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

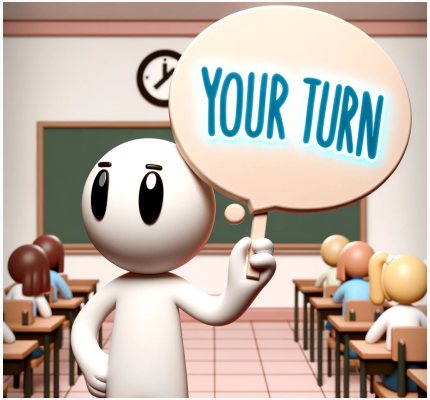
To participate in the course, you MUST submit your first mini-task **no later than October 31, 2023**.

You can choose either this week's or next week's mini-tasks. They will be posted to ILIAS (Folder Minitasks/tasks). There will be a submission guideline in the Minitasks folder.

Session 2: 18.10.2023
Importance of Practical Skills in Linguistics & R basics (1)

Importance of Practical Skills

What are Practical Linguistic Skills?



In your view, what counts as **a practical skill** in different fields of linguistics?

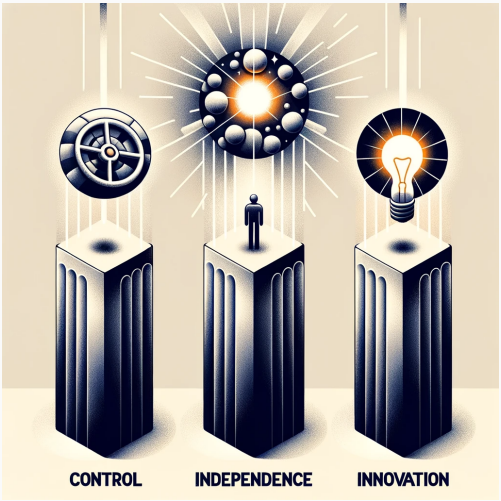
Practical Linguistic Skills (to name a few)

1. **Transcription Skills** phonetic and orthographic transcription
2. **Fieldwork Skills** elicitation techniques, recording and audio equipment
3. **Language Documentation** archiving, metadata creation
4. **Tech Proficiency** corpus & speech analysis tools, programming & libraries, machine learning algorithms
5. **Data Analysis** tools and techniques for qualitative and quantitative analysis
6. **Experimental Skills** experimental design, eye-tracking, neuroimaging (EEG, fMRI)
7. **Ethical Research Practices** informed consent, anonymization, GDPR rules, transparency and open science
8. **Other Skills** academic writing, presentation skills, teamwork, open-mindedness, emotional intelligence, organizational skills, cultural sensitivity

Why Practical Skills Matter for Linguistic Research

1. Modern linguistics is data-driven and relies heavily on empirical data for validation.
2. Linguistics has a very interdisciplinary nature. Practical skills are often the common language.
3. Increased growth and complexity of data necessitates computational skills for analysis.
4. Practical skills ensure the data is reliable and the research is valid.
5. Practical skills enable researchers to better collaborate, share their data, and contribute to open science.

Why Practical Skills Matter for a Researcher



1. Practical skills are increasingly required in job postings and ensure better career opportunities, both within academia and industry.
2. Proficiency in practical skills is empowering since it (1) gives you control over every stage of the research, (2) decreases your reliance on others to do the work for you, and (3) gives you the confidence to try out new methods and technologies.

Practical Skills Introduced in this Course

Programming & Data Management

- Basics of R, tidyverse, R markdown
- Data management fundamentals
- File operations in R
- Data cleaning and preprocessing

Packages: `base R`, `rmarkdown`, `tidyverse`

Text Processing & Analysis

- Regular expressions & string operations: `stringr`
- Textual data types (XML, HTML, JSON): `xml2`, `jsonlite`
- Corpus analysis: `tidytext`, `quanteda`
- Exploratory data analysis: `summarytools`

Practical Skills Introduced in this Course

Linguistic Tools & Annotation

- Usage of the Sketch Engine for linguistic queries
- Manual and automatic annotation techniques: `INCEpTION`, `udpipe`, `spaCy`, `NLTK`

Data Acquisition & Visualization

- Acquiring data from the web: `rvest`
- Data visualization techniques: `ggplot`

Resources for In-depth Learning

Aim: This course focuses on highlighting areas for improvement, not on achieving practical proficiency (due to time limitation).

How to become proficient?

1. Watch YouTube videos.
2. Adopt a "Challenge Accepted" attitude and work on real datasets.
3. Use Google and Stack Overflow.
4. Use Cheatsheets: `https://posit.co/resources/cheatsheets/`
5. Read relevant books and blogposts.
   - R for Data Science (2e) `https://r4ds.hadley.nz/`
   - Text Mining with R: A Tidy Approach `https://www.tidytextmining.com/`
   - R-bloggers: `https://www.r-bloggers.com/`

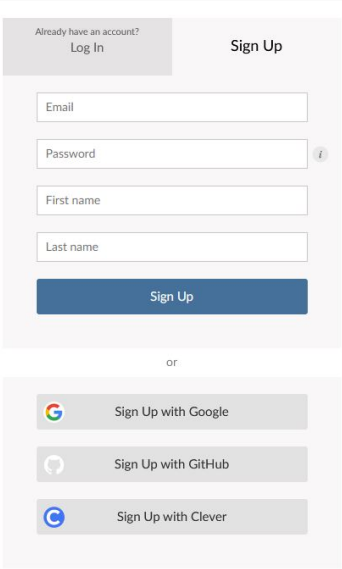
R basics (1)

R and RStudio

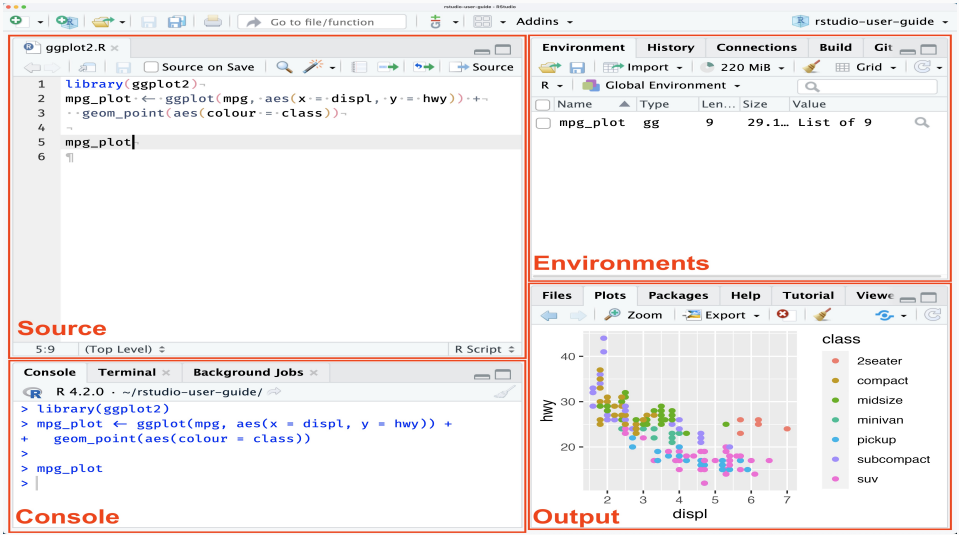
- R is a free, open-source programming language used for data mining, statistical analysis, data visualization, and machine learning.
- RStudio is an integrated development environment (IDE) designed to help you be more productive in your daily data science work.
  - Desktop version: `https://posit.co/download/rstudio-desktop/`
  - Cloud version: `https://posit.cloud/`

Posit Cloud

- Posit Cloud lets you access R and Python in your browser – no installation or complex configuration required.
- Great for sharing and collaboration



RStudio Environment



RStudio Panes

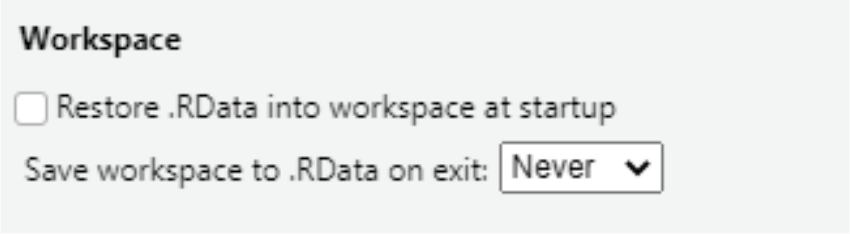
1. **Source or script editor** write and run R code
2. **Console** execute commands and see output
3. **Environment** displays temporary R objects (e.g., variables and dataframes) as created during that R session.
4. **Output pane** displays the plots, tables, or HTML outputs of executed code along with files saved to disk.

Changing Some Global Options in R

Go to the **Tools** menu. Select **Global Options**.

Go to the **General** tab:

1. Change the *Save workspace to .RData on exit* to **Never**.
2. **Uncheck** the *Restore .RData into workspace at start*.



Install & Load Packages

- Packages extend R's functionality.
- Install packages using the **install.packages()** function.
- Load packages into the current session using the **library()** function.

In R, packages only need to be installed once. After installation, you can load them into your script using the library() function.

Install & Load Packages (Example)

```
install.packages("tidyverse")
library(tidyverse)
# To comment (out) one line, use a hashtag sign (#)
# To comment multiple lines: on Windows, highlight the text and
    use CTRL + SHIFT + C. For macOS, use command + SHIFT + C.
#install.packages("tidyverse")
```

YOUR TASK

install and load the package **strex** in R.

Accessing Help Documentation

- To access all functions in a package: **help(package = "strex")**
- To access the documentation of a command:
  1. help("command name"), e.g., help("str_before_first")
  2. **?**command name, e.g., ?str_before_first
- In case you remember the command name partially, **apropos("keyword")** will list all the functions with the specific "keyword" in them, e.g., apropos("last").

```
help(package = "strex") #list of all functions of a package
help("str_after_first") #how to use a specific command
?str_after_first #how to use a specific command
apropos("last") #in case you do not remember the full name of a
    command
```

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)
)
```

Understanding the <− Operator in R

- The <− symbol is the assignment operator in R.
- It is used to assign a value to a variable.
- It is composed of a "less than" symbol and a "hyphen/minus" symbol.

```
x <- 10 #Assigning a numeric value to a variable
my_string <- "Hello, world!" #Assigning a character string to a
    variable
my_vector <- c(1, 2, 3, 4, 5) #Assigning a vector to a variable
my_df <- data.frame(name = c("Alice", "Bob"), score = c(85, 90)
   ) #Assigning a data frame to a variable
```

Note: We usually do not create data frames from scratch; instead, we import data into R from other sources.

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

Looking at Other Dataframes

R has a lot of built-in dataset in the package **datasets**. This is an inherent base R package.

- Use **data()** to load a dataframe from the datasets package, e.g., data("iris")

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 and 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.

QUESTIONS?