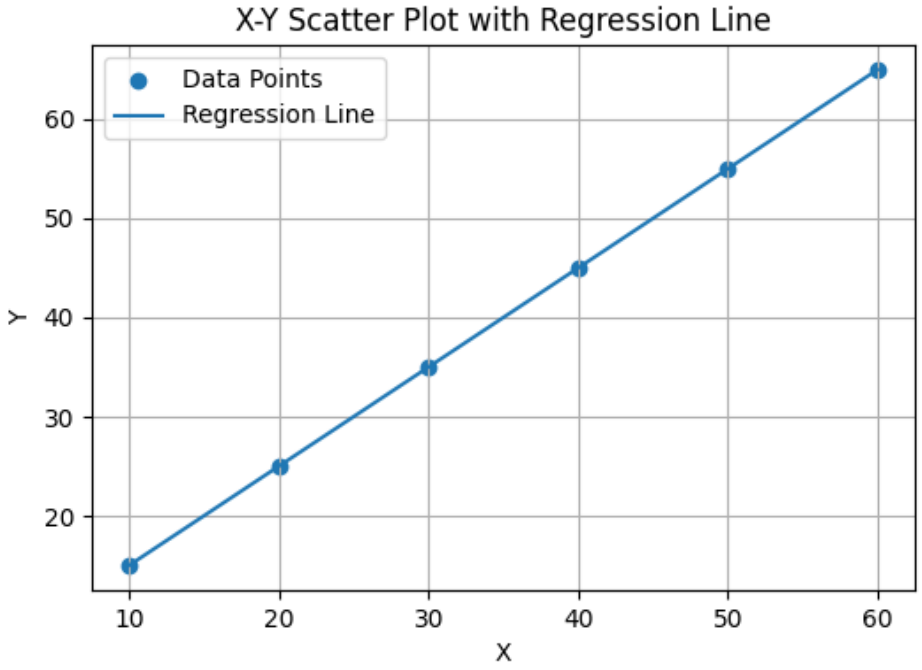
Program 9: Program to implement correlation, rank correlation and regression and plot x-y plot and heat maps of correlation matrices.

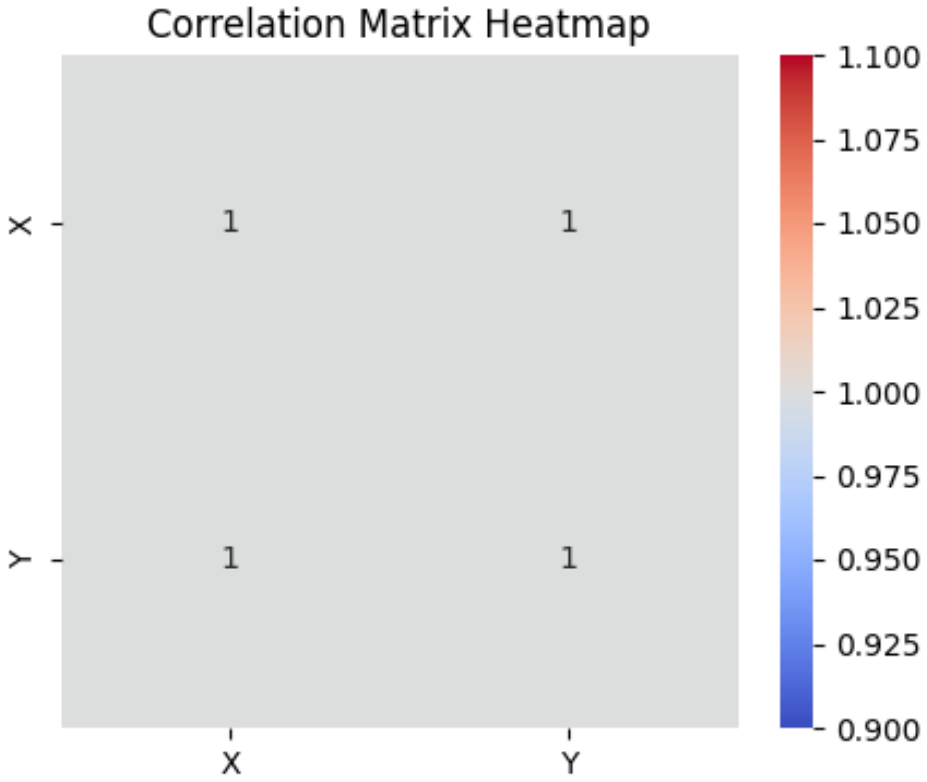
```
# Import libraries
import numpy as np
import pandas as pd
from scipy.stats import pearsonr, spearmanr
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import seaborn as sns

# Sample Dataset
x = np.array([10, 20, 30, 40, 50, 60])
y = np.array([15, 25, 35, 45, 55, 65])

df = pd.DataFrame({"X": x, "Y": y})

# ------------------------------
# 1. Pearson Correlation
# ------------------------------
pear_corr, _ = pearsonr(df["X"], df["Y"])
print("Pearson Correlation:", pear_corr)

# ------------------------------
# 2. Rank Correlation (Spearman)
# ------------------------------
rank_corr, _ = spearmanr(df["X"], df["Y"])
print("Spearman Rank Correlation:", rank_corr)

# ------------------------------
# 3. Linear Regression
# ------------------------------
X_reshaped = df[['X']]  # 2D array
model = LinearRegression()
model.fit(X_reshaped, df['Y'])

print("\nRegression Equation: Y = {:.2f}X + {:.2f}".format(model.coef_[0], model.intercept_))

# ------------------------------
# 4. XY Scatter Plot
# ------------------------------
plt.figure(figsize=(6, 4))
plt.scatter(df["X"], df["Y"], label="Data Points")
plt.plot(df["X"], model.predict(X_reshaped), label="Regression Line")
plt.xlabel("X")
plt.ylabel("Y")
plt.title("X-Y Scatter Plot with Regression Line")
plt.legend()
plt.grid(True)
plt.show()

# ------------------------------
# 5. Heatmap of Correlation Matrix
# ------------------------------
plt.figure(figsize=(5, 4))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
```

```
plt.title("Correlation Matrix Heatmap")
plt.show()
```

output:
Pearson Correlation: 0.9999999999999999
Spearman Rank Correlation: 1.0

Regression Equation: Y = 1.00X + 5.00





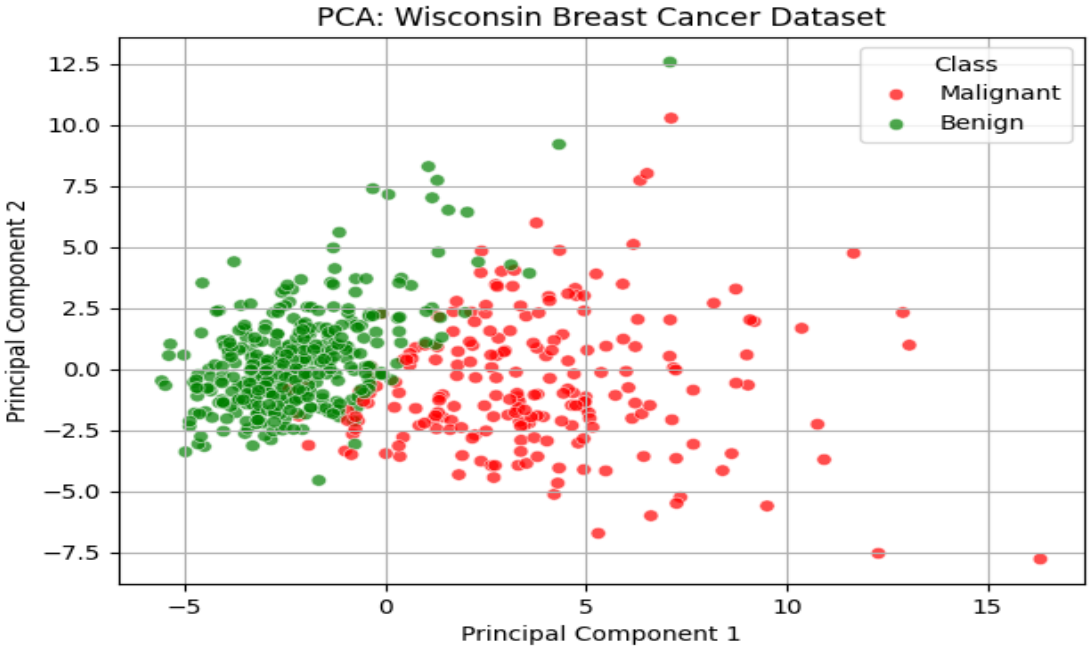
Program 10: Program to implement PCA for Wisconsin dataset, visualize and analyze the results.

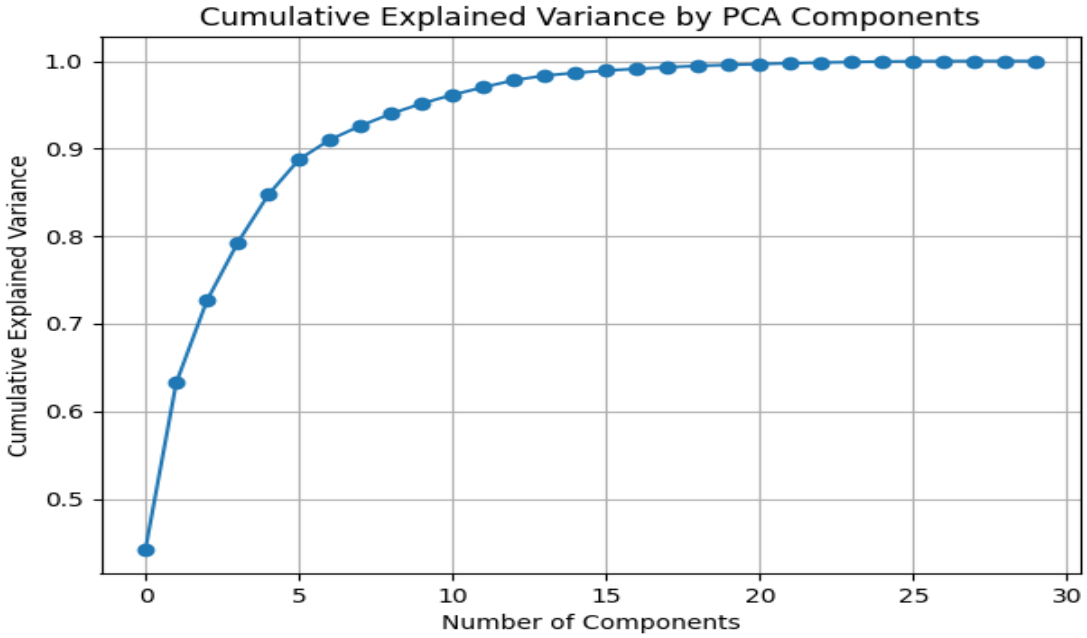
```
# PCA on Wisconsin Breast Cancer Dataset
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# -----------------------------------
# 1. Load Wisconsin Dataset
# -----------------------------------
data = load_breast_cancer()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)  # 0 = malignant, 1 = benign

print("Shape of Dataset:", X.shape)

# -----------------------------------
# 2. Standardize the Data
# -----------------------------------
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# -----------------------------------
# 3. Apply PCA (2 components)
# -----------------------------------
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("\nExplained Variance Ratio:")
print(pca.explained_variance_ratio_)

# -----------------------------------
# 4. Convert to DataFrame
# -----------------------------------
pca_df = pd.DataFrame(data=X_pca, columns=["PC1", "PC2"])
pca_df["Class"] = y.map({0: "Malignant", 1: "Benign"})

# -----------------------------------
# 5. Scatter Plot of PCA Components
# -----------------------------------
plt.figure(figsize=(7, 5))
sns.scatterplot(
    x="PC1", y="PC2", hue="Class",
    palette=["red", "green"], data=pca_df, alpha=0.7
)
plt.title("PCA: Wisconsin Breast Cancer Dataset")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.show()

# -----------------------------------
# 6. Explained Variance Plot
```

```
# ----------------------------------
pca_full = PCA()
pca_full.fit(X_scaled)

plt.figure(figsize=(7, 5))
plt.plot(np.cumsum(pca_full.explained_variance_ratio_), marker='o')
plt.title("Cumulative Explained Variance by PCA Components")
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.grid(True)
plt.show()
```

output:
Shape of Dataset: (569, 30)
Explained Variance Ratio:

[0.44272026
0.18971182]





Program 11: Program to implement the working of linear discriminant analysis using iris dataset and visualize the results.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasetsfrom sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler
# ------------------------------------------------
# 1. Load the Iris Dataset
# ------------------------------------------------
iris = datasets.load_iris()
X = iris.data       # features
y = iris.target     # labels
target_names = iris.target_names
print("Iris dataset shape:", X.shape)
# ------------------------------------------------
# 2. Feature Scaling (recommended for LDA)
# ------------------------------------------------
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# ------------------------------------------------
# 3. Apply Linear Discriminant Analysis (LDA)
# ------------------------------------------------
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)
print("Transformed LDA shape:", X_lda.shape)
# ------------------------------------------------
# 4. Visualization of LDA Components
# ------------------------------------------------
plt.figure(figsize=(10, 6))
colors = ['red', 'green', 'blue']
for color, i, target in zip(colors, [0, 1, 2], target_names):
    plt.scatter(X_lda[y == i, 0], X_lda[y == i, 1], alpha=0.7,color=color,label=target)
plt.xlabel("LDA Component 1")
plt.ylabel("LDA Component 2")
plt.title("Linear Discriminant Analysis on Iris Dataset")
plt.legend()
plt.grid(True)
plt.show()
# ------------------------------------------------
# 5. Explained Variance Ratio
# ------------------------------------------------
print("\nExplained variance ratio:")
print(lda.explained_variance_ratio_)
```

program 12: Program to Implement multiple linear regression using iris dataset, visualize and analyze the results.

```
# ============================================================
# Multiple Linear Regression on Iris Dataset
# Predict Petal Length using other Iris features
# Visualization + Analysis
# ============================================================

import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error, r2_score

# ------------------------------------------------------------
# 1. Load Iris Dataset
# ------------------------------------------------------------
iris = datasets.load_iris()
X = iris.data[:, [0, 1, 3]]
# Using Sepal length, Sepal width, Petal width as predictors
y = iris.data[:, 2]
# Predicting Petal Length

print("Feature shape:", X.shape)
print("Target shape:", y.shape)

# ------------------------------------------------------------
# 2. Train-Test Split
# ------------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=42
)

# ------------------------------------------------------------
# 3. Multiple Linear Regression Model
# ------------------------------------------------------------
model = LinearRegression()
model.fit(X_train, y_train)

# ------------------------------------------------------------
# 4. Predictions
# ------------------------------------------------------------
y_pred = model.predict(X_test)

# ------------------------------------------------------------
# 5. Model Analysis
# ------------------------------------------------------------
print("\n=========== Regression Results ==========")
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# ------------------------------------------------------------
# 6. Visualization: Actual vs Predicted
# ------------------------------------------------------------
plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)],
        [min(y_test), max(y_test)],
        linewidth=2)

plt.xlabel("Actual Petal Length")
plt.ylabel("Predicted Petal Length")
plt.title("Actual vs Predicted Petal Length (Multiple Linear Regression)")
plt.grid(True)
plt.show()

# ------------------------------------------------------------
```

```
# 7. Residual Plot
# ------------------------------------------------------------
residuals = y_test - y_pred

plt.figure(figsize=(10, 5))
plt.scatter(y_pred, residuals)
plt.axhline(0, color='black', linewidth=2)

plt.xlabel("Predicted Petal Length")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.grid(True)
plt.show()
```