Probeklausur: Einführung in die Statistische Software (R)

Stefanie Peschel, Philip Boustani

22 January 2025

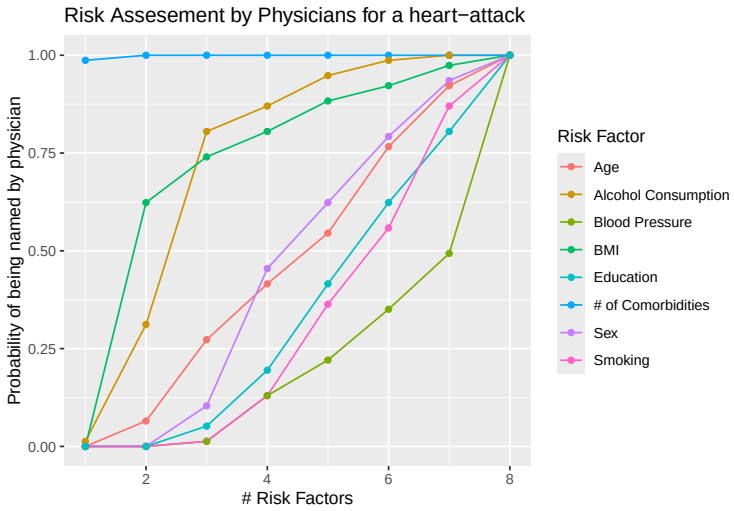
Bitte führen Sie zuerst folgenden R-Markdown-Block aus:

```
renv::restore()
```

Aufgabe 1: Plot reproduzieren (15 P.)

Führend Sie den folgenden R-Markdown-Block aus:

```
load("Q1plot.RData")
Q1plot.png
```



Reproduzieren Sie die Grafik "Q1plot.png" in Ihrem Repository so gut wie möglich. Verwenden Sie bei Bedarf die `tidyverse`-Pakete zum Preprocessen und verwenden Sie `ggplot2` zum Plotten. Die Roh-Daten für den Plot befinden sich in der Datei `umfragen.Rds` in Ihrem repository.

Eine Reproduktion in deutscher Sprache ist zulässig.

Hintergrund zur Grafik

Bei der Datenerhebung wurden tausende Mediziner gefragt, welche sie für die Top X (1-8) Risikofaktoren für einen Herz-Infarkt halten. Insgesamt konnten bis zu 8 Risikofaktoren genannt werden. Beispiel: Wenn ein einziger Risikofaktor gennant werden durfte (erste Zeile des Datensatzes), sagten 98,7013 Prozent der Ärzte "Anzahl Erkrankungen" (`numberComorbidities`) und 1,298701 Prozent gaben Alkoholkumsum an (`alcoholConsumption`). Wenn die Ärzte zwei Risikofaktoren angeben durften (zweite Zeile des Datensatzes), nannten alle Ärzte "Anzahl Erkrankungen", andere Risikofaktoren wurden unterschiedlich häufig genannt, etc.

In der Graphik zeigt - die X-Achse zeigt die Anzahl an Risikofaktoren, die genannt werden durften. - die Y-Achse zeigt den Anteil der Mediziner, die bei gegebener Anzahl an nennbaren Risikofaktoren einen der 8 möglichen Risikofaktoren gennant hat.

Ihr Code:

```
umfragen <- readRDS("umfragen.Rds")

# TODO
```

Tests:

Für diese Aufgabe gibt es keine Tests.

Aufgabe 2: Temperaturumrechnung (15 P.)

Schreiben Sie eine Funktion zur Umrechnung der Temperatur von Celsius in Fahrenheit (C -> F und F -> C).

Die Umrechnung erfolgt wie folgt: - °C = (°F - 32) * 5/9 (von Fahrenheit in Celsius) - °F = °C * 1,8 + 32 (von Celsius nach Fahrenheit)

Achten Sie darauf, dass alle Funktionsargumente (Inputs) nur erlaubte Werte und Datentypen annehmen. Die minimale erlaubte Temperatur ist -459.67.

Details zu der Funktion

Die Funktion ist wie folgt benannt: `ex02temperature_conversion`.

Eingabe

- `temp`: Numerischer Vektor ohne fehlende Werte. Die Temperatur(en), die umgewandelt werden soll(en).
- `unit`: Die Einheit, in der `temp` angegeben ist."F" für Fahrenheit

Ausgabe

`numeric(n)`.

Beispiele

```
ex02temperature_conversion(c(212, 32))
```

```
## [1] 100   0
```

```
ex02temperature_conversion(100, unit = "C")
```

```
## [1] 212
```

```
ex02temperature_conversion(-9, unit = "CC")
```

```
## Error in match.arg(unit, c("F", "C")): 'arg' should be one of "F", "C"
```

```
ex02temperature_conversion(-460)
```

```
## Error in ex02temperature_conversion(-460): Assertion on 'temp' failed: Element 1 is not >= -459.67.
```

Ihr Code:

```r
ex02temperature_conversion <- function(temp, unit = "F") {

  # TODO

}
```

Tests:

(2.1) Einfach: Ihre Funktion wandelt Temperaturen korrekt zwischen Celsius und Fahrenheit um. (5 P.)

```r
test <- "2.1"; source("evaluate_test.R")
```

(2.2) Einfach: Ihre Funktion gibt bei ungültigen Eingaben (falsche Einheit oder Temperaturen außerhalb des physikalisch sinnvollen Bereichs) Fehlermeldungen aus. (5 P.)

```r
test <- "2.2"; source("evaluate_test.R")
```

(2.3) Fortgeschritten: Ihre Funktion rechnet weitere Temperaturen korrekt um und gibt Fehlerbehandlung bei ungültigen Eingabeformaten aus (uneinheitlichen Angabe von Einheiten für Vektoreingaben). (5 P.)

```r
test <- "2.3"; source("evaluate_test.R")
```

Aufgabe 3: Einkaufsprozess (30 P.)

Betrachten Sie die Daten `SAP.Rds` mit den Datensätzen `EKKO`, `EKPO` und `LFA1`, die aus einer SAP-Datenbank stammen.

Die Datensätze (oder Tabellen) sind über die Spalten "MANDT", "EBELN" und (`EKKO` und `EKPO`) oder "MANDT" und "LIFNR" (`EKKO` und `LFA1`) miteinander verbunden. Genauer beschäftigen wir uns hier mit einem Einkaufsprozess, in dem ein Unternehmen Bestellungen ("EBELN") mit Bestellpositionen ("EBELP") aufgibt. Zu den Bestellungen und Bestellpositionen haben Sie einige zusätzlichen Informationen in den Tabellen "EKKO" und "EKPO", zu den Lieferanten, bei denen gekauft wurde in "LFA1".

Die Datei "SAP_Glossar.pdf" enthält eine Beschreibung aller Datensätze und Variablen, welche in der SAP-Datenbank enthalten sind.

Teilaufgabe A (15 P.)

Die Spalte "NETWR" in `EKPO` ist falsch formatiert. Korrigieren Sie sie, damit die Datenbank standardisiert ist.

Legen Sie dazu eine neue Spalte "WAERS" an, die die Währung aus der Zeichenkette NETWR enthält. NETWR sollte nur den Wert ohne jegliche Währung enthalten.

Es gibt vier Währungen: Euro, Pfund, Dollar und Yen.

Die auftretenden Unstimmigkeiten bestehen entweder darin, dass die Währung manchmal vor und manchmal nach dem Betrag steht, oder es gibt manchmal Leerzeichen zwischen dem Betrag und der Währung oder die Verwendung von Text und Symbolen ist uneinheitlich. Jede Währung kann auf vier Arten auftreten: Klartext (Klein- und Großbuchstaben, z. B. Yen oder Yen), Symbol (z. B. ¥) oder internationale Variante (z. B. JPY).

Die vollständige Liste umfasst:

- USD
- $
- USD
- Euro
- EUR

- €
- euro
- Yen
  - Yen
  - JPY
  - ¥
- GBP
  - gbp
  - £

Sie können dieser Anleitung folgen (es gibt aber sicher auch andere Wege zum Ziel):

i) Machen Sie alle Zeichenfolgen in NETWR zu Kleinbuchstaben.

ii) Entfernen Sie Leerzeichen aus den Zeichenfolgen ("trimmen" Sie diese).

iii) Standardisieren Sie die verschiedenen Ausdrücke für eine Währung auf "eur", "usd", "jpy" und "gbp".

iv) Ändern Sie die Ausdrücke in die Großschreibung: "EUR", "USD", "JPY" und "GBP".

v) Trennen Sie NETWR: Suchen Sie die drei Währungen in NETWR und übertragen Sie sie in eine neue Spalte WAERS.

vi) Entfernen Sie dann die Währungszeichenfolgen aus NETWR. Tipp: Die regular Expression für "alles außer Characters" ist "[^0-9.-]".

vii) Stellen Sie sicher, dass NETWR numerisch ist.

Speichern Sie die verbesserte Tabelle EKPO als "ex0303a".

```
ex0303a <- NULL
```

Ihr Code:

Tests: (3.a) `ex0303a` ist ein Dataframe mit korrekten Dimensionen, Spaltennamen, Summen- und Häufigkeitswerten in den numerischen Spalten. (15 P.)

```
test <- "3.1"; source("evaluate_test.R")
```

Teilaufgabe B (15 P.)

Unabhängig von Ihrem Ergebnis in a) verwenden Sie die in der Datei NETWR.Rds bereitgestellten Spalten als neue Werte für NETWR bzw. WAERS (d.h. überschreiben Sie die entsprechenden Spalten in EKPO mit den Spalten in `NETWR.Rds`).

Berechnen Sie den Gesamtbestellwert pro Lieferant (`LIFNR`), der vor dem 03.07.2021 bestellt (`AEDAT`) und in Euro bezahlt hat (hierzu brauchen Sie auch den `EKKO` Datensatz aus dem `SAP` Objekt).

Speichern Sie das resultierende Objekt als "ex0303b". Die erstellte Tabelle "ex0303b" soll aus zwei Spalten "LIFNR" und "NETWR" bestehen.

```
ex0303b <- NULL
```

Ihr Code:

Tests: (3.b) `ex0303b` ist ein Dataframe mit korrekten Dimensionen, Spaltennamen, Summen- und Häufigkeitswerten in den numerischen Spalten. (15 P.)

```
test <- "3.2"; source("evaluate_test.R")
```

Aufgabe 4: Runner (45 P.)

Sie implementieren das Spiel "Runner". Sie spielen das Spiel mit einem Kartendeck aus vier Farben:

- "Karo" / "Diamonds"
- "Herz" / "Hearts"
- "Pik" / "Spades"
- "Kreuz" / "Clubs")

Das Kartendeck enthält für jede Farbe jeweils Karten mit Zahlen von 1 bis 15.

Das Spiel hat folgende Regeln:

- Das Spiel kann mit einem Geber ("Dealer") und einem bis fünf Spielern gespielt werden.
- Das Spiel beginnt, wenn der Geber eine Karte vom Stapel zieht.
- Er fragt den ersten Spieler, ob die aktuelle Karte eine höhere, niedrigere oder gleiche Zahl als die vorherige Karte hat.
- Der Spieler erhält 1 Punkt, wenn er richtig liegt, und 1 Minuspunkt, wenn er falsch liegt.
- Die Karte, die der vorherige Spieler erraten musste, ist nun die Karte, die der nächste Spieler erraten muss.
- Das Spiel geht weiter, bis keine Karte mehr im Stapel ist.

Teilaufgabe A (5 P.)

Erstellen Sie einen `data.frame`, das als Datenbasis dient, um alle 60 Karten im Deck anzuzeigen, so dass jede Karte eindeutig identifiziert werden kann. Nennen Sie das `data.frame` "ex04adeck". Nennen Sie die Spalten "colour" und "number".

```
ex04adeck <- NULL
```

Ihr Code:

Tests: (4.a) `ex04adeck` ist ein Dataframe mit korrekten Dimensionen, Spaltennamen, Summen- und Häufigkeitswerten in den numerischen Spalten. (5 P.)

```
test <- "4.1"; source("evaluate_test.R")
```

Teilaufgabe B (5 P.)

Dies ist Ihr Starter Deck. Aus technischen Gründen müssen wir auch die aktuelle Karte und die Historie der Karten festhalten. Dieser Slot wird zunächst aber leer bleiben. Erstellen Sie eine Liste namens "ex04bdeck", die diese Einträge mit den Namen "card", "deck" und "history" enthält. Da wir noch nicht gespielt haben, sollten "card" und "history" leer sein.

```
ex04bdeck <- NULL
```

Ihr Code:

Tests: (4.b) `ex04bdeck` ist eine Liste mit den Einträgen `card`, `history` und `deck` wobei `deck` ein Dataframe ist mit korrekten Dimensionen, Spaltennamen, Summen- und Häufigkeitswerten in den numerischen Spalten. (5 P.)

```
test <- "4.2"; source("evaluate_test.R")
```

Teilaufgabe C (20 P.)

Implementieren Sie ein Ziehen aus dem Deck als eine `R` Funktion.

Ihre Funktion sollte eine benannte Liste zurückgeben, bei der der erste Eintrag die Reihe ist, die aus dem Deck gezogen wurde, und der zweite Eintrag das verbleibende Deck ohne die gezogene Karte ist.

Der Verlauf (history) entspricht allen Karten, die aus dem Deck gezogen wurden, in chronologischer (absteigender) Reihenfolge.

Alle Einträge sollten, wenn nicht leer, den gleichen Typ/die gleiche Klasse haben. Die Einträge sollten "card" und "deck", "history" genannt werden. Nennen Sie die Funktion `ex04draw`.

Sie benutzt das "Deck" als Eingabe und hat ein optionales Seed argument. Die Eingabe "deck" sollte das gleiche Format haben wie die Ausgabe.

Stellen Sie sicher, dass der Seed nur gesetzt wird, wenn er angegeben wird, so dass die Funktion standardmäßig zufällig arbeitet. Verwenden Sie die bereitgestellte Signatur.

Eingabe

- `deck`: Eine Liste, die den Anforderungen für ein Deck aus b) entspricht.
- `seed`: Numerisches Skalar. Der optionale Seed für das Generieren von Zufallszahlen.

Ausgabe Ein `deck`, mit einer Karte weniger in `$deck` im Vergleich zum Input. Die entfernte Karte wird in `$card` als ein `data.frame` mit einer Zeile dargestellt.

```
example_deck <- list(
  card = NULL,
  deck = data.frame(colour = c("black", "orange", "white", "black"),
                    number = c(8, 9, 1, 3)),
  history = NULL)
ex04draw(example_deck, seed = 8L)
```

Beispiele

```
## $card
##   colour number
## 4  black      3
##
## $deck
##   colour number
## 1  black      8
## 2 orange      9
## 3  white      1
##
## $history
## NULL
```

```
example_deck <- list(
  card = NULL,
  deck = data.frame(colour = c("black", "orange", "white", "black"),
                    number = c(8, 9, 1, 3),
                    irrelevant = c("A", "A", "C", "D")),
```

```
  history = NULL)
ex04draw(example_deck, seed = 8L)
```

```
## $card
##   colour number irrelevant
## 4  black      3          D
##
## $deck
##   colour number irrelevant
## 1  black      8          A
## 2 orange      9          A
## 3  white      1          C
##
## $history
## NULL
```

```
example_deck <- list(
  card = data.frame(colour = "green", number = 12),
  deck = data.frame(colour = c("black", "orange", "white", "black"),
                    number = c(8, 9, 1, 3)),
  history = NULL)
ex04draw(example_deck, seed = 8L)
```

```
## $card
##   colour number
## 4  black      3
##
## $deck
##   colour number
## 1  black      8
## 2 orange      9
## 3  white      1
##
## $history
##   colour number
## 1  green     12
```

```
ex04draw <- function(deck, seed = NULL) {

  #TODO

}
```

Ihr Code:

Tests: (4.c.1) Ihre Funktion `ex04draw` zieht die Karte korrekt, aktualisiert das verbleibende Deck richtig und belässt den Verlauf (History) als NULL, wenn das Deck frisch ist und keine vorherige Karte gezogen wurde. (5 P.)

```
test <- "4.31"; source("evaluate_test.R")
```

(4.c.2) Ihre Funktion `ex04draw` zieht die Karte korrekt, aktualisiert das verbleibende Deck richtig und verschiebt die vorherige Karte aus card in den Verlauf (History), wenn bereits eine Karte gezogen wurde. (5 P.)

```
test <- "4.32"; source("evaluate_test.R")
```

(4.c.3) Ihre Funktion `ex04draw` zieht die Karte korrekt, aktualisiert das verbleibende Deck richtig, ignoriert irrelevante Spalten und belässt den Verlauf (History) als NULL, wenn keine vorherige Karte gezogen wurde. (5 P.)

```
test <- "4.33"; source("evaluate_test.R")
```

(4.c.4) Ihre Funktion `ex04draw` zieht die Karte zufällig und produziert reproduzierbare Ergebnisse, wenn ein Seed angegeben wird. (5 P.)

```
test <- "4.34"; source("evaluate_test.R")
```

Teilaufgabe D (15 P.)

Implementieren Sie die Entscheidung eines spielenden Individuums, das seine Entscheidung auf die vorherige (vor Beginn des Spiels) Verteilung der Zahlen stützt, d.h. das Individuum wird immer "größer" spielen, wenn eine Zahl kleiner als 8 erscheint und "kleiner", wenn sie größer ist. Im Falle von 8 ist die Person unentschieden und trifft eine zufällige Entscheidung zwischen den gleich wahrscheinlichen Ereignissen (aus ihrer Sicht).

Die Funktion nimmt die `card` (die Karte, über die entschieden werden soll) und das `deck` (das verbleibende Deck zum Zeitpunkt der Entscheidung) als Eingaben.

Nennen Sie die Funktion `ex04decide_prior`.

Die Funktion gibt einen skalaren Zeichenwert aus: "larger", "equal" oder smaller" (die getroffene Entscheidung). Verwenden Sie die bereitgestellte Signatur.

Eingabe

- `card`: Ein `data.frame` mit einer Zeile und (mindestens) den zwei Spalten
- `colour` und `number`
- `deck`: Ein `data.frame` mit beliebig vielen Zeilen und (mindestens) den zwei Spalten `colour` und `number`

Ausgabe Ein skalarer Character: entweder "smaller" oder "larger" gemäß der Angabe.

```
example_deck <- list(
 card = data.frame(colour = "green", number = 12),
 deck = data.frame(colour = c("black", "orange", "white", "black"),
                   number = c(8, 9, 1, 3)),
 history = NULL)
ex04decide_prior(card = example_deck$card, deck = example_deck$deck)
```

Beispiele

```
## [1] "smaller"
```

```
example_deck <- list(
 card = data.frame(colour = "green", number = 4),
 deck = data.frame(colour = c("black", "orange", "white", "black"),
                   number = c(8, 9, 1, 3)),
 history = NULL)
ex04decide_prior(card = example_deck$card, deck = example_deck$deck)
```

```
## [1] "larger"
```

```r
ex04decide_prior <- function(card, deck) {

  #TODO

}
```

Ihr Code:

Tests: (4.d.1) Ihre Funktion `ex04decide_prior` trifft basierend auf der vorgegebenen Verteilung der Zahlen die korrekte Entscheidung, ob die gezogene Karte "smaller" oder "larger" ist, oder trifft bei einer Zahl von 8 eine zufällige Entscheidung. (5 P.)

```r
test <- "4.41"; source("evaluate_test.R")
```

(4.d.2) Ihre Funktion `ex04decide_prior` ignoriert irrelevante Eingaben und trifft basierend auf der vorgegebenen Verteilung der Zahlen die korrekte Entscheidung, ob die gezogene Karte "smaller" oder "larger" ist, oder trifft bei einer Zahl von 8 eine zufällige Entscheidung. (5 P.)

```r
test <- "4.42"; source("evaluate_test.R")
```

(4.d.3) Ihre Funktion `ex04decide_prior` trifft auch bei zusätzlichen Tests die korrekten Entscheidungen ("smaller", "larger") basierend auf der vorgegebenen Verteilung der Zahlen und entscheidet zufällig bei einer Zahl von 8. (5 P.)

```r
test <- "4.43"; source("evaluate_test.R")
```

Klausur einreichen (Wichtig!)

Reichen Sie Ihren Code ein, indem Sie Ihre Ergebnisse über Git pushen.

1) Öffnen Sie das Fenster `Git`.
2) Klicken Sie auf den Button `Commit`.
3) Setzen Sie ein Kreuz bei dem Dokument `exam.Rmd`.
4) Geben Sie eine beliebige Commit Nachricht ein.
5) Drücken Sie auf `Commit`.
6) Drücken Sie auf `Push`.

Alle Tests laufen lassen

```r
EXERCISES <- character(0); source("evaluate_submission.R")
```