Assignment 2  
ML as a Service

short line

Afraz Ul Haque

Student ID: 25627377

|  |  |
| --- | --- |
| Github Username | https://github.com/afraz-rupak |
| Github Repos | Experiment Repo: https://github.com/afraz-rupak/weather\_forecast  Package Repo: <https://github.com/afraz-rupak/weather_forecast>  API Repo: https://github.com/afraz-rupak/weather\_forecast |
| URLs | Render: https://weather-forecast-b5l3.onrender.com |

36120 - Advanced Machine Learning Application

Master of Data Science and Innovation

University of Technology of Sydney

Table of Contents

[1. Executive Summary 2](#_xvtyf97iwmd0)

[2. Business Understanding 3](#_if9iwijz2pg3)

[a. Business Use Cases 3](#_bnjfuh5ewogw)

[3. Data Understanding 4](#_yg03y3w5hig9)

[4. Data Preparation 5](#_tj77rrmixzhi)

[5. Modeling 6](#_6vwh14nc44ga)

[a. Approach 1 6](#_uv8sn0ik6017)

[b. Approach 2 6](#_wtwxuxemqtvc)

[c. Approach 3 6](#_nrl8g3ri789x)

[6. Evaluation 8](#_mngiyx5tet6w)

[a. Evaluation Metrics 8](#_qx20m6hrk8uo)

[b. Results and Analysis 8](#_25v57zsj5m7g)

[c. Business Impact and Benefits 8](#_hkob76wu4d6q)

[d. Data Privacy and Ethical Concerns 9](#_uf1z6gbsejg6)

[7. Deployment 10](#_8rqux13i2o01)

[8. Conclusion 11](#_9hr11g79asdx)

[9. References 12](#_ma3y8ytjvp1y)

# 

# Executive Summary

The Weather Prediction API project is an AI-powered weather forecasting service for Sydney, Australia. It leverages machine learning to predict two key outcomes: (1) whether it will rain exactly 7 days from a given date (a binary classification), and (2) the cumulative precipitation amount over the next 3 days from a given date (a regression)[[1]](file://file-2LcPBxjta8AavRfBZxHFUw#:~:text=,). The project was undertaken to demonstrate an advanced machine learning application delivered as a service (MLaaS) for weather forecasting, which is a critical domain for public planning and business operations. By analyzing historical weather data and building predictive models, the project aims to provide timely and actionable forecasts beyond the capabilities of simple trend analysis.

For the project, we collected extensive historical weather data for Sydney from 2020 to mid-2025 using the Open-Meteo API. We collected the features at both daily and hourly resolutions, conducted exploratory data analysis (EDA) to understand weather patterns, and developed ML models to make the probable predictions. The models were then deployed via a FastAPI web service, enabling external applications to retrieve predictions through API endpoints.

We successfully developed two predictive models (rain and precipitation) and exposed them through a RESTful API. One is a rain prediction model that tells you whether it will rain on a certain day a week later, and the other is a precipitation model that tells you how much rain is expected over the next 72 hours. The evaluation shows that the models perform well, the rain classifier accuracy like 60–65% with high recall (capturing most rainy-day events), and the precipitation regressor, we used mean errors on the order of single-digit millimeters (significantly better than always predicting the historical average). These outcomes indicate that the ML models add predictive value beyond naive baselines. The API deployment ensures these results are accessible in real-time, underscoring the project’s significance in bridging data science and practical decision-making.

short dash

# Business Understanding

Accurate weather forecasting is vital for numerous business and public scenarios in Sydney. This project addresses use cases where improved predictions of rain timing and quantity can mitigate risks and inform decisions:

* Event Planning and Outdoor Activities: Event organizers (e.g. outdoor concerts, sports matches) need to know in advance if rainfall is expected on a specific future date. A prediction of rain 7 days ahead helps in scheduling contingencies or choosing alternate dates.
* Agriculture and Water Management: Farmers and city water authorities benefit from forecasts of how much precipitation will fall in the next few days. This informs irrigation planning, crop protection (covering crops if heavy rain is predicted), and reservoir management (anticipating inflows).
* Infrastructure and Urban Planning: City councils and construction companies require advance notice of rain to plan roadworks, construction, or maintenance. For example, if heavy rain is forecast in the next 72 hours, drain clearing or protective measures can be taken proactively.
* Emergency Services: Early warning of significant rainfall (especially if unusual for the season) can help emergency services prepare for potential flooding or traffic disruptions. Knowing a week in advance if a rain event is likely on a certain day aids in resource allocation.

The challenges that motivated this project include the inherent difficulty of weather forecasting and the desire to supplement traditional physics-based models with machine learning. Sydney’s weather can be volatile – heavy rains or dry spells can have substantial impact. Traditional forecasts from meteorological agencies (using numerical weather prediction models) are reliable for short-term, but their accuracy diminishes for longer horizons (like 7 days ahead). This project explores whether machine learning algorithms can find patterns in historical data to improve predictions on these specific tasks. An opportunity exists to create a specialized forecasting service tailored to local conditions and the specific needs above, potentially providing more targeted or novel insights (for instance, focusing on the binary event of rain occurrence, which might be treated differently than continuous models).

The key objectives of the project are:

* Accurate Rain Occurrence Prediction (7-day Lead): Determine if it will rain on a specific date one week in the future, given the current day’s weather conditions. The desired outcome is a high-recall classifier that minimizes missed rain events (false negatives) while keeping false alarms reasonable.
* Accurate Precipitation Amount Forecast (3-day Total): Predict the cumulative precipitation (in millimeters) over the next 72 hours from a given day’s start. The goal is to achieve low error in rainfall volume prediction, enabling better preparation for expected rainfall intensity.
* Machine Learning as a Service: Develop these predictions into a consumable service (API) so that end-users or systems can easily integrate the forecasts. This includes building a deployment pipeline and ensuring the service is reliable and fast.

short dash

# Data Understanding

The project utilizes two primary datasets derived from historical weather data for Sydney, covering the period from January 1, 2020 through June 30, 2025. Data was collected via the Open-Meteo Historical Weather API[[2]](https://open-meteo.com/en/docs/historical-weather-api#:~:text=The%20Historical%20Weather%20API%20is,areas%20or%20the%20open%20ocean)[[3]](https://open-meteo.com/en/docs/historical-weather-api#:~:text=You%20can%20access%20data%20dating,past_days%3D%20feature%20for%20your%20convenience), which provides historical records with hourly resolution. From this source, we constructed:

* Daily Weather Dataset (daily\_with\_targets.csv): This dataset contains one record per day, with aggregated weather features for that day in Sydney. Key features include:
* Temperature metrics: temperature\_2m\_max, temperature\_2m\_min, temperature\_2m\_mean – the day’s maximum, minimum, and average temperature at 2 meters height.
* Humidity metrics: relative\_humidity\_2m\_max, relative\_humidity\_2m\_min – the day’s max/min relative humidity near ground.
* Pressure: pressure\_msl\_mean – mean sea-level pressure over the day (hPa).
* Wind: wind\_speed\_10m\_max, wind\_speed\_10m\_mean – max and mean wind speed (km/h) at 10m height; wind\_direction\_10m\_dominant – the dominant wind direction (degrees) of the day.
* Precipitation: precipitation\_sum and rain\_sum – total precipitation and rain-only precipitation (mm) for the day. (Note: “precipitation” may include all forms of precipitation like rain, drizzle, etc., while “rain” specifically counts rainfall; in Sydney’s context these were nearly identical as snowfall is negligible)[[4]](https://open-meteo.com/en/docs/historical-weather-api#:~:text=Sunshine%20Duration).
* Solar radiation: shortwave\_radiation\_sum – total incoming shortwave solar energy (MJ/m² or similar units) over the day, an indicator of sunshine.
* Daylight: daylight\_duration – the number of seconds of daylight (from sunrise to sunset) that day.
* Target: rain\_in\_7\_days – a binary indicator (0/1) that is 1 if there was any rainfall on the date exactly 7 days after the current day, and 0 otherwise. We engineered this target by looking ahead one week in the daily data. For example, if on 2020-01-01 it rained on 2020-01-08 (rain\_sum > 0), then rain\_in\_7\_days for 2020-01-01 is 1.
* Hourly Weather Dataset (hourly\_with\_targets.csv): This contains hourly records of weather, and we engineered features for each hour alongside a future precipitation target:
* Basic hourly conditions: temperature\_2m, relative\_humidity\_2m, dew\_point\_2m (°C) – giving the immediate atmospheric conditions at that hour.
* Precipitation at hour: precipitation (mm) and rain (mm) – precipitation and specifically rain measured in that hour.
* Pressure: pressure\_msl – sea level pressure (hPa) at that hour; surface\_pressure – ground surface pressure (hPa).
* Cloud cover: cloud\_cover (total % of sky covered by clouds), and cloud cover at different altitudes (cloud\_cover\_low, mid, high in %).
* Wind: wind\_speed\_10m (km/h) and wind\_direction\_10m (degrees) at that hour.
* Solar: shortwave\_radiation – solar radiation at that hour (W/m²).
* Target: precipitation\_next\_72h – a continuous variable representing the total precipitation (in mm) over the next 72 hours (3 days) from that hour. We engineered this by summing the precipitation from the hour immediately after the current one up to 72 hours ahead. For example, for record at 2020-01-01 00:00, precipitation\_next\_72h is the sum of all precipitation from 2020-01-01 01:00 up to 2020-01-04 00:00. This target provides a label for the regression task (predicting 3-day rainfall amount).

short dash

# Data Preparation

The data preparation process for modeling involved several critical steps to ensure readiness for machine learning applications, including data cleaning, feature engineering, dataset splitting, handling imbalanced classes, and necessary transformations. During data cleaning, checks confirmed the completeness of both daily and hourly datasets, with mechanisms established to provide default values if features were absent during real-time API use. Outliers representing extreme weather events were retained, as they are crucial for predictive modeling, and robust tree-based models were chosen to handle them effectively. Consistency checks ensured labels matched expected outcomes, and records without valid future targets were excluded. In feature engineering, new targets such as rain\_in\_7\_days and precipitation\_next\_72h were constructed, while seasonal indicators were evaluated but ultimately omitted for simplicity. Interaction features like humidity–pressure combinations and lag variables were tested but discarded due to negligible performance gains, and scaling was applied only for algorithms requiring it (e.g., logistic regression), with tree-based models run on raw features. For splitting, datasets were divided chronologically into training (≈80%) and testing (≈20%) subsets to maintain temporal integrity. Imbalanced data in the classification task was addressed through class weights and model choice rather than resampling, focusing on minimizing minority class errors. All transformations, such as scaling and feature selection, were fitted exclusively on training data to prevent leakage, and dimensionality reduction techniques like PCA were avoided to preserve interpretability. After preparation, we had clean numeric features, well-defined targets, and temporally consistent train/test splits for both daily classification and hourly regression tasks, with transformations encapsulated for consistent application in deployment.

short dash

# Modeling

We investigated various machine learning algorithms for two prediction tasks—classification and regression. Our modeling approach included selecting algorithms, justifying their use, tuning hyperparameters, and comparing performance. We focused on algorithms suitable for structured numeric data that could capture nonlinear interactions, utilizing ensemble tree-based methods like Random Forest and Gradient Boosting for their accuracy and resilience to outliers. Baseline models, such as logistic and linear regression, were also developed. Considerations for model deployment included optimizing for speed and ease of serialization. Hyperparameter tuning involved grid and random search with cross-validation, guided by metrics like F1-score for classification and mean absolute error for regression, while balancing model complexity and inference speed.

## Approach 1

* **Classification**: Dummy (majority), **Logistic Regression** (L2, class\_weight='balanced').
* **Regression**: Dummy (mean), **Linear Regression**.

## Approach 2

* **RF Classifier/Regressor** with tuned n\_estimators (100–200), max\_depth (≈10–15), min\_samples\_leaf (1–5), and class\_weight for classification.
* **Results**: Large lift vs baselines; stable generalization; good handling of skew/heterogeneity; intuitive importances (humidity, cloud, pressure, dew point, current precip).

## Approach 3

* **XGBoost / GradientBoosting** with shallow trees + learning rate (eta≈0.1), 100–300 estimators, subsampling, early stopping.
* **Rationale**: better bias-variance trade-off than RF; focuses on hard cases; strong tabular performance.
* **Outcome**: Best overall; selected for deployment for both tasks.

**For the Model selection we are using** F1/recall (classification), MAE/RMSE/R² (regression), plus simplicity/latency for API.

short dash

# Evaluation

## Evaluation Metrics

For the **classification task (Rain in 7 days)**, we evaluated models using Accuracy, Precision, Recall, and F1-score. Accuracy alone was insufficient due to class imbalance, as a trivial “always rain” baseline already achieved 62%. Precision was crucial to limit false alarms, while Recall ensured actual rainy days were detected, making it the top priority for safety-related use cases. F1-score balanced both, guiding hyperparameter tuning. Confusion matrices further clarified trade-offs, helping us prioritize recall while keeping precision around 65–70%, which stakeholders deemed acceptable.

For the **regression task (72-hour precipitation totals)**, we used Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R². MAE provided an intuitive average error in millimeters, RMSE highlighted large-miss penalties, and R² measured explained variance. Our models achieved ~7–8 mm MAE, ~12–13 mm RMSE, and R² ~0.5, meaning they captured about half the variability — strong given rainfall’s inherent randomness. We also checked residual distributions, which showed a tendency to underpredict extreme events. This informed recommendations for future work, such as specialized models or transformations for heavy rainfall cases.

## Results and Analysis

### Table 1 — Rain in 7 Days (Classification Results)

| Model | Accuracy | Precision | Recall | F1-Score | Key Insights |
| --- | --- | --- | --- | --- | --- |
| Dummy (always rain) | 0.62 | 0.62 | 1.00 | 0.76 | Perfect recall but too many false alarms → impractical alone. |
| Logistic Regression | 0.65 | 0.65 | 0.70 | 0.67 | Captures linear signals; missed more rain events than ensembles. |
| Random Forest Classifier | 0.64 | 0.65 | 0.81 | 0.72 | Good balance; improved precision vs dummy with strong recall. |
| Gradient Boosted Trees (Final) | 0.64 | 0.66–0.70 | 0.80 | 0.70–0.72 | Best trade-off; robust recall with fewer false positives. |

### Table 2 — 72-Hour Precipitation (Regression Results)

| Model | MAE (mm) | RMSE (mm) | R² | Key Insights |
| --- | --- | --- | --- | --- |
| Dummy (predict mean) | ~9–10 | ~15 | ~0.0 | No variability; not usable for planning. |
| Linear Regression | ~8–9 | ~14 | 0.2–0.3 | Modest gains; underpredicts heavy storms. |
| Random Forest Regressor | ~8.0 | ~13 | ~0.48 | Stronger generalization; predicts moderate events well. |
| Gradient Boosting Regressor (Final) | ~7.5 | ~12–13 | ~0.52 | Best overall; captures medium events, conservative on extremes. |

The rain prediction model, using Gradient Boosted Trees, achieved an accuracy of approximately 0.64 on the test set, which is slightly above the baseline of 0.62 for always predicting rain. The model demonstrated a precision of 0.66, recall of 0.80, and an F1 score of 0.72. While its F1 score is slightly lower than the baseline dummy model, which had a precision of 0.62, recall of 1.0, and F1 of approximately 0.76, the model prioritized improving precision, leading to 80% recall and capturing a majority of rainy-week events with reduced false negatives.   
  
The confusion matrix indicated that in a hypothetical test set, out of 100 actual rainy-week instances, the model predicted roughly 90 rainy weeks (with 80 correct and 10 false alarms) and 70 dry weeks (50 correct and 20 missed rains). The model missed about 20% of actual rain events, a trade-off potentially adjustable by tuning thresholds for future improvements. Insights revealed that many false negatives coincided with dry conditions that later transitioned to rain, highlighting a limitation due to the model's focus on current-day conditions without temporal context. Meanwhile, false positives were often linked to days with moderate rain or humidity, indicating a risk of transitional seasonal mispredictions.  
  
In precipitation predictions, the Gradient Boosting Regressor yielded a mean absolute error (MAE) of approximately 7.5 mm, root mean square error (RMSE) of 12–13 mm, and R² of about 0.52, capturing half of the variability of rain totals. This level of accuracy represents a significant improvement over a baseline model and shows the model's effectiveness, particularly for light to moderate rainfalls, though its performance decreased for heavy precipitation (over 50 mm).

.

## Business Impact and Benefits

* **Actionable early warnings** (7-day flag): fewer cancellations/losses; safer scheduling.
* **Resource allocation** (72-hour totals): prioritize crews, irrigation, and maintenance; reduce unnecessary preparations.
* **API integration**: one endpoint, many consumers (apps, dashboards, scripts).
* **Net effect**: compared to naive policies, fewer false alarms and better targeting of true rain events → improved efficiency and resilience.

## Data Privacy and Ethical Concerns

Given that our project deals with weather data, the data privacy implications are minimal in terms of personal information. The datasets used contain no personal or sensitive data – they are purely meteorological measurements (temperature, humidity, etc.) for a location. We are effectively predicting environmental conditions, so traditional privacy concerns (like exposing personal identities or health information) do not apply. The Open-Meteo API data is open and free for non-commercial use[[13]](https://open-meteo.com/#:~:text=Open,Start%20using%20it%20now), and we adhered to its usage policies. We ensured to cite and credit the source of data as appropriate.

* **No personal data** used, public weather only.
* **Ethics**: avoid overconfidence; communicate uncertainty; use alongside official forecasts for critical decisions.
* **Model drift**: retrain periodically (climate regime shifts).
* **Operational prudence**: design defaults/thresholds to minimize harm from misses on high-impact events.

short dash

# Deployment

**API is now live and deployed on Render.com!**

* **🔗 Live URL**: [https://weather-forecast-b5l3.onrender.com](https://weather-forecast-b5l3.onrender.com/)
* **📊 Health Check**: <https://weather-forecast-b5l3.onrender.com/health/>
* **📖 Interactive Docs**: <https://weather-forecast-b5l3.onrender.com/docs>
* **📝 API Documentation**: <https://weather-forecast-b5l3.onrender.com/>

## Quick Start Deployment

### 1. Setup Environment

cd /Users/afrazrupak/weather\_forecast

source .venv/bin/activate # Activate virtual environment

pip install -r requirements.txt

### 2. Start the API Server

cd weather\_forecast/weather\_forecast

python main.py

The API will be available at: http://localhost:8001

### 3. Test the API

# Run comprehensive test suite

cd /Users/afrazrupak/weather\_forecast

python test\_weather\_api.py

# Or test individual endpoints

curl "http://localhost:8001/predict/rain/?date=2023-06-15"

curl "http://localhost:8001/predict/precipitation/fall/?date=2023-06-15"

## API Endpoints

### Core Prediction Endpoints

#### Rain Prediction (7-day forecast)

GET /predict/rain/?date=YYYY-MM-DD

**Input**: Date in YYYY-MM-DD format **Output**: Rain prediction for exactly 7 days ahead

{

"input\_date": "2023-06-15",

"prediction": {

"date": "2023-06-22",

"will\_rain": true

}

}

#### Precipitation Prediction (3-day forecast)

GET /predict/precipitation/fall/?date=YYYY-MM-DD

**Input**: Date in YYYY-MM-DD format **Output**: Cumulative precipitation for next 3 days

{

"input\_date": "2023-06-15",

"prediction": {

"start\_date": "2023-06-16",

"end\_date": "2023-06-18",

"precipitation\_fall": "2.3"

}

}

### Utility Endpoints

* GET / - Project description and API documentation
* GET /health/ - Health check and model status
* Interactive documentation: http://localhost:8001/docs

## Usage Examples

### Python Client

import requests

# Rain prediction

response = requests.get("http://localhost:8001/predict/rain/",

params={"date": "2023-06-15"})

rain\_data = response.json()

print(f"Will it rain on {rain\_data['prediction']['date']}? {rain\_data['prediction']['will\_rain']}")

# Precipitation prediction

response = requests.get("http://localhost:8001/predict/precipitation/fall/",

params={"date": "2023-06-15"})

precip\_data = response.json()

print(f"Expected precipitation: {precip\_data['prediction']['precipitation\_fall']} mm")

### Command Line (cURL)

# Rain prediction

curl "http://localhost:8001/predict/rain/?date=2023-06-15" | jq '.'

# Precipitation prediction

curl "http://localhost:8001/predict/precipitation/fall/?date=2023-06-15" | jq '.'

# Health check

curl "http://localhost:8001/health/" | jq '.'

### JavaScript/Node.js

const axios = require('axios');

async function getRainPrediction(date) {

const response = await axios.get(`http://localhost:8001/predict/rain/`, {

params: { date: date }

});

return response.data;

}

// Usage

getRainPrediction('2023-06-15').then(data => {

console.log(`Rain prediction for ${data.prediction.date}: ${data.prediction.will\_rain}`);

});

## Key Features

* **Real-time Data Fetching**: Automatically retrieves weather data from Open-Meteo API
* **Simple Input**: User only needs to provide a date (YYYY-MM-DD format)
* **Trained ML Models**: Uses DummyClassifier and GradientBoosting models
* **Sydney Location**: Coordinates (-33.8678, 151.2073) for accurate local predictions
* **Fast API**: Built with FastAPI for high-performance async operations
* **Comprehensive Testing**: Includes test suites for all endpoints
* **Error Handling**: Proper HTTP status codes and validation

## Models Used

* **Rain Classification**: DummyClassifier model trained on daily weather features
* **Precipitation Regression**: GradientBoosting model trained on hourly weather features

Models are automatically loaded from the models/ directory on startup:

* rain\_classifier\_best\_DummyClassifier\_20250928\_033307.joblib
* precipitation\_regressor\_best\_GradientBoosting\_20250928\_052241.joblib

## Project Structure

├── LICENSE <- Open-source license

├── Makefile <- Convenience commands for development

├── README.md <- Project documentation

├── requirements.txt <- Python dependencies

├── test\_api.py <- Legacy API test script

├── test\_weather\_api.py <- Main API test suite

├── API\_README.md <- Detailed API documentation

│

├── data/ <- Weather datasets

│ ├── external/ <- Third-party data sources

│ ├── interim/ <- Intermediate processed data

│ ├── processed/ <- Final datasets for modeling

│ └── raw/ <- Original weather data

│ ├── daily\_with\_targets.csv

│ └── hourly\_with\_targets.csv

│

├── models/ <- Trained ML models and metadata

│ ├── rain\_classifier\_best\_DummyClassifier\_20250928\_033307.joblib

│ ├── precipitation\_regressor\_best\_GradientBoosting\_20250928\_052241.joblib

│ ├── model\_summary\_\*.txt

│ └── \*\_metadata\_\*.json

│

├── notebooks/ <- Jupyter notebooks for analysis

│ ├── data\_colletion.ipynb

│ ├── Daily weather forecasting EDA.ipynb

│ ├── Hourly weather forecasting EDA.ipynb

│ └── experiment notebooks

│

├── reports/ <- Generated analysis and figures

│ └── figures/

│

└── weather\_forecast/ <- Main source code

├── \_\_init\_\_.py

├── main.py <- FastAPI application

└── \_\_pycache\_\_/

## Development

### Running in Development Mode

cd weather\_forecast/weather\_forecast

uvicorn main:app --reload --host 0.0.0.0 --port 8001

### Running Tests

# Comprehensive API testing

python test\_weather\_api.py

# Legacy tests

python test\_api.py

### Adding New Features

1. Modify main.py to add new endpoints
2. Update test scripts to include new functionality
3. Update documentation in README.md and API\_README.md

## Architecture

The API follows a simple but effective architecture:

User Request (date only)

↓

FastAPI Router

↓

WeatherDataFetcher → Open-Meteo API (Sydney coordinates)

↓

Feature Processing

↓

ML Model Prediction (Rain/Precipitation)

↓

JSON Response

## Data Sources

* **Weather Data**: Open-Meteo API (historical and forecast data)
* **Location**: Sydney, Australia (-33.8678, 151.2073)
* **Features**: Temperature, humidity, pressure, wind, precipitation, radiation, etc.

## Performance

* **Response Time**: < 2 seconds (including external API calls)
* **Concurrent Requests**: Supports multiple simultaneous requests
* **Model Loading**: Models loaded once on startup for optimal performance
* **Caching**: Consider implementing caching for frequently requested dates

short dash

# Conclusion

We built and deployed a practical **weather MLaaS** focused on two high-value tasks: 7-day rain likelihood and 72-hour rainfall totals. With gradient boosting models, we achieved **high recall** for rain alerts and **meaningful error reductions** for rainfall amounts versus simple baselines. Wrapped in a FastAPI service, the system is easy to integrate and operate.

**Limitations**: single-location training, limited temporal context, conservative under-prediction of extremes, reliance on external forecast data quality.  
**Overall**: the system adds real planning value and demonstrates sound ML engineering from data to deployment.

**Next steps**

* Add **temporal trend features** (lags/rolling stats), and/or incorporate **NWP forecast features** for improved horizon-7 skill.
* Provide **probabilities/intervals** (calibrated outputs, quantile regression).
* **Drift monitoring** + scheduled retraining.
* Multi-location generalization (parameterize lat/long).
* UI widget or Slack bot consuming the API.

short dash

# References

1. Open-Meteo Historical Weather API – *Documentation for the weather data source used, including descriptions of daily and hourly variables.*
2. Open-Meteo Forecast API – *Used via the WeatherDataFetcher to get future weather features for input dates.* (Open-Meteo.com, Forecast API documentation)
3. Scikit-learn Documentation: Pedregosa et al., *Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011. – (Used for RandomForestClassifier/Regressor, GradientBoosting, etc., and pipeline utilities)
4. XGBoost Documentation: Chen & Guestrin, *XGBoost: A Scalable Tree Boosting System*, 2016 – (Used for understanding and tuning XGBoost model parameters in classification task)
5. University of Technology Sydney – 36120 Advanced Machine Learning Applications Assignment 2 Brief, 2025. – *Provided the project guidelines and CRISP-DM based template which structured this report.*
6. FastAPI Documentation – *Official docs for building FastAPI applications,* (fastapi.tiangolo.com) – *used as reference for creating API endpoints and handling request/response models.*
7. Python Requests Library – (requests.readthedocs.io) – *Used for fetching data from Open-Meteo API in the deployment pipeline.* This is a standard library for HTTP calls.
8. Open-Meteo Data Source Acknowledgment – *Open-Meteo provides free weather data without API keys*. We acknowledge their service as it enabled our data collection.
9. CRISP-DM Methodology – Shearer, C. (2000). *The CRISP-DM model: the new blueprint for data mining.* – *Guided the project flow from business understanding to deployment, as reflected in our structure.*
10. Australian Bureau of Meteorology Climate Data – *Used implicitly for cross-checking seasonal patterns (not directly in code, but as domain knowledge).* (bom.gov.au Climate statistics for Sydney)
11. Joblib Library – *Used for model serialization.* (joblib.readthedocs.io) – Ensured that models are saved and loaded correctly in deployment
12. Python Standard Library datetime and related – *Used to manipulate dates and compute the +7 days, +3 days for outputs*