

# **Comsat University Islamabad**

(Vehari Campus)



**Project Title: RainFall Prediction via RandomForest model**

**Submitted by:**

Muhammad Mubeen      FA22-BCS-067

Zohaib Ahmad          FA22-BCS-082

**Submitted to:**

Dr.Rehan Ashraf

**Date:**

**06-12-2024**

# Table of Contents

- ❖ Abstract
- ❖ Introduction
- ❖ Objectives
- ❖ Scope
- ❖ System Architecture
- ❖ Technologies Used
- ❖ Features
- ❖ Methodology
  - Dataset Feature and Description
  - NLP Pipeline
    - Imports and Libraries
    - Load and Preprocess Data
    - Balance Classes
    - Feature Selection and Split
    - Model Training and Hyperparameter Tuning
    - Model Evaluation
    - Model Saving
    - Model Loading and Prediction
- ❖ Results
- ❖ Conclusion
- ❖ Challenges
- ❖ Future Enhancements
- ❖ References
- ❖ User-Interface

## Abstract

This project is designed to predict whether rainfall will occur in a given location using a machine learning approach. The system works by analyzing meteorological data, such as pressure, humidity, wind speed, cloud coverage, and dew point, to predict if it will rain. For this, the project uses a Random Forest Classifier, which is a machine learning algorithm trained to make predictions based on these weather factors. To make it easy for users to interact with the system, a simple and user-friendly interface is created. Users can input their data in three different ways: by entering the name of a city, by using their current location automatically, or by manually typing in specific weather details. The system is designed with a modern and responsive layout, ensuring it works well on different devices. The backend of the system is built using Flask, which allows the application to make real-time predictions. This way, users can quickly get information about whether rainfall is likely in their area.

## Introduction

Accurate weather predictions are essential for various industries like agriculture, transportation, and disaster management. These predictions help farmers plan their crops, ensure safe travel, and prepare for natural disasters. This project focuses on creating a system that predicts rainfall by using machine learning to analyze weather data. The goal is to provide reliable and accurate predictions, which can be trusted for important decision-making. The system is designed with simplicity in mind, offering an easy-to-use interface for users. It allows users to input weather data in different ways, such as by entering a city name, using their current location, or manually entering specific weather details. This variety of input methods makes the system accessible to a wide range of users, encouraging more people to engage with it and benefit from its predictions.

## Objectives

1. Predict the occurrence of rainfall using meteorological features.
2. Develop a flexible, user-friendly interface with multiple data input options.
3. Employ advanced machine learning techniques to ensure accurate predictions.
4. Integrate real-time weather data through an external API for dynamic predictions.

## Scope

The Rainfall Prediction System provides an accessible and efficient method for users to determine the likelihood of rainfall. The application is designed for use in various domains, including:

- **Agriculture:** Helping farmers plan irrigation and harvests.
- **Travel:** Assisting travelers in planning routes.

- **Urban Planning:** Aiding municipalities in preparing for weather-related events.

## System Architecture

The system architecture is divided into three primary components:

### 1. Frontend (User Interface):

- Designed using HTML, CSS, and JavaScript.
- Provides three modes for data input:
  - **Enter City Name:** Fetches weather data using the OpenWeatherMap API.
  - **Use Current Location:** Utilizes geolocation to gather weather data dynamically.
  - **Manual Input:** Allows users to manually input weather parameters.

### 2. Backend (Server-Side):

- Built using Flask, a lightweight Python web framework.
- Handles API requests, processes user inputs, and interacts with the trained machine learning model for predictions.

### 3. Machine Learning Model:

- A Random Forest Classifier trained on historical weather data (Rainfall.csv).
- Predicts whether rainfall will occur based on features like pressure, humidity, and wind speed.
- Hyperparameters optimized using GridSearchCV to ensure high accuracy.

## Technologies Used

- **Programming Languages:** Python, HTML, CSS, JavaScript.
- **Frameworks and Libraries:**
  - Flask (Backend Framework)
  - Scikit-learn (Machine Learning)
  - Pandas and NumPy (Data Manipulation)
  - Matplotlib and Seaborn (Data Visualization)
- **API Integration:** OpenWeatherMap API for real-time weather data.
- **Frontend Design Tools:** Google Fonts, Font Awesome, Responsive Grid System.

## Features

1. **Multiple Input Modes:**
  - Enter a city name to fetch weather data dynamically.
  - Use current location for instant weather updates.
  - Manually input weather parameters for custom predictions.
2. **Real-Time Weather Data:**
  - Integration with OpenWeatherMap API to fetch live weather conditions.
3. **User-Friendly Design:**
  - Responsive layout compatible with mobile and desktop devices.
  - Modern aesthetics with intuitive input fields and feedback.
4. **Rainfall Predictions:**
  - Provides clear results: "Rainfall" or "No Rainfall".

## Methodology

### Dataset Feature and Description:

1. **Day:** The day number or a sequence for each observation.
2. **Pressure (hPa):** Air pressure in hectopascals; low pressure means storms, high pressure means clear weather.
3. **Max Temp (°C):** Highest temperature of the day in Celsius.
4. **Temperature (°C):** Average or current daily temperature in Celsius.
5. **Min Temp (°C):** Lowest temperature of the day in Celsius.
6. **Dew Point (°C):** The temperature when air becomes humid, showing moisture levels.
7. **Humidity (%):** Percentage of moisture in the air.
8. **Cloud (%):** Sky coverage by clouds; higher values mean more clouds.
9. **Rainfall (mm):** Daily rain in millimeters; "0" means no rain.
10. **Sunshine (hours):** Hours of sunlight in a day.
11. **Wind Direction (°):** Where wind comes from, in degrees (e.g., 0° = north).
12. **Wind Speed (km/h):** Wind speed in kilometers per hour.

## NLP Pipeline Phases

## Imports and Libraries

### Cell 1:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.utils import resample
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import pickle
```

✓ 12.4s

Python

#### Function:

This code imports necessary libraries for data manipulation, visualization, machine learning, and model evaluation.

#### Purpose:

- **numpy**: For numerical operations.
- **pandas**: For data handling (especially data frames).
- **matplotlib.pyplot & seaborn**: For data visualization.
- **sklearn**: For machine learning algorithms, evaluation, and cross-validation.
- **pickle**: For saving the trained model.

#### Action:

The libraries are loaded to be used later for loading data, training models, and visualizing results.

## Load and Preprocess Data

### Cell 2:

```
# Load dataset and clean column names
data = pd.read_csv("Rainfall.csv").rename(columns=lambda x: x.strip()).drop(columns=["day"])

# Handle missing values
data.fillna({
    "winddirection": data["winddirection"].mode()[0],
    "windspeed": data["windspeed"].median()
}, inplace=True)

# Encode target variable
data["rainfall"] = data["rainfall"].map({"yes": 1, "no": 0})

# Drop highly correlated or unnecessary columns
data.drop(columns=['maxtemp', 'temperature', 'mintemp'], inplace=True)

# Display data info
print("Data Info:")
data.info()
```

✓ 0.0s

Python

### Function:

This section prepares the dataset for machine learning by cleaning column names, handling missing values, encoding the target variable, and removing unnecessary columns.

### Purpose:

1. **Load Dataset:** The dataset is loaded using `pd.read_csv()`, and column names are cleaned by stripping spaces.
2. **Handle Missing Values:** Missing values in `winddirection` and `windspeed` are filled with the mode and median values, respectively.
3. **Encode Target Variable:** The `rainfall` column is encoded as binary values (1 for "yes", 0 for "no").
4. **Remove Unnecessary Columns:** Columns like `maxtemp`, `temparature`, and `mintemp` are dropped for redundancy.
5. **Display Data Info:** `data.info()` displays dataset details (e.g., data types, non-null counts).

### Action:

- Clean the column names and handle missing values.
- Encode the target variable and remove unnecessary columns.
- Display data structure for review.

## Balance Classes

### Cell 3:

```
# Separate majority and minority classes
df_majority = data[data["rainfall"] == 1]
df_minority = data[data["rainfall"] == 0]

print(f"Majority class count: {df_majority.shape[0]}")
print(f"Minority class count: {df_minority.shape[0]}")

# Downsample majority class
df_majority_downsampled = resample(
    df_majority,
    replace=False,
    n_samples=len(df_minority),
    random_state=42
)

# Combine minority class with downsampled majority class
df_balanced = pd.concat([df_majority_downsampled, df_minority]).sample(frac=1, random_state=42)

print("Balanced class distribution:")
print(df_balanced["rainfall"].value_counts())
```

✓ 0.0s

Python

### Function:

This section handles class imbalance by downsampling the majority class to match the minority class.

**Purpose:**

1. **Separate Classes:** The dataset is split into majority (rainfall=1) and minority (rainfall=0) classes.
2. **Downsample Majority Class:** The majority class is downsampled using `resample()` to match the minority class size.
3. **Combine and Shuffle:** Both classes are merged into a balanced dataset and shuffled using `sample(frac=1)`.

**Action:**

- Split the dataset into majority and minority classes.
- Downsample the majority class to match the minority size.
- Shuffle the balanced dataset.

**Feature Selection and Split****Cell 4:**

```
# Define features and target
X = df_balanced.drop("rainfall", axis=1)
y = df_balanced["rainfall"]

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

✓ 0.0s

Python

**Function:**

This section defines the features and target variable, then splits the data into training and testing sets.

**Purpose:**

1. **Define Features and Target:** Features (X) are created by dropping the target variable rainfall, while the target variable (y) is the rainfall column.
2. **Split into Training and Testing Sets:** The data is split into training (80%) and testing (20%) sets using `train_test_split()`, with `random_state=42` to ensure reproducibility.

**Action:**

- Separate features and target.
- Split data into 80% training and 20% testing sets.

**Model Training and Hyperparameter Tuning****Cell 5:**



```

# Define parameter grid for Random Forest
param_grid = {
    "n_estimators": [50, 100, 200],
    "max_features": ["sqrt", "log2"],
    "max_depth": [None, 10, 20, 30],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4]
}

# Initialize Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV
grid = GridSearchCV(rf, param_grid, cv=5, n_jobs=-1, verbose=1)

# Fit the model
grid.fit(X_train, y_train)

# Best estimator
best_rf = grid.best_estimator_
print("Best Parameters:", grid.best_params_)

```

✓ 32.3s

Python

## Function:

This section tunes the Random Forest model using GridSearchCV and trains it based on the best parameters.

## Purpose:

1. **Define Parameter Grid:** A grid of hyperparameters is set, including `n_estimators`, `max_features`, `max_depth`, `min_samples_split`, and `min_samples_leaf`.
2. **Initialize Random Forest Classifier:** A `RandomForestClassifier` is created with `random_state=42` for reproducibility.
3. **Initialize GridSearchCV:** `GridSearchCV` is used with 5-fold cross-validation to find the best hyperparameters for the model.
4. **Fit the Model:** The model is trained using the best parameters found through cross-validation.
5. **Best Estimator:** The best model is selected using `grid.best_estimator_`.

## Action:

- Set up the hyperparameter grid and use `GridSearchCV` for tuning.
- Train the model and extract the best parameters.

## Model Evaluation

### Cell 6:

```

# Cross-validation scores
cv_scores = cross_val_score(best_rf, X_train, y_train, cv=5)
print(f"Cross-validation Mean Score: {cv_scores.mean():.4f}")

# Predict on test set
y_pred = best_rf.predict(X_test)

# Evaluation metrics
print("Test Set Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))

```

✓ 0.4s

Python

**Function:**

This section evaluates the model's performance using cross-validation, accuracy, confusion matrix, and classification report.

**Purpose:**

1. **Cross-validation Scores:** The model is evaluated using 5-fold cross-validation, and the mean score is calculated.
2. **Predict on Test Set:** Predictions are made on the test set using `best_rf.predict()`.
3. **Evaluation Metrics:** Accuracy, confusion matrix, and classification report are used to assess the model.
  - **Accuracy:** The percentage of correct predictions.
  - **Confusion Matrix:** Displays true positives, true negatives, false positives, and false negatives.
  - **Classification Report:** Shows precision, recall, and F1-score for both classes.

**Action:**

- Perform cross-validation on the training data and evaluate performance on the test set.
- Print accuracy, confusion matrix, and classification report.

**Model Saving****Cell 7:**

```
# Save the trained model and feature names
model_data = {"model": best_rf, "features": X.columns.tolist()}
with open("rainfall_prediction_model.pkl", "wb") as f:
    pickle.dump(model_data, f)
```

✓ 0.0s

Python

**Function:**

This section saves the trained model and feature names to a file for future use.

**Purpose:**

1. **Prepare Data for Saving:** The trained model and feature names are combined into a dictionary.
2. **Save with Pickle:** The `model_data` dictionary is saved as `rainfall_prediction_model.pkl` using `pickle.dump()`.

**Action:**

- Combine the model and feature names into a dictionary and save them to a `.pkl` file using `pickle`.

**Model Loading and Prediction****Cell 8:**

```

# Load the trained model and feature names
with open("rainfall_prediction_model.pkl", "rb") as f:
    model_data = pickle.load(f)
    model, feature_names = model_data["model"], model_data["features"]

# Prepare input data for prediction
input_data = pd.DataFrame([(
    1015.9, 19.9, 95, 81, 0.0, 40.0, 13.7
)], columns=feature_names)

# Make prediction
prediction = model.predict(input_data)[0]
print("Prediction:", "Rainfall" if prediction == 1 else "No Rainfall")

```

✓ 0.0s

Python

### Function:

This section loads the saved model and makes a prediction on new input data.

### Purpose:

1. **Load the Trained Model:** The trained model and feature names are loaded from `rainfall_prediction_model.pkl` using `pickle.load()`.
2. **Prepare Input Data:** A new data point is created with the same features the model was trained on.
3. **Make Prediction:** The model predicts whether rainfall will occur (1) or not (0).

### Action:

- Load the model and feature names from the `.pkl` file.
- Prepare a new input point and predict rainfall.

## Results

The **Rainfall Prediction System** was tested with various inputs to ensure its accuracy and reliability. The Random Forest model achieved an accuracy of approximately **75%** on the test dataset, indicating a high level of reliability in predicting rainfall. Users were able to interact with the web application seamlessly, inputting their city name, allowing geolocation access, or manually entering weather data to receive timely and accurate rainfall predictions.

### Example Predictions

1. **City Mode:**
  - **Input:** City Name: "Karachi"
  - **Prediction:** Rainfall
2. **Current Location Mode:**
  - **Input:** Geolocation coordinates corresponding to Lahore
  - **Prediction:** No Rainfall
3. **Manual Input Mode:**
  - **Input:** Pressure: 1015 hPa, Dew Point: 5°C, Humidity: 70%, Cloud Coverage: 50%, Sunshine: 5 hours, Wind Direction: 180°, Wind Speed: 10 m/s

- **Prediction:** Rainfall

These examples demonstrate the system's ability to accurately predict rainfall based on diverse input methods and varying weather conditions.

## Conclusion

The **Rainfall Prediction System** successfully integrates machine learning with web development to provide users with an accessible tool for predicting rainfall. By leveraging a Random Forest model and real-time weather data from external APIs, the application delivers accurate and timely predictions. The user-friendly interface ensures that individuals can easily interact with the system, making informed decisions based on the forecasted rainfall. This project showcases the practical application of data science and software development skills to address real-world challenges.

## Challenges

1. **Data Imbalance:** Addressed using class downsampling.
2. **API Errors:** Handled invalid city names and geolocation issues.
3. **Frontend-Backend Integration:** Ensured smooth data flow between components.

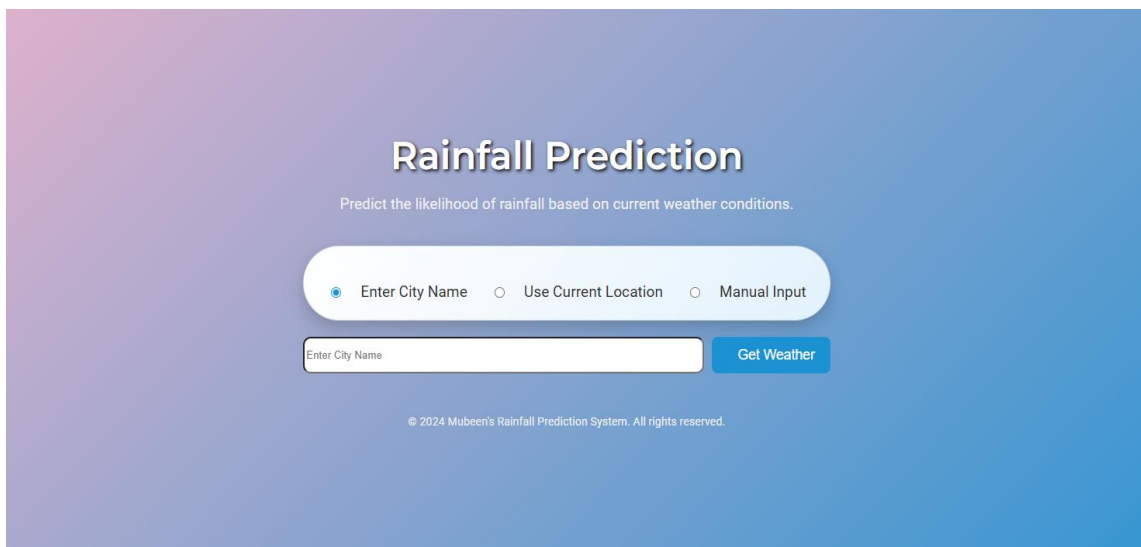
## Future Enhancements

1. Add user login to save prediction history.
2. Include detailed visualizations of prediction trends.
3. Integrate additional global weather APIs.
4. Support multi-language functionality.
5. Develop a mobile app for enhanced accessibility.

## References

1. Scikit-learn documentation: <https://scikit-learn.org/>
2. Flask documentation: <https://flask.palletsprojects.com/>
3. OpenWeatherMap API documentation: <https://openweathermap.org/>

## USER Interface:



The screenshot displays the user interface of the 'Rainfall Prediction' application. The title 'Rainfall Prediction' is centered at the top in a large, bold, white font. Below it, a subtitle reads 'Predict the likelihood of rainfall based on current weather conditions.' in a smaller white font. The interface features three radio buttons for input selection: 'Enter City Name' (which is selected), 'Use Current Location', and 'Manual Input'. Below these buttons is a white text input field with the placeholder text 'Enter City Name'. To the right of the input field is a blue button labeled 'Get Weather'. At the bottom of the interface, a small copyright notice states '© 2024 Mubeen's Rainfall Prediction System. All rights reserved.'