m01_workshop3_guided

October 7, 2025

1 M01 Python Engineering - Workshop 3

Each challenge includes a stretch option for those who are already confident or looking for an extra challenge. These stretch tasks are completely optional, so don't worry if you don't get to them. The most important thing is to focus on the core challenge, as that's where the key learning happens. If you do feel ready to go further, give the stretch a try!

1.1 Challenge: ETL with the Sakila Database

Goal: Practice building a simple ETL (Extract, Transform, Load) pipeline using Python and SQL.

1. Extract: modify the given code from the teacher demonstration to read data from the `film` table instead of the `rental` table. Focus on the `rating` column.
2. Transform: write a SQL query that:
   - Groups films by `rating`
   - Counts how many films exist in each rating group
   - (Optional, but helpful) orders the results by count (hint: `ORDER BY total_films DESC`)
3. Load: Create a new table named `rating_summary` with the columns:
   - `rating` (TEXT)
   - `total_films` (INTEGER)
   - Then insert the results from your query into this new table.
4. Verify: Use a `SELECT * FROM rating_summary` query to check that your data was inserted correctly.
5. *Stretch: Only include ratings with more than 150 films using a `HAVING` clause in your SQL.*

Starter code from the teacher demonstration:

```
import sqlite3

# Connect to the Sakila database
conn = sqlite3.connect("sqlite-sakila.db")
cursor = conn.cursor()

# --- STEP 1: Extract ---
# TODO: Change this to extract rating data from the film table instead
cursor.execute("SELECT customer_id FROM rental")
rentals = cursor.fetchall()
```

```
print(f"Total rental records: {len(rentals)}")  # TODO: Update this print message to reflect t

# --- STEP 2: Transform ---
# TODO: Update this query to group films by rating and count how many films per rating
transform_query = """
SELECT customer_id, COUNT(*) AS total_rentals
FROM rental
GROUP BY customer_id
"""
results = cursor.execute(transform_query).fetchall()

# --- STEP 3: Load ---
# TODO: Change the table name and columns to match your new transformation (e.g., rating_summa
cursor.execute("""
CREATE TABLE IF NOT EXISTS customer_rental_summary (
    customer_id INTEGER,
    total_rentals INTEGER
)
""")

# TODO: Update the insert statement to match the new table and columns (e.g., rating, total_fi
cursor.executemany("""
INSERT INTO customer_rental_summary (customer_id, total_rentals)
VALUES (?, ?)
""", results)

conn.commit()
print("Loaded summary data into customer_rental_summary table.")  # TODO: Update this message

# --- STEP 4: Verify ---
# TODO: Update table name here to reflect your new summary table
cursor.execute("SELECT * FROM customer_rental_summary LIMIT 5")
for row in cursor.fetchall():
    print(row)
```

[3]:
```
# Start challenge...
```

[10]:
```
# Close the connection to the database once you have finished the challenge
conn.close()
```

1.2 Challenge: Create a simple Flask API

Goal: Practice building and testing a basic POST API endpoint using Flask.

- Create a Flask app
- Define a POST route
- Get JSON data from the request
- Validate that JSON contains a specific key
- Return JSON responses

- Return custom error messages with status codes
- Use requests.post() to test your API
- *Stretch:*
  - Add a GET /greet?name=YourName route
  - It should return: {"message": "Hello, YourName!"}
  - If no name is provided, return: {"message": "Hello, stranger!"}
  - Test the output in browser: e.g. http://localhost:8100/greet?name=Sam

Begin with the starter code below and fill in the ...

app.py – The Flask API

```
from flask import Flask, request, jsonify

app = Flask(__name__)

# POST /echo endpoint
# TODO: Insert the appropriate HTTP method e.g. "GET", "POST"
@app.route("/echo", methods=[...])
def echo():
    # TODO: Get JSON data from the request
    data = ...

    # TODO: Print the received data for debugging
    # (This helps us confirm what's arriving in the API)
    # Example: print(data)
    ...

    # Validate that JSON exists and contains a "message" key
    if not data or "message" not in data:
        return jsonify({"error": "Missing 'message'"}), 400

    # Return the same data back
    return jsonify(data)

# Add a simple /status endpoint
@app.route("/status", methods=["GET"])
def status():
    return jsonify({"status": "ok"}), 200

# STRETCH TODO: Add a GET /greet?name=YourName route
# It should return: {"message": "Hello, YourName!"}
# If no name is provided, return: {"message": "Hello, stranger!"}

# Run the app
@app.route('/')
def home():
    return jsonify(message="Hello from the Flask app!")
```

```
if __name__ == '__main__':
    # ---- DECODED URL GENERATOR ----
    import socket
    print("https://lab{0}{1}.labs.decoded.com:8100".format(
        *socket.gethostbyname(socket.gethostname()).split('.')[2:]
    ))
    # ---- /DECODED URL GENERATOR ----
    app.run(host="0.0.0.0", port=8100, debug=True)
```

`client.py` – Test the API

```
import requests

# Valid request
response = requests.post("http://localhost:8100/echo", json={"message": "Hello, world!"})
print("Valid response:", response.status_code, response.json())

# TODO: Try sending invalid input (e.g. {"text": "Hi"}) and print the status + response
response = ...

# Test the /status endpoint
response = requests.get("http://localhost:8100/status")
print("Status check:", response.status_code, response.json())
```

1.3 Challenge: Version-control your workflow with Git

Goal: Practice using Git to version-control a simple Python file and understand the core workflow.

1. Create a new folder called `git-practice` and add a simple Python file:

```
mkdir git-practice
cd git-practice
```

2. Create a simple Python file called `hello.py`

```
echo 'print("Hello Git!")' > hello.py
```

3. Initialise Git in the folder using `git init`. If asked to identify yourself, put a user name e.g. your first name.

```
git init
```

4. Check repo status

```
git status
```

5. Stage the file

```
git add hello.py
```

6. Commit the file

```
git commit -m "Initial commit"
```

7. Make a change

```
echo 'print("This is my first commit.")' >> hello.py
```

8. View changes

```
git status
git diff
```

9. Stage and commit again

```
git add hello.py
git commit -m "Updated hello message"
```

10. View commit history

```
git log
```

11. *Stretch:*
    - Create a new branch and switch to it: `git checkout -b new-feature`
    - Make a change and commit it
    - Switch back to `main` with: `git checkout main`
      - Your main branch may be set up as `master` in which case, try `git checkout master`

1.4 Reflection Activity: Priya's Next Steps

Priya has made big progress. She now:

- Has a solid foundation in Python.
- Can work with databases using SQL in Python.
- Can serve outputs through an API.
- Can use Git to manage her code and collaborate with her team.

She's no longer just experimenting with AI — she's starting to engineer it.

In small groups, discuss the following questions:

1. What else does Priya need to learn or practice to feel confident building a real AI system?
2. What's one piece of advice you'd give Priya about the next step in becoming an AI Engineer?
3. *Stretch: What challenges might Priya face when building an AI feature for a real product?*

Tips for Discussion:

- Think about both technical skills (e.g. testing, cloud, deployment, ML models) and team skills (e.g. collaboration, communication, agile).
- Use your own learning journey — what would *you* want to know before tackling a real AI project?
- Be ready to share one key takeaway with the main group.

END