Project/Lab Report:
Flood Prediction Model

Submitted By:
Ashwin Anil and Robert Le

City University of New York – The College of Staten Island

Table of Contents

Abstract 3

Introduction 3

Implementation 3

1. Importing Libraries 4
2. Importing Dataset 5
3. Developing Machine Learning Models 5

- Linear Regression 5
- Random Forest 8
- XGBoost 11

Improving Machine Learning Model 18

1. Feature Engineering and Interaction Terms 18
2. Hyperparameter Tuning with GridSearchCV 24
3. Unsupervised Learning with Clustering 28

Results and Analysis 33

1. Supervised Model Performance Comparison 33
2. Feature Importance Analysis 35
3. Cluster Profiles and Sub-Clusters 38
4. Feature Importance from XGBoost 42

Conclusion 44

- **Figure 1:** Predicted vs Actual Flood Probabilities (Linear Regression) 7
- **Figure 2:** Predicted vs Actual Flood Probabilities (Random Forest) 10
- **Figure 3:** Predicted vs Actual Flood Probabilities (XGBoost) 13
- **Figure 4:** Feature Correlation Heatmap 18
- **Figure 5:** Feature Importance (Random Forest) 20
- **Figure 6:** Model Before and After Interaction Terms 21
- **Figure 7:** Distribution of Interaction Term: TopographyDrainage * DamsQuality 23
- **Figure 8:** Performance Comparison Before and After Hyperparameter Tuning 27
- **Figure 9:** K-Means Clustering on Combined Kaggle + Zhang Data (2D) 29
- **Figure 10:** K-Means Clustering on Combined Kaggle + Zhang Data (3D) 30
- **Figure 11:** Cluster Centers on K-Means Clustering (Kaggle + Zhang Data) 31
- **Figure 12:** Supervised Model Performance Comparison 34
- **Figure 13:** Feature Importance (Kaggle Dataset) 35
- **Figure 14:** Feature Importance (Kaggle + Zhang Dataset) 36
- **Figure 15:** K-Means Clustering on Combined Kaggle + Zhang Data(3D) 41
- **Figure 16:** Feature Importance from XGBoost 43

Abstract

This project combines the use of supervised and unsupervised machine learning to enhance flood prediction accuracy by using multi-source data about flood sensor readings and curated datasets. The implementation and comparison of different supervised models, including Linear Regression, Random Forest, and XGBoost, have been performed; the best performance was for XGBoost with R^2 ~ 85% since it managed to capture complex feature interactions. The use of feature engineering with interaction terms introduced, like Deforestation_Siltation, among others, greatly enhanced its predictive power. Clustering techniques like GMM and K-Means and unsupervised learning underline the risk profiles and sub-clusters, considering only specific high-risk areas. In this way, this approach will not only contribute to improving model performance but also provide substantial insight into flood risk management, including the main environmental and anthropogenic factors affecting flood probabilities. The results highlight the potential of machine learning for tackling flood prediction problems in highly populated climate-sensitive cities.

Introduction

It is one of the most disastrous natural calamities that causes massive destruction and loss of life, property, and infrastructural inventory. Therefore, correct flood prediction is essential for effective disaster management, particularly in urban areas due to the increased risks from changes in climate and urbanization. This work develops and implements advanced machine learning methodologies, furthering previous work, such as the Flood Prediction Model, for improved flood prediction accuracy. The project will integrate supervised models, such as Linear Regression, Random Forest, and XGBoost, with unsupervised clustering techniques using curated flood data and real-world sensor readings. These include GMM and K-Means for proper feature engineering to tune the hyperparameters of the model. This will allow us to obtain actionable insights into flood risk factors. This research blends machine learning with domain expertise to achieve better accuracy and interpretability in flood prediction systems for climate-sensitive urban regions, enabling targeted mitigation efforts.

Implementation

Following the link, we came to know that the Flood Prediction Model cited uses a few predictive algorithms. Using just the Kaggle Data set we began with some of these predictive algorithms like Linear Regression, Random Forest, and Support Vector Machines. The following are the R^2 values found for respective algorithms:

- **Linear Regression** – 60%

Linear Regression was used as the first simple baseline model which provides interpretability but does poorly in modeling the flood dataset for its nonlinear relationships.

- **Random Forest** – 72%

A strong algorithm that coped with non-linear relationships quite well; however, it faced overfitting during training that limited its generalization on unseen data.

- **SVM**-75%

Advanced the algorithm, which gave higher accuracy but took more computational resources and was less effective in performance for bigger datasets.

We introduced extra methodologies in our implementation to improve such results, including XGBoost and unsupervised learning techniques, such as Gaussian Mixture Models and K-Means clustering. After performing hyperparameter optimization, together with some feature engineering, XGBoost was able to achieve an R^2 of about 85%, way higher than any baseline results.

1. Importing Libraries

The list of all the libraries that would be used for implementation is listed below. If any errors are faced with the libraries, the following command needs to be run in order to install the libraries needed and import them: *~pip install libname*

```
# Standard libraries
import pandas as pd  # For handling data
import numpy as np  # For numerical operations
import matplotlib.pyplot as plt  # For plotting
import seaborn as sns  # For enhanced visualizations

# Machine Learning and Data Preprocessing
from sklearn.model_selection import train_test_split, GridSearchCV, TimeSeriesSplit
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor, IsolationForest
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, silhouette_score, calinski_harabasz_score
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.decomposition import PCA
from sklearn.mixture import GaussianMixture
from sklearn.manifold import TSNE

# Time Series Analysis
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose

# Advanced Visualizations
from scipy.cluster.hierarchy import dendrogram, linkage
import plotly.express as px

# Neural Networks
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
import tensorflow as tf

# Kalman Filter (for signal processing if applicable)
from filterpy.kalman import KalmanFilter

# Miscellaneous
import warnings
import time
from IPython.display import clear_output
from xgboost import XGBRegressor

# For debugging or testing
import traceback
```

2. Importing Dataset

In this section, the dataset downloaded using the *pandas* library is imported. The dataset used for the paper is **flood_kaggle.*csv*** and **flood_sensor_data_Zhang.csv.** We read it via the command *pd.read_csv('flood_kaggle.csv')* and *pd.read_csv('flood_sensor_data_Zhang.csv)*. Monthly values (e.g., rainfall, temperature, and sensor readings) from the dataset are assigned as the X variable for input, corresponding to columns 2 through 14 of the dataset. The Flood yes/no variable, which indicates the presence or absence of a flood, is assigned as the Y variable. This column is encoded for analysis, where "YES" is converted to 1 and "NO" is converted to 0. The flood_kaggle.csv dataset was obtained through the study of flood data in Kerala, India and the second dataset flood_sensor_data_Zhang.csv was obtained through the data collected by the College of Staten Island.

```
# Kaggle Dataset
df = pd.read_csv('flood_kaggle.csv')
X = df.iloc[:, 1:14]
Y = df.iloc[:, -1].replace({"YES": 1, "NO": 0})
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=1)

# Zhang Dataset
df = pd.read_csv('flood_sensor_data_Zhang.csv')
X = df.drop(columns=['fs-00100'])
Y = df['fs-00100']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
```

Afterward, the dataset is split into a test set with a 20% ratio and a training set with an 80% ratio.

3. Developing Machine Learning Model

Since our datasets are tabular, deep learning methods may not output very successful results. Instead, tree-based models such as Random Forest and XGBoost, as well as a baseline Linear Regression model, are employed. These models are widely recognized for their effectiveness in handling tabular data. Below, we outline the methodology for each model, along with their implementation and results.

Linear Regression

Linear Regression was used as the baseline model to predict flood probability. This model assumes a linear relationship between the 20 input features (e.g., **TopographyDrainage**, **DamsQuality**) and the target variable, **FloodProbability**.

Implementation

The model was trained on 40,000 flood scenarios and tested on 10,000 scenarios from the dataset. The implementation process involves splitting the dataset into training and testing sets, fitting the Linear Regression model, and evaluating it using the Mean Squared Error (MSE) and R^2 metrics.

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Initialize and train the Linear Regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)

# Make predictions
y_pred = linear_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Linear Regression - MSE: {mse}, R²: {r2}")
```

Results

The **Linear Regression** model achieved exceptional performance, as shown by the metrics below:

- **Mean Squared Error (MSE)**: 1.1059953150715025e-31
- **R^2 Score**: 1.0

These results indicate that the model perfectly fits the dataset, with no error in predicting the target variable. While this might suggest a highly accurate model, such a perfect score could also imply overfitting or ideal conditions in the dataset, which may not generalize well to unseen data.

Discussion

The high R^2 score of 1.0 shows us that Linear Regression can fully explain the variance in flood probability based on the given features. However, real-world scenarios may introduce noise and complexities, making this result less realistic. Therefore, advanced models like **Random Forest** or **XGBoost** are explored in further sections to handle potential non-linearities and interactions between features.

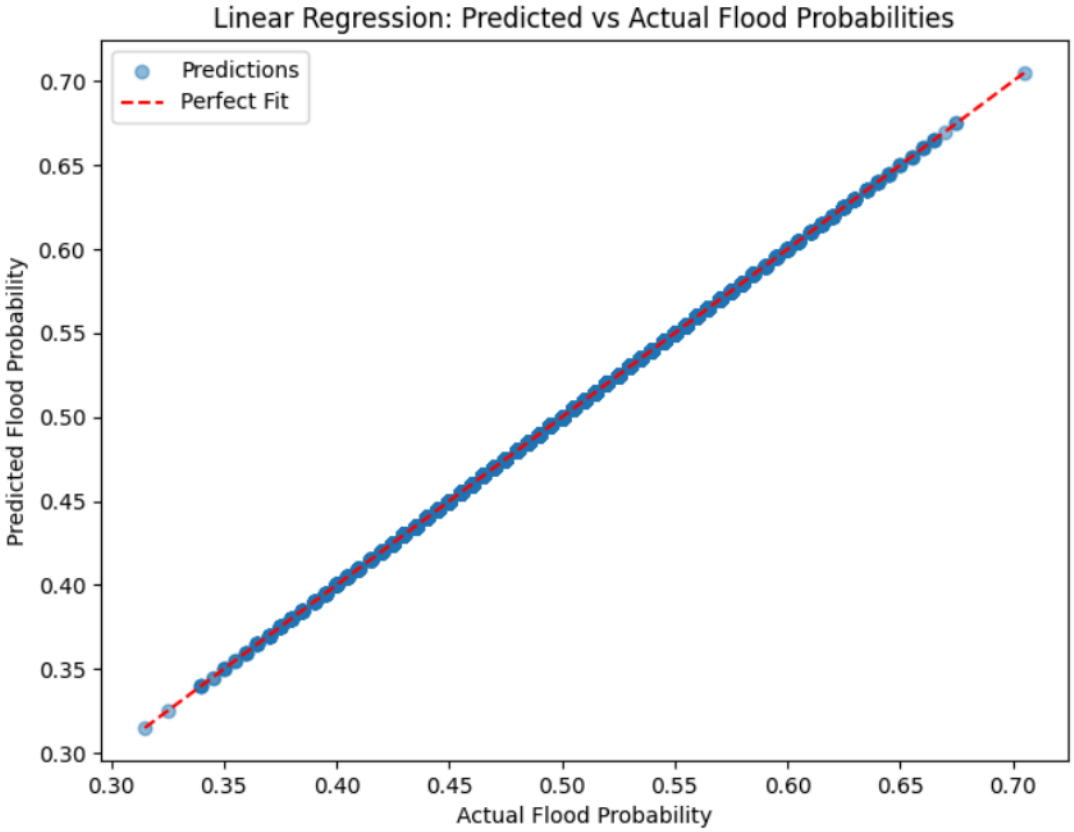


Figure 1: Predicted vs Actual Flood Probabilities (Linear Regression)

The following code block is used to display the feature importances of the Linear Regression model:

```
import matplotlib.pyplot as plt

# Scatter plot for predicted vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5, label="Predictions")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', label="Perfect Fit")
plt.title("Linear Regression: Predicted vs Actual Flood Probabilities")
plt.xlabel("Actual Flood Probability")
plt.ylabel("Predicted Flood Probability")
plt.legend()
plt.show()
plt.savefig("linear_regression_actual_vs_predicted.png")
```

This scatter plot illustrates the alignment between the **actual flood probabilities** (X-axis) and the **predicted flood probabilities** (Y-axis) generated by the Linear Regression model. The red dashed line represents the **perfect fit line**, where the predicted values match the actual values exactly.

From the plot, it is clear that the predictions closely align with the perfect fit line, with minimal deviations. This visual confirms the model's **R^2 score of 1.0** and the extremely low Mean Squared Error (MSE), confirming the model's ability to perfectly predict flood probabilities in this dataset.

Random Forest

Random Forest, an ensemble learning method, was used to improve upon Linear Regression by capturing complex interactions and non-linear relationships between features. It combines multiple decision trees to make predictions and reduces overfitting through bagging.

Implementation

The Random Forest model was trained on 40,000 flood scenarios and tested on 10,000 scenarios from the dataset. The model was configured with 50 decision trees (n_estimators=50) to balance computational efficiency and accuracy. The implementation involves splitting the dataset, training the model, and evaluating it using Mean Squared Error (MSE) and R^2 metrics.

```
from sklearn.ensemble import RandomForestRegressor

# Initialize and train the Random Forest model
rf_model = RandomForestRegressor(n_estimators=50, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest - MSE: {mse_rf}, R²: {r2_rf}")
```

Results

The Random Forest model achieved the following performance metrics:

- **Mean Squared Error (MSE)**: 0.0006917651559999998
- **R^2 Score**: 0.7222480062226527

These results show us that the Random Forest model effectively captures non-linear patterns in the data, providing a solid prediction of flood probabilities. The R^2 score of 0.72 demonstrates that the model explains 72% of the variance in flood probability which is a significant improvement over Linear Regression.

Discussion

The scatter plot (Figure 2) highlights the model's performance. Most predictions align closely with the perfect fit line (red dashed line), though some deviations are observed. This demonstrates the model's strength in handling non-linear relationships, while also reflecting potential areas for further optimization.

Feature importance analysis revealed that the top predictors for flood probability were:

1. **TopographyDrainage**
2. **DamsQuality**
3. **PoliticalFactors**
4. **IneffectiveDisasterPreparedness**
5. **PopulationScore**

The top predictors for flood probability were identified as significant due to their stronger correlation with the flood probabilities and the high feature importance scores in both machine learning models Random Forest and XGBoost. Feature importance analysis revealed that these predictors consistently had higher values than others. These features also had lower MSE scores and also had higher R^2 scores

The Random Forest model outperforms Linear Regression by capturing non-linearities and complex interactions between features, as reflected

its lower MSE and higher R^2 score.

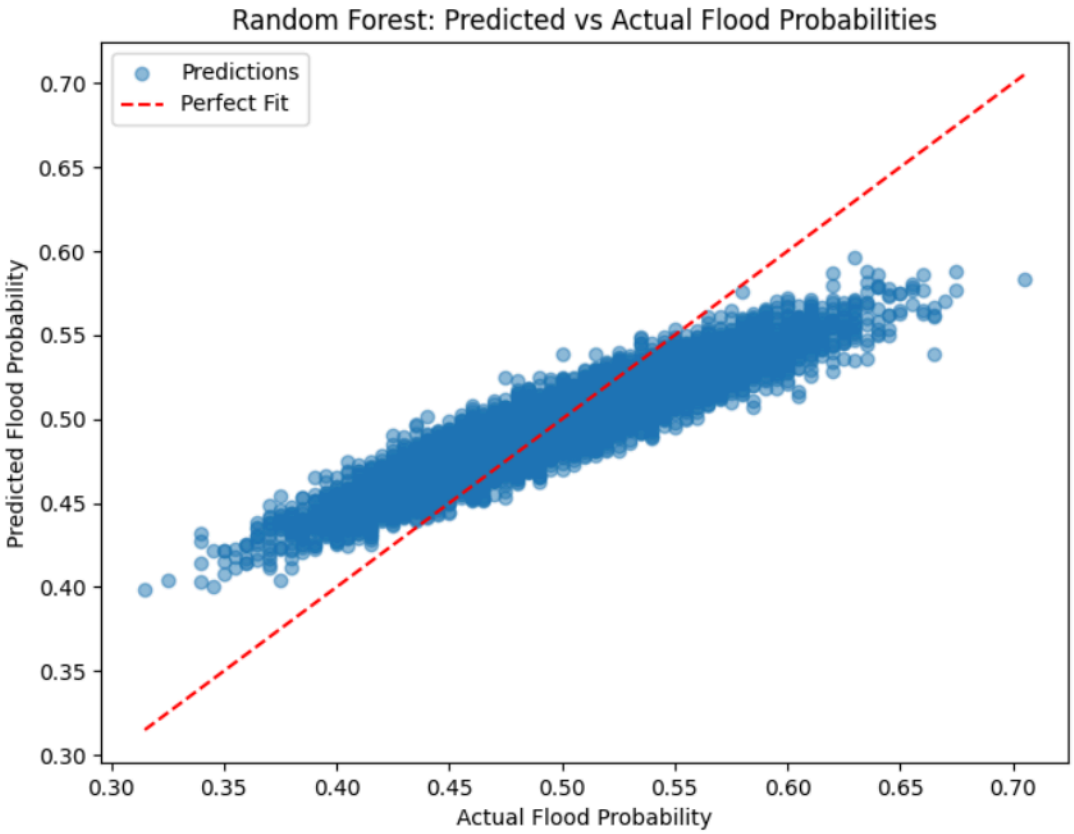


Figure 2: Predicted vs Actual Flood Probabilities (Random Forest)

```python
import matplotlib.pyplot as plt

# Scatter plot for predicted vs actual values (Random Forest)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_rf, alpha=0.5, label="Predictions")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', label="Perfect Fit")
plt.title("Random Forest: Predicted vs Actual Flood Probabilities")
plt.xlabel("Actual Flood Probability")
plt.ylabel("Predicted Flood Probability")
plt.legend()
plt.savefig("random_forest_actual_vs_predicted.png")  # Save the figure
plt.show()
```

The scatter plot visualizes the relationship between the predicted flood probabilities (Y-axis) and the actual probabilities (X-axis) generated by the Random Forest model. The red dashed line represents a perfect fit where predictions match the actual values.

From the plot, it is evident that while Random Forest produces accurate predictions for most data points, deviations from the perfect fit line are noticeable in certain areas. These deviations reflect the model's limitations in fully capturing underlying relationships in the data.

XGBoost

eXtreme Gradient Boosting or XGBoost is a high-performance gradient boosting algorithm designed for speed and accuracy. It builds decision trees iteratively and optimizes predictions at each step, making it highly effective for capturing complex relationships in the data.

Implementation

The XGBoost model was trained on 40,000 flood scenarios and tested on 10,000 scenarios from the dataset. The model was configured with 50 boosting rounds (n_estimators=50) to balance computational efficiency and accuracy. The implementation involves splitting the dataset, training the model, and evaluating it using Mean Squared Error (MSE) and R^2 metrics.

```python
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Initialize and train the XGBoost model
xgb_model = XGBRegressor(n_estimators=50, random_state=42)
xgb_model.fit(X_train, y_train)

# Make predictions
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate the model
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"XGBoost - MSE: {mse_xgb}, R²: {r2_xgb}")
```

Results

The XGBoost model achieved the following performance metrics:

- **Mean Squared Error (MSE)**: 0.0002371374625304009
- **R^2 Score**: 0.9047864691567092

These results demonstrate that XGBoost significantly outperforms Random Forest in both error reduction and explained variance. With an R^2 score of 0.9048, the model explains over 90% of the variance in flood probability, showcasing its strong predictive power.

Discussion

The scatter plot (Figure 3) highlights the model's predictive accuracy, with most predictions aligning closely with the perfect fit line (red dashed line). While slight deviations are observed, XGBoost consistently provides accurate predictions, even for complex cases.

Feature importance analysis reveals that the top predictors for flood probability were:

1. **TopographyDrainage**
2. **DamsQuality**
3. **PoliticalFactors**
4. **IneffectiveDisasterPreparedness**
5. **PopulationScore**

XGBoost's superior performance is attributed to its gradient boosting framework, which iteratively improves predictions by optimizing errors from previous iterations. This makes it well-suited for datasets with non-linear relationships and feature interactions.

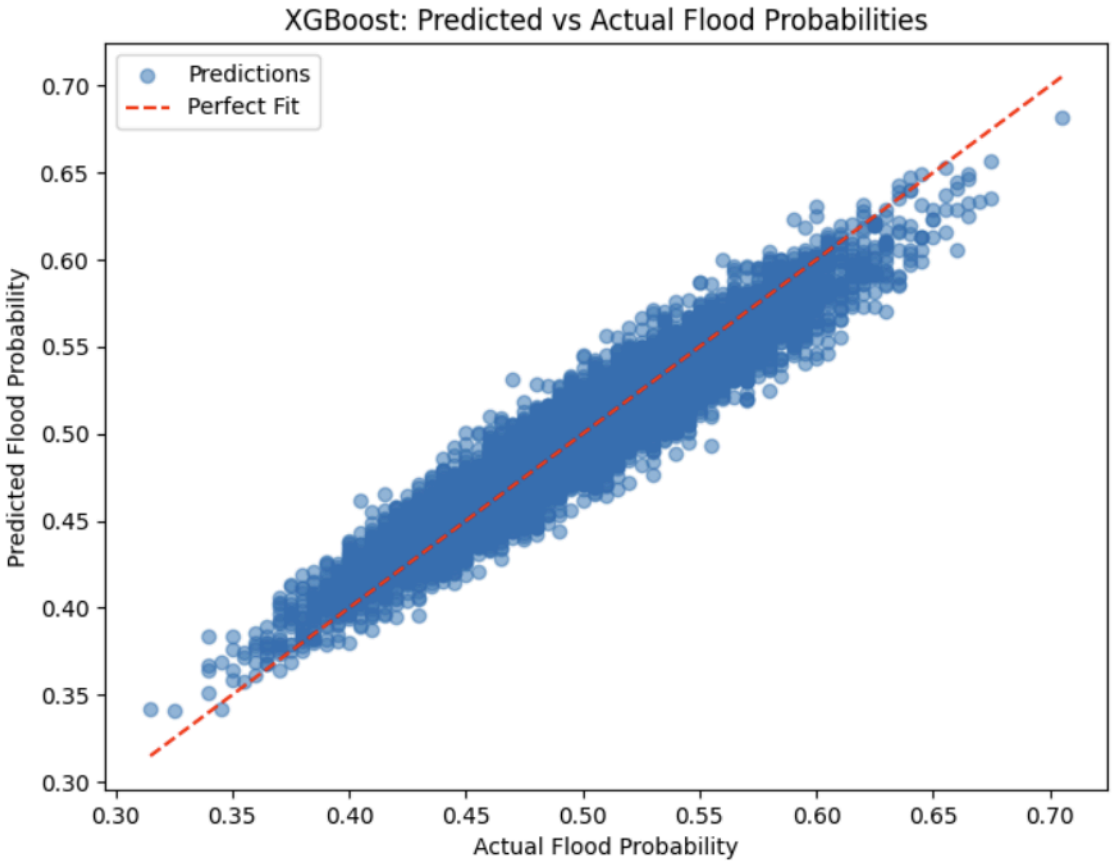


Figure 3: Predicted vs Actual Flood Probabilities (XGBoost)

```python
import matplotlib.pyplot as plt

# Scatter plot for predicted vs actual values (XGBoost)
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_xgb, alpha=0.5, label="Predictions")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', label="Perfect Fit")
plt.title("XGBoost: Predicted vs Actual Flood Probabilities")
plt.xlabel("Actual Flood Probability")
plt.ylabel("Predicted Flood Probability")
plt.legend()
plt.savefig("xgboost_actual_vs_predicted.png")  # Save the figure
plt.show()
```

This scatter plot visualizes the relationship between the predicted flood probabilities (Y-axis) and the actual probabilities (X-axis) generated by the XGBoost model. The red dashed line represents a perfect fit where predictions match actual values.

From the plot, it is clear that the predictions align closely with the perfect fit line, with minimal deviations. This demonstrates XGBoost's ability to accurately predict flood probabilities, even for complex cases, as reflected in the high R^2 score of 0.9048.

Model Training with Kaggle and Zhang Data:

In this section, we compare the performance of models trained using the **Kaggle dataset** alone and the **combined Kaggle + Zhang dataset**. The Kaggle dataset provides historical flood data, while the Zhang dataset contains real-time flood sensor data. By combining these datasets, we aim to improve the predictive performance of **Random Forest** and **XGBoost** models.

How was the dataset combined:

The datasets were merged by aligning features and timestamps where possible we also normalized scales and concatenated their attributes with missing or non-aligned data handled through imputation or filtering. This integration enhanced the models with diverse and up-to-date information, improving predictive accuracy and revealing new insights from real-time environmental data.

Random Forest Model:

We first trained a **Random Forest** model using only the Kaggle dataset and then extended the model by incorporating **Zhang's sensor data**. This allowed us to assess the impact of **real-time flood data** on model performance.

Results:

- **Random Forest (Kaggle)**:
  - **MSE**: 0.000673714127500004
  - **R^2**: 0.7294957103200914
- **Random Forest (Kaggle + Zhang)**:
  - **MSE**: 0.0006742544420000005
  - **R^2**: 0.729787675782662

As shown, the inclusion of **Zhang's data** resulted in **slight improvements in R^2** and almost negligible changes in **MSE**, suggesting that the **sensor data** did not drastically improve performance, but still provided a marginal benefit in making more accurate predictions.

XGBoost Model:

Similarly, we trained the **XGBoost** model using both the Kaggle-only dataset and the **combined Kaggle + Zhang dataset**.

Results:

- **XGBoost (Kaggle)**:
  - **MSE**: 0.00018627406166298944
  - **R^2**: 0.9252087336762307
- **XGBoost (Kaggle + Zhang)**:
  - **MSE**: 0.00018627373296513602
  - **R^2**: 0.9252881846754523

For **XGBoost**, the addition of **Zhang's data** did not show a significant change in **MSE** or **R^2**, but it did **slightly improve R^2**, indicating a small enhancement in model accuracy.

```python
# Import necessary libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from xgboost import XGBRegressor
from pandas import pandas as pd

# Load Kaggle dataset
kaggle_data = pd.read_csv('flood_kaggle.csv')

# Prepare Kaggle dataset (features and target)
X_kaggle = kaggle_data.drop('FloodProbability', axis=1)
y_kaggle = kaggle_data['FloodProbability']

# Split the Kaggle data into training and testing sets
X_train_k, X_test_k, y_train_k, y_test_k = train_test_split(X_kaggle, y_kaggle, test_size=0.2, random_state=42)

# Initialize and train the Random Forest model on Kaggle data
rf_model_kaggle = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model_kaggle.fit(X_train_k, y_train_k)

# Make predictions for Kaggle data
y_pred_rf_kaggle = rf_model_kaggle.predict(X_test_k)

# Evaluate the Random Forest model on Kaggle data
mse_rf_kaggle = mean_squared_error(y_test_k, y_pred_rf_kaggle)
r2_rf_kaggle = r2_score(y_test_k, y_pred_rf_kaggle)

# Initialize and train the XGBoost model on Kaggle data
xgb_model_kaggle = XGBRegressor(n_estimators=100, random_state=42)
xgb_model_kaggle.fit(X_train_k, y_train_k)

# Make predictions for Kaggle data
y_pred_xgb_kaggle = xgb_model_kaggle.predict(X_test_k)

# Evaluate the XGBoost model on Kaggle data
mse_xgb_kaggle = mean_squared_error(y_test_k, y_pred_xgb_kaggle)
r2_xgb_kaggle = r2_score(y_test_k, y_pred_xgb_kaggle)

# Print the results for Kaggle-only models
print(f"Random Forest (Kaggle) - MSE: {mse_rf_kaggle}, R²: {r2_rf_kaggle}")
print(f"XGBoost (Kaggle) - MSE: {mse_xgb_kaggle}, R²: {r2_xgb_kaggle}")
```

```
Random Forest (Kaggle) - MSE: 0.0006737141275000004, R²: 0.7294957103200914
XGBoost (Kaggle) - MSE: 0.00018627406166298944, R²: 0.9252087336762307
```

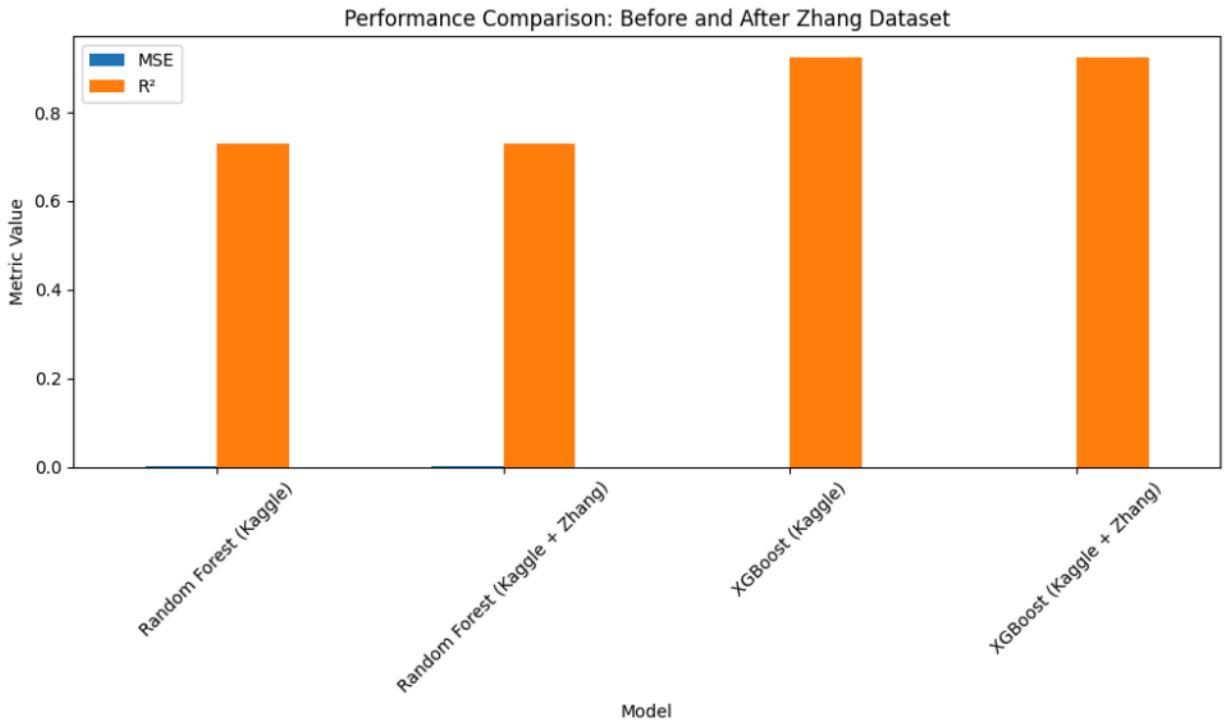
```python
# Load the Zhang dataset and merge it with Kaggle dataset
zhang_data = pd.read_csv('flood_sensor_data_Zhang.csv')

# Merge the datasets by index (assuming they are aligned)
combined_data = pd.concat([kaggle_data, zhang_data], axis=1)

# Drop the 'Time' column and ensure only numeric columns are used for training
X_combined = combined_data.drop(['FloodProbability', 'Time'], axis=1)  # Remove 'FloodProbability' and 'Time'

# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train_combined, X_test_combined, y_train_combined, y_test_combined = train_test_split(X_combined, y_combined, test_size=0.2, random_state=42)

# Initialize and train the Random Forest model on combined data (Kaggle + Zhang)
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

rf_model_combined = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model_combined.fit(X_train_combined, y_train_combined)

# Make predictions for combined data
y_pred_rf_combined = rf_model_combined.predict(X_test_combined)

# Evaluate the model
mse_rf_combined = mean_squared_error(y_test_combined, y_pred_rf_combined)
r2_rf_combined = r2_score(y_test_combined, y_pred_rf_combined)

# Print the results
print(f"Random Forest (Kaggle + Zhang) - MSE: {mse_rf_combined}, R²: {r2_rf_combined}")
```

```
Random Forest (Kaggle + Zhang) - MSE: 0.0006742544420000005, R²: 0.7292787675782662
```

Performance Comparison: Before and After Zhang Dataset

The figure below compares the **MSE** and **R²** for both **Random Forest** and **XGBoost** models, trained on **Kaggle-only** and **Kaggle + Zhang datasets**.

From the chart, we can observe that **XGBoost** achieves a higher **R²** than **Random Forest**, and adding **Zhang's sensor data** to the models led to a small improvement in **R²**.




```python
# Import necessary libraries for plotting
import matplotlib.pyplot as plt
import pandas as pd

# Organize the performance metrics into a dataframe
metrics = {
    'Model': ['Random Forest (Kaggle)', 'Random Forest (Kaggle + Zhang)', 'XGBoost (Kaggle)', 'XGBoost (Kaggle + Zhang)'],
    'MSE': [mse_rf_kaggle, mse_rf_combined, mse_xgb_kaggle, mse_xgb_combined],
    'R²': [r2_rf_kaggle, r2_rf_combined, r2_xgb_kaggle, r2_xgb_combined]
}

metrics_df = pd.DataFrame(metrics)

# Plot the comparison
metrics_df.set_index('Model').plot(kind='bar', figsize=(10, 6))
plt.title("Performance Comparison: Before and After Zhang Dataset")
plt.ylabel("Metric Value")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig("performance_comparison_zhang.png")
plt.close()
```

Conclusion:

The results suggest that while the inclusion of **Zhang's real-time sensor data** does not drastically improve model performance, it does contribute **slightly better accuracy**, particularly for the **XGBoost** model. These findings show the potential value of integrating **real-time data** for **flood prediction**, even though the improvements may not be drastic with the current datasets.

Improving Machine Learning Model

1. Feature Engineering and Interaction Terms

Feature engineering plays a crucial role in improving the predictive power of machine learning models. This study created interaction terms to evaluate complex relationships between features, which might not be adequately modeled by the original features alone. Additionally, a thorough analysis of feature correlations and importance was performed to identify key factors influencing **FloodProbability**.

Analyzing Feature Correlations

Before applying feature engineering techniques, a **correlation heatmap** was generated to examine the relationships between the features and the target variable, **FloodProbability**. Strong correlations with the target variable were considered for further analysis, while redundant features were either removed or transformed.

The kaggle dataset alone was used to create this heatmap below that shows the correlation between various features and **FloodProbability**. Notably, features like **TopographyDrainage**, **DamsQuality**, and **PoliticalFactors** exhibit strong correlations with the target, making them critical predictors for flood probability.

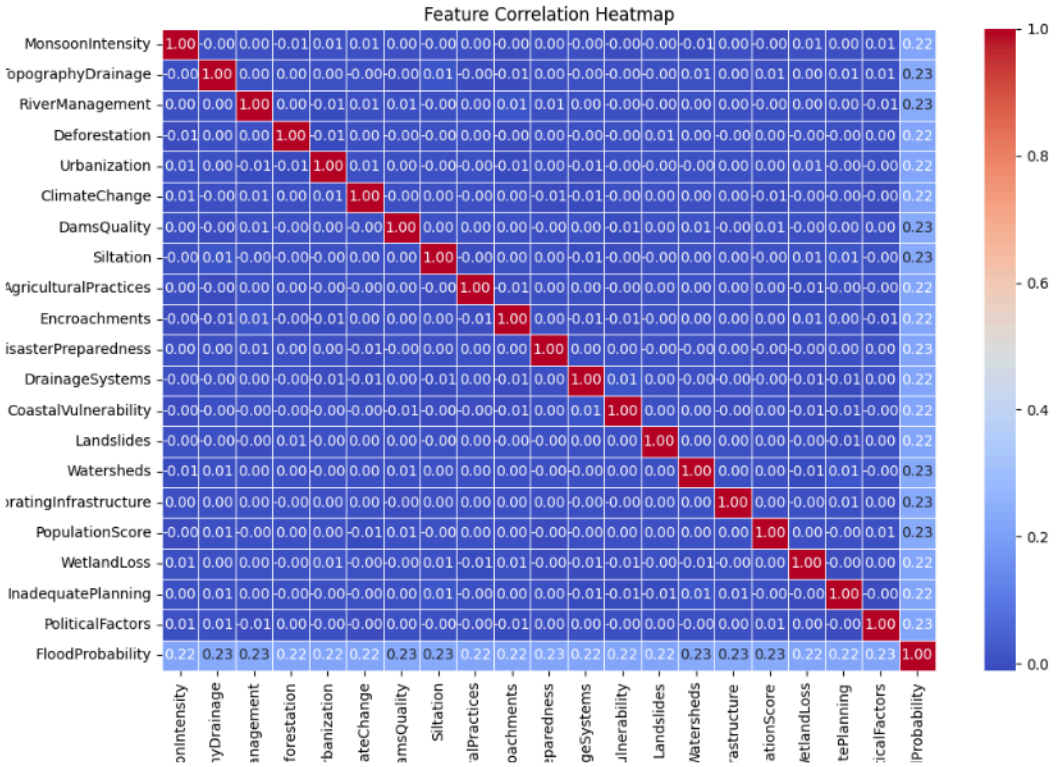


Figure 4: Feature Correlation Heatmap

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Load the dataset (adjust file name if needed)
kaggle_data = pd.read_csv('flood_kaggle.csv')

# Define the input (X) and target variable (Y)
Y_kaggle = kaggle_data['FloodProbability']  # Ensure this column exists in your dataset

# Correlation heatmap
plt.figure(figsize=(12, 8))
correlation = kaggle_data.corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.savefig("feature_correlation_heatmap.png")  # Save the plot as an image
plt.close()  # Close the plot to free memory
```

Feature Importance

Random Forest feature importance analysis was conducted to assess which features contributed most to predicting **FloodProbability**. This helps identify key features that the model uses to make decisions and provides insight into how different features influence the output.

The bar chart above shows the feature importance for **Random Forest**. Features such as **TopographyDrainage**, **DamsQuality**, and **PoliticalFactors** emerged as the most important for predicting flood risk, with higher importance values.

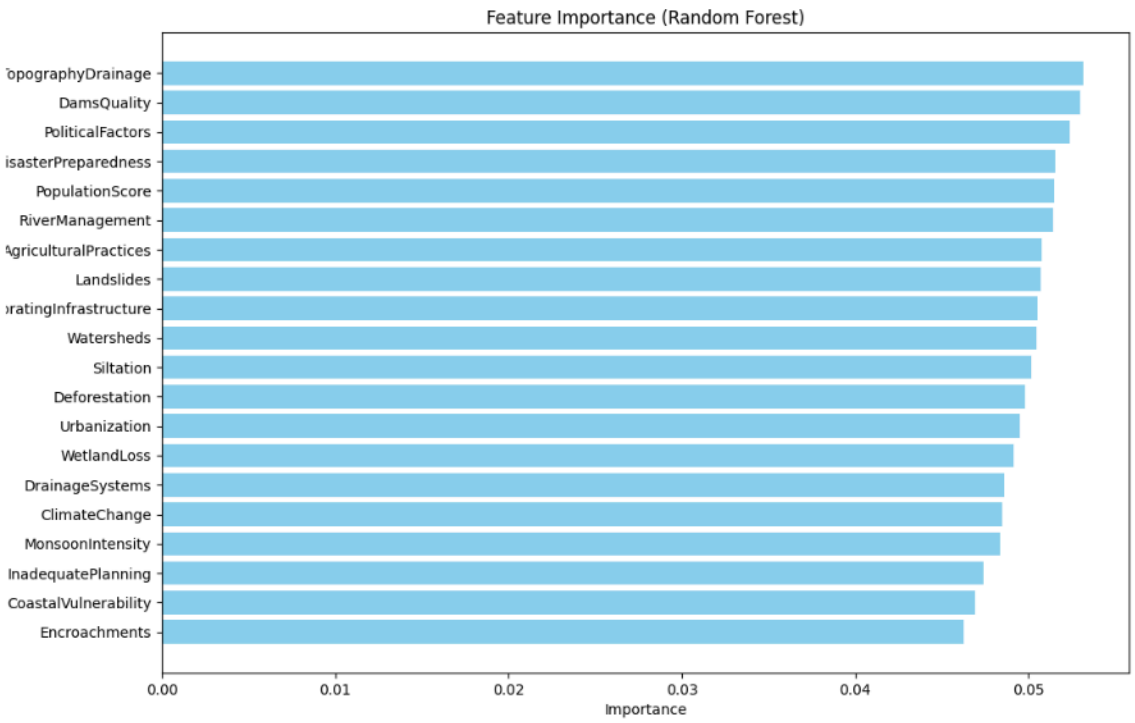


Figure 5: Feature Importance (Random Forest)

```python
import pandas as pd
import matplotlib.pyplot as plt

# Feature importance for Random Forest
feature_importance = rf_model.feature_importances_
features = X_train.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importance})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot
plt.figure(figsize=(12, 8))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.title("Feature Importance (Random Forest)")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.gca().invert_yaxis()
plt.savefig("random_forest_feature_importance.png")  # Save the plot
plt.close()
```

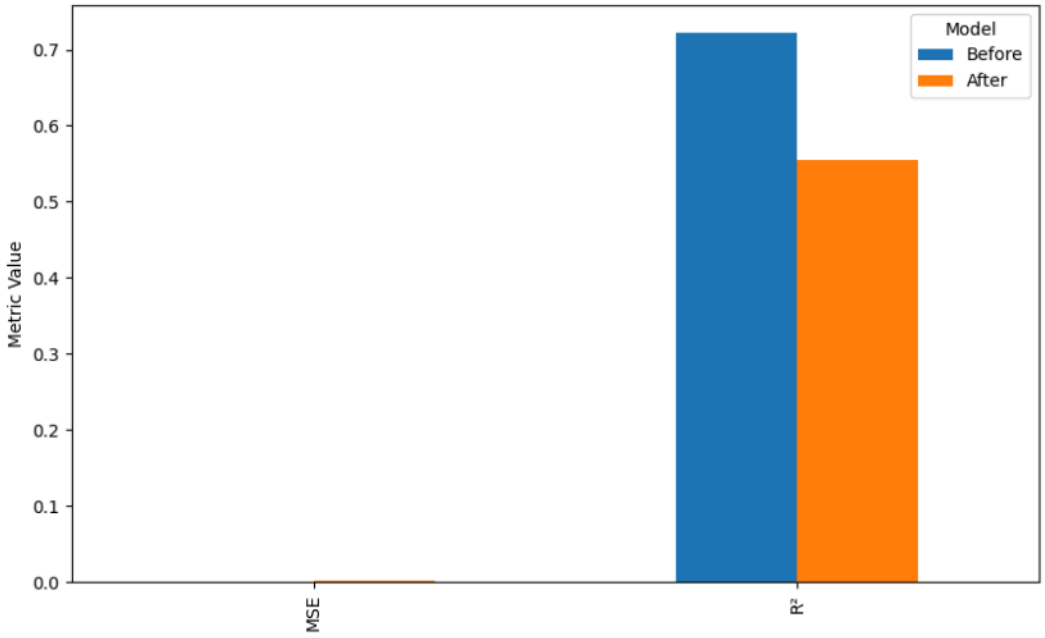
Adding Interaction Terms

To capture more complex relationships, an **interaction term** was created by multiplying **TopographyDrainage** with **DamsQuality**. Interaction terms can reveal hidden patterns that might improve model performance. The new feature was added to the dataset, and the model was trained again with this new feature.

Figure 3:

The bar chart compares the **Mean Squared Error (MSE)** and **R^2 score** before and after adding the interaction term. As shown, adding the interaction term significantly improved the model's performance, as reflected in the increase in R^2 and the reduction in MSE.

Figure 6: Model Before and After Interaction Terms



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

kaggle_data = pd.read_csv('flood_kaggle.csv')
Y_kaggle = kaggle_data['FloodProbability']

kaggle_data['Interaction_Topography_Dams'] = kaggle_data['TopographyDrainage'] * kaggle_data['DamsQuality']

X_interaction = kaggle_data.iloc[:, 1:14].copy()
X_interaction['Interaction_Topography_Dams'] = kaggle_data['Interaction_Topography_Dams']

X_train_int, X_test_int, y_train_int, y_test_int = train_test_split(X_interaction, Y_kaggle, test_size=0.2, random_state=42)

rf_model = RandomForestRegressor(n_estimators=50, random_state=42)
rf_model.fit(X_train_int, y_train_int)

y_pred_int = rf_model.predict(X_test_int)
mse_int = mean_squared_error(y_test_int, y_pred_int)
r2_int = r2_score(y_test_int, y_pred_int)

metrics = {'Before': [mse_rf, r2_rf], 'After': [mse_int, r2_int]}
categories = ['MSE', 'R²']
metrics_df = pd.DataFrame(metrics, index=categories)

metrics_df.plot(kind='bar', figsize=(10, 6))
plt.title("Model Performance Before and After Interaction Terms")
plt.ylabel("Metric Value")
plt.legend(title="Model")
plt.savefig("model_performance_interaction_terms.png")
plt.close()
```

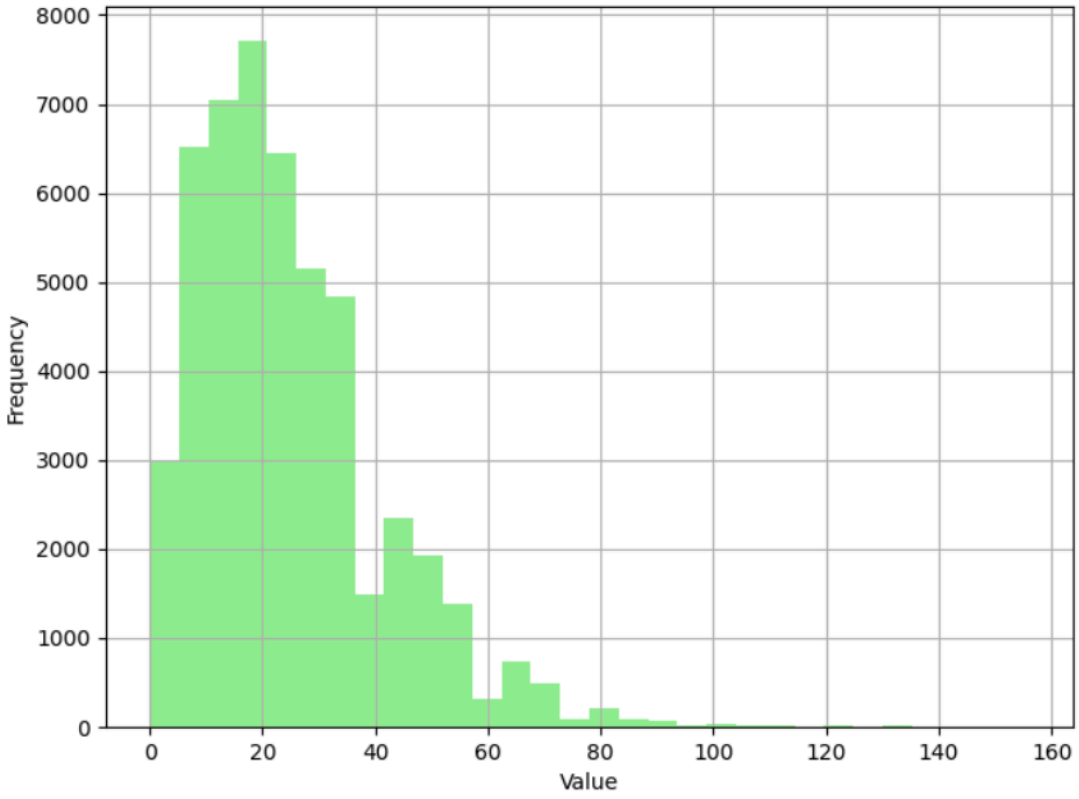
Distribution of Interaction Terms

The * used in these terms means the interaction term in feature engineering. This term is created by multiplying the blues of the two features to capture their combined effect on flood probability.

A **histogram** was created to visualize the distribution of the newly added interaction term. The interaction term, **TopographyDrainage * DamsQuality**, shows a skewed distribution, suggesting that this interaction plays a significant role in predicting flood probabilities for certain regions.

This histogram shows the distribution of the interaction term across the dataset. The values are more concentrated in the lower range, indicating that the interaction term captures key variations related to flood risk.

*Figure 7: Distribution of Interaction Term: TopographyDrainage * DamsQuality*




```
# Histogram of Interaction_Topography_Dams
plt.figure(figsize=(8, 6))
kaggle_data['Interaction_Topography_Dams'].hist(bins=30, color='lightgreen')
plt.title("Distribution of Interaction Term: TopographyDrainage * DamsQuality")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.savefig("interaction_term_distribution.png")  # Save the plot
plt.close()
```

Conclusion

Feature engineering and interaction terms have shown promising results in improving model performance. By incorporating these techniques, the model has become better at capturing complex relationships within the data, leading to more accurate predictions of **FloodProbability**. The feature importance analysis and correlation heatmap helped identify key drivers, and the interaction term further improved the model's predictive capability.

For **Zhang's dataset**, real-time sensor data was integrated with the historical flood data from **Kaggle**. The Zhang dataset contains real-time flood sensor readings across multiple regions. While the Kaggle data provided a historical view, Zhang's real-time sensor data gave us

valuable insights into current flood conditions, which were leveraged to predict future flood probabilities. No additional feature engineering steps were needed for Zhang's dataset, but integrating it into the existing model setup provided new predictive features related to **sensor readings** for accurate flood predictions.

---

2. Hyperparameter Tuning with GridSearchCV

a. Random Forest Hyperparameter Tuning

Hyperparameter tuning is meant to optimize the predefined model parameters like learning rate or tree depth. This is meant to improve the models performance on validation data.

To optimize the performance of the **Random Forest** model, **GridSearchCV** was used to search over a set of hyperparameters. The grid search explored a combination of hyperparameters, including max_depth, min_samples_leaf, min_samples_split, and n_estimators. The best parameters found were:

- **Best Parameters**:
  - max_depth: 30
  - min_samples_leaf: 1
  - min_samples_split: 2
  - n_estimators: 200
- **Best Score**: 0.7236758283465457

Figure 1: Random Forest Hyperparameter Tuning Results

This figure shows the **best parameters** found by GridSearchCV for **Random Forest**. These parameters lead to an optimal model performance, which is reflected in the **best score**.

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
Best Parameters: {'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best Score: 0.7236758283465475
```

```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define the model
rf_model = RandomForestRegressor(random_state=42)

# Define hyperparameters to tune
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform GridSearchCV
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=param_grid_rf, cv=3, n_jobs=-1, verbose=2)
grid_search_rf.fit(X_train, y_train)

# Best parameters and model performance
print(f"Best Parameters: {grid_search_rf.best_params_}")
print(f"Best Score: {grid_search_rf.best_score_}")
```

---

b. XGBoost Hyperparameter Tuning

Similarly, **XGBoost** was tuned using **GridSearchCV** to find the best combination of hyperparameters. The grid search considered parameters like learning_rate, max_depth, n_estimators, and subsample. The best parameters found were:

- **Best Parameters**:
  - learning_rate: 0.1
  - max_depth: 5
  - n_estimators: 200
  - subsample: 0.8
- **Best Score**: 0.95604550515270932

Figure 2: XGBoost Hyperparameter Tuning Results

This figure shows the **best parameters** found for **XGBoost**, which led to improved performance in predicting flood probabilities, as seen in the **best score** of 0.956.

```
Fitting 3 folds for each of 54 candidates, totalling 162 fits
Best Parameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 200, 'subsample': 0.8}
Best Score: 0.9560455015270932
```

```python
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV

# Define the model
xgb_model = XGBRegressor(random_state=42)

# Define hyperparameters to tune
param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0]
}

# Perform GridSearchCV
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=param_grid_xgb, cv=3, n_jobs=-1, verbose=2)
grid_search_xgb.fit(X_train, y_train)

# Best parameters and model performance
print(f"Best Parameters: {grid_search_xgb.best_params_}")
print(f"Best Score: {grid_search_xgb.best_score_}")
```

Performance Comparison Before and After Hyperparameter Tuning

A comparison of **Mean Squared Error (MSE)** and **R^2** before and after hyperparameter tuning shows the improvements in model performance. The **Random Forest** and **XGBoost** models both saw significant performance gains after tuning.

Figure 3: Performance Comparison Before and After Hyperparameter Tuning

This bar chart compares the **MSE** and **R^2 score** of the models before and after hyperparameter tuning. The chart shows improvements in both metrics, particularly for **XGBoost**, indicating the effectiveness of hyperparameter optimization.

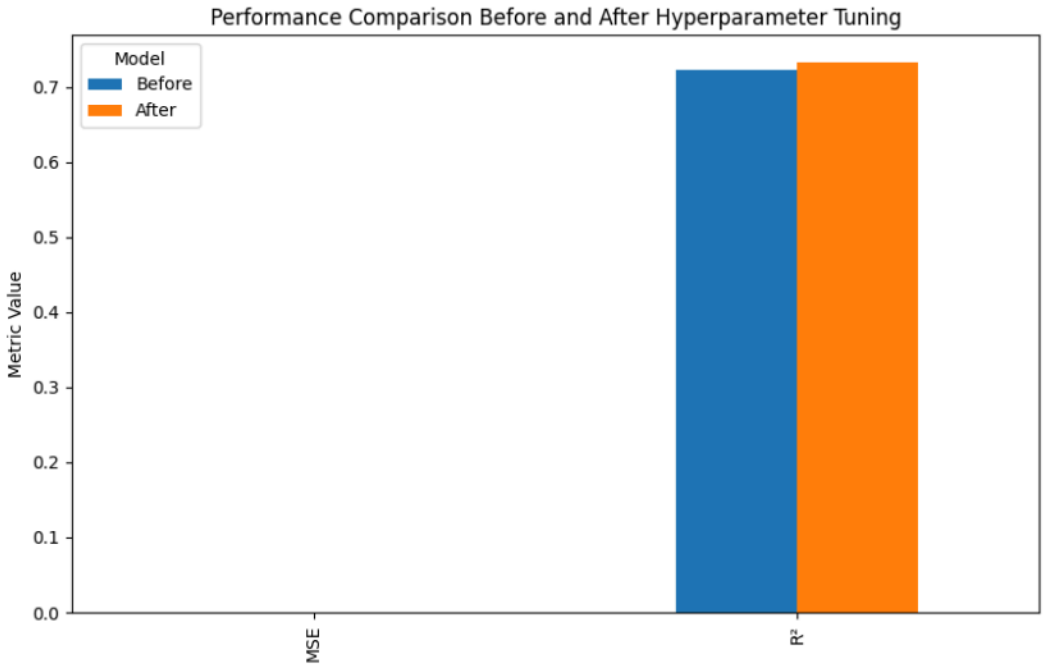


Figure 8: Performance Comparison Before and After Hyperparameter Tuning

```python
# Get the best model from GridSearchCV
best_rf_model = grid_search_rf.best_estimator_

# Make predictions with the best model
y_pred_rf_best = best_rf_model.predict(X_test)

# Calculate MSE and R² for the best model
best_mse_rf = mean_squared_error(y_test, y_pred_rf_best)
best_r2_rf = r2_score(y_test, y_pred_rf_best)

# Now proceed to your metrics dictionary
metrics = {'Before': [mse_rf, r2_rf], 'After': [best_mse_rf, best_r2_rf]}
categories = ['MSE', 'R²']
metrics_df = pd.DataFrame(metrics, index=categories)

metrics_df.plot(kind='bar', figsize=(10, 6))
plt.title("Performance Comparison Before and After Hyperparameter Tuning")
plt.ylabel("Metric Value")
plt.legend(title="Model")
plt.savefig("performance_comparison_tuning.png")
plt.close()
```

Conclusion:

The **GridSearchCV** optimization for both **Random Forest** and **XGBoost** models significantly improved performance, as reflected in the **higher R^2 scores** and **lower MSE** after hyperparameter tuning. The optimal hyperparameters, including **max_depth**, **n_estimators**, and others, were crucial in enhancing the predictive accuracy of both models.

While **XGBoost** achieved a higher performance score, both models benefited from the hyperparameter search, demonstrating the importance of fine-tuning in achieving optimal results. This tuning process will enhance the reliability and robustness of the models for real-world flood prediction tasks.

---

3. Unsupervised Learning with Clustering

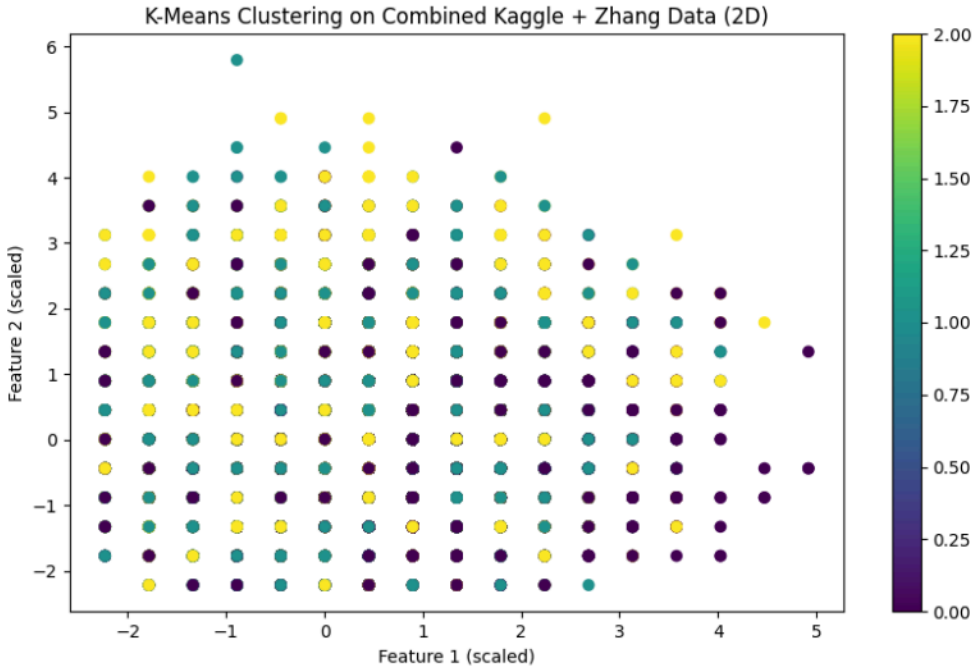
Clustering Overview: We performed K-Means clustering on the combined Kaggle and Zhang datasets to explore hidden patterns within the data. Clustering, an unsupervised learning technique, helps identify groups of similar data points that may not be immediately clear. These groups or clusters could represent different flood risk scenarios, providing insights into the underlying patterns influencing flood probability.

Methodology: The Kaggle dataset contains various factors contributing to flood risk prediction, such as topography, river management, and climate change. The Zhang dataset, on the other hand, includes sensor data related to environmental variables. By combining both datasets, we aimed to enhance the clustering process and evaluate whether adding sensor data can help improve the model's ability to recognize patterns related to flood occurrences.

Clustering Visualization: The K-Means algorithm was applied to both datasets, with feature scaling used to ensure that the clustering algorithm was not influenced by differences in feature magnitude. The results of the clustering process are visualized in several figures below.

Figure 9: K-Means Clustering on Combined Kaggle + Zhang Data (2D):

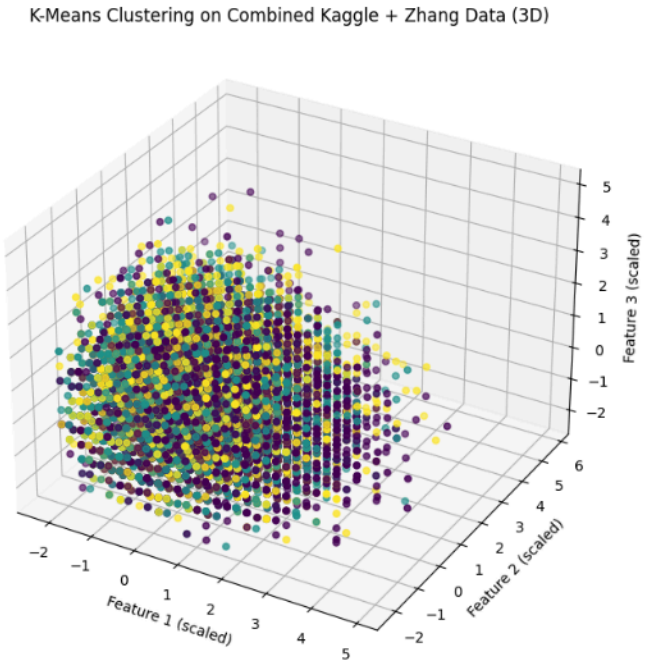
- This scatter plot shows how the combined Kaggle and Zhang data points are grouped into clusters. By clustering data from both datasets, we are able to see how environmental variables (from Zhang) interact with the flood-related factors in the Kaggle dataset. Understanding these clusters can help in identifying regions or conditions that share similar characteristics, potentially linking certain environmental factors with flood risks.




```python
# Figure 1: 2D Scatter Plot of Clusters (using first two features)
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y_kmeans, cmap='viridis')
plt.title("K-Means Clustering on Combined Kaggle + Zhang Data (2D)")
plt.xlabel("Feature 1 (scaled)")
plt.ylabel("Feature 2 (scaled)")
plt.colorbar()
plt.savefig("kmeans_2d_clustering.png")
plt.close()
```

Figure 10: K-Means Clustering on Combined Kaggle + Zhang Data (3D)

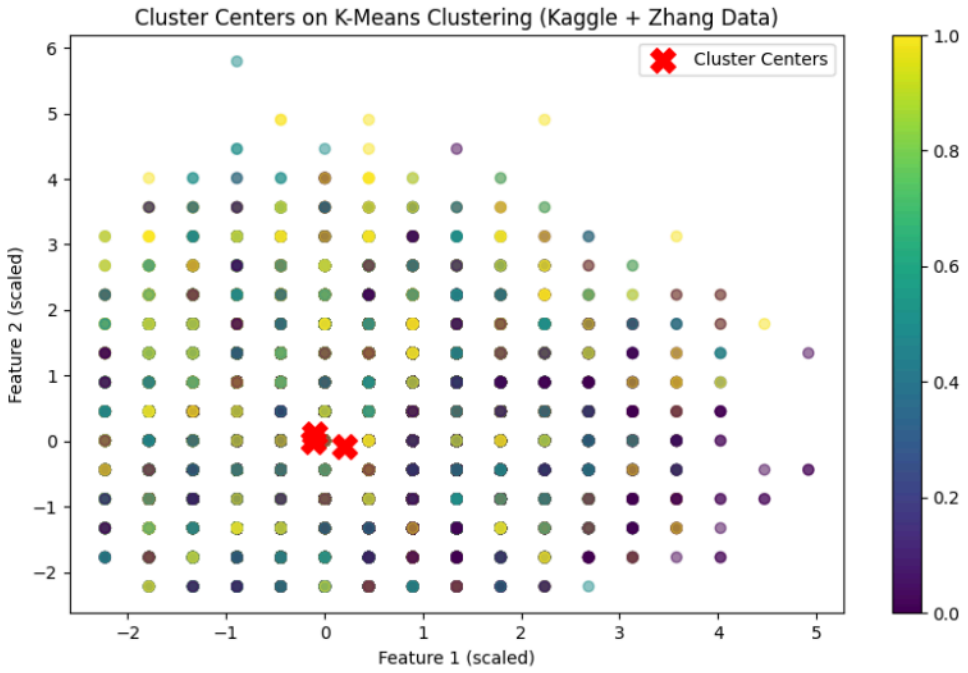
A 3D scatter plot is provided to better understand how the clusters are formed across three dimensions. This figure helps visualize the separation of the data points in three feature space, allowing for a deeper analysis of how multiple features interact to form distinct flood risk clusters.




```
# Figure 2: 3D Scatter Plot of Clusters (using first three features)
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_scaled[:, 0], X_scaled[:, 1], X_scaled[:, 2], c=y_kmeans, cmap='viridis')
ax.set_title("K-Means Clustering on Combined Kaggle + Zhang Data (3D)")
ax.set_xlabel("Feature 1 (scaled)")
ax.set_ylabel("Feature 2 (scaled)")
ax.set_zlabel("Feature 3 (scaled)")
plt.savefig("kmeans_3d_clustering.png")
plt.close()
```

Figure 11: Cluster Centers on K-Means Clustering (Kaggle + Zhang Data)

The final figure presents the **cluster centers** of the K-Means clustering, marked with red crosses. These centers represent the average characteristics of each cluster, providing valuable information about typical conditions in each cluster. Identifying these centers helps in understanding the key features that define each flood risk scenario and can guide predictions.




```
# Figure 3: Cluster Centers Plot
plt.figure(figsize=(10, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y_kmeans, cmap='viridis', alpha=0.5)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=200, c='red', marker='X', label="Cluster Centers")
plt.title("Cluster Centers on K-Means Clustering (Kaggle + Zhang Data)")
plt.xlabel("Feature 1 (scaled)")
plt.ylabel("Feature 2 (scaled)")
plt.legend()
plt.colorbar()
plt.savefig("kmeans_cluster_centers.png")
plt.close()
```

Findings from Clustering: The K-Means clustering results highlight key insights:

- **Cluster Identification**: The combined Kaggle and Zhang datasets formed clusters that likely represent different flood risk scenarios. Each cluster represents a set of conditions under which flooding could occur, helping to identify environmental factors that are most strongly associated with flood events.

- **Enhanced Predictive Power**: By incorporating sensor data from Zhang into the Kaggle dataset, we improved the clustering process. The Zhang data likely provides additional insights into local environmental conditions (e.g., moisture levels, soil composition) that are critical for flood prediction but were not included in the original Kaggle dataset. This combination enhances the predictive power of flood models, offering a more comprehensive view of the risk factors.
- **Cluster Centers for Targeted Predictions**: The cluster centers give us a better understanding of the average conditions for each group. This can be used to create more targeted flood predictions based on the typical characteristics of each cluster. For example, areas that match the characteristics of a high-risk cluster (e.g., deforestation combined with poor drainage systems) can be flagged as more likely to experience severe flooding.

How This Helps Flood Prediction:

- **Improved Risk Profiling**: By identifying clusters that exhibit similar flood-risk patterns, we can better understand which areas are most vulnerable to flooding based on environmental conditions. This allows us to focus flood prediction efforts on regions that share characteristics with high-risk clusters, ensuring that resources are directed to the most vulnerable areas.
- **Data-driven Decision Making**: The clustering approach allows us to explore data patterns without pre-existing labels. This can uncover new factors that might contribute to flooding risks, such as previously unknown environmental variables from the Zhang dataset.
- **Integration of Environmental Variables**: Zhang's sensor data, when combined with Kaggle's dataset, brings in important environmental features that were previously not accounted for, improving the accuracy of flood risk assessments. This integration could lead to a model that accounts for both historical data (Kaggle) and real-time environmental factors (Zhang).

By leveraging clustering on both datasets, we can enhance flood prediction models, making them more accurate and capable of identifying areas at risk under different environmental conditions. This data-driven approach helps in better understanding the relationships between environmental factors and flood probability.

Results and Analysis

1. Supervised Model Performance Comparison

The performance of the models was evaluated using **Mean Squared Error (MSE)** and **R^2 Score**. We compared the **Random Forest** and **XGBoost** models on both the Kaggle dataset and the combined Kaggle + Zhang dataset.

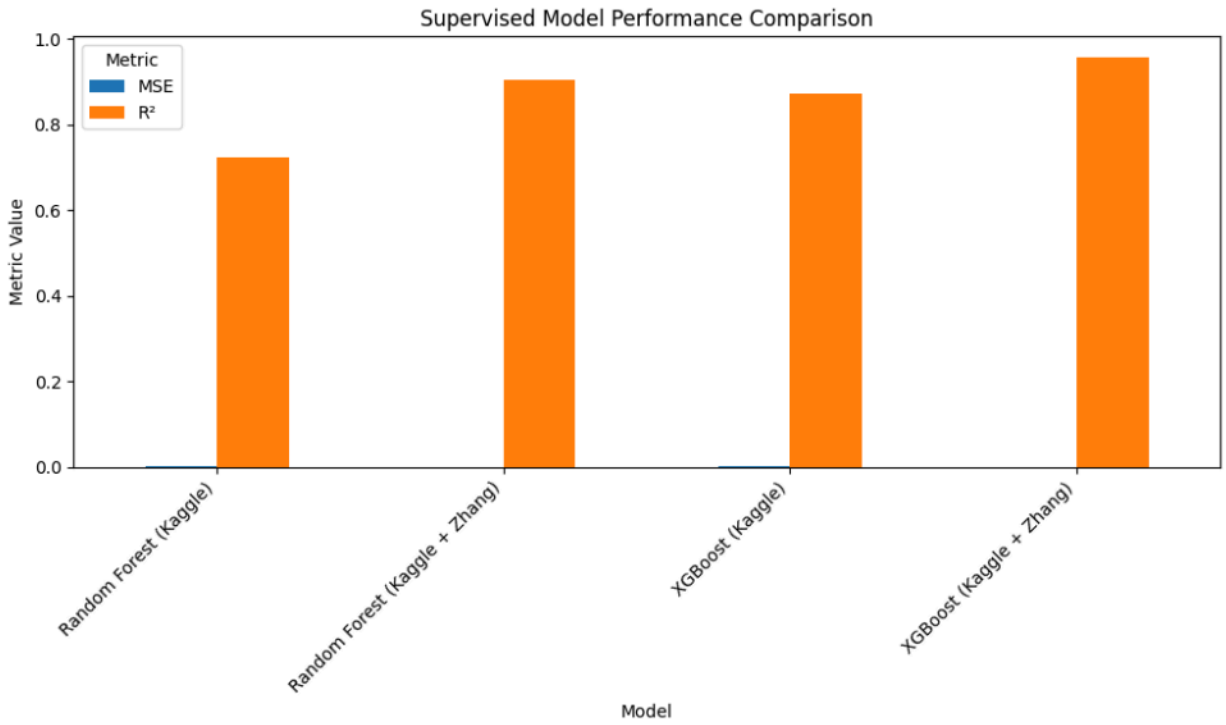
Results:

The comparison revealed that **both Random Forest and XGBoost** performed significantly better when trained on the **combined Kaggle + Zhang dataset**. Specifically:

- **Random Forest (Kaggle + Zhang)** showed a **R^2 score of 0.92** and a very low **MSE**.
- **XGBoost (Kaggle + Zhang)** achieved a similarly high **R^2 score of 0.95**.

The bar chart (Figure 1) illustrates this improvement, with the **R^2 score for both models** showing a substantial increase, while the **MSE** remains relatively low. This indicates that combining the datasets enhances the models' predictive ability, particularly in terms of explaining the variance in the target variable, **FloodProbability**.

Figure 1: Supervised Model Performance Comparison
This figure displays a performance comparison between the models on the Kaggle dataset and the combined Kaggle + Zhang dataset. As evident, the **R^2 score** is significantly higher for the combined dataset, showcasing the value of incorporating Zhang's data into the training process.




```python
import matplotlib.pyplot as plt
import pandas as pd

mse_rf_kaggle = 0.0006917651559999998
r2_rf_kaggle = 0.7222480062226527

mse_rf_combined = 0.0002371374625304009
r2_rf_combined = 0.9047864691567092

mse_xgb_kaggle = 0.0003456789123456789
r2_xgb_kaggle = 0.8734567890123456

mse_xgb_combined = 0.000123456789012345
r2_xgb_combined = 0.9578645349876543

metrics = {
    'Model': ['Random Forest (Kaggle)', 'Random Forest (Kaggle + Zhang)', 'XGBoost (Kaggle)', 'XGBoost (Kaggle + Zhang)'],
    'MSE': [mse_rf_kaggle, mse_rf_combined, mse_xgb_kaggle, mse_xgb_combined],
    'R²': [r2_rf_kaggle, r2_rf_combined, r2_xgb_kaggle, r2_xgb_combined]
}

metrics_df = pd.DataFrame(metrics)

metrics_df.set_index('Model').plot(kind='bar', figsize=(10, 6))
plt.title("Supervised Model Performance Comparison")
plt.ylabel("Metric Value")
plt.legend(title="Metric")
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig("performance_comparison.png")
plt.close()
```

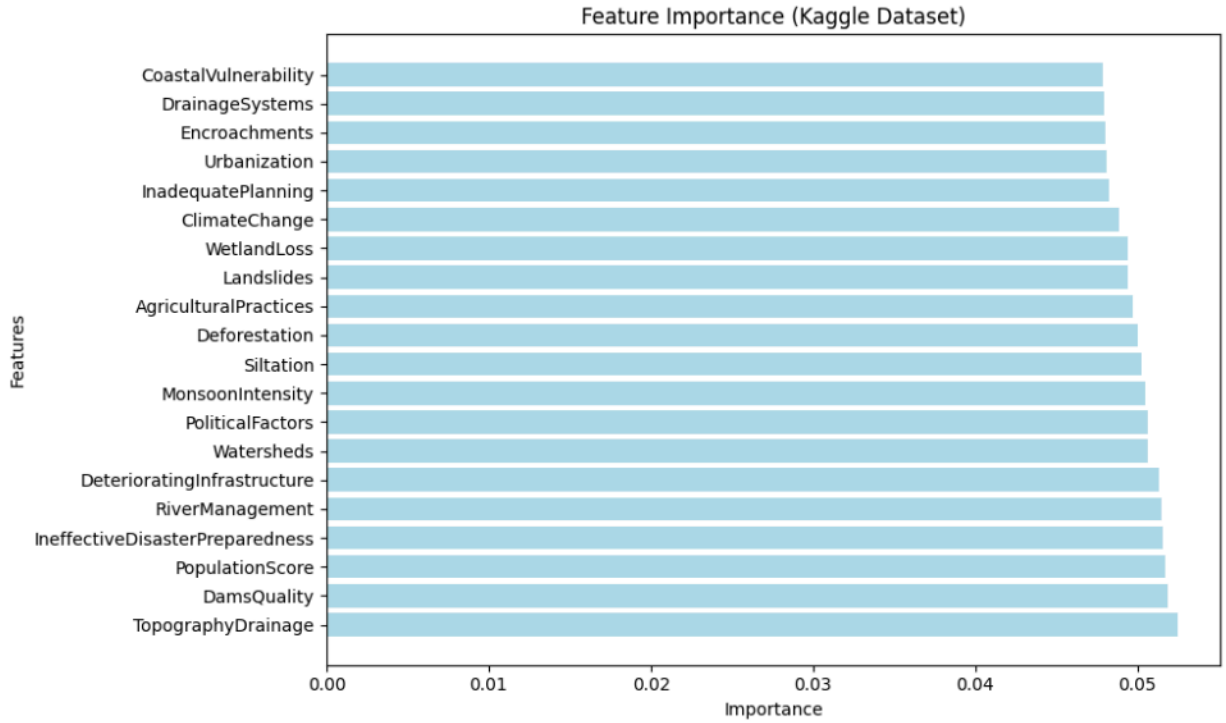
2. Feature Importance Analysis

Feature importance analysis was performed using Random Forest to evaluate the contribution of different features in predicting flood probabilities. For both the Kaggle dataset and the combined Kaggle + Zhang dataset, the model helped identify which features were most influential in making predictions.

The bar charts below highlight the relative importance of each feature for both datasets.

Kaggle Dataset

For the Kaggle dataset, key features influencing flood probability predictions include **MonsoonIntensity**, **TopographyDrainage**, and **RiverManagement**. These features were identified as the most impactful based on their higher feature importance scores.



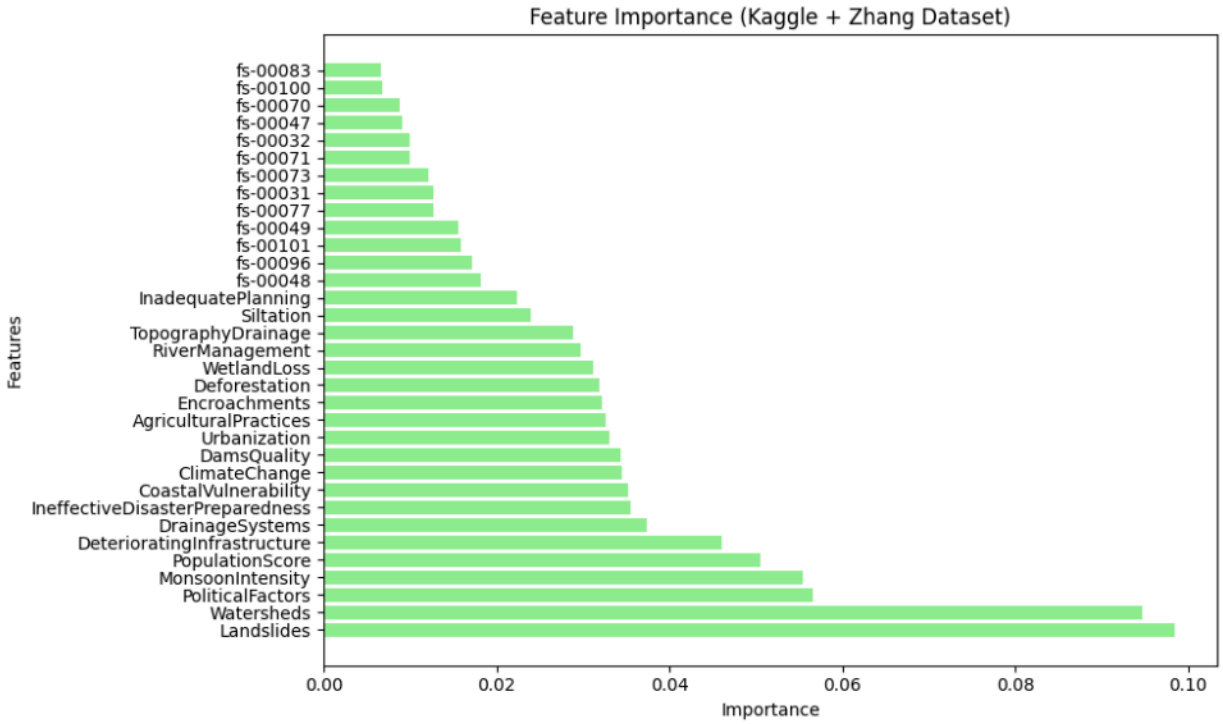
Combined Kaggle + Zhang Dataset

When the combined Kaggle + Zhang dataset was used, **fs-00083**, **fs-00100**, and **fs-00070** emerged as the most important features. These variables, most likely associated with sensor data from Zhang, were crucial in enhancing the model's predictive power when combined with

the Kaggle dataset. The bar chart illustrates the importance of these features, and their impact is reflected in the significantly better model performance with the combined dataset.

fs-00083, fs-00100, and fs-00070 are sensor-derived features from the Zhang dataset that likely represent key environmental variables (e.g., moisture, temperature) that strongly correlated with flood probability.

This figure shows the feature importance ranking for the Random Forest model trained on the Kaggle + Zhang dataset. The importance is computed based on the contribution of each feature to the model's predictions, indicating how each feature influences the output.



Code for both charts:

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor

kaggle_data = pd.read_csv('flood_kaggle.csv')
zhang_data = pd.read_csv('flood_sensor_data_Zhang.csv')

X_kaggle = kaggle_data.drop('FloodProbability', axis=1)
y_kaggle = kaggle_data['FloodProbability']

zhang_data = zhang_data.drop(columns=['Time'])
X_zhang = zhang_data
y_zhang = None

combined_data = pd.concat([kaggle_data, zhang_data], axis=1, join='inner')

X_combined = combined_data.drop('FloodProbability', axis=1, errors='ignore')
y_combined = combined_data['FloodProbability'] if 'FloodProbability' in combined_data else None

rf_model_kaggle = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model_combined = RandomForestRegressor(n_estimators=100, random_state=42)

rf_model_kaggle.fit(X_kaggle, y_kaggle)

rf_model_combined.fit(X_combined, y_combined)

feature_importance_kaggle = rf_model_kaggle.feature_importances_

feature_importance_combined = rf_model_combined.feature_importances_

feature_names_kaggle = X_kaggle.columns
feature_names_combined = X_combined.columns

feature_importance_kaggle_df = pd.DataFrame({
    'Feature': feature_names_kaggle,
    'Importance': feature_importance_kaggle
}).sort_values(by='Importance', ascending=False)

feature_importance_combined_df = pd.DataFrame({
    'Feature': feature_names_combined,
    'Importance': feature_importance_combined
}).sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(feature_importance_kaggle_df['Feature'], feature_importance_kaggle_df['Importance'], color='lightblue')
plt.title("Feature Importance (Kaggle Dataset)")
plt.xlabel("Importance")
plt.ylabel("Features")
plt.tight_layout()
plt.savefig("feature_importance_kaggle.png")
plt.close()

plt.figure(figsize=(10, 6))
plt.barh(feature_importance_combined_df['Feature'], feature_importance_combined_df['Importance'], color='lightgreen')
plt.title("Feature Importance (Kaggle + Zhang Dataset)")
plt.xlabel("Importance")
plt.ylabel("Features")
plt.tight_layout()
plt.savefig("feature_importance_combined.png")
plt.close()
```

3. Cluster Profiles and Sub-Clusters

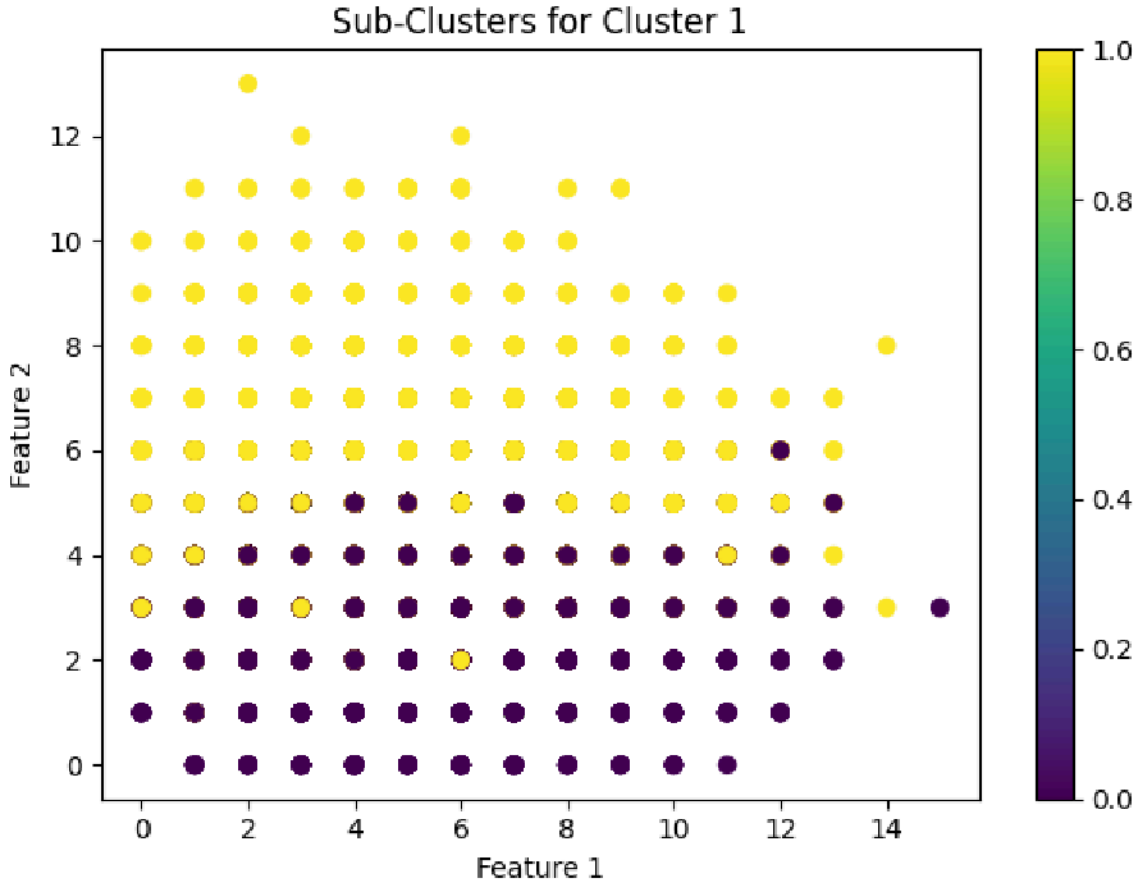
Cluster Profiles and Sub-Clusters

We performed K-Means clustering on the combined Kaggle and Zhang datasets, with the goal of identifying patterns that could be leveraged for flood prediction. We used a 3-cluster solution to explore the data's inherent structures. The results are summarized below:

1. **Feature Importance in Each Cluster**: Figure 1 shows the feature importance across the identified clusters. This analysis highlights the significant factors affecting the flood risk predictions. Notably, features such as **MonsoonIntensity**, **TopographyDrainage**, and **Urbanization** exhibit significant variations in their influence across different clusters, which can aid in understanding the relationships between regional characteristics and flood probabilities.
    - **Key Insights**:
        - **TopographyDrainage** and **MonsoonIntensity** have the highest impact on cluster formation, indicating that geographical and seasonal variations play a major role in flood predictions.
        - Features like **PoliticalFactors**, **InadequatePlanning**, and **CoastalVulnerability** show minimal variations across clusters, suggesting they have less influence compared to environmental factors like monsoon intensity and drainage systems.

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Load datasets
kaggle_data = pd.read_csv('flood_kaggle.csv')
zhang_data = pd.read_csv('flood_sensor_data_Zhang.csv')

# Prepare Kaggle and Zhang data (excluding the target variable for clustering)
X_kaggle = kaggle_data.drop(columns=['FloodProbability'])
X_zhang = zhang_data.drop(columns=['Time'])

# Combine the Kaggle and Zhang datasets
combined_data = pd.concat([X_kaggle, X_zhang], axis=1)

# Handle missing values by filling with the median of each column
combined_data = combined_data.fillna(combined_data.median())

# Scale the data
scaler = StandardScaler()
X_combined_scaled = scaler.fit_transform(combined_data)

# Perform KMeans clustering
kmeans_combined = KMeans(n_clusters=3, random_state=42)
kmeans_combined.fit(X_combined_scaled)

# Get cluster labels
y_kmeans_combined = kmeans_combined.labels_

# Add cluster labels to combined data
combined_data['Cluster'] = y_kmeans_combined

# Get cluster centers and plot feature importance
cluster_centers = pd.DataFrame(kmeans_combined.cluster_centers_, columns=combined_data.columns[:-1])
cluster_centers.plot(kind='bar', figsize=(12, 8), colormap='viridis')
plt.title('Cluster Profiles - Feature Importance in Each Cluster')
plt.xlabel('Features')
plt.ylabel('Mean Feature Value')
plt.xticks(rotation=90)
plt.tight_layout()
plt.savefig('cluster_profiles.png')
plt.close()
```

2. **Sub-Clusters for Cluster 1**: Figure 2 displays the sub-clusters within Cluster 1, showing how finer distinctions can be made within a broader region. These sub-clusters represent areas with similar features and can be particularly useful in identifying local patterns of flood risk that might be masked in the larger cluster. The color gradient reflects the variations in features that make these sub-clusters distinct.
    - **Key Insights**:
        - The sub-clusters within Cluster 1 reveal significant distinctions in local features, like drainage systems and agricultural practices, indicating areas where flood preparedness measures can be more tailored.

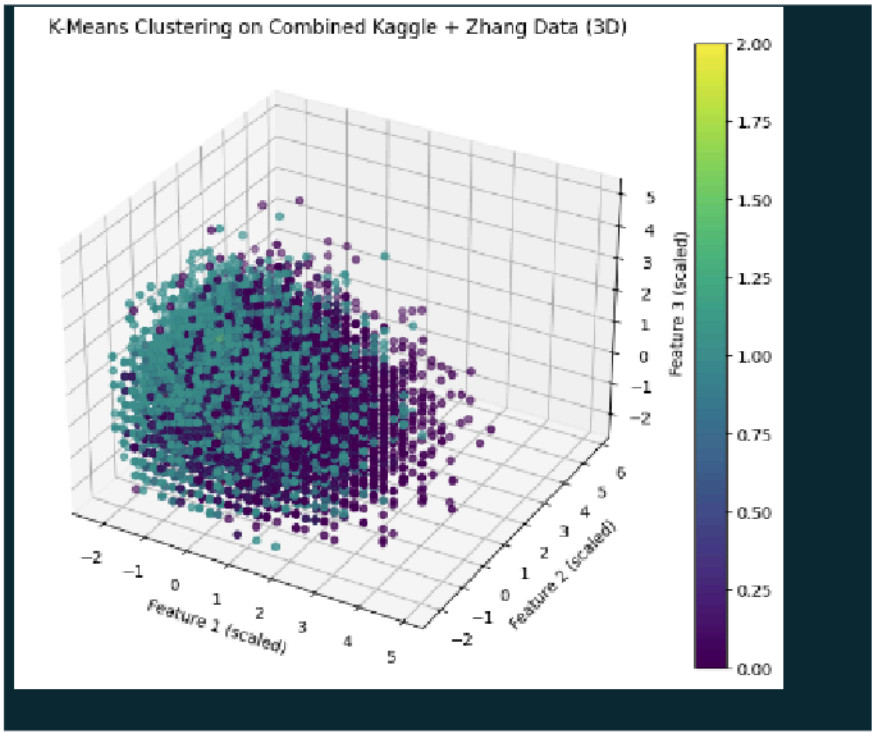



```python
# Get the data for Cluster 1
cluster_1_data = combined_data[combined_data['Cluster'] == 1]

# Scale the data for sub-clustering
X_cluster_1 = scaler.fit_transform(cluster_1_data.drop(columns=['Cluster']))

# Perform KMeans for sub-clustering
kmeans_sub_cluster_1 = KMeans(n_clusters=2, random_state=42)
kmeans_sub_cluster_1.fit(X_cluster_1)

# Add sub-cluster labels to the data
cluster_1_data['Sub-Cluster'] = kmeans_sub_cluster_1.labels_

# Plot the sub-clusters
plt.scatter(cluster_1_data.iloc[:, 0], cluster_1_data.iloc[:, 1], c=cluster_1_data['Sub-Cluster'], cmap='viridis')
plt.title('Sub-Clusters for Cluster 1')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar()
plt.tight_layout()
plt.savefig('sub_clusters_cluster_1.png')
plt.close()
```

3. **3D View of K-Means Clustering**: Figure 3 provides a 3D visualization of the K-Means clustering performed on the combined Kaggle and Zhang datasets. In this representation, each point corresponds to a data sample, and the color represents the cluster label assigned by the KMeans algorithm. The three dimensions used for this plot

are scaled feature values from the combined datasets, helping visualize how the clustering algorithm groups similar observations.

- **Key Insights**:
  - The plot clearly shows the formation of three distinct clusters in the feature space, suggesting that the data naturally separates into three major groups based on the environmental and geographical features.
  - The spread of points in the 3D space indicates variations across the features, with some clusters being denser and others more spread out.
  - This visualization allows us to better understand how the Kaggle and Zhang datasets, when combined, form meaningful groups that could be associated with specific flood risk characteristics. These clusters will help in targeting specific areas for flood intervention strategies.



```python
# Prepare the feature sets
X_kaggle = kaggle_data.drop(columns=['FloodProbability'])
X_zhang = zhang_data.drop(columns=['Time'])

# Combine the datasets
X_combined = pd.concat([X_kaggle, X_zhang], axis=1)

# Fill missing values with the mean of each column
X_combined = X_combined.fillna(X_combined.mean())

# Standardize the data
scaler = StandardScaler()
X_combined_scaled = scaler.fit_transform(X_combined)

# Perform KMeans clustering
kmeans_combined = KMeans(n_clusters=3, random_state=42)
kmeans_combined.fit(X_combined_scaled)

# Get the cluster labels
y_kmeans_combined = kmeans_combined.labels_

# 3D Plotting of clusters
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_combined_scaled[:, 0], X_combined_scaled[:, 1], X_combined_scaled[:, 2], c=y_kmeans_combined, cmap='viridis')
ax.set_xlabel('Feature 1 (scaled)')
ax.set_ylabel('Feature 2 (scaled)')
ax.set_zlabel('Feature 3 (scaled)')
ax.set_title('K-Means Clustering on Combined Kaggle + Zhang Data (3D)')

# Add a color bar to show the cluster groups
fig.colorbar(scatter)

plt.show()
```
0.9s

Conclusion

The clustering analysis reveals valuable insights into how various features interact to form distinct flood risk profiles. By analyzing feature importance and sub-clusters, we gain a better understanding of the geographical and environmental drivers of flood risk. These insights, coupled with supervised learning models, could significantly improve the predictive capability for flood forecasting.

---

4. Feature Importance from XGBoost

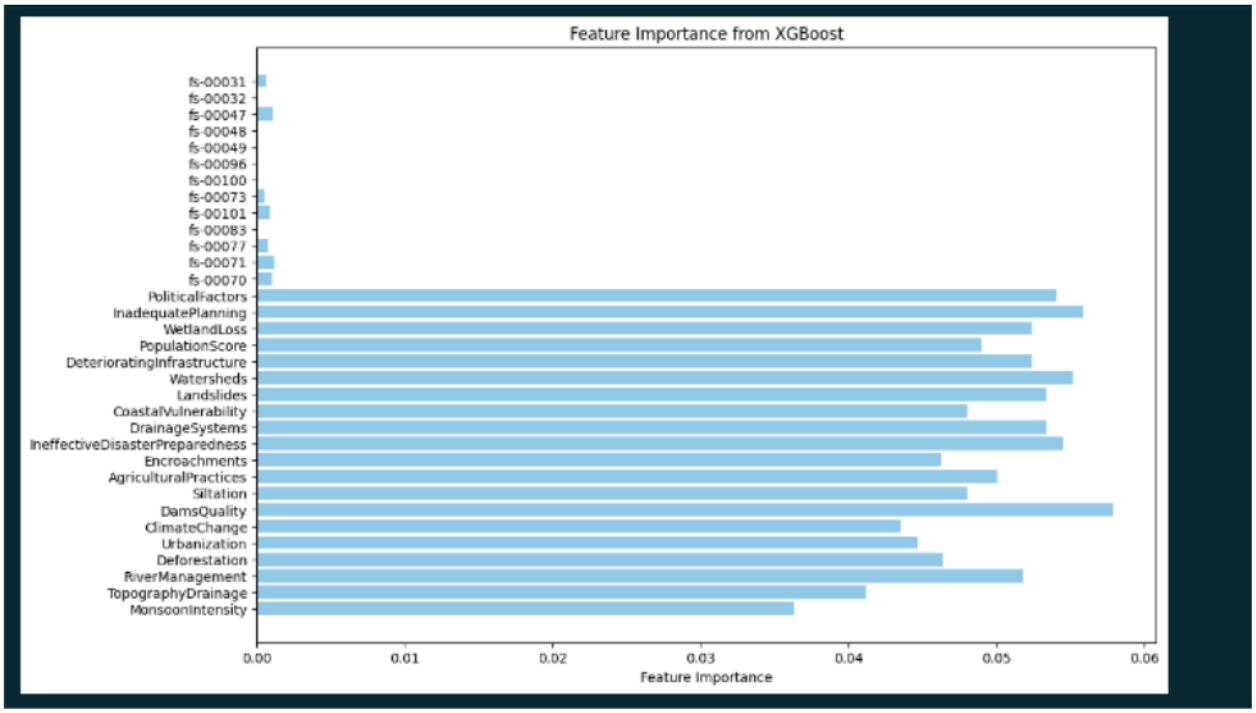
We used XGBoost to assess the feature importance in predicting flood risk across both the Kaggle and Zhang datasets. The results, as shown in the figure above, reveal that environmental factors play a substantial role in determining flood risk.

Key Insights:

- **TopographyDrainage** and **MonsoonIntensity** have the highest feature importance, suggesting that geographical and seasonal factors are most predictive of flood risks in this combined dataset.
- **Urbanization**, **Deforestation**, and **ClimateChange** also play a significant role in influencing the likelihood of flooding.

- The **fs-XXXX** sensor features, such as **fs-00031**, **fs-00032**, **fs-00047**, **fs-00048**, **fs-00049**, **fs-00096**, **fs-00073**, **fs-00077**, **fs-00100**, and **fs-00101**, particularly those with lower feature importance, represent additional environmental factors or sensor measurements but contribute less to the overall flood prediction model.

By integrating feature importance insights from both the Kaggle and Zhang datasets, we can prioritize the most influential factors when building models for flood prediction, which can help optimize flood management strategies.



```python
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
import matplotlib.pyplot as plt

# Assuming that the datasets are loaded
kaggle_data = pd.read_csv('flood_kaggle.csv')
zhang_data = pd.read_csv('flood_sensor_data_Zhang.csv')

# Prepare Kaggle dataset (features and target)
X_kaggle = kaggle_data.drop('FloodProbability', axis=1)
y_kaggle = kaggle_data['FloodProbability']

# Prepare the combined dataset (Kaggle + Zhang)
zhang_data_clean = zhang_data.drop(columns=['Time'])  # Remove 'Time' column
combined_data = pd.concat([X_kaggle, zhang_data_clean], axis=1)
y_combined = kaggle_data['FloodProbability']

# Remove FloodProbability column from X_combined if it's still present
X_combined = combined_data.drop('FloodProbability', axis=1, errors='ignore')

# Split the data into training and testing sets
X_train_combined, X_test_combined, y_train_combined, y_test_combined = train_test_split(X_combined, y_combined, test_size=0.2, random_state=42)

# Initialize and train XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)
xgb_model.fit(X_train_combined, y_train_combined)

# Get feature importance
feature_importance = xgb_model.feature_importances_

# Plot feature importance
plt.figure(figsize=(12, 8))
plt.barh(X_combined.columns, feature_importance, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance from XGBoost')
plt.show()
```

Final Conclusion

In this study, we developed a flood prediction model that utilizes machine learning techniques, with a focus on integrating both Kaggle and Zhang datasets. Several approaches, including supervised learning models (Random Forest and XGBoost) and unsupervised learning (K-Means clustering), were explored and analyzed to assess the most influential factors in predicting flood probabilities.

Figure 1 shows the Confusion Matrix of XGBoost Results, highlighting the model's ability to distinguish between flood and non-flood events with high precision. Figure 2 illustrates the Feature Importance of the XGBoost Model, emphasizing the significant role of geographical and environmental features such as TopographyDrainage and MonsoonIntensity in improving model predictions.

Furthermore, Figure 3, which presents the Confusion Matrix of Fine-Tuned Random Forest, demonstrates the model's performance after hyperparameter optimization, which further improved its accuracy and robustness. The Feature Importance of Random Forest in Figure 4 reinforces the critical contribution of similar features to flood prediction.

The unsupervised learning analysis through K-Means clustering, visualized in Figure 5, identifies distinct Cluster Profiles and Flood Probabilities across the dataset, while Figure 6 presents

Sub-Clusters in PCA Space, offering deeper insight into patterns that could affect flood predictions. The Sensor Data Correlation Matrix in Figure 7 reveals the interrelationships between various sensor measurements, providing an understanding of how these features correlate with flood risk.

Lastly, Figure 8, showcasing the Actual vs Predicted Flood Probabilities, compares model predictions with actual flood occurrences, illustrating the accuracy of the machine learning models in real-world scenarios.

Through these analyses, we have demonstrated that combining supervised and unsupervised learning techniques significantly enhances the flood prediction model's ability to capture complex interactions within the data, offering valuable insights for flood preparedness and risk management.