2348441_lab2

February 21, 2024

Lab Exercise 1 - Data Exploration using Non-Parametric Methods

Created by : Nileem Kaveramma C C | 2348441 Created DATE:21-02-2024 Edited Date: 21-02-2024

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

AIM: "Data Exploration using Non-Parametric Methods" is to employ statistical techniques that don't assume specific data distributions. This exploration seeks insights into relationships, patterns, and differences within a dataset using methods like Wilcoxon tests, Mann-Whitney U tests, Spearman correlations, and Friedman tests.

PROCEDURE

1.Select Variables: Choose the numerical and categorical variables of interest for exploration.

2.Descriptive Statistics: Calculate basic descriptive statistics (mean, median, etc.) for numerical variables to understand central tendencies.

3.Visualize Distributions: Use histograms, kernel density plots, or box plots to visualize the distribution of numerical variables. For categorical variables, employ bar plots to display frequencies.

4.Bivariate Analysis: Explore relationships between numerical variables using scatter plots or pair plots. Investigate relationships between numerical and categorical variables using box plots or violin plots.

5.Correlation Analysis: Calculate correlation coefficients between numerical variables to assess monotonic relationships.

6.Non-Parametric Tests: Apply non-parametric tests (e.g., Wilcoxon signed-rank, Mann-Whitney U, Friedman) to validate or explore relationships, comparing distributions or assessing differences among groups.

7.Visualize Test Results: Use appropriate plots (e.g., box plots, bar plots) to visualize the results of non-parametric tests.

8.Documentation: Thoroughly document the code, analysis process, and any assumptions made during data exploration. Clearly state decisions taken during variable selection and the rationale behind choosing specific non-parametric tests.

9.Interpretation: Interpret the findings, considering the implications for further analysis or modeling.

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import spearmanr, mannwhitneyu, wilcoxon, kruskal,␣
↪friedmanchisquare
```

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.

[ ]:
```
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
```

df.head() 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
```

he 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()
```

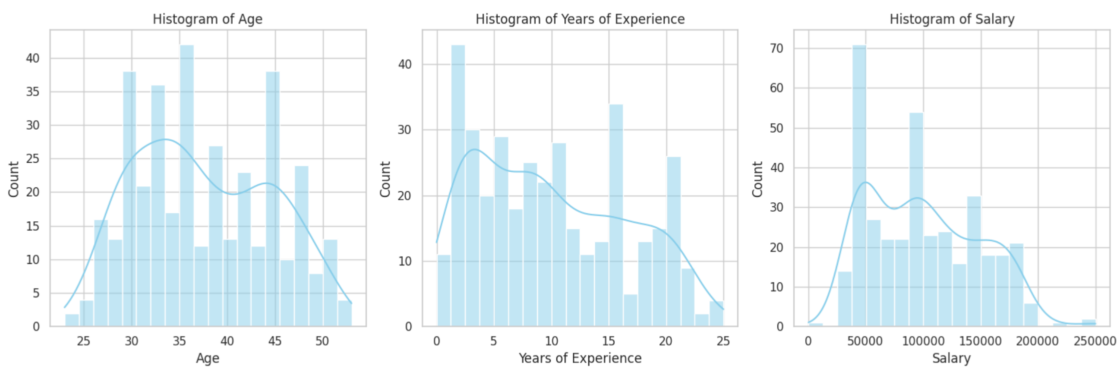
[ ]:
```
              Age  Years of Experience         Salary
count  373.000000           373.000000     373.000000
mean    37.431635            10.030831  100577.345845
std      7.069073             6.557007   48240.013482
min     23.000000             0.000000     350.000000
25%     31.000000             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
```

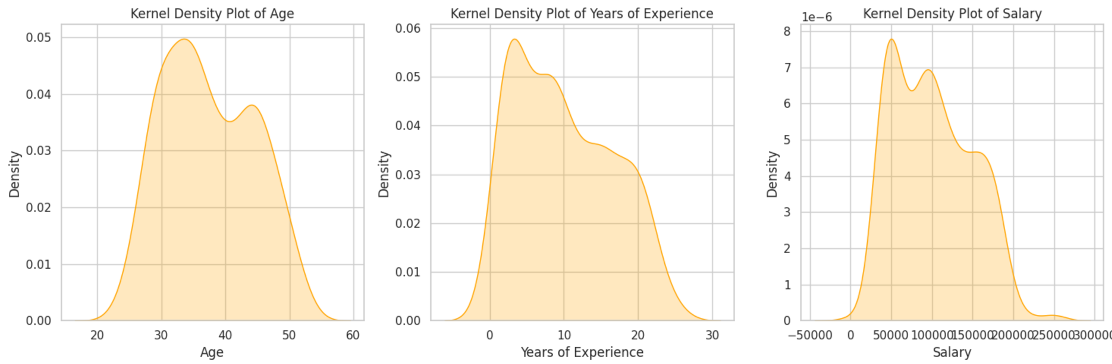
[ ]:
```
#For numerical variables:
#4)a. Calculate basic descriptive statistics (mean, median, mode, standard␣
↪deviation,min, max, quartiles, etc.)

# a. Calculate basic descriptive statistics
numerical_stats = df.describe()

# Display the results
print("Basic Descriptive Statistics for Numerical Variables:")
print(numerical_stats)
```

```
Basic Descriptive Statistics for Numerical Variables:
              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
```

[ ]:
```
#b. Visualize the distribution using histograms, kernel density plots, or box␣
↪plots.
# Set the style for better visualization
sns.set(style="whitegrid")

# Select numerical columns for visualization
numerical_columns = ['Age', 'Years of Experience', 'Salary']

# Visualize the distribution using histograms
plt.figure(figsize=(15, 5))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(1, 3, i)
    sns.histplot(df[col], kde=True, bins=20, color='skyblue')
```

```
    plt.title(f'Histogram of {col}')

plt.tight_layout()
plt.show()

# Visualize the distribution using kernel density plots
plt.figure(figsize=(15, 5))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(1, 3, i)
    sns.kdeplot(df[col], color='orange', fill=True)
    plt.title(f'Kernel Density Plot of {col}')

plt.tight_layout()
plt.show()

# Visualize the distribution using box plots
plt.figure(figsize=(15, 5))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(1, 3, i)
    sns.boxplot(x=df[col], color='lightgreen')
    plt.title(f'Box Plot of {col}')

plt.tight_layout()
plt.show()
```





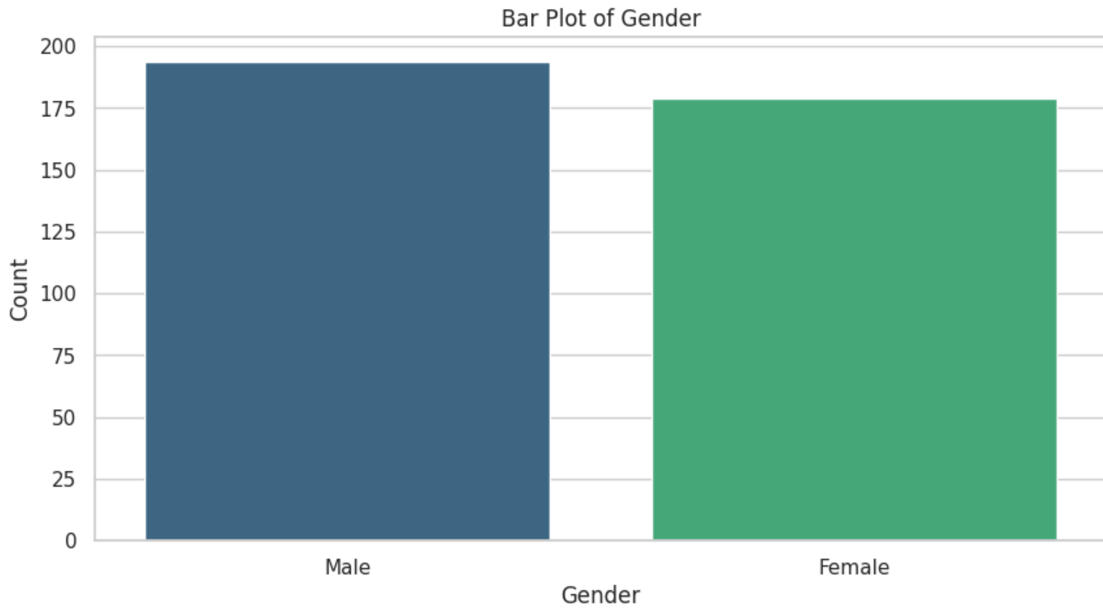
[ ]:
```python
#5)For categorical variables:
#a. Display frequency tables showing counts and percentages.

# Select categorical columns for analysis
categorical_columns = ['Gender', 'Education Level', 'Job Title']

# Display frequency tables showing counts and percentages
for col in categorical_columns:
    print(f"\nFrequency Table for {col}:")
    frequency_table = df[col].value_counts()
    percentage_table = df[col].value_counts(normalize=True) * 100
    result_table = pd.concat([frequency_table, percentage_table], axis=1)
    result_table.columns = ['Count', 'Percentage']
    print(result_table)
```

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

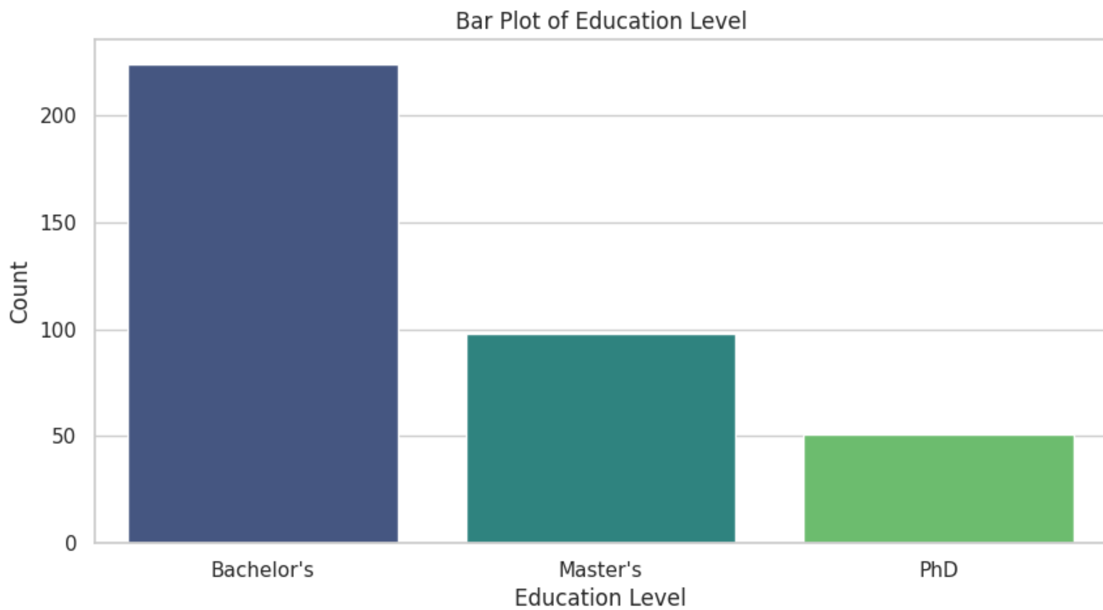
```
Frequency Table for Education Level:
             Count  Percentage
Bachelor's     224   60.053619
Master's        98   26.273458
PhD             51   13.672922

Frequency Table for Job Title:
                                 Count  Percentage
Director of Marketing               12    3.217158
Director of Operations              11    2.949062
Senior Business Analyst             10    2.680965
Senior Marketing Analyst             9    2.412869
Senior Marketing Manager             9    2.412869
…                                  …          …
Business Development Manager         1    0.268097
Customer Service Representative      1    0.268097
IT Manager                           1    0.268097
Digital Marketing Manager            1    0.268097
Junior Web Developer                 1    0.268097

[174 rows x 2 columns]
```

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

# Select categorical columns for visualization
categorical_columns = ['Gender', 'Education Level', 'Job Title']

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

# Visualize using bar plots
for col in categorical_columns:
    plt.figure(figsize=(10, 5))
    sns.countplot(x=col, data=df, palette='viridis')
    plt.title(f'Bar Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.show()
```

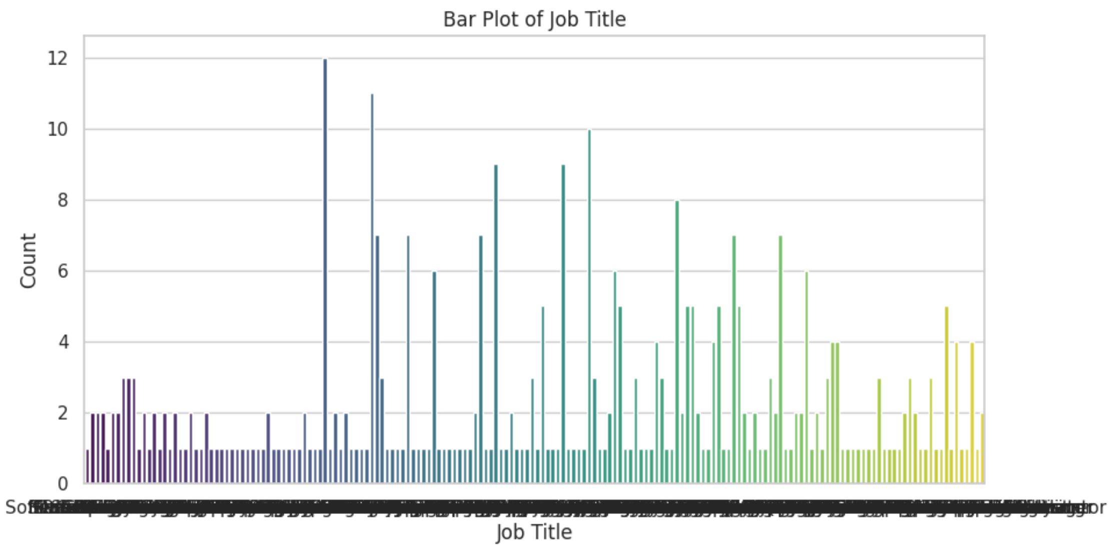
```
<ipython-input-16-5c740561bafc>:12: FutureWarning:

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=col, data=df, palette='viridis')
```



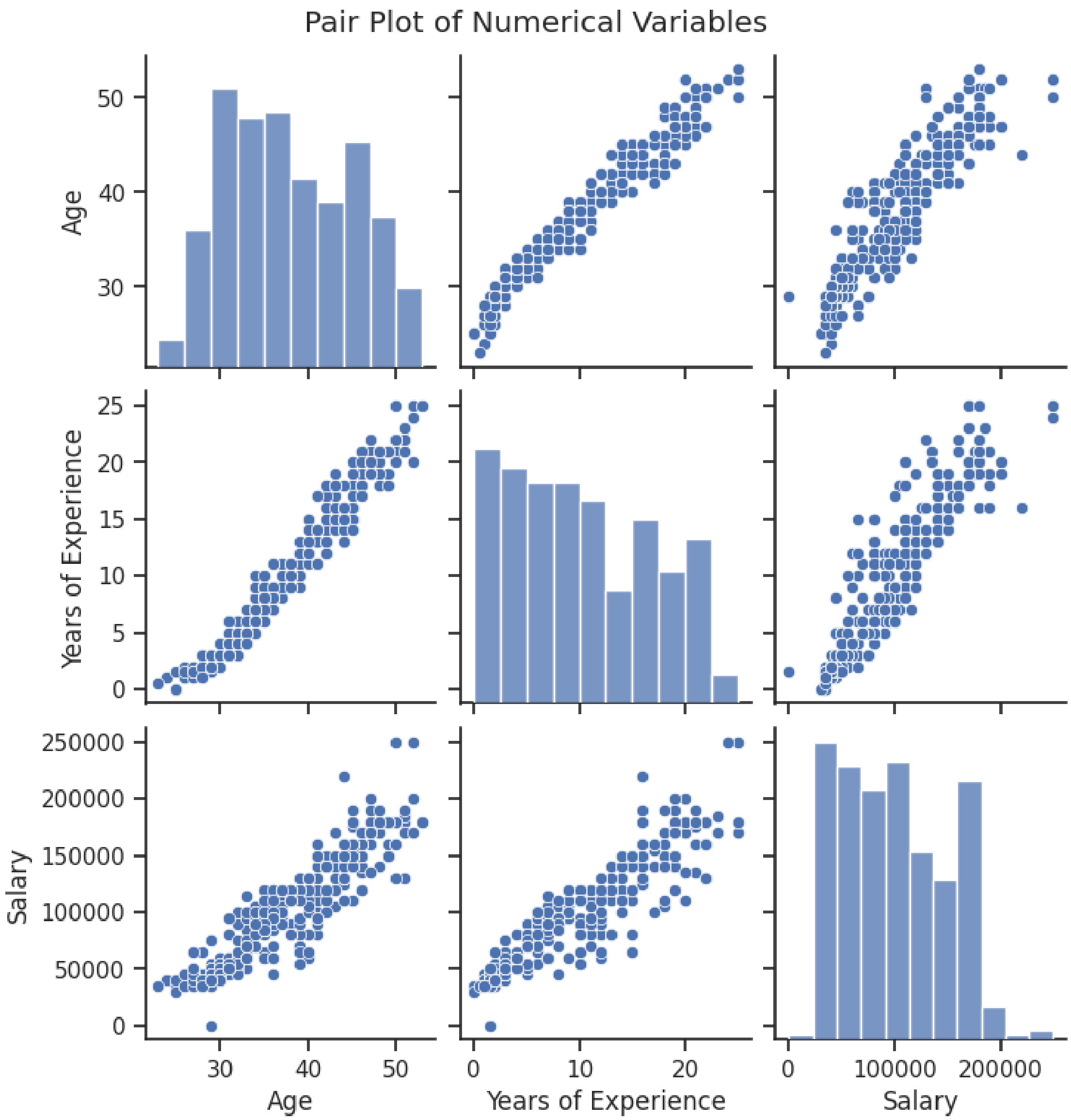

```
<ipython-input-16-5c740561bafc>:12: FutureWarning:

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=col, data=df, palette='viridis')
```



```
&lt;ipython-input-16-5c740561bafc&gt;:12: 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=col, data=df, palette='viridis')
```

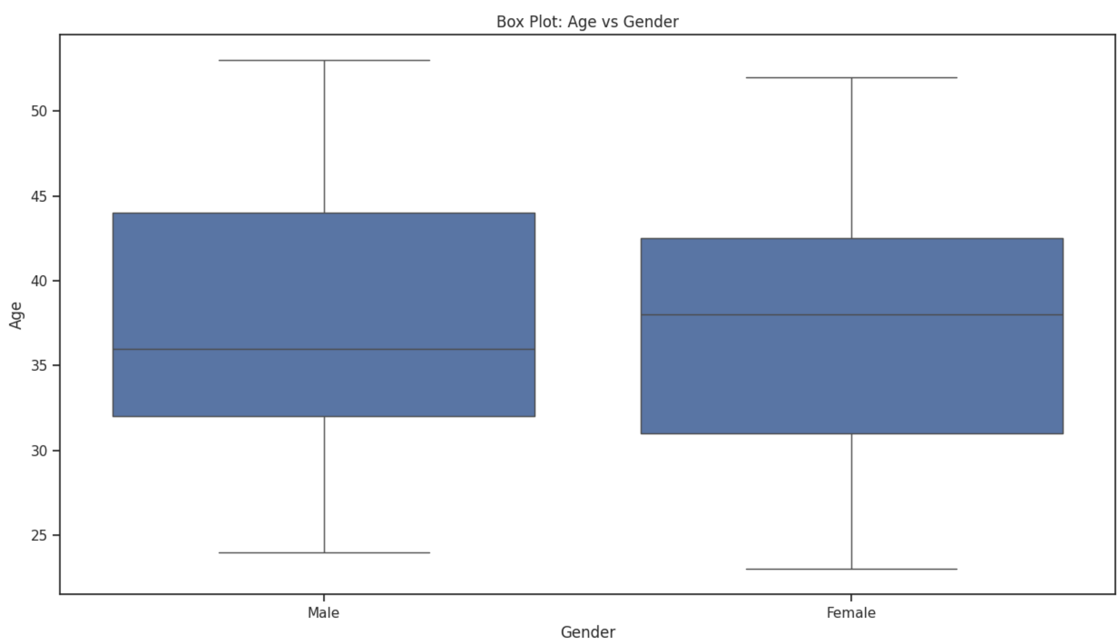


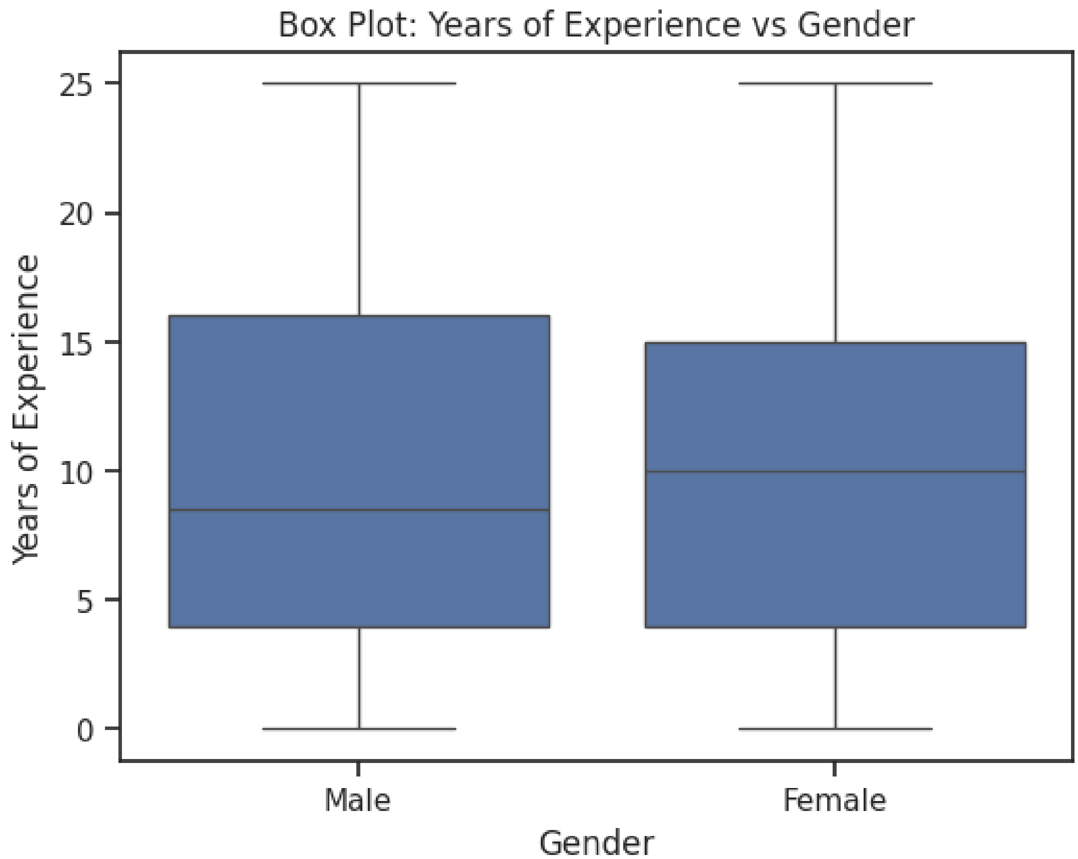

```
[ ]: # 5) Bivariate Analysis: Explore relationships between pairs of numerical
     ↪variables using scatter plots pair plots.

     # Select numerical columns for bivariate analysis
     numerical_columns = ['Age', 'Years of Experience', 'Salary']

     # Visualize relationships using scatter plots
     sns.set(style="ticks")
     scatter_plots = sns.pairplot(df[numerical_columns])
     scatter_plots.fig.suptitle("Pair Plot of Numerical Variables", y=1.02)
     plt.show()
```



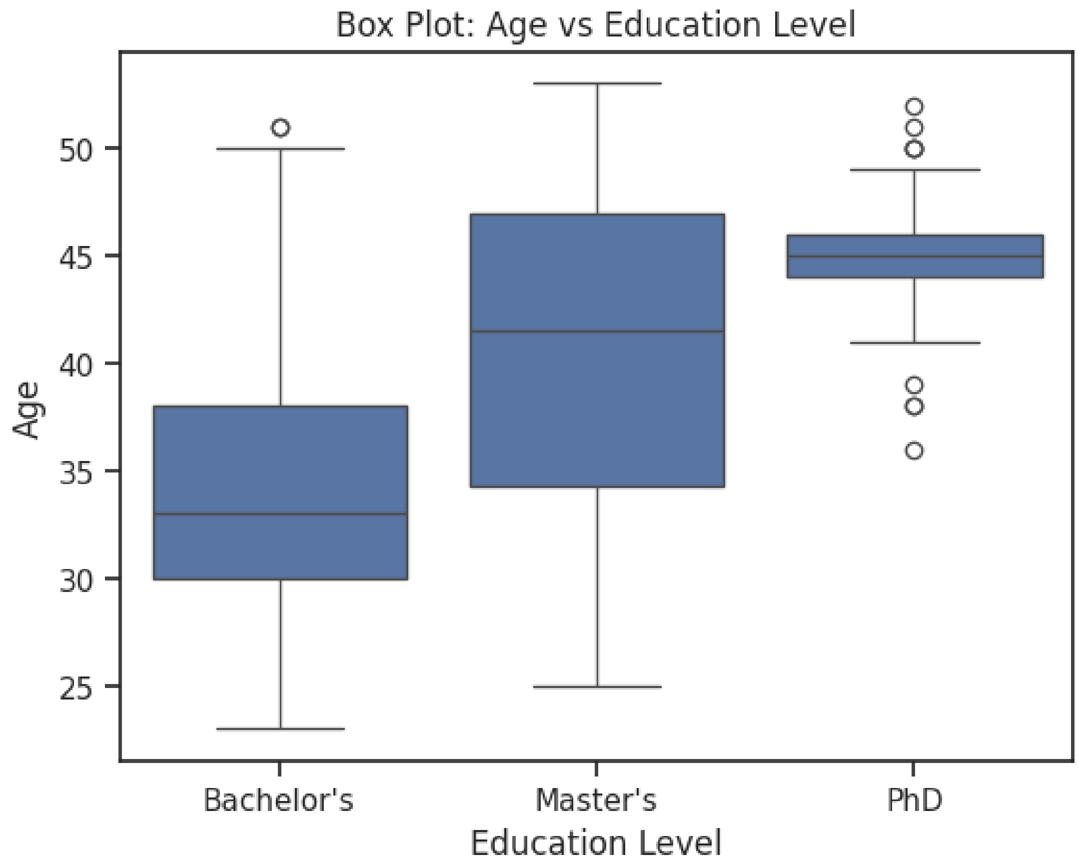

```
[ ]: #Bivariate Analysis:
     #Explore relationships between numerical and categorical variables using box
      ↪plots or violin plots.

     # Select numerical and categorical columns for bivariate analysis
     numerical_columns = ['Age', 'Years of Experience', 'Salary']
     categorical_columns = ['Gender', 'Education Level', 'Job Title']

     # Visualize relationships using box plots
     plt.figure(figsize=(15, 8))
     for cat_col in categorical_columns:
```

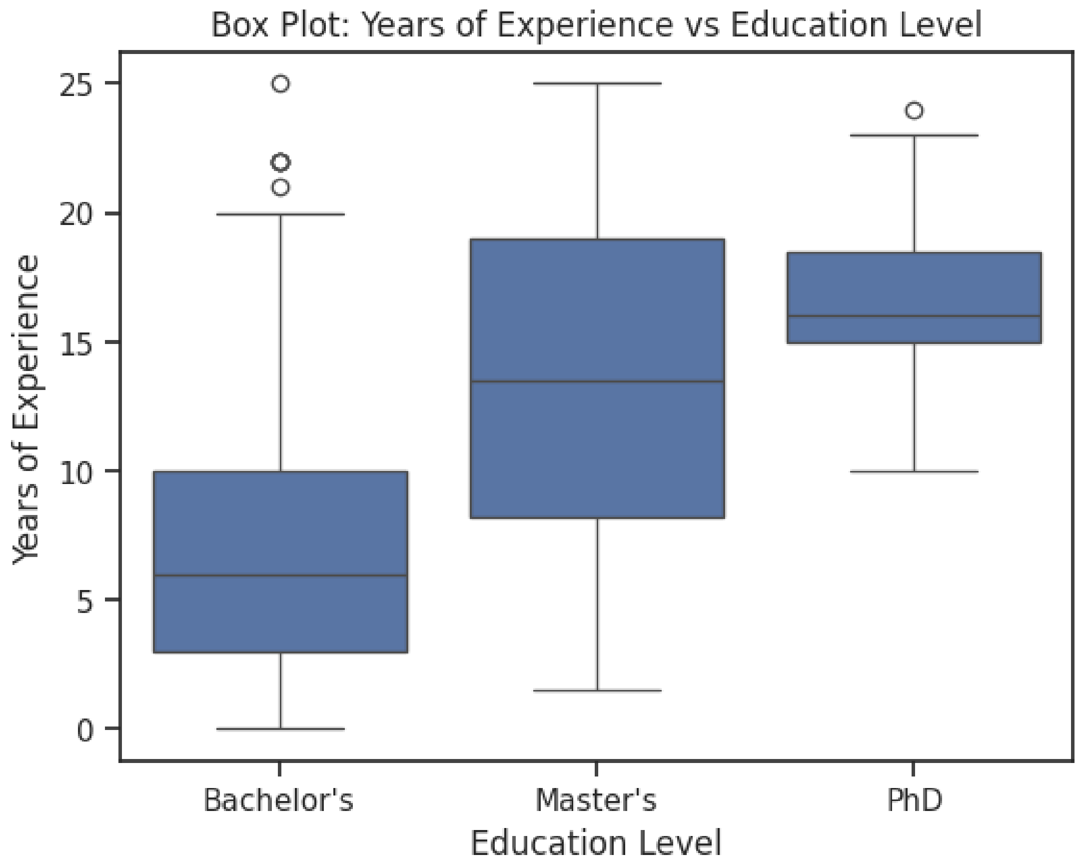
```
    sns.boxplot(x=cat_col, y='Age', data=df)
    plt.title(f'Box Plot: Age vs {cat_col}')
    plt.show()

    sns.boxplot(x=cat_col, y='Years of Experience', data=df)
    plt.title(f'Box Plot: Years of Experience vs {cat_col}')
    plt.show()

    sns.boxplot(x=cat_col, y='Salary', data=df)
    plt.title(f'Box Plot: Salary vs {cat_col}')
    plt.show()

# Visualize relationships using violin plots
plt.figure(figsize=(15, 8))
for cat_col in categorical_columns:
    sns.violinplot(x=cat_col, y='Age', data=df)
    plt.title(f'Violin Plot: Age vs {cat_col}')
    plt.show()

    sns.violinplot(x=cat_col, y='Years of Experience', data=df)
    plt.title(f'Violin Plot: Years of Experience vs {cat_col}')
    plt.show()

    sns.violinplot(x=cat_col, y='Salary', data=df)
    plt.title(f'Violin Plot: Salary vs {cat_col}')
    plt.show()
```

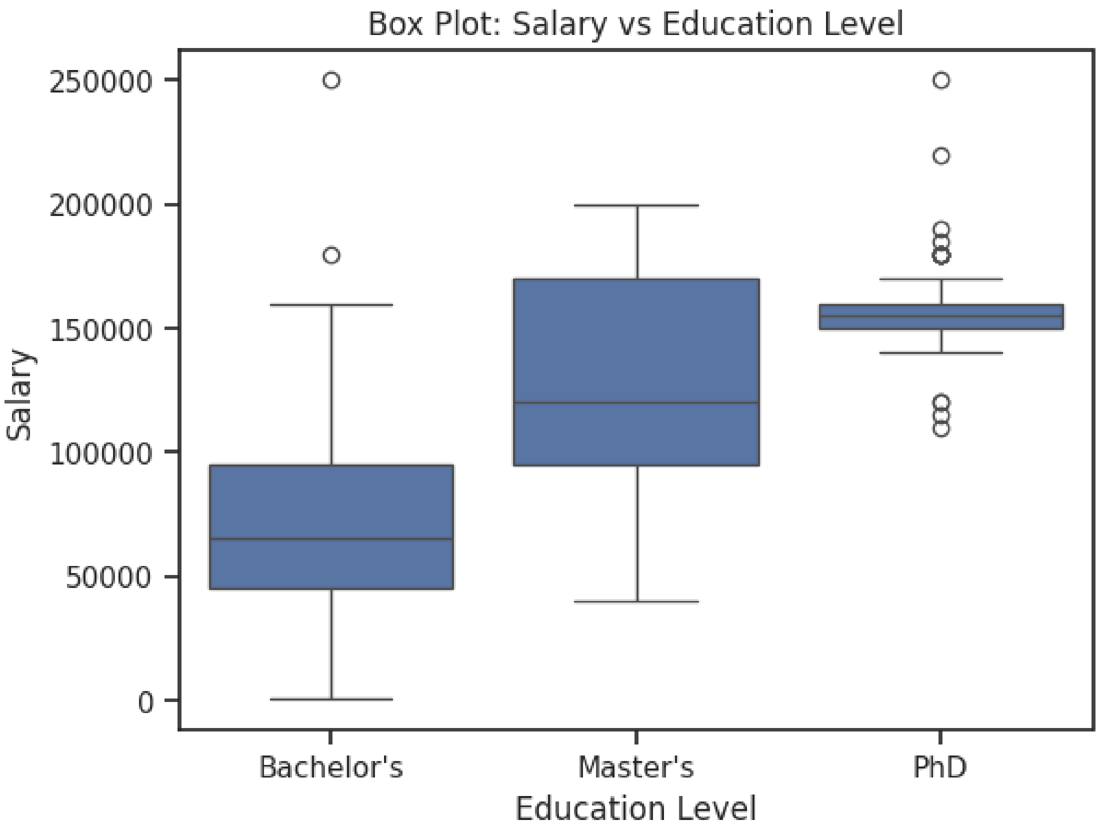


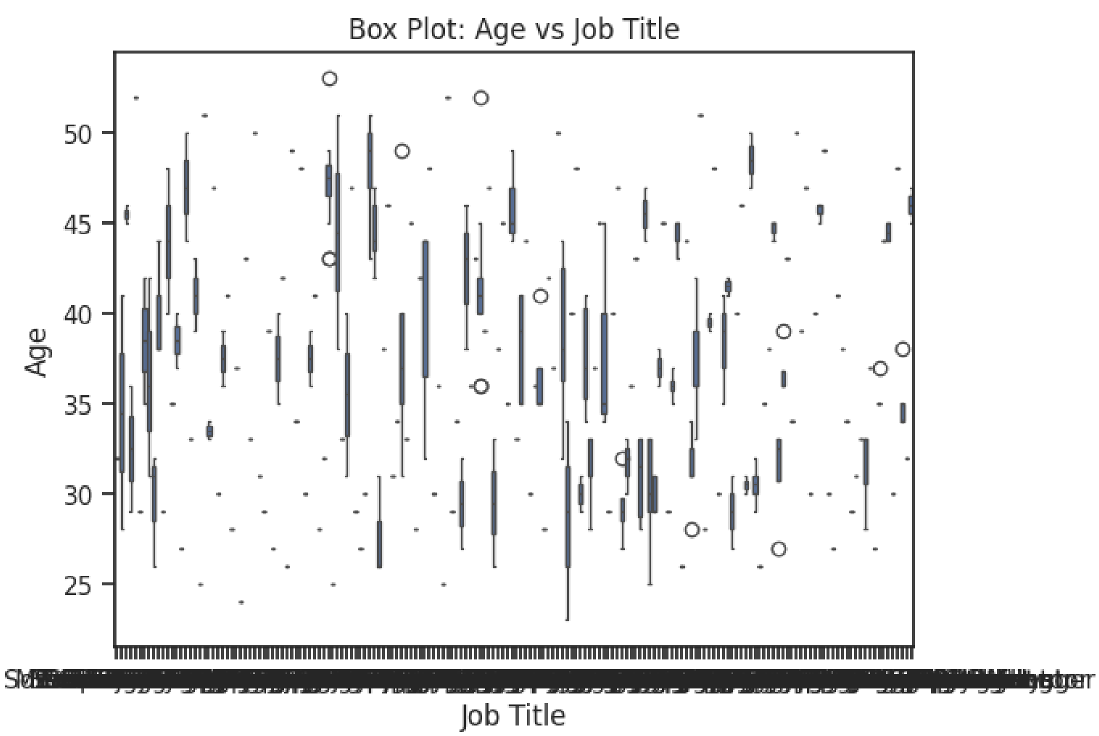

Box Plot: Years of Experience vs Gender
Years of Experience
25
20
15
10
5
0
Male
Female
Gender


Box Plot: Salary vs Gender
250000
200000
150000
100000
50000
0
Salary
Male
Female
Gender

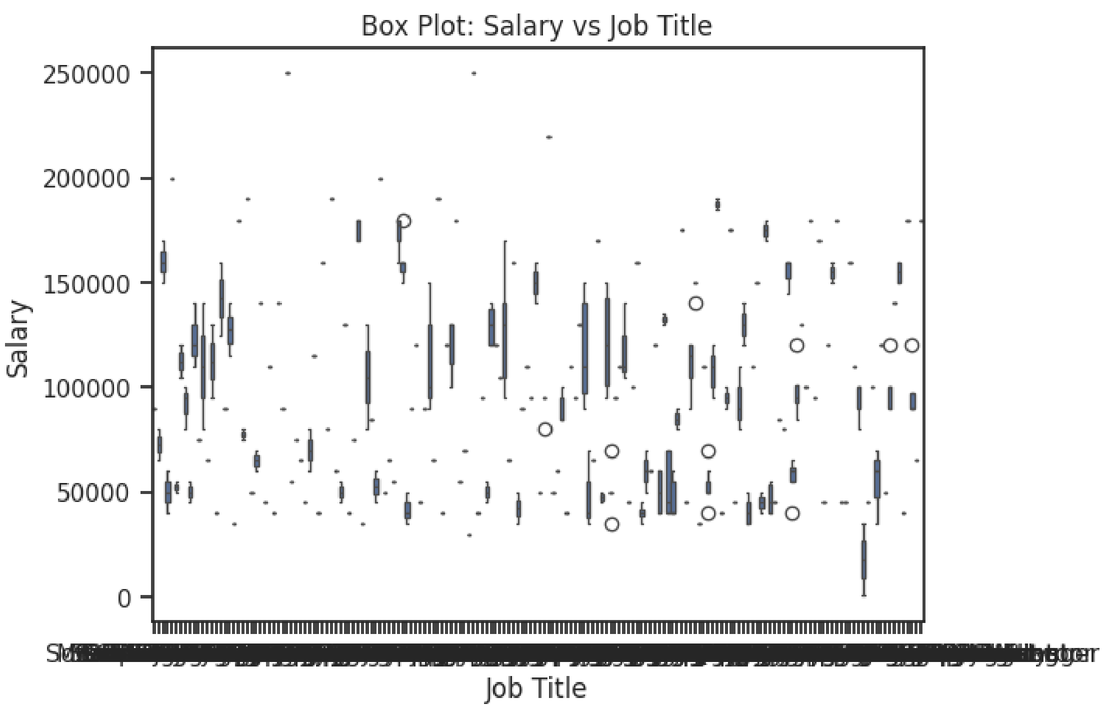

Box Plot: Age vs Education Level
50
45
40
35
30
25
Age
Bachelor's
Master's
PhD
Education Level


Box Plot: Years of Experience vs Education Level
Years of Experience
25
20
15
10
5
0
Bachelor's
Master's
PhD
Education Level

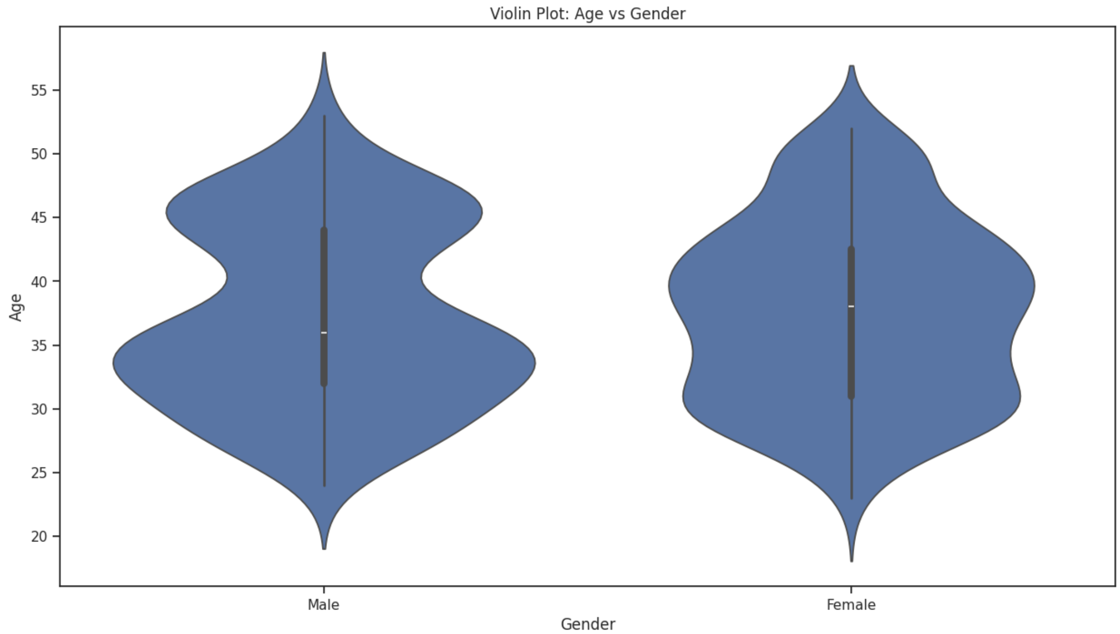

Box Plot: Salary vs Education Level
250000
200000
150000
100000
50000
0
Salary
Bachelor's
Master's
PhD
Education Level

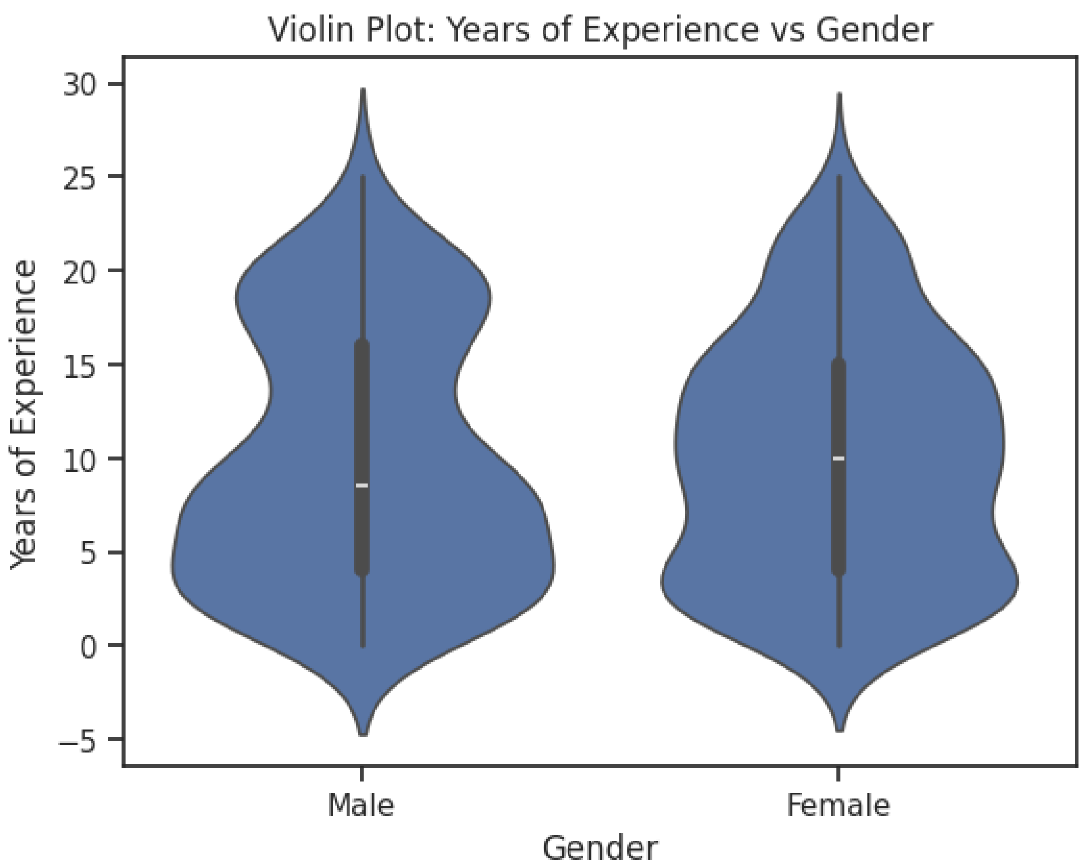

Box Plot: Age vs Job Title
50
45
40
35
30
25
Age
Job Title

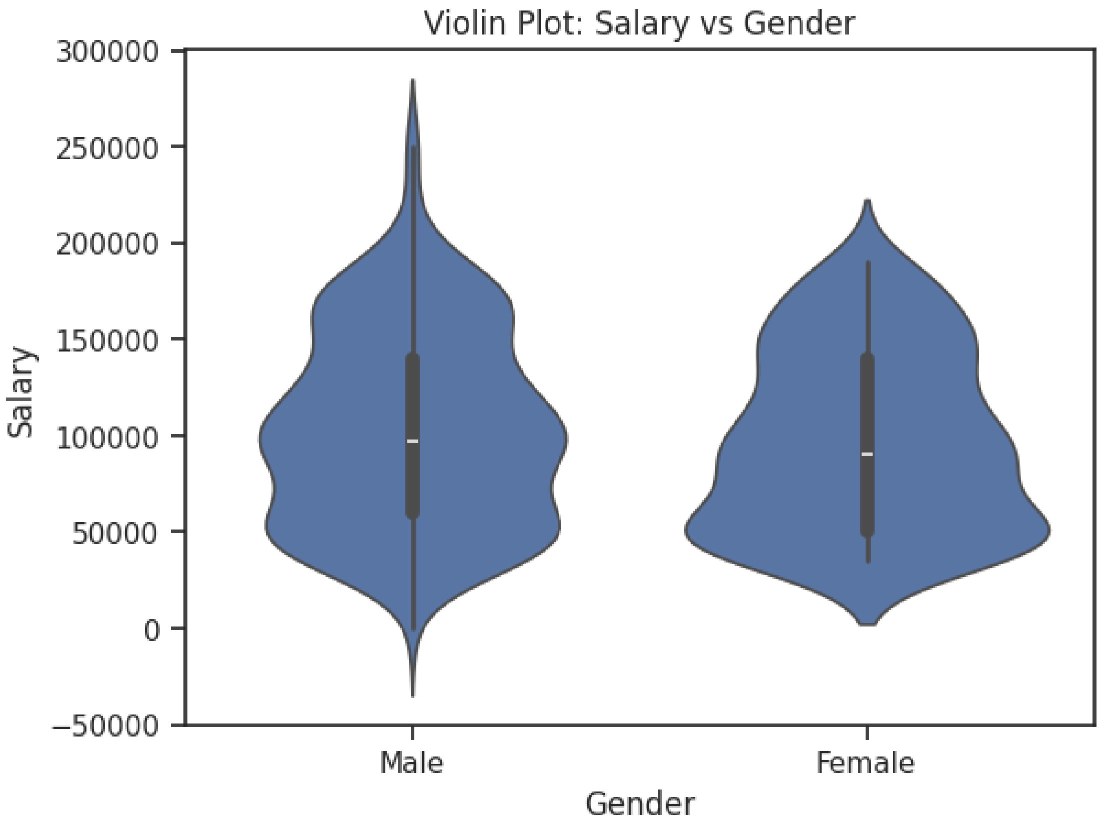
Box Plot: Years of Experience vs Job Title
Years of Experience
Job Title

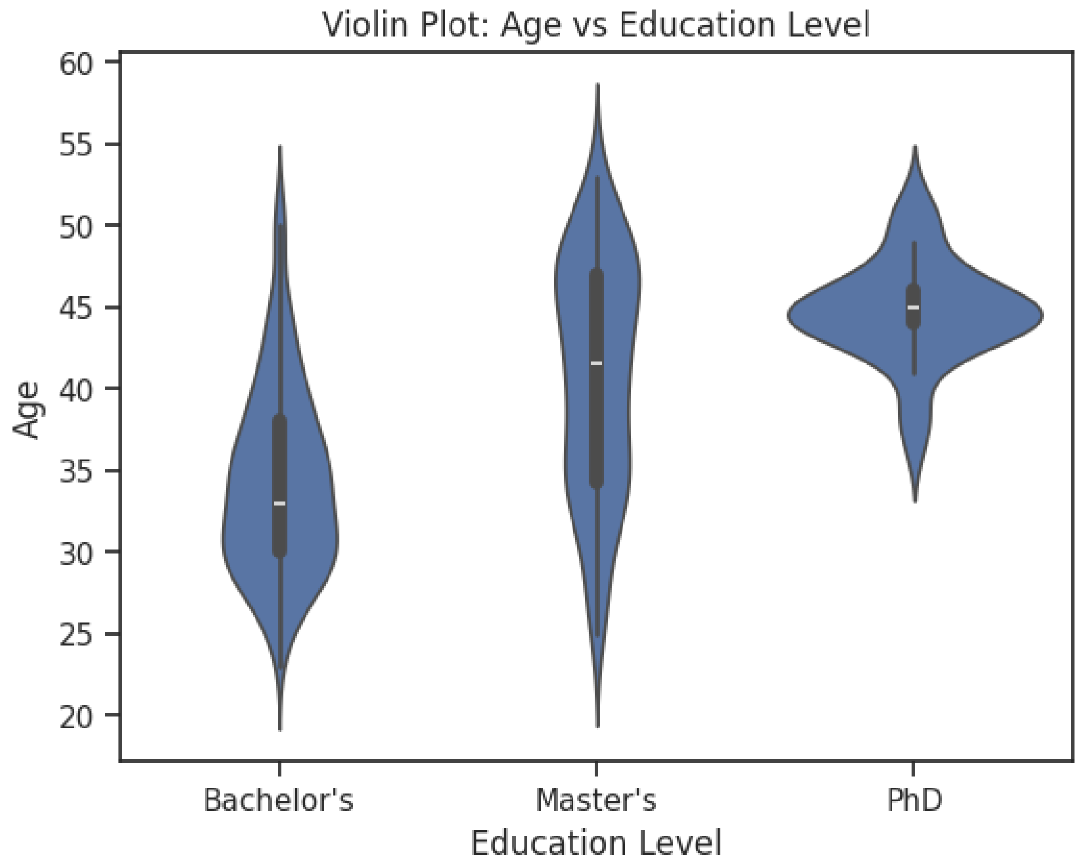


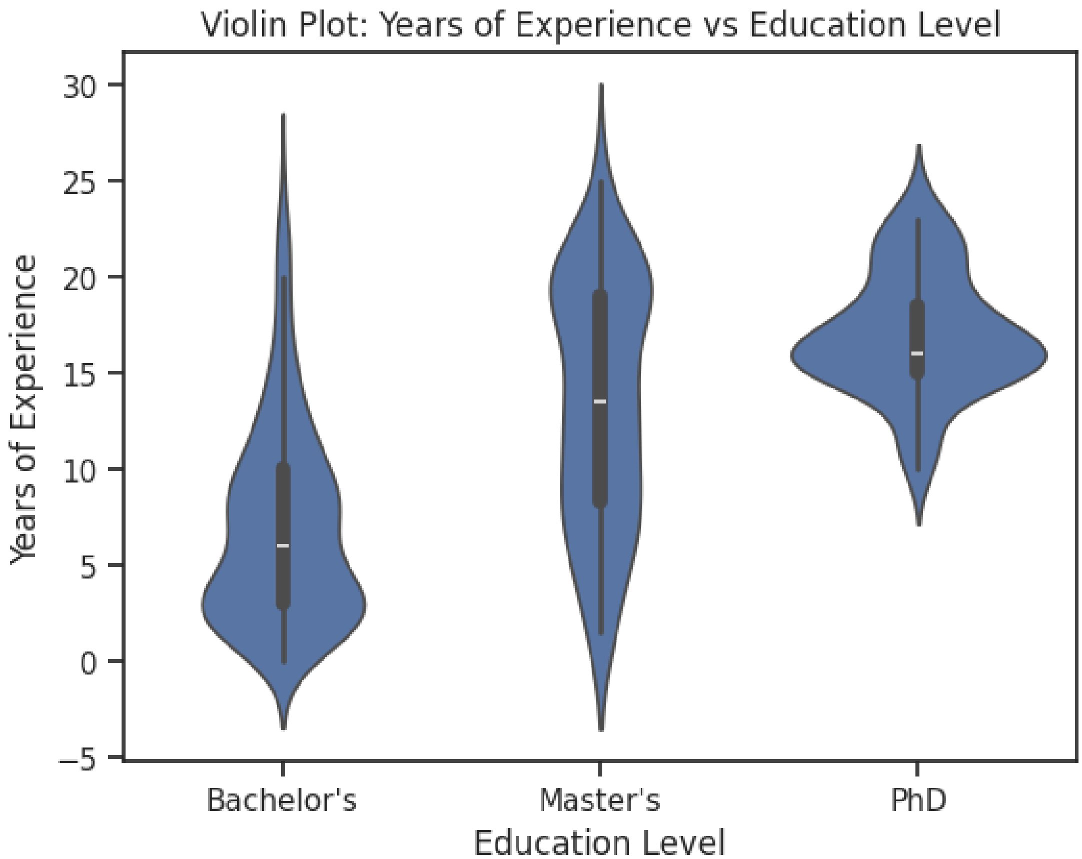
Box Plot: Salary vs Job Title
Salary
Job Title

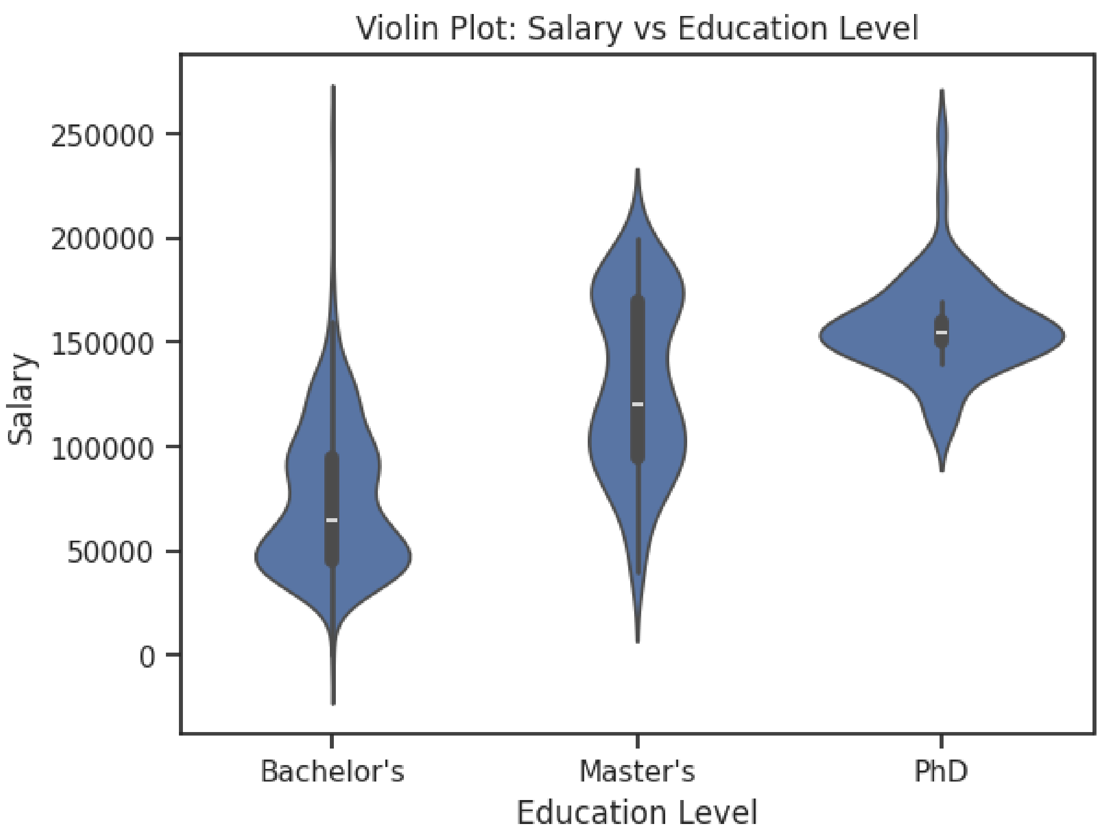

Violin Plot: Age vs Gender
Age
55
50
45
40
35
30
25
20
Male
Female
Gender

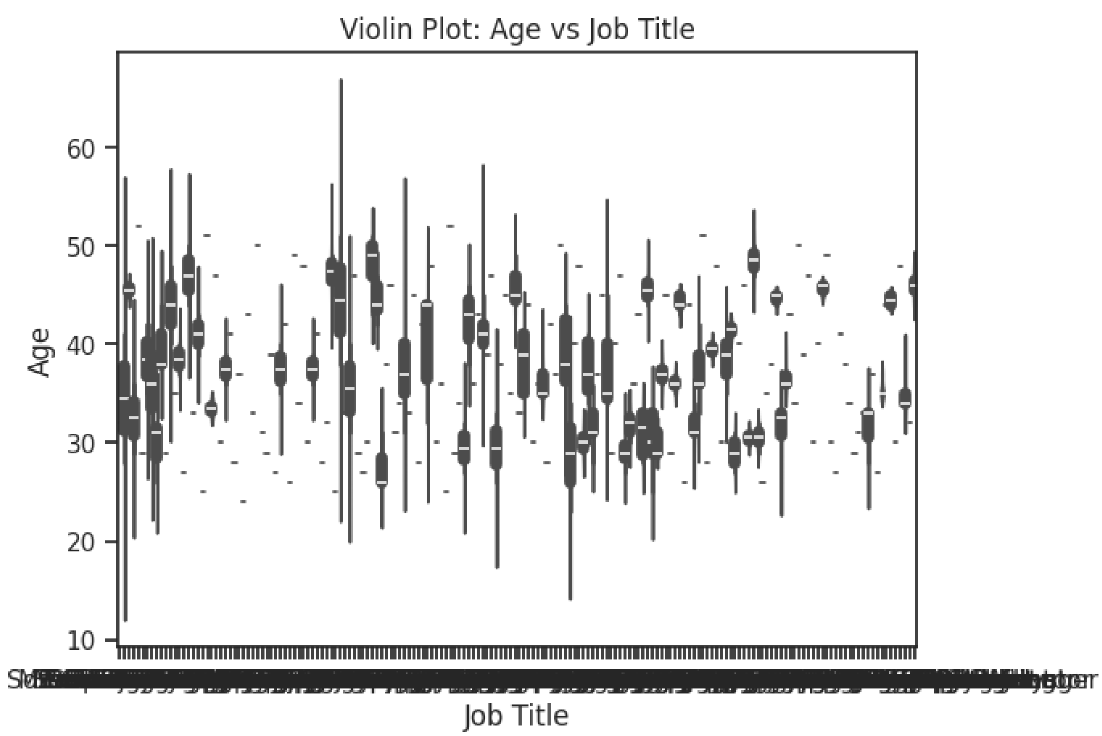

Violin Plot: Years of Experience vs Gender
Years of Experience
30
25
20
15
10
5
0
−5
Male
Female
Gender

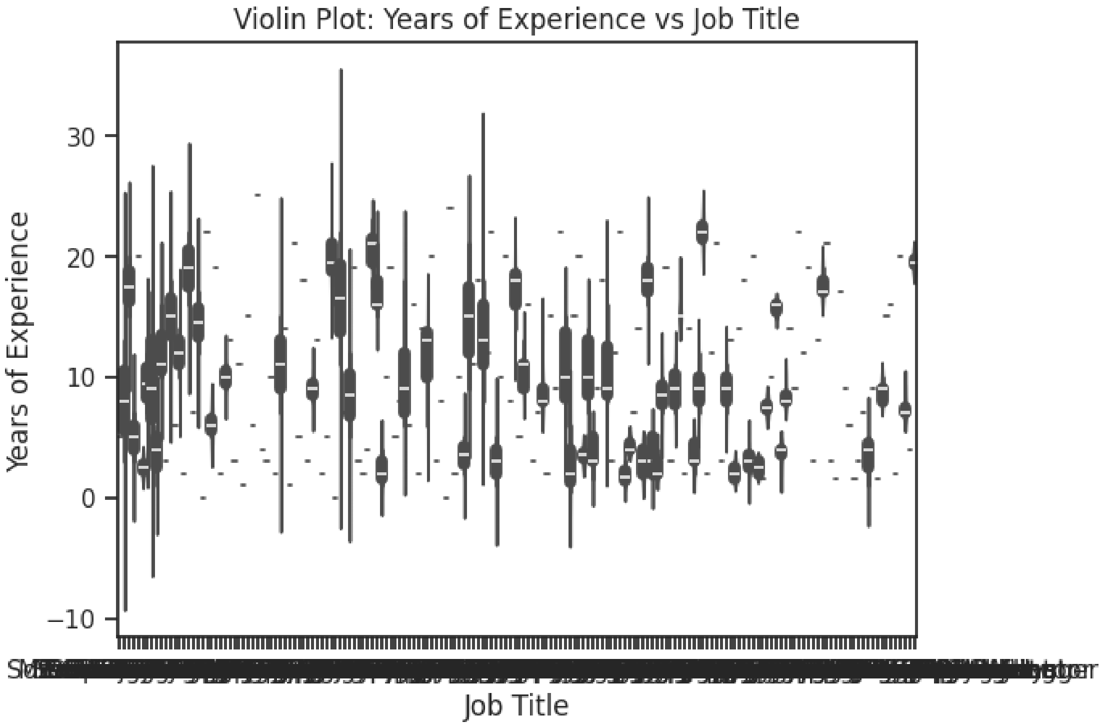

Violin Plot: Salary vs Gender
300000
250000
200000
150000
100000
50000
0
−50000
Salary
Male
Female
Gender

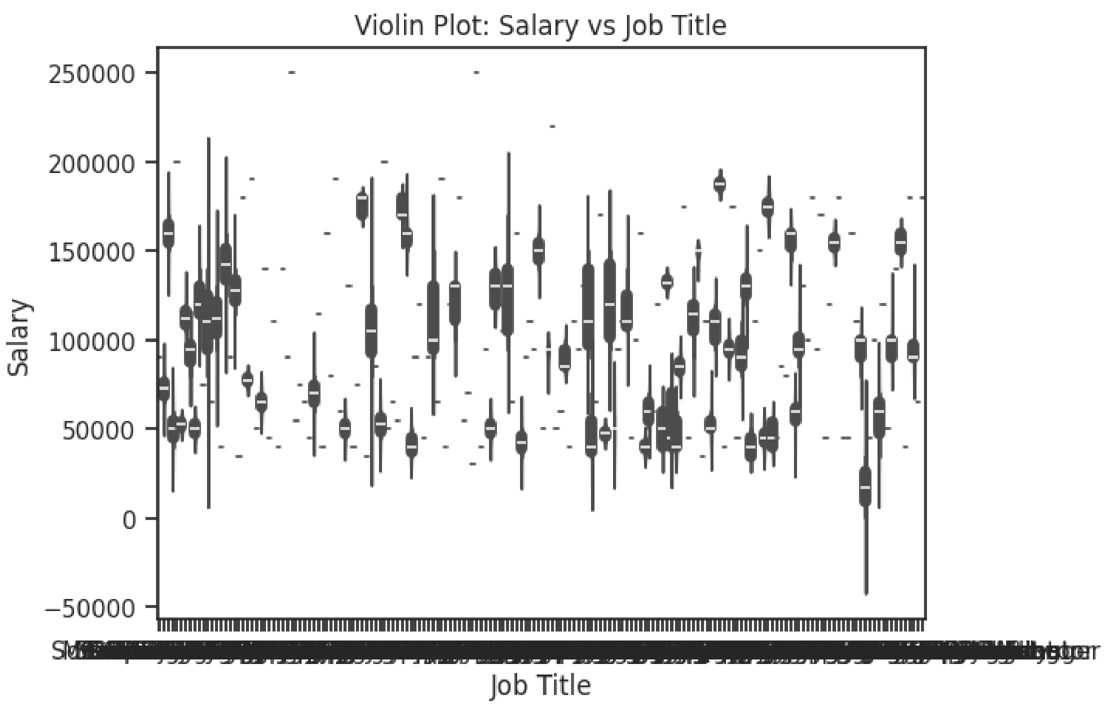

Violin Plot: Age vs Education Level
Age
60
55
50
45
40
35
30
25
20
Bachelor's
Master's
PhD
Education Level


Violin Plot: Years of Experience vs Education Level
Years of Experience
30
25
20
15
10
5
0
−5
Bachelor's
Master's
PhD
Education Level


Violin Plot: Salary vs Education Level
Salary
250000
200000
150000
100000
50000
0
Bachelor's
Master's
PhD
Education Level

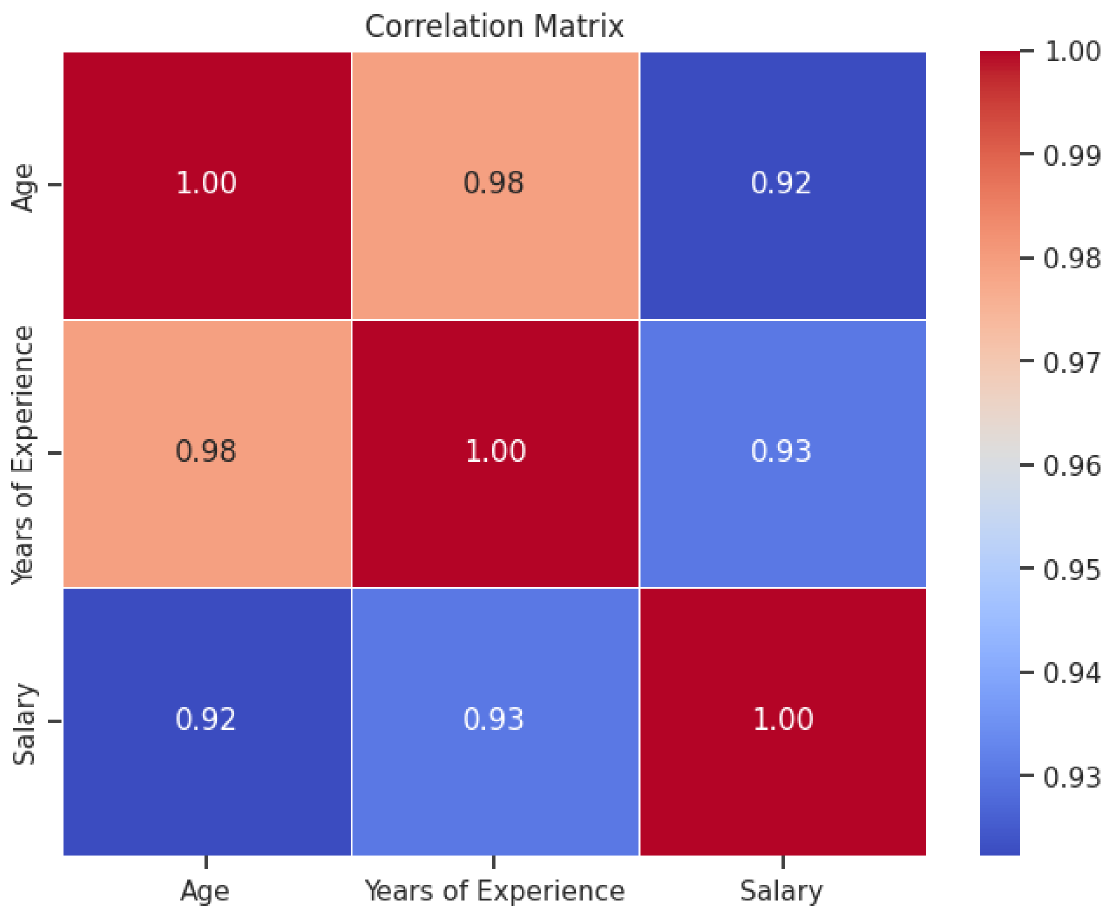

Violin Plot: Age vs Job Title
Age
60
50
40
30
20
10
Job Title


Violin Plot: Years of Experience vs Job Title
Years of Experience
Job Title
30
20
10
0
−10


Violin Plot: Salary vs Job Title
Salary
Job Title
250000
200000
150000
100000
50000
0
−50000

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

numerical_columns = ['Age', 'Years of Experience', 'Salary']

# Calculate correlation coefficients
correlation_matrix = df[numerical_columns].corr()

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

# Optional: Visualize the correlation matrix using a heatmap
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
```



[ ]:
```
#Non-parametric Methods:
#i. Apply non-parametric tests to validate or explore relationships:
#a. Spearman rank correlation for assessing monotonic relationships between␣
↪numerical variables.
# Select numerical columns for non-parametric analysis
numerical_columns = ['Age', 'Years of Experience', 'Salary']

# Apply Spearman rank correlation
for col1 in numerical_columns:
    for col2 in numerical_columns:
        if col1 != col2:
            correlation, p_value = spearmanr(df[col1], df[col2])
            print(f"Spearman Rank Correlation between {col1} and {col2}:␣
↪{correlation:.4f} (p-value: {p_value:.4f})")
```

```
Spearman Rank Correlation between Age and Years of Experience: nan (p-value:
nan)
Spearman Rank Correlation between Age and Salary: nan (p-value: nan)
Spearman Rank Correlation between Years of Experience and Age: nan (p-value:
```

```
nan)
Spearman Rank Correlation between Years of Experience and Salary: nan (p-value:
nan)
Spearman Rank Correlation between Salary and Age: nan (p-value: nan)
Spearman Rank Correlation between Salary and Years of Experience: nan (p-value:
nan)
```

[ ]:
```
# Non-parametric Methods: Mann-Whitney U test or Kruskal-Wallis H test for␣
↪comparing distributions
#across different groups.

from scipy.stats import mannwhitneyu, kruskal

# Assuming your DataFrame is named 'df'
# If not, replace 'df' with your actual DataFrame name

# Select numerical and categorical columns for non-parametric analysis
numerical_column = 'Salary'
categorical_column = 'Gender'

# Apply Mann-Whitney U test
group1 = df[df[categorical_column] == 'Male'][numerical_column]
group2 = df[df[categorical_column] == 'Female'][numerical_column]

statistic, p_value = mannwhitneyu(group1, group2)

print(f"Mann-Whitney U Test between {numerical_column} and {categorical_column}:
↪")
print(f"Statistic: {statistic:.4f}")
print(f"P-value: {p_value:.4f}")

# Select numerical and categorical columns for non-parametric analysis
numerical_column = 'Salary'
categorical_column = 'Job Title'

# Apply Kruskal-Wallis H test
groups = [df[df[categorical_column] == category][numerical_column] for category␣
↪in df[categorical_column].unique()]

statistic, p_value = kruskal(*groups)

print(f"Kruskal-Wallis H Test between {numerical_column} and␣
↪{categorical_column}:")
print(f"Statistic: {statistic:.4f}")
print(f"P-value: {p_value:.4f}")
```

```
Mann-Whitney U Test between Salary and Gender:
```

```
Statistic: 18617.0000
P-value: 0.2277
Kruskal-Wallis H Test between Salary and Job Title:
Statistic: nan
P-value: nan
```

[ ]:
```
#Non-parametric Methods: Wilcoxon signed-rank test for paired samples.

# Select two numerical columns for the paired test
numerical_column1 = 'Age'
numerical_column2 = 'Years of Experience'

# Apply Wilcoxon signed-rank test
statistic, p_value = wilcoxon(df[numerical_column1], df[numerical_column2])

print(f"Wilcoxon Signed-Rank Test between {numerical_column1} and␣
↪{numerical_column2}:")
print(f"Statistic: {statistic:.4f}")
print(f"P-value: {p_value:.4f}")
```

```
Wilcoxon Signed-Rank Test between Age and Years of Experience:
Statistic: nan
P-value: nan
```

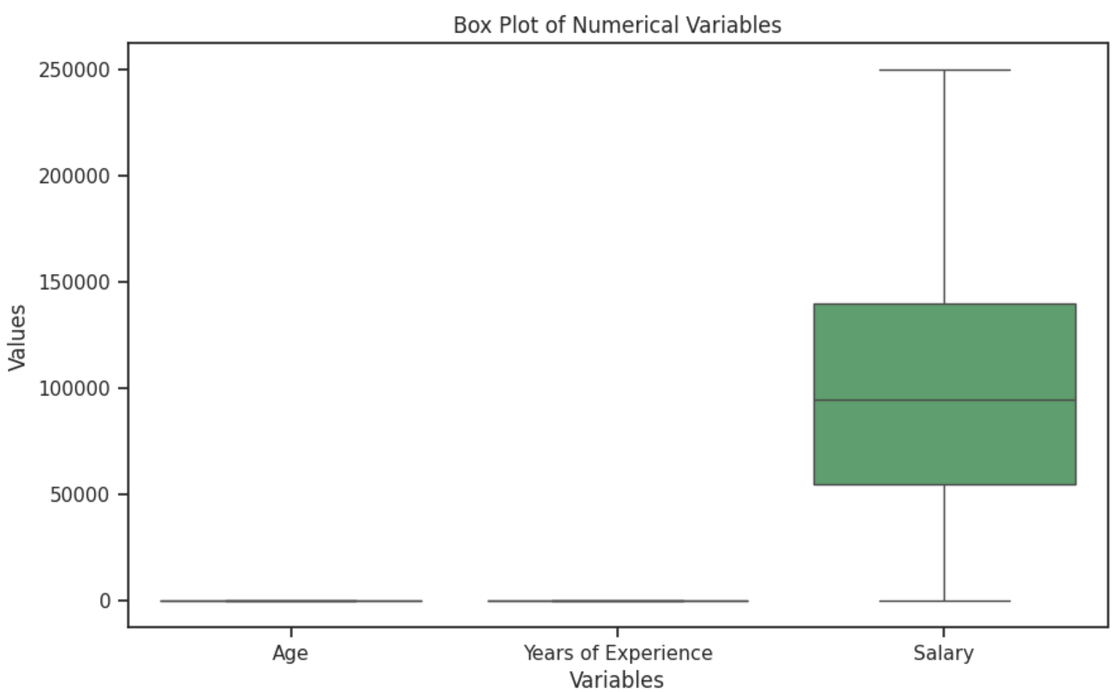
[ ]:
```
#Non-parametric Methods: Friedman test for comparing multiple paired samples.
# Select multiple numerical columns for the Friedman test
numerical_columns = ['Age', 'Years of Experience', 'Salary']

# Apply Friedman test
statistic, p_value = friedmanchisquare(df[numerical_columns[0]],␣
↪df[numerical_columns[1]], df[numerical_columns[2]])

print("Friedman Test for multiple paired samples:")
print(f"Statistic: {statistic:.4f}")
print(f"P-value: {p_value:.4f}")
```

```
Friedman Test for multiple paired samples:
Statistic: nan
P-value: nan
```

[ ]:
```
# Visualize the results using appropriate plots (e.g., box plots, bar plots,␣
↪etc.).
#Document your code and analysis process thoroughly, including any assumptions␣
↪made and decisions taken during
#data exploration.

# Select multiple numerical columns for the Friedman test
```

```
numerical_columns = ['Age', 'Years of Experience', 'Salary']

# Apply Friedman test
statistic, p_value = friedmanchisquare(df[numerical_columns[0]],␣
 ↪df[numerical_columns[1]], df[numerical_columns[2]])

print("Friedman Test for multiple paired samples:")
print(f"Statistic: {statistic:.4f}")
print(f"P-value: {p_value:.4f}")

# Visualize the results using box plots
plt.figure(figsize=(10, 6))
sns.boxplot(data=df[numerical_columns])
plt.title("Box Plot of Numerical Variables")
plt.xlabel("Variables")
plt.ylabel("Values")
plt.show()
```

```
Friedman Test for multiple paired samples:
Statistic: nan
P-value: nan
```



Conclusion

In conclusion, the provided dataset appears to be well-suited for non-parametric models. Non-parametric modeling techniques such as decision trees, random forests, support vector machines, or kernel-based methods can be applied to this dataset to capture its underlying patterns effectively without relying on strict assumptions about the data distribution or relationships. These models can adapt to the dataset's characteristics and provide robust and flexible modeling approaches.