How to Install/Run Via Pip and uv (From readme.txt)

- Step 5C: To create a new environment and run the project from scratch, use one of these methods:

- Option 1: pip (Enter the following in your terminal)

  - cd module_5
  - python -m venv .venv
  - source .venv/bin/activate # Mac/Linux; on Windows: .venv\Scripts\activate
  - pip install -r requirements.txt

- Option 2: uv

- First install uv by entering curl -LsSf https://astral.sh/uv/install.sh | sh (Mac/Linux) or pip install uv in your terminal. Then enter the following:

  - cd module_5
  - uv venv
  - source .venv/bin/activate # Mac/Linux; on Windows: .venv\Scripts\activate
  - uv pip sync requirements.txt

How to Install/Run Via Pip and uv (From readme.txt)

- Installing the Whole Project as a Package (Step 5B) –

- After installing requirements, run pip install -e . from the module_5 folder.

  - Pip: pip install -e .

  - Uv: uv pip install -e .

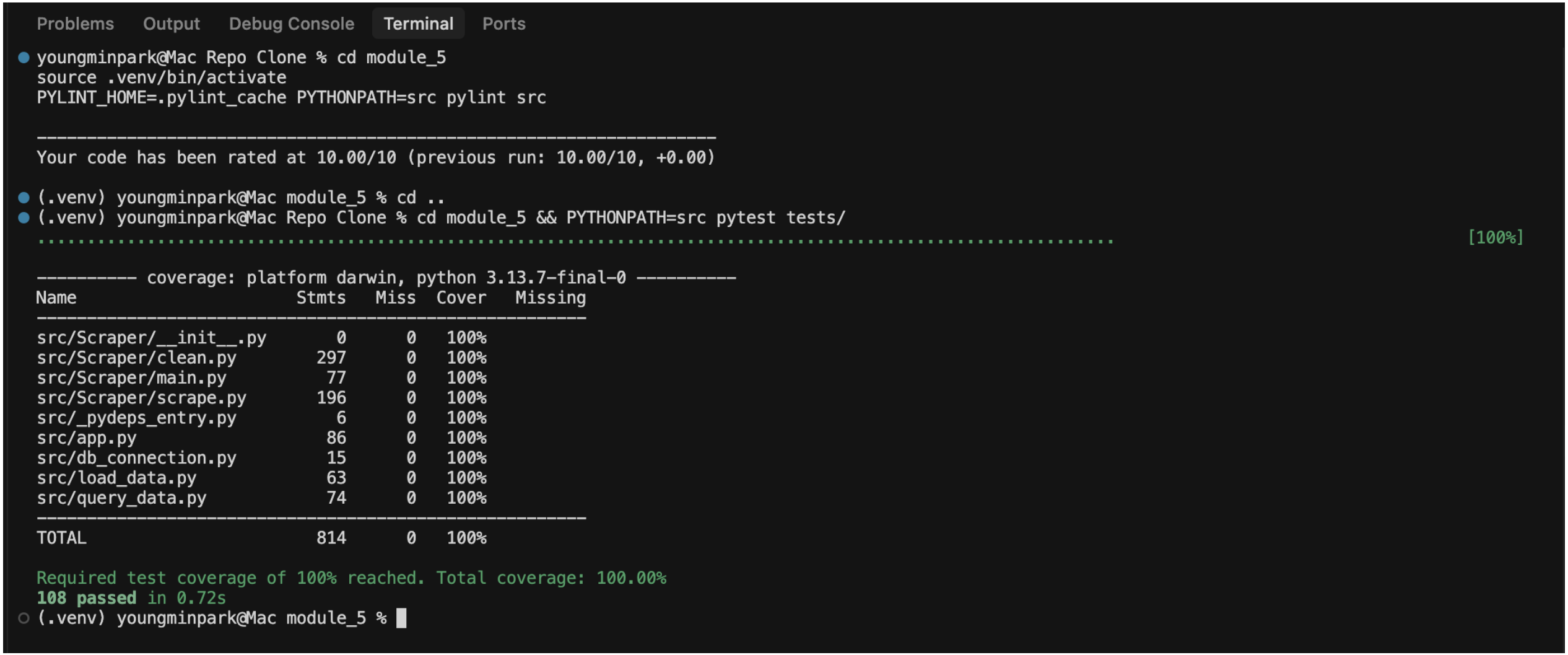
- setup.py describes my project layout and dependencies so that pip install -e . knows what to install. The package is named gradcafe-analytics in setup.py. After installing, continue with the Setup steps below (PostgreSQL, environment variables, etc.).

- Once the installations and set up are complete, initiate the Flask app by typing in the following in the module_5 .venv directory in your terminal:

  - Python -m app

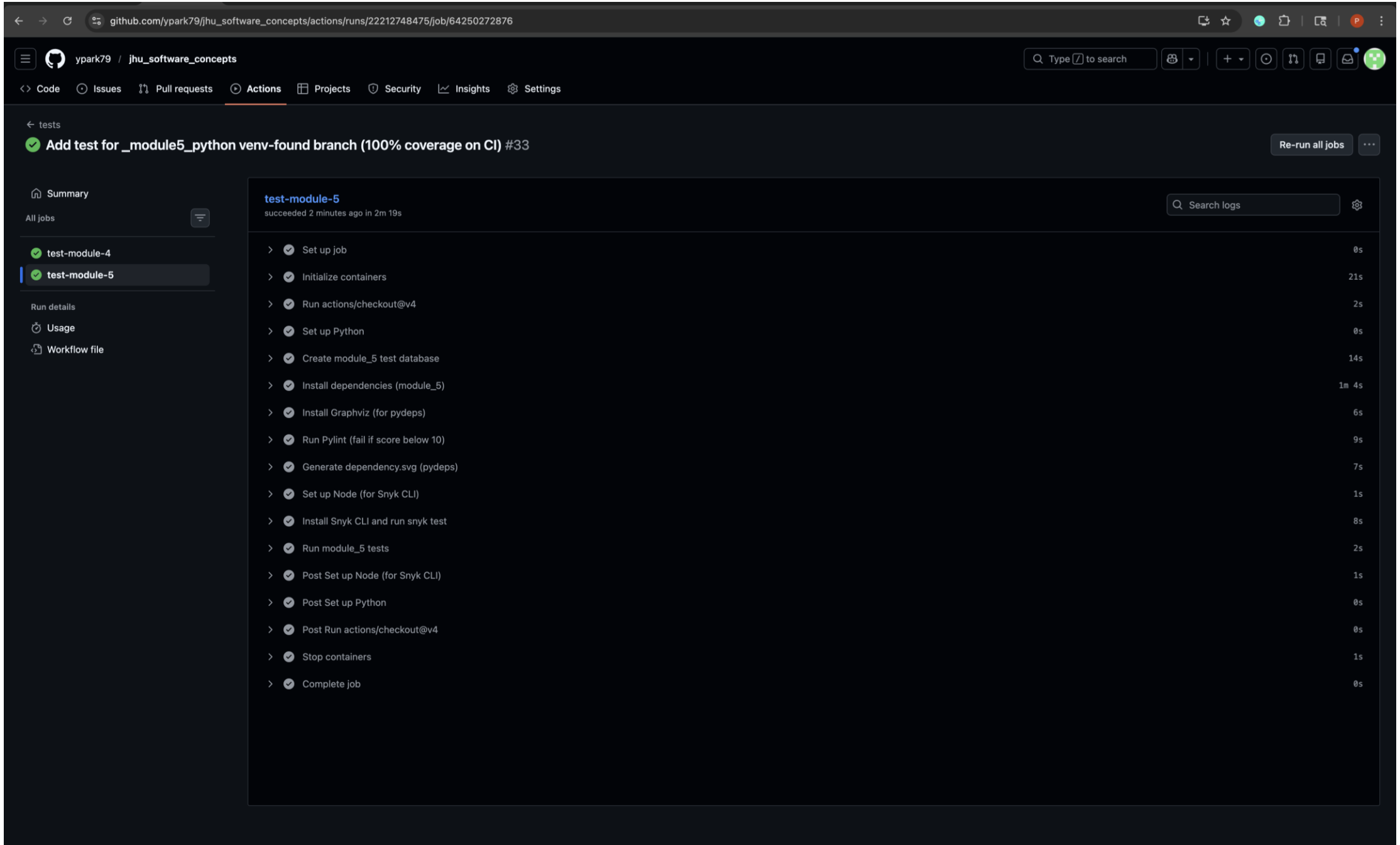
- Then open the URL in a browser, click the Pull Data button, allow the scrape/clean process to run, and then click the Update Analysis button to view the new analysis. The buttons will not be active while the scrape/clean process is ongoing – they will automatically become usable once the scrape/clean process is complete.

Step 1: 10/10 PYLINT Score and 100% Test Coverage Screenshot

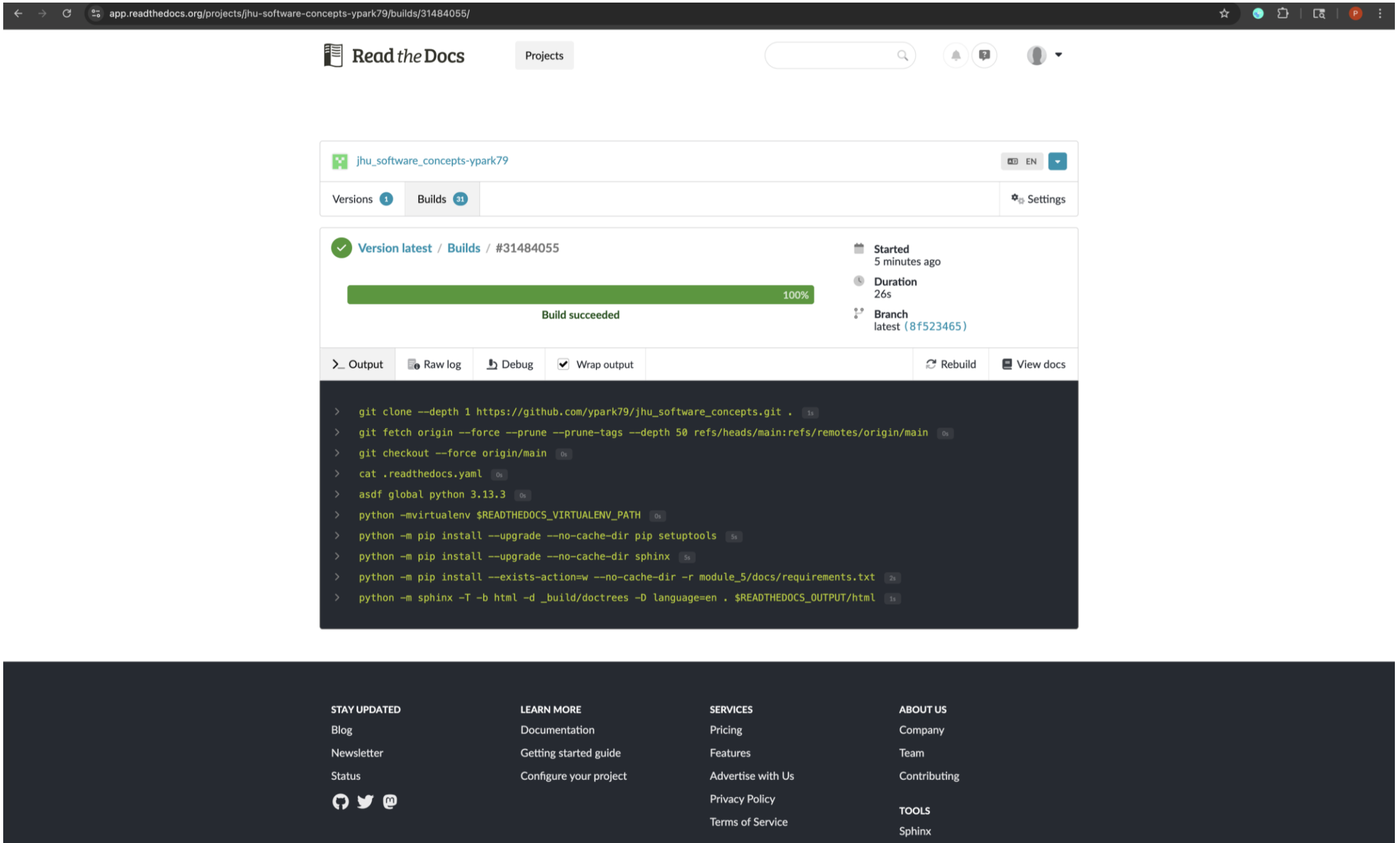


```
Problems  Output  Debug Console  Terminal  Ports

youngminpark@Mac Repo Clone % cd module_5
source .venv/bin/activate
PYLINT_HOME=.pylint_cache PYTHONPATH=src pylint src

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)

(.venv) youngminpark@Mac module_5 % cd ..
(.venv) youngminpark@Mac Repo Clone % cd module_5 && PYTHONPATH=src pytest tests/
.....................................................................................................  [100%]

---------- coverage: platform darwin, python 3.13.7-final-0 ----------
Name                       Stmts   Miss  Cover   Missing
--------------------------------------------------------
src/Scraper/__init__.py        0      0   100%
src/Scraper/clean.py         297      0   100%
src/Scraper/main.py           77      0   100%
src/Scraper/scrape.py        196      0   100%
src/_pydeps_entry.py           6      0   100%
src/app.py                    86      0   100%
src/db_connection.py          15      0   100%
src/load_data.py              63      0   100%
src/query_data.py             74      0   100%
--------------------------------------------------------
TOTAL                        814      0   100%

Required test coverage of 100% reached. Total coverage: 100.00%
108 passed in 0.72s
(.venv) youngminpark@Mac module_5 %
```

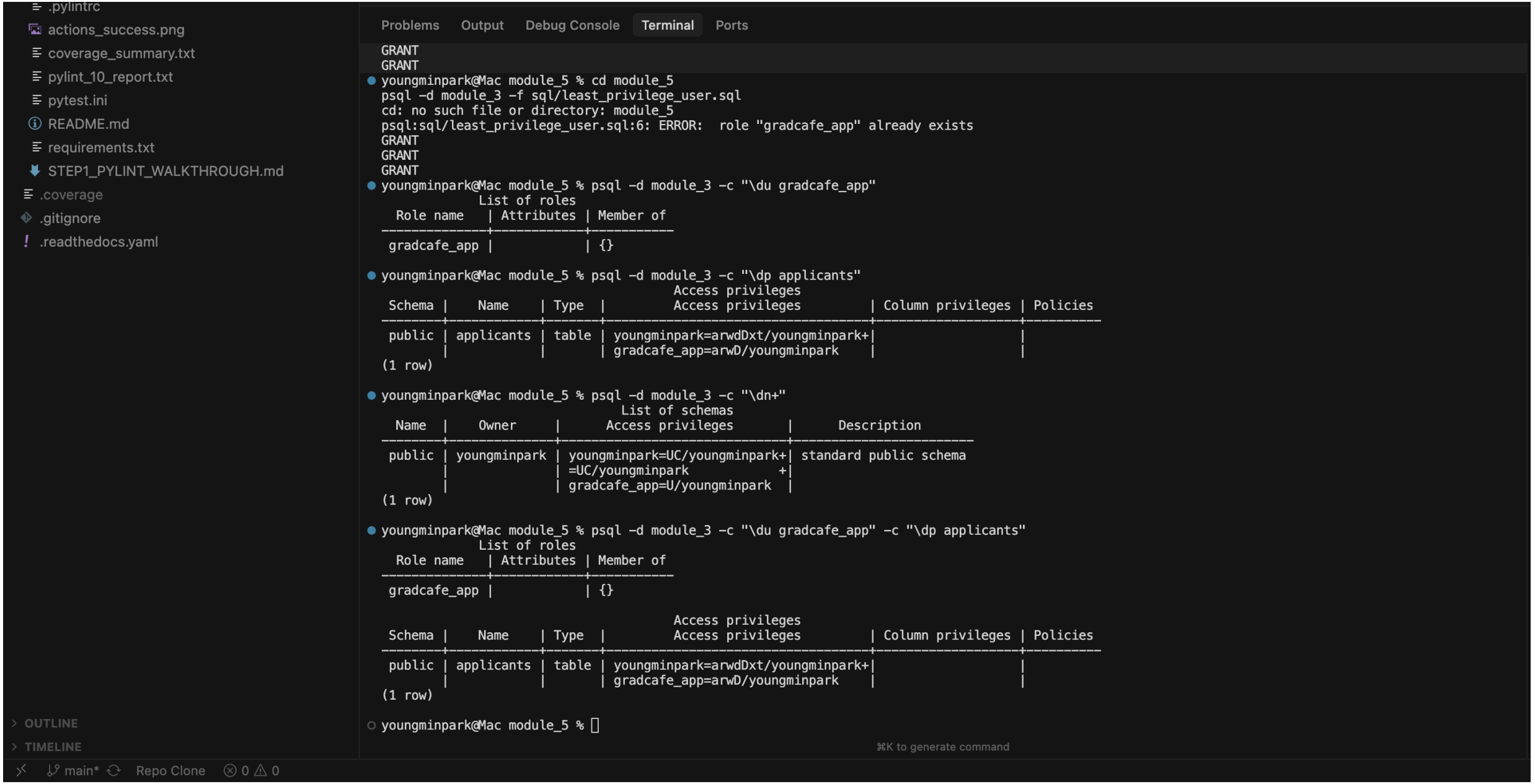
GitHub Actions Success Screenshot



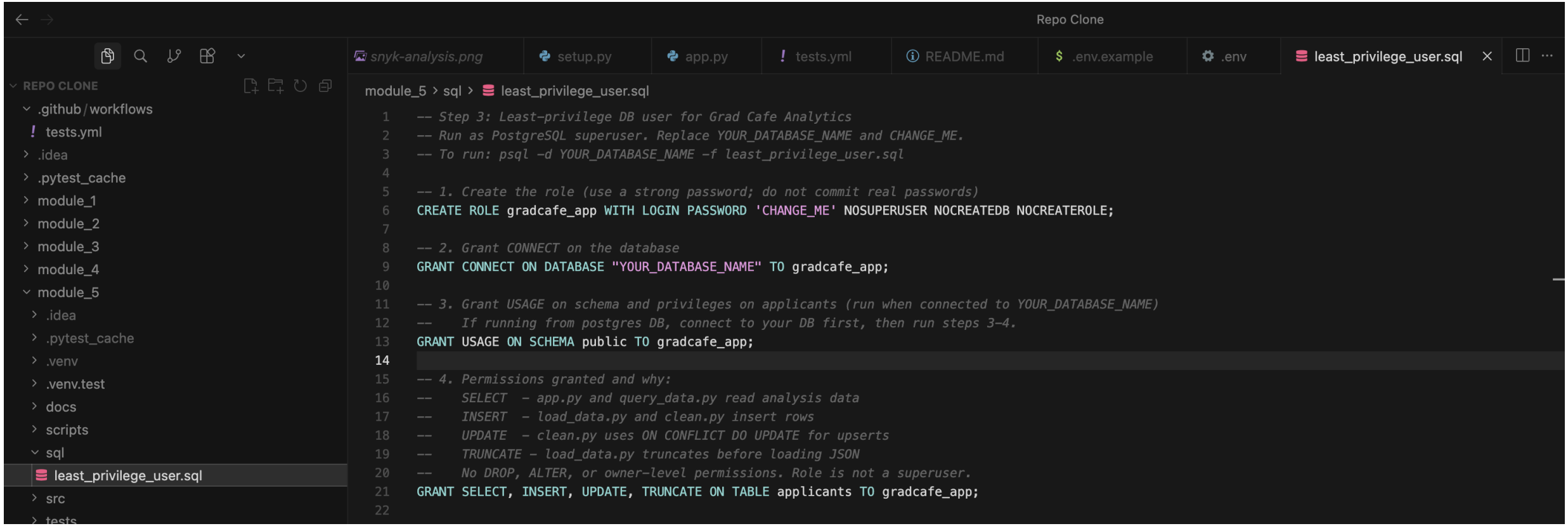
Readthedocs Updated to Module_5: Successful Build



Step 3: Screenshot of Privileges



Step 3: Screenshot of SQL Privileges



Step 3: Permissions and Justification

I created a least-privilege database user called gradcafe_app and gave it only the permissions the app actually uses. I granted CONNECT on the database so it can connect, and USAGE on the public schema so it can access the tables there. On the applicants table, I granted SELECT so the app can read data for the analysis page, INSERT so it can add rows when loading data or scraping, UPDATE so the clean pipeline can insert/update rows on conflict, and TRUNCATE so the load script can clear the table before reloading. I did not grant DROP, ALTER, or any superuser privileges, because the app never needs to change or delete tables or schemas. If the app's database is ever compromised, this setup limits an attacker to reading and modifying data in the applicants table instead of damaging the whole database.

Step 3: SQL Injection Defenses (What changed and why its safe)

I hardened every SQL query in the project so that no user or caller input is ever concatenated into SQL strings. In query_data.py, the get_sample_applicant_dict function takes a table_name argument; it used to build the query with an f-string (FROM {table_name}). I replaced that with psycopg's sql.SQL() and sql.Identifier(table_name) so the table name is always safely quoted and cannot be used for injection. In Scraper/clean.py, insert_rows_into_postgres also takes a table_name and used f-strings for the INSERT INTO and ON CONFLICT clauses; I changed those to sql.SQL() and sql.Identifier(table_name) as well. All user or caller-provided values (e.g. row data) are passed via parameter binding (%s placeholders and cursor.execute's second argument), not string formatting, so they are never interpreted as SQL. I also added LIMIT 1 to every aggregate SELECT that did not already have a limit, and introduced a maximum row limit (10,000) for bulk inserts in insert_rows_into_postgres, so a caller cannot request unlimited result sets or inserts. As a result, table and identifier inputs are safely composed with Identifier, value inputs go through parameters, and every query is bounded; the code is therefore protected against SQL injection and meets the assignment's safety and limit requirements.

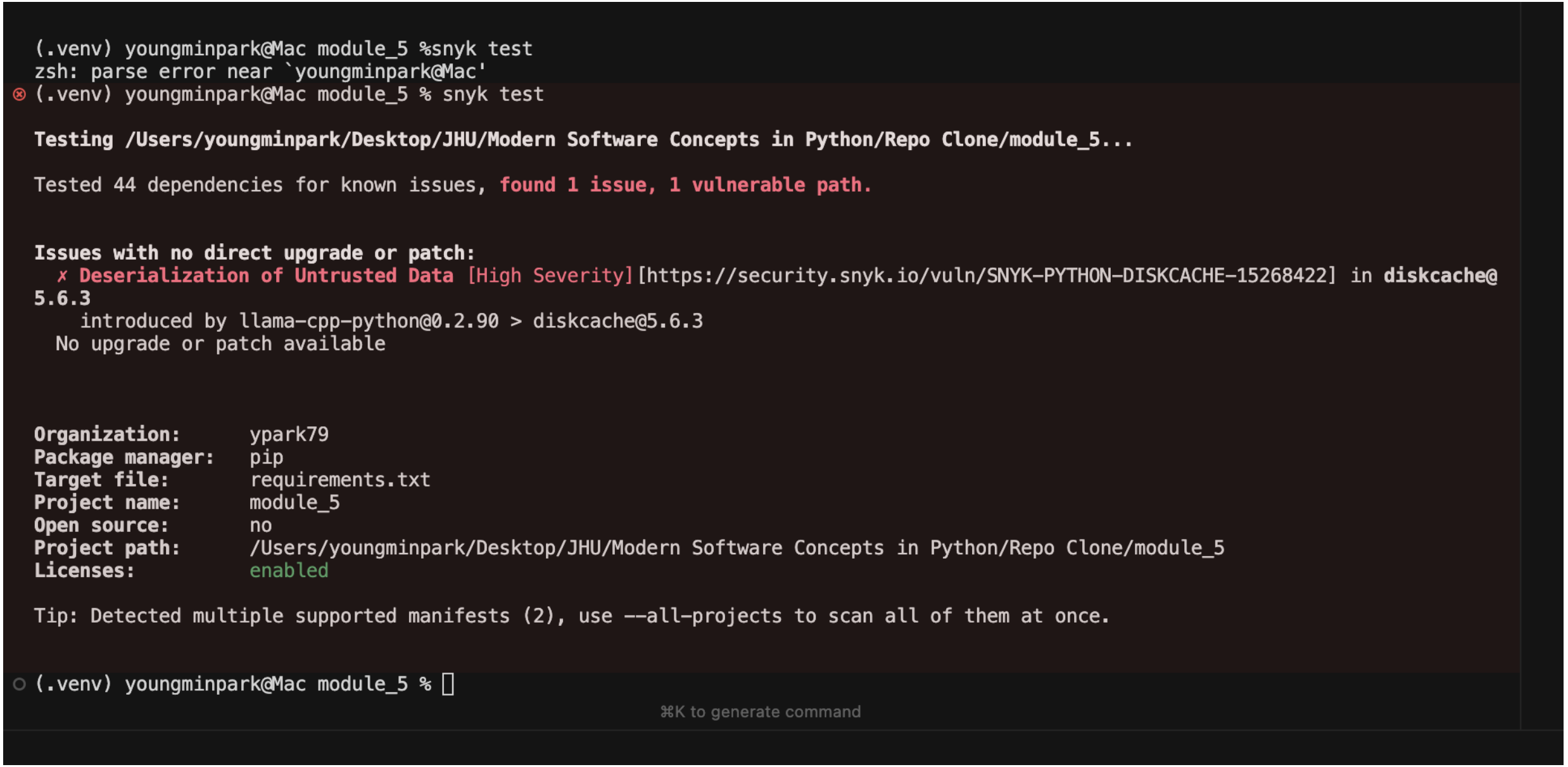
Step 4 Dependency Graph Summary (Dependencies and What They Do)

The dependency graph is built from app.py as the single entry point. It shows app.py as the main Flask application that serves the web interface and handles routes such as the analysis page and the Pull Data and Update Analysis buttons. The app depends on db_connection, a helper that creates and manages the connection to PostgreSQL using environment variables, and on Flask, which provides the web layer: routing, request handling, templating (e.g. rendering index.html). psycopg appears because db_connection uses it to connect to PostgreSQL and run SQL. The graph also includes Flask's internal modules (such as flask.app, flask.templating, flask.ctx, flask.globals, flask.helpers, and flask.wrappers) because pydeps follows every import from the entry point. Because the graph is generated only from app.py, it does not show other application modules (such as query_data or load_data); it shows app.py, db_connection, Flask, psycopg, and their imported submodules, which together form the dependency set that the graph depicts for the Grad Cafe Analytics app's main entry point.

Step 5B: Why Packaging Matters

Without packaging, Python only knows where to look for my code if I set PYTHONPATH or run from a specific folder. That works on my laptop, but a user on a different machine may use different paths, which will result in the import error "ModuleNotFoundError." Adding a setup.py makes the project installable on any machine. When the user runs pip install -e . from the module_5 folder, Python treats the project like a real package and adds it to the import path. After that, imports such as from app import create_app and from Scraper import main work the same on any machine without extra environment setup and all users use the same Python path structure. Editable installs (pip install -e .) are important because they link to the source code instead of copying it. I can edit app.py, run tests or the app, and see the changes right away without reinstalling. Packaging also helps tools like uv use setup.py when syncing environments, and it makes the project structure clear for collaborators and external users.

Step 6: snyk-analysis Screenshot



Step 6: snyk code test (First Iteration)

```
Testing /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 ...

Open Issues

 ✗ [MEDIUM] Path Traversal
   Finding ID: 7a79dacd-3472-4a7d-9a04-9445b201d0d5
   Path: src/Scraper/llm_hosting/app.py, line 365
   Info: Unsanitized input from a command line argument flows into json.dump, where it is used as a path. This may result in a Pa
th Traversal vulnerability and allow an attacker to write arbitrary files.

 ✗ [MEDIUM] Path Traversal
   Finding ID: 83a89249-fe29-422f-93a8-0d80de137e71
   Path: src/Scraper/llm_hosting/app.py, line 377
   Info: Unsanitized input from a command line argument flows into open, where it is used as a path. This may result in a Path Tr
aversal vulnerability and allow an attacker to read arbitrary files.

 ✗ [MEDIUM] Path Traversal
   Finding ID: e18fc080-3c28-4ce0-bd58-188efff91400
   Path: src/Scraper/llm_hosting/app.py, line 385
   Info: Unsanitized input from a command line argument flows into open, where it is used as a path. This may result in a Path Tr
aversal vulnerability and allow an attacker to write arbitrary files.

 ✗ [MEDIUM] Debug Mode Enabled
   Finding ID: 75bc9a20-28f9-4ada-8cf8-817eac5e85fb
   Path: src/app.py, line 369
   Info: Running the application in debug mode (debug flag is set to True in run) is a security risk if the application is access
ible by untrusted parties.


Test Summary

  Organization:      ypark79
  Test type:         Static code analysis
  Project path:      /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5

  Total issues:   4
  Ignored issues: 0 [ 0 HIGH  0 MEDIUM  0 LOW ]
  Open issues:    4 [ 0 HIGH  4 MEDIUM  0 LOW ]

⌘K to generate command
```

Snyk code test (first iteration) Issues Summary

- Issues 1– 3. Path traversal (3 issues): In the LLM helper app's file-handling code. The app has a command-line mode where you pass in file paths (e.g. code that reads JSON or writes JSON). Snyk said those paths were used without checking that they stay inside a safe folder. This introduces risk because a user could try to make the program read or write files outside the project, which is known as path traversal.
- Issue 4. Debug mode on: In the main Flask app, the main web app was started with debug mode turned on (debug=True). In debug mode, when something goes wrong, the server can show detailed error pages with bits of your code and stack traces. If the app is reachable by people you don't trust, that can leak information and is a security risk.
- Summary: Snyk Code reported 4 medium issues: three about path traversal (file paths not restricted to a safe directory) and one about debug mode being enabled in the main app. The code was corrected to address all 4 of these issues.

Sync code test (Second Iteration)

```
(.venv) youngminpark@Mac module_5 %    snyk code test
\ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Uploading batch
| Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Uploading batch
\ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
| Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
/ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
– Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
\ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
| Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
/ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
– Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
\ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
| Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
/ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
– Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
\ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
| Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
/ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
– Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
\ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
| Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
/ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
– Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
\ Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu
| Snyk Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 – Retrieving resu

Testing /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 ...


 Test Summary

   Organization:      ypark79
   Test type:         Static code analysis
   Project path:      /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5

   Total issues:   0

💡 Tip

   To view ignored issues, use the --include-ignores option.

(.venv) youngminpark@Mac module_5 %
```

⌘K to generate command