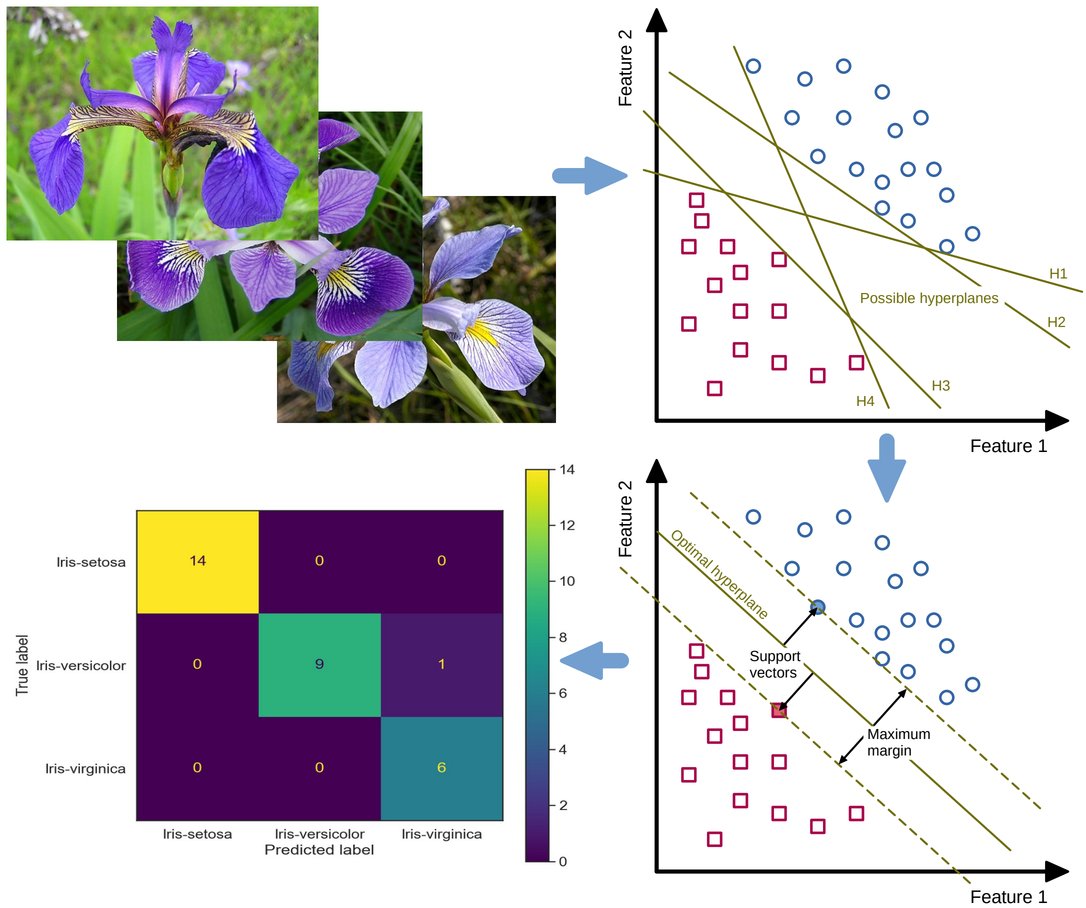
Preparing raw CSV input data from survey for analytical hierarchy process (AHP)

Björn Kasper (kasper.bjoern@bgetem.de)[1] and Henriette John (h.john@ioer.de)[2]

[1] *Berufsgenossenschaft Energie Textil Elektro Medienerzeugnisse*
[2] *Leibniz Institute of Ecological Urban and Regional Development*

January 3, 2023; version 0.2 (pre-release)



This is a placeholder for the abstract that needs to be added later.

Contents

1 Introduction

Why we use a Jupyter notebook to to publish the R program examples:

Jupyter is a new **open source** alternative to the proprietary numerical software Mathematica from **Wolfram Research** that is well on the way to become a **standard for exchanging research results** (Somers 2018; Romer 2018).

Originally Jupyter was intended as an IDE for the programming languages **Julia** and **Python**. Besides that it is also possible to install other interpreter kernels, such as the **IRkernel** for R. This can be interesting if the IDE **RStudio Desktop** is not available on the target platform used. For example, it is very difficult to install RStudio on the ARM-based embedded computer **Raspberry Pi** due to many technical dependencies. In contrast, using the R kernel in JupyterLab on the Raspberry Pi works very well and performant.

2 Global settings and dependencies

2.1 Install missing packages if not present yet

[1]:
```
# List of R packages that are used in this script
list.of.packages <- c("data.table")

# Query the already installed packages and save the missing ones in a new list
missing.packages <- list.of.packages[!(list.of.packages %in% installed.
  ↪packages()[,"Package"])]

# Install missing packages
if(length(missing.packages)) {
    install.packages(missing.packages)
} else {
```

```
    print("All required packages are installed.")
}
```

```
[1] "All required packages are installed."
```

2.2 Load package data.table

The package `data.table` is used for reading and manipulating tables (`data.table` inherits from `data.frame`). Install and load it:

[2]:
```
library(data.table)
```

2.3 Load packages knitr and IRdisplay

The `kable()` function from the package `knitr` is used to output dataframes as a markdown tables.

The `display_markdown()` function from the package `IRdisplay` renders the markdown table in the notebook as well as in the PDF version.

[3]:
```
library(knitr)
library(IRdisplay)
```

2.4 Set globally used input and output folders

[4]:
```
str_input_path = "./input_data_from_survey"
str_output_path = "./output_data_manipulated"
```

3 Functions to prepare the survey data for further analysis

The following functions are used to read the survey data from the CSV files and prepare the data structure for further analysis with the R package `ahpsurvey`.

3.1 Function to read the survey data from CSV files to dataframe objects

Define a function for reading in a CSV file to 4 different dateframes by selecting different columns.

[5]:
```
func_readCSVdata_to_dataframes <- function(str_CSVfilename) {

  # criteria (main criteria)
  df_mySurvey_1 <- fread(
    file = str_CSVfilename, encoding = "UTF-8",
    header = TRUE, sep = "\t", quote = "\"",
    # dec = ".", row.var = "CASE",
    select = c("CASE", "AK01", "AK02", "AK03",
               "RK01_01", "RK02_01", "RK03_01", "RK04_01", "RK05_01", "RK06_01")
    )

  # environmental sub-criteria
  df_mySurvey_2 <- fread(
    file = str_CSVfilename, encoding = "UTF-8",
    header = TRUE, sep = "\t", quote = "\"",
    # dec = ".", row.names = "CASE",
    select = c("CASE", "AU01", "AU02", "AU03",
               "RU01_01", "RU02_01", "RU03_01", "RU04_01", "RU05_01", "RU06_01")
```

```
    )

  # social sub-criteria
  df_mySurvey_3 <- fread(
    file = str_CSVfilename, encoding = "UTF-8",
    header = TRUE, sep = "\t", quote = "\"",
    # dec = ".", row.names = "CASE",
    select = c("CASE", "AS01", "AS02", "AS03",
               "RS01_01", "RS02_01", "RS03_01", "RS04_01", "RS05_01", "RS06_01")
    )

  # economic sub-criteria
  df_mySurvey_4 <- fread(
    file = str_CSVfilename, encoding = "UTF-8",
    header = TRUE, sep = "\t", quote = "\"",
    # dec = ".", row.names = "CASE",
    select = c("CASE", "AW01", "AW02", "AW03",
               "RW01_01", "RW02_01", "RW03_01", "RW04_01", "RW05_01", "RW06_01")
    )

  output <- list(df_mySurvey_1, df_mySurvey_2, df_mySurvey_3, df_mySurvey_4)

  return(output)
}
```

3.2 Function to format dataframes as a markdown tables

Following function formats given dataframes as markdown tables using the `kable()` function from the `knitr` package.

The `display_markdown()` function from the package `IRdisplay` renders the markdown table in the notebook as well as in the PDF version.

[6]:
```
func_render_md_tables <- function(df_table, str_table_header) {
    # format the dataframe as a markdown table using the 'kable()' function from
  ↪the 'knitr' package
    table_out <- kable(
        df_table,
        format = "markdown",
        # digits = 2,
        caption = str_table_header)

    display_markdown(as.character(table_out))
}
```

3.3 Function to prepare the data and store it in new dataframes

[7]:
```
func_scrambleData <- function(df_inputData, vec_colnames_search_1,
  ↪vec_colnames_search_2, vec_colnames_out) {
  # Generate new data frame ...
  df_outputData <- data.frame(matrix(ncol = 3, nrow = 0))
  # ... and name the columns
  colnames(df_outputData) <- vec_colnames_out

  # Generate 1. column
  for ( row_idx in 1:nrow(df_inputData) ) {
```

```r
    # filter column names by vector element
    if (df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_1[1], with=FALSE] == 1) {
      int_tmp_val <- as.integer(df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_2[1], with=FALSE])
      int_tmp_val <- int_tmp_val * -1 - 1

      df_outputData[row_idx, vec_colnames_out[1]] <- int_tmp_val
    }
    else if (df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_1[1], with=FALSE] == -1) {
      df_outputData[row_idx, vec_colnames_out[1]] <- 1
    }
    else if (df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_1[1], with=FALSE] == 2) {
      int_tmp_val <- as.integer(df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_2[2], with=FALSE])
      int_tmp_val <- int_tmp_val + 1

      df_outputData[row_idx, vec_colnames_out[1]] <- int_tmp_val
    }
  }

  # Generate 2. column
  for ( row_idx in 1:nrow(df_inputData) ) {
    # filter column names by vector element
    if (df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_1[2], with=FALSE] == 1) {
      int_tmp_val <- as.integer(df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_2[3], with=FALSE])
      int_tmp_val <- int_tmp_val * -1 - 1

      df_outputData[row_idx, vec_colnames_out[2]] <- int_tmp_val
    }
    else if (df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_1[2], with=FALSE] == -1) {
      df_outputData[row_idx, vec_colnames_out[2]] <- 1
    }
    else if (df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_1[2], with=FALSE] == 2) {
      int_tmp_val <- as.integer(df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_2[4], with=FALSE])
      int_tmp_val <- int_tmp_val + 1

      df_outputData[row_idx, vec_colnames_out[2]] <- int_tmp_val
    }
  }

  # Generate 3. column
  for ( row_idx in 1:nrow(df_inputData) ) {
    # filter column names by vector element
    if (df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_1[3], with=FALSE] == 1) {
      int_tmp_val <- as.integer(df_inputData[row_idx, colnames(df_inputData) %in% vec_colnames_search_2[5], with=FALSE])
      int_tmp_val <- int_tmp_val * -1 - 1
```

```
        df_outputData[row_idx, vec_colnames_out[3]] <- int_tmp_val
      }
      else if (df_inputData[row_idx, colnames(df_inputData) %in%␣
  ↪vec_colnames_search_1[3], with=FALSE] == -1) {
        df_outputData[row_idx, vec_colnames_out[3]] <- 1
      }
      else if (df_inputData[row_idx, colnames(df_inputData) %in%␣
  ↪vec_colnames_search_1[3], with=FALSE] == 2) {
        int_tmp_val <- as.integer(df_inputData[row_idx, colnames(df_inputData) %in%␣
  ↪vec_colnames_search_2[6], with=FALSE])
        int_tmp_val <- int_tmp_val + 1

        df_outputData[row_idx, vec_colnames_out[3]] <- int_tmp_val
      }
    }

    # return scrambled data frame
    return(df_outputData)
}
```

3.4 Function to write resulting dataframes to CSV files

[8]:
```
func_writeDataframe_to_CSVfile <- function(str_path, str_CSVfilename, df_dataframe,␣
  ↪str_filenameExtension) {
  # Split file name on second underscore, found here:
  # https://stackoverflow.com/questions/32398427/
  ↪r-split-a-character-string-on-the-second-underscore/32398489#32398489
  list_str_split <- strsplit(sub('(^[^_]+_[^_]+)_(.*)$', '\\1 \\2',␣
  ↪str_CSVfilename), ' ')

  # extend the file name prefix and glue together with old suffix
  str_CSVfilename_extended <- paste(list_str_split[[1]][1], str_filenameExtension,␣
  ↪list_str_split[[1]][2], sep="_")

  # extend file name by path
  str_CSVfilename_extended <- paste(str_path, str_CSVfilename_extended, sep="/")

  write.table(df_dataframe, file = str_CSVfilename_extended,
              fileEncoding = "UTF-8", row.names = FALSE,
              col.names = TRUE, sep = "\t", quote = TRUE)
}
```

4 Create data frame (table) handling the file names of input CSV data (raw data from survey)

[9]:
```
df_csvInputFiles <- data.table(
  file_idx = 1:4,
  keys = c("all", "CA", "NGO", "PE"),
  filenames = c("rdata_all_AHP_edible_Cities_2022-03-18_09-53.csv",
                "rdata_CA_AHP_edible_Cities_2022-03-18_10-28.csv",
                "rdata_NGO_AHP_edible_Cities_2022-03-18_10-40.csv",
                "rdata_PE_AHP_edible_Cities_2022-03-18_10-41.csv"),
```

```
  descriptions = c("all target groups together",
                   "City Administrations",
                   "Non-Governmental Organisations",
                   "Practitioners and Experts")
)

func_render_md_tables(df_csvInputFiles, "File table for handling the file names of␣
↪input CSV data (raw data from survey)")
```

Table 1: File table for handling the file names of input CSV data (raw data from survey)

file_idx	keys	filenames	descriptions
1	all	rdata_all_AHP_edible_Cities_2022-03-18_09-53.csv	all target groups together
2	CA	rdata_CA_AHP_edible_Cities_2022-03-18_10-28.csv	City Administrations
3	NGO	rdata_NGO_AHP_edible_Cities_2022-03-18_10-40.csv	Non-Governmental Organisations
4	PE	rdata_PE_AHP_edible_Cities_2022-03-18_10-41.csv	Practitioners and Experts

5 Prepare the data and store it in new CSV files for each criterion

5.1 Criteria (main criteria)

Walk over all input CSV files, select necessary columns, filter cells by given algorithm, and write the results to output CSV files:

[10]:
```
vec_colnames_search_1 <- c('AK01', 'AK02', 'AK03')
vec_colnames_search_2 <- c('RK01_01', 'RK02_01', 'RK03_01', 'RK04_01', 'RK05_01',␣
↪'RK06_01')
vec_colnames_out <- c('Envi_Soci', 'Envi_Econ', 'Soci_Econ')

for ( row_idx in 1:nrow(df_csvInputFiles) ) {
  # create list of data frames from current input CSV file
  str_filename <- paste(str_input_path, df_csvInputFiles[row_idx, filenames], sep="/
↪")
  list_dataframes <- func_readCSVdata_to_dataframes(str_filename)

  # scramble the data frames
  df_scrambledData <- func_scrambleData(list_dataframes[[1]],␣
↪vec_colnames_search_1, vec_colnames_search_2, vec_colnames_out)

  # write scrambled data frames to output CSV file
  func_writeDataframe_to_CSVfile(str_output_path, df_csvInputFiles[row_idx,␣
↪filenames], df_scrambledData, "crit")
}
```

5.2 Environmental sub-criteria

Walk over all input CSV files, select necessary columns, filter cells by given algorithm, and write the results to output CSV files:

[11]:
```
vec_colnames_search_1 <- c('AU01', 'AU02', 'AU03')
vec_colnames_search_2 <- c('RU01_01', 'RU02_01', 'RU03_01', 'RU04_01', 'RU05_01',␣
↪'RU06_01')
vec_colnames_out <- c('Clim_BDiv', 'Clim_CiEc', 'BDiv_CiEc')

for ( row_idx in 1:nrow(df_csvInputFiles) ) {
  # create list of data frames from current input CSV file
  str_filename <- paste(str_input_path, df_csvInputFiles[row_idx, filenames], sep="/
↪")
  list_dataframes <- func_readCSVdata_to_dataframes(str_filename)

  # scramble the data frames
  df_scrambledData <- func_scrambleData(list_dataframes[[2]],␣
↪vec_colnames_search_1, vec_colnames_search_2, vec_colnames_out)

  # write scrambled data frames to output CSV file
  func_writeDataframe_to_CSVfile(str_output_path, df_csvInputFiles[row_idx,␣
↪filenames], df_scrambledData, "env")
}
```

5.3 Social sub-criteria

Walk over all input CSV files, select necessary columns, filter cells by given algorithm, and write the results to output CSV files:

[12]:
```
vec_colnames_search_1 <- c('AS01', 'AS02', 'AS03')
vec_colnames_search_2 <- c('RS01_01', 'RS02_01', 'RS03_01', 'RS04_01', 'RS05_01',␣
↪'RS06_01')
vec_colnames_out <- c('KEdu_Comm', 'KEdu_Part', 'Comm_Part')

for ( row_idx in 1:nrow(df_csvInputFiles) ) {
  # create list of data frames from current input CSV file
  str_filename <- paste(str_input_path, df_csvInputFiles[row_idx, filenames], sep="/
↪")
  list_dataframes <- func_readCSVdata_to_dataframes(str_filename)

  # scramble the data frames
  df_scrambledData <- func_scrambleData(list_dataframes[[3]],␣
↪vec_colnames_search_1, vec_colnames_search_2, vec_colnames_out)

  # write scrambled data frames to output CSV file
  func_writeDataframe_to_CSVfile(str_output_path, df_csvInputFiles[row_idx,␣
↪filenames], df_scrambledData, "soc")
}
```

5.4 Economic sub-criteria

Walk over all input CSV files, select necessary columns, filter cells by given algorithm, and write the results to output CSV files:

[13]:
```
vec_colnames_search_1 <- c('AW01', 'AW02', 'AW03')
vec_colnames_search_2 <- c('RW01_01', 'RW02_01', 'RW03_01', 'RW04_01', 'RW05_01',␣
↪'RW06_01')
vec_colnames_out <- c('Qual_LVCs', 'Qual_Affo', 'LVCs_Affo')
```

```
for ( row_idx in 1:nrow(df_csvInputFiles) ) {
  # create list of data frames from current input CSV file
  str_filename <- paste(str_input_path, df_csvInputFiles[row_idx, filenames], sep="/
  ↪")
  list_dataframes <- func_readCSVdata_to_dataframes(str_filename)

  # scramble the data frames
  df_scrambledData <- func_scrambleData(list_dataframes[[4]],␣
  ↪vec_colnames_search_1, vec_colnames_search_2, vec_colnames_out)

  # write scrambled data frames to output CSV file
  func_writeDataframe_to_CSVfile(str_output_path, df_csvInputFiles[row_idx,␣
  ↪filenames], df_scrambledData, "eco")
}
```

6 Summary and outlook

7 References

Online references

Romer, Paul (Apr. 13, 2018). *Jupyter, Mathematica, and the Future of the Research Paper*. English. URL: https://paulromer.net/jupyter-mathematica-and-the-future-of-the-research-paper/ (visited on 09/08/2022) (cit. on p. 2).

Somers, James (Apr. 5, 2018). *The Scientific Paper Is Obsolete*. English. The Atlantic. URL: https://www.theatlantic.com/science/archive/2018/04/the-scientific-paper-is-obsolete/556676/ (visited on 09/08/2022) (cit. on p. 2).