

# 3. Textbasierte Daten

## Warum textbasierte Daten?

Bisher haben wir vor allem zahlenbasierte Daten betrachtet, die man beispielsweise in einem Diagramm darstellen und auch empirisch auswerten kann. Für die meisten Anwendungen reicht das vollkommen aus, aber im Internet und auch im Alltag begegnen uns vor allem textbasierte Daten, das heißt Wörter, Sätze, Dokumente und ganze Sammlungen an Texten. In dieser Einheit wagen wir einen Exkurs hin zu dieser Art von Daten, mit denen wir ganz andere Untersuchungen anstellen können als mit Zahlen allein. Zum Beispiel können wir untersuchen:

- Welche Wörter in einer Sorte Text am häufigsten vorkommen und sie mit anderen Texten vergleichen
- Wie die Stimmung (das *Sentiment*) dieser Texte ist oder
- Ob es bei Immobilien einen Zusammenhang zwischen Wortwahl und Kaufpreis gibt.

Um diesen Zielen näher zu kommen, benutzen wir die Datei [angebote\\_1000.csv](#). Sie enthält Daten zu über 200.000 Immobilienangeboten aus den Jahren 2018 und 2019. Diese Daten wurden entnommen aus [diesem](#) Kaggle-Datensatz und stammen von der Seite [immobilien-scout24.de](#).

Zunächst laden wir die Daten in unseren Arbeitsbereich und schauen sie uns an:

## Datenprojekt

### Daten vorbereiten

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.4      v readr      2.1.4
v forcats    1.0.0      v stringr    1.5.1
v ggplot2    3.4.4      v tibble     3.2.1
v lubridate  1.9.3      v tidyr      1.3.0
v purrr      1.0.2
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

```
inserate <- read_csv("data/angebote_1000.csv")
```

```
Rows: 1000 Columns: 49
```

```
-- Column specification -----
Delimiter: ","
chr (20): regio1, heatingType, telekomTvOffer, firingTypes, geo_bln, houseNu...
dbl (23): serviceCharge, telekomHybridUploadSpeed, picturecount, pricetrend,...
lgl (6): newlyConst, balcony, hasKitchen, cellar, lift, garden
```

```
i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
inserate
```

```
# A tibble: 1,000 x 49
```

	regio1	serviceCharge	heatingType	telekomTvOffer	telekomHybridUploadS~1
	<chr>	<dbl>	<chr>	<chr>	<dbl>
1	Nordrhein_We~	245	central_he~	ONE_YEAR_FREE	NA
2	Rheinland_Pf~	134	self_conta~	ONE_YEAR_FREE	NA
3	Sachsen	255	floor_heat~	ONE_YEAR_FREE	10
4	Sachsen	58.2	district_h~	ONE_YEAR_FREE	NA
5	Bremen	138	self_conta~	<NA>	NA
6	Sachsen	70	self_conta~	ONE_YEAR_FREE	10
7	Bremen	88	central_he~	ONE_YEAR_FREE	10
8	Baden_Württe~	110	oil_heating	ONE_YEAR_FREE	NA
9	Nordrhein_We~	95	self_conta~	ONE_YEAR_FREE	NA
10	Sachsen	88	<NA>	ONE_YEAR_FREE	NA

```
# i 990 more rows
```

```
# i abbreviated name: 1: telekomHybridUploadSpeed
```

```
# i 44 more variables: newlyConst <lgl>, balcony <lgl>, picturecount <dbl>,
#   pricetrend <dbl>, telekomUploadSpeed <dbl>, totalRent <dbl>,
#   yearConstructed <dbl>, scoutId <dbl>, noParkSpaces <dbl>,
#   firingTypes <chr>, hasKitchen <lgl>, geo_bln <chr>, cellar <lgl>,
#   yearConstructedRange <dbl>, baseRent <dbl>, houseNumber <chr>, ...
```

Wir können uns auch anschauen, welche Spalten es gibt und welches Format sie haben:

```
inserate %>%
  glimpse()
```

```
Rows: 1,000
Columns: 49
$ regio1          <chr> "Nordrhein-Westfalen", "Rheinland-Pfalz", "Sa~
$ serviceCharge   <dbl> 245.00, 134.00, 255.00, 58.15, 138.00, 70.00,~
$ heatingType     <chr> "central_heating", "self_contained_central_he~
$ telekomTvOffer  <chr> "ONE_YEAR_FREE", "ONE_YEAR_FREE", "ONE_YEAR_F~
$ telekomHybridUploadSpeed <dbl> NA, NA, 10, NA, NA, 10, 10, NA, NA, NA, 10, N~
$ newlyConst      <lgl> FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, FALS~
$ balcony         <lgl> FALSE, TRUE, TRUE, TRUE, TRUE, FALSE, TRUE, F~
$ picturecount    <dbl> 6, 8, 8, 9, 19, 9, 5, 5, 7, 11, 9, 4, 3, 12, ~
$ pricetrend      <dbl> 4.62, 3.47, 2.72, 1.53, 2.46, 1.01, 1.89, 3.7~
$ telekomUploadSpeed <dbl> 10.0, 10.0, 2.4, 40.0, NA, 2.4, 2.4, 40.0, 40~
$ totalRent       <dbl> 840.00, NA, 1300.00, NA, 903.00, 380.00, 584.~
$ yearConstructed <dbl> 1965, 1871, 2019, 1964, 1950, NA, 1959, 1970,~
$ scoutId         <dbl> 96107057, 111378734, 113147523, 108890903, 11~
$ noParkSpaces    <dbl> 1, 2, 1, NA, NA, NA, NA, 1, NA, NA, NA, NA, N~
$ firingTypes     <chr> "oil", "gas", NA, "district_heating", "gas", ~
$ hasKitchen      <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FAL~
$ geo_bln         <chr> "Nordrhein-Westfalen", "Rheinland-Pfalz", "Sa~
$ cellar          <lgl> TRUE, FALSE, TRUE, FALSE, FALSE, TRUE, TRUE, ~
$ yearConstructedRange <dbl> 2, 1, 9, 2, 1, NA, 2, 2, 2, 1, 1, 1, 2, 9, 4,~
$ baseRent        <dbl> 595.00, 800.00, 965.00, 343.00, 765.00, 310.0~
$ houseNumber     <chr> "244", NA, "4", "35", "10", "14", "35", NA, "~
$ livingSpace     <dbl> 86.00, 89.00, 83.80, 58.15, 84.97, 62.00, 60.~
$ geo_krs         <chr> "Dortmund", "Rhein-Pfalz-Kreis", "Dresden", "~
$ condition       <chr> "well_kept", "refurbished", "first_time_use",~
$ interiorQual    <chr> "normal", "normal", "sophisticated", NA, NA, ~
$ petsAllowed     <chr> NA, "no", NA, NA, NA, NA, NA, "no", "negotiab~
$ street          <chr> "Sch&uuml;rferstra&szlig;e", "no_information~
$ streetPlain     <chr> "Sch&uuml;rferstra&szlig;e", NA, "Turnerweg", "Gl&uuml;ck-Au~
$ lift            <lgl> FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, FALS~
```

```

$ baseRentRange      <dbl> 4, 5, 6, 2, 5, 2, 3, 4, 1, 1, 2, 5, 6, 6, 1, ~
$ typeOfFlat         <chr> "ground_floor", "ground_floor", "apartment", ~
$ geo_plz            <chr> "44269", "67459", "01097", "09599", "28213", ~
$ noRooms            <dbl> 4.0, 3.0, 3.0, 3.0, 3.0, 2.0, 3.0, 2.0, 2.5, ~
$ thermalChar        <dbl> 181.40, NA, NA, 86.00, 188.90, NA, 63.00, 138~
$ floor              <dbl> 1, NA, 3, 3, 1, 1, NA, 2, 2, 3, 1, NA, 4, 0, ~
$ numberOfFloors     <dbl> 3, NA, 4, NA, NA, 4, NA, 2, 5, NA, NA, NA, 4, ~
$ noRoomsRange       <dbl> 4, 3, 3, 3, 3, 2, 3, 2, 2, 2, 3, 4, 4, 3, 1, ~
$ garden             <lgl> TRUE, FALSE, FALSE, FALSE, FALSE, TRUE, FALSE~
$ livingSpaceRange   <dbl> 4, 4, 4, 2, 4, 3, 2, 2, 2, 1, 3, 4, 6, 4, 1, ~
$ regio2             <chr> "Dortmund", "Rhein_Pfalz_Kreis", "Dresden", "~
$ regio3             <chr> "Schüren", "Böhl_Iggelheim", "Äußere_Neustadt~
$ description        <chr> "Die ebenerdig zu erreichende Erdgeschosswohn~
$ facilities          <chr> "Die Wohnung ist mit Laminat ausgelegt. Das B~
$ heatingCosts       <dbl> NA, NA, NA, 87.23, NA, NA, 44.00, NA, NA, NA, ~
$ energyEfficiencyClass <chr> NA, NA, NA, NA, NA, NA, "B", "E", NA, NA, NA, ~
$ lastRefurbish      <dbl> NA, 2019, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
$ electricityBasePrice <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ electricityKwhPrice <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ date               <chr> "May19", "May19", "Oct19", "May19", "Feb20", ~

```

Fokussieren wir uns auf die Spalten `totalRent`, `description` und `facilities`, sowie die `scoutId`, die das jeweilige Angebot identifiziert.

```

inseerate_textdaten <-
  inseerate %>%
  select(scoutId, totalRent, description, facilities)

inseerate_textdaten

```

# A tibble: 1,000 x 4

	scoutId	totalRent	description	facilities
	<dbl>	<dbl>	<chr>	<chr>
1	96107057	840	"Die ebenerdig zu erreichende Erdgeschosswohn~	"Die Wohn~
2	111378734	NA	"Alles neu macht der Mai - so kann es auch fü~	<NA>
3	113147523	1300	"Der Neubau entsteht im Herzen der Dresdner N~	"* 9 m² B~
4	108890903	NA	"Abseits von Lärm und Abgasen in Ihre neue Wo~	<NA>
5	114751222	903	"Es handelt sich hier um ein saniertes Mehrfa~	"Diese Wo~
6	114391930	380	"Am Bahnhof 14 in Freiberg\nHeizkosten und Wa~	<NA>
7	115270775	584.	"+ Komfortabler Bodenbelag: Die Wohnung ist z~	"Rollläde~
8	106416361	690	"Diese ansprechende, lichtdurchflutete DG-Woh~	"Parkett,~
9	91383597	NA	"Sie sind auf der Suche nach einer gepflegten~	"In Ihrem~

```
10 112923517      307 "Gemütliche 2-Raum Wohnung in Chemnitz. komp~ <NA>
# i 990 more rows
```

Wir wollen nur Angebote betrachten, die einen Preis sowie eine generelle Beschreibung haben.

```
relevante_inserate <-
  inserate_textdaten %>%
  drop_na(totalRent, description)

relevante_inserate
```

```
# A tibble: 855 x 4
  scoutId totalRent description facilities
  <dbl>     <dbl> <chr>          <chr>
1  96107057      840 "Die ebenerdig zu erreichende Erdgeschosswohn~ "Die Wohn~
2 113147523     1300 "Der Neubau entsteht im Herzen der Dresdner N~ "* 9 m² B~
3 114751222      903 "Es handelt sich hier um ein saniertes Mehrfa~ "Diese Wo~
4 114391930      380 "Am Bahnhof 14 in Freiberg\nHeizkosten und Wa~ <NA>
5 115270775     584. "+ Komfortabler Bodenbelag: Die Wohnung ist z~ "Rollläde~
6 106416361      690 "Diese ansprechende, lichtdurchflutete DG-Woh~ "Parkett,~
7 112923517      307 "Gemütliche 2-Raum Wohnung in Chemnitz. komp~ <NA>
8 109842225      555 "Gern möchten wir Ihnen diese 3-Zimmer-Wohnun~ "- Wohnzi~
9 111251778      920 "Altes Sandsteinhaus unter Denkmalschutz im J~ "Die Char~
10 101730329     1150 "Die angebotene Wohnfläche befindet sich im d~ "Die Wohn~
# i 845 more rows
```

Wie wir sehen können, gibt es in den Beschreibungstexten die Zeichenfolge `\n`. Diese beschreibt einen Zeilenumbruch (das, was erzeugt wird, wenn man die Entertaste drückt). Wir möchten diese durch ein Leerzeichen ersetzen. Das geht folgendermaßen:

```
relevante_inserate <-
  relevante_inserate %>%
  mutate(description = str_replace_all(description, "\n", " "),
         facilities = str_replace_all(facilities, "\n", " "))
```

Nach diesem Entfernen können wir zum Beispiel die erste Beschreibung ganz normal lesen:

```
relevante_inserate %>%
  select(description) %>%
  head(1) %>%
```

```
pull()
```

```
[1] "Die ebenerdig zu erreichende Erdgeschosswohnung befindet sich in einem gepflegten 8-Fam.
```

## Daten analysieren

Wir können nun mit den Daten arbeiten. Schauen wir uns zum Beispiel einmal an, welche Wörter am häufigsten in den Inseratsbeschreibungen vorkommen.

Hierfür brauchen wir die Library `tidytext`.

```
install.packages("tidytext")

library(tidytext)

tokens <-
  relevante_inserate %>%
  select(description) %>%
  unnest_tokens(input = description,
                output = word,
                to_lower = TRUE)
```

## Häufigkeitsliste

Der obenstehende Code erzeugt einen Tibble mit allen Wörtern (*tokens*), die in den Texten vorkommen. Wir können nun ausgeben lassen, wie oft jedes Wort vorkommt und die Wörter nach Häufigkeit sortieren:

```
tokens %>%
  count(word, sort = TRUE)
```

```
# A tibble: 8,041 x 2
```

	word	n
	<chr>	<int>
1	und	1931
2	die	1522
3	mit	1326
4	in	1234
5	der	1075

```

6 wohnung      967
7 ein          821
8 im           791
9 sich         756
10 das         706
# i 8,031 more rows

```

Wir sehen, dass Wörter wie “und”, “die”, “mit”, ... besonders häufig vorkommen, also Wörter, die keinen Aufschluss über den Inhalt der Beschreibung geben. Wir können diese Wörter, die sogenannten *Stoppwörter*, entfernen. Hierfür brauchen wir die Library `stopwords`.

```

install.packages("stopwords")

stoppwoerter <-
  get_stopwords("de") %>%
  pull(word)

tokens_relevant <-
  tokens %>%
  filter(!word %in% stoppwoerter)

haeufigkeitsliste <-
  tokens_relevant %>%
  count(word, sort = TRUE)

haeufigkeitsliste

```

```

# A tibble: 7,875 x 2
  word      n
  <chr>    <int>
1 wohnung  967
2 befindet 311
3 zimmer   293
4 2        289
5 küche    256
6 balkon   218
7 sowie    214
8 3        200
9 wurde    200
10 verfügt 184
# i 7,865 more rows

```

Wir sehen, dass das häufigste (relevante) Wort in unseren Inseraten “Wohnung” ist, was nicht überraschend ist (wir betrachten Wohnungsinserate).

## Wordwolke

Mithilfe der Häufigkeitsliste können wir eine Wordwolke erstellen. Dafür brauchen wir die Library `wordcloud`:

```
install.packages("wordcloud")
```

```
library(wordcloud)
```

Lade nötiges Paket: `RColorBrewer`

```
wordcloud(  
  words = haeufigkeitsliste$word,  
  freq = haeufigkeitsliste$n  
)
```

Wir sehen: das sind zu viele Wörter für eine Wortwolke. Nehmen wir die 100 häufigsten Wörter:

```
wordcloud(  
  words = haeufigkeitsliste$word[1:100],  
  freq = haeufigkeitsliste$n[1:100]  
)
```

Warning in `wordcloud(words = haeufigkeitsliste$word[1:100], freq = haeufigkeitsliste$n[1:100])`: `wohnung` could not be fit on page. It will not be plotted.





## Sentimentanalyse

Nun haben wir uns angesehen, welche Wörter in den Beschreibungstexten häufig vorkommen. Nun wollen wir die Texte aber weiter auf ihre Stimmung analysieren. Im Deutschen ist das nicht ganz so komfortabel möglich wie im Englischen, wo zahlreiche Libraries zur Verfügung stehen.

Was benötigen wir, um einen Text auf seine Stimmung hin zu analysieren?

1. Den Text, den wir analysieren wollen
2. Eine Liste, die jedem Wort eine Stimmung zuordnet (positiv, negativ, neutral, ...)

Wir beginnen mit dem zweiten Schritt und laden eine Wortliste herunter, die deutsche Wörter enthält. Hierzu hat der Nutzer *georgeblck* auf GitHub ein Skript geschrieben, das unter [diesem Link](#) eingesehen und in veränderter Form unter [diesem Link](#) heruntergeladen werden kann.

Wir können das Skript komplett ausführen, indem wir auf **Source** gehen (rechts oben in der Leiste über dem Code). Der Dataframe `sentiDat` ist dann unsere Wortliste. Falls etwas nicht funktioniert, kann sie auch unter [diesem Link](#) heruntergeladen werden und dann mit diesem Code in den Arbeitsbereich geladen werden:

```
# "data/" etc. hängt natürlich vom Speicherort ab
sentiDat <- read_csv("data/sentiment_liste.csv")
```

```

Rows: 34603 Columns: 4
-- Column specification -----
Delimiter: ","
chr (3): type, POS, Wort
dbl (1): Wert

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.

```

Schauen wir uns diese Liste oder auch *Lexikon* einmal an.

```
sentiDat
```

```

# A tibble: 34,603 x 4
  type      Wert POS  Wort
  <chr>    <dbl> <chr> <chr>
1 neg    -0.058  NN   abbaus
2 neg    -0.0048 NN   abbruches
3 neg    -0.0048 NN   abdankungen
4 neg    -0.0048 NN   abdämpfungen
5 neg    -0.0048 NN   abfalles
6 neg    -0.337  NN   abfahren
7 neg    -0.346  NN   abgründe
8 neg    -0.365  NN   abhängigkeiten
9 neg    -0.512  NN   ablehnungen
10 neg   -0.0435 NN   ablenkungen
# i 34,593 more rows

```

Eine Erklärung der Spalten:

- type: ist das Wort positiv (pos) oder negativ (neg)?
- Wert: wie positiv (+) /negativ (-) ist das Wort? (Von -1 bis 1)
- POS: Part Of Speech: Welche Funktion hat das Wort? (NN: Nomen, VVIN: Verben, ADJX: Adjektive, ADV: Adverbien)
- Wort: das gefragte Wort

Wir wollen nun unsere Daten so vorbereiten, dass wir am Ende für jedes Inserat einen Sentiment-Score erhalten. Dazu füllen wir die Inserate in eine Liste (jedes Element = ein Inserat) und teilen die Inseratstexte auf in Token (wir erhalten eine Liste mit jedem Element = Tibble der Tokens).

```
relevante_inserate %>%
  select(description) %>%
  apply(1, \(\x) tibble(description = x)) %>%
  as.list() -> beschreibung_liste

token_liste <-
  beschreibung_liste %>%
  map(\(x) unnest_tokens(x, word, description))
```

Nach diesem Aufteilen wollen wir endlich zur Sentimentanalyse kommen. Hierzu suchen wir für jedes Wort, das in der Token-Liste vorkommt, seinen Wert im Lexikon und vereinen diese beiden Einträge zu einer Zeile im jeweiligen Tibble in der Token-Liste.

Hierzu ein Exkurs zu Joins.

## Joins

Wer bereits SQL kennt, denen wird der Begriff des Joins etwas sagen. Vereinfacht dargestellt ist ein Join nichts anderes als ein Zusammenfügen zweier Tabellen anhand eines gemeinsamen Kriteriums. Wir werden in diesem Kontext drei Joins besprechen:

1. Left Join (analog: Right Join)
2. Inner Join
3. Full Join

Left Join

LEFT		RIGHT		LEFT JOIN	
Wort	gemeinsame Spalte	Wort	gemeinsame Spalte	Wort	gemeinsame Spalte
frei		abfall	-0,5	frei	✓
toll		still	0,3	toll	0,6
wuchern		toll	0,6	wuchern	-0,4
möbel		wuchern	-0,4	möbel	✓

→ Es werden alle Einträge aus der linken Tabelle übernommen und deren Werte aus der rechten Tabelle in die neue Tabelle überführt.  
(Analog für Right Join)

Inner Join

LEFT		RIGHT		INNER JOIN	
Wort	gemeinsame Spalte	Wort	gemeinsame Spalte	Wert	neue Spalte
frei		abfall		-0,5	
toll		still		0,3	
wuchern		toll		0,6	
möbel		wuchern		-0,4	

→ Es werden die Einträge zusammengeführt, die in beiden Tabellen vorkommen.

### Full Join

LEFT		RIGHT		FULL JOIN	
Wort	gemeinsame Spalte	Wort	gemeinsame Spalte	Wert	neue Spalte
frei		abfall		-0,5	
toll		still		0,3	
wuchern		toll		0,6	
möbel		wuchern		-0,4	

→ Es werden alle Einträge aus beiden Tabellen zusammengeführt.

Wir führen nun einen Inner Join für jedes Inserat durch. Dabei ist die linke Tabelle unsere Token-Liste und die rechte Tabelle das Lexikon.

Hierzu benötigen wir die Library `udpipe`, die es uns erlaubt, die Grundformen von Wörtern zu bestimmen (im Lexikon stehen oft nur die Grundformen). Wir filtern die Token nach Adjektiven, Nomen und Verben (wir lassen Wörter wie “der/die/das”, “welcher/welche/welches”, Satzzeichen und Zahlen aus) und führen dann den Join durch. Für jedes Inserat berechnen wir dann das durchschnittliche Sentiment und schließlich den Durchschnitt über alle Inserate.

```
# install.packages("udpipe")
```

```

library(udpipe)

# ud_model <- udpipe_download_model("german")
ud_model <- udpipe_load_model("german-gsd-ud-2.5-191206.udpipe")

token_liste <-
  beschreibung_liste %>%
  map(\(x) pull(x)) %>%
  map(\(x) {
    udpipe_annotate(ud_model, x) %>%
    as.data.frame() %>%
    filter(upos %in% c("ADJ", "NOUN", "VERB")) %>%
    select(lemma)
  })

sentiment_list <- list()

sentiDat %>%
  distinct(Wort, .keep_all = T) -> sentiDat

for (i in seq_along(token_liste)) {
  sentiment_list[[i]] <-
    token_liste[[i]] %>%
    inner_join(sentiDat, by = join_by(lemma == Wort))
}

sentiment_list %>%
  map(\(x) {
    x %>%
      summarise(sentiment = mean(Wort)) %>%
      pull()
  }) %>%
  unlist() -> sentimente

mean(sentimente, na.rm = TRUE)

```

```
[1] 0.1537123
```

Wir sehen: Mit ungefähr 0,154 ist das Durchschnittssentiment deutlich über 0. Die Wohnungsinserate sind also sehr positiv formuliert.