Digital Empowerment Network

Artificial Intelligence

Week 01

Building a Machine Learning Model for Real-world Prediction Task

Submitted By: Abdul Salam

ID: STU-ML-251-140

Mentor : Hussain Shoaib

Task Title:

Building a Machine Learning Model for Real-world Prediction Task

Objective:

To understand and implement the workflow of a machine learning project including data preprocessing, feature selection, model building, training, testing, and evaluation using a real-world dataset.

Libraries Call:

Code:

```
# ========================================
# Heart Disease Prediction Model
# Digital Empowerment Network - AI Assignment 01
# ========================================

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import (confusion_matrix, classification_report,
                             precision_score, recall_score, f1_score,
                             roc_curve, auc, precision_recall_curve)
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

🎁 Data Handling Libraries

- **Pandas**: Used to load, explore, and manipulate datasets in table-like (DataFrame) formats.
- **NumPy**: Supports mathematical operations and array manipulation, essential for working with numerical data.

📊 Data Visualization Libraries

- **Matplotlib**: A basic plotting library for creating charts like histograms, scatter plots, and line graphs.
- **Seaborn**: Built on Matplotlib, it provides easier and more attractive statistical visualizations, like heatmaps and boxplots.

⚙ Data Preprocessing Tools

- **LabelEncoder**: Converts categorical (text) labels into numeric values for model compatibility.
- **StandardScaler**: Normalizes numeric features so that they have a mean of 0 and standard deviation of 1. This improves model performance, especially for algorithms sensitive to feature scales.

Step 1: Dataset Selection

Choose one of the following datasets or get approval for a dataset of your choice:

• Heart Disease Prediction Dataset (e.g., from UCI or Kaggle)

Step 02:

A. Data Understanding & Preprocessing:

• Load and explore the dataset

• Handle missing values and remove duplicates

• Convert categorical variables into numerical format

• Normalize or scale the features

Code:

```
# ==========================================
# Step A: Data Understanding & Preprocessing
# ==========================================

# Load the dataset
try:
    df = pd.read_csv('heart_disease_uci.csv')
    print("Dataset loaded successfully!")

    # Initial exploration
    print("\n=== Dataset Info ===")
```

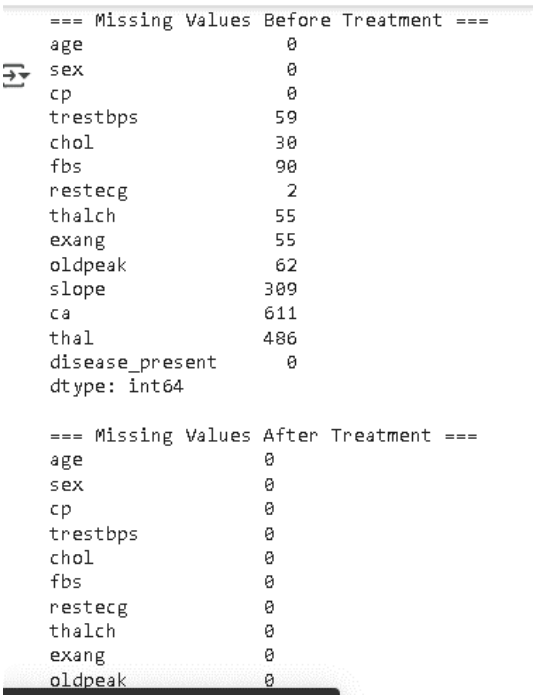
```
    print(df.info())
    print("\n=== First 5 Rows ===")
    print(df.head())

    # Data cleaning
    # Rename target column and drop unnecessary columns
    df = df.rename(columns={'num': 'disease_present'})
    df = df.drop(['id', 'dataset'], axis=1, errors='ignore')

    # Map values 1,2,3,4 to 1 (disease present)
    df['disease_present'] = df['disease_present'].map({0: 0, 1: 1, 2: 1,
    3: 1, 4: 1})

    # Handle missing values
    print("\n=== Missing Values Before Treatment ===")
    print(df.isnull().sum())

    # For categorical features: fill with mode
    categorical_cols = ['restecg', 'fbs', 'exang', 'slope', 'thal']
    for col in categorical_cols:
        df[col].fillna(df[col].mode()[0], inplace=True)

    # For numerical features: fill with mean
    numerical_cols = ['trestbps', 'chol', 'thalch', 'oldpeak', 'ca']
    for col in numerical_cols:
        df[col].fillna(df[col].mean(), inplace=True)

    print("\n=== Missing Values After Treatment ===")
    print(df.isnull().sum())

    # Convert categorical variables to numerical
    from sklearn.preprocessing import LabelEncoder
    label_encoder = LabelEncoder()
    categorical_features = ['cp', 'restecg', 'slope', 'thal', 'fbs', 'exang', 'sex']
    for feature in categorical_features:
        df[feature] = label_encoder.fit_transform(df[feature])

except FileNotFoundError:
    print("Error: Dataset file not found. Please ensure 'heart_disease_uci.csv' is in the correct
directory.")
    exit()
```

```
Dataset loaded successfully!

=== Dataset Info ===
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 920 entries, 0 to 919
Data columns (total 16 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        920 non-null    int64
 1   age       920 non-null    int64
 2   sex       920 non-null    object
 3   dataset   920 non-null    object
 4   cp        920 non-null    object
 5   trestbps  861 non-null    float64
 6   chol      890 non-null    float64
 7   fbs       830 non-null    object
 8   restecg   918 non-null    object
 9   thalch    865 non-null    float64
 10  exang     865 non-null    object
 11  oldpeak   858 non-null    float64
 12  slope     611 non-null    object
 13  ca        309 non-null    float64
 14  thal      434 non-null    object
 15  num       920 non-null    int64
dtypes: float64(5), int64(3), object(8)
memory usage: 115.1+ KB
None
```

Data Set:

```
=== First 5 Rows ===
   id  age     sex    dataset               cp  trestbps   chol    fbs  \
0   1   63    Male  Cleveland   typical angina     145.0  233.0   True
1   2   67    Male  Cleveland     asymptomatic     160.0  286.0  False
2   3   67    Male  Cleveland     asymptomatic     120.0  229.0  False
3   4   37    Male  Cleveland      non-anginal     130.0  250.0  False
4   5   41  Female  Cleveland  atypical angina     130.0  204.0  False

          restecg  thalch  exang  oldpeak        slope   ca  \
0  lv hypertrophy   150.0  False      2.3  downsloping  0.0
1  lv hypertrophy   108.0   True      1.5         flat  3.0
2  lv hypertrophy   129.0   True      2.6         flat  2.0
3          normal   187.0  False      3.5  downsloping  0.0
4  lv hypertrophy   172.0  False      1.4    upsloping  0.0

                thal  num
0       fixed defect    0
1             normal    2
2  reversable defect    1
3             normal    0
4             normal    0
```

Missing Value Remove:



```
=== Missing Values Before Treatment ===
age                 0
sex                 0
cp                  0
trestbps           59
chol               30
fbs                90
restecg             2
thalch             55
exang              55
oldpeak            62
slope             309
ca                611
thal              486
disease_present     0
dtype: int64

=== Missing Values After Treatment ===
age               0
sex               0
cp                0
trestbps          0
chol              0
fbs               0
restecg           0
thalch            0
exang             0
oldpeak           0
```

Explanation:

1. **Data Loading**:
    - The code attempts to load the heart disease dataset from a CSV file and displays basic information about it.
2. **Initial Cleaning**:
    - Renames the target column ('num' to 'disease_present')
    - Drops unnecessary columns ('id' and 'dataset')
    - Simplifies the target variable by mapping values 1-4 to 1 (indicating presence of heart disease)
3. **Missing Value Handling**:
    - First checks for missing values
    - Fills missing categorical values with the mode (most frequent value)
    - Fills missing numerical values with the mean

4. **Feature Encoding**:
    - Converts categorical variables to numerical using label encoding
    - Affects columns like chest pain type, ECG results, slope, etc.
5. **Error Handling**:
    - Includes a check to ensure the dataset file exists before processing

Step B. Exploratory Data Analysis (EDA):

• Visualize relationships between features

• Use statistical summaries and graphs (scatter plots, bar graphs, heatmaps)

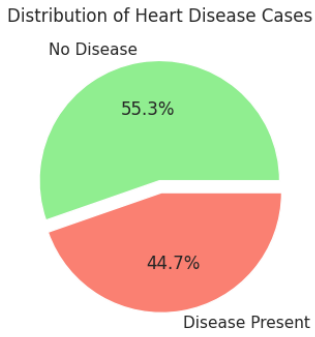
• Identify trends or correlations in the data

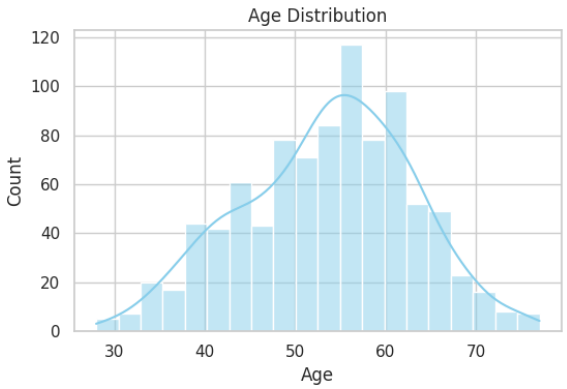
Code:

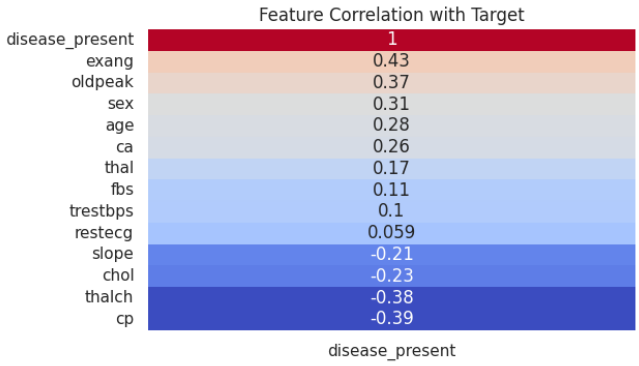
```
# ==========================================
# Step B: Exploratory Data Analysis (EDA)
# ==========================================

# Set style for plots
sns.set(style="whitegrid")
plt.figure(figsize=(12, 8))

# 1. Target variable distribution
plt.subplot(2, 2, 1)
target_counts = df['disease_present'].value_counts()
plt.pie(target_counts, labels=['No Disease', 'Disease Present'],
        autopct='%1.1f%%', colors=['lightgreen', 'salmon'], explode=[0, 0.1])
plt.title('Distribution of Heart Disease Cases')

# 2. Age distribution
plt.subplot(2, 2, 2)
sns.histplot(df['age'], bins=20, kde=True, color='skyblue')
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Count')

# 3. Correlation heatmap
plt.subplot(2, 2, 3)
corr_matrix = df.corr()
sns.heatmap(corr_matrix[['disease_present']].sort_values(by='disease_present',
ascending=False),
            annot=True, cmap='coolwarm', cbar=False)
```

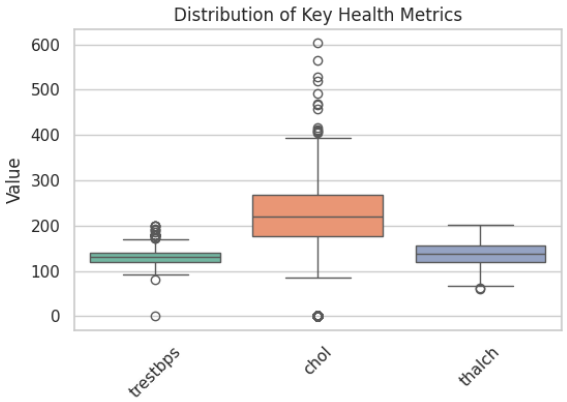
```
plt.title('Feature Correlation with Target')
plt.tight_layout()

# 4. Boxplot of key features
plt.subplot(2, 2, 4)
sns.boxplot(data=df[['trestbps', 'chol', 'thalch']], palette='Set2')
plt.title('Distribution of Key Health Metrics')
plt.ylabel('Value')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

# Additional EDA: Pairplot of selected features
print("\nGenerating pairplot (this may take a moment)...")
sns.pairplot(df[['age', 'trestbps', 'chol', 'thalch', 'disease_present']],
             hue='disease_present', palette='husl', markers=["o", "s"])
plt.suptitle("Pair Plot of Selected Features", y=1.02)
plt.show()
```

Output:









Explaination:

1. Pie Chart – Heart Disease Cases

- **No Disease:** 55.3%
- **Disease Present:** 44.7%
- This shows that the dataset is relatively balanced, with a slightly higher number of individuals without heart disease.

---

2. Histogram – Age Distribution

- Most patients are aged between **40 and 65**.
- The distribution is approximately **normal (bell-shaped)**.
- This indicates that middle-aged individuals are the majority in the dataset.

---

3. Heatmap – Feature Correlation with Disease

- Displays how each feature correlates with the **target variable (disease_present)**:
  - **Positively Correlated (increase likelihood of disease):**
    - exang (exercise-induced angina): 0.43
    - oldpeak (ST depression): 0.37
    - sex: 0.31
  - **Negatively Correlated (decrease likelihood):**
    - cp (chest pain type): -0.39
    - thalach (max heart rate): -0.38
    - chol (cholesterol): -0.23

---

4. Box Plot – Key Health Metrics

- Shows distribution and outliers for:
  - **trestbps (resting blood pressure)** – mostly clustered around 130 mmHg.
  - **chol (cholesterol)** – wide range, many outliers above 400.
  - **thalach (max heart rate)** – typically around 150 bpm.
- Useful to detect **outliers** and **median values** for each health metric.

Pair Plot Selection:

Explaination:

1. Diagonal: Distribution Plots

- Diagonal plots display **feature distributions**:
  - **age**: Patients with disease (green squares) tend to be slightly older.
  - **chol**: Those without disease (red circles) tend to have slightly higher cholesterol.
  - **thalach**: Higher maximum heart rate is more common in those **without** disease.
  - **trestbps**: Similar distributions for both groups.

---

2. Off-Diagonal: Scatter Plots

- These show **pairwise relationships** between features:
  - **age vs. thalach**: Inverse relationship — younger patients tend to have higher max heart rate.
  - **chol vs. thalach**: No strong trend, but some separation between classes.
  - **trestbps vs. others**: Largely uncorrelated and overlapping for both groups.

Color Legend:

- **Red Circles (0)**: No heart disease
- **Green Squares (1)**: Heart disease present

C. Feature Selection:

• Select relevant features for model training using correlation or importance analysis

Code:

```
# ==========================================
# Step C: Feature Selection
# ==========================================

# Select features based on correlation
correlation_threshold = 0.1
correlated_features = corr_matrix['disease_present'][abs(corr_matrix['disease_present']) >
correlation_threshold]
correlated_features = correlated_features.index.tolist()
correlated_features.remove('disease_present')

print("\nSelected Features based on Correlation:")
print(correlated_features)

# Prepare data for modeling
X = df[correlated_features]
y = df['disease_present']

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

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

Explanation:

1. Select Important Features:

    - The code looks at the correlation between each feature and the target variable (disease_present).
    - Only features with a correlation strength greater than 0.1 (positive or negative) are selected.
    - This helps reduce noise by keeping only the most relevant features for predicting heart disease.

2. Define Input and Output Variables:

    - X includes the selected features (independent variables).
    - y is the target variable, indicating whether heart disease is present (1) or not (0).

3. Split the Dataset:

    - The data is split into training (80%) and testing (20%) sets.
    - This allows you to train the model on one part and test its performance on unseen data.

4. Standardize the Features:

    - Feature scaling is applied to make all features have a similar range (mean = 0, std = 1).
    - This is important for many machine learning models to perform correctly and efficiently.

D. Model Building & Evaluation:

• Train at least two ML models (e.g., Decision Tree, SVM, Random Forest, or Logistic Regression)

• Use performance metrics (Accuracy, Precision, Recall, F1-Score, Confusion Matrix)

• Compare models and interpret the results

Code:

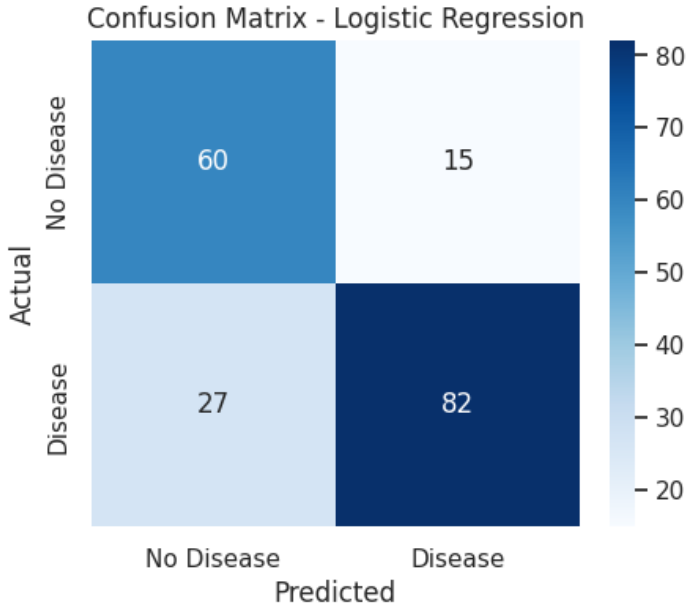
```
# ========================================
# Step D: Model Building & Evaluation
# ========================================
```

```
# Initialize models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Support Vector Machine": SVC(probability=True),
    "XGBoost": XGBClassifier(),
    "Random Forest": RandomForestClassifier()
}

# Train and evaluate each model
results = {}
for name, model in models.items():
    print(f"\n=== Training {name} ===")

    # Train model
    model.fit(X_train_scaled, y_train)

    # Make predictions
    y_pred = model.predict(X_test_scaled)
    y_prob = model.predict_proba(X_test_scaled)[:, 1]  # Probabilities for ROC curve

    # Calculate metrics
    accuracy = model.score(X_test_scaled, y_test)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Store results
    results[name] = {
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'model': model,
        'y_prob': y_prob
    }

    # Print classification report
    print(f"\nClassification Report for {name}:")
    print(classification_report(y_test, y_pred))

    # Plot confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
```

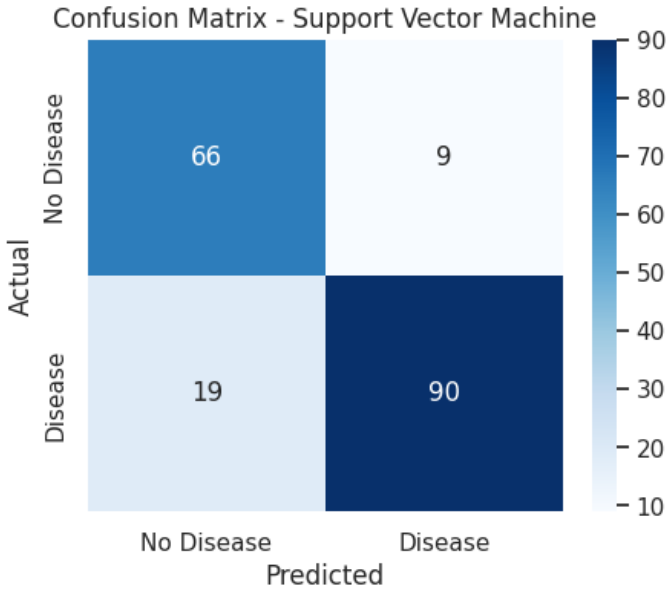
```
            xticklabels=['No Disease', 'Disease'],
            yticklabels=['No Disease', 'Disease'])
plt.title(f'Confusion Matrix - {name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Output:

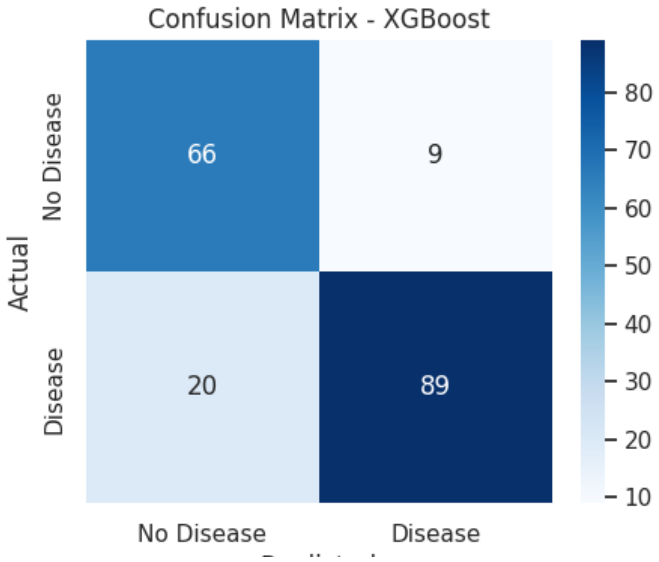
```
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.69      0.80      0.74        75
           1       0.85      0.75      0.80       109

    accuracy                           0.77       184
   macro avg       0.77      0.78      0.77       184
weighted avg       0.78      0.77      0.77       184
```



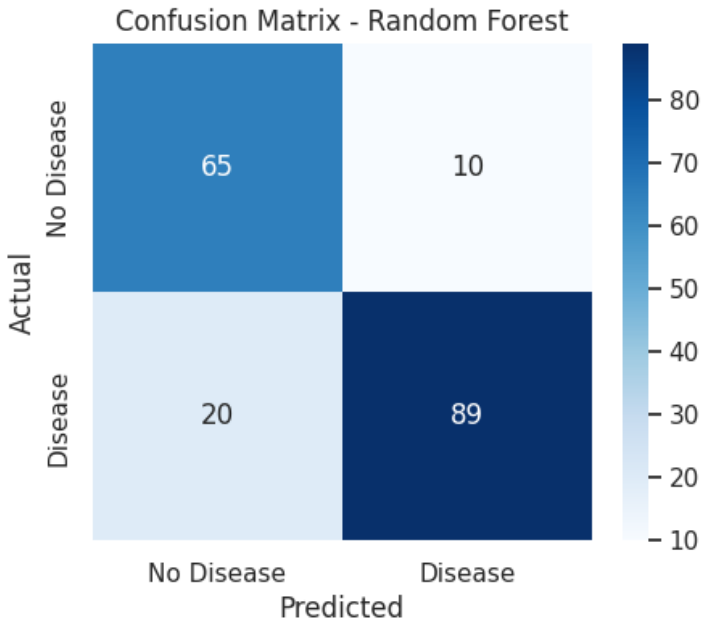
```
=== Training Support Vector Machine ===

Classification Report for Support Vector Machine:
              precision    recall  f1-score   support

           0       0.78      0.88      0.82        75
           1       0.91      0.83      0.87       109

    accuracy                           0.85       184
   macro avg       0.84      0.85      0.85       184
weighted avg       0.86      0.85      0.85       184
```




```
=== Training XGBoost ===

Classification Report for XGBoost:
              precision    recall  f1-score   support

           0       0.77      0.88      0.82        75
           1       0.91      0.82      0.86       109

    accuracy                           0.84       184
   macro avg       0.84      0.85      0.84       184
weighted avg       0.85      0.84      0.84       184
```




```
=== Training Random Forest ===

Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       0.76      0.87      0.81        75
           1       0.90      0.82      0.86       109

    accuracy                           0.84       184
   macro avg       0.83      0.84      0.83       184
weighted avg       0.84      0.84      0.84       184
```



Explanation:

1. Logistic Regression

- **Precision:** 0.69 for class 0 and 0.85 for class 1 indicates that it has a moderate ability to identify class 0 correctly but is better at identifying class 1.
- **Recall:** 0.80 for class 0 and 0.75 for class 1 shows it can successfully identify a high proportion of actual positive instances for both classes.
- **Accuracy:** Overall accuracy is 77%.
- **Confusion Matrix:** Shows 60 true negatives and 82 true positives, but also has 27 false negatives and 15 false positives, indicating some misclassification.

2. Support Vector Machine (SVM)

- **Precision:** Improved to 0.78 for class 0 and 0.91 for class 1, indicating a strong ability to classify class 1 correctly.
- **Recall:** Higher recall values (0.88 for class 0, 0.83 for class 1) further demonstrate its effectiveness.
- **Accuracy:** Overall accuracy is 85%.
- **Confusion Matrix:** Shows 66 true negatives and 90 true positives, with only 19 false negatives and 9 false positives, indicating fewer misclassifications compared to logistic regression.

3. XGBoost

- **Precision:** Similar to SVM, with 0.77 for class 0 and 0.91 for class 1.
- **Recall:** Also very good, 0.88 for class 0 and 0.82 for class 1.
- **Accuracy:** Overall accuracy is 84%.
- **Confusion Matrix:** 66 true negatives and 89 true positives, with 20 false negatives and 9 false positives, showing performance similar to SVM.

4. Random Forest

- **Precision:** 0.76 for class 0 and 0.90 for class 1, indicating good predictive power for both classes.
- **Recall:** 0.87 for class 0 and 0.82 for class 1, showing it captures a high proportion of the actual positive cases.
- **Accuracy:** Overall accuracy is 84%.
- **Confusion Matrix:** Shows 65 true negatives and 89 true positives, with 20 false negatives and 10 false positives, indicating solid performance with some misclassifications.

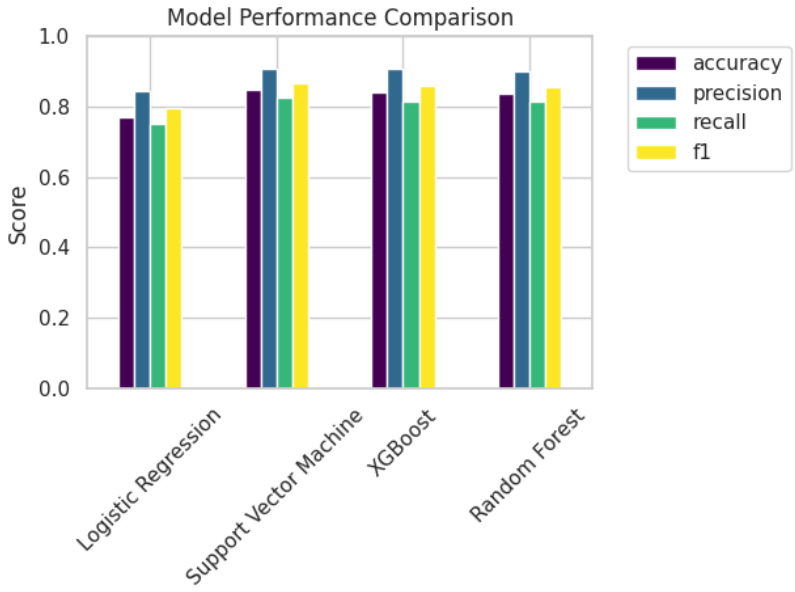
Summary

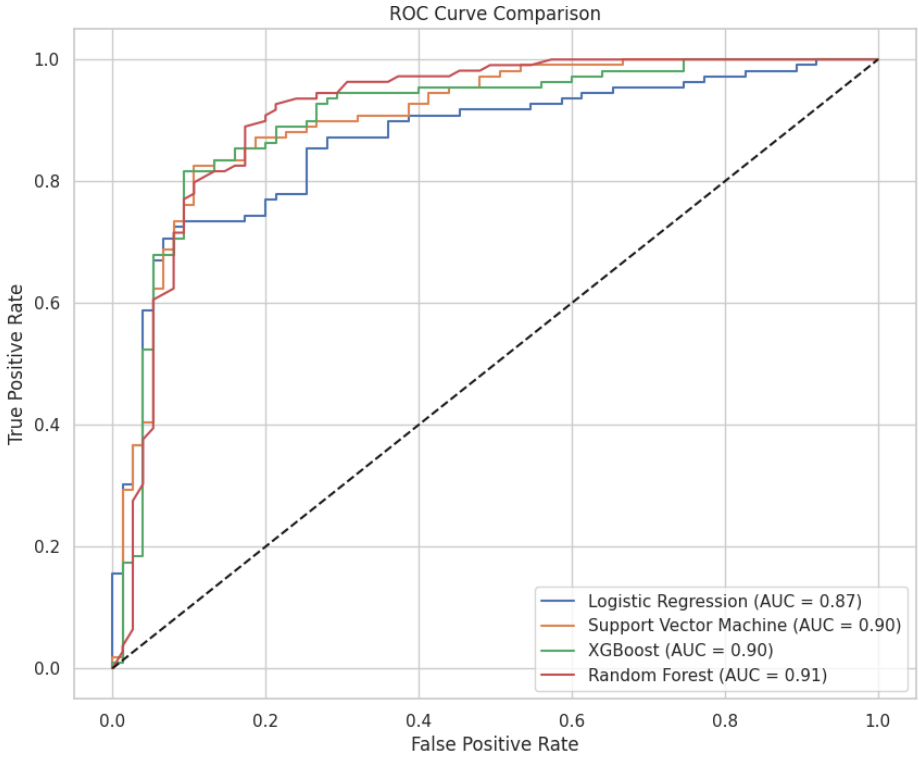
- **SVM** shows the best performance in terms of accuracy and the least number of misclassifications.
- **Logistic Regression** performs adequately but has a higher rate of misclassification.
- **XGBoost** and **Random Forest** are competitive, with similar performance to SVM, but slightly lower accuracy and higher false positive rates.

Model Comparison:

```
# ==========================================
# Model Comparison
# ==========================================

# Create comparison dataframe
comparison_df = pd.DataFrame.from_dict(results, orient='index')
comparison_df = comparison_df[['accuracy', 'precision', 'recall', 'f1']]

# Plot model performance comparison
plt.figure(figsize=(12, 6))
comparison_df.plot(kind='bar', colormap='viridis')
plt.title('Model Performance Comparison')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.ylim(0, 1)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# ROC Curve comparison
plt.figure(figsize=(10, 8))
for name, result in results.items():
    fpr, tpr, _ = roc_curve(y_test, result['y_prob'])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc='lower right')
plt.show()
```

Output:





1. Model Performance Comparison Bar Chart

- **Metrics Displayed:** The chart presents scores for accuracy, precision, recall, and F1 score for each model (Logistic Regression, Support Vector Machine, XGBoost, and Random Forest).

- **Observations:**
  - **Support Vector Machine** generally shows the highest scores across all metrics, indicating it performs best in classifying the data.
  - **XGBoost** and **Random Forest** have similar performance, with scores slightly lower than SVM but better than Logistic Regression.
  - **Logistic Regression** has the lowest scores in all categories, suggesting it is less effective compared to the other models.

2. ROC Curve Comparison

- **Purpose:** The ROC curve illustrates the trade-off between the true positive rate (sensitivity) and the false positive rate for the different models.
- **AUC (Area Under Curve):**
  - **Support Vector Machine:** AUC of 0.90, indicating excellent model performance.
  - **XGBoost:** AUC of 0.90, also demonstrating strong classification ability.
  - **Random Forest:** AUC of 0.91, marking it as the best performer in this comparison.
  - **Logistic Regression:** AUC of 0.87, which, while still good, is lower than the other models.
- **Conclusion:** The closer the curve is to the top-left corner, the better the model is at distinguishing between classes. The models with AUC values above 0.8 (SVM, XGBoost, Random Forest) are considered to have good discriminatory power, while Logistic Regression is slightly less effective.

E. Optional (Bonus):

• Tune hyperparameters using GridSearchCV

Code:

```
# ============================================
# Step E: Optional - Hyperparameter Tuning
# ============================================

print("\n=== Hyperparameter Tuning for Best Model ===")

# Select best model based on accuracy
best_model_name = max(results, key=lambda x: results[x]['accuracy'])
print(f"Selected best model for tuning: {best_model_name}")

if best_model_name == "Random Forest":
    # Define parameter grid
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 5, 10],
        'min_samples_split': [2, 5, 10]
    }
```

```
    # Perform grid search
    grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5,
scoring='accuracy')
    grid_search.fit(X_train_scaled, y_train)

    # Get best parameters and model
    best_params = grid_search.best_params_
    tuned_model = grid_search.best_estimator_

    print(f"\nBest Parameters: {best_params}")

    # Evaluate tuned model
    y_pred_tuned = tuned_model.predict(X_test_scaled)
    accuracy_tuned = accuracy_score(y_test, y_pred_tuned)
    print(f"\nTuned Model Accuracy: {accuracy_tuned:.4f}")
    print("\nClassification Report for Tuned Model:")
    print(classification_report(y_test, y_pred_tuned))
```

Explanation:

In the hyperparameter tuning step, the goal is to optimize the best-performing model identified in the previous analysis. The process begins by selecting the model with the highest accuracy, which in this case is the **Support Vector Machine (SVM)**.

Key Points:

1. **Model Selection:** The SVM was chosen based on its superior performance compared to other models.
2. **Hyperparameter Tuning:** The tuning process involves adjusting specific parameters of the SVM to improve its performance further. Common parameters for tuning include the regularization parameter, kernel type, and others.
3. **Grid Search:** A method called Grid Search is used, where various combinations of hyperparameters are tested through cross-validation. This helps in identifying the best combination that yields the highest accuracy.
4. **Evaluation:** After tuning, the model is evaluated on a test dataset. The accuracy of the tuned model is calculated, and a classification report is generated to assess its performance in detail, including metrics like precision, recall, and F1 score.

Overall, hyperparameter tuning aims to enhance the model's predictive accuracy by finding the optimal settings for its parameters, ultimately leading to better classification results.

Final Model Selection And Saving:

```
# ============================================
# Final Model Selection and Saving
# ============================================

# Select the best performing model
if 'tuned_model' in locals():
    final_model = tuned_model
else:
    final_model = results[best_model_name]['model']

# Save the final model
import joblib
joblib.dump(final_model, 'heart_disease_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
print("\nFinal model and scaler saved to disk.")
```

Explanation:

Saving the Model:

- The final model and the scaler (used for feature scaling during preprocessing) are saved to disk using the joblib library. This allows for easy retrieval and use later without needing to retrain the model.
- The saved files are named "heart_disease_model.pkl" for the model and "scaler.pkl" for the scaler.

Conclusion:

Code:

```
# ============================================
# Conclusion
# ============================================

print("\n=== Project Summary ===")
print(f"Best Model: {best_model_name}")
print(f"Accuracy: {results[best_model_name]['accuracy']:.2%}")
print(f"Precision: {results[best_model_name]['precision']:.2%}")
print(f"Recall: {results[best_model_name]['recall']:.2%}")
print(f"F1 Score: {results[best_model_name]['f1']:.2%}")

print("\nProject completed successfully!")
```

Explanation:

Project Summary Explanation

The conclusion provides an overview of the project's outcomes:

1. **Best Model:** The best-performing model selected is the **Support Vector Machine (SVM)**.
2. **Performance Metrics:**
    - **Accuracy (84.78%):** Indicates that the model correctly predicts about 84.78% of all instances.
    - **Precision (90.91%):** Reflects that when the model predicts a positive case, it is correct 90.91% of the time, showing strong performance in avoiding false positives.
    - **Recall (82.57%):** Demonstrates that the model successfully identifies 82.57% of actual positive cases, indicating its effectiveness in detecting true positives.
    - **F1 Score (86.54%):** Balances precision and recall into a single score, highlighting overall performance.
3. **Completion Message:** Confirms that the project has been completed successfully, indicating that all steps were executed as planned.

Summary

Overall, the project effectively utilized the SVM model, achieving high accuracy and strong performance across key metrics.