Celebal Assignment 6

1. Importing Libraries

Imports necessary libraries for data handling, model training, evaluation, and visualization.

```
# Importing necessary libraries for data handling and model building
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import (
accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, ConfusionMatrixDisplay,
precision_recall_curve
)
```

2. Data Loading and Preparation

- Loads the breast cancer dataset from scikit-learn.
- X contains feature values, y contains target values (0 = malignant, 1 = benign).

```
# Step 1: Load the dataset
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)  # Features
y = pd.Series(data.target)  # Target (0 = malignant, 1 = benign)
```

3. Train-Test Split

Splits data into 80% training and 20% testing.

```
# Step 2: Split the dataset into training and test 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)
```

4. Feature Scaling

Standardizes features to improve model performance, especially for SVM and Logistic Regression.

```
# Step 3: Feature scaling (Standardize features to mean=0, std=1)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # Fit and transform training data
X_test_scaled = scaler.transform(X_test)        # Transform test data with same scaling
```

5. Model Definitions

Three baseline models are defined:

- Logistic Regression
- Random Forest
- Support Vector Classifier

```
# Step 4: Define the machine learning models to train
models = {"LogisticRegression": LogisticRegression(),
"RandomForest": RandomForestClassifier(),
"SVC": SVC(probability=True)  # Enable probability for plotting precision-recall curve
}
```

6. Evaluation Function

A reusable function to evaluate models using key classification metrics.

```
# Step 5: Define a reusable function to evaluate models
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)  # Predict the test set
    return {
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1 Score": f1_score(y_test, y_pred),
        "y_pred": y_pred  # Include for confusion matrix visualization
    }
```

7. Baseline Model Training and Evaluation

- Each model is trained (with scaling applied appropriately).
- Metrics are calculated and printed for comparison.

```
# Step 6: Train baseline models and evaluate them
print("=== Baseline Model Evaluation ===")
baseline_results = {}

for name, model in models.items():
    if name in ["LogisticRegression", "SVC"]:
        model.fit(X_train_scaled, y_train)  # Scaled data for models sensitive to feature scale
        result = evaluate_model(model, X_test_scaled, y_test)
    else:
        model.fit(X_train, y_train)  # Tree-based models do not require scaling
        result = evaluate_model(model, X_test, y_test)

    baseline_results[name] = result
    print(f"\n{name}:\n", result)
```

```
=== Baseline Model Evaluation ===

LogisticRegression:
 {'Accuracy': 0.9736842105263158, 'Precision': 0.9722222222222222, 'Recall': 0.9859154929577465, 'F1 Score': 0.9790209790209791, 'y_pred': array(
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0])}

RandomForest:
 {'Accuracy': 0.9649122807017544, 'Precision': 0.958904109589041, 'Recall': 0.9859154929577465, 'F1 Score': 0.9722222222222222, 'y_pred': array([:
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0])}

SVC:
 {'Accuracy': 0.9824561403508771, 'Precision': 0.9726027397260274, 'Recall': 1.0, 'F1 Score': 0.9861111111111112, 'y_pred': array([1, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0])}
```

8. Hyperparameter Tuning

- **Logistic Regression** using GridSearchCV

```
# Step 7: Hyperparameter Tuning

# Logistic Regression - GridSearchCV (tries all combinations)
log_reg_params = {
'C': [0.01, 0.1, 1, 10],          # Regularization strength
'penalty': ['l2'],               # Regularization type
'solver': ['liblinear']          # Solver that supports 'l2'
}
log_grid = GridSearchCV(LogisticRegression(), log_reg_params, cv=5)
log_grid.fit(X_train_scaled, y_train)  # Use scaled data
```

- **Random Forest** using RandomizedSearchCV

```
# Random Forest - RandomizedSearchCV (tries random combinations for faster tuning)
rf_params = {
'n_estimators': [10, 50, 100, 200],       # Number of trees
'max_depth': [None, 10, 20, 30],          # Maximum depth of tree
'min_samples_split': [2, 5, 10]           # Min samples to split a node
}
rf_rand = RandomizedSearchCV(RandomForestClassifier(), rf_params, n_iter=10, cv=5, random_state=42)
rf_rand.fit(X_train, y_train)  # No need to scale for Random Forest
```

- **Support Vector Classifier** using GridSearchCV

```
# Support Vector Classifier - GridSearchCV
svc_params = {
'C': [0.1, 1, 10],                  # Regularization parameter
'kernel': ['linear', 'rbf'],        # Kernel type
'gamma': ['scale', 'auto']          # Kernel coefficient
}
svc_grid = GridSearchCV(SVC(), svc_params, cv=5)
svc_grid.fit(X_train_scaled, y_train)  # Use scaled data
```

Each tuned model is fit using appropriate training data.

9. Evaluation of Tuned Models

Evaluates best estimators from each tuning strategy.

```
# Step 8: Evaluate tuned models
print("\n=== Tuned Model Evaluation ===")
tuned_results = {
"LogisticRegression": evaluate_model(log_grid.best_estimator_, X_test_scaled, y_test),
"RandomForest": evaluate_model(rf_rand.best_estimator_, X_test, y_test),
"SVC": evaluate_model(svc_grid.best_estimator_, X_test_scaled, y_test)
}
```

10. Best Model Selection

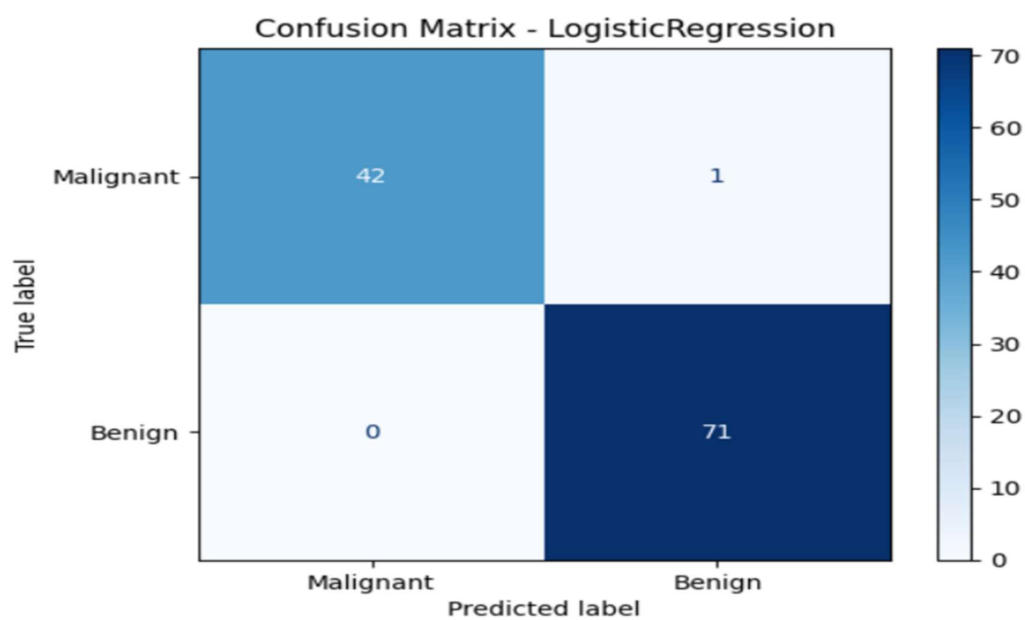
The best model is selected based on **F1 Score**, which balances precision and recall.

```
# Step 9: Select best model based on F1 Score
best_model_name, best_result = max(tuned_results.items(), key=lambda x: x[1]['F1 Score'])
print("\nBest Model Based on F1 Score:\n", best_model_name)
```

11. Confusion Matrix Visualization

Plots confusion matrix for each model to visualize performance.

```
# Step 10: Confusion Matrices for each model
for name, result in tuned_results.items():
    cm = confusion_matrix(y_test, result["y_pred"])
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Malignant", "Benign"])
    disp.plot(cmap='Blues')
    plt.title(f"Confusion Matrix - {name}")
    plt.show()
```

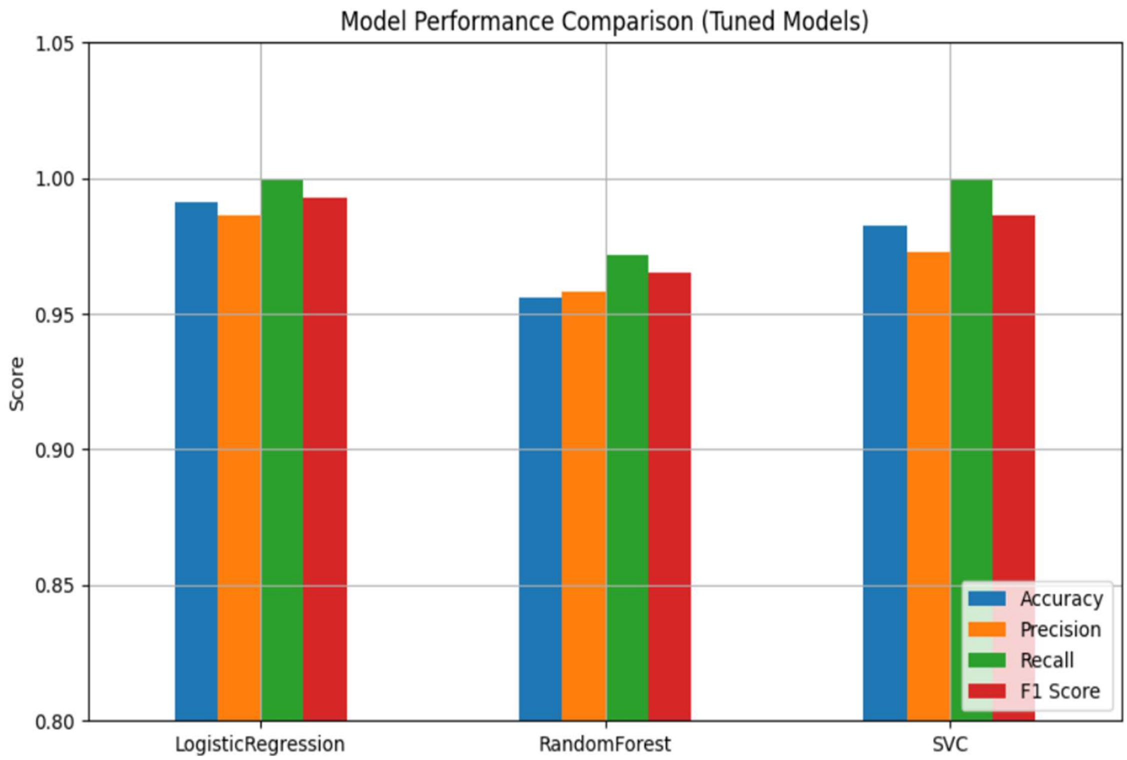


12. Bar Chart Comparison

Displays a grouped bar chart of Accuracy, Precision, Recall, and F1 Score for each tuned model.

```
# Step 11:  Bar chart of Accuracy, Precision, Recall, F1
metrics_df = pd.DataFrame({
    model: {
        k: v for k, v in result.items() if k in ['Accuracy', 'Precision', 'Recall', 'F1 Score']
    } for model, result in tuned_results.items()
}).T

metrics_df.plot(kind='bar', figsize=(10, 6))
plt.title("Model Performance Comparison (Tuned Models)")
plt.ylabel("Score")
plt.ylim(0.8, 1.05)
plt.xticks(rotation=0)
plt.grid(True)
plt.legend(loc='lower right')
plt.show()
```



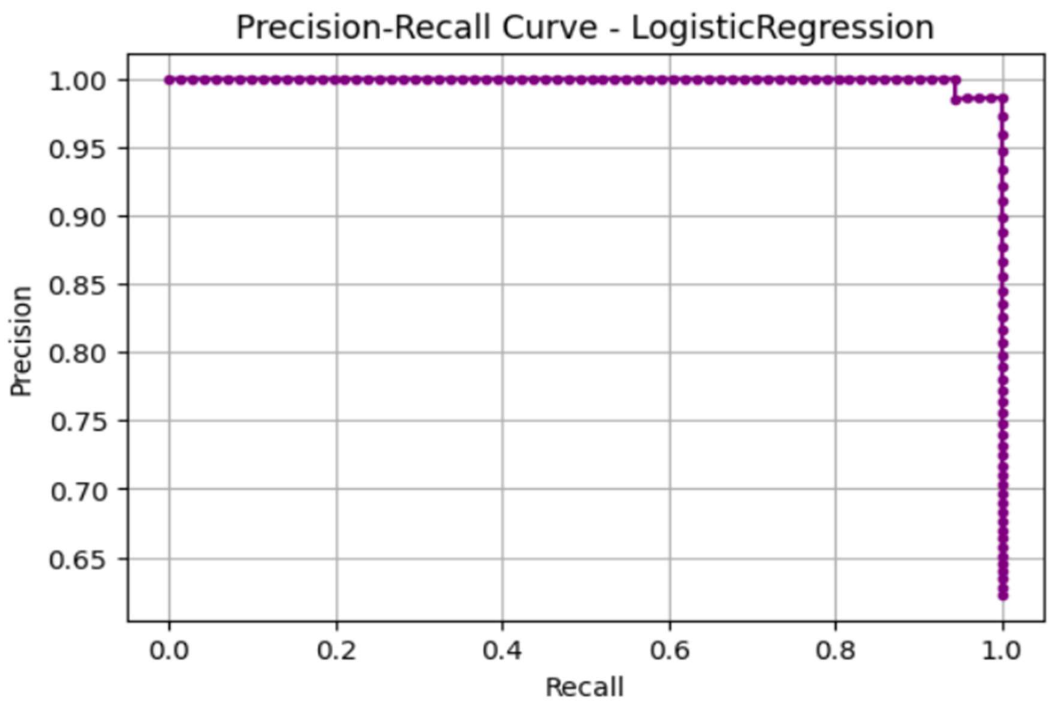
13. Precision-Recall Curve for Best Model

Visualizes the precision-recall tradeoff for the best-performing model.

```python
# Step 12:  Precision-Recall Curve for the best model
if best_model_name == "RandomForest":
    model = rf_rand.best_estimator_
    X_used = X_test
elif best_model_name == "LogisticRegression":
    model = log_grid.best_estimator_
    X_used = X_test_scaled
else:
    model = svc_grid.best_estimator_
    X_used = X_test_scaled

y_scores = model.predict_proba(X_used)[:, 1]
precision, recall, _ = precision_recall_curve(y_test, y_scores)

plt.figure(figsize=(6, 4))
plt.plot(recall, precision, marker='.', color='purple')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title(f'Precision-Recall Curve - {best_model_name}')
plt.grid(True)
plt.show()
```



Devanshi Mittal