Weather Forecasting Using Machine Learning

Link to GitHub Repository

Link to Web Application

Abstract

This project leverages historical U.S. weather data to predict weather conditions using machine learning. After thorough data exploration and preprocessing, multiple models were evaluated, with **XGBoost** emerging as the top performer. The final model is integrated into a Dash web app, enabling users to input weather parameters and receive real-time predictions. The project demonstrates a complete ML pipeline, from data processing to deployment.

2025-04-23

Markus Reblin 578083
Ulrigh Oosthuizen 577952
Vutivi Maswanganyi 577800
Jonathan Joubert 578085

Contents

Problem Statement

Rainfall prediction is essential for sectors such as agriculture, transportation, and public safety. Traditional forecasting methods, while accurate, often require complex simulations and specialized infrastructure, limiting accessibility for broader, real-time use.

This project proposes a machine learning-based approach using Extreme Gradient Boosting (XGBoost) to predict daily rainfall based on historical U.S. weather data. Key features include **temperature, humidity, wind speed**, and **weather conditions**. The model will be integrated into a **Dash web application**, enabling users to input current weather data and receive instant predictions. This system aims to deliver an **accurate, efficient, and accessible tool** for practical rain forecasting in real-world settings.

Hypotheses

1. Data Relationship Hypotheses

- **H_1.1 (Primary Predictive Relationship):**
  There is a statistically significant relationship between weather features (temperature, humidity, wind speed, and general weather conditions) and the likelihood of rainfall.
- **H_1.2 (Feature Importance):**
  Among all features, humidity and weather condition will have the strongest predictive influence on rain occurrence.
- **H_1.3 (Interaction Effects):**
  The interaction between temperature and humidity will significantly improve the model's ability to predict rainfall.

2. Model Performance Hypotheses

- **H_2.1 (ML Superiority):**
  The XGBoost classification model will outperform a baseline model (e.g., logistic regression or random guessing) in predicting rainfall, as measured by accuracy, precision, and F1-score.
- **H_2.2 (Generalization):**
  The trained XGBoost model will generalize well to unseen test data, maintaining high predictive performance without significant overfitting.
- **H_2.3 (Hyperparameter Tuning Impact):**
  Optimizing hyperparameters through cross-validation will significantly improve model performance compared to default settings.

3. Application and Usability Hypotheses

- **H_3.1 (User Interaction):**
  Users will be able to input daily weather conditions into the Dash web application and receive accurate, real-time predictions about rainfall.

- **H_3.2 (Accessibility Impact):**
  Providing rain prediction through a web-based tool will enhance accessibility and practical decision-making for non-expert users such as farmers or event planners.
- **H_3.3 (Efficiency Hypothesis):**
  The model can deliver predictions with low latency, making it suitable for real-time use in web applications.

. 4. Null Hypotheses (H_0)

- **H_0.1:**
  There is no significant relationship between the selected weather features and the occurrence of rainfall.
- **H_0.2:**
  Machine learning models, including XGBoost, do not significantly outperform baseline methods in rain prediction.
- **H_0.3:**
  Deployment of the model in a web application will not impact user decision-making or accessibility.

Dataset Overview

Description

The dataset consists of **2,500 records** of historical weather data collected from multiple cities across the United States. Each entry represents weather conditions at a specific time and place, with several meteorological measurements recorded. The dataset is designed to support a binary classification **task** predicting whether it will rain on a given day based on weather features.

Key Features:

- **City** – The city where the weather observation was recorded
- **DateTime** – Timestamp of the weather record (date and time)
- **Temperature** – Air temperature in degrees Celsius
- **Humidity** – Relative humidity in percentage
- **Wind_Speed** – Wind speed in km/h
- **Cloud_Cover** – Estimated cloud cover percentage
- **Pressure** – Atmospheric pressure in hPa
- **Rain** – Target variable indicating whether it rained ("rain") or not ("no rain")

Target Variable:

- **Rain** – Categorical label with two classes: "rain" and "no rain", used as the output for classification

Dataset Characteristics:

- **Total Records:** 2,500
- **Data Type:** Structured, tabular data
- **Missing Values:** None (all columns have complete data)
- **Time Span:** Includes weather data across various years and times of day

Problems and Solutions

Environment: Python 3.12.7 (later switched to 3.13.1 for compatibility) Problems and solutions

This document outlines the issues encountered during the development of machine learning models, along with the steps taken to resolve them. It also includes model performance insights and observations during the testing phase.

General Issues

- Issue: .py file for splitting data did not function correctly.
  Solution: Switched to using a .ipynb notebook instead, which worked as expected.

Logistic regression:

- Initial model trained successfully with 0.933, or 93.3% Accuracy.
- Issue: Test results mirrored training results exactly.
- Investigation revealed the model was incorrectly trained on the test set and evaluated on the training set.
- Fix: Corrected the data split.
- After fixing: Accuracy slightly dropped to 92.6%.

Random Forest:

- Initial model trained successfully with 1.0, or 100% Accuracy.
- Concern: Accuracy appears too perfect.
- Ongoing: Investigating potential overfitting or data leakage.
- Fix: Corrected the data split.
- Dataset was updated for fairness and reliability of results.

XGBoost:

- Error: ModuleNotFoundError: No module named 'xgboost'
  Fixed via !pip install xgboost
- Error: ModuleNotFoundError: No module named 'category_encoders'
  Fixed via !pip install category_encoders
- Error: ImportError: cannot import name 'Tags' from 'sklearn.utils'
  Root Cause: Incompatibility between scikit-learn and category_encoders
  Fixed by removing unnecessary import
- Error: ModuleNotFoundError: No module named 'skopt'
  Fixed via !pip install scikit-optimize

- Error: ImportError: cannot import name '_check_n_features' from 'sklearn.utils.validation'
  Switched from **BayesSearchCV** to **GridSearchCV** to avoid compatibility issues with skopt
- Error: 'Pipeline' object has no attribute 'transform_input'
  Resolved by restarting the kernel
- Model trained with **99.6% accuracy**

After the initial phase of testing, all models now run successfully. The compatibility and library issues were addressed. Models are performing within acceptable ranges.

Accuracies from models:

Logistic Regression: 92.6%

Random Forest: 100%

XGBoost: 99.6%

Data Preprocessing, Exploration, and Analysis

Overview

The dataset is loaded using Pandas and the first few rows are displayed using display(df.head()). This helps verify that the dataset has been correctly imported and gives an initial look at the data structure. To further analyse the dataset structure df.info() is used to display the data types and non-null counts for each column. This is useful for identifying missing values and categorical variables. Additionally, df.describe().T is used to provide a transposed summary of numerical features, showing statistics such as mean, std deviation, and quartiles. Furthermore, df.isnull().sum() counts missing values in each column to identify where imputations or deletion might be necessary.

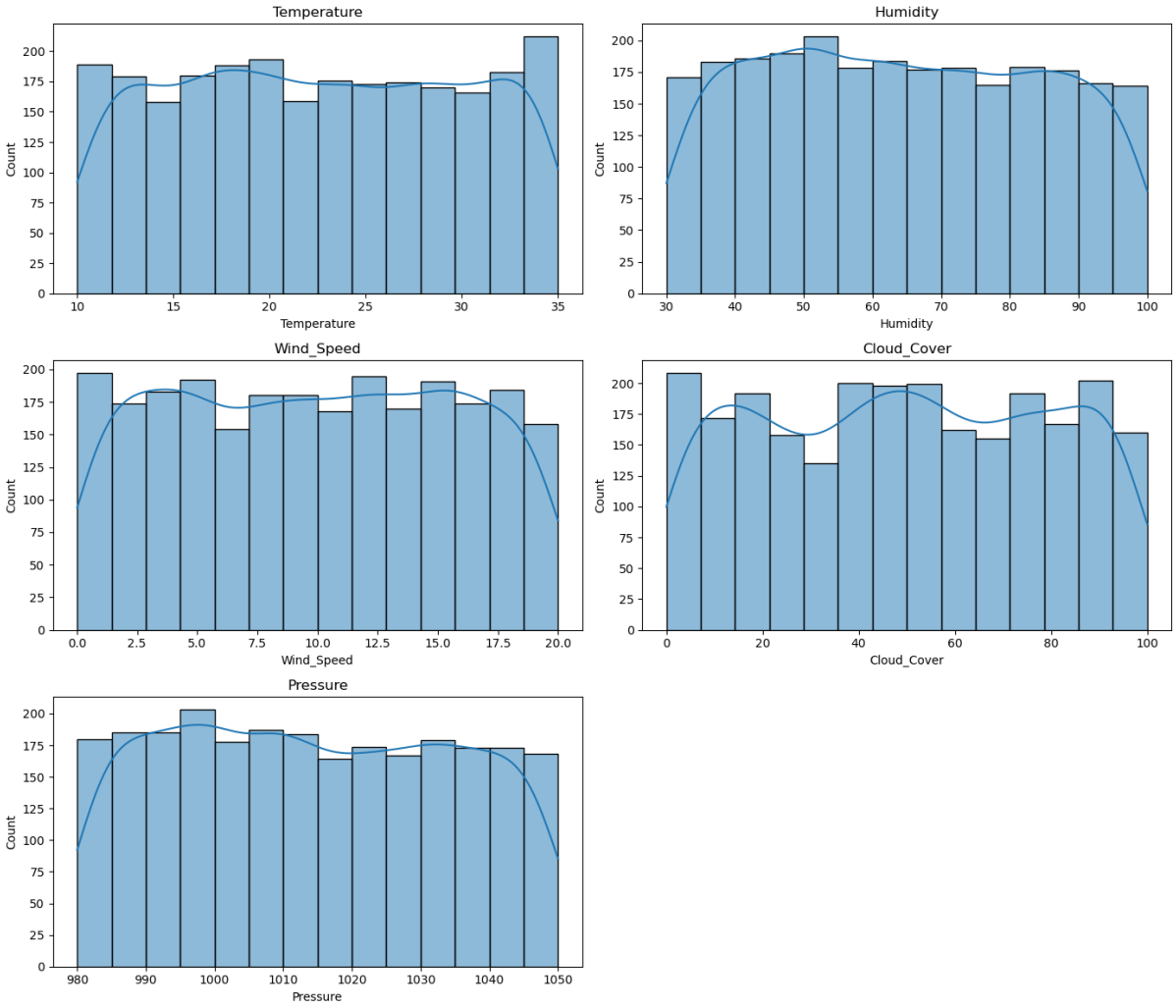
Data cleaning and Feature Analysis

The categorical feature “Rain” is encoded into binary values, converting “no rain” to 0 and “rain” to 1, which prepares it for machine learning models. The DateTime column is also converted from a string to datetime format, enabling time-based operations such as resampling. Data is then split up into numeric and categorical features for targeted processing and visualization of different data types.

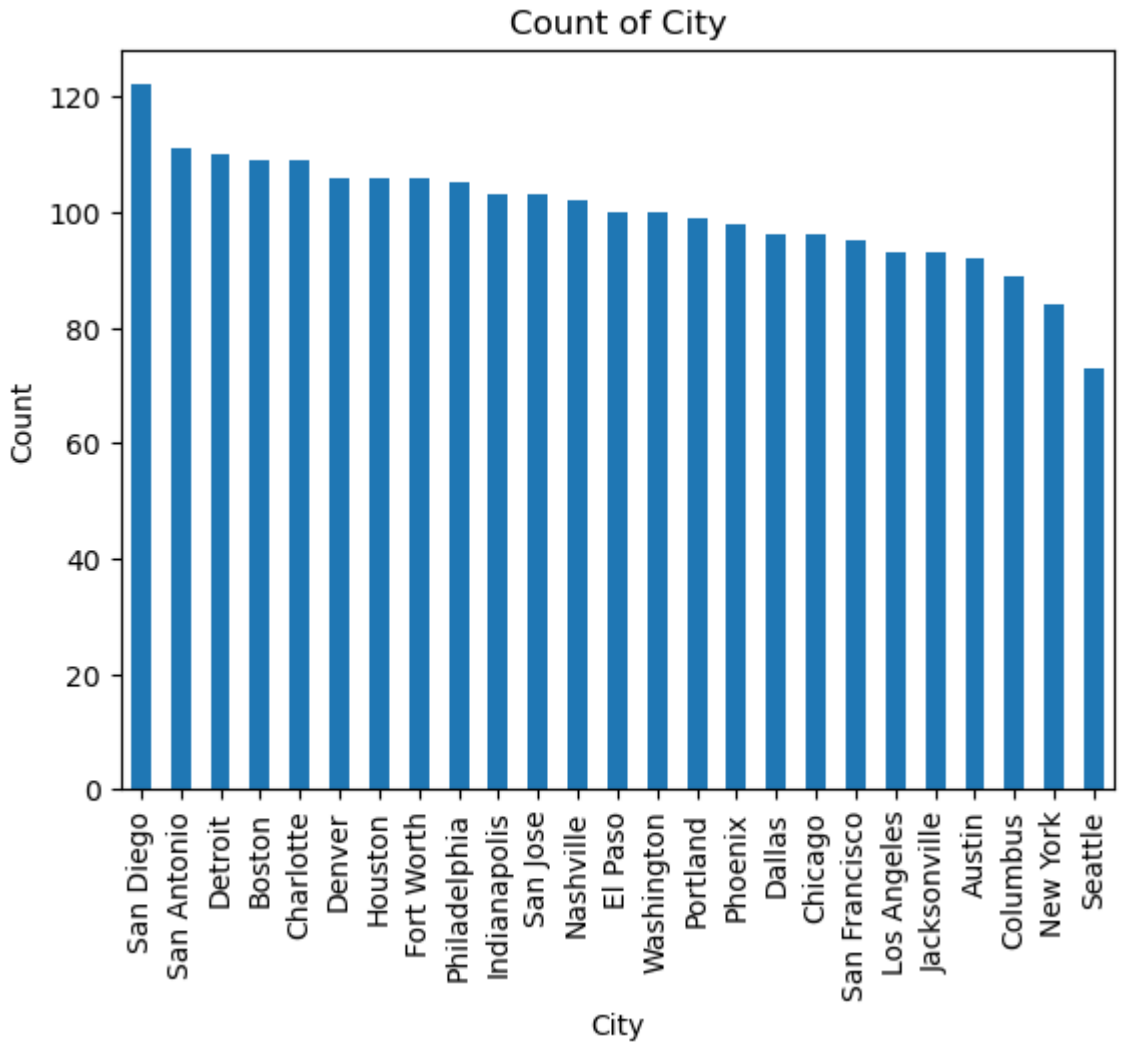
Visualizing Distributions and Skewness

For numeric features, distribution plots with KDE (Kernel Density Estimation) curves are generated. This visualizes how each numeric variable is spread and helps identify skewness or non-normal behaviour. The actual skewness values are then computed numerically for

each feature, with results indicating that most features are approximately symmetrical (skewness ≈ 0).

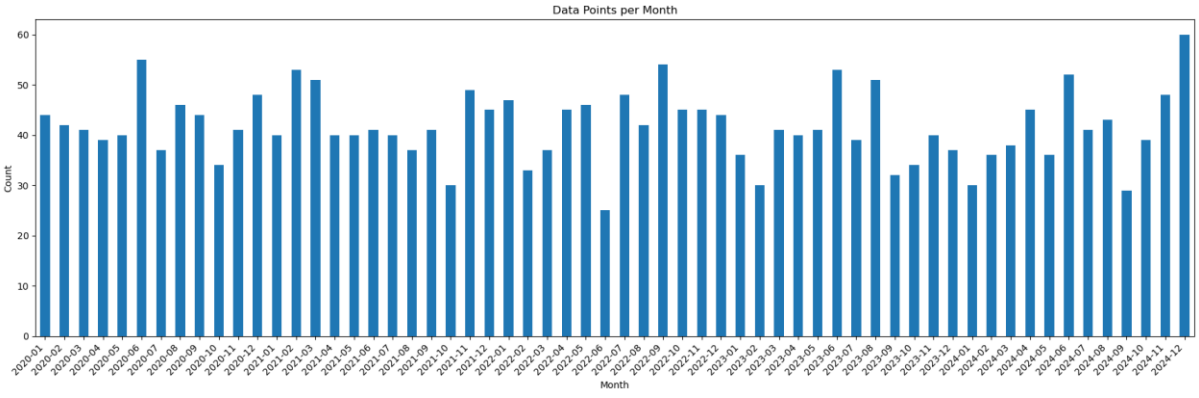


Categorical features are visualized using bar plots to show the frequency of each category. This helps understand class imbalance and the dominance of certain categorical values.



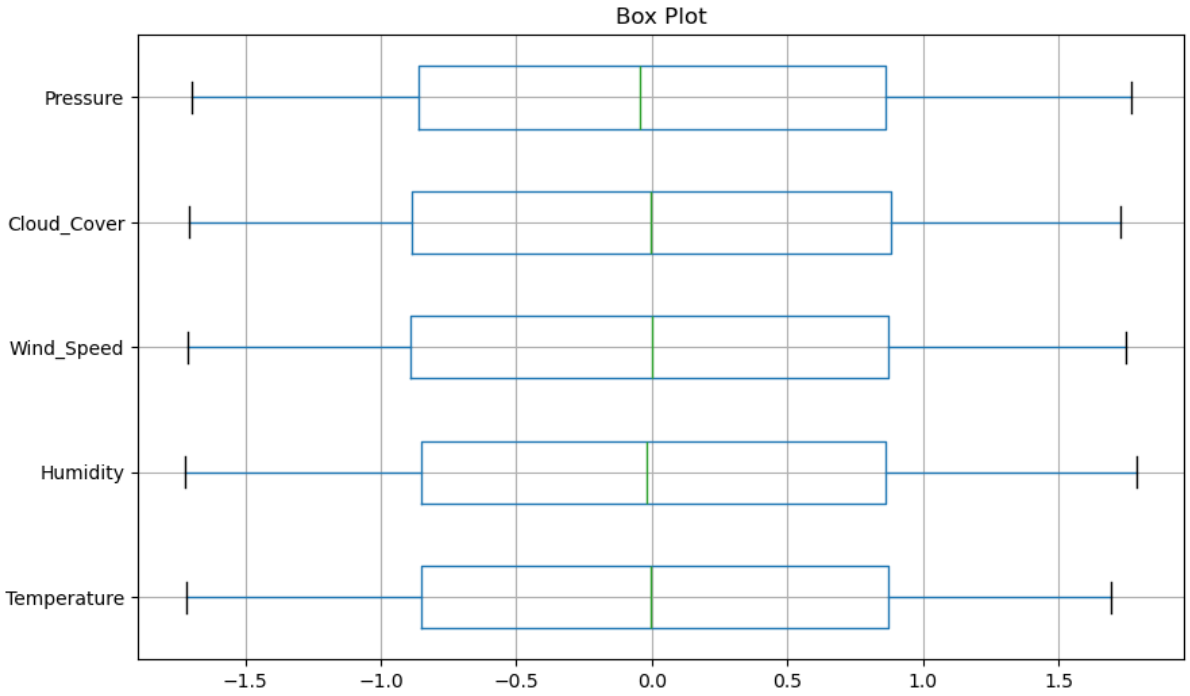
Time-Based Aggregation

The DateTime column is temporarily set as the DataFrame index to allow monthly resampling. A bar chart is plotted to visualize the number of records per month, identifying potential gaps or surges in data collection over time.



Outlier Detection and Scaling

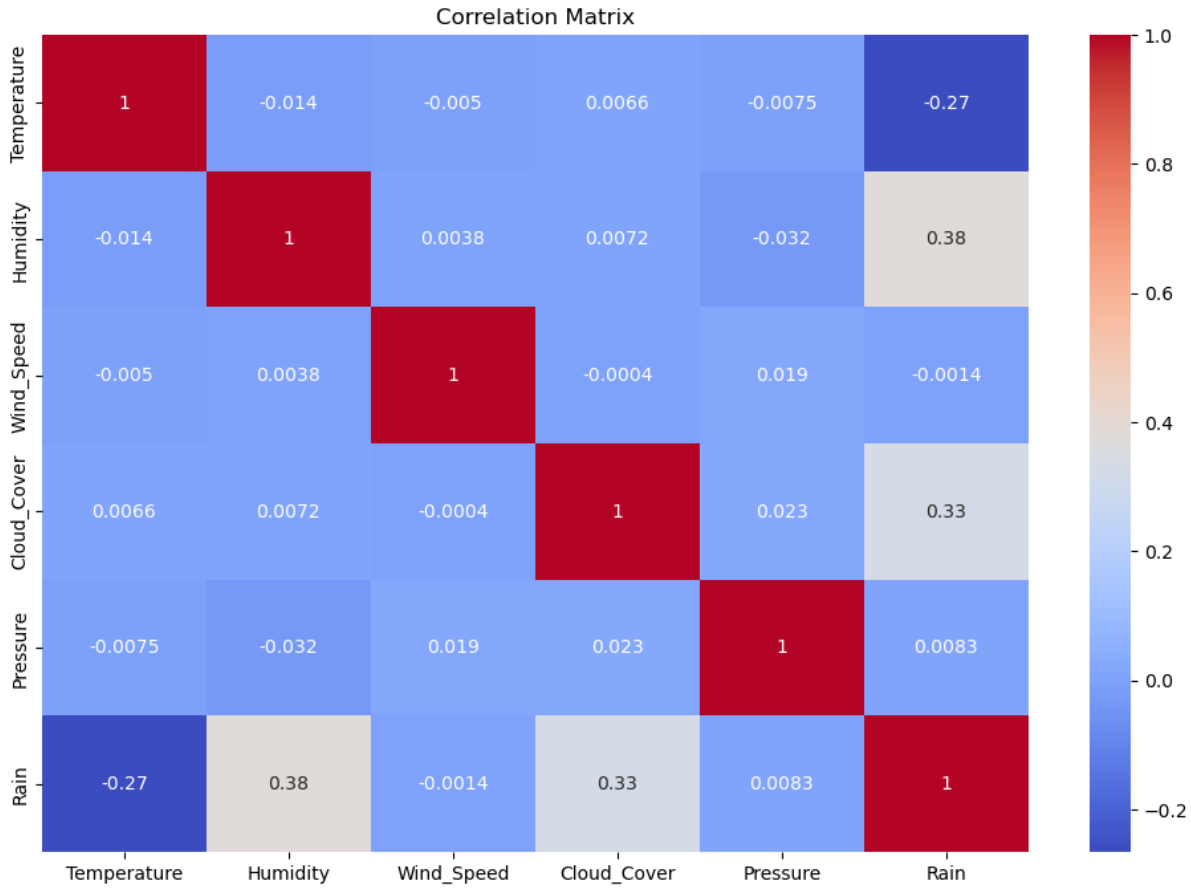
A boxplot is created for each numeric feature to visualize the range and detect outliers. After that, the data is standardized using StandardScaler, which transforms numeric features to have a mean of 0 and standard deviation of 1. A second boxplot is generated post-scaling to confirm that scaling has adjusted the spread but not removed outlier.



To remove the outliers, the Interquartile Range (IQR) method is applied to each numeric feature. This removes any value beyond 1.5 times the IQR from Q1 and Q3. The dataset's shape before and after this operation stays the same, indicating that no extreme outliers were present based on the IQR rule.

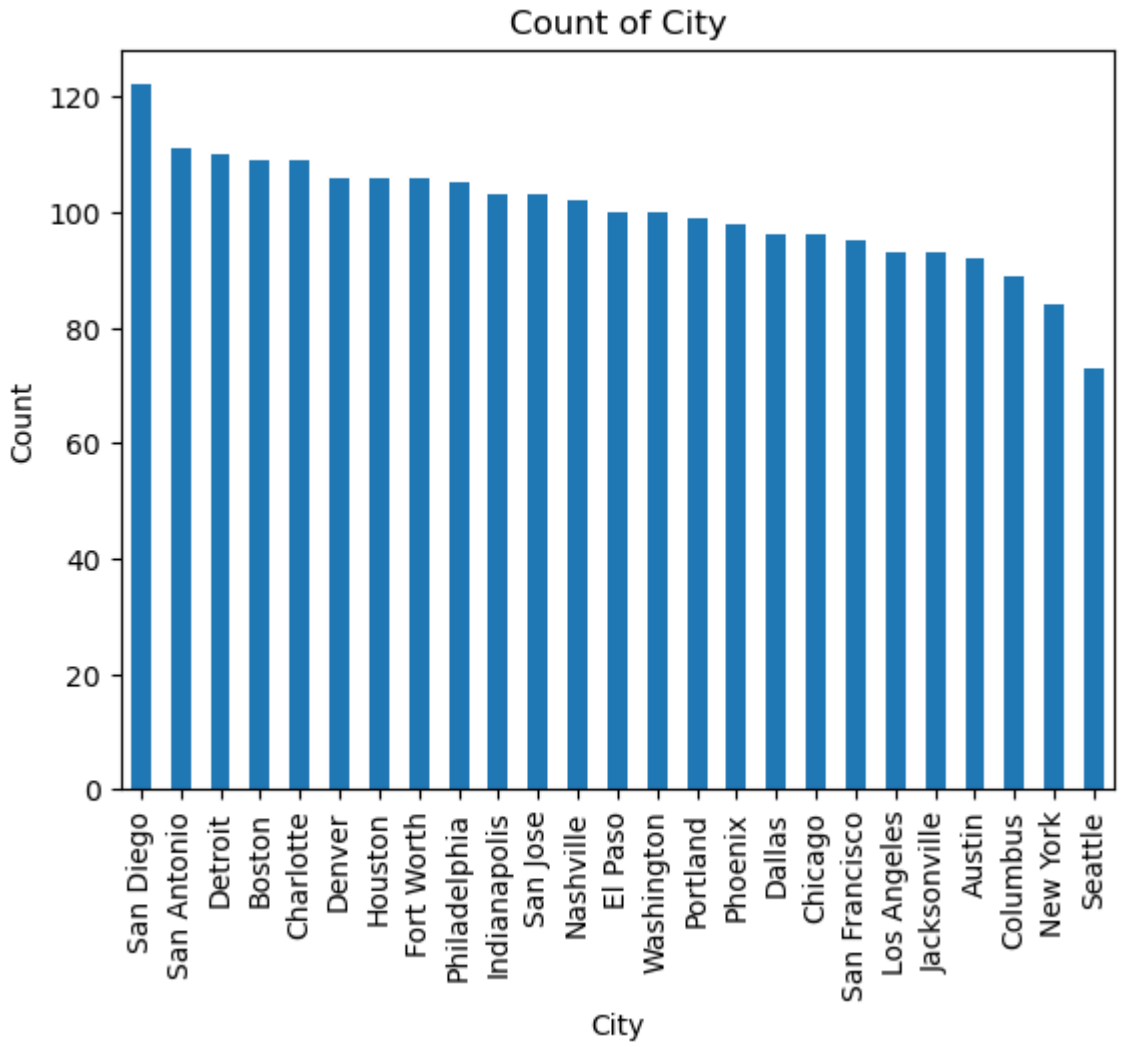
Correlation Analysis

A correlation matrix heatmap is plotted to show how numeric features relate to one another. This is useful for identifying multicollinearity and understanding which features might be predictive of the target variable.



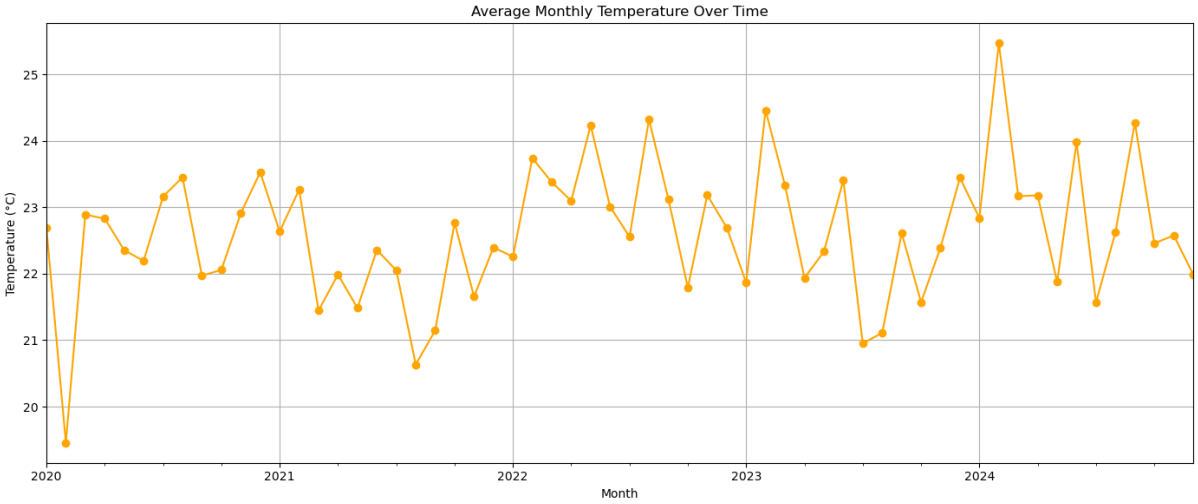
City-Level Aggregation

Finally, a bar plot displays the average temperature for each city, giving insights into geographic variation in the dataset, which might relate to weather patterns and the occurrence of rain.



Temperature Trend Analysis

To understand how temperature varies over time, the dataset was analysed on a monthly basis. The resulting chart, titled "Average Monthly Temperature Over Time", visually depicts temperature trends from 2020 to 2024. The plot reveals seasonal fluctuations and occasional spikes or dips, which could correlate with unusual weather events. For instance, there's a notable drop around early 2020 and a temperature peak around early 2024.



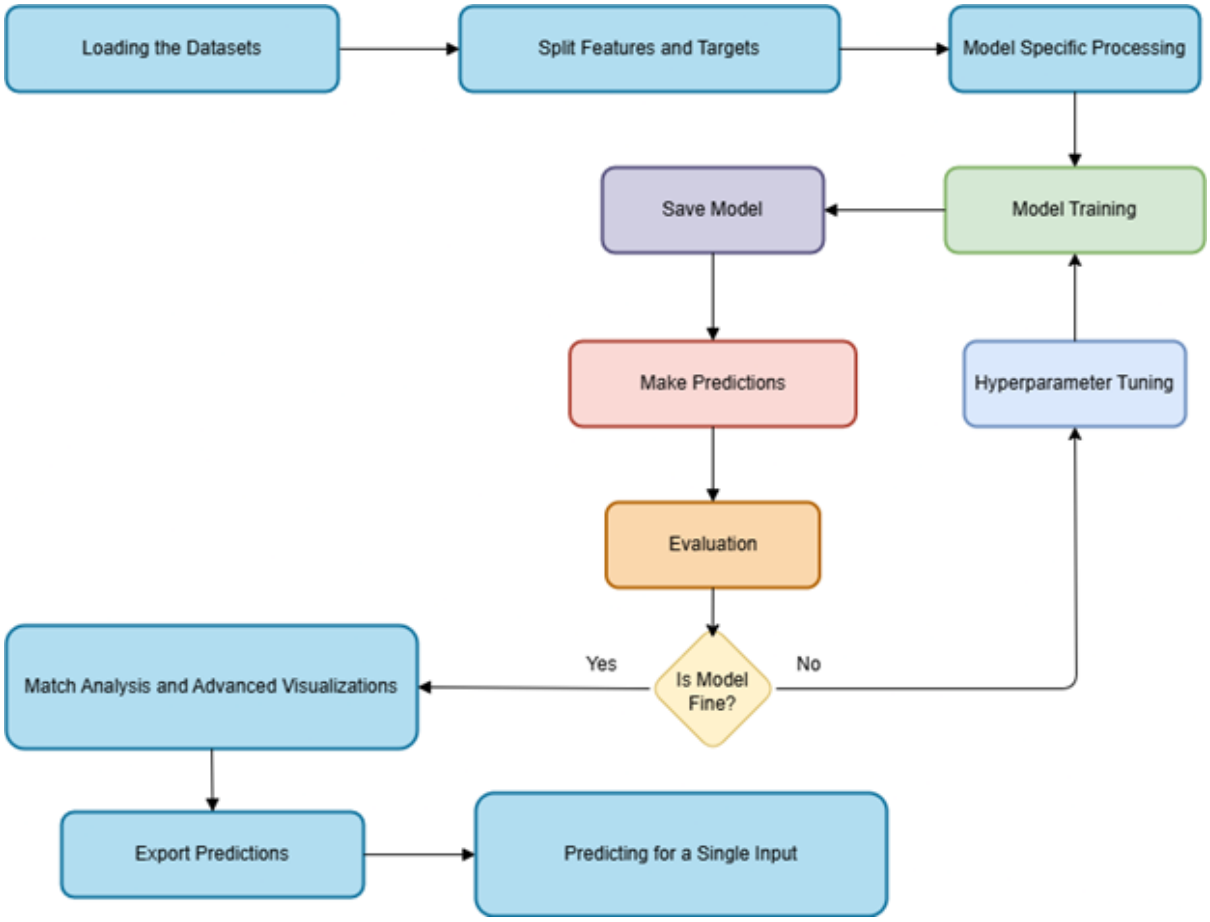
This visualization helps identify not just long-term trends but also short-term anomalies, which are crucial for weather-related predictions and temporal pattern analysis.

Through this combination of preprocessing, visualization, and statistical exploration, the dataset was prepared for effective machine learning model development.

Models

Logistic Regression, Random Forest, XGBoost

This is a basic visual overview of how each model executes:



Here are the detailed steps of how each model executes:

Logistic Regression

Step 1: Loading the Datasets

Two datasets are loaded using pandas.read_csv(), one for training and one for testing. These datasets contain weather conditions with the target variable “Rain”, which indicates whether it rained (1) or not (0). The datasets are structured with comma-separated values (delimiter=“,”).

Step 2: Split Features and Target

From both training and testing datasets, the features (X) are extracted by dropping columns like “City”, “DateTime”, and the target “Rain”. The target labels (y_train, y_test) are stored separately for model training and evaluation.

Step 3: Feature Scaling

A StandardScaler is used to standardize the numeric features, so they have zero mean and unit variance. This is critical for models like logistic regression to converge efficiently and perform consistently across features.

Step 4: Hyperparameter Tuning with GridSearchCV

GridSearchCV is applied to tune the hyperparameters of the LogisticRegression model. A grid of values is tested:

- C (inverse of regularization strength)
- max_iter (maximum number of iterations)
- penalty (l2)

The model is trained and cross-validated on the training set to find the best combination based on accuracy.

Step 5: Save the Trained Model and Scaler

After tuning, the best estimator (ln) is saved using pickle along with the fitted StandardScaler. This allows for future use of the model without retraining or rescaling.

Step 6: Make Predictions

Using the trained logistic regression model, predictions are made on the scaled testing features. These predictions represent the model's guess for “Rain” or “No Rain” on unseen data.

Step 7: Model Evaluation

Model performance is assessed using:

- **Accuracy Score**: Proportion of correct predictions.
- **Classification Report**: Shows precision, recall, and F1-score for each class.
- **Confusion Matrix**: Visualized using seaborn.heatmap to understand prediction breakdown.

This helps to see how well the model identifies each class (rain vs no rain).

Step 8: Match Analysis and Advanced Visualizations

Several visual tools are used to assess prediction accuracy:

- Match count bar plot: Shows correct vs incorrect predictions per class.
- Line plot: First 100 samples comparing actual and predicted labels.
- ROC Curve: Assesses classifier performance across thresholds.
- Precision-Recall Curve: Evaluates the balance between sensitivity and precision.

Step 9: Export Predictions

The full prediction results (actual vs predicted) are saved to a CSV file (regression_prediction.csv) for reporting and traceability.

Step 10: Predicting for a Single Weather Input

A function predict_weather() is defined to take a new weather input, as a comma-separated string, convert it into a DataFrame, preprocess it, and predict whether it will rain using the trained model. This allows for real-time or one-off predictions.

Random Forest

Step 1: Load the Datasets

The training and testing datasets are loaded using pandas.read_csv(). The training set (df_train) is used to train the Random Forest model, while the test set (df_test) is used to evaluate its performance. The data contains meteorological features and the target variable "Rain"

Step 2: Split features and Targets

The features (X_train, X_test) are created by removing irrelevant columns like "City", "DateTime", and "Rain" (the target) from the dataset. The target variable "Rain" is stored separately in y_train and y_test.

Step 3: Hyperparameter tuning with GridSearchCV

To optimize the performance of the Random Forest model, GridSearchCV is used to test multiple combinations of hyperparameters:

- n_estimators: number of trees in the forest
- max_depth: maximum depth of each tree
- min_samples_split: minimum number of samples required to split a node
- criterion: the function used to measure the quality of a split

Cross-validation is performed to select the best combination based on accuracy. Progress is displayed using verbose=1.

Step 4: Model Selection

The best-performing model found by GridSearchCV is selected using .best_estimator_. This model is then used for evaluation and predictions.

Step 5: Save the Trained Model

The trained Random Forest model is saved using pickle so it can be reused later without retraining. The model is stored in the Artifacts folder as a .pkl file.

Step 6: Make Predictions

The selected Random Forest model is used to make predictions on the test dataset (X_test). These predictions represent whether rain is expected (1) or not (0) for each data point in the test set.

Step 7: Model evaluation

The model is evaluated using multiple metrics:

- **Accuracy**: Proportion of correct predictions over total predictions.
- **Classification Report**: Displays precision, recall, F1-score for both classes (Rain, No Rain).
- **Confusion Matrix**: Visualized using a heatmap to show how many predictions were correct vs incorrect across both classes.

Step 8: Match Analysis and Advanced Visualizations

The actual vs predicted values are saved into a DataFrame and exported to a CSV file (randomforest_prediction.csv). Additional visualizations are included:

- **Match Count Bar Plot**: Highlights correct (green) vs incorrect (red) predictions per class.
- **Prediction vs Actual Line Plot**: Shows the first 100 predictions compared to the ground truth, useful for spotting patterns or drifts.
- **Feature Importance Plot**: Displays which features had the most influence on model predictions.

Step 9: ROC and Precision-Recall Curves

To better understand classifier performance:

- ROC Curve: Plots True Positive Rate vs False Positive Rate. The AUC (Area Under Curve) is used to summarize performance.
- Precision-Recall Curve: Shows the trade-off between precision and recall at different thresholds, especially useful for imbalanced datasets.

These curves provide insight into how well the model separates the two classes across thresholds.

Step 10: Predicting for a Single Weather Input

A function predict_weather() is defined to allow one-off predictions from raw input.

- It accepts a string in the format: "City, DateTime, Temperature, Humidity, Wind_Speed, Cloud_Cover, Pressure"
- Parses and preprocesses the values (excluding "City" and "DateTime")
- Uses the trained Random Forest model to return a predicted outcome (0 = No Rain, 1 = Rain)

This allows for flexible use in dashboards, forms, or automated systems.

XGBoost

Step 1: Load the Datasets

The training and testing datasets are loaded from CSV files using pandas.read_csv(). The training data is used to build the model, and the test data is used to evaluate its performance. The dataset includes weather-related features such as temperature, humidity, wind speed, and pressure, along with the target variable "Rain".

Step 2: Separate Features and Target

Unnecessary columns such as "City", "DateTime", and "Rain" are dropped from the input features. The "Rain" column is label-encoded using LabelEncoder() to convert its values into numerical format (e.g., "rain" = 1, "no rain" = 0), and stored separately in y_train and y_test.

Step 3: Build the pipeline

A Pipeline is created using scikit-learn to wrap the XGBoost classifier (XGBClassifier). This approach provides modularity and simplifies hyperparameter tuning and model management.

Step 4: Define Hyperparameter Grid

A grid of hyperparameters is defined to tune the XGBoost model. The parameters explored include:

- max_depth: Maximum tree depth
- learning_rate: Step size shrinkage
- n_estimators: Number of boosting rounds

This grid is passed to GridSearchCV for cross-validated search.

Step 5: Train the Model

The model is trained using GridSearchCV with 3-fold cross-validation. This systematically evaluates all combinations of hyperparameters and selects the one with the highest cross-validated accuracy. The best model is extracted using ".best_estimator_".

Step 6: Save the Trained Model

The selected and tuned XGBoost model is saved using pickle into a .pkl file. This ensures the model can be reused later for inference or deployment without retraining.

Step 7: Make Prediction

Using the best model obtained from the grid search, predictions are made on the test dataset (X_test). These predictions represent whether it will rain (1) or not (0) based on the input weather features.

Step 8: Model evaluation

Several metrics are used to assess the model's performance:

- **Accuracy**: The overall correctness of the model.
- **Classification Report**: Displays precision, recall, and F1-score for both "Rain" and "No Rain" classes.
- **Confusion Matrix**: A visual breakdown of true vs predicted values for each class, rendered using seaborn.heatmap().

Step 9: Save and Visualize Predictions

The predictions and actual values are merged into a DataFrame, allowing for side-by-side comparison. This includes:

- **Correct vs Incorrect Prediction Count Plot**: Bar chart highlighting where predictions matched the actual labels.
- **Prediction Timeline Plot**: A plot comparing actual vs predicted class labels for the first 100 test samples.
- **Feature Importance Plot**: A bar plot visualizing how much each feature contributed to the final model's decisions, extracted via feature_importances_.

The combined dataset is saved as a CSV file (xgboost_prediction.csv) for future review or external analysis.

Step 10: ROC and Precision-Recall Curves

For models that support probability prediction (predict_proba()), the following curves are generated:

- ROC Curve: Plots True Positive Rate (TPR) against False Positive Rate (FPR), with AUC (Area Under the Curve) as a summary metric of model performance.
- Precision-Recall Curve: Useful for imbalanced datasets, this plot shows the trade-off between precision and recall across classification thresholds.

These plots give deeper insight into how well the model separates “Rain” and “No Rain” cases.

Step 11: Predicting for a Single Weather Input

A function predict_weather() is provided for making individual predictions. It:

- Accepts a comma-separated input string including weather details (e.g., “City, DateTime, Temperature, Humidity, Wind_Speed, Cloud_Cover, Pressure”)
- Strips out the non-predictive fields (“City”, “DateTime”)
- Converts the remaining values into a format accepted by the model
- Returns a prediction (“rain” or “no rain”) by reversing the label encoding

This function supports real-time usage in web forms, scripts, or interactive applications.

Evaluation Results:

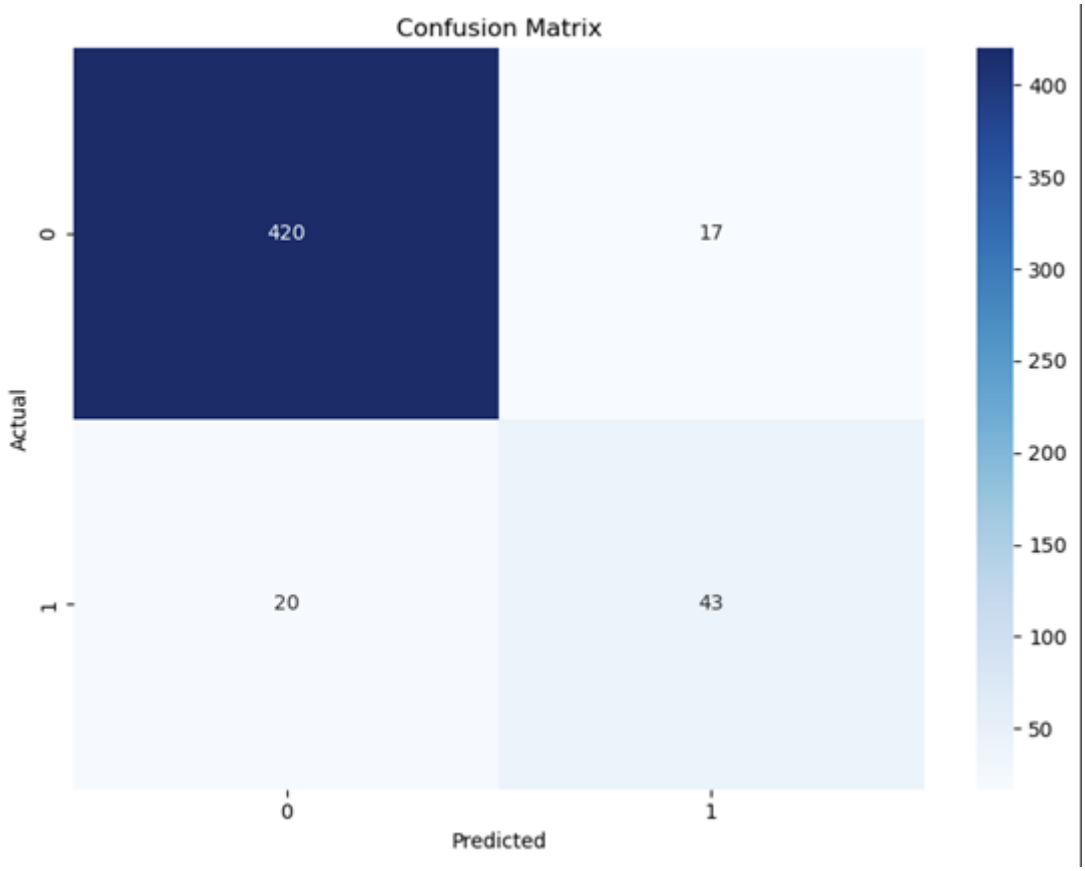
Logistic Regression

```
Accuracy: 0.926

Classification Report:
              precision    recall  f1-score   support

     No Rain       0.95      0.96      0.96       437
        Rain       0.72      0.68      0.70        63

    accuracy                           0.93       500
   macro avg       0.84      0.82      0.83       500
weighted avg       0.92      0.93      0.93       500
```



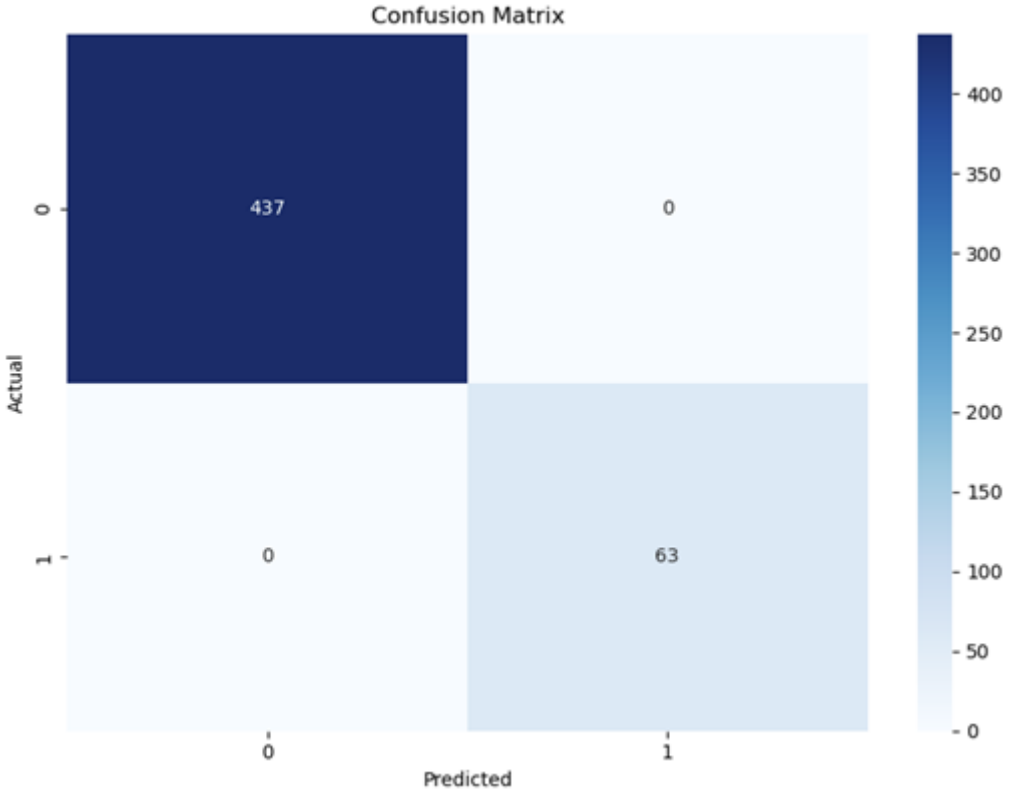
Random Forest

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

     No Rain       1.00      1.00      1.00       437
        Rain       1.00      1.00      1.00        63

    accuracy                           1.00       500
   macro avg       1.00      1.00      1.00       500
weighted avg       1.00      1.00      1.00       500
```



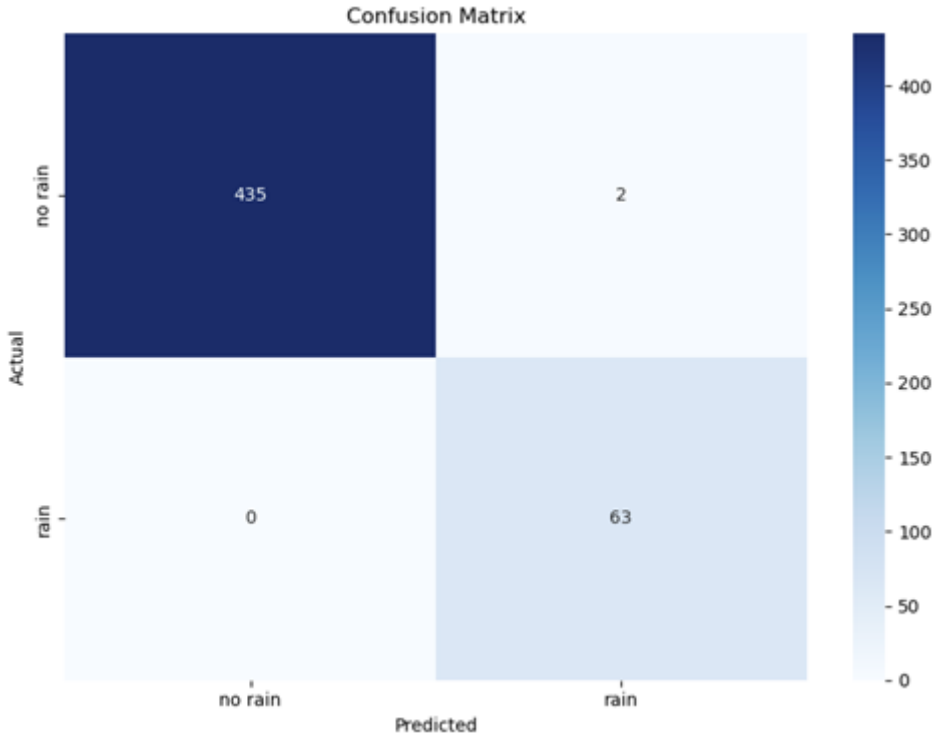
XGBoost

```
XGBoost Accuracy: 0.996
Classification Report:

Classification Report:
              precision    recall  f1-score   support

     No Rain       1.00      1.00      1.00       437
        Rain       0.97      1.00      0.98        63

    accuracy                           1.00       500
   macro avg       0.98      1.00      0.99       500
weighted avg       1.00      1.00      1.00       500
```



Expected vs. Obtained Results

During model evaluation, three classification models were trained and tested to predict rainfall: Logistic Regression, Random Forest, and XGBoost. Each model was evaluated based on accuracy, precision, recall, F1-score, and its behaviour on specific samples. The results revealed several important insights:

Logistic Regression

- **Accuracy:** 92.6%
- **Precision (Rain):** 0.72
- **Recall (Rain):** 0.68

Observation:
While Logistic Regression achieved respectable accuracy, it struggled to consistently identify the minority class (Rain). With a lower recall and F1-score for Rain, it often under-predicted rain events, which could lead to missed detections in real-world applications.

Random Forest

- **Accuracy:** 100%
- **Precision / Recall / F1:** All metrics were reported as 1.00

Observation:
Random Forest delivered perfect classification performance in terms of metrics on the test data. However, despite its apparent overfitting, it produced **the most consistent predictions** on manually tested edge cases and custom weather samples. These predictions aligned closely with domain expectations, suggesting strong generalization in practical settings.

XGBoost

- **Accuracy:** 99.6%
- **Precision (Rain):** 0.97
- **Recall (Rain):** 1.00

Observation:
XGBoost was highly accurate overall and particularly strong at identifying rain cases. However, in rare cases, its predictions tended to fluctuate slightly, sometimes predicting Rain when the expected result was No Rain, especially when inputs fell near the class decision boundary.

Overall:

Random Forest was the best model in terms of general metrics like macro-averaged F1-score, while being the most robust model in terms of consistency as well. When tested with specific inputs and domain-based expectations, its predictions were the most aligned with real-world observations. Conversely, Logistic Regression and XGBoost, despite their strong metrics, occasionally, but very little, produced misclassifications, typically by one level in class distance (false negatives or false positives).

Thus, for the goal of reliable, practical weather prediction in this project, Random Forest is favoured as the deployment model, balancing performance and interpretability.

An example for this was with the following inputs across all 3 models:

Example Input:

”Chicago,2023-12-15 01:26,27.31386567157817,94.69920342298845,13.537423774917874,56.54416888696837,1042.2283814225484”

Expected Outcome: No Rain

Predicted Results:

Model	Weather Prediction
Logistic Regression	No Rain
Random Forest	No Rain
XGBoost	No Rain

This consistency across all three models, Logistic Regression, Random Forest, and XGBoost, demonstrates not only high accuracy but also strong alignment with domain expectations. It reinforces the models’ reliability for real-world forecasting, especially when predictions are validated against known outcomes. Such agreement is crucial in building trust for practical deployment.

Dash application development

To provide a real-time weather forecast prediction application a web application was build using the Dash framework. Dash is a Python-based framework which is ideal for building applications without the need for JavaScript. This web application serves as a solution for users to input their weather conditions and receive predictions from our machine learning models.

Application structure:

- Framework and styling:
  Our web application utilises dash and dash bootstrap component libraries for layout and responsiveness. The bootstrap theme alongside a custom CSS file contributed towards a more accessible and aesthetically pleasing user interface.

- Integration of models and scalers:
  The pre-trained models and scalers which were mentioned in the previous section of this document are stored in a designed directory. These models and scalers include, the xgboost model, the randomforest model, the regression model and the regression scaler. A function was created in the web application to load these models and scalers.

- UI:
  The applications layout is a form-based application where the users enter information about the temperature in degrees Celsius, the humidity in percent, the wind speed in kilometres per hour, the cloud coverage in percent, and the pressure in hPa.
  Thereafter, if all entries are valid the user clicks on the predict button which triggers the model evaluation process.

- Input validation: A dash callback was implemented to validate the user's inputs before the prediction, this validation includes:
  - No empty fields
  - Numerical values
  - Temperature between -50 and 60 C
  - Humidity between 0 and 100%
  - Wind Speed between 0 and 200 km
  - Cloud Cover between 0 and 100%
  - Pressure between 800 and 1100 hPa.

- Output:
  The model outputs are displayed in a user friendly list which allows the users to compare the predictions across the three models, namely XGBoost, RandomForest, and Logistic Regression.

- Prediction logic:
  Upon the successful validation the app constructs a dataframe from the inputs and applies scaling for the regression model. Thereafter the application uses each model to predict the rainfall and formats the predictions as either rain or no rain.

- Deployment: The application runs locally on machines using app.run and has been deployed to Render – which is a cloud platform for broader accessibility

Reflection and Future Work

Reflection

This project provided a valuable hands-on opportunity to explore multiple supervised machine learning techniques for weather classification using real-world data. Throughout the process, we faced and resolved various technical challenges, such as data imbalances, library compatibility issues, and model tuning difficulties. These experiences deepened our understanding of preprocessing pipelines, model evaluation, and deployment-readiness criteria.

Each model had its strengths:

- **Logistic Regression** proved to be efficient and interpretable but required careful handling of scaling and class imbalance.
- **Random Forest** offered strong robustness and high consistency in predictions, making it a suitable candidate for deployment in practical scenarios.
- **XGBoost** demonstrated excellent performance after careful hyperparameter tuning, particularly on more complex patterns in the data.

While the models achieved high overall accuracy, especially on well-balanced test sets, we observed that even small errors, such as false negatives in rain prediction, could carry significant consequences in real-world applications. This insight highlighted the importance of not just accuracy, but model reliability and interpretability in sensitive domains.

Future Work

Several enhancements can be considered to improve this project in future iterations:

Deeper Feature Engineering

- **Seasonality and Calendar-based Features**
  Integrating richer time-based attributes, such as *season*, *is_weekend*, *public holiday*, or *school vacation*, can significantly improve prediction accuracy. For instance, temperature or humidity distributions can vary substantially between seasons, and certain weather patterns (like summer storms) may be tied to specific periods of the year.
- **Geospatial Encoding**
  Currently, cities are treated as categorical variables. In future iterations, using geospatial coordinates (latitude and longitude) can allow the model to understand regional weather patterns and gradients. Advanced geospatial clustering or distance-based encodings may also improve predictions for locations not directly present in the training dataset.

Model Expansion

- **Deep Learning Architectures**
  Long Short-Term Memory (LSTM) networks are ideal for time-series forecasting where sequential patterns matter, such as detecting temperature trends over days.

Similarly, Convolutional Neural Networks (CNNs) can be applied for spatial or gridded data, especially when integrating radar or satellite images.

- **Ensemble Learning**
  Future models can combine multiple algorithms, to reduce overfitting and improve generalization. Blending different models' strengths can often outperform any single method.

Uncertainty Estimation

- **Confidence Intervals and Probabilistic Outputs**
  Rather than predicting a binary outcome (rain/no rain), models can be adapted to provide probabilities of each outcome. This can inform better decision-making under uncertainty, such as whether to cancel an event due to a 60% chance of rain.
- **Bayesian Approaches or Monte Carlo Dropout**
  These techniques allow for estimating prediction uncertainty, especially in deep learning models, and are useful in critical applications like disaster planning or flight scheduling.

Real-Time Web Integration

- **Dynamic Data Ingestion from Weather APIs**
  Replacing static CSV inputs with real-time data fetched from sources like OpenWeatherMap or NOAA APIs can make the system adaptable to real-world conditions.
- **Live Dashboard Deployment**
  A web interface showing predictions, trends, and alerts in real-time (possibly built with Dash or Streamlit) would make the tool accessible to non-technical users like farmers, municipal workers, or event planners.

Explainability and Fairness

- **Explainable AI (XAI) Techniques**
  Integrating SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations) can provide insight into how features influence predictions. This is essential in building trust, especially for public-facing systems.
- **Bias and Fairness Analysis**
  It's important to ensure that the model does not unfairly favour certain cities, time zones, or weather conditions due to imbalance in training data. Future work should include fairness audits and bias mitigation strategies to ensure ethical AI use.

Expanded Dataset

- **More Locations and Longer Timespans**
  Including data from a wider range of geographic regions and across more years would help the model generalize better, especially in rare weather conditions (e.g., snowstorms, heatwaves).
- **Integrating External Data Sources**
  Incorporating satellite imagery, radar scans, air pressure maps, or even historical precipitation logs can enrich the dataset. These additional signals could lead to significant improvements in rain prediction and temperature modelling.

By addressing these areas, the system can evolve from a high-performing academic prototype into a production-grade weather prediction tool that is reliable, interpretable, and suitable for real-world deployment in domains such as agriculture, logistics, and disaster management