HTTP APIs als Datenquelle

by Woche 11

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
np.random.seed(1) # damit die Zufallszahlen reproduzierbar sind
```

Bisher lagen uns die Daten für unsere Analyse immer als CSV-Dateien vor. Tatsächlich kommt es jedoch oft vor, dass die Daten zuerst aus anderen Systemen extrahiert werden müssen.

Deshalb machen wir an dieser Stelle einen kleinen Exkurs in die Welt der APIs (Application Programming Interfaces) - Schnittstellen, über die verschiedene Computersysteme miteinander kommunizieren können. Es ist ein bisschen wie beim Bestellen im Restaurant: Du musst nicht wissen, wie das Essen zubereitet wird, aber du musst wissen, wie du es bestellen kannst.

Streng genommen gehört dieses Thema eher zum Bereich Data Engineering als zur reinen Datenanalyse. Dennoch ist es wertvolles Wissen für Data Scientists, da es dir Zugang zu einer riesigen Vielfalt an Datenquellen eröffnet - von Wetterdaten über Börsenkurse bis hin zu Social Media Trends.

Webservices und APIs

Bevor wir uns APIs ansehen, sollten wir verstehen, wie Computer überhaupt miteinander kommunizieren. Damit du das hier überhaupt lesen kannst, musste dein Computer die Inhalte dieser Website von einem Server anfordern. Und nachdem der die Anforderung erhalten hat, musste er die Informationen auch genau an deinen Computer senden. Für solche Kommunikationen zwischen Computersystemen in Rechnernetzen (wie dem Internet) kommt eine Vielzahl verschiedener Standards zum Einsatz. Dabei gibt es sowohl solche, die für die Kommunikation zwischen Diensten verwendet werden, als auch jene, die für die Kommunikation mit Endnutzern im Browser gedacht sind.

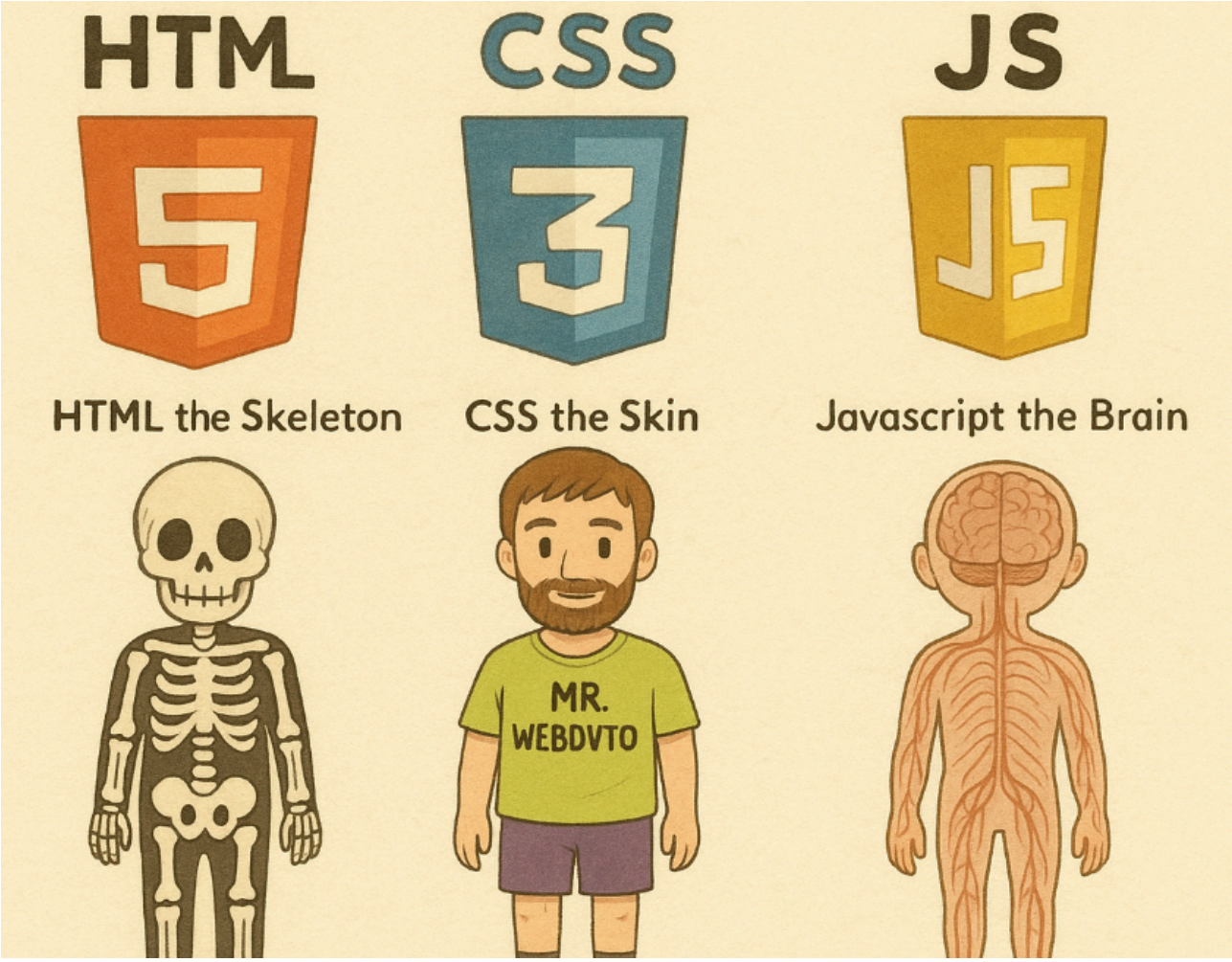
Die wichtigsten Merkmale dieser Kommunikationsstandards sind:

- **Interoperabilität**: Sie funktionieren unabhängig von der verwendeten Programmiersprache oder Plattform
- **Lose Kopplung**: Dienste können unabhängig voneinander entwickelt und gewartet werden
- **Skalierbarkeit**: Leichte Erweiterbarkeit bei steigenden Anforderungen

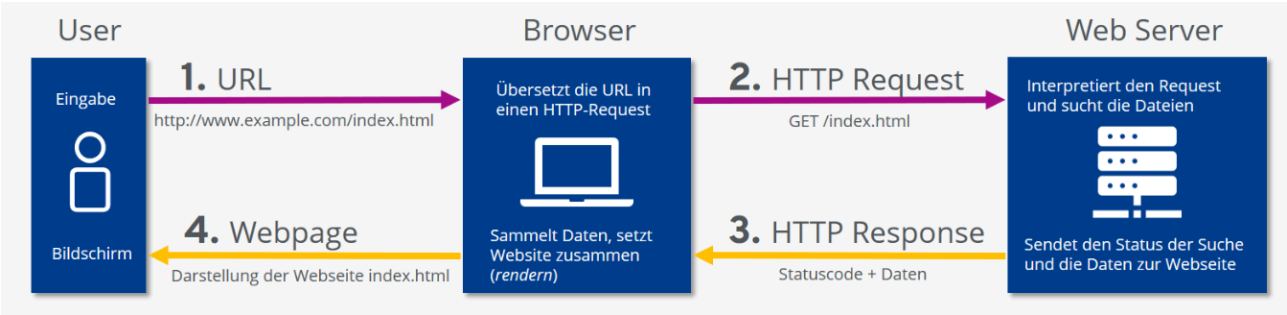
Moderne Websites funktionieren mit dem Prinzip, dass der Browser sich zunächst die Dateien für die Darstellung und Interaktivität der Seite herunterlädt:

- HTML für die Struktur des Dokuments
- CSS für die Darstellung und
- Javascript für Nutzerinteraktion und Manipulation des Dokuments

und sich dann weitere Daten dann über Schnittstellen anfragt.



Für die Kommunikation von Server zu Browser (und umgekehrt) kommt in erster Linie das **Hypertext Transfer Protocol (HTTP)** zum Einsatz. Es ist im Prinzip die Sprache des Internets mit der sich Computer untereinander verständigen. Es definiert den Aufbau von Nachrichten, Statuscodes und die grundlegenden Mechanismen für den Datenaustausch im Web.



Quelle: IONOS

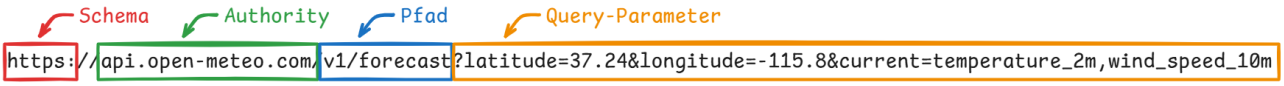
Wenn du eine Website besuchst, passiert folgendes:

1. Du gibst die URL in den Browser ein
2. Dein Browser schickt eine HTTP-Anfrage an den Server: “Hallo, ich hätte gerne die Webseite www.beispiel.de”
3. Der Server antwortet: “Hier ist die Webseite” und schickt die entsprechenden Dateien zurück
4. Dein Browser stellt diese Dateien dar - das ist die Webseite, die du siehst

Websites wie genau die, die du gerade liest, sind für Menschen gemacht und sehen im Idealfall übersichtlich und einladend aus - sie enthalten Text, Bilder, klickbare Buttons und mehr. HTTP ist aber als Schnittstelle nicht darauf beschränkt nur Webseiten zu übertragen. Es kann im Prinzip

alles mögliche übertragen werden und eigentlich ist das auch klar, wenn man z.B. an Videostreaming a la Netflix denkt.

Und so ist es eben auch möglich Daten über HTTP zu übertragen. Hier eine Beispiel-URL, die aktuelle Wetterdaten von Open-Meteo abruft (klicken auf Bild öffnet die Seite):



Wir sehen, dass eine **HTTP API** (Application Programming Interface, dt. Anwendungsprogrammierschnittstelle) genau wie eine Website beim Browsen über eine **URL** (Uniform Resource Locator), bzw. genauer eine **URI** (Uniform Resource Identifier) angesprochen wird. Diese besteht hierbei aus 4 Teilen:

1. **Schema** `https:` - definiert den Kontext oder das Protokoll der Ressource. In diesem Fall HTTPS, die verschlüsselte Form von HTTP
2. **Authority** `api.open-meteo.com` - identifiziert den Server, der die Ressource hostet, oft ein Domainname oder eine IP-Adresse
3. **Pfad** `/v1/forecast` - spezifiziert den Pfad zur gesuchten Ressource auf dem Server, oft hierarchisch organisiert
4. **Query-Parameter** `?latitude=37.24&longitude=-115.8&current=temperature_2m,wind_speed_10m` - optionale Parameter, die mit der Ressourcenanfrage übergeben werden, im Format `?name=wert&name2=wert2`

Die Parameter sind in diesem Fall also Längen- und Breitengrad, sowie die gewünschten Daten:

- `latitude=37.24`: “Ich möchte Daten für Breitengrad 37.24”
- `longitude=-115.8`: “und Längengrad −115.8”
- `current=temperature_2m,wind_speed_10m`: “Gib mir die aktuelle Temperatur und Windgeschwindigkeit”

Wie man in der API-Dokumentation von open-meteo sieht, sind noch etliche weitere Parameter möglich.

JSON

Beim Klicken auf den Link, sollte folgende Antwort zurückgegeben werden:

```
{'latitude': 37.24434, 'longitude': -115.787384, 'generationtime_ms':
0.026345252990722656, 'utc_offset_seconds': 0, 'timezone': 'GMT',
'timezone_abbreviation': 'GMT', 'elevation': 1355.0, 'current_units': {'time':
'iso8601', 'interval': 'seconds', 'temperature_2m': '°C', 'wind_speed_10m': 'km/h'},
'current': {'time': '2025-08-19T09:45', 'interval': 900, 'temperature_2m': 18.4,
'wind_speed_10m': 3.2}}
```

Und das ist vielleicht keine schöne Tabelle, aber es sind definitiv strukturierte Daten, die ein Computer perfekt verstehen kann. Genau das macht APIs so wertvoll für Data Scientists!

Genauer gesagt liegt die Antwort im JSON-Format (JavaScript Object Notation) vor. Wie der Name vermuten lässt, wurde es ursprünglich für die Übertragung und Speicherung von Datenobjekten mit Javascript konzipiert. Aufgrund seiner Flexibilität, Lesbarkeit und weiten Verbreitung wird es jedoch auch in anderen Programmiersprachen oft verwendet. Es ist ja auch im Grunde wie ein Python-Dictionary, nur als Text geschrieben. Die geschweiften Klammern `{}` bedeuten “das gehört zusammen”, und die Doppelpunkte `:` verbinden Namen mit Werten.

Als weiterer Hinweis darauf wie verbreitet JSON ist seien hier direkt drei Ansätze gezeigt, um die Daten in Python zu laden:

```
url = "https://api.open-meteo.com/v1/forecast?latitude=37.24&longitude=-115.8&current=
temperature_2m,wind_speed_10m&past_days=14"
```

```
import json
import urllib.request

with urllib.request.urlopen(url) as response:
    json.load(response)
```

```
{'latitude': 37.24434, 'longitude': -115.787384, 'generationtime_ms':
0.027298927307128906, 'utc_offset_seconds': 0, 'timezone': 'GMT',
'timezone_abbreviation': 'GMT', 'elevation': 1355.0, 'current_units': {'time':
'iso8601', 'interval': 'seconds', 'temperature_2m': '°C', 'wind_speed_10m': 'km/h'},
'current': {'time': '2025-08-19T09:45', 'interval': 900, 'temperature_2m': 18.4,
'wind_speed_10m': 3.2}}
```

```
import requests

requests.get(url).json()
```

```
{'latitude': 37.24434, 'longitude': -115.787384, 'generationtime_ms':
0.026226043701171875, 'utc_offset_seconds': 0, 'timezone': 'GMT',
'timezone_abbreviation': 'GMT', 'elevation': 1355.0, 'current_units': {'time':
'iso8601', 'interval': 'seconds', 'temperature_2m': '°C', 'wind_speed_10m': 'km/h'},
'current': {'time': '2025-08-19T09:45', 'interval': 900, 'temperature_2m': 18.4,
'wind_speed_10m': 3.2}}
```

```
pd.read_json(url)
```

```
               latitude   longitude  ...  current_units           current
time           37.24434 -115.787384  ...         iso8601  2025-08-19T09:45
interval       37.24434 -115.787384  ...         seconds               900
temperature_2m 37.24434 -115.787384  ...              °C              18.4
wind_speed_10m 37.24434 -115.787384  ...           km/h               3.2

[4 rows x 9 columns]
```

Praktischerweise versucht Pandas (`pd.read_json()`) direkt die Daten in einen DataFrame zu laden.

Tatsächlich bietet Open-Meteo auch eine CSV-API an, die die Daten im CSV-Format zurückgibt. Diese kann mit dem gleichen Link wie oben aufgerufen werden, nur dass wir den Parameter `format=csv` hinzufügen:

```
csv_url = "https://api.open-meteo.com/v1/forecast?latitude=37.24&longitude=-115.8&
current=temperature_2m,wind_speed_10m&past_days=14&format=csv"
pd.read_csv(csv_url)
```

```
           latitude            longitude  ... timezone  timezone_abbreviation
0          37.24434          -115.787384  ...      GMT                    GMT
1              time  temperature_2m (°C)  ...      NaN                    NaN
2  2025-08-19T09:45                 18.4  ...      NaN                    NaN

[3 rows x 6 columns]
```

Wenn man sich vergegenwärtigt was wir gerade gelernt haben, werden folgende Vorteile von APIs zur Datenbeschaffung deutlich:

- **Flexibilität**: APIs bieten oft eine Vielzahl von Endpunkten und Parametern, um genau die Daten abzurufen, die benötigt werden. Das ist viel flexibler als eine CSV-Datei, die nur eine bestimmte Struktur hat.
- **Echtzeitdaten**: APIs ermöglichen den Zugriff auf aktuelle Daten, die sich ständig ändern können. Das ist besonders nützlich für Anwendungen, die Echtzeitinformationen benötigen, wie Wetterdaten oder Finanzmarktdaten.
- **Automatisierung**: APIs ermöglichen die Automatisierung von Datenabrufen, was Zeit und Aufwand spart. Anstatt manuell Daten herunterzuladen und zu verarbeiten, können Skripte geschrieben werden, die dies automatisch tun.
- **Filterung**: APIs ermöglichen oft die Filterung und Aggregation von Daten auf dem Server, bevor sie an den Client gesendet werden. Das reduziert die Menge der übertragenen Daten und verbessert die Effizienz.

Verschiedene Arten von APIs

Obwohl es weit verbreitete Designprinzipien für Webschnittstellen gibt, ist es wichtig zu beachten, dass nicht alle APIs gleich sind.

Es lohnt sich immer, die Dokumentation der API zu lesen, um zu verstehen, wie sie funktioniert und welche Daten sie bereitstellt. Open-Meteo bietet eine sehr gute und ausführliche Dokumentation, die alle Endpunkte und deren Parameter erklärt. Sie bietet sogar die Möglichkeit, Anfragen im Browser zu konfigurieren und zu testen: https://open-meteo.com/en/docs

Es sind jedoch nicht alle APIs so benutzerfreundlich. Es kann in manchen Fällen notwendig sein, eine Schnittstelle, die nicht für die Datenbereitstellung gedacht ist, zu verwenden. Hierfür lohnt es sich, das weit verbreitete Designprinzip von RESTful-APIs zu kennen, da dieses oft für die Kommunikation zwischen Servern verwendet wird.

REST APIs

Das **REST** (Representational State Transfer) Prinzip für Schnittstellen definiert Grundsätze für den Aufbau von Web-APIs, die eine einfache und effiziente Kommunikation zwischen Client (z.B. dem Browser) und Server ermöglichen. “RESTful” APIs sind besonders nützlich, wenn es darum geht, Daten aus verschiedenen Quellen zu integrieren oder zu aggregieren.

Die für uns wichtigen Merkmale von REST-APIs sind:

1. Zustandslosigkeit - Jede Anfrage an die Schnittstelle muss alle für ihre Ausführung notwendigen Daten enthalten
2. Einheitliche Schnittstelle - Der Aufbau der Schnittstelle folgt dem Prinzip adressierbarer Ressourcen, welche je nach Anwendungsfall erstellt, abgerufen, geändert und/oder gelöscht werden können

HTTP-Methoden in REST

RESTful APIs nutzen die HTTP-Methoden, um verschiedene Aktionen auf Ressourcen auszuführen. Diese Methoden sind im Header der HTTP-Anfrage angegeben. Wenn wir im Browser eine URL aufrufen, verwenden wir standardmäßig die GET-Methode. Andere Methoden werden programmatisch verwendet. Es gibt folgende HTTP-Methoden, die in REST APIs häufig verwendet werden:

- **GET**: Lesen von Daten, ohne diese zu verändern
- **POST**: Erstellen neuer Ressourcen
- **PUT**: Vollständiges Aktualisieren bestehender Ressourcen
- **PATCH**: Teilweises Aktualisieren bestehender Ressourcen
- **DELETE**: Löschen von Ressourcen

Als Data Scientist verwendest du hauptsächlich **GET** - du willst ja Daten lesen, nicht verändern.

Ressourcen und Endpunkte

In REST stellen Ressourcen Entitäten oder Objekte dar, mit denen interagiert werden kann. Sie werden über eindeutige URIs (Uniform Resource Identifiers) identifiziert. Beispiele für RESTful Endpunkte:

- `GET /users` - Liste aller Benutzer abrufen
- `GET /users/123/name` - Name des Benutzers mit ID 123 abrufen
- `POST /users` - Neuen Benutzer anlegen
- `PUT /users/123` - Benutzer mit ID 123 aktualisieren
- `DELETE /users/123` - Benutzer mit ID 123 löschen

Authentifizierung

Nicht alle APIs sind öffentlich zugänglich! Viele erfordern eine Authentifizierung, um sicherzustellen, dass nur autorisierte Benutzer auf die Daten zugreifen können. Dies kann durch

API-Schlüssel, OAuth-Token oder andere Authentifizierungsmethoden erfolgen. Dabei geht es nicht unbedingt darum, dass die Daten geheim sind und nur wenige Personen Zugriff darauf haben, sondern oft auch einfach darum, den Zugriff auf die API zu kontrollieren und Missbrauch zu verhindern. Beispielsweise kann so ein API-Betreiber sicherstellen, dass nur eine bestimmte Anzahl von Anfragen pro Tag von einem bestimmten Benutzer oder einer bestimmten Anwendung gesendet werden kann.

Oben wurde Pandas direkt verwendet, um Daten via HTTP-GET von einer API anzufragen. Pandas selbst bietet keine einfache Möglichkeit, andere HTTP Methoden zu verwenden oder Anfragen mit Authentifizierung zu senden. Die hierfür am häufigsten verwendete Bibliothek für Python für HTTP-Anfragen ist die `requests`-Bibliothek. ein API-Schlüssel kann in den Header der Anfrage eingefügt werden, um die Authentifizierung zu ermöglichen:

```
import requests

response = requests.get("https://jsonplaceholder.typicode.com/users/1",
headers={"Authorization": "Bearer SGllciBnaWJ0IGVzIG5pY2h0cyB6dSBzZWhlbg=="})
print(response.json())
```

```
{'id': 1, 'name': 'Leanne Graham', 'username': 'Bret', 'email': 'Sincere@april.biz',
'address': {'street': 'Kulas Light', 'suite': 'Apt. 556', 'city': 'Gwenborough',
'zipcode': '92998-3874', 'geo': {'lat': '-37.3159', 'lng': '81.1496'}}, 'phone':
'1-770-736-8031 x56442', 'website': 'hildegard.org', 'company': {'name': 'Romaguera-
Crona', 'catchPhrase': 'Multi-layered client-server neural-net', 'bs': 'harness real-
time e-markets'}}
```

Praktische Tipps

1. Immer die Dokumentation lesen

Bevor du eine API verwendest, schau dir ihre Dokumentation an. Dort findest du:

- Welche Endpoints (URLs) verfügbar sind
- Welche Parameter du verwenden kannst
- Ob du einen API-Schlüssel brauchst
- Welche Limits es gibt (z.B. maximal 1000 Anfragen pro Stunde)

2. Fehler abfangen

APIs können ausfallen oder falsche Anfragen zurückweisen. Prüfe daher immer den Statuscode:

```
response = requests.get("https://api.example.com/data")

if response.status_code == 200:
    data = response.json()
    # Weiter mit der Datenverarbeitung
elif response.status_code == 404:
    print("Daten nicht gefunden")
elif response.status_code == 429:
    print("Zu viele Anfragen - warte einen Moment")
else:
    print(f"Unbekannter Fehler: {response.status_code}")
```

3. Rate Limits beachten

Viele APIs haben **Rate Limits** - du darfst nur eine bestimmte Anzahl von Anfragen pro Zeit stellen. Das verhindert, dass die Server überlastet werden. Wenn du viele Daten brauchst, baue Pausen in deinen Code ein:

```
import time

for i in range(10):
    response = requests.get("https://api.example.com/data")
    # 1 Sekunde warten zwischen den Anfragen
    time.sleep(1)
```

Pagination

Ein weiteres wichtiges Konzept bei der Arbeit mit RESTful APIs ist die Pagination. Wenn eine API große Datenmengen zurückgibt, werden diese oft in Seiten unterteilt, um die Übertragung und Verarbeitung zu erleichtern. Dies geschieht häufig durch die Verwendung von Query-Parametern wie `offset` und `limit`, um die Anzahl der zurückgegebenen Datensätze zu steuern. Möchte ich beispielsweise die ersten 50 Einträge einer Liste abrufen, kann ich `offset=0&limit=50` verwenden. Wenn ich die nächsten 50 Einträge abrufen möchte, kann ich `offset=50&limit=50` verwenden und Seite 3 dann `offset=100&limit=50` usw.

Um die Daten in einem DataFrame zu speichern, können wir eine Schleife verwenden, um alle Seiten abzufragen und die Ergebnisse zusammenzuführen:

```
import requests
import pandas as pd
import json

url = "https://pokeapi.co/api/v2/pokemon"
params = {
    "limit": 200,
    "offset": 0
}

data = []
df = pd.DataFrame()
while True:
    response = requests.get(url, params=params)
    # requests übersetzt die Parameter in den Query-String. Die URL sieht dann so aus:
    # https://pokeapi.co/api/v2/pokemon?limit=200&offset=0
    if response.status_code != 200:
        # Stoppt die Schleife, wenn die Anfrage nicht erfolgreich war
        break
    page_data = response.json()
    if not page_data["results"]:
        # Stoppt die Schleife, wenn keine weiteren Daten vorhanden sind
        break
    df = pd.concat((df, pd.json_normalize(page_data["results"])))
    params["offset"] += params["limit"]

df
```

```
                      name                                        url
0                 bulbasaur      https://pokeapi.co/api/v2/pokemon/1/
1                   ivysaur      https://pokeapi.co/api/v2/pokemon/2/
```

```
2                    venusaur      https://pokeapi.co/api/v2/pokemon/3/
3                  charmander      https://pokeapi.co/api/v2/pokemon/4/
4                  charmeleon      https://pokeapi.co/api/v2/pokemon/5/
..                        ...                                      ...
97    ogerpon-wellspring-mask  https://pokeapi.co/api/v2/pokemon/10273/
98   ogerpon-hearthflame-mask  https://pokeapi.co/api/v2/pokemon/10274/
99   ogerpon-cornerstone-mask  https://pokeapi.co/api/v2/pokemon/10275/
100        terapagos-terastal  https://pokeapi.co/api/v2/pokemon/10276/
101         terapagos-stellar  https://pokeapi.co/api/v2/pokemon/10277/

[1302 rows x 2 columns]
```

Web-Scraping

In einigen Fällen sind Daten zwar auf einer Website o.ä. vorhanden, werden aber gar nicht über eine strukturierte API bereitgestellt. In diesen Fällen können wir auf Web-Scraping zurückgreifen, um die Daten direkt von der Website zu extrahieren. Dies kann rechtlich jedoch problematisch sein, da viele Dienste dies in ihren Nutzungsbedingungen verbieten und mit Maßnahmen wie Captcha unterbinden. Es sollte ausschließlich auf Aufforderung oder mit ausdrücklicher Erlaubnis des Betreibers durchgeführt werden.

Beim Web-Scraping wird direkt der HTML-Inhalt der Seite, der für die Anzeige im Browser verwendet wird, abgerufen und automatisiert geparsed und somit in ein strukturiertes Datenformat überführt. Ein Beispiel würde an dieser Stelle den Rahmen sprengen, da es sehr stark von der Struktur der Seite abhängt. Es gibt jedoch eine Vielzahl von Bibliotheken, die uns dabei helfen können, wie `BeautifulSoup` oder `Scrapy`. Falls Interaktionen mit der Seite notwendig sind, wie das Klicken auf Buttons oder das Ausfüllen von Formularen, können wir auf `Playwright` zurückgreifen. Diese Bibliothek ermöglicht das Erstellen von Skripten, welche Nutzeraktionen simulieren.

Zusammenfassung

APIs eröffnen dir als Data Scientist Zugang zu einer riesigen Welt von Daten:

- **Aktuelle Daten**: Börsenkurse, Wetterdaten, Social Media Trends
- **Umfangreiche Datensätze**: Ohne komplette Dateien herunterladen zu müssen
- **Automatisierung**: Deine Analysen können sich selbst mit neuen Daten versorgen

Die wichtigsten Konzepte:

1. **URLs**: Die “Adressen” von APIs mit Parametern für spezifische Anfragen
2. **JSON**: Das Standard-Datenformat für APIs
3. **HTTP-Methoden**: GET für Daten lesen, POST/PUT/DELETE für Änderungen
4. **Authentifizierung**: Manche APIs brauchen einen “Ausweis”
5. **Pagination**: Große Datenmengen werden in Seiten aufgeteilt

Das Schöne ist: Sobald du das Grundprinzip verstanden hast, funktionieren fast alle APIs ähnlich. Du musst nur ihre spezifische Dokumentation lesen, um zu verstehen, welche Daten sie anbieten und wie du sie anfragen kannst.

Weitere Ressourcen

- Working with APIs in Python - Code in 10 Minutes
- Request API data using Python in 8 minutes!
- Public APIs Liste auf GitHub zum ausprobieren

Optional:

- Die harte Wahrheit über Web Scraping im Jahr 2025
- Am I going to jail for web scraping?

Übung

Übung 1

Verwende die Open-Meteo API um die aktuelle Temperatur von 4 deutschen Städten abzurufen und erstelle ein Balkendiagramm:

Städte und Koordinaten:

```
cities = {
    'Hamburg': (53.55, 10.00),
    'Berlin': (52.52, 13.41),
    'München': (48.14, 11.58),
    'Köln': (50.94, 6.96)
}
```

API-Endpunkt: https://api.open-meteo.com/v1/forecast Parameter: latitude, longitude, current_weather=true

Erstelle ein einfaches Balkendiagramm mit den Städtenamen auf der x-Achse und den Temperaturen auf der y-Achse.

- (A) Geschafft

Übung 2

Verwende die PokeAPI (https://pokeapi.co/api/v2/pokemon/) um Daten über Schiggy und seine beiden Entwicklungen zu sammeln und zu visualisieren.

Dokumentation: https://pokeapi.co/docs/v2

Aufgaben:

1. Finde heraus, welche Pokemon-Nummer Schiggy hat
2. Hole die Daten für Schiggy und seine beiden Entwicklungen (Schillok und Turtok)
3. Extrahiere folgende Informationen für jedes Pokemon:
   - Name (name; auf englisch)
   - Gewicht (weight; in Hektogramm)
4. Erstelle ein Diagramm mit je einem Punkt pro Pokemon (x-Achse) für das Gewicht (y-Achse) und verbinde die Punkt mit einer Linie

- (A) Geschafft