Case Study 2

AKSTA Statistical Computing 107.258

Maximilian Hagn

16. May 2022

Task 1

In order to import the .csv file, the package "readr" must first be installed:

```
install.packages("readr")
```

Afterwards the library can be included:

```
library("readr")
```

Then the file "country-codes.csv" will be imported with the help of the "read.csv" function and stored in the variable "countryCodes":

```
countryCodes <- read.csv("country-codes.csv", sep=",", header=TRUE)
```

Next, the package "dplyr" must be installed. With the help of this package the desired columns can be selected:

```
install.packages("dplyr")
```

Afterwards the library can be included:

```
library(dplyr)
```

Now the desired columns can be selected with the "select" function and assigned to the variable:

```
countryCodes <- countryCodes %>% select("official_name_en",
                        "ISO3166.1.Alpha.2",
                        "ISO3166.1.Alpha.3",
                        "Developed...Developing.Countries",
                        "Region.Name",
                        "Sub.region.Name")
```

Task 2

To import the .csv file, the same procedure as in Task 1 can be used:

```
countryYouthUnemploymentRate <- read.csv("rawdata_373.csv")
countryYouthUnemploymentRate <- countryYouthUnemploymentRate %>%
                            select("country_name",
                                   "youth_unempl_rate")
```

To import the text file the function “read.fwf” is used. The “widths” property can be used to specify how long a column is in characters. The first value is prefixed with a minus, because this column should not be imported. The “skip” property is used to specify that the first two lines should be skipped. Furthermore, it is specified how the imported columns have to be named.

```
countryMedianAge <- read.fwf("rawdata_343.txt",
                             widths = c(-8, 66, 4),
                             skip = 2,
                             col.names = c("country_name", "median"))
```

Task 3

Since we have already installed and imported the “dplyr” library in the first task, this step is not needed again. Since all observations of both tables are to be saved, the function “full_join” is used in this task. Both tables contain the column “country_name”. This column can be used to join the tables.

```
countryAgeUnemploymentUnion = full_join(countryMedianAge,
                                        countryYouthUnemploymentRate,
                                        by = c("country_name" = "country_name"))
```

Task 4

If the two tables are joined with the "full_join" function, it can be seen that the values for the average age and the unemployment rate are not inserted correctly.

```
countryCodesJoin <- countryCodes %>% full_join(countryAgeUnemploymentUnion,
                                               by = c("official_name_en" = "country_name"))
```

This is apparently because the names are not exactly the same and so there are spaces or hidden characters in the names. To remove the blanks in the table containing the average age and the unemployment rate the function "trimws" can be used. The function specifies which column is to be cleaned and where and which whitespaces are to be removed.

```
countryAgeUnemploymentUnion$country_name <-
  trimws(countryAgeUnemploymentUnion$country_name,
         which = c("both"),
         whitespace = "[ ]")
```

After the spaces are removed, the tables can be merged:

```
countryCodesJoin <- countryCodes %>% full_join(countryAgeUnemploymentUnion,
                                               by = c("official_name_en" = "country_name"))
```

However, it can be seen that not all lines are joined correctly, since some names of the countries differ from each other, although the same country is meant.

For the final merging of the tables, the Excel table "CIA_factbook_matching_table_iso.xlsx" should be used. In order to import an Excel table, the package "readxl" must be installed first.

```
install.packages("readxl")
```

Afterwards the library can be included:

```
library("readxl")
```

The next step is to import the Excel table.

```
isoCodes <- read_excel("CIA_factbook_matching_table_iso.xlsx")
```

To perform the merging, first the country identifiers from the Excel spreadsheet are added to both tables. To join the "isoCodes" with the "countryCode" table, the "ISO3166" values are used. To add the codes to the "countryAgeUnemploymentUnion" table, the name of the country is used, since this table does not contain "ISO3166" codes.

```
isoCodesCountryCodesUnion    <- countryCodes %>% left_join(isoCodes,
                                                 by = c("ISO3166.1.Alpha.2" = "ISO 3166 2",
                                                        "ISO3166.1.Alpha.3" = "ISO 3166 3"))

isoCodesAgeUnemploymentUnion <- countryAgeUnemploymentUnion %>% left_join(isoCodes,
                                                 by = c("country_name" = "Country"))
```

Finally, the "full_join" function can be used to join both table. For this, both the "ISO3166" codes and the names of the countries are now used. The result is stored in the variable "df_vars":

```
df_vars <- isoCodesCountryCodesUnion %>% full_join(isoCodesAgeUnemploymentUnion,
                                                 by = c("Country" = "country_name",
                                                        "ISO3166.1.Alpha.2" = "ISO 3166 2",
                                                        "ISO3166.1.Alpha.3" = "ISO 3166 3"))
```

Task 5

In a tidy data set, each **variable** must form a column, each **observation unit** must form a row, and each **type** of an **observational unit** must form a table. In the underlying data set, the **observational units** are each in a row. This is a name of a country in combination with a "ISO3166" code, the median age and the unemployment rate. A **variable** contains all values that measure the same attribute across all units. **Variables** in this dataset are "official_name_en", "ISO3166.1.Alpha.2", "ISO3166.1.Alpha.3", "Region.Name", "Sub.region.Name", "Country", "Developed…Developing.Countries", "median", and "youth_unempl_rate". I would call the first six **variables** mentioned **fixed**, since they describe the experimental design and were already known beforehand. Especially with the names of the countries or the "ISO3166" codes, it can be stated that these do not change and that they are fixed values. The last three **variables** are, in my opinion, **measured variables**, since they are used in the analyses to generate statistics. Furthermore, they were probably measured for these statistics and can be used for the actual measurements. The underlying dataset is tidy according to the criteria described first. Each **variable** forms a column and in this column are only values which describe the attribute of the variable. Furthermore, each **observational unit** forms a row, whereby a country is always assigned to the measured values. Since I would describe the combination of country, median age, development status and unemployment rate as a **type** of an **observational unit** and this combination should form a table, I would also state that this criteria is also fulfilled.

The information was obtained from the script chapter "Tidy data".

Task 6

To count the developed or developing countries the function "count" from the "dplyr" package can be used. The function is provided with the column containing the desired information.

```
DevelopedDevelopingCountriesCount <- df_vars %>% count(Developed...Developing.Countries)
```

It can be seen that in this column there are 66 entries with the value "Developed", 183 entries with the value "Developing" and 17 rows containing no entry.

```
DevelopedDevelopingCountriesCount
```

```
##   Developed...Developing.Countries   n
## 1                                    2
## 2                        Developed  66
## 3                       Developing 183
## 4                             <NA>  17
```

Task 7

To count how many countries per region are contained in the dataset, the function “count” can be used again. This time, however, the name of the region is specified.

```
DevelopedDevelopingCountriesCountRegion <- df_vars %>% count(Region.Name)
```

It can be seen that Africa contains 60 countries, America 57, Asia 51, Europe 52 and Oceania 29. There are again rows that do not contain any entries.

```
DevelopedDevelopingCountriesCountRegion
```

```
##   Region.Name  n
## 1              2
## 2      Africa 60
## 3    Americas 57
## 4        Asia 51
## 5      Europe 52
## 6     Oceania 29
## 7        <NA> 17
```

Task 8

In this task the values from the previous tasks are to be combined. For this purpose the ” count ” function is also used but this time both the name of the region and the column containing the developed and developing countries are specified.

```
DevelopedDevelopingCountriesCountForRegion <- df_vars %>%
  count(Region.Name, Developed...Developing.Countries)
```

Now a table is obtained in which for each region a value for the developed and a value for the developing countries is given. It is especially noticeable that in Africa not a single country is developed, whereas in Europe every country is developed.

```
DevelopedDevelopingCountriesCountForRegion
```

```
##    Region.Name Developed...Developing.Countries  n
## 1                                                2
## 2       Africa                       Developing 60
## 3     Americas                        Developed  5
## 4     Americas                       Developing 52
## 5         Asia                        Developed  3
## 6         Asia                       Developing 48
## 7       Europe                        Developed 52
## 8      Oceania                        Developed  6
## 9      Oceania                       Developing 23
## 10        <NA>                             <NA> 17
```

Task 9

In this task the countries are to be grouped according to the development status and then both the average and the standard deviation are to be calculated. For the grouping the function "group_by" can be used. Afterwards all values which contain a "NA" value or are empty are filtered out with the "filter" function. Finally, the "summarise" function can be used to create a statistic that contains the desired values. To calculate the average, the "mean" function can be used. This is applied to both the "median" column and the "youth_unempl_rate" column. For the standard deviation, the function "sd" can be used, which is also applied to both columns.

```
df_vars %>%
  group_by(Developed...Developing.Countries) %>%
  filter(Developed...Developing.Countries != NA
      | Developed...Developing.Countries != "") %>%
  summarise(
    Avg.Median.Age    = mean(median, na.rm = TRUE),
    SD.Median.Age     = sd(median, na.rm = TRUE),
    Avg.Youth.Unempl = mean(youth_unempl_rate, na.rm = TRUE),
    SD.Youth.Unempl  = sd(youth_unempl_rate, na.rm = TRUE)
  )
```

```
## # A tibble: 2 x 5
##   Developed...Dev~ Avg.Median.Age SD.Median.Age Avg.Youth.Unempl SD.Youth.Unempl
##   <fct>                     <dbl>         <dbl>            <dbl>           <dbl>
## 1 Developed                  41.9          4.19             16.4            9.78
## 2 Developing                 27.7          7.20             17.8           12.3
```

The result is a table that contains the desired values. It can be seen that the average age in the developed countries is higher than in the developing ones. It is also noticeable that youth unemployment is only slightly lower in the developed countries than in the developing countries.

Task 10

In this task, the analysis will be repeated, grouping not only by development status but also by region. Thus, the procedure remains mostly the same, but the “filter” function is now given the name of the region as a second parameter.

```
df_vars %>%
  group_by(Developed...Developing.Countries, Region.Name) %>%
  filter(Developed...Developing.Countries != NA
       | Developed...Developing.Countries != "") %>%
  summarise(
    Avg.Med.Age = mean(median, na.rm = TRUE),
    SD.Med.Age  = sd(median, na.rm = TRUE),
    Avg.Youth.U = mean(youth_unempl_rate, na.rm = TRUE),
    SD.Youth.U  = sd(youth_unempl_rate, na.rm = TRUE)
  )
```

```
## # A tibble: 8 x 6
## # Groups:   Developed...Developing.Countries [2]
##   Developed...Develop~ Region.Name Avg.Med.Age SD.Med.Age Avg.Youth.U SD.Youth.U
##   <fct>                <fct>             <dbl>      <dbl>       <dbl>      <dbl>
## 1 Developed            Americas           41.3      5.34         16.3     11.3
## 2 Developed            Asia               39.0      9.15         10.3      8.73
## 3 Developed            Europe             42.4      3.62         17.2     10.0
## 4 Developed            Oceania            37.4      0.212        11.6      0.212
## 5 Developing           Africa             20.9      4.96         18.1     14.2
## 6 Developing           Americas           32.7      5.29         16.9      9.28
## 7 Developing           Asia               30.2      6.27         17.0     11.4
## 8 Developing           Oceania            28.5      4.50         21.3     15.5
```

The result is a table containing the average and standard deviation for each region and both development stages. What is particularly noticeable in the data is that the average age is higher in the developed regions Americas and Europe. The lowest youth unemployment rates are found in the developed regions of Asia and Oceania.

Task 11

In this example, the table “df_vars” is to be extended by two additional columns. In each case, it is to be stored whether the value of the country is above the average of the region. For this purpose, a table can first be created that calculates an average value for each region. This is done as in the previous task, but now it is only grouped by the variable “Region.Name”. Furthermore, only two values are created for the respective mean value.

```
regionAverageTable <- df_vars %>%
  group_by(Region.Name) %>%
  filter(Developed...Developing.Countries != NA
       | Developed...Developing.Countries != "") %>%
  summarise(
    Avg.Med.Age = mean(median, na.rm = TRUE),
    Avg.Youth.U = mean(youth_unempl_rate, na.rm = TRUE),
  )
```

Next, a table can be created that has the same dimension as the “df_vars” table. For each country, the average value of the region is added, so that finally the values of the country can be compared with the average value of the region.

```
dfWithRegionAverage <- df_vars %>% left_join(regionAverageTable,
                                              by = c("Region.Name" = "Region.Name"))
```

Finally, for each value in the table, it is possible to check which value should be inserted in the two new columns. With the help of the function “mutate” new columns can be added. Both columns are named with the desired name. With the help of the “case_when” function a case distinction can be made. In this example, the columns are filled with the value “yes” if the corresponding value is greater than the average. If the value of the country is lower or equal to the average of the region, the value “no” is inserted. The result is a table that has been extended with the variables “above_average_median_age” and “above_average_yu”.

```
df_vars <- df_vars %>%
    mutate(above_average_median_age =
                case_when(
                  median            >   dfWithRegionAverage[,"Avg.Med.Age"] ~ "yes",
                  median            <=  dfWithRegionAverage[,"Avg.Med.Age"] ~ "no"
                ),
           above_average_yu =
                case_when(
                  youth_unempl_rate >   dfWithRegionAverage[,"Avg.Youth.U"] ~ "yes",
                  youth_unempl_rate <=  dfWithRegionAverage[,"Avg.Youth.U"] ~ "no"
                ))
```

Task 12

The last step is to write the variable “df_vars” into a .csv file. The function “write.csv2” can be used for this. By default, no rownames are written to the file, so no further adjustments are necessary. However, it must still be described that “NA” values are to be represented by a point, whereby this can be done with the property “na”.

```
write.csv2(df_vars, file="df_vars.csv", na=".")
```