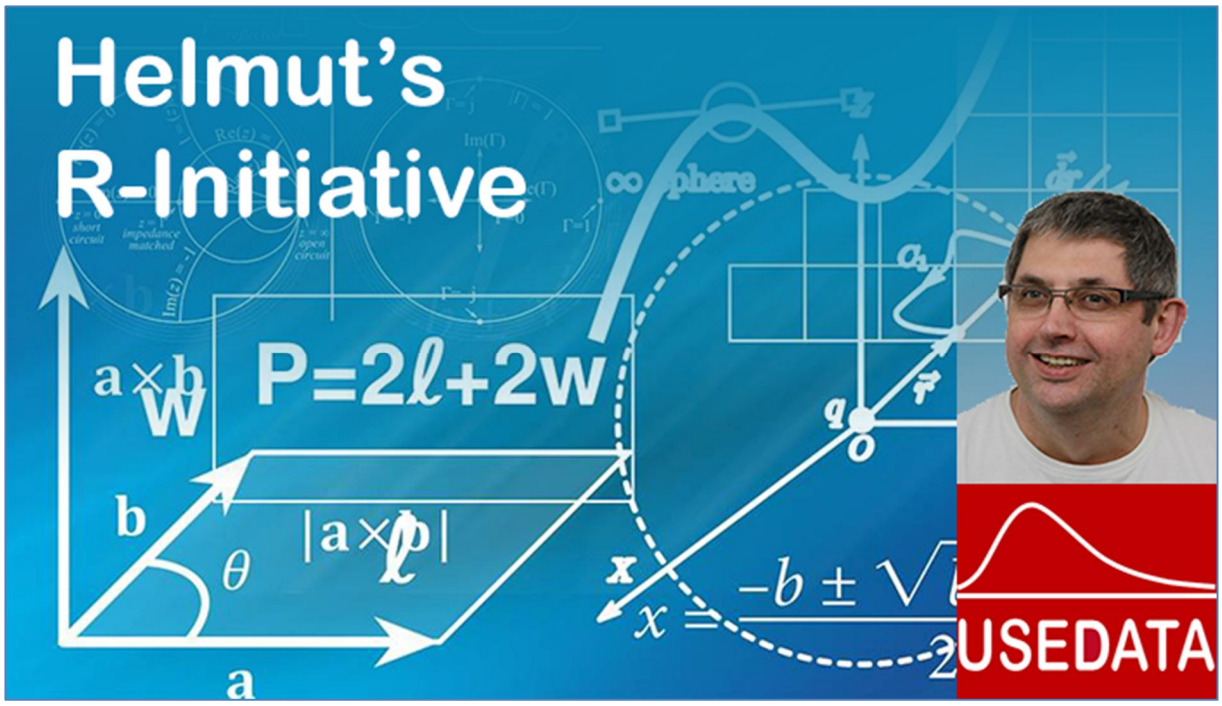
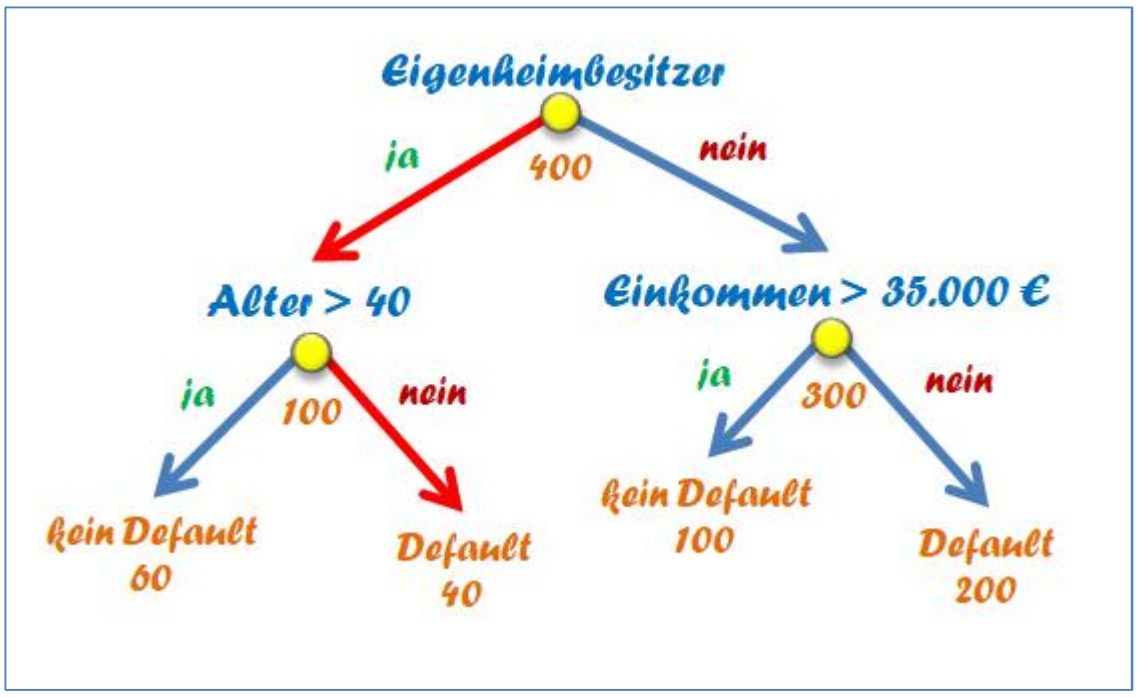


Im ersten Teil meines Blogs zum Thema Kreditscoring wurde für die Klassifizierung von Kunden die logistische Regression genutzt. Die Optimierung des Modells erfolgte mittels ***Rückwärtselimination***. Im zweiten Teil werden nun die Kunden mit Hilfe des ***binären Entscheidungsbaums*** klassifiziert.

Was versteht man unter einem binären Entscheidungsbaum?



Die o. a. Abbildung zeigt ein mögliches Ergebnis für einen Entscheidungsbaum. Dieser besteht aus Knoten und Pfaden. Ein dreißigjähriger Eigenheimbesitzer würde z. B. dem roten Pfad im Entscheidungsbaum folgen. Der rote Pfad endet mit der Entscheidung ***Default*** und

führt damit zur Ablehnung des Kredits. Der Entscheidungsbaum zeigt anschaulich, wie man zu dieser Entscheidung kommt - daher auch der Name.

Wie erstellt man einen Entscheidungsbaum?

Betrachten wir zu diesem Zweck unsere Trainingsdaten:

Default	Kreditbetrag	Scoreklasse	Wohnen	Einkommen	Alter	Jobdauer	Zins
0	5000	B	RENT	24000	33	0-15	8-11
0	2400	C	RENT	12252	31	15-30	Missing
0	10000	C	RENT	49200	24	0-15	11-13.5
0	5000	A	RENT	36000	39	0-15	Missing
0	3000	E	RENT	48000	24	0-15	Missing
0	12000	B	OWN	75000	28	0-15	11-13.5

Die Tabelle enthält sowohl metrische (Kreditbetrag, Einkommen und Alter) als auch kategoriale Merkmale (Scoreklasse, Wohnen, Jobdauer und Zins).

Bei einem binären Entscheidungsbaum wird jedes Merkmal in zwei Gruppen geteilt. Dies erfolgt bei metrischen Daten auf Basis eines Trennwertes. Alle Werte, die kleiner gleich dem Trennwert sind, werden einer Gruppe, der Rest der anderen Gruppe zugeteilt. Bei kategorialen Daten stehen zwei Töpfe zur Auswahl. Alle verfügbaren Ausprägungen werden nun beliebig auf die beiden Töpfe aufgeteilt. Es stellt sich nun die Frage, welcher Trennwert bzw. welche Aufteilung optimal ist. Oder anders formuliert:

Wie erkennt man, ob eine gewählte Aufteilung gut trennt?

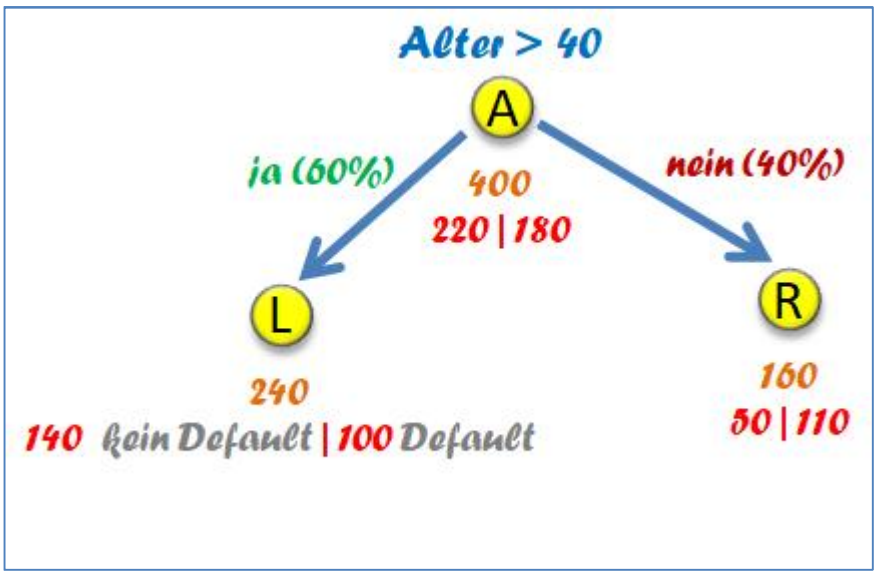
Dafür bedient man sich unterschiedlicher Techniken: Nachdem eine Aufteilung festgelegt wurde, kann man mittels χ^2-Test, Entropie, Gini-Kennzahl oder anderen Verfahren die Güte der Aufteilung beurteilen. Programmtechnisch wird aus Gründen der Einfachheit oft die ***Gini-Kennzahl*** als Gütekriterium genutzt.

Die *Gini-Kennzahl* berechnet sich wie folgt:

Von jedem Knoten kennen wir die Gesamtanzahl aller darin enthaltenen Elemente. Weiters wissen wir aus den Trainingsdaten, wie viele Elemente davon den Gruppen ***kein Default*** und ***Default*** zugeordnet wurden. Mit Hilfe dieser Informationen kann die ***Gini-Kennzahl*** berechnet werden:

$$Gini = 2 * \frac{Anzahl(kein\ Default)}{Anzahl(Gesamt)} * \frac{Anzahl(Default)}{Anzahl(Gesamt)}$$

Betrachten wir als Beispiel das Merkmal **Alter**. Das Ziel ist es, mit Hilfe der Gini-Kenzahl die Güte der Trennzahl zu beurteilen.



Betrachten wir den Knoten A (Alter): dieser enthält 400 Kunden und nehmen wir als Trennzahl ein Alter von 40 Jahren an. Diese Trennzahl teilt den Knoten in die beiden Knoten L (Links) mit 240 Kunden und den Knoten R (Rechts) mit den restlichen 160 Kunden auf. Für jeden Knoten können wir nun die Anzahl an Kunden mit der Ausprägung *Default* ermitteln. 180 Kunden beim Knoten A, 100 Kunden beim Knoten L und 110 Kunden beim Knoten R. Damit stehen alle Informationen zur Verfügung, welche benötigt werden, um die Gini-Kennzahl zu berechnen.

Konkret erhalten wir für die einzelnen Knoten (A, L und R) folgende Ergebnisse:

$$Gini_A = 2 * \frac{220}{400} * \frac{180}{400} = 0.495$$

$$Gini_L = 2 * \frac{140}{240} * \frac{100}{240} = 0.486$$

$$Gini_R = 2 * \frac{50}{160} * \frac{110}{160} = 0.430$$

Aus diesen einzelnen Gini-Kennzahlen kann man nun eine gemeinsame Kennzahl G zur Bewertung der Aufteilung ermitteln.

Für die konkrete Altersaufteilung (Alter > 40) erhalten wir folgende Kennzahl:

$$G = Gini_A - Anteil(Gini_L) * Gini_L - Anteil(Gini_R) * Gini_R$$

$$G = 0.495 - 0.6 * 0.486 - 0.4 * 0.430 = 0.031$$

Ob dieser Wert *G* tatsächlich eine optimale Lösung für die Aufteilung des Merkmals *Alter* darstellt, kann zunächst nicht beantwortet werden - es fehlen Vergleichszahlen. Eine Möglichkeit besteht darin, den Wert für unterschiedliche Altersaufteilungen zu bestimmen und sich so schrittweise der optimalen Aufteilung zu nähern. Diese Vorgangsweise ist aufwändig und mühsam. Doch dafür gibt es heute Computer und das *R*-Paket ***rpart***.

rpart steht für r*ecursive partitioning and regression tree*. Mit Hilfe von ***rpart*** kann man automatisch einen - im gewissen Sinne - optimierten Entscheidungsbaum erstellen.

Wir erstellen nun unseren ersten Entscheidungsbaum mit Hilfe der Trainingsdaten:

```
library(rpart)
modell <- rpart(Default ~ ., method = "class", data = dt)
plot(modell)
```

```
Error in plot.rpart(model1) : fit is not a tree, just a root
```

Wenn wir keine Anpassungen an den Daten vornehmen, wird auf Grund des geringen Auftretens von Defaults (rund 11 %) vom Algorithmus nur ein Knoten (*Root-Knoten*) gebildet. Die einzig mögliche Entscheidung lautet in diesem Fall: Kredit vergeben! Das entspricht aber nicht unseren Vorstellungen.

Um dieses Problem zu lösen, müssen wir unsere Trainingsdaten anpassen. Für den Algorithmus ist es wichtig, dass beide Fälle (*kein Default*|*Default*) in der Analyse in etwa gleich auftreten. Zu diesem Zweck können wir Datensätze mit der Ausprägung *"kein Default"* ***entfernen*** oder Datensätze mit der Ausprägung "*Default*" ***duplizieren***. Eine andere Alternativen besteht darin, dass man den Datensätzen entsprechende ***aprioi-Wahrscheinlichkeiten*** zuweist. Schließlich könnte man auch die Kosten für unterschiedliche Entscheidungen in einer ***Verlust-Matrix*** abbilden.

In diesem Beispiel habe ich die Trainingsdaten angepasst und Datensätze mit der Kennung *kein Default* auf rund ein Drittel reduziert. Die Datensätze mit der Kennung *Default* wurden belassen. Diese reduzierte Datei nenne ich ***undersampled_trainingsset***.

Man erkennt auch hier, dass die Datenaufbereitung viel Raum und Zeit einnimmt – dies ist jedoch Teil jeder Analyse. Wie bereits erwähnt, nutzen wir das Paket ***rpart*** um die Analyse durchzuführen. Diesmal jedoch mit dem reduzierten (***undersampled***) Trainingsbestand:

```
dtree <- rpart(Default ~ Kredit + Score + Wohnen + Gehalt + Alter + Jobdauer + Zins,
               method = "class",
               data = undersampled_trainingsset,
               control = rpart.control(cp = 0.001))
```

Die Funktion ***rpart*** enthält verschieden Argumente.

Das ***erste Argument*** ist ein Formelausdruck. Dieser stellt die binäre Responsevariable in Abhängigkeit von Kredit, Score, Wohnen, Gehalt, Alter, Jobdauer und Zins dar – ersichtlich wird dies durch die Wellenlinie innerhalb des Ausdrucks.

Das ***zweite Argument*** bestimmt die Analysemethode (*class*). Damit wird zum Ausdruck gebracht, dass es sich in diesem Beispiel um einen binären Entscheidungsbaum handelt.

Das ***dritte Argument*** beschreibt die reduzierten Trainingsdaten.

Das ***vierte Argument*** beschreibt das Abbruchkriterium für die Erstellung von neuen Knoten im Entscheidungsbaum. Im konkreten Fall wurde der ***cp-Wert*** auf 0.001 gesetzt. Falls für einen neuen Knoten der cp-Wert unterhalb dieser Grenze liegt, wird der Vorgang für diesen Zweig des Entscheidungsbaumes beendet.

Wir werden nun einige der vom Paket ***rpart*** erstellten Ausgaben näher betrachten. Als erstes analysieren wir die Tabelle mit der Darstellung der ***cp-Werte***. Diese Tabelle wird uns helfen, einen optimalen Entscheidungsbaum zu finden.

```
printcp(dtree)
```

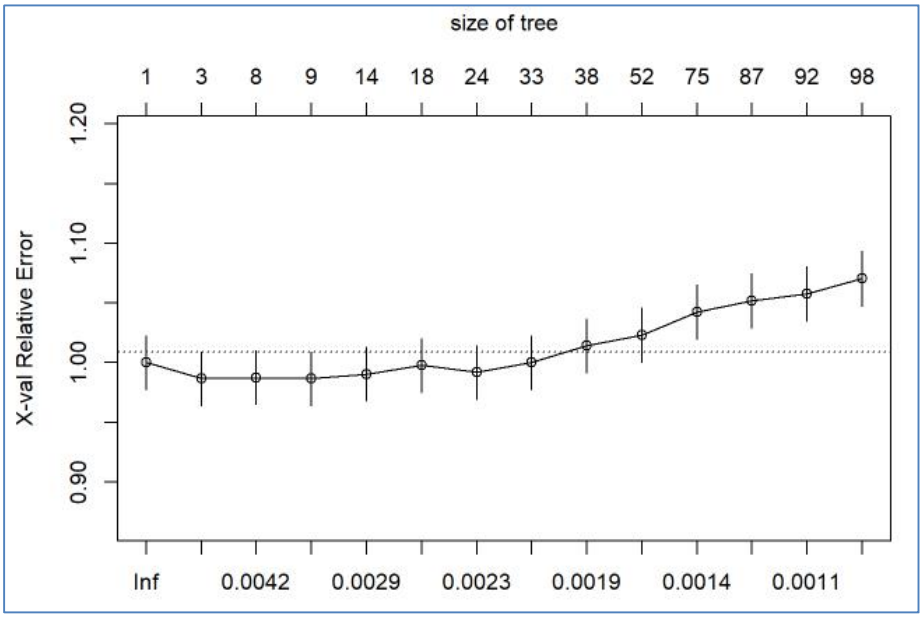
```
##
## Classification tree:
## rpart(formula = Default ~ Kredit + Score + Wohnen + Gehalt +
##     Alter + Jobdauer + Zins, data = undersampled_trainingsset,
##     method = "class", control = rpart.control(cp = 0.001))
##
## Variables actually used in tree construction:
## [1] Alter    Gehalt   Jobdauer Kredit   Score    Wohnen   Zins
##
## Root node error: 1349/4349 = 0.31019
##
## n= 4349
##
##           CP nsplit rel error  xerror     xstd
## 1  0.0155671      0   1.00000 1.00000 0.022613
## 2  0.0046948      2   0.96887 0.98666 0.022529
## 3  0.0037064      7   0.94514 0.98740 0.022534
## 4  0.0034594      8   0.94144 0.98666 0.022529
## 5  0.0024710     13   0.92365 0.99036 0.022553
## 6  0.0024092     17   0.91327 0.99778 0.022599
## 7  0.0022239     23   0.89844 0.99185 0.022562
## 8  0.0019274     32   0.87769 1.00000 0.022613
## 9  0.0018532     37   0.86805 1.01408 0.022700
## 10 0.0014826     51   0.84136 1.02298 0.022753
## 11 0.0012355     74   0.80430 1.04225 0.022866
## 12 0.0011861     86   0.78132 1.05189 0.022920
## 13 0.0011119     91   0.77539 1.05782 0.022953
## 14 0.0010000     97   0.76872 1.07042 0.023022
```

Im unteren Teil der Tabelle findet man eine ***Tabelle mit cp-Werten***. Man erkennt, dass der letzte Eintrag den cp-Wert 0.001 aufweist. Darunter findet man keinen Eintrag mehr. Dies entspricht der Vorgabe im Funktionsaufruf: `control = rpart.control(cp = 0.001)`

In der Spalte ***nsplit*** kann man die kumulierte Anzahl an Knoten, in Abhängigkeit vom cp-Wert, ablesen. Mit zunehmender Verringerung der cp-Werte findet eine Optimierung der Trainingsdaten und nicht des Problems statt. Man spricht von einer *Überanpassung*.

Betrachten wir nun die Spalte ***xerror*** der cp-Tabelle. Wir erkennen, dass der Wert zunächst abnimmt, auf niedrigem Niveau bleibt und dann wieder zunimmt. Eine grafische Darstellung dieses Sachverhalts erhalten wir mit der plotcp-Funktion.

```
plotcp(dtree)
```

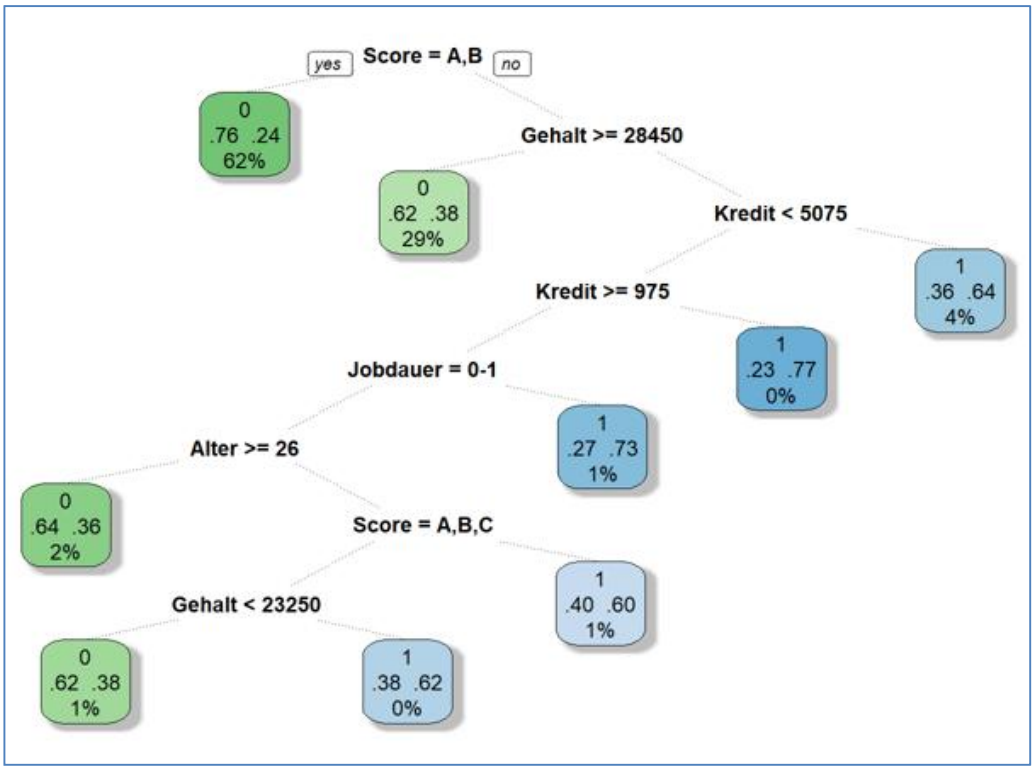


Das Minimum finden wir bei einem cp-Wert von 0.0034594. Wir erstellen nun den Entscheidungsbaum, der diesem cp-Wert entspricht.

```
prune_dtree <- prune(dtree, cp = 0.0034594)
```

Damit können wir unseren finalen Entscheidungsbaum darstellen:

```
prp(prune_dtree,
    extra = 104,
    box.palette = "GnBu",
    branch.lty = 3,
    shadow.col = "gray",
    tweak = 0.8,
    digits = -2
    )
```



Der Vorteil des Entscheidungsbaums liegt in der Klarheit seiner Darstellung. Trotzdem möchte ich auf einige Besonderheiten hinweisen: Bei allen grün eingefärbten Knoten fällt die Entscheidung zu Gunsten des Kredits. Bei allen blau eingefärbten Knoten wird der Kredit abgelehnt. Je heller die Farbe, umso weniger eindeutig ist die Entscheidung.

Trotzdem haben Entscheidungsbäume auch Nachteile: Sie liefern zumeist zu wenige Knoten, um reale Probleme zu bedienen. Sie sind jedoch eine große Hilfe beim Aufspüren von Modellen. Bei der logistischen Regression hat man dieses Problem nicht.

Sie haben nun zwei Verfahren für Binärentscheidungen kennengelernt:

- die logistische Regression und
- den binäre Entscheidungsbaum

Mit diesen beiden Verfahren habe ich die Mächtigkeit der Programmiersprache *R* aufgezeigt. Dabei handelt es sich jedoch nur um die *kleine* Spitze eines *riesigen* Eisbergs. In den nächsten Beiträgen werde ich Ihnen die Programmiersprache *R* Schritt für Schritt näher bringen.

Falls Ihnen dieser Weg, die Programmiersprache *R* zu erlernen, zu mühsam erscheint, können Sie mich gerne unter helmut.grillenberger@usedata.com kontaktieren. Weitere Informationen finden Sie auf meiner Webseite www.usedata.com. Ich berate und unterstütze Sie und Ihr Team gerne in allen Fragen zur Programmiersprache *R, Statistik und Data Science.*