Exercises with `dplyr` and `tidyr`

September 3, 2017

Quantide
[1]

[1] mailto:

Contents

Chapter 1

Introduction

In this document you will find some exercises with the `tidyverse` R packages. They are mainly based on the `nycflights13` data, taken from the `nycflights13` package.

1.1 Introduction to `nycflights13` data

The `nycflights13` package contains information about all flights that departed from NYC (e.g. EWR, JFK and LGA) in 2013: 336,776 flights in total.

```
require(nycflights13)
ls(pos = "package:nycflights13")
```

```
## [1] "airlines" "airports" "flights"  "planes"   "weather"
```

To help understand what causes delays, it includes a number of useful datasets:

- `flights`: information about all flights that departed from NYC
- `weather`: hourly meterological data for each airport;
- `planes`: construction information about each plane;
- `airports`: airport names and locations;
- `airlines`: translation between two letter carrier codes and names.

1.1.1 `flights`

This dataset contains on-time data for all flights that departed from NYC (i.e. JFK, LGA or EWR) in 2013. The data frame has 16 variables and 336776 observations. The variables are organised as follow:

- Date of departure: `year`, `month`, `day`;
- Departure and arrival times (local tz): `dep_time`, `arr_time`;
- Departure and arrival delays, in minutes: `dep_delay`, `arr_delay` (negative times represent early departures/arrivals);
- Time of departure broken in to hour and minutes: `hour`, `minute`;
- Two letter carrier abbreviation: `carrier`;
- Plane tail number: `tailnum`;
- Flight number: `flight`;
- Origin and destination: `origin`, `dest`;
- Amount of time spent in the air: `air_time`;
- Distance flown: `distance`.

```
dim(flights)
```

```
## [1] 336776     19
```

```
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
## 6  2013     1     1      554            558        -4      740
## # ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

```
str(flights)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    336776 obs. of  19 variables:
##  $ year          : int  2013 2013 2013 2013 2013 2013 2013 2013 2013 2013 ...
##  $ month         : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ day           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ dep_time      : int  517 533 542 544 554 554 555 557 557 558 ...
##  $ sched_dep_time: int  515 529 540 545 600 558 600 600 600 600 ...
```

```
##  $ dep_delay     : num  2 4 2 -1 -6 -4 -5 -3 -3 -2 ...
##  $ arr_time      : int  830 850 923 1004 812 740 913 709 838 753 ...
##  $ sched_arr_time: int  819 830 850 1022 837 728 854 723 846 745 ...
##  $ arr_delay     : num  11 20 33 -18 -25 12 19 -14 -8 8 ...
##  $ carrier       : chr  "UA" "UA" "AA" "B6" ...
##  $ flight        : int  1545 1714 1141 725 461 1696 507 5708 79 301 ...
##  $ tailnum       : chr  "N14228" "N24211" "N619AA" "N804JB" ...
##  $ origin        : chr  "EWR" "LGA" "JFK" "JFK" ...
##  $ dest          : chr  "IAH" "IAH" "MIA" "BQN" ...
##  $ air_time      : num  227 227 160 183 116 150 158 53 140 138 ...
##  $ distance      : num  1400 1416 1089 1576 762 ...
##  $ hour          : num  5 5 5 5 6 5 6 6 6 6 ...
##  $ minute        : num  15 29 40 45 0 58 0 0 0 0 ...
##  $ time_hour     : POSIXct, format: "2013-01-01 05:00:00" "2013-01-01 05:00:00" ...
```

1.1.2 airlines

This dataset contains airlines names and their respective carrier codes, it has 2 variables and 16 observations. Data structure shows that both variables involved are categorical.

```
dim(airlines)
```

```
## [1] 16  2
```

```
head(airlines)
```

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

```
str(airlines)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    16 obs. of  2 variables:
##  $ carrier: chr  "9E" "AA" "AS" "B6" ...
##  $ name   : chr  "Endeavor Air Inc." "American Airlines Inc." "Alaska Airlines Inc." "JetBlue Airways"
```

1.1.3 airports

This dataset contains useful metadata about airports, that is:

- FAA airport code: `faa`;
- Usual name of the aiport: `name`;
- Location of airport: `lat`, `lon`;
- Altitude (in feet): `alt`;
- Timezone offset from GMT: `tz`;
- Daylight savings time zone: `dst` A = Standard US DST: starts on the second Sunday of March, ends on the first Sunday of November U = unknown N = no dst

The data frame has 7 variables and 1397 observations.

```
dim(airports)
```

```
## [1] 1458    8
```

```
head(airports)
```

```
## # A tibble: 6 x 8
##     faa                           name      lat       lon  alt    tz
##   <chr>                          <chr>    <dbl>     <dbl> <int> <dbl>
## 1   04G             Lansdowne Airport 41.13047 -80.61958  1044    -5
## 2   06A  Moton Field Municipal Airport 32.46057 -85.68003   264    -6
## 3   06C            Schaumburg Regional 41.98934 -88.10124   801    -6
## 4   06N                Randall Airport 41.43191 -74.39156   523    -5
## 5   09J          Jekyll Island Airport 31.07447 -81.42778    11    -5
## 6   0A9 Elizabethton Municipal Airport 36.37122 -82.17342  1593    -5
## # ... with 2 more variables: dst <chr>, tzone <chr>
```

```
str(airports)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1458 obs. of  8 variables:
##  $ faa  : chr  "04G" "06A" "06C" "06N" ...
## $ name : chr "Lansdowne Airport" "Moton Field Municipal Airport" "Schaumburg Regional" "Rand
##  $ lat  : num  41.1 32.5 42 41.4 31.1 ...
##  $ lon  : num  -80.6 -85.7 -88.1 -74.4 -81.4 ...
##  $ alt  : int  1044 264 801 523 11 1593 730 492 1000 108 ...
##  $ tz   : num  -5 -6 -6 -5 -5 -5 -5 -5 -5 -8 ...
##  $ dst  : chr  "A" "A" "A" "A" ...
## $ tzone: chr "America/New_York" "America/Chicago" "America/Chicago" "America/New_York" ...
```

```
##  - attr(*, "spec")=List of 2
##   ..$ cols   :List of 12
##   .. ..$ id     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ name   : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ city   : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ country: list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ faa    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ icao   : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ lat    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ lon    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ alt    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_integer" "collector"
##   .. ..$ tz     : list()
##   .. .. ..- attr(*, "class")= chr  "collector_double" "collector"
##   .. ..$ dst    : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   .. ..$ tzone  : list()
##   .. .. ..- attr(*, "class")= chr  "collector_character" "collector"
##   ..$ default: list()
##   .. ..- attr(*, "class")= chr  "collector_guess" "collector"
##   ..- attr(*, "class")= chr "col_spec"
```

1.1.4 planes

This dataset contains plane metadata for all plane tailnumbers found in the FAA aircraft registry (American Airways (AA) and Envoy Air (MQ) report fleet numbers rather than tail numbers). The data frame has 9 variables and 3322 observations. The variables are organised as follow:

- Tail number: `tailnum`;
- Year manufactured: `year`;
- Type of plane: `type`;
- Manufacturer and model: `manufacturer`, `model`;
- Number of engines and seats: `engines`, `seats`;
- Average cruising speed in mph: `speed`;
- Type of engine: `engine`.

```
dim(planes)
```

```
## [1] 3322    9
```

```
head(planes)
```

```
## # A tibble: 6 x 9
##   tailnum  year                    type     manufacturer     model engines
##     <chr> <int>                   <chr>            <chr>     <chr>   <int>
## 1  N10156  2004 Fixed wing multi engine          EMBRAER EMB-145XR       2
## 2  N102UW  1998 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 3  N103US  1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 4  N104UW  1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## 5  N10575  2002 Fixed wing multi engine          EMBRAER EMB-145LR       2
## 6  N105UW  1999 Fixed wing multi engine AIRBUS INDUSTRIE  A320-214       2
## # ... with 3 more variables: seats <int>, speed <int>, engine <chr>
```

```
str(planes)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    3322 obs. of  9 variables:
##  $ tailnum     : chr  "N10156" "N102UW" "N103US" "N104UW" ...
##  $ year        : int  2004 1998 1999 1999 2002 1999 1999 1999 1999 1999 ...
## $ type     : chr  "Fixed wing multi engine" "Fixed wing multi engine" "Fixed wing multi engin
## $ manufacturer: chr  "EMBRAER" "AIRBUS INDUSTRIE" "AIRBUS INDUSTRIE" "AIRBUS INDUSTRIE" ...
##  $ model       : chr  "EMB-145XR" "A320-214" "A320-214" "A320-214" ...
##  $ engines     : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ seats       : int  55 182 182 182 55 182 182 182 182 182 ...
##  $ speed       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ engine      : chr  "Turbo-fan" "Turbo-fan" "Turbo-fan" "Turbo-fan" ...
```

1.1.5 `weather`

This dataset is about hourly meterological data for LGA, JFK and EWR. The data frame has 14 variables and 8719 observations. The variables are organised as follow:

- Weather station: `origin` (named `origin` to faciliate merging with `flights` data);
- Time of recording: `year`, `month`, `day`, `hour`;
- Temperature and dewpoint in F: `temp`, `dewp`;
- Relative humidity: `humid`;
- Wind direction (in degrees), speed and gust speed (in mph): `wind_dir`, `wind_speed`, `wind_gust`;

- Preciptation, in inches: `precip`;
- Sea level pressure in millibars: `pressure`;
- Visibility in miles: `visib`.

```
dim(weather)
```

```
## [1] 26130    15
```

```
head(weather)
```

```
## # A tibble: 6 x 15
##   origin  year month   day  hour  temp  dewp humid wind_dir wind_speed
##    <chr> <dbl> <dbl> <int> <int> <dbl> <dbl> <dbl>    <dbl>      <dbl>
## 1    EWR  2013     1     1     0 37.04 21.92 53.97      230   10.35702
## 2    EWR  2013     1     1     1 37.04 21.92 53.97      230   13.80936
## 3    EWR  2013     1     1     2 37.94 21.92 52.09      230   12.65858
## 4    EWR  2013     1     1     3 37.94 23.00 54.51      230   13.80936
## 5    EWR  2013     1     1     4 37.94 24.08 57.04      240   14.96014
## 6    EWR  2013     1     1     6 39.02 26.06 59.37      270   10.35702
## # ... with 5 more variables: wind_gust <dbl>, precip <dbl>,
## #   pressure <dbl>, visib <dbl>, time_hour <dttm>
```

```
str(weather)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    26130 obs. of  15 variables:
##  $ origin    : chr  "EWR" "EWR" "EWR" "EWR" ...
##  $ year      : num  2013 2013 2013 2013 2013 ...
##  $ month     : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ day       : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ hour      : int  0 1 2 3 4 6 7 8 9 10 ...
##  $ temp      : num  37 37 37.9 37.9 37.9 ...
##  $ dewp      : num  21.9 21.9 21.9 23 24.1 ...
##  $ humid     : num  54 54 52.1 54.5 57 ...
##  $ wind_dir  : num  230 230 230 230 240 270 250 240 250 260 ...
##  $ wind_speed: num  10.4 13.8 12.7 13.8 15 ...
##  $ wind_gust : num  11.9 15.9 14.6 15.9 17.2 ...
##  $ precip    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ pressure  : num  1014 1013 1013 1013 1013 ...
##  $ visib     : num  10 10 10 10 10 10 10 10 10 10 ...
## $ time_hour : POSIXct, format: "2013-01-01 01:00:00" "2013-01-01 02:00:00" ...
```

Chapter 2

Verb functions

In this section you will find exercises on the basic verbs of data manipulating provided by `dplyr`:

1. `select()`;
2. `filter()`;
3. `arrange()`;
4. `mutate()`;
5. `summarise()`.

2.1 `select()` and its friends

Note: all the exercises of this section are based on the `flights` dataset.

```
require(tidyverse)
require(nycflights13)
```

2.1.1 Exercise 1

Extract the following information about flights:

- month;
- day;
- air_time;
- distance.

2.1.2 Exercise 2

Extract all information about flights except hour and minute.

2.1.3 Exercise 3

Select all variables whose name ends in “time”.

2.1.4 Exercise 4

Select all variables whose name contains the word “delay”.

2.1.5 Exercise 5

Select the `tailnum` variable and rename it into `tail_num`.

2.1.6 Exercise 6

Select all the variables and rename the `tailnum` variable into `tail_num`.

2.2 `filter()` and its friends

Note: all the exercises of this section are based on the `flights` dataset.

```
require(tidyverse)
require(nycflights13)
```

2.2.1 Exercise 1

Select all flights which delayed more than 1000 minutes at departure.

2.2.2 Exercise 2

Select all flights which delayed more than 1000 minutes at departure or at arrival.

2.2.3 Exercise 3

Select all flights which took off from “EWR” and landed in “IAH” on Christmas Day.

2.2.4 Exercise 4

Select the first five flights in this dataset.

2.2.5 Exercise 5

Select the last ten flights in this dataset.

2.2.6 Exercise 6

Extract information about distance for all flights which delayed more than 1000 minutes at departure.

2.3 arrange()

Note: all the exercises of this section are based on the `flights` dataset.

```
require(tidyverse)
require(nycflights13)
```

2.3.1 Exercise 1

Sort the flights in chronological order.

2.3.2 Exercise 2

Sort the flights by decreasing arrival delay.

2.3.3 Exercise 3

Sort the flights by origin (in alphabetical order) and decreasing arrival delay.

2.4 mutate() and its friends

Note: all the exercises of this section are based on the `flights` dataset.
Times are in minutes and distances are in miles.

```
require(tidyverse)
require(nycflights13)
```

2.4.1 Exercise 1

Add the following new variables to the `flights` dataset:

- the gained time in minutes, defined as the difference between delay at departure and delay at arrival;
- the speed in miles per hour (`distance` / `air_time` * 60).

Show only the following variables: delay at departure, delay at arrival, distance, air time and the two new variables (gained time and speed).

2.4.2 Exercise 2

Redo the previous calculations keeping only the new variables.

2.4.3 Exercise 3

After sorting flights in chronological order, for each flight calculate the difference between its delay at arrival and the delay at arrival of the immediately previous flight. Have R to show only the delay variables (delay at departure, delay at arrival and the new variable).

2.4.4 Exercise 4

For each flight calculate the 'min ranking' in terms of delay at arrival.

2.4.5 Exercise 5

For each flight calculate the 'first ranking' in terms of delay at arrival.

2.4.6 Exercise 6

Create a variable which indicates if a flight took off on time, i.e. departure delay is more than -4 and less than 4 minutes late.

2.5 `summarise()`

Note: all the exercises of this section are based on the `flights` dataset.

```r
require(tidyverse)
require(nycflights13)
```

2.5.1 Exercise 1

Calculate minimum, mean and maximum delay at arrival.

2.5.2 Exercise 2

Calculate minimum, mean and maximum delay at arrival for flights in January.

2.5.3 Exercise 3

Calculate the number of flights are contained in the dataset

Chapter 3

Grouping data

3.1 `group_by()`

Note: all the exercises of this section are based on the `flights` dataset.

```
library(tidyverse)
library(nycflights13)
```

3.1.1 Exercise 1

Calculate number of flights, minimum, mean and maximum delay at arrival for flights by month.

3.1.2 Exercise 2

Calculate number of flights, mean delay at departure and arrival for flights by origin.

3.1.3 Exercise 3

Calculate the number of flights that go to each possible destination.

3.1.4 Exercise 4

Calculate the number of flights for each day. Save the result in a data frame called `per_day`.

3.1.5 Exercise 5

By exploiting `per_day`, calculate the number of flights for each month. Save the result in a data frame called `per_month`.

3.1.6 Exercise 6

Calculate the mean daily number of flights per month.

Chapter 4

Do

4.1 `do`

Note: all the exercises of this section are based on the `flights` dataset.

```
library(dplyr)
library(nycflights13)
```

4.1.1 Exercise 1

Calculate quartiles (25-, 50- and 75-percentiles) of delay at arrival per origin. Put all three quartiles in a unique column.

4.1.2 Exercise 2

Redo the previous exercise putting the three quartiles in three different columns (hint: use `summarise()`).

4.1.3 Exercise 3

Calculate mean and standard deviation of delay at arrival per origin. Put both statistics in a unique column.

4.1.4 Exercise 4

Redo the previous exercise putting mean and standard deviation in two different columns (hint: use `summarise()`).

Chapter 5

Combining data

5.1 Joins: inner_join(), left_join(), right_join(), etc.

Note: all the exercises of this section are based on `flights`, `airlines`, `airports` or `planes` datasets.

```
library(dplyr)
library(nycflights13)
```

5.1.1 Exercise 1

Keep only the following variables of the `flights` dataset: month, day, hour, origin, destination and carrier. Save this dataset in a data frame and call it `flights_red`. Through a proper join command, add the carrier name to `flights_red` (this piece of information is available in `airlines`).

5.1.2 Exercise 2

Through a proper join command, add name, latitude, longitude and altitude of the origin airport to `flights_red` (these pieces of information are available in `airports`). Do the same also for the destination airport. (If you are able to, try to keep variables about both origin and destination airports in the same final dataset).

5.1.3 Exercise 3

Through the `inner_join()` function, redo the same for the destination airport but keep only the flights whose information is available in both datasets (`flights` and `airports`).

5.1.4 Exercise 4

Redo the exercise 3 by using `full_join()` instead of `inner_join()`. What is the difference in the result?

5.1.5 Exercise 5

Through the `anti_join()` function, extract all the flights from `flights` whose information about destination airport is not available in `airports`.

5.1.6 Exercise 6

Sort the `planes` dataset by increasing year. Then create two datasets: the first will deal with planes older than 2000; the second will deal with planes of 2000 or newer. Finally create a unique dataset where the first rows will deal with the newest planes, whereas the last rows will deal with the oldest planes.

Chapter 6

Tidy data with tidyr

6.1 tidyr

```
library(tidyverse)
```

6.1.1 Exercise 1

Consider the following dataset:

```
heartrate_wide <- data.frame(
  name = c("Aldo", "Giovanni", "Giacomo"),
  surname = c("Baglio", "Storti", "Poretti"),
  morning = c(67, 80, 64),
  afternoon = c(56, 90, 50)
)
heartrate_wide
```

```
##       name surname morning afternoon
## 1     Aldo  Baglio      67        56
## 2 Giovanni  Storti      80        90
## 3  Giacomo Poretti      64        50
```

It represents the heart rate measured on three patients in the morning and in the afternoon. The dataset is in the wide format: change it to the long format through a proper `tidyr` function. Save the result in a data frame and call it `heartrate_long`.

6.1.2 Exercise 2

Starting from `heartrate_long`, come back to a dataset in a wide format through a proper `tidyr` function. The result should be obviously equal to `heartrate_wide`.

6.1.3 Exercise 3

Consider the dataset `heartrate_wide` and unite name and surname of the patients in a unique column through a proper `tidyr` function. Save the result in a new data frame called `heartrate_united`.

6.1.4 Exercise 4

Starting from `heartrate_united`, come back to a dataset where name and surname are in two different columns through a proper `tidyr` function. The result should be obviously equal to `heartrate_wide`.

Chapter 7

Handling Missing values

7.1 Data import: set working directory

Some of the data that will be used in this exercises are contained in the data folder. Hence you should set your working directory in the *data* folder, using `setwd()` function, like in this example:

```
setwd("~/Documents/datamanage/exercises/data)
```

You will work inside this folder.

```
library(tidyverse)
```

7.1.1 Exercise 1

Consider the following dataset:

```
heartrate <- data.frame(
  name = c("Aldo", "Giovanni", "Giacomo", "Aldo", "Giovanni", "Giacomo",
           "Giovanni", "Giacomo"),
  surname = c("Baglio", "Storti", "Poretti", "Baglio", "Storti", "Poretti",
              "Storti", "Poretti"),
  when = c("morning", "morning", "morning", "afternoon", "afternoon", "afternoon",
           "evening", "evening"),
  heartrate = c(67, 80, 64, 56, 90, 50, 60, 85)
)
```

It represents the heart rate measured on three patients in the morning, in the afternoon and in the evening. Make explicit any implicit missing value. How many missing values do you see?

7.1.2 Exercise 2

Import data in the file `marks.Rdta`. Missing values have been recorded as ".". What's the percentage of missing values in the data? Replace them with `NA` and drop them.

7.1.3 Exercise 3

Import the data `heartrate_NA.Rdta` . Consider all the missing values you find and replace them using the function `fill()` when possible.

Chapter 8

Dates with lubridate

8.1 lubridate

Note: all the exercises of this section are based on the `flights` dataset.

```
require(tidyverse)
require(lubridate)
require(nycflights13)
```

8.1.1 Exercise 1

Using the flights data, build the variable `dep_date` based on the variables `year`, `month` and `day`. First use the function `unite()` and then the parsing function `ydm()`. Select only the new variable and save the new data frame called `flights_date`.

8.1.2 Exercise 2

Using the dataset, shift all flights by two months. Save it in a separate data frame called `flights_date_2`.

8.1.3 Exercise 3

Take the new date (2 months ahead) and substruct the orginal variable `date`. `flights_date_2`.

Chapter 9

Manipulating strings with `stringr`

9.1 Data import: set working directory

In this section you will work with data are contained in the data folder. Hence you should set your working directory in the *data* folder, using `setwd()` function, like in this example:

```
setwd("~/Documents/datamanage/exercises/data)
```

You will work inside this folder.

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

9.1.1 Exercise 1

Import the data `aire_milano_strings.txt` which is a tab delimited file. Find how China has been codified (notice that the file is in Italian) and manipulate that string as you find more confortable for you. Save the results in a new tibble.

```
## Parsed with column specification:
## cols(
##   Residenza = col_character(),
##   MotivoIscrizioneEstero = col_character(),
##   Num = col_integer()
## )
```

9.1.2 Exercise 2

Using the data modified in exercise 1, find all the countries whose names contain non-alphanumeric characters. Identify what kind of characters they contain.

9.1.3 Exercise 3

Consider now the column with information on the reason for migrating. Count how many different reasons there are and notice that citizenship was recorded in two slightly different ways: "acquisto cittadinanza" and "per acquisto cittadinanza". Replace one of them so that they are the same.

Chapter 10

Case study

10.1 Recap exercise

In this section you will use all the tools provided throughout the course. It is a simple example on real data that shows you the overall usage of the tools you were given during this course.

In the data folder you find the following three files:

1. `rating_final.csv`
2. `chefmezcuisine.csv`
3. `userprofile.csv`

These are the files you will work on in this section.

The files contain data on the rating of some restaurants in the US, their characteristics and the characteristics of the users rating them.

This exercise will guide you to first clean and combine data, and then answer questions on people's restaurant preferences.

Before starting th exercise, you should set your working directory in the *data* folder, using `setwd()` function, like in this example:

```
setwd("/data")
```

You will work in this folder.

10.2 Exercise

1. First of all you need to import the three files into `R` using the correct `readr` function. In order to find the correct function and to set the right options, you may explore the files by opening them in csv (check which is the separator, if there are column names, etc).

2. In order to understand what you are working on, check how many columns and rows each data frame is composed of, and check the type of the variables you are working with. If the variables type has not correctly been parsed, parse it manually. Further explore those variables that you think may be interesting for understanding different people's tastes, by for example checking how many levels they have, etc etc.

3. Consider `userprofile` data frame. Create a new data frame containing only relevant variables for users profile. Among these, keep the variables: `userID`, `birth_year`, `budget`, `marital_status`, `personality`, `smoker` and `activity`. If you think there are other relevant variables, you may include them in the new data frame as well. Call the new data frame `userprofile_reduced`.

4. Focus on the data frame `userprofile_reduced`. By exploring the different values recorded for `budget`, you may notice there are missing values. What are they recorded by? Replace all missing values with `NA`. Do the same for all the variables in `userprofile_reduced`.

5. Note that for all users we have the year of birth but we do not have the age. As it may be easier to deal with their age, build a variable called `age` and drop `birth_year`.

6. All three data frames are now ready to use. Merge the three data frames (`rating_final`, `userprofile_reduced` and `chefmoicuisine`) so that you keep all rows and columns of rating and you add all the variables of `chefmezcuisine` and `userprofile_reduced`. Call the new data frame `rating_all`.

7. Group data by `placeID` and find the average rating (`rating`, `food_rating` and `service_rating`). Sort the results so that places with the highest average rating are at the top. Show id of such places and type of cuisine. What is the cuisine type of users' favourite restaurants?

8. Find mean and standard deviation of all rating variables. Do you notice differences with regards to ratings of students as compared to people that are employed? Do you find differences in smokers and non smokers? Think of other groups of users to compare.

9. What are the best restaurant for smokers? What are the best restaurant for those that are on a budget?

10. Think of other analysis that you may perform with these data and use the tidyverse toolbox to answer your questions!