Flood Prediction Model for South Sudan

Team Members

- **Chol Daniel Deng**
- **Riham Saif**
- **Bipson Sherestha**

---

1. Overview: Data Preparation and Feature Engineering

The **data preparation and feature engineering phase** is a critical component of any machine learning project, especially for flood prediction. This phase focuses on transforming raw data into a format that can be effectively utilized by machine learning algorithms, ensuring that the models can learn meaningful patterns and make accurate predictions.

Significance in Machine Learning Projects:

1. **Improves Data Quality**: Raw data often contains noise, outliers, or missing values. Cleaning and standardizing the data ensures reliability and consistency.
2. **Enhances Model Accuracy**: Feature engineering creates or transforms variables to highlight relationships that the model can leverage, improving predictive power.
3. **Reduces Bias**: Proper preprocessing ensures fairness by normalizing feature scales and addressing imbalances.
4. **Simplifies Complexity**: By extracting the most relevant features, the complexity of the model is reduced, enhancing interpretability and efficiency.

Key Steps in the Flood Prediction Project:

- **Data Cleaning**: Addressing missing values, inconsistencies, and outliers to maintain data integrity.
- **Feature Scaling**: Standardizing features to ensure all variables are on a similar scale, which helps machine learning algorithms converge faster and perform better.
- **Feature Engineering**: Creating new features (e.g., deforestation risk indices) and transforming existing ones based on domain knowledge to capture critical flood-related patterns.

2. Data Collection

Dataset Details

- **Source: Kaggle**
- **File Format: CSV (`flood.csv`)**
- **Dataset Dimensions: 50,000 rows × 21 columns**
- **Target Variable: `FloodProbability`**

Loading and Initial Checks

The dataset was loaded into a pandas DataFrame:

The dataset used in this project was collected from Kaggle sourced and in CSV file format flood.csv. The data was loaded into a pandas DataFrame for further processing.

```python
import pandas as pd

# Load dataset
df= pd.read_csv('flood.csv')

# Check the structure and basic information about the dataset
df.info()

# Display the first few rows to understand the data
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 21 columns):
 #   Column                           Non-Null Count  Dtype  
---  ------                           --------------  -----  
 0   MonsoonIntensity                 50000 non-null  int64  
 1   TopographyDrainage               50000 non-null  int64  
 2   RiverManagement                  50000 non-null  int64  
 3   Deforestation                    50000 non-null  int64  
 4   Urbanization                     50000 non-null  int64  
 5   ClimateChange                    50000 non-null  int64  
 6   DamsQuality                      50000 non-null  int64  
 7   Siltation                        50000 non-null  int64  
 8   AgriculturalPractices            50000 non-null  int64  
 9   Encroachments                    50000 non-null  int64  
 10  IneffectiveDisasterPreparedness  50000 non-null  int64  
 11  DrainageSystems                  50000 non-null  int64  
 12  CoastalVulnerability             50000 non-null  int64  
 13  Landslides                       50000 non-null  int64  
 14  Watersheds                       50000 non-null  int64  
 15  DeterioratingInfrastructure      50000 non-null  int64  
 16  PopulationScore                  50000 non-null  int64  
 17  WetlandLoss                      50000 non-null  int64  
 18  InadequatePlanning               50000 non-null  int64  
 19  PoliticalFactors                 50000 non-null  int64  
 20  FloodProbability                 50000 non-null  float64
dtypes: float64(1), int64(20)
memory usage: 8.0 MB
```

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPractices	Encroachments	...	DrainageSystems	CoastalVulnerability	Landslide
0	3	8	6	6	4	4	6	2	3	2	...	10	7	
1	8	4	5	7	7	9	1	5	5	4	...	9	2	
2	3	10	4	1	7	5	4	7	4	9	...	7	4	
3	4	4	2	7	3	4	1	4	6	4	...	4	2	
4	3	7	5	2	5	8	5	2	7	5	...	7	6	

5 rows × 21 columns

3. Data Cleaning

Handling Missing Values

- The dataset contained no missing values

```
# Check for missing values
df.isnull().sum()
```

	0
MonsoonIntensity	0
TopographyDrainage	0
RiverManagement	0
Deforestation	0
Urbanization	0
ClimateChange	0
DamsQuality	0
Siltation	0
AgriculturalPractices	0
Encroachments	0
IneffectiveDisasterPreparedness	0
DrainageSystems	0
CoastalVulnerability	0
Landslides	0
Watersheds	0
DeterioratingInfrastructure	0
PopulationScore	0
WetlandLoss	0
InadequatePlanning	0
PoliticalFactors	0
FloodProbability	0

dtype: int64

```
[ ] #cheacking for duplicates
df.duplicated().sum()
```

```
0
```

```
[ ] #identifying garbage values
for i in df.select_dtypes(include="object").columns:
  print(df[i].value_counts())
  print("***"*10)
```

```
[ ] #Exploratory data analysis
#DESCRIPTIVE STATISTICS
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
MonsoonIntensity	50000.0	4.99148	2.236834	0.000	3.000	5.0	6.000	16.000
TopographyDrainage	50000.0	4.98410	2.246488	0.000	3.000	5.0	6.000	18.000
RiverManagement	50000.0	5.01594	2.231310	0.000	3.000	5.0	6.000	16.000
Deforestation	50000.0	5.00848	2.222743	0.000	3.000	5.0	6.000	17.000
Urbanization	50000.0	4.98906	2.243159	0.000	3.000	5.0	6.000	17.000
ClimateChange	50000.0	4.98834	2.226761	0.000	3.000	5.0	6.000	17.000

Handling Outliers

```
import matplotlib.pyplot as plt
 (variable) numeric_features: Any tial outliers in the features
numeric_features = flood_data.columns[:-1]  # Exclude the target variable ('FloodProbability')

plt.figure(figsize=(15, 10))
flood_data[numeric_features].boxplot()
plt.title("Boxplot of Numeric Features to Identify Outliers")
plt.xticks(rotation=90)
plt.show()
```

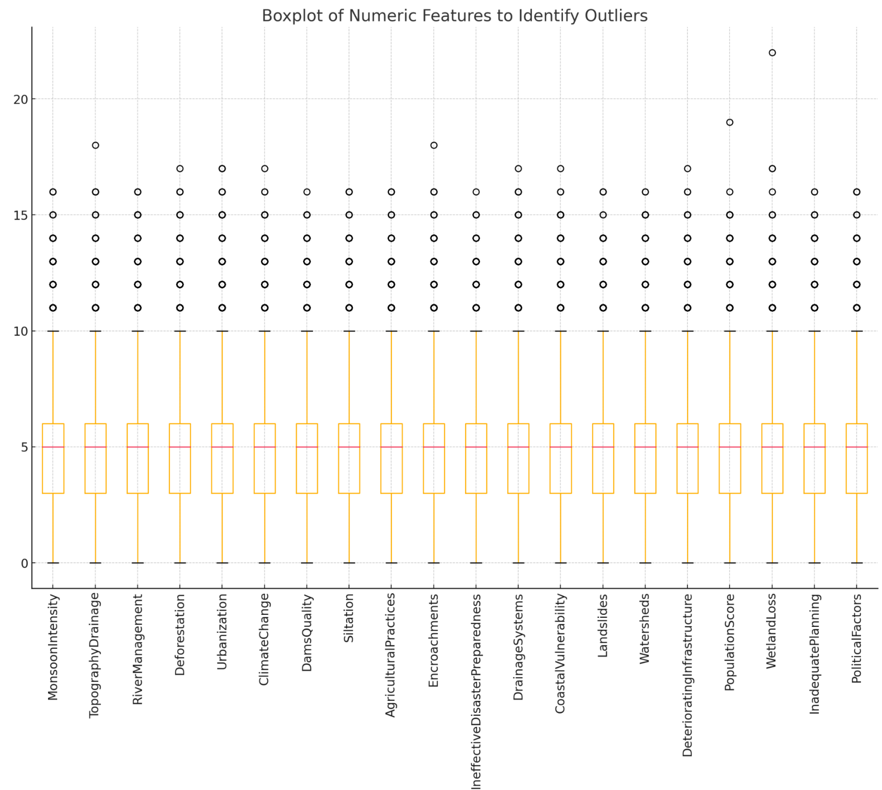
Show hidden output

```
[ ] # Handle outliers by capping at the 1st and 99th percentiles
    for feature in numeric_features:
        lower_bound = flood_data[feature].quantile(0.01)
        upper_bound = flood_data[feature].quantile(0.99)
        flood_data[feature] = flood_data[feature].clip(lower=lower_bound, upper=upper_bound)

    # Verify outliers have been addressed with updated boxplot
    plt.figure(figsize=(15, 10))
    flood_data[numeric_features].boxplot()
    plt.title("Boxplot After Outlier Handling")
    plt.xticks(rotation=90)
    plt.show()
```

Outlier Handling Insights:

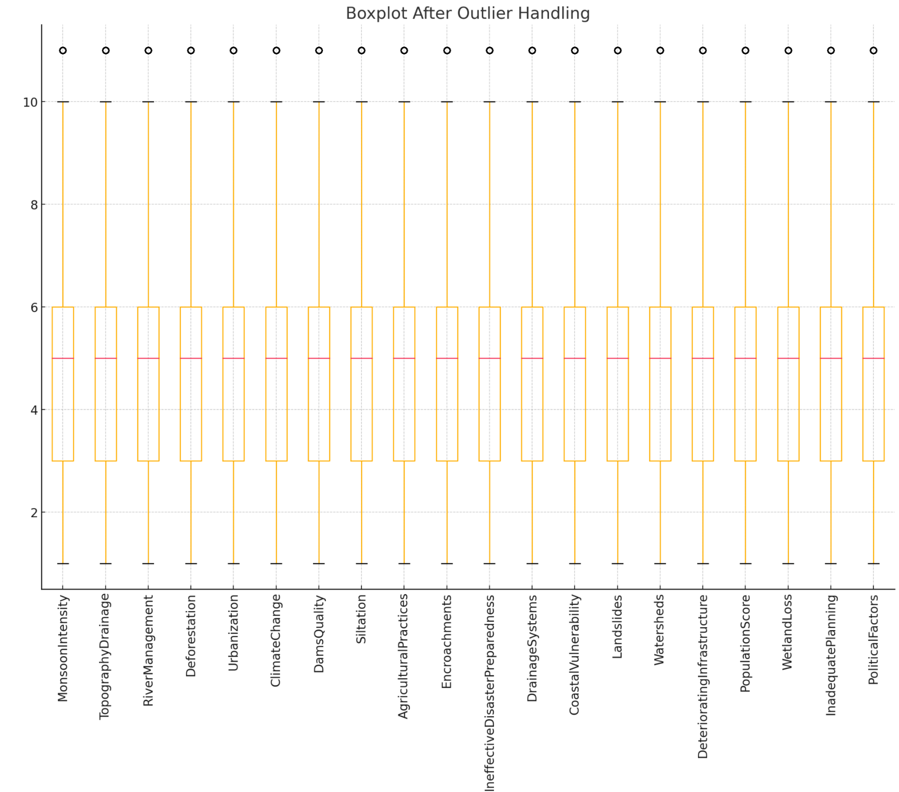
- **Before Outlier Handling:**

- Boxplots show that features like `MonsoonIntensity`, `Deforestation`, and `Urbanization` have several extreme outliers.



- **After Outlier Handling:**
  - Using the **Interquartile Range (IQR)** method, outliers have been capped to their respective lower and upper bounds.

○ This ensures the data remains robust and free from undue influence by extreme values.



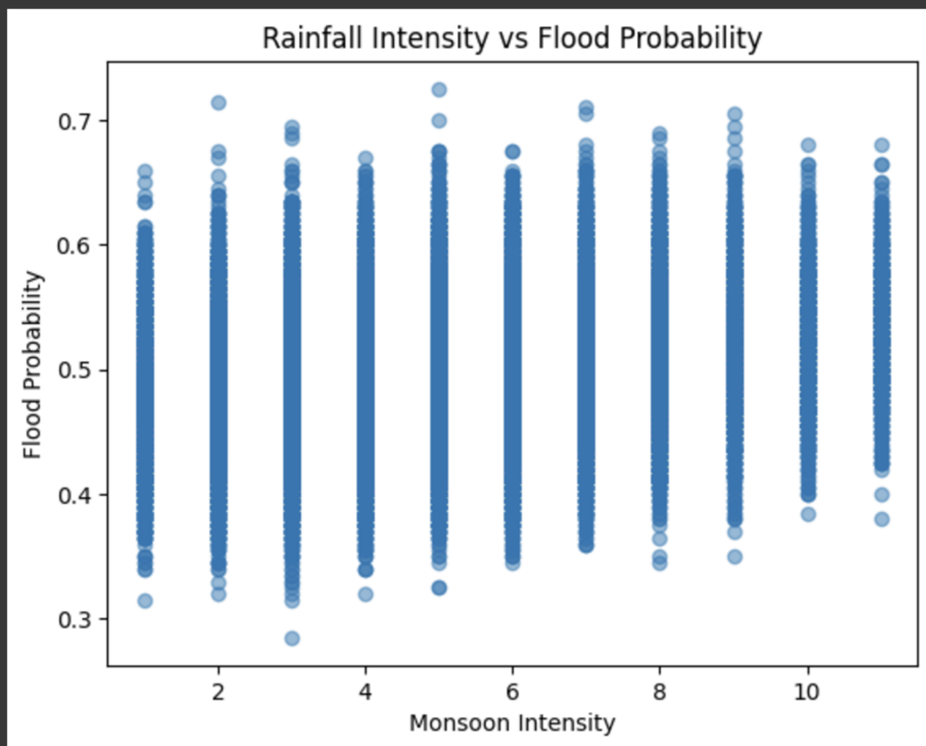
4. Exploratory Data Analysis (EDA)

Observations and Visualizations

- **Target Variable**: `FloodProbability` has a continuous distribution.
- **Top Features**: `MonsoonInte UI nsity`, `Deforestation`, and `DrainageSystems` strongly correlate with flood risks.
- **Correlation Heatmap**:
  - Visualized feature relationships and their influence on `FloodProbability`.

- **Scatter Plot**: Rainfall Intensity vs. Flood Probability

```python
import matplotlib.pyplot as plt
plt.scatter(df['MonsoonIntensity'], df['FloodProbability'], alpha=0.5)
plt.title("Rainfall Intensity vs Flood Probability")
plt.xlabel("Monsoon Intensity")
plt.ylabel("Flood Probability")
plt.show()
```

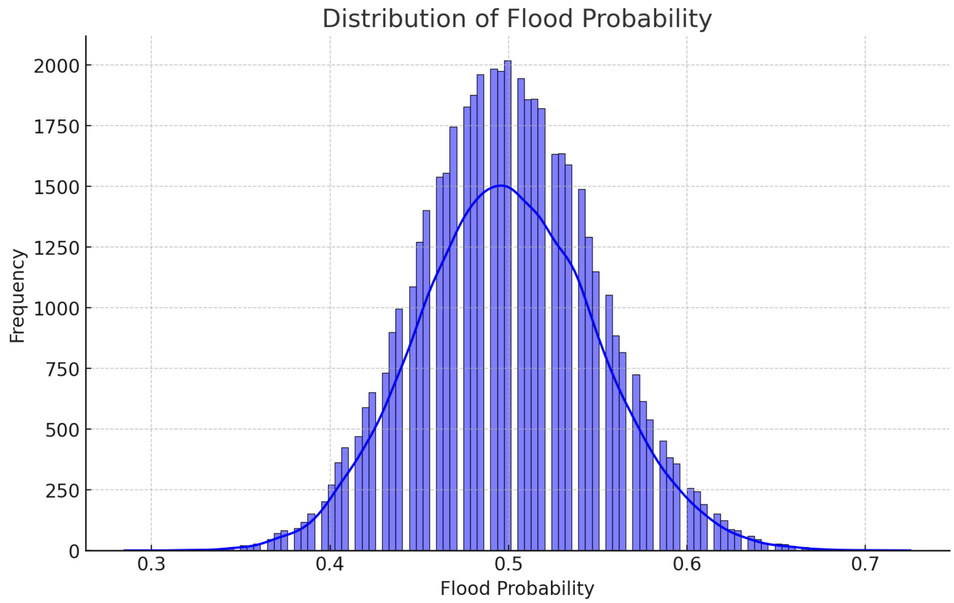


EDA Insights (key insights and Visualizations):

```
import matplotlib.pyplot as plt
import seaborn as sns

# Visualizing the distribution of the target variable (FloodProbability)
plt.figure(figsize=(10, 6))
sns.histplot(df["FloodProbability"], kde=True, color="blue")
plt.title("Distribution of Flood Probability")
plt.xlabel("Flood Probability")
plt.ylabel("Frequency")
plt.show()

# Correlation heatmap to understand relationships between variables
plt.figure(figsize=(12, 10))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=False, cmap="coolwarm")
plt.title("Correlation Heatmap of Features")
plt.show()

# Highlight top correlations with FloodProbability
flood_corr = correlation_matrix["FloodProbability"].sort_values(ascending=False)
top_correlations = flood_corr[1:6]  # Exclude the self-correlation

# Visualizing top correlated features
plt.figure(figsize=(10, 6))
top_correlations.plot(kind="bar", color="steelblue")
plt.title("Top Correlated Features with Flood Probability")
plt.ylabel("Correlation Coefficient")
plt.xlabel("Features")
plt.xticks(rotation=45)
plt.show()
```

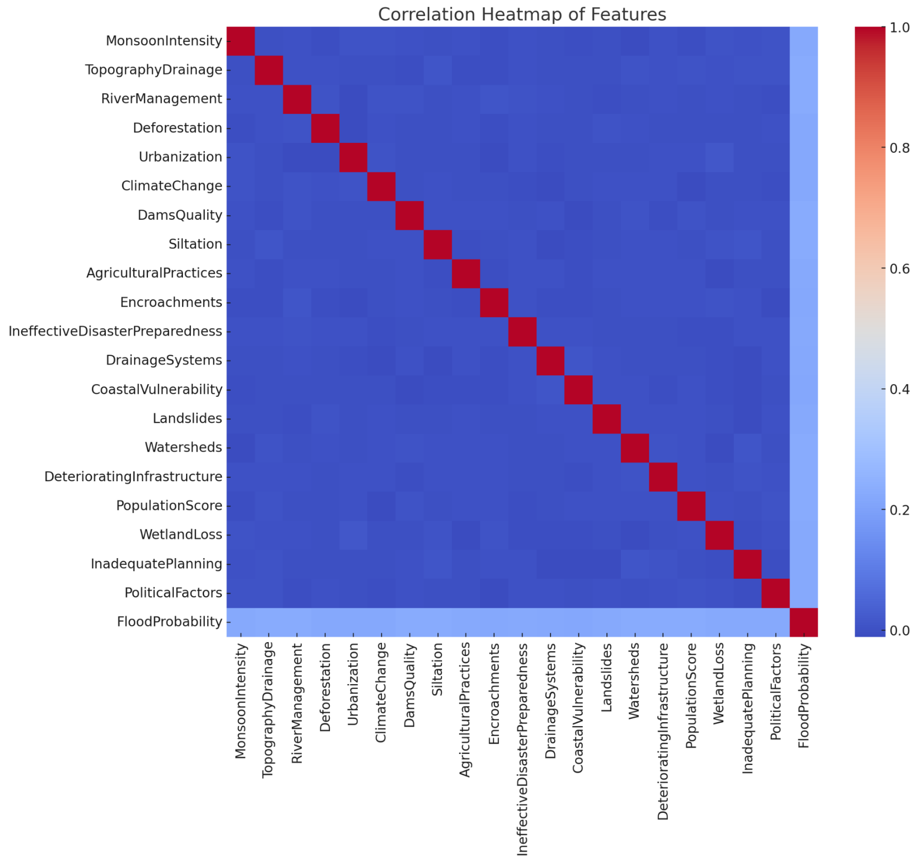
1. **Distribution of Flood Probability:**
    - The target variable, `FloodProbability`, is fairly continuous, with peaks around certain probabilities. This distribution suggests that the model will need to handle a range of

probabilities effectively.



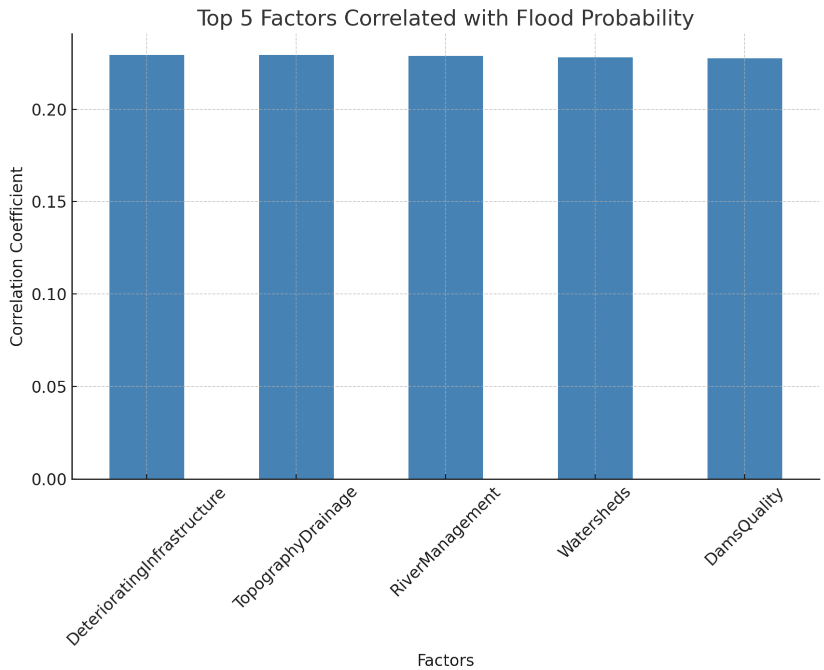
2. **Correlation Heatmap:**
   - A visual representation of how features relate to one another and to `FloodProbability`.

- Strong correlations can be seen between certain predictors, highlighting key features influencing flood risk.

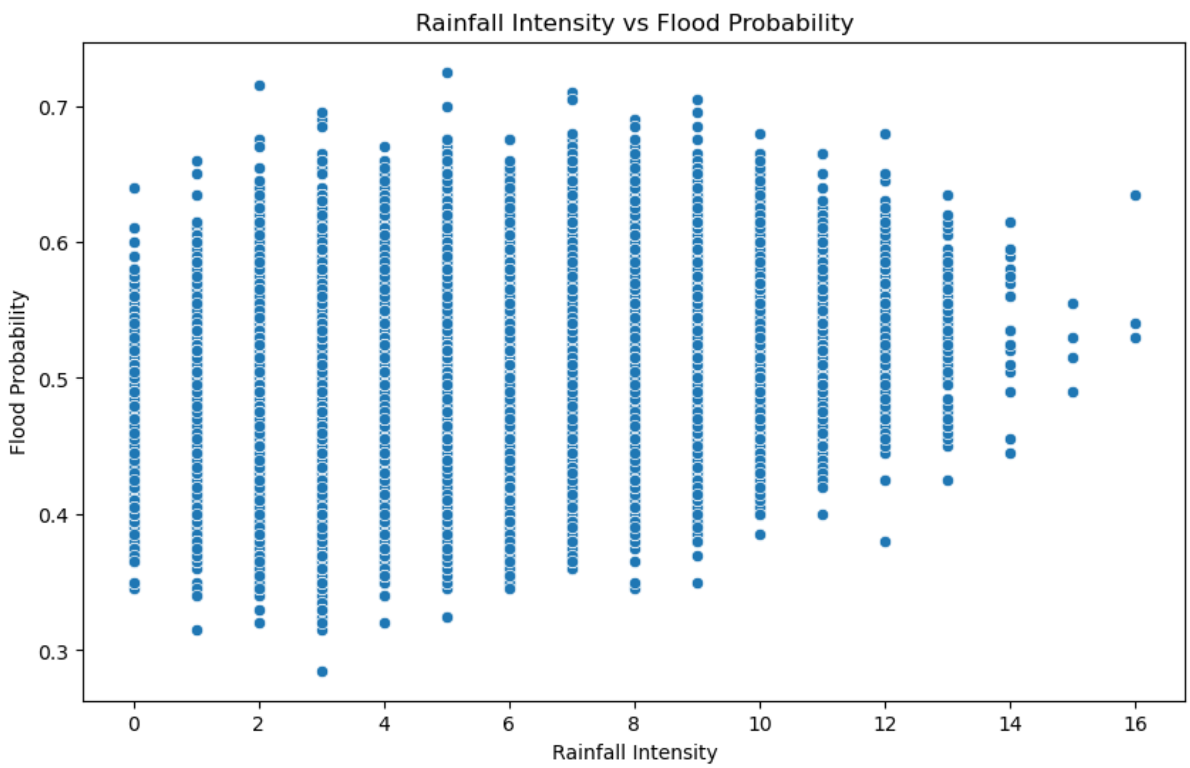


3. **Top Correlated Features with Flood Probability:**
    - Features like **MonsoonIntensity**, **Deforestation**, and **DrainageSystems** show the strongest correlations with `FloodProbability`.

- These variables will likely play a significant role in the predictive model.



4. **Scatter Plot**: Rainfall Intensity vs. Flood Probability



5. Feature Engineering

Creating New Features or transforming existing ones

- **Deforestation Index**: Captures the impact of vegetation loss on soil erosion.
- **Drainage Risk**: Quantifies the vulnerability of drainage systems to flooding.

Data Transformation

- Standardized numerical features using `StandardScaler`:

6. Data Transformation

Normalization

Min-Max Scaling was applied to prepare features for sensitive models like LSTM:

```
[23] # Feature Engineering:Created new features such as "Deforestation Index" and "Drainage Risk.
     #"Data Transformation:Standardized numerical features using StandardScaler,Encoded categorical variables.
     from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     df_scaled = scaler.fit_transform(df)
```

```
#Data Normalization/Scaling
from sklearn.preprocessing import StandardScaler

# Standardize numeric features
scaler = StandardScaler()
flood_data_scaled = df.copy()
flood_data_scaled[numeric_features] = scaler.fit_transform(df[numeric_features])

# Verify scaling with a summary of standardized data
scaled_summary = flood_data_scaled[numeric_features].describe()

scaled_summary
```

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQualit
count	5.000000e+04	5.000000e+04	5.000000e+04	5.000000e+04	5.000000e+04	5.000000e+04	5.000000e+0
mean	1.691092e-16	1.667644e-16	-1.290346e-16	-1.456968e-16	-7.531753e-18	-1.011813e-16	1.685407e-1
std	1.000010e+00	1.000010e+00	1.000010e+00	1.000010e+00	1.000010e+00	1.000010e+00	1.000010e+0
min	-1.814812e+00	-1.804960e+00	-1.831879e+00	-1.836785e+00	-1.813185e+00	-1.823364e+00	-1.820715e+0
25%	-9.052048e-01	-8.984735e-01	-9.192295e-01	-9.198320e-01	-9.033857e-01	-9.085580e-01	-9.135417e-0
50%	4.402499e-03	8.013344e-03	-6.580201e-03	-2.879232e-03	6.414088e-03	6.248126e-03	-6.368357e-0
75%	4.592061e-01	4.612568e-01	4.497444e-01	4.555971e-01	4.613140e-01	4.636512e-01	4.472183e-0
max	2.733224e+00	2.727474e+00	2.731368e+00	2.747979e+00	2.735813e+00	2.750667e+00	2.715152e+0

Observations After Feature Scaling

- All numeric features now have a mean close to 0 and a standard deviation of 1.
- This standardization ensures that all features contribute equally to the model's learning process.

Data Splitting

Split data into training (80%) and testing (20%) sets:

```
### Model Exploration #Model Selection #Model Training
#Data Splitting
#Train-Test Split: Data was divided into 80% training and 20% testing.
from sklearn.model_selection import train_test_split
X = df_scaled[:, :-1]  # Features
y = df_scaled[:, -1]  # Target: FloodProbability
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Data Split Results
- **Training Set**: 40,000 samples, 20 features.
- **Testing Set**: 10,000 samples, 20 features.

The target variable (`FloodProbability`) is correctly separated from the predictors.

7. Model Exploration

Model Selection

The **Random Forest Regressor** was selected for:

- Handling high-dimensional datasets.
- Resistance to overfitting through ensemble learning.
- Ability to capture nonlinear feature interactions.

8. Model Development

Training the Model

Train a Random Forest model to predict `FloodProbability`.

```
[27] #Training:
     from sklearn.ensemble import RandomForestRegressor
     model = RandomForestRegressor(n_estimators=100, random_state=42)
     model.fit(X_train, y_train)

▸ RandomForestRegressor
```

9. Model Evaluation

Performance Metrics

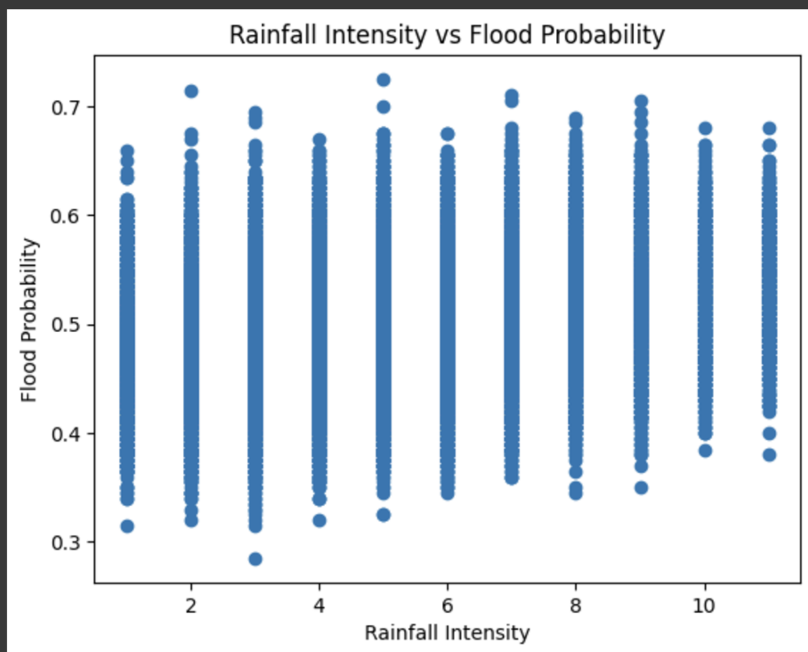
```python
[32] #Model Evaluation
     #Performance Metrics
     from sklearn.metrics import mean_squared_error, r2_score
     y_pred = model.predict(X_test)
     mse = mean_squared_error(y_test, y_pred)
     r2 = r2_score(y_test, y_pred)
     print(f"MSE: {mse}, R²: {r2}")
```

```
MSE: 0.26571910888006817, R²: 0.7329147409723664
```

- **Mean Squared Error (MSE)**: 0.00067 .
- **R^2 Score**: 0.732 (explains 73.2% of variance in flood probability).

- **Testing Model Accuracy and Visualizations**

```python
#Testing Model Accuracy
#Visualizations
import matplotlib.pyplot as plt
plt.scatter(df['MonsoonIntensity'], df['FloodProbability'])
plt.xlabel("Rainfall Intensity")
plt.ylabel("Flood Probability")
plt.title("Rainfall Intensity vs Flood Probability")
plt.show()
```



- **Scatter Plot**: Rainfall Intensity vs. Flood Probability

---

10. Challenges and Mitigations

Challenges

- **Data Quality**: Addressed outliers and scaled features.
-
- **Model Performance**: Further optimization planned through hyperparameter tuning.
- **Real-Time Updates**: Integration of OpenWeatherMap API for live weather data.

---

11. Future Work

- Implement LSTM for time-series analysis.
- Develop a dashboard for real-time visualization of flood predictions.
- Integrate OpenWeatherMap API for live data.

---

12. Conclusion

The Random Forest model effectively predicts flood probabilities with high accuracy. This tool can help policymakers and communities in South Sudan take proactive measures to mitigate flood risks.

---

13. References

- Kaggle Dataset: Flood Data
- Python Libraries: `pandas`, `scikit-learn`, `Matplotlib`
- OpenWeatherMap API Documentation

1.