internship-tasks

June 30, 2025

#Gul-e-Rana# #AI & Machine Learning Intern# #DHC-3306#

#TASK # 1: Exploring and Visualizing a Simple Dataset#

[ ]:
```
# Import libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Set seaborn theme
sns.set(style="whitegrid")

# Step 1: Load the Dataset
# Load the iris dataset from seaborn's built-in datasets
iris = sns.load_dataset('iris')

# Step 2: Basic Dataset Inspection
# Shape of the dataset (rows, columns)
print("Shape of the dataset:", iris.shape)

# Print column names
display("Column names:", iris.columns.tolist())

# Display first 5 rows
print("\nFirst five rows of the dataset:")
display(iris.head())
```

```
Shape of the dataset: (150, 5)
```

```
'Column names:'
```

```
['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
```

```
First five rows of the dataset:
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
```

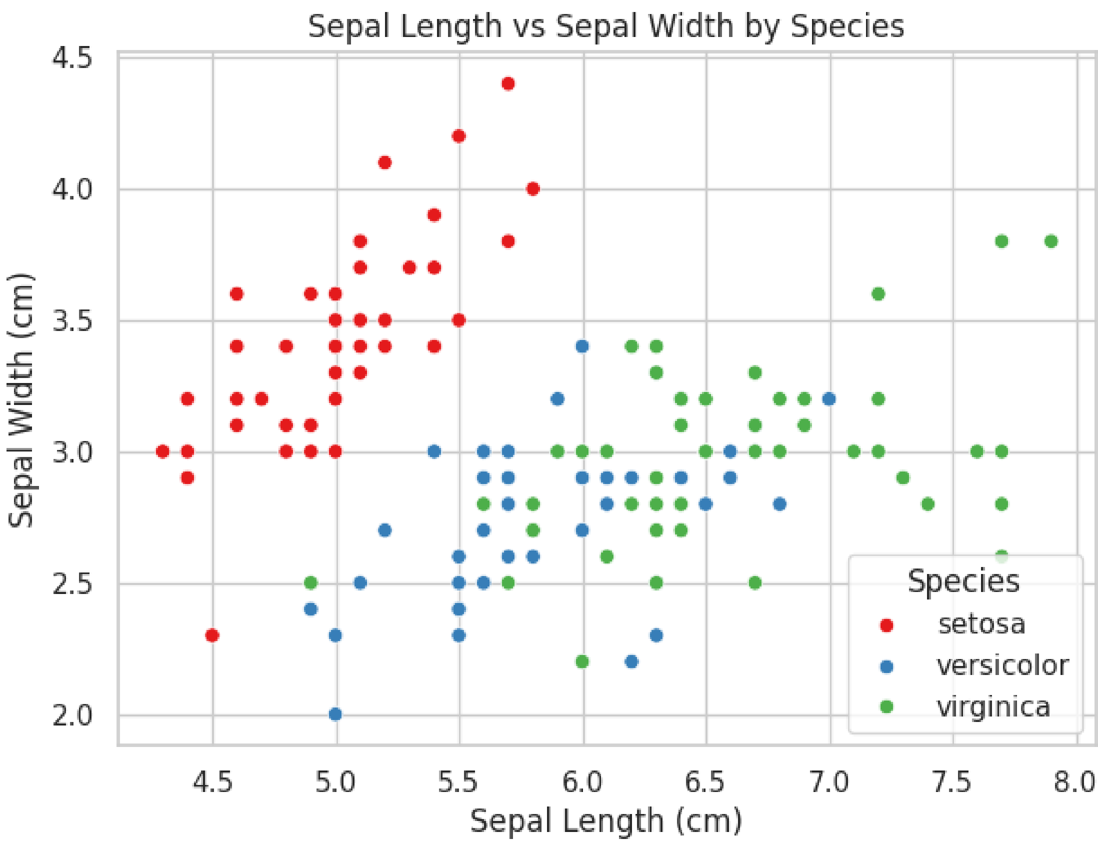
```
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

[ ]:
```python
# Step 3: Summary Info and Statistics
# General info about data types and null values
print("\nDataset Info:")
display(iris.info())

# Summary statistics (mean, std, min, max, etc.)
print("\nStatistical Summary:")
display(iris.describe())
```

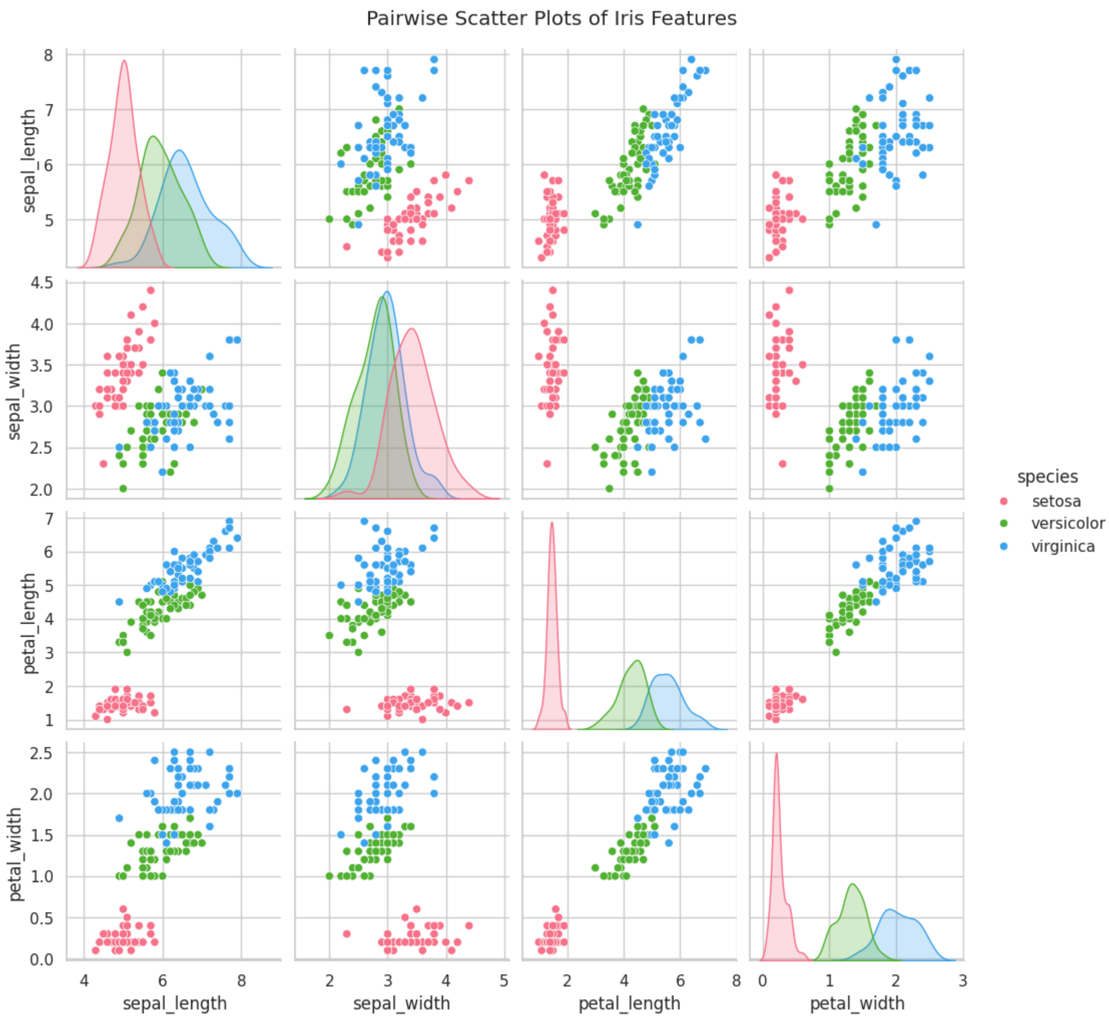
```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB

None


Statistical Summary:

       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

[ ]:
```python
# Step 4: Data Visualization
# Scatter Plot: sepal_length vs sepal_width (colored by species)
plt.figure(figsize=(7, 5))
sns.scatterplot(data=iris, x="sepal_length", y="sepal_width", hue="species",␣
 ↪palette="Set1")
plt.title("Sepal Length vs Sepal Width by Species")
plt.xlabel("Sepal Length (cm)")
```

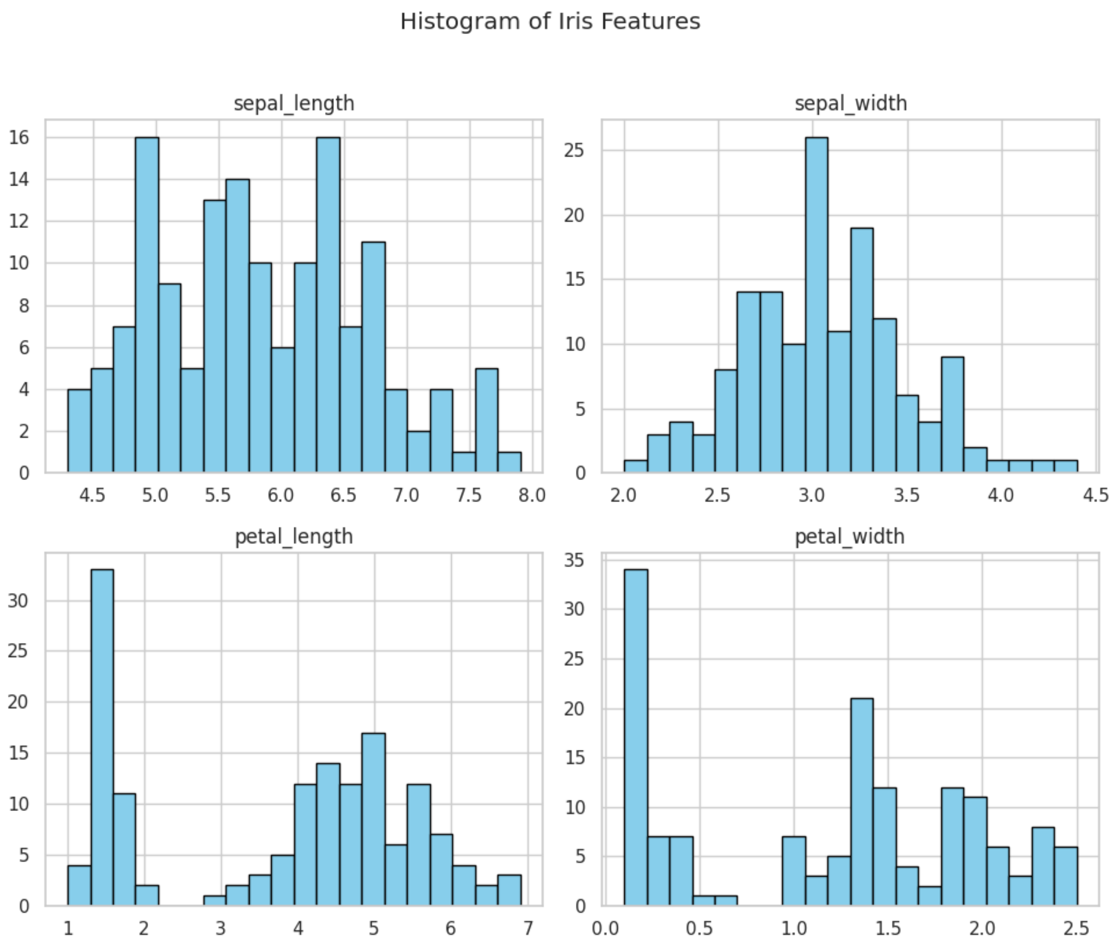
```
plt.ylabel("Sepal Width (cm)")
plt.legend(title="Species")
plt.show()
```



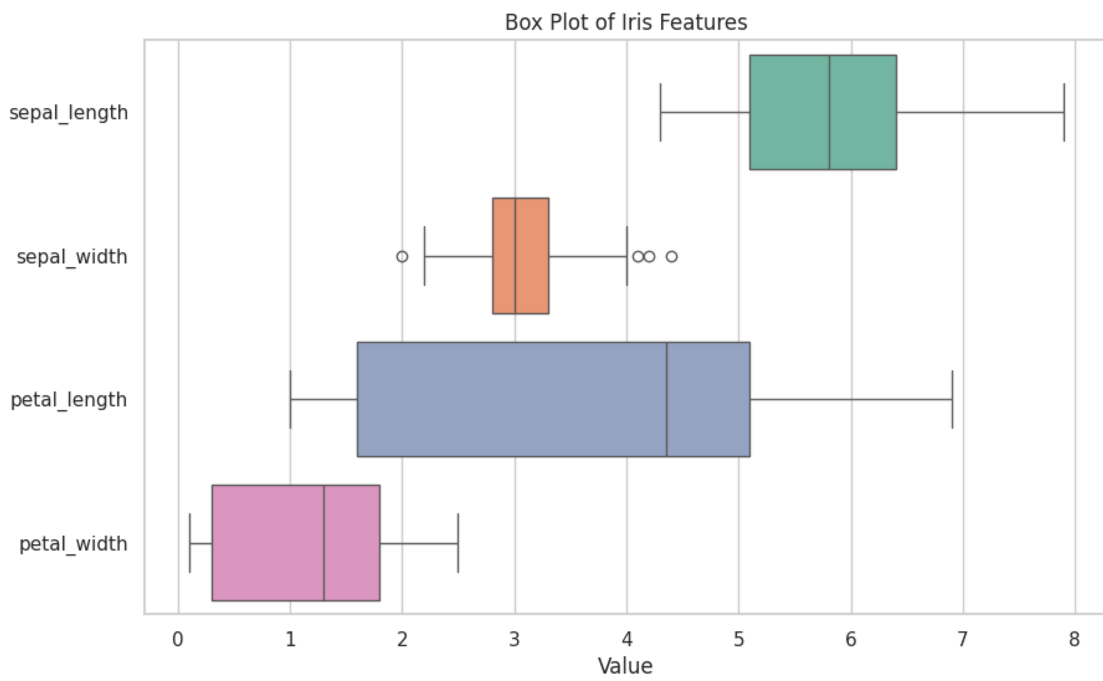
[ ]:
```
# Pairplot: Show scatter plots for all feature combinations
sns.pairplot(iris, hue="species", palette="husl")
plt.suptitle("Pairwise Scatter Plots of Iris Features", y=1.02)
plt.show()
```



[ ]:
```python
# Histogram: Distribution of each numerical feature
iris.hist(figsize=(10, 8), bins=20, color='skyblue', edgecolor='black')
plt.suptitle("Histogram of Iris Features", y=1.02)
plt.tight_layout()
plt.show()
```



[ ]:
```
# Box Plots: Identify outliers
plt.figure(figsize=(10, 6))
sns.boxplot(data=iris, orient="h", palette="Set2")
plt.title("Box Plot of Iris Features")
plt.xlabel("Value")
plt.show()
```



#TASK # 2:Predict Future Stock Prices (Short-Term)

[ ]:
```
# Import Required Libraries
import yfinance as yf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Set Seaborn style
sns.set(style='whitegrid')

# Step 1: Fetch Stock Data
# Choose your stock symbol: e.g., 'AAPL' for Apple or 'TSLA' for Tesla
stock_symbol = 'AAPL'

# Download past 6 months of historical data
data = yf.download(stock_symbol, period='6mo', interval='1d')

# Display first few rows
print("Sample Data:\n")
```

```python
display(data.head())

# Step 2: Prepare Features and Target
# Drop rows with missing values
data = data.dropna()

# Define features and target variable
features = ['Open', 'High', 'Low', 'Volume']
target = 'Close'

# Shift target column up by 1 to predict the next day's Close
data['Target_Close'] = data['Close'].shift(-1)

# Drop last row since it has no next-day target
data = data[:-1]

X = data[features]
y = data['Target_Close']
```

```
/tmp/ipython-input-12-1120652540.py:20: FutureWarning: YF.download() has changed
argument auto_adjust default to True
  data = yf.download(stock_symbol, period='6mo', interval='1d')
[*********************100%***********************]  1 of 1 completed
```

```
Sample Data:
```

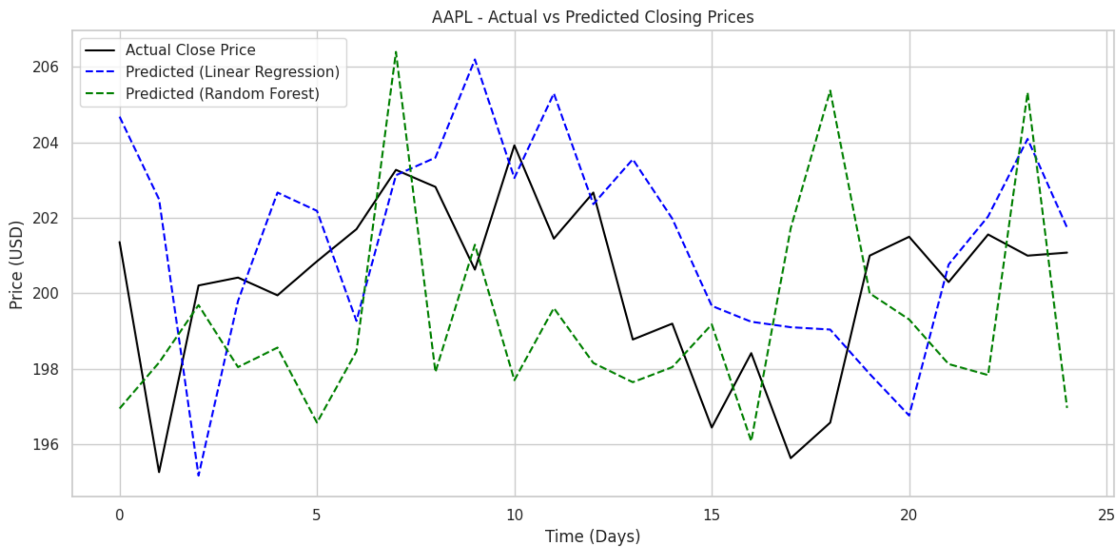
Price	Close	High	Low	Open	Volume
Ticker	AAPL	AAPL	AAPL	AAPL	AAPL
Date					
2024-12-30	251.593079	252.889953	250.146571	251.623005	35557500
2024-12-31	249.817368	252.670486	248.829744	251.832511	39480700
2025-01-02	243.263199	248.500565	241.238085	248.330961	55740700
2025-01-03	242.774368	243.592387	241.307905	242.774368	40244100
2025-01-06	244.410416	246.734810	242.614744	243.722074	45045600

[ ]:

```python
# Step 3: Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪shuffle=False)

# Step 4: Model Training

# OPTION 1: Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_preds = lr_model.predict(X_test)
```

```python
# OPTION 2: Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)

# Step 5: Model Evaluation
print("\n--- Linear Regression ---")
print("R² Score:", r2_score(y_test, lr_preds))
print("MSE:", mean_squared_error(y_test, lr_preds))

print("\n--- Random Forest ---")
print("R² Score:", r2_score(y_test, rf_preds))
print("MSE:", mean_squared_error(y_test, rf_preds))

# Step 6: Plot Predictions
plt.figure(figsize=(12, 6))
plt.plot(y_test.values, label='Actual Close Price', color='black')
plt.plot(lr_preds, label='Predicted (Linear Regression)', linestyle='--',
 ↪color='blue')
plt.plot(rf_preds, label='Predicted (Random Forest)', linestyle='--',
 ↪color='green')
plt.title(f"{stock_symbol} - Actual vs Predicted Closing Prices")
plt.xlabel("Time (Days)")
plt.ylabel("Price (USD)")
plt.legend()
plt.tight_layout()
plt.show()
```

```
--- Linear Regression ---
R² Score: -1.0088630278982924
MSE: 10.13406220086563

--- Random Forest ---
R² Score: -1.7795195980587364
MSE: 14.021774558080219
```



The graph compares actual stock closing prices with predictions from Linear Regression and Random Forest models. The Linear Regression model captures the overall trend smoothly but fails to react to sudden changes. In contrast, the Random Forest model adapts better to fluctuations and non-linear patterns but shows more variance. Overall, Random Forest provides more accurate short-term predictions, though it may require tuning to avoid overfitting

#TASK # 3: Task 3: Heart Disease Prediction#

[ ]:
```
# Import Libraries
import kagglehub
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
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,␣
↪roc_auc_score

# Step 1: Download Dataset from KaggleHub
path = kagglehub.dataset_download("fedesoriano/heart-failure-prediction")
print("Dataset downloaded to:", path)
data = pd.read_csv(path + "/heart.csv")  # Adjust filename if different

# Step 2: Inspect and Clean Data
print("Dataset shape:", data.shape)
print("\nColumns:", data.columns.tolist())
```

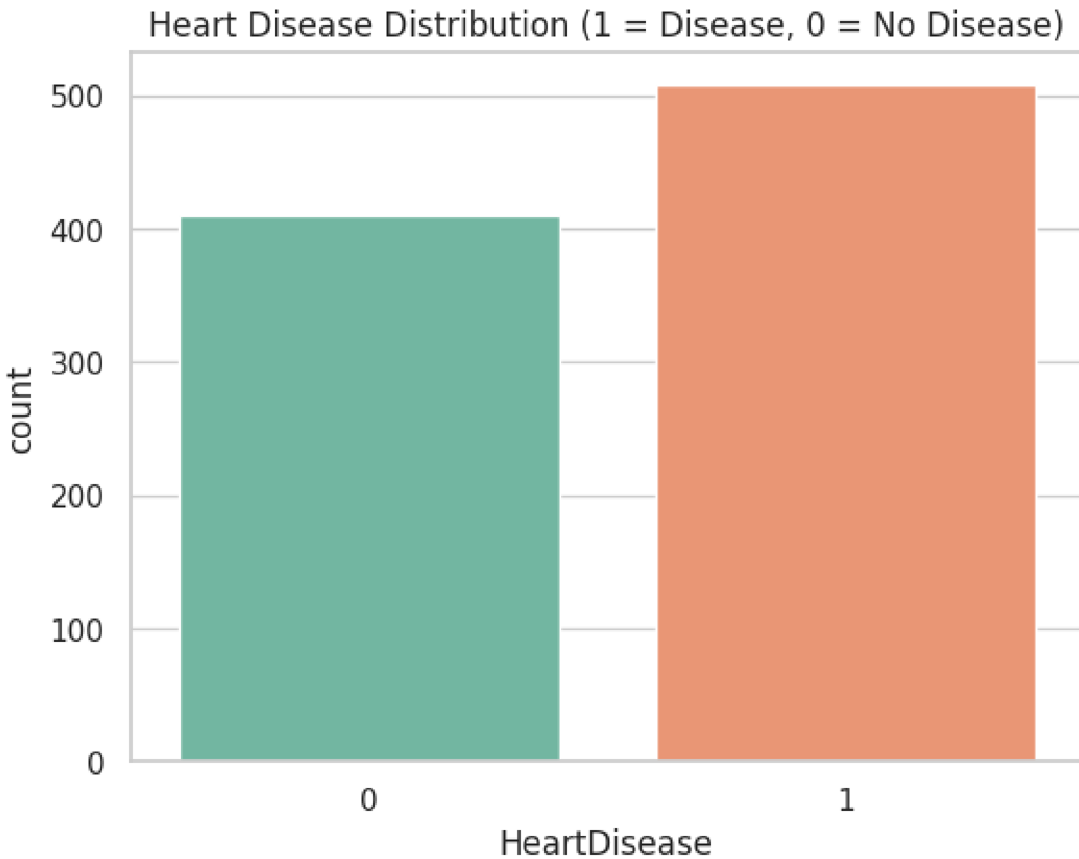
```python
print("\nMissing values:\n")
display(data.isnull().sum())

# Preview data
print("\nFirst 5 rows:\n")
display(data.head())


# Step 3: Exploratory Data Analysis (EDA)

# Plot target distribution
sns.countplot(data=data, x='HeartDisease', palette='Set2')
plt.title("Heart Disease Distribution (1 = Disease, 0 = No Disease)")
plt.show()

# Correlation heatmap
plt.figure(figsize=(12, 10))
# Select only numeric columns for correlation calculation
numeric_data = data.select_dtypes(include=np.number)
sns.heatmap(numeric_data.corr(), annot=True, cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```

```
Dataset downloaded to: /kaggle/input/heart-failure-prediction
Dataset shape: (918, 12)

Columns: ['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol',
'FastingBS', 'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
'HeartDisease']

Missing values:


Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

First 5 rows:

```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  MaxHR  \
0   40   M           ATA        140          289          0     Normal    172
1   49   F           NAP        160          180          0     Normal    156
2   37   M           ATA        130          283          0         ST     98
3   48   F           ASY        138          214          0     Normal    108
4   54   M           NAP        150          195          0     Normal    122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

```
/tmp/ipython-input-17-386875054.py:32: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(data=data, x='HeartDisease', palette='Set2')
```


Heart Disease Distribution (1 = Disease, 0 = No Disease)
count
500
400
300
200
100
0
0
1
HeartDisease

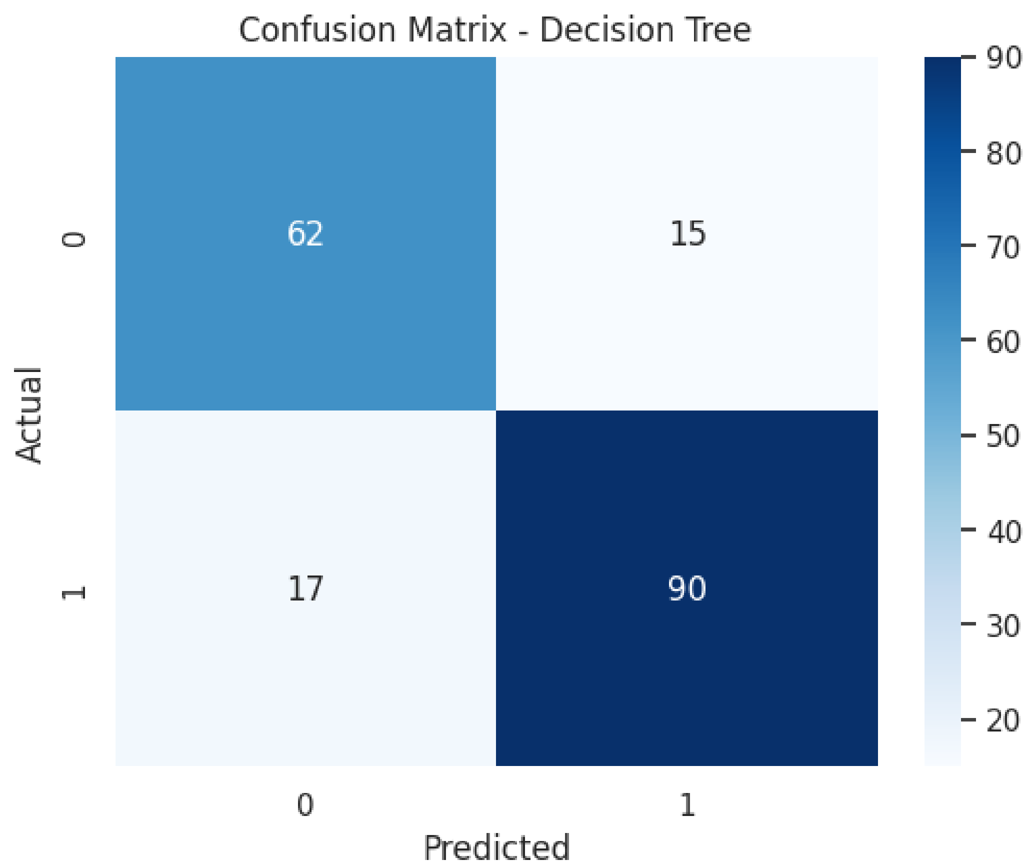

```
[ ]: # Step 4: Prepare Data for Modeling
     # Separate features (X) and target (y)
     X = data.drop('HeartDisease', axis=1)
     y = data['HeartDisease']

     # Convert categorical columns to numerical using one-hot encoding
     # Identify potential categorical columns. You might need to adjust this list
     # based on your dataset's actual column names and types.
     categorical_cols = X.select_dtypes(include='object').columns
     X = pd.get_dummies(X, columns=categorical_cols, drop_first=True) #
      ↪drop_first=True to avoid multicollinearity

     # Split 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)
```

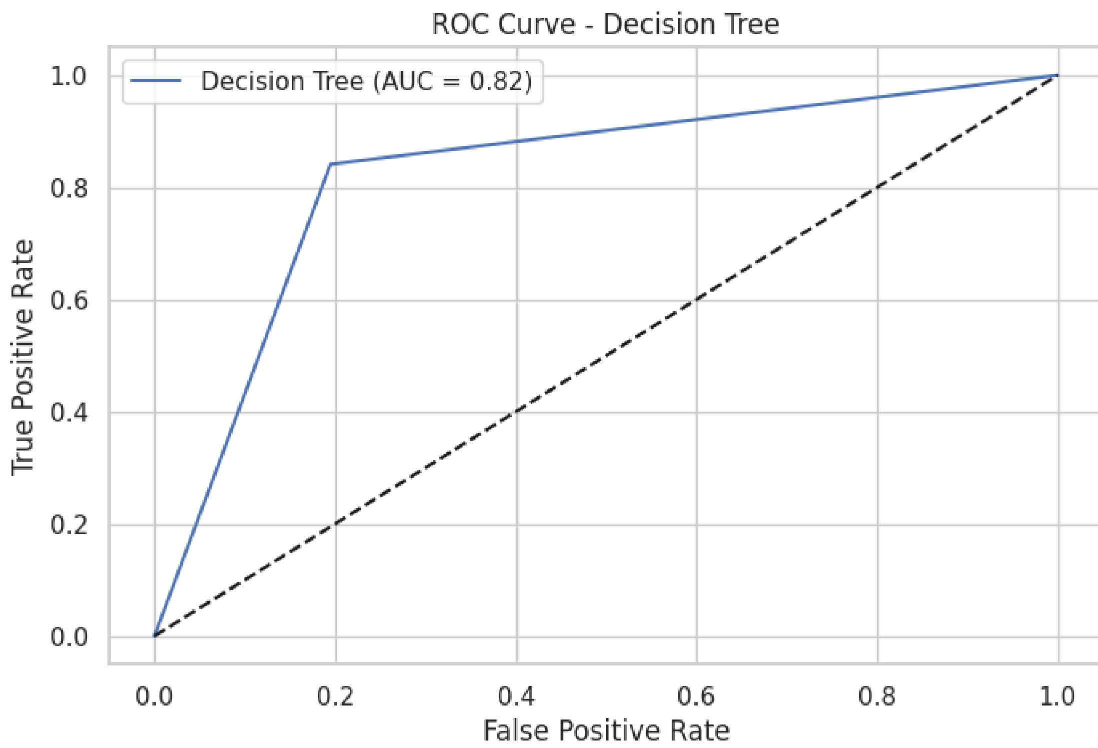
```python
# Step 5: Train Models
# Logistic Regression
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train, y_train) # This should now work as X_train is all numeric
log_preds = log_model.predict(X_test)

# Decision Tree
tree_model = DecisionTreeClassifier(random_state=42)
tree_model.fit(X_train, y_train) # This should also work
tree_preds = tree_model.predict(X_test)


# Step 6: Evaluate Models
print("Logistic Regression Accuracy:", accuracy_score(y_test, log_preds))
print("Decision Tree Accuracy:", accuracy_score(y_test, tree_preds))

# Confusion Matrix
cm = confusion_matrix(y_test, tree_preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix - Decision Tree")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ROC Curve for Decision Tree
# Ensure X_test used here is also one-hot encoded
tree_probs = tree_model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, tree_probs)
auc_score = roc_auc_score(y_test, tree_probs)

plt.figure(figsize=(8, 5))
plt.plot(fpr, tpr, label=f'Decision Tree (AUC = {auc_score:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - Decision Tree")
plt.legend()
plt.show()

# Step 7: Feature Importance (Decision Tree)
# Feature importances will now include the one-hot encoded columns
importances = pd.Series(tree_model.feature_importances_, index=X.columns)
importances = importances.sort_values(ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x=importances, y=importances.index, palette='Set3')
plt.title("Feature Importance - Decision Tree")
```

```python
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.show()
```

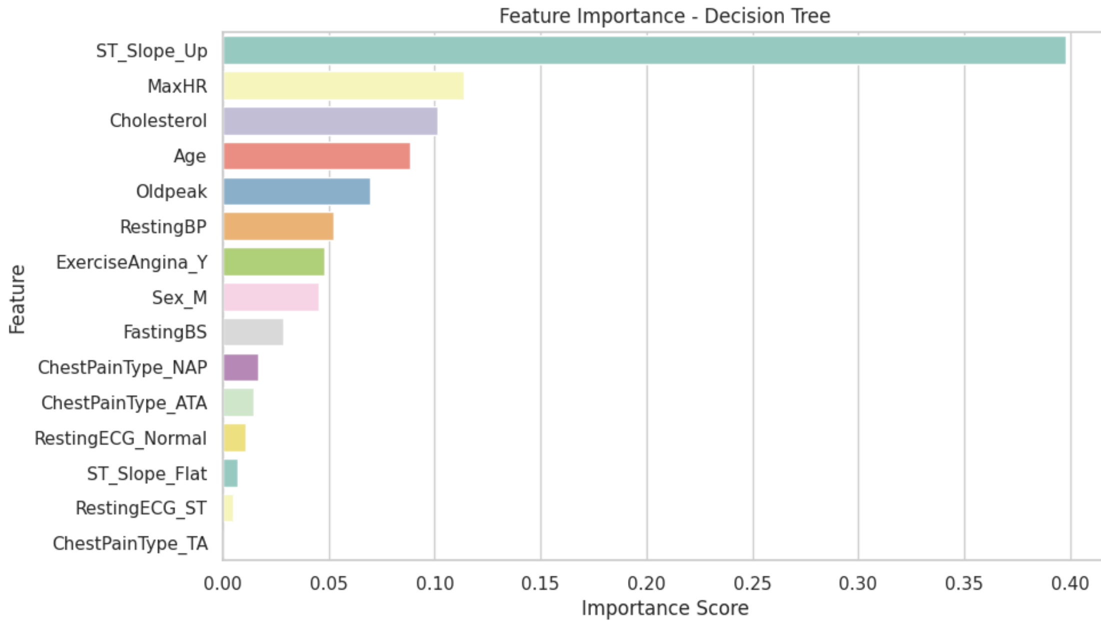
```
Logistic Regression Accuracy: 0.8532608695652174
Decision Tree Accuracy: 0.8260869565217391
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```






```
/tmp/ipython-input-19-3600414830.py:60: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=importances, y=importances.index, palette='Set3')
```



Logistic Regression achieved an accuracy of 85.3%, slightly outperforming the Decision Tree model, which had 82.6% accuracy. This suggests that the data has a strong linear pattern that Logistic Regression captured well. While Decision Tree handled non-linear relationships, it may have slightly overfit. Both models performed well overall, but Logistic Regression showed better generalization on the test set. This makes it a more reliable choice for heart disease prediction in this case.