Development Environment

The weather monitor was tested and developed with the following environment:

- Raspberry Pi 4 Model B
- Raspberry Pi OS (64-bit, with desktop)
    - Release date: September 22nd 2022
    - Kernel version: 5.15
    - Debian version: 11 (bullseye)
- 32 GB Micro SD-Karte
- Waveshare Display 10.1

Project Structure

The project is structured as follows:

```
├── autostart # Autostart script that is executed on boot
├── DEVELOPMENT.md # Development documentation
├── docker-compose-dev.yml # Docker compose file for development
├── docker-compose.yml # Docker compose file for production
├── Dockerfile # Dockerfile for production
├── download_csvs.sh # Script to download CSV files for historical weather data from the we
├── download_model.sh # Script to download the prediction model from the web
├── example.env # Example environment file. InfluxDB password is stored here
├── images # Images used in README.md
│   ├── dashboard.png
│   ├── Intro.gif
│   ├── Intro.mp4
│   └── loading_screen.gif
├── install.sh # Convenience script to install the weather monitor
├── prediction # Prediction related files
│   ├── input # Input data for the prediction model
│   │   ├── input.csv
│   │   ├── messwerte_mythenquai_2007-2021.csv
│   │   └── messwerte_tiefenbrunnen_2007-2021.csv
│   └── weather_prediction.ipynb # Jupyter notebook to train the prediction model
├── README.md # Readme file. Installation instructions are here.
├── requirements.txt # Python requirements for production and development
└── src # Source code
    ├── app.py # Main application. Flask endpoints are defined here. Contains startup hook
    ├── plotting.py # Plotting functions using Plotly. Saves plots as SVGs to the static fo
    ├── prediction.py # Prediction functions. Uses the prediction model to predict the weat
    ├── service_status.py # Service status functions. Checks if the weather monitor is runn
    ├── static # Static files. Contains the SVGs generated by plotting.py during runtime.
    │   └── images # Images used in the dashboard
    │       └── direction.png # Image used in the wind direction prediction visualization
    ├── templates # HTML templates. Contains the HTML templates for the dashboard.
    │   ├── current.html # Current weather subpage
    │   ├── index.html # Navigation and header
    │   ├── loading.html # Loading screen
    │   ├── plot.html # Detailed plot subpage
    │   ├── plots.html # Plots slider subpage
    │   ├── prediction.html # Prediction subpage
    │   └── station.html # Station subpage
    ├── tests.py # Unit tests
    ├── weather_data.py # Weather data functions. Fetches the weather data from the web and
    └── weather_repository.py # Weather repository functions. Includes query logic that is
```

weather_data.py

The `weather_data.py` file contains functions to communicate with the API and the database. The file fetches data from the API and stores it in the database.

This file was provided by the project owner and was slightly modified in order to match the use case. These modifications include the following:

- execute_query function
    - The `execute_query` function is used to execute a query on the database. It takes the query as a parameter and returns the result of the query.
    - The query tries to catch any exception that might occur and returns `None` if an exception occurs.
    - In this way, we can utilize the existing database connection and do not have to create a new connection for every query.

Handling of timezones

- The data structure returned by the database client, includes the timestamp in UTC format.
- The timestamps are converted to the local timezone (default: Europe/Zurich) by default.
- The conversion is done within the `run_query()` function in `weather_repository.py` .
- The timezone is changed by default because the data is mostly used to visualize in the dashboard. The dashboard displays the data in the local timezone of the main customer (Zurich, Switzerland).

plotting.py

The `plotting.py` file contains functions to generate plots using Plotly. The plots are saved as SVGs to the static folder. The SVGs are then embedded in the HTML templates.

prediction.py

The `prediction.py` file contains functions to predict the weather. The prediction model is loaded from the `src` folder directly. The model is trained in the `weather_prediction.ipynb` notebook. The notebook is located in the `prediction` folder.

install.sh

The installation script is used to install the weather monitor. It is a convenience script that installs the weather monitor and all its dependencies.

The installation script will install the following services and applications:

- Docker
- Docker Compose
- InfluxDB (Time series database as Docker container)
- Weather Monitor (The application as Docker container)
- Watchtower (Service to automatically update Docker containers as Docker container)
- autostart script (Script that is executed on boot. Opens the dashboard in the browser)
    - The script opens Chromium in full screen mode on boot. This can be disabled by commenting out the last line of the script.

> Through the docker restart policy, the weather monitor will automatically restart if it crashes or the Raspberry Pi reboots.

Tech stack

The weather monitor is built with the following technologies:

- Docker used to containerize the application
- Docker Compose used to orchestrate the application
- Python as programming language
  - Used version: 3.10
- Flask as web framework
- Plotly as plotting library
- sci-kit learn used to make predictions with the help of K-nearest neighbors algorithm
- schedule used to schedule the generation of predictions, plots and health checks
- NumPy used to work with arrays
- Pandas used to work with dataframes
- requests used to make HTTP requests to the API as well as to download CSV files
- InfluxDB used as database. Official python client is used to communicate with the database
- Bash scripts used to automate the download of files on build time and the installation of the weather monitor
- Intro.js used to provide a guided tour of the weather monitor
- Bulma used as CSS framework
- Pickle used to serialize and deserialize the prediction model

All python dependencies are also listed in the `requirements.txt` file with their respective version.

Environment

The weather monitor is built with Docker. The `Dockerfile` is used to build the image and the `docker-compose.yml` file is used to run the image. The `docker-compose-dev.yml` file is used to run the image in development mode.

Run in development mode (Docker)

> **Note**: The scripts used are based on a Linux environment. They may not work on other operating systems. On Windows the scripts can be run using WSL.

First the current model has to be downloaded using the script `download_model.sh` :

```
./download_model.sh
```

After that the model file has to be moved to the `src` folder:

```
mv weather_model.pkl ./src/weather_model.pkl
```

The weather monitor can be run in development mode using the `docker-compose-dev.yml` file. This file is configured to use the `Dockerfile` file in the root of the project. This means that the Docker image will be built locally instead of pulling the image from the GitHub Container Registry.

The following command can be used to run the weather monitor in development mode:

```
docker compose -f docker-compose-dev.yml up -d --build --force-recreate --remove-orphans
```

Run in development mode (Python)

Follow the steps from above apart from the last one. Instead of running the weather monitor in Docker, it can be run in Python. The following command can be used to run the weather monitor in development mode:

A minimum version of Python 3.10 is required to run the weather monitor.

> **Note**: The `requirements.txt` file has to be installed first. This can be done using the following command:
>
> ```
> pip install -r requirements.txt
> ```

```
docker compose -f docker-compose-dev.yml up -d influxdb
python3 src/app.py
```

This way we will only run the InfluxDB database in Docker and the rest of the application will be run in Python. This is useful if you want to make changes to the application and see the changes without rebuilding the Docker image every time.

Logs

Logs can be viewed through the Docker container logs. The following command can be used to view the logs of the weather monitor container:

```
sudo docker logs -f fhnw-ds-cde1-wettermonitor-weather-monitor-1
```

You may have to replace `fhnw-ds-cde1-wettermonitor-weather-monitor-1` with the name of the container. The name of the container can be found using the following command:

```
sudo docker ps
```

The matching container name will be listed in the `NAMES` column of the container running under the `ghcr.io/fhnw-ivy/fhnw-ds-cde1-wettermonitor:main` image.

If you're running the weather monitor in development mode and with Python, the logs can be viewed in the execution terminal.

Code Documentation

The code is documented using docstrings directly in the code.

Unit Testing

The unit tests are made with the coverage package. The tests are located in the `src` folder. The tests are executed with the following command:

```
cd src
python python -m tests -v
```

The tests are not covering all the code. The tests are only covering the code that is related to the query generation and some visualization code.

Accessing the application

The dashboard can be locally accessed through the following URL: http://localhost:6540

Make sure that the port `6540` is not used by another application or instance running in order to avoid conflicts.

Configuration

The configuration of the weather monitor can be done through the `docker-compose.yml` file. The following options are available:

Option	Description	Default
`INFLUXDB_HOST`	Hostname of the InfluxDB instance	`influxdb`
`INFLUXDB_PORT`	Port of the InfluxDB instance	`8086`
`INFLUXDB_PASSWORD`	Password used on the InfluxDB instance	`mysecretpassword`

The only exception is the `INFLUXDB_PASSWORD` option. This option has to be set in the environment file `.env` in the root of the project. This file is not included in the repository. The `.env` file has to be created manually. The example configuration must be copied to a `.env` file in order to utilize it.

The `.env` file is ignored by Git and will not be committed to the repository. This is done to prevent the accidental commit of sensitive information. The `.env` file is also ignored by Docker Compose. This is done to prevent the accidental commit of sensitive information to the Docker image.

You can copy and edit the example configuration using the following command (Linux):

```
cp example.env .env
nano .env
```

> **Note**: This is necessary because the `docker-compose.yml` file is configured to use the `.env` file as configuration file.

After that the docker stack has to be redeployed (if already running or not) using the following command:

```
sudo docker compose up -d
```

Updating the application

The application pulls the latest version of the Docker image from the GitHub Container Registry on every boot. This means that the application will automatically update to the latest version regularly.

This is done through the Watchtower Docker container. The Watchtower container will automatically update the Docker container running the weather monitor application as well as the InfluxDB container.

If further development is done on the application, the Docker image has to be rebuilt and pushed to the GitHub Container Registry or any other Docker registry. The Watchtower container will automatically pull the latest version of the Docker image from the registry and update the running containers.

With a registry change, the `docker-compose.yml` file has to be updated to use the new registry. The `image` option has to be updated to use the new registry.

Adding new stations

The weather monitor can be extended to support new weather stations. The following steps have to be done to add a new weather station:

1. Add the new weather station identifier (e.g. 'mythenquai') to the `stations` list within the `Config` class in the `src/weather_data.py` file.
2. Extend or create a new module with `src/weather_data.py` as a role model to fetch data from the data source that provides the weather data for the new weather station.
   i. Be sure to follow the structure of the InfluxDB when adding new data points. The structure of the InfluxDB can be found in the `src/weather_data.py` file.
3. Schedule the fetching of the new weather station data in the `src/app.py` file. The fetching of the data can be scheduled using the `schedule.every().hour.do(fetch_weather_data)` function. The `fetch_weather_data` is a placeholder for a function that can be found in the new or extended module created for that station.

If the data from the new data source and station is successfully fetched and stored in the InfluxDB, the weather monitor will automatically generate a new prediction and plot for the new weather station. The new weather station will also be available in the dashboard.

> **Note**: The weather monitor will only generate a new prediction and plot if the data for the new weather station is available in the InfluxDB. This means that the weather monitor will not generate a new prediction and plot for the new weather station if the data with the needed variables for the respective operation and new weather station is not available in the InfluxDB.

Adding new plots

The weather monitor can be extended to support new plots. The following steps have to be done to add a new plot:

1. Create a function that generate the plot in the `src/plotting.py` file.
2. Save the plot with the help of the `save_plot()` function in the `src/plotting.py` file.
3. Add the new plot identifier (e.g. 'wind_speed_avg_30h') to the list within the `get_plots()` function in the `src/plotting.py` file.
4. Add the function call of the newly created function to the `generate_plots()` function in the `src/plotting.py` file.

After that, the weather monitor will automatically generate the new plot and make it available in the dashboard.

Adding new measurements to the dashboard

The weather monitor can be extended to support new measurements. The following steps have to be done to add a new measurement:

1. Add the measurement identifier (e.g. 'wind_speed_avg_30h') matching with the corresponding field in the InfluxDB to the `show_current` list in the `src/app.py` file.
2. Add the unit of the newly added measurement to the `unit_mapping` dictionary in the `src/weather_repositor.py` file.

After that, the weather monitor will automatically show the new measurement on the dashboard.

Predictions

Currently, the application predicts the **wind speed** (m/s) and the **wind direction** (°) in a K-nearest neighbor model which was specifically trained on the pre-existing data from the Tecdottir API which provides data from both **Mythenquai** and **Tiefenbrunnen**.

As already shown in the Project Structure, there is a **prediction** directory in which everything about the weather prediction model can be accessed and modified.

The input data, which is used for training, consists of two separate CSV files (one for each station):

- messwerte_mythenquai_2007-2021.csv
- messwerte_tiefenbrunnen_2007-2021.csv

The model was trained on data from the timespan of the year 2007 until 2021. To have as much data, to rely on, as possible, the dataset was concatenated from both stations into one single dataset, tough, the source (station) was kept as an input feature through label encoding.

Handling of NA values

The volume of NA values does not significantly change the rest size of the input data which is why we just dropped rows that had NA values. Through that, still 1516655 rows persist; That is 5260 Days of data per station which seems to be enough for this use case.

List of features

To predict the **wind speed** and **wind direction** we relied on following attributes:

- Station
- Air Temperature from 10min before
- Wind Speed Average from 10min before
- Wind Direction from 10min before
- Current Day
- Current Month
- Current Year

- Current Wind Speed Average *(Goal variable)*
- Current Wind Direction *(Goal variable)*

As the list already hints, two output variables were chosen. To predict two variables from one model SciKit Learn's MultiOutputRegressor was used.

> **Note**: Since the target variables are numerical and thus continuous, the regression variant of the KNN algorithm is used.

Measuring Accuracy

SciKit Learn's Machine Learning algorithms provide a `score()` function to each fitted model. Through calling `model.fit().score()` we measured an accuracy of about 70% throughout our model. This value is calculated by dividing the correctly predicted values by all predicted values in the training set.

By using the Features list above we found that those features provide the best accuracy while training the model in a timely manner (resp. Curse of Dimensionality).

Implementation into production

To use the trained model in the application the package Pickle was used. Pickle dumps a *pkl* file onto the Filesystem which then can be used with `pickle.load(open('model.pkl', 'rb')).predict(input_variables)` to predict future measurements based on a persisted model.

Since the model with over 1.5M rows of input data is too large to keep track of in most versioning systems, an alternative flow was implemented when retraining a new model:

1. Upload the model to a cloud provider (e.g. Google Drive, Dropbox or AWS)
2. Update the model downloading bash script with your own choice of cloud reference to the uploaded model
3. Restart the host (in this case the Raspberry Pi) *this will refetch the updated model*

Shutdown

The following command can be used to shut down the weather monitor in development mode (Docker):

```
docker-compose -f docker-compose-dev.yml down
```

If you're running the weather monitor in development mode and with Python, the weather monitor can be shut down by pressing `CTRL+C` in the execution terminal.

Known Issues

- Running the application outside Docker on a Windows machine is not supported. The application can be run using WSL on Windows.
    - This issue is related to the saving of plots on a Windows file system. The application will not be able to save the plots on a Windows file system. See function `save_plot` in `src/plotting.py`.

Improvements

The current state of the weather monitor is still a proof of concept. The following improvements can be made to the weather monitor:

- The weather monitor can be extended to support more weather stations.
- Combination of the weather stations can be used to improve the prediction accuracy. Currently, no overall dashboard or prediction is available which summarizes the weather data from all weather stations. This may be a desired feature of clients.
- Possibility of customizing the dashboard. Currently, the dashboard is not customizable. Customization of the dashboard may include the possibility to add or remove weather stations from the dashboard based on the client's location or adding custom plots based on measurements from the weather stations.

Future Roadmap

- [ ] Add more stations
- [ ] Add a summary view of all stations
- [ ] Add customizable dashboard and plots
- [ ] Artificial intelligence to predict the weather
- [ ] Add more data sources (e.g. weather radar, weather satellites, etc.)