Universität zu Köln

SFB 1252 PROMINENCE IN LANGUAGE

Practical Skills for Working with Linguistic Data

Short name: Practical Linguistic Skills (PLS)

Fahime (Fafa) Same
f.same@uni-koeln.de

WiSe23/24

Important Announcement

To participate in the course, you MUST submit your first mini-task **no later than October 31, 2023**.

You can choose either last week's or this week's mini-tasks. The deadline for both is October 31, 2023. They are posted to ILIAS (Folder Minitasks).

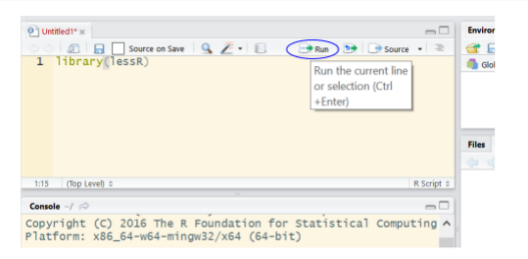
Session 3: 25.10.2023
R basics (2) and R markdown

Recap of the Previous Session: R and RStudio

- R is there to make our data life a bit easier. RStudio is an integrated development environment for R. It provides a more convenient and efficient workflow.
  NOTE: After installing both R and RStudio, there is no need to open R separately; simply launch RStudio to access R's capabilities.
- Alternative to Desktop RStudio: Posit Cloud (`https://posit.cloud/`)

Recap of the Previous Session: Dataframe

Important: To run an R command, put the cursor on the line of the command and then click the Run button at the top of the file window. Or just **press CTRL-Enter** in Windows and **press Command-Enter** in Mac.



The $<-$ symbol is the **assignment operator** in R, composed of a "less than" and a "hyphen/minus" symbol (no space between them).

What is a dataframe?

- It is a two-dimensional tabular data structure.
- Rows represent observations.
- Columns represent variables

ID	Name	Score
1	Alice	90
2	Bob	85
3	Carol	92
4	Dave	88
5	Rachel	95
6	Richard	80
7	John	83
8	Monica	86

Creating a dataframe in R

```
# Create the data frame
student_data <- data.frame(
  ID = c(1, 2, 3, 4, 5, 6, 7, 8),
  Name = c("Alice", "Bob", "Carol", "Dave", "Rachel", "Richard",
       "John", "Monica"),
  Score = c(90, 85, 92, 88, 95, 80, 83, 86)
)
```

Exploring a dataframe (df)

- **head()** Displays the first few rows of the dataframe. Default= 6 rows
- **tail()** Displays the last few rows of the dataframe.
- **str()** Provides the structure of the df, including variable names and data types
- **summary()** Generates summary statistics for each variable in the df.
- **colnames()** Displays the name of the columns

```
head(student_data) #displays the first 6 rows of df
tail(student_data, n =3) #displays the last 6 rows of df
str(student_data)
summary(student_data)
colnames(student_data)
```

Data Types

Data stored in columns can have different types:

- **Numeric** numerical values such as height, age, length
- **Character** text or strings of characters
- **Factor** categorical values with distinct levels or categories (e.g., gender, nationality)

Looking at Individual Instances in a dataframe

- Use square brackets ([ ]) to access data by their index number.
  Syntax: dataframe_name[row_index, column_index] (row first, column second)

```
student_data[2,3] #retrieves the value at the
    intersection of the 2nd row and 3rd column.
student_data[1 , ] #?
student_data[ , 1 ] #?
student_data[1:5 , ] #?
student_data[1:5 , 2:3] #?
student_data[2:3, "Name"] #you can also give
    the column name instead of its index
student_data[2:3, c("Name", "Score")] #?
student_data[1:2] #?
```

ID	Name	Score
1	Alice	90
2	Bob	85
3	Carol	92
4	Dave	88
5	Rachel	95
6	Richard	80
7	John	83
8	Monica	86

Best Practices for Coding in R

1. **Punctuation**
   - Assignment Operator ($<-$): Use for variable assignment; avoid $=$.
   - Quotes (" " or ' '): Both work, but be consistent.
   - Commas: Separate function arguments and vector/matrix elements.
   - Parentheses: Necessary for functions (func()) and control structures (if()). No space between the function name and parentheses.
2. **Indentation** Use indentation to make your code more readable.
3. **Naming Conventions** Stick to lowercase, camelCase, or separate words with underscores whenever possible (my_dataframe). Give short but meaningful names.
4. **Code Structure**
   - Comments: Use liberally to comment on your code.
   - Whitespace: Use blank lines to separate logical blocks of code.
5. **Error Messages** Pay close attention; they usually point to the issue.

Tidyverse

TIDYVERSE

- is a collection of R packages designed for data science and data manipulation.
- provides a consistent and intuitive syntax for data manipulation.
- offers powerful tools for handling and transforming data.
- is more efficient and readable than base R.
- Difficulty installing tidyverse? Use posit cloud for now!

```
install.packages("tidyverse")
# After installing a package, comment out the line using: #
    install.packages("tidyverse)
library(tidyverse)
```

Important Functions for Now

- **select()** Choose specific columns.
- **filter()** Subset rows based on conditions.
- **rename()** Renaming columns.
- **mutate()** Create new variables/columns.
- **arrange()** Sort the data based on one or more variables.

General syntax of a tidyverse command:

function_name(dataframe, arguments)

Arguments are the specific parameters or conditions that guide the function.

Function: select()

This function is used for subsetting columns and removing unwanted columns.

Components

- **Function Name:** `select`
- **Data:** The data frame you are manipulating.
- **Arguments:** The columns you want to **keep or remove**.

```
select(data, column_names)
onlyNames <- select(student_data, Name) # Keeping only the Name
    column
```

Function: filter()

This function **filters** rows based on specified conditions.

```
filter(student_data, Score > 90) # filter rows where scores is
    great than 90
below90 <- filter(student_data, Score <= 90) # Filter rows
    where scores are less or equal to 90. Assign the new
    filtered dataframe to the variable below90
filter(student_data, Name == "Alice") # filter rows that the
    name of the student is Alice
```

NOTE And (&) and or (|) operators allow you to combine two multiple conditions:

- mother tongue = German & gender = female (both conditions should met)
- mother tongue = German | age ¡ 30 (one condition suffices)

Function: rename()

As the name suggests, this function is used for renaming columns.

```
rename(data, new_name = old_name)

rename(student_data, "Number" = "ID")
```

Function: mutate()

This function is used to create new columns or modifying the existing columns.

```
mutate(data, new_column = something)
mutate(student_data, occupation = "student") # Adding an
    occupation column with the value "student"
mutate(student_data, Score = Score +1) # Increasing the scores
    by 1
mutate(student_data, NameScore = str_c(Name,"-", Score)) #
    Concatenating Name and Score columns
mutate(student_data, Uppercase = str_to_upper(Name)) # Turn
    names into uppercase
```

Pipe Operator (%>%)

- Shortcut: shift+ctrl+M (Windows) , command+shit+M (Mac)
- The "pipe" operator (%>%), allows you to chain together multiple operations so that the output of one operation becomes the input for the next.

`https://info5940.infosci.cornell.edu/slides/pipes-and-functions-in-r/#9`

Pipe Operator (%>%)

- Shortcut: shift+ctrl+M (Windows) , command+shit+M (Mac)
- The "pipe" operator (%>%), allows you to chain together multiple operations so that the output of one operation becomes the input for the next.

https://info5940.infosci.cornell.edu/slides/pipes-and-functions-in-r/#9

```
student_data_pipe <- student_data %>% #start a pipeline
  select(Name, Score) %>% #select only Name and Score columns
  rename(StudName = Name) %>% #rename Name column to StudName
  mutate(ScoreCorrected = Score + 1) #Increase the score by 1
```

QUESTIONS?