Big Data

by Woche 12

```
# Hinweis: In diesem Kapitel wird zwar beispielhaft Code gezeigt, aber nie ausgeführt.
```

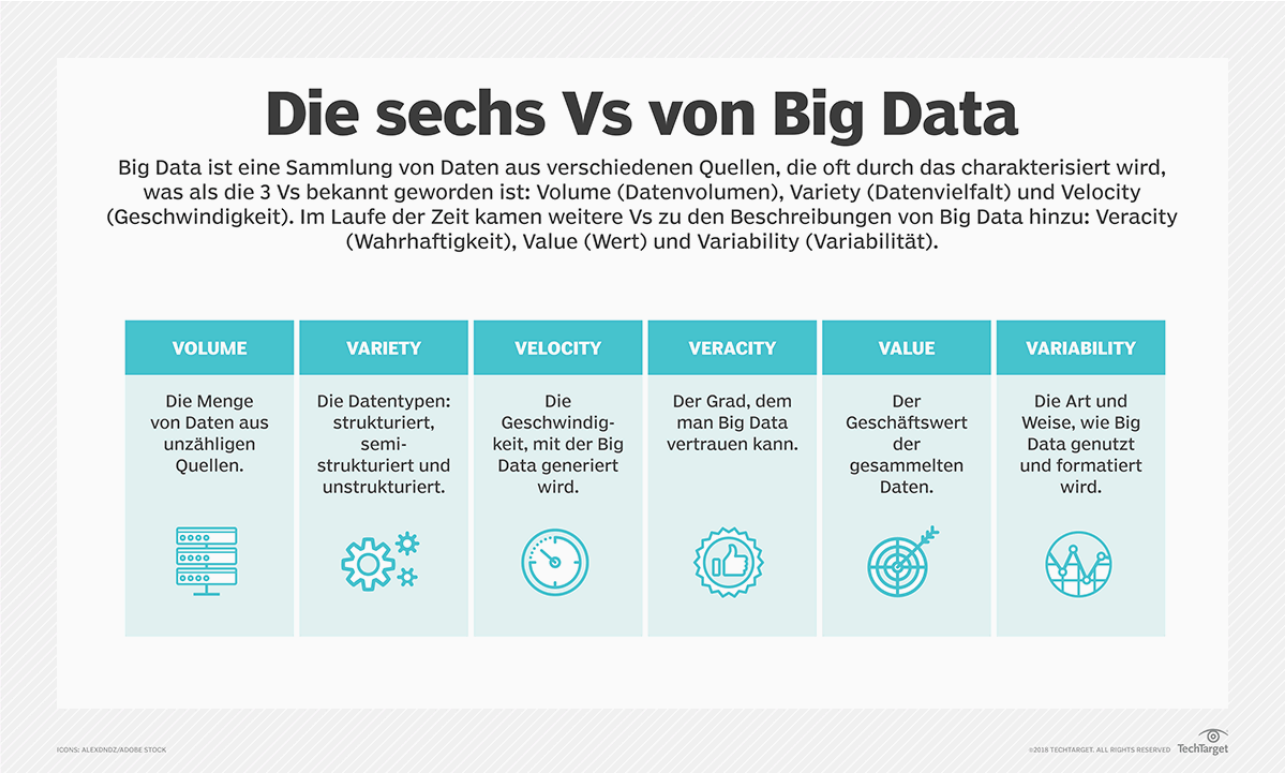
In den vorherigen Kapiteln haben wir gelernt, wie man Daten über APIs abruft und in SQL-Datenbanken speichert. Doch was passiert, wenn die Datenmengen so groß werden, dass z.B. unsere bewährten Pandas-DataFrames an ihre Grenzen stoßen? Wenn Netflix täglich das Verhalten von über 200 Millionen Abonnenten analysiert oder das CERN bei seinen Teilchenexperimenten etwa 1 Petabyte Daten pro Sekunde generiert, reichen die Methoden, die wir bisher kennengelernt haben, nicht mehr aus.

An dieser Stelle kommt **Big Data** ins Spiel - Technologien und Methoden zur Verarbeitung von Datenmengen, die zu groß, zu schnell generiert oder zu komplex für traditionelle Verfahren sind.

Was ist Big Data?

Der Begriff “Big Data” hat sich seit den frühen 2000er Jahren entwickelt und wird typischerweise durch die “V’s” charakterisiert:

- **Volume (Volumen)**: Riesige Datenmengen, oft im Tera- bis Petabyte-Bereich
- **Variety (Vielfalt)**: Unterschiedliche Datentypen und -quellen (strukturiert, semi-strukturiert, unstrukturiert)
- **Velocity (Geschwindigkeit)**: Hohe Geschwindigkeit, mit der neue Daten erzeugt und verarbeitet werden müssen
- **Veracity (Wahrhaftigkeit)**: Herausforderungen bei Datenqualität und -zuverlässigkeit
- **Value (Wert)**: Die Notwendigkeit, aus großen Datenmengen wertvolle Erkenntnisse zu gewinnen
- **Variability (Variabilität)**: Nutzungsmöglichkeiten der Daten



_Quelle: ComputeryWeekly.de

Wo ist Big Data relevant?

Big Data-Technologien sind in zahlreichen Bereichen unverzichtbar geworden:

- **E-Commerce und Einzelhandel**: Amazon analysiert Kundenverhalten für personalisierte Empfehlungen
- **Streaming-Dienste**: Netflix nutzt Sehverhalten für Content-Entscheidungen
- **Finanzwesen**: Banken erkennen Betrug in Echtzeit bei Millionen von Transaktionen
- **Gesundheitswesen**: Genomanalyse und personalisierte Medizin
- **Smart Cities**: Verkehrssteuerung durch Echtzeitanalyse von Sensordaten
- **Wissenschaft**: Klimamodellierung, Astronomie, Teilchenphysik

Diese Anwendungen sind ohne Big Data-Technologien schlichtweg nicht möglich. Konkrete Beispiele:

- Das CERN (Europäische Organisation für Kernforschung) generiert durch seine Teilchenexperimente etwa 1 Petabyte (1.000 Terabyte) Daten pro Sekunde. Selbst nach Filterung werden immer noch etwa 25 Petabyte pro Jahr gespeichert - eine Datenmenge, die mit traditionellen Methoden nicht zu bewältigen wäre.
- Ein Online-Händler wie Amazon verarbeitet nicht nur Transaktionsdaten, sondern auch Klickverhalten, Suchhistorien, Produktbewertungen, Lagerbestände in Echtzeit, Lieferdaten und mehr - alles gleichzeitig und in riesigen Mengen.
- Netflix analysiert das Sehverhalten von über 200 Millionen Abonnenten, um personalisierte Empfehlungen zu geben und Entscheidungen über neue Inhalte zu treffen.

Grundlegende Big Data-Prinzipien

Big Data erfordert ein fundamental anderes Denken als traditionelle Datenanalyse:

1. Verteilte statt zentraler Verarbeitung

Bei traditioneller Datenanalyse laden wir einen Datensatz vollständig in den Arbeitsspeicher eines einzelnen Computers (wie wir es in diesem Kurs bisher getan haben). Bei Big Data entstehen zwei Arten von Problemen:

Problem 1: Speicherplatz-Limitierung Die Daten sind schlichtweg zu groß für den Arbeitsspeicher eines Computers. Hier hilft nur verteilte Speicherung auf mehrere Computer.

Problem 2: Verarbeitungszeit-Limitierung Die Daten passen in den Speicher, aber die Berechnungen dauern zu lange. Hier kann man zunächst mehrere Prozessorkerne desselben Computers nutzen (moderne Computer haben 4, 8 oder mehr Kerne). Tools wie Dask oder multiprocessing ermöglichen das in Python. Bei echtem Big Data reichen aber auch mehrere Kerne nicht aus - dann werden Daten und Berechnungen auf viele Computer (Cluster) verteilt.

2. Parallelisierung

Aufgaben werden in kleinere Teilaufgaben zerlegt, die parallel auf verschiedenen Rechnern (oder Kernen) ausgeführt werden können. Das klingt zunächst abstrakt, wird aber mit einem konkreten Beispiel klarer:

Beispiel: Durchschnittspreis berechnen

Stellen wir uns vor, wir haben 1 Million Verkaufsdatensätze und wollen den Durchschnittspreis berechnen:

Traditionell (sequenziell):

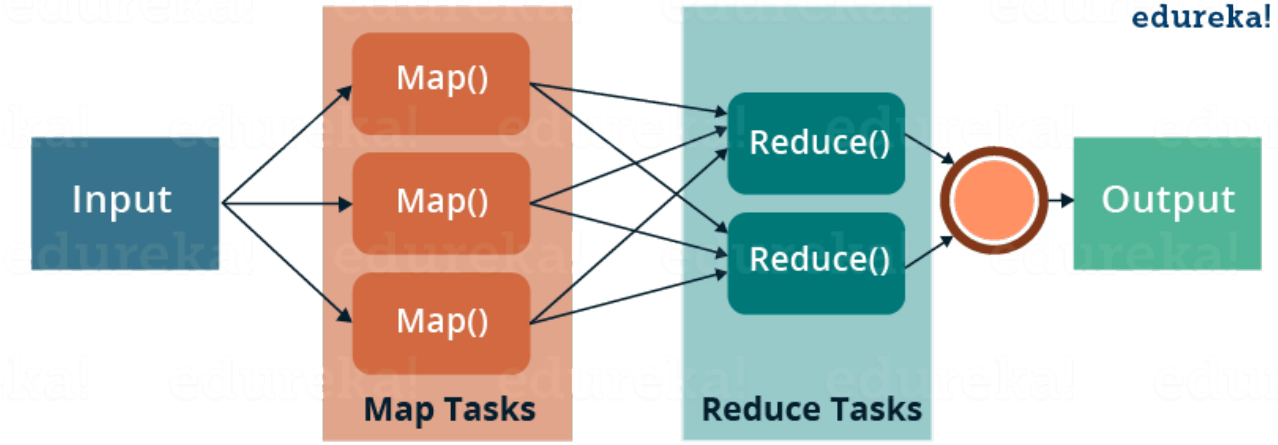
1. Alle 1 Million Datensätze laden
2. Alle Preise addieren

3. Durch die Anzahl teilen

Mit Parallelisierung:

1. **Aufteilen**: Die 1 Million Datensätze auf 10 Computer verteilen (je 100.000)
2. **Map-Phase**: Jeder Computer berechnet für seine 100.000 Datensätze:
   - Summe der Preise in seinem Teil
   - Anzahl der Datensätze in seinem Teil
3. **Reduce-Phase**: Ein Computer sammelt alle Teilergebnisse:
   - Alle Teilsummen addieren = Gesamtsumme
   - Alle Teilanzahlen addieren = Gesamtanzahl
   - Gesamtsumme ÷ Gesamtanzahl = Durchschnittspreis

Das Ergebnis ist identisch, aber die Berechnung war 10x schneller, weil 10 Computer parallel gearbeitet haben. Dieses Map-Reduce-Prinzip funktioniert für viele Datenoperationen: Gruppierungen, Aggregationen, Filterungen usw.



Quelle: edureka.co

3. Fehlertoleranz

In einem Cluster mit hunderten Rechnern sind Ausfälle normal. Das System muss automatisch damit umgehen können, ohne dass die gesamte Berechnung neu gestartet werden muss.

Das ist ein fundamentaler Unterschied zu unserem gewohnten Python-Code: Wenn in unserem Jupyter Notebook ein Fehler auftritt, stoppt das gesamte Programm und wir müssen von vorne anfangen. Bei Big Data-Systemen wäre das katastrophal - wenn von 100 Computern einer ausfällt, sollen die anderen 99 weiterlaufen und nur der ausgefallene Computer wird durch einen neuen ersetzt. Die bis dahin berechneten Ergebnisse bleiben erhalten.

4. Skalierbarkeit

Systeme müssen sowohl horizontal (mehr Maschinen hinzufügen) als auch vertikal (leistungsfähigere Maschinen verwenden) skalierbar sein.

5. Lazy Evaluation

Bei traditioneller Datenverarbeitung mit Pandas werden Operationen sofort ausgeführt:

```
# Pandas: Jede Zeile wird sofort ausgeführt
df1 = df[df['price'] > 100]   # Wird sofort ausgeführt
df2 = df1.groupby('category') # Wird sofort ausgeführt
result = df2.mean()           # Wird sofort ausgeführt
```

Big-Data-Frameworks wie Spark funktionieren anders. Sie erstellen zunächst einen "Ausführungsplan" und optimieren diesen, bevor sie ihn ausführen:

```
# Spark: Erstellt zunächst nur einen Plan
df1 = df.filter(df['price'] > 100) # Noch keine Ausführung!
df2 = df1.groupBy('category')      # Noch keine Ausführung!
result = df2.avg()                 # Noch keine Ausführung!

# Erst hier wird der optimierte Plan ausgeführt:
result.show()  # JETZT wird alles auf einmal optimal ausgeführt
```

Warum ist das vorteilhaft? Das System kann den gesamten Plan betrachten und optimieren: "Ah, wir filtern zuerst und gruppieren dann - dann können wir die Filterung direkt beim Einlesen machen und müssen weniger Daten übertragen." Solche Optimierungen sind bei sofortiger Ausführung nicht möglich.

Big Data Technologien und Frameworks

Um die Herausforderungen von Big Data zu bewältigen, wurden spezialisierte Technologien und Frameworks entwickelt:

Verteilte Speicherung und Verarbeitung

Apache Hadoop war eines der ersten großen Frameworks für verteilte Speicherung und Verarbeitung großer Datensätze. Das System funktioniert nach dem Prinzip "Bring the computation to the data" - statt riesige Dateien zwischen Servern zu kopieren, wird der Code dorthin gebracht, wo die Daten bereits liegen. Für uns als Python-Nutzer bedeutet das: Hadoop läuft zwar auf vielen Servern im Hintergrund, wir schreiben aber weiterhin Python-Code.

Apache Spark hat sich als modernere Alternative zu Hadoop etabliert und kann durch clevere Speichernutzung bis zu 100 Mal schneller sein. Das Besondere für Data Scientists ist das Python-Modul `pyspark`, das eine sehr ähnliche Syntax wie Pandas bietet, aber die Berechnungen auf viele Server verteilt:

```
import pyspark.sql as spark
df = spark.read.csv("riesige_datei.csv")
result = df.groupBy("kategorie").count()
```

Für Echtzeit-Datenströme wie Live-Tweets oder kontinuierliche Sensor-Daten kommt **Apache Kafka** zum Einsatz. Diese Streaming-Plattform verarbeitet Daten, während sie entstehen, anstatt sie erst zu sammeln und später zu analysieren. Auch hier gibt es Python-Libraries wie `kafka-python`, die den Zugriff auf diese Datenströme ermöglichen

Speicherlösungen für Big Data

Während unsere gewohnten SQL-Datenbanken bei sehr großen Datenmengen an ihre Grenzen stoßen, wurden spezialisierte NoSQL-Datenbanken entwickelt, die anders funktionieren als die relationalen Datenbanken, die wir kennengelernt haben. **MongoDB** beispielsweise speichert Daten in JSON-ähnlichen Dokumenten, was besonders bei flexiblen, sich ändernden Datenstrukturen vorteilhaft ist. **Cassandra** hingegen ist für das sehr schnelle Schreiben von Zeitreihendaten optimiert, während **Neo4j** sich auf vernetzte Daten wie soziale Netzwerke spezialisiert hat.

Das Gute für uns als Python-Anwender: Alle diese Systeme haben eigene Python-Module wie `pymongo` oder `cassandra-driver`, sodass wir weiterhin in der gewohnten Sprache arbeiten können, auch wenn im Hintergrund völlig andere Datenbank-Technologien laufen.

Cloud vs. On-Premise

Grundsätzlich kann man Big Data-Technologien auf zwei Arten nutzen. Bei der On-Premise-Variante installiert man Hadoop, Spark und andere Tools auf eigenen Servern. Das bedeutet jedoch einen sehr hohen Aufwand für Installation, Konfiguration und Wartung und ist meist nur für große Unternehmen mit entsprechenden IT-Teams praktikabel.

Die moderne Alternative sind Cloud-Services, die von großen Anbietern wie Amazon Web Services (AWS), Google Cloud oder Microsoft Azure angeboten werden. AWS bietet beispielsweise EMR (Elastic MapReduce) für managed Spark-Cluster, Simple Storage Service für die Dateispeicherung und Redshift als Data Warehouse. Google Cloud stellt entsprechend Dataflow, BigQuery und Cloud Storage zur Verfügung, während Microsoft Azure mit HDInsight und Synapse Analytics aufwartet.

Für einzelne Data Scientists sind Cloud-Services deutlich attraktiver, weil sie viel einfacher zu nutzen sind und man weiterhin Python-Code schreibt, der dann auf Cloud-Servern ausgeführt wird. Google BigQuery zeigt das sehr schön: Es gibt ein Python-Modul `google-cloud-bigquery`, mit dem man SQL-Abfragen ausführen und die Ergebnisse direkt als Pandas DataFrame erhalten kann:

```
# BigQuery mit Python nutzen:
from google.cloud import bigquery
client = bigquery.Client()
query = "SELECT kategorie, COUNT(*) FROM meine_tabelle GROUP BY kategorie"
result = client.query(query).to_dataframe()
```

Der große Vorteil der Cloud liegt darin, dass man keine eigene Server-Infrastruktur aufbauen muss und nur für die tatsächlich genutzte Rechenzeit zahlt. Während ein eigenes Hadoop-Cluster auch läuft wenn man es nicht nutzt, kann man Cloud-Ressourcen bei Bedarf starten und wieder stoppen.

Der Unterschied in der praktischen Arbeit

Als Data Scientist ist der Umgang mit Big Data in vielerlei Hinsicht anders als mit kleineren Datensätzen:

1. **Denkweise ändern**: Man nicht mehr den gesamten Datensatz auf einmal sehen oder erkunden - stattdessen musst mit Stichproben gearbeitet oder gezielt Aggregationen und Zusammenfassungen angefordert werden.
2. **Effizienz wird wichtiger**: Bei kleinen Datensätzen ist es oft egal, ob eine Operation 0,1 oder 10 Sekunden dauert. Bei Big Data kann eine ineffiziente Operation Stunden oder Tage dauern, oder die Echtzeitfähigkeit des Systems gefährden.
3. **Iteratives Arbeiten**: Man arbeitet oft mit einer kleinen Stichprobe, um einen Ansatz zu testen, bevor du er auf den gesamten Datensatz angewendet wird.

Beispiel: Datenverarbeitung mit PySpark

Ein einfaches Beispiel für die Verwendung von PySpark zur Analyse eines großen Datensatzes:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg, count

# Spark-Session erstellen - der Einstiegspunkt für alle Spark-Funktionalitäten
spark = SparkSession.builder.appName("BigDataAnalyse").getOrCreate()

# Daten laden - beachte, dass keine tatsächliche Datenverarbeitung stattfindet,
```

```
# bis eine Aktion ausgeführt wird (Lazy Evaluation)
data = spark.read.csv("hdfs://very_large_dataset.csv", header=True, inferSchema=True)

# Transformationen definieren
# Diese werden nicht sofort ausgeführt, sondern als Plan gespeichert
grouped_data = data.groupBy("kategorie").agg(
    count("*").alias("anzahl"),
    avg("preis").alias("durchschnittspreis")
)

# Aktion ausführen - erst jetzt wird der Ausführungsplan umgesetzt
# und die Berechnung durchgeführt
result = grouped_data.orderBy("durchschnittspreis", ascending=False).collect()

# Ergebnisse anzeigen
for row in result:
    print(f"Kategorie: {row['kategorie']}, Anzahl: {row['anzahl']}, Durchschnittspreis:
{row['durchschnittspreis']:.2f}")

# Spark-Session beenden
spark.stop()
```

In diesem Beispiel würden die tatsächlichen Berechnungen erst bei `collect()` ausgeführt werden, nachdem Spark die Möglichkeit hatte, den Ausführungsplan zu optimieren.

Herausforderungen bei Big Data

Die Arbeit mit Big Data bringt spezifische Herausforderungen mit sich:

- **Infrastrukturmanagement**: Aufbau und Wartung einer skalierbaren Infrastruktur erfordern spezielle Kenntnisse. Diese Aufgabe wird seltener von Data Scientists übernommen, sondern von DevOps- oder Data-Engineering-Teams.
- **Kosten**: Hardware, Speicher und Cloud-Ressourcen für Big Data können teuer sein
- **Komplexität**: Verteilte Systeme sind inhärent komplexer zu verstehen und zu debuggen
- **Sicherheit und Datenschutz**: Schutz sensibler Daten in verteilten Umgebungen ist anspruchsvoll
- **Datenqualität**: Bei großen Datenmengen werden Probleme mit der Datenqualität verstärkt und bleiben leichter unentdeckt

Wann ist Big Data sinnvoll?

Nicht jedes Problem erfordert Big-Data-Technologien. Vor der Implementierung einer Big-Data-Lösung sollten folgende Fragen gestellt werden:

1. **Ist der Datensatz wirklich zu groß für traditionelle Methoden?** Moderne Computer können Datensätze mit mehreren Gigabyte problemlos verarbeiten.
2. **Kannst du das Problem durch Sampling lösen?** Oft kann eine repräsentative Stichprobe ausreichen.
3. **Rechtfertigt der Mehrwert den Aufwand?** Big-Data-Infrastrukturen sind komplex und teuer.

Als Faustregel gilt: Wenn ein Datensatz nicht in den RAM eines leistungsstarken Servers passt oder wenn Echtzeit-Verarbeitung für kontinuierliche Datenströme gefordert sind, könnte der Einsatz von Big-Data-Technologien gefordert sein.

Fazit

Big Data erweitert die Möglichkeiten der Datenanalyse erheblich, bringt aber auch neue Komplexität mit sich. Als Data Scientist ist es wichtig, sowohl traditionelle als auch Big-Data-Methoden zu kennen und je nach Problemstellung die geeignete Herangehensweise wählen zu können.

Die gute Nachricht: Viele der statistischen und analytischen Konzepte, die wir in diesem Kurs kennengelernt haben, gelten auch im Big-Data-Kontext - nur die technische Umsetzung unterscheidet sich. Wer die statistischen Zusammenhänge versteht, und weiß, wie man Daten mit Pandas analysiert, kann auch die Konzepte von Spark DataFrames schnell erfassen.

Hier sei schließlich noch darauf hingewiesen, dass “Big Data” neben “KI”, und “Machine Learning” ein weiteres Buzz-Word ist, das in der Branche gerne verwendet wird, um Projekte zu verkaufen oder Investoren zu beeindrucken. Es ist wichtig, kritisch zu hinterfragen, ob Big Data wirklich notwendig ist oder ob die Probleme auch mit traditionellen Methoden gelöst werden können. Selbst im Wikipedia-Artikel zu Big Data steht folgender Abschnitt:

> Der Begriff „Big Data“ wird gelegentlich auch dann verwendet, wenn Daten weder groß noch komplex sind oder sich nicht schnell ändern oder mit herkömmlichen Techniken problemlos verarbeitet werden können.[9] Die zunehmende Aufweichung des Begriffs führt nach Meinung einiger Beobachter dazu, dass er immer mehr ein aussageloser Marketingbegriff werde und vielen Prognosen zufolge innerhalb der nächsten Jahre eine starke Abwertung erfahre („Tal der Enttäuschungen“ im Hype-Zyklus).

Weitere Ressourcen

- Intro to Big Data: Crash Course Statistics #38
- What is Big Data? - Computerphile
- How to work with big data files (5gb+) in Python Pandas!

Optional:

- Apache Spark Documentation
- Hadoop Documentation
- Designing Data-Intensive Applications - Ein Standardwerk zu verteilten Systemen und Big Data. ISBN 9781449373320