Inhaltsverzeichnis

1 Verteilungen

Sie haben verschiedene Verteilungen in der Vorlesung kennengelernt. Im Folgenden soll die Verwendung der Normalverteilung in R gezeigt und vertieft werden.

StatsReminder

Ein fundamentales Konzept der Wahrscheinlichkeitstheorie ist die Wahrscheinlichkeitsverteilung (ab jetzt verkürzt: Verteilung). Eine Verteilung ist das theoretische Gegenstück zu einer relativen Häufigkeitsverteilung. Die relative Häufigkeitsverteilung stellt dar, wie **oft ein Ereignis eingetreten ist**. Die (Wahrscheinlichkeits-)Verteilung stellt dar, wie **oft ein Ereignis basierend auf theoretischen Erwartungen hätte eintreten sollen**.

1.1 Stetige Zufallsvariablen

Eine Zufallsvariable wird dann als stetig bezeichnet, wenn sie überabzählbar unendlich viele Werte annimmt. In der Psychologie wäre das beispielsweise der IQ einer zufällig ausgewählten Person. Beliebte Beispiele sind auch das Körpergewicht oder die Temperatur. Die Wahrscheinlichkeitsverteilung beschreibt man im Kontext von stetigen Zufallsvariablen mithilfe der Dichtefunktion und der kumulierten Verteilungsfunktion.

StatsReminder

Die Normalverteilung (auch Gauß-Verteilung) wird Ihnen im Laufe Ihres Studiums immer wieder begegnen. Die Formel der Dichtefunktion lautet:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Die Normalverteilung besitzt einige nützliche Eigenschaften:

- Glockenförmig (eingipflig)
- Symmetrisch
- Durch zwei Parameter definiert:
  - Erwartungswert μ
  - Standardabweichung σ bzw. Varianz σ^2
- Kurz-Schreibweise $X \sim \mathcal{N}(\mu, \sigma^2)$
- Die Wendepunkte befinden sich an den Stellen $\mu \pm \sigma$
- Eine Normalverteilung mit $\mu = 0$ und $\sigma = 1$ nennt man Standardnormalverteilung oder z-Verteilung

Da diese Verteilung durch ihre Dichte- bzw. Verteilungsfunktion eindeutig beschrieben ist, kann sie in R ohne jegliche beobachtete Werte aufgerufen werden.

Das Wissen (bzw. die Annahme) der Normalverteilung von Variablen/Merkmalen ist sehr nützlich für die Bestimmung von Wahrscheinlichkeiten.

1.2 Verteilungsfunktionen in R

- Es gibt vier Funktionen in R, die sich auf die Normalverteilung beziehen: `dnorm()`, `pnorm()`, `qnorm()` und `rnorm()`.
- Alle haben die Argumente `mean` und `sd`, mit deren Hilfe man Erwartungswert und Standardabweichung angeben kann.
- Die Standardeinstellung ist `mean = 0` und `sd = 1`, so dass man sich ohne Angabe dieser Argumente immer auf die Standardnormalverteilung bezieht.
- Beim Arbeiten mit der z-Tabelle mussten bisher anders normalverteilte Werte zunächst z-transformiert werden. Beim Arbeiten mit den R Funktionen kann man entweder dies tun und dann die Standardeinstellungen der Funktionen unangetastet lassen. Einfacher ist es, Mittelwert und Standardabweichung der entsprechenden Verteilung über die Argumente in der jeweiligen Funktion anzugeben.
- Achtung Fehlerquelle: Die Kurz-Schreibweise $X \sim \mathcal{N}(\mu, \sigma^2)$ verwendet σ^2, die R-Funktionen benötigen aber σ!
- Der Name der Funktion bezieht sich darauf, was gesucht ist, und das erste Argument darauf, was gegeben ist. Dies werden wir im Folgenden mit den Fragen des Anwendungsbeispiels illustrieren.

1.3 Anwendungsbeispiel

Eine zufällig bestimmte Person absolviert einen *IQ*-Test. Die Testwerte sind normalverteilt mit dem Erwartungswert $\mu = 100$ und der Standardabweichung $\sigma = 15$.

Man schreibt auch $IQ \sim \mathcal{N}(100, 225)$.

Folgende Fragen möchten wir heute beantworten:

1. Wie können wir mithilfe von R die Dichtefunktion dieser Verteilung zeichnen?
2. Wie wahrscheinlich ist es, dass eine zufällig gezogene Person einen *IQ* von höchstens 110 erzielt?

3. Welchen Wert müsste eine Person *mindestens* erzielen, um zu den 2,5 % der Population mit den höchsten Werten zu gehören?
4. In welchem Bereich liegen die *IQ*-Werte der mittleren 90 % der Bevölkerung?

1.4 Dichte

- Gesucht ist die Dichte (engl., *density*) an einem bestimmten gegebenen Punkt x der Verteilung.
- Die passende Funktion ist `dnorm(x)`.

```
dnorm(x = 110,              # Bestimmter Wert der normalverteilten Variable
      mean = 100, sd = 15)  # Parameter der IQ-Verteilung
```

```
[1] 0.02129653
```

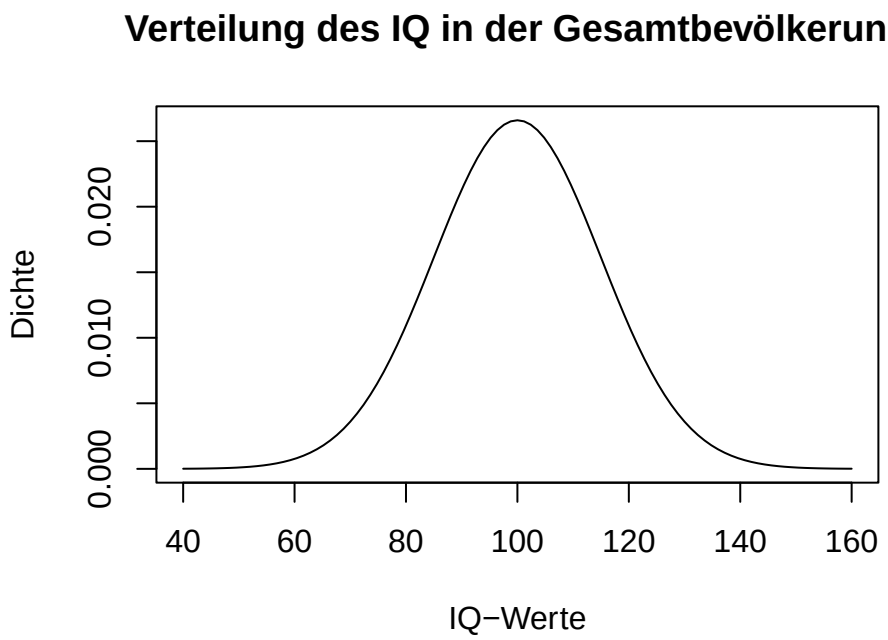
Der konkrete Wert der Dichte ist bei stetigen Verteilungen für uns selten von Interesse. Die Dichte dient v. a. zur graphischen Darstellung, da für eine Normalverteilungskurve der Wert auf der Y-Achse die Dichte ist.

Am einfachsten ist die Erstellung eine Dichtekurve mit der Funktion `curve()`. Bei ihrer Verwendung muss kein konkreter x-Wert vorgegeben werden, da die Funktion für alle möglichen x-Werte evaluiert wird. Daher bleibt für die zu plottende Funktion einfach der Platzhalter `x` stehen.

Wichtig wird dann aber die Angabe eines sinnvoll abzubildenden Bereichs der X-Achse über das bereits bekannte Argument `xlim`, für unser Beispiel $\mu \pm 4\sigma$.

Durch die Schachtelung von `dnorm()` in `curve()` muss auf die Zuordnung der Argumente geachtet werden: `mean` und `sd` gehören zu `dnorm()`, alle graphischen Argumente zu `curve()`.

```
curve(dnorm(x,                      # Platzhalter für alle möglichen Werte
            mean = 100, sd = 15),   # Parameter der IQ-Verteilung
      xlim = c(40, 160),            # 100- 4*15 und 100+ 4*15
      ylab = "Dichte",
      xlab = "IQ-Werte",
      main = "Verteilung des IQ in der Gesamtbevölkerung")
```



1.5 Kumulative Wahrscheinlichkeiten

- Gesucht ist die kumulative Wahrscheinlichkeit (engl., *probability*) bis zu einem konkret gegebenen Quantil q.
- Die passende Funktion ist `pnorm(q)`.
- Im Gegensatz zu `dnorm(x)` gibt es ein zusätzliches Argument mit Standardeinstellung, nämlich `lower.tail = TRUE`.
- Der *tail* ist das Verteilungsende, von dem aus kumuliert wird, im Standardfall das untere Ende, also wird ausgegeben $P(-\infty < X \leq x_i) = P(X \leq x_i)$.
- Das x in der Formel ist in der Funktion das Argument `q`.
- Bezogen auf das Anwendungsbeispiel, gesucht: $P(IQ \leq 110)$:

```
pnorm(q = 110,                    # konkreter IQ-Wert
      mean = 100, sd = 15,        # Parameter der IQ-Verteilung
      lower.tail = TRUE)          # vom unteren Ende kumulieren (Standardeinstellung)
```

```
[1] 0.7475075
```

- Die Wahrscheinlichkeit, dass eine zufällig bestimmte Person einen IQ-Testwert von höchstens 110 erzielt, beträgt $P(IQ \leq 110) = 0,75$.
- Auch: Rund 75% der Bevölkerung haben einen IQ, der 110 oder kleiner ist.

Falls statt $P(X \leq x_i)$ einmal $P(X \geq x_i)$ gesucht sein sollte, gibt es zwei einfache Möglichkeiten, diese Wahrscheinlichkeit zu berechnen:

(a) Berechnung als Gegenwahrscheinlichkeit

```
1 - pnorm(q = 110, mean = 100, sd = 15, lower.tail = TRUE)
```

```
[1] 0.2524925
```

(b) Anpassung des Zusatzarguments:

```
pnorm(q = 110, mean = 100, sd = 15, lower.tail = FALSE)
```

```
[1] 0.2524925
```

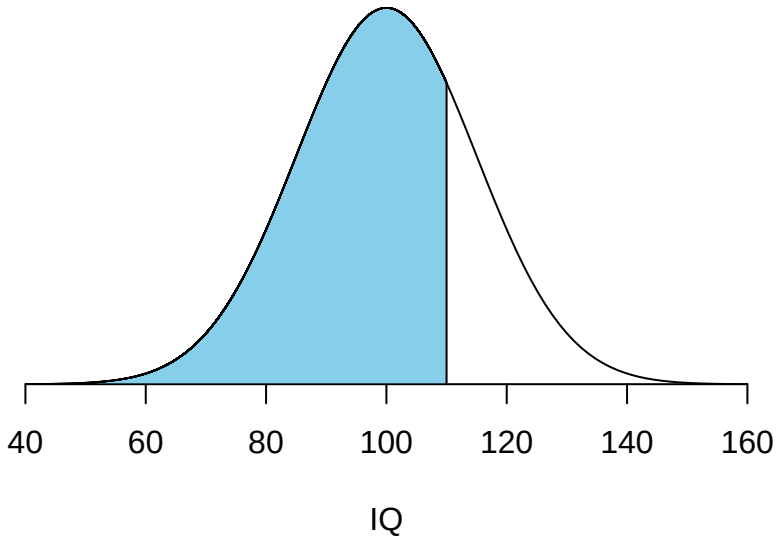
1.5.1 Veranschaulichung mit `shadeNV()`

- Um die hier verwendete Funktion `shadeNV()` verwenden zu können, muss einmal der Befehl im Skript **shadeNV.R** ausgeführt werden. Das geht durch normales Öffnen und Ausführen oder — wenn die Datei im Arbeitsverzeichnis liegt — über folgenden Befehl:

```
source(file = "functions/shadeNV.R")
```

- Nun steht in der Umgebung (rechts oben in RStudio) die Funktion zur Verfügung.

```
shadeNV(M = 100, SD = 15,    # Verteilungskennwerte
        ub = 110,            # Obergrenze (upper bound)
        shade = "lower",     # Bereich DARUNTER einfärben
        xlab = "IQ")         # Beschriftung X-Achse
```




```
[1] 0.7475075
```

1.6 Bestimmung von Quantilen

- Falls andersherum die Wahrscheinlichkeit p gegeben und das Quantil q gesucht ist, ist die passende Funktion `qnorm(p)`.
- Auch hier gibt es das Argument `lower.tail = TRUE`.
- Übertragung auf das Anwendungsbeispiel c) Welchen Wert müsste die Person *mindestens* erzielen, um zu den 2,5% der Population mit den höchsten Werten zu gehören?
- Gesucht ist x_i, für das gilt: $P(X \geq x_i) = 0,025$ bezogen auf die Verteilung des IQ.
- Da sich die angegebene Wahrscheinlichkeit auf den rechten Rand der Verteilung bezieht, setzen wir `lower.tail = FALSE`:

```
qnorm(p = .025,             # Wahrscheinlichkeit, die am OBEREN Ende abgeschnitten wird
      mean = 100, sd = 15,  # Parameter der IQ-Verteilung
      lower.tail = FALSE)   # p vom OBEREN Ende
```

```
[1] 129.3995
```

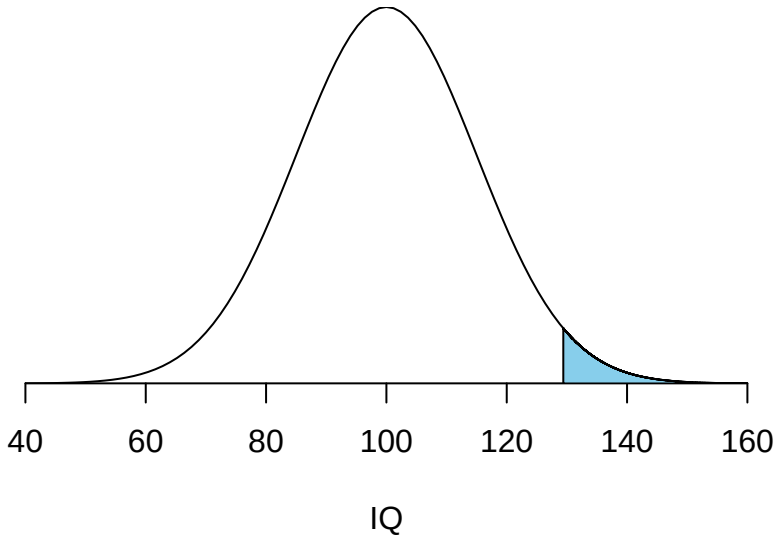
- Alternativ hätten wir die Wahrscheinlichkeit umformulieren können, denn wenn $P(X \geq x_i) = 0,025$, dann $P(X \leq x_i) = 0,975$, gesucht ist also das 97,5% Quantil.

```
qnorm(p = .975,             # Wahrscheinlichkeit, die am UNTEREN Ende abgeschnitten wird
      mean = 100, sd = 15,  # Parameter der IQ-Verteilung
      lower.tail = TRUE)    # p vom UNTEREN Ende
```

```
[1] 129.3995
```

- Die Person müsste mindestens einen IQ-Testwert von 129,4 erzielen, um zu den 2.5% der Bevölkerung mit den höchsten Werten zu gehören (Hochbegabung ist definiert als $IQ \geq 130$, also $\mu + 2\sigma$).
- Darstellung:

```
shadeNV(M = 100, SD = 15,    # Verteilungskennwerte
        lb = 129.4,          # Untergrenze (lower bound)
        shade = "upper",     # Bereich DARÜBER einfärben
        xlab = "IQ")         # Beschriftung X-Achse
```




```
[1] 0.0249979
```

1.7 Zentrale Schwankungsintervalle

- Die Bestimmung von Quantilen kann auch zur Bestimmung der Grenzen von zentralen Schwankungsintervallen dienen.
- Die Intervallgrenzen liegen symmetrisch um den Erwartungswert und schließen einen bestimmten Flächenanteil ein bzw. aus.
- Bezogen auf das Anwendungsbeispiel d) In welchem Bereich liegen die IQ-Werte der mittleren 90% der Bevölkerung?
- Gesucht sind die Quantile IQ_{p1} und IQ_{p2}, für die gilt: $F(IQ_{p2}) - F(IQ_{p1}) = 0,90$.
- Dies ist das zentrale 90% Schwankungsintervall.
- Vorteil Standardnormalverteilung: Da der Mittelwert 0 ist, reicht es aus, einen der Werte zu bestimmen und für den zweiten Wert das Vorzeichen zu ändern: $F(z_p) - F(-z_p) = 0,90$.

Bestimmung der Werte für das zentrale 90% Schwankungsintervall der IQ-Verteilung:

1. Flächenanteil bestimmen, der auf jeder Seite der Verteilung abgeschnitten werden soll, da $P(IQ \leq IQ_{p1}) = P(IQ \geq IQ_{p2})$ (zur Not im Kopf):

```
(1 - .9) / 2
```

```
[1] 0.05
```

2. Bestimmung der beiden IQ-Werte, die links bzw. rechts von sich jeweils 5% der Verteilung abschneiden:

```
qnorm(p = .05, mean = 100, sd = 15)
```

```
[1] 75.3272
```

```
qnorm(p = .95, mean = 100, sd = 15) # oder
```

```
[1] 124.6728
```

```
qnorm(p = .05, mean = 100, sd = 15, lower.tail = FALSE)
```

```
[1] 124.6728
```

Die mittleren 90% der Bevölkerung liegen im Bereich zwischen den IQ-Werten 75 und 125.

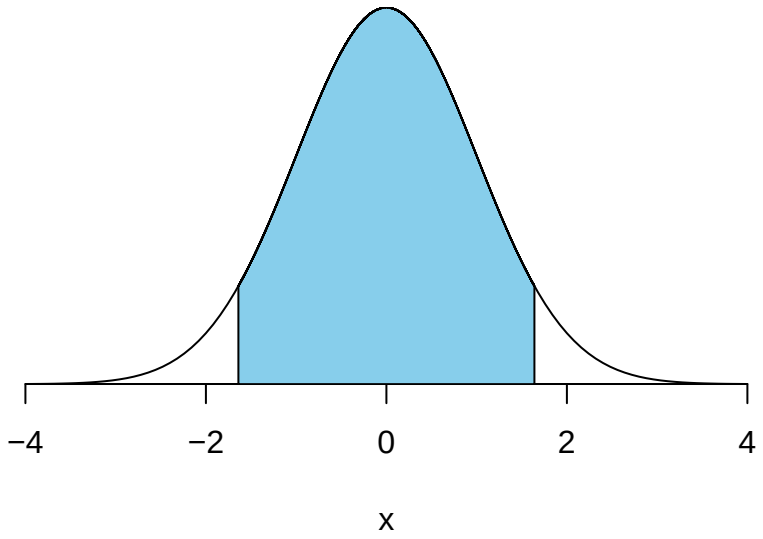
Für die Standardnormalverteilung $-z_p = -1,64$ und $z_p = 1,64$:

```
qnorm(p = .05)
```

```
[1] -1.644854
```

- Darstellung (bezogen auf Standardnormalverteilung)

```
shadeNV(Z = 1.64, shade = "inner")
```




```
[1] 0.8989948
```

1.7.1 Weitere zentrale Schwankungsintervalle

- Welches sind die Grenzwerte des zentralen Schwankungsintervalls der Standardnormalverteilung, das die mittleren 95% der Werte umfasst?

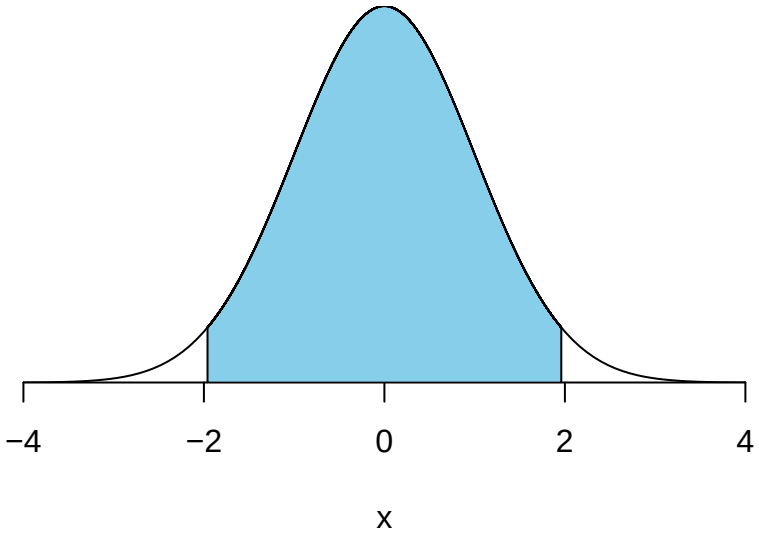
```
qnorm(.025)
```

```
[1] -1.959964
```

```
qnorm(.975)
```

```
[1] 1.959964
```

```
shadeNV(Z=1.96, shade = "inner")
```




```
[1] 0.9500042
```

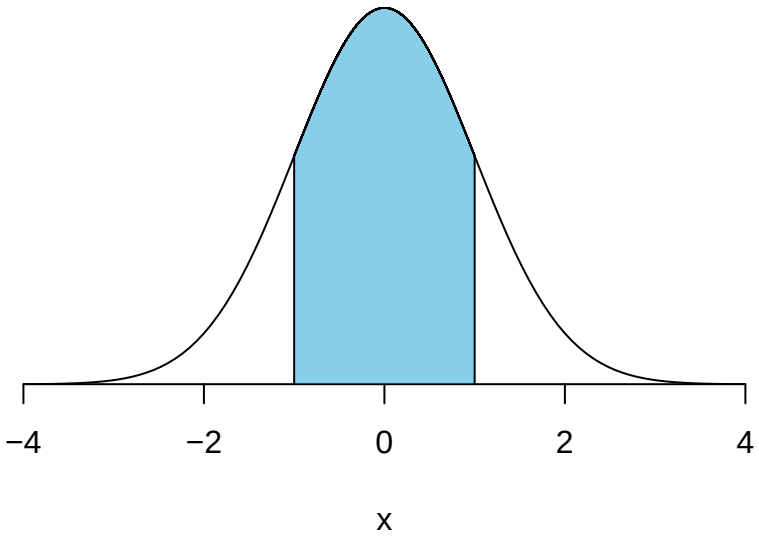
Die mittleren (wahrscheinlichsten) 95% der z-Werte liegen im Bereich zwischen -1,96 und +1,96.

- Wie groß ist der Teil der Werte, die bei der Standardnormalverteilung im Bereich zwischen -1 und 1 liegen?

```
pnorm(1) - pnorm(-1)
```

```
[1] 0.6826895
```

```
shadeNV(Z = 1, shade = "inner")
```




```
[1] 0.6826895
```

Rund 68% der z-Werte liegen im Bereich zwischen -1 und 1. Bezogen auf das IQ-Beispiel zwischen 85 und 115, da die Standardabweichung 15 beträgt.

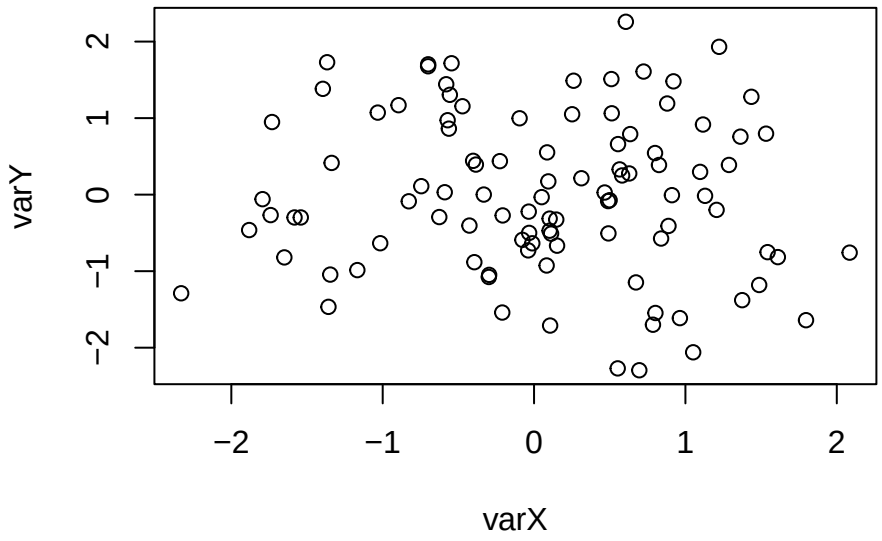
1.8 Generierung von Zufallszahlen

- Die vierte Funktion, die sich auf die Normalverteilung bezieht, ist `rnorm(n)`.
- Sie erzeugt n zufällige (engl., *random*) Werte aus einer Normalverteilung.
- Auch hier können die Parameter `mean` und `sd` angegeben werden.
- Diese Funktion ist manchmal nützlich, wenn keine Daten vorhanden sind, man aber Werte für Funktionen benötigt:

```
varX <- rnorm(n = 100)
varY <- rnorm(n = 100)
cor(varX, varY)
```

```
[1] -0.07154409
```

```
plot(varY ~ varX)
```



1.9 Zusammenfassung

Normalverteilung	Bedeutung
`dnorm(x, mean, sd)`	**d** steht für *density* und bezieht sich auf die Wahrscheinlichkeitsverteilung
`pnorm(q, mean, sd, lower.tail)`	**p** steht für *probability* und bezieht sich auf die Verteilungsfunktion
`qnorm(p, mean, sd, lower.tail)`	**q** steht für *quantile* und bezieht sich auf die Quantilsfunktion
`rnorm(n, mean, sd)`	**r** steht für *random* bezieht sich auf zufällige Ziehen von Werten aus einer Normalverteilung

Das Argument `x` der Funktion `dnorm()` nimmt eine Zahl oder einen Vektor aus Zahlen entgegen. Für jede Zahl in `x` wird eine Wahrscheinlichkeit berechnet.
Das Argument `q` der Funktion `pnorm()` nimmt eine Zahl oder einen Vektor Vektor aus Zahlen entgegen. Für jedes Quantil in `q` wird die kumulative Wahrscheinlichkeit berechnet.
Das Argument `p` der Funktion `qnorm` nimmt eine Wahrscheinlichkeit oder einen Vektor aus Wahrscheinlichkeiten entgegen. Für jede Wahrscheinlichkeit in `p` wird das Quantil berechnet.
Das Argument `n` der Funktion 'rnorm()" gibt einen Vektor aus normalverteilten Zufallswerten zurück.

1.10 Weitere Verteilungen

Dasselbe Set an Funktionen gibt es für viele andere Verteilungen. Im Folgenden eine Auswahl der Kürzel, die nach `d` (Dichte), `p` (Wahrscheinlichkeit), `q` (Quantile) und `r` (Zufallszahlen) gesetzt werden müssen. Die notwendigen Parameter unterscheiden sich dabei je nach Verteilungsfamilie.

Verteilungsfamilie	Funktions-Suffix
Gleichverteilung	`unif`

Verteilungsfamilie	Funktions-Suffix
Exponential	`exp`
Binomial	`binom`
Logistisch	`logis`
χ^2	`chisq`
t	`t`
F	`f`
Poisson	`pois`

Für eine Liste aller Verteilungen, die in R implementiert sind, kann man mit dem folgenden Befehl eine entsprechende Hilfeseite aufrufen:

```
?distributions
```

2 Diskrete Verteilungen

2.1 Diskrete Zufallsvariablen

Eine diskrete Zufallsvariable ist eine Abbildung der Ergebnisse im Ergebnisraum Ω auf Zahlen (meistens die reellen Zahlen $\mathbb{R}$). Diskret beschreibt dabei, dass die Zufallsvariable (im Gegensatz zur stetigen Zufallsvariable) nur endlich viele oder abzählbar unendlich viele Werte annehmen kann. Ein Beispiel für eine endliche Wertemenge ist die Anzahl defekter Schrauben aus einer Stichprobe aus Schrauben. Eine abzählbar unendliche Wertemenge wäre beispielsweise die Anzahl der Würfe einer Münze bis zum ersten Mal Zahl erscheint. In welcher *Form* diskrete Zufallsvariablen Werte einnehmen, wird durch Verteilungen beschrieben.

2.2 Bernoulliverteilung

StatsReminder

Die Bernoulliverteilung beschreibt die Wahrscheinlichkeit von Ereignissen in Versuchen mit nur zwei mögliche Versuchsausgängen, z. B. die Wahrscheinlichkeit Zahl oder Kopf zu werfen beim einmaligen Münzwurf. Die Formel einer Bernoulliverteilung sieht wie folgt aus:

$$P(X = x) = \pi^x (1 - \pi)^{(1-x)}$$

Im folgenden dient der Münzwurf als Beispiel.

Die obige Formel ist durch die Funktion `dbinom()` in R implementiert. Mit dieser kann man die Wahrscheinlichkeit für das als *Erfolg* bezeichnete Ereignis berechnen.

```
dbinom(x = 1, size = 1, prob = 0.5)
```

```
[1] 0.5
```

- `size` ist die Anzahl der Münzwürfe
  - falls `size` nicht auf 1 gesetzt wird, sprechen wir nicht die Bernoulli-verteilte Zufallsvariable an, sondern eine binomialverteilte Variable. (die Münze soll/wird nur einmal geworfen)
- `x = 1` beschreibt das für uns interessante Ereignis (beispielsweise: Kopf)
  - in unserem Beispiel wäre also $\{Kopf\} = 1$ und $\{Zahl\} = 0$
- `prob = 0.5` ist die Erfolgswahrscheinlichkeit, welche hier auf 50% gesetzt wurde (d.h., die Münze ist fair)

Der Output von `dbinom` ist 0.5. Die Wahrscheinlichkeit das die Münze auf Kopf landet ist also 0.5. In mathematischer Notation:

$$P(\{\text{Kopf}\}) = P(X = 1) = 0.5$$

Die Berechnung der Wahrscheinlichkeit, dass die Münze auf Zahl fällt, funktioniert nahezu gleich:

```
dbinom(x = 0, size = 1, prob = 0.5)
```

```
[1] 0.5
```

- Hier wurde lediglich das `x` Argument auf 0 gesetzt
- Auch hier erhalten wir eine Wahrscheinlichkeit von 50%, schließlich ist die Münze fair (Kopf und Zahl haben dieselbe Wahrscheinlichkeit aufzutreten)

In Abbildung 1 sind zwei Bernoulliverteilungen graphish dargestellt. Links ist die bereits behandelte Verteilung, in welcher die Münze fair ist. Rechts ist eine Bernoulliverteilung, in welcher die Münze unfair ist; das Auftreten von Kopf ist deutlich wahrscheinlicher (70%) im Vergleich zu Zahl.

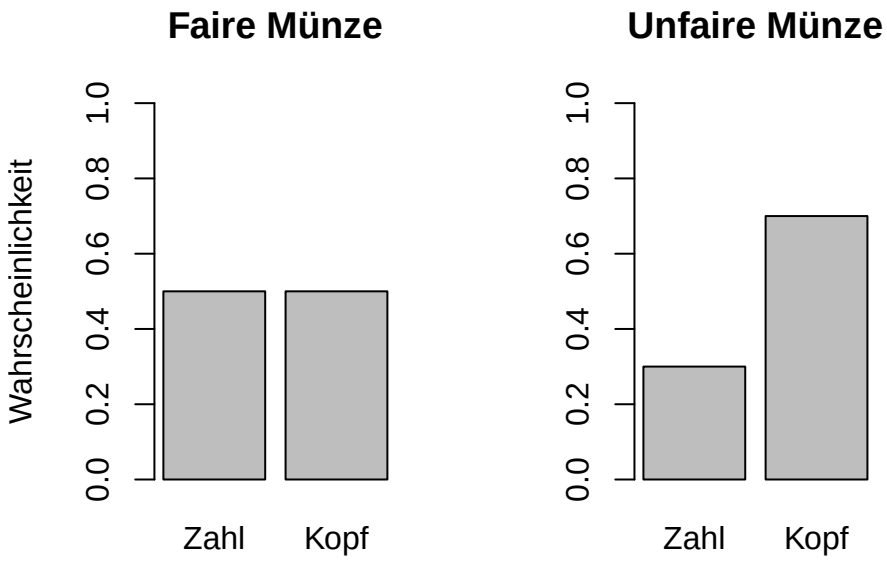


Abbildung 1: Vergleich der Wahrscheinlichkeitsverteilungen (Bernoulli) einer fairen und einer unfairen Münze.

2.3 Binomialverteilung

Die Bernoulliverteilung ist ein Spezialfall der Binomialverteilung. Im Gegensatz zur Bernoulliverteilung kann man mit der Binomialverteilung beliebig oft die Münze werfen (≥ 1). Daher lassen sich mit der Binomialverteilung auch Fragen der Art "wie wahrscheinlich ist es 22 Köpfe zu erhalten, wenn die Münze 30 mal geworfen wird?" beantworten. Da die Bernoulliverteilung ja ein Spezialfall der Binomialverteilung ist, wird für die Binomialverteilung auch `dbinom()` verwendet.

```
dbinom(x = 22, size = 30, prob = 0.5)
```

```
[1] 0.005450961
```

- Mit `x = 22` und `size = 30` wird die obige Frage beantwortet
  - hier wird wieder von einer fairen Münze ausgegangen (`prob = 0.5`)

Genau wie für die Wahrscheinlichkeitsverteilung gibt es in R auch eine Funktion für die Verteilungsfunktion (kumulierte Wahrscheinlichkeit). Für die Binomialverteilung heißt diese `pbinom()`.

```
pbinom(q = 22, size = 30, prob = 0.5)
```

```
[1] 0.9973886
```

- Übersetzt: Wie hoch ist die Wahrscheinlichkeit, dass die Münze 22 mal **oder weniger** auf Kopf landet (bei 30 Würfen der Münze)
- Anstatt `x` wird nun `q` übergeben
  - `q` steht hier für das Quantil
- `pbinom()` summiert die Wahrscheinlichkeiten für jeden Wert von 0 bis einschließlich 22 (geschlossenes Intervall, $[0, 22]$) auf. Sie könnten also auch jeden einzelnen Wert `dbinom()` übergeben und dann die einzelnen Wahrscheinlichkeiten addieren

Beispiel:

```
# kumulierte Wahrscheinlichkeit mit dbinom
sum(
  dbinom(x = 0, size = 3, prob = 0.5),
  dbinom(x = 1, size = 3, prob = 0.5),
  dbinom(x = 2, size = 3, prob = 0.5)
)
```

```
[1] 0.875
```

```
# kumulierte Wahrscheinlichkeit mit pbinom
pbinom(q = 2, size = 3, prob = 0.5)
```

```
[1] 0.875
```

- Hier wird die Frage beantwortet wie wahrscheinlich es ist, dass die Münze bei 3 Würfen 2 mal, 1 mal oder gar nicht auf Kopf landet
- Man kann sehen, dass das Ergebnis von `pbinom()` lediglich die Summe der Einzelwahrscheinlichkeiten darstellt

Übrigens: `dbinom()` kann auch Vektoren für die Anzahl der Treffer `x` verarbeiten.

```
dbinom(x = c(0, 1, 2), size = 3, prob = 0.5)
```

```
[1] 0.125 0.375 0.375
```

- Hier wird `x` ein Vektor aus Zahlen übergeben
  - `dbinom()` berechnet dann die Wahrscheinlichkeit jedes Elements in `x` und gibt einen Vektor aus Wahrscheinlichkeiten zurück
  - die erste Wahrscheinlichkeit `0.125` ist das Resultat des ersten Elements des `x`-Vektors, also `0` usw.

Mit `pbinom()` kann man somit nicht nur die Wahrscheinlichkeit der Form $P(X \leq x)$ berechnen, sondern auch die entsprechende Gegenwahrscheinlichkeit $1 - P(X \leq x) = P(X > x)$. Übersetzt auf die obige Fragestellung würde das bedeuten, dass wir danach fragen, wie wahrscheinlich es ist, genau 3 Köpfe zu werfen (bei 3 Würfen).

```
1 - pbinom(q = 2, size = 3, prob = 0.5)
```

```
[1] 0.125
```

Alternativ:

```
pbinom(q = 2, size = 3, prob = 0.5, lower.tail = FALSE)
```

```
[1] 0.125
```

- Mit `lower.tail = FALSE` wird die Wahrscheinlichkeit $P(X > x)$ berechnet
  - der default ist `lower.tail = TRUE`, was $P(X \leq x)$ entspricht

- Achtung! Das $>$ in $P(X > x)$ ist, im Gegensatz zu stetigen Verteilungen, ausschließlich als *größer* und **nicht** als *größer gleich* zu interpretieren

Alternativ:

```
dbinom(x = 3, size = 3, prob = 0.5)
```

```
[1] 0.125
```

- Die Frage lässt sich auch einfach mit `dbinom()` beantworten

In Abbildung 2 ist die Verteilung und die Verteilungsfunktion einer Zufallsvariablen, die einer Binomialverteilung folgt, graphisch dargestellt. Versuchen Sie die Graphik zusammen mit der Beschreibung zu verstehen. Führen Sie danach die Befehle, die in der x-Achsenbeschriftung stehen, einmal selbst aus. Setzen Sie danach eigene Werte für `x` und `q` ein und versuchen Sie die Verbindung zu den beiden Graphiken herzustellen. Beispiel: Wie würden sich die grauen Einfärbungen in den Graphiken verändern, wenn Sie `pbinom(q = 3, size = 7, prob = 0.2, lower.tail = TRUE)` ausführen würden?

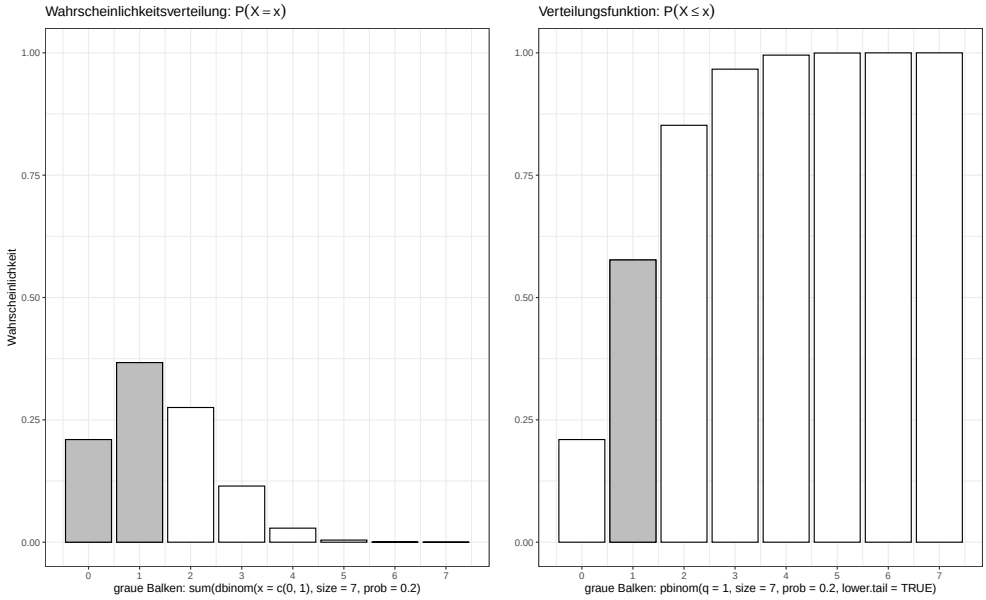


Abbildung 2: Links: Wahrscheinlichkeitsverteilung einer binomialverteilten Zufallsvariable mit einer Anzahl der Wiederholungen von 7 und einem Erwartungswert von 0.2. Rechts: Die dazugehörige Verteilungsfunktion. Die Summe der Wahrscheinlichkeiten der grau eingefärbten Balken in der Wahrscheinlichkeitsverteilung entsprechen dem grauen Balken in der Verteilungsfunktion. Unter der jeweiligen Graphik steht der R Befehl mit dem Sie die grauen Balken, also die Wahrscheinlichkeiten, berechnen

3 Übersicht

3.1 Neue wichtige Konzepte

- **Wahrscheinlichkeitsverteilung**
- **Normalverteilung**
- **Kumulative Wahrscheinlichkeitsfunktion**
- **Quantilsfunktion**
- **Schwankungsintervall**

3.1.1 Neue wichtige Befehle, Argumente, Operatoren

- SNV = Standardnormalverteilung (falls Standardeinstellungen für `mean` und `sd` verwendet werden)

Funktion	Ausgabe
`dnorm(x, mean = 0, sd = 1)`	Dichte einer SNV an der Stelle x
`pnorm(q, mean = 0, sd = 1, lower.tail = TRUE)`	Flächenanteil p unter der SNV-Kurve *links* von Wert q
`pnorm(q, mean = 0, sd = 1, lower.tail = FALSE)`	Flächenanteil p unter der SNV-Kurve *rechts* von Wert q
`qnorm(p, mean = 0, sd = 1, lower.tail = TRUE)`	Quantil q, das *links* von sich den Flächenanteil p unter SNV-Kurve abschneidet
`qnorm(p, mean = 0, sd = 1, lower.tail = FALSE)`	Quantil q, das *rechts* von sich den Flächenanteil p unter der SNV-Kurve abschneidet
`curve(function(x), xlim = c(0, 1))`	Zeichnet die von `function()` erzeugte Kurve über das vorgegebene Intervall.

3.1.2 Neue optionale Befehle, Argumente, Operatoren

Funktion	Ausgabe
`rnorm(n, mean = 0, sd = 1)`	Zieht n zufällige Werte aus einer SNV
`source()`	Führt alle Befehle einer Skriptdatei aus
`shadeNV()`	Zeichnet die NV Kurve mit eingefärbten Bereichen
`dbinom(x, size, prob)`	Wahrscheinlichkeit einer binomialverteilten Zufallsvariable an der Stelle `x`
`pbinom(q, size, prob, lower.tail = TRUE)`	Flächenanteil p unter binomialverteilten Kurve *links* von Wert q
`qbinom(p, size, prob)`	Quantil q, das *links* von sich den Flächenanteil p unter binomialverteilten Kurve abschneidet
`rbinom(n, size, prob)`	Zieht n zufällige Werte aus einer binomialverteilten Zufallsvariable

4 Appendix

4.1 Wahrscheinlichkeitstheorie in R

Ihre bisherige Statistiklaufbahn drehte sich größtenteils um deskriptive Statistik, also um das Beschreiben und Zusammenfassen von Daten. Im kommenden Semester werden Sie sich mit der Inferenzstatistik (auch schließende Statistik) befassen. Die Grundlage für die Inferenzstatistik bildet die Wahrscheinlichkeitstheorie.

4.1.1 Mengenlehre

Die Operatoren der Mengenlehre, welche Sie in der Vorlesung kennengelernt haben, findet man auch in R wieder.

```
A <- c("Frodo", "Sam", "Pippin", "Merry")
B <- c("Legolas", "Frodo", "Gandalf", "Gimli", "Merry")
```

- A und B entsprechen den Mengen
  - Menge A in Mengennotation: $A = \{\text{Frodo, Sam, Pippin, Merry}\}$
  - Menge B in Mengennotation: $B = \{\text{Legolas, Frodo, Gandalf, Gimli, Merry}\}$

```
is.element("Boromir", B)
```

```
[1] FALSE
```

- `is.element` entspricht dem $\in$-Operator
  - das obige Ergebnis in Mengennotation: Boromir $\notin B$
  - `is.element` gibt entweder `TRUE` oder `FALSE` aus. In diesem Fall bedeutet das, dass `Boromir` kein Element der Menge B ist

```
is.element("Frodo", A)
```

```
[1] TRUE
```

- `Frodo` ist Element der Menge A
  - Frodo $\in A$

```
intersect(A, B)
```

```
[1] "Frodo" "Merry"
```

- `intersect()` entspricht dem Schnittmengenoperator $\cap$
  - Die Menge {Frodo, Merry} ist die Menge aller Elemente, die sowohl in A als auch in B sind

```
union(A, B)
```

```
[1] "Frodo"   "Sam"     "Pippin"  "Merry"   "Legolas" "Gandalf" "Gimli"
```

- `union()` entspricht dem Operator der Vereinigung $\cup$
  - Die neue Menge {Frodo, Sam, Pippin, Merry, Legolas, Gandalf, Gimli} ist die Menge aller Elemente, die in beiden Mengen, A und B, auftreten

```
setdiff(A, B)
```

```
[1] "Sam"    "Pippin"
```

- `setdiff()` entspricht der Differenzmenge $\setminus$
  - für `setdiff(A, B)` spielt die Reihenfolge der Argumente eine Rolle: $A \setminus B$
  - die neue Menge {Sam, Pippin} ist die Menge aller Elemente, die in A aber nicht in B sind

```
setdiff(B, A)
```

```
[1] "Legolas" "Gandalf" "Gimli"
```

- entspricht $B \setminus A$

4.1.2 Kombinatorik

Um die Fakultät einer natürlichen Zahl k zu berechnen, kann `factorial()` genutzt werden.

```
factorial(4)
```

```
[1] 24
```

```
factorial(500)
```

```
[1] Inf
```

- Fakultäten wachsen extrem schnell. Deshalb gibt R bereits bei relativ *kleinen* Zahlen `Inf` (Unendlich) aus

Für die Berechnung des Binomialkoeffizienten gibt es in R ebenfalls eine Funktion `choose()`.

```
choose(4, 2)
```

```
[1] 6
```

- Hier wird also die Anzahl der möglichen Ergebnisse errechnet; wieviele Wege gibt es 2 Personen aus 4 Personen auszuwählen
  - das erste Argument `4` entspricht der Anzahl der Elemente der Grundgesamtheit
  - das zweite Argument `2` entspricht der Anzahl der Wiederholungen

4.1.3 `sample()`

Mit `sample()` lässt sich eine zufällig gezogene *Stichprobe* generieren.

```
sample(x = 1:10, size = 5, replace = FALSE)
```

```
[1]  1  8 10  3  2
```

- `x`; beschreibt den Vektor aus Elementen aus denen gezogen werden soll
- `size`; beschreibt die Anzahl der Elemente, die aus `x` gezogen werden sollen
- `replace`; mit `replace` kann bestimmt werden, ob ein gezogenes Objekt zurückgelegt werden soll
  - wenn `replace` auf `FALSE` gesetzt wird, kann ein Objekt nur einmal gezogen werden
  - wenn `replace` auf `TRUE` gesetzt wird, kann ein Objekt mehrmals gezogen werden

Die Funktion `sample()` ist nützlicherer, als es zunächst vermuten lässt. So kann man mit dieser Funktion bereits kleinere *Simulationen* durchführen. In einer Simulation werden die Regeln, wie es zum Ziehen aus einer Stichprobe kommt (z. B., das Ziehen von Murmeln aus einer Urne) vom Nutzer in R nachgestellt. Beispielsweise kann man das Werfen einer Münze simulieren. Stellen Sie sich vor Sie haben gerade keine Münze zur Hand. Sie wollen aber dennoch eine Münze werfen. In R kann man dieses Zufallsexperiment mit `sample()` simulieren.

```
münzwurf_ergebnisse <- c("Kopf", "Zahl")
sample(x = münzwurf_ergebnisse, size = 1)
```

```
[1] "Zahl"
```

- Hier wird ein (fairer) Münzwurf simuliert:
  - der Zeichenvektor `münzwurf_ergebnisse` ist der Ergebnisraum aus dem eine Stichprobe gezogen werden soll
  - mit `size = 1` spezifizieren wir, dass der Vorgang nur einmal stattfinden soll
  - es wurde `sample()` ein Zeichenvektor übergeben (die Funktion arbeitet aber genau gleich; es wird zufällig eines der Elemente ausgewählt)

Um den Befehl mehrmals hintereinander auszuführen, können Sie den obigen Befehl immer wieder in der Konsole ausführen. Eine praktischere Methode ist es `size` auf den bevorzugten Wert zu setzen und `replace = TRUE` zu übergeben.

```
sample(x = münzwurf_ergebnisse, size = 20, replace = TRUE)
```

```
 [1] "Zahl" "Zahl" "Kopf" "Kopf" "Zahl" "Kopf" "Zahl" "Zahl" "Zahl" "Zahl"
[11] "Zahl" "Kopf" "Kopf" "Zahl" "Kopf" "Kopf" "Zahl" "Kopf" "Zahl" "Kopf"
```

- Das entspricht dem 20-maligen Werfen einer Münze

```
sim_münzwurf <- sample(x = münzwurf_ergebnisse, size = 20, replace = TRUE)
table(sim_münzwurf)
```

```
sim_münzwurf
Kopf Zahl
  12    8
```

- Mit `table()` kann man das Resultat der simulierten Münzwürfe zusammenfassen

4.1.4 `set.seed()`

Es ist Ihnen vielleicht aufgefallen, dass `sample()` bei der erneuten Ausführung unterschiedliche Ergebnisse ausgibt. Das macht auch intuitiv Sinn, schließlich wollen wir einen Zufallsprozess simulieren (wir gehen beim Werfen der Münze auch davon aus, dass das Werfen *zufällig* passiert).
Allerdings ergibt sich hierbei ein Problem. Damit andere Studierende oder Wissenschaftler Simulationsstudien nachvollziehen und überprüfen können, müssen die berechneten Simulationen replizierbar sein (bei erneuter Ausführung muss das gleiche Ergebnis berechnet werden).
Um die Replizierbarkeit zu ermöglichen, muss der Prozess, der `sample()`[1] ermöglicht, zufällig Stichproben zu generieren, *fixiert* werden. Das lässt sich mit `set.seed()` realisieren. Mit dieser Funktion lässt sich der Zufallszahlengenerator (Englisch: RNG für *random number generator*), der für den Zufall der Ziehung verantwortlich ist, auf einen selbst gewählten Startzustand legen. Wenn bei einer zukünftigen Ausführung derselbe *Startzustand* gewählt wird (`set.seed()` auf denselben Wert gesetzt wird), dann wird die zufällig gezogene Stichprobe auf dieselbe Art und Weise generiert.

```
set.seed(123)
```

- An `set.seed()` kann eine beliebige Zahl übergeben werden
- Wenn dieser Befehl vor der eigentlichen Simulation ausgeführt wird, können die Ergebnisse repliziert werden

Beispiel:

```
set.seed(42)
```

- Zuerst setzen wir einen beliebigen *seed*
  - falls der Befehl bereits ausgeführt wurde, wird er mit dem erneuten Ausführen überschrieben

```
münzwurf_ergebnisse <- c("Kopf", "Zahl")
sample(münzwurf_ergebnisse, size = 6, replace = TRUE)
```

```
[1] "Kopf" "Kopf" "Kopf" "Kopf" "Zahl" "Zahl"
```

Wenn Sie diese drei Zeilen in Ihrem RStudio ausführen, werden Sie feststellen, dass Sie das gleiche Ergebnis erhalten werden. So lassen sich Simulationen replizieren.

[1] das gilt auch für alle anderen Funktionen, die auf Zufallsprozessen basieren (bspw.: rnorm(), rbinom(),...)