**1.INTRODUCTION**

Rainfall prediction is a critical aspect of meteorology, impacting agriculture, water resource management, and daily life. Traditional methods of weather prediction, which rely on statistical and numerical models, have limitations in handling large datasets and complex patterns. Machine learning (ML) offers a promising approach to improve the accuracy and efficiency of rainfall prediction by leveraging vast amounts of historical weather data to identify patterns and make predictions.

**1.1 Problem Statement**

The primary goal of this project is to develop a machine learning model that accurately predicts rainfall. The model will be trained using historical weather data and evaluated on its performance in predicting rainfall amounts.

**1.2 Existing System**

Agriculture is the strength of our Indian economy. Farmer only depends upon monsoon to be their cultivation. The good crop productivity needs good soil, fertilizer and also good climate. Weather forecasting is the very important requirement of the each farmer. Due to the sudden changes in climate/weather, The people are suffered economically and physically. Weather prediction is one of the challenging problems in current state. The main motivation of this paper to predict the weather using various data mining techniques. Such as classification, clustering, decision tree and also neural networks. Weather related information is also called the meteorological data. In this paper the most commonly used weather parameters are rainfall, wind speed, temperature and cold

**1.3 Proposed System**

The proposed system employs machine learning techniques, specifically regression models, to predict rainfall. By utilizing large datasets and advanced algorithms, the system aims to provide more accurate and timely rainfall predictions.

. **2.SOFTWARE REQUIREMENTS SPECIFICATIONS**

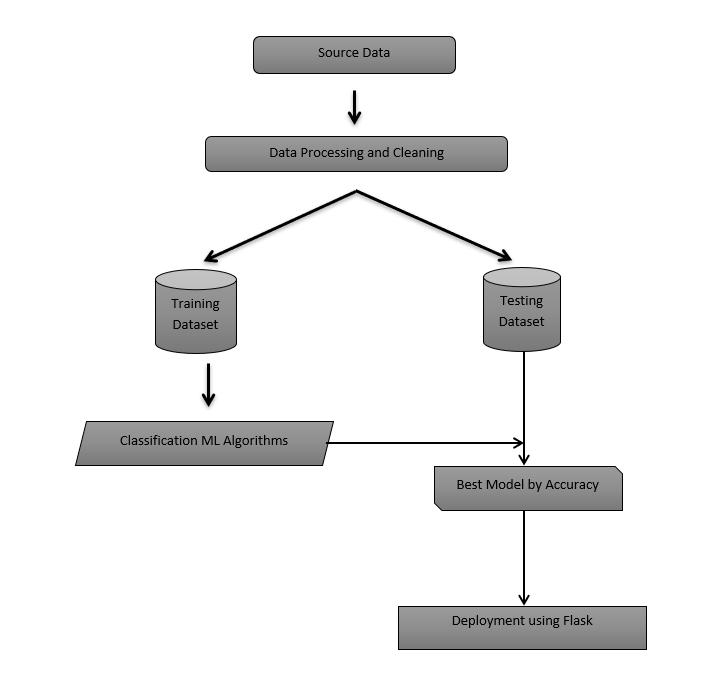
**2.1 Software Requirements**

* O/S : Windows 11.
* Language : Python
* Front End : Anaconda Navigator – Spyder

**2.2 Hardware Requirements**

* System : Pentium IV 2.4 GHz
* Ram : 4GB

**2.3 System Architecture / Flow Chart:**



**3**.**DESIGN & IMPLEMENTATION**

**3.1 Environmental Setup:**

1. Install Visual Studio Code.
2. Install Python from the official website.
3. Install required packages:

pip install pandas numpy scikit-learn matplotlib seaborn

1. Set up the development environment and ensure all libraries are properly installed.

**3.2 Implementation:**

**Step 1: Data Collection and Preprocessing**

* Collect historical weather data from reliable sources.
* Clean and preprocess the data, handling missing values and outliers.

**Step 2: Feature Selection**

* Select relevant features that influence rainfall, such as temperature, humidity, wind speed, and pressure.

**Step 3: Model Development**

* Split the data into training and testing sets.
* Develop regression models (e.g., Linear Regression, Random Forest, Support Vector Machine) to predict rainfall.

**Step 4: Model Training and Evaluation**

* Train the models using the training dataset.
* Evaluate the models based on metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared.

**Step 5: Model Deployment**

* Deploy the best-performing model for real-time rainfall prediction.

#### 4. Results & Discussion

**4.1 Results:**

1. **Data Visualization:**
   * Visualize the weather data to understand trends and correlations.
   * Example: A plot showing the relationship between temperature and rainfall.
2. **Model Performance:**
   * Present the performance metrics of different models.
   * Example: A table comparing the MAE, MSE, and R-squared values of the models.
3. **Predictions:**
   * Display sample predictions from the deployed model.
   * Example: A chart showing actual vs. predicted rainfall.

**4.2 Test Cases:**

* Develop and document test cases to validate the system’s functionality.
* Example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| Fig-1: logistic Regression roc curve |  |  |  |  |

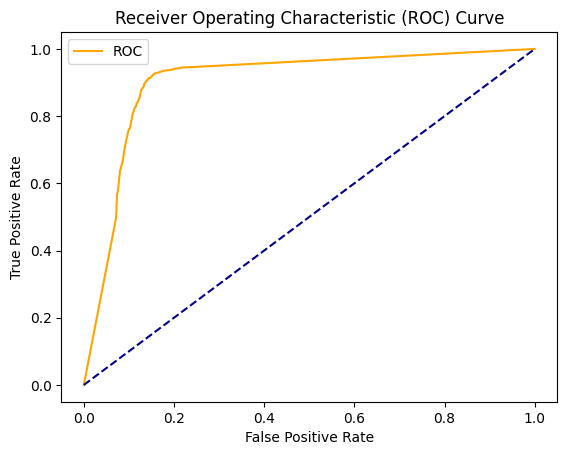


Fig-2: Decesion tree roc curve

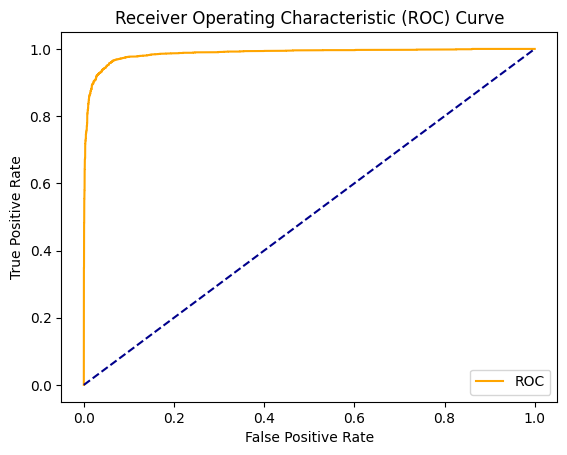


Fig-3: Randomforest roc curve

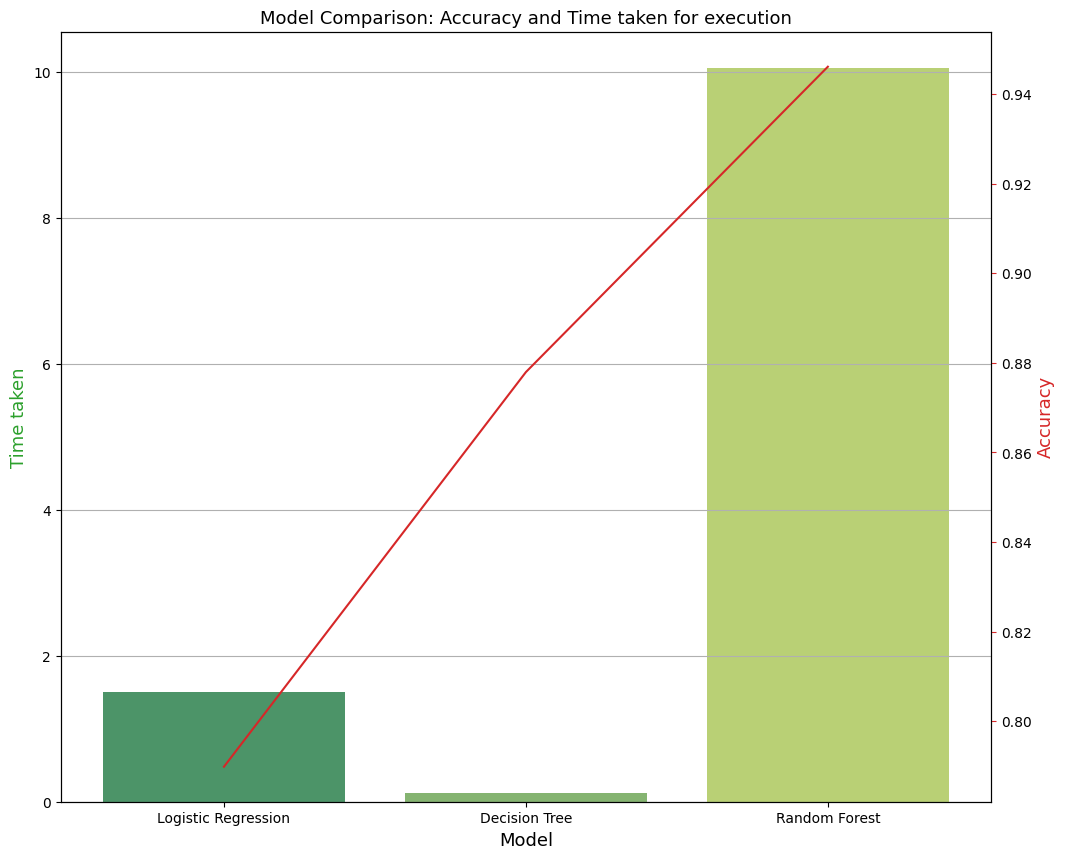


Fig-4: model comparision:Acurray and time taken for execution

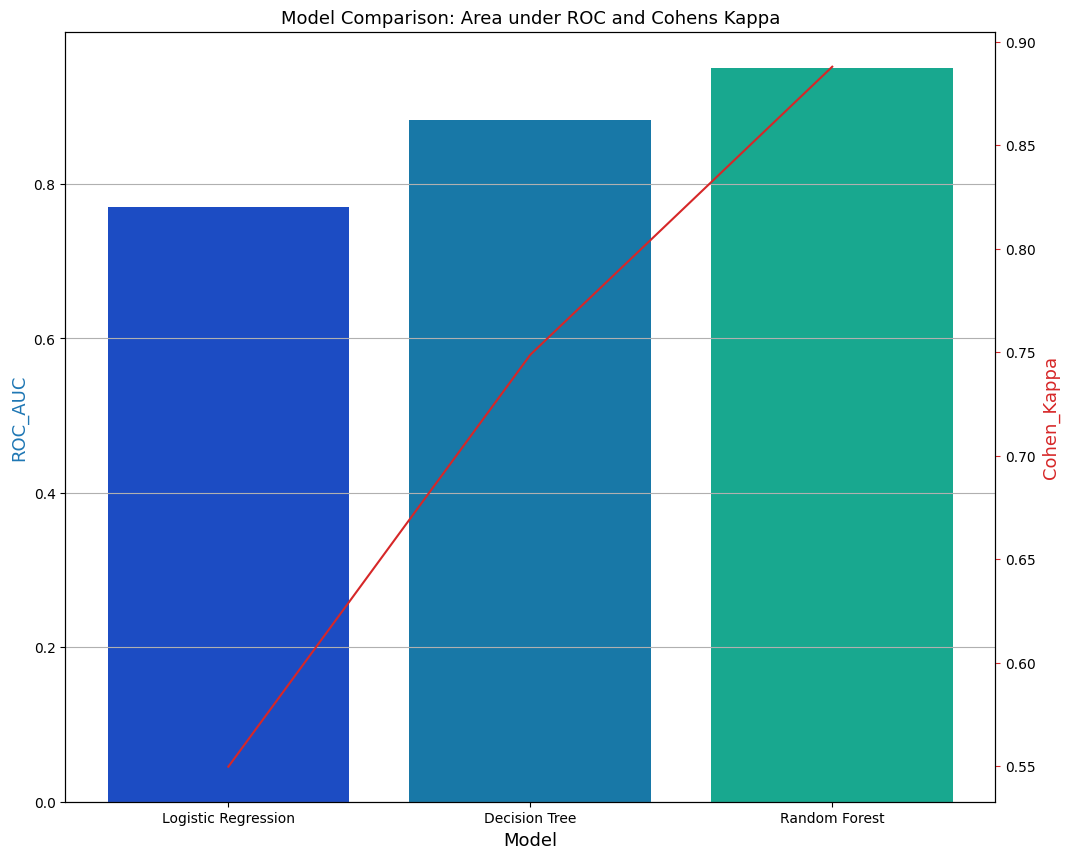
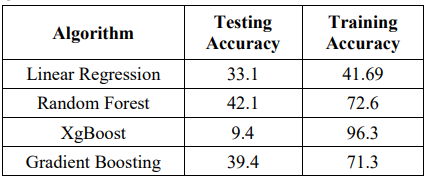


Fig-5: Model Comparision:Area under Roc and cohens kappa



**5.CONCLUSION AND FUTURE ENHANCEMENTS**

**5.1 Conclusion**

The machine learning-based rainfall prediction system demonstrates significant improvements over traditional methods. By leveraging historical weather data and advanced algorithms, the system provides accurate and timely rainfall predictions, benefiting various sectors. Randomforest Algorithm gives the best accurate result.

**5.2 Future Enhancements**

 **Incorporate Real-Time Data:** Integrate real-time weather data to continuously update the model.

 **Enhance Model Accuracy:** Explore advanced algorithms and deep learning techniques to further improve prediction accuracy.

 **User Interface:** Develop a user-friendly interface for easy access and visualization of predictions.

 **Geographic Scalability:** Expand the model to predict rainfall in different geographic regions with varied climatic conditions.

**REFERENCES**

* <https://www.w3schools.com/python/python_ml_getting_started.asp>
* <https://www.geeksforgeeks.org>
* <https://www.tutorialspoint.com>

**APPENDIX: PSEUDO CODE**

import pandas as pd

full\_data = pd.read\_csv('weatherAUS.csv')

full\_data.head()

full\_data['Date'] = pd.to\_datetime(full\_data['Date'])

full\_data['year'] = full\_data['Date'].dt.year

full\_data['month'] = full\_data['Date'].dt.month

full\_data['day'] = full\_data['Date'].dt.day

full\_data.drop(['Date'], axis = 1,inplace=True)

full\_data.head()

full\_data.shape

full\_data.info()

full\_data['RainToday'].replace({'No': 0, 'Yes': 1},inplace = True)

full\_data['RainTomorrow'].replace({'No': 0, 'Yes': 1},inplace = True)

full\_data.head()

import matplotlib.pyplot as plt

fig = plt.figure(figsize = (20,5))

ax=full\_data.RainTomorrow.value\_counts(normalize = True).plot(kind='bar', color= ['skyblue','navy'], alpha = 0.9, rot=0)

plt.title('RainTomorrow Indicator No(0) and Yes(1) in the Imbalanced Dataset')

for p in ax.patches:

    ax.annotate(str(round(p.get\_height(),2)), (p.get\_x() \* 1.01 , p.get\_height() \* 1.01))

plt.show()

from sklearn.utils import resample

no = full\_data[full\_data.RainTomorrow == 0]

yes = full\_data[full\_data.RainTomorrow == 1]

yes\_oversampled = resample(yes, replace=True, n\_samples=len(no), random\_state=42)

oversampled = pd.concat([no, yes\_oversampled])

fig = plt.figure(figsize = (20,5))

ax=oversampled.RainTomorrow.value\_counts(normalize = True).plot(kind='bar', color= ['skyblue','navy'], alpha = 0.9, rot=0)

plt.title('RainTomorrow Indicator No(0) and Yes(1) after Oversampling (Balanced Dataset)')

for p in ax.patches:

    ax.annotate(str(round(p.get\_height(),2)), (p.get\_x() \* 1.01 , p.get\_height() \* 1.01))

plt.show()

# Logistic Regression

from sklearn.linear\_model import LogisticRegression

params\_lr = {'penalty': 'l1', 'solver':'liblinear'}

model\_lr = LogisticRegression(\*\*params\_lr)

model\_lr, accuracy\_lr, roc\_auc\_lr, coh\_kap\_lr, tt\_lr = run\_model(model\_lr, X\_train, y\_train, X\_test, y\_test)

# Decision Tree

from sklearn.tree import DecisionTreeClassifier

params\_dt = {'max\_depth': 16,

             'max\_features': "sqrt"}

model\_dt = DecisionTreeClassifier(\*\*params\_dt)

model\_dt, accuracy\_dt, roc\_auc\_dt, coh\_kap\_dt, tt\_dt = run\_model(model\_dt, X\_train, y\_train, X\_test, y\_test)

# Random Forest

from sklearn.ensemble import RandomForestClassifier

params\_rf = {'max\_depth': 16,

             'min\_samples\_leaf': 1,

             'min\_samples\_split': 2,

             'n\_estimators': 100,

             'random\_state': 42}

model\_rf = RandomForestClassifier(\*\*params\_rf)

model\_rf, accuracy\_rf, roc\_auc\_rf, coh\_kap\_rf, tt\_rf = run\_model(model\_rf, X\_train, y\_train, X\_test, y\_test)

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

import itertools

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.ensemble import RandomForestClassifier

from mlxtend.classifier import EnsembleVoteClassifier

from mlxtend.plotting import plot\_decision\_regions

value = 1.80

width = 0.90

clf1 = LogisticRegression(random\_state=42)

clf2 = DecisionTreeClassifier(random\_state=42)

clf3 = MLPClassifier(random\_state=42, verbose = 0)

clf4 = RandomForestClassifier(random\_state=42)

clf5 = lgb.LGBMClassifier(random\_state=42, verbose = 0)

X\_list = MiceImputed[["Sunshine", "Humidity9am", "Cloud3pm"]] #took only really important features

X = np.asarray(X\_list, dtype=np.float32)

y\_list = MiceImputed["RainTomorrow"]

y = np.asarray(y\_list, dtype=np.int32)

# Plotting Decision Regions

gs = gridspec.GridSpec(3,3)

fig = plt.figure(figsize=(18, 14))

labels = ['Logistic Regression',

          'Decision Tree',

          'Random Forest',

          ]

for clf, lab, grd in zip([clf1, clf2, clf3, clf4, clf5],

                         labels,

                         itertools.product([0, 1, 2],

                         repeat=2)):

    clf.fit(X, y)

    ax = plt.subplot(gs[grd[0], grd[1]])

    fig = plot\_decision\_regions(X=X, y=y, clf=clf,

                                filler\_feature\_values={2: value},

                                filler\_feature\_ranges={2: width},

                                legend=2)

    plt.title(lab)

plt.show()

fig, ax3 = plt.subplots(figsize=(12,10))

ax3.set\_title('Model Comparison: Area under ROC and Cohens Kappa', fontsize=13)

color = 'tab:blue'

ax3.grid()

ax3.set\_xlabel('Model', fontsize=13)

ax3.set\_ylabel('ROC\_AUC', fontsize=13, color=color)

ax4 = sns.barplot(x='Model', y='ROC\_AUC', data = data, palette='winter')

ax3.tick\_params(axis='y')

ax4 = ax3.twinx()

color = 'tab:red'

ax4.set\_ylabel('Cohen\_Kappa', fontsize=13, color=color)

ax4 = sns.lineplot(x='Model', y='Cohen\_Kappa', data = data, sort=False, color=color)

ax4.tick\_params(axis='y', color=color)

plt.show()