

# Phonetic fieldwork and experiments with the phonfieldwork package for R

George Moroz

Linguistic Convergence Laboratory, NRU HSE

08 August 2020, Grupo de estadística para el estudio del lenguaje

Presentation is available here: [tinyurl.com/y6lf5ch4](https://tinyurl.com/y6lf5ch4)



# Outline of the talk

Introduction

Installation of the package

Creating your presentation

Data renaming

Data merging

Data annotation

Data extraction

Data visualization

Creating a data viewer – template for the data sharing

Reading data from different linguistic sources

Get help and cite

## Most phonetic research consists of the following steps:

1. Formulate a research question. Think of what kind of data is necessary to answer this question, what is the appropriate amount of data, what kind of annotation you will do, what kind of statistical models and visualizations you will use, etc.
2. Create a list of stimuli.
3. Elicit list of stimuli from speakers who signed an Informed Consent statement, agreeing to participate in the experiment to be recorded on audio and/or video.
4. Annotate the collected data.
5. Extract the information from annotated data.
6. Create visualizations.
7. Evaluate your statistical models.
8. Report your results.
9. Publish your data.

## Most phonetic research consists of the following steps:

1. Formulate a research question. Think of what kind of data is necessary to answer this question, what is the appropriate amount of data, what kind of annotation you will do, what kind of statistical models and visualizations you will use, etc.
2. Create a list of stimuli.
3. Elicit list of stimuli from speakers who signed an Informed Consent statement, agreeing to participate in the experiment to be recorded on audio and/or video.
4. Annotate the collected data.
5. Extract the information from annotated data.
6. Create visualizations.
7. Evaluate your statistical models.
8. Report your results.
9. Publish your data.

The `phonfieldwork` package is created for helping with items 3, 4 (partially), 5, 6, and 9.

## Why/when do you need the phonfieldwork package?

These ideal plan hides some technical subtasks:

- creating a presentation for elicitation task
- renaming and concatenating multiple sound files
- automatic annotation in Praat TextGrids [[Boersma and Weenink 2019](#)]
- creating a searchable .html table with annotations, spectrograms and ability to hear sound
- converting multiple formats (Praat, ELAN [[Brugman et al. 2004](#)] and EXMARaLDA [[Schmidt and Wörner 2009](#)])

## Why/when do you need the phonfieldwork package?

These ideal plan hides some technical subtasks:

- creating a presentation for elicitation task
- renaming and concatenating multiple sound files
- automatic annotation in Praat TextGrids [[Boersma and Weenink 2019](#)]
- creating a searchable .html table with annotations, spectrograms and ability to hear sound
- converting multiple formats (Praat, ELAN [[Brugman et al. 2004](#)] and EXMARaLDA [[Schmidt and Wörner 2009](#)])

All of these tasks can be solved by a mixture of different tools:

- any programming language can handle automatic file renaming
- Praat contains scripts for concatenating and renaming files

# Why/when do you need the phonfieldwork package?

These ideal plan hides some technical subtasks:

- creating a presentation for elicitation task
- renaming and concatenating multiple sound files
- automatic annotation in Praat TextGrids [[Boersma and Weenink 2019](#)]
- creating a searchable .html table with annotations, spectrograms and ability to hear sound
- converting multiple formats (Praat, ELAN [[Brugman et al. 2004](#)] and EXMARaLDA [[Schmidt and Wörner 2009](#)])

All of these tasks can be solved by a mixture of different tools:

- any programming language can handle automatic file renaming
- Praat contains scripts for concatenating and renaming files

phonfieldwork provides a functionality that will make it easier to solve those tasks independently of any additional tools. You can also compare the functionality with other packages: rPraat [[Bořil and Skarnitzl 2016](#)], textgRid [[Reidy 2016](#)], pympfi [[Lubbers and Torreira 2013](#)]

## Philosophy of the phonfieldwork package

- each stimulus is a separate file
- researcher carefully listens to consultants to make sure that they are producing the kind of speech they wanted
- in case a speaker does not produce clear repetitions, researcher ask them to repeat the task

There are some phoneticians who prefer to record everything for language documentation purposes. I think that should be a separate task. If you insist on recording everything, it is possible to run two recorders at the same time: one could run during the whole session, while the other is used to produce small audio files. You can also use special software to record your stimuli automatically on a computer (e.g. PsychoPy [[Peirce et al. 2009](#)]).

# Outline of the talk

Introduction

Installation of the package

Creating your presentation

Data renaming

Data merging

Data annotation

Data extraction

Data visualization

Creating a data viewer – template for the data sharing

Reading data from different linguistic sources

Get help and cite

## Install phonfieldwork

phonfieldwork is an R package, so you need to install R, RStudio (optional) or use [rstudio.cloud](#). There are two possibilities for installing package in R:

- official version from CRAN

```
install.packages("phonfieldwork")
```

- development version from GitHub

```
devtools::install_github("agricolamz/phonfieldwork")
```

## Install phonfieldwork

phonfieldwork is an R package, so you need to install R, RStudio (optional) or use [rstudio.cloud](#). There are two possibilities for installing package in R:

- official version from CRAN

```
install.packages("phonfieldwork")
```

- development version from GitHub

```
devtools::install_github("agricolamz/phonfieldwork")
```

Since this package is under [rOpenSci review](#) there is a chance that in couple months the address could be changed to `ropensci/phonfieldwork`, and documentation page will be moved from [agricolamz.github.io/phonfieldwork](https://agricolamz.github.io/phonfieldwork) to [docs.ropensci.org/phonfieldwork](https://docs.ropensci.org/phonfieldwork).

```
library("phonfieldwork")
packageVersion("phonfieldwork") # Unreleased version
```

```
## [1] '0.0.8'
```

# Outline of the talk

Introduction

Installation of the package

**Creating your presentation**

Data renaming

Data merging

Data annotation

Data extraction

Data visualization

Creating a data viewer – template for the data sharing

Reading data from different linguistic sources

Get help and cite

## Make a list of your stimuli

There are several ways to enter information about a list of stimuli into R:

- list them with the `c()` command

```
my_stimuli <- c("tip", "tap", "top")
```

- read from .csv file with the `read.csv()` function:

```
my_stimuli_df <- read.csv("my_stimuli_df.csv")
```

- read from .xls or .xlsx file with the `read_xls()` or `read_xlsx()` function from the `readxl` package (run `install.packages("readxl")` in case you don't have it installed):

```
library("readxl")
my_stimuli_df <- read_xlsx("my_stimuli_df.xlsx")
```

## Create a presentation based on a list of stimuli

```
create_presentation(stimuli = my_stimuli, # it is "tip" "tap" "top"  
                    output_file = "first_example",  
                    output_dir = "data/")
```

Here is the result.

# Create a presentation based on a list of stimuli

```
create_presentation(stimuli = my_stimuli, # it is "tip" "tap" "top"
                    output_file = "first_example",
                    output_dir = "data/")
```

Here is [the result](#).

It is also possible to use images (or .gif) as a stimuli:

```
my_image <- system.file("extdata", "r-logo.png", package = "phonfieldwork")
my_image
```

```
## [1] "/home/agricolamz/R/x86_64-pc-linux-gnu-library/4.0/phonfieldwork/extdata/r-
logo.png"
```

```
create_presentation(stimuli = c("rzeka", "drzewo", my_image),
                    external = 3,
                    output_file = "second_example",
                    output_dir = "data/")
```

Here is [the result](#).

# Outline of the talk

Introduction

Installation of the package

Creating your presentation

## Data renaming

Data merging

Data annotation

Data extraction

Data visualization

Creating a data viewer – template for the data sharing

Reading data from different linguistic sources

Get help and cite

## Obtained data

After collecting data and removing soundfiles with unsuccessful elicitations, one could end up with the following structure:

```
## └── s1
##   ├── 01.wav
##   ├── 02.wav
##   └── 03.wav
## └── s2
##   ├── 01.wav
##   ├── 02.wav
##   └── 03.wav
```

## Rename collected data

```
rename_soundfiles(stimuli = my_stimuli, # it is "tip" "tap" "top"  
                  prefix = "s1_",  
                  path = "data/s1/")
```

```
## └── s1  
##   ├── backup  
##   │   ├── 01.wav  
##   │   ├── 02.wav  
##   │   └── 03.wav  
##   ├── s1_tap.wav  
##   ├── s1_tip.wav  
##   └── s1_top.wav  
## └── s2  
##   ├── 01.wav  
##   ├── 02.wav  
##   └── 03.wav
```

## Rename collected data

```
rename_soundfiles(stimuli = my_stimuli, # it is "tip" "tap" "top"
                  prefix = paste0(1:3, "_"),
                  suffix = "_s2",
                  path = "data/s2/",
                  backup = FALSE)
```

```
## └── s1
##   ├── backup
##   │   ├── 01.wav
##   │   ├── 02.wav
##   │   └── 03.wav
##   ├── s1_tap.wav
##   ├── s1_tip.wav
##   └── s1_top.wav
## └── s2
##   ├── 1_tip_s2.wav
##   ├── 2_tap_s2.wav
##   └── 3_top_s2.wav
```

## Rename collected data

Sometimes it is better to keep an order that will deal with the computer sorting:

```
add_leading_symbols(1:105)
```

```
## [1] "001" "002" "003" "004" "005" "006" "007" "008" "009" "010" "011" "012"  
## [13] "013" "014" "015" "016" "017" "018" "019" "020" "021" "022" "023" "024"  
## [25] "025" "026" "027" "028" "029" "030" "031" "032" "033" "034" "035" "036"  
## [37] "037" "038" "039" "040" "041" "042" "043" "044" "045" "046" "047" "048"  
## [49] "049" "050" "051" "052" "053" "054" "055" "056" "057" "058" "059" "060"  
## [61] "061" "062" "063" "064" "065" "066" "067" "068" "069" "070" "071" "072"  
## [73] "073" "074" "075" "076" "077" "078" "079" "080" "081" "082" "083" "084"  
## [85] "085" "086" "087" "088" "089" "090" "091" "092" "093" "094" "095" "096"  
## [97] "097" "098" "099" "100" "101" "102" "103" "104" "105"
```

So it is better to use the result of `add_leading_symbols()` as a prefix during the renaming of a huge amount of files.

## Get sound duration

Sometimes it is useful to get information about sound duration:

```
get_sound_duration("data/s1/s1_tap.wav")
```

```
##           file  duration
## 1 s1_tap.wav 0.4299093
```

```
get_sound_duration(sounds_from_folder = "data/s2/")
```

```
##           file  duration
## 1 1_tip_s2.wav 0.5866440
## 2 2_tap_s2.wav 0.5343991
## 3 3_top_s2.wav 0.6650113
```

# Outline of the talk

Introduction

Installation of the package

Creating your presentation

Data renaming

**Data merging**

Data annotation

Data extraction

Data visualization

Creating a data viewer – template for the data sharing

Reading data from different linguistic sources

Get help and cite

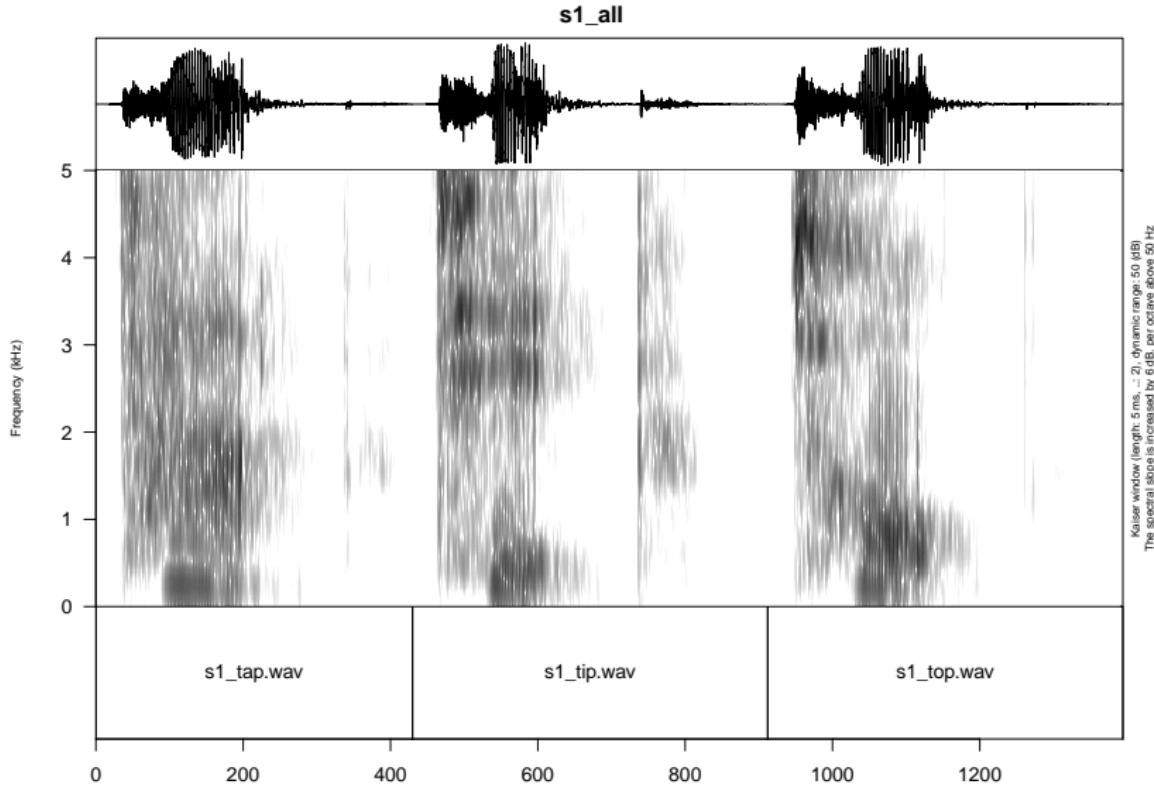
## Merge all data together

After all the files are renamed, you can merge them into one:

```
concatenate_soundfiles(path = "data/s1/",  
                      result_file_name = "s1_all")
```

```
## data/s1  
## └── backup  
##   ├── 01.wav  
##   ├── 02.wav  
##   └── 03.wav  
## └── s1_all.TextGrid  
## └── s1_all.wav  
## └── s1_tap.wav  
## └── s1_tip.wav  
## └── s1_top.wav
```

# Merge all data together



# Outline of the talk

Introduction

Installation of the package

Creating your presentation

Data renaming

Data merging

**Data annotation**

Data extraction

Data visualization

Creating a data viewer – template for the data sharing

Reading data from different linguistic sources

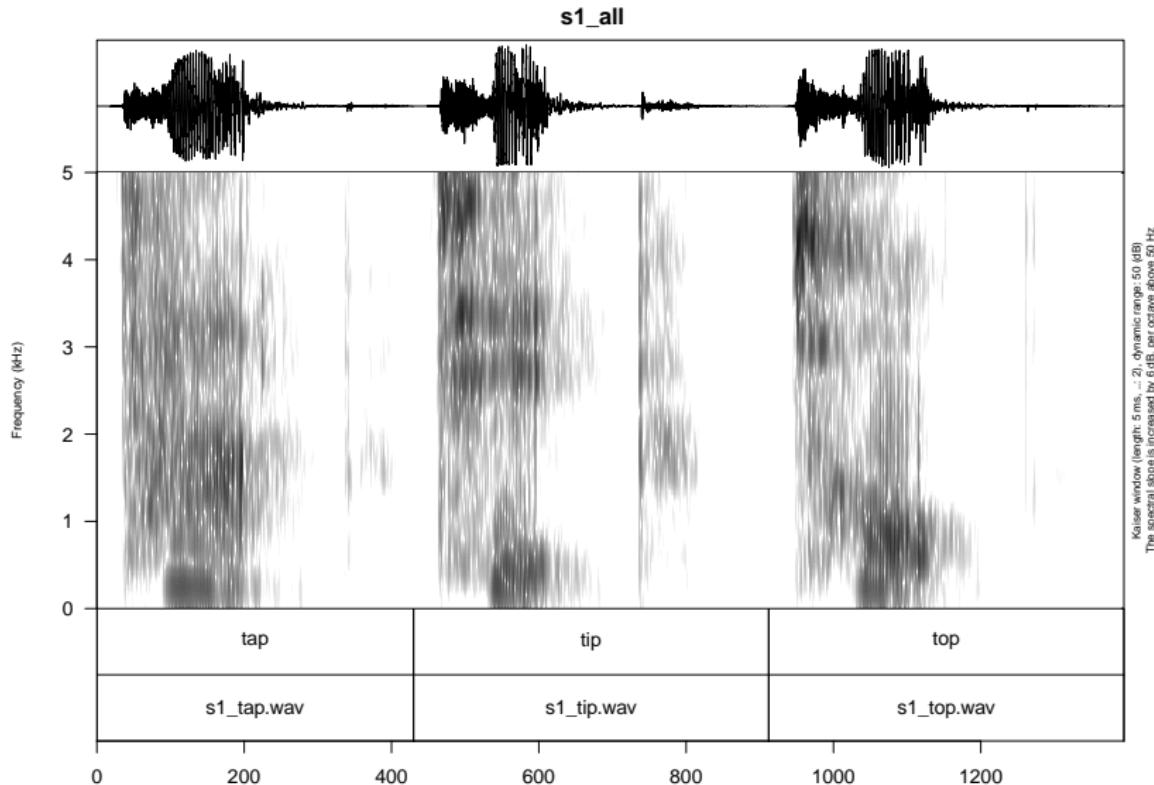
Get help and cite

## Annotate your data

It is possible to create annotation in advance (since file concatenation is made according to files sorted on the computer I use the `sort()` function in order to make correct annotation):

```
annotate_textgrid(annotation = sort(my_stimuli),  
                  textgrid = "data/s1/s1_all.TextGrid")
```

# Annotate your data

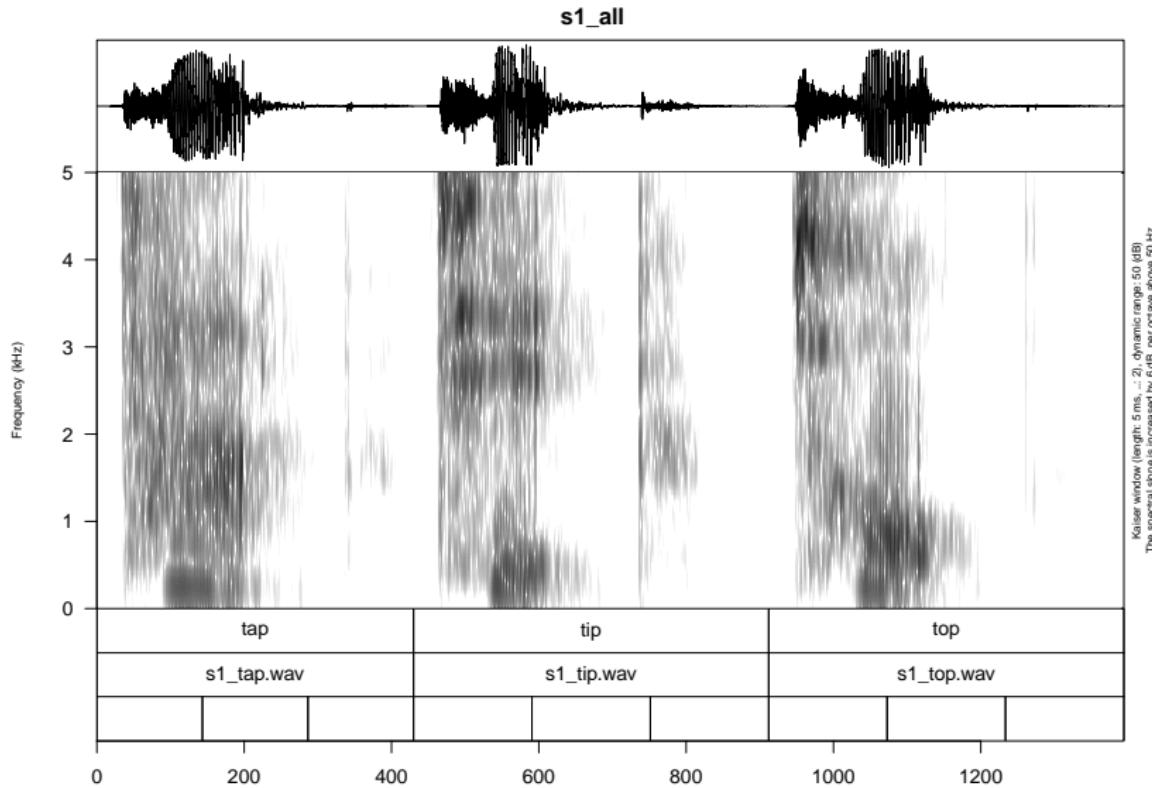


## Create subannotation

Imagine that we are interested in annotation of vowels. The most common solution will be open Praat and create new annotations. But it is also possible to create them in advance. The idea that you choose some baseline tier that later will be automatically cutted into smaller pieces on the other tier.

```
create_subannotation(textgrid = "data/s1/s1_all.TextGrid",
                     tier = 1, # this is a baseline tier
                     n_of_annotations = 3) # how many empty annotations per unit?
```

# Annotate subannotation in advance



## Annotate subannotation in advance

Now we can annotate created tier:

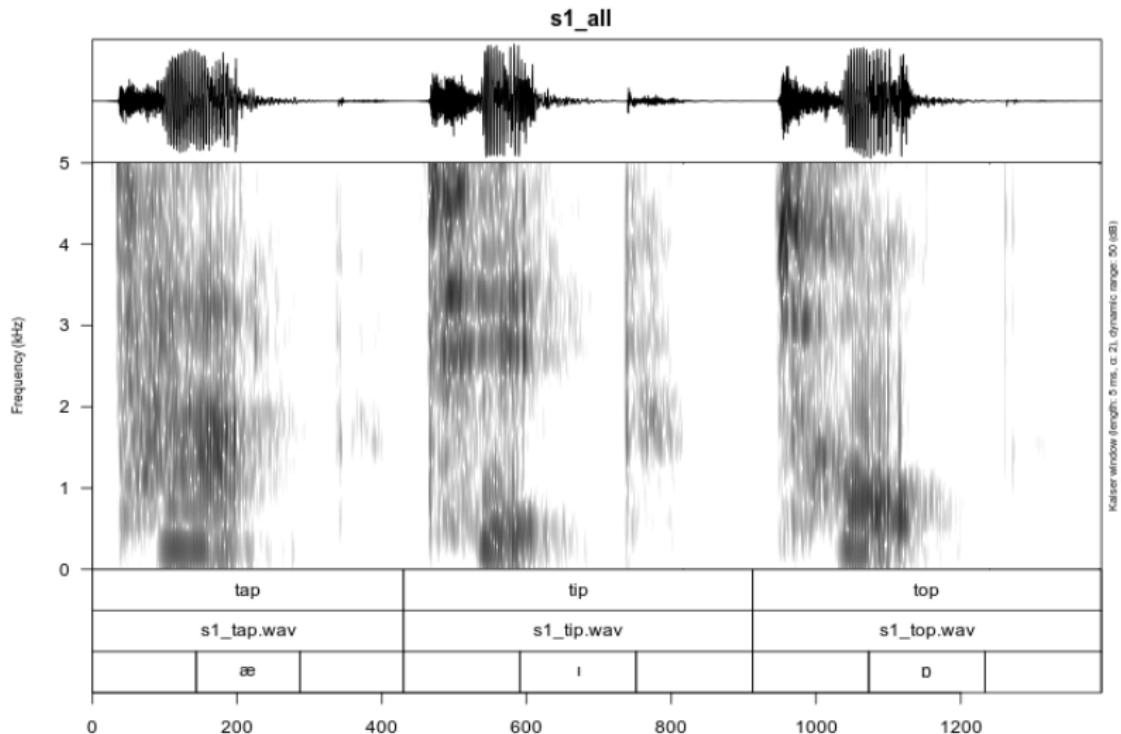
```
annotate_textgrid(annotation = c("", "æ", "", "", "i", "", "", "ɒ", ""),
                  textgrid = "data/s1/s1_all.TextGrid",
                  tier = 3,
                  backup = FALSE)
```

List annotations by hand is a boring task, so if you have a prepared list of annotations, the merge could be done with the following code:

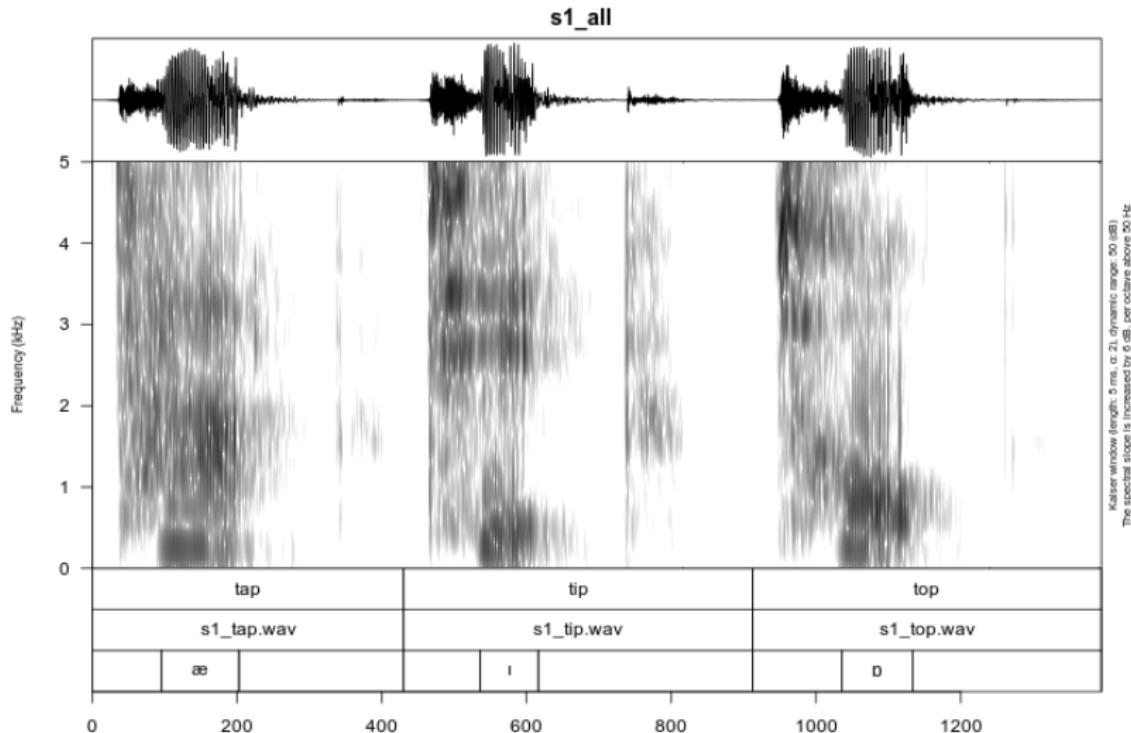
```
vowels <- c("æ", "i", "ɒ")
unlist(lapply(vowels, function(x){c("", x, "")}))
```

```
## [1] ""  "æ"  ""  ""  "i"  ""  ""  "ɒ"  ""
```

# Create subannotation



# The only thing left is to move annotation boundaries



# Outline of the talk

Introduction

Installation of the package

Creating your presentation

Data renaming

Data merging

Data annotation

**Data extraction**

Data visualization

Creating a data viewer – template for the data sharing

Reading data from different linguistic sources

Get help and cite

## Data viewer

Sound viewer (here is an [example](#)) is a useful tool that combine together your annotations and make it searchable. It is also produce a ready to go .html file that could be uploaded on the server (e. g. to Github Pages) and be available for anyone in the world.

In order to do that we need:

- separate folder with soundfiles
- separate folder with spectorgrams (optional)
- `data.frame` with data about utterances or speakers

## Data extraction

First, it is important to create a folder where all of the extracted files will be stored:

```
dir.create("data/s1/s1_sounds")
```

It is possible extract to extract all annotated files based on an annotation tier:

```
extract_intervals(file_name = "data/s1/s1_all.wav",
                  textgrid = "data/s1/s1_all.TextGrid",
                  tier = 3,
                  path = "data/s1/s1_sounds/",
                  prefix = "s1_")
```

## Data extraction

After those commands, one could end up with the following structure:

```
## data/s1
##   └── backup
##     ├── 01.wav
##     ├── 02.wav
##     └── 03.wav
##   └── s1_all.TextGrid
##   └── s1_all.wav
##   └── s1_sounds
##     ├── 1_s1_a.wav
##     ├── 2_s1_i.wav
##     └── 3_s1_o.wav
##   └── s1_tap.wav
##   └── s1_tip.wav
##   └── s1_top.wav
```

# Outline of the talk

Introduction

Installation of the package

Creating your presentation

Data renaming

Data merging

Data annotation

Data extraction

**Data visualization**

Creating a data viewer – template for the data sharing

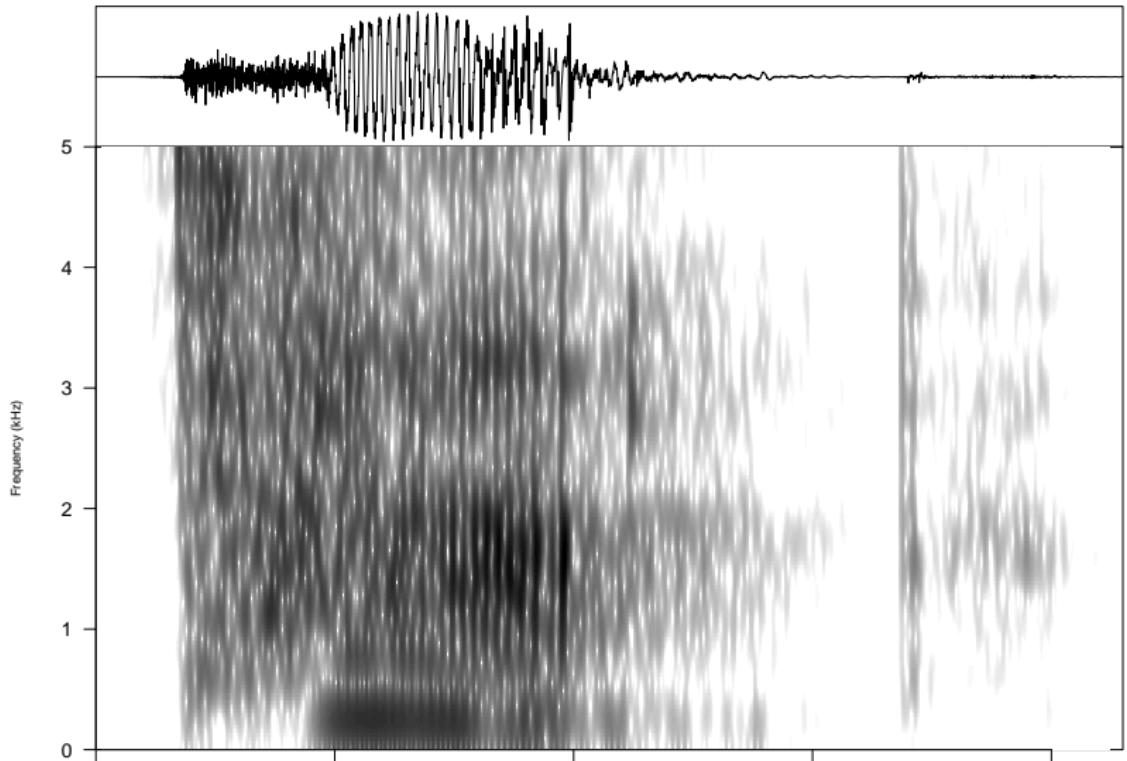
Reading data from different linguistic sources

Get help and cite

# Sound visualization in phonfieldwork

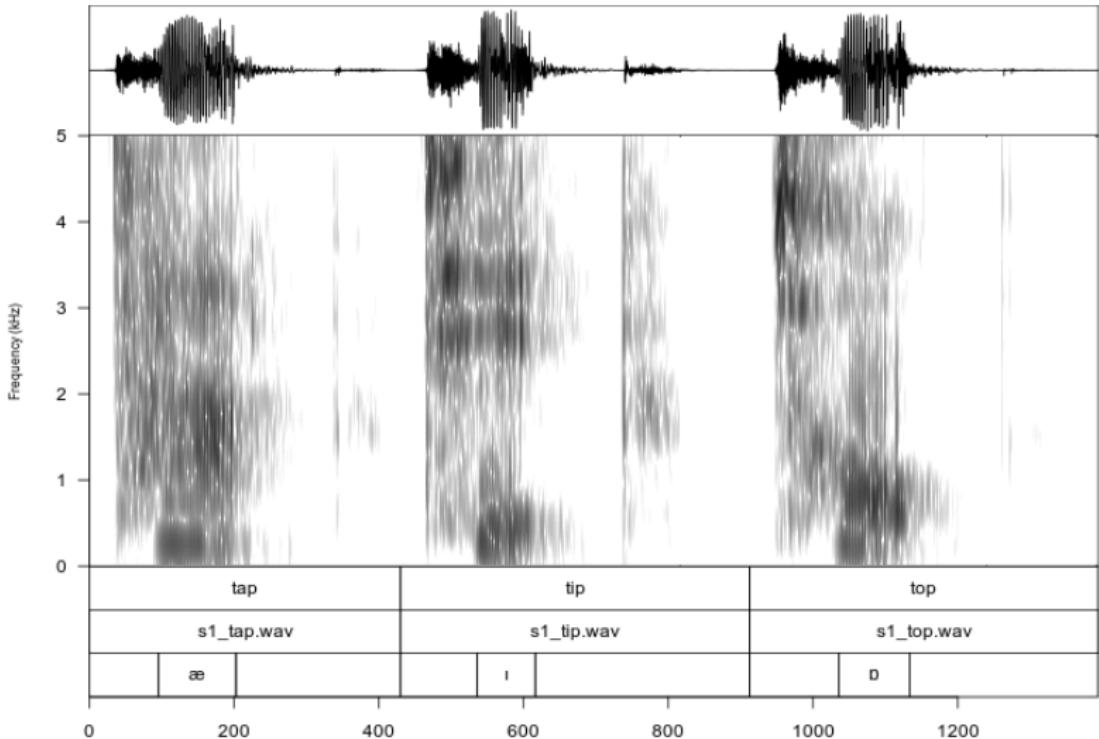
The easiest way to visualise sound in phonfieldwork:

```
draw_sound(file_name = "data/s1/s1_tap.wav")
```



# Sound visualization in phonfieldwork

```
draw_sound(file_name = "data/s1/s1_all.wav",
           annotation = "data/s1/s1_all.TextGrid")
```



# Create multiple spectrograms

```
draw_sound(sounds_from_folder = "data/s1/s1_sounds/",  
          pic_folder_name = "s1_pics")
```

```
## data/s1  
##   └── backup  
##       ├── 01.wav  
##       ├── 02.wav  
##       └── 03.wav  
##   ├── s1_all.TextGrid  
##   ├── s1_all.wav  
##   ├── s1_pics  
##       ├── 1_s1_a.png  
##       ├── 2_s1_i.png  
##       └── 3_s1_o.png  
##   └── s1_sounds  
##       ├── 1_s1_a.wav  
##       ├── 2_s1_i.wav  
##       └── 3_s1_o.wav
```

# Outline of the talk

Introduction

Installation of the package

Creating your presentation

Data renaming

Data merging

Data annotation

Data extraction

Data visualization

**Creating a data viewer – template for the data sharing**

Reading data from different linguistic sources

Get help and cite

## Create a viewer

- separate folder with soundfiles
- separate folder with spectrograms (optional)
- → `data.frame` with data about utterances or speakers

```
df <- data.frame(word = c("tap", "tip", "top"),
                  sounds = c("ə", "ɪ", "ɒ"))
```

```
df
```

```
##   word sounds
## 1  tap      ə
## 2  tip      ɪ
## 3  top      ɒ
```

```
create_viewer(audio_dir = "data/s1/s1_sounds/",
              picture_dir = "data/s1/s1_pics/",
              table = df,
              output_dir = "data/s1/",
              output_file = "stimuli_viewer")
```

# Outline of the talk

Introduction

Installation of the package

Creating your presentation

Data renaming

Data merging

Data annotation

Data extraction

Data visualization

Creating a data viewer – template for the data sharing

Reading data from different linguistic sources

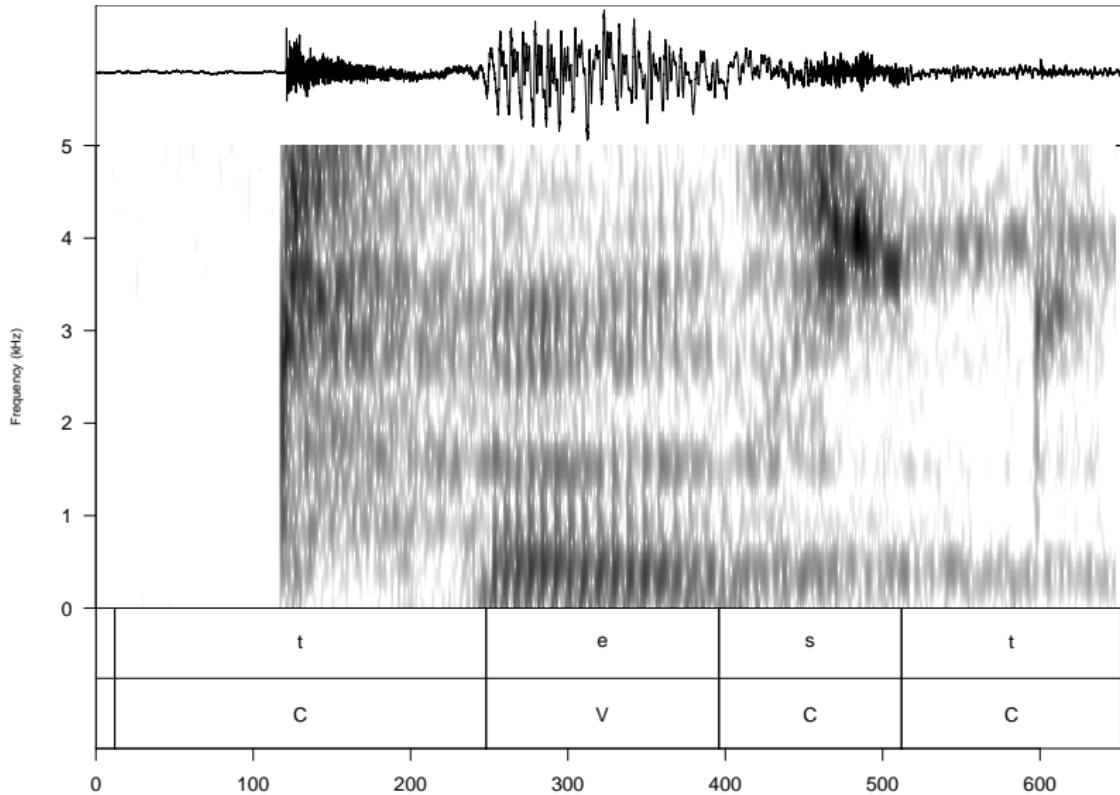
Get help and cite

## Read linguistic files into R

- `textgrid_to_df()` (Praat)
- `eaf_to_df()` (ELAN)
- `exb_to_df()` (EXMARaLDA)
- `srt_to_df()` (subtitle file)
- `audacity_to_df()` (Audacity)
- `flextext_to_df()` (FieldWorks)

# Read linguistic files into R

```
draw_sound(file_name = "data/test.wav",  
annotation = eaf_to_df("data/test.eaf"))
```



# Outline of the talk

Introduction

Installation of the package

Creating your presentation

Data renaming

Data merging

Data annotation

Data extraction

Data visualization

Creating a data viewer – template for the data sharing

Reading data from different linguistic sources

**Get help and cite**

## Get help and cite

You can always write an email or open an [issue on GitHub](#), asking some questions.

The most recent citation information is available with this command:

```
citation("phonfieldwork")
```

```
##  
## Moroz G (2020). _Phonetic fieldwork and experiments with phonfieldwork  
## package_. <URL: https://CRAN.R-project.org/package=phonfieldwork>.  
##  
## A BibTeX entry for LaTeX users is  
##  
## @Manual{,  
##   title = {Phonetic fieldwork and experiments with phonfieldwork package},  
##   author = {George Moroz},  
##   year = {2020},  
##   url = {https://CRAN.R-project.org/package=phonfieldwork},  
## }
```

## References

- Boersma, P. and Weenink, D. (2019). Praat: doing phonetics by computer (version 5.3.51)[computer program] version 6.0.25, retrieved 15 november 2019 from <http://www.praat.org/>.
- Bořil, T. and Skarnitzl, R. (2016). Tools rpraat and mpraat. In Sojka, P., Horák, A., Kopeček, I., and Pala, K., editors, *Text, Speech, and Dialogue: 19th International Conference, TSD 2016, Brno, Czech Republic, September 12-16, 2016, Proceedings*, pages 367–374, Cham. Springer International Publishing.
- Brugman, H., Russel, A., and Nijmegen, X. (2004). Annotating Multi-media/Multi-modal Resources with ELAN. In *LREC*.
- Lubbers, M. and Torreira, F. (2013). Pympi-ling: a Python module for processing ELANs EAF and Praats TextGrid annotation files.

## References

- Peirce, J. W., Gray, J. R., Simpson, S., MacAskill, M. R., Höchenberger, R., Sogo, H., Kastman, E., and Lindeløv, J. (2009). Psychopy2: experiments in behavior made easy. *Behavior Research Methods*, 51(1):195–203.
- Reidy, P. (2016). *textgridRid: Praat TextGrid Objects in R*. R package version 1.0.1.
- Schmidt, T. and Wörner, K. (2009). Exmaralda—creating, analysing and sharing spoken language corpora for pragmatic research. *Pragmatics. Quarterly Publication of the International Pragmatics Association (IPrA)*, 19(4):565–582.