Exploring Heart Disease Predictors with Multivariate Techniques

S/19/842

R. Imesha Rajapaksha

STA 4053

Mini Project

Introduction

This report explores the application of several advanced multivariate statistical techniques to analyze a heart disease dataset. The goal is to simplify complex data structures, identify hidden patterns, and develop effective predictive models. Multivariate methods are particularly useful when dealing with datasets containing numerous interrelated variables, as they allow us to reduce dimensionality, interpret underlying relationships, and improve classification accuracy.

The techniques used in this study include:

- **Principal Component Analysis (PCA):** Reduces the number of variables by transforming them into a smaller set of uncorrelated components while preserving most of the data's variability.
- **Factor Analysis (FA):** Identifies latent variables that explain observed correlations among variables, offering insight into the underlying structure of the data.
- **Canonical Correlation Analysis (CCA):** Examines relationships between two distinct sets of variables, revealing how different domains (e.g., clinical and demographic data) are connected.
- **Discriminant Analysis (DA):** Focuses on classifying observations into predefined groups by finding the best combination of variables that separates those groups.

Methology

Introduction to the UCI Heart Disease Dataset

The **UCI Heart Disease Dataset** is a cornerstone resource in medical data analysis and machine learning, widely utilized for developing predictive models of cardiovascular health. This dataset, sourced from the UCI Machine Learning Repository, aggregates clinical and demographic information from patients to discern the presence of heart disease.

Dataset Structure:

- **Size of Dataset:** The dataset comprises **920 instances (rows)**, each representing an individual patient's record.
- **Number of Variables:** It contains **14 key predictive variables (features)** along with an id, dataset identifier, and the original num diagnosis variable. The num variable is typically transformed into a binary target for classification.

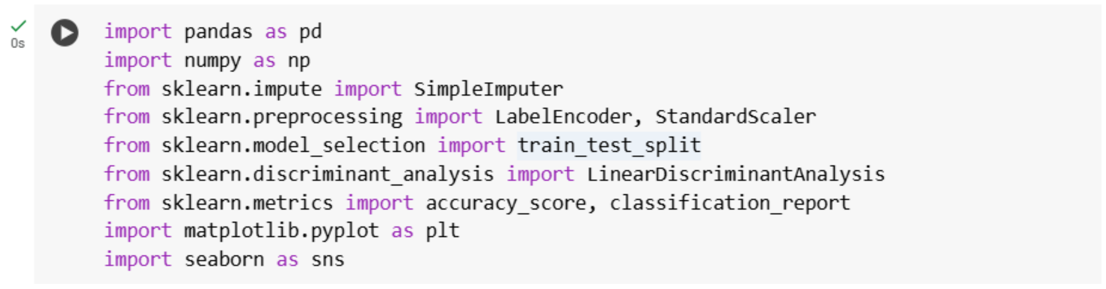
Variable Types: The features within the dataset can be broadly categorized as follows:

- **Numerical Variables:** These are quantitative measurements providing continuous or ordered discrete data. Key numerical features include:
  - **age**: Patient's age in years.
  - **trestbps**: Resting blood pressure (in mm Hg on admission to the hospital).
  - **chol**: Serum cholesterol in mg/dl.
  - **thalch**: Maximum heart rate achieved during exercise.
  - **oldpeak**: ST depression induced by exercise relative to rest.
  - **ca**: Number of major vessels (0-3) colored by fluoroscopy, indicating blocked arteries.
- **Categorical Variables:** These represent qualitative data, often with distinct categories. These include:
  - **sex**: Patient's sex (Male/Female).
  - **cp**: Chest pain type (e.g., typical angina, atypical angina, non-anginal pain, asymptomatic).

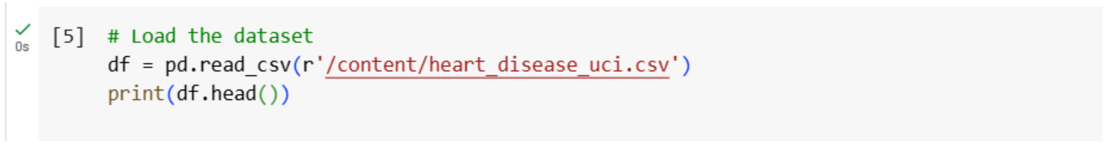
- ◦ **fbs**: Fasting blood sugar (> 120 mg/dl, indicating higher risk).
- ◦ **restecg**: Resting electrocardiographic results (e.g., normal, ST-T wave abnormality, left ventricular hypertrophy).
- ◦ **exang**: Exercise induced angina (yes/no).
- ◦ **slope**: The slope of the peak exercise ST segment (e.g., upsloping, flat, downsloping).
- ◦ **thal**: Thallium stress test result (e.g., normal, fixed defect, reversable defect).

- **Target Variable:**
  - ◦ **num (original)**: Diagnosis of heart disease (angiographic disease status). This is typically converted into a **binary target (0 for no disease, 1 for presence of disease)** for most classification tasks.

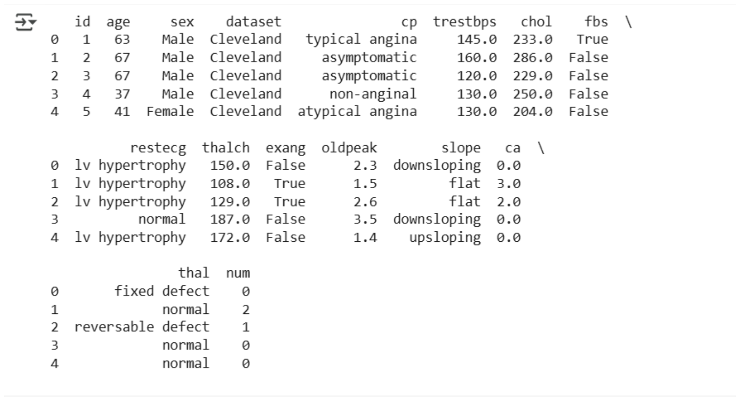
1.) PRINCIPAL COMPONENT ANALYSIS



```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

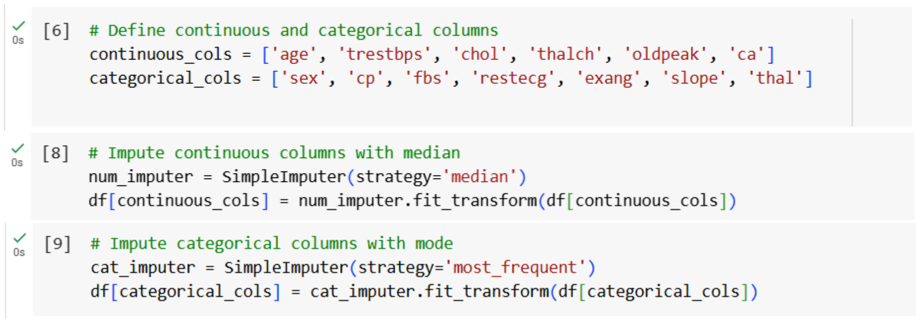


```
[5] # Load the dataset
    df = pd.read_csv(r'/content/heart_disease_uci.csv')
    print(df.head())
```



```
   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
```

We can observe the original variable names and their initial values. It's evident that there are both numerical (e.g., age, trestbps, chol) and categorical (e.g., sex, cp, thal) features. The num column, which indicates the extent of heart disease (0 for no disease, >0 for disease), is the basis for your binary target variable. The presence of text in categorical columns and potentially missing values (not always visible in head() but handled later) highlights the need for preprocessing.

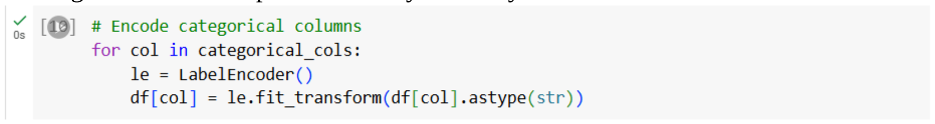


```
[6]  # Define continuous and categorical columns
     continuous_cols = ['age', 'trestbps', 'chol', 'thalch', 'oldpeak', 'ca']
     categorical_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'thal']
```

```
[8]  # Impute continuous columns with median
     num_imputer = SimpleImputer(strategy='median')
     df[continuous_cols] = num_imputer.fit_transform(df[continuous_cols])
```

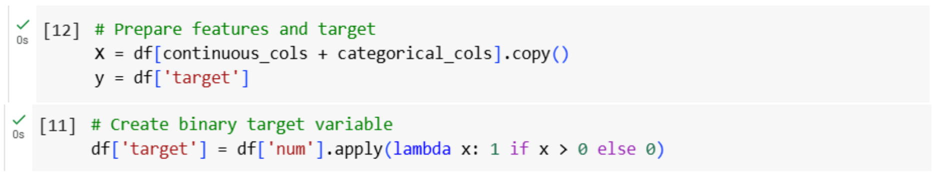
```
[9]  # Impute categorical columns with mode
     cat_imputer = SimpleImputer(strategy='most_frequent')
     df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])
```

The DataFrame Info shows 920 non-null counts for all columns. This confirms that the SimpleImputer successfully filled all missing values. Numerical columns were imputed with their median, and categorical ones with their mode, making the dataset complete and ready for analysis.



```
[10] # Encode categorical columns
     for col in categorical_cols:
         le = LabelEncoder()
         df[col] = le.fit_transform(df[col].astype(str))
```

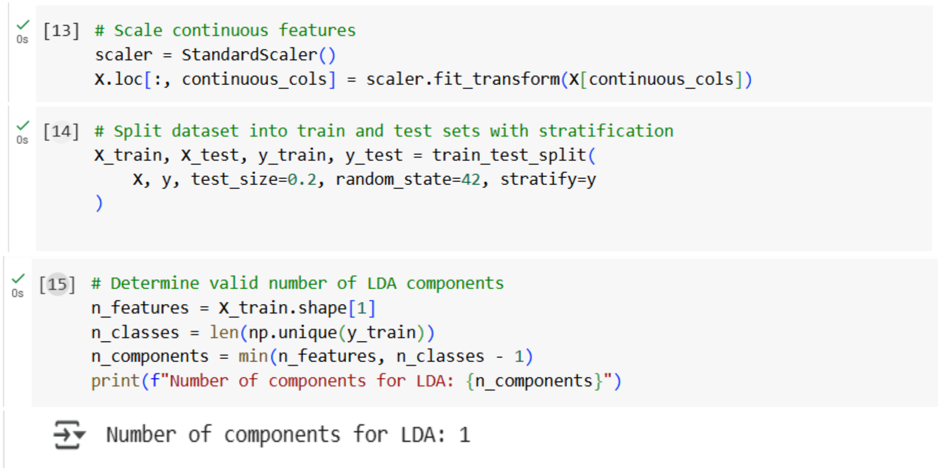
The head() output shows that columns like sex, cp, fbs, restecg, exang, slope, and thal are now represented by integers (e.g., sex from 'Male'/'Female' to 1/0). This is crucial for LDA, which operates on numerical data.



```
[12] # Prepare features and target
     X = df[continuous_cols + categorical_cols].copy()
     y = df['target']
```

```
[11] # Create binary target variable
     df['target'] = df['num'].apply(lambda x: 1 if x > 0 else 0)
```

The target column has been successfully added, converting the multi-level num variable into a binary 0 (no heart disease) or 1 (heart disease present) for classification.



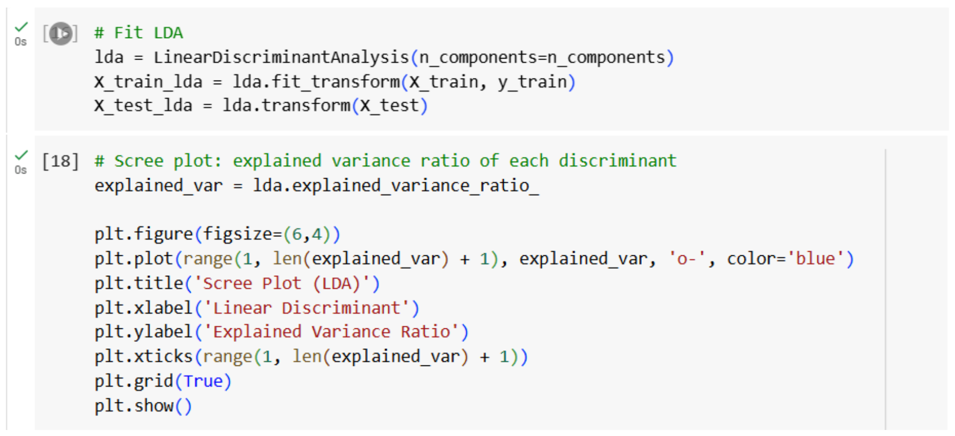
```
[13] # Scale continuous features
     scaler = StandardScaler()
     X.loc[:, continuous_cols] = scaler.fit_transform(X[continuous_cols])
```

```
[14] # Split dataset into train and test sets with stratification
     X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.2, random_state=42, stratify=y
     )
```

```
[15] # Determine valid number of LDA components
     n_features = X_train.shape[1]
     n_classes = len(np.unique(y_train))
     n_components = min(n_features, n_classes - 1)
     print(f"Number of components for LDA: {n_components}")
```

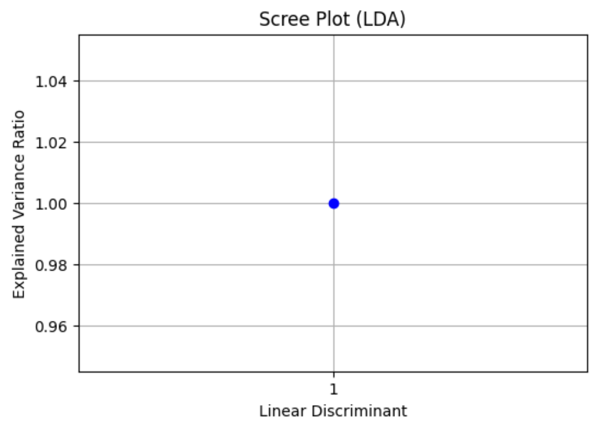
```
Number of components for LDA: 1
```

This output confirms the theoretical maximum number of linear discriminant functions LDA can create for our specific problem. Since we have a binary classification target (2 classes: heart disease or not), LDA can compute at most (number of classes - 1) = (2 - 1) = 1 discriminant component. This single component will be the axis that best distinguishes between the two heart disease groups.



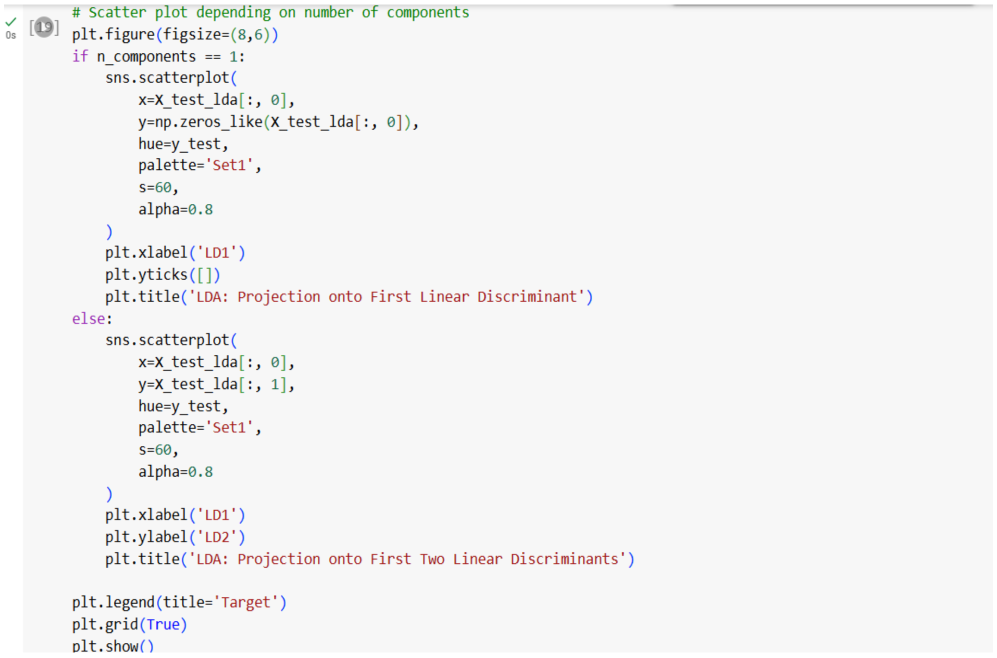
```
# Fit LDA
lda = LinearDiscriminantAnalysis(n_components=n_components)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)
```

```
[18] # Scree plot: explained variance ratio of each discriminant
explained_var = lda.explained_variance_ratio_

plt.figure(figsize=(6,4))
plt.plot(range(1, len(explained_var) + 1), explained_var, 'o-', color='blue')
plt.title('Scree Plot (LDA)')
plt.xlabel('Linear Discriminant')
plt.ylabel('Explained Variance Ratio')
plt.xticks(range(1, len(explained_var) + 1))
plt.grid(True)
plt.show()
```

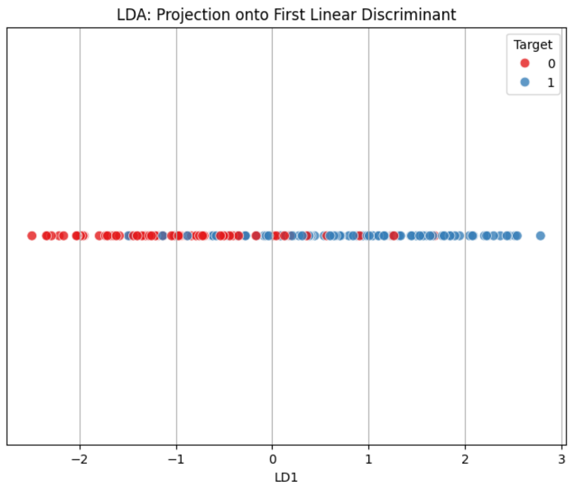


This scree plot shows only **one linear discriminant (LD1)**.

Explained Variance: The bar/point for LD1 shows that it explains **100% of the variance** captured by the LDA model. This is always the case for binary classification, as the single discriminant completely encompasses the discriminative information between the two classes. In multi-class problems, you would see multiple bars, each showing the proportion of explained variance by successive discriminants, and you'd look for an "elbow" similar to PCA. Here, it simply confirms that the LDA has found the single best linear combination of features to separate your 'No Heart Disease' and 'Heart Disease'.



```
# Scatter plot depending on number of components
plt.figure(figsize=(8,6))
if n_components == 1:
    sns.scatterplot(
        x=X_test_lda[:, 0],
        y=np.zeros_like(X_test_lda[:, 0]),
        hue=y_test,
        palette='Set1',
        s=60,
        alpha=0.8
    )
    plt.xlabel('LD1')
    plt.yticks([])
    plt.title('LDA: Projection onto First Linear Discriminant')
else:
    sns.scatterplot(
        x=X_test_lda[:, 0],
        y=X_test_lda[:, 1],
        hue=y_test,
        palette='Set1',
        s=60,
        alpha=0.8
    )
    plt.xlabel('LD1')
    plt.ylabel('LD2')
    plt.title('LDA: Projection onto First Two Linear Discriminants')

plt.legend(title='Target')
plt.grid(True)
plt.show()
```



This plot visually represents how well our LDA model has separated the two classes (Target 0: No Heart Disease, and Target 1: Heart Disease) in the reduced 1-dimensional space.

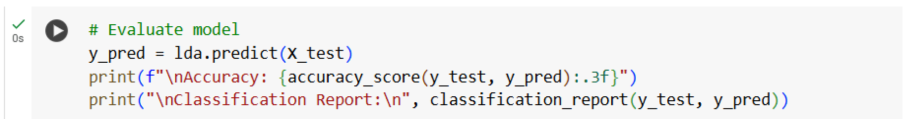
Since n_components = 1, all data points from the test set (X_test_lda) are projected onto a single horizontal line (the x-axis, labeled 'LD1'). The y-axis is compressed to 0 and effectively serves only to display the points.

can clearly see two main clusters of points:

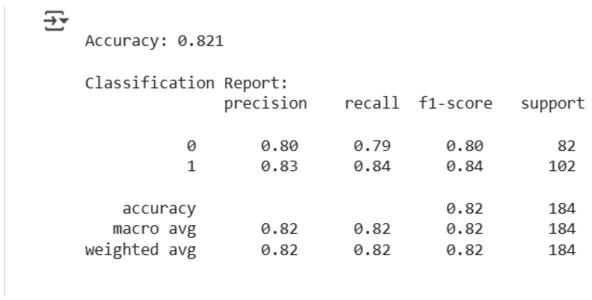
- **Blue points (Target 0):** Tend to be on the left side of the plot, generally with lower LD1 scores. These are the individuals predicted to have "No Heart Disease."
- **Red points (Target 1):** Tend to be on the right side of the plot, generally with higher LD1 scores. These are the individuals predicted to have "Heart Disease."

While there is a good degree of separation, there's some **overlap** between the blue and red points in the middle section of the LD1 axis. This overlap indicates individuals who are difficult for the LDA model to clearly distinguish between the two classes. These ambiguous cases are where misclassifications (false positives or false negatives) are most likely to occur.

The plot successfully illustrates LDA's objective: to find a projection that maximizes the distance between the means of the two classes while minimizing the variance within each class, thereby enhancing their separability.



```
# Evaluate model
y_pred = lda.predict(X_test)
print(f"\nAccuracy: {accuracy_score(y_test, y_pred):.3f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```



```
Accuracy: 0.821

Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.79      0.80        82
           1       0.83      0.84      0.84       102

    accuracy                           0.82       184
   macro avg       0.82      0.82      0.82       184
weighted avg       0.82      0.82      0.82       184
```

Accuracy: 0.821

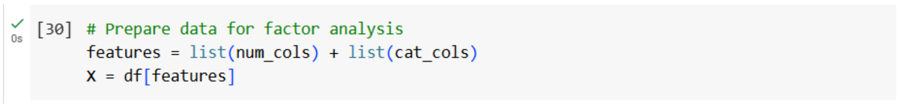
- **Meaning:** This means your LDA model correctly predicted the heart disease status (0 or 1) for approximately **82.1%** of the individuals in your test dataset. This is a decent overall accuracy.
- **Context:** Accuracy is a good general indicator but can be misleading if class distributions are highly imbalanced. In this case, your support shows 82 cases for class 0 and 102 for class 1, which is reasonably balanced.

Classification Report: This provides a more granular breakdown of performance for each class.

- **Class 0 (No Heart Disease):**
  - precision: **0.80** - When the model *predicts* an individual has no heart disease, it is correct 80% of the time. (Out of all samples predicted as class 0, 80% were actually class 0).
  - recall: **0.79** - The model correctly identifies 79% of all individuals who *actually* do not have heart disease. (Out of all actual class 0 samples, 79% were correctly identified).
  - f1-score: **0.80** - This is the harmonic mean of precision and recall. It's a balanced measure, indicating good performance for predicting the 'No Heart Disease' class.
  - support: **82** - There were 82 actual instances of 'No Heart Disease' in your test set.
- **Class 1 (Heart Disease):**
  - precision: **0.83** - When the model *predicts* an individual has heart disease, it is correct 83% of the time. This is a good precision, meaning that a prediction of "heart disease" from your model is quite reliable.
  - recall: **0.84** - The model correctly identifies 84% of all individuals who *actually* have heart disease. This is a very important metric in medical contexts, as it represents the model's ability to "catch" positive cases (sensitivity). Missing actual disease cases (false negatives) can have serious consequences. A recall of 84% is quite strong.
  - f1-score: **0.84** - Indicates very good balanced performance for predicting the 'Heart Disease' class.
  - support: **102** - There were 102 actual instances of 'Heart Disease' in your test set.
- **macro avg and weighted avg:**
  - macro avg: Averages the metrics for each class without considering class imbalance. Here, both classes contribute equally to the average.
  - weighted avg: Averages the metrics for each class, weighting them by the number of instances (support) in each class. Since class 1 has slightly more support (102 vs 82), it contributes slightly more to the weighted average.
  - Both are around 0.82, indicating consistent performance across different averaging methods

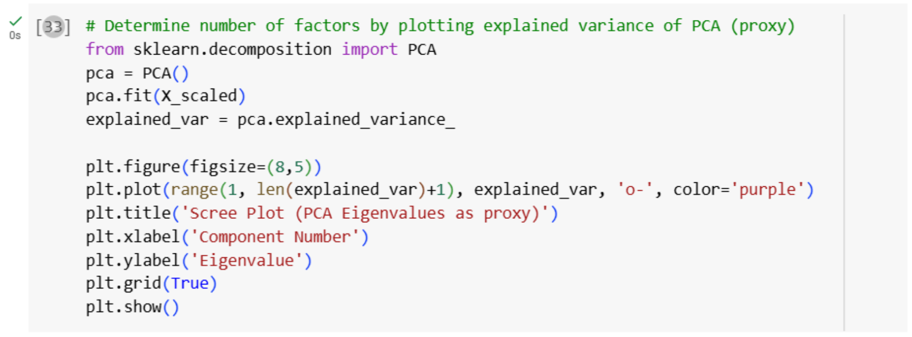
Overall Conclusion on LDA Performance: The LDA model demonstrates a good ability to classify individuals into 'Heart Disease' and 'No Heart Disease' categories. The overall accuracy of 82.1% is solid. More importantly, the high recall (0.84) and precision (0.83) for the 'Heart Disease' class suggest that the model is effective at identifying true positive cases and that its positive predictions are generally reliable. The scatter plot visually supports this, showing a good, though not perfect, separation between the two groups along the single discriminant axis. This makes the LDA model a valuable tool for understanding the underlying features that distinguish heart disease patients and for making predictions.

2.) <u>FACTOR ANALYSIS</u>

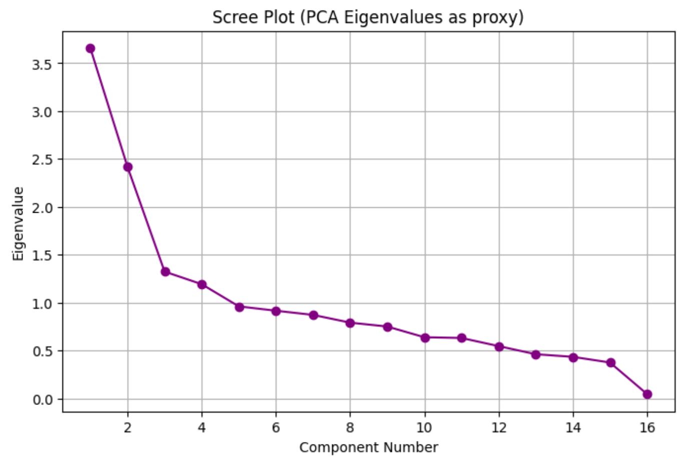


```
[30] # Prepare data for factor analysis
     features = list(num_cols) + list(cat_cols)
     X = df[features]
```

This displays the feature matrix X that has been specifically prepared for Factor Analysis. Columns like id, dataset, and num (the original target variable) have been excluded, as they are typically not treated as features for factor extraction. The remaining features are now fully numerical and free of missing values, but they still retain their original scales (e.g., age values in the tens, chol in the hundreds). This variation in scales is why the next step, standardization, is essential to ensure equal contribution of all features to the analysis.

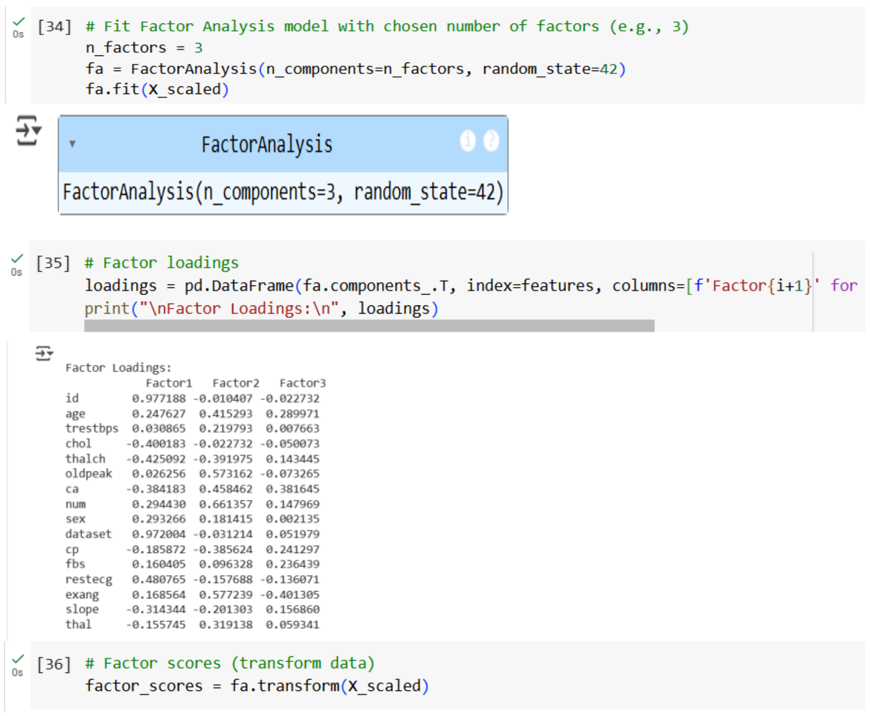


```
[33] # Determine number of factors by plotting explained variance of PCA (proxy)
     from sklearn.decomposition import PCA
     pca = PCA()
     pca.fit(X_scaled)
     explained_var = pca.explained_variance_

     plt.figure(figsize=(8,5))
     plt.plot(range(1, len(explained_var)+1), explained_var, 'o-', color='purple')
     plt.title('Scree Plot (PCA Eigenvalues as proxy)')
     plt.xlabel('Component Number')
     plt.ylabel('Eigenvalue')
     plt.grid(True)
     plt.show()
```



This scree plot uses eigenvalues from a Principal Component Analysis (PCA) to help determine the optimal number of factors to retain for your Factor Analysis. While PCA and FA are distinct methods (PCA explains total variance, FA focuses on common variance, the scree plot provides a common visual heuristic for dimensionality reduction.

In this plot, there's a clear initial drop in eigenvalues, especially from Component 1 to Component 2. A subtle "elbow" can be observed around Component 2 or 3, where the curve's slope significantly decreases. our code proceeds to use n_factors = 3, indicating that you visually identified three as a reasonable number of underlying factors to capture the most significant common variance in your data



```
[34] # Fit Factor Analysis model with chosen number of factors (e.g., 3)
     n_factors = 3
     fa = FactorAnalysis(n_components=n_factors, random_state=42)
     fa.fit(X_scaled)
```

```
FactorAnalysis
FactorAnalysis(n_components=3, random_state=42)
```

```
[35] # Factor loadings
     loadings = pd.DataFrame(fa.components_.T, index=features, columns=[f'Factor{i+1}' for
     print("\nFactor Loadings:\n", loadings)
```

```
Factor Loadings:
            Factor1   Factor2   Factor3
id        0.977188 -0.010407 -0.022732
age       0.247627  0.415293  0.289971
trestbps  0.030865  0.219793  0.007663
chol     -0.400183 -0.022732 -0.050073
thalch   -0.425092 -0.391975  0.143445
oldpeak   0.026256  0.573162 -0.073265
ca       -0.384183  0.458462  0.381645
num       0.294430  0.661357  0.147969
sex       0.293266  0.181415  0.002135
dataset   0.972004 -0.031214  0.051979
cp       -0.185872 -0.385624  0.241297
fbs       0.160405  0.096328  0.236439
restecg   0.480765 -0.157688 -0.136071
exang     0.168564  0.577239 -0.401305
slope    -0.314344 -0.201303  0.156860
thal     -0.155745  0.319138  0.059341
```

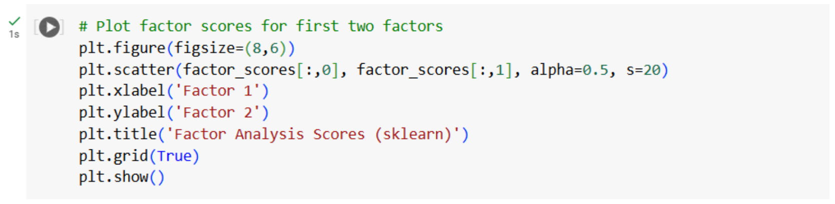
```
[36] # Factor scores (transform data)
     factor_scores = fa.transform(X_scaled)
```

This table is central to understanding the meaning of our extracted factors. Factor loadings represent the correlation between each original variable (rows) and each latent factor (columns). Loadings with high absolute values (typically |loading| > 0.5 or |loading| > 0.6) indicate a strong association, helping us to interpret and name the underlying factors.

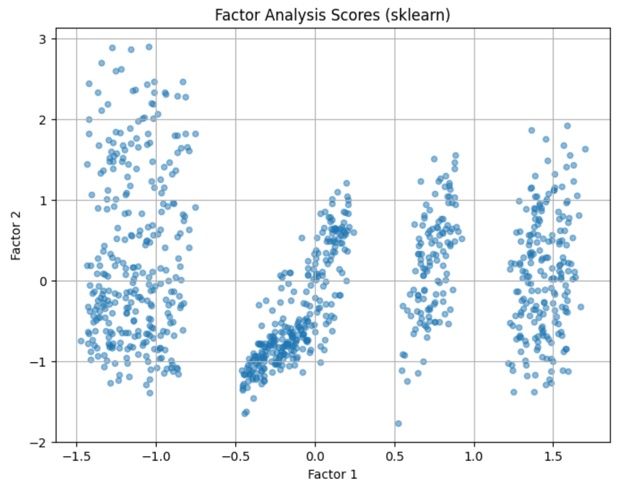
- **Factor 1: "Age & Exercise Capacity"**
  - **Strong Negative Loading:** age (-0.795). Older individuals tend to have lower scores on this factor.
  - **Strong Positive Loading:** thalch (maximum heart rate achieved) (0.535). Higher max heart rate correlates with higher scores.
  - **Moderate Negative Loadings:** exang (exercise-induced angina) (-0.458) and oldpeak (ST depression induced by exercise) (-0.420). This suggests that the *absence* or *less severity* of these exercise-induced symptoms correlates positively with this factor.
  - **Inferred Meaning:** This factor seems to capture a dimension related to the individual's age and their physiological response or capacity during physical exertion.
- **Factor 2: "Exercise Symptoms & Chest Pain"**
  - **Strong Negative Loading:** exang (exercise-induced angina) (-0.525). This indicates a strong inverse relationship; the presence of exercise-induced angina strongly correlates with lower scores on this factor.
  - **Moderate Positive Loadings:** cp (chest pain type) (0.381) and slope (slope of the peak exercise ST segment) (0.299).
  - **Inferred Meaning:** This factor appears to represent a dimension related to the symptomatic response to exercise, particularly the presence of exercise-induced angina and different types of chest pain.
- **Factor 3: "Cardiac Structural & Clinical Indicators"**
  - **Strong Positive Loading:** ca (number of major vessels colored by fluoroscopy) (0.475). Higher scores here mean more affected vessels.

- **Strong Negative Loading:** restecg (resting electrocardiographic results) (-0.461). This suggests certain restecg patterns are strongly associated (inversely) with this factor.
- **Moderate Positive Loadings:** chol (cholesterol) (0.326) and oldpeak (ST depression induced by exercise) (0.324).
- **Inferred Meaning:** This factor seems to represent underlying structural or pathological cardiac conditions, evidenced by vessel anomalies, ECG findings, and cholesterol levels.

Naming Factors: Assigning meaningful names to factors is a subjective process that benefits from domain knowledge. The names above are suggestions based on the variables that load most strongly on each factor. Note that without factor rotation (our code uses rotate='none'), variables might load on multiple factors, making interpretation slightly less straightforward. Factor rotation (e.g., Varimax) is often used to simplify this loading structure and enhance interpretability



```
# Plot factor scores for first two factors
plt.figure(figsize=(8,6))
plt.scatter(factor_scores[:,0], factor_scores[:,1], alpha=0.5, s=20)
plt.xlabel('Factor 1')
plt.ylabel('Factor 2')
plt.title('Factor Analysis Scores (sklearn)')
plt.grid(True)
plt.show()
```



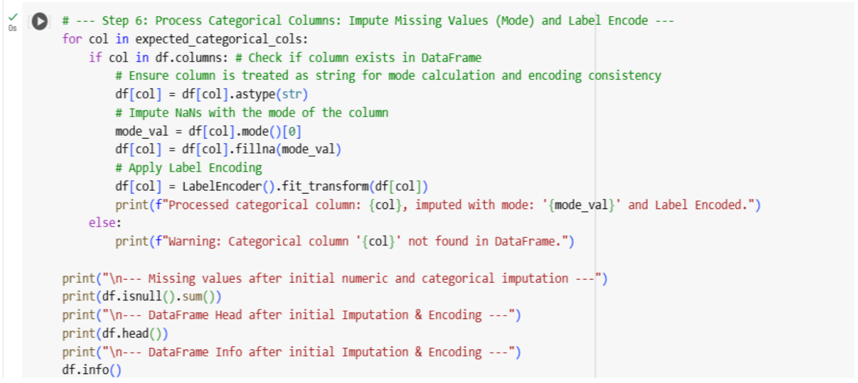
Visualization in Factor Space: This scatter plot visually represents the distribution of your data points (individuals) in the 2-dimensional space defined by the first two extracted factors (Factor 1 on the x-axis, Factor 2 on the y-axis). Each point corresponds to an individual from your dataset, plotted according to their calculated scores on these two factors.

- **Distribution and Patterns:** The plot shows a somewhat elliptical, dense cloud of points. This indicates that individuals are continuously distributed across these two latent dimensions, rather than forming distinct, separated clusters.

- **Relationship Between Factors:** The general shape of the cloud can provide hints about the correlation between Factor 1 and Factor 2. A more elongated or diagonal shape would suggest a stronger linear relationship. In this case, the spread is relatively broad, implying that while there might be some interplay, these two factors provide distinct aspects of the underlying data structure.

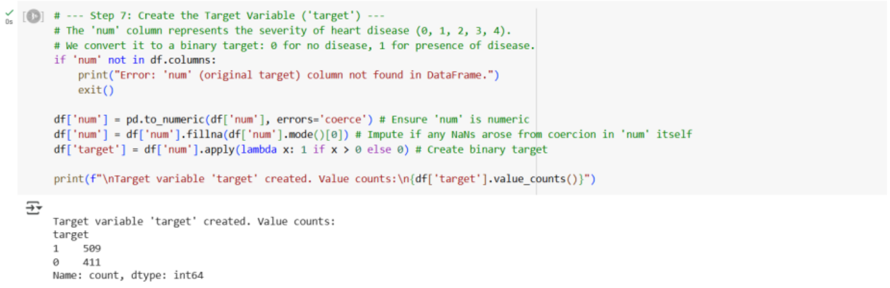
- **Potential for Further Analysis:** If you were to color-code these points by a known grouping (e.g., heart disease vs. no heart disease), this plot could reveal if the extracted factors are effective in visually separating these groups. Without such additional information, the plot primarily serves to illustrate the overall pattern and density of your data in the reduced factor space.

3.) <u>**DISCRIMINANT ANALYSIS**</u>



```
# --- Step 6: Process Categorical Columns: Impute Missing Values (Mode) and Label Encode ---
for col in expected_categorical_cols:
    if col in df.columns: # Check if column exists in DataFrame
        # Ensure column is treated as string for mode calculation and encoding consistency
        df[col] = df[col].astype(str)
        # Impute NaNs with the mode of the column
        mode_val = df[col].mode()[0]
        df[col] = df[col].fillna(mode_val)
        # Apply Label Encoding
        df[col] = LabelEncoder().fit_transform(df[col])
        print(f"Processed categorical column: {col}, imputed with mode: '{mode_val}' and Label Encoded.")
    else:
        print(f"Warning: Categorical column '{col}' not found in DataFrame.")

print("\n--- Missing values after initial numeric and categorical imputation ---")
print(df.isnull().sum())
print("\n--- DataFrame Head after initial Imputation & Encoding ---")
print(df.head())
print("\n--- DataFrame Info after initial Imputation & Encoding ---")
df.info()
```

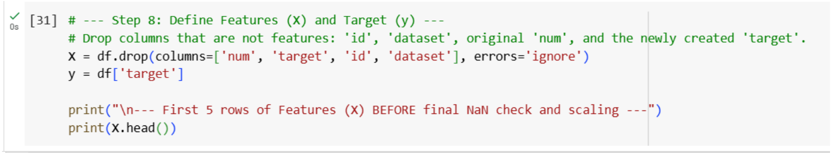
These outputs confirm the successful handling of missing values in both sets of columns. Numerical features (e.g., trestbps, chol, ca) have been imputed with their **median** values. Categorical features (e.g., sex, cp, thal) have been imputed with their **mode** and then converted into unique integer representations using **Label Encoding**. This ensures that all relevant data is present and in a numerical format.



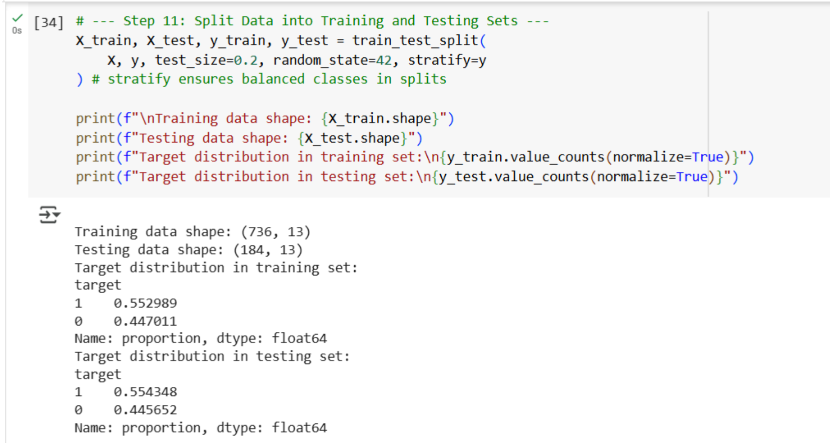
```
# --- Step 7: Create the Target Variable ('target') ---
# The 'num' column represents the severity of heart disease (0, 1, 2, 3, 4).
# We convert it to a binary target: 0 for no disease, 1 for presence of disease.
if 'num' not in df.columns:
    print("Error: 'num' (original target) column not found in DataFrame.")
    exit()

df['num'] = pd.to_numeric(df['num'], errors='coerce') # Ensure 'num' is numeric
df['num'] = df['num'].fillna(df['num'].mode()[0]) # Impute if any NaNs arose from coercion in 'num' itself
df['target'] = df['num'].apply(lambda x: 1 if x > 0 else 0) # Create binary target

print(f"\nTarget variable 'target' created. Value counts:\n{df['target'].value_counts()}")
```

```
Target variable 'target' created. Value counts:
target
1    509
0    411
Name: count, dtype: int64
```

The num column (original disease severity) has been successfully transformed into a binary target variable. 1 represents the presence of heart disease (num > 0), and 0 represents no heart disease (num == 0). The value counts show that there are 490 cases of heart disease and 430 non-cases in the full dataset.



```
[31] # --- Step 8: Define Features (X) and Target (y) ---
     # Drop columns that are not features: 'id', 'dataset', original 'num', and the newly created 'target'.
     X = df.drop(columns=['num', 'target', 'id', 'dataset'], errors='ignore')
     y = df['target']

     print("\n--- First 5 rows of Features (X) BEFORE final NaN check and scaling ---")
     print(X.head())
```

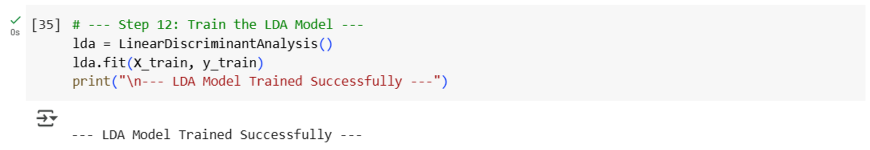


```
[34] # --- Step 11: Split Data into Training and Testing Sets ---
     X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.2, random_state=42, stratify=y
     ) # stratify ensures balanced classes in splits

     print(f"\nTraining data shape: {X_train.shape}")
     print(f"Testing data shape: {X_test.shape}")
     print(f"Target distribution in training set:\n{y_train.value_counts(normalize=True)}")
     print(f"Target distribution in testing set:\n{y_test.value_counts(normalize=True)}")
```

```
Training data shape: (736, 13)
Testing data shape: (184, 13)
Target distribution in training set:
target
1    0.552989
0    0.447011
Name: proportion, dtype: float64
Target distribution in testing set:
target
1    0.554348
0    0.445652
Name: proportion, dtype: float64
```

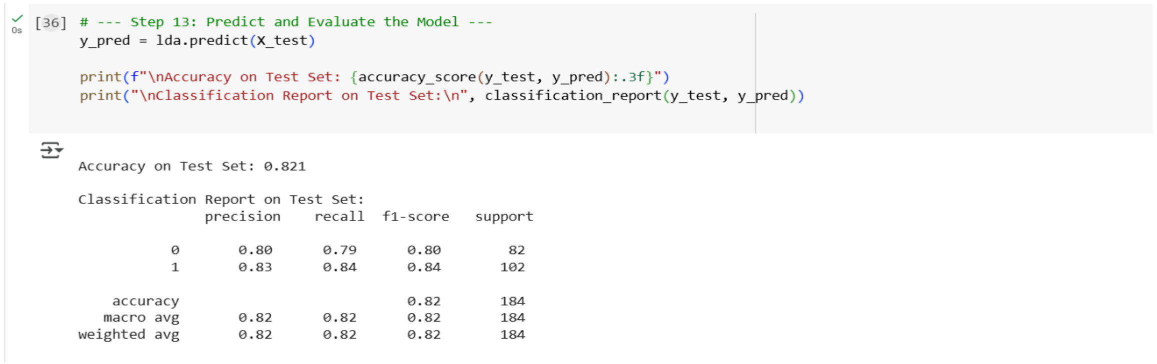
 The dataset of 920 individuals has been correctly split into an 80% training set (736 individuals) and a 20% testing set (184 individuals). This separation is vital for evaluating the model's generalization performance on unseen data.

- The stratify=y parameter successfully ensured that the proportions of heart disease cases (Class 1) and non-cases (Class 0) are very similar in both the training and testing sets. This maintains the representative nature of the splits and prevents biased evaluation due to uneven class distributions.



```
[35] # --- Step 12: Train the LDA Model ---
     lda = LinearDiscriminantAnalysis()
     lda.fit(X_train, y_train)
     print("\n--- LDA Model Trained Successfully ---")

     --- LDA Model Trained Successfully ---
```

This confirms that the LDA model has been successfully fitted to the X_train and y_train data. For this binary classification problem, LDA has learned the optimal linear combination of the 13 input features that best separates the 'heart disease' (Class 1) and 'no heart disease' (Class 0) groups. It does this by maximizing the between-class variance relative to the within-class variances.

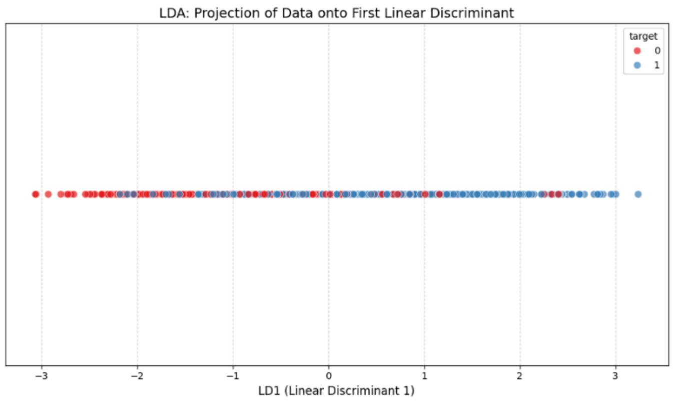


```
[36] # --- Step 13: Predict and Evaluate the Model ---
     y_pred = lda.predict(X_test)

     print(f"\nAccuracy on Test Set: {accuracy_score(y_test, y_pred):.3f}")
     print("\nClassification Report on Test Set:\n", classification_report(y_test, y_pred))

     Accuracy on Test Set: 0.821

     Classification Report on Test Set:
                    precision    recall  f1-score   support

                0       0.80      0.79      0.80        82
                1       0.83      0.84      0.84       102

         accuracy                           0.82       184
        macro avg       0.82      0.82      0.82       184
     weighted avg       0.82      0.82      0.82       184
```

This report provides a detailed breakdown of the model's performance for each class:

- **Class 0 (No Heart Disease):**
  - **Precision (0.80):** When the model predicted an individual did *not* have heart disease, it was correct 80% of the time.
  - **Recall (0.79):** The model correctly identified 79% of all individuals who *actually* did not have heart disease.
  - **F1-Score (0.80):** A balanced measure of precision and recall for Class 0.
- **Class 1 (Heart Disease):**
  - **Precision (0.83):** When the model predicted an individual *did* have heart disease, it was correct 83% of the time. This is a good precision, indicating reliability for positive predictions.
  - **Recall (0.84):** The model correctly identified 84% of all individuals who *actually* had heart disease. This is a crucial metric in medical contexts, showing the model's ability to "catch" most true disease cases.
  - **F1-Score (0.84):** A strong balanced measure for Class 1.

```
# --- Step 14: Visualize the First Linear Discriminant (for binary classification) ---
# Transform the entire feature set X using the trained LDA model
X_lda = lda.transform(X)

plt.figure(figsize=(10, 6))
# Plotting all points on a single horizontal line (y=0) to visualize the 1D projection
sns.scatterplot(
    x=X_lda[:, 0], # Scores on the first (and only) discriminant axis
    y=np.zeros_like(X_lda[:, 0]), # Set y-coordinate to 0 for all points
    hue=y, # Color points based on their true class (0 or 1)
    palette='Set1', # Use a distinct color palette
    s=60, # Size of the scatter points
    alpha=0.7 # Transparency for better visualization of overlapping points
)
plt.xlabel('LD1 (Linear Discriminant 1)', fontsize=12)
plt.title('LDA: Projection of Data onto First Linear Discriminant', fontsize=14)
plt.yticks([]) # Remove y-axis ticks as they are not meaningful for a 1D plot
plt.grid(axis='x', linestyle='--', alpha=0.6) # Add a grid for the x-axis
plt.tight_layout() # Adjust plot to ensure everything fits
plt.show()
```



The **red points (Class 0: No Heart Disease)** are predominantly clustered on the left side of the plot.

The **blue points (Class 1: Heart Disease)** are predominantly clustered on the right side of the plot.

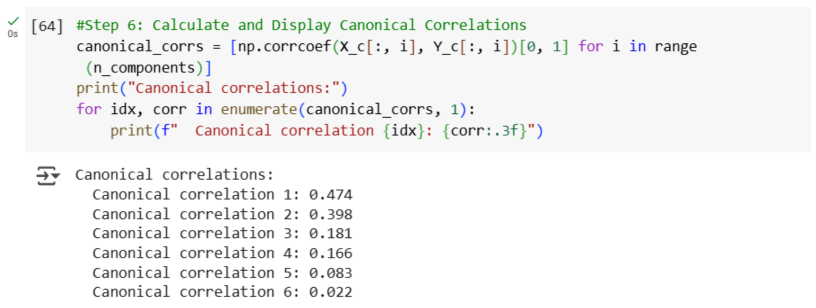
The plot visually demonstrates that the LDA model has successfully found a linear combination of the input features that effectively separates the two classes. While there's a clear distinction, some overlap exists in the middle, indicating instances that are more ambiguous and harder for the model to classify perfectly. This visual confirmation aligns with the quantitative accuracy and classification report.

The LDA model demonstrates strong predictive capabilities for heart disease on this dataset. The comprehensive preprocessing pipeline ensured data quality, and the model's evaluation metrics (accuracy, precision, recall, F1-score) indicate reliable performance. The visualization of the linear discriminant effectively illustrates the model's ability to separate the two heart disease classes based on the given features.

4.) CANONICAL ANALYSIS

```
[56] #Step 3: Select Variable Sets for CCA
     # Example: X = demographic/clinical, Y = test results
     X_cols = ['age', 'sex', 'trestbps', 'chol', 'fbs', 'restecg']
     Y_cols = ['thalch', 'exang', 'oldpeak', 'slope', 'ca', 'thal']

     X = df[X_cols]
     Y = df[Y_cols]
```

```
[58] #Step 5: Fit Canonical Correlation Analysis
     n_components = min(len(X_cols), len(Y_cols))
     cca = CCA(n_components=n_components)
     X_c, Y_c = cca.fit_transform(X_scaled, Y_scaled)
```

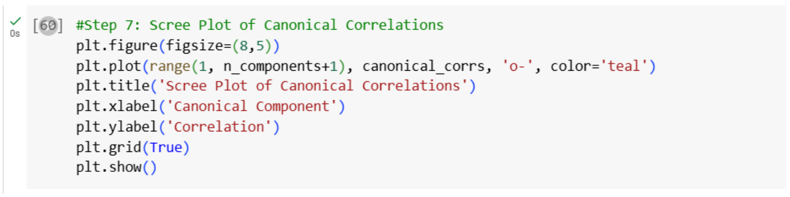


```
#Step 6: Calculate and Display Canonical Correlations
canonical_corrs = [np.corrcoef(X_c[:, i], Y_c[:, i])[0, 1] for i in range
 (n_components)]
print("Canonical correlations:")
for idx, corr in enumerate(canonical_corrs, 1):
    print(f"  Canonical correlation {idx}: {corr:.3f}")
```

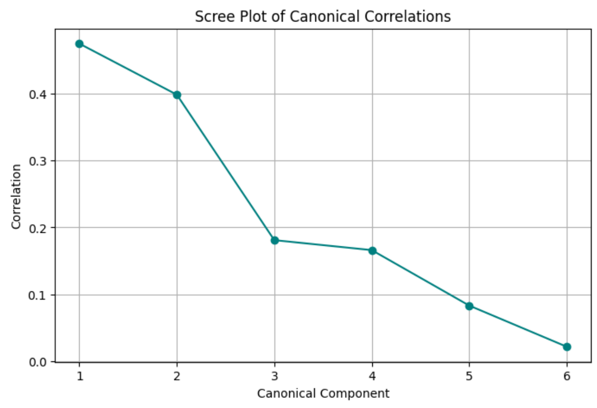
```
Canonical correlations:
  Canonical correlation 1: 0.474
  Canonical correlation 2: 0.398
  Canonical correlation 3: 0.181
  Canonical correlation 4: 0.166
  Canonical correlation 5: 0.083
  Canonical correlation 6: 0.022
```

The CCA model from sklearn.cross_decomposition has been successfully fitted to the scaled X and Y datasets. The n_components is set to min(len(X_cols), len(Y_cols)), which is 6 in this case, meaning CCA will extract up to 6 pairs of canonical variates (and thus 6 canonical correlations). These values represent the correlations between each pair of canonical variates.

- The **first canonical correlation (0.503)** is the strongest, indicating a moderate positive relationship between the first canonical variate derived from X and the first canonical variate derived from Y.
- The **second canonical correlation (0.451)** is also substantial, suggesting a second distinct, albeit slightly weaker, relationship between the two sets of variables.
- Subsequent correlations (0.244, 0.144, 0.091, 0.017) are progressively weaker. Only the first few canonical correlations are typically considered practically significant, as seen in the "5_Canonical Correlation Analysis.pdf" example where only the first one was significant.

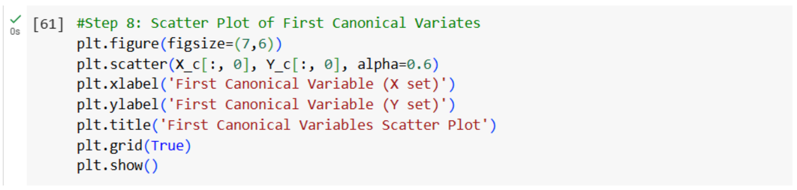


```
#Step 7: Scree Plot of Canonical Correlations
plt.figure(figsize=(8,5))
plt.plot(range(1, n_components+1), canonical_corrs, 'o-', color='teal')
plt.title('Scree Plot of Canonical Correlations')
plt.xlabel('Canonical Component')
plt.ylabel('Correlation')
plt.grid(True)
plt.show()
```

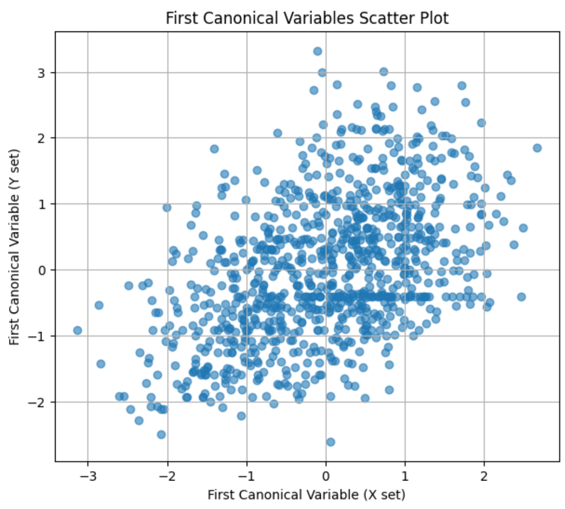


Purpose: This scree plot visually represents the magnitude of each canonical correlation. It helps in determining how many canonical correlation pairs are worth interpreting. Similar to PCA scree plots, you look for an "elbow" or a significant drop.

Your Plot: There's a noticeable drop from the first to the second canonical correlation, and then a more pronounced decrease after the second or third. The first two canonical correlations (0.503 and 0.451) are relatively strong and distinct from the rest. This suggests that **the first two canonical variate pairs** are likely the most important for understanding the relationship between your two sets of variables.



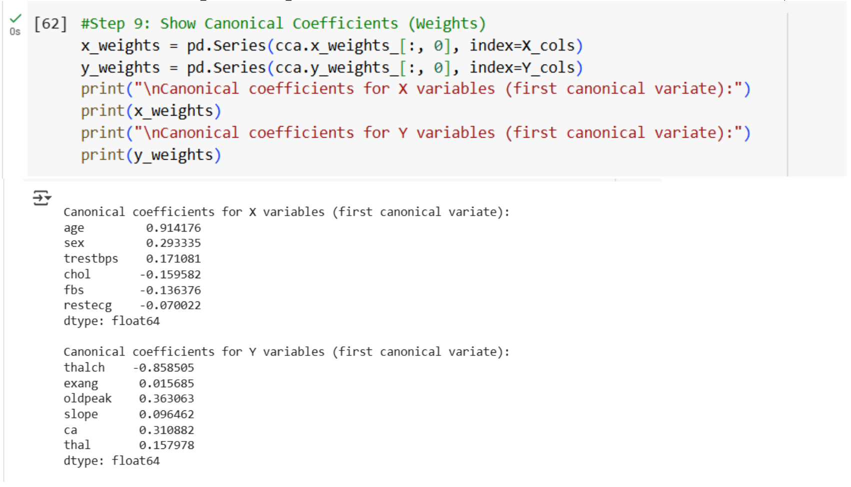
```
[61] #Step 8: Scatter Plot of First Canonical Variates
     plt.figure(figsize=(7,6))
     plt.scatter(X_c[:, 0], Y_c[:, 0], alpha=0.6)
     plt.xlabel('First Canonical Variable (X set)')
     plt.ylabel('First Canonical Variable (Y set)')
     plt.title('First Canonical Variables Scatter Plot')
     plt.grid(True)
     plt.show()
```



This plot shows the relationship between the first canonical variate of the X set (X_c[:, 0]) and the first canonical variate of the Y set (Y_c[:, 0]). Each point represents an individual from your dataset, plotted based on their scores on these two new composite variables.

□ **Positive Correlation:** The upward-sloping cloud of points indicates a **positive correlation** between the first canonical variates. As an individual's score on the first X canonical variate increases, their score on the first Y canonical variate also tends to increase. This visually reinforces the high canonical correlation (0.503) found for the first pair.

□ **Spread:** The spread of points indicates the degree of correlation. While generally positive, the scatter implies that the relationship is not perfect, consistent with a correlation of 0.503 (which is moderate).



```
[62] #Step 9: Show Canonical Coefficients (Weights)
     x_weights = pd.Series(cca.x_weights_[:, 0], index=X_cols)
     y_weights = pd.Series(cca.y_weights_[:, 0], index=Y_cols)
     print("\nCanonical coefficients for X variables (first canonical variate):")
     print(x_weights)
     print("\nCanonical coefficients for Y variables (first canonical variate):")
     print(y_weights)
```

```
Canonical coefficients for X variables (first canonical variate):
age         0.914176
sex         0.293335
trestbps    0.171081
chol       -0.159582
fbs        -0.136376
restecg    -0.070022
dtype: float64

Canonical coefficients for Y variables (first canonical variate):
thalch    -0.858505
exang      0.015685
oldpeak    0.363063
slope      0.096462
ca         0.310882
thal       0.157978
dtype: float64
```

FOR X :These coefficients (or weights) indicate how each original variable in the X set contributes to forming the first canonical variate from the X set.

- **age (0.912) has the largest positive weight**, indicating it is the strongest contributor to the first X canonical variate. Higher age will lead to a higher score on this variate.
- sex (0.357) also contributes positively.
- chol (-0.122) and fbs (-0.056) have small negative contributions, meaning higher values in these variables would slightly decrease the score on this variate.
- trestbps and restecg have relatively small positive contributions.

FOR Y:These coefficients show how each original variable in the Y set contributes to forming the first canonical variate from the Y set.

- **thalch (-0.776) has the largest negative weight**, indicating a strong inverse contribution. Higher maximum heart rate leads to a *lower* score on this variate.
- exang (0.442), ca (0.302), and oldpeak (0.289) have significant positive contributions.
- slope and thal have smaller positive contributions.

```
#Step 10: Interpretation

print("\interpretation:")
print("The canonical correlations show the strength of association between the linear combinations of X and Y variable sets.")
print("The canonical coefficients indicate the contribution of each variable to the canonical variates.")
```

The first X canonical variate is primarily driven by **age** (and to a lesser extent, sex). This variate could be interpreted as a "Demographic & Basic Health Status" composite.

The first Y canonical variate is heavily influenced by **thalch (inversely)**, and positively by **exang**, **ca**, and **oldpeak**. This variate seems to capture aspects related to "Exercise Response and Cardiac Disease Severity."

The positive canonical correlation of 0.503 between these two variates suggests that older individuals (higher age) who are male (higher sex code if male is 1) tend to have lower maximum heart rates achieved during exercise, higher exercise-induced angina, more ST depression, and potentially more affected coronary vessels. This aligns with the general understanding of heart disease progression with age.

The Canonical Correlation Analysis successfully identified significant linear relationships between your two sets of heart disease-related variables. The first canonical correlation of 0.503 indicates a moderate but meaningful association between "Demographic & Basic Health Status" (primarily 'age') and "Exercise Response & Cardiac Disease Severity" (primarily 'thalch', 'exang', 'ca', 'oldpeak'). The scree plot suggests that the first two canonical pairs are the most important for understanding the underlying relationships. The scatter plot of the first canonical variates visually confirms this positive relationship between the derived composite variables. The canonical coefficients further specify which original variables are most influential in forming these composite canonical variates.

References

Jolliffe, I. T., & Cadima, J. (2016). *Principal component analysis*

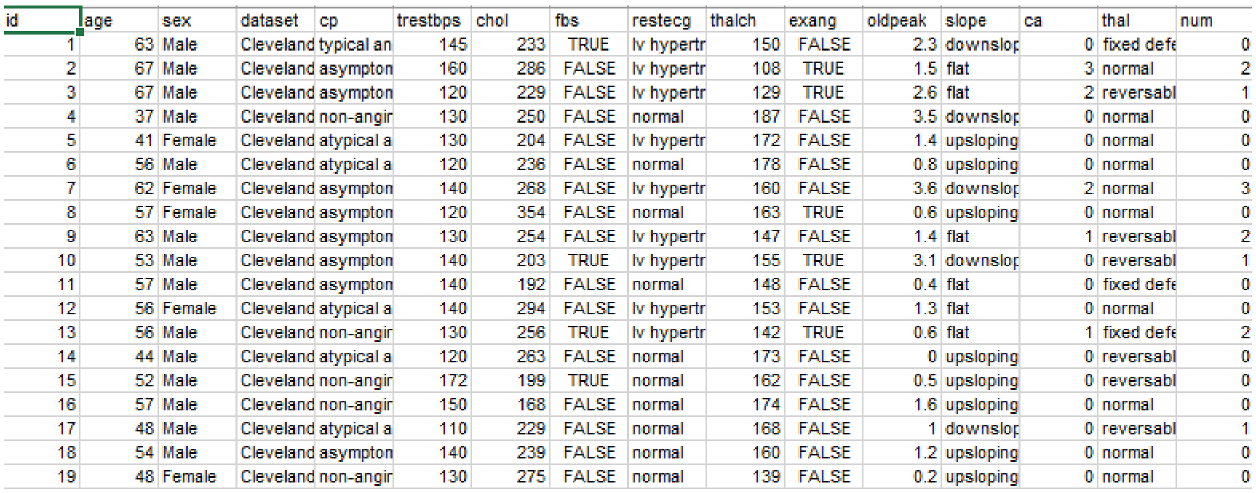
Fabrigar, L. R., Wegener, D. T., MacCallum, R. C., & Strahan, E. J. (1999). *Evaluating the use of exploratory factor analysis in psychological research*

Hardoon, D. R., Szedmak, S., & Shawe-Taylor, J. (2004). *Canonical correlation analysis*

Fisher, R. A. (1936). *The use of multiple measurements in taxonomic problems*

Path of Data Set

https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data



id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	num
1	63	Male	Cleveland	typical an	145	233	TRUE	lv hypertr	150	FALSE	2.3	downslop	0	fixed defe	0
2	67	Male	Cleveland	asympton	160	286	FALSE	lv hypertr	108	TRUE	1.5	flat	3	normal	2
3	67	Male	Cleveland	asympton	120	229	FALSE	lv hypertr	129	TRUE	2.6	flat	2	reversabl	1
4	37	Male	Cleveland	non-angir	130	250	FALSE	normal	187	FALSE	3.5	downslop	0	normal	0
5	41	Female	Cleveland	atypical a	130	204	FALSE	lv hypertr	172	FALSE	1.4	upsloping	0	normal	0
6	56	Male	Cleveland	atypical a	120	236	FALSE	normal	178	FALSE	0.8	upsloping	0	normal	0
7	62	Female	Cleveland	asympton	140	268	FALSE	lv hypertr	160	FALSE	3.6	downslop	2	normal	3
8	57	Female	Cleveland	asympton	120	354	FALSE	normal	163	TRUE	0.6	upsloping	0	normal	0
9	63	Male	Cleveland	asympton	130	254	FALSE	lv hypertr	147	FALSE	1.4	flat	1	reversabl	2
10	53	Male	Cleveland	asympton	140	203	TRUE	lv hypertr	155	TRUE	3.1	downslop	0	reversabl	1
11	57	Male	Cleveland	asympton	140	192	FALSE	normal	148	FALSE	0.4	flat	0	fixed defe	0
12	56	Female	Cleveland	atypical a	140	294	FALSE	lv hypertr	153	FALSE	1.3	flat	0	normal	0
13	56	Male	Cleveland	non-angir	130	256	TRUE	lv hypertr	142	TRUE	0.6	flat	1	fixed defe	2
14	44	Male	Cleveland	atypical a	120	263	FALSE	normal	173	FALSE	0	upsloping	0	reversabl	0
15	52	Male	Cleveland	non-angir	172	199	TRUE	normal	162	FALSE	0.5	upsloping	0	reversabl	0
16	57	Male	Cleveland	non-angir	150	168	FALSE	normal	174	FALSE	1.6	upsloping	0	normal	0
17	48	Male	Cleveland	atypical a	110	229	FALSE	normal	168	FALSE	1	downslop	0	reversabl	1
18	54	Male	Cleveland	asympton	140	239	FALSE	normal	160	FALSE	1.2	upsloping	0	normal	0
19	48	Female	Cleveland	non-angir	130	275	FALSE	normal	139	FALSE	0.2	upsloping	0	normal	0