

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import (
    train_test_split, GridSearchCV, RandomizedSearchCV,
    cross_val_score, KFold
)
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix
)

# -----
# 1. Load Dataset
# -----
df = pd.read_excel("health_dataset.xlsx")
print("Dataset Loaded Successfully")
```

Dataset Loaded Successfully

```
In [2]: # -----
# 2. Encode categorical columns
# -----
cat_cols = df.select_dtypes(include=['object']).columns.tolist()

if "Target" in cat_cols:
    cat_cols.remove("Target")

le = LabelEncoder()
for col in cat_cols:
    df[col] = le.fit_transform(df[col])
```

```
if df["Target"].dtype == 'object':  
    df["Target"] = le.fit_transform(df["Target"])
```

```
In [3]: # -----  
# 3. Split Features/Target  
# -----  
X = df.drop("Target", axis=1)  
y = df["Target"]  
  
# Scaling  
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(X)  
  
X_train, X_test, y_train, y_test = train_test_split(  
    X_scaled, y, test_size=0.2, random_state=42, stratify=y  
)  
  
# Set cross-validation  
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

```
In [4]: # -----  
# 5. KNN TUNING  
# -----  
knn_params = {  
    "n_neighbors": list(range(1, 30)),  
    "weights": ["uniform", "distance"],  
    "metric": ["euclidean", "manhattan", "minkowski"]  
}  
  
knn = KNeighborsClassifier()  
  
knn_grid = GridSearchCV(knn, knn_params, cv=5, scoring="accuracy", n_jobs=-1)  
knn_grid.fit(X_train, y_train)  
  
print("\nBest KNN Parameters:", knn_grid.best_params_)  
knn_best = knn_grid.best_estimator_  
y_pred_knn = knn_best.predict(X_test)  
  
# Cross Validation  
cv_knn = cross_val_score(knn_best, X_scaled, y, cv=kfold).mean()
```

```
Best KNN Parameters: {'metric': 'euclidean', 'n_neighbors': 1, 'weights': 'uniform'}
```

```
In [5]: # =====
# 6. DECISION TREE TUNING
# =====
dt_params = {
    "criterion": ["gini", "entropy"],
    "max_depth": list(range(2, 50)),
    "min_samples_split": [2, 5, 10, 20],
    "min_samples_leaf": [1, 2, 5, 10]
}

dt = DecisionTreeClassifier(random_state=42)

dt_random = RandomizedSearchCV(
    dt, dt_params, cv=5, scoring="accuracy",
    n_iter=50, random_state=42, n_jobs=-1
)
dt_random.fit(X_train, y_train)

print("\nBest Decision Tree Parameters:", dt_random.best_params_)

dt_best = dt_random.best_estimator_
y_pred_dt = dt_best.predict(X_test)

# Cross Validation
cv_dt = cross_val_score(dt_best, X_scaled, y, cv=kfold).mean()
```

```
Best Decision Tree Parameters: {'min_samples_split': 20, 'min_samples_leaf': 1, 'max_depth': 25, 'criterion': 'gini'}
```

```
In [6]: # =====
# 7. RANDOM FOREST TUNING
# =====
rf_params = {
    "n_estimators": [50, 100, 200, 300, 500],
    "max_depth": [None, 10, 20, 30, 40, 50],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "bootstrap": [True, False]
}
```

```
rf = RandomForestClassifier(random_state=42)

rf_random = RandomizedSearchCV(
    rf, rf_params, cv=5, scoring="accuracy",
    n_iter=50, random_state=42, n_jobs=-1
)
rf_random.fit(X_train, y_train)

print("\nBest Random Forest Parameters:", rf_random.best_params_)

rf_best = rf_random.best_estimator_
y_pred_rf = rf_best.predict(X_test)

# Cross Validation
cv_rf = cross_val_score(rf_best, X_scaled, y, cv=kfold).mean()
```

Best Random Forest Parameters: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': 40, 'bootstrap': True}

In [7]:

```
# =====
# 8. METRIC FUNCTION
# =====

def evaluate(name, y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, average="weighted")
    rec = recall_score(y_true, y_pred, average="weighted")
    f1 = f1_score(y_true, y_pred, average="weighted")

    print(f"\n==== {name} ====")
    print("Accuracy:", acc)
    print("Precision:", prec)
    print("Recall:", rec)
    print("F1 Score:", f1)

    return acc, prec, rec, f1

acc_knn, prec_knn, rec_knn, f1_knn = evaluate("Optimized KNN", y_test, y_pred_knn)
acc_dt, prec_dt, rec_dt, f1_dt = evaluate("Optimized Decision Tree", y_test, y_pred_dt)
acc_rf, prec_rf, rec_rf, f1_rf = evaluate("Optimized Random Forest", y_test, y_pred_rf)
```

```
===== Optimized KNN =====
Accuracy: 0.9900990099009901
Precision: 0.9903818953323903
Recall: 0.9900990099009901
F1 Score: 0.9900968682202419

===== Optimized Decision Tree =====
Accuracy: 0.9603960396039604
Precision: 0.9645648775403856
Recall: 0.9603960396039604
F1 Score: 0.9607345349919606

===== Optimized Random Forest =====
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

```
In [8]: # -----
# 9. BEST MODEL
# -----
accuracies = {
    "KNN": acc_knn,
    "Decision Tree": acc_dt,
    "Random Forest": acc_rf
}

best_model = max(accuracies, key=accuracies.get)

print("\n=====")
print(" BEST MODEL AFTER TUNING:", best_model)
print("=====")
```

```
=====
BEST MODEL AFTER TUNING: Random Forest
=====
```

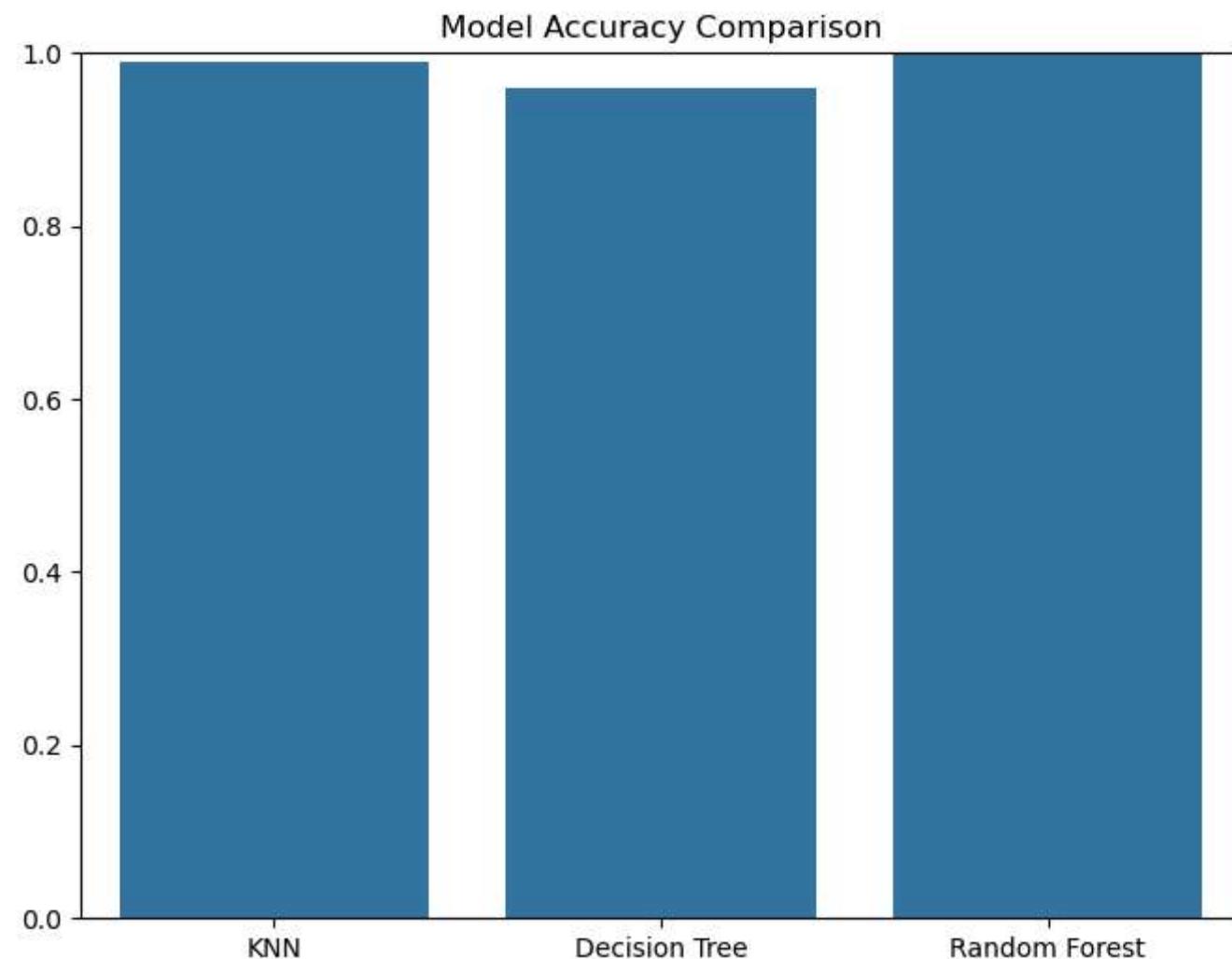
```
In [9]: # -----
# 10. GRAPHS
# -----
```

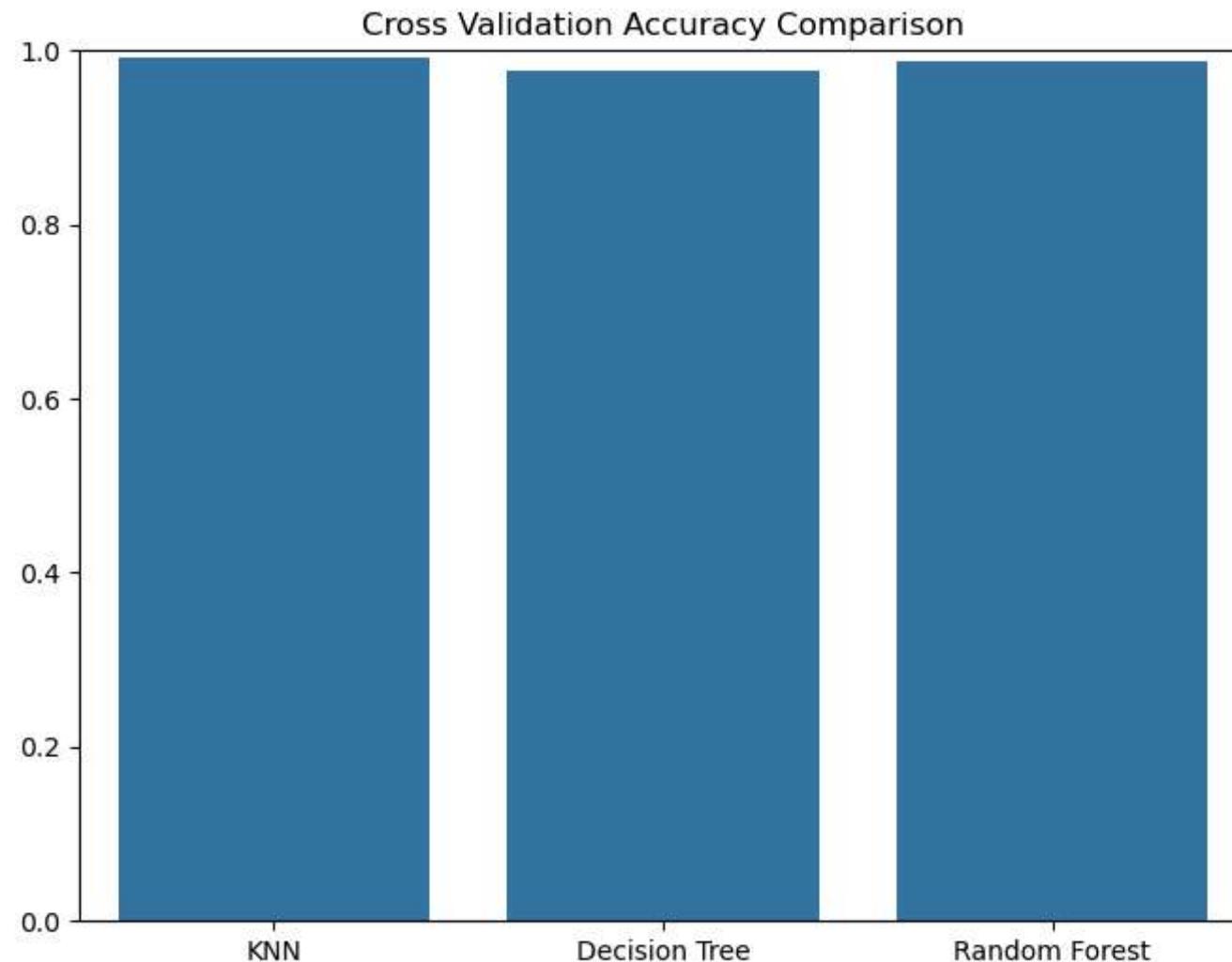
```
# --- Accuracy Comparison ---
plt.figure(figsize=(8,6))
models = ["KNN", "Decision Tree", "Random Forest"]
acc_values = [acc_knn, acc_dt, acc_rf]

sns.barplot(x=models, y=acc_values)
plt.title("Model Accuracy Comparison")
plt.ylim(0, 1)
plt.show()

# --- Cross Validation Comparison ---
plt.figure(figsize=(8,6))
cv_scores = [cv_knn, cv_dt, cv_rf]

sns.barplot(x=models, y=cv_scores)
plt.title("Cross Validation Accuracy Comparison")
plt.ylim(0, 1)
plt.show()
```



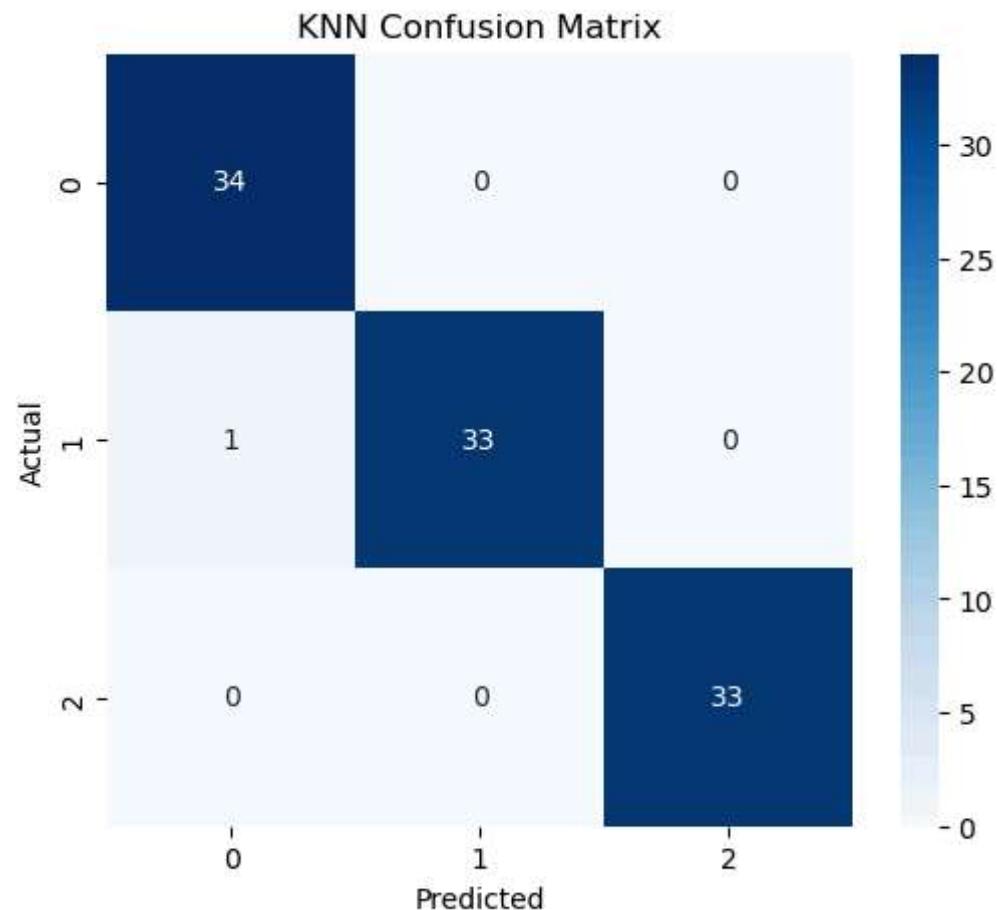


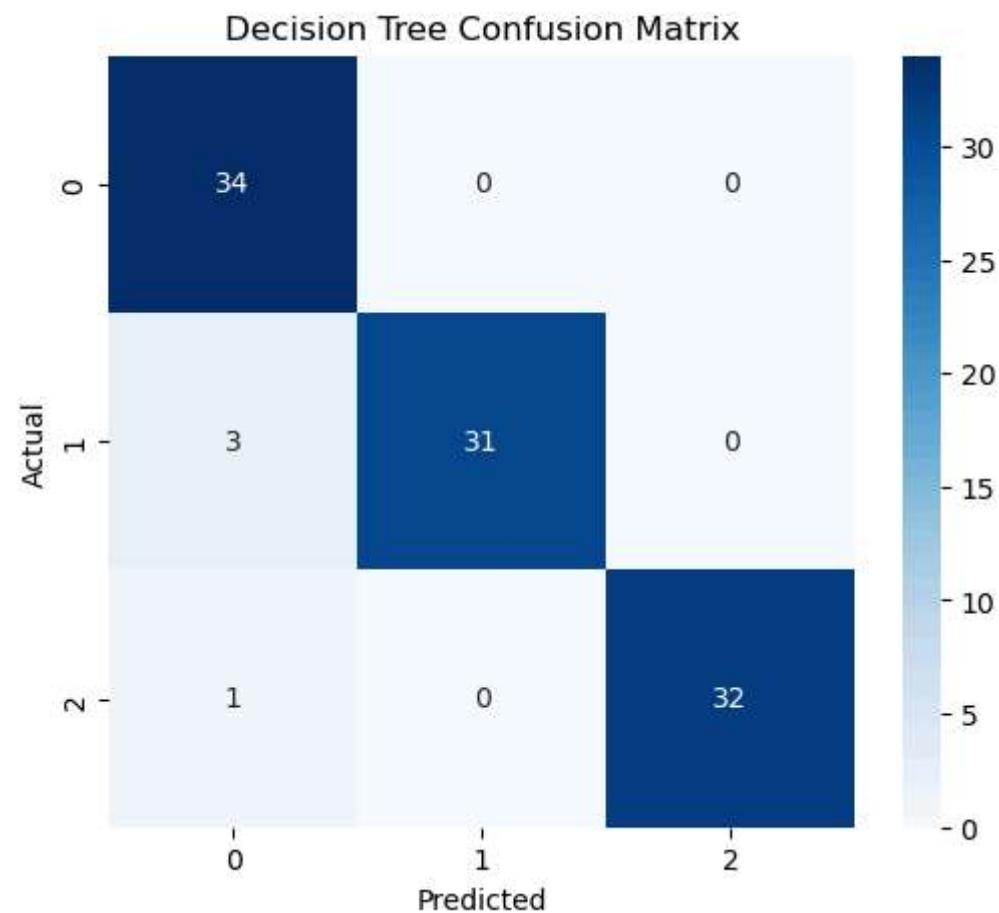
```
In [12]: # --- Confusion Matrices ---
```

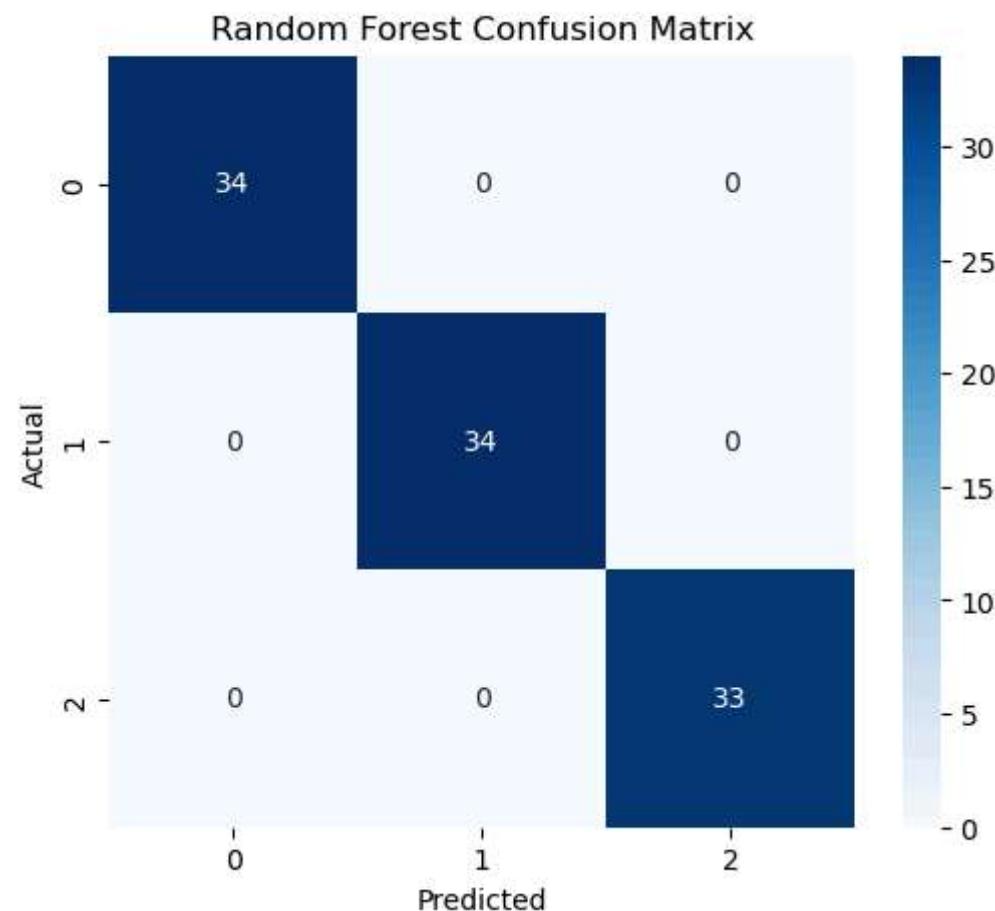
```
def plot_cm(y_true, y_pred, title):  
    plt.figure(figsize=(6,5))  
    cm = confusion_matrix(y_true, y_pred)  
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  
    plt.title(title)  
    plt.xlabel("Predicted")
```

```
plt.ylabel("Actual")
plt.show()

plot_cm(y_test, y_pred_knn, "KNN Confusion Matrix")
plot_cm(y_test, y_pred_dt, "Decision Tree Confusion Matrix")
plot_cm(y_test, y_pred_rf, "Random Forest Confusion Matrix")
```







In []: