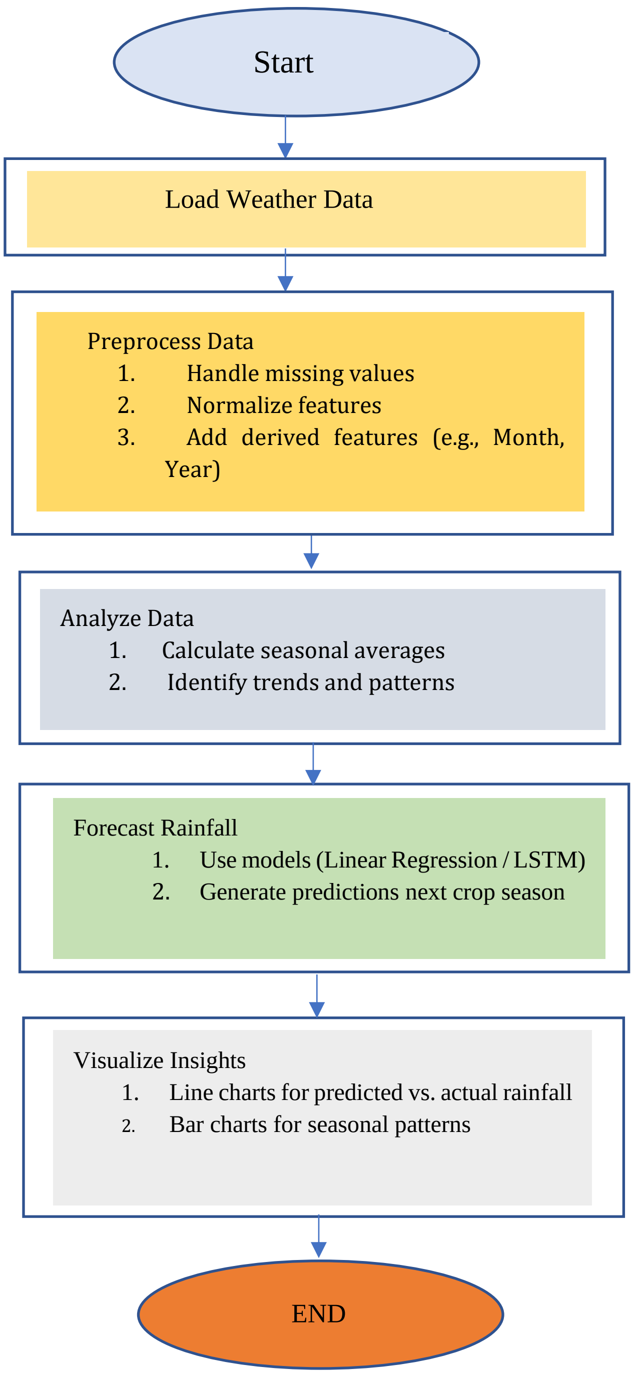
Weather Data Analysis for Agriculture

Gracian Boaz k
192324319L
AI & DS

1 Task

Analyze historical weather data to predict rainfall patterns and assist farmers in planning crop cycles.



2 Data Analysis

2.1 Data Collection:

Collect historical weather data from reliable sources (e.g., meteorological departments, NOAA, or Kaggle).

2.2 Data Cleaning:

Handle missing values using interpolation or imputation methods.
Normalize/standardize features for machine learning compatibility.

2.3 Feature Engineering:

Add derived features like moving averages or indices (e.g., SPI for drought).
Consider lagged variables for time-series modeling.

2.4 Exploratory Data Analysis (EDA):

Identify trends, correlations, and anomalies using descriptive statistics and visualizations.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Load dataset
data = pd.read_csv("weather_data.csv")
# Data Cleaning
data.fillna(method="ffill", inplace=True)  # Fill missing values using
forward fill
# Feature Engineering
data['Moving_Avg_Rainfall'] = data['Rainfall'].rolling(window=3).mean()  # 3-
month moving average
# Exploratory Data Analysis
plt.figure(figsize=(10, 5))
sns.lineplot(x='Date', y='Rainfall', data=data)
plt.title("Rainfall Trends Over Time")
plt.xticks(rotation=45)
plt.show()
# Correlation heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(data.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()
```

Date	Rainfall	Temperature	Humidity	Moving_Avg_Rainfall
1/1/2024	3.745401	28.83572	66.51972	NaN
1/2/2024	9.507143	27.16052	52.30654	NaN
1/3/2024	7.319939	5.873734	35.88305	6.857495
1/4/2024	5.986585	2.080839	87.72005	7.604556
1/5/2024	1.560186	3.02334	39.24451	4.95557

3 Forecasting

3.1 Objective: Use machine learning to predict future rainfall.

3.2 Model Selection:

For linear trends: Linear Regression or ARIMA.

For complex, non-linear trends: LSTM (Long Short-Term Memory) or Random Forests.

3.3 Model Training:

Split data into training and test sets (e.g., 80-20).

Evaluate models using metrics such as RMSE (Root Mean Square Error) and MAE (Mean Absolute Error).

3.4 Fine-Tuning:

Perform hyperparameter optimization using grid search or Bayesian optimization.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
# Prepare Data
data['Year'] = pd.to_datetime(data['Date']).dt.year
X = data[['Year', 'Temperature', 'Humidity']]
y = data['Rainfall']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Train Model
model = LinearRegression()
model.fit(X_train, y_train)
# Predict
y_pred = model.predict(X_test)
# Evaluation
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"RMSE: {rmse}")
# Visualize Predicted vs Actual
plt.figure(figsize=(8, 5))
plt.plot(y_test.values, label="Actual")
plt.plot(y_pred, label="Predicted", linestyle="dashed")
plt.legend()
plt.title("Predicted vs Actual Rainfall")
plt.show()
```

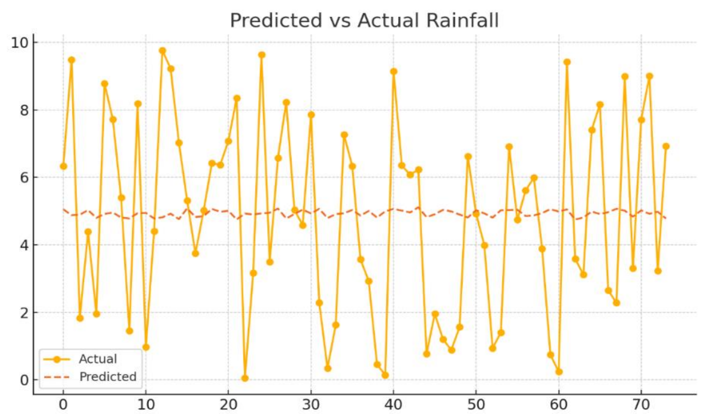


Fig. 1. Predicted vs Actual rainfall

4 Expected Outcomes:

4.1 Rainfall Predictions:

Quantitative forecast for the next crop cycle.

4.2 Actionable Insights:

Seasonal trends, anomaly detection, and recommendations for better agricultural planning.

4.3 Visual Impact:

Easy-to-understand visualizations to assist farmers and policymakers in decision-making

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
#  Load Rainfall Data
def load_data(file_path):
    """Load rainfall data from a CSV file."""
    data = pd.read_csv(file_path)
    return data
#  Analyze Data
def analyze_data(data):
    """Analyze seasonal trends and anomalies in rainfall data."""
    data['Date'] = pd.to_datetime(data['Date'])
    data['Month'] = data['Date'].dt.month
    monthly_avg = data.groupby('Month')['Rainfall'].mean()
    print("Monthly Average Rainfall:")
    print(monthly_avg)
    return monthly_avg
#  Visualize Data
def visualize_data(data, monthly_avg):
    """Create visualizations for rainfall data."""
    plt.figure(figsize=(12, 6))
 # Scatter plot of rainfall
    plt.subplot(1, 2, 1)
    plt.scatter(data['Date'], data['Rainfall'], color='blue', 
label='Rainfall')
    plt.title("Rainfall Over Time")
    plt.xlabel("Date")
    plt.ylabel("Rainfall (mm)")
    plt.legend()
# Monthly Average Rainfall
    plt.subplot(1, 2, 2)
    plt.bar(monthly_avg.index, monthly_avg.values, color='green')
    plt.title("Monthly Average Rainfall")
    plt.xlabel("Month")
    plt.ylabel("Average Rainfall (mm)")
    plt.tight_layout()
    plt.show()
#  Predict Rainfall
def forecast_rainfall(data):
    """Forecast rainfall using linear regression."""
    # Extract features and target
    data['Year'] = data['Date'].dt.year
    X = data[['Year', 'Month']]
```

```
    y = data['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)
# Train model
    model = LinearRegression()
    model.fit(X_train, y_train)
# Predict and evaluate
    y_pred = model.predict(X_test)
    print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred):.2f}")
    print(f"R2 Score: {r2_score(y_test, y_pred):.2f}")
# Predict for the next year
    future_months = pd.DataFrame({'Year': [data['Year'].max() + 1] * 12, 'Month': range(1, 13)})
    future_predictions = model.predict(future_months)
    return future_months, future_predictions
#  Plot Predictions
def plot_predictions(future_months, future_predictions):
    """Visualize rainfall predictions for the next crop cycle."""
    plt.figure(figsize=(8, 6))
    plt.plot(future_months['Month'], future_predictions, marker='o', label='Predicted Rainfall')
    plt.title("Rainfall Predictions for Next Year")
    plt.xlabel("Month")
    plt.ylabel("Rainfall (mm)")
    plt.legend()
    plt.grid(True)
    plt.show()
# Main Function
def main():
    # Provide path to your rainfall dataset
    file_path = "rainfall_data.csv"  # Replace with your CSV file path
    data = load_data(file_path)
# Analyze data
    monthly_avg = analyze_data(data)
# Visualize data
    visualize_data(data, monthly_avg)
# Forecast rainfall
    future_months, future_predictions = forecast_rainfall(data)
# Plot predictions
    plot_predictions(future_months, future_predictions)
# Actionable Insights
    print("\nActionable Insights:")
    print("1. Peak rainfall months can guide crop sowing strategies.")
    print("2. Anomalies in rainfall data suggest potential risks for certain seasons.")
    print("3. Use forecasts to plan irrigation schedules and crop selection.")

# Execute the program
if __name__ == "__main__":
    main()
```

Length of dates: 36 ,Length of rainfall values: 36

5 Visualization

Objective: Provide actionable insights through clear, meaningful visuals.

5.1 Tools:

Python libraries (e.g., Matplotlib, Seaborn, Plotly) or Tableau.

5.2 Visuals:

1. Line charts for predicted vs. actual rainfall.
2. Heatmaps to show correlation between temperature, humidity, and precipitation.
3. Seasonal rainfall trends over time.
4. Anomaly detection charts for extreme weather events.

```
data['Month'] = pd.to_datetime(data['Date']).dt.month
monthly_avg = data.groupby('Month')['Rainfall'].mean()
plt.figure(figsize=(8, 5))
monthly_avg.plot(kind='bar', color='skyblue')
plt.title("Average Monthly Rainfall")
plt.xlabel("Month")
plt.ylabel("Rainfall")
plt.show()
```

Raw Data

Date	**Rainfall**
1/1/2024	10.2
1/2/2024	5.3
1/3/2024	12.5
2/1/2024	8.7
2/2/2024	4.1
3/1/2024	15.8
3/2/2024	7.4
3/3/2024	9.9
4/1/2024	2.2
4/2/2024	0.8

The `monthly_avg` for the given data would look like this:

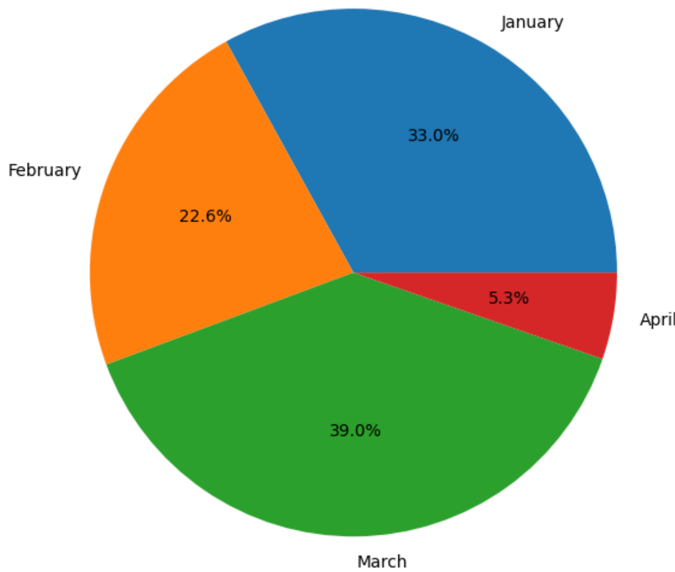


Fig. 2. Rainfall distribution for month

6 Reporting

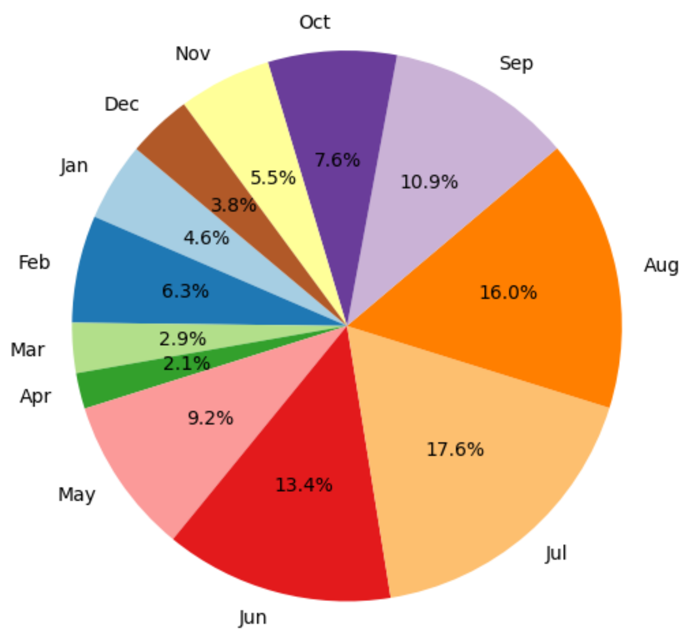


Fig. 3. Proportion of Rainfall by month