Project Documentation and Demonstration

Project Proposal: Enhancing Rainfall Prediction for Sustainable Agriculture and Water Resource Management

Project_Initialization_and_Planning_phase

1. Introduction

Accurate rainfall prediction is pivotal for effective agricultural planning and water resource management, especially in regions like Maharashtra, India, where monsoon variability significantly impacts crop yields and water availability. Leveraging machine learning (ML) techniques offers a promising avenue to improve the precision of rainfall forecasts, thereby aiding farmers and policymakers in making informed decisions.

2. Objectives

- Develop a machine learning-based model to predict daily rainfall with high accuracy.
- Integrate the model into a user-friendly platform for stakeholders in agriculture and water management.
- Evaluate the model's performance using standard metrics to ensure reliability and robustness.

3. Methodology

3.1 Data Collection and Preprocessing

- Data Sources: Historical weather data, including parameters like temperature, humidity, wind speed, and past rainfall records, will be collected from reputable meteorological departments.
- Preprocessing Steps:
    - Handling missing values through imputation techniques.

  - Normalizing data to ensure uniformity.
  - Encoding categorical variables if present.

3.2 Model Development

- Algorithm Selection: Based on preliminary analyses, algorithms such as Random Forest, Gradient Boosting, and Long Short-Term Memory (LSTM) networks will be considered due to their proven efficacy in time-series forecasting.
- Training and Validation:
  - The dataset will be split into training and testing subsets.
  - Cross-validation techniques will be employed to prevent overfitting.
  - Hyperparameter tuning will be conducted to optimize model performance.

3.3 Performance Evaluation

- Metrics:
  - Mean Absolute Error (MAE)
  - Root Mean Square Error (RMSE)
  - R-squared (R^2) score
- These metrics will provide insights into the model's accuracy and reliability.

---

4. Implementation Plan

Phase 1: Data Acquisition and Cleaning

- Gather historical weather data.
- Perform data cleaning and preprocessing.

Phase 2: Model Development

- Train multiple ML models.

- Evaluate and select the best-performing model.

Phase 3: Integration and Deployment

- Develop a user interface for stakeholders.
- Integrate the model into the platform.
- Deploy the system for pilot testing.

Phase 4: Feedback and Iteration

- Collect feedback from users.
- Refine the model and interface based on insights.

5. Expected Outcomes

- A robust ML model capable of accurately predicting daily rainfall.
- A user-friendly platform accessible to farmers and water resource managers.
- Enhanced decision-making capabilities leading to optimized agricultural practices and water usage.

6. Future Enhancements

- Incorporate Real-time Data: Integrate live weather data feeds to provide up-to-date forecasts.
- Expand Geographical Scope: Adapt the model for use in other regions with similar climatic conditions.
- Integrate with IoT Devices: Utilize sensors for real-time soil moisture and atmospheric data to further refine predictions.

By implementing this project, we aim to empower stakeholders in agriculture and water management with precise rainfall forecasts, facilitating proactive and informed decision-making that promotes sustainability and resilience against climatic uncertainties.

2)Data Collection and Preprocessing Phase

Data Collection :

We have used the following data for the model to train on :

https://docs.google.com/spreadsheets/d/1RA2OO0LZTeQykl_mvnensAjp6LM4YzWI1Tz0SUG5-Ao/edit?gid=121883362#gid=121883362

Data Quality Report :

Analyzing the data:

```python
# Analyze the data
data.head()
```

[5] Python

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	...	Humidity3pm	Pressure9am	Pressure3pm	Cloud
0	2008-12-01	Delhi	13.4	22.9	0.6	NaN	NaN	W	44.0	W	...	22.0	1007.7	1007.1	
1	2008-12-02	Delhi	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	...	25.0	1010.6	1007.8	I
2	2008-12-03	Delhi	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	...	30.0	1007.6	1008.7	I
3	2008-12-04	Delhi	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	...	16.0	1017.6	1012.8	I
4	2008-12-05	Delhi	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	...	33.0	1010.8	1006.0	

5 rows × 24 columns

```python
data.describe()
```

[6] Python

	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9a
count	143975.000000	144199.000000	142199.000000	82670.000000	75625.000000	135197.000000	143693.000000	142398.000000	142806.000000	140953.000000	130395.000
mean	12.194034	23.221348	2.360918	5.468232	7.611178	40.035230	14.043426	18.662657	68.880831	51.539116	1017.6499
std	6.398495	7.119049	8.478060	4.193704	3.785483	13.607062	8.915375	8.809800	19.029164	20.795902	7.1065
min	-8.500000	-4.800000	0.000000	0.000000	0.000000	6.000000	0.000000	0.000000	0.000000	0.000000	980.5000
25%	7.600000	17.900000	0.000000	2.600000	4.800000	31.000000	7.000000	13.000000	57.000000	37.000000	1012.9000
50%	12.000000	22.600000	0.000000	4.800000	8.400000	39.000000	13.000000	19.000000	70.000000	52.000000	1017.6000
75%	16.900000	28.200000	0.800000	7.400000	10.600000	48.000000	19.000000	24.000000	83.000000	66.000000	1022.4000
max	33.900000	48.100000	371.000000	145.000000	14.500000	135.000000	130.000000	87.000000	100.000000	100.000000	1041.0000

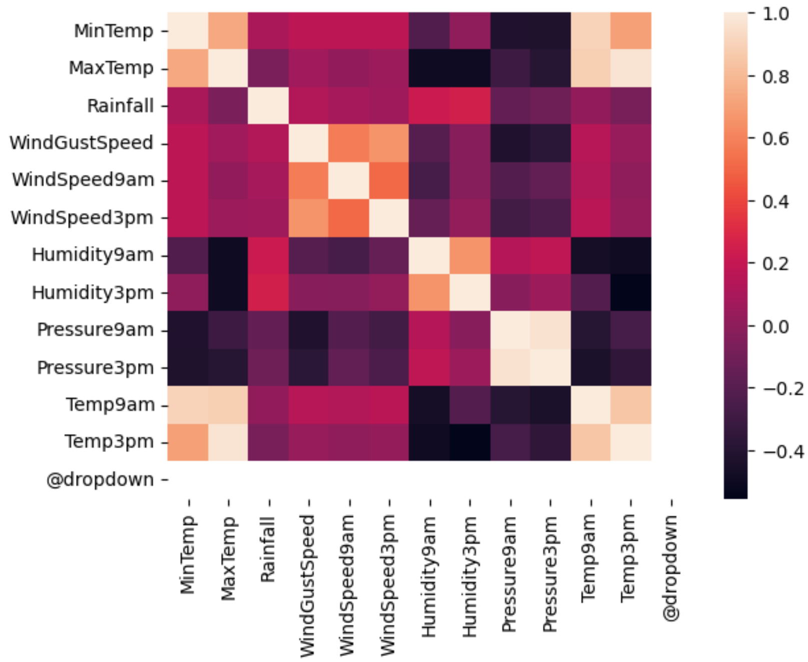
```
[7]
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145460 entries, 0 to 145459
Data columns (total 24 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Date           145460 non-null  object 
 1   Location       145460 non-null  object 
 2   MinTemp        143975 non-null  float64
 3   MaxTemp        144199 non-null  float64
 4   Rainfall       142199 non-null  float64
 5   Evaporation    82670 non-null   float64
 6   Sunshine       75625 non-null   float64
 7   WindGustDir    135134 non-null  object 
 8   WindGustSpeed  135197 non-null  float64
 9   WindDir9am     134894 non-null  object 
 10  WindDir3pm     141232 non-null  object 
 11  WindSpeed9am   143693 non-null  float64
 12  WindSpeed3pm   142398 non-null  float64
 13  Humidity9am    142806 non-null  float64
 14  Humidity3pm    140953 non-null  float64
 15  Pressure9am    130395 non-null  float64
 16  Pressure3pm    130432 non-null  float64
 17  Cloud9am       89572 non-null   float64
 18  Cloud3pm       86102 non-null   float64
 19  Temp9am        143693 non-null  float64
...
 22  RainTomorrow   142207 non-null  object 
 23  @dropdown      0 non-null       float64
dtypes: float64(17), object(7)
memory usage: 26.6+ MB
```

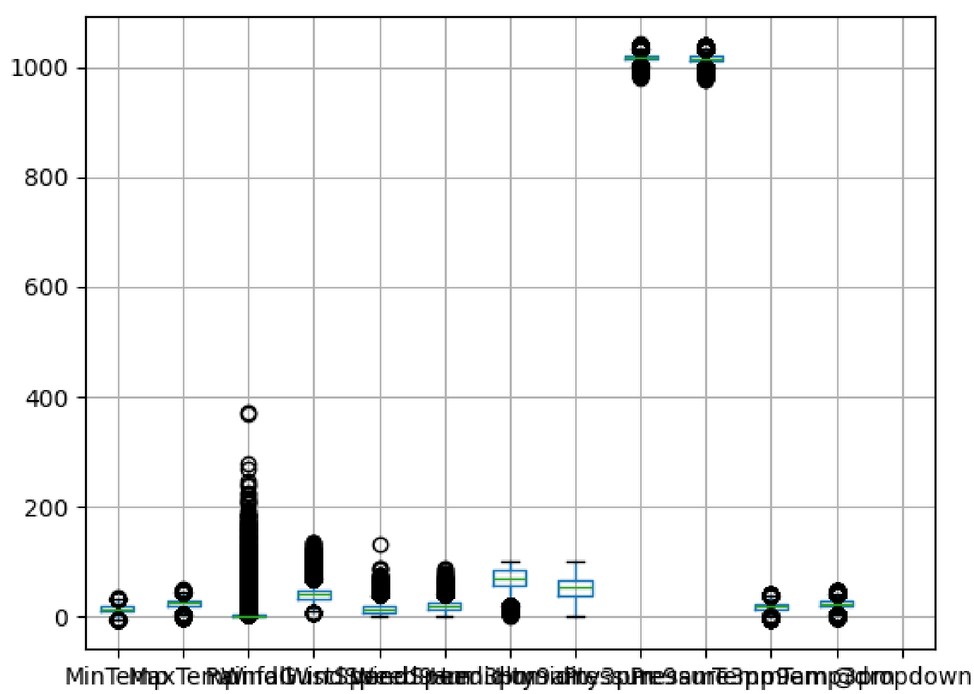
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Preprocessing Report : (Visualisation Report)

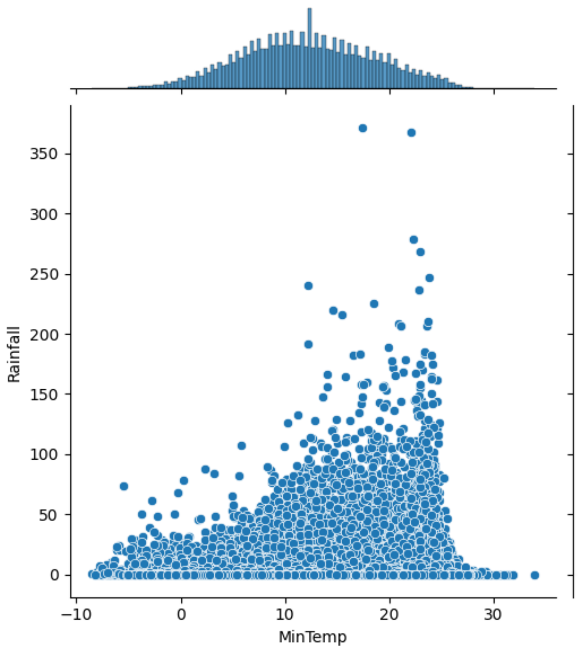
1.The HeatMap of the data



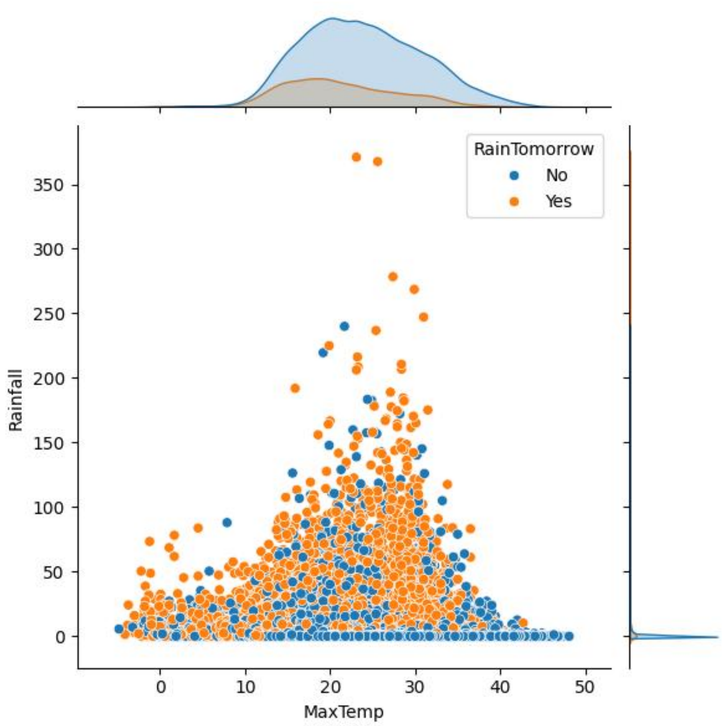
2. The Boxplot



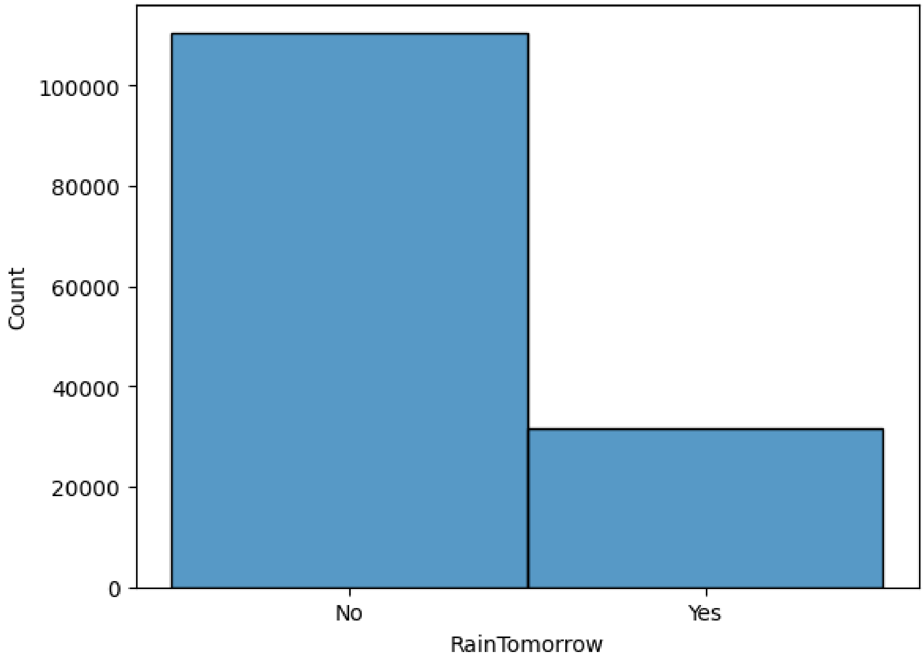
3. The Jointplot - 1



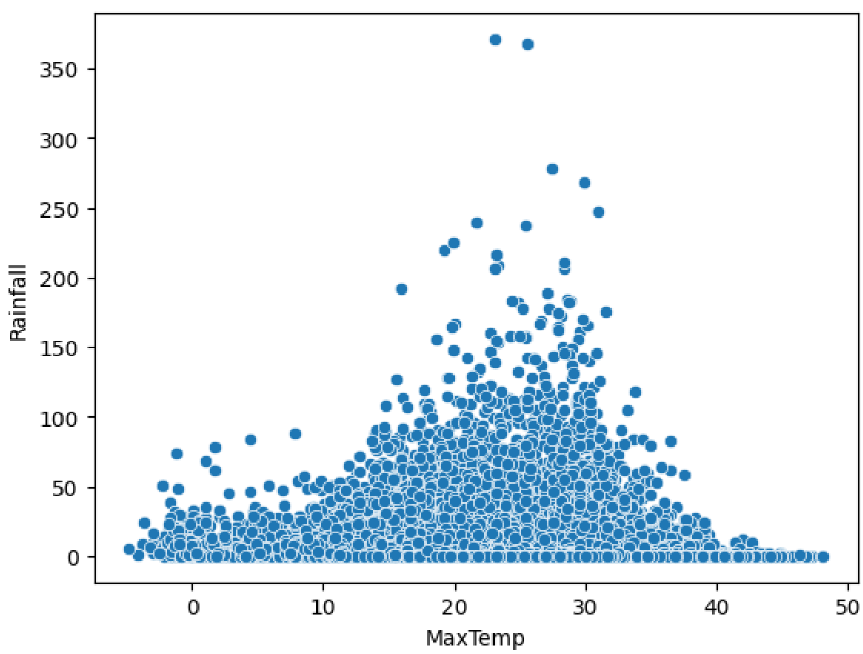
4. The Jointplot - 2



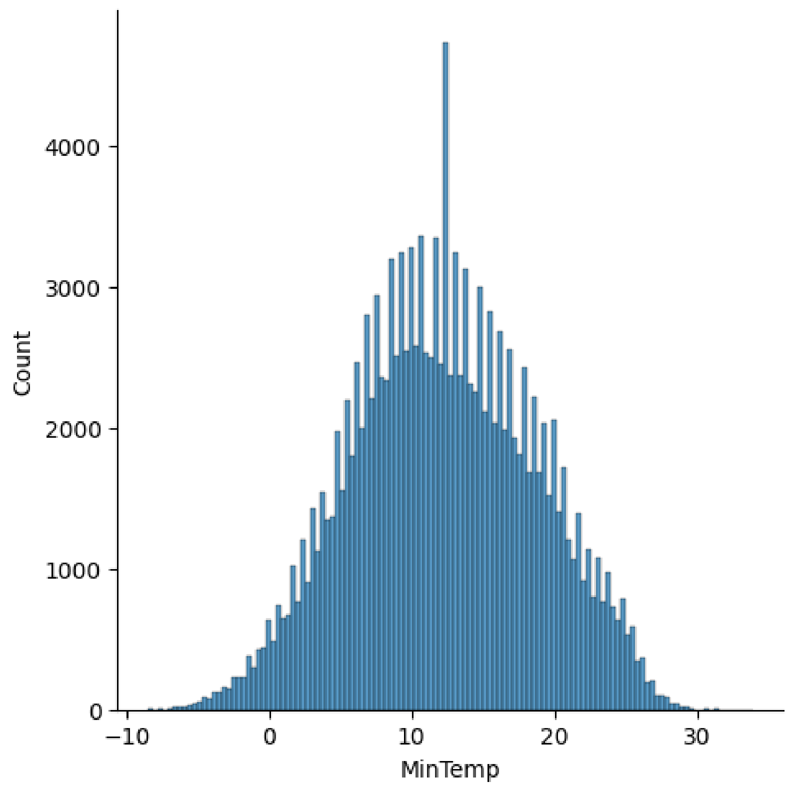
5. The Histogram



6. The Scatter Plot



7. The Distribution Plot



Splitting the data:

```
# Splitting x and y values
y = data['RainTomorrow']
x = data.drop('RainTomorrow',axis=1)
```
[27]

```
from sklearn.preprocessing import StandardScaler
```
[28]

```
# Splitting x and y values
y = data['RainTomorrow']
x = data.drop('RainTomorrow',axis=1)
```
[29]

```
names = x.columns #Loading the names of the x_features
```
[30]

```
names
```
[31]

```
Index(['Date', 'Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'WindGustSpeed',
       'WindSpeed9am', 'WindSpeed3pm', 'Humidity9am', 'Humidity3pm',
       'Pressure9am', 'Pressure3pm', 'Temp9am', 'Temp3pm', '@dropdown',
       'RainToday', 'WindGustDir', 'WindDir9am', 'WindDir3pm'],
      dtype='object')
```

```
# Splitting the data into train and test


x_train,x_test,y_train,y_test =
model_selection.train_test_split(x,y,test_size=0.2, random_state=0)
```

3)Model Development Phase

Initial Training Code :

```python
# Splitting the data into train and test

x_train,x_test,y_train,y_test = model_selection.train_test_split(x,y,test_size=0.2, random_state=0)
```
[35] Python

```python
# Model Initialization
import sklearn
XGBoost = xgboost.XGBRFClassifier()
Rand_forest = sklearn.ensemble.RandomForestClassifier()
svm = sklearn.svm.SVC()
Dtree = sklearn.tree.DecisionTreeClassifier()
GBM = sklearn.ensemble.GradientBoostingClassifier()
log = sklearn.linear_model.LogisticRegression()
```
[ ] Python

```python
# Fitting the model
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBRFClassifier

# --- Encode target variable ---
le = LabelEncoder()
y = le.fit_transform(data['RainTomorrow'])

# --- One-hot encode features ---
x = pd.get_dummies(data.drop('RainTomorrow', axis=1))
```

```python
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBRFClassifier

# --- Encode target variable ---
le = LabelEncoder()
y = le.fit_transform(data['RainTomorrow'])

# --- One-hot encode features ---
x = pd.get_dummies(data.drop('RainTomorrow', axis=1))

# --- Handle missing values ---
imputer = SimpleImputer(strategy="mean")
x = imputer.fit_transform(x)

# --- Split data ---
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# --- Initialize models ---
XGBoost = XGBRFClassifier(n_estimators=50)
Rand_forest = RandomForestClassifier(n_estimators=50)
Dtree = DecisionTreeClassifier()
GBM = GradientBoostingClassifier(n_estimators=50)
log = LogisticRegression(max_iter=1000)

# --- Train models ---
XGBoost.fit(x_train, y_train)
Rand_forest.fit(x_train, y_train)
Dtree.fit(x_train, y_train)
GBM.fit(x_train, y_train)
log.fit(x_train,y_train)
```
37] Python

```
Generate  + Code  + Markdown  |  Run All  Restart  Clear All Outputs  Go To  |  Jupyter Variables  Outline  ...        base (Python 3.11.7)
```

```python
GBM = GradientBoostingClassifier(n_estimators=50)
log = LogisticRegression(max_iter=1000)

# --- Train models ---
XGBoost.fit(x_train, y_train)
Rand_forest.fit(x_train, y_train)
Dtree.fit(x_train, y_train)
GBM.fit(x_train, y_train)
log.fit(x_train,y_train)
```

[37] Python

```
c:\Users\shukr\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\impute\_base.py:637: UserWarning: Skipping features without any observed values:
  warnings.warn(
c:\Users\shukr\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\linear_model\_logistic.py:470: ConvergenceWarning: lbfgs failed to converge after
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=1000).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression
▶ Parameters
```

Model Evaluation and Validation Report :

- **Accuracy Score**

```
# checking the accuracy score
print("xgboost:",metrics.accuracy_score(y_train,p1))
print("Rand_forest:",metrics.accuracy_score(y_train,p2))
# print("svm:",metrics.accuracy_score(y_train,p3))
print("Dtree:",metrics.accuracy_score(y_train,p4))
print("GBM:",metrics.accuracy_score(y_train,p5))
print("log:",metrics.accuracy_score(y_train,p6))
[40]
```

```
xgboost: 0.8344476144644576
Rand_forest: 0.9997078234566203
Dtree: 1.0
GBM: 0.836570191117833
log: 0.8214887254227966
```

```
# Accuracy_score
from sklearn import metrics  # if not already imported

t1 = XGBoost.predict(x_test)
t2 = Rand_forest.predict(x_test)
# t3 = svm.predict(x_test)
t4 = Dtree.predict(x_test)
t5 = GBM.predict(x_test)
t6 = log.predict(x_test)


print("xgboost:", metrics.accuracy_score(y_test, t1))
print("Rand_forest:", metrics.accuracy_score(y_test, t2))
# print("svm:", metrics.accuracy_score(y_test, t3))
print("Dtree:", metrics.accuracy_score(y_test, t4))
print("GBM:", metrics.accuracy_score(y_test, t5))
print("log:", metrics.accuracy_score(y_test, t6))
[45]
```

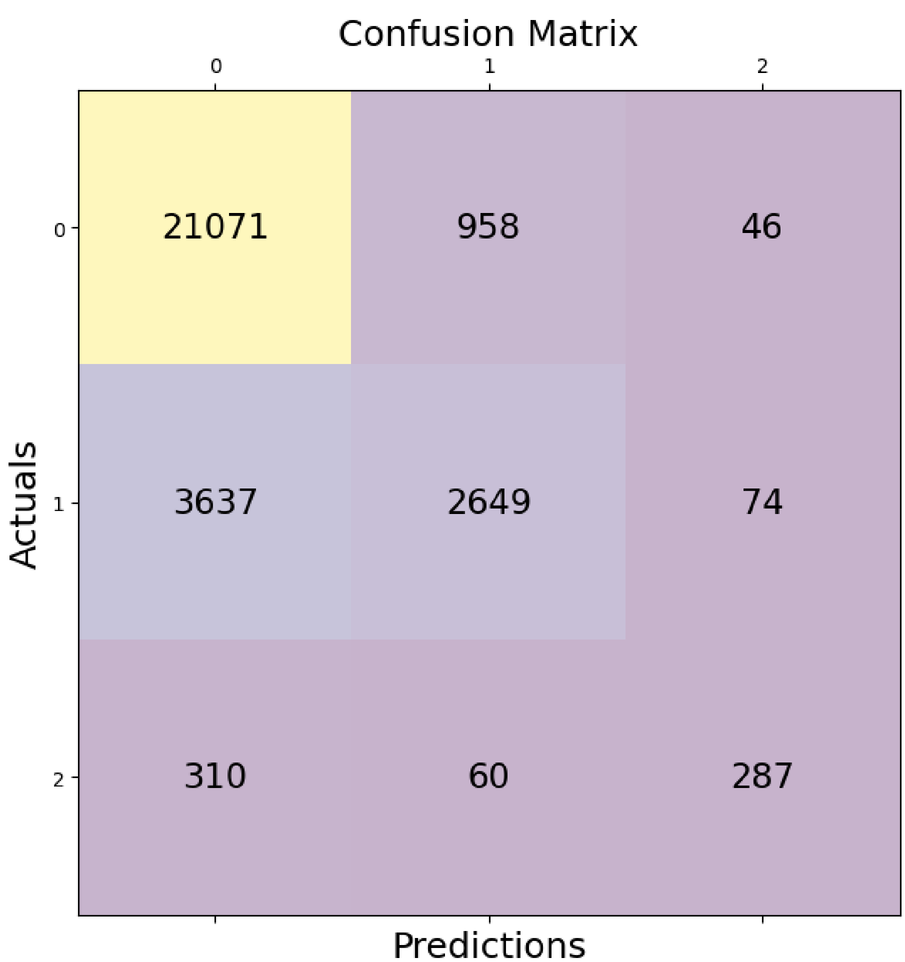
```
xgboost: 0.8252096796370136
Rand_forest: 0.8323594115220679
Dtree: 0.7958545304551079
GBM: 0.8327031486319263
log: 0.8202254915440671
```

Confusion Matrix

```python
# Confusion Matrix
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
```

```python
fig,ax = plt.subplots(figsize=(7.5,7.5))
ax.matshow(conf_matrix,alpha=0.3)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j,y=i,s=conf_matrix[i,j],va='center',ha='center',size='xx-large')

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals',fontsize=18)
plt.title('Confusion Matrix',fontsize=18)
plt.show()
```



Save The Model:

- **Model is saved using pickle**

```
import pickle
[86]
```

```
model = Rand_forest

pickle.dump(model,open('rainfall.pkl','wb'))  #model
pickle.dump(le,open('encoder.pkl','wb'))  #encoder saving
pickle.dump(imp_mode,open('impter.pkl','wb'))   #imputer saving
pickle.dump(sc,open('scale.pkl','wb'))    #scaling the data
[88]
```