

Einführung in R für SozialwissenschaftlerInnen

Angelina Mooseder

Technical University of Munich

Bavarian School of Public Policy

mooseder@in.tum.de





Gliederung des Vorlesungsskripts

Woche 0:

Administratives

Woche 1: Grundlagen

- Einführung
- Einrichten der Programmierumgebung
- Organisation in R
- Grundlagen in R
- Datentypen

Woche 2: Datenstruktur "Vektor"

- Vektoren erstellen
- Rechnen mit Vektoren
- <u>Informationen über Vektoren erhalten</u>
- Vektor-Elemente ansprechen
- Factor-Vektoren



Gliederung des Vorlesungsskripts

Woche 3: Weitere Datenstrukturen

- Matrizen
- <u>Listen</u>
- Dataframes

Woche 4: Arbeiten mit Dataframes

- Umgang mit Dataframes I: Datentypen und sortieren
- Umgang mit Dataframes II: Fehlende Werte
- Umgang mit Dataframes III: Dataframes verändern

Woche 5: Informationen aus Dataframes

- <u>Visualisierungen</u>
- Umgang mit Dataframes IV: Gruppen bilden



Gliederung des Vorlesungsskripts

Woche 6: Wichtige Funktionen

- Funktionen schreiben
- Apply-Funktion
- Merge-Funktion

Woche 7: Fortgeschrittene Themen

- <u>Umgang mit Zeichenketten</u>
- Umgang mit Datums- und Zeitangaben
- Schleifen und Bedingungen



WOCHE 0



Administratives





Personal

- Angelina Mooseder
 - Vorlesungs- und Übungsmaterialien
- Margareta Maier
 - Tutorien
 - Support über Slack



Voraussetzungen

- Keine Vorkenntnisse notwendig!
- Anmeldung zu einer der beiden Lehrveranstaltungen in TUM Online
- Anmeldung zur Prüfung in TUM Online
- Zugang zu Slack



1) Vorlesung:

- Videos auf YouTube
- 7 Wochen lang

2) Tutorien:

- Fragestunde und Wiederholung von Stoff
- 2 Termine:
 - Di, 09:30-11, in H.103 (bitte 3-G-Nachweis mitbringen!)
 - Do, 09:30-11, online über Zoom



3) Hausaufgaben:

- Woche 1-2:
 - Freiwillige Übungsblätter
 - Besprechung eine Woche später in den Tutorien
 - Musterlösung wird hochgestellt
- Woche 3-7:
 - Benotete Übungsblätter
 - Abgabe über ein Online-Abgabesystem
 - Für jedes Blatt 2 Wochen Zeit zur Bearbeitung, für das letzte Blatt 3 Wochen
 - Wiederholte Abgabe möglich
 - Musterlösung wird nicht hochgestellt, aber in den Tutorien besprochen



4) Prüfung:

- Keine Prüfung in Form einer Klausur, sondern die Hausaufgaben 3-7 gelten als Prüfungsleistung
- Die Hausaufgaben müssen alleine bearbeitet werden
- Trotzdem zur Prüfung in TUM-Online anmelden!



Woche	Dienstag, 08:00	Dienstag, 09:30- 11:00, Präsenz	Donnerstag, 09:30- 11:00, Online	Donnerstag, 12:00
18.1024.10.	Videos (Woche 1)			
25.1031.10.	Videos (Woche 2)	Tutorium Woche 1	Tutorium Woche 1	Hausaufgabe 1
01.1007.11.	Videos (Woche 3)	Tutorium Woche 2 + Besprechung H1	Tutorium Woche 2 + Besprechung H1	Hausaufgabe 2
08.1014.11.	Videos (Woche 4)	Tutorium Woche 3 + Besprechung H2	Tutorium Woche 3 + Besprechung H2	Hausaufgabe 3 (Abgabe 25.11.)
15.1021.11.	Videos (Woche 5)	Fällt aus (Fachschafts-Versammlung)	Tutorium Woche 4	Hausaufgabe 4 (Abgabe 02.12)
22.1028.11	Videos (Woche 6)	Tutorium Woche 5	Tutorium Woche 5	Hausaufgabe 5 (Abgabe 09.12.)
29.1005.12	Videos (Woche 7)	Tutorium Woche 6 + Besprechung H3	Fällt aus (Dies Academicus)	Hausaufgabe 6 (Abgabe 16.12.)
06.1212.12.		Tutorium Woche 7 + Besprechung H4	Tutorium Woche 7 + Besprechung H4	Hausaufgabe 7 (Abgabe 30.12)
13.1219.12.		Besprechung H5	Besprechung H5	
20.1209.01.			Ferien	
10.0102.01.		Besprechung H6, H7	Besprechung H6, H7	

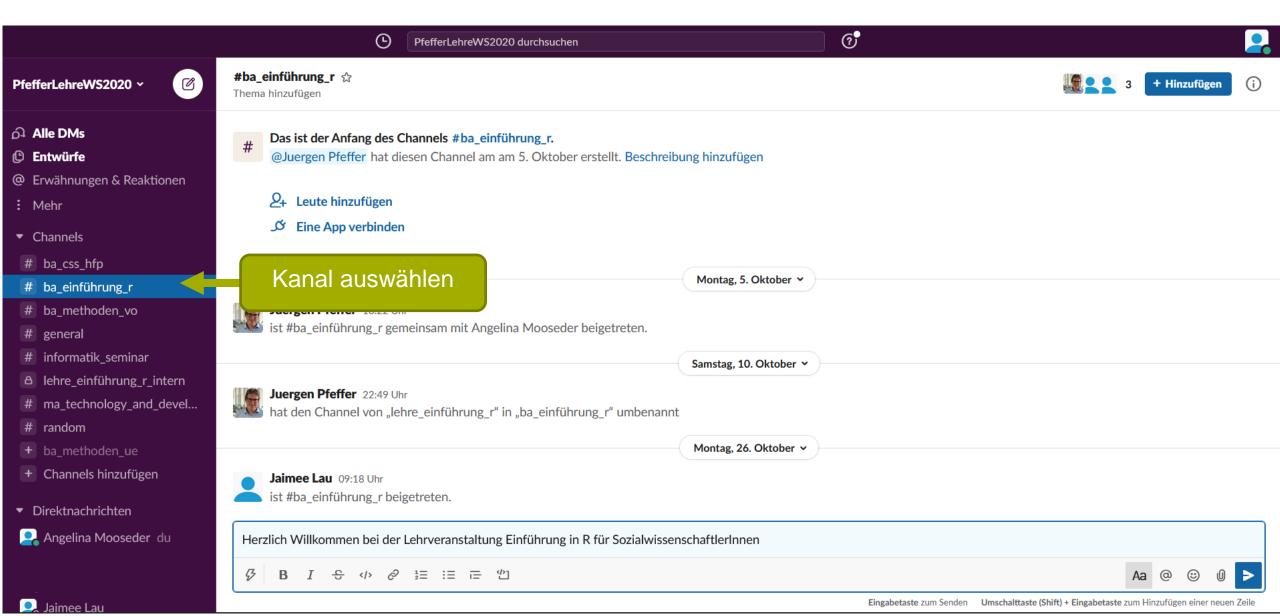


5) Slack:

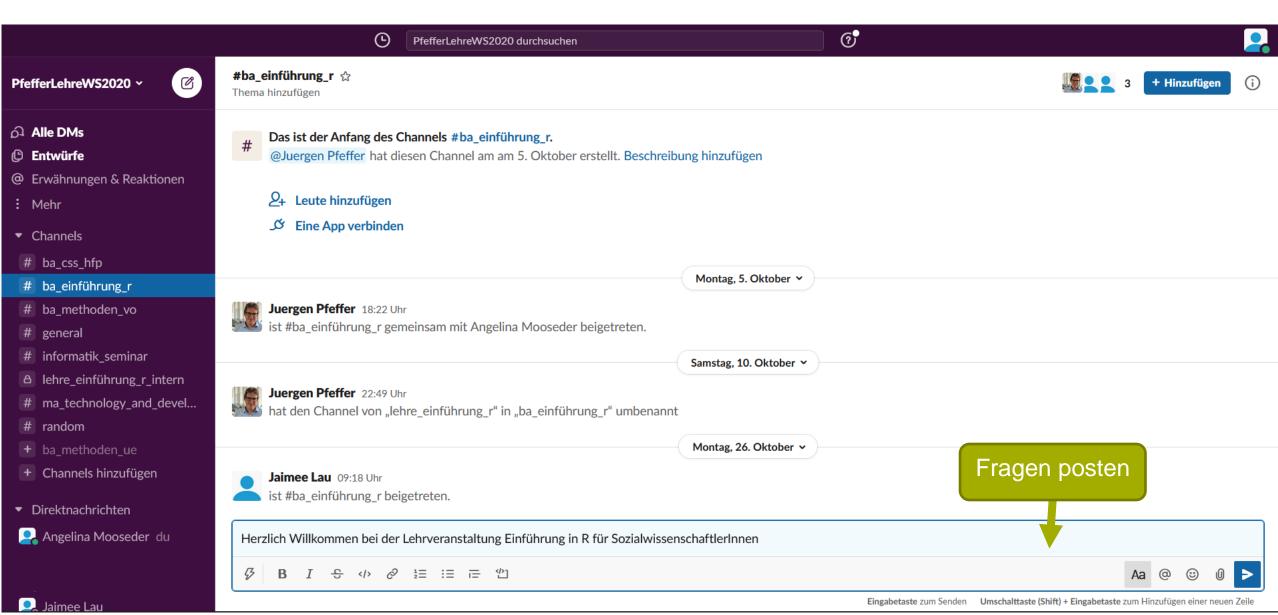
- Fragen stellen
- Fragen von Kommilitonen beantworten
- Neuigkeiten rund um die Vorlesung erfahren

Aber Vorsicht: Keine Fragen, die den anderen Informationen über die Lösung der Hausaufgaben geben könnten, das wäre Unterschleif!!! (Insbesondere keinen Code, kein "Ich habe Lösung X probiert, aber das klappt nicht",...)

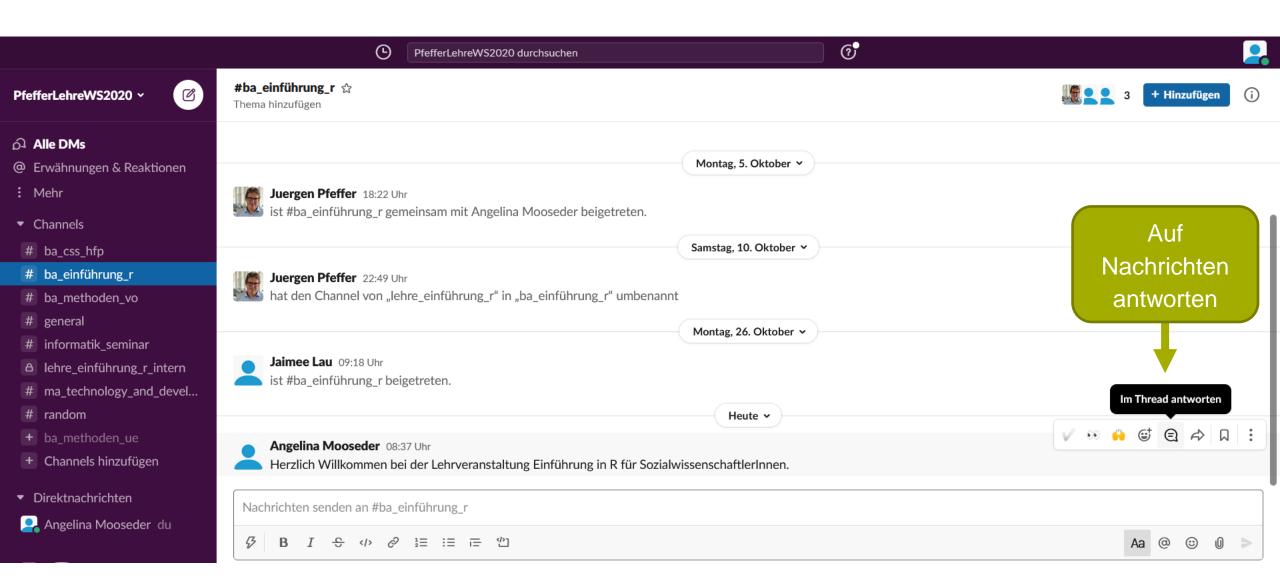




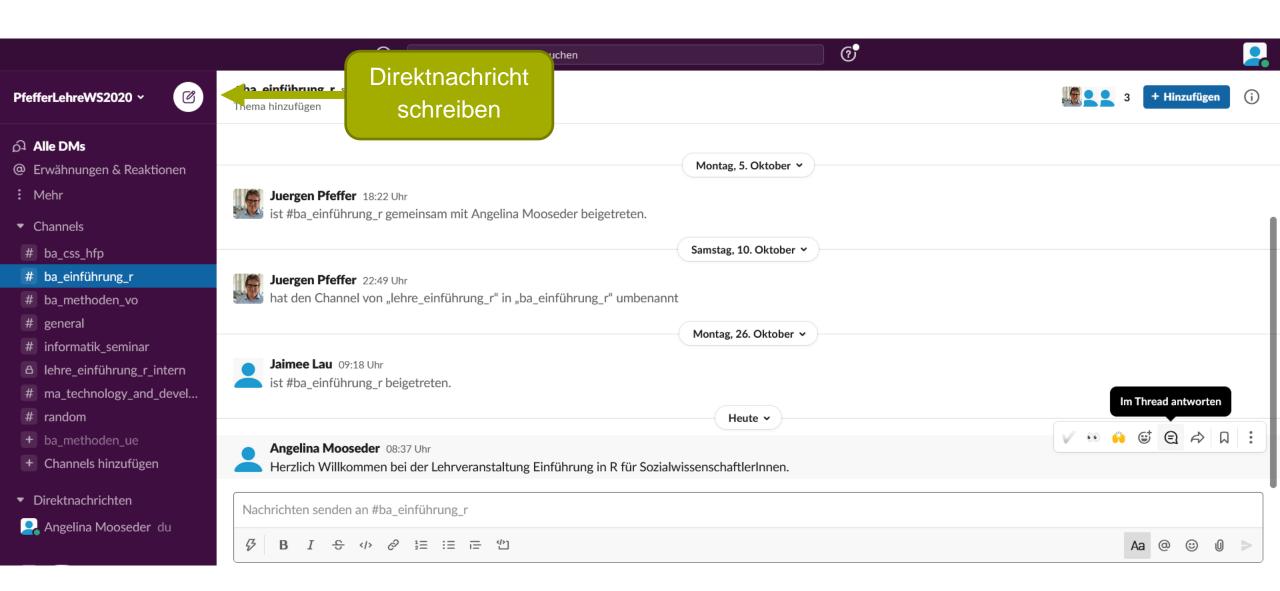










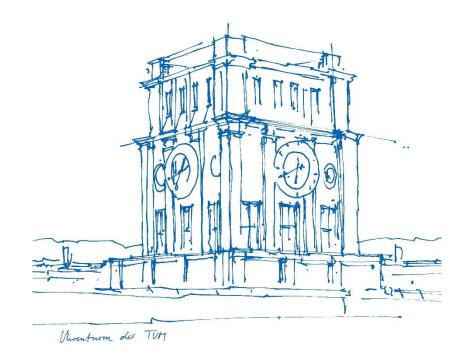




WOCHE 1



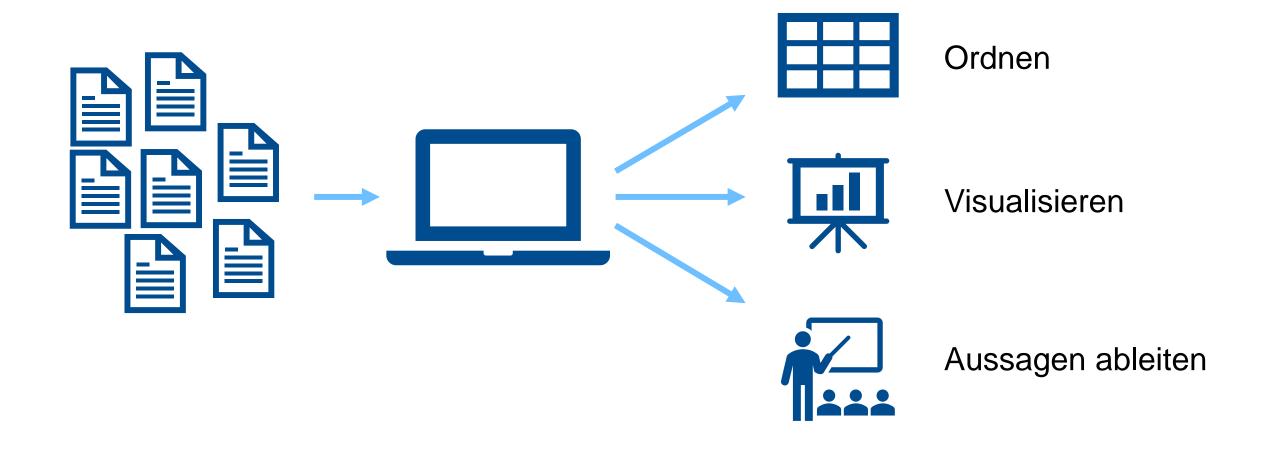
Einführung













Gliederung

1) Woche 1: Grundlagen

- Einrichten der Programmierumgebung
- Sich organisieren in R
- Grundlagen in R
- Datentypen

2) Woche 2: Datenstruktur "Vektor"

- Vektoren erstellen
- Rechnen mit Vektoren
- Informationen über Vektoren erhalten
- Vektor-Elemente ansprechen
- Factor-Vektoren



Gliederung

3) Woche 3: Weitere Datenstrukturen

- Matrizen
- Listen
- Dataframes

4) Woche 4: Arbeiten mit Dataframes

- Umgang mit Dataframes I: Datentypen und sortieren
- Umgang mit Dataframes II: Fehlende Werte
- Umgang mit Dataframes III: Dataframes verändern

5) Woche 5: Informationen aus Dataframes

- Visualisierungen
- Umgang mit Dataframes IV: Gruppen bilden



Gliederung

6) Woche 6: Wichtige Funktionen

- Funktionen schreiben
- Apply-Funktion
- Merge-Funktion

7) Woche 7: Fortgeschrittene Themen

- Arbeiten mit Zeichenketten
- Arbeiten mit Datums- und Zeitangaben
- Schleifen und Bedingungen

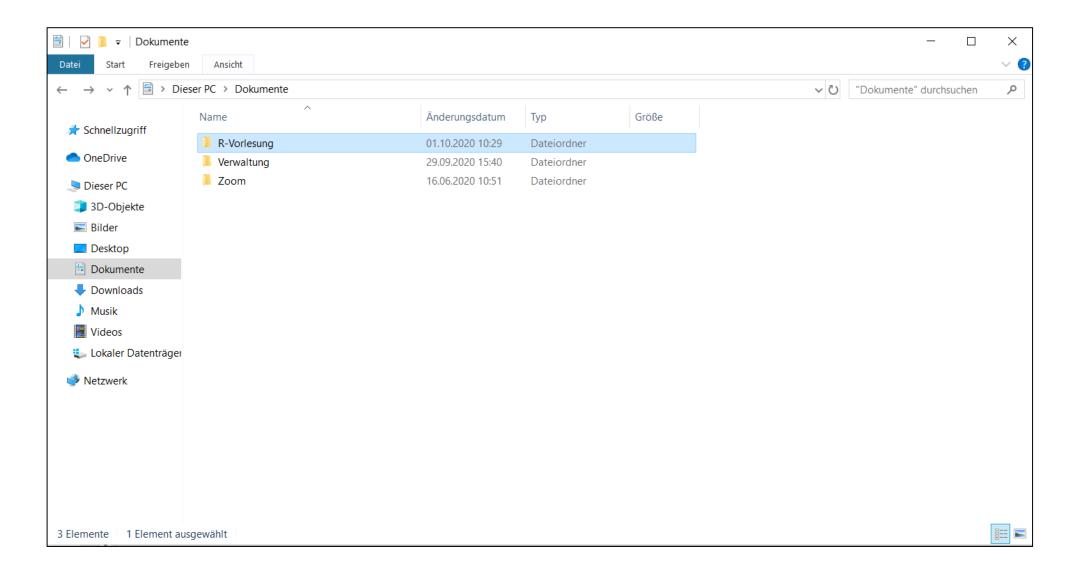


Einrichten der Programmierumgebung





Was brauchen wir?

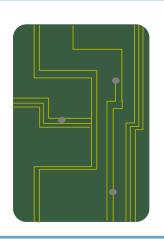




Was brauchen wir?



Programmiersprache





Entwicklungsumgebung







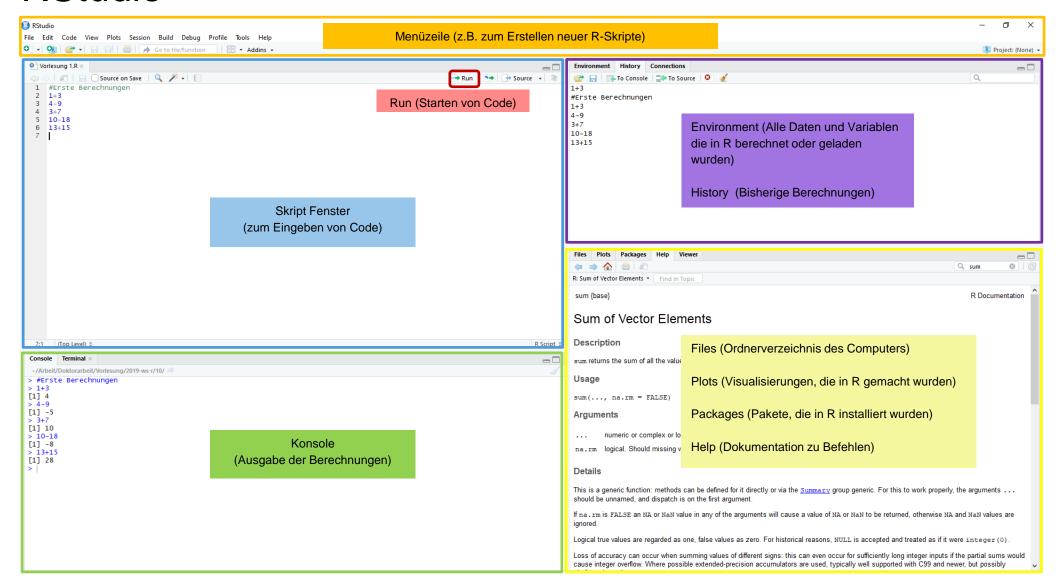
Installation

- 1) R installieren
 - http://cran.r-project.org/bin/
 - Version: 4.0.2

- 2) R Studio installieren
 - https://www.rstudio.com
 - Version: RStudio Desktop Open Source



RStudio





Organisation in R





Organisation in R

1) Daten einlesen

- Working-Directory setzen: setwd("~/R-Vorlesung")
- .txt-Datei einlesen: read.table("Test.txt")
- .csv-Datei einlesen: read.csv("Test.csv") oder read.csv2("Test.csv")

2) Daten speichern

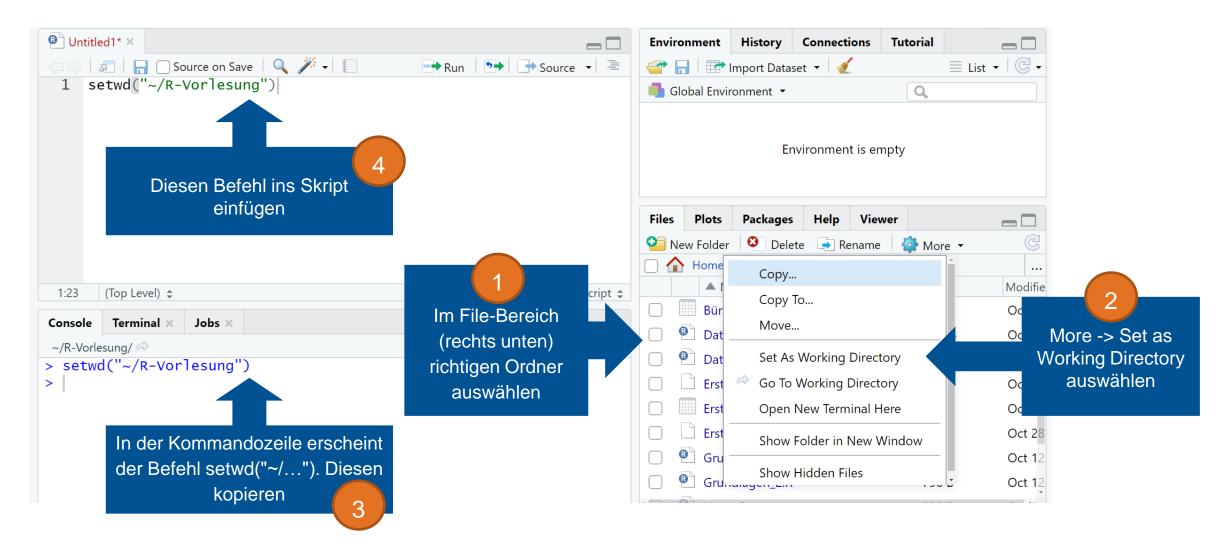
- Working-Directory setzen: setwd("~/R-Vorlesung")
- .txt-Datei speichern: write.table(daten, file="Meine_Daten.txt")
- csv-Datei speichern: write.csv(daten, file="Meine_Daten.csv") oder write.csv2(daten, file="Meine_Daten.csv")

3) Pakete installieren

- Einmal Installieren: install.packages("TeachingDemos")
 Anführungszeichen nicht vergessen!
- Immer zu Beginn der Session einlesen: library(TeachingDemos)



Organisation in R: Working-Directory mit R-Studio setzen





Grundlagen





Grundlagen

1) Kommentar

Kommentar hinter Rautenzeichen: 3+4 #Dies ist eine Addition

2) Funktion

- Argumente durch Komma trennen
- Reihenfolge beachten: round(3.47, 2)
- Argumentnamen angeben: round(x=3.47, digits=2)

3) Variable

- Zuweisung: zahl<-8
- Variable wiederverwendbar: zahl<-zahl+1



Grundlagen: Mathematische Operationen

<u>Operatoren</u>		<u>Funktionen</u>	
	Addition	oum()	Cummo
+	Addition	sum()	Summe
-	Subtraktion	abs()	Betrag
*	Multiplikation	exp()	Exponentialfunktion
^	Potenz	sqrt()	Wurzel
/	Division	round()	Runden
%/%	Ganzzahlige Division		
%%	Modulo-Division		



Datentypen





Datentypen: Rechnen mit Logicals

Und				Oder				Entweder Oder			
&	TRUE	FALSE			TRUE	FALSE		xor()	TRUE	FALSE	
TRUE	TRUE	FALSE		TRUE	TRUE	TRUE		TRUE	FALSE	TRUE	
FALSE	FALSE	FALSE		FALSE	TRUE	FALSE		FALSE	TRUE	FALSE	
Beispiel: a<-24 a<23 & a<30 => FALSE				Beispiel: a<-24 a<23 a<30 => TRUE			3	Beispiel: a<-24 xor(a<23,a<30) => TRUE			



Datentypen

1) Datentypen

Numeric: 1 oder 3.4

• Complex: 2i

NULL: NULL

Logical: TRUE oder FALSE

• Character: "Banane!" oder '1'

2) Datentypen verwenden

- Numeric und Complex f
 ür mathematische Operationen
- Logical f
 ür logische Verkn
 üpfungen

3) Datentypen prüfen

- Is-Funktion: is.numeric(3)
- Mode-Funktion: mode(3)



WOCHE 2



Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de





1) Ein Vektor enthält immer nur einen Datentyp!!!

2) Erstellen

- Kombination: c(2,5,6)
- Folge: 1:5
- Sequenz: seq(from=1, to=10, by=2)
- Wiederholung: rep(c(1,2,3), each=3, times=2, length.out=15)

3) Rechnen

- Mathematische Operatoren
- Mathematische Funktionen
- Reihenfolge verändern: rev(1:5), sort(c(1,4,3,2,5))



4) Informationen erhalten

- Vektorlänge: length(vek)
- Minimum, Maximum: min(vek), max(vek)
- Mean, Median: mean(vek), median(vek)
- Summary: summary(vek)

5) Elemente ansprechen

- Über Position ansprechen: vek[3:7], vek[-4]
- Über Namen ansprechen: vek ["Dienstag"]
- Über Logicals ansprechen: vek[vek<4]
- Names-Funktion verwenden: names(vek)<-c("Montag","Dienstag")
- Vektorelemente verändern: vek[3]<-27

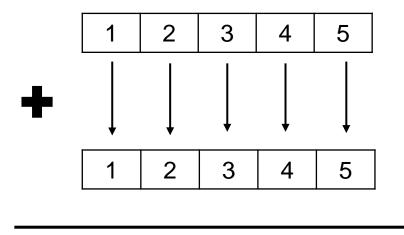


6) Factor-Vektoren

- Erstellen: vek<- factor(vek,levels = c("klein", "mittel", "groß"), ordered = TRUE)
- Kategorien herausfinden: levels(vek)
- Verteilung herausfinden: summary(vek)

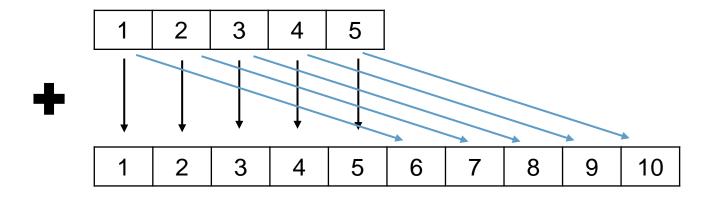


Vektoren: Vektor-Addition*





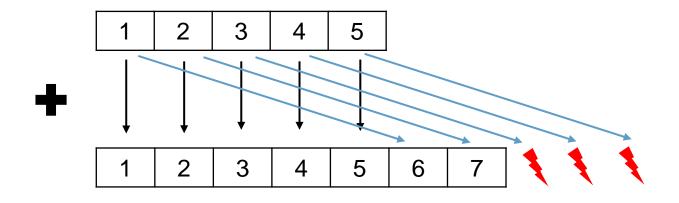
Vektoren: Vektor-Addition*



 2
 4
 6
 8
 10
 7
 9
 11
 13
 15



Vektoren: Vektor-Addition*





^{*}Dieses Prinzip gilt auch für -, *, ^, /, %/% und %%.

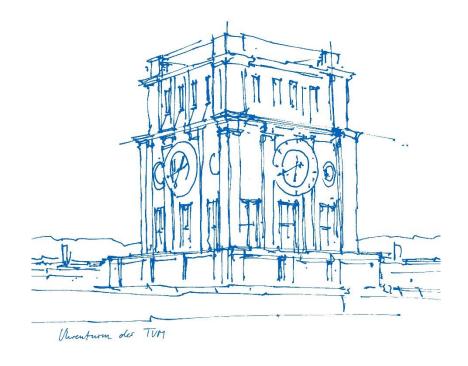


WOCHE 3



Matrizen

Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de





Matrizen

1) Eine Matrix enthält immer nur einen Datentyp!!!

2) Erstellen

- Matrix-Funktion: m<-matrix(vek, nrow=3, ncol=4, byrow=TRUE)
- Vektoren zeilenweise zusammenfügen: m<-rbind(vek1, vek2)
- Vektoren spaltenweise zusammenfügen: m<-cbind(vek1, vek2)

3) Elemente ansprechen

- Über Position ansprechen: m[2,3], m[2,], m[,3]
- Über Namen ansprechen: m ["Peter", "Englisch"]
- Über Logicals ansprechen: m[m>3]
- Names-Funktion verwenden: colnames(m)<-c("A","B"), rownames(m)<-c("A","B")
- Matrizen-Element verändern: m[2,3] <-1

4) Rechnen

• Ähnlich zu Vektoren: m+m, summary(m)



Listen

Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de





Listen

- 1) Listen enthalten verschiedene Datentypen und Strukturen
- 2) Erstellen
 - List-Funktion: lis<-list(Zahlen=1:5, Buchstaben=c("A","B","C"), Nummer=3)
- 3) Elemente ansprechen
 - Über Position ansprechen: lis[[2]][1]
 - Über Namen ansprechen: lis\$Buchstaben
 - Über Logicals ansprechen: lis\$Zahlen[lis\$Zahlen<5]
 - Names-Funktion verwenden: names(lis)<-c("Montag", "Dienstag", "Mittwoch")
 - Listen- Element verändern: lis[[2]][1] <- 1 (Richtigen Datentyp beachten!)

4) Rechnen

Mit Listenelementen rechnen und Funktionen anwenden: sum(y\$Zahlen), summary(y\$Faktoren)



Data-Frames

Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de





DataFrames

 Dataframes sind Listen, die Vektoren gleicher Länge enthalten. Jeder Vektor kann einen anderen Datentyp enthalten.

2) Erstellen

- Data-Frame aus Vektoren: d<-data.frame(v1,v2,v3)
 mit Namen: d<-data.frame(Studium=v1, Semester=v2,Geschlecht=v3, row.names=c("Peter","Anna"))
 mit Factor- statt Char-Vektoren: d<-data.frame(v1,v2,v3,stringsAsFactors=TRUE)
- Data-Frame aus Matrix: d<-as.data.frame(m)

3) Elemente ansprechen

- Names-Funktion verwenden:
 - Alle Namen verändern: rownames(d)<- c("A","B","C"), colnames(d)<- c("A","B","C")
 - Einzelne Namen verändern: colnames(d)[colnames(d) == "alterName"] <- "neuerName,"



DataFrames

Elemente ansprechen II

- Über Position ansprechen:
 - Wie Matrix, also d[Zeile, Spalte]: d[3,1], d[3,], d[,1]
 - Wie Liste, also d[[Spalte]][Zeile]: d[[1]][3], d[[1]]
- Über Namen ansprechen:
 - Wie Matrix, also d[Zeile, Spalte]: d["Anna", "Semester"], d["Anna",], d["Semester"]
 - Wie Liste, also d\$Spalte: d\$Semester
- Über Logicals ansprechen:
 - Listenelement erhalten: d\$Semester[d\$Semester<4]
 - Ganze Zeile erhalten: d[d\$Semester<4,]
 - Mit Subset-Funktion: subset(d,d\$Semester<4, select=Hauptfach)
- Datenelement verändern: d[3,1]<-3

4) Rechnen

Mit Spalten rechnen und Funktionen anwenden: sum(y\$Zahlen), summary(y\$Faktoren)



Übersicht

	Matrix	Liste	DataFrame			
Besteht aus	1 Vektor = gleicher Datentyp	Versch. Datentypen und Strukturen	Versch. Vektoren mit gleicher Länge			
	A B Tom 1 2 Eva 3 4	A TRUE FALSE B 3 C 1 2 3 4	A B Tom TRUE 1 Eva FALSE 2			
Erstellen	matrix(v1)	list(v1, z1, m1)	data.frame(v1, v2)			
Erstellen mit Namen	m<-matrix(v1) rownames(m)<-c("Tom", "Eva") colnames(m)<-c(A,B)	list(A=v1, B=z1, C=m1)	data.frame(A=v1, B=v2, row.names=c("Tom", "Eva"))			
Über Position anspr.	x[2,1]	x[[2]][1]	x[2,1] und x[[1]][2]			
Über Namen ansprechen	x["Tom", "A"] x["Tom",] x["A"] /	/ / x["A"] x\$A x\$A[1]	x["Tom","A"] x["Tom",] x["A"] x\$A x\$A[1]			



WOCHE 4



Arbeiten mit Data-Frames I Datentypen und Sortieren

Angelina Mooseder

Technical University of Munich

Bavarian School of Public Policy

mooseder@in.tum.de





Szenario



Herr Müller Integration



Frau Schmidt Klimaschutz



Herr Paolini Bildungsreformen



Arbeiten mit DataFrames I

1) Spalten sortieren

Erst Spalte 3, dann 1, dann 2: d<-d[,c(3,1,2)]

2) Datentypen verändern

- In Char umwandeln: d\$Spalte <-as.character(d\$Spalte)
- In Numeric umwandeln: d\$Spalte <-as.numeric(d\$Spalte)
- In Factor umwandeln: d\$Spalte <-as.factor(d\$Spalte) oder d\$Spalte <-factor(d\$Spalte, levels=c("m","w","d")))
- Kommata durch Punkte ersetzen: gsub(x=d\$Spalte, pattern=",",replacement = ".")



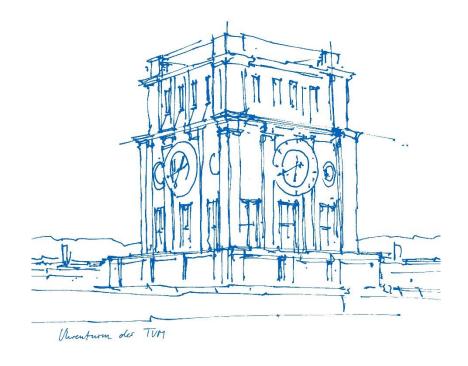
Arbeiten mit Data-Frames II Fehlende Werte

Angelina Mooseder

Technical University of Munich

Bavarian School of Public Policy

mooseder@in.tum.de





Arbeiten mit DataFrames II

3) Mit fehlenden Werten umgehen

- Fehlende Werte einer Spalte erkennen: is.na(d\$Spalte)
- Vollständige Werte einer Spalte erkennen: complete.cases(d\$Spalte)
- Fehlende Werte eines dataframes erkennen: is.na(d)
- Prüfen, ob fehlende Wert in einer Spalte enthalten sind: any(is.na(d\$Spalte))
- Zeilen mit fehlenden Werten in einer Spalte löschen:
 d<-d[!is.na(d\$Spalte),] oder d<-d[complete.cases(d\$Spalte),]
- Zeilen mit fehlenden Werten in dem dataframe löschen: bu<-na.omit(bu)
- Berechnung nur auf kompletten Daten ausführen: mean(d\$Spalte, na.rm=TRUE)



Arbeiten mit Data-Frames III Dataframes verändern

Angelina Mooseder

Technical University of Munich

Bavarian School of Public Policy

mooseder@in.tum.de





Arbeiten mit DataFrames III

4) Dataframes verändern

- Spalte hinzufügen: d\$Neu<-vek oder d[,"Neu"]<-vek
- Spalte entfernen: d<-data.frame(d[,c(1,3:7)])oder d<-data.frame(d[,-2]) oder d[,2]<-NULL oder d\$Alt<-NULL
- Zeile hinzufügen: d[49,]<-list("Müller",24,3) oder d<-rbind(d,list("Müller",24,3))
- Zeile entfernen: d<-data.frame(d[c(1,3:7),]) oder d<-data.frame(d[-2,])



WOCHE 5

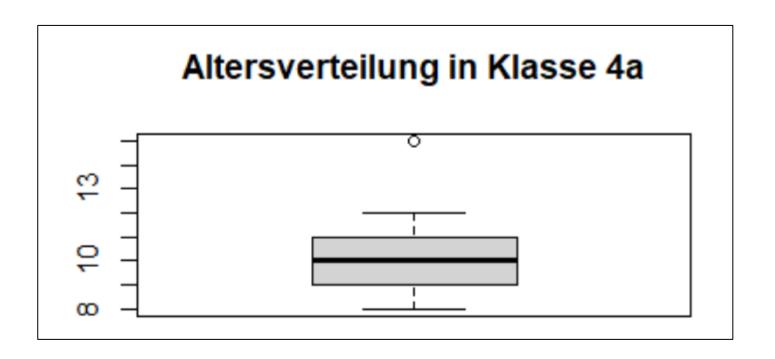


Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de



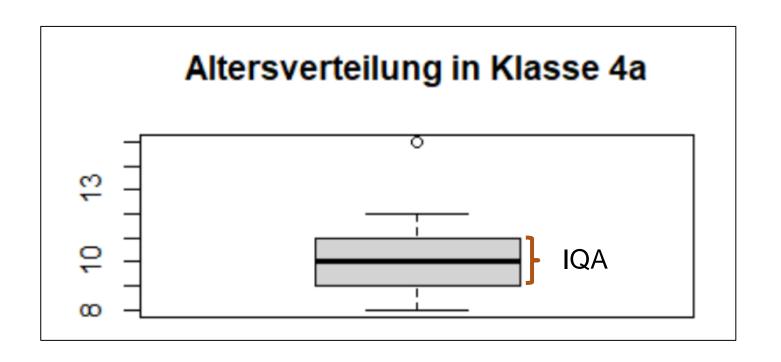


Ausreißer in Boxplots



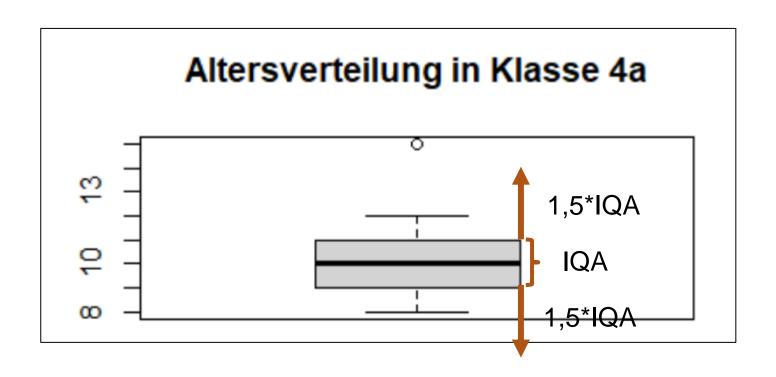


Ausreißer in Boxplots



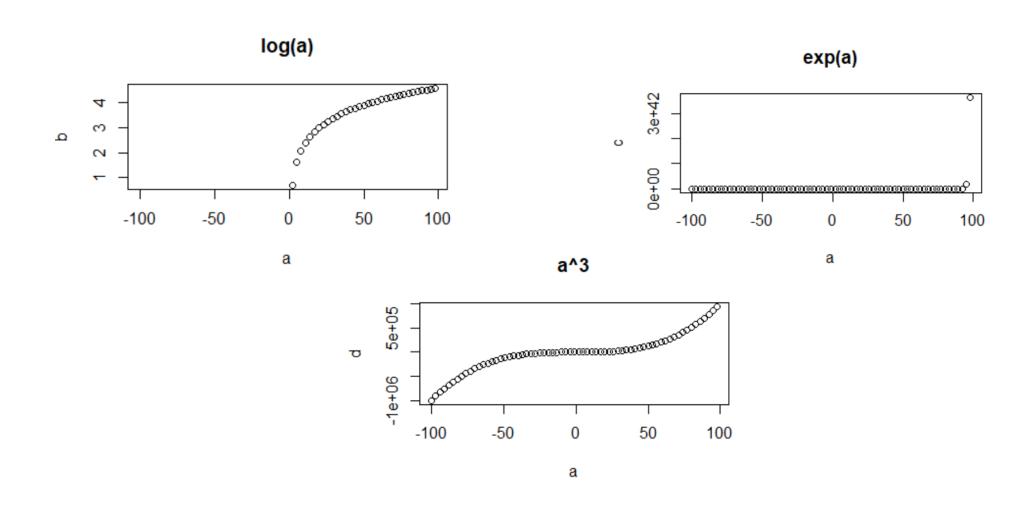


Ausreißer in Boxplots

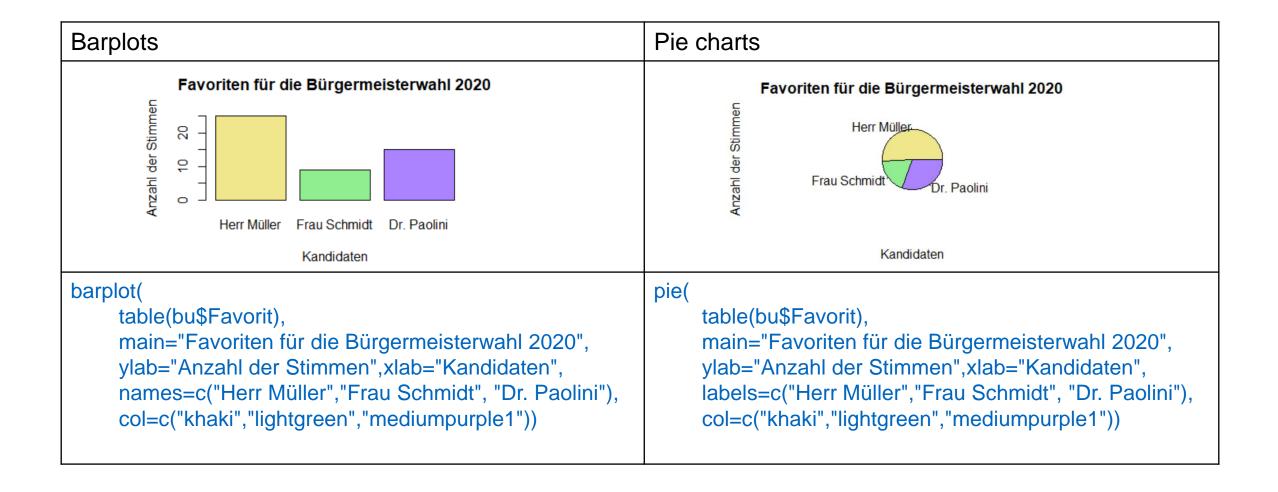




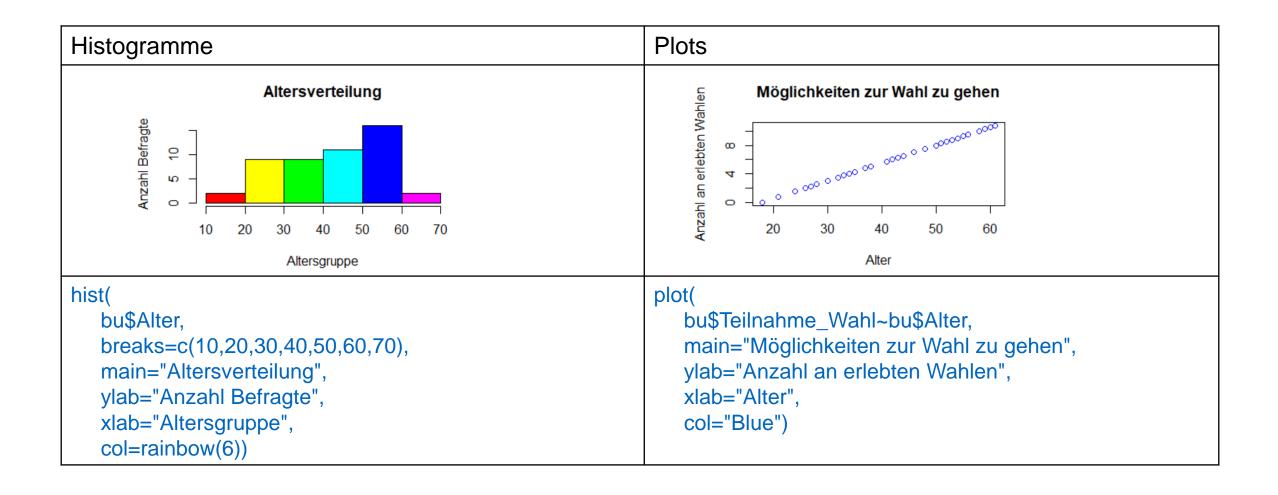
Muster in Plots



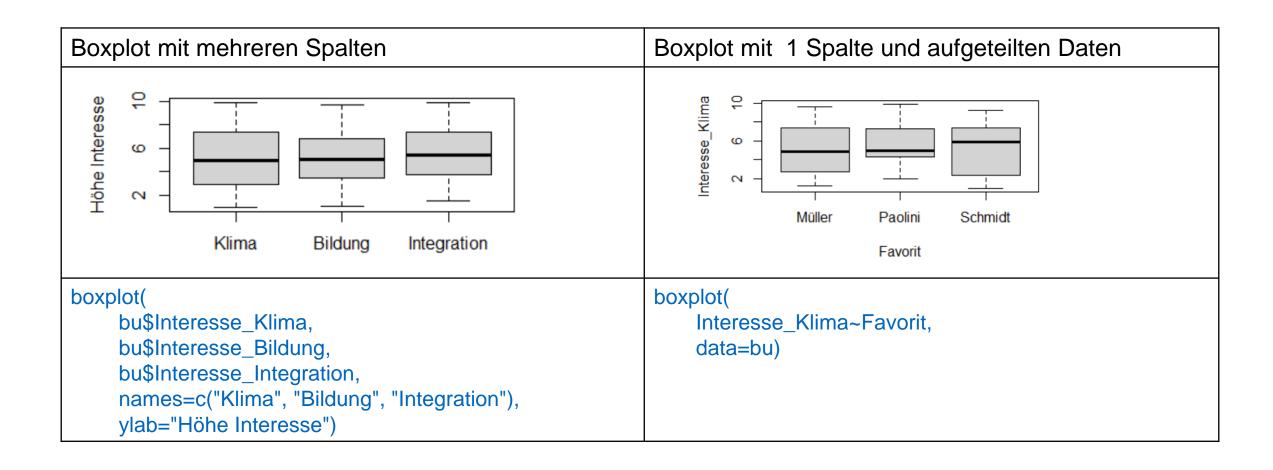














Arbeiten mit Data-Frames IV Gruppen bilden

Angelina Mooseder

Technical University of Munich

Bavarian School of Public Policy

mooseder@in.tum.de







1) Datensatz Bu auswählen

Alter	lnKli.	Favorit	
16	2	Müller	
34	6	Müller	
50	1	Schmidt	
32	3	Schmidt	
27	5	Paolini	



2) Nach Favoriten gruppieren

			Alter	lnKli.	Favorit
Alter	InKli.	Favorit	16	2	Müller
16	2	Müller	34	6	Müller
34	6	Müller	Alter	InKli.	Favorit
50	1	Schmidt	50	1	Schmidt
32	3	Schmidt	32	3	Schmidt
27	5	Paolini			Comme
			Alter	lnKli.	Favorit
			27	5	Paolini

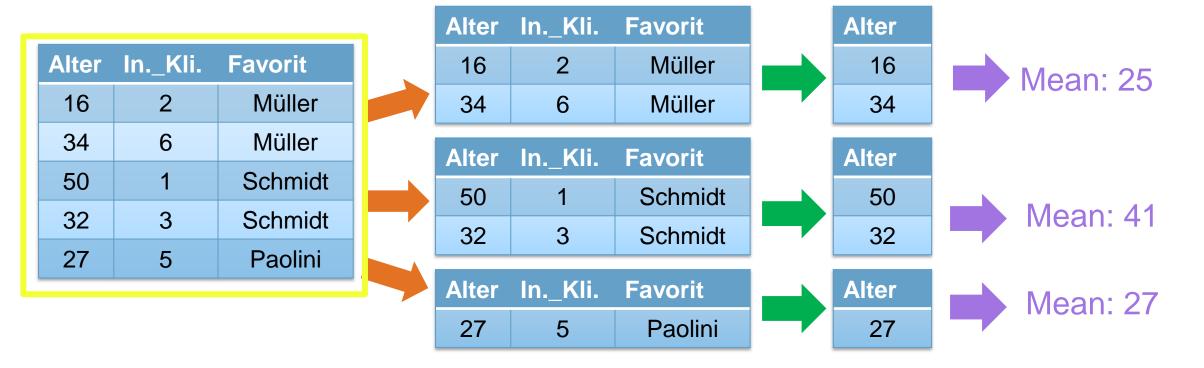


3) Interesse_Klima auswählen





4) Mean berechnen





Arbeiten mit DataFrames IV

5) Gruppen bilden

- Datenset aufteilen:
 split(d\$Noten, f= d\$Geschlecht) (Teilt Spalte Noten nach Kategorien aus Spalte Geschlecht auf)
- Aufgeteiltes Datenset ansprechen:
 split(d\$Noten, f= d\$Geschlecht)\$männlich (Gibt die Noten der männlichen Schüler aus)
- Funktion auf Gruppe anwenden: aggregate (Hausaufgaben~Geschlecht, data=d, FUN=sum) (Berechnet die Summe der Hausaufgaben aus dem Datensatz d für jede Gruppe aus der Kategorie Geschlecht)

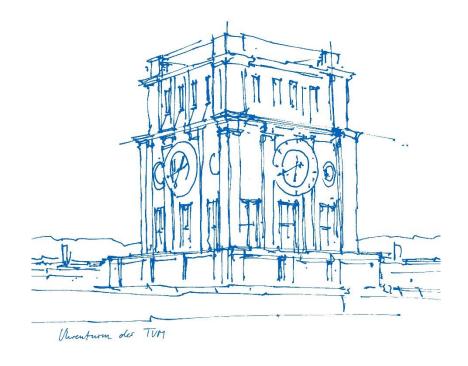


WOCHE 6



Funktionen schreiben

Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de





Eigene Funktionen schreiben

1) Mit einem Argument

Z.B. eine Funktion, die Werte verdoppelt:

```
verdoppeln <- function (x) {
     x*2
}</pre>
```

2) Zusammengesetzte Funktionen

Z.B. eine Funktion, die das Zweifache eines Werts mit dem Dreifachen eines zweiten Werts addiert



Apply-Funktion

Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de





Apply-Funktion

1) Einfache Funktionen

- Mittelwert von jeder Spalte in d: apply(d,2,mean)
- Mittelwert von jeder Zeile in d: apply(d,1,mean)

2) Zusammengesetzte Funktionen

- Spalten herausfinden, die fehlende Werte enthalten:
 - Möglichkeit 1: Funktion aufteilen: apply(is.na(da),2,any)
 - Möglichkeit 2: Funktion definieren: apply(d, 2, function(x){any(is.na(x))})
- Für jede Spalten 1-4 aus dem Datensatz d nach der Spalte Kategorie aus dem Datensatz d gruppieren und von diesen Gruppen den Mittelwert berechnen:
 - apply(d[,1:4],2,function(x){aggregate(x~Kategorie, data=d, mean)})



Merge-Funktion

Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de





Merge-Funktion

1) Datensets verbinden

- Datensets bei gleichem Namen verbinden: m=merge(d, df, by="Namen")
- Datensets bei mehreren Namen verbinden: m=merge(d, df, by=c("Vorname","Nachname"))
- Datensets bei verschiedenen Namen verbinden : m=merge(d, df, by.x="Namen_d", by.y ="Namen_df")
- Datensets bei Zeilennamen verbinden: m=merge(d, df, by="row.names")

2) Datensets mit nicht zusammenpassenden Zeilen verbinden

- Mit allen Werten aus X: m=merge(d, df, by="Namen", all.x=TRUE)
- Mit allen Werten aus Y: m=merge(d, df, by="Namen", all.y=TRUE)
- Mit allen Werten: m=merge(d, df, by="Namen", all=TRUE)



Weitere Funktionen

• Doppelte Zeilen aus Datensatz entfernen: d<-d[!duplicated(d),]



WOCHE 7



Umgang mit Zeichenketten

Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de





Umgang mit Zeichenketten

1) Allgemeine Funktionen

- Anzahl der Zeichen erhalten: nchar(satz)
- Satz in Kleinbuchstaben umwandeln: tolower(satz)
- Satz in Großbuchstaben umwandeln: toupper(satz)

2) Aufteilen und Zusammenfügen

- Zeichenkette splitten: strsplit(satz, split="ein")
- Zeichenkette zusammenfügen: paste("Dies ist", "die Nummer", 1:10)
- Teil der Zeichenkette erhalten: substring(satz, first=1, last=3)

3) Finden und Ersetzen

- Vektorelemente mit Zeichen finden: grep(x=wortvektor,pattern="i")
- Zeichen im Vektor ersetzen: gsub(x=wortvektor,pattern="s", replacement="l")
- Zeichen in Zeichenkette ersetzen: gsub(x=satz,pattern="s", replacement="l")
- Zwei Zeichen ersetzen: gsub(x=gsub(x=satz,pattern="s", replacement="l"), pattern="S", replacement="L")



Umgang mit Datums- und Zeitangaben

Angelina Mooseder

Technical University of Munich

Bavarian School of Public Policy

mooseder@in.tum.de





Datum konvertieren

- %S Sekunde (00–61) (wegen Schaltsekunden nicht bis nur 59)
- %M Minute (00–59)
- %d Tag (01–31)
- %H Stunde (00–23)
- %I Stunde (01–12)
- %m Monat (01–12)
- %b Monatsname (abgekürzt in aktueller Ländereinstellung)
- %Y Jahreszahl mit Jahrhundert (z.B. 2019)
- %y Jahr ohne Jahrhundert (00–99; Die Zahlen ab 69 werden mit einer 19 davor geschrieben, der Rest mit einer 20 davor)



Umgang mit Datums- und Zeitangaben

1) Erstellen

- Datum ohne Zeitangabe (Typ Date):
 - datum <- "05.08.20"
 - datum <- as.Date(datum, "%d.%m.%y")
- Datum mit Zeitangabe (Typ POSIXt):
 - datum <- "05.08.20, 13:30"
 - datum <- strptime(datum, "%d.%m.%y, %H:%M")

2) Rechnen

- Zeitdifferenz berechnen: difftime(datum1,datum2, units="weeks")
- Wochentage erhalten: weekdays(datum)
- Funktionen auf Datumsvektoren anwenden: min(c(datum1,datum2)), mean(c(datum1,datum2))

3) Formatieren

In character umwandeln (für Date und POSIXt): format(datum, "%b %d, %Y: %H Uhr %M")



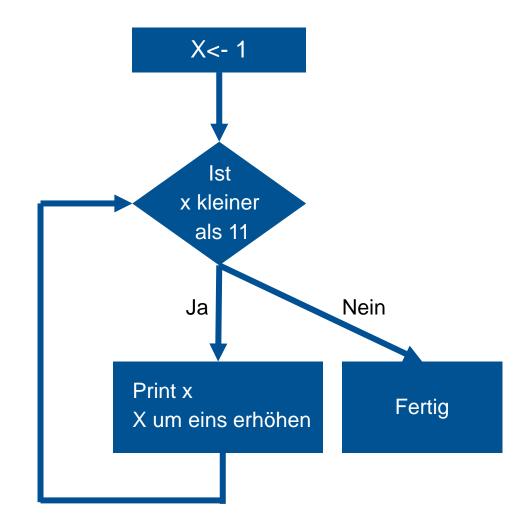
Angelina Mooseder
Technical University of Munich
Bavarian School of Public Policy
mooseder@in.tum.de





Die While-Schleife

```
x<-1
while (x<11) {
    print(x)
    x<-x+1
}</pre>
```





1) Entweder-Oder-Struktur

Mit dem Statement if ... else...:
x<-1
if(x<3)
 {print("x ist kleiner als 3")}
else</pre>

{"x ist größer/gleich 3"}

Mit der ifelse-Funktion:

```
x<-1 ifelse(x<3,"x ist kleiner als 3","x ist größer/gleich 3")
```



2) Schleifen

```
    For-Schleife, z.B.:
        for (x in 1:10) {
            print(x)
        }
    While-Schleife, z.B.:
        x=1
        while (x<11) {
            print(x)
            x<-x+1
        }</li>
```



Weitere Funktionen

- Duplicated-Funktion, z.B. doppelte Zeilen aus Datensatz entfernen: d<-d[!duplicated(d),]
- Print-Funktion, z.B. Wert x in die Konsole schreiben: print(x)
- Sample-Funktion,
 - z.B. zufällige Zahlen aus dem Bereich von 1-100 erhalten: sample(1:100, size=40)
 - z.B. zufällige Zahlen aus dem Bereich von 1-100 erhalten, wobei jede Zahle mehrfach vorkommen darf:
 sample(1:100, size=40,replace=TRUE)