Final Project Report

1. Introduction
    1.1. Project overviews
    1.2. Objectives
2. Project Initialization and Planning Phase
    2.1. Define Problem Statement
    2.2. Project Proposal (Proposed Solution)
    2.3. Initial Project Planning
3. Data Collection and Preprocessing Phase
    3.1. Data Collection Plan and Raw Data Sources Identified
    3.2. Data Quality Report
    3.3. Data Exploration and Preprocessing
4. Model Development Phase
    4.1. Feature Selection Report
    4.2. Model Selection Report
    4.3. Initial Model Training Code, Model Validation and Evaluation Report
5. Model Optimization and Tuning Phase
    5.1. Hyperparameter Tuning Documentation
    5.2. Performance Metrics Comparison Report
    5.3. Final Model Selection Justification
6. Results
    6.1. Output Screenshots
7. Advantages & Disadvantages
8. Conclusion
9. Future Scope
10. Appendix
    10.1. Source Code
    10.2. GitHub & Project Demo Link

1.INTRODUCTION

1.1 Project Overview: Rising Waters – A Machine Learning Approach to Flood Prediction

Rising Waters is a machine learning-based flood prediction system designed to forecast flood events with high accuracy. By analyzing historical weather data, river levels, terrain features, and other environmental factors, the model aims to provide early warnings and actionable insights to reduce the impact of floods on lives and infrastructure.

The project focuses on three key areas:

- **Early Warning Systems**: Leveraging real-time data to predict flood risks and issue timely alerts, enabling communities to take preventive actions such as evacuation and reinforcement of flood defenses.
- **Disaster Response Planning**: Supporting emergency services with accurate predictions to plan rescue operations, allocate resources efficiently, and coordinate effective relief efforts during flood events.
- **Infrastructure Resilience**: Assisting urban planners and engineers in designing flood-resilient infrastructure by incorporating risk assessments into development projects, including drainage systems, flood barriers, and green solutions.

This initiative enhances preparedness, minimizes damage, and supports sustainable urban planning in flood-prone regions.

1.2 Objectives

1. Knowledge of Machine Learning Algorithms.
2. Knowledge of Python Language with Machine Learning
3. You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.
4. You will be able to know how to pre-process/clean the data using different data pre-processing techniques.
5. Applying different algorithms according to the dataset and based on visualization.
6. Real-Time Analysis of Project
7. Building ease of User Interface (UI)
8. Navigation of ideas towards other projects(creativeness)
9. Knowledge of building ML models.
10. How to build web applications using the Flask framework.

2.PROJECT INITIALIZATION AND PLANNING PHASE

2.1 Define Problem Statements (Flood Prediction Problem Statement):

The increasing unpredictability of weather patterns poses a serious challenge to both rural and urban communities, particularly in flood-prone regions. Residents, farmers, and local authorities face significant difficulties in anticipating floods, which leads to unpreparedness, property damage, and disruption of livelihoods. The lack of timely and accurate flood forecasting, compounded by the complexity of environmental data such as rainfall, humidity, and temperature, results in delayed response and higher risk.

This situation affects not only the safety and well-being of individuals but also the efficiency of disaster response systems. By leveraging machine learning to build predictive models based on key environmental indicators, we aim to provide early warnings for potential flood events. This solution will empower decision-makers, improve community preparedness, and minimize losses. Our goal is to create an intelligent, data-driven support system that enhances resilience and trust in flood management efforts.

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	A local resident in a flood prone area	Stay prepared for possible flood events	I don't receive any early warnings	Weather data is scattered and hard to interpret	Worried and unprepared

2.2 Project Proposal (Proposed Solution) report

The proposal aims to transform flood prediction using machine learning, improving the accuracy and timeliness of flood risk assessments. It addresses current gaps in early warning systems, promising better disaster preparedness, minimized damage, and improved community safety. Key features include a machine learning-based flood prediction model, real-time alerts, and data-driven support for disaster response planning.

Proposed Solution	
Approach	We collected and preprocessed historical meteorological and flood data, performing univariate, bivariate, and multivariate analysis to understand patterns and relationships. Outliers were removed, and features were standardized before training machine learning models to predict flood occurrences, followed by evaluation using metrics like accuracy, precision, and recall.
Key Features	The model predicts floods using machine learning by analyzing historical rainfall, temperature, humidity, and other environmental data. It enables early warnings and supports effective disaster preparedness and response.

2.3 Initial Project Planning

Sprint	**Functional Requirement (Epic)**	**User Story Number**	**User Story / Task**	**Priority**	**Team Members**	**Sprint Start Date**	**Sprint End Date (Planned)**
Sprint-1	Data Collection and Preprocessing	SL-3	Understanding & loading data	Low	Angarag Seal	2025/06/12	2025/06/13
Sprint-1	Data Collection and Preprocessing	SL-4	Data cleaning	High	Angarag Seal	2025/06/14	2025/06/15
Sprint-1	Data Collection and Preprocessing	SL-5	EDA	Medium	Angarag Seal	2025/06/16	2025/06/17
Sprint-4	Project Report	SL-20	Report	Medium	Vineetha	2025/06/18	2025/06/18
Sprint-2	Model Development	SL-8	Training the model	Medium	Kundana	2025/06/19	2025/06/20
Sprint-2	Model Development	SL-9	Evaluating the model	Medium	Kundana	2025/06/20	2025/06/21
Sprint-2	Model tuning and testing	SL-13	Model tuning	High	Vineetha	2025/06/22	2024/03/23
Sprint-2	Model tuning and testing	SL-14	Model testing	Medium	Vineetha	2025/06/23	2025/06/23

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Team Members	Sprint Start Date	Sprint End Date (Planned)
Sprint-3	Web integration and Deployment	SL-16	Building HTML templates	Low	Maitreyee Mukherjee	2025/06/20	2025/06/21
Sprint-3	Web integration and Deployment	SL-17	Local deployment	Medium	Maitreyee Mukherjee	2025/06/22	2025/06/22

3. DATA COLLECTION AND PREPROCESSING PHASE

3.1 Data Collection Plan & Raw Data Sources Identification Report:

Elevate your data strategy with the Data Collection plan and the Raw Data Sources report, ensuring meticulous data curation and integrity for informed decision-making in every analysis and decision-making endeavor.

Data Collection Plan:

Section	Description
Project Overview	The machine learning project aims to predict the likelihood of a flood event based on key environmental and climatic parameters. Using a dataset with features such as rainfall patterns, temperature, humidity, cloud cover, and runoff levels, the objective is to build a robust classification model that determines whether a flood is expected (1) or not expected (0). This predictive capability enables early warnings, disaster preparedness, and more informed decisionmaking in flood-prone regions.
Data Collection Plan	● Searched for datasets related to meteorological and hydrological data influencing flood occurrences. ● Prioritized datasets with a wide range of environmental parameters across different seasons and regions. ● Focused on obtaining data that represents realworld weather variation to reduce overfitting and improve model generalization.

Raw Data Sources Identified	The raw data sources for this project include publicly available datasets from meteorological repositories and environmental research studies, supplemented with curated synthetic samples to balance the classes and enhance model learning. The final dataset includes attributes such as annual rainfall, seasonal rainfall distribution (e.g., Jan–Feb, Jun–Sep), average temperature and humidity, cloud cover, and surface runoff. These variables were used in the machine learning pipeline to build, evaluate, and deploy models such as XGBoost, Random Forest, Decision Tree, and KNN for flood prediction.,

Raw Data Sources Report:

Source Name	Description	Location/URL	Format	Size	Access Permissions
Kaggle Dataset	The dataset comprises key flood prediction variables such as climatic and hydrological indicators, including temperature, humidity, cloud cover, annual and seasonal rainfall distributions (e.g., Jan–Feb, Mar–May, Jun–Sep, Oct–Dec), average June rainfall, and subsurface runoff.	https://www.kaggle.com/datasets/arbethi/rainfall-dataset	Excel Sheet	16 kB	Public

Kaggle Dataset	This data concerns rainfall in India from 1901-2015	https://www.kaggle.com/datasets/arbethi/rainfall-dataset	Excel Sheet	503 kB	Public

3.2Data Quality Report:

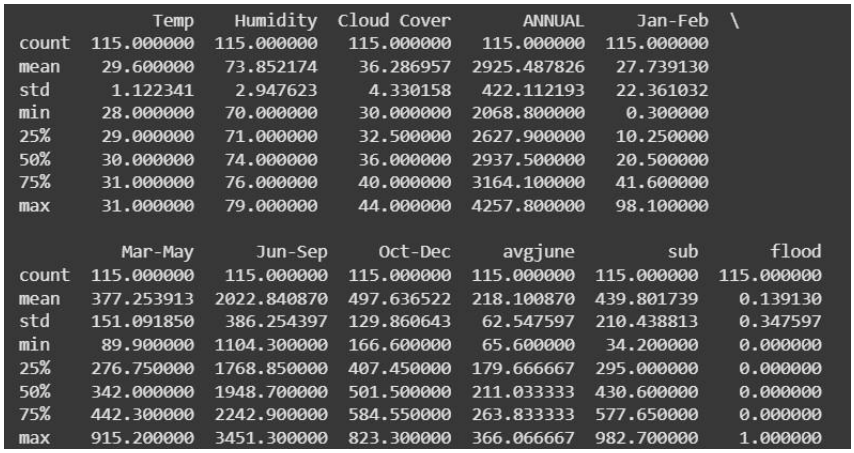
The Data Quality Report will summarize data quality issues from the selected source, including severity levels and resolution plans. It will aid in systematically identifying and rectifying data discrepancies.

Data Quality Report:

Data Source	**Data Quality Issue**	**Severity**	**Resolution Plan**
Kaggle Dataset	Missing values in columns such as Temperature, Humidity, Cloud Cover, Rainfall (seasonal), or Runoff	Moderate	Use mean/median imputation.
Kaggle Dataset	Presence of categorical or mixed-type columns (e.g., possibly mislabeled or regionspecific identifiers)	Moderate	Apply appropriate encoding or remove unused categorical columns
Kaggle Dataset	Outliers in continuous variables like Rainfall and Temperature	Moderate	Use IQR-based outlier removal techniques
Kaggle Dataset	Class imbalance between flood and no-flood cases	Moderate	Apply class balancing techniques such as SMOTE or class weights

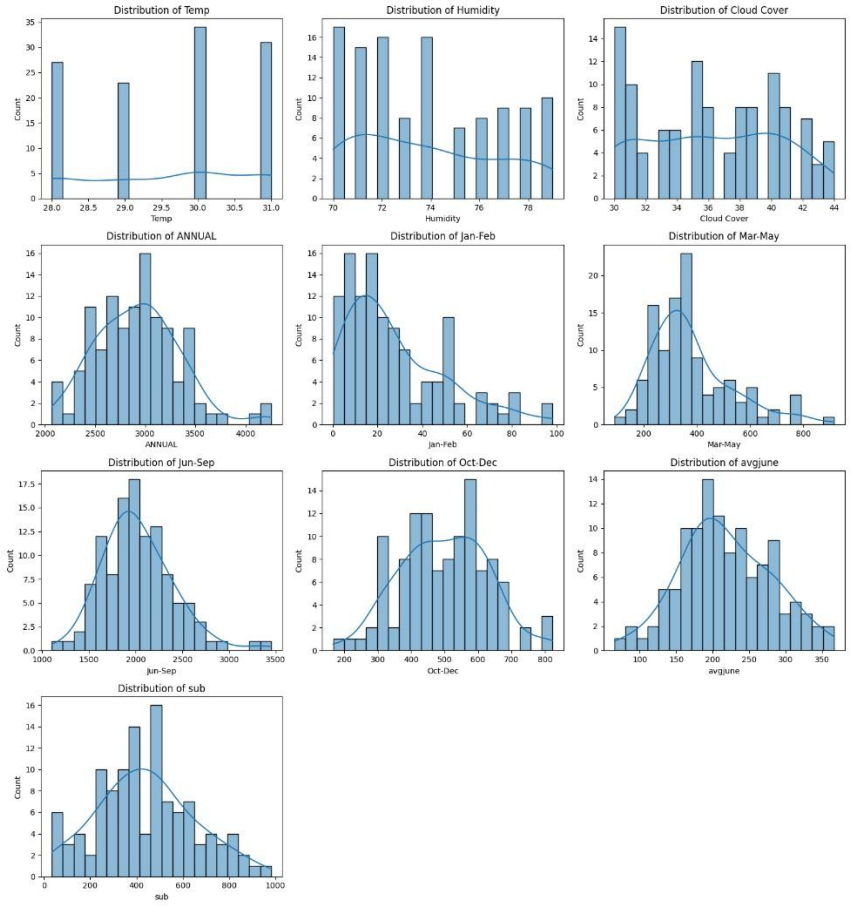
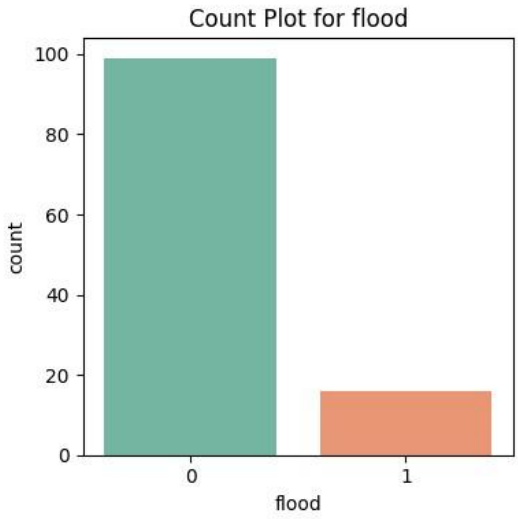
3.3 Data Exploration and Preprocessing Report

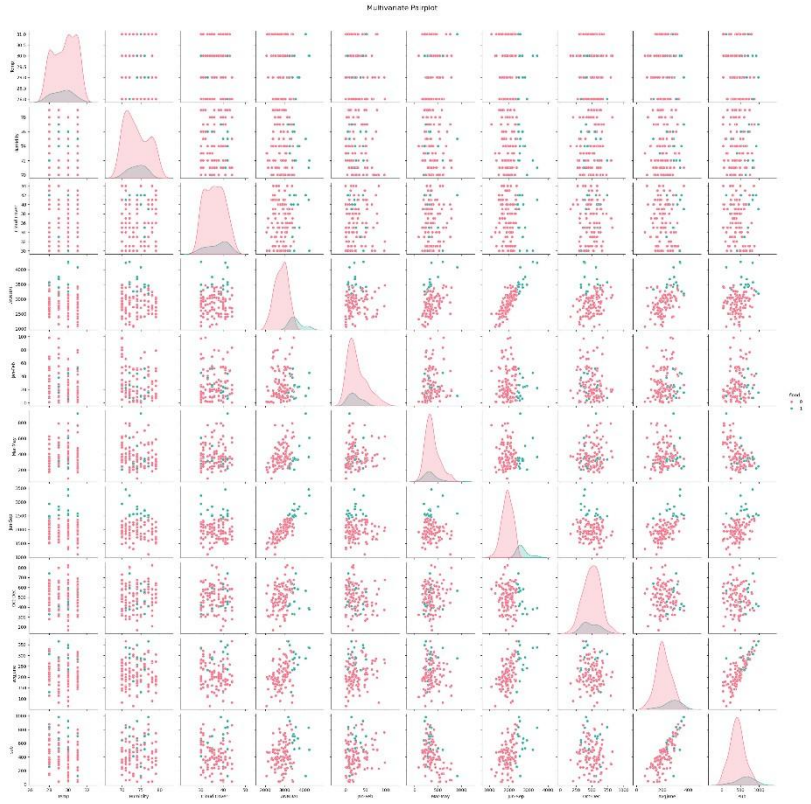
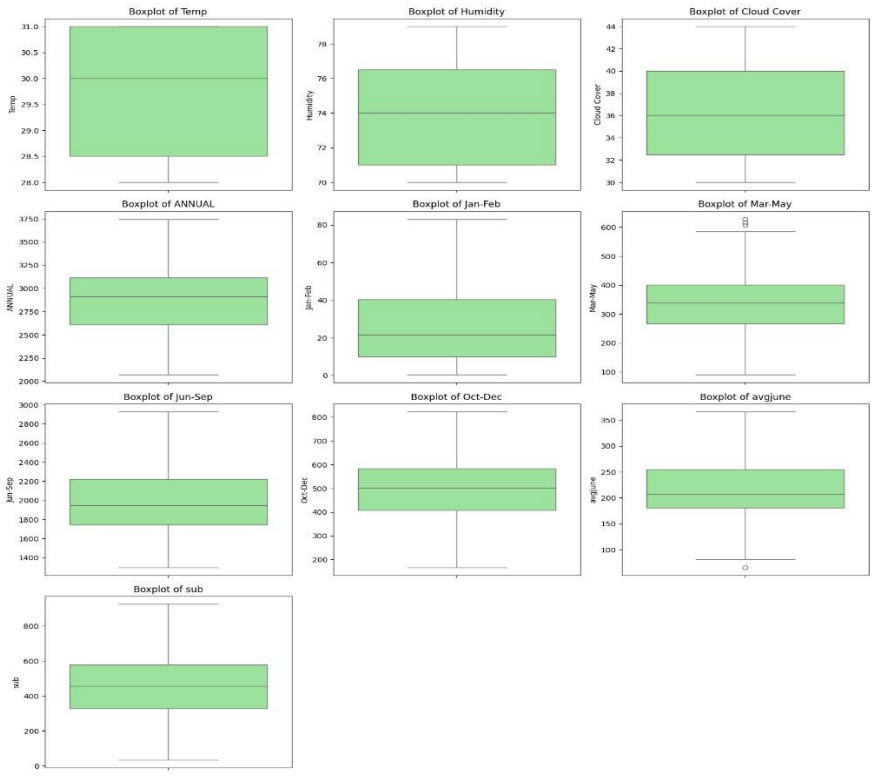
Dataset variables will be statistically analyzed to identify patterns and outliers, with Python employed for preprocessing tasks like normalization and feature scaling. Data cleaning will address missing values and outliers, ensuring quality for subsequent analysis and modeling, and forming a strong foundation for insights and predictions.

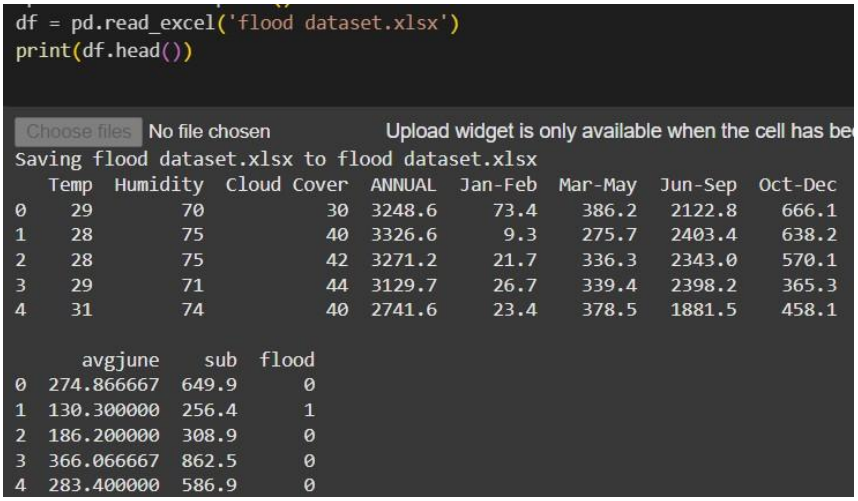
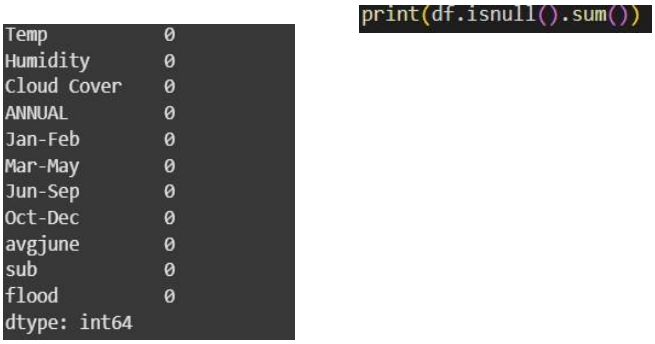
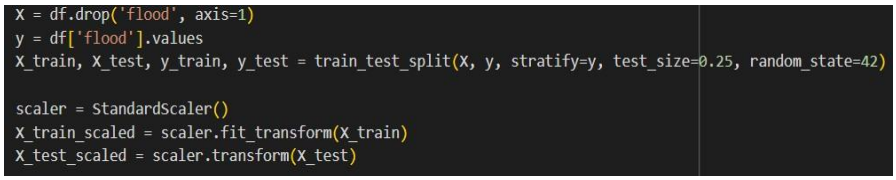
Section	Description
Data Overview	 Dimension: 116 rows × 11 columns Descriptive statistics:

	Temp	Humidity	Cloud Cover	ANNUAL	Jan-Feb	\
count	115.000000	115.000000	115.000000	115.000000	115.000000	
mean	29.600000	73.852174	36.286957	2925.487826	27.739130	
std	1.122341	2.947623	4.330158	422.112193	22.361032	
min	28.000000	70.000000	30.000000	2068.800000	0.300000	
25%	29.000000	71.000000	32.500000	2627.900000	10.250000	
50%	30.000000	74.000000	36.000000	2937.500000	20.500000	
75%	31.000000	76.000000	40.000000	3164.100000	41.600000	
max	31.000000	79.000000	44.000000	4257.800000	98.100000	

	Mar-May	Jun-Sep	Oct-Dec	avgjune	sub	flood
count	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000
mean	377.253913	2022.840870	497.636522	218.100870	439.801739	0.139130
std	151.091850	386.254397	129.860643	62.547597	210.438813	0.347597
min	89.900000	1104.300000	166.600000	65.600000	34.200000	0.000000
25%	276.750000	1768.850000	407.450000	179.666667	295.000000	0.000000
50%	342.000000	1948.700000	501.500000	211.033333	430.600000	0.000000
75%	442.300000	2242.900000	584.550000	263.833333	577.650000	0.000000
max	915.200000	3451.300000	823.300000	366.066667	982.700000	1.000000

Univariate Analysis	
Categorical Count Plot	

Multivariate Analysis	
Outliners and Anomalies	

Data Preprocessing Code Screenshots	
Loading Data	
Handling Missing Data	
Data Transformation	
Feature Scaling	Attached the codes in final submission
Save Processed Data	-

4. MODEL DEVELPOMENT PHASE

4.1 Feature Selection Report Template

In the forthcoming update, each feature will be accompanied by a brief description. Users will indicate whether it's selected or not, providing reasoning for their decision. This process will streamline decision-making and enhance transparency in feature selection.

Feature	Description	Selected(Yes/No)	Reasoning
Temp	Temperature (°C)	Yes	Temperature can influence evapora on and atmospheric pressure, affec ng rainfall and flood dynamics.
Humidity	Humidity (%)	Yes	High humidity o en correlates with increased rainfall and flood risks.
Cloud Cover	Cloud cover (%)	Yes	Indicates sky condi ons; higher coverage may signal more rainfall and poten al flooding.
Jan-Feb	Rainfall during JanuaryFebruary (mm)	Yes	Early year rainfall can saturate soil, affec ng its capacity to absorb future rain.
Mar-May	Rainfall during March-May (mm)	Yes	. Pre-monsoon rainfall can contribute to ground saturation, indirectly affecting flood risk.
Jun-Sep	Rainfall during June-September (mm)	Yes	Primary monsoon period—direct contributor to flooding events.

Oct-Dec	Rainfall during October-December (mm)	Yes	Post-monsoon rainfall can extend waterlogging or delayed flooding situa ons.
avgjune	Average rainfall in June (mm)	Yes	Early monsoon performance is a strong early indicator of flood likelihood.
sub	Sub-basin water accumula on	Yes	Indicates local water collec on; high values increase flood probability.
flood	Flood occurrence (0: No, 1: Yes	Yes	Target variable for classifica on—essen al for model training and evalua on.

4.2 Model Selection Report:

In the forthcoming Model Selection Report, various models will be outlined, detailing their descriptions, hyperparameters, and performance metrics, including Accuracy or F1 Score. This comprehensive report will provide insights into the chosen models and their effectiveness.

Model	**Description**	**Hyperparameters**	**Performance Metric (e.g., Accuracy, F1 Score**
1. **Logistic Regression**	A linear classifier; effective for binary flood prediction, easy to interpret, and useful as a baseline model for detecting flood risk.	random_state=42	Accuracy score =92.31%

2. **Support Vector Classifier**	A margin-based classifier; handles complex, highdimensional flood data using kernel tricks, offering robust separation between flood and no-flood zones.	random_state=42	Accuracy score = 92.31%
3. **Decision Tree Classifier**	A tree-based model; captures conditional rules in flood indicators, easy to visualize, and suitable for understanding environmental triggers.	random_state=42	Accuracy score = 96.15%
4. **Random Forest Classifier**	An ensemble of decision trees; reduces overfitting, handles noisy environmental data well, and improves flood prediction accuracy.	random_state=42	Accuracy score = 96.15%
5. **K-Nearest Neighbors**	An instance-based algorithm; predicts flood events by comparing with similar historical patterns, adaptive and simple to implement.	random_state=42	Accuracy score = 96.15%
6. **Naive Bayes**	A probabilistic model; fast and efficient on sparse or small environmental datasets, useful when flood indicators are conditionally independent.	random_state=42	Accuracy score = 92.31%
7. **XGBoost Classifier**	A powerful gradient boosting model; handles large-scale environmental data, captures non-linear patterns, and offers high accuracy in flood risk prediction.	random_state=42	Accuracy score = 96.15%

4.3 Initial Model Training Code, Model Validation and Evaluation Report

The initial model training code will be showcased in the future through a screenshot. The model validation and evaluation report will include classification reports, accuracy, and confusion matrices for multiple models, presented through respective screenshots.

Initial Model Training Code:

```
# Importing and building the Logistic Regression model
model = LogisticRegression(random_state=42)
model.fit(X_train_scaled, y_train)

# Making predictions for both training and testing data
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)

# Printing training and testing accuracy
print("Logistic Regression")
print(f"Train Accuracy: {accuracy_score(y_train, y_train_pred):.2%}")
print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred):.2%}")
```

```
# Importing and building the K-Nearest Neighbors (KNN) model
model = KNeighborsClassifier()
model.fit(X_train_scaled, y_train)

# Predictions
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)

# Accuracy
print("KNN")
print(f"Train Accuracy: {accuracy_score(y_train, y_train_pred):.2%}")
print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred):.2%}")
```

```
# Importing and building the Naive Bayes model
model = GaussianNB()
model.fit(X_train_scaled, y_train)

# Predictions
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)

# Accuracy
print("Naive Bayes")
print(f"Train Accuracy: {accuracy_score(y_train, y_train_pred):.2%}")
print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred):.2%}")
```

```python
# Importing and building the Support Vector Classifier (SVC) model
model = SVC(random_state=42)
model.fit(X_train_scaled, y_train)

# Making predictions
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)

# Printing accuracies
print("SVC")
print(f"Train Accuracy: {accuracy_score(y_train, y_train_pred):.2%}")
print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred):.2%}")
```

```python
# Importing and building the Random Forest model
model = RandomForestClassifier(random_state=42)
model.fit(X_train_scaled, y_train)

# Predicting
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)

# Accuracy
print("Random Forest")
print(f"Train Accuracy: {accuracy_score(y_train, y_train_pred):.2%}")
print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred):.2%}")
```

```python
# Importing and building the Decision Tree model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train_scaled, y_train)

# Predicting
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)

# Accuracy
print("Decision Tree")
print(f"Train Accuracy: {accuracy_score(y_train, y_train_pred):.2%}")
print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred):.2%}")
```

```python
# Importing and building the XGBoost model
model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
model.fit(X_train_scaled, y_train)

# Predictions
y_train_pred = model.predict(X_train_scaled)
y_test_pred = model.predict(X_test_scaled)

# Accuracy
print("XGBoost")
print(f"Train Accuracy: {accuracy_score(y_train, y_train_pred):.2%}")
print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred):.2%}")
```

Model	Classification Report	F1 Score	Confusion Matrix

Logis c Regression	`print("\nClassification Report:")`
	`print(classification_report(y_test, y_test_pred))`
	Classification Report:
	precision recall f1-score support
	0 0.96 0.96 0.96 23
	1 0.67 0.67 0.67 3
	accuracy 0.92 26
	macro avg 0.81 0.81 0.81 26
	weighted avg 0.92 0.92 0.92 26
		67%	`print("Confusion Matrix:")`
			`print(confusion_matrix(y_test, y_test_pred))`
			Confusion Matrix:
			[[22 1]
			[1 2]]

Model Validation and Evaluation Report:

SVC	`print("\nClassification Report:")`
	`print(classification_report(y_test, y_test_pred))`
	
	Classification Report:
	precision recall f1-score support
	
	0 0.92 1.00 0.96 23
	1 1.00 0.33 0.50 3
	
	accuracy 0.92 26
	macro avg 0.96 0.67 0.73 26
	weighted avg 0.93 0.92 0.91 26
		50%	`print("Confusion Matrix:")`
			`print(confusion_matrix(y_test, y_test_pred))`
			
			Confusion Matrix:
			[[23 0]
			[2 1]]

Decision Tree	`print("\nClassification Report:")`
	`print(classification_report(y_test, y_test_pred))`
	
	Classification Report:
	precision recall f1-score support
	
	0 0.96 1.00 0.98 23
	1 1.00 0.67 0.80 3
	
	accuracy 0.96 26
	macro avg 0.98 0.83 0.89 26
	weighted avg 0.96 0.96 0.96 26
		80%	`print("Confusion Matrix:")`
			`print(confusion_matrix(y_test, y_test_pred))`
			
			Confusion Matrix:
			[[23 0]
			[1 2]]

Random Forest	`print("\nClassification Report:")`
	`print(classification_report(y_test, y_test_pred))`
	
	Classification Report:
	precision recall f1-score support
	
	0 0.96 1.00 0.98 23
	1 1.00 0.67 0.80 3
	
	accuracy 0.96 26
	macro avg 0.98 0.83 0.89 26
	weighted avg 0.96 0.96 0.96 26
		80%	`print("Confusion Matrix:")`
			`print(confusion_matrix(y_test, y_test_pred))`
			
			Confusion Matrix:
			[[23 0]
			[1 2]]

K-Nearest Neighbors	`print("\nClassification Report:")`
	`print(classification_report(y_test, y_test_pred))`
	
	Classification Report:
	precision recall f1-score support
	
	0 0.96 1.00 0.98 23
	1 1.00 0.67 0.80 3
	
	accuracy 0.96 26
	macro avg 0.98 0.83 0.89 26
	weighted avg 0.96 0.96 0.96 26
		80%	`print("Confusion Matrix:")`
			`print(confusion_matrix(y_test, y_test_pred))`
			
			Confusion Matrix:
			[[23 0]
			[1 2]]

<table>
<tr>
<td>Naive
Bayes</td>
<td>

```
print("\nClassification Report:")
print(classification_report(y_test, y_test_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        23
           1       0.67      0.67      0.67         3

    accuracy                           0.92        26
   macro avg       0.81      0.81      0.81        26
weighted avg       0.92      0.92      0.92        26
```

</td>
<td>67%</td>
<td>

```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred))
```

```
Confusion Matrix:
[[22  1]
 [ 1  2]]
```

</td>
</tr>
<tr>
<td>XGBoost</td>
<td>

```
print("\nClassification Report:")
print(classification_report(y_test, y_test_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        23
           1       1.00      0.67      0.80         3

    accuracy                           0.96        26
   macro avg       0.98      0.83      0.89        26
weighted avg       0.96      0.96      0.96        26
```

</td>
<td>80%</td>
<td>

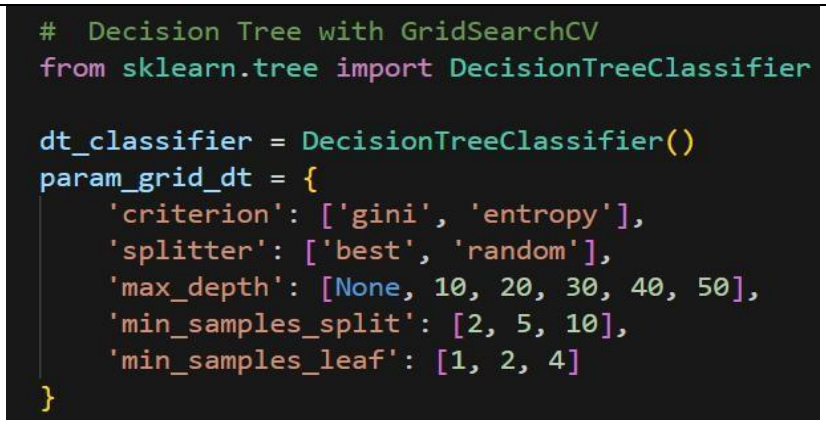
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred))
```

```
Confusion Matrix:
[[23  0]
 [ 1  2]]
```

</td>
</tr>
</table>

5. MODEL OPTIMIZATION AND TUNING PHASE

5.1 Hyper-parameter Tuning Documentation:

MODEL	Tuned Hyperparameters
Decision Tree	

```
#  Decision Tree with GridSearchCV
from sklearn.tree import DecisionTreeClassifier

dt_classifier = DecisionTreeClassifier()
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Random Forest	

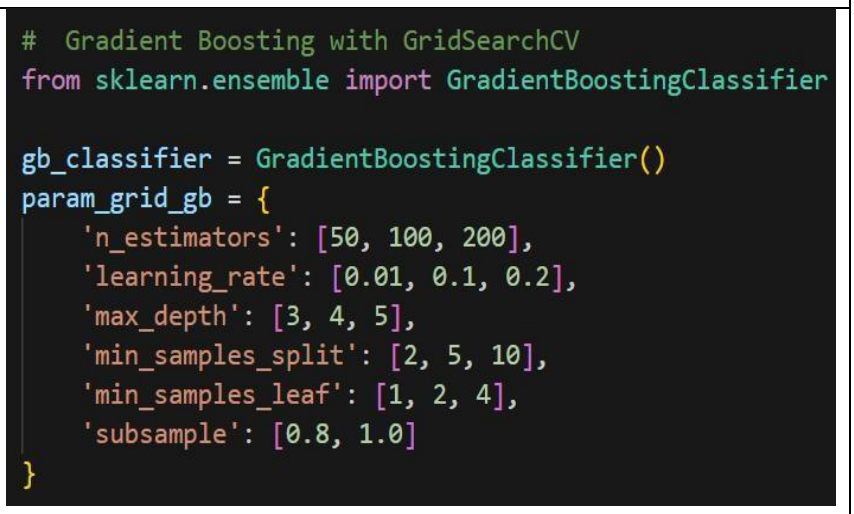
```
#  Random Forest with GridSearchCV
from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier()
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

KNN	

```
#  KNN with GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier()
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}
```

Gradient Boosting	

```
#  Gradient Boosting with GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier

gb_classifier = GradientBoostingClassifier()
param_grid_gb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 1.0]
}
```

Optimal Values

```
grid_dt = GridSearchCV(estimator=dt_classifier, param_grid=param_grid_dt, cv=5)
grid_dt.fit(x_train, y_train)
y_pred_dt = grid_dt.predict(x_test)
print("Optimal Hyperparameters:", grid_dt.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
```

```
Optimal Hyperparameters: {'criterion': 'entropy', 'max_depth': 30, 'min_samples_leaf': 2, 'min_samples_split': 2, 'splitter': 'random'}
Accuracy: 1.0
```

```
grid_rf = GridSearchCV(estimator=rf_classifier, param_grid=param_grid_rf, cv=5)
grid_rf.fit(x_train, np.ravel(y_train))
y_pred_rf = grid_rf.predict(x_test)
print("Optimal Hyperparameters:", grid_rf.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Optimal Hyperparameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Accuracy: 0.9615384615384616
```

```
grid_rf = GridSearchCV(estimator=rf_classifier, param_grid=param_grid_rf, cv=5)
grid_rf.fit(x_train, np.ravel(y_train))
y_pred_rf = grid_rf.predict(x_test)
print("Optimal Hyperparameters:", grid_rf.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
```

```
Optimal Hyperparameters: {'n_neighbors': 5, 'p': 1, 'weights': 'uniform'}
Accuracy: 0.9615384615384616
```

```
grid_knn = GridSearchCV(estimator=knn_classifier, param_grid=param_grid_knn, cv=5)
grid_knn.fit(x_train, np.ravel(y_train))
y_pred_knn = grid_knn.predict(x_test)
print("Optimal Hyperparameters:", grid_knn.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
```

```
Optimal Hyperparameters: {'learning_rate': 0.2, 'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50, 'subsample'
Accuracy: 1.0
```

5.2 Performance Metrics Comparison Report:

Model	Optimized Metric
Decision Tree	

```
print(classification_report(y_test, y_pred_dt))
print(confusion_matrix(y_test, y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        24
           1       1.00      1.00      1.00         2

    accuracy                           1.00        26
   macro avg       1.00      1.00      1.00        26
weighted avg       1.00      1.00      1.00        26

[[24  0]
 [ 0  2]]
```

Random Forest	

```
print(classification_report(y_test, y_pred_rf))
print(confusion_matrix(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        24
           1       1.00      0.50      0.67         2

    accuracy                           0.96        26
   macro avg       0.98      0.75      0.82        26
weighted avg       0.96      0.96      0.96        26

[[24  0]
 [ 1  1]]
```

KNN

```
print(classification_report(y_test, y_pred_knn))
print(confusion_matrix(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        24
           1       0.67      1.00      0.80         2

    accuracy                           0.96        26
   macro avg       0.83      0.98      0.89        26
weighted avg       0.97      0.96      0.96        26

[[23  1]
 [ 0  2]]
```

Gradient Boosting

```
print(classification_report(y_test, y_pred_gb))
print(confusion_matrix(y_test, y_pred_gb))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        24
           1       1.00      1.00      1.00         2

    accuracy                           1.00        26
   macro avg       1.00      1.00      1.00        26
weighted avg       1.00      1.00      1.00        26

[[24  0]
 [ 0  2]]
```

5.3 Final Model Selection Justification:

Our Dataset is imbalanced.

Model	**Accuracy**	**Recall (1)**	**F1 (1)**
Decision Tree	1.00	1.00	1.00
Random Forest	0.96	0.50	0.67
KNN	0.96	1.00	0.80

Gradient Boosting	1.00	1.00	1.00

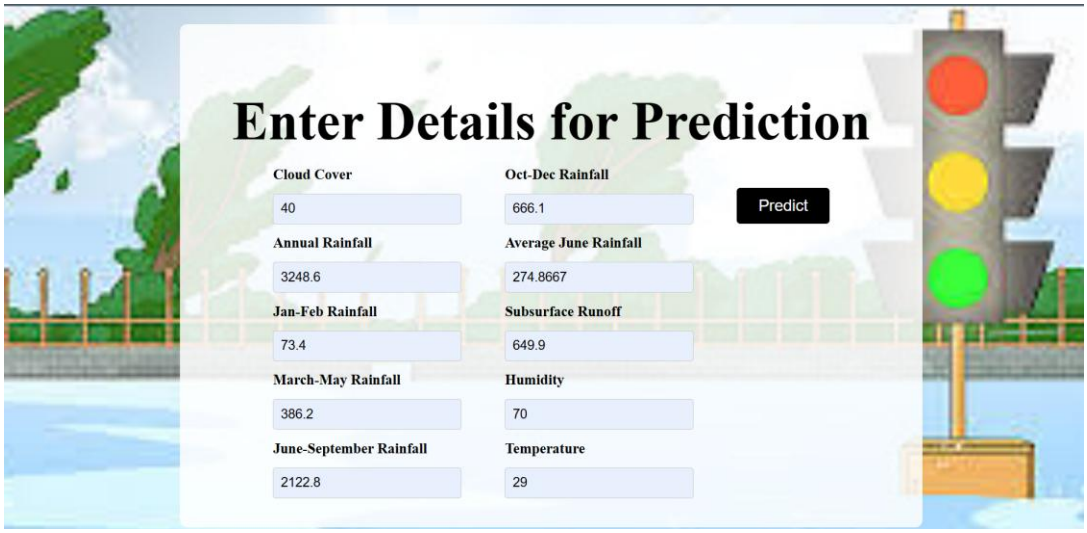
After tuning and evaluating all models, **Gradient Boosting Classifier** was selected as the final model.

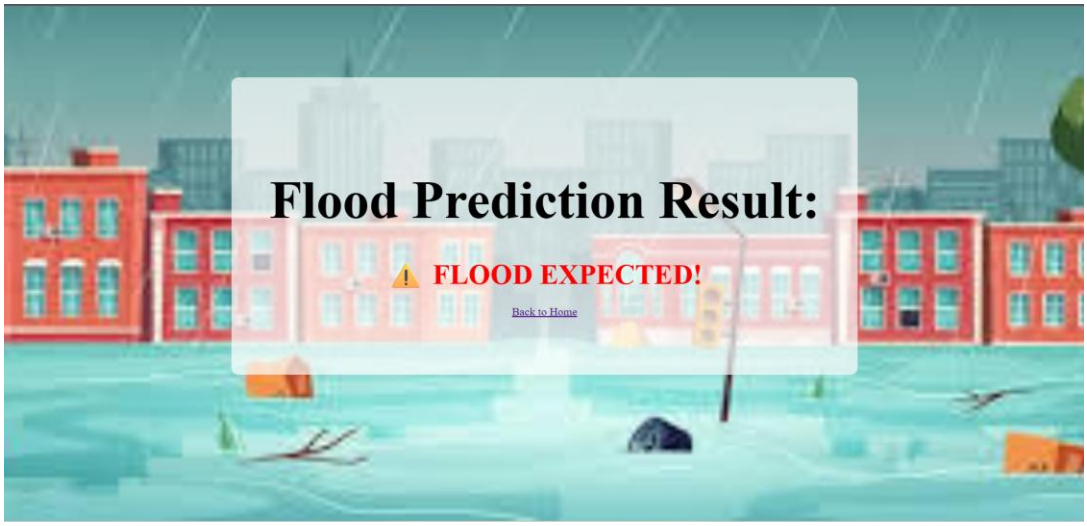
- It achieved **100% accuracy** and **perfect scores (1.00)** in all metrics (precision, recall, F1) for both flood and non-flood classes.
- It **handled class imbalance** well and ensured **zero false negatives**, which is crucial in flood detection.
- Compared to Random Forest and KNN, it performed better on the rare flood cases.
- Unlike Decision Tree, it is **less prone to overfitting** and combines multiple learners for stronger performance.
- Gradient Boosting provides **better control over learning rate and depth**, making it ideal for **future improvements**.

In real-world flood prediction, **missing a flood is riskier than a false alarm**, and this model gives the **most reliable and safe predictions**.

6.FLASK APPLICATION

6.1 Result of the application


Enter Details for Prediction
Cloud Cover
40
Oct-Dec Rainfall
666.1
Predict
Annual Rainfall
3248.6
Average June Rainfall
274.8667
Jan-Feb Rainfall
73.4
Subsurface Runoff
649.9
March-May Rainfall
386.2
Humidity
70
June-September Rainfall
2122.8
Temperature
29


Flood Prediction Result:
FLOOD EXPECTED!
Back to Home

10.ADVANTAGES & DISADVANTAGES

☑ Advantages:

1. Early Warning Capability
    - Enables timely alerts, allowing communities and authorities to take preventive actions and reduce loss of life and property.
2. Data-Driven Decision Making
    - Utilizes historical and real-time data for more accurate and objective flood risk assessment.
3. Resource Optimization
    - Helps emergency services allocate resources more effectively during disaster response operations.
4. Infrastructure Planning Support
    - Aids city planners and engineers in designing flood-resilient infrastructure by providing location-specific risk insights.
5. Scalability
    - Can be scaled and customized for different geographic regions and types of flood-prone environments.
6. Cost-Effective in the Long Run
    - Reduces long-term economic damage by enabling proactive rather than reactive flood management.

❌ Disadvantages:

1. Data Dependency
    - Requires large volumes of high-quality, real-time data (weather, hydrology, topography), which may not be available in all regions.
2. Model Limitations
    - Machine learning models may produce inaccurate predictions if trained on biased, incomplete, or outdated data.
3. High Initial Setup Cost
    - Implementation involves costs for sensors, data collection systems, infrastructure, and skilled professionals.
4. Complexity of Natural Systems
    - Floods are influenced by numerous unpredictable factors (e.g., sudden dam breaks, urbanization effects), which models may not fully capture.
5. Maintenance and Updates
    - Models must be frequently updated with new data to remain accurate, requiring continuous monitoring and tuning.

6. Limited Interpretability
    - Some machine learning models (like neural networks) can be seen as "black boxes," making it difficult to understand how predictions are made.

8.CONCLUSION

The **"Rising Waters"** project demonstrates the powerful potential of machine learning in addressing real-world challenges like flood prediction and disaster management. By leveraging historical and real-time data, the system offers timely and accurate flood forecasts that can save lives, protect infrastructure, and support efficient disaster response. While challenges such as data availability and model accuracy remain, the benefits of early warning, informed planning, and reduced economic loss make this approach a valuable tool for flood-prone regions. With continued improvement and integration into broader disaster management systems, machine learning can play a critical role in building safer, more resilient communities.

9. FUTURE SCOPE

The potential for enhancing flood prediction using machine learning is vast. Future developments in the **"Rising Waters"** project could include:

1. **Integration with IoT and Real-Time Sensors**
    a. Deploying IoT devices for real-time data collection (e.g., rainfall, river levels, soil moisture) can improve model accuracy and responsiveness.
2. **Use of Satellite and Remote Sensing Data**
    a. Incorporating satellite imagery and remote sensing data can enhance flood mapping and enable predictions even in remote or inaccessible areas.
3. **Advanced Deep Learning Models**
    a. Implementing deep learning techniques like CNNs or LSTMs could better capture spatial and temporal patterns in complex flood scenarios.
4. **Geo-Spatial Risk Mapping**
    a. Creating interactive flood risk maps using GIS tools can aid planners and the public in visualizing vulnerable areas.
5. **Mobile Alert Systems**
    a. Building mobile applications or SMS-based alert systems can ensure that warnings reach local populations quickly and effectively.

6. **Climate Change Impact Analysis**
    a. Enhancing the model to account for long-term climate change trends could support more robust planning and adaptation strategies.
7. **Cross-Regional Model Generalization**
    a. Making the model adaptable across multiple geographic locations by training on diverse datasets for broader applicability.
8. **Policy and Decision-Making Support**
    a. Integrating predictions into government planning systems to guide investments in flood prevention infrastructure and emergency services.

10. APPENDIX

10.1 Source Code (app.py):

app.py > predict_flood

```python
from flask import Flask, render_template, request
import joblib
import pandas as pd

app = Flask(__name__)

# ✅ Load both model and scaler
model, scaler = joblib.load(open('flood.save', 'rb'))

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/intro')
def intro():
    return render_template('intro.html')

@app.route('/predict')
def predict():
    return render_template('index.html')

@app.route('/predict_flood', methods=['POST'])
def predict_flood():
    # Extract input values
    temp = float(request.form['Temp'])
    humidity = float(request.form['Humidity'])
    cloud_cover = float(request.form['cloud_cover'])
    annual_rainfall = float(request.form['ANNUAL'])
    jan_feb = float(request.form['Jan-Feb'])
    mar_may = float(request.form['Mar-May'])
    jun_sep = float(request.form['Jun-Sep'])
    oct_dec = float(request.form['Oct-Dec'])
    avg_june = float(request.form['avgjune'])
    sub = float(request.form['sub'])

```

```python
    # Prepare input DataFrame
    input_data = pd.DataFrame([[temp, humidity, cloud_cover, annual_rainfall, jan_feb,
                                mar_may, jun_sep, oct_dec, avg_june, sub]],
                              columns=['Temp', 'Humidity', 'Cloud Cover', 'ANNUAL', 'Jan-Feb',
                                       'Mar-May', 'Jun-Sep', 'Oct-Dec', 'avgjune', 'sub'])

    # ✅ Scale the input
    input_scaled = scaler.transform(input_data)

    # ✅ Predict
    prediction = model.predict(input_scaled)
    print("Prediction:", prediction[0])

    return render_template('imageprediction.html', prediction=prediction[0])

if __name__ == '__main__':
    app.run(debug=True)
```

Source code (model_testing.py):

model_testing.py > ...

```python
import joblib
import pandas as pd

# ✅ Load both model and scaler from flood.save
model, scaler = joblib.load(open('flood.save', 'rb'))

# ✅ Test input (replace with real values if needed)
sample_input = pd.DataFrame([[30, 65, 60, 1200, 100, 200, 300, 400, 35, 1]],
    columns=['Temp', 'Humidity', 'Cloud Cover', 'ANNUAL', 'Jan-Feb',
             'Mar-May', 'Jun-Sep', 'Oct-Dec', 'avgjune', 'sub'])

# ✅ Scale the input
scaled_input = scaler.transform(sample_input)

# ✅ Predict
prediction = model.predict(scaled_input)

# ✅ Output
result = "🌊 Flood Likely" if prediction[0] == 1 else "✅ No Flood"
print("📌 Model Test Result:", result)
```

10.2 GitHub & Project Demo Link

- https://github.com/Vineetha-Jakkilinki/flood_analysis/tree/main
- working prototype.mp4