Programmieren mit statistischer Software

Eva Endres, M.Sc.

Institut für Statistik
Ludwig-Maximilians-Universität München

Textverarbeitung

Text computations I

- Textverarbeitung ist wichtig
  - zur Datenaufbereitung
  - zum Arbeiten mit Text-Daten
- Typische Aufgaben:
  Sortierung, Suche nach Zeichenketten, Musterabgleich, Datenverdichtung
- Datenbeispiel: Better Life Index

```
> bli <- read.csv2("better-life-index.csv",
+                  stringsAsFactors = FALSE)

> head(names(bli))

[1] "COUNTRY"                           "Income_Households.income"
[3] "Income_Household.financial.wealth" "Jobs_Employment.rate"
[5] "Jobs_Personal.earnings"            "Jobs_Job.security"

> summary(bli)
> str(bli)
> head(bli)
```

Character vectors I

- Strings können mit doppelten oder einfachen Anführungszeichen eingegeben werde

```
> t <- c("The quick brown fox", "jumps over the lazy dog")  # bevorzugt
> t
[1] "The quick brown fox"     "jumps over the lazy dog"
> t2 <- c('The quick brown fox', 'jumps over the lazy dog')
> t2
[1] "The quick brown fox"     "jumps over the lazy dog"
```

- Anführungszeichen innerhalb der Textsequenz

```
> t3 <- c('The quick brown fox', 'jumps over the "lazy" dog')
> t3
[1] "The quick brown fox"         "jumps over the \"lazy\" dog"
> t4 <- c("The quick brown fox", "jumps over the 'lazy' dog")
> t4
[1] "The quick brown fox"       "jumps over the 'lazy' dog"
```

Character vectors II

```
> # ?Quotes
> # \n   newline
> # \t  tab
```

- Länge des Vektors

```
> length(t)

[1] 2
```

- Anzahl der Zeichen pro Element

```
> nchar(t)

[1] 19 23
```

- Aneinanderhängen von Text

```
> #?paste
> paste("Today is", date())

[1] "Today is Fri May 12 09:21:40 2017"

> paste("Today is", date(), sep = ": ")

[1] "Today is: Fri May 12 09:21:40 2017"
```

Character vectors III

- paste funktioniert auch vektorisiert

  ```
  > paste("A", 1:6, sep = "")
  [1] "A1" "A2" "A3" "A4" "A5" "A6"
  > # aequivalent
  > paste0("A", 1:6)
  [1] "A1" "A2" "A3" "A4" "A5" "A6"
  ```

- Zusammenfügen von Informationen in einem String (Vektor der Länge 1)

  ```
  > paste("A", 1:6, collapse=",", sep="")
  [1] "A1,A2,A3,A4,A5,A6"
  > paste(t, collapse = " ")
  [1] "The quick brown fox jumps over the lazy dog"
  > paste(t, collapse = " ... ")
  [1] "The quick brown fox ... jumps over the lazy dog"
  ```

weitere nützliche Funktionen I

- formatierte Kombination von Text und Variablenwerten

```
> #?sprintf
> sprintf("%s is %f feet tall", "Sven", 7.1)
[1] "Sven is 7.100000 feet tall"
```

- alles in Kleinbuchstaben/Großbuchstaben

```
> #?tolower
> tolower(t)
[1] "the quick brown fox"     "jumps over the lazy dog"
> #?toupper
> toupper(t)
[1] "THE QUICK BROWN FOX"     "JUMPS OVER THE LAZY DOG"
```

- Text auf eine bestimmte Länge kürzen

```
> #?strtrim
> strtrim(t, c(5,10))
[1] "The q"      "jumps over"
```

weitere nützliche Funktionen II

- Umbrechen von `character`-Zeichenketten

```
> #?strwrap
> strwrap(t, width=5)
[1] "The"   "quick" "brown" "fox"   "jumps" "over"  "the"   "lazy"  "dog"
> strwrap(t, width=10)
[1] "The quick" "brown fox" "jumps"     "over the"  "lazy dog"
> strwrap(t, width=2)
[1] "The"   "quick" "brown" "fox"   "jumps" "over"  "the"   "lazy"  "dog"
```

weitere nützliche Funktionen III

- Abkürzen von Strings

```
> #?abbreviate
> abbreviate(names.arg=t, minlength = 4)
    The quick brown fox jumps over the lazy dog
                 "Tqbf"                 "jotld"
> abbreviate(names.arg=t, minlength = 10)
    The quick brown fox jumps over the lazy dog
           "Thqckbrwnf"            "jmpsovrtld"
> abbreviate(names.arg=t, minlength = 2)
    The quick brown fox jumps over the lazy dog
                 "Tqbf"                 "jotld"
```

String matching I

- exakte Übereinstimmung von Text
  - gibt den Index zurück

    ```
    > #?match
    > match(c("a","y"), letters)
    [1]  1 25
    > match(c("a", 1), letters)
    [1]  1 NA
    ```

  - gibt logischen Vektor zurück

    ```
    > c("a",1) %in% letters
    [1]  TRUE FALSE
    ```

String matching II

- Partial String Matching

```
> #?pmatch
> pmatch("med", c("mean", "mode"))

[1] NA

> pmatch("med", c("mean", "median", "mode"))

[1] 2

> pmatch("med", c("mean", "median", "mode", "median2"))

[1] NA

> #?charmatch
> charmatch("med", c("mean", "mode"))

[1] NA

> charmatch("med", c("mean", "median", "mode"))

[1] 2

> charmatch("med", c("mean", "median", "mode", "median2"))

[1] 0
```

String matching III

- Spalten eines Datensatzes identifizieren

```
> #names(bli)
>
> match(c("Housing_Rooms.per.person",
+         "Environment_Air.pollution"),
+       names(bli))
```

```
[1]  8 22
```

- Prüfen, ob bestimmte Variablen in einem Datensatz enthalten sind

```
> match(c("Housing_Rooms.per.person", "Housing_Rooms.per.person_xxx",
+         "Environment_Air.pollution"),
+       names(bli))
```

```
[1]  8 NA 22
```

```
> ( c("Housing_Rooms.per.person", "Housing_Rooms.per.person_xxx",
+   "Environment_Air.pollution") %in%
+ names(bli) )
```

```
[1]  TRUE FALSE  TRUE
```

Substrings I

- Extrahieren oder Ersetzen von Substrings in einem `character`-Vektor basierend auf Positionen

```
> #?substr
> # substr(x, start, stop)
```

- Anfangs- und Endposition müssen übergeben werden

```
> substr(t, 5, 10)
[1] "quick " "s over"
> substr(t, 4, 11)
[1] " quick b" "ps over "
> substr(t[2], 21, 23)
[1] "dog"
> substr(t[2], 21, 23) <- "cat"
> t
[1] "The quick brown fox"     "jumps over the lazy cat"
```

Substrings II

- kein Recycling

```
> substr(t[2], 21, 23) <- "zo"
> t
[1] "The quick brown fox"     "jumps over the lazy zot"
> substr(t[2], 21, 23) <- "zoxx"
> t
[1] "The quick brown fox"     "jumps over the lazy zox"
> # substring(text, first, last = 1000000L)
```

- nur die Anfangsposition muss übergeben werden

```
> substring(t[2], 21)
[1] "zox"
> substring(t[2], 21) <- "bee"
> t
[1] "The quick brown fox"     "jumps over the lazy bee"
> substring(t[2], 21) <- "zoxx"
> t
[1] "The quick brown fox"     "jumps over the lazy zox"
```

Substrings III

- Anwendung auf eine Spalte in einem Datensatz

```
> substr(bli$Income_Households.income,
+        start = 1,
+        stop = nchar(bli$Income_Households.income) - 4)
 [1] "26927" "27541" "26734" ""      "27138" "8618"  "16614" "23213"
 [9] "13149" "24958" "27789" "27692" "22134" "13696" ""      "24156"
[17] ""      "23917" "23458" "16570" "35321" "11106" "25740" "18601"
[25] "30465" "14508" "18689" "13911" "15840" "19334" "23541" "26633"
[33] "27756" ""      "26552" "37708"
```

```
> Income_Households.income_num <-
+   as.numeric(
+     substr(bli$Income_Households.income,
+            start = 1,
+            stop = nchar(bli$Income_Households.income) - 4)
+     )
```

Pattern matching I

- Regular expressions

  ```
  > #?regex
  ```

- Treffer - Index bzw. ja-nein: grep und grepl

  ```
  > t2 <- c("Programmieren", "mit", "statistischer", "Software", "SS2017")
  > # ?grep
  > # grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
  > #      fixed = FALSE, useBytes = FALSE, invert = FALSE)
  ```

  - grep gibt Index mit Treffern aus

    ```
    > grep("a", t2)
    [1] 1 3 4
    > grep("[[:alpha:]]", t2)
    [1] 1 2 3 4 5
    > grep("[[:digit:]]", t2)
    [1] 5
    ```

Pattern matching II

- grepl gibt Vektor mit TRUE-FALSE zurück

  ```
  > grepl("a", t2)
  [1]  TRUE FALSE  TRUE  TRUE FALSE
  > grepl("[[:alpha:]]", t2)
  [1] TRUE TRUE TRUE TRUE TRUE
  > grepl("[[:digit:]]", t2)
  [1] FALSE FALSE FALSE FALSE  TRUE
  ```

```
> t3 <- c("2012-07-10", "2012-01-20", "May 5, 2012")
>
> grep("\\d{4}-\\d{2}-\\d{2}", t3)
[1] 1 2
> grepl("\\d{4}-\\d{2}-\\d{2}", t3)
[1]  TRUE  TRUE FALSE
```

Pattern matching III

- Index des Treffers innerhalb jedes Elements

```
> #?regexpr
> # regexpr(pattern, text, ignore.case = FALSE, perl = FALSE,
> #         fixed = FALSE, useBytes = FALSE)

> regexpr("a", t2)

[1]  6 -1  3  6 -1
attr(,"match.length")
[1]  1 -1  1  1 -1
attr(,"useBytes")
[1] TRUE

> regexpr("[[:alpha:]]", t2)

[1] 1 1 1 1 1
attr(,"match.length")
[1] 1 1 1 1 1
attr(,"useBytes")
[1] TRUE
```

Pattern matching IV

```
> regexpr("[[:digit:]]", t2)
[1] -1 -1 -1 -1  3
attr(,"match.length")
[1] -1 -1 -1 -1  1
attr(,"useBytes")
[1] TRUE
> # gregexpr("a", t2)
> # gregexpr("[[:alpha:]]", t2)
> # gregexpr("[[:digit:]]", t2)
```

Zugriff auf Teile des Musters: `sub` und `gsub` I

```
> # sub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,
> #     fixed = FALSE, useBytes = FALSE)
> sub("(\\d{4})-\\d{2}-\\d{2}", "\\1", t3)
[1] "2012"        "2012"        "May 5, 2012"
> sub("(\\d{4})-(\\d{2})-(\\d{2})", "\\3.\\2.\\1", t3)
[1] "10.07.2012"  "20.01.2012"  "May 5, 2012"
> gsub("(\\d{4})-\\d{2}-\\d{2}", "\\1", t3)
[1] "2012"        "2012"        "May 5, 2012"
> gsub("(\\d{4})-(\\d{2})-(\\d{2})", "\\3.\\2.\\1", t3)
[1] "10.07.2012"  "20.01.2012"  "May 5, 2012"
```

Split the Elements of a Character Vector I

- Split the Elements of a Character Vector

```
> #?strsplit
> # strsplit(x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)

> x <- c(as = "asfef", qu = "qwerty", "yuiop[", "b", "stuff.blah.yech")
> strsplit(x, "e")

$as
[1] "asf" "f"

$qu
[1] "qw"  "rty"

[[3]]
[1] "yuiop["

[[4]]
[1] "b"

[[5]]
[1] "stuff.blah.y" "ch"
```

Split the Elements of a Character Vector II

```
> t4 <- c("89. Derdiyok fuer Schuerrle",
+          "69. Kohr fuer L. Bender")
> #?regexec
> m <- regexec("(\\d\\d)\\. (.+) fuer (.+)", t4)
> #?regmatches          # Extract or Replace Matched Substrings
> regmatches(t4, m)
[[1]]
[1] "89. Derdiyok fuer Schuerrle" "89"
[3] "Derdiyok"                    "Schuerrle"

[[2]]
[1] "69. Kohr fuer L. Bender" "69"
[3] "Kohr"                    "L. Bender"
```

Anwendung auf einen Datensatz I

- Index der Variablennamen, die Muster enthalten

```
> grep("Income_", names(bli))
```

```
[1] 2 3
```

- Extraktion der Zahlen (digits)

```
> sub("(\\d+) .+", "\\1", bli$Income_Households.income)
```

```
 [1] "26927" "27541" "26734" ""      "27138" "8618"  "16614" "23213"
 [9] "13149" "24958" "27789" "27692" "22134" "13696" ""      "24156"
[17] ""      "23917" "23458" "16570" "35321" "11106" "25740" "18601"
[25] "30465" "14508" "18689" "13911" "15840" "19334" "23541" "26633"
[33] "27756" ""      "26552" "37708"
```

Data analysis I

```
> satisfaction <- data.frame(
+   letter = substr(bli$COUNTRY, 1, 1),
+   value  = as.numeric(
+     sub("(.+) rate", "\\1", bli$Life.Satisfaction_Life.Satisfaction)))
>
> # satisfaction
>
> # barplot(sapply(split(satisfaction$value, satisfaction$letter), mean))

> # Objekt mit Nummer der Iteration im Namen
> for(i in 1:3){
+   name <- paste0("res",i)
+   assign(name,i)
+ }
> res1

[1] 1

> res2

[1] 2

> res3

[1] 3
```