JUPYTER NOTEBOOK SCRENNSHOTS FOR HEART DESEASE ML PROJECT

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("C:/Users/Admin/Downloads/heart.csv")

# Display basic info
print("Dataset Shape:", df.shape)
print("\nFirst 5 Rows:")
df.head()
```

```
Dataset Shape: (1025, 14)

First 5 Rows:
```

[2]:

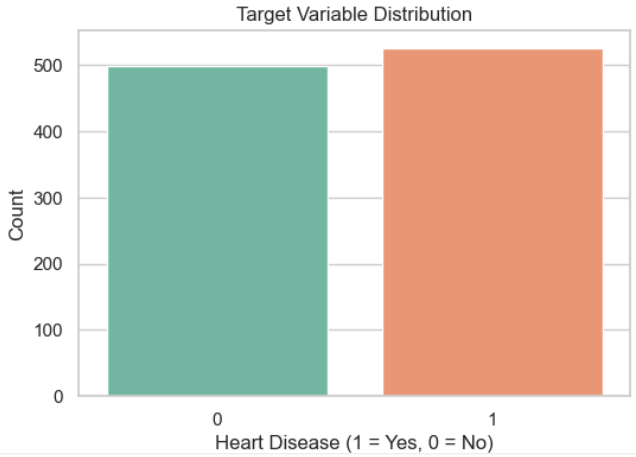
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

[3]:
```
# Check for missing values
print("Missing Values:\n", df.isnull().sum())

# Drop rows with all NaN values (if any)
df.dropna(how='all', inplace=True)

# Fill missing values (if needed)
df.fillna(0, inplace=True)  # Replace NaNs with 0 (for binary symptoms)
```

```
Missing Values:
 age         0
sex          0
cp           0
trestbps     0
chol         0
fbs          0
restecg      0
thalach      0
exang        0
oldpeak      0
slope        0
ca           0
thal         0
target       0
dtype: int64
```

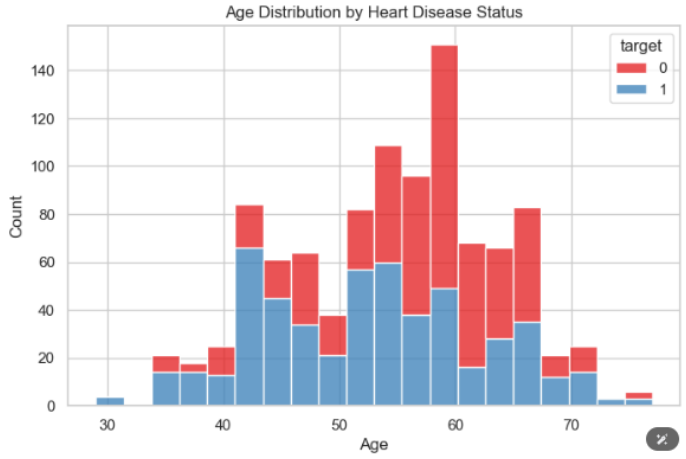
[7]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(style="whitegrid")

plt.figure(figsize=(6, 4))
sns.countplot(x='target', data=df, palette='Set2')
plt.title('Target Variable Distribution')
plt.xlabel('Heart Disease (1 = Yes, 0 = No)')
plt.ylabel('Count')
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_14408\1407634300.py:7: FutureWarning:

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

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



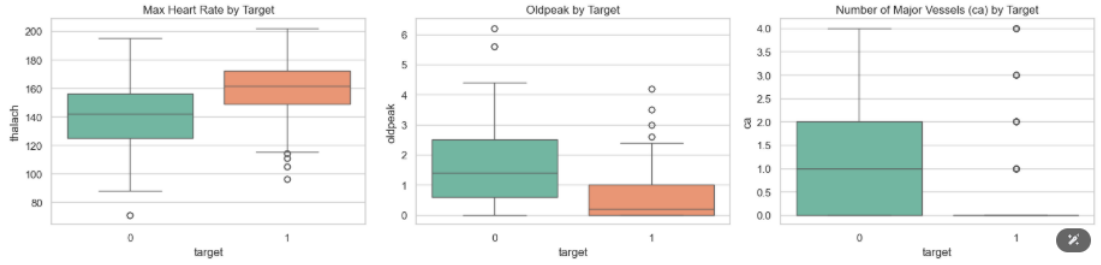
[9]:
```
plt.figure(figsize=(12, 10))
corr = df.corr()
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Feature Correlation Heatmap')
plt.show()
```

[10]:
```
plt.figure(figsize=(8, 5))
sns.histplot(data=df, x='age', hue='target', multiple='stack', palette='Set1', bins=20)
plt.title('Age Distribution by Heart Disease Status')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```



[11]:
```
plt.figure(figsize=(16, 4))
# Max heart rate
plt.subplot(1, 3, 1)
sns.boxplot(x='target', y='thalach', data=df, palette='Set2')
plt.title('Max Heart Rate by Target')
# ST depression
plt.subplot(1, 3, 2)
sns.boxplot(x='target', y='oldpeak', data=df, palette='Set2')
plt.title('Oldpeak by Target')
# Number of major vessels
plt.subplot(1, 3, 3)
sns.boxplot(x='target', y='ca', data=df, palette='Set2')
plt.title('Number of Major Vessels (ca) by Target')
plt.tight_layout()
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_14408\2589475323.py:5: 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.boxplot(x='target', y='thalach', data=df, palette='Set2')
C:\Users\Admin\AppData\Local\Temp\ipykernel_14408\2589475323.py:10: 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.boxplot(x='target', y='oldpeak', data=df, palette='Set2')
C:\Users\Admin\AppData\Local\Temp\ipykernel_14408\2589475323.py:15: 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.boxplot(x='target', y='ca', data=df, palette='Set2')
```



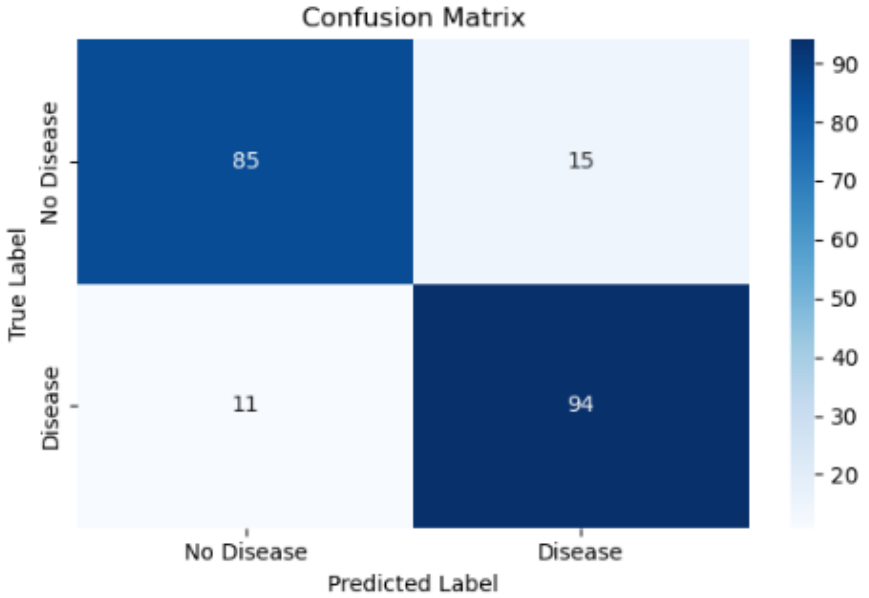
[3]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, recall_score, confusion_matrix, classification_report

# Load dataset
df = pd.read_csv("C:/Users/Admin/Downloads/heart.csv")

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

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)

# Decision Tree model
dt = DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42)
dt.fit(X_train, y_train)

# Predictions
y_pred_dt = dt.predict(X_test)

# Evaluation
print("Decision Tree Classifier")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Recall:", recall_score(y_test, y_pred_dt))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
print("Classification Report:\n", classification_report(y_test, y_pred_dt))
```

```
Decision Tree Classifier
Accuracy: 0.8731707317073171
Recall: 0.8952380952380953
Confusion Matrix:
 [[85 15]
 [11 94]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.85      0.87       100
           1       0.86      0.90      0.88       105

    accuracy                           0.87       205
   macro avg       0.87      0.87      0.87       205
weighted avg       0.87      0.87      0.87       205
```

[7]:
```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Make predictions using your model (example: decision tree)
# Replace 'dt' with the actual model you're using (e.g., rf, xgb, log_reg)
y_pred = dt.predict(X_test)

# Step 2: Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Step 3: Print text-based confusion matrix and classification report
print("Confusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Step 4: Plot confusion matrix heatmap
labels = ['No Disease', 'Disease']  # Adjust based on your dataset
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()
```

```
Confusion Matrix:
 [[85 15]
 [11 94]]
```

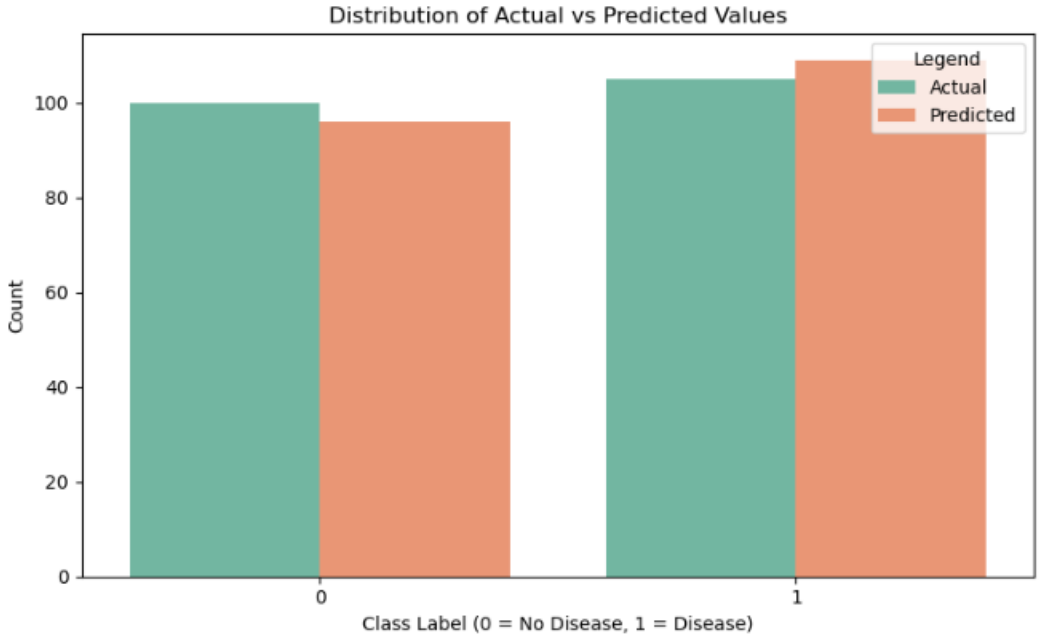
```
[11 94]]

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.85      0.87       100
           1       0.86      0.90      0.88       105

    accuracy                           0.87       205
   macro avg       0.87      0.87      0.87       205
weighted avg       0.87      0.87      0.87       205
```



[10]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, recall_score, confusion_matrix, classification_report

# Load the dataset
df = pd.read_csv("C:/Users/Admin/Downloads/heart.csv")

# Separate features and target
X = df.drop("target", axis=1)
y = df["target"]

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# Create and train the model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict
y_pred = rf.predict(X_test)

# Evaluate
print("Random Forest Classifier Results")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

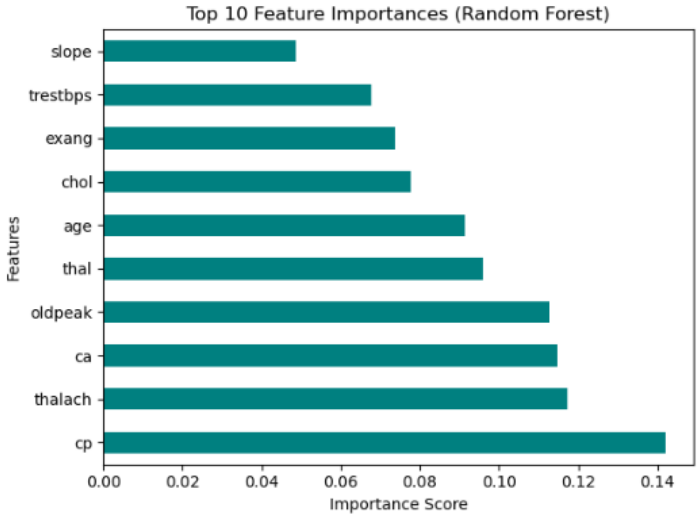
```
print("Classification Report:\n", classification_report(y_test,
```

```
Random Forest Classifier Results
Accuracy: 1.0
Recall: 1.0
Confusion Matrix:
 [[100   0]
 [  0 105]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       105

    accuracy                           1.00       205
   macro avg       1.00      1.00      1.00       205
weighted avg       1.00      1.00      1.00       205
```

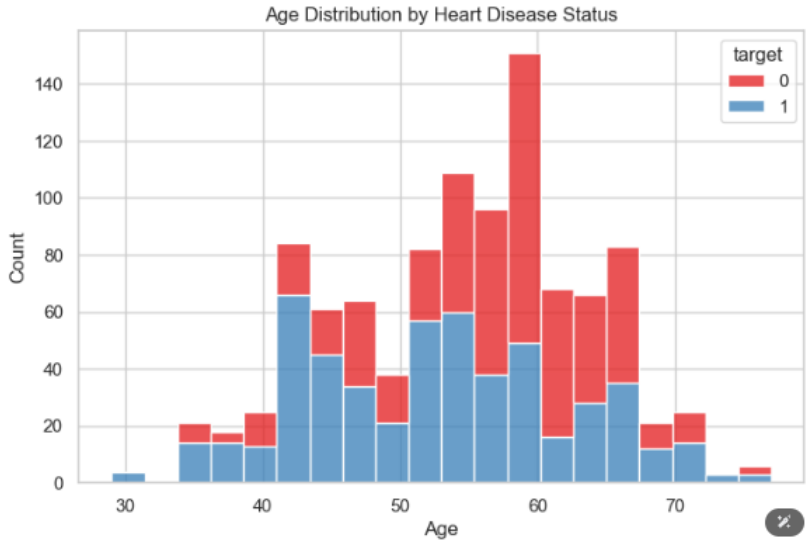


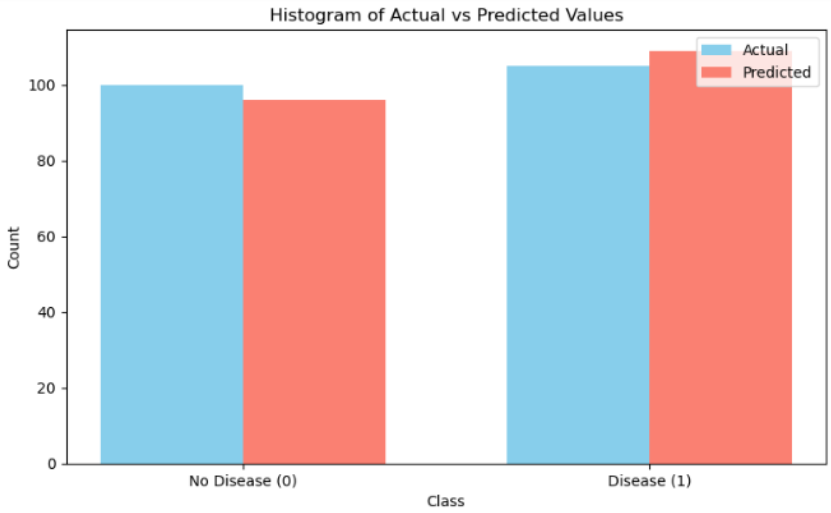
[8]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Ensure you have actual and predicted values
# y_test: actual labels
# y_pred: predicted labels from your model
# If not done already:
# y_pred = model.predict(X_test)

# Create a DataFrame to compare
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

# Plot the count of actual vs predicted values
plt.figure(figsize=(8, 5))
sns.countplot(data=pd.melt(comparison_df), x='value', hue='variable', palette='Set2')
plt.title('Distribution of Actual vs Predicted Values')
plt.xlabel('Class Label (0 = No Disease, 1 = Disease)')
plt.ylabel('Count')
plt.legend(title='Legend')
plt.tight_layout()
plt.show()
```

[11]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot feature importances
feat_importances = pd.Series(rf.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh', color='teal')
plt.title("Top 10 Feature Importances (Random Forest)")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.tight_layout()
plt.show()
```



[10]:
```
plt.figure(figsize=(8, 5))
sns.histplot(data=df, x='age', hue='target', multiple='stack', palette='Set1', bins=20)
plt.title('Age Distribution by Heart Disease Status')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```




Histogram of Actual vs Predicted Values
Actual
Predicted
Count
0
20
40
60
80
100
No Disease (0)
Disease (1)
Class