1.Upload the file

```
from google.colab import files
uploaded = files.upload()
```

```
Choose Files garbage_bin_data.csv
• garbage_bin_data.csv(text/csv) - 759 bytes, last modified: 3/20/2025 - 100% done
Saving garbage_bin_data.csv to garbage_bin_data.csv
```

2.Load the dataset:

```
import pandas as pd
df = pd.read_csv('garbage_bin_data.csv')
print(df.head())  # Check if data is loaded correctly
```

```
   Bin ID        Date   Time Location  Week No  Fill Level (liters)  \
0     101  2025-03-20  12:00   Area A       12                   40
1     102  2025-03-20  12:30   Area B       12                   30
2     103  2025-03-20  13:00   Area C       12                   50
3     104  2025-03-20  13:30   Area D       12                   25
4     105  2025-03-20  14:00   Area E       12                   45

   Total (liters)  Fill Percentage  Latitude  Longitude  Temperature (°C)  \
0              50               80   40.7128    -74.006                25
1              50               60   40.7328    -73.996                27
2              50              100   40.7528    -73.986                29
3              50               50   40.7728    -73.976                24
4              50               90   40.7928    -73.966                28

   Battery Level
0             80
1             78
2             75
3             82
4             77
```

1 Install & Import Dependencies

```
# Install necessary libraries (if not installed)
!pip install xgboost seaborn matplotlib scikit-learn pandas numpy

# Import required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classi
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.14.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
```

```
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
```

2 Load & Explore Dataset

```
# Upload dataset manually in Google Colab (run this cell and upload the CSV file)
from google.colab import files
uploaded = files.upload()

# Load dataset
df = pd.read_csv('garbage_bin_data.csv')  # Change filename if needed

# Remove extra spaces from column names
df.columns = df.columns.str.strip()

# Display first few rows
print(df.head())

# Check for missing values
print("Missing Values:\n", df.isnull().sum())
```

```
Choose Files garbage_bin_data.csv
• garbage_bin_data.csv(text/csv) - 759 bytes, last modified: 3/20/2025 - 100% done
Saving garbage_bin_data.csv to garbage_bin_data (1).csv
   Bin ID        Date   Time Location  Week No  Fill Level (liters)  \
0     101  2025-03-20  12:00   Area A       12                   40
1     102  2025-03-20  12:30   Area B       12                   30
2     103  2025-03-20  13:00   Area C       12                   50
3     104  2025-03-20  13:30   Area D       12                   25
4     105  2025-03-20  14:00   Area E       12                   45

   Total (liters)  Fill Percentage  Latitude  Longitude  Temperature (°C)  \
0              50               80   40.7128    -74.006                25
1              50               60   40.7328    -73.996                27
2              50              100   40.7528    -73.986                29
3              50               50   40.7728    -73.976                24
4              50               90   40.7928    -73.966                28

   Battery Level
0             80
1             78
2             75
3             82
4             77
Missing Values:
 Bin ID                0
Date                   0
Time                   0
Location               0
Week No                0
Fill Level (liters)    0
Total (liters)         0
Fill Percentage        0
Latitude               0
Longitude              0
Temperature (°C)       0
Battery Level          0
dtype: int64
```

3 Data Preprocessing

```
# Drop irrelevant columns
df = df.drop(columns=['Bin ID', 'Date', 'Time', 'Location'])


# Handle missing values (fill or drop)
df = df.dropna()
```

```
# Ensure 'Fill Percentage' exists in dataset
if 'Fill Percentage' not in df.columns:
    raise ValueError("Error: 'Fill Percentage' column not found in dataset.")

# Define feature variables (X) and target variable (y)
X = df.drop(columns=['Fill Percentage'])
y = (df['Fill Percentage'] >= 80).astype(int)  # Binary classification (1: Full, 0: Not Full)

# Split dataset into training & testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalize feature variables
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print("Data Preprocessing Done ✅")
```

```
Data Preprocessing Done ✅
```

4️⃣ Train & Evaluate Models

```
# Define models
models = {
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='logloss')
}

# Train & Evaluate Models
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(f"\n📊 {name} Model Evaluation:")
    print("✅ Accuracy:", accuracy_score(y_test, y_pred))
    print("🎯 Precision:", precision_score(y_test, y_pred))
    print("🔁 Recall:", recall_score(y_test, y_pred))
    print("⚖️ F1 Score:", f1_score(y_test, y_pred))
    print("📉 Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("📄 Classification Report:\n", classification_report(y_test, y_pred))
```

```
📊 Logistic Regression Model Evaluation:
✅ Accuracy: 0.5
🎯 Precision: 0.5
🔁 Recall: 1.0
⚖️ F1 Score: 0.6666666666666666
📉 Confusion Matrix:
 [[0 1]
 [0 1]]
📄 Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
📊 Random Forest Model Evaluation:
✅ Accuracy: 0.5
🎯 Precision: 0.5
🔁 Recall: 1.0
⚖️ F1 Score: 0.6666666666666666
📉 Confusion Matrix:
 [[0 1]
 [0 1]]
📄 Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2


📊 XGBoost Model Evaluation:
✅ Accuracy: 1.0
🎯 Precision: 1.0
🔁 Recall: 1.0
⚖️ F1 Score: 1.0
📉 Confusion Matrix:
 [[1 0]
 [0 1]]
📄 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
```

5 Hyperparameter Tuning (Grid Search)

```
from sklearn.model_selection import GridSearchCV

# Convert X_train back to DataFrame (if needed)
X_train_df = pd.DataFrame(X_train, columns=X.columns)

# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 150],  # Number of trees in forest
    'max_depth': [5, 10, 20],        # Depth of each tree (removed None)
    'min_samples_split': [2, 5, 10]  # Min samples required to split a node
}

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(RandomForestClassifier(random_state=42),
                           param_grid,
                           cv=3,  # Use 3-fold cross-validation for faster tuning
                           scoring='accuracy',
                           n_jobs=-1)  # Use all CPU cores for faster processing

# Fit the model
grid_search.fit(X_train_df, y_train)

# Print Best Parameters
print("\n🏆 Best Parameters:", grid_search.best_params_)

# Use the best model from GridSearch
best_rf = grid_search.best_estimator_
```

```
🏆 Best Parameters: {'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 50}
```

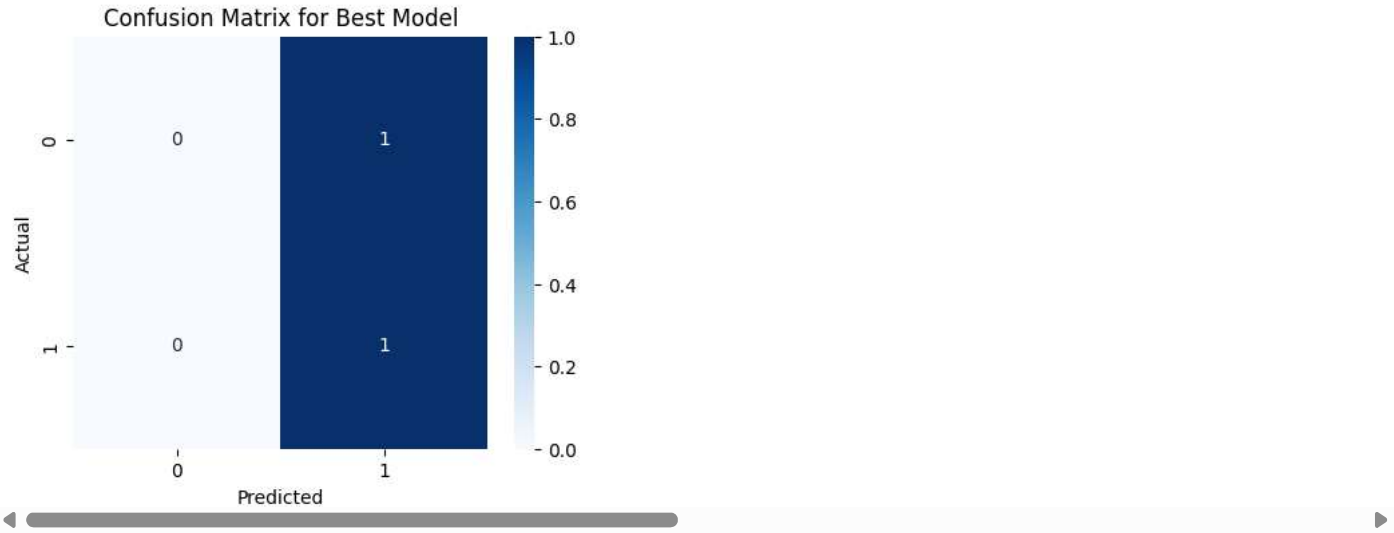
6 Confusion Matrix for Best Model

```
# Predict using best model
y_pred_best = best_rf.predict(X_test)

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_best)

# Plot heatmap
plt.figure(figsize=(5, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix for Best Model")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but RandomFo
  warnings.warn(
```




```
print("GridSearchCV Status:", hasattr(grid_search, 'best_estimator_'))
```

```
GridSearchCV Status: True
```

```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)

# Check for NaN values
print("Missing values in X_train:\n", pd.DataFrame(X_train).isnull().sum())
print("Missing values in y_train:\n", pd.DataFrame(y_train).isnull().sum())

# Check unique values in y_train
print("Unique values in y_train:", np.unique(y_train))
```

```
X_train shape: (8, 7)
y_train shape: (8,)
Missing values in X_train:
 0    0
1    0
2    0
3    0
4    0
5    0
6    0
dtype: int64
Missing values in y_train:
 Fill Percentage    0
dtype: int64
Unique values in y_train: [0 1]
```

```
# Assign the best model found from GridSearchCV
best_model = grid_search.best_estimator_



from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}

# Run GridSearchCV with cv=2
try:
    grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=2, scoring='accuracy')
    grid_search.fit(X_train, y_train)  # Fit the model

    # Assign best model
    best_model = grid_search.best_estimator_
    print("\n🏆 Best Parameters:", grid_search.best_params_)

except Exception as e:
    print("❌ GridSearchCV Failed! Error:", str(e))
```

```
🏆 Best Parameters: {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 50}
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classi

# Make predictions on the test set
y_pred = best_model.predict(X_test)

# Evaluate model performance
print("\n📊 Model Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
📊 Model Evaluation:
Accuracy: 0.5
Precision: 0.5
Recall: 1.0
F1 Score: 0.6666666666666666

Confusion Matrix:
 [[0 1]
 [0 1]]

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      1.00      0.67         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
import pandas as pd

# Ensure correct feature names
feature_names = X.columns.tolist()

# Example new data (modify values as needed)
new_data_values = [[11, 500, 1000, 40.618, -70.0060, 24, 78]]  # Ensure same number of features

# Convert to DataFrame with correct column names
new_data = pd.DataFrame(new_data_values, columns=feature_names)

# Debugging Step: Check new data format
print("\nNew Data Shape:", new_data.shape)
print("Expected Number of Features:", len(feature_names))
print("New Data Columns:", new_data.columns)

# Scale new data using previously fitted scaler
new_data_scaled = scaler.transform(new_data)

# Predict bin status using the best model
prediction = best_model.predict(new_data_scaled)

print("\n🗑️ Predicted Bin Status (1=Full, 0=Not Full):", prediction[0])
```

```
New Data Shape: (1, 7)
Expected Number of Features: 7
New Data Columns: Index(['Week No', 'Fill Level (liters)', 'Total (liters)', 'Latitude',
       'Longitude', 'Temperature (°C)', 'Battery Level'],
      dtype='object')

🗑️ Predicted Bin Status (1=Full, 0=Not Full): 1
```

```
import joblib

# Save the best model
joblib.dump(best_rf, 'garbage_bin_model.pkl')

# Download the model
from google.colab import files
files.download('garbage_bin_model.pkl')

print("🎉 Model Saved & Ready for Deployment!")
```