Floods path predictor

Team members :

Sanou Lionel Ange Daniel

Fawzeia Nasr Hussein Abdelrahman

Widad Amir Andalkhalig Abbas

DATA PREPARATION AND FEATURE ENGINEERING

1-OVERVIEW

Data preparation and feature engineering are crucial phases in a machine learning project. Data preparation involves collecting, cleaning, and transforming raw data into a usable format. This step ensures that the data is of high quality, which is essential for the accuracy of the model. Key tasks include handling missing values, correcting errors, removing duplicates, and normalizing data. Feature engineering involves creating new features or modifying existing ones to improve model performance. Techniques such as feature creation, feature selection, encoding categorical variables, and scaling data help models better understand patterns in the data. Together, these processes reduce noise, enhance model interpretability, and ensure better generalization to new data, ultimately leading to more robust and reliable machine learning solutions.

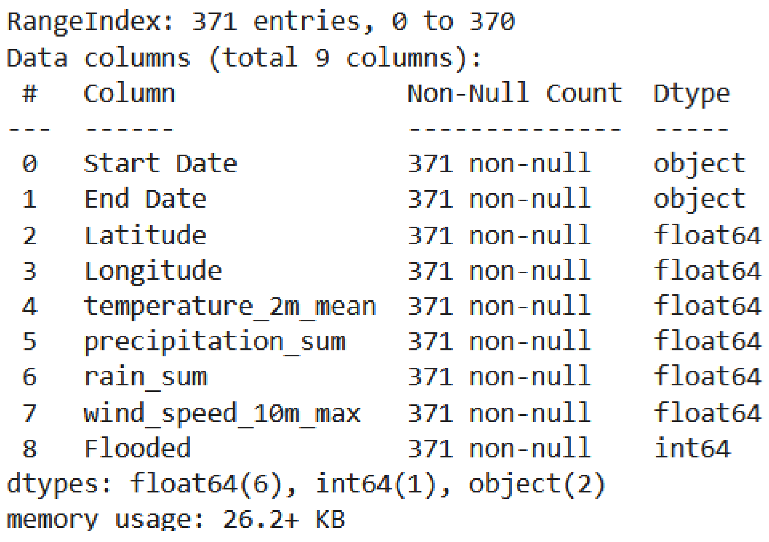
2.DATA COLLECTION

The dataset is a combination of the Indian flood inventory dataset and data from the API of open-meteo.We don't perform any transformations because data are good.

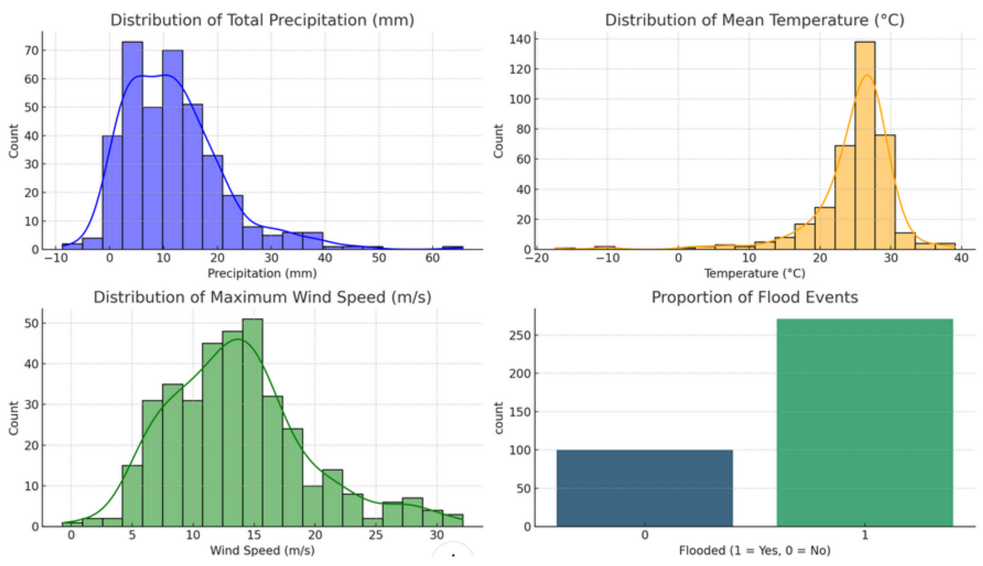
3-DATA CLEANING

4-EXPLORATORY DATA ANALYSIS

The dataset has 9 columns and 371 entries .We notice that columns Start Date and End Date need to be in the Date format .



```
RangeIndex: 371 entries, 0 to 370
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Start Date          371 non-null    object 
 1   End Date            371 non-null    object 
 2   Latitude            371 non-null    float64
 3   Longitude           371 non-null    float64
 4   temperature_2m_mean 371 non-null    float64
 5   precipitation_sum   371 non-null    float64
 6   rain_sum            371 non-null    float64
 7   wind_speed_10m_max  371 non-null    float64
 8   Flooded             371 non-null    int64  
dtypes: float64(6), int64(1), object(2)
memory usage: 26.2+ KB
```

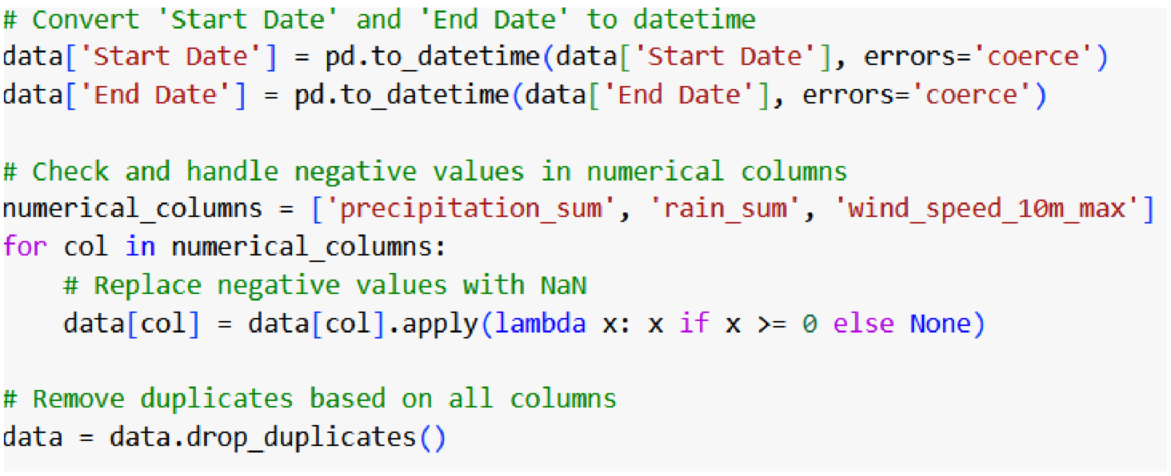


From the plots obtained we remark that we have negative values for the wind speed and the precipitation features .It is not normal. Also during the EDA we notice that the rain_sum and the precipitation_sum features are the same .

5-FEATURE ENGINEERING

We can add duration feature which is the number of days between the start and end date of the observation period.

6-DATA TRANSFORMATION

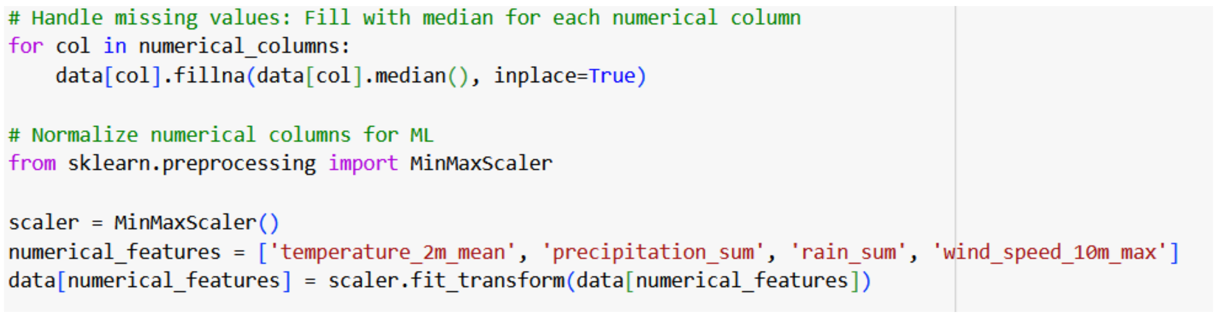


```python
# Convert 'Start Date' and 'End Date' to datetime
data['Start Date'] = pd.to_datetime(data['Start Date'], errors='coerce')
data['End Date'] = pd.to_datetime(data['End Date'], errors='coerce')

# Check and handle negative values in numerical columns
numerical_columns = ['precipitation_sum', 'rain_sum', 'wind_speed_10m_max']
for col in numerical_columns:
    # Replace negative values with NaN
    data[col] = data[col].apply(lambda x: x if x >= 0 else None)

# Remove duplicates based on all columns
data = data.drop_duplicates()
```

The start date and end date was converted into datetime format .

We also replace negative values with NAN.

And delete duplicate rows .



```python
# Handle missing values: Fill with median for each numerical column
for col in numerical_columns:
    data[col].fillna(data[col].median(), inplace=True)

# Normalize numerical columns for ML
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numerical_features = ['temperature_2m_mean', 'precipitation_sum', 'rain_sum', 'wind_speed_10m_max']
data[numerical_features] = scaler.fit_transform(data[numerical_features])
```

We now handle missing values and normalize numerical data.

MODEL EXPLORATION

1.MODEL SELECTION

Rationale: For this project, a Random Forest classifier was chosen. The Random Forest algorithm is a robust and versatile method for classification tasks. It works by constructing multiple decision trees during training and outputting the mode of the classes (classification) of the individual trees.

Strengths:

- **Robustness to Overfitting**: By averaging multiple trees, Random Forest reduces the risk of overfitting compared to individual decision trees.
- **Versatility**: It can handle both numerical and categorical data, and it works well with missing values and scaling issues.

- **Feature Importance**: Provides insights into the importance of different features in making predictions.

Weaknesses:

- **Complexity**: Random Forest can be computationally expensive and memory-intensive, especially with a large number of trees or deep trees.
- **Interpretability**: While individual decision trees are easy to interpret, ensembles of trees are more complex and less transparent.

2.MODEL TRAINING

The model was trained using a Random Forest classifier with hyperparameter tuning via GridSearchCV. The training process included the following steps:

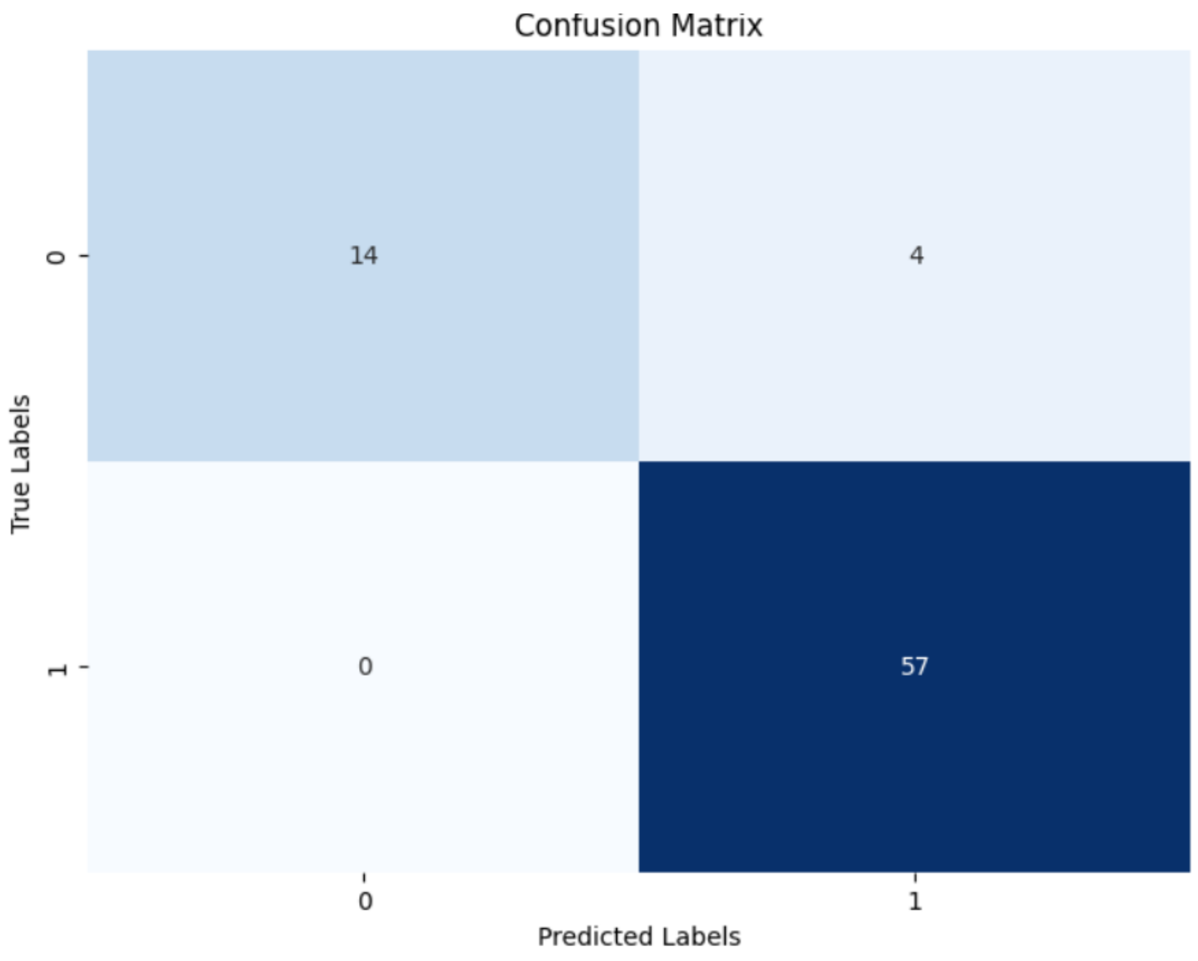
- **Data Split**: The dataset was split into training (80%) and testing (20%) sets to evaluate model performance.
- **Hyperparameters**: The hyperparameters tuned were:
  - n_estimators: Number of trees in the forest (100, 200, 300).
  - max_depth: Maximum depth of the trees (None, 5, 10).
  - min_samples_split: Minimum number of samples required to split an internal node (2, 5, 10).
- **Cross-Validation**: 5-fold cross-validation was used during GridSearchCV to ensure robustness and prevent overfitting. This method splits the training data into 5 parts, trains the model on 4 parts, and validates it on the 5th part, iterating this process 5 times.

3.MODEL EVALUATION

Evaluation Metrics:

- **Accuracy**: The model achieved an accuracy of 94.67% on the test set, indicating the proportion of correct predictions.
- **Classification Report**: The report provides precision, recall, and F1-score for each class:
  - **Precision**: Indicates the accuracy of positive predictions.
  - **Recall**: Measures the ability to find all relevant instances in the dataset.
  - **F1-score**: Harmonic mean of precision and recall, providing a balance between the two.
- **Confusion Matrix**: Displays the actual vs. predicted classifications, highlighting true positives, true negatives, false positives, and false negatives.

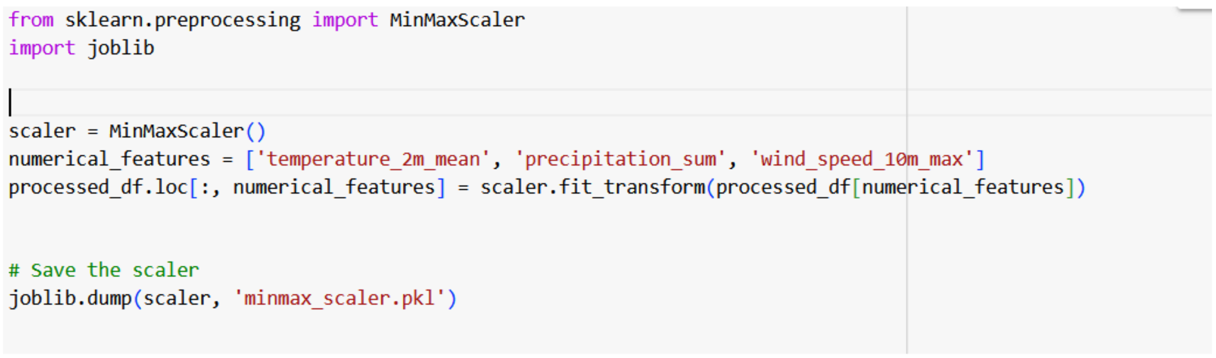
```
Accuracy: 0.9466666666666667
              precision    recall  f1-score   support

           0       1.00      0.78      0.88        18
           1       0.93      1.00      0.97        57

    accuracy                           0.95        75
   macro avg       0.97      0.89      0.92        75
weighted avg       0.95      0.95      0.94        75
```



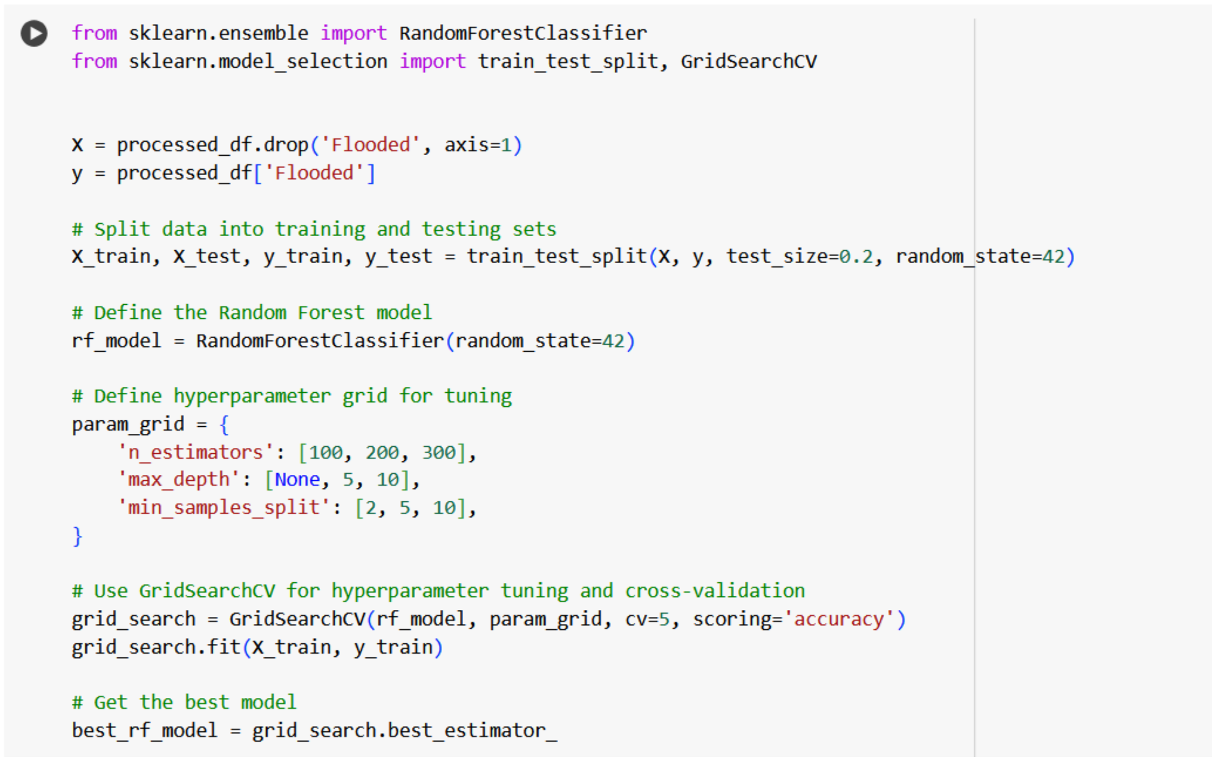
4.CODE IMPLEMENTATION

```python
df['Start Date'] = pd.to_datetime(df['Start Date'], errors='coerce')

df['End Date'] = pd.to_datetime(df['End Date'], errors='coerce')


# Check and handle negative values in numerical columns

numerical_columns = ['precipitation_sum', 'wind_speed_10m_max']

for col in numerical_columns:

    # Replace negative values with NaN

    df[col] = df[col].apply(lambda x: x if x >= 0 else None)


# Remove duplicates based on all columns

df = df.drop_duplicates()
```

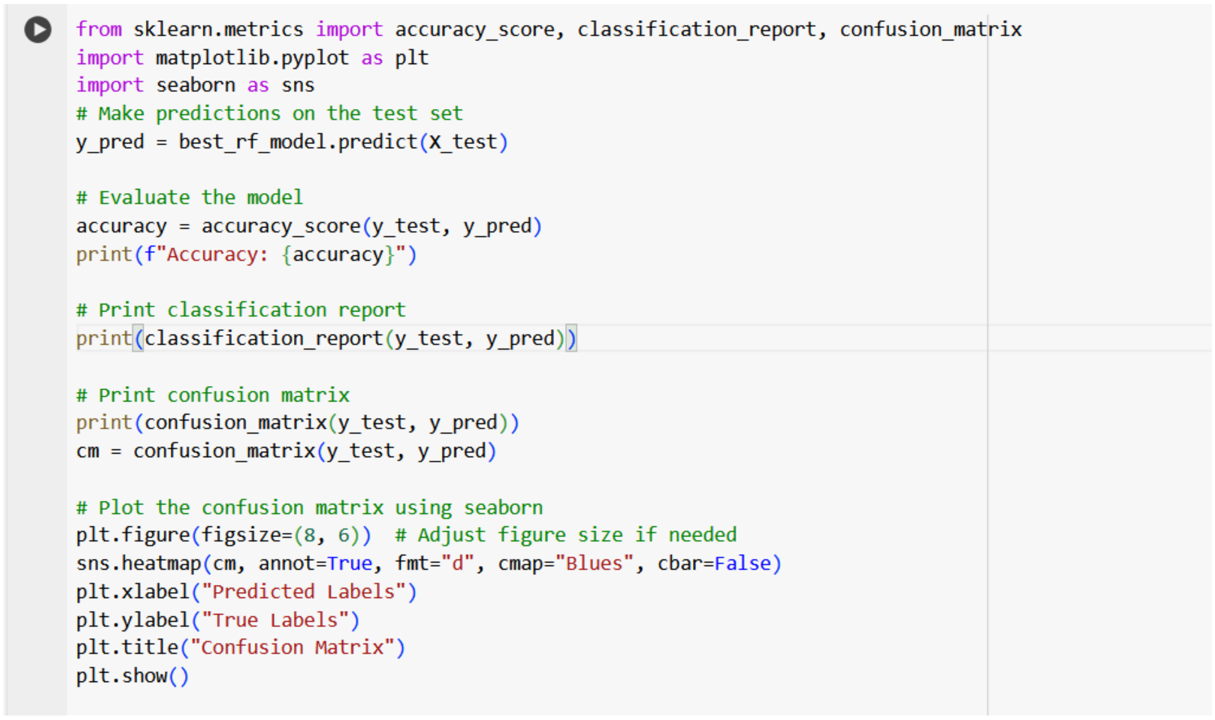
```python
for col in numerical_columns:
    df[col] = df[col].fillna(df[col].median())
#create new column duration
df["Duration"] = (df["End Date"] - df["Start Date"]).dt.days
# Select relevant columns for modeling

processed_df = df[

    [

        "Latitude",

        "Longitude",

        "temperature_2m_mean",

        "precipitation_sum",


        "wind_speed_10m_max",

        "Duration",

        "Flooded",

    ]

]
```



```python
from sklearn.preprocessing import MinMaxScaler
import joblib


scaler = MinMaxScaler()
numerical_features = ['temperature_2m_mean', 'precipitation_sum', 'wind_speed_10m_max']
processed_df.loc[:, numerical_features] = scaler.fit_transform(processed_df[numerical_features])


# Save the scaler
joblib.dump(scaler, 'minmax_scaler.pkl')
```



```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV


X = processed_df.drop('Flooded', axis=1)
y = processed_df['Flooded']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Define hyperparameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
}

# Use GridSearchCV for hyperparameter tuning and cross-validation
grid_search = GridSearchCV(rf_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best model
best_rf_model = grid_search.best_estimator_
```



```python
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
# Make predictions on the test set
y_pred = best_rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Print classification report
print(classification_report(y_test, y_pred))

# Print confusion matrix
print(confusion_matrix(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(8, 6))  # Adjust figure size if needed
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

The code prepares a dataset for machine learning by cleaning and processing the data, scales numerical features, trains a Random Forest model, and evaluates its performance using various metrics and visualizations.