

Python, Data Science & Deployment: Von den Grundlagen zur Praxis

Diese Präsentation bietet einen klaren und umfassenden Einstieg in Python, Data Science und Deployment. Sie erlernen die Python-Grundlagen, effiziente Datenanalyse mit Pandas, Datenvisualisierung mit Matplotlib/Plotly, die Entwicklung interaktiver Web-Apps mit Streamlit und das Deployment mit Docker. Der Kurs deckt alles Wichtige ab – von den ersten Schritten bis zur erfolgreichen Anwendung.



Bereit für Python & Data Science?

Zielgruppe & Fokus

Dieser Kurs richtet sich an Studierende ohne Python-Vorkenntnisse. Er vermittelt die essenziellen Python-Grundlagen, die für Data-Analytics-, Machine-Learning- und Deep-Learning-Kurse erforderlich sind. Sie erlernen zentrale Konzepte und führen erste Datenanalysen durch.

Was ist Python?

- Leicht erlernbare, universelle Programmiersprache mit intuitiver, englischähnlicher Syntax
- Führende Sprache in Data Science und Machine Learning
- Kostenfrei und Open Source
- Breite Unterstützung durch eine aktive Community und zahlreiche spezialisierte Bibliotheken



Jupyter Notebooks

Interaktives Tool für Python-Code und Ergebnisvisualisierung. Optimal zum Lernen, da es Text, Code und Ausgaben in einem Dokument vereint.



Optimal für Einsteiger

Python wurde für maximale Lesbarkeit und Verständlichkeit konzipiert. Der Fokus auf Klarheit und Einfachheit macht es zur idealen Sprache für den Einstieg in die Programmierung.

Teil 1 – Python Basics

Variablen – Die „Datenbehälter“

Variablen sind zentrale Bausteine jeder Programmiersprache, vergleichbar mit beschrifteten Boxen zum Speichern von Daten.

```
# Das ist ein Kommentar – wird nicht ausgeführt

# Variablen erstellen (wie Boxen mit Etiketten)
name = "Anna"    # Text (String)
alter = 25       # Ganze Zahl (Integer)
groesse = 1.68   # Dezimalzahl (Float)
ist_student = True # Wahr/Falsch (Boolean)

# Ausgeben mit print()
print("Name:", name)
print("Alter:", alter)
```

- Python weist Datentypen automatisch basierend auf dem Wert zu.
- `print()` gibt den Variableninhalt auf dem Bildschirm aus.
- Verwenden Sie aussagekräftige Variablennamen (ohne Leerzeichen).
- Variablenwerte sind jederzeit änderbar.

Pythons dynamische Typisierung: In Python müssen Datentypen nicht explizit deklariert werden. Das System erkennt und passt den Typ automatisch an, im Gegensatz zu statisch typisierten Sprachen wie Java oder C++.

Operationen mit Variablen

Rechnen und Textbearbeitung in Python

Python unterstützt alle grundlegenden arithmetischen Operationen und bietet intuitive Textbearbeitung. Die Syntax ist unkompliziert und ähnelt der mathematischen Schreibweise.

Arithmetische Operationen

```
# Rechnen mit numerischen Variablen  
a = 10  
b = 3  
  
print("Addition:", a + b)      # Ergebnis: 13  
print("Subtraktion:", a - b)    # Ergebnis: 7  
print("Multiplikation:", a * b)  # Ergebnis: 30  
print("Division:", a / b)       # Ergebnis: 3.333...  
print("Ganzzahldivision:", a // b) # Ergebnis: 3  
print("Rest:", a % b)          # Ergebnis: 1  
print("Potenz:", a ** b)        # Ergebnis: 1000 (103)
```

Text-Operationen (Strings)

```
# Texte (Strings) zusammenfügen  
vorname = "Max"  
nachname = "Mustermann"  
  
vollname = vorname + " " + nachname  
print("Vorname:", vollname) # Ergebnis: Max Mustermann  
  
# Strings vervielfältigen  
wiederholung = "Ha" * 3  
print(wiederholung) # Ergebnis: HaHaHa  
  
# Formatierte Strings (f-Strings)  
alter = 25  
ausgabe = f"{vorname} ist {alter} Jahre alt."  
print(ausgabe) # Max ist 25 Jahre alt.
```

- ⓘ Hinweis: Operatoren können je nach Datentyp unterschiedliche Bedeutungen haben. Der `+`-Operator dient beispielsweise sowohl zur Addition von Zahlen als auch zur String-Verkettung (Konkatenation).

Diese grundlegenden Operationen bilden den Grundstein für komplexere Berechnungen und Datenmanipulationen – essenzielle Fähigkeiten für die Datenanalyse.

Datentypen & Datenstrukturen in Python

Ein vollständiger Überblick

Primitive Datentypen

Integer (int) – Ganze Zahlen

```
alter = 25
anzahl = -10
grosse_zahl = 1_000_000 # Unterstriche zur Lesbarkeit
```

Float – Gleitkommazahlen

```
preis = 19.99
temperatur = -5.5
wissenschaftlich = 1.5e-4 # 0.00015
```

String (str) – Textdaten

```
name = "Anna"
text = 'Hallo Welt'
mehrzeilig = """Zeile 1
Zeile 2"""


```

Boolean (bool) – Wahrheitswerte

```
ist_aktiv = True
ist_fertig = False
```

NoneType – Kein Wert

```
ergebnis = None # Repräsentiert "nichts" oder "leer"
```

Datenstrukturen

Liste (list) – Geordnet, veränderbar

```
zahlen = [1, 2, 3, 4, 5]
gemischt = [1, "Text", True, 3.14]
verschachtelt = [[1, 2], [3, 4]]
```

Tuple (tuple) – Geordnet, unveränderbar

```
koordinaten = (10, 20)
rgb = (255, 128, 0)
# Tuples können nicht geändert werden!
```

Dictionary (dict) – Key-Value-Paare

```
person = {
    "name": "Max",
    "alter": 30,
    "stadt": "Berlin"
}
```

Set (set) – Ungeordnet, eindeutige Werte

```
farben = {"rot", "grün", "blau"}
zahlen_set = {1, 2, 3, 3, 3} # Ergebnis: {1, 2, 3}
```

💡 **Typ prüfen:** Mit `type(variable)` können Sie den Datentyp einer Variable ermitteln. Mit `isinstance(variable, typ)` prüfen Sie, ob eine Variable einen bestimmten Typ hat.

Typkonvertierung:

- `int("42") → 42`
- `float("3.14") → 3.14`
- `str(100) → "100"`
- `list("abc") → ['a', 'b', 'c']`

Listen in Python

Geordnete Sammlungen von Werten

Listen sind fundamentale und vielseitige Datenstrukturen in Python. Sie ermöglichen die effiziente Verwaltung geordneter Sammlungen unterschiedlicher Datentypen.

```
# Listen erstellen (wie eine Einkaufsliste)
einkaufsliste = ["Milch", "Brot", "Äpfel", "Käse"]
zahlen = [1, 5, 3, 9, 2]

# Auf Elemente zugreifen (Indexierung)
print("Erstes Element: ", einkaufsliste[0]) # Listenindizes beginnen bei 0!
print("Letztes Element: ", einkaufsliste[-1]) # -1 liefert das letzte Element

# Neue Elemente hinzufügen
einkaufsliste.append("Butter")
print("Nach dem Hinzufügen: ", einkaufsliste)

# Länge einer Liste ermitteln
print("Anzahl Elemente: ", len(einkaufsliste))
```



Wichtige Listenoperationen:

- `liste.remove("Element")` – Entfernt ein spezifisches Element.
- `liste.sort()` – Sortiert die Liste aufsteigend.
- `liste.reverse()` – Kehrt die Reihenfolge der Elemente um.
- `liste[1:3]` – Erstellt eine Teilliste (Slice) bestimmter Elemente.
- `if "Element" in liste:` – Überprüft die Existenz eines Elements in der Liste.



Wichtiger Hinweis: Die Indexierung in Python beginnt bei **0**. Das erste Element einer Liste hat somit den Index 0.

Listen sind zentral für die Datenanalyse, da sie die Basis für komplexere Strukturen wie DataFrames (Pandas) bilden. Sie ermöglichen effizientes Arbeiten und Manipulieren von Datenreihen.

Schleifen

Schleifen: Effiziente Wiederholung und Automatisierung von Code

Schleifen sind fundamentale Programmierkonstrukte, die die mehrfache Ausführung eines Code-Blocks ermöglichen. Dies verhindert redundanten Code und automatisiert wiederkehrende Prozesse, was für die effiziente Verarbeitung von Sequenzen wie Listen unerlässlich ist.

For-Schleife: Iteration über Sequenzen

```
# For-Schleife: Iteration durch eine Liste  
fruechte = ["Apfel", "Banane", "Orange"]  
print("Alle Früchte:")  
for frucht in fruechte:  
    print("- " + frucht)  
  
# Zahlenreihen mit range() durchlaufen  
print("\nZahlen 1 bis 5:")  
for zahl in range(1, 6): # range(1, 6) generiert Zahlen von 1 bis 5  
    print(zahl)
```

While-Schleife: Bedingungsgesteuerte Wiederholung

```
# While-Schleife: Wiederholung, solange eine Bedingung wahr ist  
counter = 0  
print("\nWhile-Schleife:")  
while counter < 3:  
    print(f"Counter ist: {counter}")  
    counter += 1 # Kurzform für counter = counter + 1  
  
# Schleifen vorzeitig beenden  
print("\nVorzeitiger Abbruch:")  
for i in range(10):  
    if i == 5:  
        break # Beendet die Schleife vollständig  
    print(i)
```

ⓘ **Wichtiger Hinweis: Einrückung ist obligatorisch!** In Python definiert die Einrückung (standardmäßig vier Leerzeichen) den Gültigkeitsbereich eines Code-Blocks. Alle unter einer for- oder while-Anweisung eingerückten Zeilen werden innerhalb der Schleife ausgeführt. Im Gegensatz zu vielen anderen Sprachen werden hier keine Klammern verwendet.

Die Beherrschung von Schleifen ist eine Schlüsselkompetenz für Programmierer. Sie ermöglicht die effiziente Verarbeitung großer Datenmengen und die Automatisierung komplexer Aufgaben – essentiell in Bereichen wie Datenanalyse, Webentwicklung und maschinellem Lernen.

Bedingungen

Bedingte Anweisungen: Logik im Code

Bedingte Anweisungen steuern den Programmablauf, indem sie Entscheidungen auf Grundlage definierter Kriterien ermöglichen. Dadurch lässt sich flexible Logik in den Code integrieren.

```
# Eine einfache Bedingung
alter = 20
if alter >= 18:
    print("Du bist volljährig.")
else:
    print("Du bist minderjährig.")
```

```
# Mehrere Bedingungen prüfen (elif)
note = 85
if note >= 90:
    print("Sehr gut!")
elif note >= 80: # elif = "else if"
    print("Gut!")
elif note >= 70:
    print("Befriedigend.")
else:
    print("Verbesserung nötig.")
```

Vergleichsoperatoren

- == – ist gleich
- != – ist ungleich
- >, < – größer/kleiner als
- >=, <= – größer/gleich, kleiner/gleich

Logische Operatoren

```
# Kombination mehrerer Bedingungen
alter = 25
einkommen = 2500

if alter > 18 and einkommen > 2000:
    print("Kredit bewilligt.")

# Bedingungen verneinen (not)
if not (alter < 18):
    print("Du bist kein Minderjähriger.")
```

Bedingungen sind essenziell in der Datenverarbeitung, um Datensätze zu filtern, zu kategorisieren oder komplexe Entscheidungen zu treffen. Dies ist auch grundlegend für Machine-Learning-Modelle.

Wichtige Bibliotheken

Python-Bibliotheken: Leistungsstarke Erweiterungen

Definition: Bibliotheken (auch „Pakete“ oder „Module“) sind vordefinierte Code-Sammlungen, die spezifische Aufgaben vereinfachen und die Entwicklungseffizienz steigern.

Warum Bibliotheken nutzen?

Bibliotheken bieten bewährte, von der Python-Community entwickelte Lösungen. Dies spart wertvolle Entwicklungszeit, da keine Neuentwicklung erforderlich ist. Insbesondere in der Data Science sind diese gut getestete und optimierte Werkzeuge unverzichtbar für effiziente, komplexe Analysen.

Top-Bibliotheken für Data Science

- **Pandas:** Datenmanagement und -analyse
- **NumPy:** Numerische Berechnungen mit Arrays
- **Matplotlib / Plotly:** Datenvisualisierung
- **Scikit-learn:** Machine-Learning-Modelle
- **TensorFlow / PyTorch:** Deep-Learning-Projekte
- **Streamlit:** Interaktive Webanwendungen

- **Installation von Bibliotheken:** Bibliotheken installieren Sie einfach mit dem Paketmanager pip (z. B. pip install pandas). In Umgebungen wie Google Colab sind viele Bibliotheken bereits vorinstalliert.

In diesem Kurs konzentrieren wir uns zunächst auf Pandas und Visualisierungsbibliotheken, da sie das Fundament für effektive Datenanalyse und wertvolle Erkenntnisse bilden.

Pandas: Der unverzichtbare Daten-Manager

Effiziente Datenstrukturierung und -analyse mit Pandas

Pandas ist die zentrale Python-Bibliothek für die Arbeit mit tabellarischen Daten. Ihr Kernstück, der **DataFrame**, ist eine leistungsstarke Datenstruktur zur effizienten Analyse und Manipulation strukturierter Datensätze.

```
# Pandas importieren (üblicherweise als 'pd')
import pandas as pd

# DataFrame erstellen (wie eine Excel-Tabelle)
daten = {
    'Name': ['Anna', 'Ben', 'Clara', 'David'],
    'Alter': [25, 30, 35, 28],
    'Stadt': ['Berlin', 'Hamburg', 'München', 'Köln']
}
df = pd.DataFrame(daten)
print(df)
```

Ausgabe:

```
Name Alter Stadt
0 Anna 25 Berlin
1 Ben 30 Hamburg
2 Clara 35 München
3 David 28 Köln
```



Was ist ein DataFrame?

Ein **DataFrame** ist eine 2-dimensionale Datenstruktur mit:

- Benannten Spalten mit diversen Datentypen
- Indizierten Zeilen für einfachen Zugriff
- Leistungsfähiger als herkömmliche Tabellenkalkulationen
- Optimiert für schnelle Datenmanipulation und -analyse
- Ermöglicht komplexe Abfragen und Aggregationen

Pandas schließt die Lücke zwischen Python-Programmierung und intuitiver Tabellenansicht. Es ermöglicht die effiziente Bearbeitung von Datensätzen, die sowohl für Entwickler als auch für Datenanalysten intuitiv ist.

Arbeiten mit DataFrames

Grundlegende DataFrame-Operationen

Pandas bietet leistungsstarke Funktionen zur effizienten Inspektion und Manipulation von Daten. Diese grundlegenden Operationen sind der Schlüssel für den Einstieg in die Datenanalyse.

Daten erkunden

```
# Erste 2 Zeilen anzeigen  
print(df.head(2))  
  
# Informationen über den DataFrame erhalten  
print(df.info())  
  
# Statistische Zusammenfassung  
print(df.describe())  
  
# Spaltenübersicht  
print(df.columns)
```

Daten auswählen und filtern

```
# Eine einzelne Spalte auswählen  
print(df['Name'])  
  
# Mehrere Spalten auswählen  
print(df[['Name', 'Alter']])  
  
# Zeilen basierend auf einer Bedingung filtern  
aeltere_personen = df[df['Alter'] > 28]  
print(aeltere_personen)  
  
# Spezifische Zeilen und Spalten mit .loc  
print(df.loc[0:2, ['Name', 'Stadt']])
```

Daten hinzufügen

Neue Spalte berechnen:

```
df['Geburtsjahr'] = 2023 - df['Alter']
```

Daten gruppieren

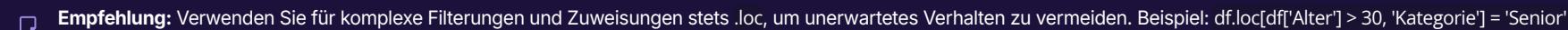
Nach Kategorien zusammenfassen:

```
df.groupby('Stadt')['Alter'].mean()
```

Daten sortieren

Nach einer oder mehreren Spalten ordnen:

```
df.sort_values('Alter', ascending=False)
```

 **Empfehlung:** Verwenden Sie für komplexe Filterungen und Zuweisungen stets .loc, um unerwartetes Verhalten zu vermeiden. Beispiel: df.loc[df['Alter'] > 30, 'Kategorie'] = 'Senior'

Mit diesen grundlegenden Operationen sind Sie optimal ausgerüstet, um Ihre Daten effektiv zu erkunden, zu bereinigen und für weiterführende Analysen oder Visualisierungen vorzubereiten. Starten Sie Ihre Datenreise mit Pandas!

Matplotlib & Plotly

Datenvisualisierung: Aussagekräftige Diagramme erstellen

Matplotlib

Matplotlib ist die grundlegende Python-Bibliothek für die Datenvisualisierung, ideal für statische, publikationsreife Grafiken.

```
# Einfaches Balkendiagramm mit Matplotlib
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))
plt.bar(df['Name'], df['Alter'])
plt.title('Alter der Personen')
plt.xlabel('Name')
plt.ylabel('Alter in Jahren')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Plotly

Plotly bietet interaktive Grafiken mit Zoom- und Hover-Funktionen, ideal für Webanwendungen und Dashboards.

```
# Interaktives Diagramm mit Plotly
import plotly.express as px

fig = px.bar(df, x='Name', y='Alter',
             color='Stadt',
             title='Alter nach Person und Stadt')
fig.update_layout(xaxis_title='Name',
                  yaxis_title='Alter in Jahren')
fig.show()
```

Bibliotheksauswahl

- **Matplotlib**: Wissenschaftliche Publikationen, maximale Anpassbarkeit, statische Berichte.
- **Plotly**: Interaktive Dashboards, Webanwendungen, dynamische Datenexploration.
- **Seaborn**: Basiert auf Matplotlib; ideal für fortgeschrittene statistische Visualisierungen mit minimalem Codeaufwand.
- **Pandas .plot()**: Schnellste Methode für einfache Visualisierungen direkt aus DataFrames.

Die Auswahl der geeigneten Visualisierungsbibliothek richtet sich nach Projekt und Zielen. Für interaktive Dashboards in diesem Kurs empfehlen wir Plotly, während Matplotlib/Seaborn für präzise statische Analysen optimal ist.

Data Science Best Practices: Pandas & Visualisierung

Datenimport

Effizienter Datenimport aus diversen Quellen:

- `pd.read_csv('daten.csv')` – für CSV-Dateien
- `pd.read_excel('daten.xlsx')` – für Excel-Dateien
- `pd.read_parquet('daten.parquet')` – für effiziente Parquet-Dateien
- `pd.read_sql('SELECT * FROM tabelle', verbindung)` – direkt aus Datenbanken

⚠ Häufige Fallstricke vermeiden:

- Das `SettingWithCopyWarning` in Pandas (nutzen Sie `.loc` für Zuweisungen).
- Große Datenmengen auf einmal laden (verwenden Sie Chunking oder `Dask` für Big Data).
- Übermäßig komplexe Visualisierungen (setzen Sie auf Einfachheit und klare Botschaften).

Diese Best Practices legen den Grundstein für professionellen, effizienten und wartbaren Code – unverzichtbar für erfolgreiche Data-Science-Projekte.

Datenbereinigung & -vorbereitung

- **Vektorisierung:** Nutzen Sie direkte Spaltenoperationen (z. B. `df['Gehalt'] * 1.1`) statt Python-Schleifen für maximale Performance.
- **Speicheroptimierung:** Konvertieren Sie Datentypen (z. B. `string` zu `category`) zur Ressourceneffizienz.
- **Fehlende Werte:** Behandeln Sie fehlende Daten mit `df.fillna()` oder `df.dropna()`.

Analyse & Visualisierung

- **Reproduzierbarkeit:** Strukturieren Sie Analysen mittels wiederverwendbarer Funktionen oder Pipelines (z. B. `sklearn.pipeline`).
- **Dashboarding:** `Plotly` und `Streamlit` eignen sich ideal für interaktive Analysen.
- **Storytelling:** Erzählen Sie durch eine logische Abfolge von Visualisierungen eine klare Geschichte.

Teil 3 – Streamlit

Streamlit: Interaktive Web-Apps direkt aus Python

Was ist Streamlit?

Streamlit ist ein Framework, das Ihnen ermöglicht, anspruchsvolle **Webanwendungen** für Data Science ohne Vorkenntnisse in HTML, CSS oder JavaScript zu entwickeln.

Mit Streamlit können Sie:

- Interaktive Dashboards erstellen
- Dynamische Datenvizualisierungen teilen
- ML-Modelle mit ansprechenden Benutzeroberflächen versehen
- Prototypen schnell entwickeln und iterieren

Funktionsweise

Erstellen Sie ein Python-Skript; Streamlit wandelt es automatisch in eine interaktive Webanwendung um. Der Ablauf:

1. Python-Code in einer ` `.py` -Datei verfassen
2. Streamlit-Befehle für die Gestaltung der Benutzeroberfläche (UI) nutzen
3. Die Anwendung mit `streamlit run app.py` starten
4. Änderungen im Code führen zur sofortigen Aktualisierung der App

⊗  **Wichtiger Hinweis:** Streamlit-Code kann **NICHT** direkt in Jupyter Notebooks ausgeführt werden. Eine separate ` `.py` -Datei und der Start über das Terminal sind erforderlich.

Streamlit hat die Bereitstellung von Data-Science-Ergebnissen revolutioniert. Es überbrückt die Lücke zwischen Datenanalyse und Anwendungsentwicklung, indem es die Erstellung von Web-Apps erheblich vereinfacht und zugänglicher macht.

Ihre erste Streamlit-App

Erste Schritte: Code und Ausführung

Schritt 1: Code in 'meine_app.py' speichern

```
import streamlit as st

st.title("Meine erste Streamlit App")
st.write("Willkommen zu Ihrer ersten interaktiven Anwendung.")
```

Streamlit-Kernkomponenten

Die Bausteine für interaktive Web-Apps



Textelemente

- `st.title()`: Haupttitel
- `st.header()`: Abschnittsüberschrift
- `st.subheader()`: Untergeordnete Überschrift
- `st.write()`: Flexible Inhaltsausgabe
- `st.markdown()`: Markdown-Formatierung



Interaktive Eingaben (Widgets)

- `st.slider()`: Schieberegler
- `st.selectbox()`: Dropdown-Auswahl
- `st.radio()`: Exklusive Auswahl (Radio-Buttons)
- `st.checkbox()`: Ja/Nein-Kontrollkästchen
- `st.text_input()`: Freie Texteingabe
- `st.date_input()`: Datumsauswahl
- `st.file_uploader()`: Datei-Upload



Anordnung & Struktur

- `st.sidebar`: Seitliche Navigationsleiste
- `st.columns()`: Spaltenlayout
- `st.expander()`: Ausklappbarer Bereich
- `st.tabs()`: Registerkarten-Organisation
- `st.container()`: Elementgruppierung

Datenvisualisierung

- `st.dataframe()`: Interaktive Datentabelle
- `st.table()`: Statische Datentabelle
- `st.line_chart()`: Liniendiagramm
- `st.bar_chart()`: Balkendiagramm
- `st.plotly_chart()`: Plotly-Visualisierung
- `st.map()`: Geodaten auf Karte

Erweiterte Funktionen

- `@st.cache_data`: Daten-Caching
- `st.progress()`: Fortschrittsanzeige
- `st.download_button()`: Daten-Download-Button
- `st.session_state`: Zustandsverwaltung
- `st.secrets`: Sichere Datenspeicherung

Diese vielseitigen Komponenten ermöglichen die Entwicklung leistungsstarker, maßgeschneideter Datenanwendungen. Beginnen Sie einfach und erweitern Sie die Funktionalität schrittweise, um Streamlit optimal zu nutzen.

Docker – Anwendungen containerisieren

Effiziente Bereitstellung für Data Science

Was ist Docker?

Docker verpackt Software und all ihre Abhängigkeiten (z. B. Python-Versionen, Bibliotheken) in standardisierte, isolierte Einheiten: **Container**.

Analogie

Denken Sie an einen **Schiffscontainer**: Unabhängig vom Inhalt wird er von jedem Kran (System) transportiert. Der Inhalt bleibt isoliert und standardisiert verpackt.

Vorteile für Data Science

- **Reproduzierbarkeit:** Garantiert identische Ausführung von Analysen und Modellen auf jedem System.
- **Einfaches Deployment:** Einmal containerisiert, überall ausführbar – lokal, Cloud oder Serverless.
- **Isolation:** Projekte mit unterschiedlichen Python- oder Bibliotheksversionen beeinflussen sich nicht gegenseitig.
- **Skalierbarkeit:** Nahtlose Integration in Cloud-Dienste und Orchestrierungstools wie Kubernetes.
- **Versionierung:** Container-Images lassen sich versionieren und speichern.

- Docker ist für Data-Science-Teams von unschätzbarem Wert. Es fördert die Zusammenarbeit und gewährleistet die konsistente Funktion von Analysen und Modellen, vom Entwickler-Laptop bis zur Produktionsumgebung.

Dieser Kurs zeigt Ihnen, wie Sie Ihre Python-Anwendungen, insbesondere Streamlit-Dashboards, mit Docker containerisieren und effizient bereitstellen.

Docker-Workflow

Vom Bauplan zum laufenden Container

Dockerfile

Ein **Bauplan** in Textform, der alle Schritte zur Erstellung eines Docker-Images definiert.

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY ..
EXPOSE 8501
CMD ["streamlit", "run", "app.py"]
```

Die wichtigsten Befehle im Dockerfile sind:

- FROM: Basis-Image festlegen.
- COPY: Dateien in den Container kopieren.
- RUN: Befehle während des Build-Prozesses ausführen.
- CMD: Startbefehl für den Container definieren.

Image

Ein **fertiges Abbild** eines Containers, das alles Nötige zur Ausführung enthält:

- Das Betriebssystem (z. B. Alpine Linux)
- Python und alle installierten Bibliotheken
- Ihren Anwendungscode
- Konfigurationen und Umgebungsvariablen

Images werden in Registries (z. B. Docker Hub) gespeichert und können so global geteilt werden.

Container

Eine **laufende Instanz** eines Images.

- Container sind start-, stopp- und neustartbar.
- Sie sind isoliert von anderen Containern und dem Host-System.
- Die Kommunikation mit der Außenwelt erfolgt über Ports.
- Daten können mit dem Host über „Volumes“ geteilt werden.

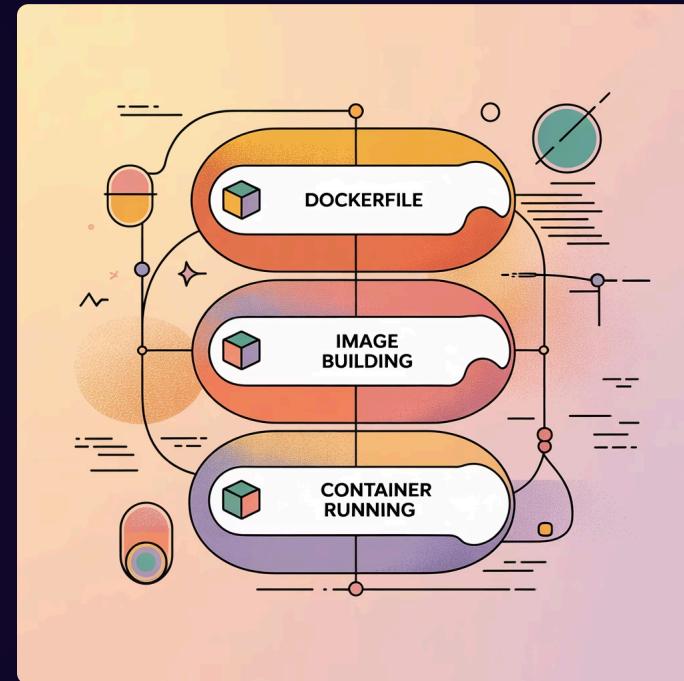
Die wichtigsten Docker-Befehle

```
# Image erstellen  
docker build -t meine-streamlit-app .
```

```
# Container starten  
docker run -p 8501:8501 meine-streamlit-app
```

```
# Laufende Container anzeigen  
docker ps
```

```
# Container stoppen  
docker stop container_id
```



Der Parameter `-p 8501:8501` leitet den Container-Port 8501 auf Port 8501 Ihres Hosts um. Ihre Streamlit-App ist dann über <http://localhost:8501> im Browser zugänglich.

Docker Compose: Multi-Container-Anwendungen verwalten

Was ist Docker Compose?

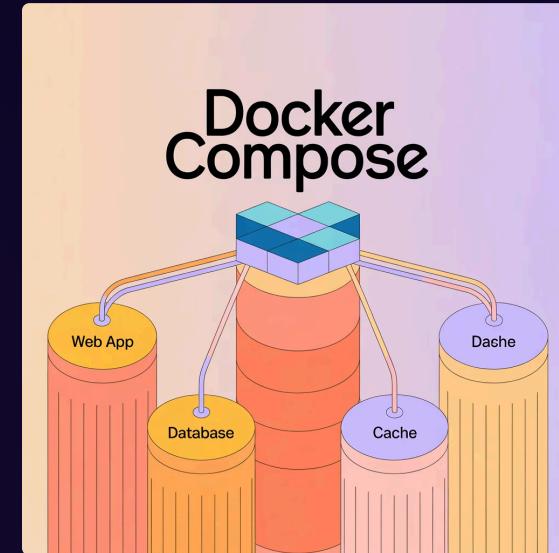
Docker Compose ist ein Tool zur Definition und zum Start komplexer **Multi-Container-Anwendungen**, die aus voneinander abhängigen Diensten (z.B. App, Datenbank, API) bestehen. Die Konfiguration erfolgt über eine einzige Datei: die docker-compose.yml.

Beispiel docker-compose.yml

```
version: '3'
services:
  streamlit-app:
    build: .
    ports:
      - "8501:8501"
    volumes:
      - ./data:/app/data
    environment:
      - DATABASE_URL=postgres://user:pass@db:5432/dbname

  db:
    image: postgres:13
    environment:
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=pass
      - POSTGRES_DB=dbname
    volumes:
      - postgres_data:/var/lib/postgresql/data

  volumes:
    postgres_data:
```



Ihre Vorteile mit Docker Compose

- **Einfacher Start:** Die gesamte Entwicklungsumgebung startet mit nur **einem Befehl** (docker-compose up).
- **Versionierbare Konfiguration:** Infrastruktur als Code (IaC) ermöglicht Versionskontrolle der Konfiguration.
- **Nahtlose Kommunikation:** Dienste kommunizieren einfach über ihre Servicenamen.
- **Zentrale Verwaltung:** Umgebungsvariablen und Volumes werden effizient an einem Ort verwaltet.
- **Skalierbarkeit:** Dienste lassen sich mühelos skalieren (z.B. mit docker-compose up --scale).

ⓘ **Profi-Tipp:** Docker Compose ist auch für Einzel-Apps empfehlenswert! Es vereinfacht den Startprozess erheblich, sichert Konsistenz und erleichtert den Übergang zu komplexeren Setups bei Projektwachstum.

Wichtige Docker Compose Befehle

```
# Alle Dienste starten  
docker-compose up
```

```
# Im Hintergrund starten  
docker-compose up -d
```

```
# Status der Dienste anzeigen  
docker-compose ps
```

```
# Alle Dienste stoppen  
docker-compose down
```

```
# Logs anzeigen  
docker-compose logs -f
```

Docker Compose ist besonders wertvoll für Data-Science-Projekte, da es die Verwaltung von Umgebungen vereinfacht, die typischerweise mehrere Komponenten wie Datenbanken, ML-Modell-Server und Visualisierungs-Apps umfassen.

Ihr Weg zum Data Scientist: Rückblick und Ausblick

Python-Grundlagen

- Variablen und Datentypen
- Listen und Operationen
- Schleifen und Bedingungen
- Funktionen und Module

Docker-Container

- Grundlagen der Containerisierung
- Dockerfile-Erstellung
- Arbeit mit Images und Containern
- Orchestrierung (Docker Compose)



Datenanalyse

- Pandas DataFrames beherrschen
- Effektive Datenmanipulation
- Datenvisualisierung (Plotly)
- Statistische Analysen

Streamlit-Apps

- Interaktive Webanwendungen
- Intuitive UI-Komponenten
- Dynamische Daten-Dashboards
- Effiziente Benutzerinteraktionen

Ihre nächsten Schritte im Kurs

Woche 2

Entwicklung fortgeschrittener Streamlit-Anwendungen mit erweiterten UI-Elementen und Datenvisualisierungen.

1

2

3

Woche 4–6

Cloud-Deployment (AWS, Azure, GCP) und Erstellung einer individuellen Fallstudie als Abschlussprojekt.

Woche 3

Integration von Machine-Learning-Modellen (Scikit-learn, TensorFlow) in Anwendungen.

Glossar: Fachbegriffe von A bis Z

Bibliothek/Library: Eine Sammlung von vorgefertigtem Code (Funktionen, Klassen), die von anderen Programmen genutzt werden kann, um spezifische Aufgaben zu lösen.

Container: Eine standardisierte, isolierte Umgebung, die alles Notwendige zur Ausführung einer Software enthält (Code, Laufzeitumgebung, Systemtools, Bibliotheken).

CSV (Comma-Separated Values): Ein einfaches Textdateiformat, das Daten in tabellarischer Form speichert, wobei Werte durch Kommas getrennt sind.

Dashboard: Eine grafische Benutzeroberfläche, die wichtige Kennzahlen und Datenvisualisierungen auf einen Blick darstellt, oft interaktiv.

Data Science: Ein interdisziplinäres Feld, das wissenschaftliche Methoden, Prozesse, Algorithmen und Systeme nutzt, um Erkenntnisse aus strukturierten und unstrukturierten Daten zu gewinnen.

Datenanalyse: Der Prozess der Untersuchung, Bereinigung, Transformation und Modellierung von Daten, um nützliche Informationen zu entdecken, Schlussfolgerungen zu ziehen und Entscheidungen zu unterstützen.

Datenvisualisierung: Die grafische Darstellung von Daten, um Muster, Trends und Ausreißer leichter erkennbar und verständlich zu machen.

DataFrame: Eine zweidimensionale, tabellenartige Datenstruktur in Pandas mit Spalten und Zeilen, ähnlich einer Tabelle in einer Datenbank oder einem Excel-Arbeitsblatt.

Deployment: Der Prozess der Bereitstellung einer Anwendung auf einem Server oder in einer Cloud-Umgebung, damit sie für Endbenutzer zugänglich ist.

Docker: Eine Plattform zum Entwickeln, Ausliefern und Ausführen von Anwendungen in Containern, die Isolation und Portabilität bietet.

Docker Compose: Ein Tool zum Definieren und Ausführen von Multi-Container-Docker-Anwendungen, das die Konfiguration in einer YAML-Datei zusammenfasst.

Dockerfile: Eine Textdatei mit Anweisungen, die Docker verwendet, um ein Image zu bauen.

f-String: Ein Python-Literal für formatierte String-Literale, das das Einbetten von Ausdrücken in Strings ermöglicht.

Image (Docker): Ein schreibgeschütztes Template mit Anweisungen zum Erstellen eines Docker-Containers, das alles zur Ausführung einer Anwendung benötigt.

Indexierung: Der Prozess des Zugriffs auf einzelne Elemente oder Teilmengen von Datenstrukturen (z. B. Listen, Strings) anhand ihrer Position oder ihres Labels.

Liste: Eine geordnete, veränderliche Sammlung von Elementen in Python, die verschiedene Datentypen enthalten kann.

Machine Learning: Ein Teilbereich der künstlichen Intelligenz, der es Systemen ermöglicht, aus Daten zu lernen und Vorhersagen oder Entscheidungen zu treffen, ohne explizit programmiert zu sein.

Matplotlib: Eine beliebte Python-Bibliothek zur Erstellung statischer, animierter und interaktiver Visualisierungen in Python.

Modul: Eine einzelne Datei in Python, die Python-Code (Variablen, Funktionen, Klassen) enthält und in andere Programme importiert werden kann.

Package: Eine Sammlung von Python-Modulen, die in einem Verzeichnis organisiert sind und über einen einheitlichen Namespace angesprochen werden können.

Pandas: Eine leistungsstarke Python-Bibliothek für Datenanalyse und -manipulation, die DataFrames als zentrale Datenstruktur verwendet.

Plotly: Eine Open-Source-Grafikbibliothek für interaktive, webbasierte Datenvisualisierungen.

Port: Eine Kommunikationsendpunktnummer, die es Anwendungen ermöglicht, Daten über ein Netzwerk auszutauschen.

Streamlit: Ein Open-Source-Python-Framework zum schnellen Erstellen und Teilen interaktiver Webanwendungen und Dashboards für Data Science und Machine Learning.

Volume: Eine persistente Speichermöglichkeit für Docker-Container, die es erlaubt, Daten außerhalb des Container-Dateisystems zu speichern und zu teilen.

Webanwendung: Eine Anwendung, die über einen Webbrowser zugänglich ist und typischerweise auf einem Server ausgeführt wird.

Ressourcen & Lernmaterialien

Offizielle Dokumentationen

- [Python-Dokumentation](#) – Umfassende Referenz und Einführung.
- [Pandas-Dokumentation](#) – Detaillierte API-Referenz und praxisnahe Tutorials.
- [Streamlit-Dokumentation](#) – Praktische Guides und Referenzen zu Komponenten.
- [Docker-Dokumentation](#) – Anleitungen, Best Practices und detaillierte Referenzen.

Praxis-Ressourcen

- [Kaggle](#) – Datensätze, Wettbewerbe und Community-Notebooks.
- [GitHub](#) – Open-Source-Projekte zum Lernen und zur Zusammenarbeit.
- [Stack Overflow](#) – Problemlösungen und Antworten auf Programmierfragen.