

Zusammenfassung *(automatisiert erstellt)*

Das Skript liest Patientendaten ein, prüft auf Existenz, Format, Anzahl von Zeilen sowie auf leere Zellen. Bei Fehlern wird stets eine `osa_result.csv` mit Hinweis ausgegeben, um den Workflow robust zu gestalten. Es berechnet einen Score anhand von Antworten und weist Risikokategorien zu. Das bisherige `applymap` wird durch `str.strip()` auf alle Objekt-Spalten ersetzt, für Kompatibilität. Das Hauptprogramm ist klar strukturiert, führt bei Fehlern sofort eine standardisierte Resultate-Datei aus. Bei Lösungserfordernissen für Batch-Verarbeitung oder mehrere Patienten passen wir das gern an. Das Skript sorgt so für konstante Rückmeldungen bei allen Fehlerfällen.

Systemprompt

Du bist ein intelligentes und unterstützendes KI-Assistenzsystem für alle Hochschulangehörigen der HAWK Hildesheim/Holzminden/Göttingen. Dein Ziel ist es, Studierende, Lehrende, Forschende und Mitarbeitende in ihrer akademischen Arbeit, beim Lernen, Forschen, Lehren und verwalterischen Aufgaben zu unterstützen. Dabei förderst du kollaboratives Arbeiten, wissenschaftliches Denken und eine kreative Problemlösung. Beziehe dich auf wissenschaftliche Methoden und Theorien, argumentiere sachlich und reflektiere kritisch. Sei objektiv und verzichte auf unbegründete Meinungen. Fördere akademische Integrität und unterstütze keine Plagiate. Sei inklusiv, wertschätzend und respektiere Vielfalt.

Verlauf:

Bousfiha, Tanae:

hallo meine container auf xnat failed und sagt

Traceback (most recent call last):

```
File "/app/OSA_xnat.py", line 104, in  
    main()
```

```
File "/app/OSA_xnat.py", line 63, in main  
    if not input_file_path or not
```

```
os.path.exists(input_file_path):
```

UnboundLocalError: local variable

'input_file_path'; referenced before
assignment

meine Skript sieht so aus

```
import pandas as pd
```

```
import os
```

```
import sys
```

```
import subprocess
```

```
input_file_path = sys.argv[1] if len(sys.argv) > 1 else
```

```
None # Default input file path
```

```
output_dir = sys.argv[2] if len(sys.argv) > 2 else
```

```
"/app/output" # Default output directory
```

```
output_file = "osa_result.csv"
```

```
def calculate_osa_risk_single(row):
```

```
    positive_answers = {"yes",
```

```
                        "JA", "ja", "x",
```

```
                        "whar", "true", "1",
```

```
                        "wahr", "X",}
```

```
    negative_answers = {"no",
```

```
                        "nein", "0", "falsch",
```

```
                        "false", "nix", "nichts",
```

```
                        "nichts davon",}
```

```
    man_words = {"m", "mann",
```

```
                 "maennlich", "männlich",
```

```
                 "manlich",}
```

```
woman_words = {'w', 'frau',
                'weiblich', 'woman',
                'feminine'}
```

```
row = {k.strip().lower(): str(v).strip().lower() for k, v in
row.items() if pd.notnull(v)}
```

```
score = 0
```

```
for value in row.values():
    if value in positive_answers or value in man_words:
        score += 1
    elif value in negative_answers or value in
woman_words:
        score += 0
    else:
        pd.DataFrame([{'stopbang_score':
'Fehler', 'osa_risiko':
'Keine Eingabedatei gefunden'}])
```

```
risiko = 'hoch' if score >= 5 else
'mittel' if score >= 3 else
'niedrig'
return pd.DataFrame([{'stopbang_score':
score, 'osa_risiko': risiko}])
```

```
def dataformat():
    data_format = {'csv', 'tsv',
'txt'}
    if input_file_path.endswith(data_format): # https://stackoverflow.com/questions/34192705/python-how-to-check-if-cell-in-csv-file-is-empty
        with open(input_file_path, 'r',
encoding='utf-8'):
            reader = pd.read(input_file_path, sep=None,
engine='python')
            if reader.empty:
```

```

        return
pd.DataFrame([{"stopbang_score":
"Fehler", "osa_risiko":
"Keine Eingabedatei gefunden"}])
    else:
        return reader

def find_void(input_file_path, delimiter=','):
    df = pd.read_csv(input_file_path, delimiter=delimiter,
dtype=str, encoding='utf-8', na_filter=False)
    df = df.applymap(lambda x: x.strip() if isinstance(x, str)
else x)
    df = df.replace(',', pd.NA)
    if df.isna().any().any():
        print("Die Datei enthält leere Zellen. Bitte
nochmal ausfüllen")

def result_file():
    with open(output_file, 'w',
encoding='utf-8') as f:
        f.write("stopbang_score,osa_risiko\n")
        f.write("Fehler,Keine Eingabedatei
gefunden\n")

def main():
    print("Start Osa-Analyse")
    os.makedirs(output_dir, exist_ok=True)

    # Sicherstellen, dass eine gültige Datei übergeben
wird
    if not input_file_path or not
os.path.exists(input_file_path):
        print("Fehler: Keine oder ungültige
Eingabedatei angegeben.")
    return

```

```

# Falls ein Verzeichnis übergeben wurde, erste
passende Datei darin suchen
if os.path.isdir(input_file_path):
    found = False
    for fname in os.listdir(input_file_path):
        if fname.lower().endswith(('".csv",
".tsv", ".txt"')):
            input_file_path = os.path.join(input_file_path,
fname)
            found = True
            break
    if not found:
        print("Fehler: Keine CSV-, TSV- oder
TXT-Datei im Verzeichnis gefunden.")
        return

```

```

ext = os.path.splitext(input_file_path)[1].lower()
if ext == &#039;.csv&#039;:
    delimiter = &#039;,&#039;
elif ext == &#039;.tsv&#039;:
    delimiter = &#039;\t&#039;
else:
    delimiter = &#039;,&#039;

```

```

find_void(input_file_path, delimiter=delimiter)

```

```

df = pd.read_csv(input_file_path, delimiter=delimiter,
dtype=str, encoding=&#039;utf-8&#039;)
if df.empty or len(df) != 1:
    print("Fehler: Datei ist leer oder enthält mehr
als einen Patienten.")
    return

```

```

result_df =
calculate_osa_risk_single(df.iloc[0].to_dict())

```

```

output_path = os.path.join(output_dir, output_file)

```

```

    result_df.to_csv(output_path, index=False)
    print(f"Ergebnis gespeichert unter:
{output_path}",)

    subprocess.run(f"ls -al {output_dir}",
shell=True)

if __name__ == "__main__":
    main()

```

HAWKI (gpt-4.1):

Dein Fehler

...

UnboundLocalError: local variable 'input_file_path' referenced before assignment
...

tritt auf, weil `input_file_path` innerhalb von `main()` ****neu zugewiesen**** wird (in der if-Verzeichnis-Prüfung), ohne es mit `global` zu deklarieren. Dadurch bezieht sich Python nach dieser Zuweisung innerhalb von `main()` nicht mehr auf die Modulvariable, sondern meint eine lokale Variable – die dann aber teilweise noch ****vor dieser lokalen Zuweisung**** referenziert wird.

****Konkret**:**

```

```python
if os.path.isdir(input_file_path):
 # HIER wird input_file_path verändert (= lokale Variable!)
 input_file_path = os.path.join(input_file_path, fname)
...

```

Damit ist ab hier `input\_file\_path` **\*\*lokal\*\*** in der Funktion, also geht das Referenzieren weiter oben schief.

## ## Lösungsmöglichkeiten

### ### 1. Verwende lokale Variablen innerhalb von `main()`

Lade `input\_file\_path` am Anfang der Funktion in eine **\*\*lokale Variable\*\***, verwende **\*\*nur diese\*\*** innerhalb von `main()`. Das ist sauberer als mit `global` zu arbeiten.

```

```python
def main():
    print("Start Osa-Analyse")
    os.makedirs(output_dir, exist_ok=True)

    # Lokale Kopie!
    input_path = input_file_path

    # Sicherstellen, dass eine gültige Datei übergeben wird
    if not input_path or not os.path.exists(input_path):
        print("Fehler: Keine oder ungültige Eingabedatei angegeben.")
        return

    # Falls ein Verzeichnis übergeben wurde, erste passende Datei darin suchen
    if os.path.isdir(input_path):
        found = False
        for fname in os.listdir(input_path):
            if fname.lower().endswith((".csv", ".tsv", ".txt")):
                input_path = os.path.join(input_path, fname)
                found = True
                break
        if not found:
            print("Fehler: Keine CSV-, TSV- oder TXT-Datei im Verzeichnis gefunden.")
            return

    ext = os.path.splitext(input_path)[1].lower()
    if ext == '.csv':
        delimiter = ','
    elif ext == '.tsv':
        delimiter = '\t'
    else:
        delimiter = ','

    find_void(input_path, delimiter=delimiter)

    df = pd.read_csv(input_path, delimiter=delimiter, dtype=str, encoding='utf-8')
    if df.empty or len(df) != 1:

```

```
print("Fehler: Datei ist leer oder enthält mehr als einen Patienten.")
return
```

```
result_df = calculate_osa_risk_single(df.iloc[0].to_dict())
```

```
output_path = os.path.join(output_dir, output_file)
result_df.to_csv(output_path, index=False)
print(f"Ergebnis gespeichert unter: {output_path}")
```

```
subprocess.run(f"ls -al {output_dir}", shell=True)
...
```

****Vorteil:**** Keine Probleme mehr mit scope.

****Extra-Tipp:**** Du kannst ``input_file_path``, ``output_dir``, etc. direkt in ``main()`` aus den ``sys.argv`` ziehen, statt außerhalb zu definieren.

2. Argumente sauber in der ``main()``-Signatur anlegen

Du kannst dein Skript auch so umbauen (stärker modularisiert):

```
```python
def main(input_file_path, output_dir="/app/output"):
 # ... wie oben, alles mit input_file_path (jetzt Parameter)

if __name__ == "__main__":
 input_file_path = sys.argv[1] if len(sys.argv) > 1 else None
 output_dir = sys.argv[2] if len(sys.argv) > 2 else "/app/output"
 main(input_file_path, output_dir)
...

```

---

### 3. Nutze **\*\*nicht\*\*** ``global``

Die Nutzung von ``global`` wäre **\*\*nicht\*\*** zu empfehlen, da das Design unsauber wäre.

---



## Schnell-Fazit

Dein Bug kommt von Python-Scope durch Neu-Zuweisung in `main()`.

Setze ganz am Anfang von `main()` einfach

```
```python
input_path = input_file_path
```
```

und ab dann verwende nur noch `input\_path` in `main()`.

Dadurch bist du Scope-Problemen aus dem Weg gegangen.

---

### **\*\*Hinweis\*\***

Der Fehler war **\*\*kein XNAT-spezifisches Problem\*\***, sondern rein Python-Logik!

Wenn du danach wieder Fehler im **\*\*XNAT-Container-Kontext\*\*** bekommst, kann das an Pfaden/Daten/Dateirechten liegen — dann gern mit Log-Auszügen davon nachfragen.

---

Wenn du weitere Unterstützung benötigst (z./B. bei DataFrame Container-Aufruf), poste gern deinen vollständigen Container-Call oder Beispielinput!

**Bousfiha, Tanae:**

so ist richtig

```
import pandas as pd
```

```
import os
```

```
import sys
```

```
import subprocess
```

```
input_file_path = sys.argv[1] if len(sys.argv) > 1 else None # Default input file path
output_dir = sys.argv[2] if len(sys.argv) > 2 else "/app/output" # Default
output directory
output_file = "osa_result.csv";
```

```

def calculate_osa_risk_single(row):
 positive_answers = {'yes', 'JA', 'ja', 'x', 'whar', 'true', '1', 'wahr', 'X'}
 negative_answers = {'no', 'nein', '0', 'falsch', 'false', 'nix', 'nichts', 'nichts davon'}
 man_words = {'m', 'mann', 'maennlich', 'männlich', 'manlich'}
 woman_words = {'w', 'frau', 'weiblich', 'woman', 'feminine'}

 row = {k.strip().lower(): str(v).strip().lower() for k, v in row.items() if pd.notnull(v)}

 score = 0

 for value in row.values():
 if value in positive_answers or value in man_words:
 score += 1
 elif value in negative_answers or value in woman_words:
 score += 0
 else:
 pd.DataFrame({'stopbang_score': 'Fehler',
 'osa_risiko': 'Keine Eingabedatei gefunden'})

 risiko = 'hoch' if score >= 5 else 'mittel' if score >= 3
 else 'niedrig'
 return pd.DataFrame({'stopbang_score': score,
 'osa_risiko': risiko})

def dataformat():
 data_format = {'csv', 'tsv', 'txt'}
 if input_file_path.endswith(data_format): # https://stackoverflow.com/questions/34192705/python-how-to-check-if-cell-in-csv-file-is-empty
 with open(input_file_path, 'r', encoding='utf-8'):
 reader = pd.read(input_file_path, sep=None, engine='python')

```

```

 if reader.empty:
 return pd.DataFrame([{"stopbang_score": "Fehler",
 "osa_risiko": "Keine Eingabedatei gefunden"}])
 else:
 return reader

```

```

def find_void(input_file_path, delimiter=','):
 df = pd.read_csv(input_file_path, delimiter=delimiter, dtype=str,
encoding='utf-8', na_filter=False)
 df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)
 df = df.replace('', pd.NA)
 if df.isna().any().any():
 print("Die Datei enthält leere Zellen. Bitte nochmal ausfüllen")

```

```

def result_file():
 with open(output_file, 'w', encoding='utf-8') as f:
 f.write("stopbang_score,osa_risiko\n")
 f.write("Fehler,Keine Eingabedatei gefunden\n")

```

```

def main():
 print("Start Osa-Analyse")
 os.makedirs(output_dir, exist_ok=True)

 # Lokale Kopie!
 input_path = input_file_path

 # Sicherstellen, dass eine gültige Datei übergeben wird
 if not input_path or not os.path.exists(input_path):
 print("Fehler: Keine oder ungültige Eingabedatei angegeben.")
 return

 # Falls ein Verzeichnis übergeben wurde, erste passende Datei darin suchen
 if os.path.isdir(input_path):
 found = False
 for fname in os.listdir(input_path):

```

```

 if fname.lower().endswith(('".csv", ".tsv",
".txt")):
 input_path = os.path.join(input_path, fname)
 found = True
 break
 if not found:
 print(""Fehler: Keine CSV-, TSV- oder TXT-Datei im Verzeichnis
gefunden."")
 return

 ext = os.path.splitext(input_path)[1].lower()
 if ext == ''.csv':
 delimiter = '','
 elif ext == ''.tsv':
 delimiter = ''\t'
 else:
 delimiter = '','

 find_void(input_path, delimiter=delimiter)

 df = pd.read_csv(input_path, delimiter=delimiter, dtype=str,
encoding='utf-8')
 if df.empty or len(df) != 1:
 print(""Fehler: Datei ist leer oder enthält mehr als einen Patienten."")
 return

 result_df = calculate_osa_risk_single(df.iloc[0].to_dict())

 output_path = os.path.join(output_dir, output_file)
 result_df.to_csv(output_path, index=False)
 print(f""Ergebnis gespeichert unter: {output_path}"")

 subprocess.run(f""ls -al {output_dir}"", shell=True)

if __name__ == '__main__':
 input_file_path = sys.argv[1] if len(sys.argv) > 1 else None
 output_dir = sys.argv[2] if len(sys.argv) > 2 else '"/app/output"'

```

main()

## **HAWKI (gpt-4.1):**

Dein Ansatz ist schon auf dem richtigen Weg, aber im letzten Teil gibt es noch einen Fehler bezüglich der Einrückung und der Reihenfolge am Skriptende.

**\*\*Das Problem:\*\***

Du hast das Block-Statement

```
```python
if __name__ == "__main__":
    input_file_path = sys.argv[1] if len(sys.argv) > 1 else None
    output_dir = sys.argv[2] if len(sys.argv) > 2 else "/app/output"
    main()
```
```

**\*\*in die main()-Funktion eingerückt!\*\***

Dadurch wird es **\*\*nie\*\*** ausgeführt, wenn du das Skript von außen startest – sondern nur, wenn main() explizit aufgerufen wird.

---

## **## Korrekte Version**

Du solltest die folgenden Punkte beachten:

1. Der `if __name__ == "__main__":`-Block muss **\*\*am obersten Level\*\*** (also ohne Einrückung) stehen.
2. Variablen wie `input_file_path` und `output_dir` sollten entweder NUR im Hauptblock gesetzt oder als Argumente an `main` übergeben werden, damit sie nicht doppelt gesetzt werden.
3. Die Variable `output_file` kann, wenn sie konstant ist, ruhig außerhalb bleiben oder als Default-Argument übergeben werden.

## **### Konkreter Vorschlag:**

```
```python
```

```

import pandas as pd
import os
import sys
import subprocess

output_file = "osa_result.csv"

def calculate_osa_risk_single(row):
    positive_answers = {"yes", "JA", "ja", "x", "whar", "true", "1", "wahr", "X"}
    negative_answers = {"no", "nein", "0", "falsch", "false", "nix", "nichts", "nichts davon"}
    man_words = {"m", "mann", "maennlich", "männlich", "manlich"}
    woman_words = {"w", "frau", "weiblich", "woman", "feminine"}
    row = {k.strip().lower(): str(v).strip().lower() for k, v in row.items() if pd.notnull(v)}
    score = 0
    for value in row.values():
        if value in positive_answers or value in man_words:
            score += 1
        # elif value in negative_answers or value in woman_words: # <- das brauchst du
        # nicht, da score unverändert bleibt
        #     score += 0
        # Andere Fälle werden (evtl. zu Recht) still ignoriert
    risiko = "hoch" if score >= 5 else "mittel" if score >= 3 else "niedrig"
    return pd.DataFrame([{"stopbang_score": score, "osa_risiko": risiko}])

def find_void(input_file_path, delimiter=','):
    df = pd.read_csv(input_file_path, delimiter=delimiter, dtype=str, encoding='utf-8',
na_filter=False)
    df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)
    df = df.replace("", pd.NA)
    if df.isna().any().any():
        print("Die Datei enthält leere Zellen. Bitte nochmal ausfüllen")

def main(input_file_path, output_dir="/app/output"):
    print("Start Osa-Analyse")
    os.makedirs(output_dir, exist_ok=True)
    input_path = input_file_path
    if not input_path or not os.path.exists(input_path):

```

```
print("Fehler: Keine oder ungültige Eingabedatei angegeben.")
return
```

```
if os.path.isdir(input_path):
    found = False
    for fname in os.listdir(input_path):
        if fname.lower().endswith((".csv", ".tsv", ".txt")):
            input_path = os.path.join(input_path, fname)
            found = True
            break
    if not found:
        print("Fehler: Keine CSV-, TSV- oder TXT-Datei im Verzeichnis gefunden.")
        return
```

```
ext = os.path.splitext(input_path)[1].lower()
if ext == '.csv':
    delimiter = ','
elif ext == '.tsv':
    delimiter = '\t'
else:
    delimiter = ','
```

```
find_void(input_path, delimiter=delimiter)
```

```
df = pd.read_csv(input_path, delimiter=delimiter, dtype=str, encoding='utf-8')
if df.empty or len(df) != 1:
    print("Fehler: Datei ist leer oder enthält mehr als einen Patienten.")
    return
```

```
result_df = calculate_osa_risk_single(df.iloc[0].to_dict())
output_path = os.path.join(output_dir, output_file)
result_df.to_csv(output_path, index=False)
print(f"Ergebnis gespeichert unter: {output_path}")
subprocess.run(f"ls -al {output_dir}", shell=True)
```

```
if __name__ == "__main__":
    input_file_path = sys.argv[1] if len(sys.argv) > 1 else None
    output_dir = sys.argv[2] if len(sys.argv) > 2 else "/app/output"
```

```
main(input_file_path, output_dir)
...
```

****Zusätzliche Hinweise****

- Statt die Argumente zweimal zu setzen, werden sie nur im Hauptblock ausgelesen und an main übergeben.
- Die Funktion `dataformat` und `result_file` wird in deinem Code nie verwendet. Du kannst sie entfernen oder ggf. bei Bedarf anpassen.
- Achte darauf, dass du bei der Verwendung im Container korrekte ****Dateipfade und Rechte**** hast.

Falls du noch weitere Anpassungen brauchst, z.B. mehrere Patienten in einer Datei oder ausführlichere Fehlerbehandlung, gerne nachfragen!

Bousfiha, Tanae:

stdout:

Start Osa-Analyse

Fehler: Datei ist leer oder enthält mehr als einen Patienten.

stdin:

/app/OSA_xnat.py:25: FutureWarning: DataFrame.applymap has been deprecated.
Use DataFrame.map instead.

```
df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)
```

HAWKI (gpt-4.1):

Gern helfe ich dir mit der Analyse und den nächsten Schritten!

****1. Fehlermeldung:****

...

Fehler: Datei ist leer oder enthält mehr als einen Patienten.

...

****Bedeutung:****

Dein Skript erwartet, dass die eingelesene Datei ****genau eine Datenzeile**** enthält (also `len(df) == 1`). Es gibt zwei Möglichkeiten, warum der Fehler kommt:

- Die Datei ist leer oder
- **Es sind mehrere Zeilen (Patienten) in der Datei**.

Prüfe also bitte:

- Ist deine Input-Datei tatsächlich leer?
- Enthält sie mehr als eine Zeile mit Patientendaten?
- Hast du den Header (Spaltennamen) korrekt und darunter exakt eine Datenzeile?

Beispiel für eine korrekte CSV:

```
```csv
gender,age,question1,question2
m,45,ja,nein
```
```

Nur **eine Zeile** mit Patientendaten.

2. FutureWarning:

```
```
```

FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.

```
df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)
```
```

Bedeutung:

Die Methode `DataFrame.applymap` ist in der genutzten Pandas-Version veraltet und wird in Zukunft entfernt.

Lösung:

Bei DataFrames nimm stattdessen:

```
```python
df = df.map(lambda x: x.strip() if isinstance(x, str) else x)
```
```

Achtung: Bisher ist `.map()` auf DataFrame-Ebene ab Pandas 2.2+ verfügbar. Für maximale Kompatibilität und Klarheit kannst du stattdessen lieber:

```
```python
for col in df.select_dtypes(['object']):
 df[col] = df[col].str.strip()
```
```

Das ist explizit und kompatibel mit allen Pandas-Versionen.

Zusammengefasst

****a) Überprüfe deinen Input:****

- Öffne die Datei im Editor.
- Siehst du mehr als eine Zeile mit Daten? (Dann `len(df) > 1``)
- Ist sie komplett leer? (`len(df) == 0``)
- Ist der Header da und darunter exakt eine Zeile?

****b) Setze das DataFrame-Aufräumen wie folgt:****

```
```python
for col in df.select_dtypes(['object']):
 df[col] = df[col].str.strip()
```
```

****Ersetze die alte `applymap``-Zeile dadurch.****

****Weiterführende Tipps:****

- Wenn du mehrere Patienten analysieren willst, entferne das `len(df) != 1``-Kriterium und iteriere über `df.iterrows()``, dann kann jede Zeile einzeln bewertet werden!
- Wenn du möchtest, gib gern Beispielinput und den genauen Aufruf deines Containers/Jobs an, dann kann ich noch gezielter helfen.

Bei Rückfragen oder weiteren Zielen (z./B. Batch-Analyse) gerne

Bousfiha, Tanae:

kann ich das skript umwandeln dass er trotzdem eine datei ausgefüllt mit
bitte die fragen nochmal antworten hochlädt

HAWKI (gpt-4.1):

Das ist ein sehr sinnvoller Wunsch — du möchtest, dass ****immer**** eine
Ergebnisdatei geschrieben wird, auch wenn die Input-Datei fehlerhaft ist (leer, mehr

als ein Patient, o.ä.). Die Datei soll dann einen Standardhinweis (z.B. „bitte die fragen nochmal beantworten“) als Ergebnis enthalten, sodass der (xnat) Workflow nicht fehlschlägt und Du in jedem Fall ein Feedback bekommst.

So geht's: Beispiel-Anpassung

Passe `main` folgendermaßen an:

```
```python
def write_error_result(output_path, message="bitte die fragen nochmal beantworten"):
 pd.DataFrame([{"stopbang_score": "Fehler", "osa_risiko":
message}]).to_csv(output_path, index=False)

def main(input_file_path, output_dir="/app/output"):
 print("Start Osa-Analyse")
 os.makedirs(output_dir, exist_ok=True)
 input_path = input_file_path

 if not input_path or not os.path.exists(input_path):
 print("Fehler: Keine oder ungültige Eingabedatei angegeben.")
 output_path = os.path.join(output_dir, output_file)
 write_error_result(output_path)
 return

 if os.path.isdir(input_path):
 found = False
 for fname in os.listdir(input_path):
 if fname.lower().endswith((".csv", ".tsv", ".txt")):
 input_path = os.path.join(input_path, fname)
 found = True
 break
 if not found:
 print("Fehler: Keine CSV-, TSV- oder TXT-Datei im Verzeichnis gefunden.")
 output_path = os.path.join(output_dir, output_file)
 write_error_result(output_path)
 return

 ext = os.path.splitext(input_path)[1].lower()
```

```

if ext == '.csv':
 delimiter = ','
elif ext == '.tsv':
 delimiter = '\t'
else:
 delimiter = ','

df = pd.read_csv(input_path, delimiter=delimiter, dtype=str, encoding='utf-8')
for col in df.select_dtypes(['object']):
 df[col] = df[col].str.strip()
if df.empty or len(df) != 1:
 print("Fehler: Datei ist leer oder enthält mehr als einen Patienten.")
 output_path = os.path.join(output_dir, output_file)
 write_error_result(output_path)
 return

result_df = calculate_osa_risk_single(df.iloc[0].to_dict())
output_path = os.path.join(output_dir, output_file)
result_df.to_csv(output_path, index=False)
print(f"Ergebnis gespeichert unter: {output_path}")
subprocess.run(f"ls -al {output_dir}", shell=True)
...

```

#### ### Hinweise

- **write\_error\_result** erzeugt immer eine gültige Ergebnistabelle mit Hinweis.
- Du bekommst egal bei welchem Fehler immer eine „osa\_result.csv“ Datei zurück.
- Für das Mapping: Du kannst gern weitere Fehlerfälle in anderen Bereichen von main so auffangen.

#### ### Optional: Rückfrage einbauen

Falls du magst, kannst du **write\_error\_result** noch einen spezifischen Fehlertext übergeben (je nach Fehlerursache).

#### ### Vollständige Integration

Am besten, du ersetzt in **main** in jedem Rückgabefall (`return`` nach Fehler) das bisherige Verhalten durch `write_error_result(...)`.

---

**\*\*Fazit:\*\***

So bleiben Container und Workflows robust und geben Nutzer:innen immer Feedback!

Wenn du magst, kann ich dir ein komplettes, lauffähiges Minimalbeispiel posten. Sag gern Bescheid!

**Bousfiha, Tanae:**

gib mir bitte das gesamte skript

**HAWKI (gpt-4.1):**

Sehr gern! Hier ist ein komplettes, robustes Skript – ausgelegt für deinen Container-Workflow mit **\*\*immer erzeugter Ergebnisdatei\*\***, auch im Fehlerfall. Die wichtigsten Verbesserungen sind:

- **\*\*Jede Fehleingabe (z./B. leere Datei, mehrere Patienten, u einer `osa\_result.csv` mit Hinweis.\*\***
- **\*\*Kompatible `.strip()`-Variante statt veraltetem `applymap`.\*\***
- **\*\*Stabile, modulare Struktur, Start ab `if \_\_name\_\_ == "\_\_main\_\_":`\*\***
- **\*\*Alle Anwendungsfälle robust abgedeckt.\*\***

```
```python
```

```
import pandas as pd
```

```
import os
```

```
import sys
```

```
import subprocess
```

```
output_file = "osa_result.csv"
```

```
def calculate_osa_risk_single(row):
```

```
    positive_answers = {"yes", "ja", "x", "whar", "true", "1", "wahr"}
```

```
    negative_answers = {"no", "nein", "0", "falsch", "false", "nix", "nichts", "nichts davon"}
```

```

man_words = {"m", "mann", "maennlich", "männlich", "manlich"}
woman_words = {"w", "frau", "weiblich", "woman", "feminine"}
row = {k.strip().lower(): str(v).strip().lower() for k, v in row.items() if pd.notnull(v)}
score = 0
for value in row.values():
    if value in positive_answers or value in man_words:
        score += 1
    # weder noch = keine Erhöhung
risiko = "hoch" if score >= 5 else "mittel" if score >= 3 else "niedrig"
return pd.DataFrame([{"stopbang_score": score, "osa_risiko": risiko}])

def write_error_result(output_path, message="bitte die fragen nochmal beantworten"):
    pd.DataFrame([{"stopbang_score": "Fehler", "osa_risiko":
message}]).to_csv(output_path, index=False)

def main(input_file_path, output_dir="/app/output"):
    print("Start Osa-Analyse")
    os.makedirs(output_dir, exist_ok=True)
    input_path = input_file_path
    output_path = os.path.join(output_dir, output_file)

    # Prüfung: Existiert Pfad?
    if not input_path or not os.path.exists(input_path):
        print("Fehler: Keine oder ungültige Eingabedatei angegeben.")
        write_error_result(output_path)
        return

    # Ist Pfad ein Verzeichnis – dann erste passende Datei auswählen
    if os.path.isdir(input_path):
        found = False
        for fname in os.listdir(input_path):
            if fname.lower().endswith((".csv", ".tsv", ".txt")):
                input_path = os.path.join(input_path, fname)
                found = True
                break
        if not found:
            print("Fehler: Keine CSV-, TSV- oder TXT-Datei im Verzeichnis gefunden.")
            write_error_result(output_path)

```

```
return
```

```
ext = os.path.splitext(input_path)[1].lower()
delimiter = ',' if ext == ".csv" else '\t' if ext == ".tsv" else ','
```

```
try:
```

```
    df = pd.read_csv(input_path, delimiter=delimiter, dtype=str, encoding='utf-8')
```

```
except Exception as e:
```

```
    print(f"Fehler beim Einlesen der Datei: {e}")
```

```
    write_error_result(output_path)
```

```
    return
```

```
# Strings trimmen
```

```
for col in df.select_dtypes(['object']):
```

```
    df[col] = df[col].str.strip()
```

```
# Prüfung auf leere Datei oder mehrere Zeilen (Patienten)
```

```
if df.empty or len(df) != 1:
```

```
    print("Fehler: Datei ist leer oder enthält mehr als einen Patienten.")
```

```
    write_error_result(output_path)
```

```
    return
```

```
# Optional: Prüfung auf leere Zellen
```

```
if df.isna().any().any():
```

```
    print("Die Datei enthält leere Zellen.")
```

```
    write_error_result(output_path, "Bitte alle Fragen beantworten und Datei erneut  
einreichen.")
```

```
    return
```

```
# Analyse: Score berechnen
```

```
result_df = calculate_osa_risk_single(df.iloc[0].to_dict())
```

```
result_df.to_csv(output_path, index=False)
```

```
print(f"Ergebnis gespeichert unter: {output_path}")
```

```
subprocess.run(f"ls -al {output_dir}", shell=True)
```

```
if __name__ == "__main__":
```

```
    input_file_path = sys.argv[1] if len(sys.argv) > 1 else None
```

```
output_dir = sys.argv[2] if len(sys.argv) > 2 else "/app/output"
main(input_file_path, output_dir)
...
```

****Wie funktioniert das Skript?****

- Es prüft sorgfältig, ob die Eingabedatei existiert, korrekt ist, nur einen Patienten enthält und keine leeren Felder hat.
- Bei jedem Fehlerfall schreibt es trotzdem eine `osa_result.csv` ins Output-Verzeichnis – mit aussagekräftigem Hinweis für die Nutzer:innen.
- Ist alles korrekt, erfolgt die übliche Analyse und das Ergebnis wird ausgegeben.

****Wenn du möchtest, kannst du die Fehlertexte im `write_error_result`-Aufruf weiter spezifizieren!****

Falls du Batch-Analyse brauchst (mehrere Patienten pro Datei), sag bitte Bescheid – dann passe ich dir die Logik entsprechend an.