Automated Weather Data Collection and Analysis

**Samuel Asuoha**

C00305107

# 

# 

# 

# 

# **TABLE OF CONTENTS**

[**Introduction 2**](#_8h1mhywbwsvq)

[**Methodology 2**](#_f3uxyrbafwuu)

[**Data Source 2**](#_nt05jktolo40)

[**Database Schema 3**](#_vf15s0uquc2u)

[**Automation Process 3**](#_mam7f7rf2q3q)

[**Analysis and Findings 3**](#_shb98i1uotg5)

[**Lessons Learned 4**](#_hndy4h60k0w7)

[**Challenges Faced 4**](#_ijmdcme66yif)

[**Suggestions for Improvement 5**](#_omcy2dye66qn)

[**References 5**](#_k4hyjqr52mr3)

# 

# 

# 

# 

# 

# 

# **Introduction**

This report outlines the implementation of an automated weather data collection and analysis system for Dublin. The system fetches temperature and precipitation data daily from WeatherAPI, stores it in an SQLite database, and generates visualizations to observe trends over time. The objective of this assignment is to demonstrate data extraction using an API, database management, automation, and data visualization.

# 

# **Methodology**

The chosen data retrieval method is a JSON API (WeatherAPI). Web scraping was initially considered but rejected due to the potential issues with website structure changes, rate limits, and bot restrictions. APIs provide structured, reliable, and up-to-date weather data, ensuring a more stable data collection process.

Steps involved in data collection:

1. **Fetching data from WeatherAPI:** The script makes an HTTP GET request to the WeatherAPI endpoint using the requests library.
2. **Processing JSON Response:** The returned JSON data is parsed to extract temperature (in Celsius) and precipitation (in millimeters).
3. **Storing Data in SQLite:** The extracted data is stored in a structured format inside an SQLite database to maintain historical records.
4. **Automated Execution:** The script is scheduled to run daily at 08:00 AM to ensure consistent data collection.
5. **Data Analysis and Visualization:** The collected data is analyzed and displayed using tables and visual charts for better insights.

# 

# **Data Source**

* **WeatherAPI** (<https://www.weatherapi.com/>)
* Provides real-time weather information, including temperature (in Celsius) and precipitation (in millimeters).
* Requires an API key for access.

# **Database Schema**

The collected weather data is stored in an SQLite database named **weather\_data.db**. The schema consists of a single table named weather, which has the following columns:

| **Column Name** | **Data Type** | **Description** |
| --- | --- | --- |
| id | INTEGER (Primary Key, Auto Increment) | Unique identifier for each record |
| date | TEXT (Unique) | The date of data collection (YYYY-MM-DD) |
| temperature | REAL | Temperature in Celsius |
| precipitation | REAL | Precipitation in millimeters |

# **Automation Process**

* The script runs daily at 08:00 AM using the **schedule** Python library.
* It fetches weather data, stores it in the database, and updates visualizations.
* The script continues running indefinitely, ensuring continuous data collection.
* If executed for the first time, it ensures at least four days' worth of data are collected before entering the daily schedule.
* The script remains active indefinitely, checking every minute for scheduled tasks using **schedule.run\_pending()**.

# **Analysis and Findings**

* The collected data was visualized using two approaches:
  + **Line Chart**: Displays temperature and precipitation trends over time, helping to observe daily variations and extreme weather conditions.
  + **Heatmap**: Highlights temperature and precipitation variations for easier trend recognition.
* Observations:
  + Temperature fluctuates based on daily weather patterns.
  + Precipitation levels vary significantly, reflecting rain patterns in Dublin.
  + The line chart clearly depicts temperature increases and decreases, while the heatmap helps in quickly spotting trends.

# 

# **Lessons Learned**

* **APIs are more reliable than web scraping:** The use of a structured API eliminates the need to handle frequent changes in website structures.
* **Automating scripts enhances efficiency:** Scheduling daily execution ensures consistent and timely data collection.
* **Database management is essential for structured storage:** Using SQLite allows for systematic storage and retrieval of data, ensuring scalability.
* **Data visualization simplifies trend analysis:** Graphs and heatmaps make it easier to identify trends compared to raw numerical data.
* **Unicode encoding must be handled carefully in Windows:** Special characters can cause encoding errors, requiring explicit UTF-8 encoding in Python scripts.

# **Challenges Faced**

* **Initial extraction of precipitation data:** The API response structure required careful parsing to ensure correct precipitation values were obtained.
* **Time zone considerations for scheduled execution:** Ensuring that the script runs at the correct local time required verifying time zone differences.
* **Handling missing or incomplete data:** Sometimes, the API does not provide precipitation data, requiring exception handling to avoid breaking the script.
* **Challenges with web scraping:** Early attempts at web scraping were hindered by website structure changes, inconsistent HTML tags, rate limits, and ethical concerns. The transition to an API-based approach solved these issues.

# 

# **Suggestions for Improvement**

* **Develop a web-based dashboard:** A user-friendly dashboard using Flask or Django would allow real-time weather monitoring.
* **Expand data collection parameters:** Including humidity, wind speed, and atmospheric pressure would provide deeper insights into weather patterns.
* **Support multiple cities:** Extending the script to collect weather data for multiple cities would enable comparative weather analysis.
* **Improve visualization techniques:** Adding bar charts and moving averages would enhance trend analysis and make findings more intuitive.

# 

# **References**

* **WeatherAPI:**<https://www.weatherapi.com/>
* **Python Libraries Used:**
  + **requests**: Fetches data from the API.
  + **sqlite3**: Manages database operations.
  + **schedule**: Automates daily execution.
  + **matplotlib & seaborn**: Creates visualizations.
  + **pandas**: Processes tabular data.
  + **tabulate**: Formats table output in the console.

This report shows a structured approach to automated weather data collection, storage, and visualization using Python, databases, and APIs. Integrating automation with visualization improves efficiency and accessibility, enabling reliable monitoring of weather patterns over time.