ERT Assessment - Maeva Pourpoint - 05/23/2024

Assessment Coordinate Transformation

- Delivery
    - The code for this assessment can be found under the ./challenge_2 folder
- Implementation
    - To complete this assessment, I implemented a CLI program in C. This program has two modes:
        - "r2g" mode which handles the conversion from Radar to GIS coordinates
            - ./conversion r2g bearing distance lon1 lat1
            - bearing is the initial bearing in decimal degrees
            - distance is the distance between the starting and end points in meters
            - lon1 is the longitude of the starting point in decimal degrees
            - lat1 is the latitude of the starting point in decimal degrees
        - "g2r" mode which handles the conversion from GIS to Radar coordinates
            - ./conversion g2r lon1 lat1 lon2 lat2
            - lon1 is the longitude of the starting point in decimal degrees
            - lat1 is the latitude of the starting point in decimal degrees
            - lon2 is the longitude of the end point in decimal degrees
            - lat2 is the latitude of the end point in decimal degrees
    - Some validation steps were also implemented within the code base to ensure that the input parameters are valid (e.g., longitude between -180° and 180° or positive distance).
- Notes
    - The formulas used to go from Radar to GIS coordinates and vice versa are outlined and described in the Movable Type Scripts website [1].
    - These formulas are valid if we assume a spherical earth (i.e., ignoring ellipsoidal effects) which leads to error up to 0.3% [2].
    - I tested my application in both modes by comparing its outputs to the ones in the Movable Type Scripts website.
- References

    - [1]http://www.movable-type.co.uk/scripts/latlong.html
    - [2]https://gis.stackexchange.com/questions/25494/how-accurate-is-approximating-earth-as-sphere#25580

Assessment Data Collection

- Delivery
    - The code for this assessment can be found under the ./challenge_3 folder

- Task 1 - Implementation
    - Development: To address task 1, I implemented a Python application (swpc_monitoring) using the following libraries:
        - requests: Most commonly used Python library for working with APIs (i.e., retrieving and sending data to a server)
        - influxdb-client: Client library that integrates with InfluxDB 2.x and Flux.
        - python-dotenv: Library to handle environment variables stored in a .env file
        - scheduler: Library to schedule recurring tasks
    - Application: The purpose of the swpc_monitoring app is to retrieve data from the NOAA SWPC API and store them in a time series database that allows for real-time analytics, event monitoring and data visualization.
    - Requirements:
        - Persistence: The data were stored in a database (InfluxDB) to make them persistent.
        - Extensibility: The specifics of RTSW file format weren't hard coded.
        - Cadence, real-time requirements, network constraints: Data are retrieved from the NOAA SWPC API and written to the InfluxDB database every minute since the RTSW data appears to be updated every minute. Although InfluxDB can handle duplicate data points, only new data points are written to the database (one-point overlap is allowed for redundancy). A proper way to handle data backfill when the connection to the NOAA SWPC API drops for more than one hour remains to be implemented.
    - Usage:
        - To run the swpc_monitoring app on your local machine:
            - Install InfluxDB [1]
            - To start influxdb, open terminal window 1 and run:
                - influxdb
            - To start the swpc_monitoring app, open terminal window 2 and enter:
                - cd .../challenge_3/
                - conda create -n swpc_monitoring python=3.11
                - conda activate swpc_monitoring
                - pip install -e .
                - swpc_monitoring

- Task 1 - Notes
    - InfluxDB - I decided to use a purpose-built time series database for its:
        - Performance: InfluxDB data stores are optimized for time series data allowing for high availability data storage and retrieval (especially, for any data volume higher than a million data points). Its optimized time series data store also allows for low disk space requirements. [2][3]

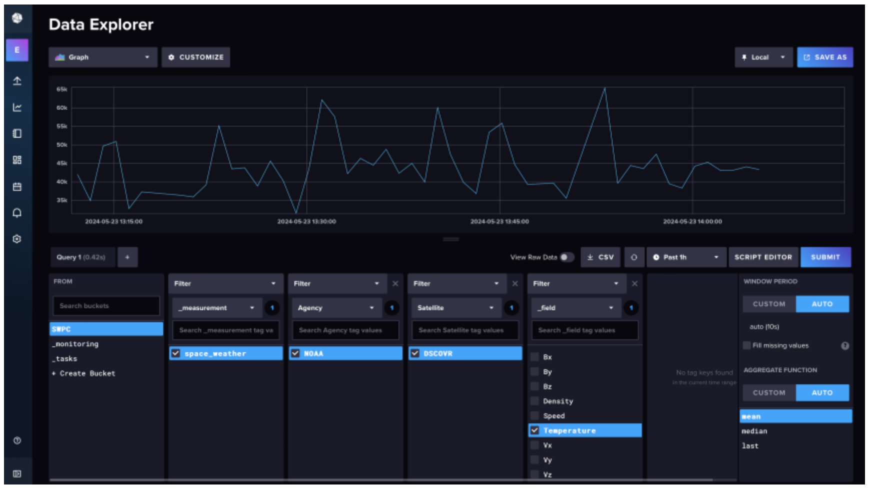
        - Scalability: There is no limit to the number of tags and fields that can be used. Timestamps in InfluxDB can be second, millisecond, microsecond, or nanosecond precision making it useful for scientific data with high-time accuracy. [2][3]
        - Integrated functionalities allowing to process, visualize and monitor data in one place.
        - Open-source and low barrier adoption
    - Software development best practices:
        - Implemented in this release:
            - Logging to report various events happening during the application run (implemented with built-in logging module)
            - Linting to follow the PEP8 style guide (performed with Flake8)
            - Type hinting to statically indicate the value types within the code base (checked with Mypy)
            - Application configuration to facilitate packaging and distribution (handled with setuptools and associated files)
        - Still needed:
            - Application testing via unit and integration tests in CI/CD pipelines

- Task 2 - Implementation
    - To address the task 2, I used docker and created the following files:
        - A docker file to produce an image for my python application
        - A docker-compose.yaml file to manage services (python and influxdb), networks (for cross-communication between containers) and volumes (database storage)
        - A .env file to pass environment variables into the docker containers and to allow docker to start InfluxDB with automated setup [4]
    - Usage:
        - Install Docker
        - Open terminal window and enter:
            - cd .../challenge_3/
            - docker-compose up --detach
        - Open InfluxDB in a web browser using the following URL: http://localhost:8086
        - Use the Data Explorer tab to view the time series
        - ...

- Task 2 - Notes
    - Software development best practices:
        - Still needed: Use Docker secrets to manage sensitive data like passwords and token

- Task 3 - Implementation
    - InfluxDB comes with a UI dashboard (Data Explorer) that allows to view the time series and easily create aggregate metrics such as mean, min, max, ... [5]. It also allows to easily monitor data and create/send alerts for user-defined events [6].

      Example of time series visualization in InfluxDB (Temperature data from RTSW data set)



- Another open-source tool I would use to monitor real-time data would be Grafana. It has a plugin to connect to InfluxDB and could be relatively easily configured to query data from our InfluxDB data source and build dashboards using existing templates. It also allows to add more than one panel to the dashboard.
- Another option would be using Dash [7] to create a web dashboard for time-series data visualization. This option would require a bit more effort since I would have to design/create the layout of the webpage.

- Task 3 - Notes
  - Current approach to early warning of solar storms
    - Reviewing the real-time solar wind data recorded over the last year [8], I noticed a significant increase in the wind speed and magnetic field values prior and during a solar storm event.
    - I would use this information to define threshold values for wind speed and magnetic field data. Above that given threshold, I would trigger an alert.

- Task 4 - Notes
  - I didn't have time to work on implementing a solution for this task. But given more time, I would have look more into AutoTS (an automated time series forecasting package for Python) [9]

- References

  - [1]https://docs.influxdata.com/influxdb/v2/install/
  - [2]https://www.influxdata.com/the-best-way-to-store-collect-analyze-time-series-data/
  - [3]https://www.influxdata.com/blog/why-use-purpose-built-time-series-database/
  - [4]https://hub.docker.com/_/influxdb
  - [5]https://docs.influxdata.com/influxdb/cloud/visualize-data/
  - [6]https://docs.influxdata.com/influxdb/cloud/monitor-alert/
  - [7]https://dash.plotly.com
  - [8]https://www.swpc.noaa.gov/products/real-time-solar-wind
  - [9]https://winedarksea.github.io/AutoTS/build/html/index.html

Assessment Interpolation

- Delivery
  - The code for this assessment can be found under the ./challenge_4 folder
- Task - Implementation
  - To complete this assessment, I used Python and the following libraries/methods:
    - scipy.interpolate.griddata to interpolate the given point locations. The griddata method is particularly appropriate for unstructured and scattered data like the provided dataset [1]
    - matplotlib to visualize the results of the interpolation
    - numpy for array management
  - I used three different methods to interpolate the data
    - Nearest-neighbor interpolator in N > 1 dimensions [2]
    - Piecewise linear interpolator in N > 1 dimensions [3]
    - Piecewise cubic, C1 smooth, curvature-minimizing interpolator in 2D [4]
  - I used Jupyter lab to allow for interactive display of the interpolation results.
  - Usage:
    - Open terminal window and enter:
      - cd .../challenge_4
      - python3 -m venv .venv
      - source .venv/bin/activate
      - pip install -r requirements.txt
      - jupyter lab
    - In web browser opened by Jupyter lab:
      - Double-click on the Interpolation.ipynb notebook to start it
      - Run through each notebook cell
- Task - Notes
  - The data points are best fit using the linear or the cubic interpolator methods.
  - In order to choose the "best" (i.e., most physically reasonable) interpolation model, I would need to have some additional information about the data set that we are trying to model.
- References

  - [1]https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html
  - [2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.NearestNDInterpolator.html
  - [3]https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.LinearNDInterpolator.html
  - [4]https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CloughTocher2DInterpolator.html

Assessment Convolution

- Delivery
  - The code for this assessment can be found under the ./challenge_5 folder

- Task - Implementation
  - To complete this assessment, I used Python and the following libraries/methods:
    - scipy.signal.convolve2d to convolve two two-dimensional arrays [1]
    - matplotlib to visualize the input data and the results of the convolution
    - numpy for array management
  - I used Jupyter lab to allow for interactive display of the interpolation results.
  - Usage:
    - Open terminal window and enter:
      - cd .../challenge_5
      - python3 -m venv .venv
      - source .venv/bin/activate
      - pip install -r requirements.txt
      - jupyter lab
    - In web browser opened by Jupyter lab:
      - Double-click on the Convolution.ipynb notebook to start it
      - Run through each notebook cell

- Task - Notes
  - For relatively small arrays, the convolve2d method is an appropriate convolution method. It also handles image boundaries (although not necessarily relevant for this dataset). For larger arrays (n > ~500), the fftconvolve method would be more appropriate [2].
  - The convolution result of the raw_0 and raw_1 data sets seems reasonable based on the expected feature combinations, and the fact that the maximum value of the convolution result is in the same order of magnitude as the value from multiplying the maximum values of raw_0 and raw_1 (i.e., max(raw_0)*max(raw_1)/max(conv) = 1.42)

- References

  - [1] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve2d.html
  - [2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.fftconvolve.html#scipy.signal.fftconvolve