
1 Big Data und Data Science

Zuerst wollen wir uns etwas mit Begriffen beschäftigen, um zu verstehen, worum es beim Thema Data Science geht. Aufbauend auf dem Begriff Big Data wird aufgezeigt, was eigentlich alles zu Data Science gehört und welche Fähigkeiten Data Scientists benötigen.

1.1 Einführung in Big Data

Den Begriff *Big Data* gibt es jetzt bereits seit einigen Jahren und der ursprüngliche mit diesem Thema verbundene Hype ist längst Vergangenheit. Stattdessen gibt es neue Buzzwords, wie das *Internet der Dinge* (engl. *Internet of Things*), die *künstliche Intelligenz* (engl. *Artificial Intelligence*), und hierbei insbesondere auch die *tiefen neuronalen Netze* (engl. *Deep Neural Network*, *Deep Learning*). Nichtsdestotrotz ist Big Data mit diesen neuen Themen eng verbunden und häufig eine Voraussetzung oder zumindest eine verwandte Technologie.

Trotz der anhaltenden Relevanz des Themas ist dennoch häufig kein gutes Verständnis für den Unterschied zwischen vielen Daten und Big Data vorhanden. Ein gutes Verständnis der Besonderheiten und Eigenschaften von Big Data und von den damit verbundenen Implikationen und Problemen ist jedoch zwingend notwendig, wenn man auf Big Data aufbauende Technologien in Projekten einsetzen will. Der Grund für Missverständnisse rund um den Begriff Big Data ist einfach: Wir denken intuitiv an »große Datenmengen«. Eine derart vereinfachte Begriffsdefinition ignoriert jedoch wesentliche Aspekte von Big Data. Backups sind ein gutes Beispiel für große Datenmengen, die nicht Big Data sind. In modernen Rechenzentren werden Backups auf Hintergrundspeichern mit einer hohen Bitstabilität, aber auch einer hohen Latenz gespeichert. Dort lagern häufig riesige Datenmengen in der Hoffnung, dass sie nie gebraucht werden, bevor sie gelöscht oder überschrieben werden. Es gibt noch einen weiteren Grund, warum es unpraktisch ist, Big Data nur über das Datenvolumen zu definieren: Wir müssten die Definition ständig anpassen, da die Speicherkapazitäten, die Rechenkraft und der Arbeitsspeicher stetig wachsen.

Eine bessere und allgemein akzeptierte Definition für Big Data basiert auf den *drei Vs*¹.

Definition von Big Data:

Als Big Data bezeichnet man Daten, die ein hohes *Volumen*, eine hohe *Geschwindigkeit* (engl. *velocity*) und eine hohe *Vielfalt* (engl. *variety*) haben, sodass man kosteneffiziente und innovative Informationsverarbeitungsmethoden benötigt, um Wissen zu generieren, Entscheidungen zu treffen oder Prozesse zu automatisieren.

Zum besseren Verständnis zerlegen wir nun diese Definition in ihre Einzelteile und betrachten diese genauer. Hierbei wird klar werden, warum Big Data mehr ist als nur Datenvolumen.

1.1.1 Volumen

Auch wenn das Datenvolumen nicht der einzig wichtige Faktor ist, ist es dennoch entscheidend. Nicht umsonst heißt es Big Data. Dass keine bestimmte Datengröße das Kriterium sein kann, wird schon klar, wenn man sich überlegt, dass Google die Forschungsarbeit, in der MapReduce vorgestellt wurde, bereits 2006 publiziert hat [Dean & Ghemawat 2008]. Zu diesem Zeitpunkt war ein Terabyte noch ein sehr großes Datenvolumen. Im Jahr 2021 ist dies lediglich die Festplattengröße des Laptops, auf dem dieses Buch geschrieben wurde. Ein weiteres Beispiel ist das Wachstum des Datenvolumens, das im Internet jährlich übertragen wird (Abb. 1–1).

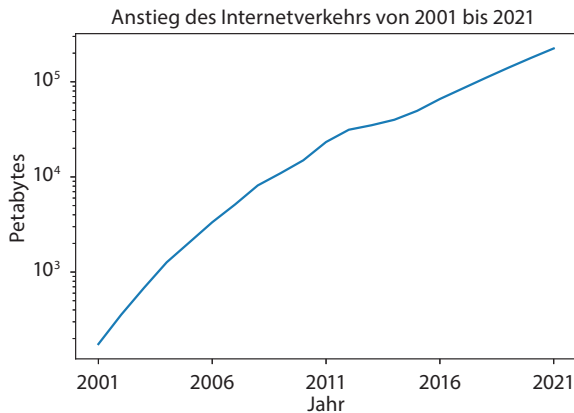


Abb. 1–1 Wachstum des Datenvolumens im Internetverkehr

1. <https://www.gartner.com/en/information-technology/glossary/big-data>

Eine vereinfachte Richtlinie für das Datenvolumen lautet, dass es mehr Daten sein müssen, als in den Arbeitsspeicher moderner Server passen. Besser ist es jedoch, wenn man sich einfach die Frage stellt, ob es möglich ist, die Daten (oft) zu kopieren, insbesondere auch über Netzwerkverbindungen. Ist dies nicht mehr der Fall, handelt es sich vermutlich um genug Daten, um von Big Data zu sprechen. In extremen Fällen sind die Daten sogar so groß, dass man sie gar nicht über das Netzwerk kopieren kann. Stattdessen nutzt man das *Sneaker Net*²: Die Daten werden direkt auf Festplatten verschickt. In Bezug auf den Datendurchsatz ist ein mit Festplatten beladenes Transportflugzeug unschlagbar. Die Latenz lässt jedoch zu wünschen übrig. Ein Beispiel für eine Anwendung, die ohne das Sneaker Net nicht geklappt hätte, ist die Erstellung des ersten Bilds von einem schwarzen Loch [Whitwam 2019].

1.1.2 Velocity/Geschwindigkeit

Die Velocity ist die *Geschwindigkeit*, mit der neue Daten generiert, verarbeitet und/oder ausgewertet werden müssen. Es gibt viele Beispiele für Daten, die eine hohe Geschwindigkeit haben, zum Beispiel die durch Sensoren wie LIDAR und Kameras erfassten Daten von autonomen Fahrzeugen. Derartige Daten können in kürzester Zeit ein sehr hohes Volumen erreichen. Die Firma Waymo hat zum Beispiel einen zwei Terabyte großen Datensatz, der während elf Fahrstunden gesammelt wurde, veröffentlicht³. Daten, die mehr oder weniger kontinuierlich in hoher Geschwindigkeit generiert werden, nennt man auch *Streamingdaten*.

Eine besondere Schwierigkeit beim Umgang mit Streamingdaten besteht darin, dass diese oft in nahezu Echtzeit verarbeitet werden müssen. Beim autonomen Fahren ist dies sofort klar, schon alleine wegen der Sicherheit. Doch das gilt auch für viele andere Anwendungen, zum Beispiel für das Sortieren des Nachrichtenstreams in sozialen Netzwerken. Hier kommt zwar niemand zu Schaden, die Nutzer würden einen Dienst aber schnell verlassen, wenn die Ladezeiten zu lang sind. Entsprechend müssen beim Umgang mit Streamingdaten in der Regel zwei Anforderungen gleichzeitig erfüllt werden: Daten müssen sehr schnell empfangen werden und dürfen dann auch nicht lange in einem Zwischenspeicher landen bzw. sich dort befinden, sondern müssen sofort verarbeitet und ausgewertet werden. Hierdurch ergibt sich eine Art inverse Korrelation zwischen der Geschwindigkeit und dem Datenvolumen: Je höher die Geschwindigkeit, desto weniger Daten reichen aus, um Daten zu Big Data werden zu lassen. Oder an einem Beispiel: Ein Gigabyte pro Tag zu verarbeiten ist einfacher als ein Gigabyte pro Sekunde.

2. <https://en.wikipedia.org/wiki/Sneakernet>

3. <https://waymo.com/open/>

1.1.3 Variety/Vielfalt

Die Vielfalt der Daten ist der dritte große Aspekt von Big Data. Mittlerweile ist die Analyse von Bildern, Videos und Texten zu einer normalen Anwendung geworden. Dies war jedoch noch nicht der Fall, als der Begriff Big Data geprägt wurde. Im Zeitraum um die Jahrtausendwende lagen die Daten, die analysiert werden sollten, üblicherweise strukturiert vor, zum Beispiel in relationalen Datenbanken. Die Daten waren entweder numerisch oder in feste Kategorien eingeteilt. Das änderte sich im Laufe der 2000er-Jahre, dadurch dass das Internet allgegenwärtig wurde und wir immer mehr Computertechnik in unseren Alltag übernommen haben, zum Beispiel in Form von Smartphones. Hier entstanden Daten eher auf unstrukturierte Weise, zum Beispiel durch Webseiten, die ad hoc von Nutzern erstellt wurden. Es ist daher kein Zufall, dass Google den Begriff Big Data und die damit verbundenen Technologien mitgeprägt hat: Die Indizierung des stetig wachsenden und komplexer werdenden Internets zwang dazu, die vorhandenen Techniken rapide weiterzuentwickeln. Hierbei mussten nicht nur immer größere Datenmengen verarbeitet werden, sondern vor allem eine Vielfalt von Datenformaten, insbesondere Text- und Bilddaten, später auch Videodaten.

Insgesamt gibt es viel mehr unstrukturierte Daten als strukturierte Daten. Dies wird üblicherweise als Pyramide dargestellt (Abb. 1–2), in der zwischen *unstrukturierten*, *quasistrukturierten*, *semistrukturierten* und *strukturierten* Daten unterschieden wird.

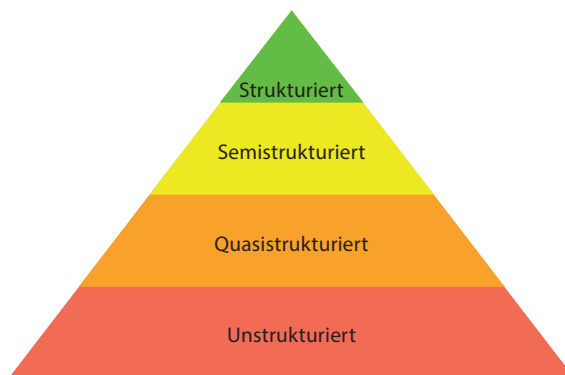


Abb. 1–2 Datenpyramide

An der Spitze der Pyramide sind die strukturierten Daten, zum Beispiel Tabellen in relationalen Datenbanken, *Comma-Separated-Value*-(CSV-)Dateien und Ähnliches. Strukturierte Daten kann man im Normalfall direkt in ein Analysetool laden, ohne dass Vorverarbeitungsschritte notwendig sind. Die Vorverarbeitung beschränkt sich daher bei strukturierten Daten höchstens auf Aufgaben wie das Säubern der Daten, um beispielsweise ungültige Datenpunkte oder Ausreißer zu filtern.

Als Nächstes kommen die semistrukturierten Daten, zum Beispiel XML und JSON. Der Hauptunterschied zwischen strukturierten und semistrukturierten Daten ist die Flexibilität der Datenformate. Bei strukturierten Daten ist beispielsweise in der Regel der Typ einer Spalte fest definiert. Dies ist bei semistrukturierten Daten anders: Hier trifft man oftmals auf verschachtelte Strukturen, die man für die Analyse erst aufbrechen muss. Außerdem gibt es häufig optionale Felder, wodurch die Verarbeitung komplizierter werden kann. Dennoch kann man mit semistrukturierten Daten überwiegend einfach arbeiten, da diese auch ohne großen Aufwand in viele Analyseumgebungen importiert werden können.

Im Allgemeinen gilt für strukturierte und semistrukturierte Daten gemeinsam, dass es feste Datenformate und/oder Anfragesprachen gibt, mit denen man einfach die benötigten Informationen extrahieren und laden kann. Dies ist in den beiden unteren Ebenen der Pyramide nicht mehr der Fall. Quasistrukturierte Daten haben zwar eine fest definierte Struktur, es ist aber ein gewisser Aufwand erforderlich, um an die benötigten Informationen zu kommen. Als Beispiel betrachten wir die Ausgabe des Befehls `ls -l`, mit dem man sich in einem Linuxterminal die Dateien in einem Ordner anzeigen lassen kann.

```
%ls -l
```

```
total 6792
drwxr-xr-x 1 sherbold sherbold    512 Mar 24 14:23 data/
-rw-r--r-- 1 sherbold sherbold   2957 May  5 14:30 howto.ipynb
drwxr-xr-x 1 sherbold sherbold    512 Apr 27 17:04 images/
-rw-r--r-- 1 sherbold sherbold   63683 May  6 11:52 kapitel_01.ipynb
-rw-r--r-- 1 sherbold sherbold  113117 May 10 10:11 kapitel_02.ipynb
-rw-r--r-- 1 sherbold sherbold   24576 May 10 10:08 kapitel_03.ipynb
-rw-r--r-- 1 sherbold sherbold  886609 May 10 10:10 kapitel_04.ipynb
-rw-r--r-- 1 sherbold sherbold   39543 May  6 16:44 kapitel_05.ipynb
-rw-r--r-- 1 sherbold sherbold 1664391 May  6 18:50 kapitel_06.ipynb
-rw-r--r-- 1 sherbold sherbold 2679078 May 10 09:05 kapitel_07.ipynb
-rw-r--r-- 1 sherbold sherbold 199328  May 10 09:23 kapitel_08.ipynb
-rw-r--r-- 1 sherbold sherbold 446103  May 10 09:39 kapitel_09.ipynb
-rw-r--r-- 1 sherbold sherbold 579843  May 10 09:50 kapitel_10.ipynb
-rw-r--r-- 1 sherbold sherbold 148082  May 10 09:58 kapitel_11.ipynb
-rw-r--r-- 1 sherbold sherbold   54300 May  6 12:11 kapitel_12.ipynb
-rw-r--r-- 1 sherbold sherbold    5967 May  6 15:16 kapitel_13.ipynb
-rw-r--r-- 1 sherbold sherbold    4927 May  5 14:30 notations.ipynb
-rw-r--r-- 1 sherbold sherbold    3887 May  6 11:59 vorwort.ipynb
```

Man sieht eine klare Struktur in den Daten: Die meisten Zeilen beinhalten die Benutzerrechte, gefolgt von der Anzahl der Symlinks auf die Datei, dem Benutzer und der Gruppe, die die Datei besitzen, der Dateigröße, dem Datum der letzten Änderung und zuletzt dem Namen. Diese Struktur kann man nutzen, um einen Parser zu schreiben, der die Daten einliest, zum Beispiel mithilfe von einem regulären Ausdruck. Wir können also eine Struktur über quasistrukturierte Daten legen, indem wir die Struktur durch einen Parser selbst definieren. Dies ist zwar mehr Aufwand

als bei strukturierten und semistrukturierten Daten, aber man kommt dennoch zuverlässig an die benötigten Informationen. Trotzdem kann es sehr leicht passieren, dass sich die Struktur ändert und der selbst geschriebene Parser nicht mehr funktioniert. Daher ist das Lesen von quasistrukturierten Daten fehleranfällig, da man eventuell Sonderfälle übersieht oder sich das Datenformat ändern kann. Man sollte also den oft signifikanten Wartungsaufwand bei der Verarbeitung von quasistrukturierten Daten für die Nutzung im Produktivbetrieb berücksichtigen.

Die unstrukturierten Daten sind auf der untersten Ebene der Pyramide. Hier befindet sich der Großteil der Daten: Bilder, Videos und Text. Die Herausforderung bei diesen Daten ist es, eine Struktur zu bestimmen, die man später verarbeiten kann. Hier gibt es keine Faustformel, es hängt stattdessen von den konkreten Daten und der geplanten Anwendung ab. Hinzu kommt, dass unstrukturierte Daten häufig vermischt sind. Dieses Buch ist ein gutes Beispiel: Wir haben eine Mischung aus natürlicher Sprache, Bildern, Formatinformationen (z.B. Überschriften, Listen) und Quelltext.

1.1.4 Innovative Informationsverarbeitungsmethoden

Auch wenn die drei Vs als die zentralen Eigenschaften von Big Data angesehen werden, sind die anderen Teile der Definition auch wichtig, um zu verstehen, dass Big Data mehr ist als einfach nur viele Daten, die möglicherweise schnell generiert werden und verschiedene Formate haben. Der nächste Teil der Definition spricht von dem Bedarf an *innovativen Informationsverarbeitungsmethoden*. Das bedeutet, dass man für Big Data nicht einen normalen Arbeitsplatzrechner oder sogar ein traditionelles Batch-System in einem Großrechner, in dem sich viele Rechenknoten einen Speicher über das Netzwerk teilen, nutzen kann. Stattdessen ist die *Datenlokalität* (engl. *data locality*) wichtig, da man in der Regel keine Kopien der Daten über das Netzwerk erzeugen kann. Dies hat zu einer Transformation geführt, sodass es immer mehr Infrastrukturen gibt, in denen Rechenkraft und schneller, verteilter Speicher direkt integriert sind. Als Big Data ein neues Konzept war, gab es solche Technologien noch nicht. Heutzutage hat man viele Möglichkeiten, allein bei der Apache Foundation⁴ gibt es unter anderem Hadoop, Spark, Storm, Kafka, Cassandra, Hive, HBase, Giraph und viele weitere Technologien, die hochprofessionell als Open Source entwickelt und von vielen Unternehmen zur Verarbeitung von Big Data eingesetzt werden.

4. <https://www.apache.org/>

1.1.5 Wissen generieren, Entscheidungen treffen, Prozesse automatisieren

Der letzte Teil der Big-Data-Definition bedeutet, dass Big Data kein Selbstzweck ist. Man spricht nur dann von Big Data, wenn man die Daten auch zum Erreichen eines Ziels nutzt. Ziele können der reine Erkenntnisgewinn sein, die Unterstützung der Entscheidungsfindung oder sogar die Automatisierung ganzer Geschäftsprozesse. Dieser Aspekt der Definition ist so wichtig, dass er häufig als weiteres V betrachtet wird: *Value*.

1.1.6 Noch mehr Vs

Die Definition von Gartner, die wir hier im Buch verwenden, hat »nur« drei Vs. Die Definition von Big Data durch Wörter, die mit V anfangen, ist jedoch so populär, dass es verschiedene Erweiterungen gibt mit bis zu 42 (!) Vs [Shafer 2017]. Die 42 Vs sollte man aber eher als Satire verstehen, die zeigen soll, dass mehr Vs nicht immer zu einer besseren Definition führen. Nichtsdestotrotz gibt es seriöse Definitionen mit bis zu zehn Vs⁵. Ein zusätzliches V hatten wir bereits: *Value*, also die Wertschöpfung durch Big Data. Die *Korrektheit* (engl. *veracity*) ist ein weiteres V, was häufig als hochrelevant eingeschätzt wird. Je mehr Daten man auswertet, desto schwieriger wird es, sicherzustellen, dass die Daten zuverlässig sind und sich Ergebnisse reproduzieren lassen. Dies ist insbesondere dann schwer, wenn sich die Datenquellen oft ändern, zum Beispiel bei der Analyse von Nachrichten oder der sozialen Medien. Volume, Velocity, Variety, Veracity und Value zusammen ergeben die Fünf-V-Definition von Big Data, die stark verbreitet ist. Weitere Vs betrachten wir an dieser Stelle nicht mehr.

1.2 Einführung in Data Science

Auch wenn der Begriff *Data Science* als Buzzword sehr populär ist, existiert noch keine allgemein akzeptierte Definition. Hierfür gibt es vermutlich zwei Gründe: Erstens ist der Begriff sehr generisch, sodass jede Verwendung von Daten, die in irgendeiner Form als »wissenschaftlich« betrachtet wird, als Data Science bezeichnet werden kann. Und zweitens gibt es einen großen Hype um diesen Begriff, weshalb Firmen, Fördermittelgeber und öffentliche Institutionen mit diesem Begriff Werbung für sich betreiben wollen.

Aus diesem Grund können wir hier leider auch keine kurze und einprägsame Definition für diesen Begriff geben. Stattdessen betrachten wir, welche Konzepte unter anderem als Data Science angesehen werden, und schauen uns Beispiele für Data-Science-Anwendungen an.

5. <https://tdwi.org/articles/2017/02/08/10-vs-of-big-data.aspx>

1.2.1 Was gehört zu Data Science?

Data Science kombiniert Methoden aus der Mathematik, Statistik und Informatik mit dem Ziel, datengetriebene Anwendungen zu entwickeln oder Wissen zu generieren. Die Wertschöpfung ist also sehr ähnlich zu Big Data. Der Hauptunterschied zwischen den Begriffen liegt auf dem Fokus auf große Datenmengen bei Big Data, der bei Data Science nicht gegeben sein muss.

Mathematik ist häufig das Fundament, auf dem die Datenanalyse definiert wird. Die Modelle über die Daten, die erstellt werden, sind im Endeffekt nichts anderes als eine mathematische Beschreibung von Aspekten der Daten. Man könnte also Data Science als mathematische Modellierung verstehen. Die Rolle der Mathematik ist jedoch größer als die einer »Beschreibungssprache« für Modelle. Teilgebiete der Mathematik liefern die Methoden, die man braucht, um Modelle zu bestimmen.

- *Optimierung* beschreibt, wie man die optimale Lösung für eine Zielfunktion findet, sodass die gefundene Lösung gewisse Nebenbedingungen erfüllt. Die Zielfunktion und die Nebenbedingungen werden bei Data Science häufig direkt aus den Daten ermittelt, sodass die Lösung optimal für die gegebenen Daten ist.
- *Stochastik* ist ein Teilgebiet der Mathematik, das sich mit zufälligen Verhalten durch Zufallsvariablen und stochastischen Prozessen befasst. Stochastik bildet daher eine wichtige Grundlage für die Theorie des maschinellen Lernens, und stochastische Modelle werden häufig genutzt, um Daten zu beschreiben.
- Ohne die *Geometrie* könnte man keine Daten, die räumlich verteilt sind, analysieren, zum Beispiel geografische Daten oder der 3-dimensionale Raum vor einem Fahrzeug.
- *Wissenschaftliches Rechnen* wird immer häufiger nicht nur genutzt, um Daten für Analysen zu simulieren, sondern auch, um Modelle über Daten durch Simulation zu bestimmen.

Die Statistik befasst sich mit der Analyse von Daten durch Stichproben, zum Beispiel das Schätzen der den Daten zugrunde liegenden Verteilungen, Zeitreihenanalysen oder die Entwicklung von statistischen Tests, mit denen man auswerten kann, ob Effekte zufällig oder signifikant sind.

- *Lineare Modelle* sind eine vielfältig einsetzbare Methode, um Daten zu beschreiben, um daraus Zusammenhänge zu erkennen oder Trends zu ermitteln.
- *Inferenz* ist ein ähnliches Verfahren, nur dass Wahrscheinlichkeitsverteilungen statt linearer Modelle genutzt werden, um die Daten zu beschreiben.
- *Zeitreihenanalyse* nutzt die interne Struktur von Daten, die über die Zeit gemessen wurden. Hierbei werden zum Beispiel saisonale Effekte oder andere sich wiederholende Muster genutzt, um die Struktur der Zeitreihe zu modellieren und zukünftige Werte vorherzusagen.

Ohne die Informatik wären die mathematischen und statistischen Verfahren nicht durch Computer ausführbar. Doch auch die theoretische Informatik liefert wichtige Grundlagen für Data Science.

- *Datenstrukturen und Algorithmen* sind die Grundlage von jeder effizienten Umsetzung von Algorithmen. Ohne das Verständnis von Datenstrukturen, wie Bäumen, Hashmaps und Listen, sowie von der Laufzeitkomplexität von Algorithmen wären Datenanalysemethoden nicht skalierbar auf große Datenmengen.
- Die *Informationstheorie* liefert das nötige Verständnis über die Entropie (Unsicherheit in den Daten) und gemeinsame Information von Daten und ist damit die Grundlage für viele Algorithmen des maschinellen Lernens.
- *Datenbanken* werden benötigt, um Daten effizient zu speichern und zugreifbar zu machen. SQL ist als Anfragesprache nicht nur bei relationalen Datenbanken, sondern auch bei NoSQL-Datenbanken allgegenwärtig.
- *Paralleles und verteiltes Rechnen* sind notwendig, um das Arbeiten mit großen Datenmengen und hoher Rechenkraft zu ermöglichen.
- Die klassische *künstliche Intelligenz* liefert die Grundlagen für die logische Modellierung und die Definition von Regelsystemen für viele Data-Science-Anwendungen. In diesem Buch unterscheiden wir explizit zwischen künstlicher Intelligenz und maschinellem Lernen. Wir benutzen den Begriff *künstliche Intelligenz*, um Anwendungen wie Deep Blue [Newborn 1997], die regelbasierte Schach-Engine, die als erster Computer Gary Kasparow im Schach besiegt hat, zu beschreiben.
- *Softwaretechnik* ist für die Operationalisierung von Anwendungen und das Management von Data-Science-Projekten notwendig.

Und zuletzt gibt es natürlich noch das *maschinelle Lernen*, was häufig auch als definierender Aspekt von Data Science gesehen wird. Das maschinelle Lernen kombiniert Mathematik, Statistik und Informatik auf geschickte Art und Weise. Je nachdem welche Methoden man betrachtet, können alle drei Disziplinen das maschinelle Lernen für sich beanspruchen. Durch maschinelles Lernen versucht man, Wissen aus Daten zu gewinnen, sodass das Wissen die Daten nicht nur beschreibt, sondern auch auf weitere Daten und Applikationen angewendet werden kann, zum Beispiel durch neuronale Netze, Entscheidungsbäume und Mustererkennung.

1.2.2 Beispielanwendungen

So unterschiedlich wie die Grundlagen von Data Science sind, so verschieden sind auch die Anwendungen in der Forschung, Industrie und Gesellschaft. Hier sind fünf kurze Beispiele:

- *Alpha Go* ist ein Beispiel für ein intelligentes selbstlernendes System. Vor einigen Jahren überraschte Alpha Go die Fachwelt, weil es scheinbar aus dem Nichts kam und einen der weltbesten Spieler im Go besiegte. Go gilt als besonders schwieriges Spiel, zum Beispiel im Verhältnis zu Schach, und Computer waren bis dahin gerade mal auf dem Niveau von Amateuren und stellten keine Herausforderung für professionelle Spieler dar. Alpha Go kombinierte klassische regelbasierte künstliche Intelligenz mit statistischen Monte-Carlo-Simulationen und selbstlernenden neuronalen Netzen, um dies zu erreichen.
- Die *Robotik* nutzt maschinelles Lernen, um den Robotern beizubringen, sich zu bewegen. Mit der Zeit können Roboter zum Beispiel lernen, durch welche Bewegungen sie das Umfallen verhindern können [Hwangbo et al. 2019].
- *Marketing* setzt auf Data Science, um im Internet gezielt Werbung schalten zu können. Die dahinter liegende Industrie erwirtschaftet Milliarden, indem Nutzern relevante Werbung basierend auf ihrem Verhalten im Internet gezeigt wird.
- In der *Medizin* werden Daten immer häufiger genutzt, um Entscheidungen zu unterstützen. IBM Watson, das erste Computerprogramm, das Menschen im Jeopardy besiegt hat⁶, wird heutzutage zum Beispiel auch eingesetzt, um geeignete Krebstherapien auszuwählen⁷. (Auch wenn das nicht so gut klappt, wie man es sich ursprünglich erhofft hat [Strickland 2019].)
- Im *autonomen Fahren* wird maschinelles Lernen für verschiedene Aufgaben genutzt, zum Beispiel für die Erkennung von Objekten, wie Verkehrsschildern, anderen Autos oder Fußgängern.

6. <https://www.ibm.com/ibm/history/ibm100/us/en/icons/watson/>

7. <https://www.ibm.com/marketplace/clinical-decision-support-oncology>

1.3 Fähigkeiten von Data Scientists

Data Scientists sind weder Informatikerinnen, Mathematikerinnen, Statistikerinnen oder Domänenexpertinnen. Der perfekte Data Scientist bringt eine Kombination von allem als Fähigkeiten mit:

- Gute mathematische Fähigkeiten, insbesondere über Optimierung und Stochastik
- Sicherer Umgang mit Methoden aus der Statistik, insbesondere Regression, statistische Tests und Inferenz
- Gute Programmierkenntnisse, sicherer Umgang mit Datenbanken, Datenstrukturen, parallelem Rechnen und Big-Data-Technologien
- Problemloser Wechsel zwischen den obigen Fähigkeiten und sicherer Umgang mit Technologien, die in der Schnittstelle liegen, insbesondere dem maschinellen Lernen
- Genug Wissen über die Domäne, um die Daten zu verstehen, Fragestellungen zu definieren und zu erarbeiten, ob und wie diese Fragen mithilfe von Daten beantwortet werden können

Außerdem müssen Data Scientists teamfähig sein, um mit Domänenexpertinnen auf der einen Seite und technischen Expertinnen auf der anderen Seite zusammenarbeiten zu können. Die Domänenexpertinnen helfen den Data Scientists, die Daten, Fragestellungen und Projektziele zu verstehen. Die technischen Expertinnen helfen bei der Umsetzung von Projekten, insbesondere bei der Operationalisierung.

Nicht zuletzt sollten Data Scientists zwar den notwendigen Enthusiasmus mitbringen, um sich für die Arbeit mit Daten begeistern zu können, aber auch die notwendige Skepsis, um die Problemstellung nach wissenschaftlichen Prinzipien anzugehen. Das heißt insbesondere auch, dass man alles tun sollte, um auszuschließen, dass etwas nur aus Zufall funktioniert, und rigoros überprüfen muss, ob Modelle wirklich wie gewünscht funktionieren.

Wenn man sich dieses Fähigkeitsprofil anschaut, wird schnell klar, dass die Anzahl der Personen, die alles mitbringen, begrenzt ist. Microsoft Research hat sich daher mit der Fragestellung befasst, was Data Scientists im Arbeitsalltag leisten und welche Arten von Data Scientists es gibt [Kim et al. 2017]. Hierbei wurden acht Arten von Data Scientists bestimmt:

- *Polymath* sind die Alleskönner, die das gesamte oben beschriebene Profil erfüllen und alles von der zugrunde liegenden Mathematik bis hin zu den Big-Data-Infrastrukturen verstehen.
- *Data Evangelists* analysieren selbst Daten, verbreiten aber auch die Erkenntnisse und Modelle. Sie setzen sich insbesondere auch dafür ein, dass aus ihren Modellen Produkte entwickelt werden.

- *Data Preparers* sammeln Daten und bereiten diese für die Analyse auf.
- *Data Analyzers* analysieren Daten, die ihnen zur Verfügung gestellt werden.
- *Data Shapers* kombinieren die beiden vorigen Rollen, das heißt, sie sammeln und analysieren Daten.
- *Platform Builders* sammeln nicht nur Daten, sondern entwickeln und administrieren ganze Plattformen, die zur Datensammlung und Analyse genutzt werden können.
- *Moonlighters 50%* und *Moonlighters 20%* sind Teilzeit-Data-Scientists, die zwar auch eine Data-Science-Rolle ausfüllen, aber nur als ein Bruchteil ihrer täglichen Arbeit.
- *Insight Actors* sind die Nutzer von Analysen und Modellen.

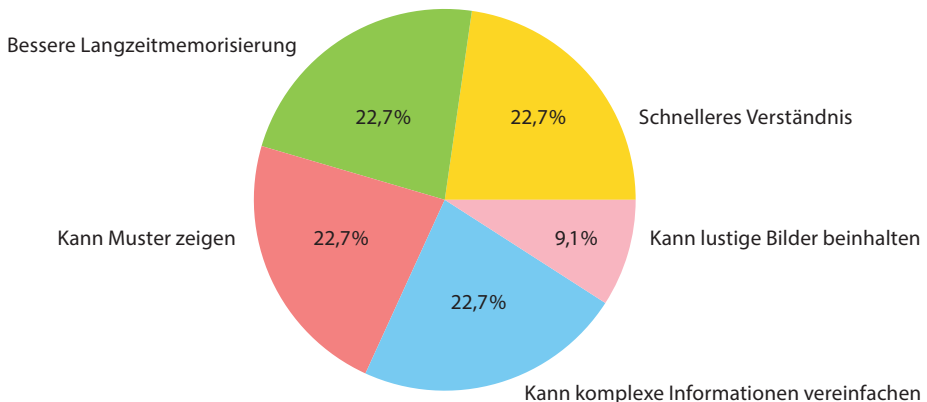
4.3 Visualisierung

Visualisierungen sind ein sehr mächtiges Werkzeug, um etwas über Daten zu lernen und diese zu verstehen. Hier sehen Sie ein Beispiel von einem Tortendiagramm (die Prozentzahlen sind ausgedacht und haben keine wissenschaftliche Bedeutung).

```
# we use matplotlib for the creation of visualizations
import matplotlib.pyplot as plt
# this is only required for Jupyter notebooks
%matplotlib inline

# data to plot
labels = ['Schnelleres Verständnis', 'Bessere Langzeitmemorisierung',
         'Kann Muster zeigen', 'Kann komplexe Informationen vereinfachen',
         'Kann lustige Bilder beinhalten']
sizes = [25, 25, 25, 25, 10]
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'pink']

# plot
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%')
plt.show()
```



Es gibt viele Vorteile von Visualisierungen gegenüber anderen Arten, Daten darzustellen, zum Beispiel Tabellen oder deskriptive Statistik. Im Allgemeinen verarbeitet man visuelle Informationen schneller. Hierdurch kann man in der Regel schneller etwas über die Daten lernen. Hinzu kommt, dass viele Menschen sich visualisierte Informationen besser merken können.

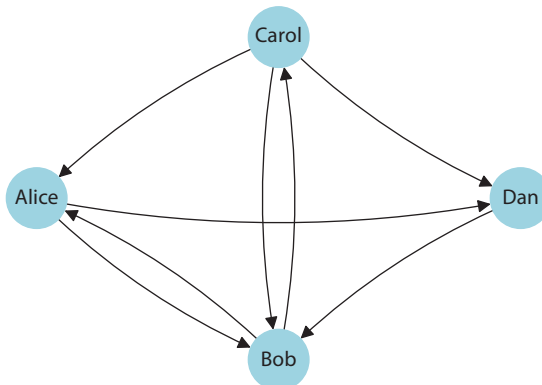
Die Vorteile von Visualisierungen gehen jedoch über die Verarbeitungsgeschwindigkeit und die Memorisierung hinaus. Visualisierungen können auch ein Verständnis der Daten ermöglichen, das man andernfalls nicht erlangen würde, zum Beispiel über Muster in Daten und komplexe Zusammenhänge, die man sonst nicht sehen könnte. Hierfür betrachten wir das folgende Beispiel. Alice kennt Bob und Dan, Dan kennt Bob, Bob kennt Carol und Alice, und Carol kennt Alice, Bob und

Dan. Diese textuelle Beschreibung ist sehr komplex und schwer zu verstehen. Außerdem bekommt man kein intuitives Verständnis der Beziehung zwischen Alice, Bob, Carol und Dan. Jetzt betrachten wir das Gleiche als gerichteten Graphen:

```
import networkx as nx # networkx is a powerful library for working with graphs

# Create the graph by adding edges
# The vertices are implicitly defined through the endpoints of the edges
graph = nx.DiGraph()
graph.add_edge('Alice', 'Bob')
graph.add_edge('Alice', 'Dan')
graph.add_edge('Dan', 'Bob')
graph.add_edge('Bob', 'Carol')
graph.add_edge('Bob', 'Alice')
graph.add_edge('Carol', 'Alice')
graph.add_edge('Carol', 'Bob')
graph.add_edge('Carol', 'Dan')

# Plot the graph with a shell layout
nx.draw_shell(graph, with_labels=True, node_size=2000, node_color='lightblue',
              arrowsize=20, connectionstyle='arc3, rad = 0.1')
```



Dieser Graph ist einfach zu lesen und gibt uns ein intuitives Verständnis der Beziehungen.

Bitte beachten Sie, dass wir bei allen Grafiken in diesem Kapitel Legenden und andere Details bewusst weglassen. Stattdessen generieren wir die Grafiken mit möglichst wenig Quelltext, sodass sie trotzdem die gewünschten Erkenntnisse liefern. Der Grund hierfür ist, dass wir hier Visualisierungen als Werkzeug zur Erkundung von Daten betrachten. Wenn man ansonsten Visualisierungen für Texte (Bücher und sonstige Publikationen) oder Präsentationen erstellt, sind andere Aspekte ebenfalls relevant, zum Beispiel die konsistente Beschriftung, Farbwahl, Legenden und Titel.

4.3.1 Anscombes Quartett

Anscombes Quartett ist ein berühmtes Beispiel für Daten, in dem deskriptive Statistiken irreführend sind. Das Beispiel basiert auf vier Datensätzen $(x_1, y_1), \dots, (x_4, y_4)$ mit je elf Paaren aus zwei Variablen x_i und y_i , $i=1, \dots, 4$. Tabelle 4–1 zeigt die Werte für jedes der Paare.

$x_{1,2,3}$	y_1	y_2	y_3	x_4	y_4
10	8,04	9,14	7,46	8	6,58
8	6,95	8,14	6,77	8	5,76
13	7,58	8,74	12,74	8	7,71
9	8,81	8,77	7,11	8	8,84
11	8,33	9,26	7,81	8	8,47
14	9,96	8,10	8,84	8	7,04
6	7,24	6,13	6,08	8	5,25
4	4,26	3,10	5,39	19	12,50
12	10,84	9,13	8,15	8	5,56
7	4,82	7,26	6,42	8	7,91
5	5,68	4,74	5,73	8	6,89

Tab. 4–1 Anscombes Quartett

Wenn wir das arithmetische Mittel und die Standardabweichung betrachten, sehen wir, dass die Werte für alle x_i und für alle y_i gleich sind.

```
import numpy as np # we now also need numpy, the commonly used numerics
                    library for Python

x = np.array([10, 8, 13, 9, 11, 14, 6, 4, 12, 7, 5]) # same for x1, x2, and x3
y1 = np.array([8.04, 6.95, 7.58, 8.81, 8.33, 9.96,
               7.24, 4.26, 10.84, 4.82, 5.68])
y2 = np.array([9.14, 8.14, 8.74, 8.77, 9.26,
               8.10, 6.13, 3.10, 9.13, 7.26, 4.74])
y3 = np.array([7.46, 6.77, 12.74, 7.11, 7.81,
               8.84, 6.08, 5.39, 8.15, 6.42, 5.73])
x4 = np.array([8, 8, 8, 8, 8, 8, 8, 19, 8, 8, 8])
y4 = np.array([6.58, 5.76, 7.71, 8.84, 8.47, 7.04,
               5.25, 12.50, 5.56, 7.91, 6.89])

print('Arithmetisches Mittel')
print('x1, x2, x3 = ', statistics.mean(x))
print('x4         = ', statistics.mean(x4))
print('y1         = ', statistics.mean(y1))
print('y1         = ', statistics.mean(y2))
print('y1         = ', statistics.mean(y3))
print('y1         = ', statistics.mean(y4))
↓
```

```

print('Standardabweichung')
print('x1, x2, x3 = ', statistics.stdev(x))
print('x4      = ', statistics.stdev(x4))
print('y1      = ', statistics.stdev(y1))
print('y1      = ', statistics.stdev(y2))
print('y1      = ', statistics.stdev(y3))
print('y1      = ', statistics.stdev(y4))

```

Arithmetisches Mittel

```

x1, x2, x3 = 9
x4      = 9
y1      = 7.500909090909091
y1      = 7.500909090909091
y1      = 7.5
y1      = 7.500909090909091

```

Standardabweichung

```

x1, x2, x3 = 3.3166247903554
x4      = 3.3166247903554
y1      = 2.031568135925815
y1      = 2.0316567355016177
y1      = 2.030423601123667
y1      = 2.0305785113876023

```

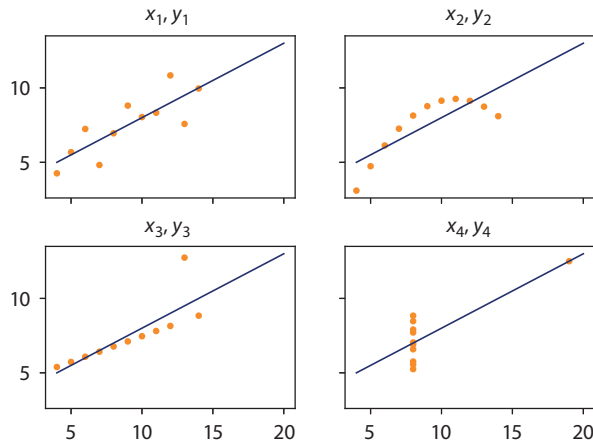
Es gibt sogar noch mehr Gemeinsamkeiten. Wenn wir eine lineare Regression von y_i durch x_i bestimmen würden (siehe Kap. 8), würden wir jedes Mal die gleiche Regressionsgerade $y = 3 + 0,5 \cdot x$ finden. Statistisch sind sich diese vier Datensätze also sehr ähnlich. Wenn wir uns die Daten mit einem einfachen *Scatterplot* visualisieren, sehen wir, dass die Datensätze eigentlich sehr unterschiedlich sind.

```

xfit = np.array([4, 20])
yfit = 3+0.5*xfit

f, axes = plt.subplots(2, 2, sharey=True, sharex=True)
axes[0, 0].plot(x, y1, color='darkorange', marker='.', linestyle='none')
axes[0, 0].plot(xfit, yfit, color='navy', lw=1)
axes[0, 0].set_title('$x_1, y_1$')
axes[0, 1].plot(x, y2, color='darkorange', marker='.', linestyle='none')
axes[0, 1].plot(xfit, yfit, color='navy', lw=1)
axes[0, 1].set_title('$x_2, y_2$')
axes[1, 0].plot(x, y3, color='darkorange', marker='.', linestyle='none')
axes[1, 0].plot(xfit, yfit, color='navy', lw=1)
axes[1, 0].set_title('$x_3, y_3$')
axes[1, 1].plot(x4, y4, color='darkorange', marker='.', linestyle='none')
axes[1, 1].plot(xfit, yfit, color='navy', lw=1)
axes[1, 1].set_title('$x_4, y_4$')
plt.show()

```

Die orangen Punkte visualisieren die Daten selbst, die blauen Linien zeigen die Regressionsgerade für $y = 3 + 0,5 \cdot x$, die optimal ist für die Daten. Nur auf Basis der statistischen Informationen über die Daten würde man erwarten, dass die Daten ungefähr so wie beim Paar x_1, y_1 aussehen: Die Daten haben in etwa eine lineare Beziehung mit einer leichten Streuung ohne ein klar erkennbares Muster um die Regressionsgerade. Eine lineare Beziehung zwischen x und y bedeutet, dass wenn x sich verändert, sich y um ein konstantes Vielfaches von x verändert. Das bedeutet umgekehrt auch, dass man die Beziehung von x und y sich etwa als Gerade vorstellen kann. Beim Paar x_2, y_2 ist dies nicht der Fall, stattdessen sehen die Daten eher aus wie eine auf den Kopf gestellte Parabel. Das Paar x_3, y_3 ist eigentlich perfekt linear, bis auf einen einzelnen Datenpunkt, der nach oben ausreißt und dafür sorgt, dass die blaue Regressionsgerade nicht zur eigentlichen Geraden, auf der die Daten liegen, passt. Das Paar x_4, y_4 passt nicht zu dem, was die Statistiken aussagen, die auch wieder von einem Ausreißer stark beeinflusst werden.

Die Aussage von Anscombes Quartett ist somit eindeutig: Auch wenn Statistiken gut geeignet sein können, Daten zusammenzufassen, ist es ebenso möglich, dass die Statistiken irreführend sind. Die kritische Leserin oder der aufmerksame Leser wird vielleicht bemerkt haben, dass die durch Ausreißer verursachten Probleme beim arithmetischen Mittel und der Standardabweichung zu erwarten sind. Die Probleme von Statistiken sind jedoch grundlegender und es gibt weitere Beispiele, die auch mehr statistische Marker berücksichtigen [Matejka & Fitzmaurice 2017].

4.3.2 Einzelne Merkmale

Eine grundlegende Betrachtung der Daten besteht in der Visualisierung einzelner Merkmale der Daten. Hierdurch kann man die Verteilung dieser Merkmale verstehen, ähnlich zur Beschreibung durch Statistiken. Hierzu schauen wir uns Histogramme, Densityplots, Rugs und Boxplots an.

Wir betrachten dieses Feature anhand von Daten über Hauspreise aus Boston, die 1978 veröffentlicht wurden, die wir im Folgenden einfach als Bostondaten bezeichnen werden.

```
# sklearn is a large machine learning library that we use
from sklearn import datasets
from textwrap import TextWrapper

# we use this wrapper to avoid printing lines that are too long because this
# should be readable as a book
# usually you would still just use print
wrapper = TextWrapper(width=65, replace_whitespace=False, break_long_words=False)
def wrap_print(string):
    for line in string.split('\n'):
        print('\n'.join(wrapper.wrap(line)))

boston = datasets.fetch_openml(data_id=531)
wrap_print(boston.DESCR)
```

****Author**:**

****Source**:** Unknown - Date unknown

****Please cite**:**

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. Used in Belsley, Kuh & Welsch, 'Regression diagnostics ...', Wiley, 1980. N.B. Various transformations are used in the table on pages 244-261 of the latter. Variables in order:

CRIM	per capita crime rate by town
ZN	proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS	proportion of non-retail business acres per town
CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX	nitric oxides concentration (parts per 10 million)
RM	average number of rooms per dwelling
AGE	proportion of owner-occupied units built prior to 1940
DIS	weighted distances to five Boston employment centres
RAD	index of accessibility to radial highways
TAX	full-value property-tax rate per \$10,000
PTRATIO	pupil-teacher ratio by town
B	$1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town
LSTAT	% lower status of the population
MEDV	Median value of owner-occupied homes in \$1000's

Information about the dataset

CLASSTYPE: numeric

CLASSINDEX: last

Downloaded from openml.org.

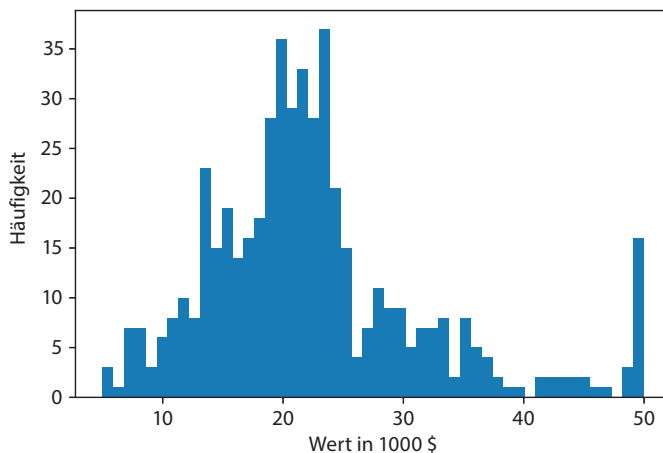
Die obige Beschreibung ist ein Beispiel für Metadaten, in diesem Fall die Dokumentation der Daten.

Bemerkung:

Die Bostondaten werden immer häufiger kritisiert und wurden zum Beispiel auch als Beispieldatensatz aus scikit-learn entfernt. Der Grund hierfür ist, dass die Daten aus ethischer Sicht hochproblematisch sind: Die Daten sind teilweise rassistisch, insbesondere das Merkmal **B**. In diesem Buch verwenden wir diese Daten dennoch weiter und nutzen den Datensatz, um nicht nur zu zeigen, wie man Daten analysieren kann, sondern auch als Beispiel, dass man ethische Aspekte von Daten und Modellen niemals vergessen sollte.

Wir werfen jetzt einen genaueren Blick auf das Merkmal **MEDV**, den Median des Werts der Eigenheime in 1000 Dollar. *Histogramme* sind eine einfache und oft effektive Art, etwas über die Verteilung von Daten zu lernen.

```
fig, ax = plt.subplots()
ax.hist(boston.target, bins=50)
ax.set_xlabel('Wert in 1000 $')
ax.set_ylabel('Häufigkeit')
plt.show()
```



Das Histogramm zeigt, wie häufig Werte vorkommen. Hierfür wird der Wertebereich in sogenannte *Bins* unterteilt. Der Begriff *Bin* ist vom englischen Wort für Gruppieren abgeleitet. Das Histogramm gibt an, wie viele Datenpunkte in jedem Bin liegen. Im obigen Beispiel haben wir 50 Bins. Das Histogramm verrät uns viel über die Daten.

- Die Daten zwischen 0 und 30 scheinen normalverteilt zu sein. Dies erkennt man daran, dass die Daten ungefähr eine *Glockenform* (engl. *bell-shape*) bilden, was für die Normalverteilung typisch ist. Der Mittelwert dieser Normalverteilung

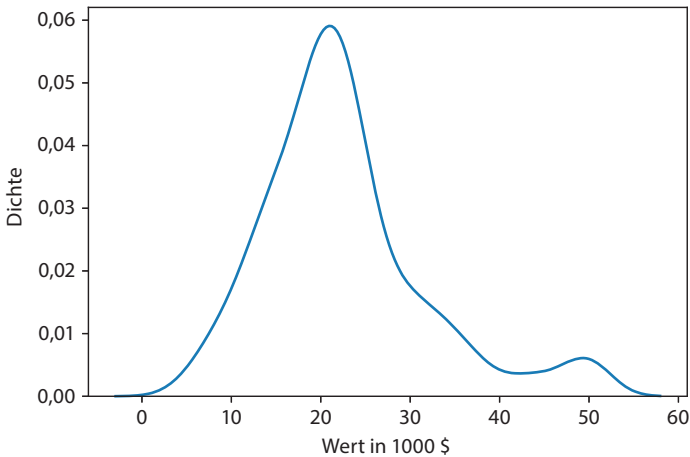
liegt ca. bei 22. Dies erkennt man am *Peak* in diesem Wertebereich. Die Standardabweichung kann man nicht genau ablesen, aber mithilfe der oben diskutierten 68-95-99-Regel grob abschätzen. Der Wert sollte sich zwischen 7 und 11 befinden.

- Neben dieser Normalverteilung gibt es noch viele Werte am rechten Rand der Grafik, also bei genau 50. Dies deutet darauf hin, dass es in Wirklichkeit vermutlich auch noch Werte oberhalb von 50 gibt, diese aber zusammengefasst wurden. Die eigentliche Bedeutung des Werts 50 sind somit nicht Häuser mit einem Wert von 50.000 Dollar, sondern Häuser, die mindestens 50.000 Dollar wert sind.
- Es gibt einen *Tail* auf der rechten Seite des Grafik, also Häuser, die teuer sind. Diese Häuser kommen zwar vor, lassen sich aber nicht mehr durch die Normalverteilung erklären. Solche Daten nennt man auch *Rechtsschief* (engl. *right skew*).

Wir können uns **MEDV** auch mithilfe von einem Densityplot anschauen.

```
import seaborn as sns # seaborn is a visualization library build on top of matplotliblib

fig, ax = plt.subplots()
sns.kdeplot(boston.target, ax=ax)
ax.set_xlabel('Wert in 1000 $')
ax.set_ylabel('Dichte')
plt.show()
```



Vereinfacht gesagt, zeigen Densityplots eine Schätzung der Dichtefunktion der Wahrscheinlichkeitsverteilung der Daten. Wir können uns das als eine Art kontinuierliches Histogramm vorstellen. Der Vorteil von Densityplots gegenüber Histogrammen ist, dass es oft einfacher ist, die Verteilung der Daten zu erkennen. Im obigen Beispiel treten die Glockenform und die Position des Peaks deutlicher hervor und man kann dadurch die Normalverteilung in der linken Hälfte des Plots besser er-

kennen. Densityplots haben jedoch auch einige Nachteile im Vergleich zu Histogrammen. Ein kleiner Nachteil ist, dass es schwierig ist, die Werte der y-Achse zu interpretieren. Während ein Histogramm eine klare Aussage über die Anzahl der Datenpunkte in einem Bin erlaubt, sieht man im Densityplot lediglich die *Dichte* als Wert zwischen 0,0 und 1,0. Die Dichte ist mehr oder weniger der Anteil der Daten, der an einem bestimmten Punkt auf der x-Achse zu erwarten ist.

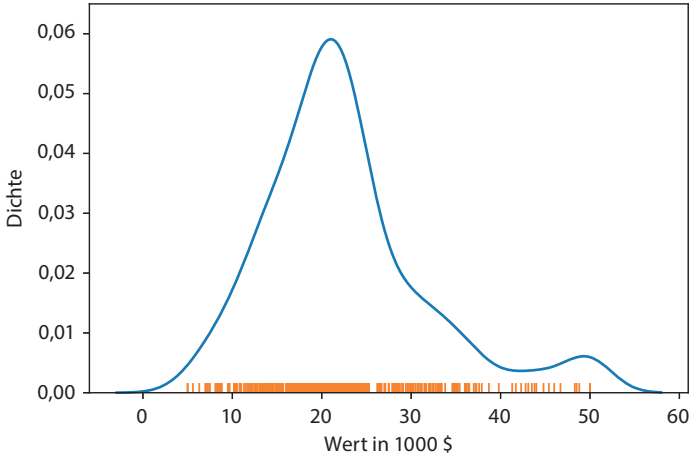
Der zweite Nachteil der Densityplots ist gravierender. Das Histogramm zeigt uns klar die vielen Datenpunkte mit dem Wert 50 und auch, dass es keine Datenpunkte mit einem höheren Wert gibt. Im Densityplot sieht man bei der 50 nur einen kleinen Peak, den man auch als weitere Normalverteilung mit dem Mittelwert 50 am rechten Rand der Daten interpretieren könnte. Das ist irreführend und versteckt die wahre Verteilung der Daten. Dieses Risiko lässt sich bei Densityplots auch nicht vermeiden, da es mit der Technik, wie die Dichte geschätzt wird, zusammenhängt. Solche Fehlinterpretationen der Daten sind dann wahrscheinlich, wenn die Daten nicht sehr dicht verteilt sind (großer Datenbereich mit verhältnismäßig wenig Datenpunkten), sowie an den Grenzen der beobachteten Daten. Hier geht die geschätzte Dichtefunktion automatisch über den beobachteten Datenbereich hinaus, was eventuell aber der Interpretation eines Merkmals widerspricht. Man erkennt im obigen Plot zum Beispiel auch, dass die Werte kleiner 0 nicht eine Dichte von 0 haben, was bedeuten würde, dass einige Eigenheime einen negativen Wert hätten.

Bemerkung:

Densityplots werden mithilfe von *Kernel Density Functions* (KDE) erstellt, in der Regel mittels der Dichtefunktion der Normalverteilung. Diese Dichtefunktion wird dann an jedem Datenpunkt mit *Skalierungs-* und *Bandbreitenparametern* geschätzt. Anschließend werden alle diese geschätzten Dichtefunktionen aufaddiert, um den Densityplot zu erstellen. Hierdurch kann man auch das Verhalten an den Grenzen erklären: Die Dichtefunktion, die an den äußersten Punkten der Daten geschätzt wird, geht aufgrund der Symmetrie der Normalverteilung automatisch über den Datenbereich hinaus. Die Skalierung der Schätzung führt dazu, dass ein seltsames Verhalten, wie viele gleiche Werte an einer Grenze in der aufsummierten Dichtefunktion, nicht stark auffällt.

Ein einfaches Mittel, dieses Problem zu vermeiden, besteht darin, einen *Rug* (dt. Teppich) anzuzeigen. Der *Rug* hat seinem Namen daher, weil er wie ein Teppich unter den Plot gelegt wird.

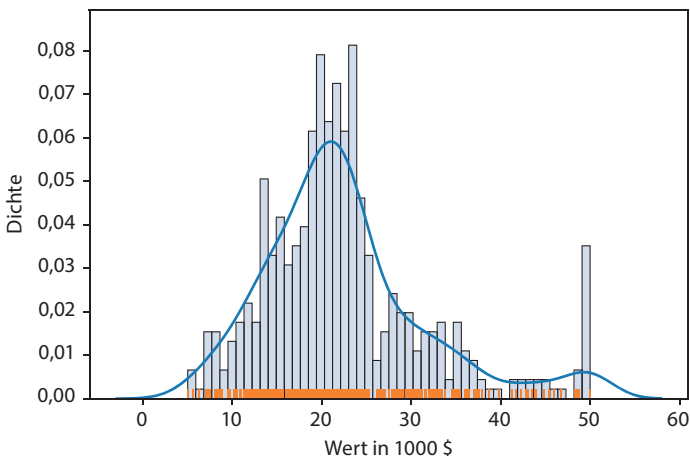
```
fig, ax = plt.subplots()
sns.kdeplot(boston.target, ax=ax)
sns.rugplot(boston.target, ax=ax)
ax.set_xlabel('Wert in 1000 $')
ax.set_ylabel('Dichte')
plt.show()
```




Der Rug zeigt, wo sich wirklich Datenpunkte befinden. In Kombination mit dem Densityplot können wir also sehen, dass es keine Datenpunkte kleiner als 5 oder größer 50 gibt. Wir erkennen auch, dass die Region zwischen 38 und 50 nur relativ dünn besiedelt ist, wobei die Punkte in diesem Bereich etwa gleichverteilt zu sein scheinen. Hierzu passt, dass der Densityplot in diesem Bereich fast parallel zur x-Achse verläuft. Dies zeigt, dass der Rug hilfreich ist, um weitere Erkenntnisse über die Daten zu gewinnen und Fehlinterpretationen zu vermeiden.

Man kann auch einfach alle oben betrachteten Ansätze kombinieren und Histogramm, Densityplot und Rug zusammen visualisieren.

```
fig, ax = plt.subplots()  
# stat='density' scales the plot to match the y-axis of the density plot  
# alpha=0.2 makes the bars transparent for better readability  
sns.histplot(boston.target, stat='density', alpha=0.2, bins=50, ax=ax)  
sns.kdeplot(boston.target, ax=ax)  
sns.rugplot(boston.target, ax=ax)  
ax.set_xlabel('Wert in 1000 $')  
ax.set_ylabel('Dichte')  
plt.show()
```



Diese Leseprobe haben Sie beim
 **edv-buchversand.de** heruntergeladen.
Das Buch können Sie online in unserem
Shop bestellen.

[Hier zum Shop](#)