2348441_lab_4

March 9, 2024

Lab Exercise 4 -Regression Analysis

Created by : Nileem Kaveramma C C | 2348441 Created DATE:08-03-2024 Edited Date: 09-03-2024

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
- from sklearn.decomposition import PCA: Imports the Principal Component Analysis (PCA) module from scikit-learn for dimensionality reduction and feature extraction.
- from sklearn.preprocessing import StandardScaler: Imports StandardScaler from scikit-learn, a method for standardizing numerical features by removing the mean and scaling to unit variance.

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 Principal Component Analysis (PCA) is to reduce the dimensionality of a dataset while retaining as much of the original variability as possible. By transforming the original features into a new set of uncorrelated variables called principal components, PCA simplifies data representation, aids in identifying patterns, and facilitates efficient analysis and visualization, particularly useful in high-dimensional datasets.

PROCEDURE:

Data Collection: Gather a dataset with numerical features for which dimensionality reduction is desired.

Data Standardization: Standardize the data by subtracting the mean and scaling to unit variance. This step ensures that all features contribute equally to the analysis.

Covariance Matrix Computation: Calculate the covariance matrix of the standardized data. The covariance matrix represents the relationships between different features.

Eigenvalue and Eigenvector Calculation: Find the eigenvalues and corresponding eigenvectors of the covariance matrix. These eigenvectors represent the principal components, and the eigenvalues indicate their importance.

Sort Eigenvalues and Eigenvectors: Arrange the eigenvalues and their corresponding eigenvectors in descending order. The principal components with higher eigenvalues capture more variance in the data.

Selection of Principal Components: Determine the number of principal components to retain based on the explained variance. This decision often involves selecting components that explain a significant percentage (e.g., 95%) of the total variance.

Projection of Data onto Principal Components: Project the original data onto the selected principal components to obtain a reduced-dimensional representation of the dataset.

Analysis and Visualization: Analyze the transformed data using the selected principal components. Visualize the results to gain insights into the structure and patterns within the dataset.

Interpretation of Principal Components: Interpret the principal components in the context of the original features to understand the underlying patterns and relationships in the data.

Validation and Iteration: Validate the results and iterate the procedure as needed, adjusting the number of retained components or exploring alternative preprocessing steps.

1) Import all the necessary libraries

[ ]:
```
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
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

2)Loading the dataset

[ ]:
```
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]
```

3). Perform some basic EDA

[ ]:
```python
columns_names=df.columns.tolist()
print("Columns names:")
print(columns_names)
```

```
Columns names:
['Age', 'Gender', 'Education Level', 'Job Title', 'Years of Experience',
'Salary']
```

df.columns.tolist() fetches all the columns and then convert it into list type.This step is just to check out all the column names in our data.Columns are also called as features of our datasets.

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

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

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

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

[ ]:
```python
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()
```

[ ]:
```
Age                    375
Gender                 375
Education Level        375
Job Title              375
Years of Experience    375
Salary                 375
dtype: int64
```

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
```

The 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-12-2f6f6606aa2c>: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()
```

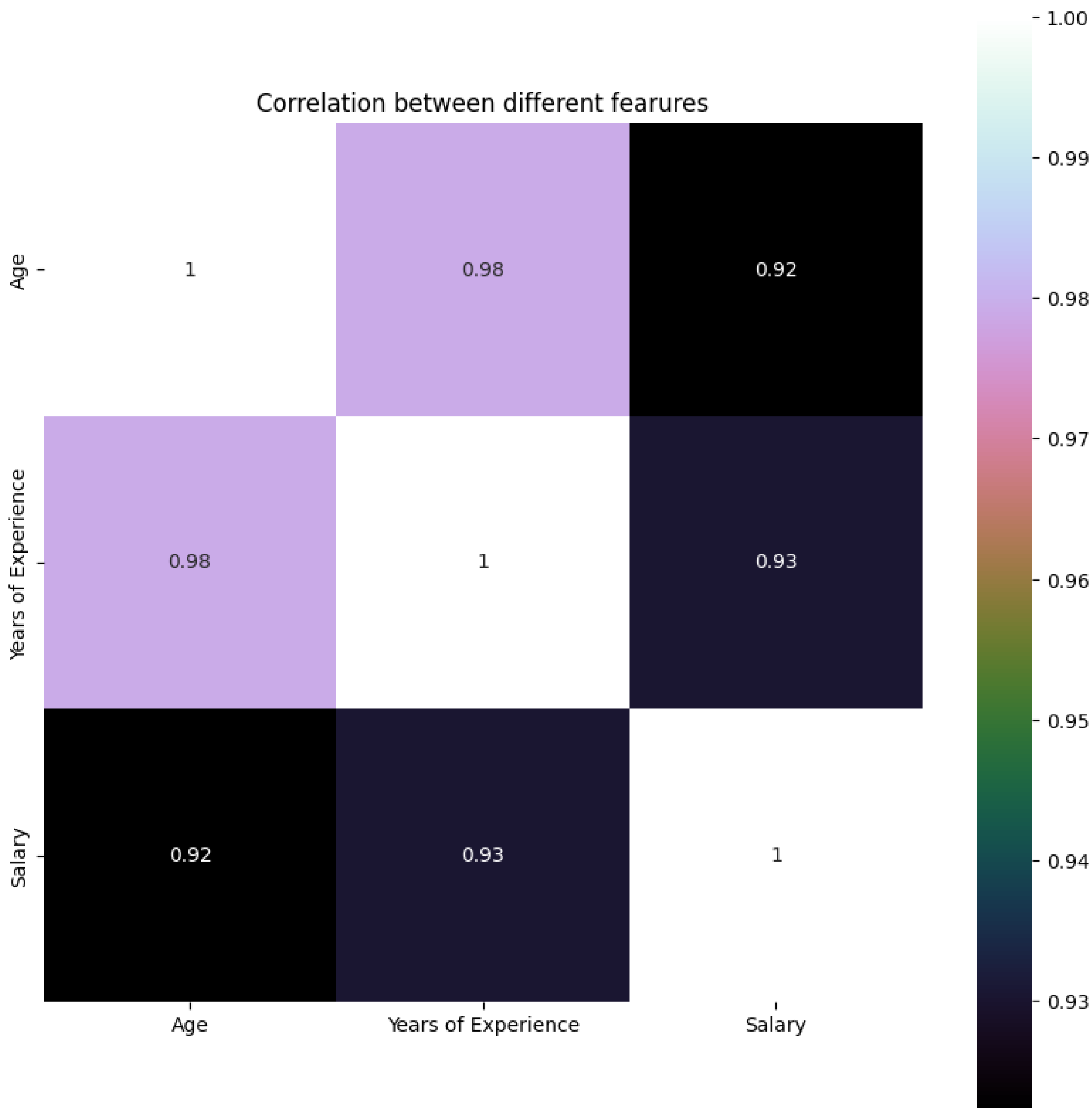
[ ]:

	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

[ ]:
```
correlation = df.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, vmax=1, square=True,annot=True,cmap='cubehelix')

plt.title('Correlation between different fearures')
```

```
<ipython-input-13-effb445d3340>: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.
  correlation = df.corr()
```

[ ]:
```
Text(0.5, 1.0, 'Correlation between different fearures')
```



Performing some visualisation before moving onto PCA

[ ]:
```
grouped_data = df.groupby('Job Title').sum()
print(grouped_data)
```

```
                             Age  Years of Experience    Salary
Job Title
Account Manager             32.0                  5.0   75000.0
Accountant                  31.0                  4.0   55000.0
Administrative Assistant    75.0                 18.0  100000.0
Business Analyst            67.0                 12.0  155000.0
Business Development Manager  34.0                8.0   90000.0
…                             …                   …         …
```

```
UX Designer                     34.0                     5.0   80000.0
UX Researcher                   27.0                     2.0   65000.0
VP of Finance                   47.0                    19.0  200000.0
VP of Operations                47.0                    19.0  190000.0
Web Developer                   33.0                     6.0   65000.0

[174 rows x 3 columns]

<ipython-input-14-9d5f552b0293>:1: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  grouped_data = df.groupby('Job Title').sum()
```

[ ]:
```
df['Job Title'].unique()
```

[ ]:
```
array(['Software Engineer', 'Data Analyst', 'Senior Manager',
       'Sales Associate', 'Director', 'Marketing Analyst',
       'Product Manager', 'Sales Manager', 'Marketing Coordinator',
       'Senior Scientist', 'Software Developer', 'HR Manager',
       'Financial Analyst', 'Project Manager', 'Customer Service Rep',
       'Operations Manager', 'Marketing Manager', 'Senior Engineer',
       'Data Entry Clerk', 'Sales Director', 'Business Analyst',
       'VP of Operations', 'IT Support', 'Recruiter', 'Financial Manager',
       'Social Media Specialist', 'Software Manager', 'Junior Developer',
       'Senior Consultant', 'Product Designer', 'CEO', 'Accountant',
       'Data Scientist', 'Marketing Specialist', 'Technical Writer',
       'HR Generalist', 'Project Engineer', 'Customer Success Rep',
       'Sales Executive', 'UX Designer', 'Operations Director',
       'Network Engineer', 'Administrative Assistant',
       'Strategy Consultant', 'Copywriter', 'Account Manager',
       'Director of Marketing', 'Help Desk Analyst',
       'Customer Service Manager', 'Business Intelligence Analyst',
       'Event Coordinator', 'VP of Finance', 'Graphic Designer',
       'UX Researcher', 'Social Media Manager', 'Director of Operations',
       'Senior Data Scientist', 'Junior Accountant',
       'Digital Marketing Manager', 'IT Manager',
       'Customer Service Representative', 'Business Development Manager',
       'Senior Financial Analyst', 'Web Developer', 'Research Director',
       'Technical Support Specialist', 'Creative Director',
       'Senior Software Engineer', 'Human Resources Director',
       'Content Marketing Manager', 'Technical Recruiter',
       'Sales Representative', 'Chief Technology Officer',
       'Junior Designer', 'Financial Advisor', 'Junior Account Manager',
       'Senior Project Manager', 'Principal Scientist',
       'Supply Chain Manager', 'Senior Marketing Manager',
       'Training Specialist', 'Research Scientist',
       'Junior Software Developer', 'Public Relations Manager',
```

```
 'Operations Analyst', 'Product Marketing Manager',
 'Senior HR Manager', 'Junior Web Developer',
 'Senior Project Coordinator', 'Chief Data Officer',
 'Digital Content Producer', 'IT Support Specialist',
 'Senior Marketing Analyst', 'Customer Success Manager',
 'Senior Graphic Designer', 'Software Project Manager',
 'Supply Chain Analyst', 'Senior Business Analyst',
 'Junior Marketing Analyst', 'Office Manager', 'Principal Engineer',
 'Junior HR Generalist', 'Senior Product Manager',
 'Junior Operations Analyst', 'Senior HR Generalist',
 'Sales Operations Manager', 'Senior Software Developer',
 'Junior Web Designer', 'Senior Training Specialist',
 'Senior Research Scientist', 'Junior Sales Representative',
 'Junior Marketing Manager', 'Junior Data Analyst',
 'Senior Product Marketing Manager', 'Junior Business Analyst',
 'Senior Sales Manager', 'Junior Marketing Specialist',
 'Junior Project Manager', 'Senior Accountant', 'Director of Sales',
 'Junior Recruiter', 'Senior Business Development Manager',
 'Senior Product Designer', 'Junior Customer Support Specialist',
 'Senior IT Support Specialist', 'Junior Financial Analyst',
 'Senior Operations Manager', 'Director of Human Resources',
 'Junior Software Engineer', 'Senior Sales Representative',
 'Director of Product Management', 'Junior Copywriter',
 'Senior Marketing Coordinator', 'Senior Human Resources Manager',
 'Junior Business Development Associate', 'Senior Account Manager',
 'Senior Researcher', 'Junior HR Coordinator',
 'Director of Finance', 'Junior Marketing Coordinator', nan,
 'Junior Data Scientist', 'Senior Operations Analyst',
 'Senior Human Resources Coordinator', 'Senior UX Designer',
 'Junior Product Manager', 'Senior Marketing Specialist',
 'Senior IT Project Manager', 'Senior Quality Assurance Analyst',
 'Director of Sales and Marketing', 'Senior Account Executive',
 'Director of Business Development', 'Junior Social Media Manager',
 'Senior Human Resources Specialist', 'Senior Data Analyst',
 'Director of Human Capital', 'Junior Advertising Coordinator',
 'Junior UX Designer', 'Senior Marketing Director',
 'Senior IT Consultant', 'Senior Financial Advisor',
 'Junior Business Operations Analyst',
 'Junior Social Media Specialist',
 'Senior Product Development Manager', 'Junior Operations Manager',
 'Senior Software Architect', 'Junior Research Scientist',
 'Senior Financial Manager', 'Senior HR Specialist',
 'Senior Data Engineer', 'Junior Operations Coordinator',
 'Director of HR', 'Senior Operations Coordinator',
 'Junior Financial Advisor', 'Director of Engineering'],
dtype=object)
```

```
[ ]: groupby_sales=df.groupby('Job Title').mean()
     groupby_sales
```

```
<ipython-input-16-455fffa59049>:1: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.mean is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  groupby_sales=df.groupby('Job Title').mean()
```

```
[ ]:                                 Age  Years of Experience    Salary
     Job Title
     Account Manager                32.0                  5.0   75000.0
     Accountant                     31.0                  4.0   55000.0
     Administrative Assistant       37.5                  9.0   50000.0
     Business Analyst               33.5                  6.0   77500.0
     Business Development Manager   34.0                  8.0   90000.0
     …                               …                  …        …
     UX Designer                    34.0                  5.0   80000.0
     UX Researcher                  27.0                  2.0   65000.0
     VP of Finance                  47.0                 19.0  200000.0
     VP of Operations               47.0                 19.0  190000.0
     Web Developer                  33.0                  6.0   65000.0

     [174 rows x 3 columns]
```

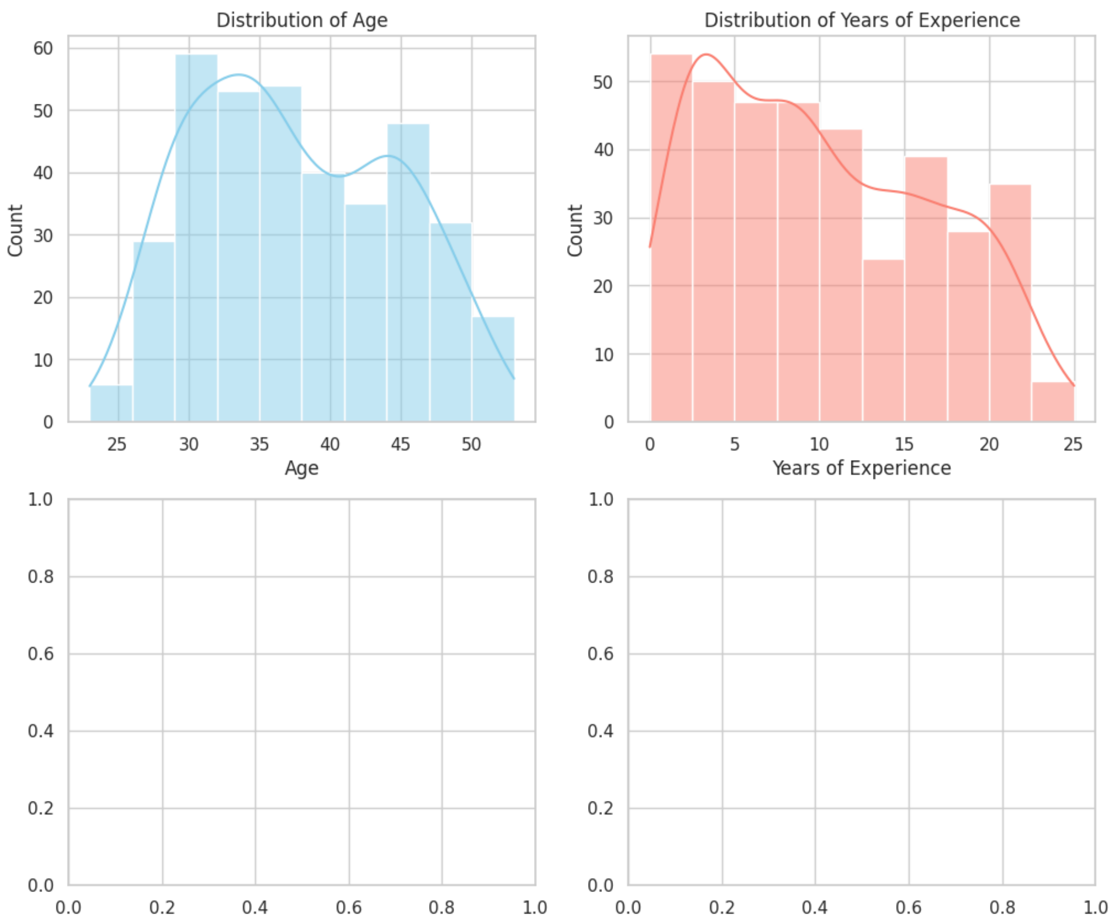
```
[ ]: #Calculate basic descriptive statistics (mean, median, mode, standard␣
      ↪deviation, min, max, quartiles, etc.

     # Mean
     mean_salary = df['Salary'].mean()
     print("Mean Salary:", mean_salary)

     # Median
     median_salary = df['Salary'].median()
     print("Median Salary:", median_salary)

     # Mode
     mode_salary = df['Salary'].mode()[0]
     print("Mode Salary:", mode_salary)

     # Standard Deviation
     std_salary = df['Salary'].std()
     print("Standard Deviation Salary:", std_salary)

     # Minimum and Maximum
     min_salary = df['Salary'].min()
     max_salary = df['Salary'].max()
```

```
print("Minimum Salary:", min_salary)
print("Maximum Salary:", max_salary)

# Quartiles
first_quartile = df['Salary'].quantile(0.25)
second_quartile = df['Salary'].quantile(0.5)
third_quartile = df['Salary'].quantile(0.75)

print("First Quartile (25th percentile):", first_quartile)
print("Second Quartile (Median):", second_quartile)
print("Third Quartile (75th percentile):", third_quartile)
```

```
Mean Salary: 100577.34584450402
Median Salary: 95000.0
Mode Salary: 40000.0
Standard Deviation Salary: 48240.013481882655
Minimum Salary: 350.0
Maximum Salary: 250000.0
First Quartile (25th percentile): 55000.0
Second Quartile (Median): 95000.0
Third Quartile (75th percentile): 140000.0
```

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

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

# Create subplots
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 10))

# Plot histograms
sns.histplot(df['Age'], kde=True, ax=axes[0, 0], color='skyblue')
axes[0, 0].set_title('Distribution of Age')

sns.histplot(df['Years of Experience'], kde=True, ax=axes[0, 1], color='salmon')
axes[0, 1].set_title('Distribution of Years of Experience')
```

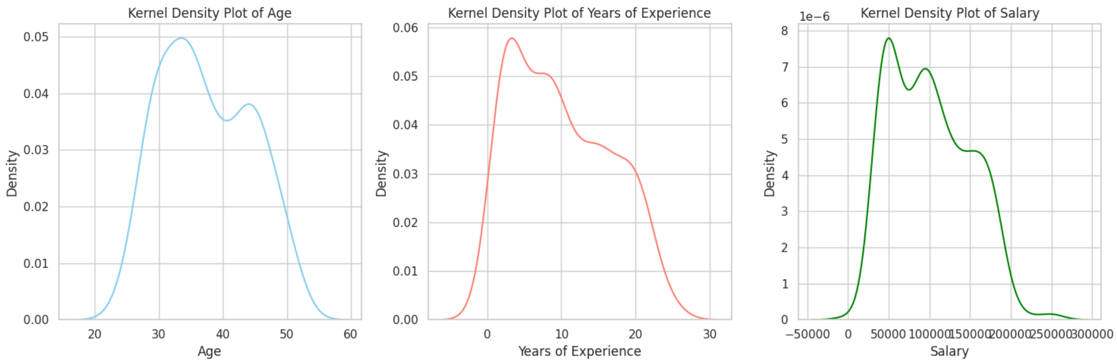
[ ]:
```
Text(0.5, 1.0, 'Distribution of Years of Experience')
```




```
[ ]:
```

```
[ ]: # Set the style for seaborn
sns.set(style="whitegrid")

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

# Plot kernel density plots
sns.kdeplot(df['Age'], ax=axes[0], color='skyblue')
axes[0].set_title('Kernel Density Plot of Age')

sns.kdeplot(df['Years of Experience'], ax=axes[1], color='salmon')
axes[1].set_title('Kernel Density Plot of Years of Experience')

sns.kdeplot(df['Salary'], ax=axes[2], color='green')
axes[2].set_title('Kernel Density Plot of Salary')
```

```
# Adjust layout
plt.tight_layout()

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



[ ]:
```
#For categorical variables:
#a. Display frequency tables showing counts and percentages.

# Display frequency table for the 'Gender' column
gender_counts = df['Gender'].value_counts()
gender_percentages = df['Gender'].value_counts(normalize=True) * 100

gender_table = pd.DataFrame({
    'Count': gender_counts,
    'Percentage': gender_percentages
})

print("Frequency Table for Gender:")
print(gender_table)
print("\n" + "="*30 + "\n")

# Display frequency table for the 'Education Level' column
education_counts = df['Education Level'].value_counts()
education_percentages = df['Education Level'].value_counts(normalize=True) * 100

education_table = pd.DataFrame({
    'Count': education_counts,
    'Percentage': education_percentages
})

print("Frequency Table for Education Level:")
print(education_table)
```

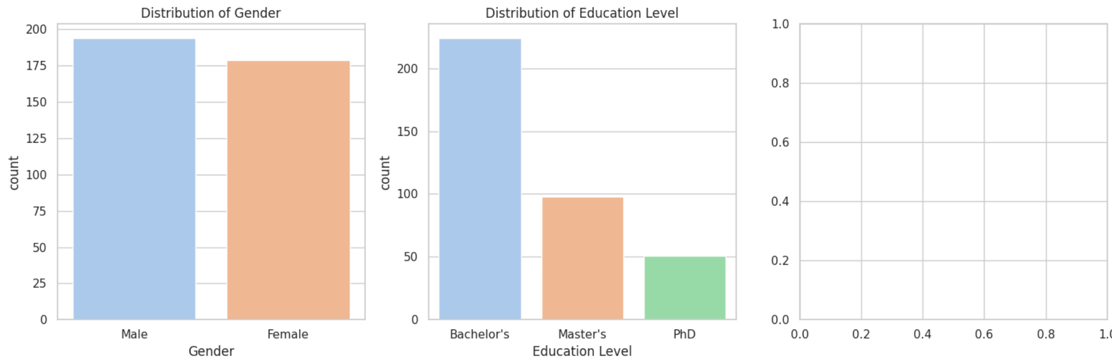
```
Frequency Table for Gender:
        Count  Percentage
Male      194   52.010724
Female    179   47.989276

==============================

Frequency Table for Education Level:
            Count  Percentage
Bachelor's    224   60.053619
Master's       98   26.273458
PhD            51   13.672922
```

[ ]:
```python
#For categorical variables:
#b. Visualize using bar plots.

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

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

# Plot bar plots for categorical variables
sns.countplot(x='Gender', data=df, ax=axes[0], palette='pastel')
axes[0].set_title('Distribution of Gender')

sns.countplot(x='Education Level', data=df, ax=axes[1], palette='pastel')
axes[1].set_title('Distribution of Education Level')



# Adjust layout
plt.tight_layout()

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

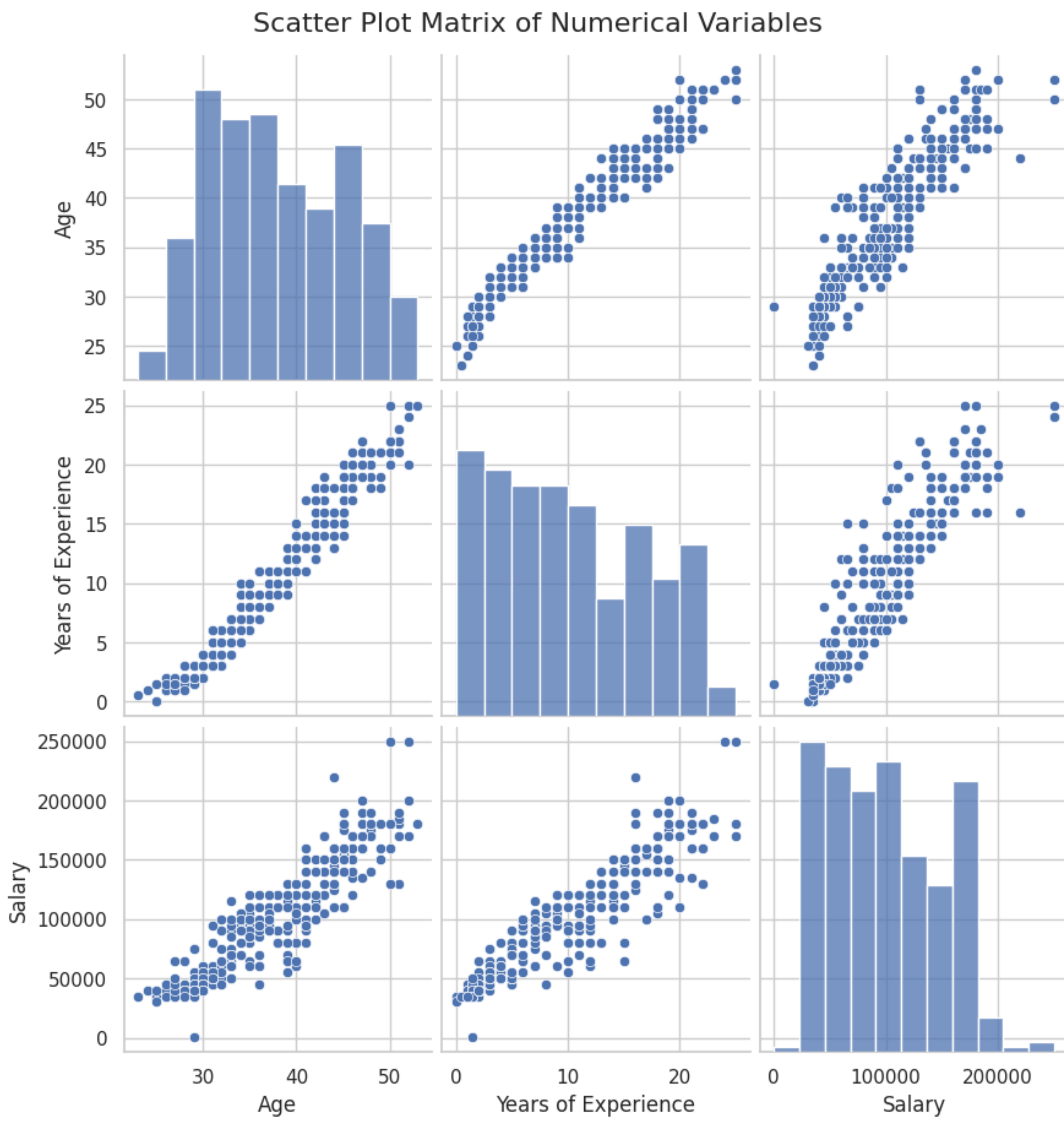
```
<ipython-input-21-9bcf1f635e29>:11: FutureWarning:

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

  sns.countplot(x='Gender', data=df, ax=axes[0], palette='pastel')
<ipython-input-21-9bcf1f635e29>:14: FutureWarning:

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

effect.

```
sns.countplot(x='Education Level', data=df, ax=axes[1], palette='pastel')
```



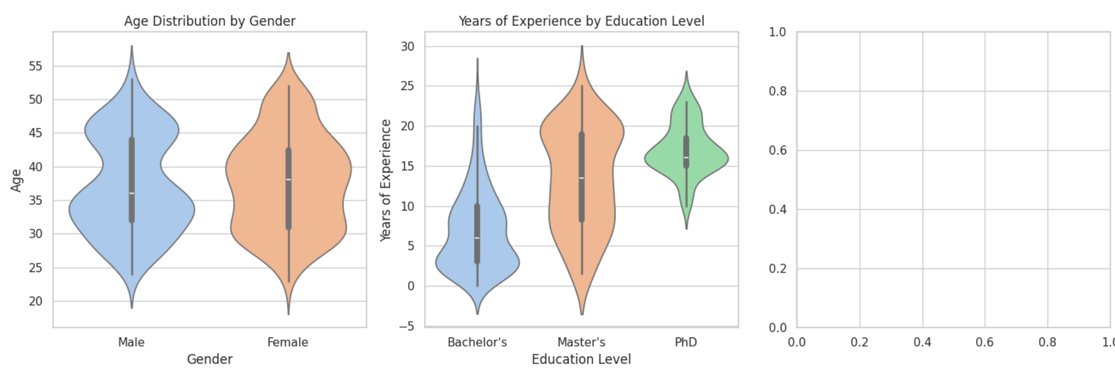
[ ]:
```
#Bivariate Analysis:
#Explore relationships between pairs of numerical variables using scatter plots

# Assuming your DataFrame is named df

# Select numerical columns for the scatter plot matrix
numerical_columns = ['Age', 'Years of Experience', 'Salary']

# Create a pair plot for numerical variables
sns.pairplot(df[numerical_columns], height=3)
plt.suptitle('Scatter Plot Matrix of Numerical Variables', y=1.02, size=16)
plt.show()
```




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

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

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))

# Violin plots for numerical vs categorical variables
sns.violinplot(x='Gender', y='Age', data=df, ax=axes[0], palette='pastel')
```

```
axes[0].set_title('Age Distribution by Gender')

sns.violinplot(x='Education Level', y='Years of Experience', data=df,␣
 ↪ax=axes[1], palette='pastel')
axes[1].set_title('Years of Experience by Education Level')
# Adjust layout
plt.tight_layout()

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

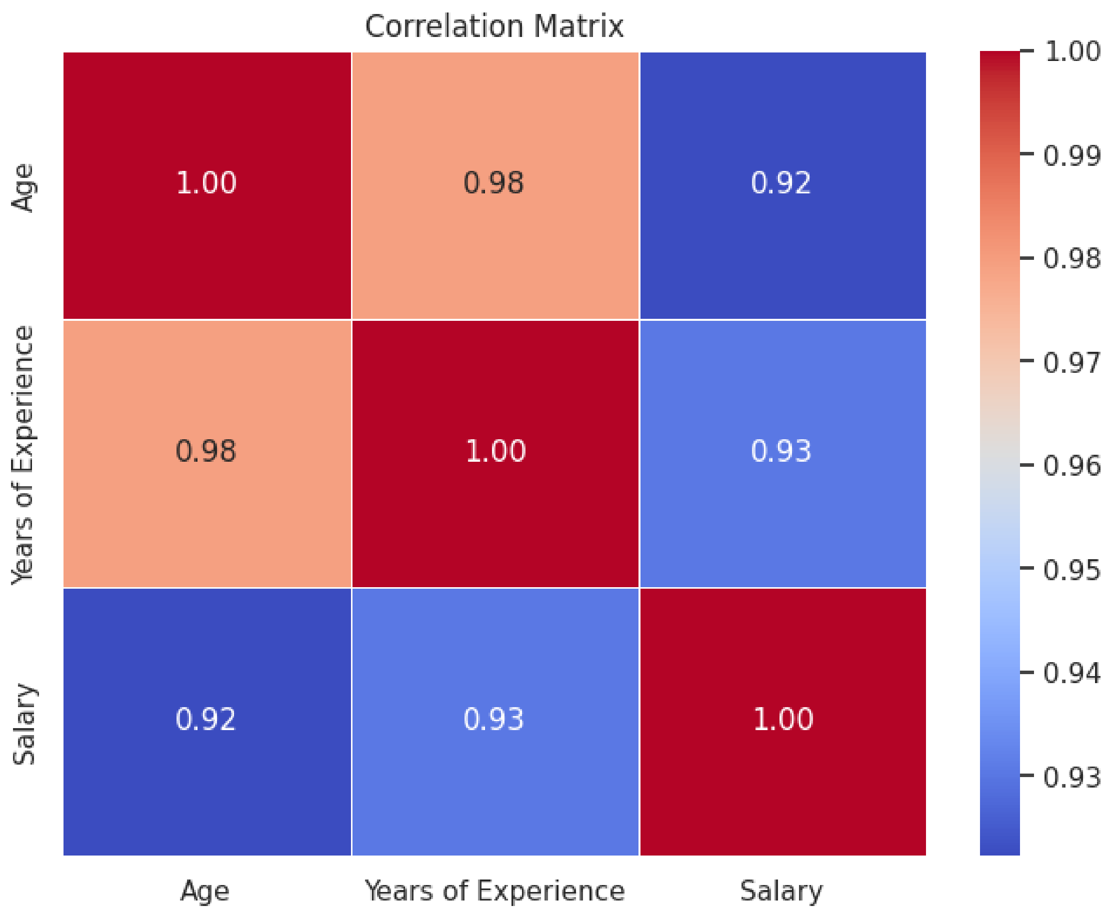
```
<ipython-input-23-80e7303caab0>:11: FutureWarning:

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.violinplot(x='Gender', y='Age', data=df, ax=axes[0], palette='pastel')
<ipython-input-23-80e7303caab0>:14: 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.violinplot(x='Education Level', y='Years of Experience', data=df,
ax=axes[1], palette='pastel')
```



[ ]:
```
#Calculate correlation coefficients between numerical variables.

# Selecting only numerical columns for correlation analysis
numerical_columns = df[['Age', 'Years of Experience', 'Salary']]

# Calculate correlation coefficients
correlation_matrix = numerical_columns.corr()
```

```python
# Print correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)

# If you want to visualize the correlation matrix as a heatmap using seaborn
import seaborn as sns
import matplotlib.pyplot as plt

# Set the style for seaborn
sns.set(style="white")

# Create a heatmap of the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",␣
↪linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```

```
Correlation Matrix:
                         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
```



[ ]:
```python
# Drop the non-required columns

# List of non-required columns to be dropped
columns_to_drop = ['Gender', 'Education Level']

# Drop the specified columns
df_dropped = df.drop(columns=columns_to_drop)

# Display the modified DataFrame
print(df_dropped)
```

```
      Age                      Job Title  Years of Experience    Salary
0    32.0              Software Engineer                  5.0   90000.0
1    28.0                   Data Analyst                  3.0   65000.0
2    45.0                 Senior Manager                 15.0  150000.0
3    36.0                Sales Associate                  7.0   60000.0
4    52.0                       Director                 20.0  200000.0
..    …                              …                   …         …
370  35.0       Senior Marketing Analyst                  8.0   85000.0
```

```
371  43.0         Director of Operations                 19.0  170000.0
372  29.0        Junior Project Manager                   2.0   40000.0
373  34.0  Senior Operations Coordinator                  7.0   90000.0
374  44.0        Senior Business Analyst                 15.0  150000.0

[375 rows x 4 columns]
```

[ ]:
```
# Re-arrange columns / features
desired_columns_order = ['Age', 'Gender', 'Education Level', 'Job Title',␣
 ↪'Years of Experience', 'Salary']

# Reorder columns in the DataFrame
df_rearranged = df[desired_columns_order]

# Display the rearranged DataFrame
print(df_rearranged)
```

```
      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]
```

[ ]:
```
#Separate the features (X and y)
# Separate features (X) and target variable (y)
```

```
X = df.drop('Salary', axis=1)  # Drop the 'Salary' column to get the features
y = df['Salary']  # 'Salary' is the target variable

# Display the first few rows of X and y
print("Features (X):")
print(X.head())

print("\nTarget Variable (y):")
print(y.head())
```

```
Features (X):
    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

Target Variable (y):
0     90000.0
1     65000.0
2    150000.0
3     60000.0
4    200000.0
Name: Salary, dtype: float64
```

[ ]:
```
#Perform Standardization:Apply Standard Scalar / MinMax Scalar / Robust Scalar
 ↪based on the
#requirement to standardize the data

# Extract numerical columns for standardization
numerical_columns = ['Age', 'Years of Experience', 'Salary']

# Create a DataFrame containing only numerical columns
numerical_df = df[numerical_columns]

# Initialize the scalers
standard_scaler = StandardScaler()
minmax_scaler = MinMaxScaler()
robust_scaler = RobustScaler()

# Standardize the data using each scaler
standardized_data_standard = standard_scaler.fit_transform(numerical_df)
standardized_data_minmax = minmax_scaler.fit_transform(numerical_df)
standardized_data_robust = robust_scaler.fit_transform(numerical_df)

# Convert the standardized data back to a DataFrame
```

```python
standardized_df_standard = pd.DataFrame(standardized_data_standard,␣
↪columns=numerical_columns)
standardized_df_minmax = pd.DataFrame(standardized_data_minmax,␣
↪columns=numerical_columns)
standardized_df_robust = pd.DataFrame(standardized_data_robust,␣
↪columns=numerical_columns)

# Display the standardized DataFrames
print("Standard Scaler:")
print(standardized_df_standard.head())

print("\nMinMax Scaler:")
print(standardized_df_minmax.head())

print("\nRobust Scaler:")
print(standardized_df_robust.head())
```

```
Standard Scaler:
        Age  Years of Experience    Salary
0 -0.769398            -0.768276 -0.219559
1 -1.336003            -1.073702 -0.738498
2  1.072068             0.758859  1.025892
3 -0.202793            -0.462849 -0.842285
4  2.063627             1.522426  2.063768

MinMax Scaler:
        Age  Years of Experience    Salary
0  0.300000                 0.20  0.359103
1  0.166667                 0.12  0.258963
2  0.733333                 0.60  0.599439
3  0.433333                 0.28  0.238935
4  0.966667                 0.80  0.799720

Robust Scaler:
        Age  Years of Experience    Salary
0 -0.307692            -0.363636 -0.058824
1 -0.615385            -0.545455 -0.352941
2  0.692308             0.545455  0.647059
3  0.000000            -0.181818 -0.411765
4  1.230769             1.000000  1.235294
```

Conventional-way of PCA:

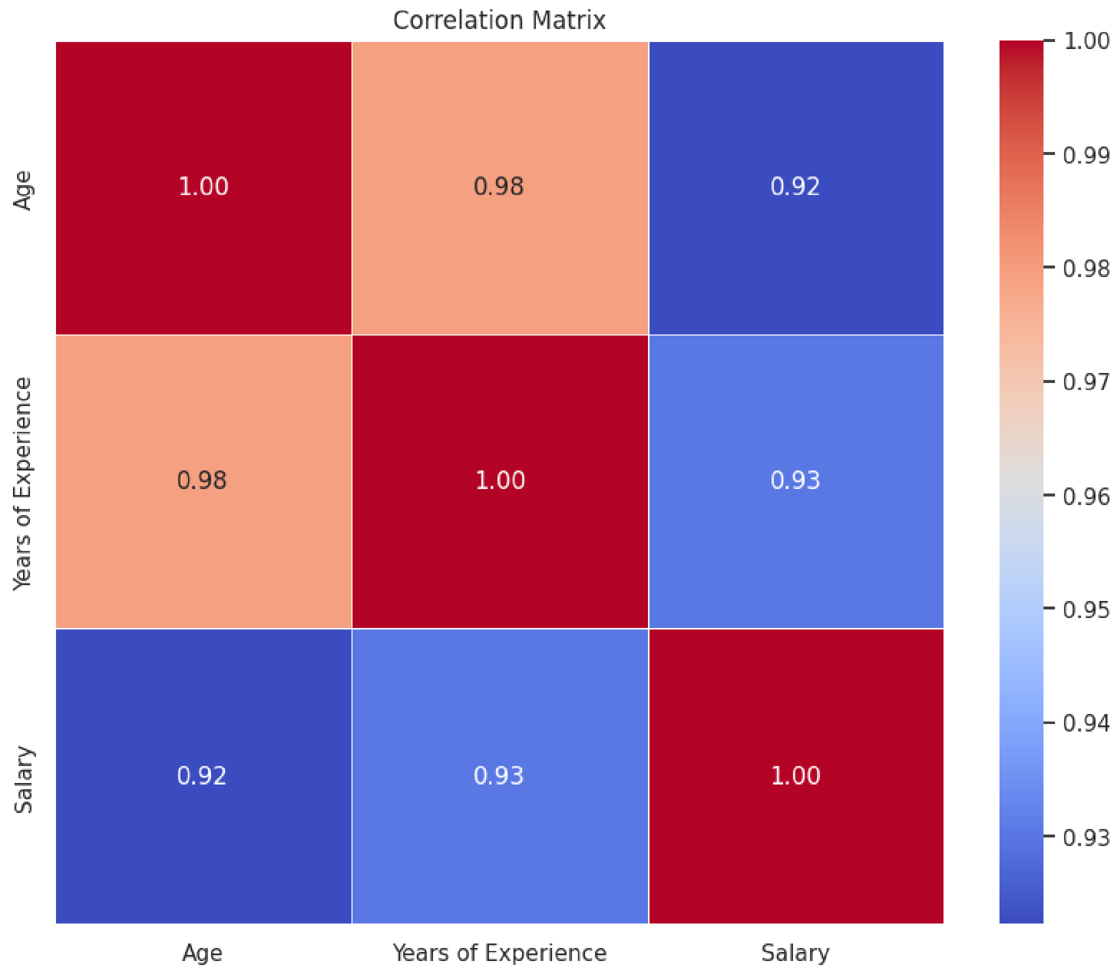
[ ]:
```python
#Compute Eigenvectors and Eigenvalues

# Drop rows with missing values
df_cleaned = df.dropna()
```

```python
# Extract numerical columns for PCA
numerical_columns = ['Age', 'Years of Experience', 'Salary']

# Create a DataFrame containing only numerical columns
numerical_df = df_cleaned[numerical_columns]

# Standardize the data (optional but often recommended before PCA)
standardized_data = (numerical_df - numerical_df.mean()) / numerical_df.std()

# Check for NaN or Inf values after standardization
if np.any(np.isnan(standardized_data)) or np.any(np.isinf(standardized_data)):
    raise ValueError("NaN or Inf values found in the standardized data. Handle␣
 ↪missing values before PCA.")

# Compute the covariance matrix
covariance_matrix = np.cov(standardized_data, rowvar=False)

# Compute eigenvectors and eigenvalues
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# Display the results
print("Eigenvalues:")
print(eigenvalues)

print("\nEigenvectors:")
print(eigenvectors)
```

```
Eigenvalues:
[2.88809241 0.02056164 0.09134595]

Eigenvectors:
[[-0.58016321 -0.67969213 -0.4488087 ]
 [-0.58175427  0.73145357 -0.35572129]
 [-0.57006369 -0.05471997  0.81977626]]
```

[ ]:
```python
#Check the Correlation between features

# Calculate the correlation matrix
correlation_matrix = df.corr()

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',␣
 ↪linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```

```
<ipython-input-38-631b40c7a995>: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()
```



[ ]:
```python
#Select Principal components with Covariance
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming your DataFrame is named df

# Handle missing values (fill with the mean of each column)
df_filled = df.fillna(df.mean())

# Selecting numerical columns for PCA
```

```python
numerical_columns = df_filled.select_dtypes(include=['float64']).columns

# Extract numerical data
data = df_filled[numerical_columns]

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

# Apply PCA with covariance
pca = PCA()
principal_components = pca.fit_transform(scaled_data)

# Create a DataFrame with principal components
columns = [f'PC{i+1}' for i in range(principal_components.shape[1])]
principal_df = pd.DataFrame(data=principal_components, columns=columns)

# Display the explained variance ratio for each principal component
print("Explained Variance Ratio:")
print(pca.explained_variance_ratio_)

# Optional: You can also print the cumulative explained variance
print("\nCumulative Explained Variance:")
print(pca.explained_variance_ratio_.cumsum())

# Optional: Print the loadings (coefficients) of each variable on each␣
 ↪principal component
loadings_df = pd.DataFrame(data=pca.components_.T, index=data.columns,␣
 ↪columns=columns)
print("\nLoadings:")
print(loadings_df)
```

```
Explained Variance Ratio:
[0.96269747 0.03044865 0.00685388]

Cumulative Explained Variance:
[0.96269747 0.99314612 1.        ]

Loadings:
                          PC1       PC2       PC3
Age                  0.580163 -0.448809 -0.679692
Years of Experience  0.581754 -0.355721  0.731454
Salary               0.570064  0.819776 -0.054720
```

```
<ipython-input-48-436792ddddc3>:9: FutureWarning: The default value of
numeric_only in DataFrame.mean is deprecated. In a future version, it will
default to False. In addition, specifying 'numeric_only=None' is deprecated.
Select only valid columns or specify the value of numeric_only to silence this
```

```
warning.
  df_filled = df.fillna(df.mean())
```

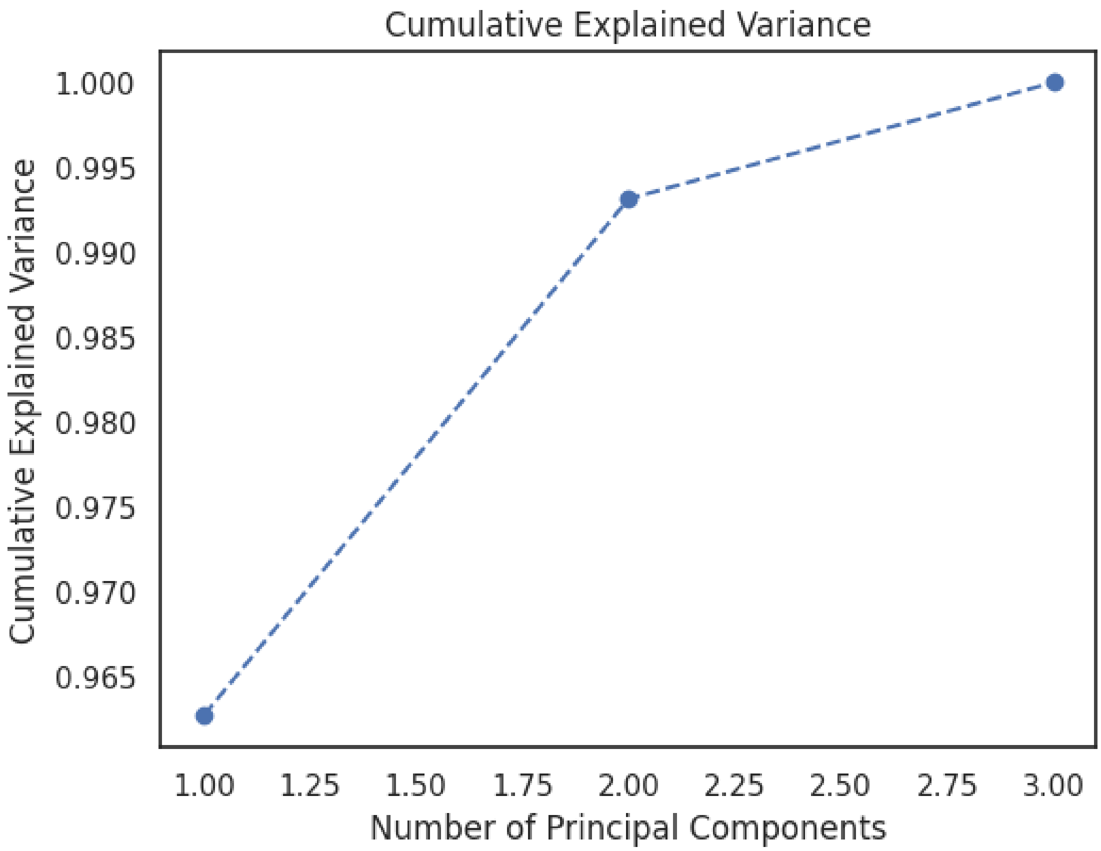
[ ]:
```python
#Projection Matrix
import numpy as np

# Assuming your DataFrame is named df

# Extract numerical columns for simplicity
numerical_data = df[['Age', 'Years of Experience', 'Salary']].values

# Handle NaN values by replacing them with the mean of each column
numerical_data = np.nan_to_num(numerical_data, nan=np.nanmean(numerical_data,␣
 ↪axis=0))

# Calculate the covariance matrix
covariance_matrix = np.cov(numerical_data, rowvar=False)

# Perform eigendecomposition of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# Sort eigenvectors based on eigenvalues
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvectors_sorted = eigenvectors[:, sorted_indices]

# Choose the number of dimensions for projection (e.g., 2 for 2D projection)
num_dimensions = 2

# Create the projection matrix
projection_matrix = eigenvectors_sorted[:, :num_dimensions]

# Project the data onto the subspace defined by the projection matrix
projected_data = np.dot(numerical_data, projection_matrix)

# Display the projection matrix
print("Projection Matrix:")
print(projection_matrix)
```

```
Projection Matrix:
[[ 1.35158649e-04  7.57625050e-01]
 [ 1.26455831e-04  6.52690010e-01]
 [ 9.99999983e-01 -1.84936039e-04]]
```

[ ]:
```python
#Calculate PCA with Scikit-learn

# Extracting numerical columns for PCA
numerical_columns = ['Age', 'Years of Experience', 'Salary']
```

```python
data_for_pca = df[numerical_columns]

# Handle missing values by imputing with mean (you can choose a different␣
↪strategy)
imputer = SimpleImputer(strategy='mean')
data_for_pca_imputed = pd.DataFrame(imputer.fit_transform(data_for_pca),␣
↪columns=numerical_columns)

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

# Apply PCA
pca = PCA()
pca_result = pca.fit_transform(scaled_data)

# Create a DataFrame with the PCA results
pca_df = pd.DataFrame(data=pca_result, columns=['PC1', 'PC2', 'PC3'])

# Print explained variance ratios
explained_variance_ratios = pca.explained_variance_ratio_
print("Explained Variance Ratios:")
print(explained_variance_ratios)

# Plot the cumulative explained variance
cumulative_variance = explained_variance_ratios.cumsum()
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,␣
↪marker='o', linestyle='--')
plt.title('Cumulative Explained Variance')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```

```
Explained Variance Ratios:
[0.96269747 0.03044865 0.00685388]
```



Interpret the results with your conclusion to substantiate that the principal components shall provide equivalent understanding of all the features.

Explained Variance Ratios:

Higher values indicate more substantial contributions of corresponding principal components to the total variance, reflecting their importance in capturing information from the original features. Cumulative Explained Variance Plot:

Reveals diminishing returns as additional principal components are considered. Common practice is to retain components until a point on the x-axis where further inclusion doesn't significantly enhance cumulative explained variance. This chosen point represents a trade-off between dimensionality reduction and retaining a high percentage of total variance.

Conclusion:

If a small number of principal components can explain a high percentage of the total variance, it suggests that these components provide a condensed representation of the original features. This can be valuable for dimensionality reduction, simplifying the analysis while retaining most of the information.

Look at the cumulative explained variance plot to determine how many principal components are needed to achieve a satisfactory level of data representation. If a small subset of components can capture a significant portion of the variance, it supports the idea that these components offer an equivalent understanding of the original features.

PCA is particularly useful when dealing with high-dimensional data, and it allows you to focus on the most informative components while discarding less significant ones. The interpretation of these components in the context of your specific data domain is crucial for extracting meaningful insights.