Project: Rainfall Prediction Using a Tuned Random Forest Classifier

1. Project Objective

The primary goal of this project was to develop a reliable machine learning model to predict daily rainfall based on atmospheric conditions. The project involved a complete data science workflow, including data cleaning, exploratory data analysis, feature engineering, handling class imbalance, and building an optimized classification model. The final model is saved and ready for integration into predictive applications.

2. Dataset and Features

The analysis was conducted on the Rainfall.csv dataset, which contains 366 daily meteorological records.

Key Features Used for Prediction:

- **pressure**: Atmospheric pressure
- **dewpoint**: Dew point temperature
- **humidity**: Relative humidity
- **cloud**: Cloud cover percentage
- **sunshine**: Hours of sunshine
- **winddirection**: Wind direction
- **windspeed**: Wind speed
- **rainfall**: **(Target)** Binary indicator of rainfall ('yes' or 'no')

3. Methodology and Key Steps

a. Data Cleaning and Preprocessing

The raw data was first rigorously cleaned and prepared for modeling:

1. **Column Name Standardization:** Leading and trailing whitespaces were removed from column names to ensure consistency.
2. **Missing Value Imputation:** Null values in winddirection and windspeed were handled using **mode** and **median** imputation, respectively, to maintain data integrity without introducing bias.
3. **Target Variable Encoding:** The categorical rainfall column was encoded into a binary format (**yes=1, no=0**) to make it suitable for machine learning algorithms.

b. Exploratory Data Analysis (EDA) & Feature Engineering

EDA was crucial for understanding data relationships and guiding feature selection:

1. **Visual Analysis:** Histograms and boxplots were generated to inspect feature distributions and identify outliers.
2. **Correlation Heatmap:** A correlation matrix revealed strong **multicollinearity** between temperature-related features (maxtemp, temparature, mintemp).

3. **Feature Selection:** To avoid model instability, maxtemp, temparature, and mintemp were dropped, streamlining the feature set to only the most impactful variables.

c. Handling Class Imbalance

The dataset showed a significant imbalance, with far more "rainfall" days than "no rainfall" days.

- To prevent the model from being biased toward the majority class, **downsampling** was employed. The majority class was randomly sampled to match the size of the minority class, creating a balanced dataset for training.

```
from sklearn.utils import resample

df_majority = data[data["rainfall"] == 1]
df_minority = data[data["rainfall"] == 0]

df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples=len(df_minority),
                                   random_state=42)

df_downsampled = pd.concat([df_majority_downsampled, df_minority])
```

d. Model Building and Optimization

A **Random Forest Classifier** was selected for its robustness and high performance. The model was trained and optimized using the following process:

1. **Data Splitting:** The balanced dataset was split into **80% for training** and **20% for testing**.
2. **Hyperparameter Tuning: GridSearchCV** was utilized to perform an exhaustive search for the optimal model hyperparameters (e.g., n_estimators, max_depth), ensuring the final model was finely tuned for maximum accuracy.

4. Results and Model Evaluation

The optimized Random Forest model achieved a final **accuracy of 74.5%** on the unseen test data.

Key Performance Metrics:

- **Precision and Recall:** The model demonstrated a strong balance in identifying both "Rainfall" and "No Rainfall" days correctly, as shown in the detailed classification report.
- **Confusion Matrix:** Confirmed the model's effectiveness in minimizing false positives and false negatives.

5. Model Deployment and Usage

For practical application, the trained model and its feature names were serialized into a **pickle** file (rainfall_prediction_model.pkl).

This allows for easy deployment and integration. New predictions can be made by simply loading the file and passing new data, without needing to retrain the model.

Example Prediction Code:

```
# Load the trained model from the pickle file
with open("rainfall_prediction_model.pkl", "rb") as file:
    model_data = pickle.load(file)
model = model_data["model"]
feature_names = model_data["feature_names"]

# Prepare new data and make a prediction
input_data = (1015.9, 19.9, 95, 81, 0.0, 40.0, 13.7)
input_df = pd.DataFrame([input_data], columns=feature_names)
prediction = model.predict(input_df)

# Output the result
print("Prediction:", "Rainfall" if prediction[0] == 1 else "No Rainfall")
```

6. Conclusion and Future Improvements

This project successfully demonstrates an end-to-end machine learning pipeline for rainfall prediction. The final model is accurate, robust, and ready for real-world use.

Potential next steps include:

- **Exploring Advanced Models:** Experimenting with models like Gradient Boosting or SVMs to potentially improve accuracy.
- **Alternative Imbalance Techniques:** Implementing oversampling methods like **SMOTE** to compare performance.
- **API Deployment:** Wrapping the model in a **Flask** or **FastAPI** application to create a predictive web service.