerical meaning. This is for instance the case when the respondents or samples are identified through numerical codes: In that case, it is clear that respondent 2 is not twice larger than respondent 1 for instance. But since the software cannot guess that those numbers are *identifiers* rather than *numbers*, the variables should be declared as nominal. The procedure explaining how to convert the type of the variables will be explained in the next section.

For nominal data, two particular types of data are of interest:

- Character or `char`;

- Factor or `fct`.

Variables defined as character or factor take strings as input. However, these two types differ in terms of structure of their levels:

- For `character`, there are no particular structure, and the variables can take any values (e.g. open-ended question);
- For `factor`, the inputs of the variables are structured into `levels`.

To evaluate the number of levels, different procedure are required:

- For `character`, one should count the number of unique element using `length()` and `unique()`;
- For `factor`, the levels and the number of levels are direcly provided by `levels()` and `nlevels()`.

Let's compare a variable set as `factor` and `character` by using the `Judge` column from `TFEQ_data`:

```
example <- TFEQ_data %>%
  dplyr::select(Judge) %>%
  mutate(Judge_fct = as.factor(Judge))

print("Summary:")
```

```
## [1] "Summary:"
```

```
summary(example)
```

```
##     Judge              Judge_fct
##  Length:107         J1     :  1
##  Class :character   J10    :  1
##  Mode  :character   J100   :  1
##                     J101   :  1
##                     J103   :  1
##                     J105   :  1
##                     (Other):101
```

```
print("As Character:")
```

```
## [1] "As Character:"
```

```
unique(example$Judge)
```

```
##   [1] "J48"  "J61"  "J60"  "J97"  "J38"  "J26"  "J103" "J91"  "J13"  "J73"
##  [11] "J49"  "J62"  "J14"  "J98"  "J15"  "J39"  "J74"  "J64"  "J99"  "J75"
##  [21] "J108" "J76"  "J1"   "J65"  "J63"  "J2"   "J24"  "J27"  "J3"   "J50"
##  [31] "J4"   "J77"  "J66"  "J5"   "J67"  "J6"   "J100" "J90"  "J92"  "J7"
##  [41] "J79"  "J68"  "J69"  "J85"  "J101" "J70"  "J51"  "J52"  "J109" "J8"
##  [51] "J93"  "J53"  "J54"  "J110" "J94"  "J111" "J86"  "J16"  "J112" "J29"
##  [61] "J95"  "J96"  "J118" "J17"  "J117" "J55"  "J30"  "J40"  "J41"  "J9"
##  [71] "J31"  "J10"  "J56"  "J87"  "J71"  "J42"  "J43"  "J32"  "J81"  "J58"
##  [81] "J19"  "J33"  "J34"  "J44"  "J72"  "J113" "J45"  "J105" "J114" "J46"
##  [91] "J20"  "J82"  "J115" "J59"  "J116" "J21"  "J88"  "J83"  "J22"  "J11"
## [101] "J35"  "J89"  "J120" "J12"  "J36"  "J23"  "J119"
```

```
length(unique(example$Judge))
```

```
## [1] 107
```

```
print("As Factor:")
```

```
## [1] "As Factor:"
```

```
levels(example$Judge_fct)
```

```
##   [1] "J1"   "J10"  "J100" "J101" "J103" "J105" "J108" "J109" "J11"  "J110"
##  [11] "J111" "J112" "J113" "J114" "J115" "J116" "J117" "J118" "J119" "J12"
##  [21] "J120" "J13"  "J14"  "J15"  "J16"  "J17"  "J19"  "J2"   "J20"  "J21"
##  [31] "J22"  "J23"  "J24"  "J26"  "J27"  "J29"  "J3"   "J30"  "J31"  "J32"
##  [41] "J33"  "J34"  "J35"  "J36"  "J38"  "J39"  "J4"   "J40"  "J41"  "J42"
##  [51] "J43"  "J44"  "J45"  "J46"  "J48"  "J49"  "J5"   "J50"  "J51"  "J52"
##  [61] "J53"  "J54"  "J55"  "J56"  "J58"  "J59"  "J6"   "J60"  "J61"  "J62"
##  [71] "J63"  "J64"  "J65"  "J66"  "J67"  "J68"  "J69"  "J7"   "J70"  "J71"
##  [81] "J72"  "J73"  "J74"  "J75"  "J76"  "J77"  "J79"  "J8"   "J81"  "J82"
##  [91] "J83"  "J85"  "J86"  "J87"  "J88"  "J89"  "J9"   "J90"  "J91"  "J92"
## [101] "J93"  "J94"  "J95"  "J96"  "J97"  "J98"  "J99"
```

```
nlevels(example$Judge_fct)
```

```
## [1] 107
```

Although `Judge` and `Judge_fct` look the same, they are structurally different, and those differences play an important role that one should consider when running certain analyses, or building tables and graphs.

When set as `character`, the number of levels of a variable is directly read from the data, and its levels' order would either match the way they appear in the data, or are ordered alphabetically. This means that any data collected using a structured scale will lose its natural order. When set as `factor`, the number and order of the factor levels are informed, and does not depend on the data itself: If a level has never been selected, or if certain groups have been filtered, this information is still present in the data.

To illustrate this, let's re-arrange the levels from `Judge_fct` by ordering them numerically in such a way `J2` follows `J1` rather than `J10`.

```
judge <- str_sort(levels(example$Judge_fct),numeric=TRUE)
judge
```

```
##   [1] "J1"   "J2"   "J3"   "J4"   "J5"   "J6"   "J7"   "J8"   "J9"   "J10"
##  [11] "J11"  "J12"  "J13"  "J14"  "J15"  "J16"  "J17"  "J19"  "J20"  "J21"
##  [21] "J22"  "J23"  "J24"  "J26"  "J27"  "J29"  "J30"  "J31"  "J32"  "J33"
##  [31] "J34"  "J35"  "J36"  "J38"  "J39"  "J40"  "J41"  "J42"  "J43"  "J44"
##  [41] "J45"  "J46"  "J48"  "J49"  "J50"  "J51"  "J52"  "J53"  "J54"  "J55"
##  [51] "J56"  "J58"  "J59"  "J60"  "J61"  "J62"  "J63"  "J64"  "J65"  "J66"
##  [61] "J67"  "J68"  "J69"  "J70"  "J71"  "J72"  "J73"  "J74"  "J75"  "J76"
##  [71] "J77"  "J79"  "J81"  "J82"  "J83"  "J85"  "J86"  "J87"  "J88"  "J89"
##  [81] "J90"  "J91"  "J92"  "J93"  "J94"  "J95"  "J96"  "J97"  "J98"  "J99"
##  [91] "J100" "J101" "J103" "J105" "J108" "J109" "J110" "J111" "J112" "J113"
## [101] "J114" "J115" "J116" "J117" "J118" "J119" "J120"
```

```
levels(example$Judge_fct) <- judge
```

Now the levels are sorted, let's 'remove' some respondents by only keeping the 20 first ones (J1 to J20, as J18 does not exist), and re-run the previous code:

```
example <- TFEQ_data %>%
  dplyr::select(Judge) %>%
  mutate(Judge_fct = as.factor(Judge)) %>%
  filter(Judge %in% paste0("J",1:20))

dim(example)
```

```
## [1] 19  2
```

```
print("As Character:")
```

```
## [1] "As Character:"
```

```
unique(example$Judge)
```

```
##  [1] "J13" "J14" "J15" "J1"  "J2"  "J3"  "J4"  "J5"  "J6"  "J7"  "J8"  "J16"
## [13] "J17" "J9"  "J10" "J19" "J20" "J11" "J12"
```

```
length(unique(example$Judge))
```

```
## [1] 19
```

```
print("As Factor:")
```

```
## [1] "As Factor:"
```

```
levels(example$Judge_fct)
```

```
##   [1] "J1"   "J10"  "J100" "J101" "J103" "J105" "J108" "J109" "J11"  "J110"
##  [11] "J111" "J112" "J113" "J114" "J115" "J116" "J117" "J118" "J119" "J12"
##  [21] "J120" "J13"  "J14"  "J15"  "J16"  "J17"  "J19"  "J2"   "J20"  "J21"
##  [31] "J22"  "J23"  "J24"  "J26"  "J27"  "J29"  "J3"   "J30"  "J31"  "J32"
##  [41] "J33"  "J34"  "J35"  "J36"  "J38"  "J39"  "J4"   "J40"  "J41"  "J42"
##  [51] "J43"  "J44"  "J45"  "J46"  "J48"  "J49"  "J5"   "J50"  "J51"  "J52"
##  [61] "J53"  "J54"  "J55"  "J56"  "J58"  "J59"  "J6"   "J60"  "J61"  "J62"
##  [71] "J63"  "J64"  "J65"  "J66"  "J67"  "J68"  "J69"  "J7"   "J70"  "J71"
##  [81] "J72"  "J73"  "J74"  "J75"  "J76"  "J77"  "J79"  "J8"   "J81"  "J82"
##  [91] "J83"  "J85"  "J86"  "J87"  "J88"  "J89"  "J9"   "J90"  "J91"  "J92"
## [101] "J93"  "J94"  "J95"  "J96"  "J97"  "J98"  "J99"
```

```
nlevels(example$Judge_fct)
```

```
## [1] 107
```

After filtering some respondents, it can be noticed that the variable set as character only contains 19 elements, whereas the column set as factor still contains the 107 respondents (most of them not having any recordings). This property can be seen as an advantage or a disadvantage depending on the situation:

- For frequencies, it may be relevant to remember all the options, including the ones that may never be selected, and to order the results logically (use of `factor`).
- For hypothesis testing (e.g. ANOVA) on subset of data (e.g. the data being split by gender), the `Judge` variable set as `character` would have the correct number of degrees of freedom (18 in our example) whereas the variable set as factor would use 106 degrees of freedom in all cases!

The latter point is particularly critical since the analysis is incorrect and will either return an error or worse return erroneous results!

Last but not least, variables defined as factor allow having their levels being renamed (and eventually combined) very easily. Let's consider the `Living area` variable from `TFEQ_data` as example. From the original excel file, it can be seen that it has three levels, `1` corresponding to *urban area*, `2` to *rurban area*, and `3` to *rural area*. Let's start by renaming this variable accordingly:

```
example = TFEQ_data %>%
  mutate(Area = factor(`Living area`, levels=c(1,2,3), labels=c("Urban", "Rurban", "Rural")))

levels(example$Area)
```

```
## [1] "Urban"  "Rurban" "Rural"
```

```
nlevels(example$Area)
```

```
## [1] 3
```

```
table(example$`Living area`, example$Area)
```

```
##
##     Urban Rurban Rural
##   1    72      0     0
##   2     0     12     0
##   3     0      0    23
```

As can be seen, the variable `Area` is the factor version (including its labels) of `Living area`. If we would also consider that `Rurban` should be combined with `Rural`, and that `Rural` should appear before `Urban`, we can simply modify the code as such:

```
example = TFEQ_data %>%
  mutate(Area = factor(`Living area`, levels=c(2,3,1), labels=c("Rural", "Rural", "Urban")))

levels(example$Area)
```

```
## [1] "Rural" "Urban"
```

```
nlevels(example$Area)
```

```
## [1] 2
```

```
table(example$`Living area`, example$Area)
```

```
##
##      Rural Urban
##   1      0    72
##   2     12     0
##   3     23     0
```

This approach of renaming and re-ordering factor levels is very important as it simplifies the readability of tables and figures. Some other transformations can be applied to factors thanks to the `{forcats}` package. Amongst other relevant functions, particular attention can be given to:

- `fct_reorder`/`fct_reorder2` and `fct_relevel` which reorder the levels of a factor;
- `fct_recode` which helps recoding a factor (as an alternative to `factor` used in the previous example);
- `fct_collapse` and `fct_lump` which allows aggregating different levels together (`fct_lump` regroups automatically all the rare levels).

Although it hasn't been done here, manipulating strings is also possible. To do so, the `{stringr}` package provides a lot of of interesting functions, such as:

- `str_to_upper`/`str_to_lower` which automatically convert strings to uppercase or lowercase;
- `str_c`, `str_sub` which combine or subset strings;
- `str_trim` and `str_squish` which help remove white spaces;
- `str_extract`, `str_replace`, `str_split` to extract, replace, or split strings or part of the strings.

5.2.2 Changing the type of a variable

Transforming the type of variables using `mutate()`:

- from logical to numerical;
- from numerical to character/factor;
- from character/factor to numerical

5.3 Data Organization

Presentation of the different shapes of the tables based on objectives

5.4 Data Manipulation

5.4.1 Type of table

matrix, data frame, and tibble.

how to check the type? `class()` how to test it? `is.data.frame()`, `is.matrix()`, `is_tibble()` how to convert it to another format? (see below)

Note on `{FactoMineR}` and `{SensoMineR}` which require data frames or matrix (not tibble) so introduction to `column_to_rownames()` and `rownames_to_columns()` as well as `as.data.frame()` and `as_tibble()`.

5.4.2 Data organisation

`select()`, `filter()`, `arrange()` `mutate()` and `transmute()`

5.4.3 Data re-structuration

`pivot_wider()` and `pivot_longer()` `full_join()`, `inner_join()`, `left_join()` and `right_join()`

`unnest_token()` from **{tidytext}**

5.5 Cleaning and Quality Assessment

5.5.1 Renaming

renaming columns using `rename()` or `select()` renaming elements using `factor()` and **{forcats}**

5.5.2 Recoding

Example of recoding (could be renaming, or replacing NAs, etc.) by combining `mutate()` and `ifelse()`

5.5.3 Handling Missing Values

Removing and replacing NAs

5.5.4 Quality Assessment

Graphics?

Chapter 6

Data Analysis

6.1 Transformation

6.2 Exploration

6.3 Modeling

Chapter 7

Data Visualization

7.1 Principles

7.2 Table Mechanics

7.3 Chart Mechanics

7.4 Examples

Chapter 8

Insight Delivery

8.1 Design principles

8.2 Scientific inquiry vs storytelling

8.3 Research reformulation

8.4 Interactive reporting

Reproducible Research

Chapter 9

Tools for Collaboration

9.1 Principles

9.2 Tools

9.2.1 GitHub

9.2.2 R scripts

9.2.3 RMarkdown

9.2.4 Shiny

9.3 Documentation

9.4 Version control

9.5 Online repositories for team collaboration

9.6 Building a code base

9.6.1 Internal functions

9.6.2 Packages

Chapter 10

Automated Reporting

10.1 Excel

10.2 Word

10.3 PowerPoint

10.3.1 Charts

10.3.2 Tables

10.3.3 Bullet Points

10.3.4 Images

10.4 HTML

Additional Topics

Chapter 11

Machine Learning

11.1 Concepts and general workflow (training/test)

11.2 Unsupervised learning

11.2.1 Cluster analysis

11.2.2 Factor analysis

11.2.3 Principle components analysis

11.2.4 t-SNE

11.3 Semisupervised learning

11.3.1 PLS regression

11.4 Supervised learning

11.4.1 Regression

11.4.2 K-nearest neighbors

11.4.3 Decision trees

11.4.4 Black boxes

11.4.4.1 Random forests

11.4.4.2 SVMs

11.4.4.3 Neural networks

11.5 Predictive modeling

Chapter 12

Text Analysis

12.1 Data import

12.1.1 Data sources

12.1.2 Tokenizing

12.1.3 Lemmatization, stemming, and stop word removal

12.2 Analysis

12.2.1 Frequency counts and summary statistics

12.2.2 Word clouds

12.2.3 Contrast plots

12.2.4 Sentiment analysis

12.2.5 Bigrams and word graphs

Chapter 13

Graph Databases

Conclusion

Chapter 14

Conclusion

Appendices

Bryan, J. (2018). Happy git and github for the useR. GitHub. Available from https://happygitwithr.com/ Gillespie, C., & Lovelace, R. (2016). Efficient R programming: A practical guide to smarter programming. " O'Reilly Media, Inc.". Grolemund, G. Getting Started with R. Rstudio Support. Available from https://support.rstudio.com/hc/en-us/articles/201141096-Getting-Started-with-R Norman, D. (2013). The design of everyday things: Revised and expanded edition. Basic books. Peng, R., Caffo, B., & Leek, J. Data Science Specialization [Online course]. Available from https://www.coursera.org/specializations/jhu-data-science Peng, R., Caffo, B., & Leek, J. Executive data science specialization [Online course]. Available from https://www.coursera.org/specializations/executive-data-science Wickham, H., & Grolemund, G. (2016). R for data science: import, tidy, transform, visualize, and model data." O'Reilly Media, Inc.". Wickham, H. (2016). ggplot2: elegant graphics for data analysis. Springer. Zuur, A., Ieno, E. N., & Meesters, E. (2009). A Beginner's Guide to R. Springer Science & Business Media.

Breiman, L. (2001). Statistical modeling: The two cultures (with comments and a rejoinder by the author). Statistical science, 16(3), 199-231. Cleveland, W. S. (2001). Data science: an action plan for expanding the technical areas of the field of statistics. International statistical review, 69(1), 21-26. Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. AI magazine, 17(3), 37-37. Naur, P. (1966). The science of datalogy. Communications of the ACM, 9(7), 485. Tukey, J. W. (1962). The future of data analysis. The annals of mathematical statistics, 33(1), 1-67. Tukey, J. W. (1977). Exploratory data analysis. Reading, Mass: Addison-Wesley Pub. Co. Wu, C. F. J. (1997) "Statistics = Data Science?" Lecture notes available online at http://www2.isye.gatech.edu/~jeffwu/presentations/datascience.pdf