

# DramaAnalysis

*Nils Reiter*

*2019-05-15*



# Contents

<b>Preface</b>	<b>5</b>
<b>1 Introduction</b>	<b>7</b>
1.1 Technical basics . . . . .	7
<b>2 Data</b>	<b>9</b>
2.1 Origin: TEI-XML . . . . .	9
2.2 Preprocessed data . . . . .	9
2.3 Sample data . . . . .	9
<b>I Structure Analysis</b>	<b>11</b>
<b>3 Who’s talking how much?</b>	<b>15</b>
3.1 Character names instead of identifiers . . . . .	18
3.2 Stacked bar plot . . . . .	19
<b>4 Who’s talking how often?</b>	<b>21</b>
4.1 When are characters talking? . . . . .	22
4.2 Adding act boundaries . . . . .	23
<b>5 Configuration</b>	<b>25</b>
5.1 Matrices . . . . .	25
5.2 Copresence . . . . .	28
<b>6 Character Exchange</b>	<b>35</b>
6.1 Hamming Distance . . . . .	35
<b>7 Text Collections</b>	<b>39</b>
7.1 Defining collections . . . . .	39
7.2 Loading multiple plays . . . . .	39
<b>II Text Analysis</b>	<b>41</b>
<b>8 Word Field Analysis</b>	<b>43</b>
8.1 Single word field . . . . .	43
8.2 Multiple Word Fields . . . . .	45
8.3 Dictionary Based Distance . . . . .	48
8.4 Development over the course of the play . . . . .	50
<b>9 Advanced Text Analysis</b>	<b>55</b>
9.1 When are characters mentioned? . . . . .	55

**10 Resterampe****57**

# Preface

*This book is in alpha stadium and not ready. It pertains to the (currently unreleased) version 3.0 of the DramaAnalysis package. Use at your own risk.*



# Chapter 1

## Introduction

### 1.1 Technical basics

#### 1.1.1 Classes

#### 1.1.2 Pipelining with `margrittr`

## DramaAnalysis 3.0

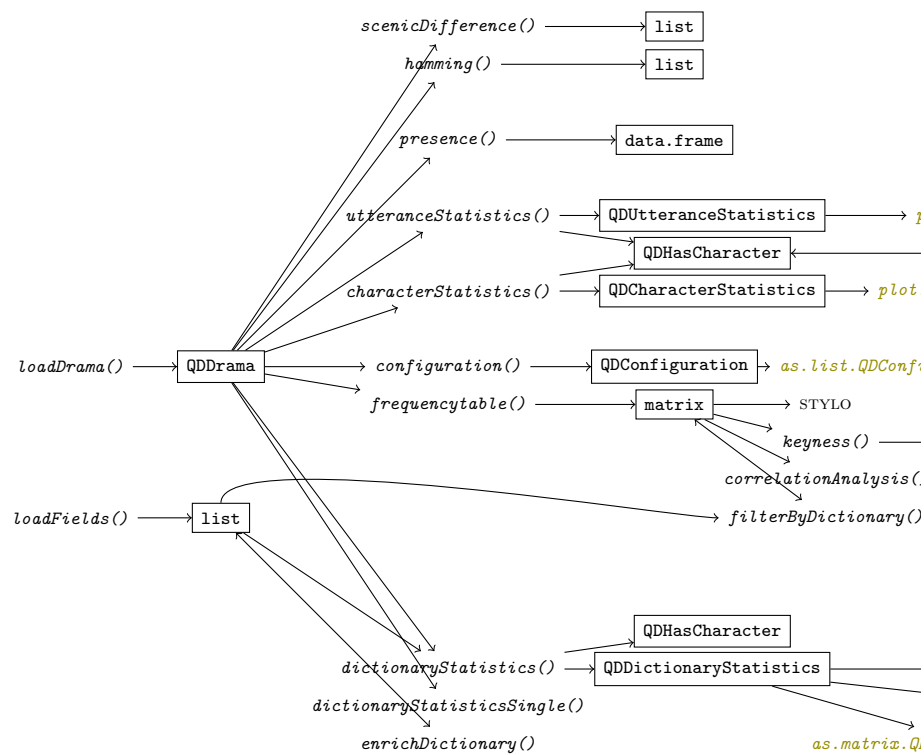


Figure 1.1: Some caption.



# Chapter 2

## Data

Before analysing any data, it needs to be imported, and converted into the proper structure. In QuaDramaA, we process dramatic texts in multiple stages, described below.

### 2.1 Origin: TEI-XML

The base format that we use (and in which we put all our annotations) is an XML format known as TEI. This format is used by most researchers doing quantitative drama analysis. An excellent source for dramas in the proper format is DraCor, maintained by Frank Fischer.

While we are using GerDraCor as a basis, we have added linguistic annotations to a number of plays, and integrated more plays (e.g., translations) into the corpus. This corpus can be found [here](#).

### 2.2 Preprocessed data

As a first step, we process all dramatic texts using our DramaNLP pipeline. The result of this processing is a set of CSV files for each play that contains the information in the play in a format suitable for analysis with R. This repository contains two plays in the format, which are also included in our R package.

Table 2.1: Different CSV files used in the analysis. *ID* is a placeholder for a unique identifier for the play

File	Description
<i>ID</i> .Metadata.csv	Meta data for the play (author, title, language, ...)
<i>ID</i> .Characters.csv	Characters of the play, with some character specific information
<i>ID</i> .Entities.csv	All discourse entities (including characters, but also all other coreference chains)
<i>ID</i> .Mentions.csv	Mentions associated with characters
<i>ID</i> .StageDirections.csv	The stage directions of the play
<i>ID</i> .UtterancesWithTokens.csv	All character utterances of the play
<i>ID</i> .Segments.csv	Information about acts and scenes

### 2.3 Sample data

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

```
# Load Miß Sara Sampson  
data(rjmw.0)  
  
text <- combine(rksp.0, rjmw.0)
```

Part I

**Structure Analysis**



We're assuming here that we have loaded some texts using `loadDrama()`, and that this text is stored as a `QDDrama`-object in the variable `text`. For demo purposes, we will use the two plays that are included in the R-package, Lessing's *Emilia Galotti* and *Miss Sara Sampson*. Both have been preprocessed by the DramaNLP pipeline.



## Chapter 3

# Who's talking how much?

First, we calculate summary statistics over all characters.

```
charStats <- characterStatistics(text)
```

```
charStats
```

##	corpus	drama	character	tokens	types	utterances
## 1	test	rjmw.0	sir_william	2056	698	23
## 2	test	rjmw.0	waitwell	1826	568	41
## 3	test	rjmw.0	der_wirt	324	177	7
## 4	test	rjmw.0	mellefont	7981	1722	201
## 5	test	rjmw.0	norton	1001	407	46
## 6	test	rjmw.0	betty	482	223	20
## 7	test	rjmw.0	sara	9121	1908	166
## 8	test	rjmw.0	marwood	7225	1757	155
## 9	test	rjmw.0	hannah	258	155	16
## 10	test	rjmw.0	der_bediente	44	34	2
## 11	test	rjmw.0	arabella	398	162	13
## 12	test	rksp.0	der_prinz	5303	1257	157
## 13	test	rksp.0	der_kammerdiener	42	34	6
## 14	test	rksp.0	conti	764	325	24
## 15	test	rksp.0	marinelli	5567	1324	221
## 16	test	rksp.0	camillo_rota	106	62	6
## 17	test	rksp.0	claudia_galotti	2098	657	73

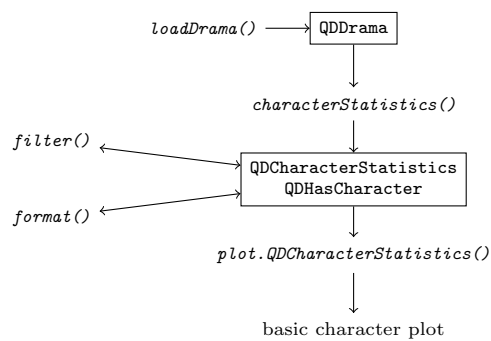


Figure 3.1: Relevant classes and functions in this chapter

```
## 18 test rksp.0      pirro      343    196      25
## 19 test rksp.0      odoardo    3248   891     108
## 20 test rksp.0      angelo      635    300      28
## 21 test rksp.0      emilia     2275   630      64
## 22 test rksp.0      appiani    1112   426      48
## 23 test rksp.0      battista   195    112      11
## 24 test rksp.0      orsina     2832   743      64
##      utteranceLengthMean utteranceLengthSd firstBegin lastEnd
## 1          89.39130         108.948169      597 170838
## 2          44.53659         59.768762      670 158683
## 3          46.28571         45.492543     3571  5695
## 4          39.70647         50.034972     5848 170188
## 5          21.76087         23.601492     6328 170456
## 6          24.10000         36.404019     9014 156987
## 7          54.94578         77.785973    11766 167457
## 8          46.61290         62.383230    28320 141005
## 9          16.12500         15.654073    28392  58523
## 10         22.00000         12.727922    31532  31790
## 11         30.61538         31.967532    43987  52559
## 12         33.77707         40.807176      426 136067
## 13          7.00000          5.215362     1149  24954
## 14         31.83333         40.141778     2654  12212
## 15         25.19005         29.844599    13147 134486
## 16         17.66667         19.179851    25577  26914
## 17         28.73973         29.131897    27006 112526
## 18         13.72000         10.663489    27113  50425
## 19         30.07407         40.399571    27385 135555
## 20         22.67857         19.573326    28777  64877
## 21         35.54688         48.159105    36769 134940
## 22         23.16667         25.525401    44452  56735
## 23         17.72727         32.948720    67465  86481
## 24         44.25000         50.977119    88094 112182
```

This already gives us a lot of information about the characters. In particular, the function `characterStatistics()` returns a table (of the types `QDCharacterStatistics`, `QDHasCharacter` and `data.frame`) with information about:

- the number of tokens a character speaks (`tokens`),
- the number of different tokens a character speaks (`types`),
- the number of utterances (`utterances`),
- the average length of the utterances (`utteranceLengthMean`),
- their standard deviation (`utteranceLengthSd`),
- the character position of the start of the first utterance (`firstBegin`), and
- the character position of the end of the last utterance (`lastEnd`).

The function `characterStatistics()` provides a number of options to control its exact behaviour. Entering `?characterStatistics` in the R console opens the documentation for the function with a description of all the options. We'll describe some frequently used options here as well:

- **Punctuation:** By default, all punctuation marks are counted as tokens. This behaviour can be changed by setting `filterPunctuation=TRUE`.
- **Normalization:** The values in the table above are all absolute values. When comparing to other texts, one is often interested in normalized values. If the option `normalize` is set to `TRUE`, all values will be normalised (if applicable).
- **Segmentation:** By default, the function extracts values for the entire play. With the option `segment`,



it is possible to extract statistics by act or scene, as shown in the example below. Except for the additional column `Act`, the columns in the table are the same as before.

```
characterStatistics(rksp.0,
                  segment="Act")
```

##	corpus	drama	Act	character	tokens	types	utterances
## 1	test	rksp.0	I	der_prinz	2740	795	71
## 2	test	rksp.0	I	der_kammerdiener	42	34	6
## 3	test	rksp.0	I	conti	764	325	24
## 4	test	rksp.0	I	marinelli	1038	412	31
## 5	test	rksp.0	I	camillo_rota	106	62	6
## 6	test	rksp.0	II	claudia_galotti	1252	477	50
## 7	test	rksp.0	II	pirro	343	196	25
## 8	test	rksp.0	II	odoardo	611	295	17
## 9	test	rksp.0	II	angelo	385	214	19
## 10	test	rksp.0	II	emilia	1166	417	28
## 11	test	rksp.0	II	appiani	1112	426	48
## 12	test	rksp.0	II	marinelli	527	255	28
## 13	test	rksp.0	III	marinelli	1814	605	56
## 14	test	rksp.0	III	der_prinz	945	398	29
## 15	test	rksp.0	III	angelo	250	147	9
## 16	test	rksp.0	III	battista	187	106	9
## 17	test	rksp.0	III	emilia	413	175	16
## 18	test	rksp.0	III	claudia_galotti	614	241	16
## 19	test	rksp.0	IV	der_prinz	803	319	30
## 20	test	rksp.0	IV	marinelli	1358	495	72
## 21	test	rksp.0	IV	battista	8	7	2
## 22	test	rksp.0	IV	orsina	2832	743	64
## 23	test	rksp.0	IV	odoardo	706	275	30
## 24	test	rksp.0	IV	claudia_galotti	232	125	7
## 25	test	rksp.0	V	marinelli	830	363	34
## 26	test	rksp.0	V	der_prinz	815	321	27
## 27	test	rksp.0	V	odoardo	1931	620	61
## 28	test	rksp.0	V	emilia	696	251	20
##	utteranceLengthMean		utteranceLengthSd		firstBegin	lastEnd	
## 1	38.59155		39.891130		426	26559	
## 2	7.00000		5.215362		1149	24954	
## 3	31.83333		40.141778		2654	12212	
## 4	33.48387		34.234360		13147	23619	
## 5	17.66667		19.179851		25577	26914	
## 6	25.04000		28.927750		27006	56666	
## 7	13.72000		10.663489		27113	50425	
## 8	35.94118		41.800823		27385	36244	
## 9	20.26316		19.674694		28777	32436	
## 10	41.64286		63.826277		36769	48503	
## 11	23.16667		25.525401		44452	56735	
## 12	18.82143		17.346263		50431	55936	
## 13	32.39286		35.245982		56826	79337	
## 14	32.58621		48.327393		56914	72918	
## 15	27.77778		19.466495		63132	64877	
## 16	20.77778		36.033935		67465	75603	
## 17	25.81250		27.352559		67517	79537	
## 18	38.37500		28.765431		74777	79747	
## 19	26.76667		38.699268		79838	95409	

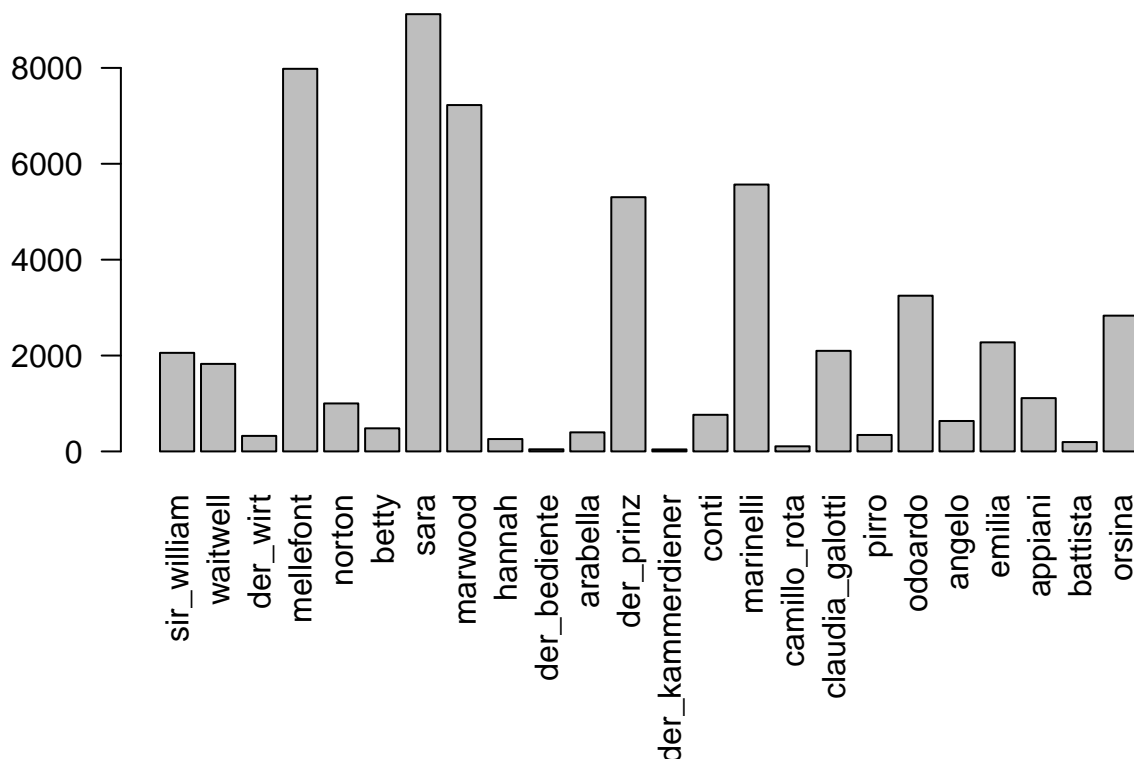
## 20	18.86111	20.750229	79972	103998
## 21	4.00000	2.828427	86363	86481
## 22	44.25000	50.977119	88094	112182
## 23	23.53333	18.528140	101533	112733
## 24	33.14286	29.952343	110037	112526
## 25	24.41176	36.802600	112824	134486
## 26	30.18519	37.235257	113752	136067
## 27	31.65574	47.382094	114998	135555
## 28	34.80000	34.284568	128910	134940

Of course, the values in the above table can be directly plotted:

```
charStats <- characterStatistics(text, normalize=FALSE)

par(mar=c(9,3,2,2)) # increase plot margins, so that the labels can be plotted

barplot(charStats$tokens, # these are the values to be plotted
        names.arg = charStats$character, # the labels on the x-axis
        las=2 # rotate the labels on both axes
)
```



### 3.1 Character names instead of identifiers

By default, all our functions identify characters using technical ids, which may or may not be human-readable. Even if they are, it's usually a good idea to replace them with nice to read labels before publication. We therefore provide the function `format()`, which can be applied to any table that contains a column with character ids (i.e., any object of type `QDHasCharacter`).

```
charStats <- characterStatistics(text, normalize=FALSE)
```

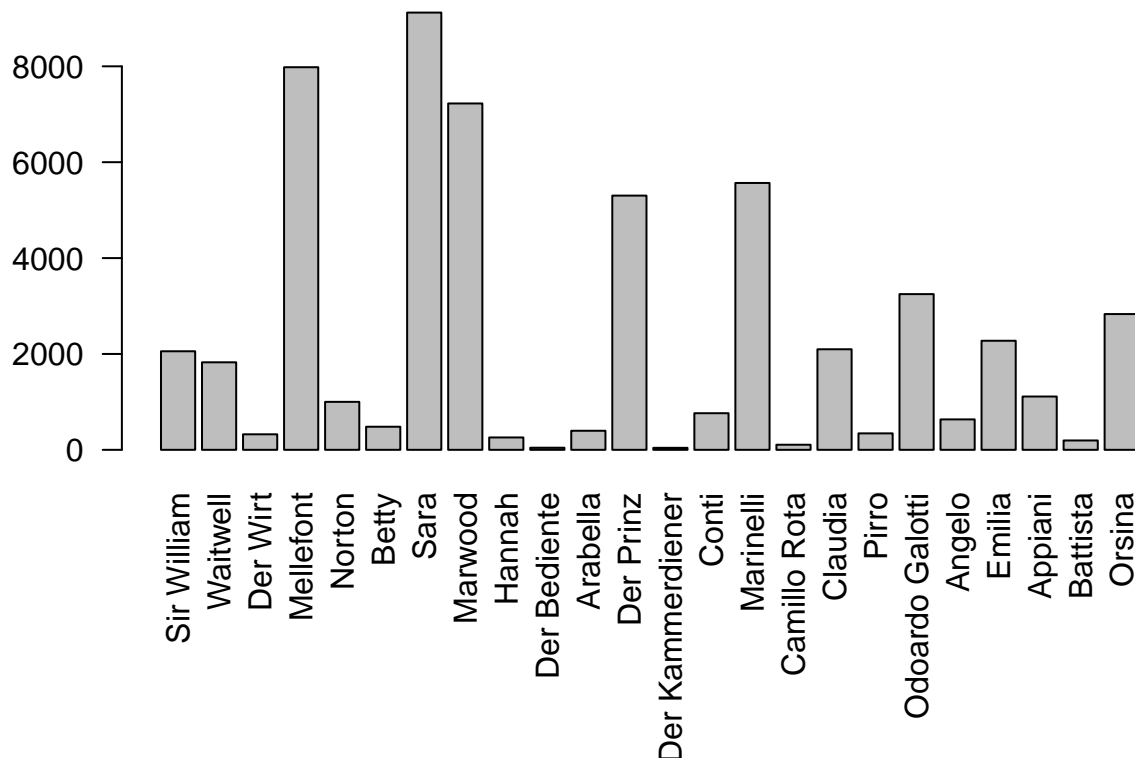
```

charStats <- format(charStats, text) # Replace ids with names

par(mar=c(9,3,2,2)) # increase plot margins, so that the labels can be plotted

barplot(charStats$tokens, # these are the values to be plotted
        names.arg = charStats$character, # the labels on the x-axis
        las=2 # rotate the labels on both axes
)

```



As can be seen above `format()` requires two arguments: The table in which we want to replace ids by characters, and the original drama object (that we got from calling the function `loadDrama()`).

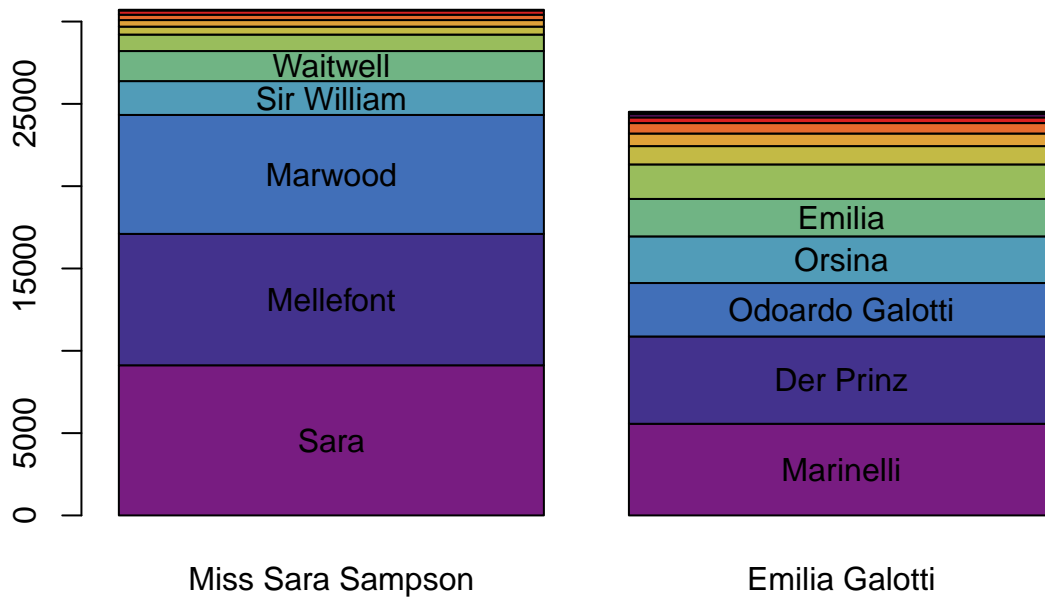
## 3.2 Stacked bar plot

The plot shown above is quite wide, and some aspects (like the ranking in terms of spoken tokens), is hard to see. We often use another way of visualizing this, which can be used if you supply the `QDCharacterStatistics` directly into the `barplot()` function. In combination with the `margrittr` pipes, we can call it like this (enter `?barplot.QDCharacterStatistics` for details on the special `barplot` function):

```

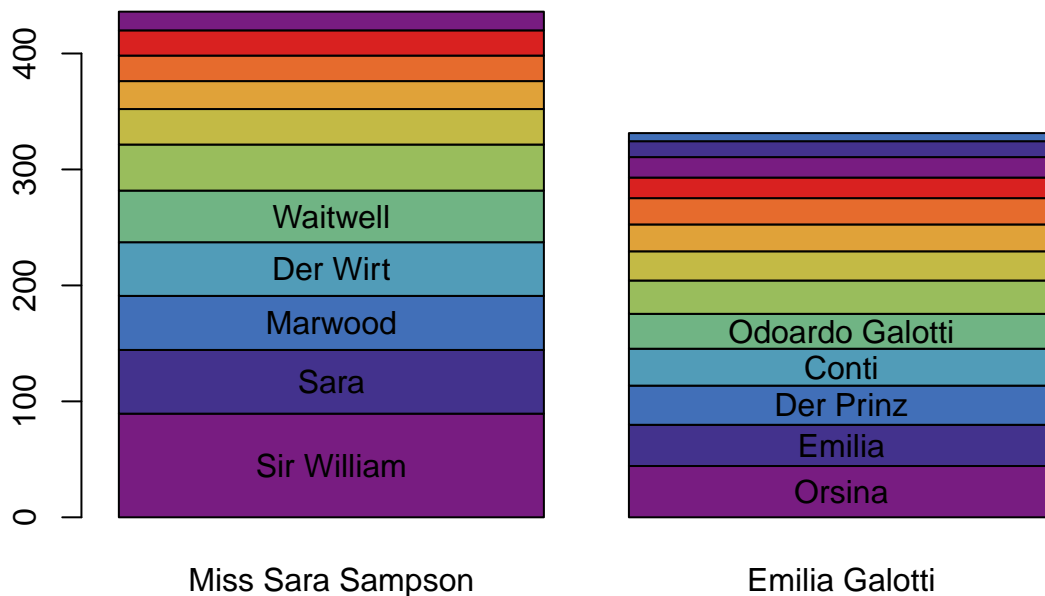
characterStatistics(text, normalize=FALSE) %>%
  format(text) %>%
  barplot(names.arg=c("Miss Sara Sampson", "Emilia Galotti"))

```



By default, the `barplot.QDCharacterStatistics()` function visualises the number of tokens spoken by the characters (and ranks the characters accordingly). This can be changed by supplying the option `column`, and specifying another value.

```
characterStatistics(text, normalize=FALSE) %>%
  format(text) %>%
  barplot(names.arg=c("Miss Sara Sampson", "Emilia Galotti"),
    column = "utteranceLengthMean") # show mean utterance length
```



This picture looks quite different! Other interesting columns to experiment with are `types` and `utteranceLengthSd`.

## Chapter 4

# Who's talking how often?

So far, we have counted words for characters. Now we will turn to utterances, and their properties.

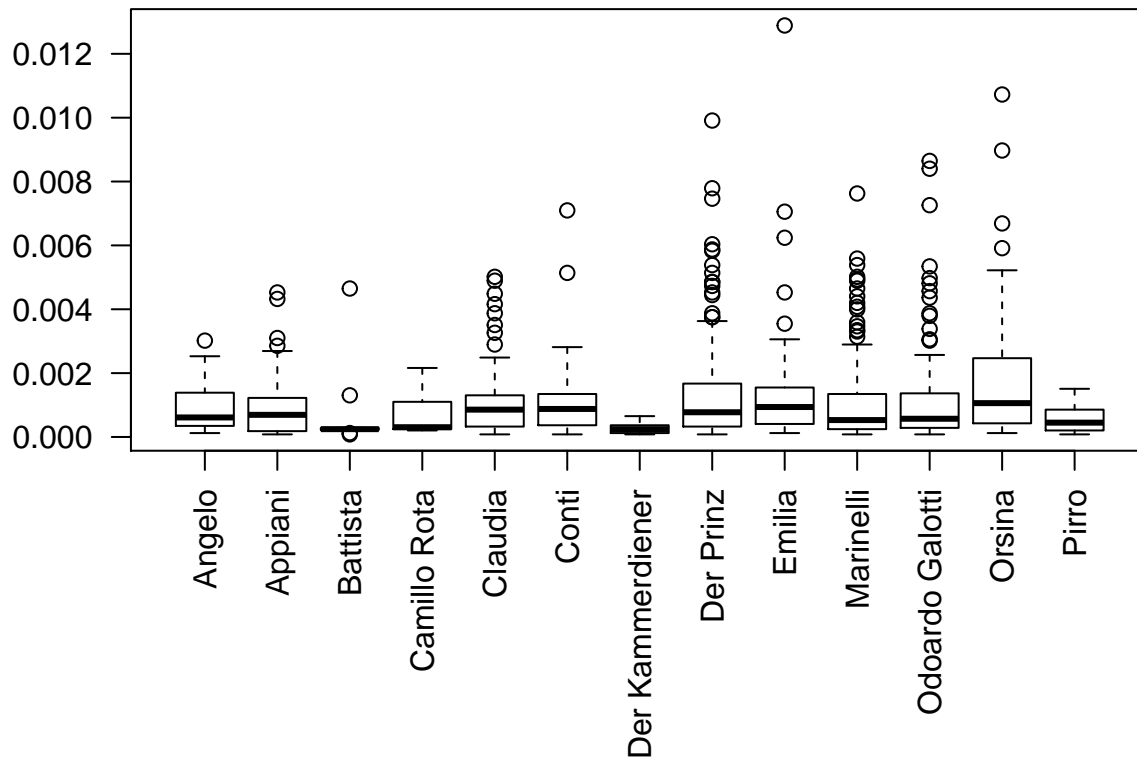
First, we will use the function `utteranceStatistics()` to extract quantitative information about utterances:

```
utteranceStatistics(rksp.0)
```

##	corpus	drama	character	utteranceBegin	utteranceLength
## 1	test	rksp.0	der_prinz	426	3.874388e-03
## 2	test	rksp.0	der_kammerdiener	1149	8.156607e-05
## 3	test	rksp.0	der_prinz	1178	2.895595e-03
## 4	test	rksp.0	der_kammerdiener	1526	6.525285e-04
## 5	test	rksp.0	der_prinz	1655	2.854812e-04
## 6	test	rksp.0	der_kammerdiener	1697	1.631321e-04
## 7	test	rksp.0	der_prinz	1739	1.019576e-03
## 8	test	rksp.0	der_kammerdiener	1857	3.262643e-04
## 9	test	rksp.0	der_prinz	1919	2.528548e-03
## 10	test	rksp.0	der_kammerdiener	2299	3.670473e-04

This creates a table that is very long, which is why we only show the first 10 rows here. The table contains one row for each utterance, and information about the speaker of the utterance, its length (measured in tokens) and its starting position (character position). We can now inspect the variance in utterance length:

```
ustat <- utteranceStatistics(rksp.0) %>%  
  format(rksp.0) # use names instead of character ids (see above)  
  
par(mar=c(9,4,2,2)) # increase margin  
  
boxplot(utteranceLength ~ character, # what do we want to correlate  
  data=ustat,  
  las = 2 # rotate axis labels  
)
```



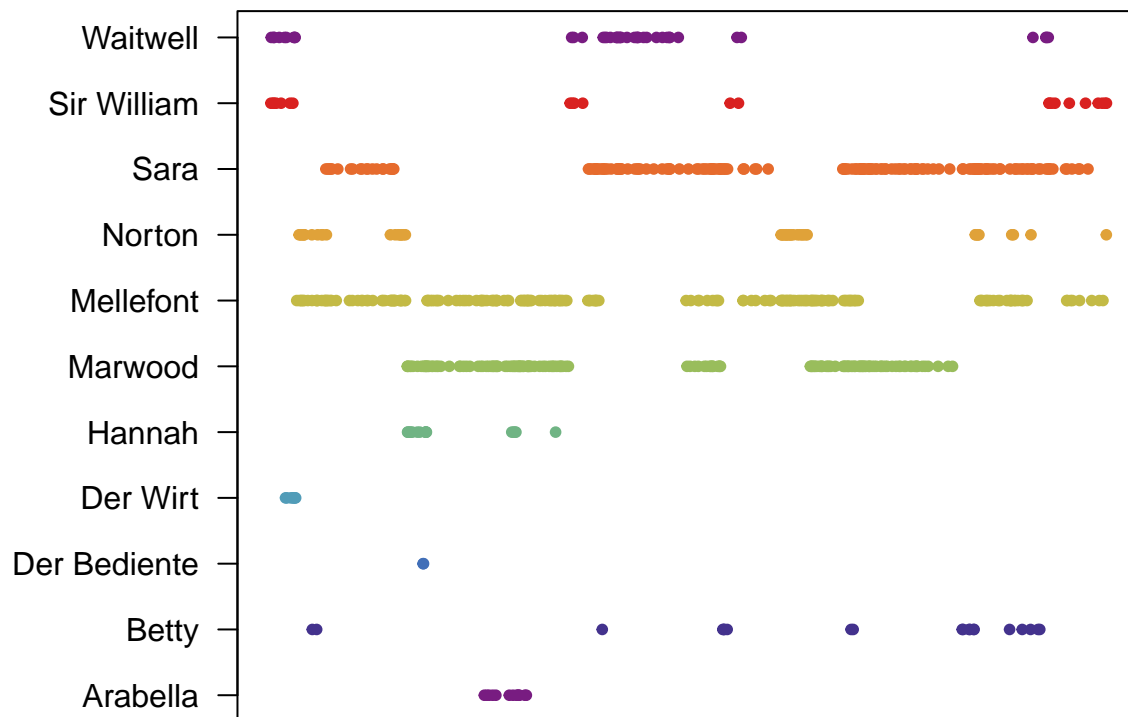
This uses the regular `boxplot()` function, enter `?boxplot` for documentation. `utteranceLength ~ character` is called a formula in R and (in this case) expresses that we want to look at the column `utteranceLength`, grouped by the column `character`. The boxplot is a useful way to grasp the dispersion of a set of values (in this case: the lengths of all utterances by a character).

## 4.1 When are characters talking?

While the above displays the *length* of utterances, we can also display the position of utterances (remember the column `utteranceBegin`?). The following snippet visualizes when characters are talking, this time for Lessings Miss Sara Sampson:

```
par(mar=c(2,7,2,2))

utteranceStatistics(rjmw.0) %>%
  format(rjmw.0) %>%      # character names instead of ids
  plot()                  # calling plot.QDUtteranceStatistics()
```

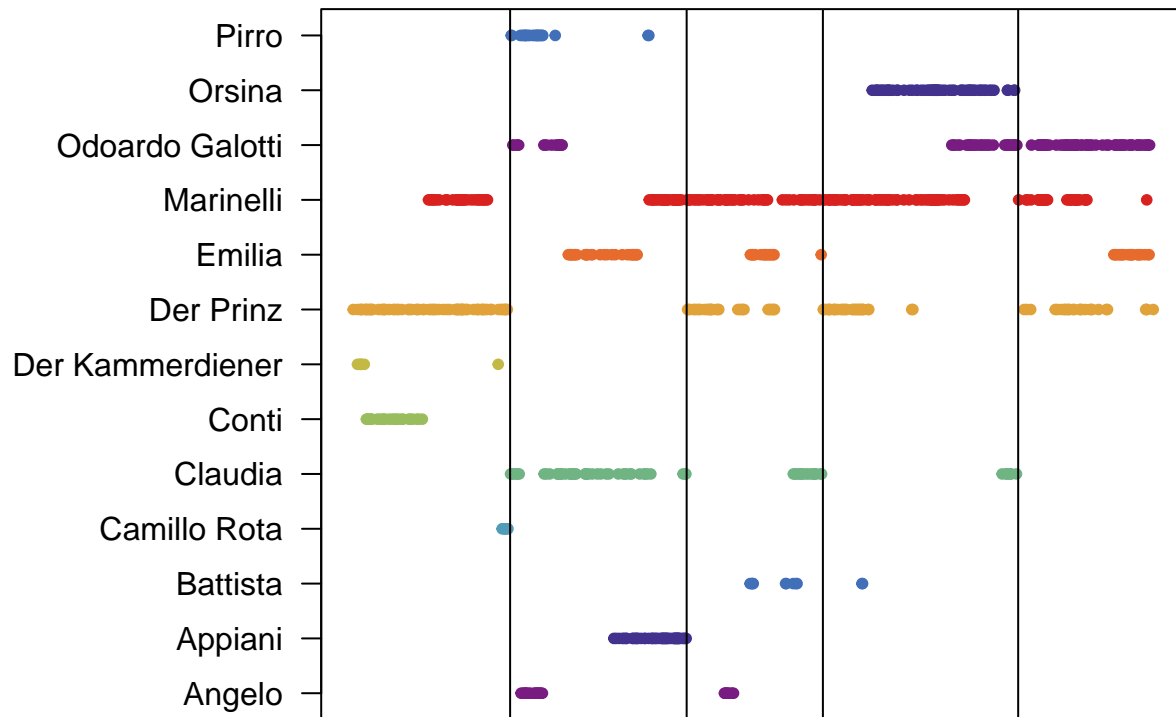


Each dot in this plot represents one utterance, the x-axis is measured in character positions. This is not really intuitive, but the flow from left to right represents the flow of the text. More technically, we again apply the function `format()` to display character names instead of character ids. This is the same function as above, just applied to a different table. It can be applied to any table of the type `QDHasCharacter`. Secondly, the call to the function `plot()` gets rerouted to the function `plot.QDUtteranceStatistics()`, because the object we supply as argument is of the type `QDUtteranceStatistics`. Information about this function can be retrieved by entering `?plot.QDUtteranceStatistics`.

## 4.2 Adding act boundaries

Now it would be useful to include information on act/scene boundaries in this plot. This can be done by supplying the original drama object as a second argument to the `plot()` function:

```
par(mar=c(2,8,2,2))
utteranceStatistics(rksp.0) %>%
  format(rksp.0) %>%
  plot(rksp.0) # adding the `QDDrama` object here creates the act boundaries.
```



Please note that the information contained in this plot is very similar to the information in configuration matrices.



## Chapter 5

# Configuration

*This section does not refer to the configuration of the R-package, but to the literary analysis concept configuration matrix (Pfister, 1988).*

### 5.1 Matrices

Configuration matrices can be extracted with the function `configuration()`. As usual, entering `?configuration` provides more detailed information about the options the function provides.

```
configuration(rksp.0)
```

##	corpus	drama	character	1	2	3	4	5
## 1	test rksp.0		der_prinz	2740	0	945	803	815
## 2	test rksp.0		der_kammerdiener	42	0	0	0	0
## 3	test rksp.0		conti	764	0	0	0	0
## 4	test rksp.0		marinelli	1038	527	1814	1358	830
## 5	test rksp.0		camillo_rota	106	0	0	0	0
## 6	test rksp.0		claudia_galotti	0	1252	614	232	0
## 7	test rksp.0		pirro	0	343	0	0	0
## 8	test rksp.0		odoardo	0	611	0	706	1931
## 9	test rksp.0		angelo	0	385	250	0	0
## 10	test rksp.0		emilia	0	1166	413	0	696
## 11	test rksp.0		appiani	0	1112	0	0	0
## 16	test rksp.0		battista	0	0	187	8	0
## 22	test rksp.0		orsina	0	0	0	2832	0

This creates a basic configuration matrix, but instead of just containing the presence or absence of a figure, it contains the number of spoken tokens for each act for each character. This information is in fact similar to what we can extract with `characterStatistics(rksp.0, segment="Act")`, but in a different form and structure. The above table contains a lot of information that can be visualised.

We first need to extract the numeric content from the above table. This can easily be done with the function `as.matrix()` (this will, in detail, be rerouted to the function `as.matrix.QDCConfiguration()`, which knows what part of the table needs to be in the matrix). A matrix containing only numbers can easily be plotted using the regular `barplot()` function, as shown below.

```
c <- configuration(rksp.0)
```

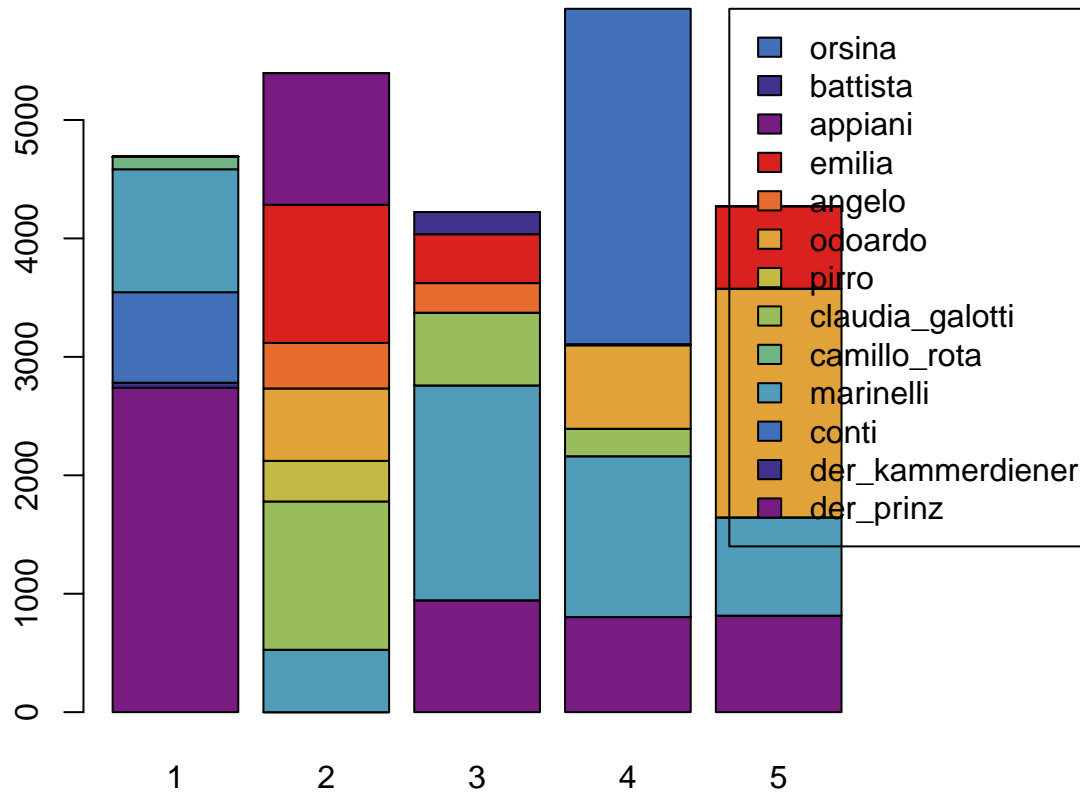
```
mat <- as.matrix(c)
```

```

par(mar=c(2,2,2,10))

barplot(mat,
  legend.text = c$character, # legend text
  args.legend = list(x=ncol(mat)+3, # legend x position
                    y=max(colSums(mat)) # legend y pos
                    ),
  col=qd.colors)

```



### 5.1.1 Filtering unimportant characters

This is informative, but doesn't look very nice and some colors are difficult to associate with characters because colors are repeating. We will therefore use the function `filter()`, which is very similar to `format()`: It can be applied to any object of the type `QDHasCharacter` and removes rows according to certain criteria. In this case, we filter every character except the five most talkative ones. As usual, see `?filter.QDHasCharacter` to see other options.

```

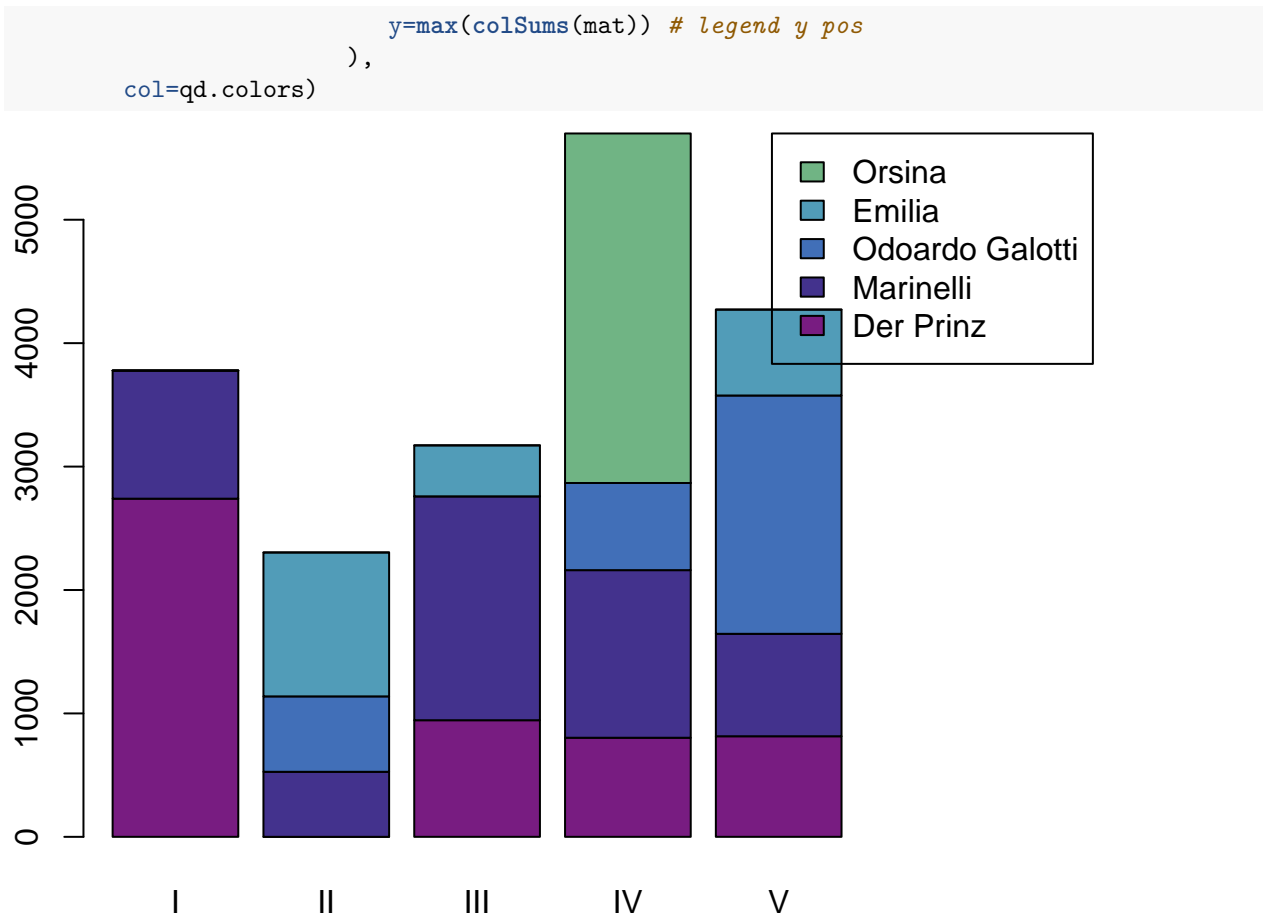
c <- configuration(rksp.0) %>%
  filter(rksp.0, threshold = 5) %>%
  format(rksp.0)

mat <- as.matrix(c)

par(mar=c(2,2,2,10))

barplot(mat,
  names.arg = as.roman(1:ncol(mat)), # roman act numbers
  legend.text = c$character, # legend text
  args.legend = list(x=ncol(mat)+3, # legend x position

```



Of course, the speech of characters that do not speak often is now removed, leaving only a portion of each act covered in the plot.

### 5.1.2 Normalization

Since each act has a different length, it is often useful to normalize each block, according to the total number of spoken tokens. This way, we can display the relative active presence of each character in each act. In combination with the filtering we did before, however, we need to be careful: If the scale with the filtered matrix, a certain portion of the character speech gets lost. The following snippet shows how to scale using the original matrix, but still only include the top eight characters into the matrix.

```

c <- configuration(rksp.0) # getting the full configuration (with all characters)
mat_orig <- as.matrix(c)   # extracting the matrix

c_filtered <- c %>% filter(rksp.0, threshold=8) # filtering, so that only the top 8 characters remain
mat_filtered <- as.matrix(c_filtered) # extracting the filtered matrix

# scaling, using values from the unfiltered matrix
mat <- scale(mat_filtered,
             center=FALSE,
             scale=colSums(mat_orig))

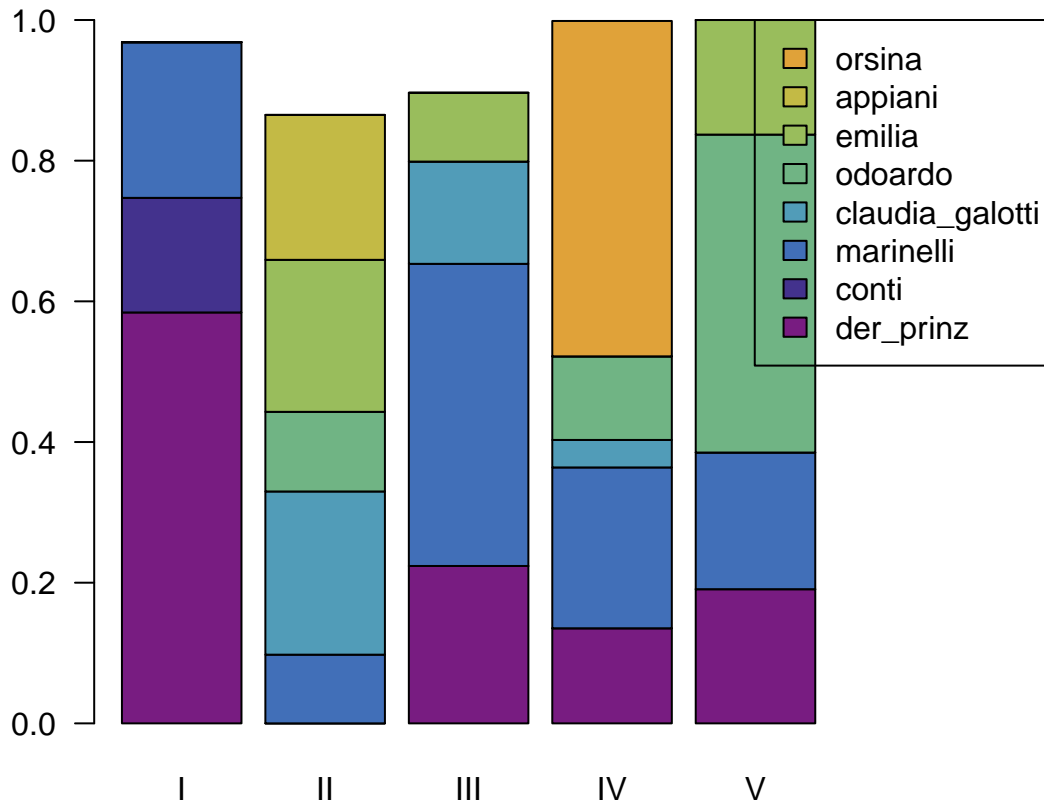
# plot, as above
par(mar=c(2,3,2,10))
barplot(mat,

```

```

names.arg = as.roman(1:ncol(mat)), # roman act numbers
# legend
legend.text=c_filtered$character, # set legend text
args.legend = list(x=ncol(mat)+3, # legend x position
                  y=max(colSums(mat)) # legend y pos
                  ),
# visual
las=1, # rotate labels on y axis
col=qd.colors # provide colors
)

```



## 5.2 Copresence

Configuration matrices are also often used to get an overview of who is copresent on stage. This can also be achieved using the function `configuration()`. First, we create a configuration matrix that only represents presence or absence of a figure (and we switch to scenes). Obviously, the resulting matrix has many more columns, we include only the first 10 below.

```
configuration(rksp.0, onlyPresence = TRUE, segment="Scene")
```

##	corpus	drama	character	1	2	3	4	5	6
## 1	test	rksp.0	der_prinz	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
## 2	test	rksp.0	der_kammerdiener	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
## 4	test	rksp.0	conti	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
## 9	test	rksp.0	marinelli	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
## 14	test	rksp.0	camillo_rota	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
## 15	test	rksp.0	claudia_galotti	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
## 16	test	rksp.0	pirro	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

```
## 17 test rksp.0      odoardo FALSE FALSE FALSE FALSE FALSE FALSE
## 20 test rksp.0      angelo  FALSE FALSE FALSE FALSE FALSE FALSE
## 25 test rksp.0      emilia  FALSE FALSE FALSE FALSE FALSE FALSE
## 27 test rksp.0      appiani FALSE FALSE FALSE FALSE FALSE FALSE
## 47 test rksp.0      battista FALSE FALSE FALSE FALSE FALSE FALSE
## 65 test rksp.0      orsina  FALSE FALSE FALSE FALSE FALSE FALSE
##          7
## 1  TRUE
## 2  TRUE
## 4  FALSE
## 9  FALSE
## 14 FALSE
## 15 FALSE
## 16 FALSE
## 17 FALSE
## 20 FALSE
## 25 FALSE
## 27 FALSE
## 47 FALSE
## 65 FALSE
```

Creating a co-occurrence matrix is a simple matter of matrix multiplication, and we already know how to create a matrix.

```
# extract the configuration
c <- configuration(rksp.0, onlyPresence = TRUE, segment="Scene")

# extract a matrix
mat <- as.matrix(c)

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

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

copresence
```

```
##          der_prinz der_kammerdiener conti marinelli camillo_rota
## der_prinz          17             2      2           9           1
## der_kammerdiener    2             2      0           0           0
## conti              2             0      2           0           0
## marinelli          9             0      0          19           0
## camillo_rota       1             0      0           0           1
## claudia_galotti    0             0      0           3           0
## pirro              0             0      0           1           0
## odoardo            2             0      0           4           0
## angelo             0             0      0           1           0
## emilia             2             0      0           4           0
## appiani           0             0      0           2           0
## battista          1             0      0           3           0
## orsina             1             0      0           3           0
##          claudia_galotti pirro odoardo angelo emilia appiani
```

## der_prinz	0	0	2	0	2	0
## der_kammerdiener	0	0	0	0	0	0
## conti	0	0	0	0	0	0
## marinelli	3	1	4	1	4	2
## camillo_rota	0	0	0	0	0	0
## claudia_galotti	13	3	3	0	3	4
## pirro	3	4	1	1	0	1
## odoardo	3	1	12	0	2	0
## angelo	0	1	0	2	0	0
## emilia	3	0	2	0	7	1
## appiani	4	1	0	0	1	5
## battista	2	0	0	0	1	0
## orsina	1	0	3	0	0	0
##	battista	orsina				
## der_prinz	1	1				
## der_kammerdiener	0	0				
## conti	0	0				
## marinelli	3	3				
## camillo_rota	0	0				
## claudia_galotti	2	1				
## pirro	0	0				
## odoardo	0	3				
## angelo	0	0				
## emilia	1	0				
## appiani	0	0				
## battista	4	0				
## orsina	0	6				

The resulting copresence matrix shows the number of scenes in which two characters are both present. The diagonal shows the number of scenes in which a character is present in total (because each character is always copresent with itself, so to speak).

There are multiple ways to visualise this copresence. One option is a heat map, as shown below.

### 5.2.1 As Heatmap

The copresence can be visualised in a simple heat map. We first focus on the lower triangle and also remove the diagonal values. The actual plotting is a bit more complicated in this case, because we are just using the values in the copresence matrix as pixel intensities in the plot. Also, the axes need to be suppressed first, and can be added later with the proper names of the characters. If needed the code can also be used to include labels into the heat map.

```
c <- configuration(rksp.0, onlyPresence = TRUE, segment="Scene") %>%
  filter(rksp.0, threshold = 7) %>%
  format(rksp.0)

# extract a matrix
mat <- as.matrix(c)

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

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

```

colnames(copresence) <- c$character

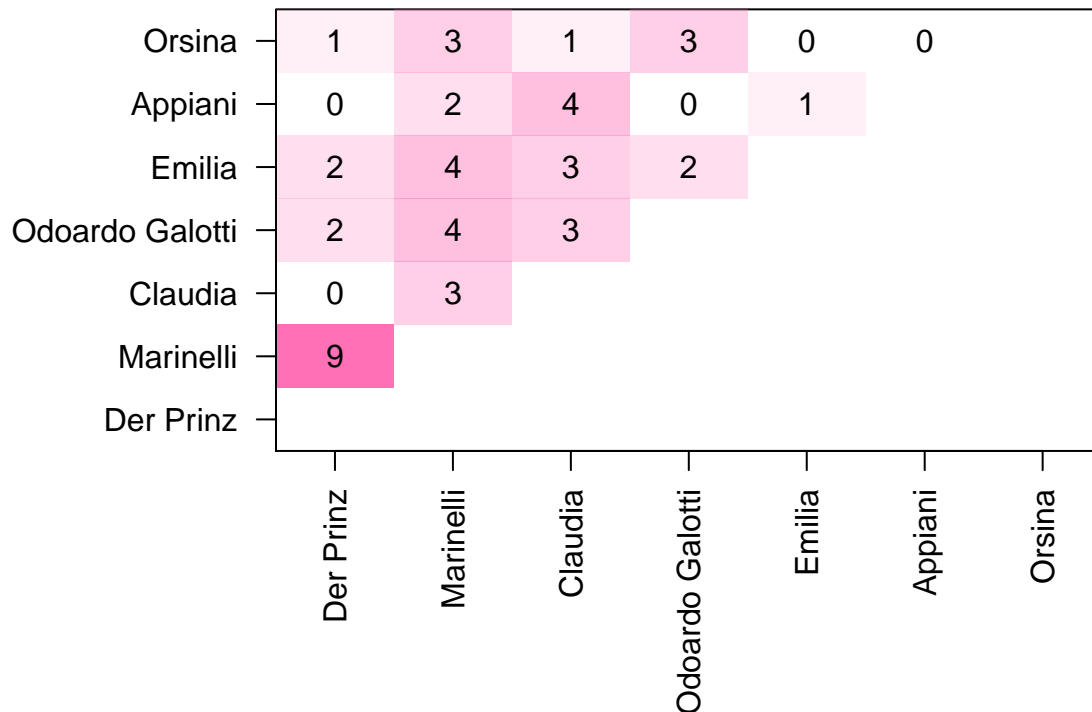
# since it's a square matrix, we don't need the bottom right triangle
# and diagonals.
copresence[lower.tri(copresence,diag=TRUE)] <- NA

par(mar=c(10,10,1,1)) # plot margins
image(copresence,
      col = rgb(256,111,184, alpha=(seq(0,255))),
      maxColorValue = 256),
      xaxt= "n", # no x axis
      yaxt= "n", # no y axis
      frame=TRUE # print a frame around the heatmap
    )

# include values as labels
text(y=(rep(1:ncol(copresence), each=nrow(copresence))-1)/(nrow(copresence)-1),
     x=(1:nrow(copresence)-1)/(nrow(copresence)-1),
     labels=as.vector(copresence))

# add the x axis
axis(1, at = seq(0,1,length.out = length(c$character)), labels = c$character, las=3)
# add the y axis
axis(2, at = seq(0,1,length.out = length(c$character)), labels = c$character, las=1)

```



Apparently, Marinelli and the prince have the most shared scenes. Marinelli also shares a scene with most other figures (sum of the vertical bar: 16).

### 5.2.2 As Network

The same information can also be visualized as a copresence network. For this, we employ the R-package `igraph`. A nice introduction to `igraph` can be found in (Arnold and Tilton, 2015), particularly for literary networks.

Technically, the matrix we created before is an adjacency matrix. It is therefore simple to convert it to a graph, and `igraph` offers the function `graph_from_adjacency_matrix()` for this.

```
c <- configuration(rksp.0, onlyPresence = TRUE, segment="Scene") %>%
  filter(rksp.0, threshold = 7) %>%
  format(rksp.0)

# extract a matrix
mat <- as.matrix(c)

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

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

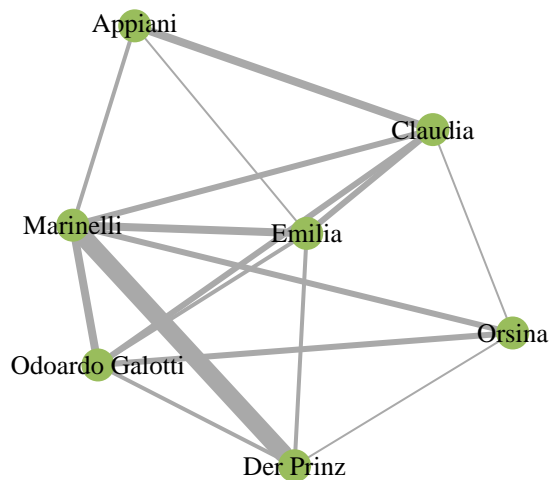
# convert the adjacency matrix to a graph object
g <- igraph::graph_from_adjacency_matrix(copresence,
                                         weighted=TRUE,      # weighted graph
                                         mode="undirected",    # no direction
                                         diag=FALSE           # no looping edges
                                         )
```

Now the variable `g` holds the graph object. There are different things we can do with the graph. First, we can visualise it. `igraph` uses the same mechanism of R that we have used before for plotting, specifying a `plot()` function that can plot graph objects.

```
# Now we plot
plot(g,
      layout=layout_with_gem,      # how to lay out the graph
      main="Copresence Network: Emilia Galotti", # title
      vertex.label.cex=0.8,        # label size
      vertex.label.color="black",  # font color
      vertex.color=qd.colors[6],   # vertex color
      vertex.frame.color=NA,       # no vertex border
      edge.width=E(g)$weight       # scale edges according to their weight
)
```



## Copresence Network: Emilia Galotti



### 5.2.3 Network properties

`igraph` offers a number of function to extract graph properties. We will show them below without explanation. The documentation of each function can be found [here](#).

#### 5.2.3.1 Density

Function: `graph.density()`

```
graph.density(g)
```

```
## [1] 0.7619048
```

#### 5.2.3.2 Average nearest neighbor degree

Function: `knn()`

```
knn(g)
```

```
## $knn
##      Der Prinz      Marinelli      Claudia Odoardo Galotti
##      4.214286      2.760000      4.714286      5.214286
##      Emilia      Appiani      Orsina
##      6.166667      7.285714      8.375000
##
## $knnk
## [1]      NaN      NaN 7.285714 6.294643 5.365079 2.760000
```

#### 5.2.3.3 Edge connectivity

Function: `edge_connectivity()`

```
edge_connectivity(g)
```

```
## [1] 3
```

### 5.2.4 Graph Export

As a final step, one might want to further work on the graph using Gephi, or other tools. In order to do so, one can export the graph into an appropriate file:

```
igraph::write_graph(g,
                    "rksp.0.graphml",    # specify the file name
                    format="graphml"     # specify the file format
)
```

This results in a file called `rksp.0.graphml`, that starts similarly as this:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
                              http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <!-- Created by igraph -->
  <key id="v_name" for="node" attr.name="name" attr.type="string"/>
  <key id="e_weight" for="edge" attr.name="weight" attr.type="double"/>
  <graph id="G" edgedefault="undirected">
    <node id="n0">
      <data key="v_name">DER KAMMERDIENER</data>
    </node>
    <node id="n1">
      <data key="v_name">DER PRINZ</data>
    </node>
    ...
```

This file can be opened with Gephi.

## Chapter 6

# Character Exchange

Character exchange – entering and leaving the stage – often takes place at scene boundaries. We offer multiple ways to measure and visualize this exchange over the course of the play.

### 6.1 Hamming Distance

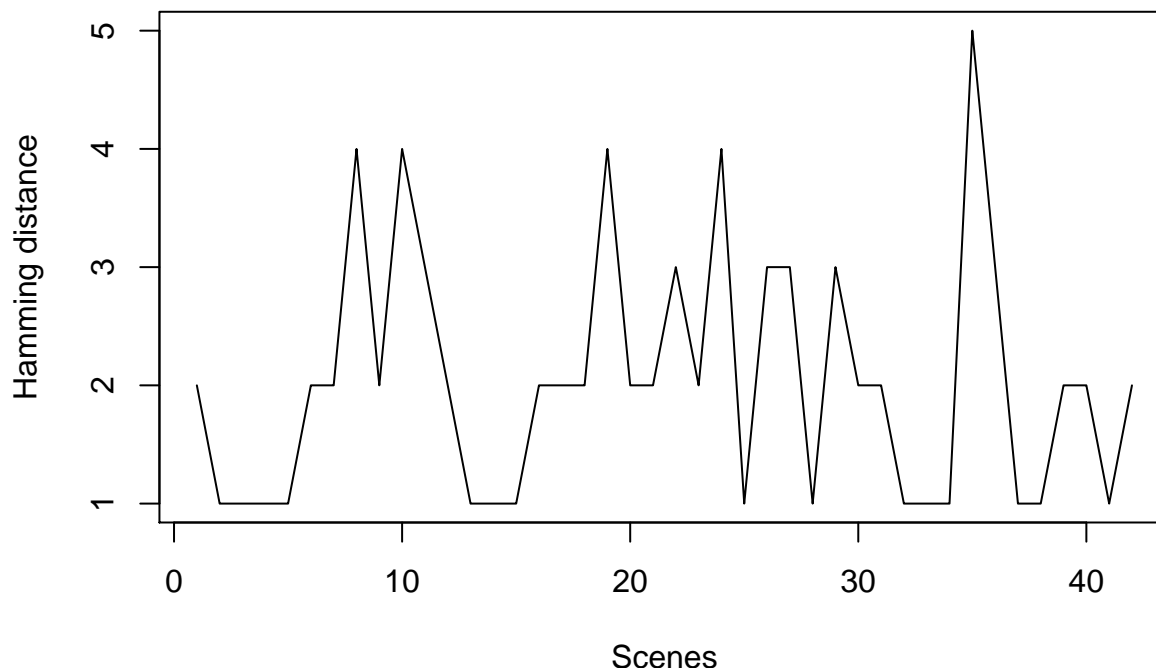
The hamming distance has been introduced in information theory by Richard Hamming (Hamming, 1950). Intuitively, the hamming distance expresses the number of characters that either enter or leave the stage. Its application to a dramatic text is straightforward, using the function `hamming()`. By default, the hamming distance is calculated over scene boundaries.

```
hamming(rksp.0, variant="Hamming")
```

```
## [1] 2 1 1 1 1 2 2 4 2 4 3 2 1 1 1 2 2 2 4 2 2 3 2 4 1 3 3 1 3 2 2 1 1 1 5
## [36] 3 1 1 2 2 1 2
```

The return value of the function is just a vector of numbers, one less than the number of scenes in the play. Of course, this can be plotted:

```
plot(hamming(rksp.0, variant="Hamming"),
     type="l",
     ylab="Hamming distance",
     xlab="Scenes")
```



Hamming distance, in its original definition as well as the `Hamming` variant in our function, returns absolute values. As usual, these values should be normalized, and the function `hamming()` provides two ways of doing that.

### 6.1.1 NormalizedHamming

Straightforwardly, one can normalise by the overall number of characters in the play. This value is only maximal, if at once scene boundary, one subset of characters enters the stage, and the entire other subset leaves the stage. As we can see below, this is not the case in our demo text.

```
hamming(rksp.0, variant="NormalizedHamming")
```

```
## [1] 0.15384615 0.07692308 0.07692308 0.07692308 0.07692308 0.15384615
## [7] 0.15384615 0.30769231 0.15384615 0.30769231 0.23076923 0.15384615
## [13] 0.07692308 0.07692308 0.07692308 0.15384615 0.15384615 0.15384615
## [19] 0.30769231 0.15384615 0.15384615 0.23076923 0.15384615 0.30769231
## [25] 0.07692308 0.23076923 0.23076923 0.07692308 0.23076923 0.15384615
## [31] 0.15384615 0.07692308 0.07692308 0.07692308 0.38461538 0.23076923
## [37] 0.07692308 0.07692308 0.15384615 0.15384615 0.07692308 0.15384615
```

### 6.1.2 Trilcke

One more possible normalization has been proposed by Trilcke et al. (2017). Instead of normalizing with all characters in the play, the Trilcke variant only normalizes with the characters in the two adjacent scenes. Thus, if all characters leave the stage, and a new set of characters enter it, the distance is maximal. This however does not have to include *all* characters in the play.

```
hamming(rksp.0, variant="Trilcke")
```

```
## [1] 0.6666667 0.5000000 0.5000000 0.5000000 0.5000000 0.6666667 0.6666667
## [8] 1.0000000 0.6666667 1.0000000 0.7500000 0.6666667 0.5000000 0.3333333
## [15] 0.3333333 0.5000000 0.5000000 0.6666667 1.0000000 0.6666667 0.6666667
## [22] 0.7500000 0.5000000 0.8000000 0.3333333 0.7500000 0.7500000 0.3333333
## [29] 0.7500000 0.6666667 0.6666667 0.3333333 0.3333333 0.3333333 1.0000000
```

```
## [36] 1.0000000 0.5000000 0.5000000 0.6666667 0.6666667 0.5000000 0.5000000
```

The Trilcke variant is the default setting of the `hamming()`-function.



# Chapter 7

## Text Collections

So far, we have mostly processed single texts. In this chapter, we will cover the important things to know when processing collections of texts – be it genres or the works of an author.

The following code can only be reproduced if the GerDraCor data set and the collection repository have been installed. This can be done with the following code (assuming an internet connection).

```
installData("gdc")

## Pulling new data from https://github.com/quadrama/data_gdc.git.
## Already up-to-date

installCollectionData()

## Warning in utils::unzip(tf, exdir = dataDirectory): Fehler 1 während des
## Extrahierens aus Zipfile

## Warning in file.rename(from = file.path(repoDirectory, "collections"), to =
## getOption("qd.collectionDirectory")): kann Datei '/Users/reiterns/QuaDrama/
## Data2/metadata-master/collections' nicht in '/Users/reiterns/QuaDrama/
## Data2/collections' umbenennen. Grund 'No such file or directory'
```

### 7.1 Defining collections

Before processing, it's necessary to define a collection of texts, by assembling their ids in a list. These are considered to be sets of plays without internal structure (e.g., no play is marked as prototypical).

```
# load ids that are comedies and tragedies
ids.tragedy <- unique(loadSet(c("trauerspiel", "tragödie", "tragoedie")))
ids.comedy <- unique(loadSet(c("lustspiel", "komödie", "komoedie")))

# make sure we're using the GerDraCor corpus
ids.tragedy <- sub("tg:", "gdc:", ids.tragedy, fixed = TRUE)
ids.comedy <- sub("tg:", "gdc:", ids.comedy, fixed = TRUE)
```

### 7.2 Loading multiple plays

Loading multiple plays at once is straightforward, and not different from loading a single one.





## Part II

# Text Analysis



## Chapter 8

# Word Field Analysis

What we consider a word field here may differ from specific uses in linguistics. In this context, a word field is a list of words that belong to a common theme / topic / semantic group. Multiple word fields can be assembled to create a dictionary. On a technical level, what we describe in the following works for arbitrary lists of words. A semantic relation between the words is technically not required. Thus, the following pieces of code can be used with arbitrary word lists.

For **demo purposes** (this is really a toy example), we will define the word field of **Love** as containing the words **Liebe** (love) and **Herz** (heart). In R, we can put them in a character vector, and we use lemmas:

```
wf_love <- c("liebe", "herz")
```

We will test this word field on Emilia Galotti, which should be about love.

```
data("rksp.0")
```

### 8.1 Single word field

The core of the word field analysis is collecting statistics about a dictionary. Therefore, we use the function called `dictionaryStatisticsSingle()` (single, because we only want to analyse a single word field):

```
dictionaryStatisticsSingle(  
  rksp.0,          # the text we want to process  
  wordfield=wf_love # the word field  
)
```

```
##   corpus  drama      character x  
## 1  test rksp.0      angelo 0  
## 2  test rksp.0      appiani 0  
## 3  test rksp.0      battista 0  
## 4  test rksp.0      camillo_rota 0  
## 5  test rksp.0      claudia_galotti 2  
## 6  test rksp.0      conti 3  
## 7  test rksp.0      der_kammerdiener 0  
## 8  test rksp.0      der_prinz 8  
## 9  test rksp.0      emilia 4  
## 10 test rksp.0      marinelli 9  
## 11 test rksp.0      odoardo 6  
## 12 test rksp.0      orsina 6  
## 13 test rksp.0      pirro 0
```

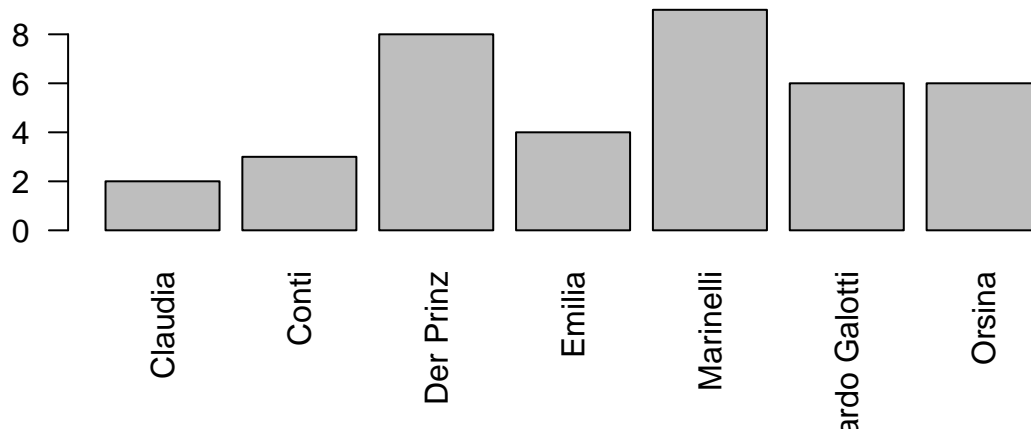
What this table shows us the number of times the characters in the play use words that appear in this list. By default, these are absolute numbers.

We can visualise these counts in a simple bar chart:

```
# retrieve counts and replace character ids by names
dstat <- dictionaryStatisticsSingle(rksp.0, wordfield = wf_love) %>%
  format(rksp.0)

# remove figures not using these words at all
dstat <- dstat[dstat$x>0,]

# create a bar plot
barplot(dstat$x,                                # what to plot
        names.arg = dstat$character,           # x axis labels
        las=2                                   # turn axis labels
)
```



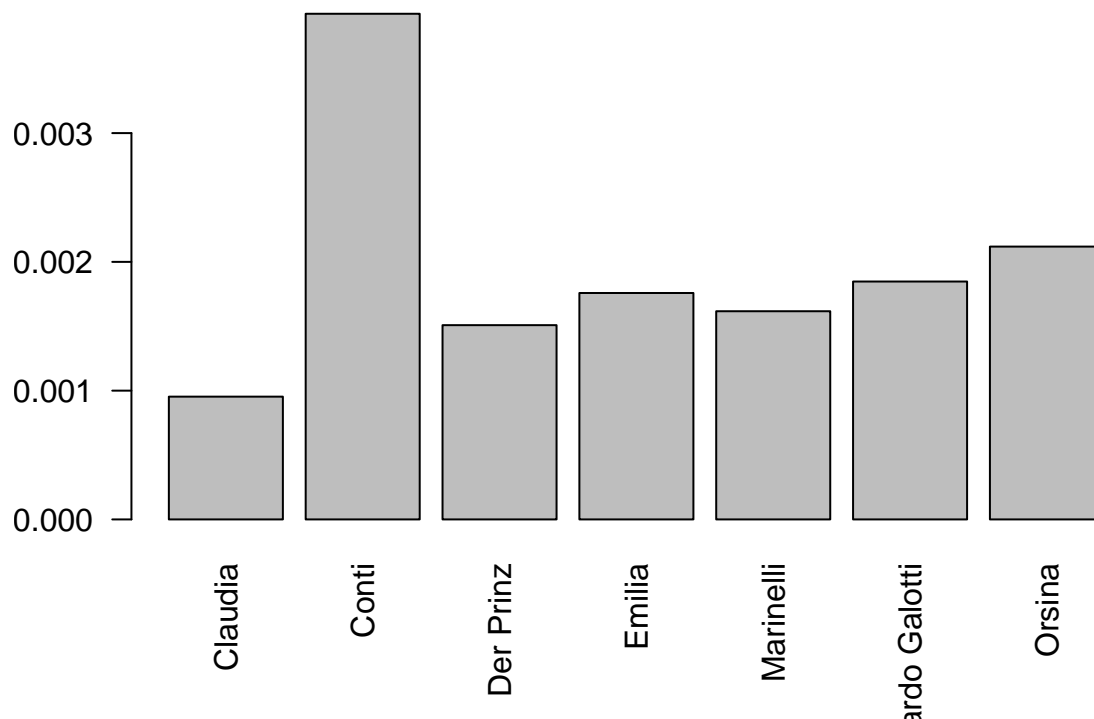
Apparently, the prince and Marinelli are mentioning these words a lot more than the other characters.

When comparing characters, it often (but not always) makes sense to normalize the counts according to the total number of spoken words by a character. This can be enabled by setting the argument `normalizeByFigure=TRUE`. This will divide the number of words in this field by the total number of words a character speaks.

```
dstat <- dictionaryStatisticsSingle(rksp.0, wordfield=wf_love,
  normalizeByFigure = TRUE # apply normalization
) %>% format(rksp.0)      # reformat character names

# remove figures not using these words at all
dstat <- dstat[dstat$x>0,]

barplot(dstat$x,
        names.arg = dstat$character,
        las=2
)
```



## 8.2 Multiple Word Fields

The function `dictionaryStatistics()` can be used to analyse multiple dictionaries at once. To this end, dictionaries are represented as lists of character vectors. The (named) outer list contains the keywords, the vectors are just words associated with the keyword.

New dictionaries can be easily created like this:

```
wf <- list(Liebe=c("liebe", "herz", "schmerz"), Hass=c("hass", "hassen"))
```

This dictionary contains the two entries `Liebe` (love) and `Hass` (hate), with 3 respective 2 entries. Dictionaries can be created in code, like shown above. In addition, the function `loadFields()` can be used to download dictionary content from a URL or a directory. By default, the function loads this dictionary from GitHub (that we used in publications), for the keywords `Liebe` and `Familie` (family). Since version 2.3.0, this dictionary is included in the package as `base_dictionary` and can be used right away (without internet connection). It is also the default dictionary used by `dictionaryStatistics()`.

The function `loadFields()` offers parameters to load from different URLs via http or to load from plain text files that are stored locally. The latter can be achieved by specifying the directory as `baseurl`. Entries for each keyword should then be stored in a file named like the keyword, and ending with `txt` (by default, can be overwritten). See `?loadFields` for details. Some of the options can also be specified through `dictionaryStatistics()`, as exemplified below.

The following examples use the `base_dictionary`, i.e., a specific version of the fields we have been using in `QuaDramA`.

```
dstat <- dictionaryStatistics(
  rksp.0, # the text
  fieldnames = # fields we want to measure (from base_dictionary)
    c("Liebe", "Familie", "Ratio", "Krieg", "Religion"),
  normalizeByFigure = TRUE, # normalization by figure
  normalizeByField = TRUE # normalization by field
```

```
)
dstat
```

```
## corpus drama character Liebe Familie Ratio
## 1 test rksp.0 angelo 5.025967e-05 4.314529e-05 7.290755e-05
## 2 test rksp.0 appiani 8.610133e-05 1.355080e-04 6.661338e-05
## 3 test rksp.0 battista 0.000000e+00 1.404988e-04 0.000000e+00
## 4 test rksp.0 camillo_rota 1.003613e-04 1.292324e-04 4.367575e-04
## 5 test rksp.0 claudia_galotti 6.591891e-05 2.350575e-04 3.972037e-05
## 6 test rksp.0 conti 1.531692e-04 5.379043e-05 3.635835e-05
## 7 test rksp.0 der_kammerdiener 0.000000e+00 0.000000e+00 0.000000e+00
## 8 test rksp.0 der_prinz 7.021317e-05 6.974600e-05 4.714313e-05
## 9 test rksp.0 emilia 7.014262e-05 3.311757e-04 4.884005e-05
## 10 test rksp.0 marinelli 8.026020e-05 1.451804e-04 3.991777e-05
## 11 test rksp.0 odoardo 5.895608e-05 2.319657e-04 6.271666e-05
## 12 test rksp.0 orsina 5.259046e-05 8.223048e-05 4.250366e-05
## 13 test rksp.0 pirro 0.000000e+00 1.597508e-04 2.699492e-05
## Krieg Religion
## 1 5.624297e-05 0.000000e+00
## 2 4.817575e-05 0.000000e+00
## 3 0.000000e+00 0.000000e+00
## 4 8.423181e-05 0.000000e+00
## 5 4.681329e-05 5.853528e-05
## 6 2.337322e-05 0.000000e+00
## 7 0.000000e+00 0.000000e+00
## 8 2.862261e-05 3.308290e-05
## 9 3.139717e-05 8.482745e-05
## 10 3.528446e-05 1.890842e-05
## 11 4.123417e-05 4.861291e-05
## 12 2.522195e-05 1.238973e-05
## 13 0.000000e+00 1.022966e-04
```

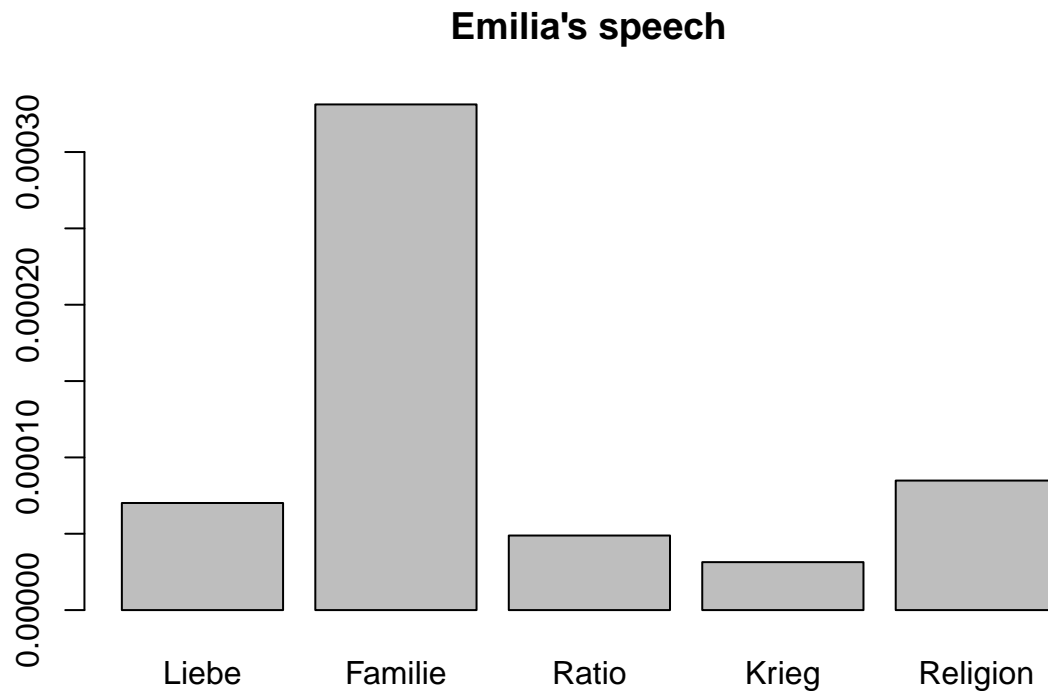
The variable `dstat` now contains multiple columns, one for each word field. We have been using the option `normalizeByFigure` before. When comparing multiple fields, it often happens that they have a different size (i.e., different number of words). In this case, it makes sense to *also* normalize with the number of words in the word field. This is achieved by `normalizeByField=TRUE`. This makes the numbers very small, but they should be used in comparison anyway.

### 8.2.1 Bar plot by character

It is now straightforward to show the distribution of fields for a single character:

```
mat <- as.matrix(dstat)

barplot(mat[9,],      # we select Emilia's line
        main="Emilia's speech",
        names.arg=colnames(mat)
)
```



### 8.2.2 Bar plot by field

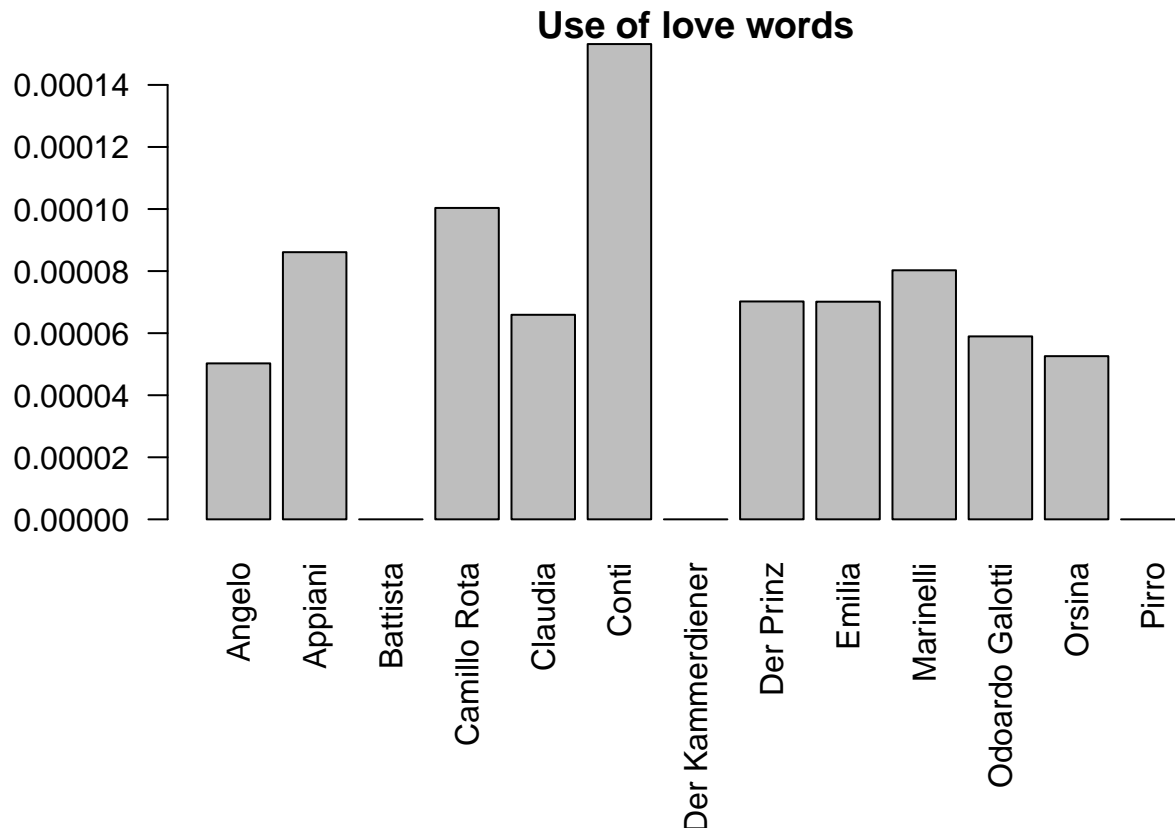
Conversely, we can also show who uses words of a certain field how often:

```
dstat <- dictionaryStatistics(
  rksp.0, # the text
  fieldnames = # fields we want to measure (from base_dictionary)
    c("Liebe", "Familie", "Ratio", "Krieg", "Religion"),
  normalizeByFigure = TRUE, # normalization by figure
  normalizeByField = TRUE # normalization by field
) %>%
  format(rksp.0)

mat <- as.matrix(dstat)

par(mar=c(9,4,1,1))

barplot(mat[,1], # Select the row for 'love'
  main="Use of love words", # title for plot
  beside = TRUE, # not stacked
  names.arg = dstat$character, # x axis labels
  las=2 # rotation for labels
)
```



### 8.3 Dictionary Based Distance

Technically, the output of `dictionaryStatistics()` is a `data.frame`. This is suitable for most uses. In some cases, however, it's more suited to work with a matrix that only contains the raw numbers (i.e., number of family words). Calculating character distance based on dictionaries, for instance. For these cases, the package provides an S3 method that extracts the numeric part of the `data.frame` and creates a matrix. We have used this function `as.matrix()` already above.

The matrix doesn't have row names by default. The snippet below can be used to add row names.

```
ds <- dictionaryStatistics(rksp.0,
                          fieldnames=c("Liebe", "Familie", "Ratio", "Krieg", "Religion"),
                          normalizeByFigure=TRUE)

m <- as.matrix(ds)
rownames(m) <- ds$character
m
```

	Liebe	Familie	Ratio	Krieg
angelo	0.004724409	0.003149606	0.007874016	0.006299213
appiani	0.008093525	0.009892086	0.007194245	0.005395683
battista	0.000000000	0.010256410	0.000000000	0.000000000
camillo_rota	0.009433962	0.009433962	0.047169811	0.009433962
claudia_galotti	0.006196378	0.017159199	0.004289800	0.005243089
conti	0.014397906	0.003926702	0.003926702	0.002617801
der_kammerdiener	0.000000000	0.000000000	0.000000000	0.000000000
der_prinz	0.006600038	0.005091458	0.005091458	0.003205733
emilia	0.006593407	0.024175824	0.005274725	0.003516484
marinelli	0.007544458	0.010598168	0.004311119	0.003951859



```
## odoardo      0.005541872 0.016933498 0.006773399 0.004618227
## orsina       0.004943503 0.006002825 0.004590395 0.002824859
## pirro        0.000000000 0.011661808 0.002915452 0.000000000
##              Religion
## angelo       0.000000000
## appiani      0.000000000
## battista     0.000000000
## camillo_rota 0.000000000
## claudia_galotti 0.003336511
## conti        0.000000000
## der_kammerdiener 0.000000000
## der_prinz    0.001885725
## emilia       0.004835164
## marinelli    0.001077798
## odoardo      0.002770936
## orsina       0.000706214
## pirro        0.005830903
```

Every character is now represented with five numbers, which can be interpreted as a vector in five-dimensional space. This means, we can easily apply distance metrics supplied by the function `dist()` (from the default package `stats`). By default, `dist()` calculates Euclidean distance.

```
cdist <- dist(m)
# output not shown
```

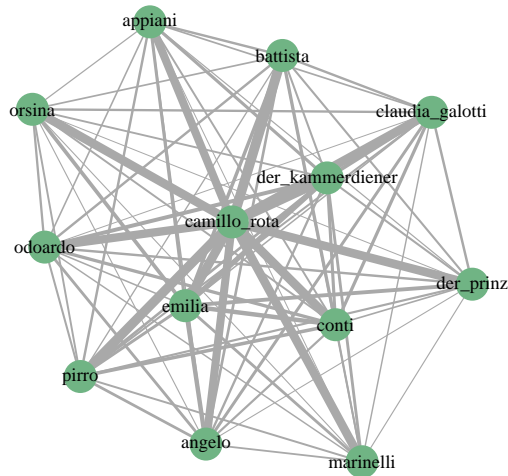
The resulting data structure is similar to the one in the weighted configuration matrix, which means everything from Section 5.2.2 can be applied here. In particular, we can convert these distances into a network:

```
require(igraph)
g <- graph_from_adjacency_matrix(as.matrix(cdist),
                                weighted=TRUE,      # weighted graph
                                mode="undirected",  # no direction
                                diag=FALSE         # no looping edges
                                )
```

This network can of course be visualised again.

```
# Now we plot
plot.igraph(g,
  layout=layout_with_fr,      # how to lay out the graph
  main="Dictionary distance network", # title
  vertex.label.cex=0.6,      # label size
  vertex.label.color="black", # font color
  vertex.color=qd.colors[5], # vertex color
  vertex.frame.color=NA,     # no vertex border
  edge.width=E(g)$weight*100 # scale edges according to their weight
                              # (since the distances are so small, we multiply)
)
```

## Dictionary distance network



Although this network is similar to the one shown in Section 5.2.2 (both undirected and weighted), it displays a totally different kind of information. The networks based on copresence connect characters that appear together on stage, while this network connects characters with similar thematic profile in their speech (within the limits of being able to detect thematic profiles via word fields).

## 8.4 Development over the course of the play

Finally, the function `dictionaryStatistics()` can be used to track word field for segments of the play. To do that, we use the parameter `segment`, and set it to either “Act” or “Scene”.

### 8.4.1 Individual characters

We can now plot the theme progress over the course of the play. This can be done for specific characters, as shown below.

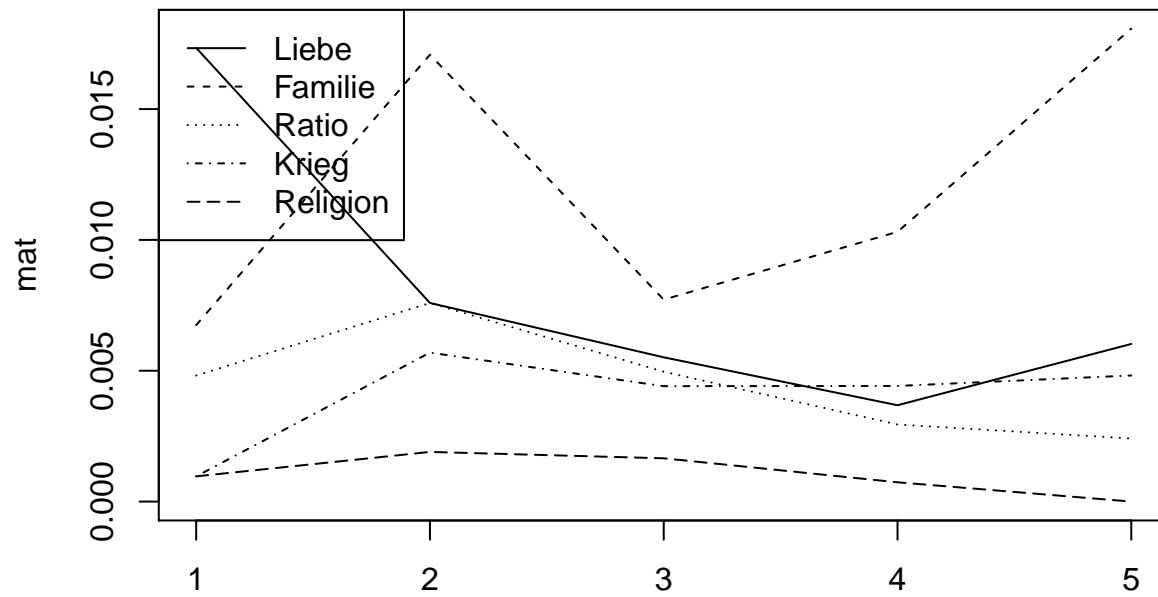
```

dsl <- dictionaryStatistics(rksp.0,
                           fieldnames=c("Liebe", "Familie", "Ratio", "Krieg", "Religion"),
                           normalizeByFigure=TRUE,
                           segment="Act")

mat <- as.matrix(dsl[dsl$character=="marinelli",])

matplot(mat, type="l", col="black")
legend(x="topleft", legend=colnames(mat), lty=1:ncol(dsl))

```



Depending on the use case, it might be easier to interpret if the numbers are cumulatively added up. The snippet below shows how this works.

```

dsl <- dictionaryStatistics(rksp.0,
                           fieldnames=c("Liebe", "Familie", "Ratio", "Krieg", "Religion"),
                           normalizeByFigure=TRUE,
                           segment="Act")

mat <- as.matrix(dsl[dsl$character=="marinelli",])

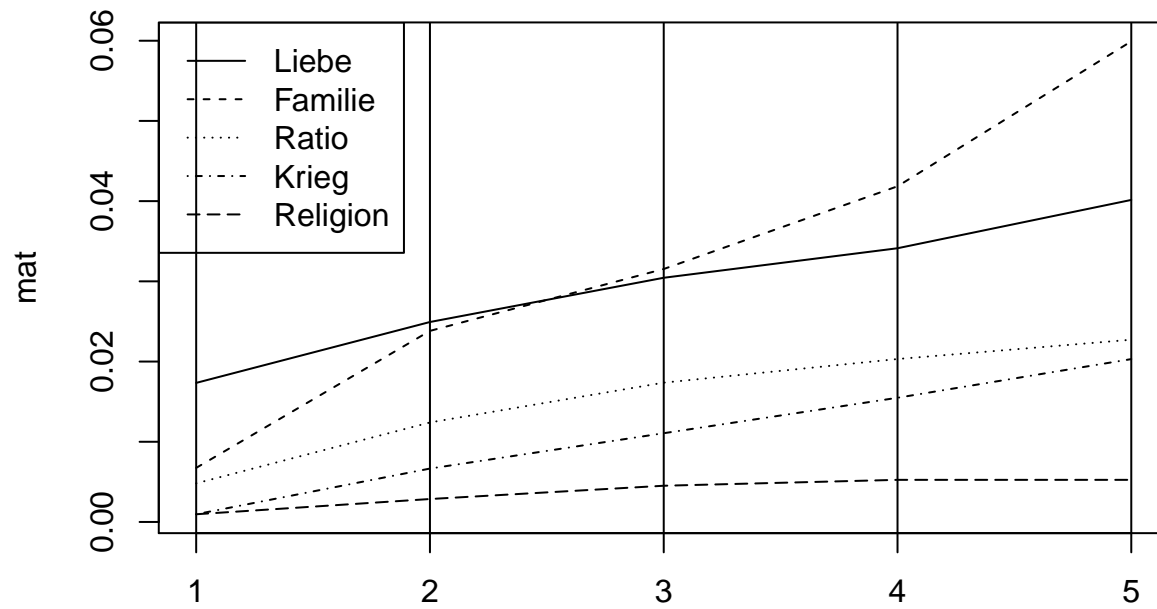
mat <- apply(mat,2,cumsum)

matplot(mat, type="l", col="black")

# Add act lines
abline(v=1:nrow(mat))

legend(x="topleft", legend=colnames(mat), lty=1:5)

```



### 8.4.2 Comparing characters

Simultaneously to the setting above, we now want to compare the development of several characters for a single word field. This unfortunately requires some reshuffling of the data, using the function `reshape` (from the `stats` package).

```

dsl <- dictionaryStatistics(rksp.0,
                           fieldnames=c("Liebe"),
                           normalizeByFigure=TRUE,
                           segment="Act") %>%

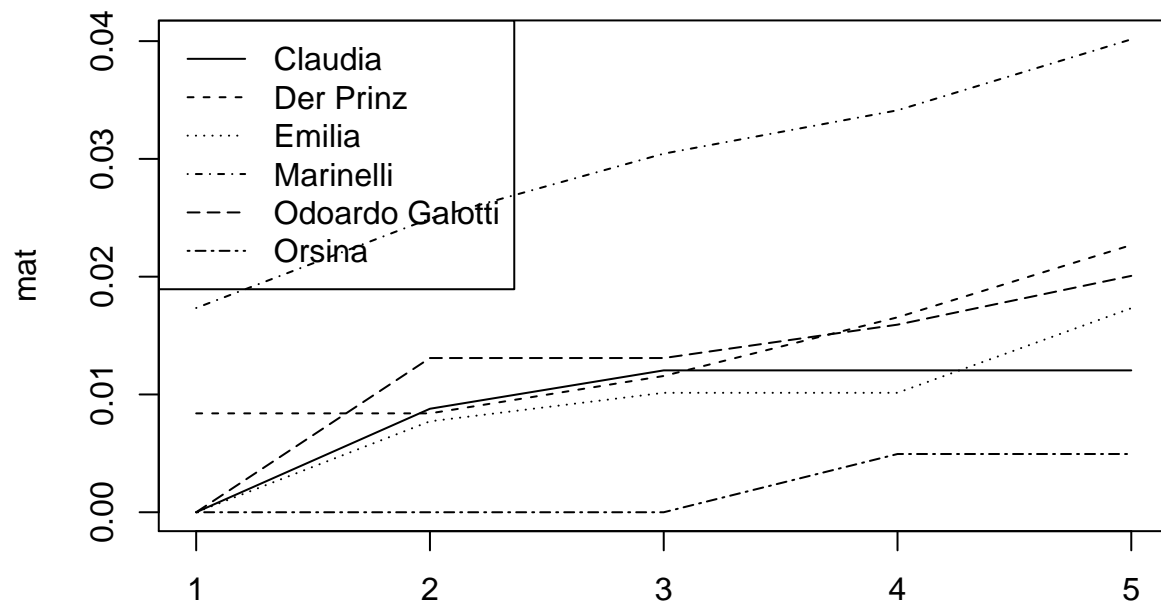
  filter(rksp.0,
         threshold = 6) %>%
  format(rksp.0)

dsl <- reshape(dsl, direction = "wide", # the table becomes wider
               timevar = "Number.Act", # the column that specifies multiple readings
               times = "Liebe",        # the number to distribute
               idvar=c("corpus","drama","character") # what identifies a character
)

mat <- as.matrix.data.frame(dsl[,4:ncol(dsl)])
rownames(mat) <- dsl$character
mat <- apply(mat,1,cumsum)

matplot(mat, type="l", lty = 1:ncol(mat), col="black", main="Liebe")
legend(x="topleft", legend=colnames(mat), lty=1:ncol(mat))

```

**Liebe**



## Chapter 9

# Advanced Text Analysis

### 9.1 When are characters mentioned?

When characters are speaking on stage, they are actively present. But they can also be passively present, if other characters refer to them. Both levels of presence can be extracted with the `presence()` function:

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

pres <- presence(rksp.0)
pres
```

##	corpus	drama	character	scenes	actives	passives	presence
## 1	test	rksp.0	angelo	43	2	1	0.02325581
## 2	test	rksp.0	appiani	43	5	14	-0.20930233
## 3	test	rksp.0	battista	43	4	6	-0.04651163
## 4	test	rksp.0	camillo_rota	43	1	2	-0.02325581
## 5	test	rksp.0	claudia_galotti	43	13	14	-0.02325581
## 6	test	rksp.0	conti	43	1	3	-0.04651163
## 7	test	rksp.0	der_kammerdiener	43	1	2	-0.02325581
## 8	test	rksp.0	der_prinz	43	14	21	-0.16279070
## 9	test	rksp.0	emilia	43	7	21	-0.32558140
## 10	test	rksp.0	marinelli	43	19	21	-0.04651163
## 11	test	rksp.0	odoardo	43	12	11	0.02325581
## 12	test	rksp.0	orsina	43	6	10	-0.09302326
## 13	test	rksp.0	pirro	43	4	5	-0.02325581

As we can see, each character has a few numbers associated: The column `actives` shows the number of scenes in which the character is actively present. This is equivalent to the information in the configuration matrix. The column `passives` shows the number of scenes in which a character is mentioned. By default, this excludes the scenes in which they are present themselves (this behaviour can be changed by adding the parameter `passiveOnlyWhenNotActive = TRUE` to the call of the `presence` function).

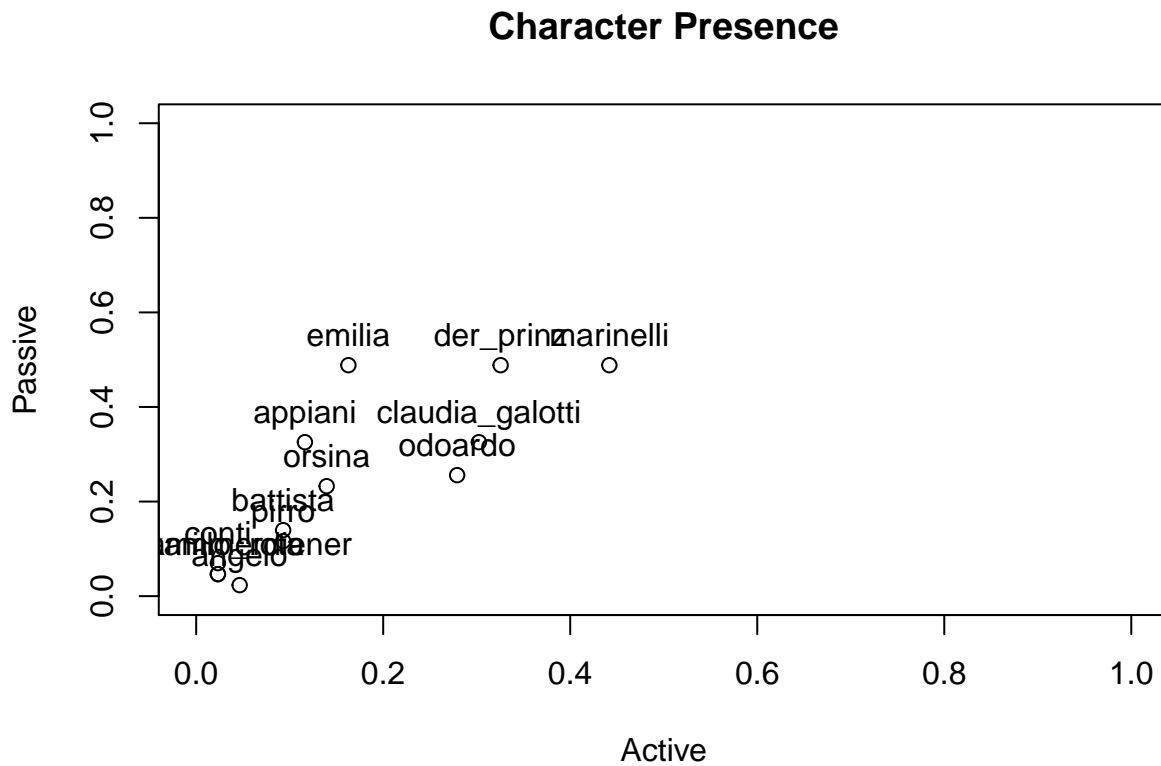
A simple visualisation that shows the characters active and passive presence in one plot can be generated like this: The first line (`plot()`) is responsible for the plotting of the symbols, the second line (`text()`) adds the character names or ids numbers.

```
plot(x=pres$active/pres$scenes,
     y=pres$passive/pres$scenes,
     xlim=c(0,1),
     ylim=c(0,1),
```

```

xlab="Active",
ylab="Passive",
main="Character Presence")
text(x=pres$actives/pres$scenes,
y=pres$passives/pres$scenes,
labels=substr(pres$character,0,20),
pos=3)

```





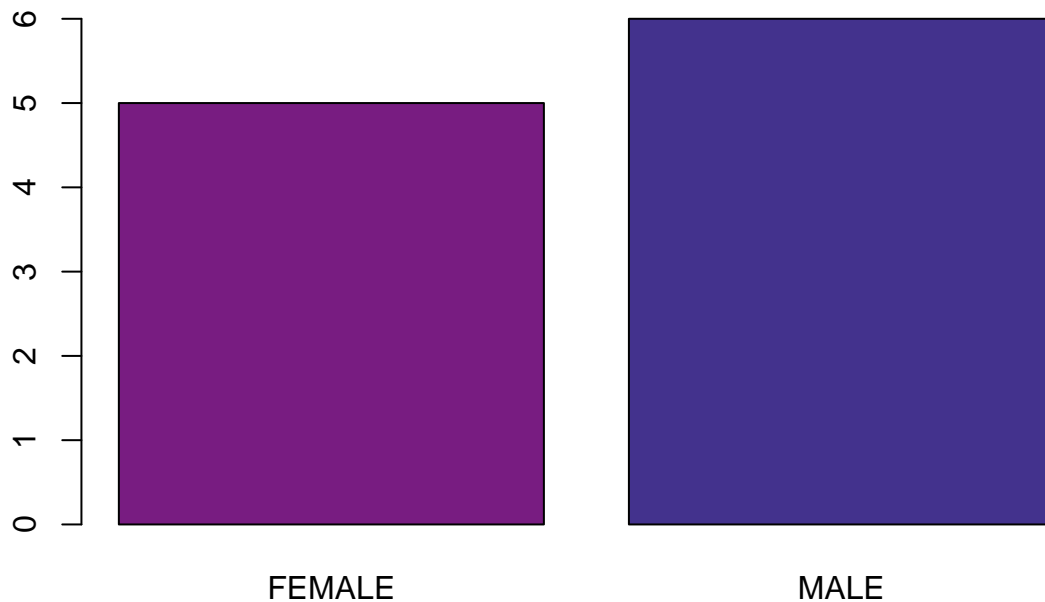
## Chapter 10

# Resterampe

### 10.0.1 Character meta data

We will now combine this information with additional meta data about characters, i.e., gender.

```
# Proportion of male / female characters  
barplot(table(text$characters$Gender),col=qd.colors)
```



### 10.0.2 Character groups

Next, we want to make the same analysis not for individual characters, but for character groups, based on categories such as gender.

```
ustat <- utteranceStatistics(rjmw.0,  
                             normalizeByDramaLength = FALSE # use absolute values  
                             )  
  
characterdata <- rjmw.0$characters  
  
ustat <- merge(ustat, characterdata,  
               by.x = c("corpus", "drama", "character"),
```



# Bibliography

Arnold, T. and Tilton, L. (2015). *Humanities Data in R*. Springer International Publishing.

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147 – 160.

Pfister, M. (1988). *The Theory and Analysis of Drama*. European Studies in English Literature. Cambridge University Press.

Trilcke, P., Fischer, F., Göbel, M., Kampkaspar, D., and Kittel, C. (2017). Netzwerkdynamik, Plotanalyse – Zur Visualisierung und Berechnung der ›progressiven Strukturierung‹ literarischer Texte. In *Book of Abstracts of DHD 2017*, Bern, Switzerland.