PREFLIGHT-CHECK

Arbeitsumgebung für ML-Projekte

Selbstdiagnose & Mikro-Lernmodule

Weil ein ML-Projekt nur so gut ist wie seine Arbeitsumgebung.

Worum geht es?

Dieses Dokument hilft dir, deine Arbeitsumgebung professionell einzurichten. In Teil 1 findest du eine ehrliche Selbstdiagnose — kein Test, keine Bewertung, sondern ein Werkzeug für dich selbst. In Teil 2 gibt es für jeden Bereich ein kompaktes Lernmodul mit konkreten Schritten.

Unser Workflow: Wir arbeiten mit dem Datei-Explorer (Windows) bzw. Finder (Mac) für die Dateiverwaltung und mit Visual Studio Code als Entwicklungsumgebung. Python-Environments werden über die VS Code-Extension „Python Environments“ erstellt und verwaltet.

Warum das wichtig ist: Datei- und Umgebungsmanagement ist kein Vorwissen — es ist eine Kernkompetenz in jedem ML-Projekt. So wie Piloten vor jedem Flug eine Checkliste durchgehen, durchläufst du vor jeder Arbeitssession deinen Preflight-Check.

Teil 1: Selbstdiagnose

Gehe die folgenden Aussagen ehrlich durch. Kreuze an, was du sicher kannst. Bereiche mit mehreren „Nein“-Antworten zeigen dir, welches Mikro-Lernmodul in Teil 2 für dich relevant ist.

A — Dateisystem & Pfade

✓	Aussage	Kann ich?
☐	Ich weiß, wo auf meinem Rechner Dateien gespeichert werden, wenn ich sie herunterlade.	Ja / Nein
☐	Ich kann den Unterschied zwischen einem lokalen Ordner und einem Cloud-Ordner (OneDrive, iCloud, Google Drive) erklären.	Ja / Nein
☐	Ich verstehe den Unterschied zwischen einem absoluten Pfad (z. B. C:\Users\max\projekte) und einem relativen Pfad (z. B. ./daten/train.csv).	Ja / Nein
☐	Ich kann eine Datei gezielt von einem Ordner in einen anderen kopieren oder verschieben — ohne sie zu suchen.	Ja / Nein

→ **Bei 2+ × „Nein“:** Arbeite Modul A in Teil 2 durch.

B — Projektordner & Struktur

✓	Aussage	Kann ich?
☐	Ich lege für jedes Projekt einen eigenen Ordner an einem festen Ort an (nicht auf dem Desktop, nicht in Downloads).	Ja / Nein
☐	Ich kann innerhalb eines Projektordners Unterordner erstellen (z. B. data/, notebooks/, models/).	Ja / Nein
☐	Wenn ich eine neue .ipynb- oder .py-Datei erstelle, speichere ich sie bewusst im richtigen Projektordner.	Ja / Nein
☐	Ich verstehe, warum alle Projektdateien im selben Ordnerbaum liegen müssen, damit relative Pfade funktionieren.	Ja / Nein

→ **Bei 2+ × „Nein“:** Arbeite Modul B in Teil 2 durch.

C — Datei-Explorer & VS Code

✓	Aussage	Kann ich?
☐	Ich kann im Datei-Explorer (Windows) bzw. Finder (Mac) gezielt zu einem Ordner navigieren.	Ja / Nein
☐	Ich kann einen Ordner per Rechtsklick in VS Code öffnen („Open with Code“ / „Mit Code öffnen“).	Ja / Nein
☐	Ich verstehe, dass VS Code immer einen Ordner als Arbeitsbereich öffnet — und dass dieser Ordner bestimmt, wo relative Pfade starten.	Ja / Nein
☐	Ich kann in der VS Code-Seitenleiste Dateien und Ordner erstellen, umbenennen und verschieben.	Ja / Nein

→ **Bei 2+ × „Nein“:** Arbeite Modul C in Teil 2 durch.

D — Python-Environments (VS Code)

✓	Aussage	Kann ich?
☐	Ich weiß, was ein virtuelles Python-Environment ist und warum es existiert.	Ja / Nein
☐	Ich kann in VS Code über die „Python Environments“-Extension ein neues Environment erstellen.	Ja / Nein
☐	Ich kann in VS Code den richtigen Python-Interpreter (aus meinem Environment) auswählen.	Ja / Nein
☐	Ich verstehe, warum Pakete nur innerhalb des aktiven Environments verfügbar sind.	Ja / Nein

→ **Bei 2+ × „Nein“:** Arbeite Modul D in Teil 2 durch.

E — Speicherplatz & Systemhygiene

✓	Aussage	Kann ich?
☐	Ich weiß, wie viel freier Speicherplatz auf meiner Festplatte ist.	Ja / Nein
☐	Ich kann große, nicht mehr benötigte Dateien finden und löschen.	Ja / Nein
☐	Ich weiß, dass Jupyter-Notebooks mit eingebetteten Outputs sehr groß werden können, und kann Outputs vor dem Speichern löschen.	Ja / Nein

→ **Bei 2+ × „Nein“:** Arbeite Modul E in Teil 2 durch.

F — Dateien orchestrieren

✓	Aussage	Kann ich?
☐	Ich kann nachvollziehen, wie ein Skript (z. B. train.py) ein Modell speichert und ein anderes Skript (z. B. app.py) es wieder lädt.	Ja / Nein
☐	Ich kann eine .pkl- oder .joblib-Datei an einem definierten Ort speichern und von einem anderen Skript aus darauf zugreifen.	Ja / Nein
☐	Ich verstehe, warum die Ordnerstruktur für ein Gradio-Deployment wichtig ist (Modell + App im selben Kontext).	Ja / Nein

→ **Bei 2+ × „Nein“:** Arbeite Modul F in Teil 2 durch.

Kurzauswertung

0–3 × „Nein“ insgesamt: Du bist solide aufgestellt. Nutze die Module als Nachschlagewerk.

4–8 × „Nein“: Es gibt klare Lücken. Arbeite die relevanten Module in Teil 2 gezielt durch — das ist eine Investition, die sich sofort auszahlt.

9+ × „Nein“: Arbeite alle Module der Reihe nach durch, bevor du mit dem nächsten ML-Projekt weitermachst. Das klingt nach viel, aber es sind jeweils nur 15–20 Minuten.

Teil 2: Mikro-Lernmodule

Jedes Modul ist in sich abgeschlossen und dauert ca. 15–20 Minuten. Du brauchst nur die Module zu bearbeiten, die deine Selbstdiagnose als Lücke identifiziert hat.

Modul A: Dateisystem & Pfade

Das Problem

Viele Probleme entstehen, weil Dateien an unerwarteten Orten liegen — in Cloud-Sync-Ordnern, im Downloads-Ordner oder verstreut über den Desktop. Jupyter und Python finden diese Dateien dann nicht, oder sie verhalten sich unvorhersehbar, weil OneDrive oder iCloud dazwischenfunken.

Die Regel

Alle Projektdateien müssen lokal liegen — nicht in einem Cloud-synchronisierten Ordner. Du musst jederzeit wissen, wo eine Datei liegt, und den Pfad dorthin angeben können.

⚠ Typische Falle

OneDrive-Falle: Unter Windows liegt der Desktop oft in C:\Users\name\OneDrive\Desktop — alles dort wird synchronisiert und kann Probleme verursachen. Prüfe, ob dein Pfad „OneDrive“ enthält. Falls ja: arbeite nicht von dort!

Schritt für Schritt

1. Öffne den Datei-Explorer (Windows) oder Finder (Mac).
2. Navigiere zu deinem Benutzerordner. Dort liegt ein klar erkennbarer Pfad, z. B. C:\Users\deinname\ oder /Users/deinname/.
3. Erstelle dort einen neuen Ordner namens projekte (Rechtsklick → Neuer Ordner).
4. Merke dir diesen Pfad. Er ist ab jetzt dein Anker für alle Kurs-Projekte.

Konkret am Rechner

Windows (Explorer)	macOS (Finder)
1. Explorer öffnen (Windows-Taste + E)	1. Finder öffnen (Cmd + Leertaste → „Finder“)
2. Links auf „Dieser PC“ klicken	2. Menü: Gehe zu → Benutzerordner
3. Doppelklick auf C: → Benutzer → dein Name	3. Rechtsklick → Neuer Ordner → „projekte“ nennen
4. Rechtsklick → Neu → Ordner → „projekte“ nennen	4. Prüfe: Liegt der Ordner unter /Users/deinname/projekte? (Rechtsklick → Informationen)
5. Prüfe: Enthält der Pfad „OneDrive“? Falls ja: Du bist im Cloud-Ordner!	5. Achtung: iCloud-Ordner meiden!

Konkret am Terminal

Windows (CMD / PowerShell)	macOS (Terminal)

`REM Benutzerordner anzeigen`
`echo %USERPROFILE%`
 	
`REM Projektordner erstellen`
`mkdir %USERPROFILE%\projekte`
	`# Benutzerordner anzeigen`
	`echo $HOME`
	
	`# Projektordner erstellen`
	`mkdir -p ~/projekte`

Modul B: Projektordner & Struktur

Das Problem

Wer Dateien unsortiert ablegt, verliert den Überblick und erzeugt Pfad-Fehler. Besonders bei ML-Projekten, wo Daten, Notebooks, trainierte Modelle und Deployment-Skripte zusammenspielen müssen, ist Struktur unverzichtbar.

Die Regel

Jedes Projekt bekommt einen eigenen Ordner mit fester Unterstruktur. Immer. Ohne Ausnahme.

⚠ Typische Falle

Relative Pfade funktionieren nur, wenn das Notebook im erwarteten Ordner liegt. Wenn du ein Notebook vom Desktop aus öffnest, zeigt „./data/“ ins Leere. Öffne immer den Projektordner in VS Code — nicht einzelne Dateien!

Schritt für Schritt

1. Öffne im Datei-Explorer / Finder deinen projekte-Ordner.
2. Erstelle darin einen neuen Ordner für dein aktuelles Projekt, z. B. sklearn-kurs.
3. Erstelle innerhalb von sklearn-kurs vier Unterordner: data, notebooks, models, scripts.
4. Speichere ab jetzt jede Datei bewusst im richtigen Unterordner.
5. Teste: Öffne den Projektordner in VS Code und prüfe, ob die Struktur in der Seitenleiste sichtbar ist.

Konkret am Rechner

Windows (Explorer)	macOS (Finder)
1. Explorer → C:\Users\name\projekte	1. Finder → /Users/name/projekte
2. Rechtsklick → Neu → Ordner → „sklearn-kurs“	2. Rechtsklick → Neuer Ordner → „sklearn-kurs“
3. Doppelklick auf sklearn-kurs	3. Doppelklick auf sklearn-kurs
4. Vier neue Ordner erstellen: data, notebooks, models, scripts	4. Vier neue Ordner erstellen: data, notebooks, models, scripts
5. Rechtsklick auf sklearn-kurs → „Mit Code öffnen“	5. Rechtsklick auf sklearn-kurs → „Open with Code“

Konkret am Terminal

Windows (CMD / PowerShell)	macOS (Terminal)

`cd %USERPROFILE%\projekte`
`mkdir sklearn-kurs`
`cd sklearn-kurs`
`mkdir data notebooks models scripts`
 	
`REM Ergebnis prüfen`
`dir`
	`cd ~/projekte`
	`mkdir sklearn-kurs`
	`cd sklearn-kurs`
	`mkdir data notebooks models scripts`
	
	`# Ergebnis prüfen`
	`ls -la`

Modul C: Datei-Explorer & VS Code

Das Problem

Viele Studierende öffnen einzelne Dateien in VS Code statt den ganzen Projektordner. Das führt dazu, dass VS Code den falschen Arbeitsbereich hat, relative Pfade nicht funktionieren und das Python-Environment nicht erkannt wird.

Die Regel

Du öffnest immer den Projektordner in VS Code — nie eine einzelne Datei. Der Ordner bestimmt den Arbeitsbereich.

⚠ Typische Falle

Häufiger Fehler: Datei → Öffnen → einzelne .ipynb-Datei auswählen. Dann hat VS Code keinen Ordner-Kontext, findet keine relativen Pfade und erkennt das Environment nicht. Immer: Datei → Ordner öffnen!

Schritt für Schritt

1. Navigiere im Datei-Explorer / Finder zu deinem Projektordner (z. B. sklearn-kurs).
2. Rechtsklick auf den Ordner → „Mit Code öffnen“ (Windows) bzw. „Open with Code“ (Mac).
3. In VS Code: Prüfe die Seitenleiste links — dort sollte dein Projektordner mit allen Unterordnern sichtbar sein.
4. Neue Dateien erstellst du direkt in VS Code: Rechtsklick auf den gewünschten Ordner in der Seitenleiste → „Neue Datei“.
5. Wichtig: Wenn VS Code oben in der Titelleiste nur einen Dateinamen zeigt (nicht den Ordnernamen), hast du eine einzelne Datei geöffnet — schließen und den Ordner öffnen!

Konkret am Rechner

Windows (Explorer)	macOS (Finder)
1. Explorer → zum Projektordner navigieren	1. Finder → zum Projektordner navigieren
2. Rechtsklick auf den Ordner → „Mit Code öffnen“	2. Rechtsklick auf den Ordner → „Open with Code“
3. VS Code: Seitenleiste zeigt die Ordnerstruktur	3. VS Code: Seitenleiste zeigt die Ordnerstruktur
4. Neue Datei: Rechtsklick in der Seitenleiste → „Neue Datei“	4. Neue Datei: Rechtsklick in der Seitenleiste → „New File“
5. Titelleiste prüfen: Ordnername muss sichtbar sein	5. Titelleiste prüfen: Ordnername muss sichtbar sein

Modul C, Teil II: Terminal-Grundlagen

Das Problem

Viele Studierende haben noch nie ein Terminal benutzt und empfinden es als einschüchternd. Aber: Environment-Aktivierung, Jupyter-Start und Paketinstallation funktionieren nur über das Terminal. Es ist kein optionales Werkzeug — es ist das Cockpit.

Die Regel

Du musst vier Befehle beherrschen: Wo bin ich? Was ist hier? Gehe dorthin. Führe etwas aus.

⚠ ***Typische Falle***

Häufiger Fehler: Das Terminal steht in C:\Users\name\ oder ~, aber das Environment und die Dateien liegen woanders. Immer zuerst cd in den richtigen Ordner!

Schritt für Schritt

1. *Öffne ein Terminal (Windows: Suche nach "cmd" oder "PowerShell"; Mac: Spotlight → "Terminal").*
2. *Finde heraus, wo du bist: pwd (Mac) oder cd ohne Argument (Windows).*
3. *Navigiere in deinen Projektordner: cd pfad/zum/ordner.*
4. *Lass dir den Inhalt anzeigen: ls (Mac) oder dir (Windows).*
5. *Tipp: In VS Code kannst du ein Terminal direkt im Projektordner öffnen (Menü → Terminal → Neues Terminal).*

Konkret am Terminal

Windows (CMD / PowerShell)

```
REM Wo bin ich?
cd

REM In den Projektordner wechseln
cd %USERPROFILE%\projekte\sklearn-kurs

REM Inhalt anzeigen
dir

REM Einen Ordner nach oben
cd ..
```

macOS (Terminal)

```
# Wo bin ich?
pwd

# In den Projektordner wechseln
cd ~/projekte/sklearn-kurs

# Inhalt anzeigen
ls -la

# Einen Ordner nach oben
cd ..
```

Modul D: Python-Environments (VS Code)

Das Problem

Ohne Environment installiert pip Pakete systemweit — was zu Versionskonflikten führt, oder schlimmer: auf Firmenrechnern schlicht nicht funktioniert. Außerdem: Wenn VS Code das falsche Environment nutzt, findet es deine installierten Pakete nicht.

Die Regel

Jedes Projekt hat ein eigenes Environment. Du erstellst und verwaltest es über die „Python Environments“-Extension in VS Code.

⚠ Typische Falle

Das Environment ist an den Projektordner gebunden. Wenn du VS Code mit einem anderen Ordner öffnest oder eine einzelne Datei öffnest, ist das Environment nicht aktiv! Immer: zuerst den richtigen Projektordner öffnen, dann arbeiten.

Schritt für Schritt mit Explorer oder Finder

1. Stelle sicher, dass die Extension „Python Environments“ in VS Code installiert ist (Extensions-Seitenleiste → suche nach „Python Environments“).
2. Öffne deinen Projektordner in VS Code.
3. Klicke in der Seitenleiste auf das Python-Environments-Symbol (oder öffne die Command Palette mit Ctrl+Shift+P / Cmd+Shift+P und suche „Create Environment“).
4. Wähle „Venv“ als Environment-Typ und den gewünschten Python-Interpreter.
5. VS Code erstellt automatisch einen .venv-Ordner in deinem Projekt.
6. Prüfe: Unten links in der Statusleiste von VS Code sollte jetzt der Name deines Environments stehen (z. B. .venv oder ml-env).
7. Pakete installieren: Öffne das VS Code-Terminal (Menü → Terminal → Neues Terminal) — das Environment ist dort automatisch aktiviert. Tippe: pip install scikit-learn pandas jupyter gradio.

Konkret am Rechner

Windows (Explorer)	macOS (Finder)
1. VS Code öffnen mit Projektordner	1. VS Code öffnen mit Projektordner
2. Extensions: „Python Environments“ installieren	2. Extensions: „Python Environments“ installieren
3. Ctrl+Shift+P → „Create Environment“	3. Cmd+Shift+P → „Create Environment“
4. Typ: „Venv“ wählen	4. Typ: „Venv“ wählen
5. Python-Version auswählen (z. B. Python 3.11)	5. Python-Version auswählen (z. B. Python 3.11)
6. Statusleiste unten links: Environment-Name prüfen	6. Statusleiste unten links: Environment-Name prüfen

Modul D, Teil II: Python-Environments (VS Code)

Schritt für Schritt für den Terminal

Die Regel

Jedes Projekt hat ein eigenes Environment. Du aktivierst es BEVOR du Jupyter startest oder pip benutzt. Immer.

⚠ Typische Falle

Das Environment muss bei jeder neuen Terminal-Sitzung erneut aktiviert werden — es bleibt nicht dauerhaft aktiv! Wenn (ml-env) fehlt, ist nichts aktiviert.

1. *Öffne ein Terminal und navigiere in deinen Projektordner.*
2. *Erstelle ein Environment (einmalig).*
3. *Aktiviere es (jedes Mal, wenn du arbeitest).*
4. *Prüfe, ob es aktiv ist: Du siehst (ml-env) am Anfang der Zeile.*
5. *Installiere dann deine Pakete: pip install scikit-learn pandas jupyter gradio.*
6. *Starte Jupyter aus dem aktiven Environment heraus.*

Konkret am Terminal

Windows (CMD / PowerShell)

```
cd %USERPROFILE%\projekte\sklearn-kurs

REM Environment erstellen (einmalig)
python -m venv ml-env

REM Environment aktivieren
ml-env\Scripts\activate

REM Pakete installieren
pip install scikit-learn pandas jupyter

REM Jupyter starten
jupyter notebook
```

macOS (Terminal)

```
cd ~/projekte/sklearn-kurs

# Environment erstellen (einmalig)
python3 -m venv ml-env

# Environment aktivieren
source ml-env/bin/activate

# Pakete installieren
pip install scikit-learn pandas jupyter

# Jupyter starten
jupyter notebook
```

Modul E: Speicherplatz & Systemhygiene

Das Problem

Zu wenig Speicherplatz führt zu kryptischen Fehlern: Jupyter stürzt ab, pip-Installationen schlagen fehl, Modelle können nicht gespeichert werden. Besonders auf Firmenrechnern mit kleinen SSDs wird das schnell kritisch.

Die Regel

Prüfe vor jeder Arbeitssession, ob mindestens 5 GB frei sind. Räume regelmäßig auf.

⚠ Typische Falle

Jupyter-Notebooks mit Outputs (besonders Bilder, Plots, DataFrames) können hunderte MB groß werden. Lösche Outputs vor dem Teilen immer! Auch alte .venv-Ordner können jeweils mehrere GB belegen.

Schritt für Schritt

1. Prüfe deinen freien Speicherplatz (siehe Anleitung rechts).
2. Identifiziere Platzfresser: große Downloads, alte Installationen, alte Environments.
3. In Jupyter: Vor dem Speichern eines Notebooks → Menü → Kernel → Restart & Clear Output. Das kann die Dateigröße drastisch reduzieren.
4. Lösche nicht benötigte alte Environments (den .venv-Ordner löschen reicht).

Konkret am Rechner

Windows (Explorer)	macOS (Finder)
1. Explorer öffnen → „Dieser PC“	1. Apple-Menü → Über diesen Mac → Festplatten
2. Rechtsklick auf C: → Eigenschaften	2. (oder: Systemeinstellungen → Allgemein → Speicher)
3. Freier Speicher wird als Balken angezeigt	3. Freier Speicher wird farblich dargestellt
4. Aufräumen: Einstellungen → System → Speicher	4. Aufräumen: Große Dateien im Finder suchen (Cmd+F → Größe)
5. Alte .venv-Ordner: Im Explorer löschen	5. Alte .venv-Ordner: Im Finder löschen

Konkret am Terminal

Windows (CMD / PowerShell)	macOS (Terminal)

`REM Freien Speicherplatz anzeigen`
`wmic logicaldisk get size,freespace,caption`
 	
`REM Größe eines Ordners prüfen`
`dir /s %USERPROFILE%\projekte`
 	
`REM Altes Environment löschen`
`rmdir /s /q altes-env`
	`# Freien Speicherplatz anzeigen`
	`df -h /`
	
	`# Größe eines Ordners prüfen`
	`du -sh ~/projekte/*`
	
	`# Altes Environment löschen`
	`rm -rf altes-env`

Modul F: Dateien orchestrieren

Das Problem

Ein ML-Projekt besteht nie aus einer einzigen Datei. Es gibt Trainingsskripte, gespeicherte Modelle, Deployment-Code und Daten. Wenn diese Dateien nicht sauber miteinander verknüpft sind, funktioniert das Zusammenspiel nicht — besonders beim Gradio-Deployment.

Die Regel

Du musst den Datenfluss deines Projekts kennen: Welche Datei erzeugt was, und welche Datei liest es ein? Alles verbindet sich über Pfade und Dateinamen.

⚠ Typische Falle

Häufiger Fehler: train.py und app.py verwenden unterschiedliche Pfade für das Modell, oder app.py wird außerhalb des Projektordners geöffnet. Relative Pfade sind immer relativ zum VS Code-Arbeitsbereich!

Schritt für Schritt

1. Zeichne dir den Datenfluss auf (auch auf Papier): Daten → Training → Modell-Datei → App.
2. Stelle sicher, dass train.py das Modell in models/ speichert (z. B. models/model.pkl).
3. Stelle sicher, dass app.py das Modell aus models/ lädt — mit demselben Pfad.
4. Wichtig: Beide Skripte müssen im selben VS Code-Arbeitsbereich liegen, damit die relativen Pfade stimmen.

Konkret am Rechner

Windows (Explorer)	macOS (Finder)
1. Projektstruktur in VS Code prüfen:	1. Im train.py:
2. sklearn-kurs/	2. import joblib
3. ├─ data/train.csv	3. joblib.dump(model, 'models/model.pkl')
4. ├─ notebooks/exploration.ipynb	
5. ├─ models/model.pkl ← von train.py erzeugt	5. Im app.py:
6. ├─ scripts/train.py ← liest data/, schreibt models/	6. import joblib
7. ├─ scripts/app.py ← liest models/, startet Gradio	7. model = joblib.load('models/model.pkl')
8. └─ .venv/ ← das Environment	8. Beide Dateien im selben VS Code-Projekt!

Konkret am Terminal

Windows (CMD / PowerShell)

```
sklearn-kurs/
├── data/
│   └── train.csv
├── notebooks/
│   └── exploration.ipynb
├── models/
│   └── model.pkl   ← erzeugt von train.py
├── scripts/
│   ├── train.py    ← liest data/, schreibt models/
│   └── app.py      ← liest models/, startet Gradio
└── ml-env/          ← das Environment
```

macOS (Terminal)

```
# Im train.py:
import joblib
joblib.dump(model, '../models/model.pkl')

# Im app.py:
import joblib
model = joblib.load('../models/model.pkl')

# Starten aus dem scripts-Ordner:
cd ~/projekte/sklearn-kurs/scripts
python app.py
```

Teil 3: Der Preflight-Check (Kurzversion)

Diese Kurzcheckliste kannst du vor jeder Arbeitssession in 2–3 Minuten durchgehen. Drucke sie aus oder halte sie als Lesezeichen bereit.

✓	Vor jeder Session prüfen	Kann ich?
☐	Ich bin im richtigen Projektordner (nicht Desktop, nicht Downloads, nicht OneDrive).	Ja / Nein
☐	Ich habe den Projektordner in VS Code geöffnet (nicht eine einzelne Datei).	Ja / Nein
☐	In der VS Code-Statusleiste steht mein Python-Environment (z. B. .venv).	Ja / Nein
☐	Ich habe mindestens 5 GB freien Speicherplatz.	Ja / Nein
☐	Meine neue Datei liegt im richtigen Unterordner (data/, notebooks/, scripts/, models/).	Ja / Nein
☐	Ich weiß, welche Dateien mein heutiges Skript liest und schreibt — und wo sie liegen.	Ja / Nein

Wenn du alle sechs Punkte abhaken kannst, bist du startklar. Falls nicht: Löse das Problem jetzt — nicht erst, wenn es den Unterricht blockiert. Das ist keine Schwäche, das ist professionelles Arbeiten.