Quantitative Drama Analytics

Part 2: lab session

January 7, 2020

Your setup

- https://quadrama.github.io/rstudio
  Virtuelle Maschine, die bei Microsoft Azure läuft
- Logins:
  capulet romeo juliet montague balthasar
  peter tybalt mercutio ophelia hamlet
  horatio francisco laertes solanio arragon
  nerissa lorenzo antonio bianca katherine
  oliver corin dennis celia phoebe
  lysander helena hermia oberon theseus
  volumnia brutus virgilia valeria sicinius

What you really need to know about R

R Basics

- R is a programming language
  - Mostly used for statistical data analysis (“data science”)
  - First version: 1993, current stable release: 3.6
  - Website
  - Open source
- Three important concepts we need to talk about
  - Objects/Types
  - Variables
  - Functions

R Basics

Objects and Types

- Objects live in the computer memory (or on disk)
- Objects represent the things we want to analyse (e.g., dramatic texts, words, or numbers)
- An object has one or more types

R Basics

Objects and Types

- Objects live in the computer memory (or on disk)
- Objects represent the things we want to analyse (e.g., dramatic texts, words, or numbers)
- An object has one or more types
- The type of an object determines what we can do with it
    - E.g., a knife allows other operations than a fork

R Basics

Objects and Types

- Objects live in the computer memory (or on disk)
- Objects represent the things we want to analyse (e.g., dramatic texts, words, or numbers)
- An object has one or more types
- The type of an object determines what we can do with it
  - E.g., a knife allows other operations than a fork
- Types: Numbers, character sequences ('strings'), lists, tables, …
  - Numbers allow arithmetic operations
    - E.g., summation: `sum(3,5)` (evaluates to 8, equivalent to 3+5)
  - Character sequences allow character-based operations
    - E.g., conversion to lower case: `tolower("ABC")` (evaluates to `"abc"`)

R Basics

Objects and Types

- Objects live in the computer memory (or on disk)
- Objects represent the things we want to analyse (e.g., dramatic texts, words, or numbers)
- An object has one or more types
- The type of an object determines what we can do with it
  - E.g., a knife allows other operations than a fork
- Types: Numbers, character sequences ('strings'), lists, tables, …
  - Numbers allow arithmetic operations
    - E.g., summation: `sum(3,5)` (evaluates to 8, equivalent to 3+5)
  - Character sequences allow character-based operations
    - E.g., conversion to lower case: `tolower("ABC")` (evaluates to `"abc"`)
- "evalutes to": result of the operation/function

R Basics

Objects and Types

Type	Example	Description
Numeric	`5`	A numeric value
Character	`"Heidelberg"`	A sequence of characters (note the double quotes!)
Logical	`TRUE/FALSE`	A truth value
Vector	`c(5,4,1)`	Sequence of objects *of the same type*
List	`list(5,"Hd",TRUE)`	Sequence of objects
Matrix		Table of objects *of the same type*
Data frame		Table of objects

R Basics

Objects and Types

In R, everything is a vector!

- Entering 5 creates a numeric vector of length 1
- Entering `"Bla"` creates a character vector of length 1

(In this way, R is different from other programming languages)

```
5
# Creates a vector consisting of the numbers 1 to 50
1:50
```

R Basics

Variables

- We usually do not interact with the objects directly
  - Because they are not known in advance (but loaded from files)
- Variables
  - A way to *name* objects
  - Used as a placeholder for objects
  - The actual operation takes place on the objects (R takes care of this)
- Creating a variable a: `a <- 3` (think of this as an arrow)

```
> a <- 3
> b <- 5
> a + b
[1] 8
>
```

R Basics

Functions

- “Mini programs”: A collection of instructions that you can use as a single instruction
- Input: Functions take *arguments* as input
- Output: Functions return an object (that stores the result of the instructions)

R Basics

Functions

- “Mini programs”: A collection of instructions that you can use as a single instruction
- Input: Functions take *arguments* as input
- Output: Functions return an object (that stores the result of the instructions)
- Functions have a name (typically lower case) and can be reognized by the round parentheses
  `function(argument1, argument2, argument3, ...)`
- The return value of a function can be stored in a variable
  `variable <- function(arg1, arg2, ...)`

R Basics

Functions

- “Mini programs”: A collection of instructions that you can use as a single instruction
- Input: Functions take *arguments* as input
- Output: Functions return an object (that stores the result of the instructions)
- Functions have a name (typically lower case) and can be reognized by the round parentheses
  `function(argument1, argument2, argument3, ...)`
- The return value of a function can be stored in a variable
  `variable <- function(arg1, arg2, ...)`
- Some functions not only return a value, but also do something (e.g., display a plot)
- Pipeline: Multiple functions operating in succession

R Basics

Functions

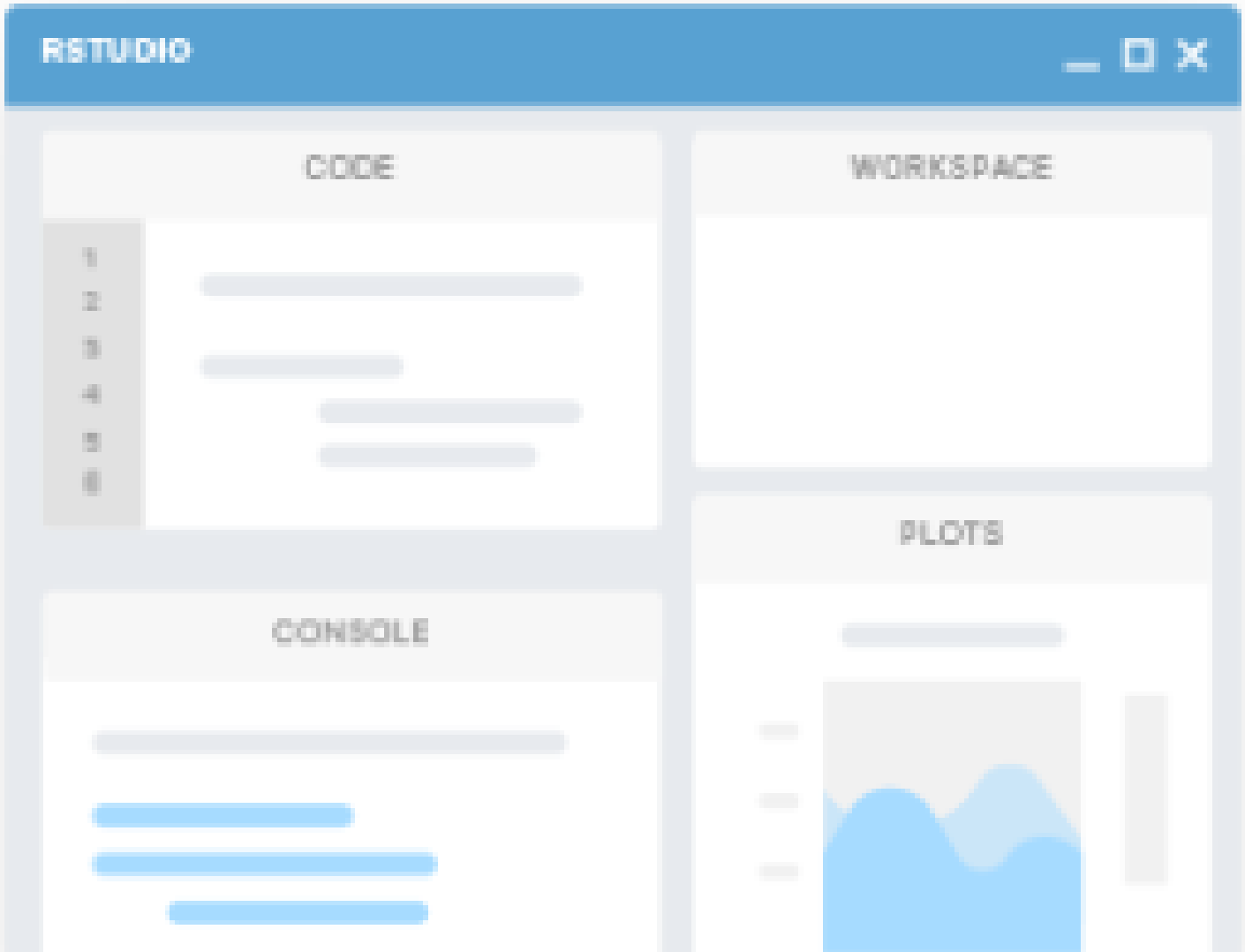
```
sum(5,1)          # 5 + 1 is only an abbreviation
s <- sum(5,1)     # stores the result in a
                  # variable, no output
s                 # prints the value of the variable
s <- 7            # overwrites the previous value of
                  # the variable
s <- sum(s,3)     # overwrites the value of the
                  # variable
```

What is the value of s now?

RStudio

RStudio

- An integrated development environment (IDE) for R
- Capable workbench for data analysis



RStudio

Four Panes

- Console: Where you enter R code and get the result immediately
- Environment: Shows the objects currently in memory
- Plots: Shows plots
- Editor/Code: Allows editing R code and inspecting tables

We will focus on the console and plot area

DramaAnalysis

Outline

- Introduction/Installation and Overview
- Three areas for you to play with
  1. Global character statistics
  2. Word fields
  3. Copresence and network analysis

Introduction

- R Package: A collection of functions and/or data sets
- Function: Mini program
- DramaAnalysis: Functions for drama analysis (surprise!)
    - Today: Third iteration, extensive rewrite
- Philosophy: Construction kit

Installation

Installation

Code

```
install.packages("DramaAnalysis")
library(DramaAnalysis)  # no quotes

 # additional package
library(magrittr)
```

Installation

Code

```
install.packag
library(Drama
 # additional
library(magrit
```

Figure 1: René Magritte: The Treachery of Images

Installation

Data

- Dramatic texts are initially stored as TEI/XML files
- Language processing (e.g., identification of parts of speech) takes place in a UIMA pipeline
  - https://github.com/quadrama/DramaNLP
- Output of the pipeline: Several CSV files for each play (meta data, character data, …)
- CSV files analysed in R

Installation

Data

- Dramatic texts are initially stored as TEI/XML files
- Language processing (e.g., identification of parts of speech) takes place in a UIMA pipeline
  - https://github.com/quadrama/DramaNLP
- Output of the pipeline: Several CSV files for each play (meta data, character data, …)
- CSV files analysed in R

Two corpora today:

```
installData("qd") # German literary canon
# or
installData("shakedracor") # English Shakespeare plays
```

Installation

Data

The function `installData()`

- Clones a git repository from `github.com/quadrama` into a local directory
- Allows easy update of data files
- German literary canon (`qd`)
  - TextGrid $\rightarrow$ GerDraCor $\rightarrow$ QuaDramA
- English Shakespeare plays (`shakedracor`)
  - Folger $\rightarrow$ DraCor $\rightarrow$ QuaDramA
- Two demo plays included in the package
  - Including manual coreference annotation
  - Lessing's *Emilia Galotti* and *Miss Sara Sampson* (German)

Inspecting data

```
# Collect all play ids into a vector
loadAllInstalledIds() %>%
  # Extract metadata for each play,
  # put it into a table
  loadMeta()          %>%
  # Have RStudio display a nice table
  View()
```

Inspecting data

```
# Collect all play ids into a vector
loadAllInstalledIds() %>%
  # Extract metadata for each play,
  # put it into a table
  loadMeta()          %>%
  # Have RStudio display a nice table
  View()
```

Filter

	corpus	drama	documentTitle	language	Name	Pn
1	shakedracor	1H4	Henry IV, Part I	en	William Shakespeare	Wi
2	shakedracor	1H6	Henry VI, Part 1	en	William Shakespeare	Wi
3	shakedracor	2H4	Henry IV, Part II	en	William Shakespeare	Wi
4	shakedracor	2H6	Henry VI, Part 2	en	William Shakespeare	Wi
5	shakedracor	3H6	Henry VI, Part 3	en	William Shakespeare	Wi

Figure 2: Metadata table in RStudio

Loading a play

- We first have to load plays into the environment
- Each play has an associated id
- Select one and create a variable to store the id (less typing in the future)

Loading a play

- We first have to load plays into the environment
- Each play has an associated id
- Select one and create a variable to store the id (less typing in the future)

```
# General form: collection colon play
# (allows comparison across collections)
myId <- "shakedracor:Rom"

play <- loadDrama(myId)
```

Online help

- Each function is documented
- Entering question mark followed by the function name opens the help view
  - `?loadDrama`
- Documentation
  - What does the function do?
  - What arguments does it expect, which default values are defined?
  - What does it return?
  - Usage example

Online help

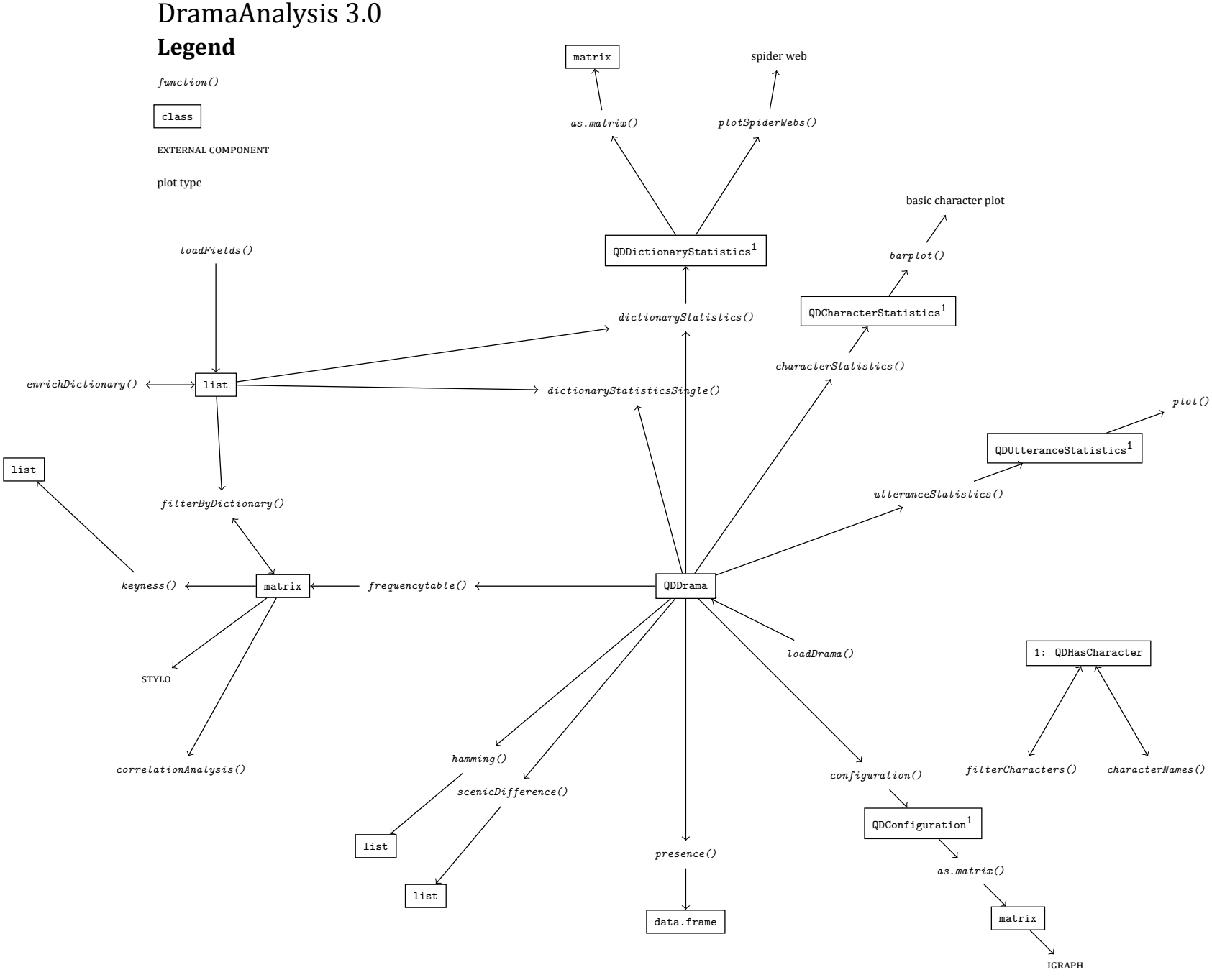
- Each function is documented
- Entering question mark followed by the function name opens the help view
  - `?loadDrama`
- Documentation
  - What does the function do?
  - What arguments does it expect, which default values are defined?
  - What does it return?
  - Usage example

Package documentation: https://quadrama.github.io/DramaAnalysis/3.0.0

Tutorial: https://quadrama.github.io/DramaAnalysis/tutorial/3/index.html

What can we do?

Function overview



Three parts

1. Global character/utterance statistics
2. Word fields
3. Character relations

1. Global character statistics

Global character statistics

Two functions:

- `characterStatistics()`: Characters in focus
- `utteranceStatistics()`: Utterances in focus

Function characterStatistics

```
cs <- characterStatistics(play)
```

Returns a table (in R: data.frame) with

- corpus: The collection id
- drama: The play id
- character: the character id
- tokens: Number of tokens (for this character)
- types: Number of different tokens (for this character)
- utterances: Number of utterances (for this character)
- utteranceLengthMean: Mean utterance length
- utteranceLengthSd: Utterance length standard deviation
- firstBegin: Starting position of the first utterance
- lastEnd: End position of the last utterance

(The function View() can be used to get browsable table in RStudio.)

Function characterStatistics I

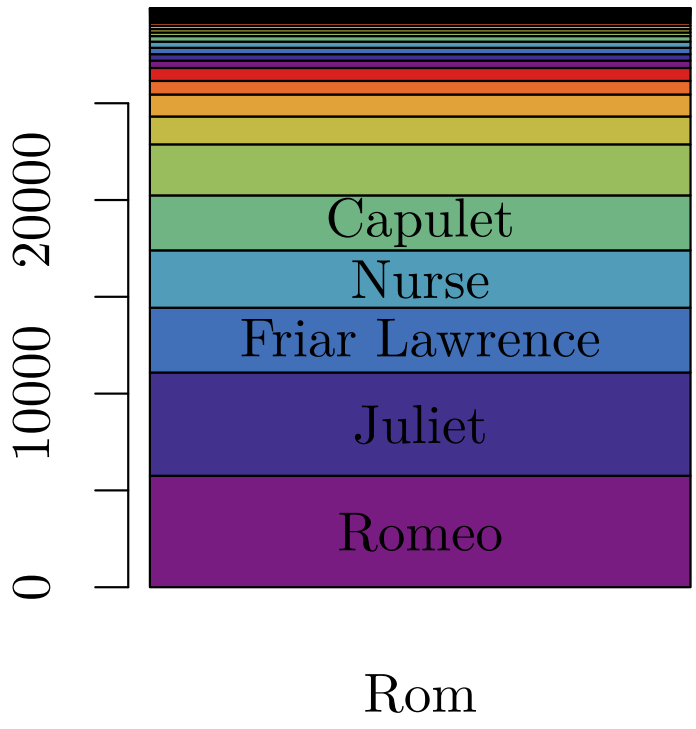
Plotting

```
# load a play
play <- loadDrama("shakedracor:Rom")

# call the function
characterStatistics(play) %>%
  # replace character ids by character names
  characterNames(play) %>%
  # plot them stacked
  barplot()
```

Function `characterStatistics` II

Plotting



Function `utteranceStatistics`

```
us <- utteranceStatistics(play)
```

Returns a table with one row for each utterance

- `corpus`: The collection id
- `drama`: The play id
- `character`: the character id
- `utteranceBegin`: Character position of the first character
- `utteranceLength`: Portion of this utterance with the total play

(The function `View()` can be used to get browsable table in RStudio.)

Function utteranceStatistics I

Plotting

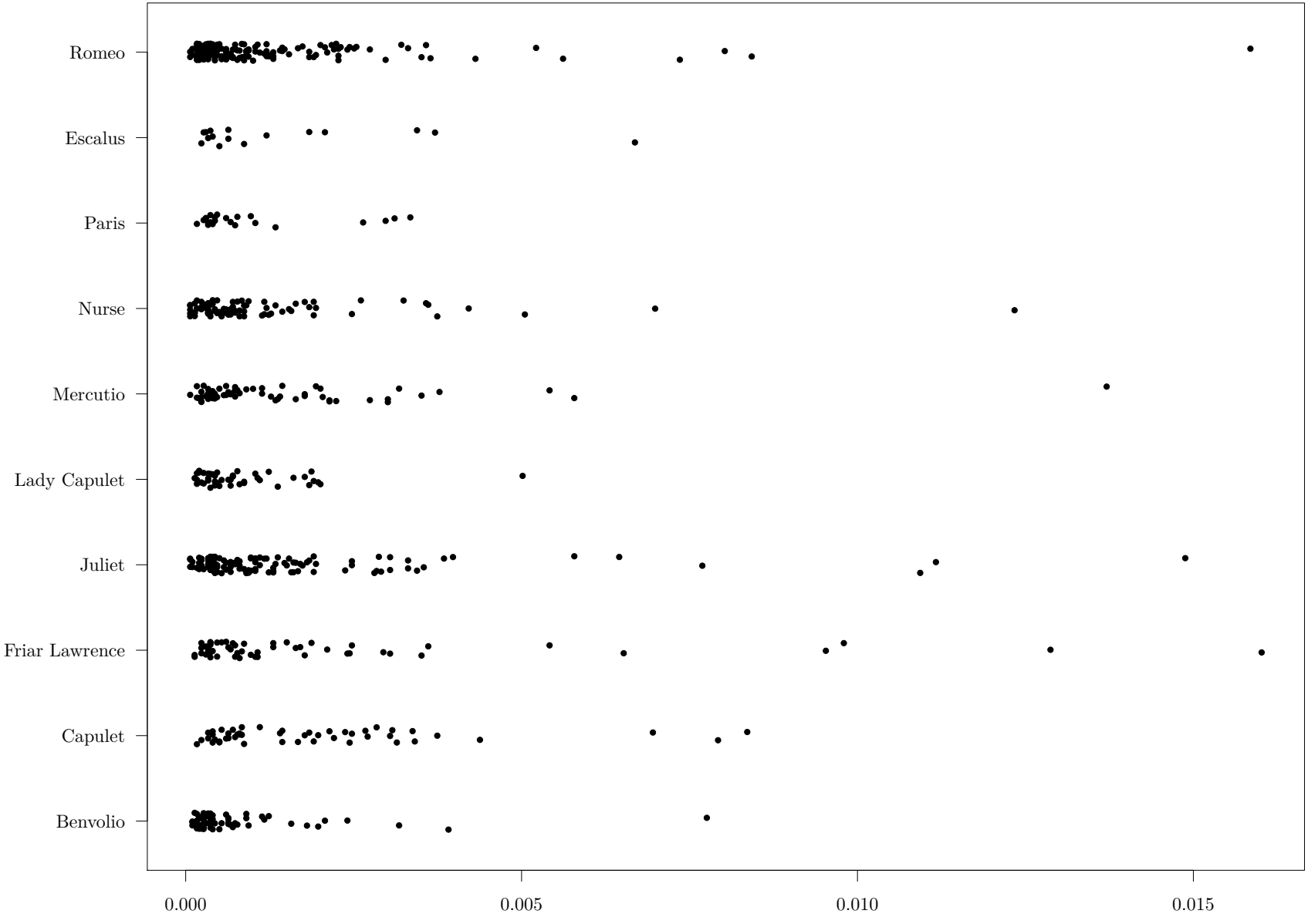
```
play <- loadDrama("shakedracor:Rom")

# get utterance statistics
us <- utteranceStatistics(play) %>%
  # remove uninteresting characters
  filterCharacters(play) %>%
  # replace ids by names
  characterNames(play)

# plot boundaries
par(mar=c(2,7,1,1))
# plot the utterances
stripchart(utteranceLength ~ character,
        data = us,
        las=1,
        pch=20,
        method="jitter")
```

Function `utteranceStatistics` II

Plotting



2. Word fields

Word fields

Word fields: Semantically related words

- Represented as a vector of strings in R
- E.g., love, heart is a word field related to love

Work steps

1. Define a word field: base R, `loadFields()`
2. Apply it to text(s): `dictionaryStatistics()`

Word Fields

Define a word field

Definition of a word field manually on the fly

```
fields <- list(
  # words related to family
  Family=c("marriage", "parents", "ancestors", ...),
  # words related to love
  Love=c("love", "heart", "kiss", ...))
```

Creates a named list of lists

Word Fields

Define a word field: Function ‘loadFields()‘

- Function to load word fields from URLs or files
- Load pre-defined (German) word lists

```
fields <- loadFields(fieldnames=c("Liebe", "Familie"))
```

Returns a named list of lists

Word Fields

Other sources

- Defining word fields manually is not trivial (historic language(s), bias, …)
- Existing dictionaries can be used as sources
- Enriching fields with distributionally similar words

Word Fields

Application: ‘dictionaryStatistics()‘

```
play <- loadDrama("shakedracor:Rom")
ds <- dictionaryStatistics(play, fields)
```

Returns a table with columns

- `corpus`, `drama`: See above
- `character`: The character id
- one column for each field

Word Fields

Application: ‘dictionaryStatistics()‘

```
play <- loadDrama("shakedracor:Rom")
ds <- dictionaryStatistics(play, fields)
```

Returns a table with columns

- `corpus`, `drama`: See above
- `character`: The character id
- one column for each field

```
##         corpus drama         character Family Love
## 1 shakedracor   Rom   Apothecary_Rom       0    0
## 2 shakedracor   Rom     Benvolio_Rom       0    9
## 3 shakedracor   Rom CITIZENS.0.1_Rom       0    0
```

Word Fields

Normalization

- Different characters have different portions of speech in the play
- Word fields may be differently large

Word Fields

Normalization

- Different characters have different portions of speech in the play
- Word fields may be differently large
- Normalization is important
  - No one fits all mechanic
  - It depends on the research question

Word Fields

Normalization

- Different characters have different portions of speech in the play
- Word fields may be differently large
- Normalization is important
  - No one fits all mechanic
  - It depends on the research question

Two parameters (both can be set to TRUE/FALSE)

- `normalizeByCharacter`
- `normalizeByField`

Word Fields

Normalization

- Different characters have different portions of speech in the play
- Word fields may be differently large
- Normalization is important
  - No one fits all mechanic
  - It depends on the research question

Two parameters (both can be set to TRUE/FALSE)

- `normalizeByCharacter`
- `normalizeByField`

Normalized numbers tend to be very small, but that does not hinder their meaningfulness

Word Fields I

Plotting

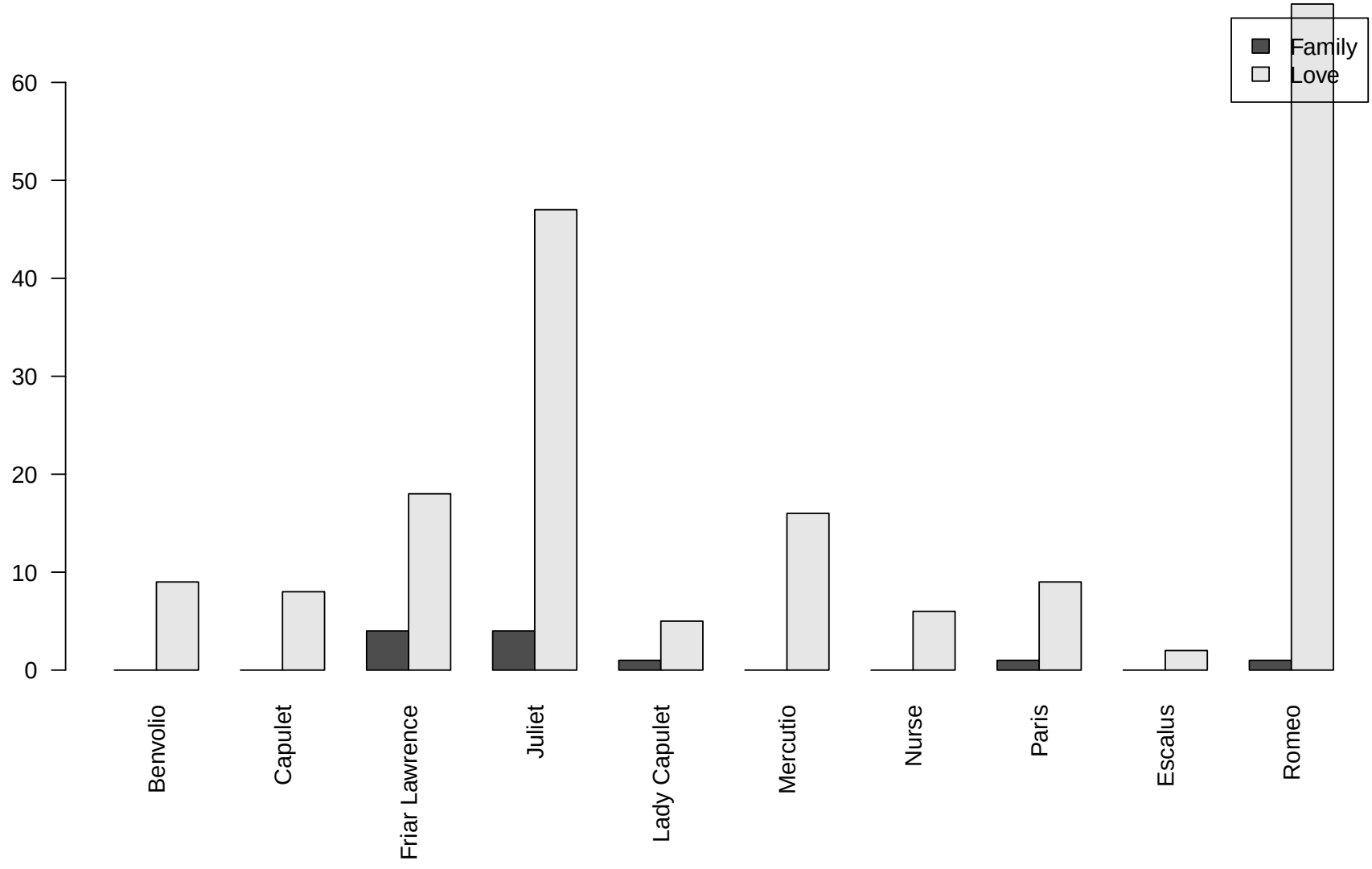
```
ds <- dictionaryStatistics(play, fields) %>%
  filterCharacters(play) %>%
  characterNames(play)

dsm <- as.matrix(ds)

par(mar=c(10,2,1,1))
barplot(t(dsm),
        beside=TRUE,
        names.arg = ds$character,
        legend.text = colnames(dsm),
        las=2)
```

Word Fields II

Plotting



3. Character Relations

Character Relations

- Configuration: A matrix showing who is on stage when

Functions

- `configuration()`
- `presence()`

Package `igraph`

Configuration

Function ‘configuration()‘

```
play <- loadDrama("shakedracor:Rom")
conf <- configuration(play)
```

Table with columns

- `corpus`, `drama`, `character`
- One column per segment, filled with the number of words spoken by a character

Configuration I

Plotting

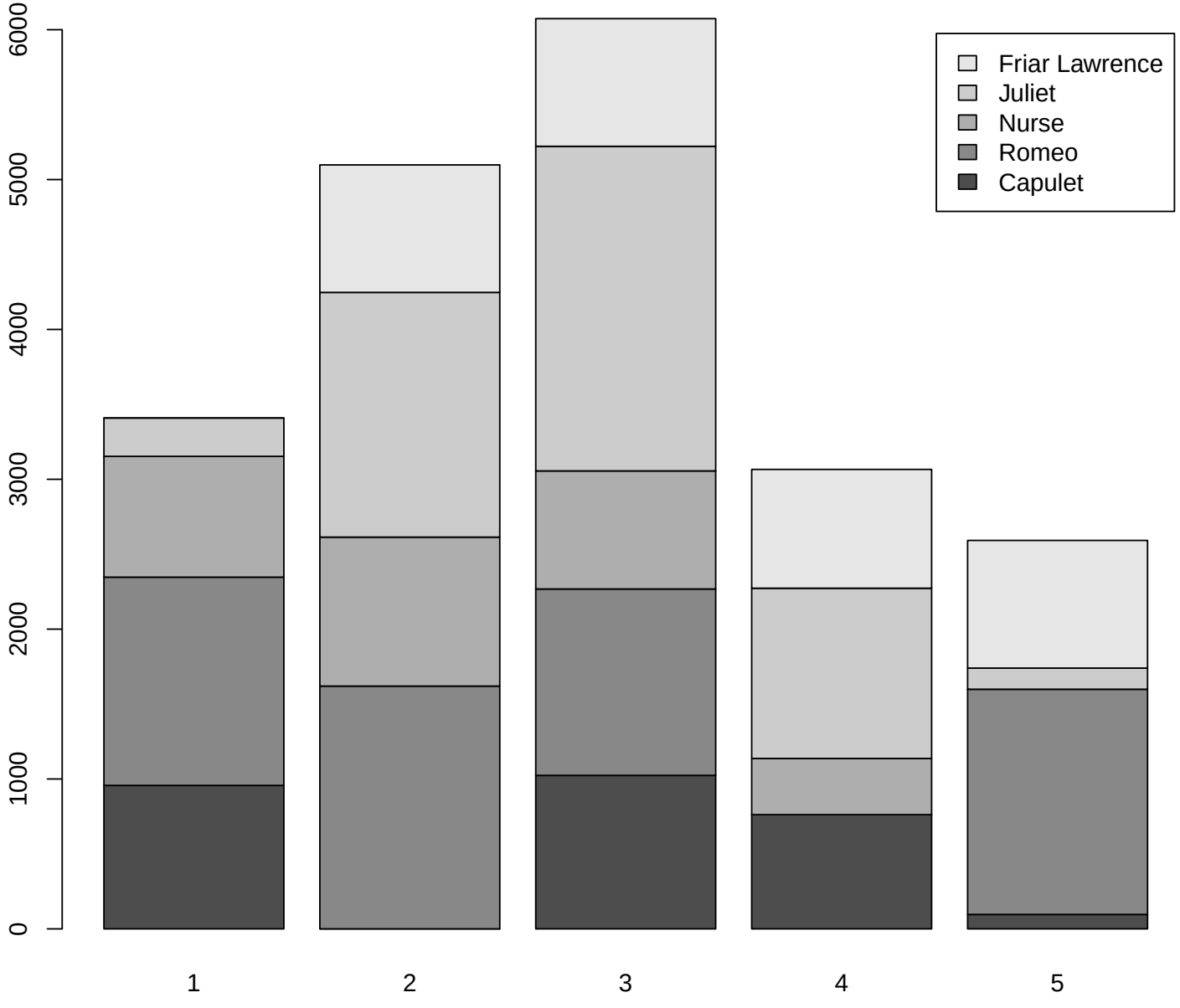
```
c <- configuration(play) %>%
  filterCharacters(play, n=5) %>%
  characterNames(play)

mat <- as.matrix(c)
par(mar=c(2,2,2,10))

barplot(mat,
        legend.text = c$character)
```

Configuration II

Plotting



Character Network

Step 1: Create an adjacency matrix

```
c <- configuration(play,
                   onlyPresence = TRUE,
                   segment = "Scene") %>%
  filterCharacters(play) %>%
  characterNames(play)
mat <- as.matrix(c)

# multiply the matrix with its inverse
# this creates the adjacency matrix
adjMatrix <- mat %*% t(mat)

# add character names
rownames(adjMatrix) <- c$character
colnames(adjMatrix) <- c$character
```

Character Network

Step 2: Create graph and plot it

Using the library `igraph`:

```
library(igraph)
# convert the adjacency matrix to a graph object
g <- graph_from_adjacency_matrix(copresence,
                                 weighted=TRUE,
                                 mode="undirected",
                                 diag=FALSE)

# plot it
plot.igraph(g,
     layout=layout_in_circle,
     main="Copresence Network: Romeo & Juliet",
     edge.width=E(g)$weight)
```

Character Presence I

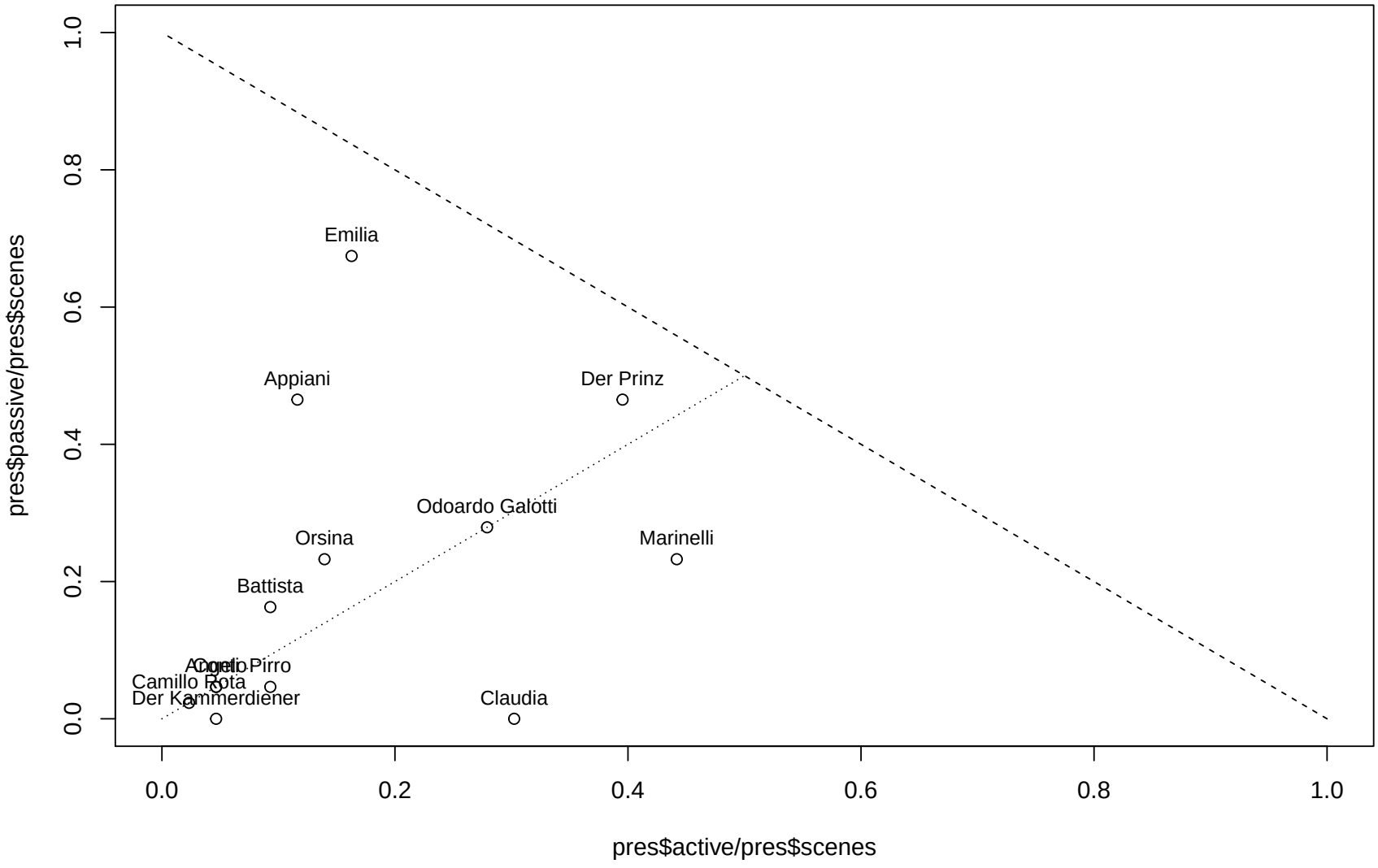
This (currently) only works for manually annotated plays

```
data(rksp.0) # load Emilia Galotti

# calculate presence
pres <- presence(rksp.0) %>%
  characterNames(rksp.0)

# plot points
plot(x=pres$active/pres$scenes,
     y=pres$passive/pres$scenes,
     xlim=c(0,1),
     ylim=c(0,1))
# add labels
text(x=pres$actives/pres$scenes,
     y=pres$passives/pres$scenes,
     labels=substr(pres$character,0,20),
     pos=3,
     cex=0.8)
# add lines
lines(x=seq(0,0.5,0.1),seq(0,0.5,0.1), lty=3)
lines(x=1:0,y=0:1, lty=2)
```

Character Presence II



Lab session

Lab session

… and now, it's your turn!

Pick one or more plays, and do one of the analyses, or follow your own ideas!

(don't be afraid, you can't break anything)

Getting help

- question mark plus function name: `?presence`
- Package documentation: https://quadrama.github.io/DramaAnalysis/3.0.0/
- Tutorial: https://quadrama.github.io/DramaAnalysis/tutorial/3/
- … and we're here for you too!