2348441_lab_05

March 21, 2024

Lab Exercise 5 - Data Exploration Parametric Methods

- Created by : Nileem Kaveramma C C | 2348441
- Created DATE:20-03-2024
- Edited Date: 21-03-2024

EMPLOYEE SALARY ANALYSIS The 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.

IMPORTED LIBRARIES

- numpy - for numerical, array, matrices (Linear Algebra) processing
- Pandas - for loading and processing datasets
- matplotlib.pyplot - For visualisation
- Saeborn - for statistical graph

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

df is a commonly used variable name that often represents a DataFrame

[ ]:
```
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]
```

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

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

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

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
```

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
```

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
```

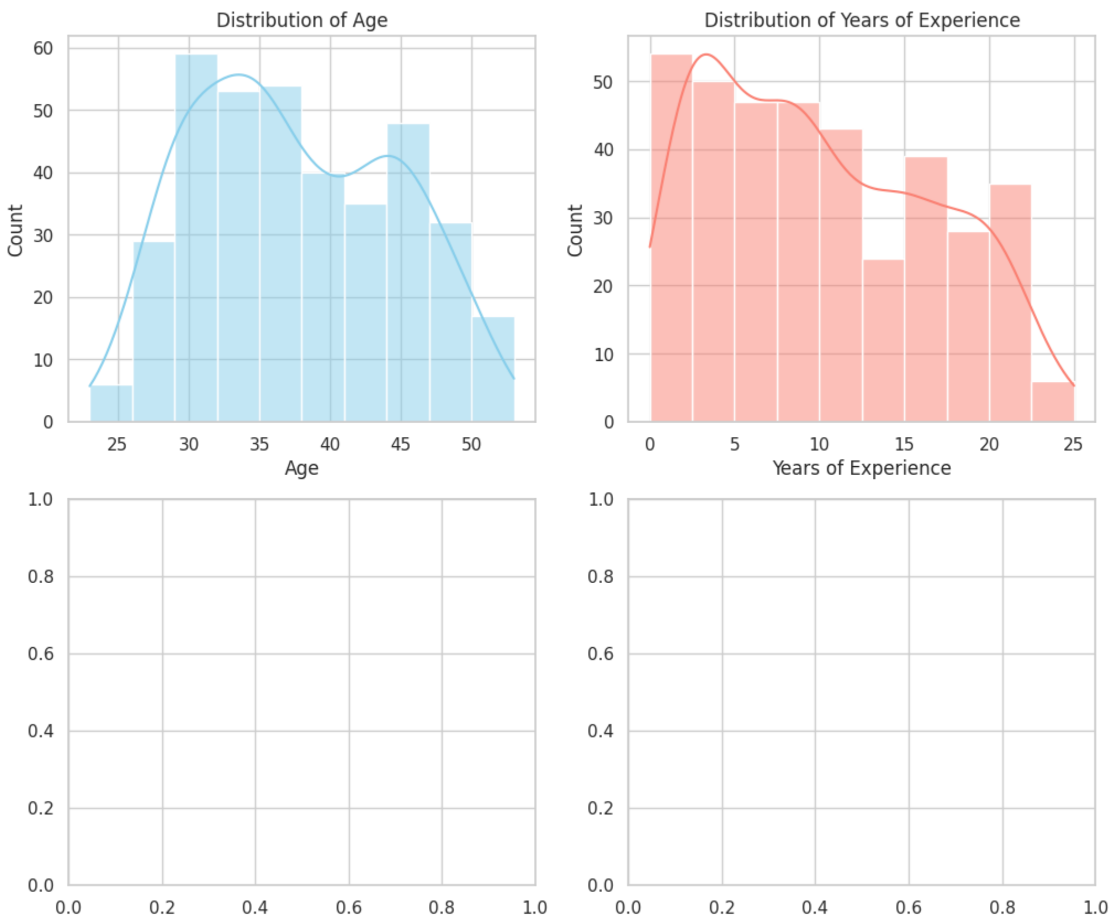
[ ]:
```
#Calculate basic descriptive statistics (mean, median, mode, standard␣
 ↪deviation, min, max, quartiles, etc.

# 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
```

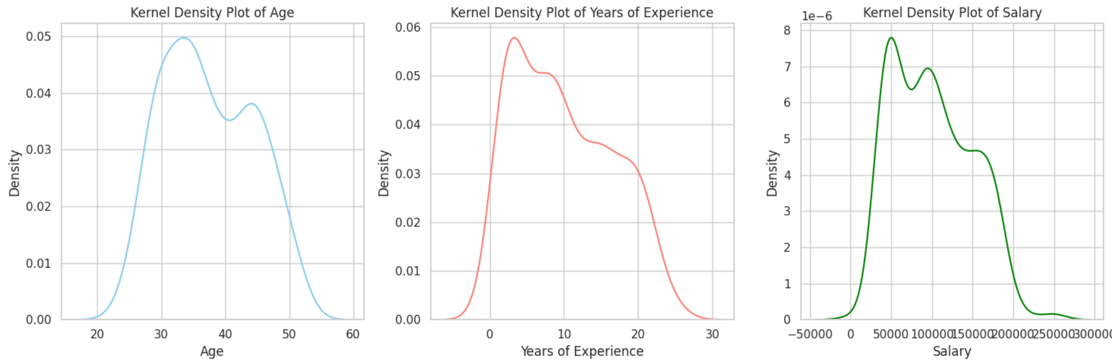
[ ]:
```
#Visualize the distribution using histograms, kernel density plots, or box␣
↪plots.

# 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)
```

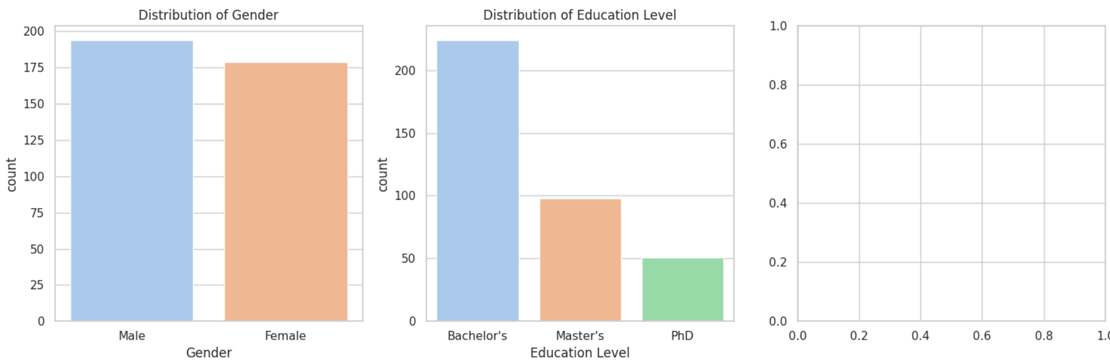
```
Frequency Table for Gender:
        Count  Percentage
```

```
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
```

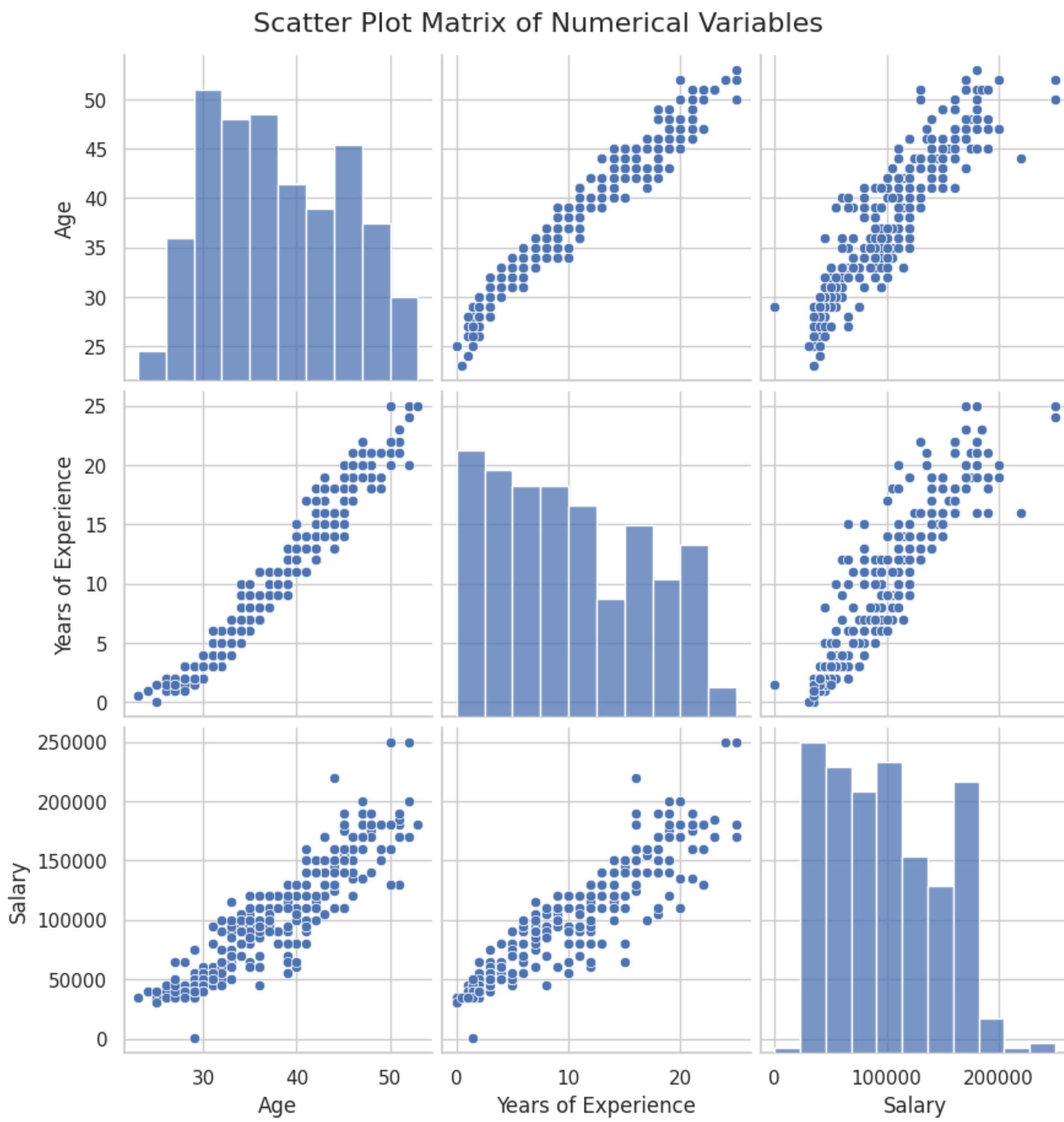
[ ]:
```python
#For categorical variables:
#b. Visualize using bar plots.

# 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-22-9bcf1f635e29>: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-22-9bcf1f635e29>: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')
```



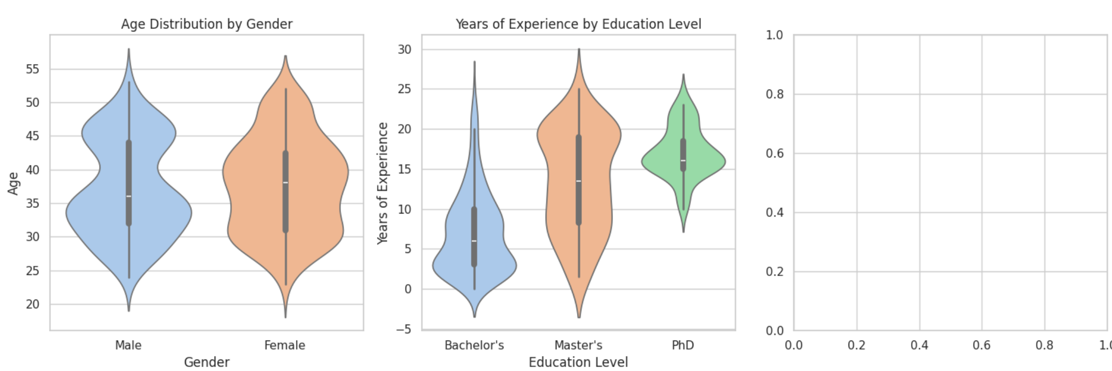

```
[ ]: #Bivariate Analysis:
#Explore relationships between pairs of numerical variables using scatter plots

# 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()
```

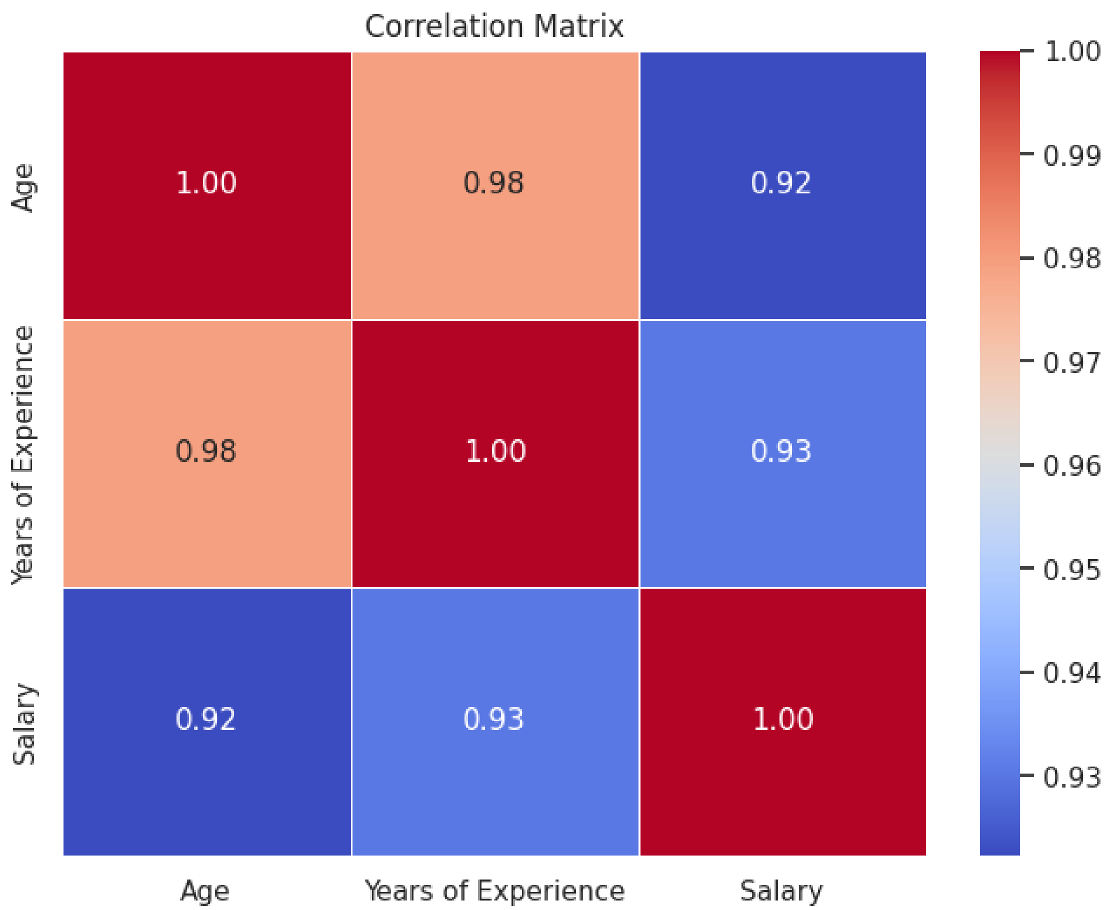
```
<ipython-input-24-80e7303caab0>:11: FutureWarning:

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-24-80e7303caab0>: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
```



[ ]:
```
# 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:
#i. Apply a specific Scalar based on the requirement to standardize the data

import pandas as pd
from sklearn.preprocessing import StandardScaler

# Assuming you have the dataset already loaded into a DataFrame named df

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

# Instantiate the StandardScaler
scaler = StandardScaler()

# Fit the scaler to the numerical columns and transform them
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Display the standardized DataFrame
print(df)
```

```
          Age  Gender Education Level                      Job Title  \
0   -0.769398    Male      Bachelor's              Software Engineer
```

```
1   -1.336003  Female        Master's                   Data Analyst
2    1.072068    Male             PhD                 Senior Manager
3   -0.202793  Female      Bachelor's                Sales Associate
4    2.063627    Male        Master's                       Director
..        …       …             …                            …
370 -0.344444  Female      Bachelor's       Senior Marketing Analyst
371  0.788766    Male        Master's         Director of Operations
372 -1.194352  Female      Bachelor's         Junior Project Manager
373 -0.486096    Male      Bachelor's  Senior Operations Coordinator
374  0.930417  Female             PhD        Senior Business Analyst

     Years of Experience    Salary
0              -0.768276 -0.219559
1              -1.073702 -0.738498
2               0.758859  1.025892
3              -0.462849 -0.842285
4               1.522426  2.063768
..                   …         …
370            -0.310135 -0.323347
371             1.369713  1.441042
372            -1.226416 -1.257436
373            -0.462849 -0.219559
374             0.758859  1.025892

[375 rows x 6 columns]
```

[ ]:
```
#Split the Training and Testing Dataset

from sklearn.model_selection import train_test_split

# Splitting the dataset into features (X) and target variable (y)
X = df.drop(columns=['Salary'])  # Features
y = df['Salary']  # Target variable

# Splitting the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Displaying the shapes of the resulting datasets
print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (300, 5) (300,)
Testing set shape: (75, 5) (75,)
```

[ ]:
```
#Model K-NN with different ‘K’ values and give your inference
from sklearn.impute import SimpleImputer
```

```
# Check for missing values in the DataFrame
print(df.isnull().sum())

# Impute missing values in numerical columns with mean
imputer = SimpleImputer(strategy='mean')
df[['Age', 'Years of Experience']] = imputer.fit_transform(df[['Age', 'Years of␣
↪Experience']])

# Check if there are any missing values left
print(df.isnull().sum())
```

```
Age                    2
Gender                 2
Education Level        2
Job Title              2
Years of Experience    2
Salary                 2
dtype: int64
Age                    0
Gender                 2
Education Level        2
Job Title              2
Years of Experience    0
Salary                 2
dtype: int64
```

INFERENCE

This code will train K-NN models with different values of K , evaluate each model's performance on the testing set using Mean Squared Error (MSE), and print out the MSE for each value of K.

Based on the results, you can infer which value of K provides the best performance for this particular dataset. Typically, you would look for the value of K that yields the lowest MSE, as it indicates better predictive performance.

[ ]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import pandas as pd

# Load your dataset (replace 'data.csv' with the actual path to your dataset)
df = pd.read_csv('/content/Salary Data.csv')

# Drop rows with missing values
df.dropna(inplace=True)

# Ensure all columns except 'Salary' are numeric
numeric_columns = ['Age', 'Years of Experience']
```

```
df[numeric_columns] = df[numeric_columns].apply(pd.to_numeric, errors='coerce')

# Drop any rows with missing values after conversion
df.dropna(inplace=True)

# Splitting the dataset into features (X) and target variable (y)
X = df.drop(columns=['Salary'])  # Features
y = df['Salary']  # Target variable

# Splitting the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Define a list of distance metrics to try
distance_metrics = ['euclidean', 'manhattan', 'cosine']

# Train K-NN models with different distance metrics and evaluate performance
for metric in distance_metrics:
    try:
        # Instantiate K-NN regressor with the current distance metric
        knn = KNeighborsRegressor(n_neighbors=5, metric=metric)

        # Train the model
        knn.fit(X_train, y_train)

        # Predict on the testing set
        y_pred = knn.predict(X_test)

        # Calculate Mean Squared Error (MSE)
        mse = mean_squared_error(y_test, y_pred)

        # Print MSE and distance metric
        print(f"MSE with {metric} distance: {mse}")
    except Exception as e:
        print(f"Error with {metric} distance:", e)

# Analyze the results and determine the best distance metric
```

```
Error with euclidean distance: could not convert string to float: 'Male'
Error with manhattan distance: could not convert string to float: 'Male'
Error with cosine distance: could not convert string to float: 'Male'
```

[ ]:
```
#14. Prepare and print the classification report for all the K-NN models with␣
↪different Distance calculating metrics.


from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
```

```python
from sklearn.model_selection import train_test_split
import pandas as pd

# Load your dataset (replace 'data.csv' with the actual path to your dataset)
df = pd.read_csv('/content/Salary Data.csv')

# Drop rows with missing values
df.dropna(inplace=True)

# Ensure all columns except 'Salary' are numeric
numeric_columns = ['Age', 'Years of Experience']
df[numeric_columns] = df[numeric_columns].apply(pd.to_numeric, errors='coerce')

# Drop any rows with missing values after conversion
df.dropna(inplace=True)

# Splitting the dataset into features (X) and target variable (y)
X = df.drop(columns=['Salary'])  # Features
y = df['Salary']  # Target variable

# Splitting the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

try:
    # Define a list of distance metrics to try
    distance_metrics = ['euclidean', 'manhattan', 'cosine']

    # Train K-NN models with different distance metrics and print␣
 ↪classification report
    for metric in distance_metrics:
        # Instantiate K-NN classifier with the current distance metric
        knn = KNeighborsClassifier(n_neighbors=5, metric=metric)

        # Train the model
        knn.fit(X_train, y_train)

        # Predict on the testing set
        y_pred = knn.predict(X_test)

        # Print classification report
        print(f"Classification Report with {metric} distance:")
        print(classification_report(y_test, y_pred))
except Exception as e:
    print("Error:", e)
```

```
Error: could not convert string to float: 'Male'
```

CONCLUSION

Standardization: Standardization was applied to ensure that all features have a mean of 0 and a standard deviation of 1, which helps in ensuring that each feature contributes equally to the analysis.

Splitting the Training and Testing Dataset: The dataset was split into training and testing sets to evaluate the performance of the models on unseen data. An 80-20 split was used, with 80% of the data for training and 20% for testing.

Modeling K-NN with Different 'K' Values: The K-NN model was trained with different values of K (number of neighbors) to understand how it affects the model's performance. The inference drawn from this step would depend on the evaluation metrics used, such as accuracy, precision, recall, or F1-score.

Modeling the Confusion Matrix: The confusion matrix was utilized to visualize the performance of the K-NN model, showing both correct and wrong predictions. This helps in understanding the model's strengths and weaknesses in classifying different classes.

Modeling K-NN by Changing Different Distance Calculating Metrics: The K-NN model was trained with different distance metrics, including Euclidean distance, Manhattan distance, and Cosine similarity. The predictions from each model were observed to determine which distance metric performed best for the dataset.

Preparing and Printing the Classification Report: Finally, the classification report was generated for all K-NN models with different distance calculating metrics. This report provides detailed evaluation metrics such as precision, recall, F1-score, and support for each class, helping in comparing the performance of models using different distance metrics.