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

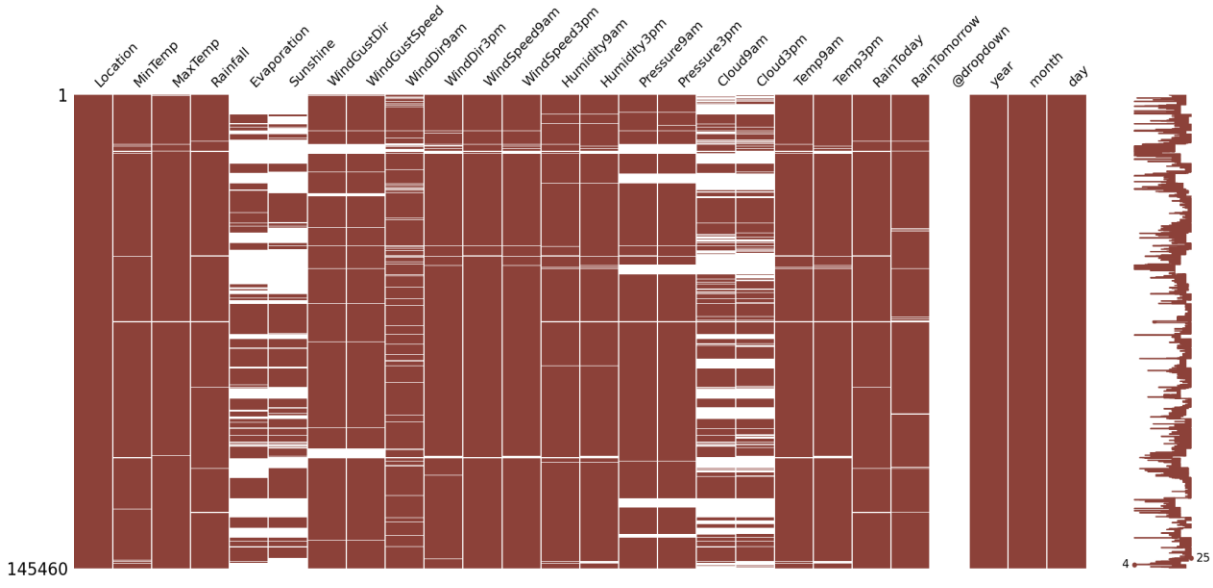
After loading the dataset, missing values were identified and handled appropriately. For numerical columns, missing values were replaced with the mean, while for categorical columns, the most frequent category was used. A missing data heatmap was also visualized to identify null value patterns.

Missing Data Visualisation using Missingo

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

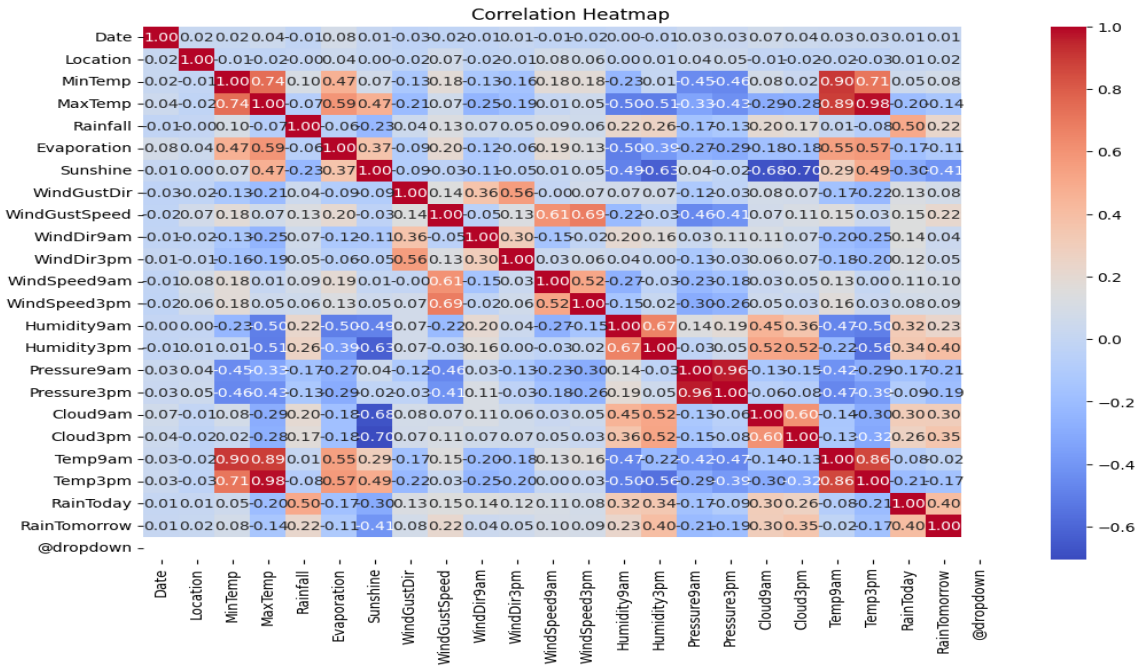
Categorical features were converted into numerical form using encoding techniques. Feature scaling was applied using StandardScaler to normalize numerical features, ensuring uniform contribution of each feature during analysis.

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

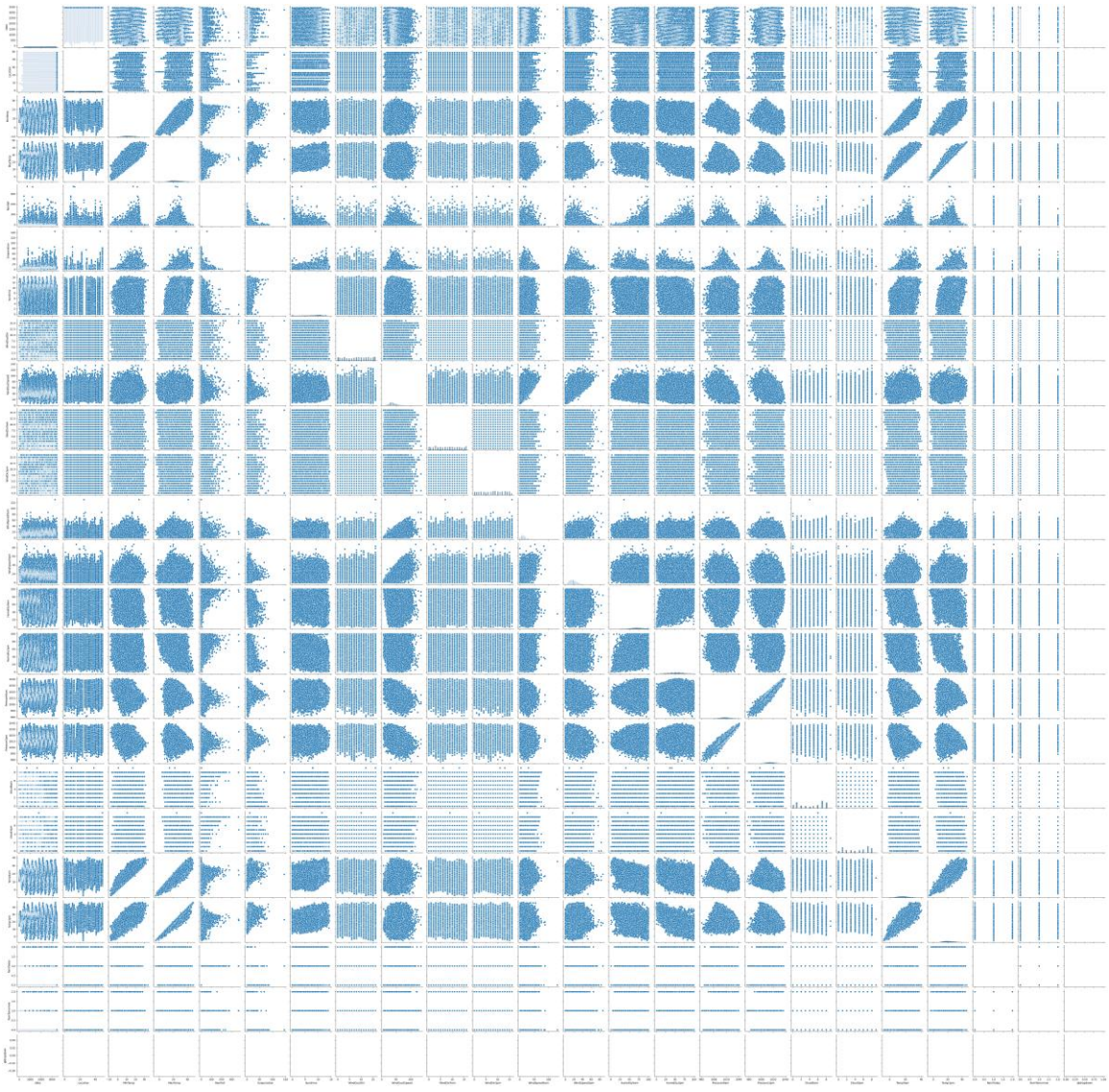
Correlation Heatmap

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.



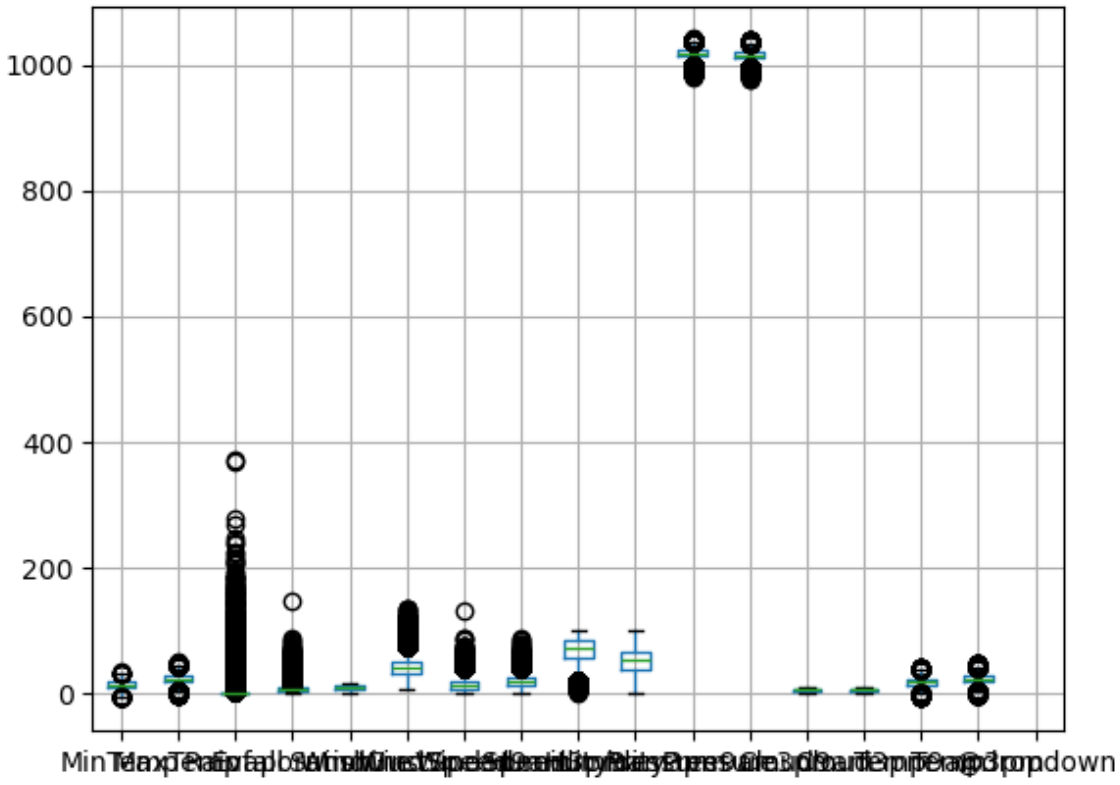
Pair plot

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

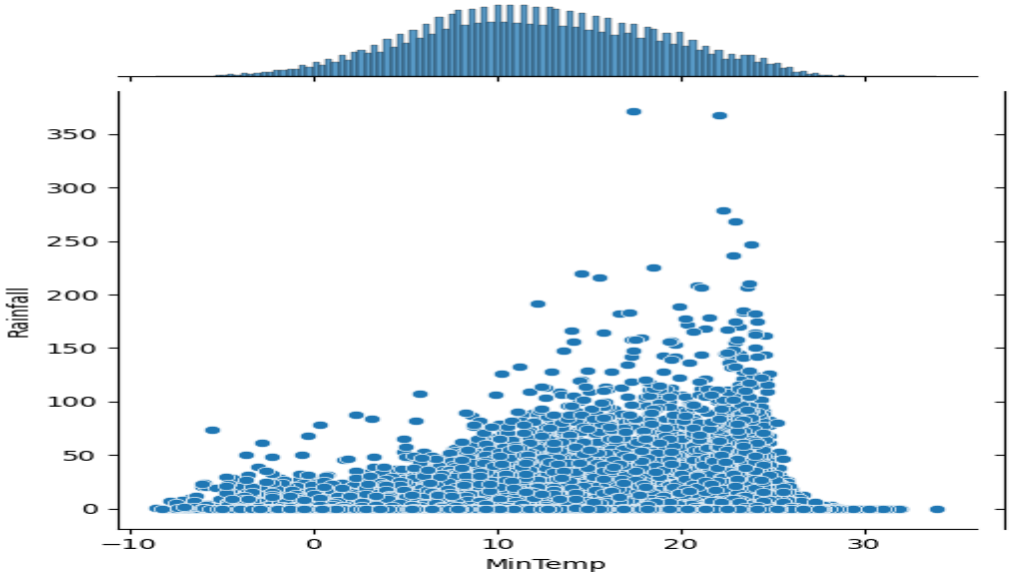
Box-plot is a type of chart often used in explanatory data analysis. Box plots visually show the distribution of numerical data and skewness by displaying the data quartiles (or percentiles) and averages.
Box plots are useful as they show the average score of a data set. The median is the average value 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

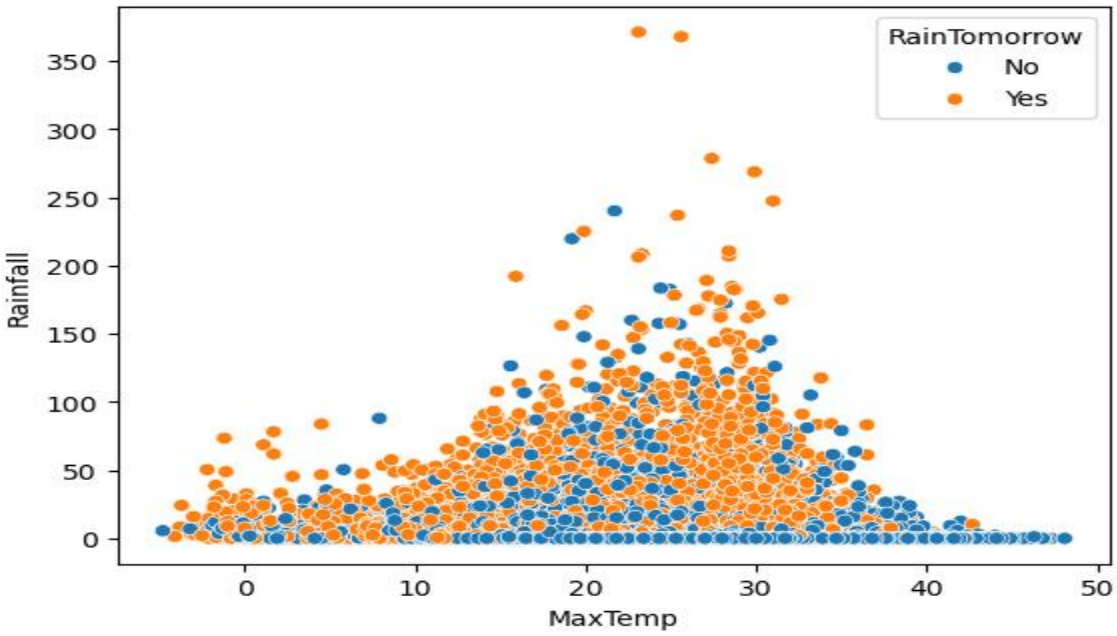


From the above box plot, we infer how the data points are spread and the existence of the outliers

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.

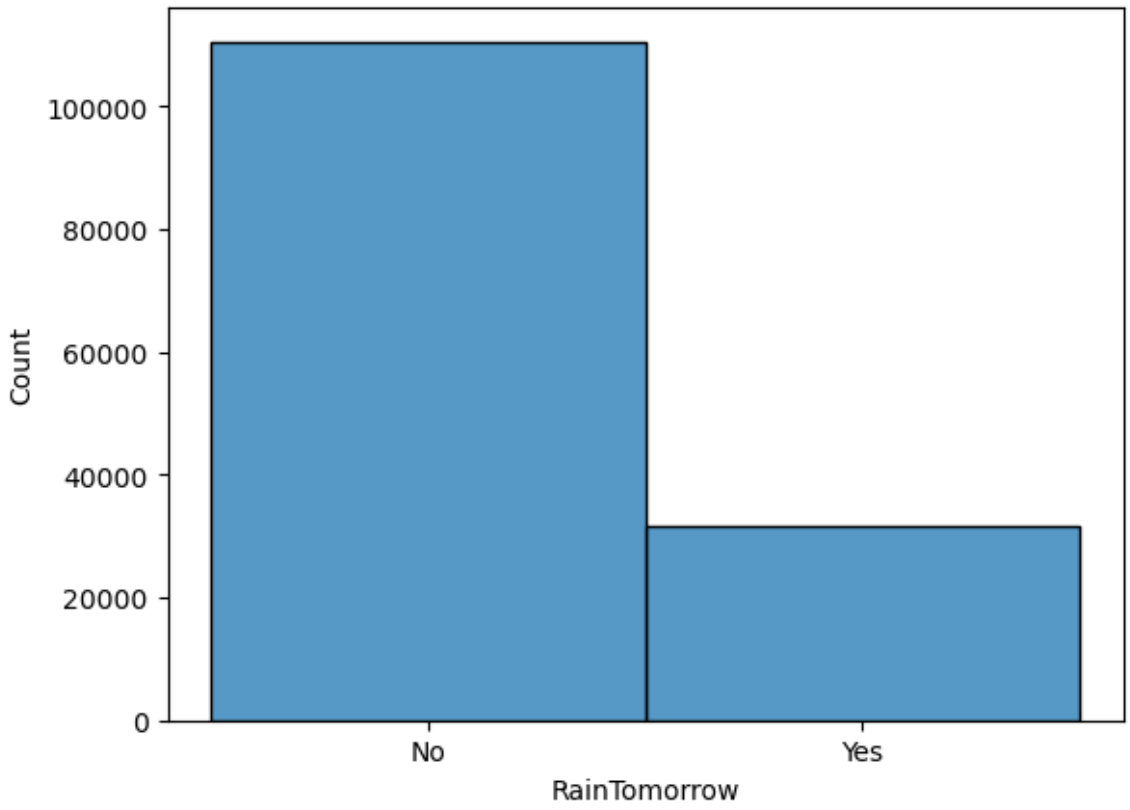




This plot is able to generate the relationship between 3 variables, in the above plot we can see the relation between MaxTemp,Rainfall and class segregation of the data

Hist plot

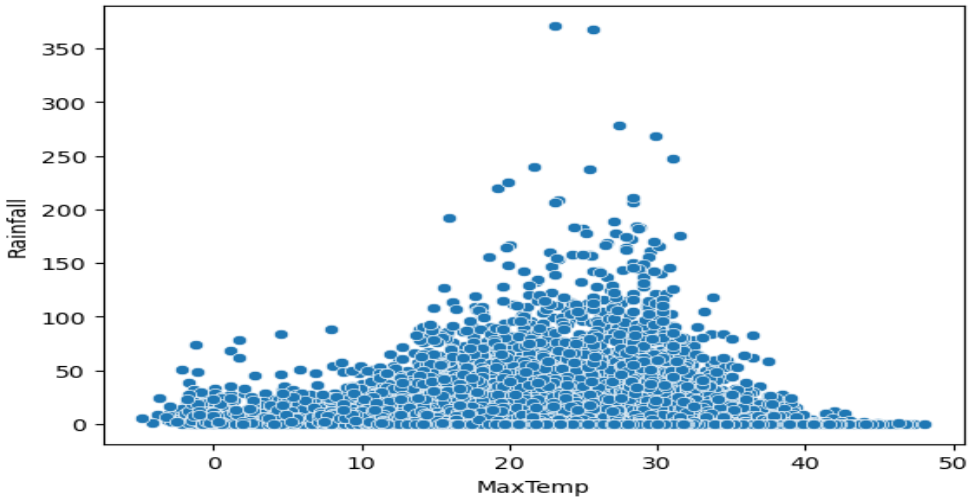
Plot univariate or bivariate histograms to show distributions of datasets. A histogram is a classic visualization tool that represents the distribution of one or more variables by counting the number of observations that fall within discrete bins.



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

Scatter-plot

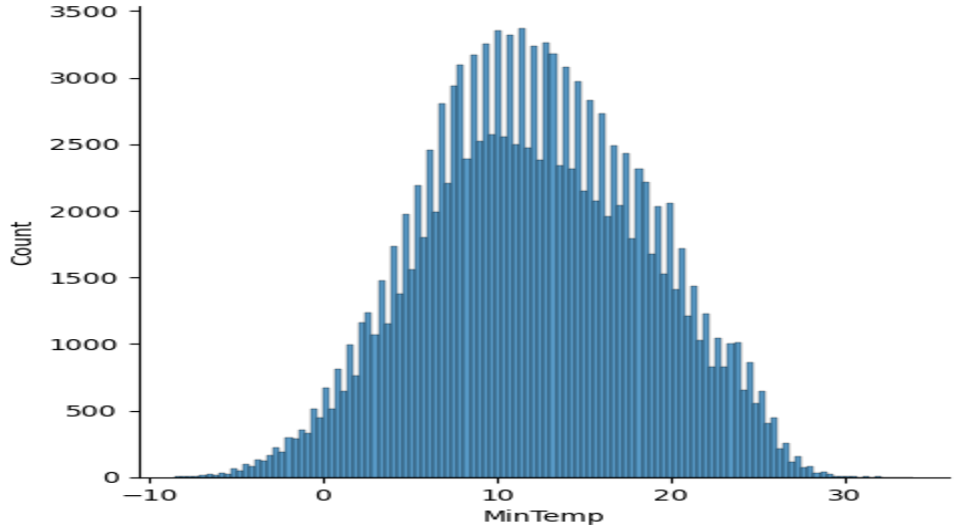
A scatter plot (also called a scatterplot, scatter graph, scatter chart, scattergram, or scatter diagram) is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data



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

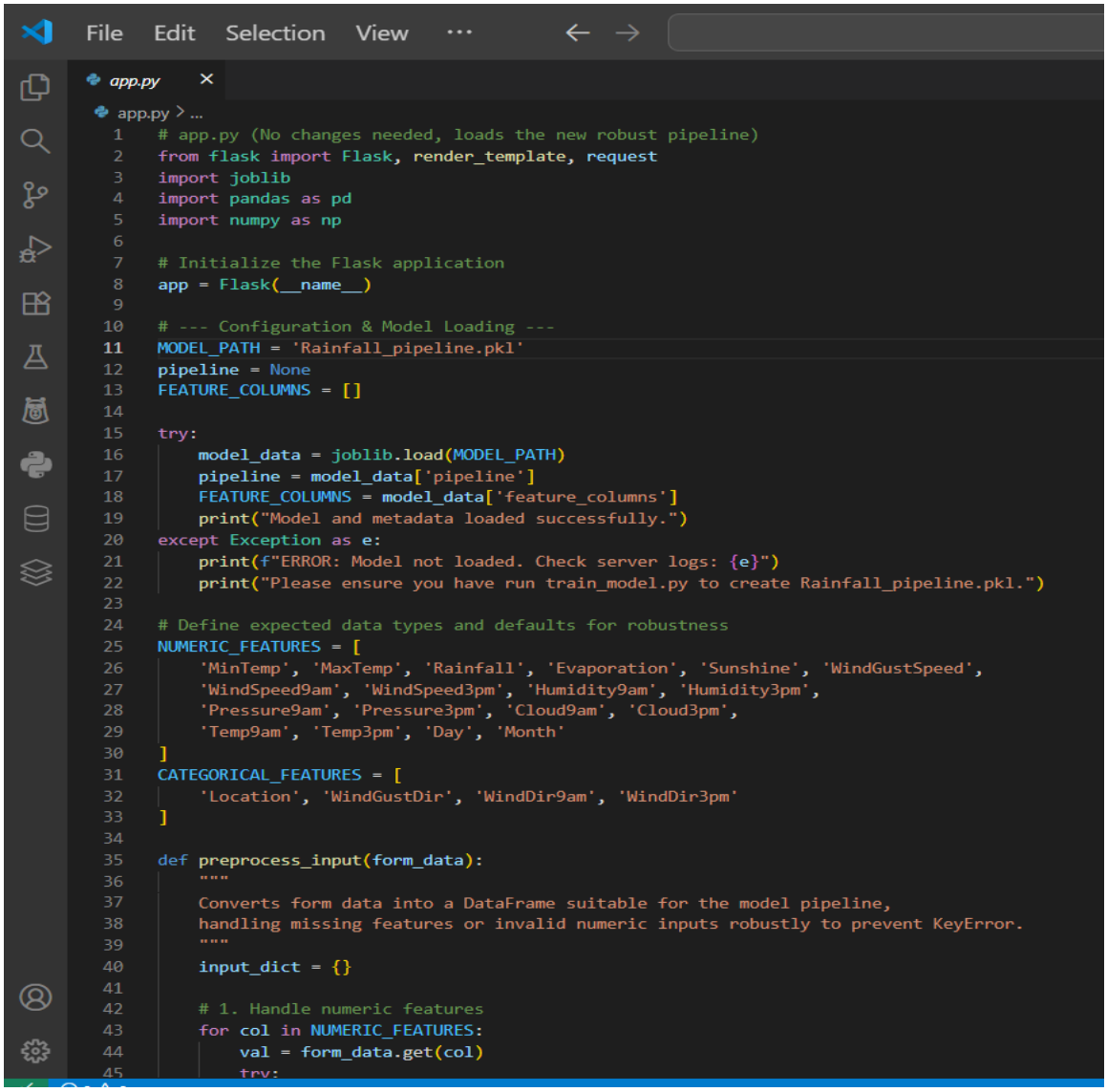
References

• 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



app.py

```python
def preprocess_input(form_data):
        val = form_data.get(col)
        try:
            input_dict[col] = [float(val)]
        except (ValueError, TypeError):
            input_dict[col] = [np.nan]

    # 2. Handle categorical features
    for col in CATEGORICAL_FEATURES:
        input_dict[col] = [form_data.get(col, '')]

    # 3. Handle 'RainToday' feature (1 for 'Yes', 0 otherwise)
    rain_today_str = form_data.get('RainToday')
    input_dict['RainToday'] = [1 if rain_today_str == 'Yes' else 0]

    # 4. Create DataFrame in the exact feature order
    X_new = pd.DataFrame(input_dict)[FEATURE_COLUMNS]

    return X_new

# --- Routes ---

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        if pipeline is None:
            return render_template('noChance.html', message="Model not loaded. Ensure Rainfall_pipeline.pkl exists.")

        try:
            X_new = preprocess_input(request.form)

            # The pipeline uses the correct feature names saved during training
            prediction_proba = pipeline.predict_proba(X_new)[0, 1]
            prediction = 1 if prediction_proba >= 0.5 else 0

            if prediction == 1:
                prob_percent = f"{prediction_proba * 100:.2f}%"
                return render_template('chance.html', probability=prob_percent)
            else:
                return render_template('noChance.html')

        except Exception as e:
            print(f"An unexpected error occurred during prediction: {e}")
            return render_template('noChance.html', message=f"Prediction Error: {e}. Check console for details.")
```

Figure 2: Model.ipynb

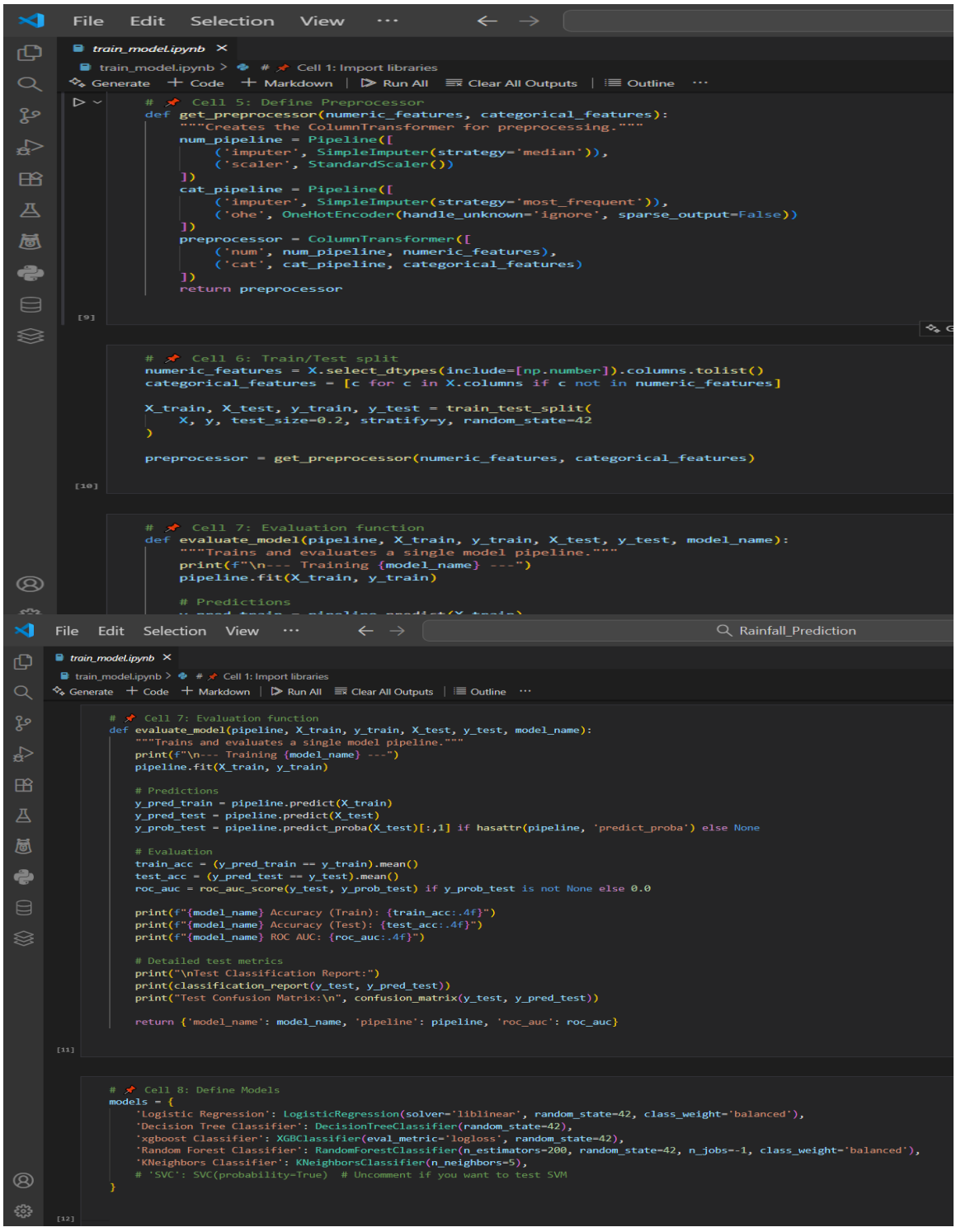
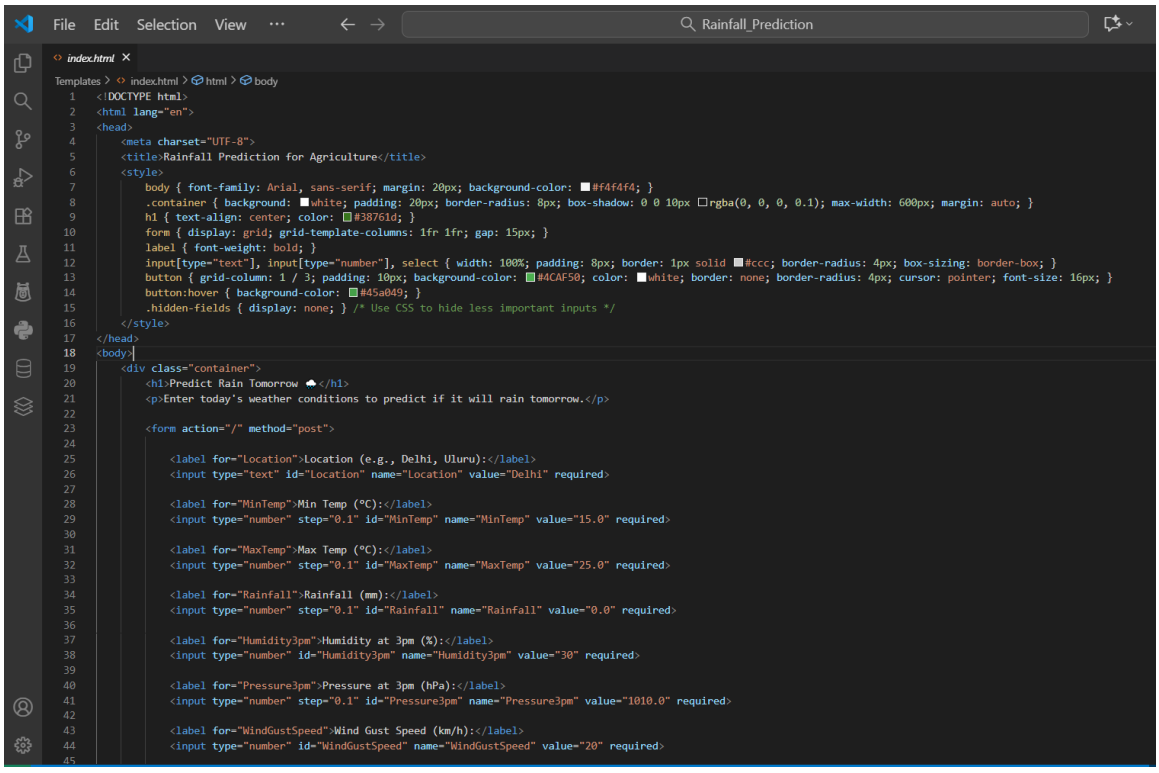
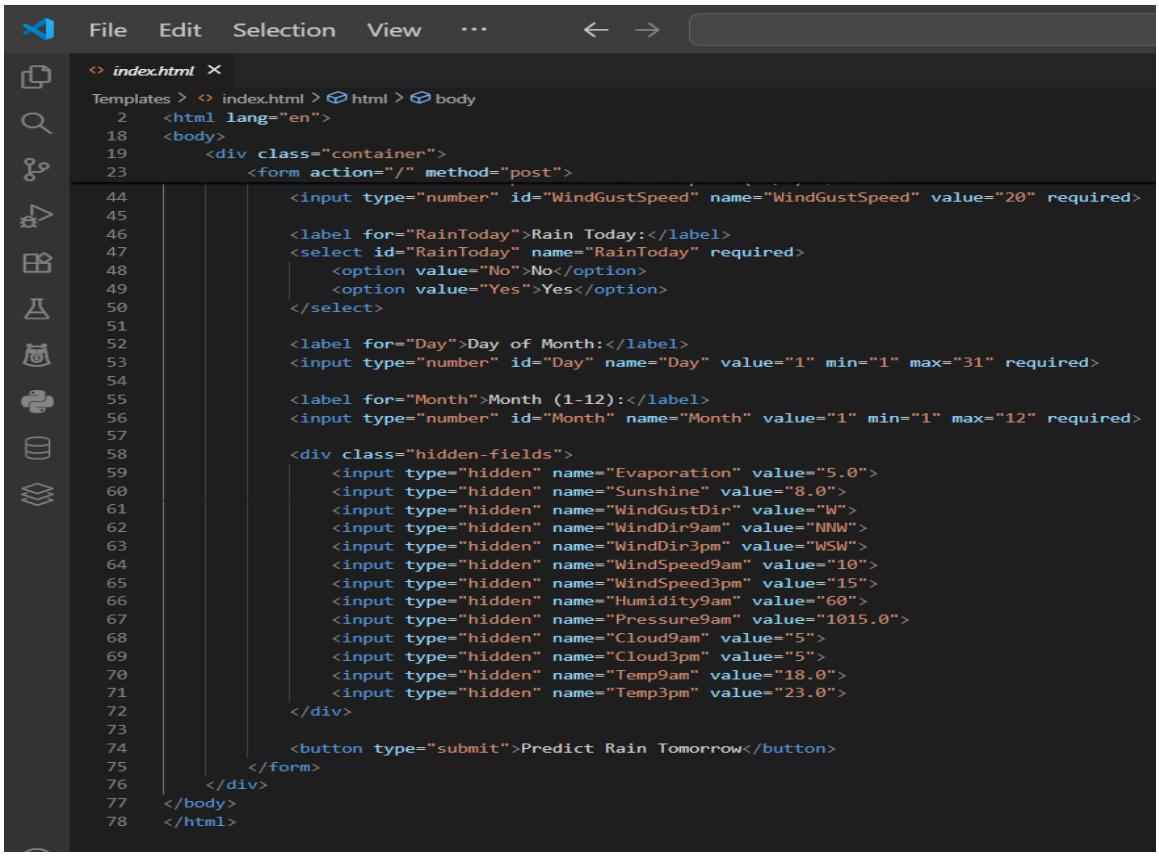
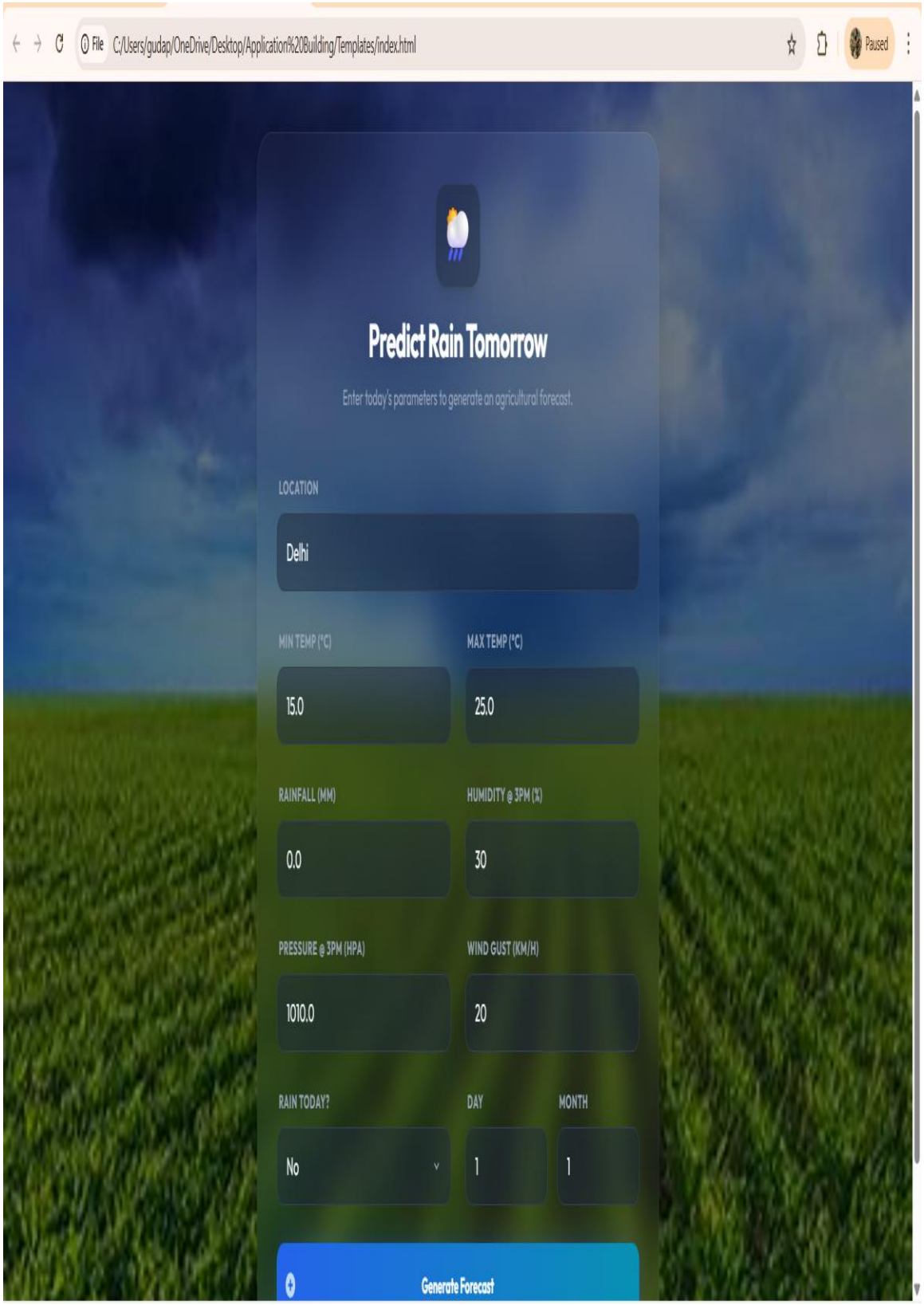


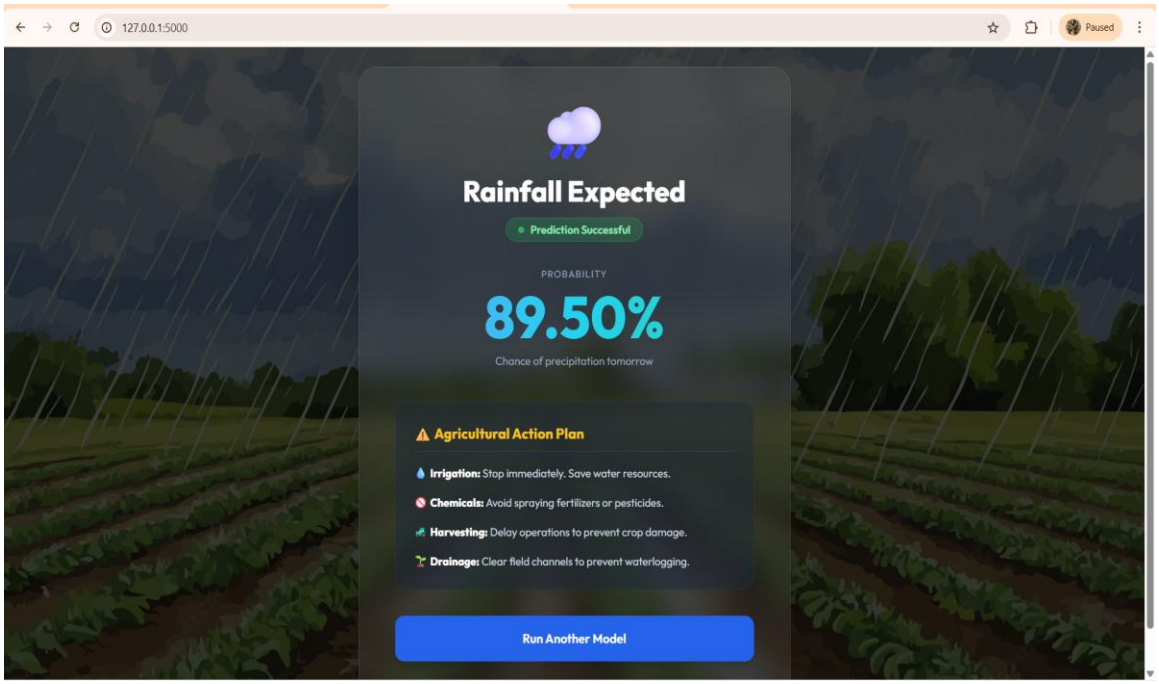
Figure 3: Main Html Script

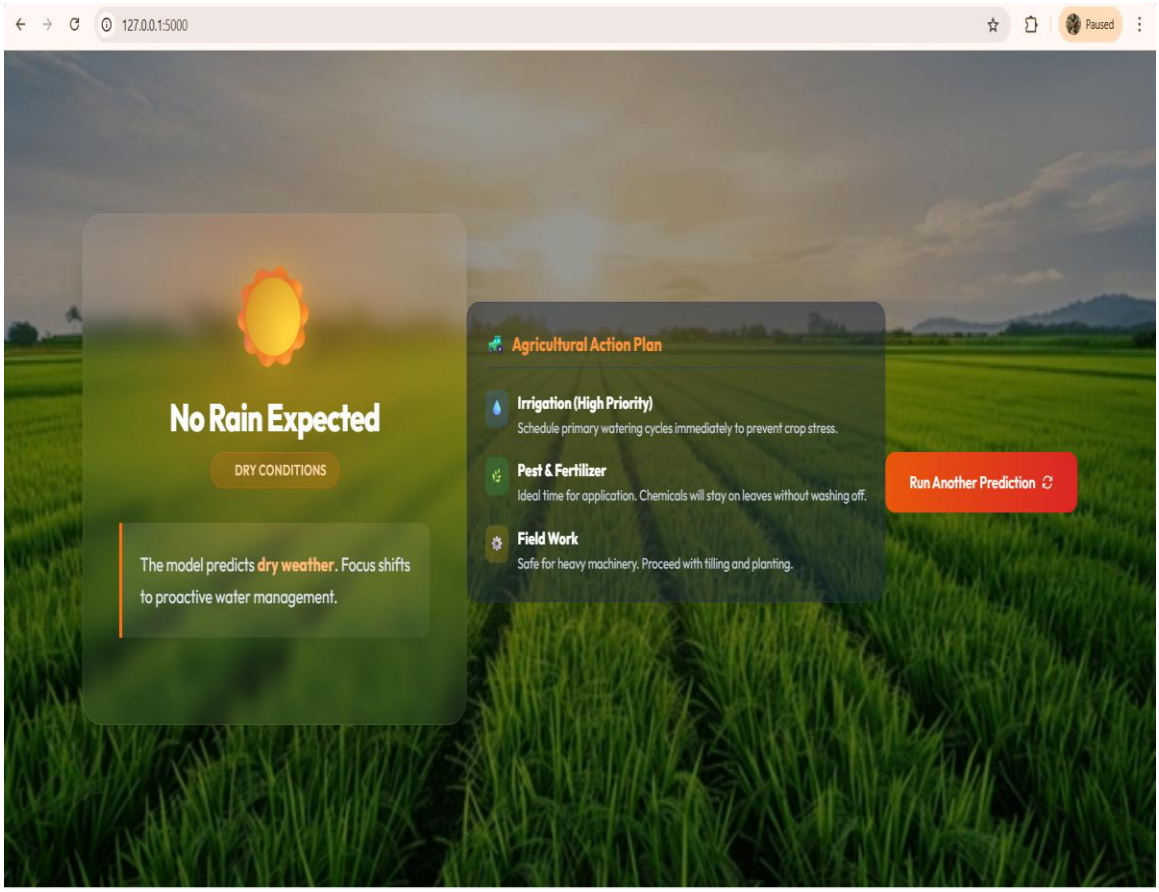




Web Application Index Page




127.0.0.1:5000
Paused
Rainfall Expected
Prediction Successful
PROBABILITY
89.50%
Chance of precipitation tomorrow
Agricultural Action Plan
Irrigation: Stop immediately. Save water resources.
Chemicals: Avoid spraying fertilizers or pesticides.
Harvesting: Delay operations to prevent crop damage.
Drainage: Clear field channels to prevent waterlogging.
Run Another Model


127.0.0.1:5000
Paused
No Rain Expected
DRY CONDITIONS
The model predicts dry weather. Focus shifts to proactive water management.
Agricultural Action Plan
Irrigation (High Priority)
Schedule primary watering cycles immediately to prevent crop stress.
Pest & Fertilizer
Ideal time for application. Chemicals will stay on leaves without washing off.
Field Work
Safe for heavy machinery. Proceed with tilling and planting.
Run Another Prediction