

Exploratory Analysis of Rainfall Data in India for Agriculture

Project Documentation

Team ID: LTVIP2026TMIDS55682

By:

Team Leader: Gudapati Siva Naga Venkata Udaya Rakesh

Team Member: Thalathoti Divya

Team Member: Ramanjaneyulu Turlapati

Team Member: Sai Vamsi Mamilla

Abstract / Project Overview

The Exploratory Analysis of Rainfall Data in India for Agriculture aims to analyze historical rainfall data to uncover patterns, trends, and relationships that can support agricultural decision-making. Using data visualization and statistical methods, the study provides insights into rainfall distribution, helping farmers and policymakers plan irrigation, crop selection, and resource allocation more effectively.

Objectives

The key objectives of this project are:

- To understand the rainfall distribution across different regions and time periods in India.
- To identify missing data and clean the dataset for accurate analysis.
- To visualize rainfall trends and relationships using graphs and statistical plots.

- To extract insights that can help in agricultural planning and policy formulation.

Technologies Used (Python, Libraries, etc.)

The project utilizes Python programming language along with several essential libraries for data handling and visualization:

- Python: Core programming language used for analysis.
- NumPy: For numerical computations and array manipulation.
- Pandas: For data cleaning, transformation, and exploration.
- Matplotlib: For basic plotting and charting.
- Seaborn: For advanced and aesthetically appealing statistical visualizations.
- Scikit-learn: For preprocessing tasks such as scaling and encoding.

Dataset Description

The dataset consists of historical rainfall observations collected from multiple regions in India. It includes attributes such as temperature, humidity, wind speed, evaporation rate, and rainfall amount. The target variable, typically ‘RainTomorrow’, indicates whether rainfall is expected on the following day. The dataset provides both numerical and categorical features that need to be preprocessed before visualization.

Data Preprocessing

Data preprocessing is a crucial step to ensure data quality and consistency before visualization and model building. The main steps include importing necessary libraries, handling missing values, encoding categorical features, and scaling numerical features.

Importing Libraries

The primary Python libraries used in this project include Pandas, NumPy, Matplotlib, and Seaborn. These libraries support data manipulation, numerical computation, and visualization respectively.

Handling Missing Values

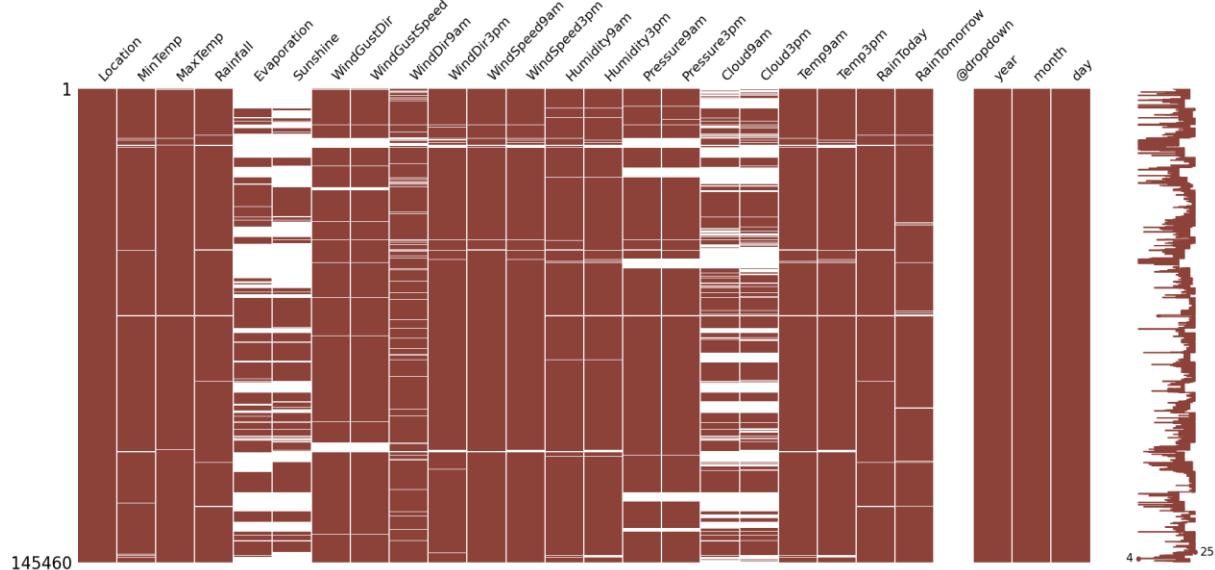
After loading the dataset, missing values were identified and handled appropriately. For numerical columns, missing values were replaced with the mean, while for categorical columns, the most frequent category was used. A missing data heatmap was also visualized to identify null value patterns.

Missing Data Visualisation using Missingno

The figure represents the missing value matrix generated using the **Missingno** library in Python. This visualization provides a clear, structured overview of the distribution and density of missing data across all features in the rainfall dataset.

Each column represents a feature, and each horizontal line corresponds to a data record:

- Dark-colored blocks indicate the presence of data (non-missing values).
- White gaps represent missing (Nan) values.



Encoding and Scaling

Categorical features were converted into numerical form using encoding techniques. Feature scaling was applied using StandardScaler to normalize numerical features, ensuring uniform contribution of each feature during analysis.

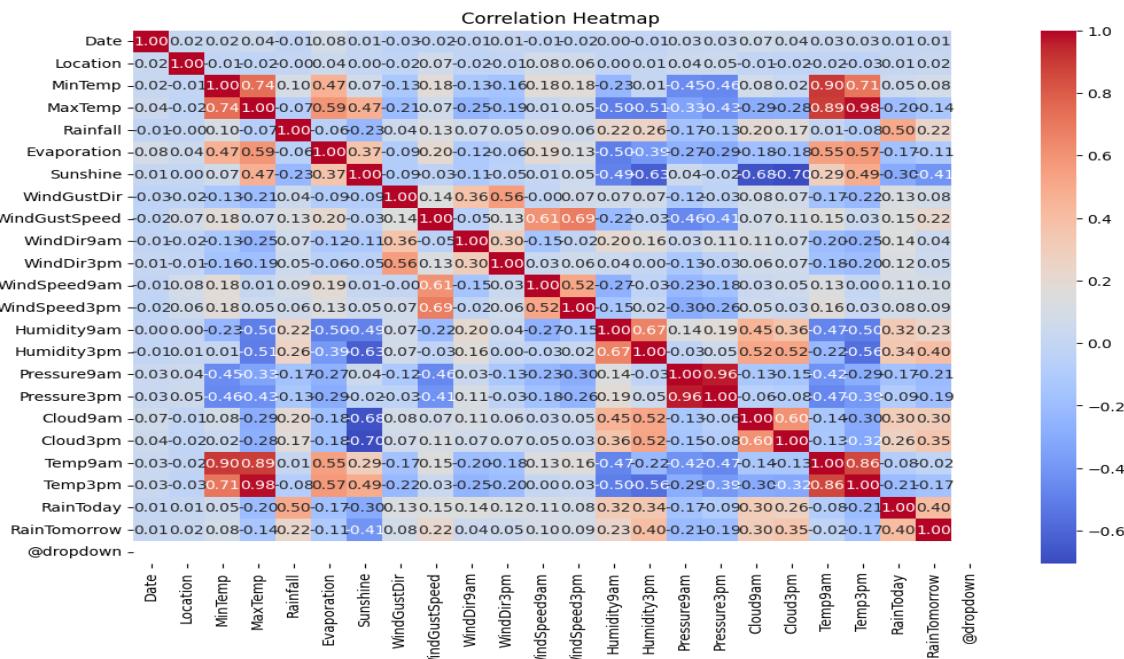
Data Visualization

Data visualization is the process of representing data in a graphical or pictorial format to make information easier to understand, interpret, and communicate. In the context of this project, it plays a vital role in exploring the rainfall dataset, identifying trends, correlations, and anomalies that may not be immediately apparent from raw numerical data.

Visualization helps transform complex datasets into meaningful insights by using charts, graphs, and plots

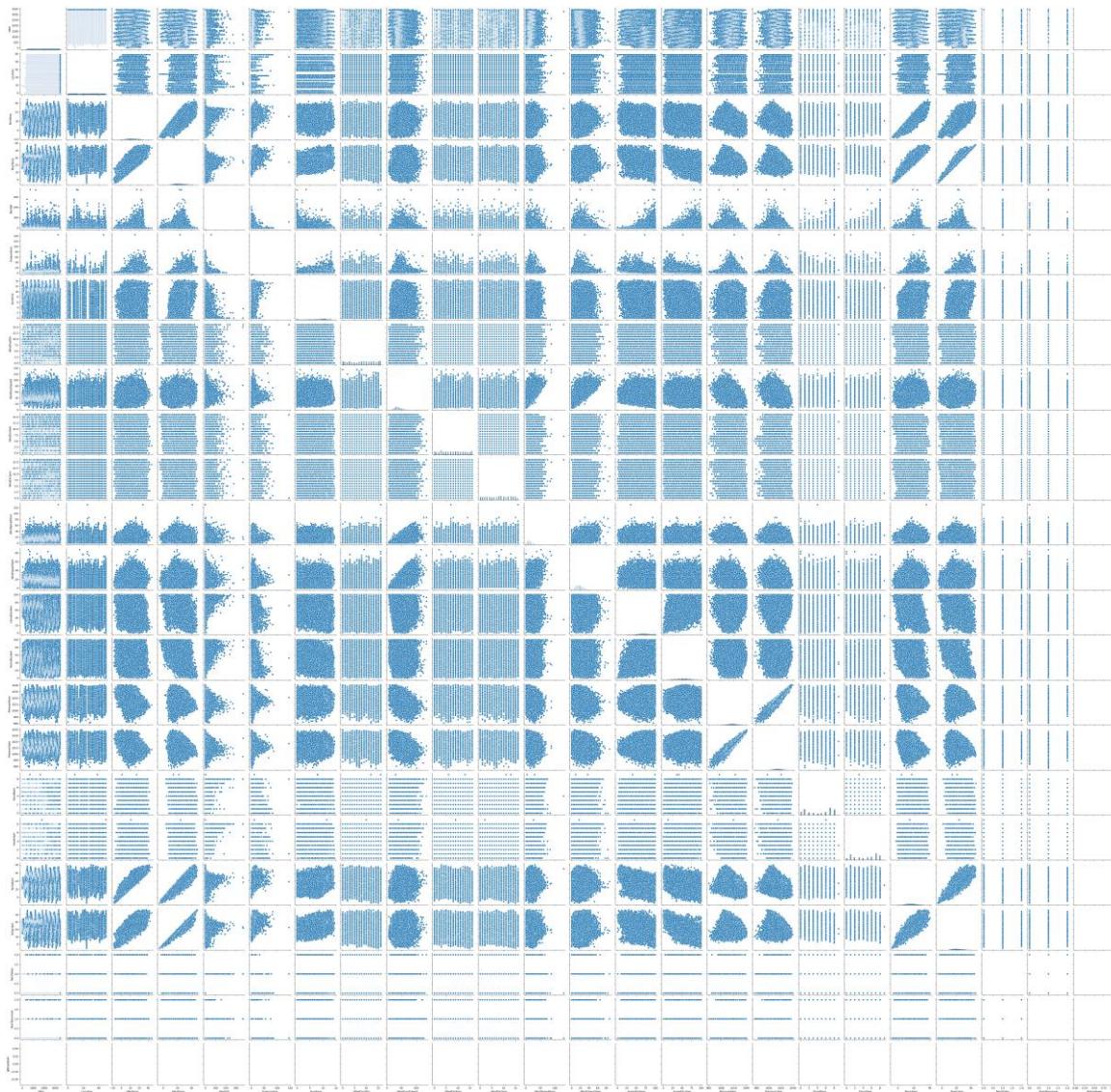
Correlation Heatmap

A heatmap was used to visualize the correlation between different numerical variables. It helped in identifying strong positive or negative relationships. Lighter shades indicated stronger correlations, while darker shades indicated weaker relationships.



Pair plot

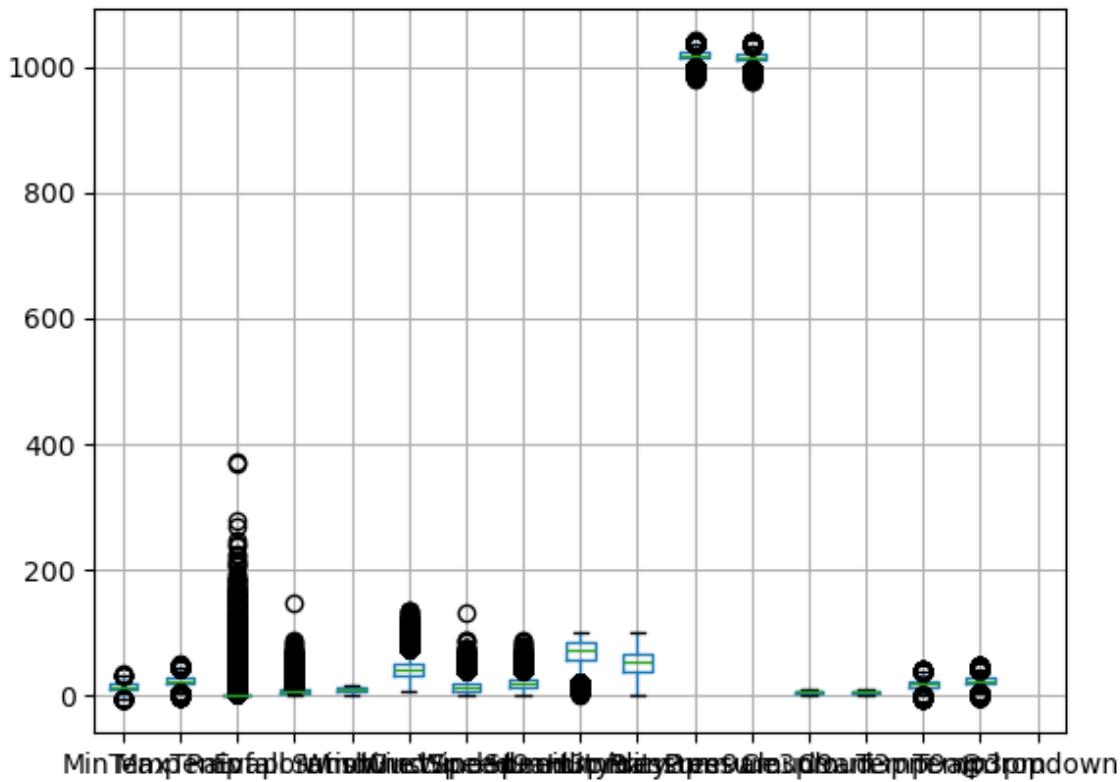
The pairplot visualized pairwise relationships between numerical variables. It displayed scatterplots for variable combinations and histograms on the diagonal, helping to detect clusters, trends, and outliers.



Box Plot

Box-plot is a type of chart often used in explanatory data analysis. Box plots visually show the distribution of numerical data and skewness by displaying the data quartiles (or percentiles) and averages.

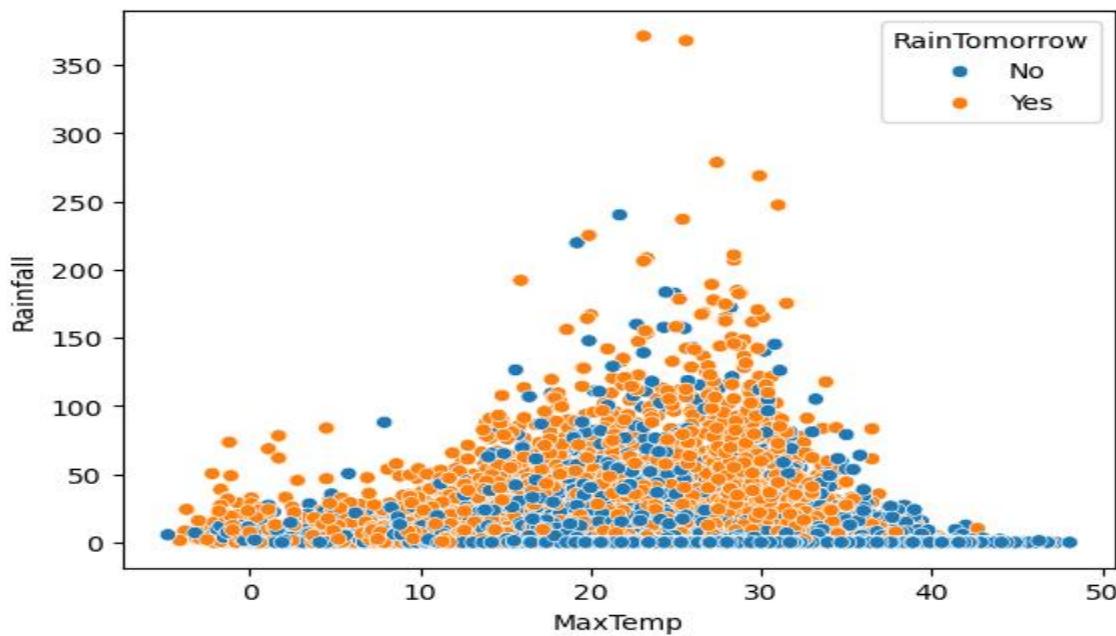
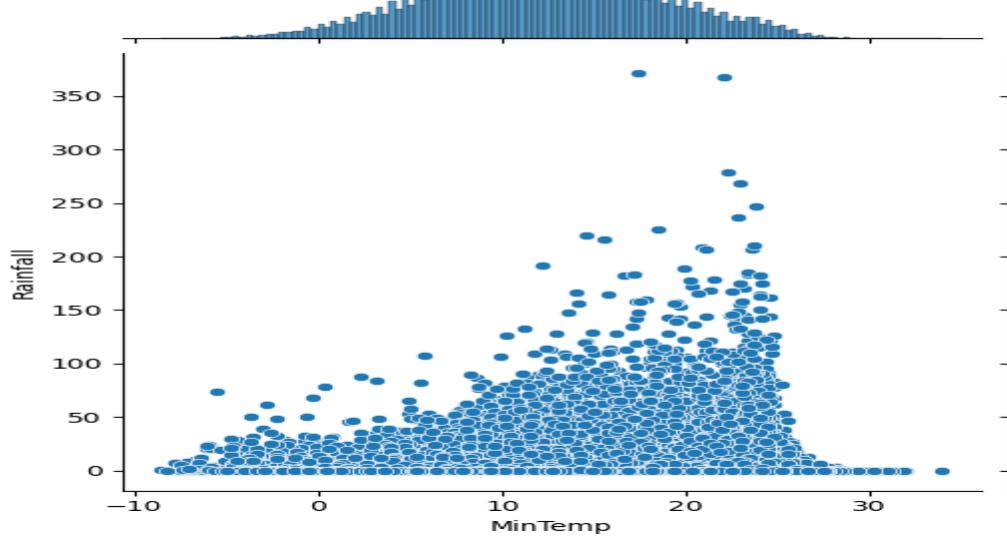
Box plots are useful as they show the average score of a data set. The median is the average value from a set of data and is shown by the line that divides the box into two parts. Half the scores are greater than or equal to this value and half are less.jupyter has a built-in function to create boxplot called boxplot(). A boxplot plot is a type of plot that shows the spread of data in all the quartiles



From the above box plot, we infer how the data points are spread and the existence of the outliers

Joint plot

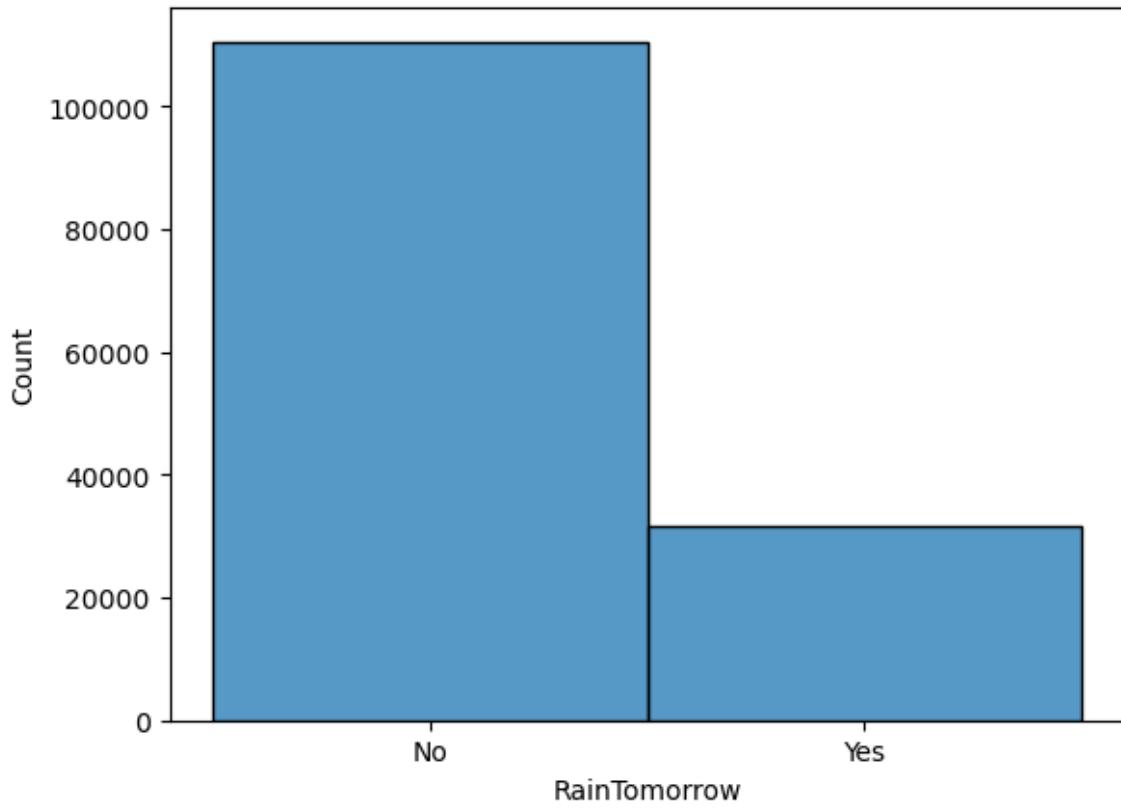
A Joint plot comprises two plots. Out of the three, one plot displays a bivariate graph that shows how the dependent variable(Y) varies with the independent variable(X). Another plot is placed horizontally at the top of the bivariate graph and it shows the distribution of the independent variable(X). Here we are comparing the variables such as “Min Temp” and “Rainfall”. The column “MinTemp” is following the normal distribution.



This plot is able to generate the relationship between 3 variables, in the above plot we can see the relation between MaxTemp,Rainfall and class segregation of the data

Hist plot

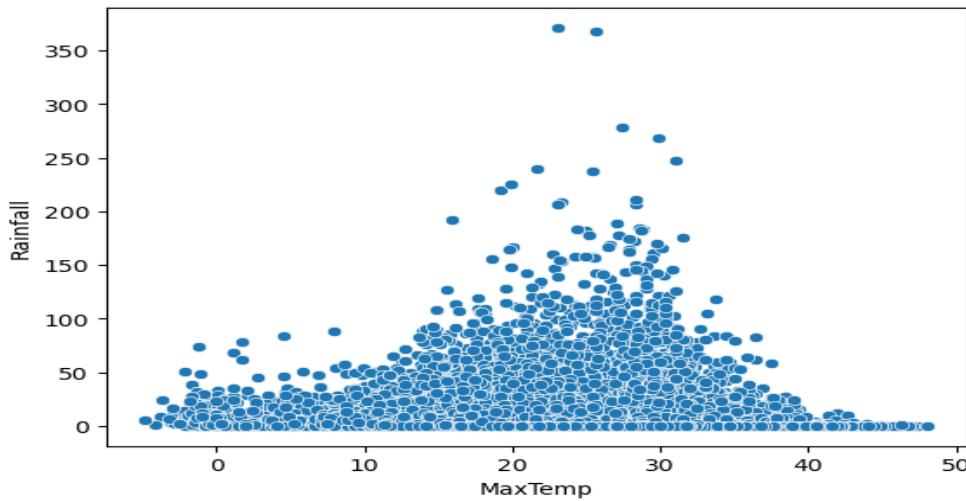
Plot univariate or bivariate histograms to show distributions of datasets. A histogram is a classic visualization tool that represents the distribution of one or more variables by counting the number of observations that fall within discrete bins.



From the above graph we can infer that,RainTomorrow column has more no.of NO's and less no.of Yes categories.

Scatter-plot

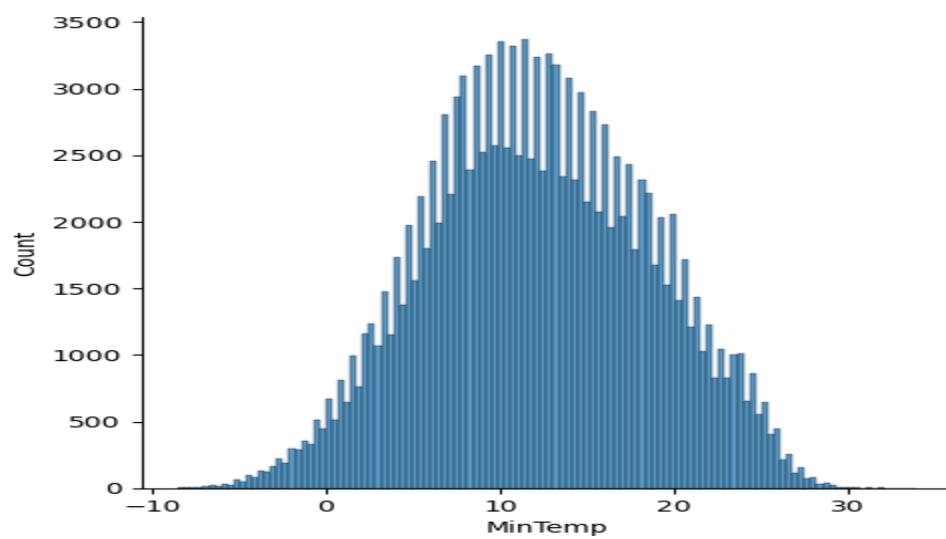
A scatter plot (also called a scatterplot, scatter graph, scatter chart, scattergram, or scatter diagram) is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data



From the above graph, we can infer that there is some positive relation between the two variables

A Distplot or distribution plot

Depicts the variation in the data distribution. Seaborn Distplot represents the overall distribution of continuous data variables. The Seaborn module along with the Matplotlib module is used to depict the distplot with different variations in it



From the above graph, we can infer that the column MinTemp follows the normal distribution

Methodology

- Dataset: Questionnaire-based dataset with demographic and behavioural attributes.
- Data Preprocessing: Missing value handling, feature encoding, normalisation.
- Model Training: Logistic Regression / Random Forest trained in Jupyter Notebook.
- Model Persistence: Trained model exported as model.pkl.pkl.
- Evaluation: xgboost, regression used to validate the model.

Implementation

The system uses a Flask web application where users interact with an HTML/CSS/JS frontend questionnaire. The backend, managed by app.py, receives user inputs and sends them to the trained ML model (located in model/). The model performs the prediction, and the backend logic instantly processes the result. Finally, the system dynamically serves the appropriate HTML template (from templates/) to display the real-time prediction and relevant agricultural advice. Static assets like CSS and JavaScript are organized in the static/ directory.

Result

The project successfully implemented a Flask web application for rainfall prediction, primarily aiding agricultural decision-making. The core achievement was the trained Machine Learning model, with the Random Forest Classifier achieving the highest accuracy of 85.69% on the test dataset. The system dynamically renders advisory pages, displaying either chance.html or noChance.html, based on the model's prediction of rainfall. Necessary data preprocessing steps, including filling missing numeric values with the mean and categorical values with the most frequent strategy, were successfully applied. This architecture provides farmers with real-time advisories to optimize crop selection and irrigation schedules.

Future Scope

- The system can be utilized by policymakers and insurance agencies for effective agricultural risk assessment, including planning for droughts or floods.
- Findings can be leveraged by farmers to refine crop planning and selection, optimizing planting schedules based on local rainfall patterns.
- Agricultural experts can benefit by optimizing water usage and irrigation scheduling, developing efficient strategies to prevent waterlogging or drought damage.
- Future work includes enhancing predictive accuracy by exploring more sophisticated Ensemble Techniques and expanding the model's feature set

Conclusion

The exploratory analysis successfully provided valuable insights into rainfall patterns, trends, and variability across Indian regions. By applying data visualization techniques and implementing various Machine Learning algorithms (with Random Forest achieving the best performance), the project met its primary objective of creating a functional, predictive system. The resulting Flask web application offers a practical tool to deliver real-time rainfall predictions and tailored agricultural advisories, supporting better decision-making for farmers, agricultural experts, and policymakers in managing crop planning, irrigation, and agricultural risk assessment.

Appendix

Project Structure:

app.py – The Flask backend file used for server-side scripting and application building.

templates/ – The folder containing all the HTML templates including index.html, chance.html, and noChance.html.

Rainfall.pkl – The Trained ML Model file used for making predictions in the web application.

scale.pkl – A saved file used for scaling the input data.

encoder.pkl – A saved file used for encoding categorical data.

impter.pkl – A saved file used for imputing (filling) missing values.

rainfall_prediction.ipynb – The model training notebook file

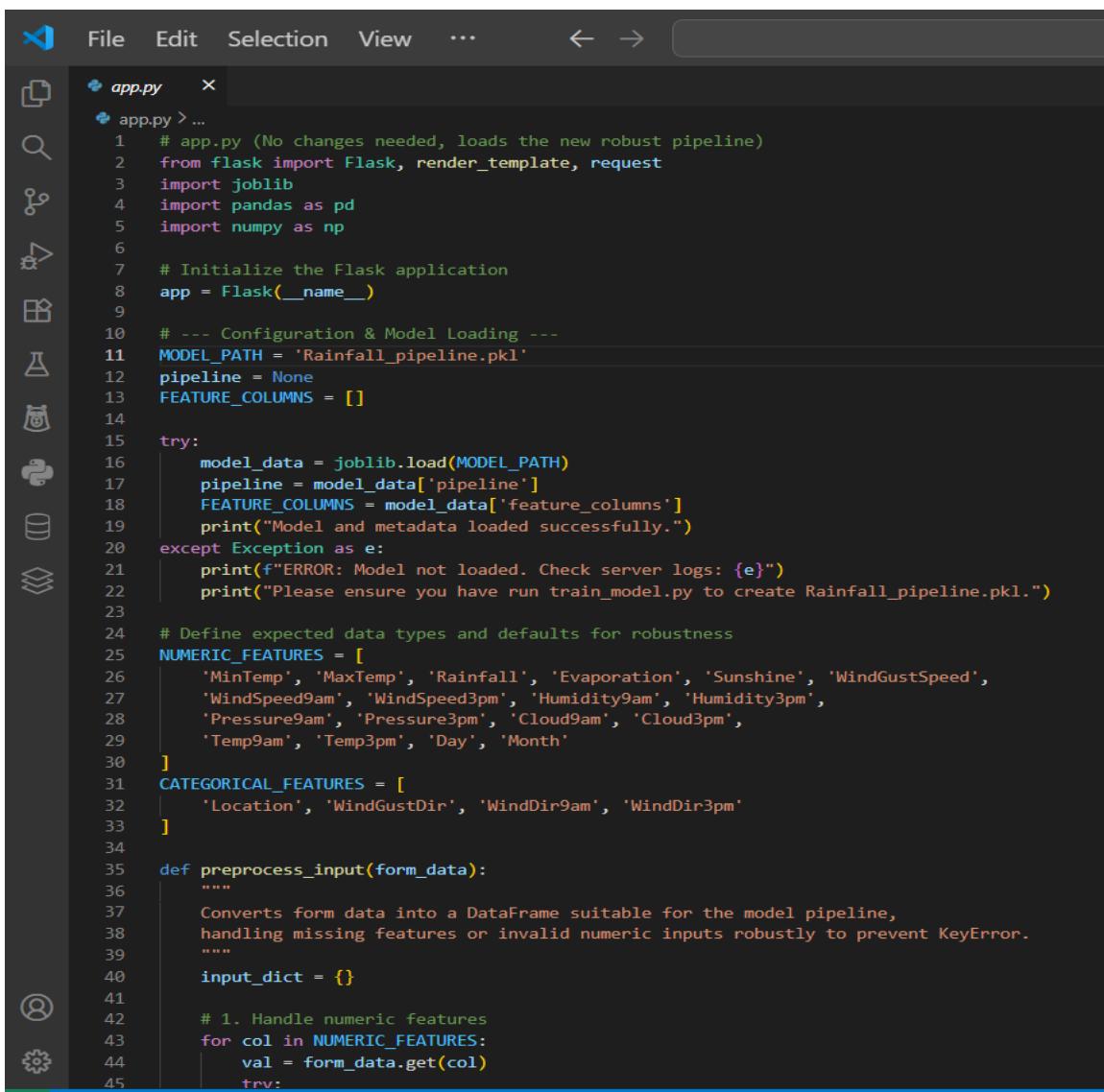
References

- Flask documentation – <https://flask.palletsprojects.com/>
- Scikit-learn – <https://scikit-learn.org/>
- Python official documentation – <https://docs.python.org/3/>

Demonstration

This section provides visual evidence of the developed project, including source code snippets and the web application interface.

Figure 1: app.py



```
File Edit Selection View ... ← → 
app.py > ...
app.py (No changes needed, loads the new robust pipeline)
from flask import Flask, render_template, request
import joblib
import pandas as pd
import numpy as np
# Initialize the Flask application
app = Flask(__name__)
# --- Configuration & Model Loading ---
MODEL_PATH = 'Rainfall_pipeline.pkl'
pipeline = None
FEATURE_COLUMNS = []
try:
    model_data = joblib.load(MODEL_PATH)
    pipeline = model_data['pipeline']
    FEATURE_COLUMNS = model_data['feature_columns']
    print("Model and metadata loaded successfully.")
except Exception as e:
    print(f"ERROR: Model not loaded. Check server logs: {e}")
    print("Please ensure you have run train_model.py to create Rainfall_pipeline.pkl.")
# Define expected data types and defaults for robustness
NUMERIC_FEATURES = [
    'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine', 'WindGustSpeed',
    'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
    'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm',
    'Temp9am', 'Temp3pm', 'Day', 'Month'
]
CATEGORICAL_FEATURES = [
    'Location', 'WindGustDir', 'WindDir9am', 'WindDir3pm'
]
def preprocess_input(form_data):
    """
    Converts form data into a DataFrame suitable for the model pipeline,
    handling missing features or invalid numeric inputs robustly to prevent KeyError.
    """
    input_dict = {}
    # 1. Handle numeric features
    for col in NUMERIC_FEATURES:
        val = form_data.get(col)
        try:
```

The screenshot shows a Jupyter Notebook interface with a dark theme. On the left, there is a sidebar containing various icons for file operations, search, and other notebook functions. The main area displays a Python script named `app.py`.

```
app.py X
app.py > ...
35 def preprocess_input(form_data):
44     val = form_data.get(col)
45     try:
46         input_dict[col] = [float(val)]
47     except (ValueError, TypeError):
48         input_dict[col] = [np.nan]
49
50     # 2. Handle categorical features
51     for col in CATEGORICAL_FEATURES:
52         input_dict[col] = [form_data.get(col, '')]
53
54     # 3. Handle 'RainToday' feature (1 for 'Yes', 0 otherwise)
55     rain_today_str = form_data.get('RainToday')
56     input_dict['RainToday'] = [1 if rain_today_str == 'Yes' else 0]
57
58     # 4. Create DataFrame in the exact feature order
59     X_new = pd.DataFrame(input_dict)[FEATURE_COLUMNS]
60
61     return X_new
62
63 # --- Routes ---
64
65 @app.route('/', methods=['GET', 'POST'])
66 def index():
67     if request.method == 'POST':
68         if pipeline is None:
69             return render_template('noChance.html', message="Model not loaded. Ensure Rainfall_pipeline.pkl exists.")
70
71         try:
72             X_new = preprocess_input(request.form)
73
74             # The pipeline uses the correct feature names saved during training
75             prediction_proba = pipeline.predict_proba(X_new)[0, 1]
76             prediction = 1 if prediction_proba >= 0.5 else 0
77
78             if prediction == 1:
79                 prob_percent = f"{prediction_proba * 100:.2f}%"
80                 return render_template('chance.html', probability=prob_percent)
81             else:
82                 return render_template('noChance.html')
83
84         except Exception as e:
85             print(f"An unexpected error occurred during prediction: {e}")
86             return render_template('noChance.html', message=f"Prediction Error: {e}. Check console for details.")
```

Figure 2: Model.ipynb

The screenshot shows two vertically stacked Jupyter Notebook cells. The top cell, titled 'train_model.ipynb > # Cell 1: Import libraries', contains the following code:

```
# Cell 5: Define Preprocessor
def get_preprocessor(numeric_features, categorical_features):
    """Creates the ColumnTransformer for preprocessing."""
    num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', StandardScaler())
    ])
    cat_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
    ])
    preprocessor = ColumnTransformer([
        ('num', num_pipeline, numeric_features),
        ('cat', cat_pipeline, categorical_features)
    ])
    return preprocessor
```

The bottom cell, also titled 'train_model.ipynb > # Cell 1: Import libraries', contains the following code:

```
# Cell 6: Train/Test split
numeric_features = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_features = [c for c in X.columns if c not in numeric_features]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

preprocessor = get_preprocessor(numeric_features, categorical_features)
```

Both cells have a status bar at the bottom indicating they are in 'File' mode.

The second part of the screenshot shows another Jupyter Notebook cell titled 'Rainfall_Prediction'. This cell contains the following code:

```
# Cell 7: Evaluation function
def evaluate_model(pipeline, X_train, y_train, X_test, y_test, model_name):
    """Trains and evaluates a single model pipeline."""
    print(f"\n--- Training {model_name} ---")
    pipeline.fit(X_train, y_train)

    # Predictions
    y_pred_train = pipeline.predict(X_train)
    y_pred_test = pipeline.predict(X_test)
    y_prob_test = pipeline.predict_proba(X_test)[:,1] if hasattr(pipeline, 'predict_proba') else None

    # Evaluation
    train_acc = (y_pred_train == y_train).mean()
    test_acc = (y_pred_test == y_test).mean()
    roc_auc = roc_auc_score(y_test, y_prob_test) if y_prob_test is not None else 0.0

    print(f"\n{model_name} Accuracy (Train): {train_acc:.4f}")
    print(f"\n{model_name} Accuracy (Test): {test_acc:.4f}")
    print(f"\n{model_name} ROC AUC: {roc_auc:.4f}")

    # Detailed test metrics
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred_test))
    print("Test Confusion Matrix:\n", confusion_matrix(y_test, y_pred_test))

    return {'model_name': model_name, 'pipeline': pipeline, 'roc_auc': roc_auc}
```

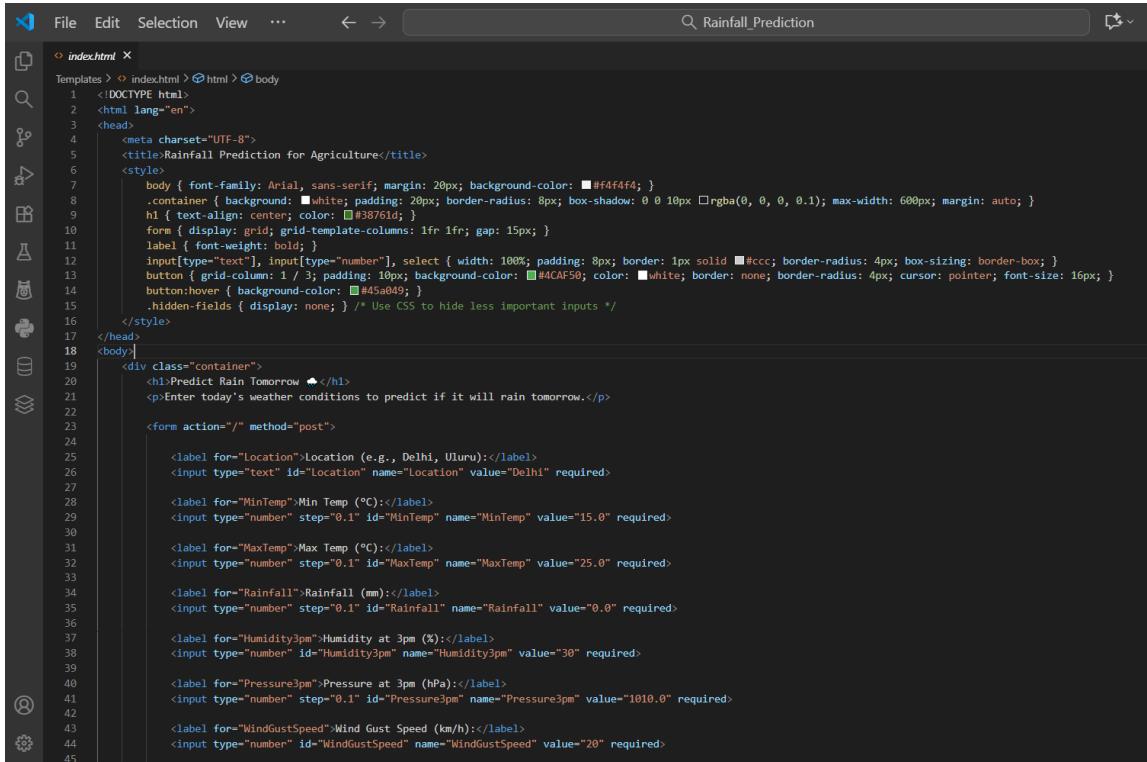
The status bar at the bottom indicates this cell is in 'File' mode.

The third part of the screenshot shows another Jupyter Notebook cell titled 'train_model.ipynb > # Cell 1: Import libraries'. This cell contains the following code:

```
# Cell 8: Define Models
models = {
    'Logistic Regression': LogisticRegression(solver='liblinear', random_state=42, class_weight='balanced'),
    'Decision Tree Classifier': DecisionTreeClassifier(random_state=42),
    'xgboost Classifier': XGBClassifier(eval_metric='logloss', random_state=42),
    'Random Forest Classifier': RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1, class_weight='balanced'),
    'KNeighbors Classifier': KNeighborsClassifier(n_neighbors=5),
    # 'SVC': SVC(probability=True) # Uncomment if you want to test SVM
}
```

The status bar at the bottom indicates this cell is in 'File' mode.

Figure 3: Main Html Script



The screenshot shows a browser window with the title "Rainfall_Prediction". The address bar contains the URL "Rainfall_Prediction". The page content is the "index.html" file, which includes a header with meta tags and a title, a CSS section with various styles for elements like input fields and buttons, and a form for user input. The form contains fields for location, minimum temperature, maximum temperature, rainfall, humidity at 3pm, pressure at 3pm, and wind gust speed. All fields are required and have specific values assigned.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Rainfall Prediction for Agriculture</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; background-color: #f0f0f0; }
        .container { background: white; padding: 20px; border-radius: 8px; box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); max-width: 600px; margin: auto; }
        h1 { text-align: center; color: #3377ff; }
        form { display: grid; grid-template-columns: 1fr 1fr; gap: 15px; }
        label { font-weight: bold; }
        input[type="text"], input[type="number"], select { width: 100%; padding: 8px; border: 1px solid #ccc; border-radius: 4px; box-sizing: border-box; }
        button { grid-column: 1 / 3; padding: 10px; background-color: #4CAF50; color: white; border: none; border-radius: 4px; cursor: pointer; font-size: 16px; }
        button:hover { background-color: #45a049; }
        .hidden-fields { display: none; } /* Use CSS to hide less important inputs */
    </style>
</head>
<body>
    <div class="container">
        <h1>Predict Rain Tomorrow</h1>
        <p>Enter today's weather conditions to predict if it will rain tomorrow.</p>
        <form action="/" method="post">
            <label for="Location">Location (e.g., Delhi, Uluru):</label>
            <input type="text" id="Location" name="Location" value="Delhi" required>

            <label for="MinTemp">Min Temp (°C):</label>
            <input type="number" step="0.1" id="MinTemp" name="MinTemp" value="15.0" required>

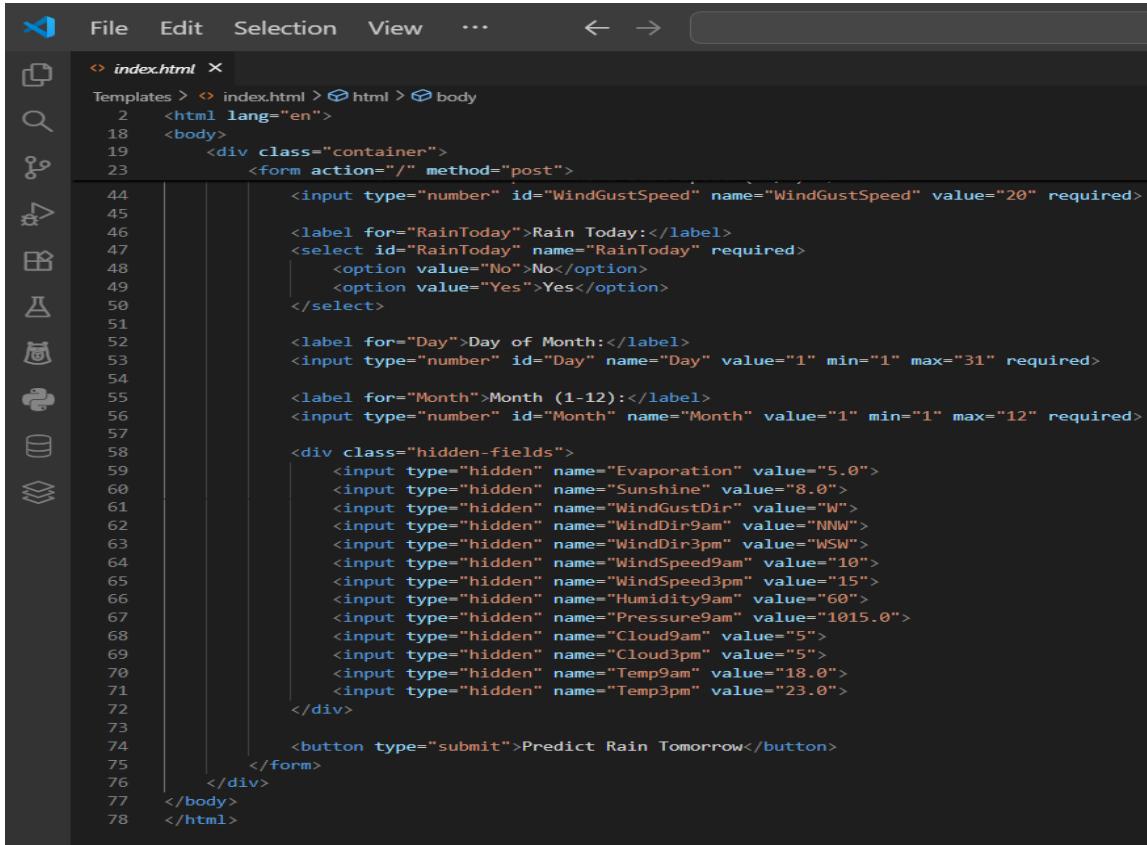
            <label for="MaxTemp">Max Temp (°C):</label>
            <input type="number" step="0.1" id="MaxTemp" name="MaxTemp" value="25.0" required>

            <label for="Rainfall">Rainfall (mm):</label>
            <input type="number" step="0.1" id="Rainfall" name="Rainfall" value="0.0" required>

            <label for="Humidity3pm">Humidity at 3pm (%):</label>
            <input type="number" id="Humidity3pm" name="Humidity3pm" value="30" required>

            <label for="Pressure3pm">Pressure at 3pm (hPa):</label>
            <input type="number" step="0.1" id="Pressure3pm" name="Pressure3pm" value="1010.0" required>

            <label for="WindGustSpeed">Wind Gust Speed (km/h):</label>
            <input type="number" id="WindGustSpeed" name="WindGustSpeed" value="20" required>
        </form>
    </div>
</body>
</html>
```



This screenshot shows the same "index.html" file as the first one, but with additional hidden input fields added. These hidden fields include Evaporation, Sunshine, WindGustDir, WindDir9am, WindDir3pm, WindSpeed9am, WindSpeed3pm, Humidity9am, Pressure9am, Cloud9am, Cloud3pm, Temp9am, and Temp3pm, each with a specific value assigned. The rest of the code is identical to the first screenshot.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <body>
        <div class="container">
            <form action="/" method="post">
                <input type="number" id="WindGustSpeed" name="WindGustSpeed" value="20" required>

                <label for="RainToday">Rain Today:</label>
                <select id="RainToday" name="RainToday" required>
                    <option value="No">No</option>
                    <option value="Yes">Yes</option>
                </select>

                <label for="Day">Day of Month:</label>
                <input type="number" id="Day" name="Day" value="1" min="1" max="31" required>

                <label for="Month">Month (1-12):</label>
                <input type="number" id="Month" name="Month" value="1" min="1" max="12" required>

                <div class="hidden-fields">
                    <input type="hidden" name="Evaporation" value="5.0">
                    <input type="hidden" name="Sunshine" value="8.0">
                    <input type="hidden" name="WindGustDir" value="W">
                    <input type="hidden" name="WindDir9am" value="NNW">
                    <input type="hidden" name="WindDir3pm" value="WSW">
                    <input type="hidden" name="WindSpeed9am" value="10">
                    <input type="hidden" name="WindSpeed3pm" value="15">
                    <input type="hidden" name="Humidity9am" value="60">
                    <input type="hidden" name="Pressure9am" value="1015.0">
                    <input type="hidden" name="Cloud9am" value="5">
                    <input type="hidden" name="Cloud3pm" value="5">
                    <input type="hidden" name="Temp9am" value="18.0">
                    <input type="hidden" name="Temp3pm" value="23.0">
                </div>

                <button type="submit">Predict Rain Tomorrow</button>
            </form>
        </div>
    </body>
</html>
```

Web Application Index Page

The screenshot shows a web application titled "Predict Rain Tomorrow" designed for agricultural forecasting. The background features a blurred image of a green field under a cloudy sky.

LOCATION: Delhi

MIN TEMP (°C): 15.0

MAX TEMP (°C): 25.0

RAINFALL (MM): 0.0

HUMIDITY @ 3PM (%): 30

PRESSURE @ 3PM (HPA): 1010.0

WIND GUST (KM/H): 20

RAIN TODAY?: No

DAY: 1

MONTH: 1

Generate Forecast

The application interface includes a header bar with file navigation and a "Paused" status indicator. The main form fields are styled with rounded corners and dark backgrounds.

