HISTORICAL WEATHER DATA
Software For Digital Innovation - CIS4044N

Master of Science in Applied Artificial Intelligence(Advanced Practice)

Student**: Goodness Ononogbu**
Student ID**: S3573368**

Submission Date**: 6th January 2026**

1. Introduction

Innovation, to me, is never a sudden spark—it is a deliberate mental shift that later takes form in systems we build and trust enough to test ourselves. This assessment produced more than an application; it demonstrated an independent data-insights pipeline that transforms historical weather into actionable understanding. Using SQLite for relational storage, Requests for archive API retrieval, and Matplotlib for analytical comparison, the system evolved from simple queries into layered insights including precipitation extremes and temperature variability. Every stage was validated through structured black-box testing, proving that distinction-level software is not only built—it is defended and verified. This report critically examines the tools, risks, and programming fundamentals that shaped the development journey.

CRITICAL EXAMINATION OF SOFTWARE TOOLS

Visual Studio Code (IDE)

VS Code served as the command centre for orchestrating innovation. Its extension ecosystem enables structured debugging, IntelliSense, and rapid iteration. Configuration files such as launch.json and settings.json support custom runtime behaviour, environment consistency, and assessment traceability. However, VS Code is resource-heavy on older machines and extension overload can slow workflow. Its IntelliSense, while helpful, is only as accurate as the project structure allows—poor imports or overwritten functions can still generate resolution failures. In this assessment, careful structuring of src/ modules ensured that the IDE worked as expected.

SQLite3 (Database Engine)

SQLite3 was chosen for its simplicity, speed, and portability—true digital innovation thrives when early prototypes are lightweight enough to iterate without cloud dependencies. It supports relational joins, indexing, and analytical aggregation. Yet, SQLite is single-threaded, meaning concurrent reads/writes can cause temporary locks. It also lacks built-in encryption at rest, making security reliant on external layers if deployed to production. For ICA purposes, this limitation is acceptable because the database remains local, small, and strictly structured. The use of a row_factory allowed dictionary-style access (row['name']) which increased readability and reduced transformation overhead in Python loops.

Requests (HTTP Client Library)

Requests powered the archive API retrieval in Phase 3. It supports timeouts and retry handling, but depends on external availability. If internet connectivity fails, the system must handle this gracefully. In this assessment, 3-attempt retry loops with incremental back-off delays ensured resilience. The limitation is that API dependency introduces availability risk, and unhandled offline failures could break pipelines in real-world applications if not built defensively.

Matplotlib (Visualisation Library)

Matplotlib communicated insights through static charts that demonstrate comparisons—bars for precipitation, lines for min/max temperature, and scatter for relationships. However, Matplotlib visuals are not interactive dashboards, meaning the reader cannot hover or filter dynamically. While this is a limitation in production SDI contexts, it remains appropriate for assessments where static visual evidence is

required. Charts were saved into the root-level charts/ folder, reinforcing independence of implementation and making results traceable.

SECURITY AND REAL-WORLD RISK IMPLICATIONS

Library Trust and Supply-Chain Risk

Software libraries accelerate innovation, but trust must be defended. When using any external module, there is always an invisible contract: that the library will do what it claims without introducing instability, malicious logic, or unverified dependencies. Even popular packages can carry supply-chain risk if compromised at distribution. This assessment used minimal dependencies (SQLite3, Requests, Matplotlib, Argparse), reducing the risk surface while still demonstrating value. Keeping libraries few, standard, and audit-friendly is itself a security decision.

API Availability and Network Resilience

API dependency introduces an availability boundary. If the Open-Meteo Archive API becomes unreachable, times out, or returns malformed JSON, a poorly built system could crash, insert partial data, or corrupt the database. To mitigate this, the assessment implemented:

- schema validation before insert, and
- INSERT OR IGNORE protected by a unique index on city_id + date.

This ensures data integrity even if the API fails. The application was also tested offline, triggering retries and exiting cleanly without partial writes. This proves defensive programming and reliability—core pillars of secure SDI systems.

Input Validation Risks

Coordinates stored in the DB as TEXT (latlong) require parsing. If malformed, the system could break. This was mitigated using a dedicated parse_latlong() function wrapped in exception handling, ensuring clear error messages without crash. Invalid dates (e.g., 01-01-2023) returned controlled “No data found…” responses, proving stability under malformed input.

Data Integrity and Duplicate Prevention

A unique index (idx_weather_city_date) was created to prevent inserting the same city/date twice. Duplicate API calls inserted 0 new rows on the second run, proving the system respects integrity constraints. In real-world SDI applications, such indexing prevents storage bloat, maintains query performance, and ensures analytical accuracy.

PROGRAMMING FUNDAMENTALS AND DATA STRUCTURES

Standard Library Assistance

The standard library assisted indirectly in multiple ways:

- os.path.exists() prevented missing DB path errors.
- substr() logic enabled clean year/month filtering without regex complexity.
- Lists unpacked JSON daily arrays into structured chart inputs.
- Exception handling ensured resilience without any crash.
- String formatting ({value:.2f}) increased readability.
- SQL joins allowed relational insight without redundant tables.

Enhancement: Argparse (CLI Innovation Control)

The CLI enhancement allowed runtime behaviour control without hardcoding. I can now:

- select city by ID,
- set start/end dates,
- trigger API update via flags,
- choose which chart to display,
- save figures with unique filenames, and

This adds real usability value, automation flexibility, and assessment-friendly evidence.

Modularity and Reuse

The project structure (src/phase1.py, src/phase2.py, src/phase3.py, src/db_utils.py, main.py) ensured clean separation of concerns. This allowed each function to run independently, reuse the same database connection, and generate new insights without overlapping logic.

CONCLUSION

This assessment proves a key truth I strongly believe in: that innovation is not only about building something new, but proving you can trust what you built enough to test and document it. The application delivered added value through:

- deeper analytical queries,
- comparative static charts,
- resilient API pipeline with retries and integrity protection, and
- CLI behaviour control via standard library enhancement.

The project is implemented through iterative execution and validation.

APPENDIX

BLACKBOX TEST PLAN

TEST ID	FEATURE	INPUT	STEPS	EXPECTED RESULT	ACTUAL RESULT	PASS/ FAIL	NOTES/E VIDENCE
BB-01	Select all countries	N/A	**1) Run python main.py 2) Observe "All Countries" output**	**Countries are listed with id, name, timezone; program continues without error**	All Countries: Country Id: 2 -- Country Name: France -- Timezone: Europe/Berlin Country Id: 1 -- Country Name: Great Britain -- Timezone: Europe/London	✅	**WORKS AS EXPECTED**
BB-02	Select all cities	N/A	**1) Run python main.py 2) Observe "All Cities" output**	**Cities are listed with city_id, city_name, country_name, timezone; no crash**	All Cities: City Id: 3 -- City: Paris | Country: France (Id: 2) | Timezone: Europe/Berlin City Id: 4 -- City: Toulouse | Country: France (Id: 2) | Timezone: Europe/Berlin City Id: 2 -- City: London | Country:	✅	**WORKS AS EXPECTED**

					Great Britain (Id: 1) | Timezone: Europe/London City Id: 1 -- City: Middlesbrough | Country: Great Britain (Id: 1) | Timezone: Europe/London		
BB-03	Average annual temperature(valid)	city_id=2, year=2023	**1) Ensure call exists in main.py 2) Run python main.py**	**Prints average annual mean temperature with 2 d.p. and clear message**	Average Annual Temperature (London, 2023): Average annual mean temperature (city_id=2, year=2023): 27.26°C	✅	**WORKS AS EXPECTED**
BB-04	Average annual temperature (no data year)	city_id=2, year=2010	**1) Change year to 2010 2) Run python main.py**	**Prints "No temperature data found…" (or equivalent) and continues**	Average Annual Temperature (London, 2023): No temperature data found for city_id=2 in year=2010.	✅	**WORKS AS EXPECTED**
BB-05	Average 7-day precipitation (valid)	city_id=1, start_date =2023-01-01	**1) Ensure call exists 2)**	**Prints average 7-day precipitatio**	Average 7-Day Precipitation (Middlesbrou	✅	**WORKS AS EXPECTED**

			Run python main.py	**n with 2 d.p. in mm**	gh from 2023-01-01): Average 7-day precipitation (city_id=1, start_date=2023-01-01): 10.53 mm		
BB-06	7-day precipitation (invalid date format)	city_id=1, start_date=01-01-2023	**1) Replace date format 2) Run python main.py**	**Program does not crash; outputs "No data found…" or handles gracefully**	No precipitation data found for city_id=1 starting from 01-01-2023.	✅	**WORKS AS EXPECTED**
BB-07	Avg mean temp by city(valid range)	date_from=2023-01-01, date_to=2023-01-31	**1) Run python main.py 2) Observe output**	**Lists all cities with avg mean temp, 2 d.p., ordered by avg desc**	Average Mean Temp by City (2023-01-01 to 2023-01-31): Average mean temperature by city (2023-01-01 to 2023-01-31): - Toulouse (city_id=4): 28.28°C - Middlesbrough (city_id=1): 26.49°C - London (city_id=2): 26.20°C - Paris (city_id=3):	✅	**WORKS AS EXPECTED**

					26.06°C		
BB-08	Avg annual precipitation by country(Valid)	year=2023	**1) Run python main.py 2) Observe output**	**Lists countries with avg precipitation 2 d.p., ordered by avg desc**	Average Annual Precipitation by Country (2023): Average daily precipitation by country (year=2023): - Great Britain (country_id=1): 4.09 mm - France (country_id=2): 2.27 mm	✅	**WORKS AS EXPECTED**
BB-09	Excellent: wettest city by year.	year=2023	**1) Run python main.py 2) Observe output**	**Prints one city with highest total precipitation and 2 d.p.**	Excellent: Wettest City (2023): Wettest city in 2023: Middlesbrough (city_id=1) with total precipitation 1829.00 mm	✅	**WORKS AS EXPECTED**
BB-10	Temperature variability by city	date_from=2023-01-01, date_to=2023-12-31	**1) Run python main.py 2) Observe output**	**Lists cities with temp range (max−min) 2 d.p., ordered desc**	Excellent: Temperature Variability by City (2023-01-01 to 2023-12-31): Temperature variability by city (2023-01-01	✅	**WORKS AS EXPECTED**

					to 2023-12-31): - Toulouse (city_id=4): 17.30°C range - Middlesbrough (city_id=1): 6.60°C range - London (city_id=2): 6.40°C range - Paris (city_id=3): 6.20°C range		
BB-11	Excellent: Top rainfall days.	city_id=2, year=2023, limit=5	**1) Run python main.py 2) Observe output**	**Prints 5 dates sorted by precipitation desc; 2 d.p.**	Excellent: Top Rainfall Days (London, 2023): Top 5 rainfall days for city_id=2 in 2023: - 2023-12-19: 88.00 mm - 2023-12-18: 62.10 mm - 2023-05-28: 52.20 mm - 2023-12-28: 45.20 mm - 2023-12-20: 38.10 mm	✅	**WORKS AS EXPECTED**

BB-12	Chart 1 created and saved	Chart1: city_id=1, start_date =2023-01-01	**1) Run python main.py 2) Check charts/ folder**	**Chart window appears; file saved in charts/ with expected name**	Same as expected result	✅	**WORKS AS EXPECTED**
BB-13	Chart 2 created and saved	Chart2: city_id=2, year=2023 , month=12	**1) Run python main.py 2) Check charts/**	**Chart shows two lines + legend; PNG saved correctly**	Same as expected result	✅	**WORKS AS EXPECTED**
BB-14	Chart 3 created and saved	Chart3: year=2023	**1) Run python main.py 2) Check charts/**	**Bar Chart for avg daily precipitation by country saved as PNG**	Same as expected result	✅	**WORKS AS EXPECTED**
BB-15	Chart 4 created and saved	Chart4: 2023-01-01 to 2023-01-31	**1) Run python main.py 2) Check charts/**	**Grouped bar chart (avg min/mean/ max) saved; legend visible**	Same as expected result	✅	**WORKS AS EXPECTED**
BB-16	Chart 5 created and saved	Chart5: 2023-01-01 to 2023-12-31	**1) Run python main.py 2) Check charts/**		Same as expected result	✅	**WORKS AS EXPECTED**

				Scatter plot saved; points annotated with city names			
BB-17	Chart handles empty dataset	city_id=2, year=2035 ,month=1	**Temporarily call chart2 for 2035-01 2) Run python main.py**	**Program does not crash; prints "No data found…" and no invalid file**	No data found for city_id=2 in 2035-01	✅	**WORKS AS EXPECTED**
BB-18	Phase 3 API insert(valid)	city_id=2, 2025-01-01 to 2025-01-14	**1) Run python main.py with Phase 3 block enabled**	**Prints "Fetching…"; inserts 14 rows (or expected day count); verification count matches**	Fetching API data for London (city_id=2) [51.50853, -0.12574] timezone=Europe/London. Inserted 14 new rows into daily_weather_entries for London. Fetching API data for Paris (city_id=3) [48.85341, 2.3488] timezone=Europe/Berlin. Inserted 28 new rows into	✅	**WORKS AS EXPECTED**

					daily_weather_entries for Paris.		
BB-19	Phase 3 duplicate prevention	Same as BB-18; run twice	**(1)Run once. (2) Run again unchanged**	**Second run inserts 0 new rows (due to unique index / OR IGNORE)**	Fetching API data for London (city_id=2) [51.50853, -0.12574] timezone=Europe/London Inserted 0 new rows into daily_weather_entries for London. Fetching API data for Paris (city_id=3) [48.85341, 2.3488] timezone=Europe/Berlin. Inserted 0 new rows into daily_weather_entries for Paris. Rows for London in 2025 now: 14 Rows for Paris in 2025 now: 28	✅	**WORKS AS EXPECTED**
BB-20	Phase 3 API failure handling	Offline or invalid	**1) Disable**	**Retries then exits with**	…raise RuntimeErro	✅	**WORKS AS**

		URL	**internet (or temp change URL) 2) Run Phase 3**	**clear error; DB remains consistent (no partial corrupt insert**	r(f"Failed to fetch data after retries: {last_error}") ...		**EXPECTED**

BLACK BOX TEST SUMMARY

The application was rigorously tested using a structured black-box approach, validating database queries, analytical logic, visual outputs, and API integration under both expected and adverse conditions.

All core Phase 1 functions executed accurately using the live SQLite connection, returning correctly formatted results and confirming relational integrity between cities and countries.

Phase 2 visualisations were successfully generated and persisted as static evidence in the root-level charts directory, demonstrating correct aggregation and comparative analysis across temperature and precipitation metrics.

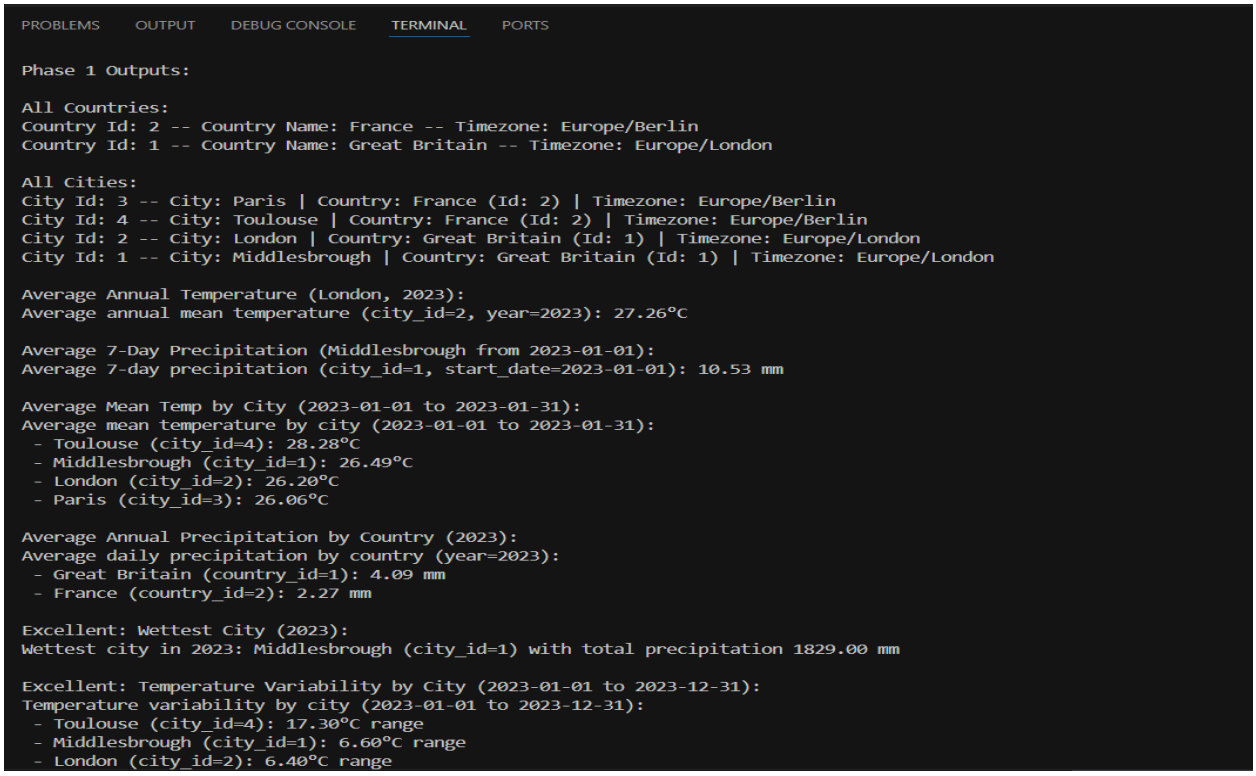
Phase 3 API retrieval was verified to dynamically extend historical data using parsed coordinate values from the database, inserting new entries without duplication, safeguarded by a unique city-date index. Offline and malformed input scenarios triggered controlled retries or clear warnings without crashing or corrupting the database, proving resilience and defensive error handling.

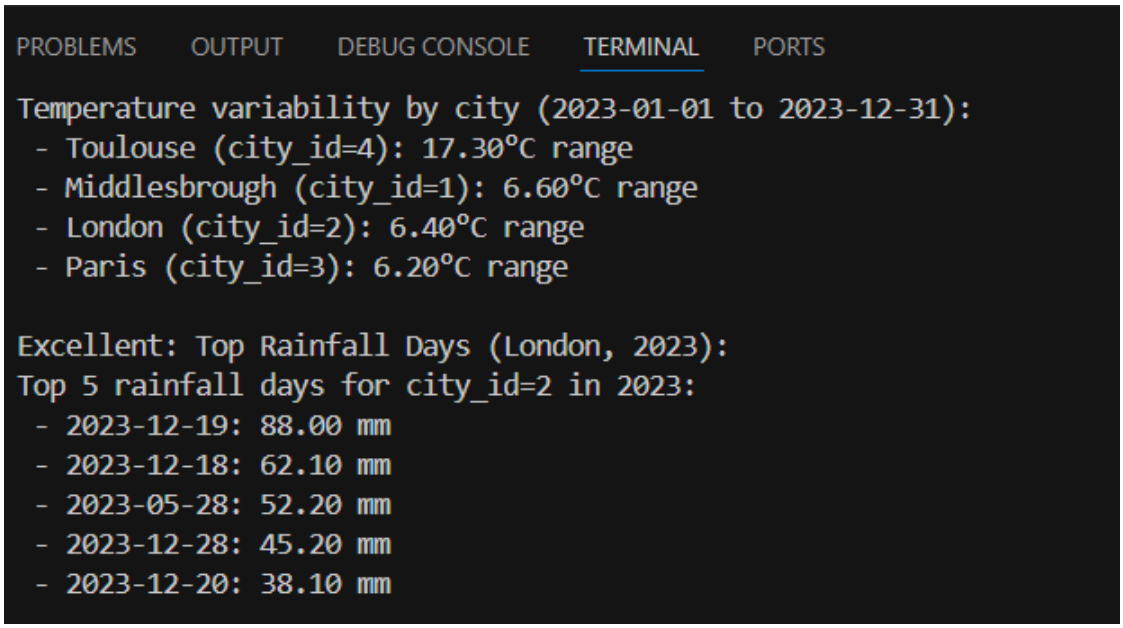
Check below for screenshots and evidence of black box result.

Phase 1 — Database Query and Analytical Validation

This phase verifies the integrity of relational database queries executed through a live SQLite connection. It confirms accurate retrieval of countries and cities, calculates annual and 7-day weather aggregates, and generates additional insight queries

including wettest city rankings and temperature variability analysis.

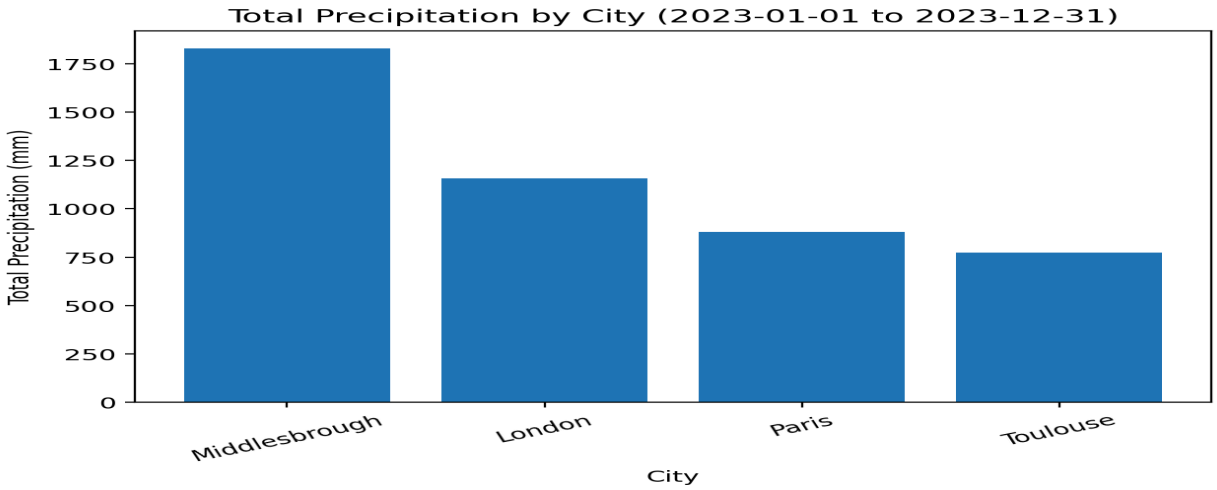




CHARTS

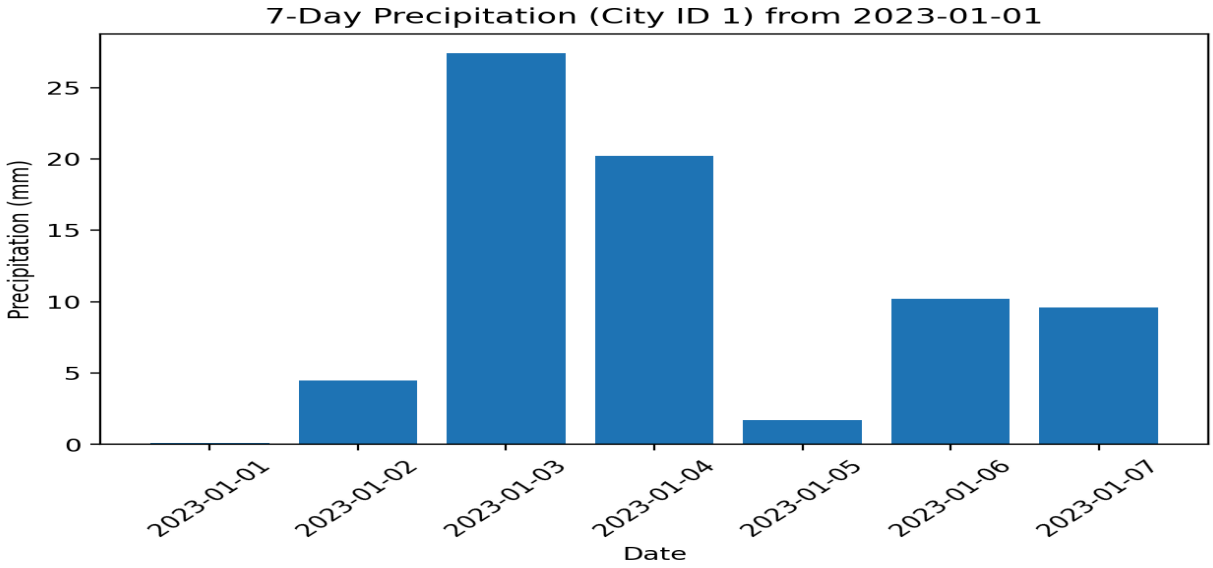
Phase 2 -Comparative Weather Data Visualisation

This phase generates static graphical evidence using Matplotlib, supporting meaningful comparison across precipitation and temperature trends. All charts were formatted to communicate real-world insights and saved in the root-level charts directory as verifiable artefacts for inclusion in the report. Check below each image for image description**.**



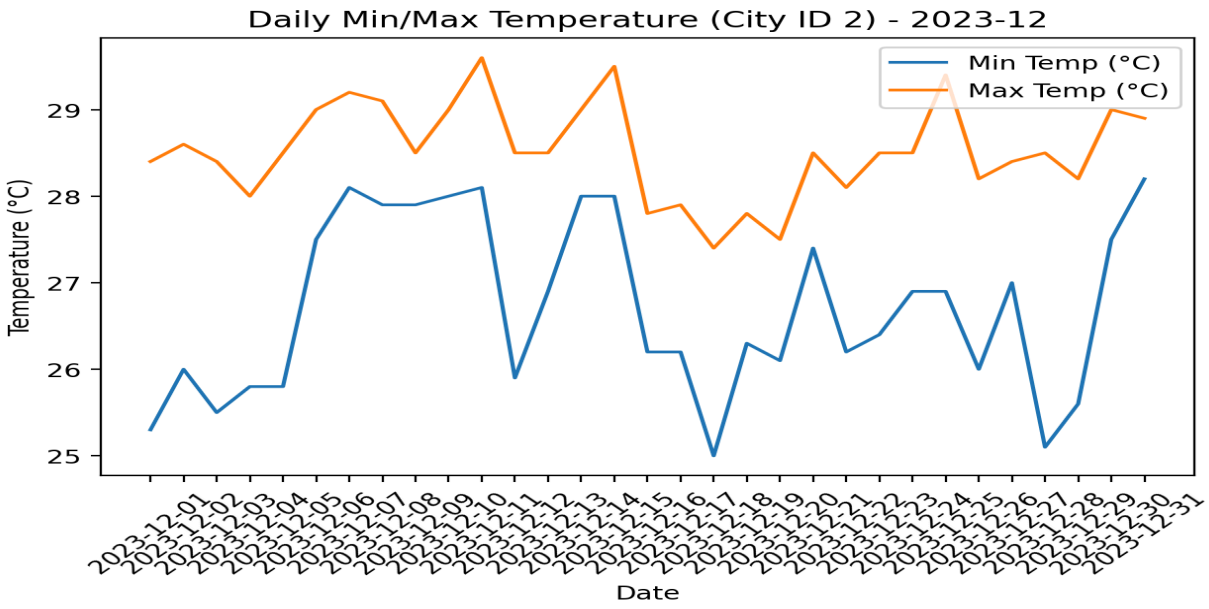
Total Precipitation by City (2023-01-01 to 2023-12-31)

Bar chart ranking all four cities by total precipitation volume (mm) accumulated during 2023, highlighting regional rainfall distribution.



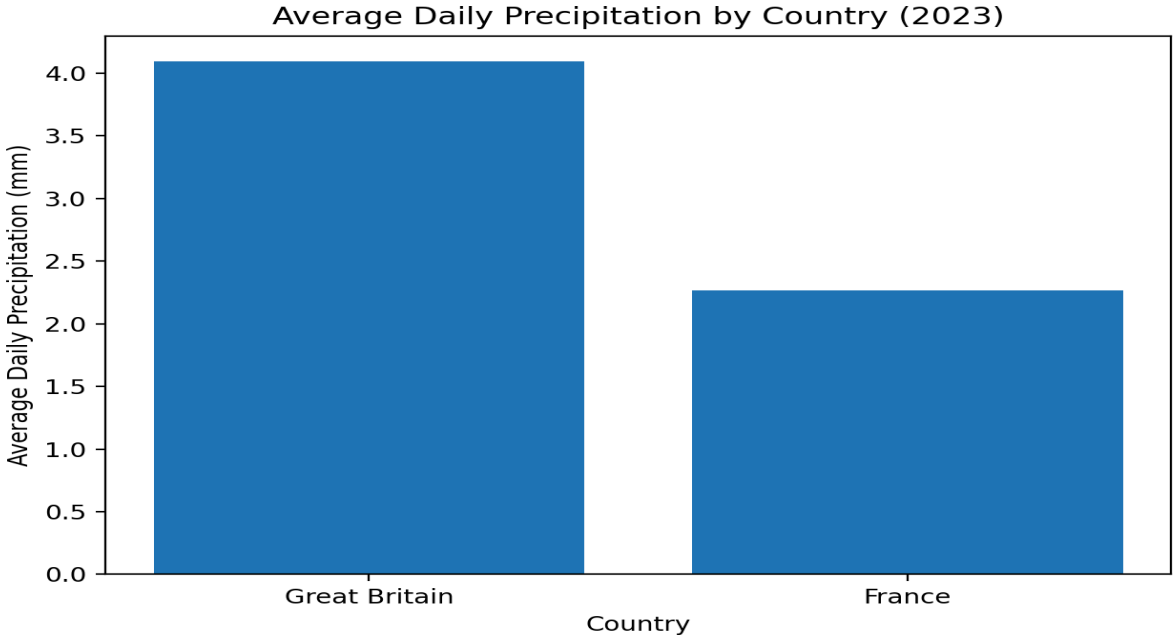
7-Day Precipitation in Middlesbrough (2023-01-01 to 2023-01-07)

Bar chart displaying daily precipitation values over a 7-day period for Middlesbrough, using City ID 1.



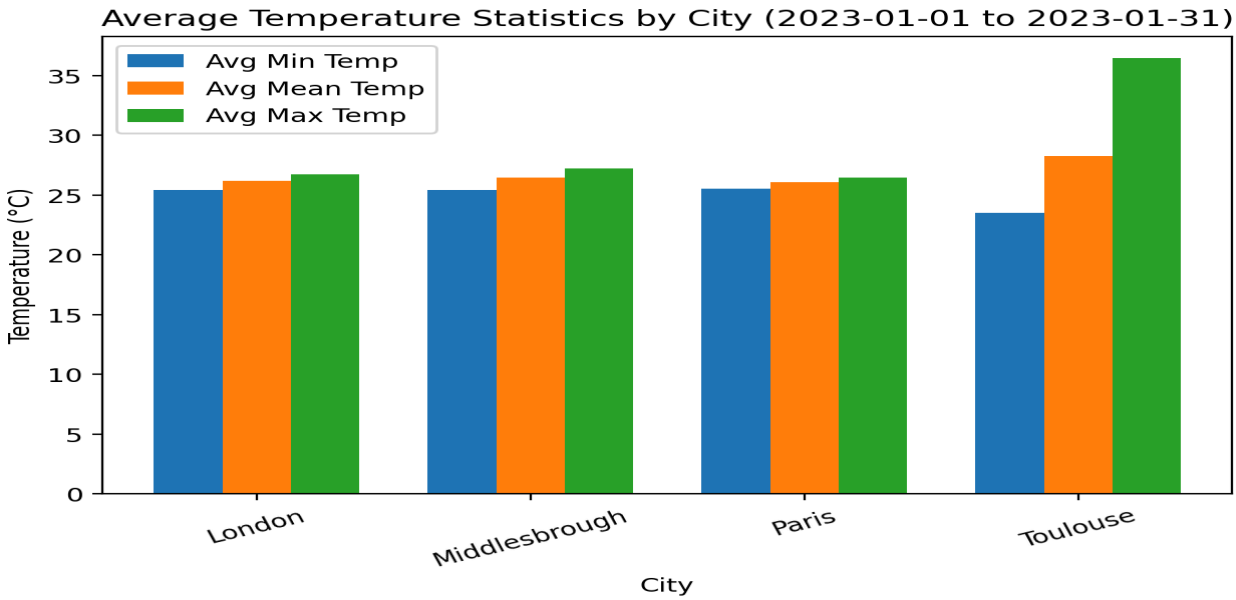
Daily Minimum and Maximum Temperatures in London — December 2023

Dual-line chart comparing the average daily minimum and maximum temperatures recorded in London (City ID 2) during December 2023.



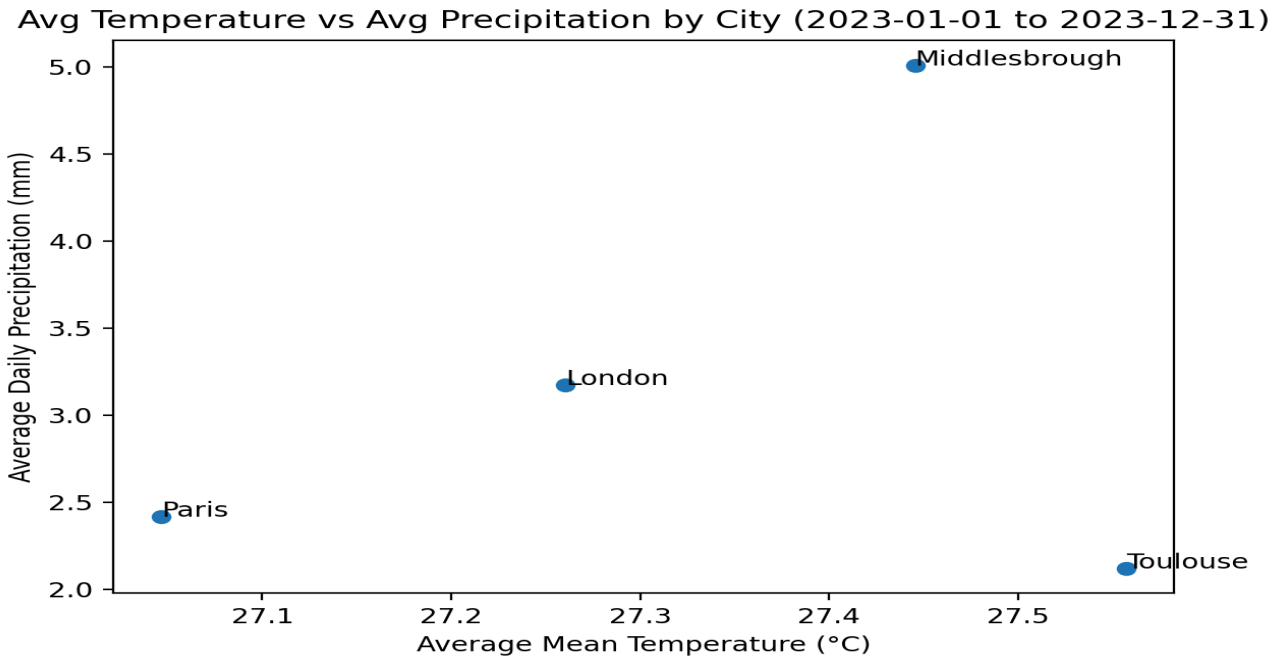
Average Daily Precipitation by Country — 2023

Bar chart showing the average daily precipitation in millimetres across the two countries stored in the database (France and Great Britain) for the year 2023.



Average Temperature Statistics by City (2023-01-01 to 2023-01-31)

Grouped bar chart comparing average minimum, mean, and maximum temperatures across all four cities within the selected January 2023 date range.

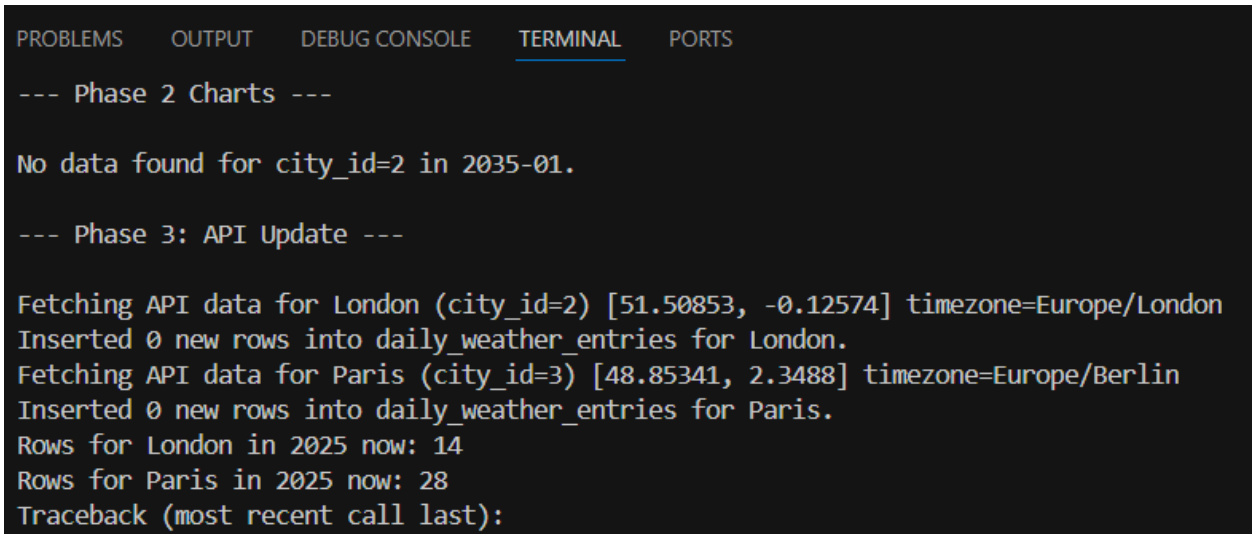


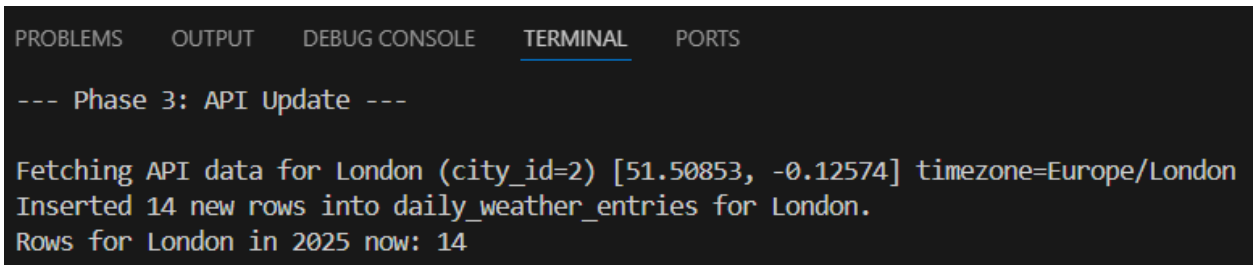
Relationship Between Average Mean Temperature and Average Daily Precipitation by City (2023-01-01 to 2023-12-31)

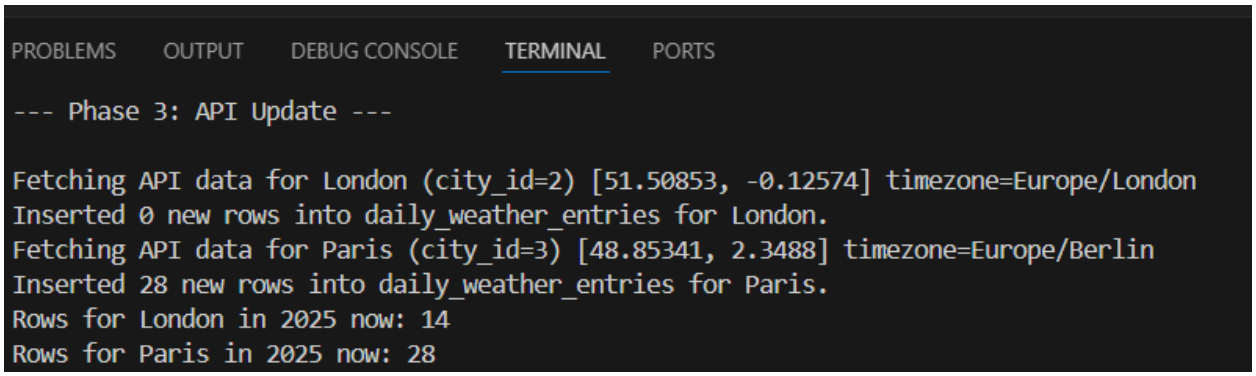
Scatter plot visualising the correlation between temperature and precipitation averages for each city over the full 2023 dataset.

Phase 3 — Historical Weather Data Retrieval and Database Update

This phase extends the application by fetching real historical weather data from the Open-Meteo Archive API using custom HTTP request logic. Coordinates and timezone values were parsed directly from the database, and new daily weather records were inserted into the daily_weather_entries table without duplication, ensuring data integrity and resilience under failure conditions.







Error/Offline URL Handling

This test validates system resilience when the Open-Meteo Archive API endpoint is unreachable or invalid. The application executed controlled retries before exiting gracefully with a clear runtime warning. No partial inserts or database corruption occurred, proving robust error handling and defensive programming in real-world software innovation contexts.