2348441_lab_06

April 1, 2024

Lab Exercise 6 -Multi Dimensional Scaling (MDS)

Created by : Nileem Kaveramma C C | 2348441 Created DATE:30-03-2024 Edited Date: 01-04-2024

IMPORTED LIBRARIES

- numpy - for numerical, array, matrices (Linear Algebra) processing
- Pandas - for loading and processing datasets
- matplotlib.pyplot - For visualisation
- Saeborn - for statistical graph
- scipy.stats use a variety of statistical functions
- %matplotlib inline: Enables inline plotting in Jupyter notebooks, displaying matplotlib plots directly below the code cell.
- from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler: Imports three different scaling techniques

EMPLOYEE SALARY ANALYSIS he provided dataset captures information relevant to employee salary prediction, encompassing various attributes such as age, gender, education level, job title, years of experience, and salary. With a diverse set of features, the dataset offers valuable insights into the characteristics of individuals within an organizational context. This dataset becomes particularly relevant for exploring patterns and relationships that could contribute to predicting employee salaries. Through descriptive statistics, visualizations, and parametric tests, analysts can discern trends, potential disparities, and factors influencing salary variations among employees.

AIM: The aim of Multi-Dimensional Scaling (MDS) in machine learning is to reduce the dimensionality of a dataset while preserving the pairwise distances between data points. By projecting high-dimensional data into a lower-dimensional space, MDS aims to reveal the underlying structure and relationships within the dataset in a more interpretable and visualizable form.

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

[ ]:
```
df = pd.read_csv('/content/Salary Data.csv')
df
```

```
[ ]:       Age  Gender Education Level                          Job Title  \
     0    32.0    Male      Bachelor's                  Software Engineer
     1    28.0  Female        Master's                       Data Analyst
     2    45.0    Male             PhD                     Senior Manager
     3    36.0  Female      Bachelor's                    Sales Associate
     4    52.0    Male        Master's                           Director
     ..    …     …            …                            …
     370  35.0  Female      Bachelor's           Senior Marketing Analyst
     371  43.0    Male        Master's             Director of Operations
     372  29.0  Female      Bachelor's             Junior Project Manager
     373  34.0    Male      Bachelor's  Senior Operations Coordinator
     374  44.0  Female             PhD            Senior Business Analyst

          Years of Experience    Salary
     0                    5.0   90000.0
     1                    3.0   65000.0
     2                   15.0  150000.0
     3                    7.0   60000.0
     4                   20.0  200000.0
     ..                    …        …
     370                  8.0   85000.0
     371                 19.0  170000.0
     372                  2.0   40000.0
     373                  7.0   90000.0
     374                 15.0  150000.0

     [375 rows x 6 columns]
```

Perform some basic EDA

df.shape - attribute is used to get the dimensions of the DataFrame.

```
[ ]: df.shape
```

```
[ ]: (375, 6)
```

df.head() method is used to display the first few rows of a DataFrame.

```
[ ]: df.head()
```

```
[ ]:     Age  Gender Education Level          Job Title  Years of Experience  \
     0  32.0    Male      Bachelor's  Software Engineer                  5.0
     1  28.0  Female        Master's       Data Analyst                  3.0
     2  45.0    Male             PhD     Senior Manager                 15.0
     3  36.0  Female      Bachelor's    Sales Associate                  7.0
     4  52.0    Male        Master's           Director                 20.0

          Salary
```

```
0     90000.0
1     65000.0
2    150000.0
3     60000.0
4    200000.0
```

df.tail() method is used to display the last few rows of a DataFrame.

[ ]:
```
df.tail()
```

[ ]:
```
      Age  Gender Education Level                      Job Title  \
370  35.0  Female      Bachelor's       Senior Marketing Analyst
371  43.0    Male        Master's         Director of Operations
372  29.0  Female      Bachelor's         Junior Project Manager
373  34.0    Male      Bachelor's  Senior Operations Coordinator
374  44.0  Female             PhD        Senior Business Analyst

     Years of Experience    Salary
370                  8.0   85000.0
371                 19.0  170000.0
372                  2.0   40000.0
373                  7.0   90000.0
374                 15.0  150000.0
```

df.columns attribute is used to retrieve the column labels or names of the DataFrame.

[ ]:
```
df.columns
```

[ ]:
```
Index(['Age', 'Gender', 'Education Level', 'Job Title', 'Years of Experience',
       'Salary'],
      dtype='object')
```

df.dtypes attribute is used to retrieve the data types of each column in a DataFrame

[ ]:
```
df.dtypes
```

[ ]:
```
Age                    float64
Gender                  object
Education Level         object
Job Title               object
Years of Experience    float64
Salary                 float64
dtype: object
```

the code df.isnull().count() in Pandas is used to count the total number of rows for each column in a DataFrame, including both missing (null or NaN) and non-missing values.

df.isnull().count()

df.info() method in Pandas provides a concise summary of a DataFrame, including information about the data types, non-null values, and memory usage

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 375 entries, 0 to 374
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Age                  373 non-null    float64
 1   Gender               373 non-null    object
 2   Education Level      373 non-null    object
 3   Job Title            373 non-null    object
 4   Years of Experience  373 non-null    float64
 5   Salary               373 non-null    float64
dtypes: float64(3), object(3)
memory usage: 17.7+ KB
```

he df.describe() method in Pandas is used to generate descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution

[ ]:
```
df.describe()
```

[ ]:

	Age	Years of Experience	Salary
count	373.000000	373.000000	373.000000
mean	37.431635	10.030831	100577.345845
std	7.069073	6.557007	48240.013482
min	23.000000	0.000000	350.000000
25%	31.000000	4.000000	55000.000000
50%	36.000000	9.000000	95000.000000
75%	44.000000	15.000000	140000.000000
max	53.000000	25.000000	250000.000000

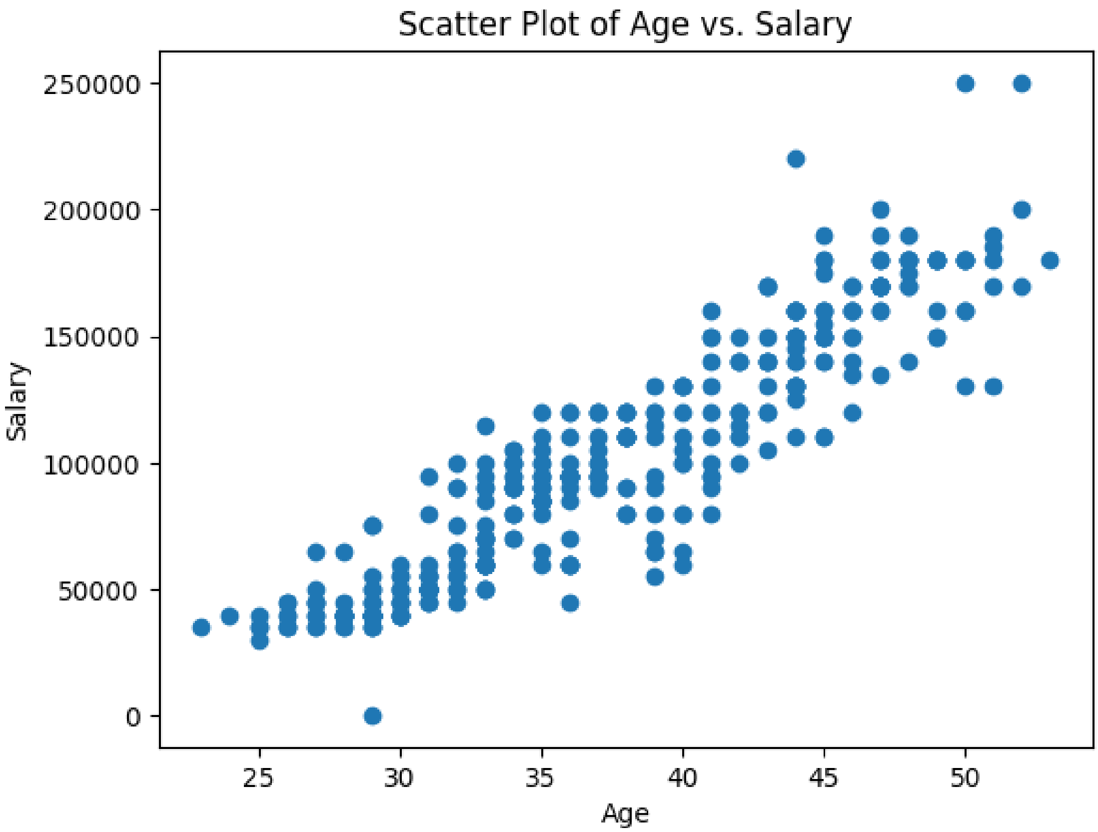
[ ]:
```
df.corr()
```

```
<ipython-input-10-2f6f6606aa2c>:1: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  df.corr()
```

[ ]:

	Age	Years of Experience	Salary
Age	1.000000	0.979128	0.922335
Years of Experience	0.979128	1.000000	0.930338
Salary	0.922335	0.930338	1.000000

[ ]:
```
# Scatter plot
plt.scatter(df['Age'], df['Salary'])
plt.title('Scatter Plot of Age vs. Salary')
plt.xlabel('Age')
plt.ylabel('Salary')
plt.show()
```



[ ]:
```
## Correlation
df.corr()
```

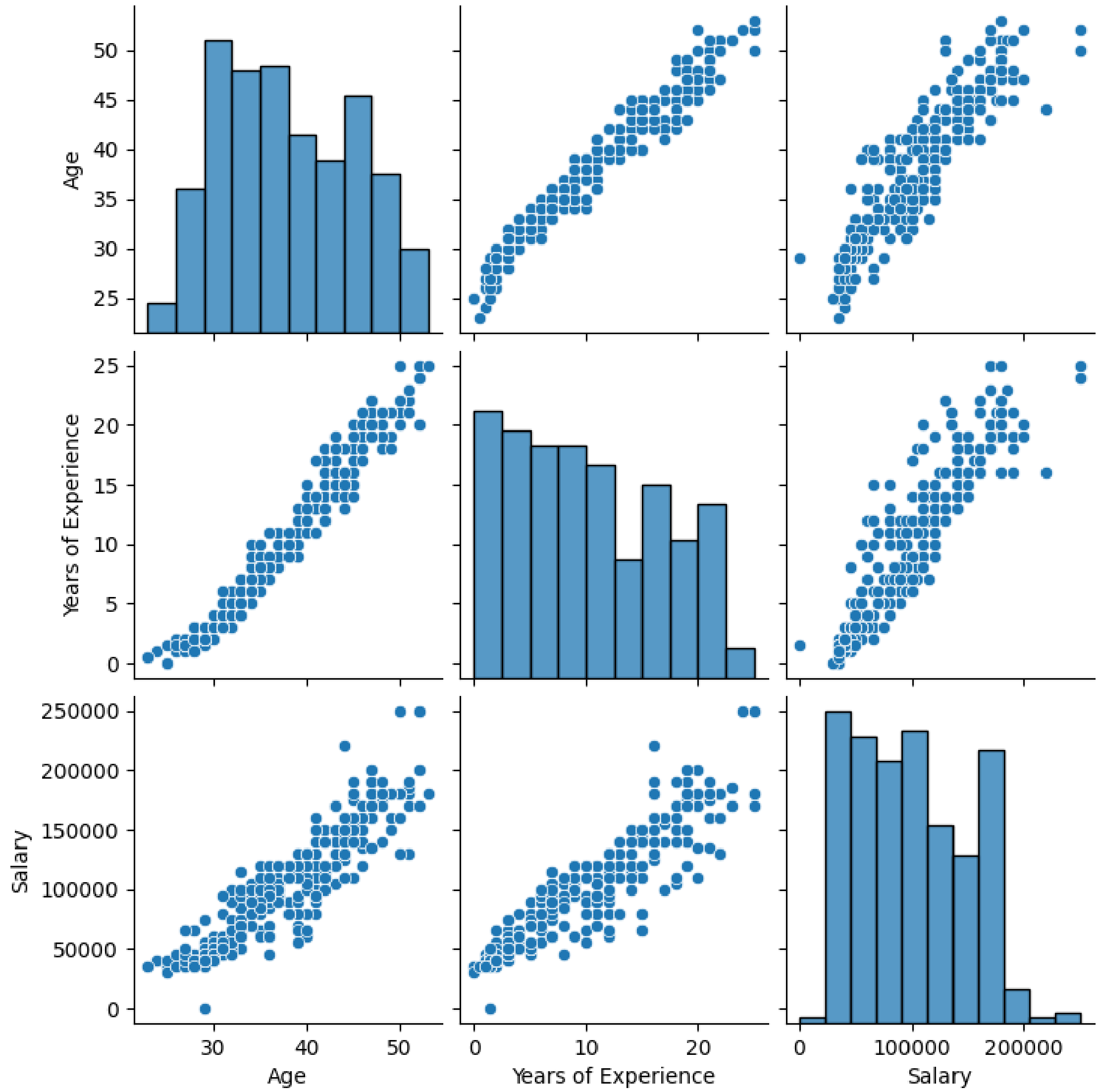
```
<ipython-input-12-2d23776439fc>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  df.corr()
```

[ ]:

	Age	Years of Experience	Salary
Age	1.000000	0.979128	0.922335
Years of Experience	0.979128	1.000000	0.930338
Salary	0.922335	0.930338	1.000000

[ ]:
```
## Seaborn for visualization
import seaborn as sns
sns.pairplot(df)
```

[ ]: <seaborn.axisgrid.PairGrid at 0x7dff2ea19b10>



[ ]:
```
X=df['Salary']
X
```

[ ]:
```
0      90000.0
1      65000.0
2     150000.0
3      60000.0
```

```
4      200000.0
         …
370     85000.0
371    170000.0
372     40000.0
373     90000.0
374    150000.0
Name: Salary, Length: 375, dtype: float64
```

[ ]:
```
X.shape
```

[ ]:
```
(375,)
```

[ ]:
```
## Independent and dependent features
X=df[['Salary']] ### independent features should be data frame or 2␣
 ↪dimesnionalarray
X
y=df['Years of Experience'] ## this variiable can be in series or 1d array
y
```

[ ]:
```
0       5.0
1       3.0
2      15.0
3       7.0
4      20.0
       …
370     8.0
371    19.0
372     2.0
373     7.0
374    15.0
Name: Years of Experience, Length: 375, dtype: float64
```

[ ]:
```
X_series=df['Salary']
np.array(X_series).shape
```

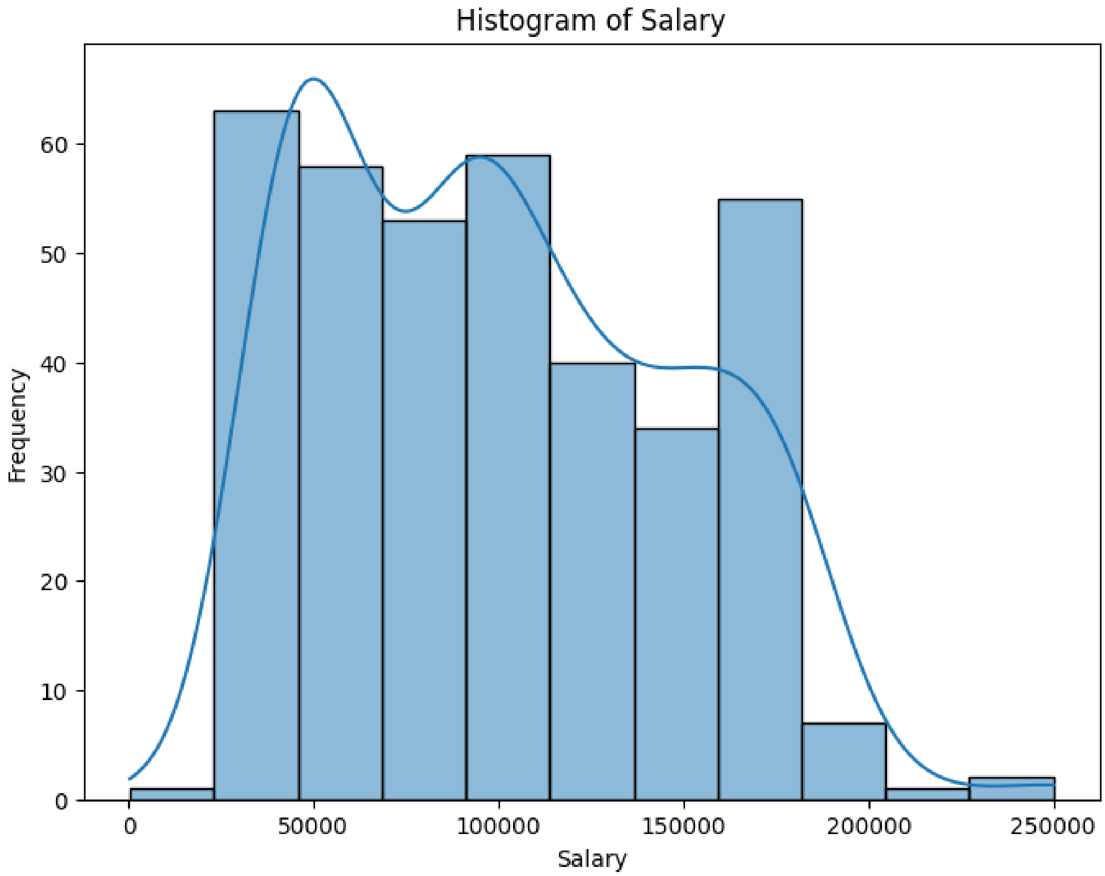
[ ]:
```
(375,)
```

[ ]:
```
#Univariate Analysis:
#For numerical variables: a. Calculate basic descriptive statistics (mean,␣
 ↪median, mode, standard deviation,
#min, max, quartiles, etc.).

mean_value = df['Salary'].mean()
median_value = df['Salary'].median()
mode_value = df['Salary'].mode().iloc[0]  # For handling multiple modes
std_deviation = df['Salary'].std()
```

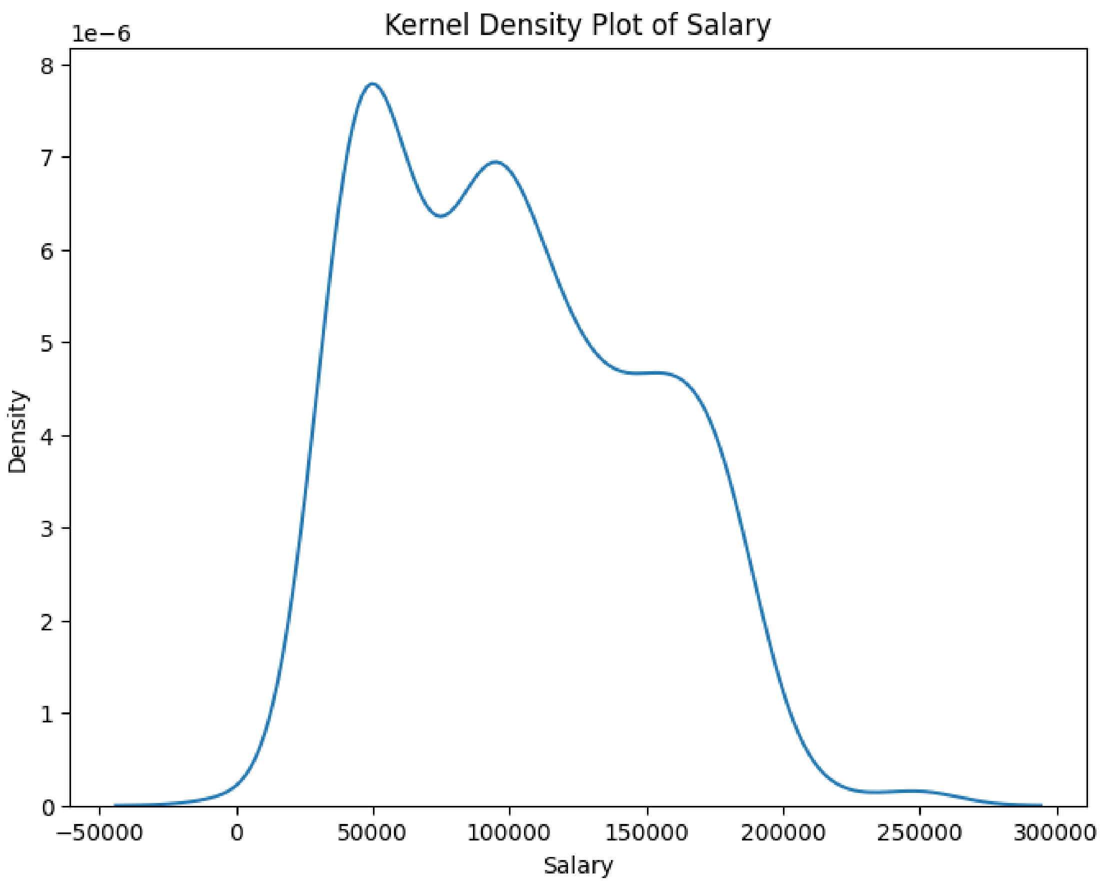
```
min_value = df['Salary'].min()
max_value = df['Salary'].max()


print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Mode: {mode_value}")
print(f"Standard Deviation: {std_deviation}")
print(f"Min: {min_value}")
print(f"Max: {max_value}")
```

```
Mean: 100577.34584450402
Median: 95000.0
Mode: 40000.0
Standard Deviation: 48240.013481882655
Min: 350.0
Max: 250000.0
```

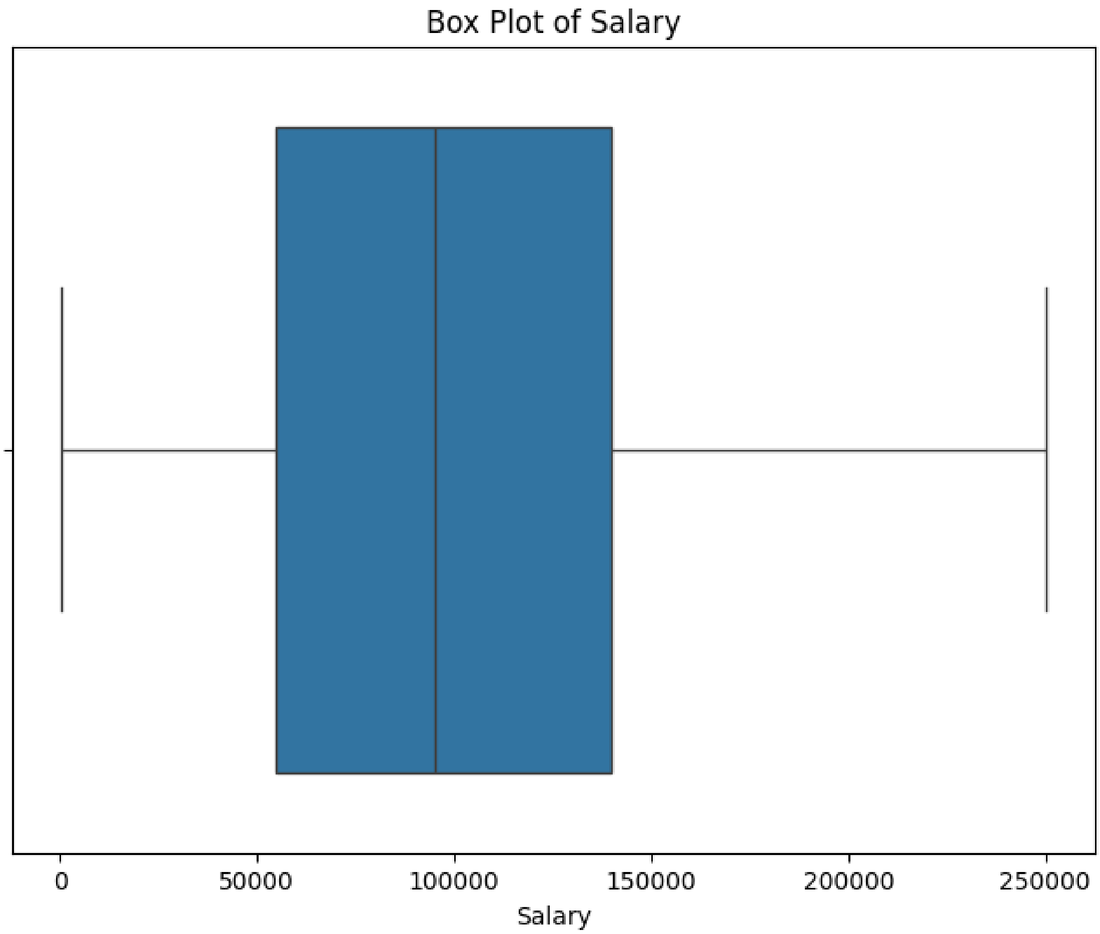
[ ]:
```
#b. Visualize the distribution using histograms, kernel density plots, or box␣
↪plots.

# Plot a simple histogram
plt.figure(figsize=(8, 6))
sns.histplot(df['Salary'], kde=True)
plt.title('Histogram of Salary')
plt.xlabel('Salary')
plt.ylabel('Frequency')

# Show the plot
plt.show()
```



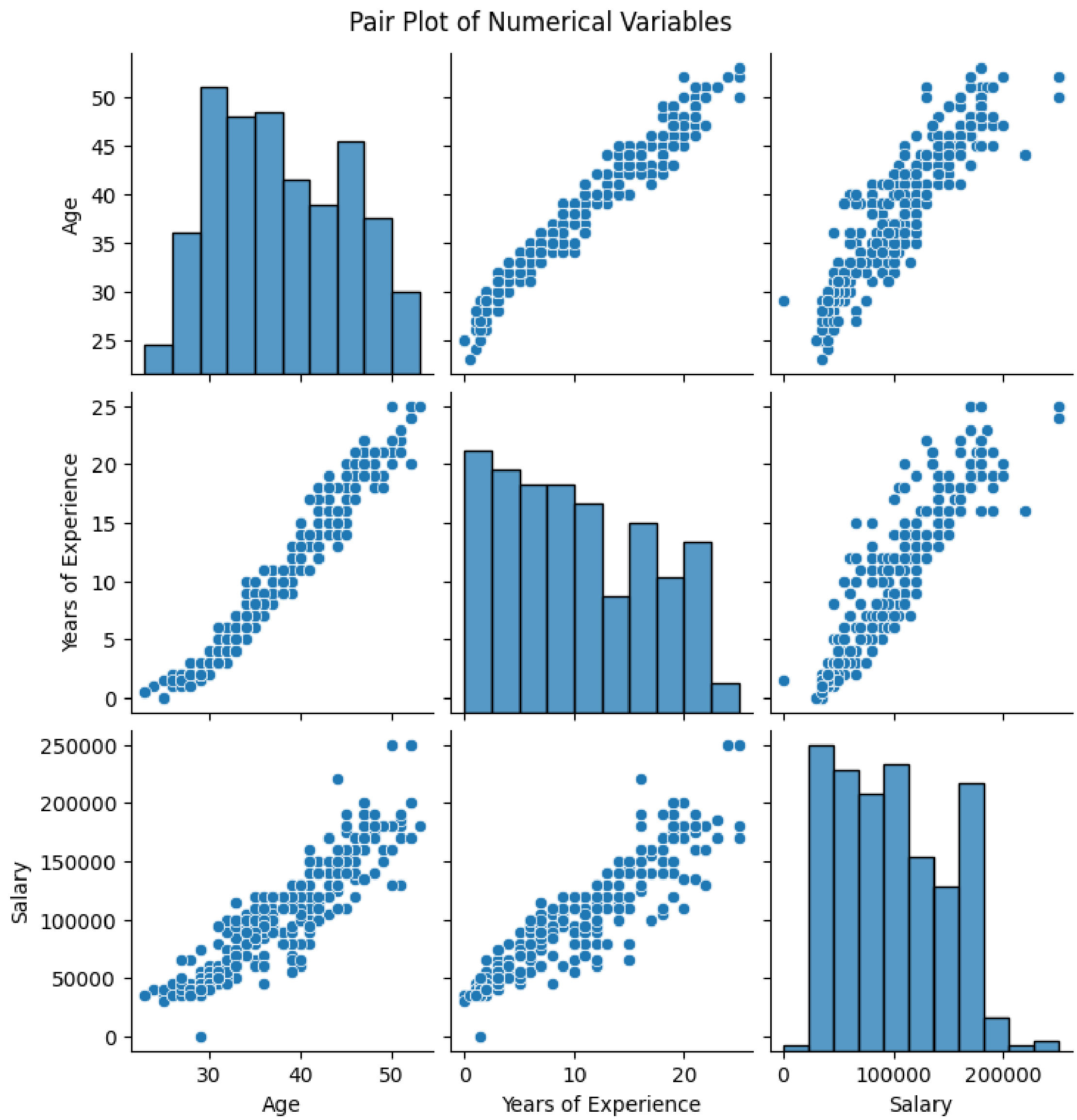

```
[ ]: # Plot a simple kernel density plot
plt.figure(figsize=(8, 6))
sns.kdeplot(df['Salary'])
plt.title('Kernel Density Plot of Salary')
plt.xlabel('Salary')
plt.ylabel('Density')

# Show the plot
plt.show()
```



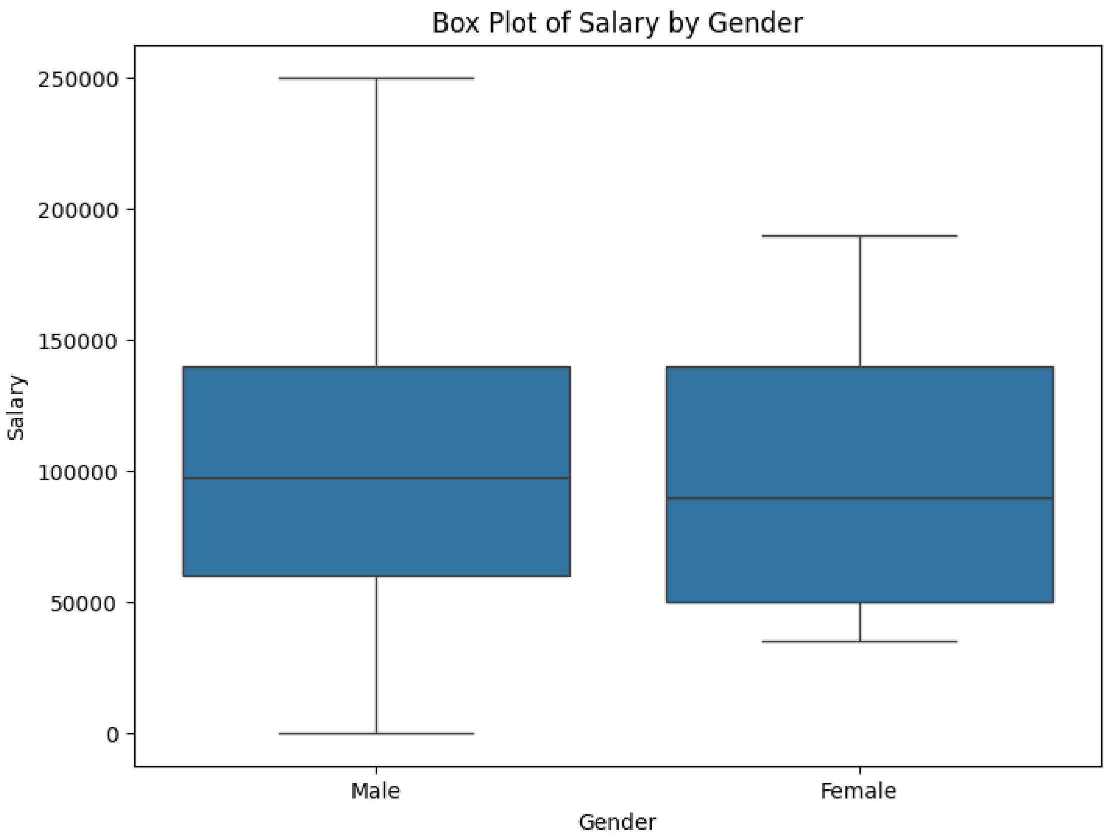
[ ]:
```
# Plot a simple box plot
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['Salary'])
plt.title('Box Plot of Salary')
plt.xlabel('Salary')

# Show the plot
plt.show()
```



[ ]:
```
#Bivariate Analysis: Explore relationships between pairs of numerical variables␣
 ↪using scatter plots, pair plots.

# Create a pair plot for numerical variables
sns.pairplot(df)
plt.suptitle('Pair Plot of Numerical Variables', y=1.02)
plt.show()
```




```
[ ]: #Bivariate Analysis: Explore relationships between numerical and categorical␣
      ↪variables using box plots or violin plots.

     # Box plot for 'Salary' vs 'Gender'
     plt.figure(figsize=(8, 6))
     sns.boxplot(x='Gender', y='Salary', data=df)
     plt.title('Box Plot of Salary by Gender')
     plt.xlabel('Gender')
     plt.ylabel('Salary')

     # Show the plot
     plt.show()
```



[ ]:
```python
#Bivariate Analysis: Calculate correlation coefficients between numerical␣
↪variables.

# Calculate correlation coefficients
correlation_matrix = df.corr()

# Display the correlation matrix
print("Correlation Coefficients:")
print(correlation_matrix)
```

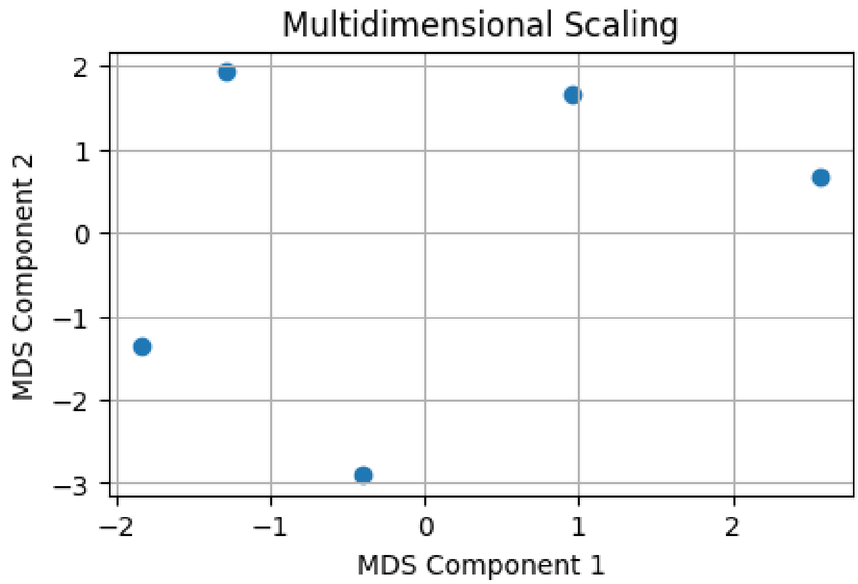
```
Correlation Coefficients:
                          Age  Years of Experience    Salary
Age                  1.000000             0.979128  0.922335
Years of Experience  0.979128             1.000000  0.930338
Salary               0.922335             0.930338  1.000000
```

```
<ipython-input-24-3a9b5b137c87>:4: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  correlation_matrix = df.corr()
```

Reduce the dimensionality of the dataset while preserving the pairwise distances

[ ]:
```
import pandas as pd
from sklearn.manifold import MDS
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import euclidean_distances
import matplotlib.pyplot as plt

# Convert categorical variables into numerical labels
df_encoded = df.copy()
label_encoder = LabelEncoder()
df_encoded['Gender'] = label_encoder.fit_transform(df['Gender'])
df_encoded['Education Level'] = label_encoder.fit_transform(df['Education␣
↪Level'])
df_encoded['Job Title'] = label_encoder.fit_transform(df['Job Title'])

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_encoded)

# Compute dissimilarity matrix
dissimilarities = euclidean_distances(scaled_data)

# Perform Multidimensional Scaling (MDS)
mds = MDS(n_components=2, dissimilarity='precomputed', random_state=42)
mds_data = mds.fit_transform(dissimilarities)

# Plot MDS results
plt.figure(figsize=(5, 3))
plt.scatter(mds_data[:, 0], mds_data[:, 1])
plt.xlabel('MDS Component 1')
plt.ylabel('MDS Component 2')
plt.title('Multidimensional Scaling')
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_mds.py:299:
FutureWarning: The default value of `normalized_stress` will change to `'auto'`
in version 1.4. To suppress this warning, manually set the value of
`normalized_stress`.
  warnings.warn(
```



[ ]:
```
# Perform Multidimensional Scaling (MDS)
mds = MDS(n_components=2, dissimilarity='euclidean', random_state=42)
mds_data = mds.fit_transform(scaled_data)
mds_data
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_mds.py:299:
FutureWarning: The default value of `normalized_stress` will change to `'auto'`
in version 1.4. To suppress this warning, manually set the value of
`normalized_stress`.
  warnings.warn(
```

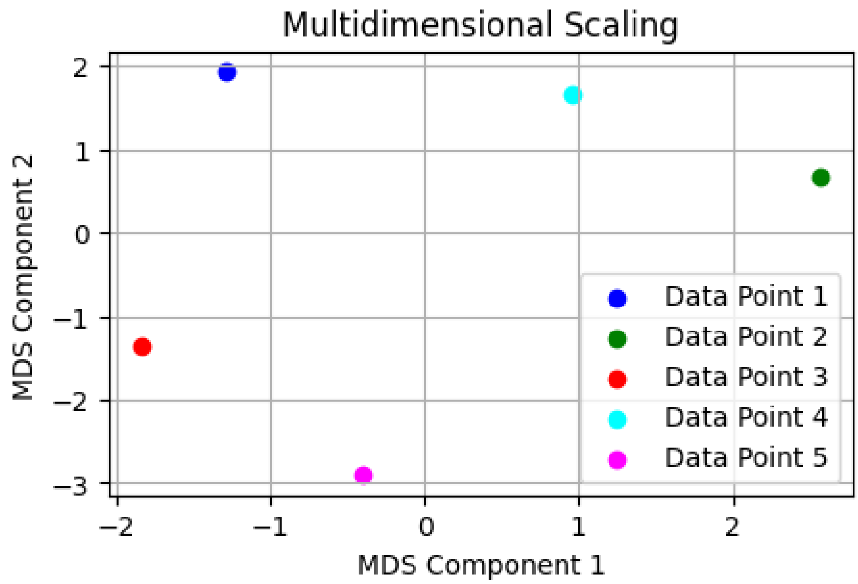
[ ]:
```
array([[-0.19465353,  3.00717262],
       [-2.75561293, -2.41538606],
       [ 3.473977  ,  1.07223545],
       [-3.19298606,  1.29350504],
       [ 2.66927552, -2.95752705]])
```

You are tasked with implementing Multi-Dimensional Scaling (MDS) to analyze and visualize the structure of a dataset containing pairwise dissimilarities or distances between a set of objects. Your goal is to reduce the dimensionality of the dataset while preserving the pairwise distances as much as possible.

[ ]:
```
import pandas as pd
from sklearn.manifold import MDS
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import euclidean_distances
import matplotlib.pyplot as plt
```

```python
# Encode categorical variables
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
df['Education Level'] = label_encoder.fit_transform(df['Education Level'])
df['Job Title'] = label_encoder.fit_transform(df['Job Title'])

# Standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# Compute pairwise dissimilarities
dissimilarities = euclidean_distances(scaled_data)

# Perform Multidimensional Scaling (MDS)
mds = MDS(n_components=2, dissimilarity='precomputed', random_state=42)
mds_data = mds.fit_transform(dissimilarities)

# Plot MDS results with colors
plt.figure(figsize=(5, 3))
colors = ['blue', 'green', 'red', 'cyan', 'magenta']  # Colors for different␣
↪categories
for i in range(len(mds_data)):
    plt.scatter(mds_data[i, 0], mds_data[i, 1], c=colors[i], label=f'Data Point␣
↪{i+1}')

plt.xlabel('MDS Component 1')
plt.ylabel('MDS Component 2')
plt.title('Multidimensional Scaling')
plt.legend()
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_mds.py:299:
FutureWarning: The default value of `normalized_stress` will change to `'auto'`
in version 1.4. To suppress this warning, manually set the value of
`normalized_stress`.
  warnings.warn(
```

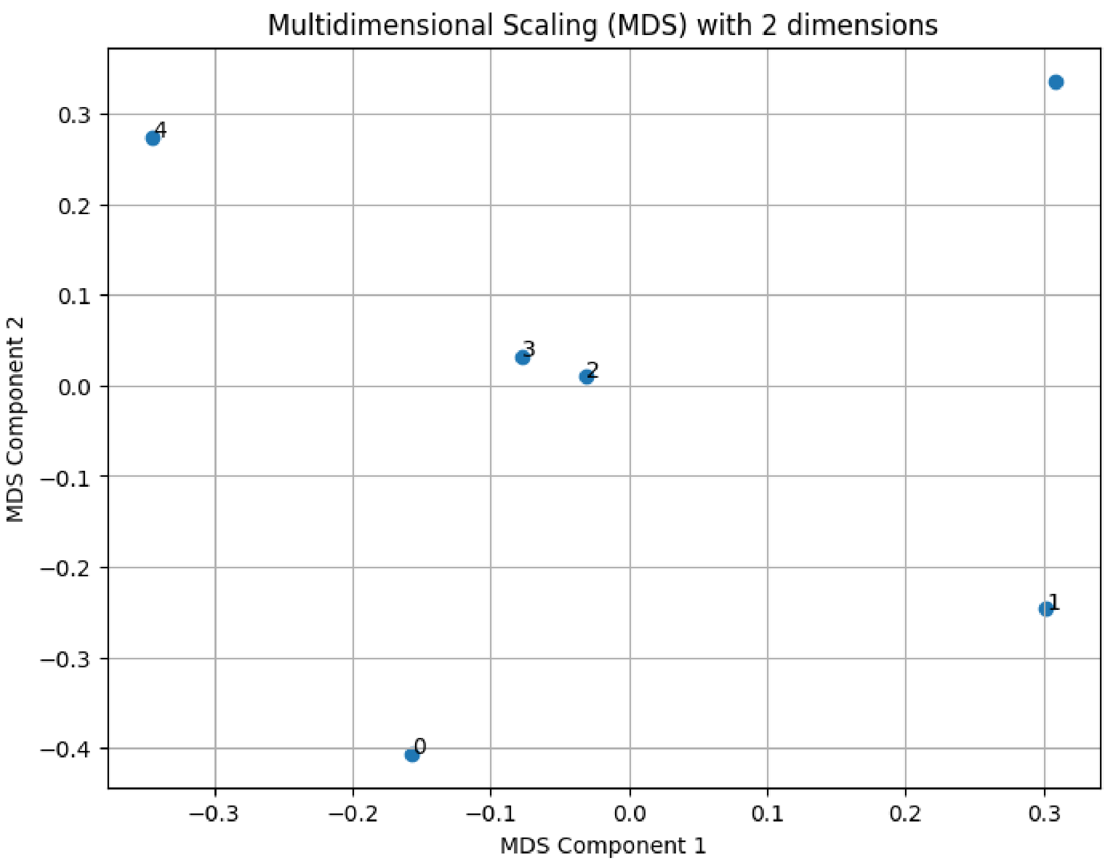


Implement Multi-Dimensional Scaling (MDS) using the sklearn.manifold.MDS module to reduce the dimensionality of the dataset to k dimensions, where k is a user-defined parameter. Visualize the objects in a k-dimensional scatter plot based on the MDS results. Each point on the plot represents an object, and the position of the points should reflect their relative similarities or dissimilarities as accurately as possible. Provide appropriate labels for the objects on the scatter plot to make it clear which point corresponds to which object.

[ ]:
```
import pandas as pd
from sklearn.manifold import MDS
import matplotlib.pyplot as plt


# Compute dissimilarity matrix
dissimilarities = df.corr()

# Define the number of dimensions
k = 2

# Perform Multidimensional Scaling (MDS)
mds = MDS(n_components=k, dissimilarity='precomputed', random_state=42)
mds_data = mds.fit_transform(dissimilarities)

# Plot MDS results
plt.figure(figsize=(8, 6))
plt.scatter(mds_data[:, 0], mds_data[:, 1])

# Annotate points with object labels
```

```
for i, txt in enumerate(df.index):
    plt.annotate(txt, (mds_data[i, 0], mds_data[i, 1]))

plt.xlabel(f'MDS Component 1')
plt.ylabel(f'MDS Component 2')
plt.title(f'Multidimensional Scaling (MDS) with {k} dimensions')
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_mds.py:299:
FutureWarning: The default value of `normalized_stress` will change to `'auto'`
in version 1.4. To suppress this warning, manually set the value of
`normalized_stress`.
  warnings.warn(
```

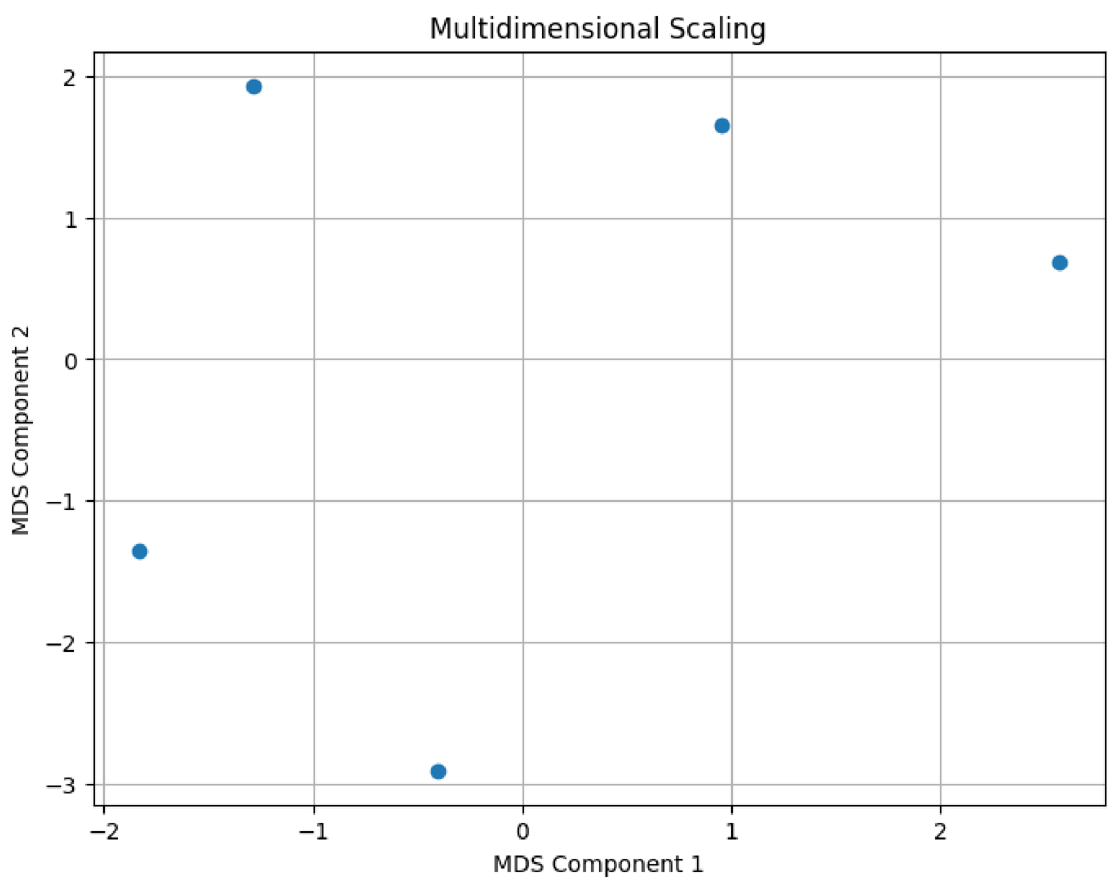


Use Python and any necessary libraries for data manipulation, dimensionality reduction, and visualization. Ensure that your code is flexible and can handle datasets of varying sizes and dimensions. Allow the user to specify the number of dimensions k for the MDS algorithm. Include comments where necessary to explain your approach and any important steps. Test your code with synthetic datasets of different sizes and dimensions to ensure its robustness and efficiency.

[ ]:
```python
import pandas as pd
from sklearn.manifold import MDS
from sklearn.preprocessing import StandardScaler, LabelEncoder
import matplotlib.pyplot as plt

def preprocess_dataset(df):
    """
    Preprocess the dataset if necessary.
    """
    # Handle missing values, encode categorical variables, etc.
    # For simplicity, assume no preprocessing is required in this example.
    return df

def perform_mds(df, n_components=2):
    """
    Perform Multidimensional Scaling (MDS) to reduce the dimensionality of the␣
 ↪dataset.
    """
    # Standardize the data
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(df)

    # Perform MDS
    mds = MDS(n_components=n_components, dissimilarity='euclidean',␣
 ↪random_state=42)
    mds_data = mds.fit_transform(scaled_data)

    return mds_data

def visualize_data(mds_data):
    """
    Visualize the reduced-dimensional data.
    """
    plt.figure(figsize=(8, 6))
    plt.scatter(mds_data[:, 0], mds_data[:, 1])
    plt.xlabel('MDS Component 1')
    plt.ylabel('MDS Component 2')
    plt.title('Multidimensional Scaling')
    plt.grid(True)
    plt.show()

def main():
    # Load the dataset (assuming it's already loaded)
    # df = pd.read_csv("your_dataset.csv")

    # Preprocess the dataset if necessary
    # df = preprocess_dataset(df)
```

```python
    # Specify the number of dimensions for MDS
    k = 2  # Specify the desired number of dimensions here


    # Perform MDS
    mds_data = perform_mds(df, n_components=k)

    # Visualize the reduced-dimensional data
    visualize_data(mds_data)

if __name__ == "__main__":
    main()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_mds.py:299:
FutureWarning: The default value of `normalized_stress` will change to `'auto'`
in version 1.4. To suppress this warning, manually set the value of
`normalized_stress`.
  warnings.warn(
```



CONCLUSION: In conclusion, Multi-Dimensional Scaling (MDS) offers a valuable tool for dimensionality reduction in machine learning. By transforming high-dimensional data into a lower-dimensional space while preserving pairwise distances, MDS facilitates visualization, exploration, and interpretation of complex datasets. Its ability to uncover underlying structure and relationships within the data makes it a valuable technique for data analysis and visualization tasks. Overall, MDS provides insights that can aid in understanding the intrinsic relationships among data points, thus contributing to informed decision-making and knowledge discovery in various domains.