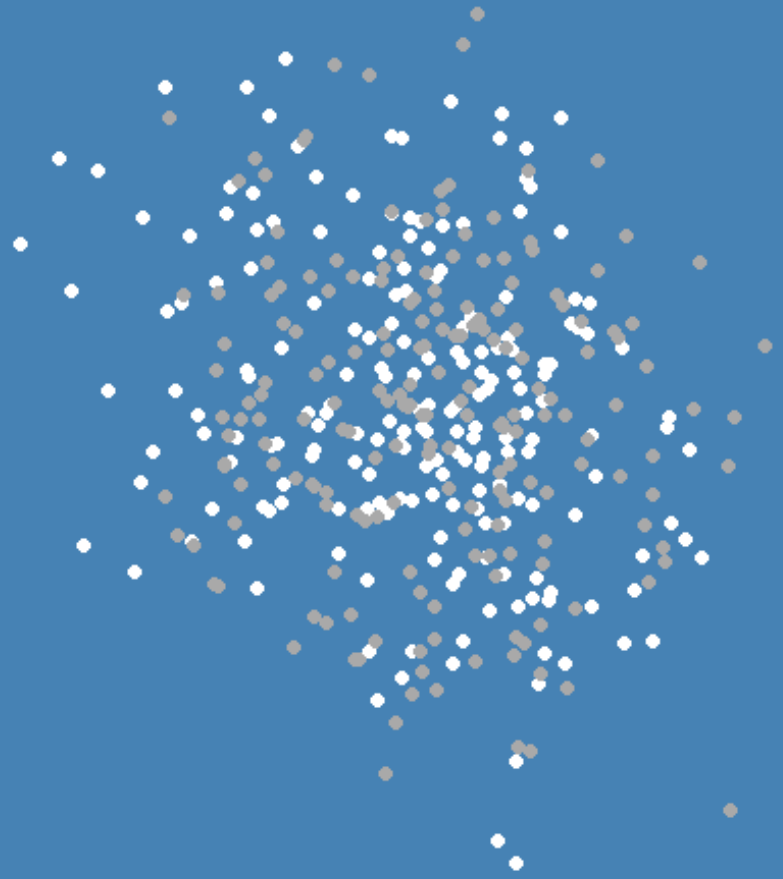
Introduction to R

1.5 Accessing, Subsetting and Naming Objects

Lion Behrens, M.Sc.

UNIVERSITY OF MANNHEIM

University of Mannheim
Chair of Social Data Science and Methodology
Chair of Quantitative Methods in the Social Sciences



Accessing and Subsetting Objects

Accessing and Subsetting Objects

- Often, we only want to access a subset of the information that is stored in an object.
- This holds equally if we are working with vectors, matrices, data frames or lists.
- Accessing and subsetting objects works very similar across different object types.

```
character_vector <- c("Austria", "England", "Brazil", "Germany")
```

```
character_vector[1] # accessing the first element
```

```
## [1] "Austria"
```

Accessing and Subsetting Vectors

- Often, we only want to access only a subset of the information that is stored in an object.
- This holds equally if we are working with vectors, matrices, data frames or lists.
- Accessing and subsetting objects works very similar across different object types.

```
character_vector <- c("Austria", "England", "Brazil", "Germany")
```

```
character_vector[length(character_vector)] # accessing the last element
```

```
## [1] "Germany"
```

Accessing and Subsetting Matrices

- Since matrices have two dimensions, we need to specify the row and column of the element that we want to retreive.

The general logic goes

```
matrix_object[specific_row, specific_column]
```

Note: These operations equally hold for data frames.

Accessing and Subsetting Matrices

- Since matrices have two dimensions, we need to specify the row and column of the element that we want to retreive.

```
matrix_example <- matrix(1:20, nrow = 4, ncol = 5)
print(matrix_example)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

```
matrix_example[1,1] # 1st row, 1st column
matrix_example[2,] # 2nd row
matix_example[,4] # 4th column
matrix_example[1:2, 3:4] # rows 1 and 2, columns 3 and 4
```

Accessing and Subsetting Data Frames

- When working with data frames, you can select individual elements, rows and columns in the same way as with matrices.
- Additionally, you can take advantage of the the fact that columns (variables) actually have names.

```
df_example <-
  data.frame(
    country = c("Austria", "England", "Brazil", "Germany"),
    capital = c("Vienna", "London", "Brasília", "Berlin"),
    elo = c(1761, 1938, 2166, 1988)
  )
```

- You can access individual variables using the $ sign:

```
df_example$country
```

```
## [1] "Austria" "England" "Brazil"  "Germany"
```

Accessing and Subsetting Data Frames

- When working with data frames, you can select individual elements, rows and columns in the same way as with matrices.
- Additionally, you can take advantage of the the fact that columns (variables) actually have names.

```
df_example <-
  data.frame(
    country = c("Austria", "England", "Brazil", "Germany"),
    capital = c("Vienna", "London", "Brasília", "Berlin"),
    elo = c(1761, 1938, 2166, 1988)
  )
```

- An alternative is indexing by variable name.

```
df_example["country"]
```

```
##   country
## 1 Austria
## 2 England
## 3  Brazil
## 4 Germany
```

Accessing and Subsetting Data Frames

- When working with data frames, you can select individual elements, rows and columns in the same way as with matrices.
- Additionally, you can take advantage of the the fact that columns (variables) actually have names.

```
df_example <-
  data.frame(
    country = c("Austria", "England", "Brazil", "Germany"),
    capital = c("Vienna", "London", "Brasília", "Berlin"),
    elo = c(1761, 1938, 2166, 1988)
  )
```

- Or using the order of the variables:

```
df_example[1]
```

```
##   country
## 1 Austria
## 2 England
## 3  Brazil
## 4 Germany
```

Accessing and Subsetting Lists

```
list_example
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "Austria" "England" "Brazil"  "Germany"
##
## [[3]]
## [1]  TRUE FALSE FALSE  TRUE
##
## [[4]]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
##
## [[5]]
##   country  capital  elo
## 1 Austria   Vienna 1761
## 2 England   London 1938
## 3  Brazil Brasília 2166
## 4 Germany   Berlin 1988
```

Accessing and Subsetting Lists

Next to indexing with [ ] there exists a different subsetting operator in R using [[ ]].

- [ ] always returns object of the same class
- [[ ]] can return objects of different classes

This distinction comes in very handy when working with lists.

```
list_example[2] # returns a sublist of list_example, which is a list itself
```

```
## [[1]]
## [1] "Austria" "England" "Brazil"  "Germany"
```

```
x <- list_example[2]
class(x) # check whether it's a list
```

```
## [1] "list"
```

Accessing and Subsetting Lists

Next to indexing with [ ] there exists a different subsetting operator in R using [[ ]].

- [ ] always returns object of the same class
- [[ ]] can return objects of different classes

This distinction comes in very handy when working with lists.

```
list_example[[2]] # returns the object that is stored in the second sublist
```

```
## [1] "Austria" "England" "Brazil"  "Germany"
```

```
x <- list_example[[2]]
class(x) # check whether it's a vector
```

```
## [1] "character"
```

```
is.vector(x)
```

```
## [1] TRUE
```

Combining [[ ]] and [ ]

We can even combine both subsetting operators when working with lists. Type list_example[[2]][1].

- [[2]] refers to the vector stored in the second sublist in list_example
- [1] refers to the first element of this vector

Question: Which object will be retained by the above command?

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "Austria" "England" "Brazil"  "Germany"
```

```
list_example[[2]][1]
```

```
## [1] "Austria"
```

Assigning Names to Objects

Naming Objects

We can name the individual elements of different elements using the names() function.

First, let's check whether the vectors that we generated have any names assigned to them.

```
numeric_vector <- c(1, 2, 3, 4, 5)
names(numeric_vector)
```

```
## NULL
```

Now, let's assign names.

```
names(numeric_vector) <- c("Name A", "Name B", "Name C", "Name D", "Name E")
numeric_vector
```

```
## Name A Name B Name C Name D Name E
##      1      2      3      4      5
```

Naming Objects

We can name the individual elements of different elements using the names() function.

First, let's check whether the vectors that we generated have any names assigned to them.

The same holds for lists.

```
names(list_example) <- c("numeric_vector", "character_vector", "logical_vector", "matrix", "df")
names(list_example)
```

```
## [1] "numeric_vector"   "character_vector" "logical_vector"
## [4] "matrix"           "df"
```

Naming Objects

We can name the individual elements of different elements using the names() function.

First, let's check whether the vectors that we generated have any names assigned to them.

As well as data frames.

```
names(df_example) <- c("Var 1", "Var 2", "Var 3")
```

Note: For matrices we need to use the commands colnames() and rownames().

References

Parts of this course are inspired by the following resources:

- Wickham, Hadley and Garrett Grolemund, 2017. *R for Data Science - Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly.
- Bahnsen, Oke and Guido Ropers, 2022. *Introduction to R for Quantitative Social Science*. Course held as part of the GESIS Workshop Series.
- Breuer, Johannes and Stefan Jünger, 2021. *Introduction to R for Data Analysis*. Course held as part of the GESIS Summer School in Survey Methodology.
- Teaching material developed by Verena Kunz, David Weyrauch, Oliver Rittmann and Viktoriia Semenova.