



R Studio vs. SPSS :: Datenaufbereitung

Daten laden

SPSS:

Datei → Daten importieren **oder** Öffnen **oder** Neu

R:

- Working Directory definieren: `setwd()`
Windows: `setwd("C:/Dateipfad")`
MAC: `setwd("/Users/Dateipfad")`
(**Wichtig:** / nicht \)

- Dateiformate einlesen:

```
data <- read.csv("Name des Datensatzes", header = TRUE)
#read.csv #CSV-Format
#read_excel #Excel (Package: readxl)
#read_spss #SPSS (Package: haven) oder
#read_sav #SPSS (Package: haven)
```

→ **Tipp:** R benennt alle missing values aus d. Datensatz in "NA" um. Wenn Sie Ihre Kennzeichnung für fehlende Werte (bspw. -9) beibehalten möchten, nutzen Sie folgenden Befehl: `data <- read_spss("Name des Datensatzes", user_na = TRUE)`

Tipp: Legen Sie Ihren Datensatz als Tibble an. Dann enthält Ihr Datensatz Zusatzinformationen (z.B. Zahl der enthaltenen Befragten) und ist schöner formatiert: (Package: `tidyverse`) `data <- as_tibble(data)`

Tipp: Wenn Sie für bspw. die Befragten eine Identifikationsnummer haben, können Sie auch folgenden Befehl benutzen, um die Identifikationsnummer in die Zeilennummer umzuwandeln (Package: `dplyr`): `data <- data %>% mutate(identifikationsnummer = row_number())` So ist Befragte/r 1 auch Fall/Zeile 1 in R.

Tipp: Wenn Sie nachprüfen möchten, wie Ihre aktuelle Working Directory definiert ist, können Sie einfach den Befehl `getwd()` ausführen.

Tipp: Wenn Sie nicht mehr genau wissen, welche Dokumente sich in der von Ihnen gesetzten Working Directory befinden, nutzen Sie den Befehl `list.files()`. So können Sie auch Ihren Datensatznamen kopieren und Rechtschreibfehler umgehen (bspw. weil Sie „.csv“ vergessen hätten).

Im Code kommentieren

SPSS:

```
*kommentar
FREQUENCIES VARIABLES=ALTER GESCHLECHT
/ORDER=ANALYSIS
```

```
R: #kommentar1
data %>%
  tab_frequencies( #kommentar2
    alter, #kommentar3
    geschlecht) #kommentar4
#kommentar5
```

→ **Nicht:** am Anfang der Zeile oder vor `%>%` oder in einer Klammer, die nicht durch einen Absatz aufgetrennt wird.

Zwei Datensätze zusammenfügen:

SPSS:

- Längsschnittdaten: Daten → Dateien zusammenfügen → Variablen hinzufügen
- Querschnittdaten: Daten → Dateien zusammenfügen → Fälle hinzufügen

R: (Package: `dplyr`)

- Variablen aus einem Datensatz ergänzen:

a. Neuer Datensatz = Die Fälle, die in beiden Datensätzen vorkommen, zusammenfügen: `data_komplett <- inner_join(datensatz1, datensatz2, by = "case_id")` Die Variable `case_id` gibt uns in diesem Beispiel die Identifikationsnummer der Befragten an. Dadurch kann R die Fälle zusammenfügen, die in beiden Datensätzen dieselbe Nummer haben. Die Befragten, die nur in einem Datensatz vorkommen, werden gelöscht.

b. Neuer Datensatz = Die Fälle, die (in der Klammer) im linken Datensatz enthalten sind, werden mit Informationen aus dem rechten Datensatz ergänzt: `data_komplett <- left_join(datensatz1, datensatz2, by = "case_id")` Bei diesem Befehl werden alle Variablen, die im rechten Datensatz (= Datensatz, der in der Klammer an der zweiten Position steht) neu sind, zum linken Datensatz hinzugefügt. Die Fälle aus dem linken Datensatz werden dabei alle behalten. Alle Fälle, die nur im rechten Datensatz vorkommen (und nicht im linken), werden gelöscht.

c. Neuer Datensatz = Die Fälle, die (in der Klammer) im rechten Datensatz enthalten sind, werden mit Informationen aus dem linken Datensatz ergänzt: `data_komplett <- right_join(datensatz1, datensatz2, by = "case_id")` Bei diesem Befehl werden alle Variablen, die im linken Datensatz (= Datensatz, der in der Klammer an der ersten Position steht) neu sind, zum rechten Datensatz hinzugefügt. Die Fälle aus dem rechten Datensatz werden dabei alle behalten. Alle Fälle, die nur im linken Datensatz vorkommen (und nicht im rechten), werden gelöscht.

- Fälle ergänzen: `data_komplett <- full_join(datensatz1, datensatz2, by = "case_id")` Bei diesem Befehl werden alle Fälle aus beiden Datensätzen zusammengefügt (doppelte Fälle werden dabei auch ergänzt). Bei Variablen, die nicht in beiden Datensätzen vorkommen, werden NAs für die Fälle eingetragen, die bei der Variable keinen Wert haben.

Hinweis: Die Variablen im kompletten Datensatz haben nun ein Suffix `.x` oder `.y`: `.x` kommt aus dem ersten Datensatz in der Klammer und `.y` aus dem zweiten Datensatz.

Hinweis: Alles, was nicht im Argument `by =` definiert wird, wird neu angelegt.

Transformation des Datensatzes: Wide-/ Long-Format:

SPSS:

- Vom Wide-Format zum Long-Format: Daten → Umstrukturieren → Umstrukturieren ausgewählter Variablen in Fälle → Definieren: Angabe von Fallgruppen/zu transportierende Variablen bzw. Zielvariable (wie Messwiederholungen)/Variable(n) mit festem Format, bzw. nicht mehrfach gemessene Variable(n) → Definieren d. Anzahl von Indexvariablen → Art des Indexwerts (Codes, vs. Variablenamen) → Fertigstellen

- Vom Long-Format zum Wide-Format: Daten → Umstrukturieren → Umstrukturieren ausgewählter Fälle in Variablen → ID-Variablen und Indexvariablen auswählen → Sollen die Daten sortiert werden? (Ja) → Fertigstellen

Wichtig: Bei beiden Varianten den originalen Datensatz separat abspeichern (äquivalent zu `<-` in R)

R: (Package: `tidyr`)

- Vom Wide-Format zum Long-Format: `data_long <- data %>% pivot_longer(cols = everything(), names_to = c("variable1", "variable2", "variable3"), values_to = "value")`
- Vom Long-Format zum Wide-Format: `data_wide <- data %>% pivot_wider(names_from = c("Spaltenname1", "Spaltenname2", "Spaltenname3"), values_from = "value")`

Datenansicht/Variablenansicht:

SPSS: In der unteren Leiste zwischen Datenansicht und Variablenansicht switchen

R:

- Datenansicht: klicken Sie doppelt auf den Datensatz in der Environment. Es öffnet sich ein Fenster. **Oder:** Nutzen Sie die Funktion `View(data)` **Achtung:** Bei `View()` wird das V großgeschrieben!
- Variablenansicht: Klicken Sie dafür auf den blauen Pfeil in der Environment. So sehen Sie, ob die Variablentypen character/numeric sind, etc.

data 878 obs. of 216 variables

Oder: (Package: `magittr` für Pipe-Operator `%>%`)
Nutzen Sie folgenden Befehl: `data %>% str()`

Operatoren

Bedeutung	SPSS	R
Mathematische Symbole		
Gleiche Symbole	+ - * / =	+ - * / =
Exponentialfunktion	**	^
Modulo Operation		%%
Integer Division		%/%
Logische Operatoren		
Gleiche Symbole	< > <= >= & xor TRUE FALSE	< > <= >= & xor TRUE FALSE
„Ist gleich“	=	== (außer bei d. Schreibweise v. Funktionen)
Negation	~	!
„Enthalten in“		%in%
„von ... bis ...“		:
Der Rest/Alles andere	ELSE	other

Variablen/Spalten umbenennen:

SPSS: Doppelt auf die Variable klicken und den neuen Namen eintragen
R: (Package: dplyr)
 1. Ausgeben der einzelnen Variablenamen: `data %>% names()`
 2. Eine Spalte umbenennen: `data <- data %>% rename("Neuer Name" = alter_name)`
 3. Mehrere Spalten umbenennen: `data <- data %>% rename(c("Neuer Name1" = alter_name1, "Neuer Name2" = alter_name2))`

Reihenfolge der Spalten ändern:

SPSS: Variable/n markieren und nach oben/unten verschieben
R: (Package: dplyr)
 1. Verschiebe die Spalte/Variable1 hinter die Spalte/Variable2: `data <- data %>% relocate(variable1, .after = variable2)`
 2. Verschiebe die Spalte/Variable2 vor die Spalte/Variable1: `data <- data %>% relocate(variable2, .before = variable1)`

Fälle/Spalten im Datensatz zählen (N; n):

SPSS:
 1. Datenansicht (=N)
 2. n: Analysieren → Deskriptive Statistiken → Häufigkeiten → OK
 3. Variablenanzahl: Variablenansicht
R: (Package: dplyr)
 1. N: `data %>% nrow()` (auch kombinierbar mit Bedingungen, bspw.: `data %>% na.omit() %>% nrow()` gibt aus, wie viele Fälle im Datensatz sind, die keine NAs haben) **Oder:** wie viele obs. werden in der Environment beim Datensatz angezeigt? (= N)
 2. n: `data %>% summarize(count = n())`
 3. Anzahl der Spalten: `data %>% ncol()`

Missing Values/ Fehlende Werte/ NAs identifizieren:

SPSS: Transformieren → Variable berechnen → MISSING()
R: `is.na(data)`
Auch: (Package: dplyr)
 1. Liegt überhaupt ein fehlender Wert im Datensatz vor?: `data %>% is.na() %>% any()` (Ausgabe: **TRUE** = Ja, es gibt mind. einen NA; **FALSE** = es gibt keinen NA)
 2. Wie viele fehlende Werte hat der Datensatz?: `data %>% is.na() %>% sum()` **oder umgedreht:** Wie viele gültige Fälle hat der Datensatz? `sum(!is.na(data))`
 3. Liegt irgendwo womöglich ein Fehler vor? Ausgeben eines Überblicks über jede Spalte/Variable (Mittelwert, Median, NAs, Minimum, Maximum): `data %>% summary()`

Missing Values/ Fehlende Werte/ NAs löschen oder ersetzen (Imputation):

SPSS:
 1. Werte löschen: Daten → Fälle auswählen → Falls Bedingung zutrifft → ~ MISSING() → Nicht ausgewählte Fälle löschen
 2. Fälle ersetzen: Transformieren → Fehlende Werte ersetzen → Name und Methode: neue Variable mit den ersetzten Werten benennen → gewünschte Methode auswählen
R: (Package: dplyr)
 1. Löschen: `data_ohne_na <- data %>% na.omit()` (löscht alle Zeilen, die mind. einen fehlenden Wert enthalten)
 2. Fehlende Werte einer Variable ersetzen: (Package: tidyr) `data <- data %>% mutate(variable = replace_na(variable, neuer Wert))`
 3. Fehlende Werte im ganzen Datensatz ersetzen: (Package: tidyr) `data <- data %>% mutate(across(everything(), ~replace_na(., gewünschter Wert)))`

Missing Values/Fehlende Werte/ NAs definieren:

SPSS: Fehlend → Einzelne fehlende Werte/Spaltenweite
R: (Package: tidycomm) `data <- setna_scale(variable, value = Wert)`
Wichtig: Die Variable, in der der NA-Wert definiert bzw. gesetzt wurde, heißt nun Variablenname **_na** (**oder overwrite = TRUE**, sh. nächste Seite).

Z-Score berechnen/ Ausreißer identifizieren:

SPSS: Analysieren → Deskriptive Statistiken → Deskriptive Statistik... → Haken bei „Standardwerte als Variablen speichern“ setzen → OK
Wichtig: Die neue Variable heißt nun Zvariablenname.
R: (Package: dplyr)
 1. Folgende Funktion berechnet den Z-Score für alle Zellen von numerischen Variablen im Datensatz: `data_z_scores <- data %>% select(where(is.numeric)) %>% scale()`
 2. Folgende Funktion berechnet den Z-Score für ausgewählte Variablen: `data <- data %>% mutate(z_variable1 = scale(variable1), z_variable2 = scale(variable2))`

Duplikate ermitteln und entfernen:

SPSS:
 1. Duplikate ermitteln: Daten → Doppelte Fälle ermitteln
 2. Duplikate entfernen:
 a. Löschen: Einzelne Duplikate mit Rechtsklick löschen
 b. Filtern: Daten → Fälle auswählen → Funktion **PrimaryLast=1** eingeben
R:
 1. Duplikate ermitteln: `data_duplikate <- duplicated(data)` (Ergebnis in der Console zeigt, welche Fälle Duplikate sind: **TRUE** = Duplikat; **FALSE** = nicht Duplikat)
 2. Duplikate entfernen: (Package: dplyr)
 a. Alle Duplikate entfernen und gesäuberten Datensatz in neuem Objekt speichern: `data_neu <- data %>% distinct()` **oder:** `data_neu <- unique(data)`
 b. Duplikate von ausgewählten Variablen entfernen: `data_neu <- data %>% distinct(variable(n), .keep_all = TRUE)`

Dummy-Variable erstellen:

SPSS:
 1. Manuell: Transformieren → Umcodieren in andere Variablen → Die Ausgangsvariable definieren und die Zielvariable benennen → Alte und neue Werte... → Weiter → OK
 2. Automatisch: Transformieren → Dummy-Variable erstellen → Dummy-Variable erstellen für: Ausgangsvariable definieren → Stammnamen definieren → OK
R: (Package: dplyr, tidycomm) `data <- data %>% select(variable) %>% dummify_scale(variable)`

Gruppen erstellen & Variablen umkodieren:

SPSS: Transformieren → Umkodieren in andere Variable → Alte und neue Werte

R:

- Nominal- oder ordinalskalierte Variablen umkodieren: (Package: `tidycomm`, `dplyr`)
 - String-Werte: `data <- data %>% recode_cat_scale(variable, assign = c("alter Wert1" = "neuer Wert1", "alter Wert2" = "neuer Wert2"))`
 - Integers (Zahlenwerte): `data <- data %>% recode_cat_scale(variable, assign = c('Zahl1' = "Bezeichnung1", 'Zahl2' = "Bezeichnung2"), overwrite = TRUE)`
 → Wenn "alle anderen Werte" als "other" (in SPSS: ELSE) betitelt werden sollen: `data <- data %>% recode_cat_scale(variable, assign = c("alter Wert1" = "neuer Wert1", "alter Wert2" = "neuer Wert2"), other = "other")`

Wichtig: Die umkodierte Variable heißt jetzt `variable_rec`

- Intervall- bzw. verhältnisskalierte Skala in eine nominal- oder ordinalskalierte Skala umkodieren und Gruppen erstellen: (Package: `tidycomm`) `data <- data %>% categorize_scale(variable, lower_end = Minimum der alten Skala, upper_end = Maximum der alten Skala, breaks = c(Werte, nach denen eine neue Gruppe beginnt), labels = c("Name Gruppe1", "Name Gruppe2"))`

Wichtig: Die neue Variable heißt jetzt `variable_cat`

Parameter `overwrite = TRUE` beim Umkodieren mit `tidycomm`:

R: Mit `overwrite = TRUE` (sh. bspw. oben „Gruppen erstellen & Variablen umkodieren“) wird eine Variable überschrieben. Wenn dieser Parameter im Beispiel von der Funktion `recode_cat_scale()` angewendet wird, wird also keine neue Variable `variable_cat` erstellt. Anwendbar ist dieser Parameter immer außer bei `dummify_scale()`.

Datensatz nach Variable(n) gruppieren/ gruppenweise auswerten:

SPSS: Daten → Datei aufteilen → Ausgabe nach Gruppen aufteilen → Dann: gewünschten Befehl ausführen

R: (Package: `dplyr`) `data %>% group_by(variable) %>% ...` → Diese Funktion ist für weitere Berechnungen/ Funktionen wie `tab_frequencies()` nützlich.

Fälle und Variablen auswählen:

SPSS: Daten → Fälle auswählen → Falls Bedingung zutrifft (Ausgabe: Nicht ausgewählte Fälle filtern)

R: (Package: `dplyr`)

- Ganze Variablen auswählen (Spalte ausgeben): `data %>% select(variable)`
Nützliche Funktionen, die Sie in `select()` einbetten können:
`select(starts_with("String"), ends_with("String"), contains("String"), last_col(), where(is.character), is.numeric, is.integer, is.double, is.logical, is.factor)`
- Bestimmte Fälle/Werte einer Variable auswählen (Zeile ausgeben): `data %>% filter(variable1 == Wert & variable2 == "String")`
 Tipp: (Package: `stringr`) Wenn Sie alle Fälle mit einem bestimmten String ausgegeben haben möchten, können Sie auch folgende Funktion verwenden: `str_detect(variable, "String, der enthalten ist")`
Da `str_detect()` sensibel bzgl. Groß-/ Kleinschreibung ist, kann man bei Zweifeln, ob der String groß/kleingeschrieben wird, auch `[]` nutzen. Bspw. `[sS]tring`

Tipp: Statt bei vielen Bedingungen wiederholt „|“ zu nutzen, können Sie auch `c(..., ..., ...)` nutzen (bspw. `select(c("variable1", "variable2", "variable3"))`)

Wichtig: Während in SPSS die Fälle ausgewählt werden, die man ausschließen möchte, werden in R mit `filter()` und `select()` die Fälle ausgewählt, die man behalten möchte!

Fälle sortieren:

SPSS: Daten → Fälle sortieren → (Sortierreihenfolge) Auf/Absteigend

R: (Package: `dplyr`)

- Aufsteigend: `data %>% arrange(variable)`
- Absteigend: `data %>% arrange(-variable) oder data %>% desc(variable)`

Invertieren von Variablen (hier: 5 wird 1 und 1 wird 5):

SPSS: Transformieren → Umkodieren in andere Variable → Alte und neue Werte

R: (Package: `tidycomm`) `data <- data %>% reverse_scale(variable, lower_end = 1, upper_end = 5)`

Wichtig: Die umkodierte Variable heißt jetzt `variable_rev` (`oder overwrite = TRUE`).

Stetige Variable zentrieren:

SPSS: Daten → Aggregieren → gewünschte Variable in „Zusammenfassungen von Variablen“ ziehen (unter „Funktion...“ prüfen, ob der Mittelwert berechnet wird) → OK → Dann: Transformieren → Variable berechnen → Zielvariable benennen → Numerischer Ausdruck: Variable Minus die zuvor berechnete Variable_mean

R: (Package: `tidycomm`) `data <- data %>% center_scale(variable)`

Wichtig: Die umkodierte Variable heißt jetzt `variable_centered` (`oder overwrite = TRUE`).

Z-standardisierte Variable:

SPSS: Analysieren → Deskriptive Statistiken → Deskriptive Statistik → Haken setzen bei „Standardisierte Werte als Variable speichern“

Oder: Transformieren → Variable berechnen → Formel für die Z-Standardisierung eingeben

R: (Package: `tidycomm`) `data <- data %>% z_scale(variable)`

Wichtig: Die umkodierte Variable heißt jetzt `variable_z` (`oder overwrite = TRUE`).

Range/ Spannweite der Skala erweitern (hier: 1 bis 5 wird 1 bis 10; die Abstände bleiben gleich groß):

SPSS: /

R: (Package: `tidycomm`) `data <- data %>% minmax_scale(variable, change_to_min = 1, change_to_max = 10)`

Wichtig: Die umkodierte Variable heißt jetzt `variable_1to10` (`oder overwrite = TRUE`).

Perzentile angeben lassen (Perzentiltabelle):

SPSS: Analysieren → Deskriptive Statistiken → Perzentile

Oder: Analysieren → Deskriptive Statistiken → Häufigkeiten → Statistiken (Quartile/Perzentile)

R: (Package: `tidycomm`) `data %>% tab_percentiles(variable)`

Häufigkeiten:

SPSS: Analysieren → Deskriptive Statistiken → Häufigkeiten → Statistiken

R: (Package: `tidycomm`) `data %>% tab_frequencies(variable)`

Kennzahlen für nominal- und ordinalskalierte Variablen (N, Missing, Modus, wie oft Modus vorkommt, Anzahl der Kategorien):

SPSS: Analysieren → Deskriptive Statistiken → Deskriptive Statistik → Optionen

Oder: Analysieren → Deskriptive Statistiken → Häufigkeiten → Statistiken

R: (Package: `tidycomm`) `data %>% describe_cat(variable)`

Streuungsmaße und Lagemaße für metrische Variablen (N, Missing, Mittelwert, SD, Min, Max, Q25, Median, Q75, Range, Konfidenzintervalle (CI_95LL, CI_95_UL)):

SPSS: Analysieren → Deskriptive Statistiken → Deskriptive Statistik → Optionen

Oder: Analysieren → Deskriptive Statistiken → Häufigkeiten → Statistiken

R: (Package: `tidycmm`) `data %>% describe(variable)`

Wichtig: Den **Interquartilsabstand** berechnet R hier nicht. Durch den folgenden Befehl können Sie diesen aber mitberechnen lassen: (Package: `tidycmm`, `dplyr`) `data %>% describe(variable) %>% mutate(IQR = Q75 - Q25)`

Dasselbe mit der Varianz: Fügen Sie dafür einfach folgenden Befehl hinzu: `mutate(Variance = SD^2)`

Tipp: Mit dem Befehl `?tidycmm::describe()` werden die Spalten der Ausgabetable genau erklärt.

Alternativ: Streuungsmaße mit `dplyr` berechnen:

```
data %>% summarize(Range = max(variable) - min(variable), Q25 =  
quantile(variable, 0.25), Q75 = quantile(variable, 0.75), Perc80 =  
quantile(variable, 0.80), IQR = IQR(variable), Variance = var(variable), SD =  
sd(variable), V = sd(variable) / mean(variable))
```

Tipp: Mit dem Befehl `na.rm = TRUE` werden die NAs ignoriert:

```
data %>% group_by(gruppierungsvariable) %>% summarize(MAX = max(variable, na.rm =  
TRUE), UQ = quantile(variable, 0.75, na.rm = TRUE), Mdn = median(variable, na.rm  
= TRUE), M = mean(variable, na.rm = TRUE), Mode =  
names(which.max(table(variable))), SD = sd(variable, na.rm = TRUE), LQ =  
quantile(variable, 0.25, na.rm = TRUE), MIN = min(variable, na.rm = TRUE), count  
= n())
```

Variablen als Grafiken veranschaulichen:

SPSS: Grafik → Diagrammerstellung...

R: (Package: `tidycmm`)

1. Für nominal- oder ordinalskalierte Variablen:

`describe_cat()` `%>% visualize()`: Balkendiagramm, das absolute Häufigkeiten abbildet
`tab_frequencies()` `%>% visualize()`: Balkendiagramm, das relative Häufigkeiten abbildet

2. Für metrische Variablen:

`describe()` `%>% visualize()`: Boxplot
`tab_frequencies()` `%>% visualize()`: Histogramm

Variablen konvertieren:

SPSS: /

R:

1. Variablentypen prüfen: `is.character(variable)` (Ausgabe: `TRUE` = es ist ein character; `FALSE` = es ist kein character). Sie können auch die weiteren Variablentypen (sh. unten) prüfen. Dafür schreiben Sie einfach vor `numeric/ etc.` ein `is`.

2. Variable konvertieren: `data <- data %>% mutate(variable =`
`as.character(variable))`
`as.numeric(variable)`
`as.factor(variable)`
`as.vector(variable)`

Weitere nützliche Funktionen in R:

1. Zu Beginn des Codes können Sie die Funktion `rm(list = ls())` ausführen. Dieser Befehl löscht alles, was aktuell in der Environment ist. Sie können mithilfe von `rm(data)` auch einen speziellen Datensatz (oder mehrere) entfernen.
2. Folgende Funktion zeigt Ihnen alle Objekte (Datensätze, Variablen), die Sie aktuell in der Environment haben, an: `ls()`
3. Mit folgenden Funktionen können Sie Hilfe zu bestimmten Funktionen bekommen. Nach der Ausführung dieser Befehle öffnet sich ein neues Ausgabefenster mit einer Beschreibung: `help(Funktion)` **oder:** `?Funktion`
4. Mit `example(Funktion)` können Sie sich einen Beispiel-Code für eine bestimmte Funktion ausgeben lassen. Damit können Sie Funktionen womöglich besser nachvollziehen und ihren Aufbau verstehen.
5. Wenn eine Berechnung einen großen Output hat, können Sie die Ergebnisse in einem neuen R-Objekt speichern und die Ergebnisse mit der Funktion `View(output_als_r_objekt)` in einem sich neu öffnenden Fenster betrachten. **Oder** Sie schreiben um den Befehl `View()`: `View(Befehl)` Dann wird Ihr Ergebnis nicht abgespeichert, aber in einem neuen Fenster angezeigt.