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

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving diabetes.csv to diabetes.csv
```

```
column_names = [
    "ID", "Diagnosis",
    "radius1", "texture1", "perimeter1", "area1", "smoothness1", "compactness1", "concavity1", "concave_
    "radius2", "texture2", "perimeter2", "area2", "smoothness2", "compactness2", "concavity2", "concave_
    "radius3", "texture3", "perimeter3", "area3", "smoothness3", "compactness3", "concavity3", "concave_
]
import pandas as pd

df = pd.read_csv("wdbc.csv", header=None, names=column_names)
output_path = "D:\ml\ass4\diagnostic\diabetes.csv"
df.to_csv(output_path, index=False)
from google.colab import files
files.download(output_path)

print(f"Saved successfully to: {output_path}")
```

```
<>:10: SyntaxWarning: invalid escape sequence '\m'
<>:10: SyntaxWarning: invalid escape sequence '\m'
/tmp/ipython-input-684899759.py:10: SyntaxWarning: invalid escape sequence '\m'
  output_path = "D:\ml\ass4\diagnostic\diabetes.csv"
Saved successfully to: D:\ml\ass4\diagnostic\diabetes.csv
```

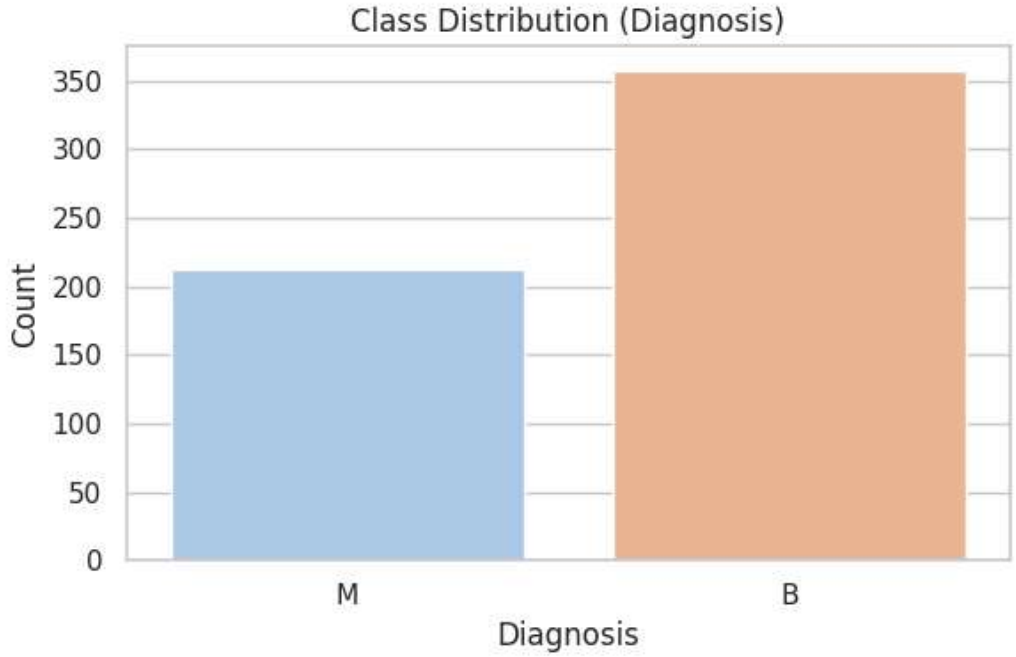
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Set visual style
sns.set(style="whitegrid")

# ------------------------------
# 1. Class Distribution (Diagnosis)
# ------------------------------
plt.figure(figsize=(6, 4))
sns.countplot(x='Diagnosis', data=df, palette='pastel')
plt.title("Class Distribution (Diagnosis)")
plt.xlabel("Diagnosis")
plt.ylabel("Count")
plt.tight_layout()
plt.show()

# ------------------------------
# 2. Correlation Heatmap
# ------------------------------
numeric_df = df.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numeric_df.corr()

plt.figure(figsize=(16, 14))
```

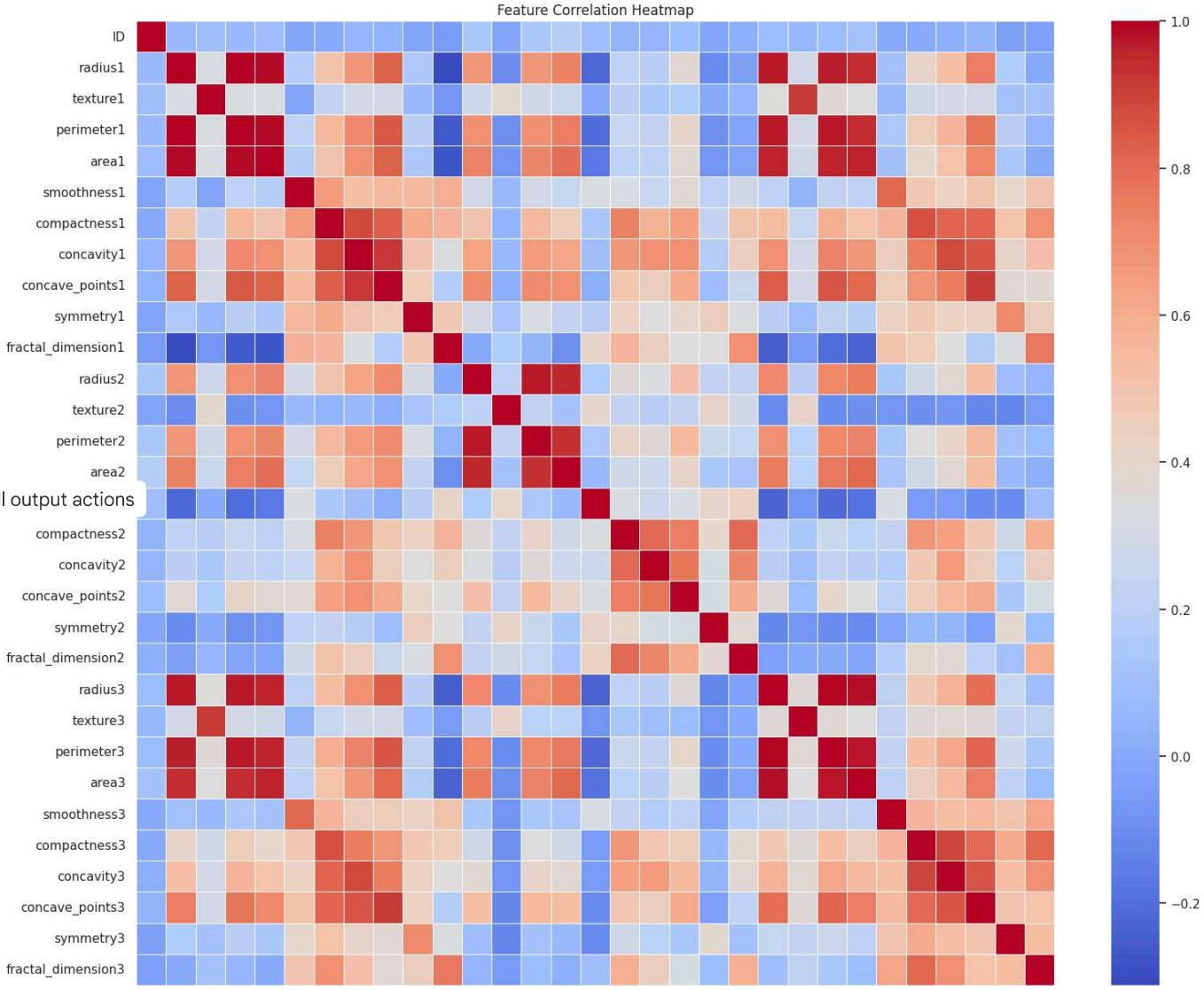
Code cell output actions

```
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.tight_layout()
plt.show()

# ------------------------------
# 3. Top 5 Correlated Feature Pairs
# ------------------------------
# Extract upper triangle without diagonal
corr_pairs = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(bool))
top_corr = corr_pairs.unstack().dropna().abs().sort_values(ascending=False).head(5)

print(" Top 5 most correlated feature pairs:")
print(top_corr)
```

Code cell output actions

```
/tmp/ipython-input-1980949047.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `

  sns.countplot(x='Diagnosis', data=df, palette='pastel')
```





Code cell output actions


```
 Top 5 most correlated feature pairs:
perimeter1  radius1       0.997855
perimeter3  radius3       0.993708
area1       radius1       0.987357
            perimeter1    0.986507
area3       radius3       0.984015
dtype: float64
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder


label_encoder = LabelEncoder()
df['Diagnosis'] = label_encoder.fit_transform(df['Diagnosis'])  # M=1, B=0

# Split features and target
X = df.drop('Diagnosis', axis=1)
y = df['Diagnosis']

# Split the dataset (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Feature scaling (standardization)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Check shapes
print("Training set:", X_train_scaled.shape)
print("Test set:", X_test_scaled.shape)
```

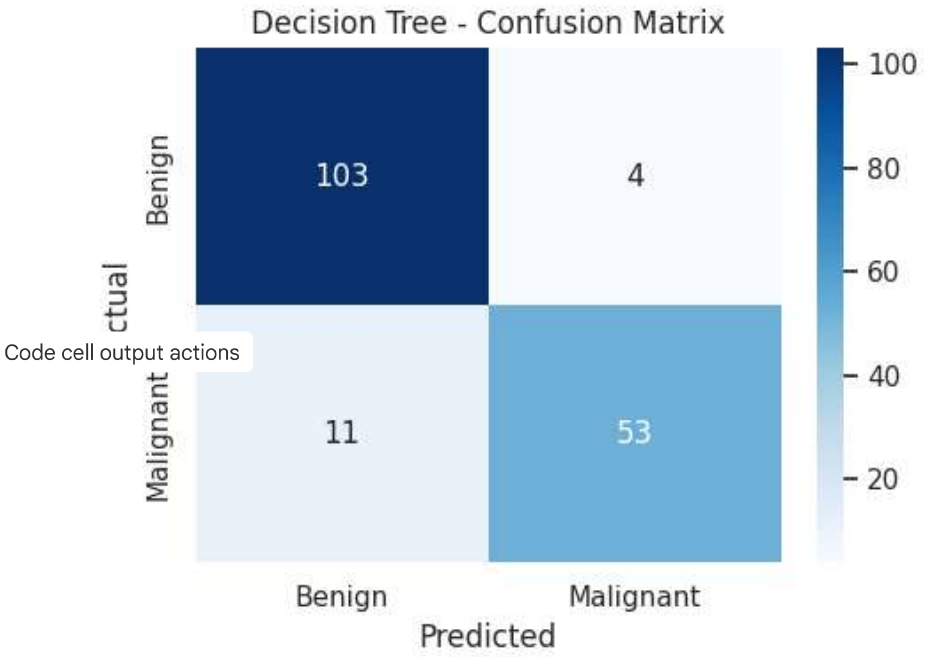
Code cell output actions

```
: (398, 31)
est set: (171, 31)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train Decision Tree Classifier
```

```
# Train Decision Tree Classifier
dt_model = DecisionTreeClassifier(criterion="gini",random_state=42)
dt_model.fit(X_train_scaled, y_train)

# Predict
y_pred = dt_model.predict(X_test_scaled)

# Evaluate
print(" Accuracy:", accuracy_score(y_test, y_pred))
print("\n Classification Report:\n", classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malignant'], yticklabels=['Beni
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title("Decision Tree - Confusion Matrix")
plt.tight_layout()
plt.show()
```

```
Accuracy: 0.9122807017543859

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.96      0.93       107
           1       0.93      0.83      0.88        64

    accuracy                           0.91       171
   macro avg       0.92      0.90      0.90       171
weighted avg       0.91      0.91      0.91       171
```



Code cell output actions

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# Define parameter grid
param_grid = {
```

```
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize base model
dt_model = DecisionTreeClassifier(random_state=42)

# Setup GridSearchCV
grid_search = GridSearchCV(
    estimator=dt_model,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    verbose=1,
    n_jobs=-1
)

# Fit on training data (use scaled data)
grid_search.fit(X_train_scaled, y_train)

# Best hyperparameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best Parameters: {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 1
Best Accuracy: 0.9445886075949368
```

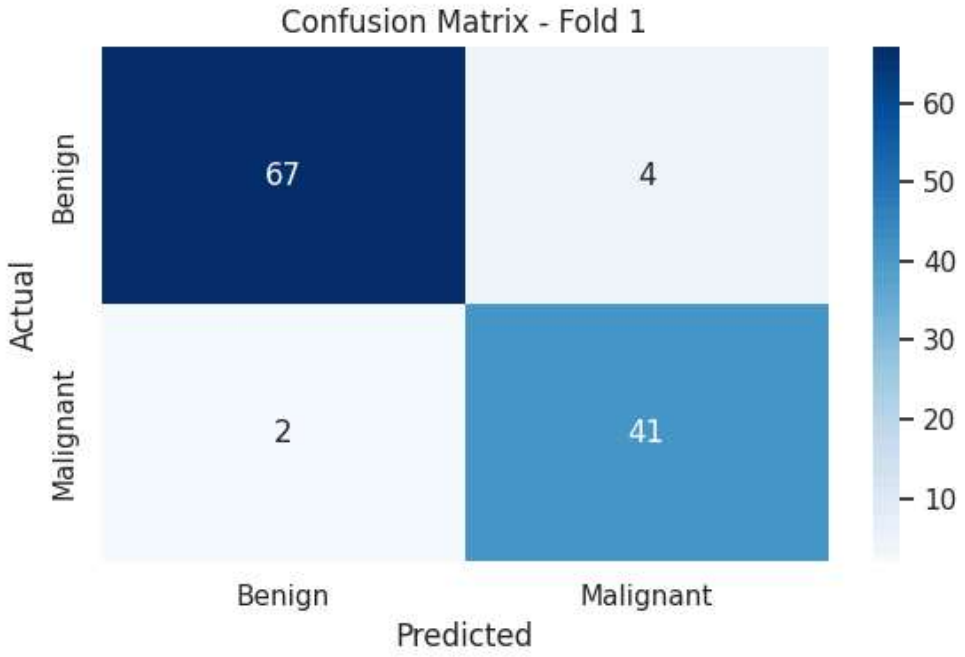
```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Get best model from grid search
best_dt = grid_search.best_estimator_

# Define 5-fold stratified cross-validation
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

fold = 1
accuracies = []

for train_index, test_index in skf.split(X, y):
    # Split using iloc (works for pandas)
    X_train_fold, X_test_fold = X.iloc[train_index], X.iloc[test_index]
    y_train_fold, y_test_fold = y.iloc[train_index], y.iloc[test_index]

    # Scale (if you were scaling manually earlier)
    X_train_fold_scaled = scaler.fit_transform(X_train_fold)
    X_test_fold_scaled = scaler.transform(X_test_fold)

    # Train model
    best_dt.fit(X_train_fold_scaled, y_train_fold)
```

Code cell output actions

```
        # Predict
        y_pred_fold = best_dt.predict(X_test_fold_scaled)

        # Accuracy
        acc = accuracy_score(y_test_fold, y_pred_fold)
        accuracies.append(acc)

        print(f"\n===== Fold {fold} =====")
        print("Accuracy:", acc)
        print("Classification Report:\n", classification_report(y_test_fold, y_pred_fold))

        # Confusion Matrix
        cm = confusion_matrix(y_test_fold, y_pred_fold)
        plt.figure(figsize=(6, 4))
        sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                    xticklabels=['Benign', 'Malignant'],
                    yticklabels=['Benign', 'Malignant'])
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.title(f"Confusion Matrix - Fold {fold}")
        plt.tight_layout()
        plt.show()

        fold += 1

    # Print overall summary
    print("\n==== Cross-validation Summary ====")
    print("Accuracies per fold:", accuracies)
    print("Mean Accuracy:", np.mean(accuracies))
    print("Std Deviation:", np.std(accuracies))
```

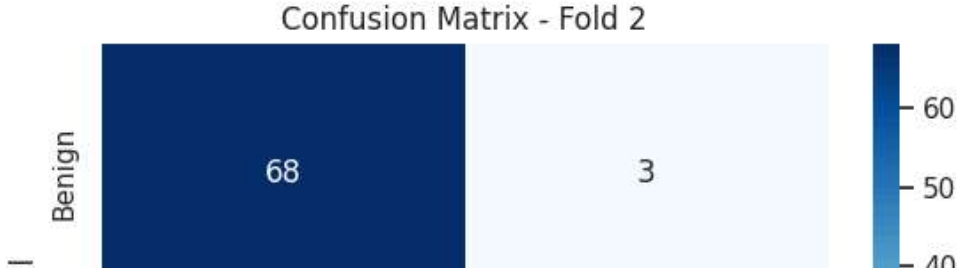
Code cell output actions

```
===== Fold 1 =====
Accuracy: 0.9473684210526315
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.94      0.96        71
           1       0.91      0.95      0.93        43

    accuracy                           0.95       114
   macro avg       0.94      0.95      0.94       114
weighted avg       0.95      0.95      0.95       114
```




```
===== Fold 2 =====
Accuracy: 0.8947368421052632
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92        71
           1       0.92      0.79      0.85        43

    accuracy                           0.89       114
   macro avg       0.90      0.87      0.88       114
weighted avg       0.90      0.89      0.89       114
```

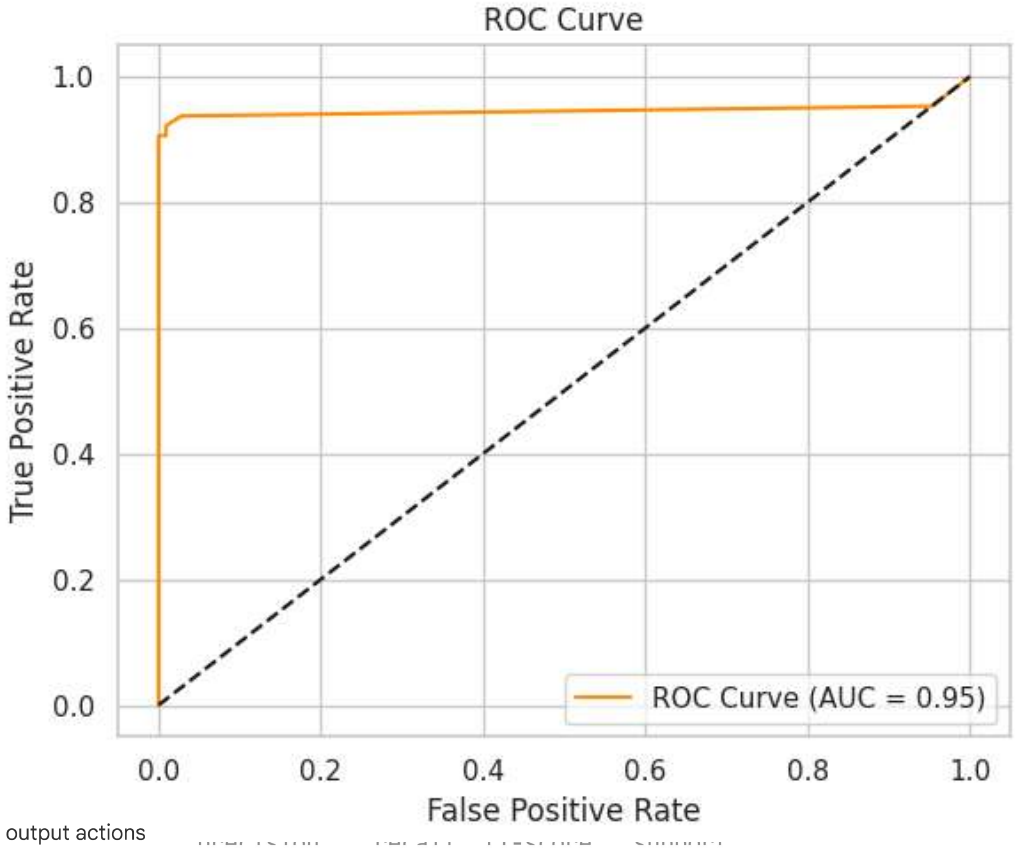
Code cell output actions




```
from sklearn.metrics import roc_curve, auc

# Get predicted probabilities for class 1 (Malignant)
y_test_proba = best_dt.predict_proba(X_test_scaled)[:, 1]
```

```
# Compute ROC curve and AUC
fpr, tpr, _ = roc_curve(y_test, y_test_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='darkorange', label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--')  # diagonal line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
weighted avg       0.93      0.93      0.93       114
```



Code cell output actions

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Define AdaBoost with Decision Tree base
adaboost = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(random_state=42),
    random_state=42
)
```

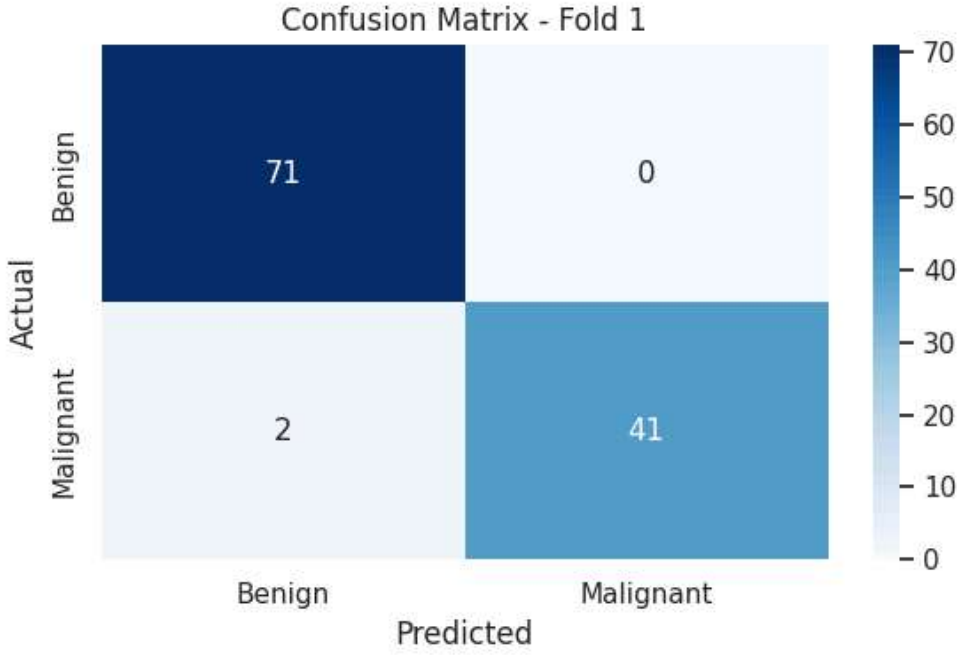
```
# Hyperparameter grid for AdaBoost
param_grid = {
    "estimator__max_depth": [1, 2, 3],
    "n_estimators": [50, 100, 200],
    "learning_rate": [0.01, 0.1, 1.0]
}

# Grid search
grid_search_ada = GridSearchCV(
    estimator=adaboost,
    param_grid=param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
grid_search_ada.fit(X_train_scaled, y_train)

# Best model
best_ada = grid_search_ada.best_estimator_
print("Best Parameters:", grid_search_ada.best_params_)

# ===== 5-Fold Cross Validation Evaluation =====
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
fold = 1
accuracies = []

for train_index, test_index in skf.split(X, y):
    X_train_fold, X_test_fold = X.iloc[train_index], X.iloc[test_index]
    y_train_fold, y_test_fold = y.iloc[train_index], y.iloc[test_index]

    # Scale manually (if not inside pipeline)
    X_train_fold_scaled = scaler.fit_transform(X_train_fold)
    X_test_fold_scaled = scaler.transform(X_test_fold)

    # Train on current fold
    best_ada.fit(X_train_fold_scaled, y_train_fold)

    # Predict
    y_pred_fold = best_ada.predict(X_test_fold_scaled)

    # Accuracy
    acc = accuracy_score(y_test_fold, y_pred_fold)
    accuracies.append(acc)

    print(f"\n===== Fold {fold} =====")
    print("Accuracy:", acc)
    print("Classification Report:\n", classification_report(y_test_fold, y_pred_fold))

    # Confusion Matrix
    cm = confusion_matrix(y_test_fold, y_pred_fold)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Benign', 'Malignant'],
                yticklabels=['Benign', 'Malignant'])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - Fold {fold}")
    plt.tight_layout()
```

Code cell output actions

```
        plt.show()

        fold += 1

    # Print overall summary
    print("\n==== Cross-validation Summary ====")
    print("Accuracies per fold:", accuracies)
    print("Mean Accuracy:", np.mean(accuracies))
    print("Std Deviation:", np.std(accuracies))
```

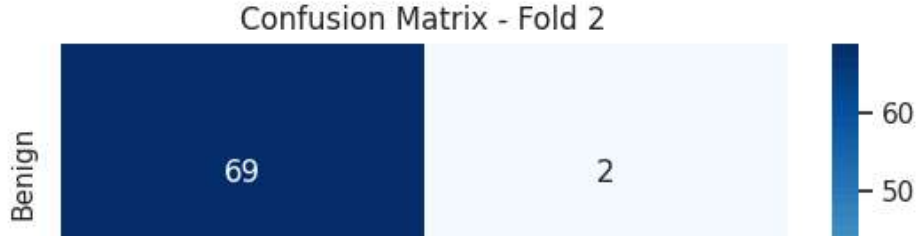
Code cell output actions

```
Best Parameters: {'estimator__max_depth': 1, 'learning_rate': 1.0, 'n_estimators': 200}

===== Fold 1 =====
Accuracy: 0.9824561403508771
Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        71
           1       1.00      0.95      0.98        43

    accuracy                           0.98       114
   macro avg       0.99      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```




```
===== Fold 2 =====
Accuracy: 0.9649122807017544
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        71
           1       0.95      0.95      0.95        43

    accuracy                           0.96       114
   macro avg       0.96      0.96      0.96       114
weighted avg       0.96      0.96      0.96       114
```

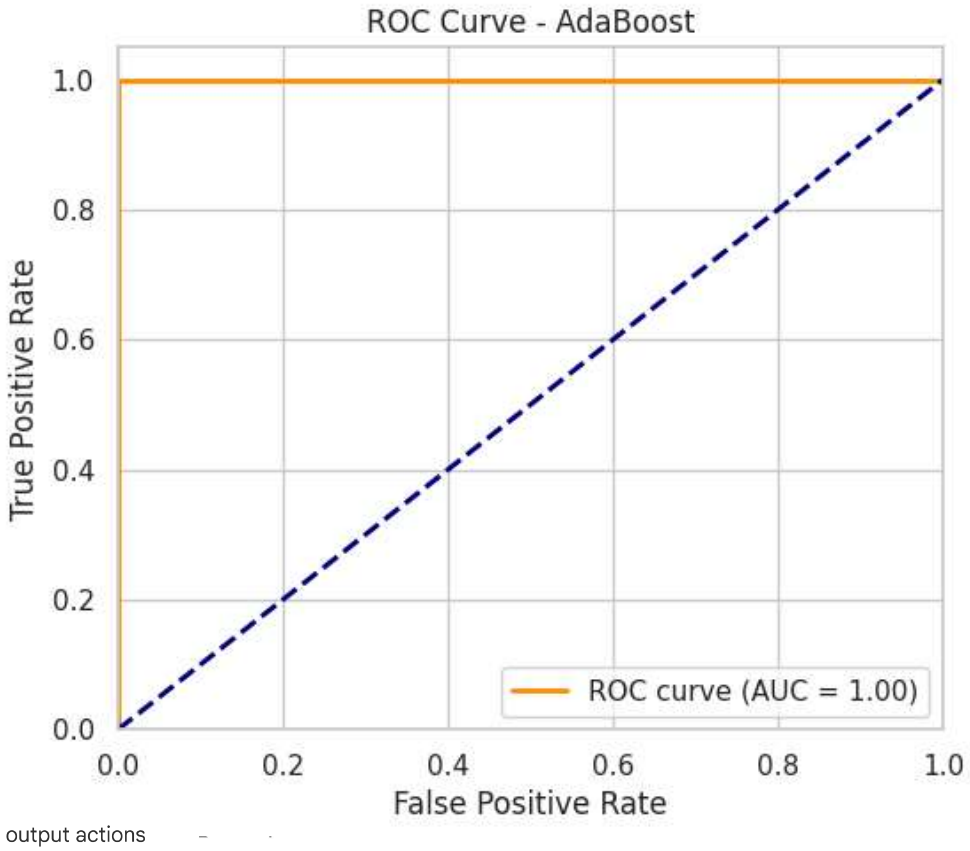
Code cell output actions




```
from sklearn.metrics import roc_curve, auc, RocCurveDisplay

# After training best_ada on the full training set:
y_test_pred_proba = best_ada.predict_proba(X_test_scaled)[:, 1]  # probability of class 1
```

```
# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color="darkorange", lw=2, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")  # diagonal line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - AdaBoost")
plt.legend(loc="lower right")
plt.show()
```

```
macro avg       0.97      0.96      0.96       114
```



Code cell output actions

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Define Gradient Boosting model
gb = GradientBoostingClassifier(random_state=42)

# Hyperparameter grid
param_grid = {
    "n_estimators": [50, 100, 200],
    "learning_rate": [0.01, 0.1, 0.2],
```

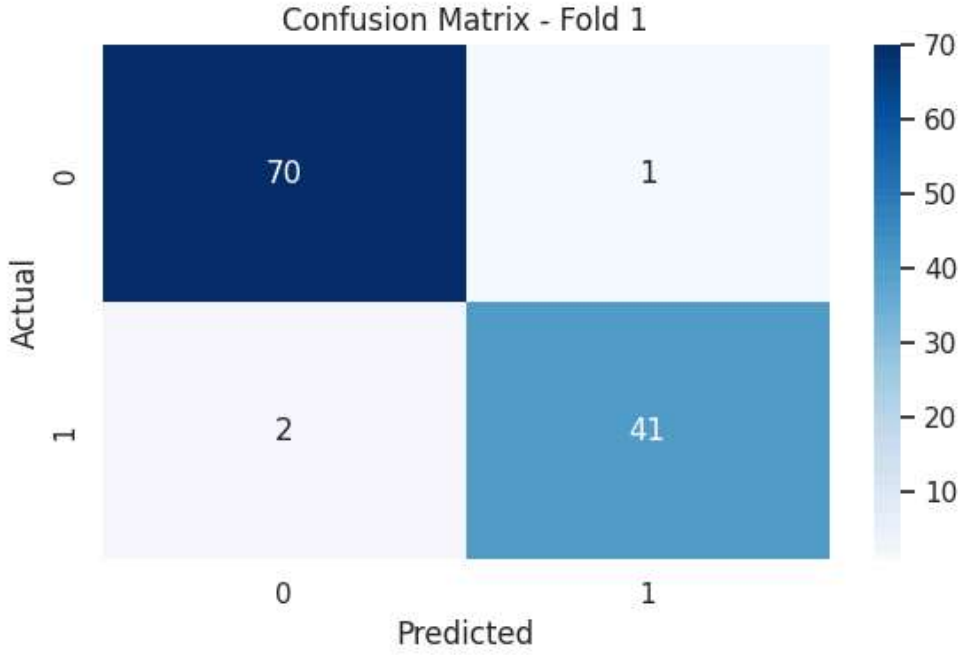
```
    "max_depth": [2, 3, 4]
}

# Grid search
grid_search_gb = GridSearchCV(
    estimator=gb,
    param_grid=param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
grid_search_gb.fit(X_train_scaled, y_train)

# Best model
best_gb = grid_search_gb.best_estimator_
print("Best Parameters:", grid_search_gb.best_params_)

# ===== 5-Fold Cross Validation =====
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
fold = 1
accuracies = []

for train_index, test_index in skf.split(X, y):
    X_train_fold, X_test_fold = X.iloc[train_index], X.iloc[test_index]
    y_train_fold, y_test_fold = y.iloc[train_index], y.iloc[test_index]

    # Scale manually
    X_train_fold_scaled = scaler.fit_transform(X_train_fold)
    X_test_fold_scaled = scaler.transform(X_test_fold)

    # Train
    best_gb.fit(X_train_fold_scaled, y_train_fold)

    # Predict
    y_pred_fold = best_gb.predict(X_test_fold_scaled)

    # Accuracy
    acc = accuracy_score(y_test_fold, y_pred_fold)
    accuracies.append(acc)

    print(f"\n===== Fold {fold} =====")
    print("Accuracy:", acc)
    print("Classification Report:\n", classification_report(y_test_fold, y_pred_fold))

    # Confusion Matrix
    cm = confusion_matrix(y_test_fold, y_pred_fold)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=[0, 1],
                yticklabels=[0, 1])
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix - Fold {fold}")
    plt.tight_layout()
    plt.show()

    fold += 1
```

Code cell output actions

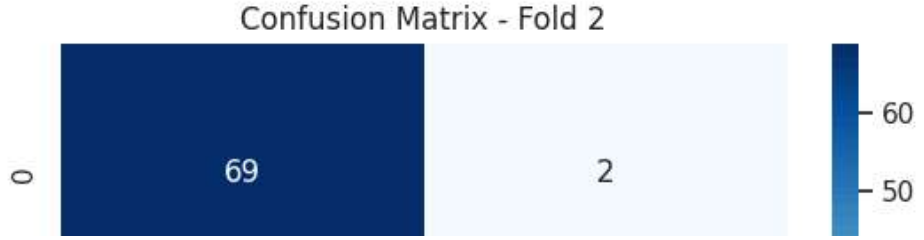
```
# Print overall CV summary
print("\n==== Cross-validation Summary ====")
print("Accuracies per fold:", accuracies)
print("Mean Accuracy:", np.mean(accuracies))
print("Std Deviation:", np.std(accuracies))

# ===== Final ROC curve on test set =====
y_test_pred_proba = best_gb.predict_proba(X_test_scaled)[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_test_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color="darkorange", lw=2, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Gradient Boosting (Test Set)")
plt.legend(loc="lower right")
plt.show()
```

Code cell output actions

```
Best Parameters: {'learning_rate': 0.2, 'max_depth': 2, 'n_estimators': 100}

===== Fold 1 =====
Accuracy: 0.9736842105263158
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        71
           1       0.98      0.95      0.96        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```




```
===== Fold 2 =====
Accuracy: 0.9210526315789473
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        71
           1       0.95      0.84      0.89        43

Code cell output actions                           0.92       114
   macro avg       0.93      0.90      0.91       114
weighted avg       0.92      0.92      0.92       114
```

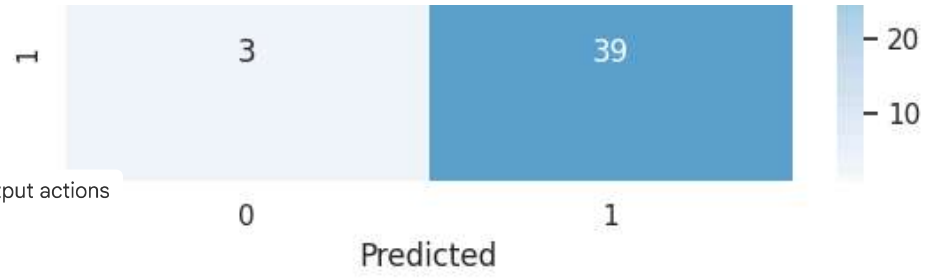



```
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, auc
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import numpy as np

# Define XGBoost model
xgb = XGBClassifier(use_label_encoder=False, eval_metric="logloss", random_state=42)

# Hyperparameter grid
param_grid = {
    "n_estimators": [50, 100, 200],
    "learning_rate": [0.01, 0.1, 0.2],
    "max_depth": [2, 3, 4],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.8, 1.0]
}

# Grid search
grid_search_xgb = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
grid_search_xgb.fit(X_train_scaled, y_train)

# Best model
best_xgb = grid_search_xgb.best_estimator_
print("Best Parameters:", grid_search_xgb.best_params_)

# ===== 5-Fold Cross Validation (per-fold accuracy only) =====
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
fold = 1
accuracies = []

for train_index, test_index in skf.split(X, y):
    X_train_fold, X_test_fold = X.iloc[train_index], X.iloc[test_index]
    y_train_fold, y_test_fold = y.iloc[train_index], y.iloc[test_index]

    # Scale manually
    X_train_fold_scaled = scaler.fit_transform(X_train_fold)
    X_test_fold_scaled = scaler.transform(X_test_fold)

    # Train
    best_xgb.fit(X_train_fold_scaled, y_train_fold)

    # Predict
    y_pred_fold = best_xgb.predict(X_test_fold_scaled)

    # Accuracy
    acc = accuracy_score(y_test_fold, y_pred_fold)
    accuracies.append(acc)

    print(f"\n===== Fold {fold} =====")
    print("Accuracy:", acc)
    print("Classification Report:\n", classification_report(y_test_fold, y_pred_fold))

    fold += 1

# Print overall CV summary
```

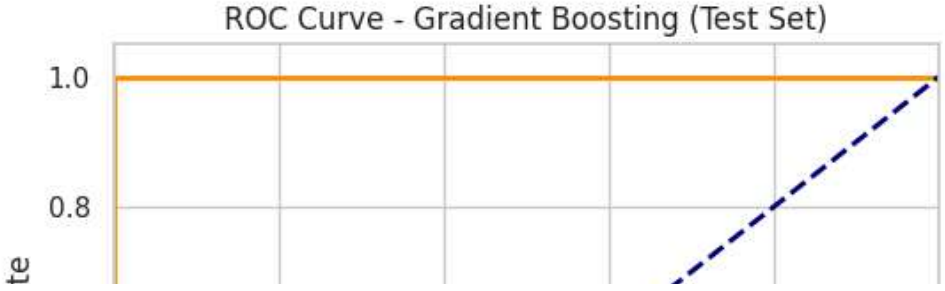
Code cell output actions

```
print("\n==== Cross-validation Summary ====")
print("Accuracies per fold:", accuracies)
print("Mean Accuracy:", np.mean(accuracies))
print("Std Deviation:", np.std(accuracies))

# ===== Final Evaluation on Test Set =====
y_test_pred = best_xgb.predict(X_test_scaled)
y_test_pred_proba = best_xgb.predict_proba(X_test_scaled)[:, 1]

# Confusion Matrix
cm = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=[0, 1],
            yticklabels=[0, 1])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - XGBoost (Test Set)")
plt.tight_layout()
plt.show()

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_test_pred_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color="darkorange", lw=2, label=f"ROC curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - XGBoost (Test Set)")
plt.legend(loc="lower right")
plt.show()
```

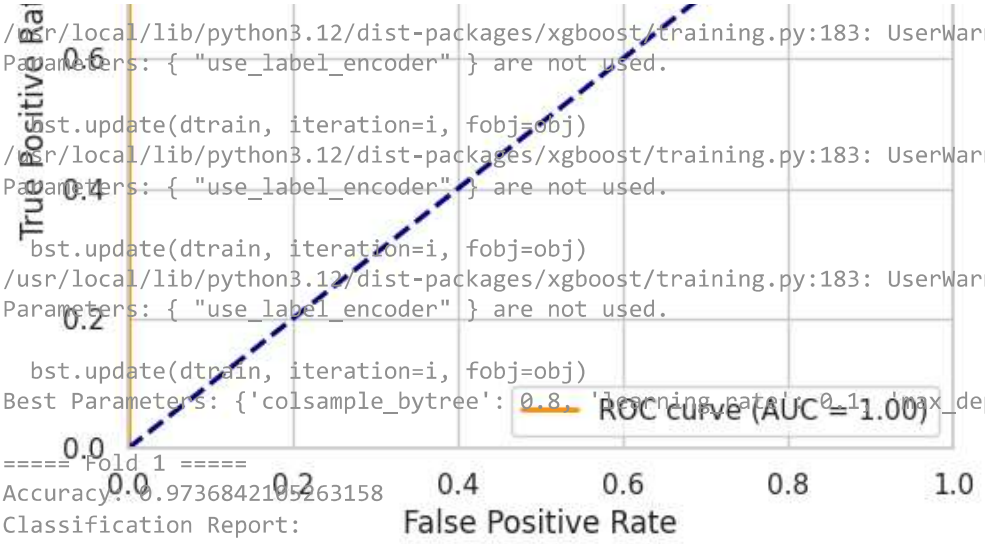


Code cell output actions

```
==== Cross-validation Summary ====
Accuracies per fold: [0.9736842105263158, 0.9210526315789473, 0.9473684210526315, 0.956140350877193,
Mean Accuracy: 0.9525694767893185
Std Deviation: 0.018020664322408134
```



```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:36:22] WARNING: /w
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:36:22] WARNING: /w
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:36:22] WARNING: /w
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Best Parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 2, 'n_estimators': 100
```




```
===== Fold 1 =====
Accuracy: 0.9736842105263158
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98        71
           1       0.95      0.98      0.97        43

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114


===== Fold 2 =====
Accuracy: 0.956140350877193
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.97        71
           1       0.95      0.93      0.94        43

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:36:22] WARNING: /w
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:36:22] WARNING: /w
Parameters: { "use_label_encoder" } are not used.
```

Code cell output actions

```
  bst.update(dtrain, iteration=i, fobj=obj)
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [04:36:22] WARNING: /w
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)

===== Fold 3 =====
Accuracy: 0.9473684210526315
Classification Report:
              precision    recall  f1-score   support
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import StratifiedKFold, GridSearchCV
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, r
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Example dataset (replace with your own diabetes dataset)
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=500, n_features=10, n_classes=2, random_state=42)

# ------------------------------
# Step 1: 5-fold Cross-validation
# ------------------------------

# Random Forest parameters
rf_params = {
    'n_estimators': 100,
    'max_depth': None,
    'min_samples_split': 2,
    'min_samples_leaf': 1,
    'max_features': 'sqrt',
    'random_state': 42
}

# Initialize model
model = RandomForestClassifier(**rf_params)

# Prepare for 5-fold CV
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Initialize performance lists
accuracy_list = []
precision_list = []
recall_list = []
f1_list = []

# Store all true and predicted labels for final metrics
y_true_all = []
y_pred_all = []
y_prob_all = []

for fold, (train_idx, test_idx) in enumerate(kf.split(X, y), 1):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # Fit the model
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    # Store results
    y_true_all.extend(y_test)
    y_pred_all.extend(y_pred)
    y_prob_all.extend(y_prob)

    # Compute metrics for this fold
    acc = accuracy_score(y_test, y_pred)
```

Code cell output actions

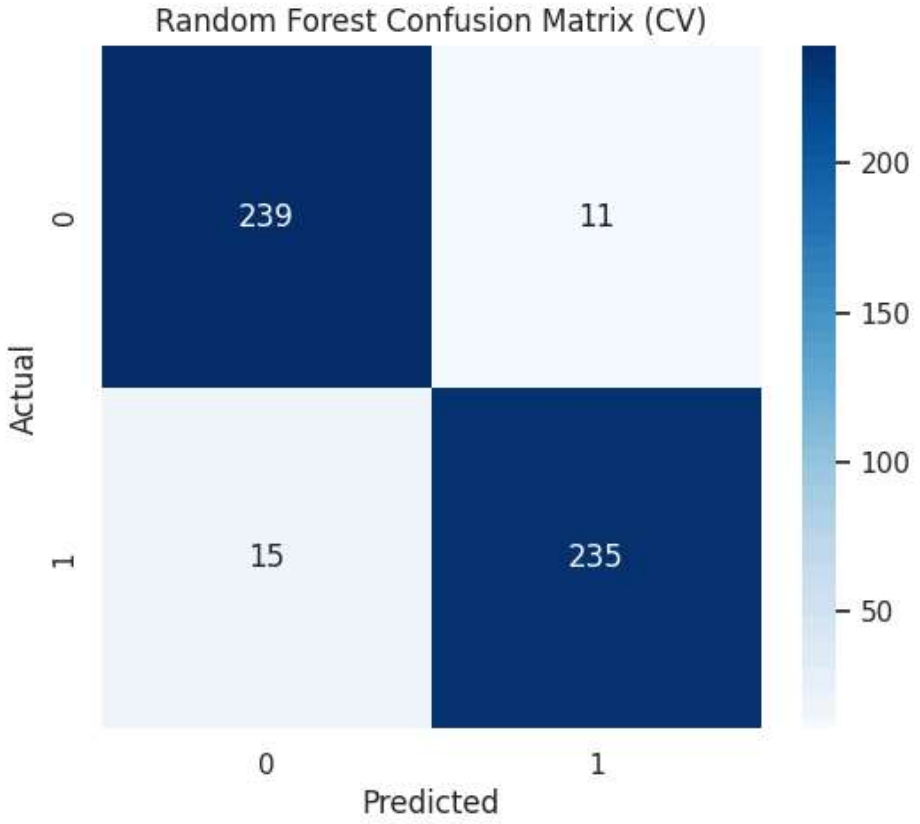
```
        prec = precision_score(y_test, y_pred)
        rec = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        accuracy_list.append(acc)
        precision_list.append(prec)
        recall_list.append(rec)
        f1_list.append(f1)

        print(f"===== Fold {fold} =====")
        print(f"Accuracy: {acc:.4f}, Precision: {prec:.4f}, Recall: {rec:.4f}, F1-Score: {f1:.4f}")

    # Average performance across folds
    print("\n===== Average Performance Across Folds =====")
    print(f"Accuracy: {np.mean(accuracy_list):.4f}")
    print(f"Precision: {np.mean(precision_list):.4f}")
    print(f"Recall: {np.mean(recall_list):.4f}")
    print(f"F1-Score: {np.mean(f1_list):.4f}")

    # Confusion Matrix
    cm = confusion_matrix(y_true_all, y_pred_all)
    plt.figure(figsize=(6,5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=[0,1], yticklabels=[0,1])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Random Forest Confusion Matrix (CV)')
    plt.show()

    # ROC Curve
    fpr, tpr, thresholds = roc_curve(y_true_all, y_prob_all)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8,6))
    plt.plot(fpr, tpr, color='green', label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='red', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Random Forest ROC Curve (CV)')
    plt.legend(loc='lower right')
    plt.show()

    # ----------------
    # Step 2: Grid Search for best hyperparameters
    # ----------------------------

    param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [None, 5, 10],
        'min_samples_split': [2, 5],
        'min_samples_leaf': [1, 2],
        'max_features': ['sqrt', 'log2']
    }

    grid_search = GridSearchCV(
        estimator=RandomForestClassifier(random_state=42),
        param_grid=param_grid,
        scoring='roc_auc',
        cv=5,
```

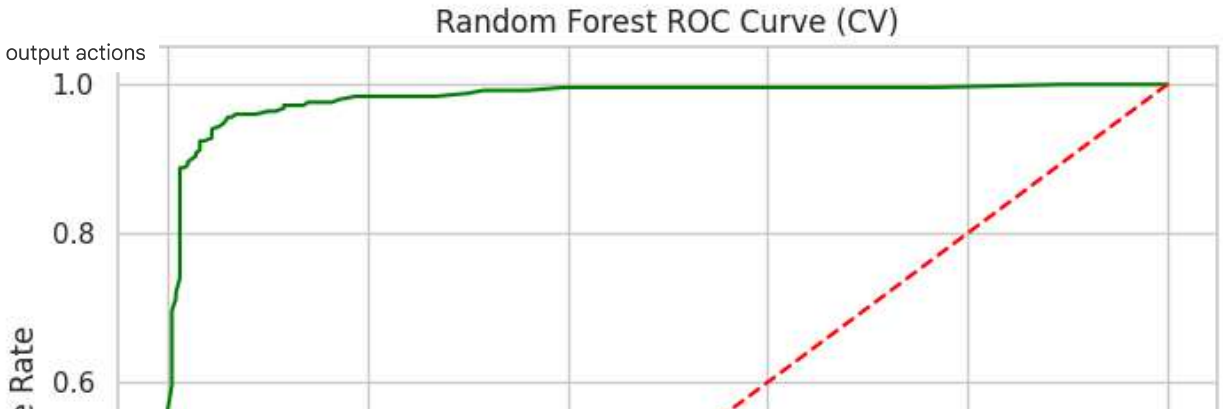
Code cell output actions

```
        n_jobs=-1,
        verbose=1
    )

    grid_search.fit(X, y)

    print("\n===== Best Grid Search Parameters =====")
    print(grid_search.best_params_)

    # Fit the best model
    best_model = grid_search.best_estimator_
    y_pred_best = best_model.predict(X)
    y_prob_best = best_model.predict_proba(X)[:, 1]

    # Final metrics after Grid Search
    print("\n===== Performance of Best Model =====")
    print(classification_report(y, y_pred_best))

    # Confusion Matrix
    cm_best = confusion_matrix(y, y_pred_best)
    plt.figure(figsize=(6,5))
    sns.heatmap(cm_best, annot=True, fmt='d', cmap='Greens', xticklabels=[0,1], yticklabels=[0,1])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Random Forest Confusion Matrix (Best Model)')
    plt.show()

    # ROC Curve
    fpr_best, tpr_best, thresholds_best = roc_curve(y, y_prob_best)
    roc_auc_best = auc(fpr_best, tpr_best)

    plt.figure(figsize=(8,6))
    plt.plot(fpr_best, tpr_best, color='blue', label=f'ROC curve (AUC = {roc_auc_best:.2f})')
    plt.plot([0, 1], [0, 1], color='red', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Random Forest ROC Curve (Best Model)')
    plt.legend(loc='lower right')
    plt.show()
```

Code cell output actions

```
===== Fold 1 =====
Accuracy: 0.9300, Precision: 0.9057, Recall: 0.9600, F1-Score: 0.9320
===== Fold 2 =====
Accuracy: 0.9600, Precision: 0.9600, Recall: 0.9600, F1-Score: 0.9600
===== Fold 3 =====
Accuracy: 0.9500, Precision: 0.9787, Recall: 0.9200, F1-Score: 0.9485
===== Fold 4 =====
Accuracy: 0.9700, Precision: 0.9608, Recall: 0.9800, F1-Score: 0.9703
===== Fold 5 =====
Accuracy: 0.9300, Precision: 0.9778, Recall: 0.8800, F1-Score: 0.9263

===== Average Performance Across Folds =====
Accuracy: 0.9480
Precision: 0.9566
Recall: 0.9400
F1-Score: 0.9474
```





Code cell output actions

```
# Install required packages
!pip install scikit-learn seaborn --quiet

import numpy as np
import pandas as pd
```

```
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, r
from sklearn.ensemble import StackingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import seaborn as sns

# Example dataset (replace with your own diabetes dataset)
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=500, n_features=10, n_classes=2, random_state=42)

# ------------------------------
# Step 1: 5-fold Cross-validation
# ------------------------------

# Base models
estimators = [
    ('svm', SVC(probability=True, random_state=42)),
    ('nb', GaussianNB()),
    ('dt', DecisionTreeClassifier(random_state=42))
]

# Stacking classifier
stack_model = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression(),
    cv=5,
    n_jobs=-1
)

# Prepare for 5-fold CV
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Initialize performance lists
accuracy_list = []
precision_list = []
recall_list = []
f1_list = []
```

Code cell output actions

```
# Store all true and predicted labels for final metrics
y_true_all = []
y_pred_all = []
y_prob_all = []

for fold, (train_idx, test_idx) in enumerate(kf.split(X, y), 1):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # Fit the stacking model
    stack_model.fit(X_train, y_train)

    # Predictions
    y_pred = stack_model.predict(X_test)
    y_prob = stack_model.predict_proba(X_test)[:, 1]
```

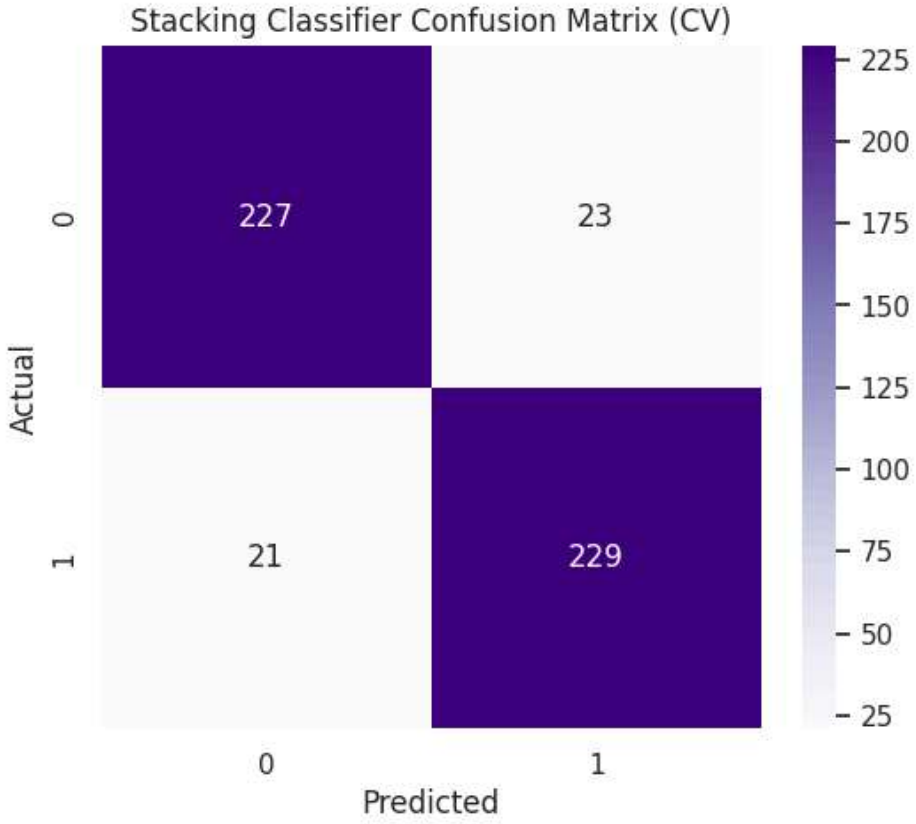
```
        # Store results
        y_true_all.extend(y_test)
        y_pred_all.extend(y_pred)
        y_prob_all.extend(y_prob)

        # Compute metrics for this fold
        acc = accuracy_score(y_test, y_pred)
        prec = precision_score(y_test, y_pred)
        rec = recall_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        accuracy_list.append(acc)
        precision_list.append(prec)
        recall_list.append(rec)
        f1_list.append(f1)

        print(f"===== Fold {fold} =====")
        print(f"Accuracy: {acc:.4f}, Precision: {prec:.4f}, Recall: {rec:.4f}, F1-Score: {f1:.4f}")

    # Average performance across folds
    print("\n===== Average Performance Across Folds =====")
    print(f"Accuracy: {np.mean(accuracy_list):.4f}")
    print(f"Precision: {np.mean(precision_list):.4f}")
    print(f"Recall: {np.mean(recall_list):.4f}")
    print(f"F1-Score: {np.mean(f1_list):.4f}")

    # Confusion Matrix
    cm = confusion_matrix(y_true_all, y_pred_all)
    plt.figure(figsize=(6,5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Purples', xticklabels=[0,1], yticklabels=[0,1])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Stacking Classifier Confusion Matrix (CV)')
    plt.show()

    # ROC Curve
    fpr, tpr, thresholds = roc_curve(y_true_all, y_prob_all)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8,6))
    plt.plot(fpr, tpr, color='purple', label=f'ROC curve (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='red', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Stacking Classifier ROC Curve (CV)')
    plt.legend(loc='lower right')
    plt.show()

    # ------------------------------
    # Step 2: Grid Search for hyperparameters
    # ------------------------------

    param_grid = {
        'dt__max_depth': [None, 3, 5, 10],
        'dt__min_samples_split': [2, 5],
        'svm__C': [0.1, 1, 10],
        'final_estimator__C': [0.1, 1, 10]
    }
```

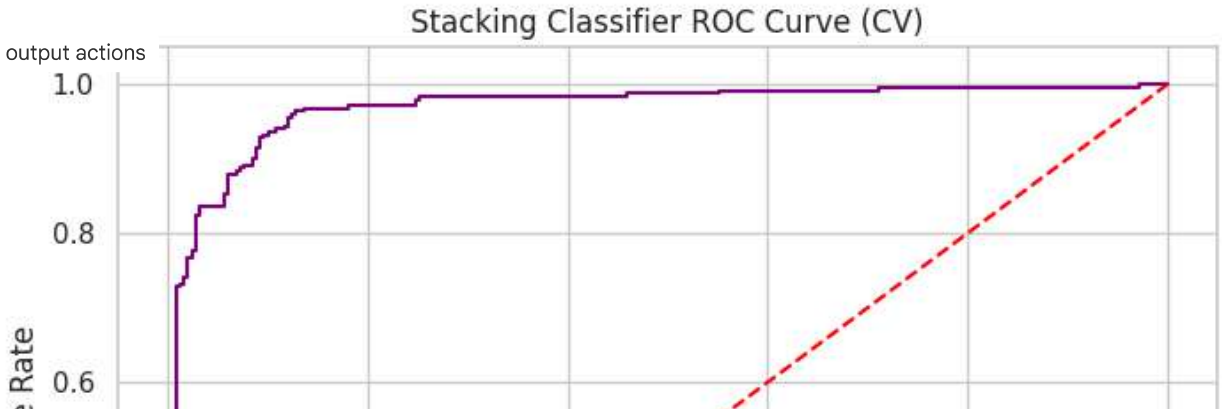
Code cell output actions

```
grid_search = GridSearchCV(
    estimator=stack_model,
    param_grid=param_grid,
    scoring='roc_auc',
    cv=5,
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X, y)

print("\n===== Best Grid Search Parameters =====")
print(grid_search.best_params_)

# Fit the best stacking model
best_stack_model = grid_search.best_estimator_
y_pred_best = best_stack_model.predict(X)
y_prob_best = best_stack_model.predict_proba(X)[:, 1]

# Final metrics after Grid Search
print("\n===== Performance of Best Stacking Model =====")
print(classification_report(y, y_pred_best))

# Confusion Matrix
cm_best = confusion_matrix(y, y_pred_best)
plt.figure(figsize=(6,5))
sns.heatmap(cm_best, annot=True, fmt='d', cmap='Purples', xticklabels=[0,1], yticklabels=[0,1])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Stacking Classifier Confusion Matrix (Best Model)')
plt.show()

# ROC Curve
fpr_best, tpr_best, thresholds_best = roc_curve(y, y_prob_best)
roc_auc_best = auc(fpr_best, tpr_best)

plt.figure(figsize=(8,6))
plt.plot(fpr_best, tpr_best, color='blue', label=f'ROC curve (AUC = {roc_auc_best:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Stacking Classifier ROC Curve (Best Model)')
plt.legend(loc='lower right')
plt.show()
```

Code cell output actions

```
===== Fold 1 =====
Accuracy: 0.9300, Precision: 0.9057, Recall: 0.9600, F1-Score: 0.9320
===== Fold 2 =====
Accuracy: 0.8900, Precision: 0.9149, Recall: 0.8600, F1-Score: 0.8866
===== Fold 3 =====
Accuracy: 0.9200, Precision: 0.9375, Recall: 0.9000, F1-Score: 0.9184
===== Fold 4 =====
Accuracy: 0.9100, Precision: 0.8868, Recall: 0.9400, F1-Score: 0.9126
===== Fold 5 =====
Accuracy: 0.9100, Precision: 0.9020, Recall: 0.9200, F1-Score: 0.9109

===== Average Performance Across Folds =====
Accuracy: 0.9120
Precision: 0.9094
Recall: 0.9160
F1-Score: 0.9121
```



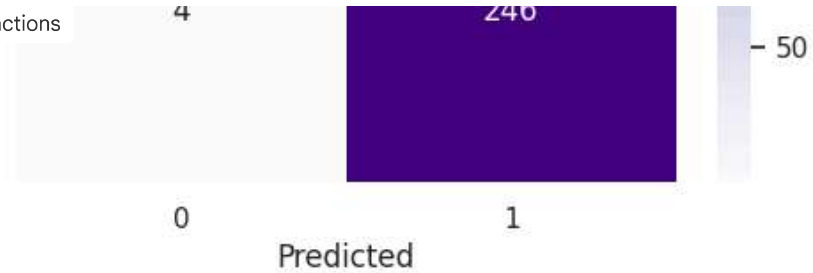


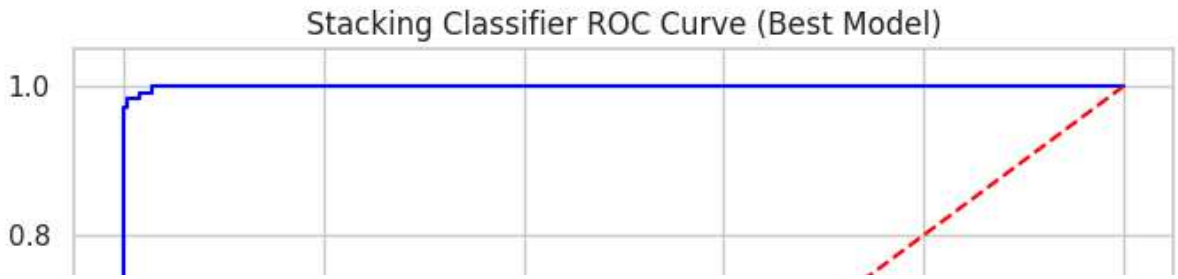
Code cell output actions

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

# Base models
```

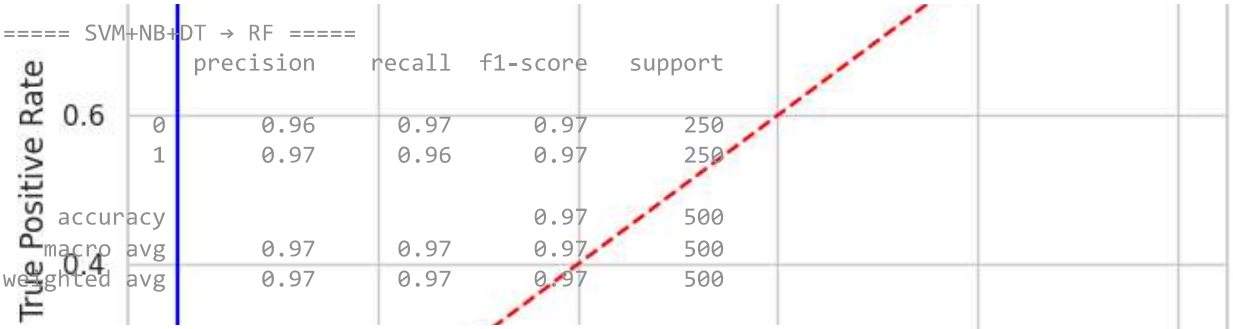
```
estimators_rf = [
    ('svm', SVC(probability=True, random_state=42)),
    ('nb', GaussianNB()),
    ('dt', DecisionTreeClassifier(random_state=42))
]

# Stacking classifier with Random Forest as final estimator
stack_rf_model = StackingClassifier(
    estimators=estimators_rf,
    final_estimator=RandomForestClassifier(n_estimators=100, random_state=42),
    cv=5,
    n_jobs=-1
)

# Fit and evaluate
stack_rf_model.fit(X, y)
y_pred_rf = stack_rf_model.predict(X)
y_prob_rf = stack_rf_model.predict_proba(X)[:, 1]

print("===== SVM+NB+DT → RF =====")
print(classification_report(y, y_pred_rf))

# Confusion Matrix
cm_rf = confusion_matrix(y, y_pred_rf)
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Oranges', xticklabels=[0,1], yticklabels=[0,1])
plt.title('Stacking SVM+NB+DT → RF Confusion Matrix')
plt.show()

# ROC Curve
fpr_rf, tpr_rf, _ = roc_curve(y, y_prob_rf)
roc_auc_rf = auc(fpr_rf, tpr_rf)

plt.figure(figsize=(8,6))
plt.plot(fpr_rf, tpr_rf, color='orange', label=f'ROC curve (AUC = {roc_auc_rf:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.title('Stacking SVM+NB+DT → RF ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```

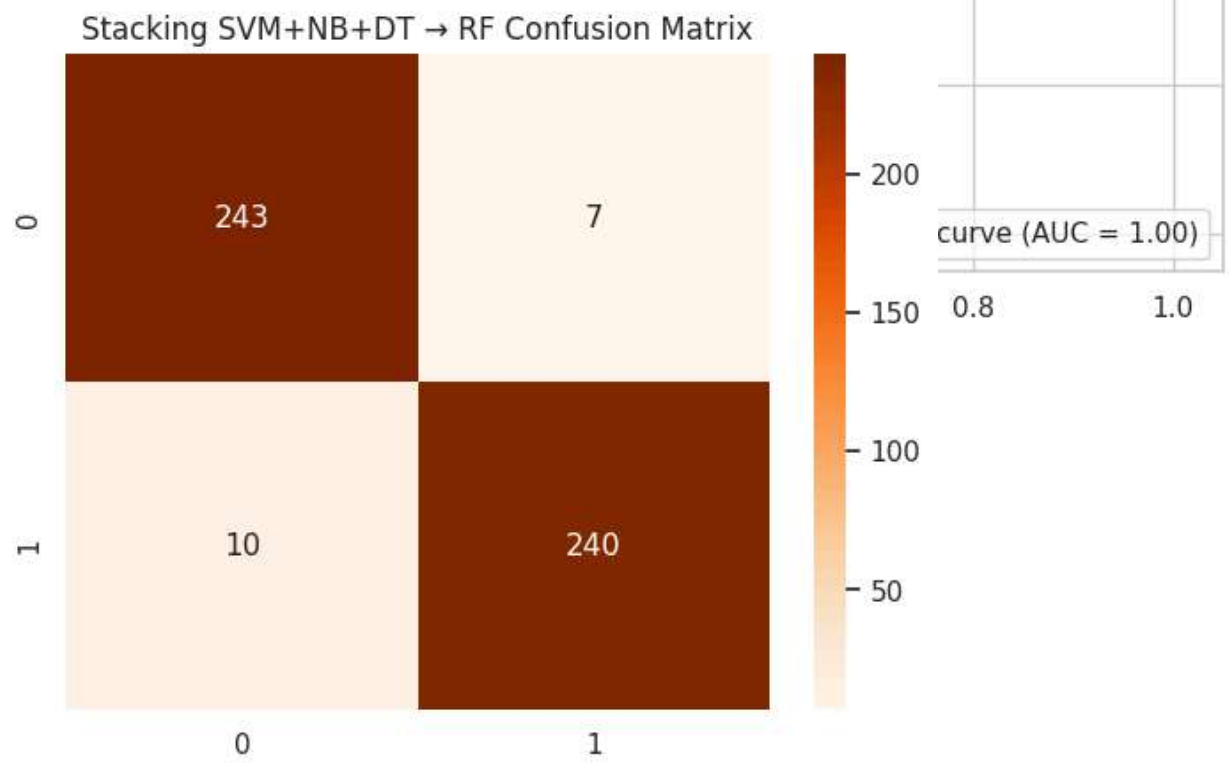
Code cell output actions





```
===== SVM+NB+DT → RF =====
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       250
           1       0.97      0.96      0.97       250

    accuracy                           0.97       500
   macro avg       0.97      0.97      0.97       500
weighted avg       0.97      0.97      0.97       500
```






```
# Standardize features for SVM & KNN
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Base models
```

Code cell output actions

```
    ('svm', SVC(probability=True, random_state=42)),
    ('dt', DecisionTreeClassifier(random_state=42)),
    ('knn', KNeighborsClassifier())
]

# Stacking classifier with Logistic Regression as final estimator
stack_lr_model = StackingClassifier(
```