

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 -Lsf 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

github.com/ypark79/jhu_software_concepts/actions/runs/22212748475/job/64250272876

ypark79 / jhu_software_concepts

Q Type ↵ to search

+

+

+

+

+

+

+

+

<> Code

Issues

Pull requests

Actions

Projects

Security

Insights

Settings

← tests

✓ Add test for _module5_python venv-found branch (100% coverage on CI) #33

Re-run all jobs

...

Summary

All jobs

✓ test-module-4

✓ test-module-5

Run details

Usage

Workflow file

test-module-5

succeeded 2 minutes ago in 2m 19s

Search logs

> ✓ Set up job0s

> ✓ Initialize containers21s

> ✓ Run actions/checkout@v42s

> ✓ Set up Python0s

> ✓ Create module_5 test database14s

> ✓ Install dependencies (module_5)1m 4s

> ✓ Install Graphviz (for pydeps)6s

> ✓ Run Pylint (fail if score below 10)9s

> ✓ Generate dependency.svg (pydeps)7s

> ✓ Set up Node (for Snyk CLI)1s

> ✓ Install Snyk CLI and run snyk test8s

> ✓ Run module_5 tests2s

> ✓ Post Set up Node (for Snyk CLI)1s

> ✓ Post Set up Python0s

> ✓ Post Run actions/checkout@v40s

> ✓ Stop containers1s

> ✓ Complete job0s

Readthedocs Updated to Module_5: Successful Build

app.readthedocs.org/projects/jhu-software-concepts-ypark79/builds/31484055/

Read the Docs Projects

jhu_software_concepts-ypark79

Versions 1 Builds 31 Settings

Version latest / Builds / #31484055

Build succeeded 100%

Started 5 minutes ago
Duration 26s
Branch latest (8f523465)

Output Raw log Debug Wrap output Rebuild View docs

```
> git clone --depth 1 https://github.com/ypark79/jhu_software_concepts.git . 1s
> git fetch origin --force --prune --prune-tags --depth 50 refs/heads/main:refs/remotes/origin/main 0s
> git checkout --force origin/main 0s
> cat .readthedocs.yaml 0s
> asdf global python 3.13.3 0s
> python -mvirtualenv $READTHEDOCS_VIRTUALENV_PATH 0s
> python -m pip install --upgrade --no-cache-dir pip setuptools 5s
> python -m pip install --upgrade --no-cache-dir sphinx 5s
> python -m pip install --exists-action=w --no-cache-dir -r module_5/docs/requirements.txt 2s
> python -m sphinx -T -b html -d _build/doctrees -D language=en . $READTHEDOCS_OUTPUT/html 1s
```

STAY UPDATED

[Blog](#)

[Newsletter](#)

[Status](#)



LEARN MORE

[Documentation](#)

[Getting started guide](#)

[Configure your project](#)

SERVICES

[Pricing](#)

[Features](#)

[Advertise with Us](#)

[Privacy Policy](#)

[Terms of Service](#)

ABOUT US

[Company](#)

[Team](#)



[Contributing](#)

TOOLS

[Sphinx](#)

Step 3: Screenshot of Privileges

```

= .pylintrc
 actions_success.png
≡ coverage_summary.txt
≡ pylint_10_report.txt
≡ pytest.ini
① README.md
≡ requirements.txt
↓ STEP1_PYLINT_WALKTHROUGH.md
≡ .coverage
 .gitignore
! .readthedocs.yaml

```

Problems Output Debug Console **Terminal** Ports

```
GRANT
GRANT
youngminpark@Mac module_5 % cd module_5
psql -d module_3 -f sql/least_privilege_user.sql
cd: no such file or directory: module_5
psql:sql/least_privilege_user.sql:6: ERROR:  role "gradcafe_app" already exists
GRANT
GRANT
GRANT
```

```
● youngminpark@Mac module_5 % psql -d module_3 -c "\du gradcafe_app"
```

List of roles

Role name	Attributes	Member of
gradcafe_app		{}

```
● youngminpark@Mac module_5 % psql -d module_3 -c "\dp applicants"
```

Access privileges

Schema	Name	Type	Access privileges	Column privileges	Policies
public	applicants	table	youngminpark=arwdDxt/youngminpark+ gradcafe_app=arwD/youngminpark		

(1 row)

```
● youngminpark@Mac module_5 % psql -d module_3 -c "\dn+"

```

List of schemas

Name	Owner	Access privileges	Description
public	youngminpark	youngminpark=UC/youngminpark+ =UC/youngminpark + gradcafe_app=U/youngminpark	standard public schema

(1 row)

```
● youngminpark@Mac module_5 % psql -d module_3 -c "\du gradcafe_app" -c "\dp applicants"
```

List of roles

Role name	Attributes	Member of
gradcafe_app		{}

Access privileges

Schema	Name	Type	Access privileges	Column privileges	Policies
public	applicants	table	youngminpark=arwdDxt/youngminpark+ gradcafe_app=arwD/youngminpark		

(1 row)

```
○ youngminpark@Mac module_5 %
```

> OUTLINE

Step 3: Screenshot of SQL Privileges

The screenshot shows a code editor interface with a file explorer on the left and a code editor on the right. The file explorer is titled "REPO CLONE" and shows a directory structure with files like `tests.yml`, `.idea`, `.pytest_cache`, `module_1`, `module_2`, `module_3`, `module_4`, `module_5`, `.idea`, `.pytest_cache`, `.venv`, `.venv.test`, `docs`, `scripts`, `sql`, `least_privilege_user.sql`, `src`, and `tests`. The file `least_privilege_user.sql` is selected and its content is displayed in the code editor. The code editor has a tab bar at the top with tabs for `snyk-analysis.png`, `setup.py`, `app.py`, `tests.yml`, `README.md`, `.env.example`, `.env`, and `least_privilege_user.sql`. The code in the editor is as follows:

```
module_5 > sql > least_privilege_user.sql
1  -- Step 3: Least-privilege DB user for Grad Cafe Analytics
2  -- Run as PostgreSQL superuser. Replace YOUR_DATABASE_NAME and CHANGE_ME.
3  -- To run: psql -d YOUR_DATABASE_NAME -f least_privilege_user.sql
4
5  -- 1. Create the role (use a strong password; do not commit real passwords)
6  CREATE ROLE gradcafe_app WITH LOGIN PASSWORD 'CHANGE_ME' NOSUPERUSER NOCREATEDB NOCREATEROLE;
7
8  -- 2. Grant CONNECT on the database
9  GRANT CONNECT ON DATABASE "YOUR_DATABASE_NAME" TO gradcafe_app;
10
11 -- 3. Grant USAGE on schema and privileges on applicants (run when connected to YOUR_DATABASE_NAME)
12 -- If running from postgres DB, connect to your DB first, then run steps 3-4.
13 GRANT USAGE ON SCHEMA public TO gradcafe_app;
14
15 -- 4. Permissions granted and why:
16 -- SELECT - app.py and query_data.py read analysis data
17 -- INSERT - load_data.py and clean.py insert rows
18 -- UPDATE - clean.py uses ON CONFLICT DO UPDATE for upserts
19 -- TRUNCATE - load_data.py truncates before loading JSON
20 -- No DROP, ALTER, or owner-level permissions. Role is not a superuser.
21 GRANT SELECT, INSERT, UPDATE, TRUNCATE ON TABLE applicants TO gradcafe_app;
22
```

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

```
(.venv) youngminpark@Mac module_5 %snyk test
zsh: parse error near `youngminpark@Mac'
❌ (.venv) youngminpark@Mac module_5 % snyk test

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

Tested 44 dependencies for known issues, found 1 issue, 1 vulnerable path.

Issues with no direct upgrade or patch:
  x Deserialization of Untrusted Data [High Severity] [https://security.snyk.io/vuln/SNYK-PYTHON-DISKCACHE-15268422] in diskcache@5.6.3
    introduced by llama-cpp-python@0.2.90 > diskcache@5.6.3
    No upgrade or patch available

Organization:      ypark79
Package manager:   pip
Target file:       requirements.txt
Project name:      module_5
Open source:       no
Project path:      /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5
Licenses:          enabled

Tip: Detected multiple supported manifests (2), use --all-projects to scan all of them at once.

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

⌘K to generate command

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 Path 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 Traversal 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 Traversal 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 accessible 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]

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
\ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Uploading batch
| Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Uploading batch
\ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
| Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
/ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
- Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
\ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
| Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
/ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
- Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
\ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
| Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
/ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
- Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
\ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
| Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
/ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
- Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
\ Snky Code analysis for /Users/youngminpark/Desktop/JHU/Modern Software Concepts in Python/Repo Clone/module_5 - Retrieving resu
| Snky 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