# 2348441_Lab3

February 29, 2024

**Lab Exercise 3 -Regression Analysis**

Created by : Nileem Kaveramma C C | 2348441 Created DATE:28-02-2024 Edited Date: 29-02-2024

**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
- train_test_split
- LinearRegression
- mean_squared_error, r2_score

**AIM** : Aim: To explore and apply Regression Analysis techniques in Machine Learning for predictive modeling, with the goal of understanding and quantifying relationships between variables, predicting future outcomes, and optimizing decision-making processes.

**PROCEDURE:**

1.Model Selection: Choose an appropriate regression model based on data characteristics.

2.Data Preparation: Clean and preprocess data, handling missing values and outliers.

3.Feature Selection: Identify relevant independent variables for analysis.

4.Training the Model: Train the regression model using the prepared dataset.

5.Model Evaluation: Assess performance using metrics like MSE, RMSE, and R-squared.

6.Interpretation of Coefficients: Analyze coefficients to understand variable relationships.

7.Assess Goodness-of-Fit: Examine R-squared for overall model fit.

8.Validate the Model: Verify performance on a separate test dataset.

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

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.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
```

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
```

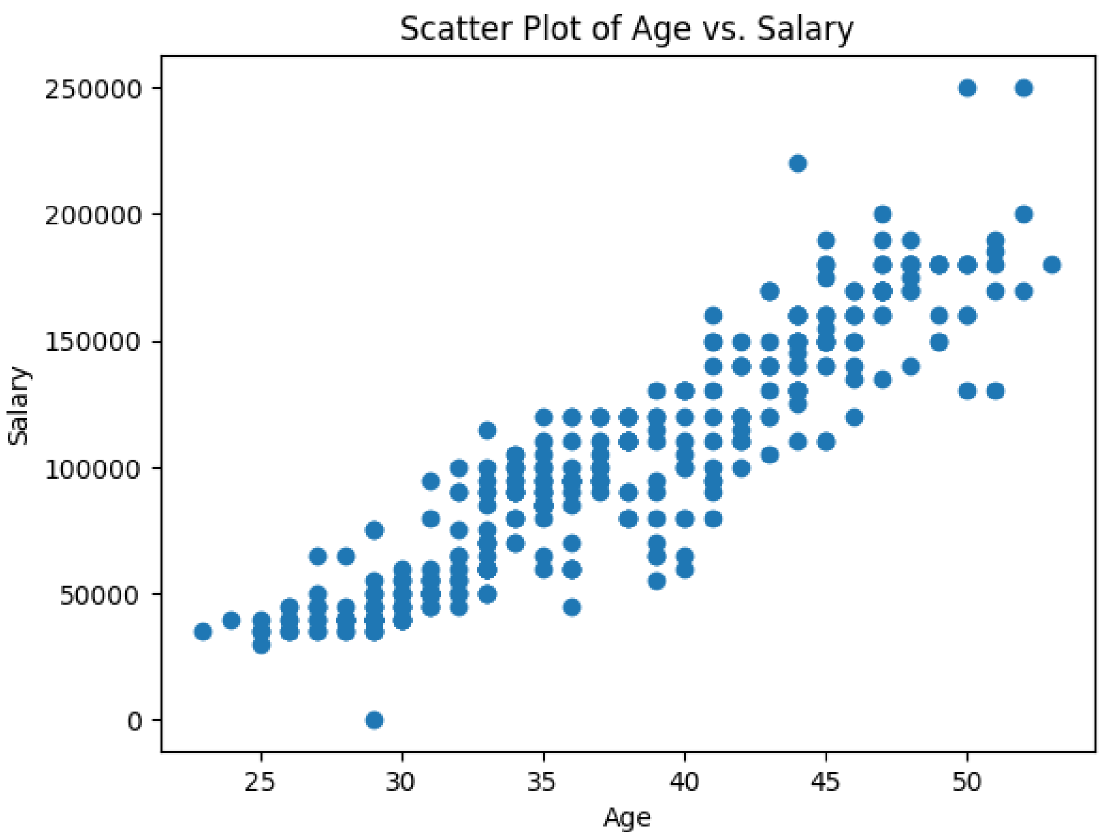
The df.describe() method in Pandas is used to generate descriptive statistics that summarize the central tendency, dispersion, and shape of a dataset's distribution

[ ]:
```
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
```

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



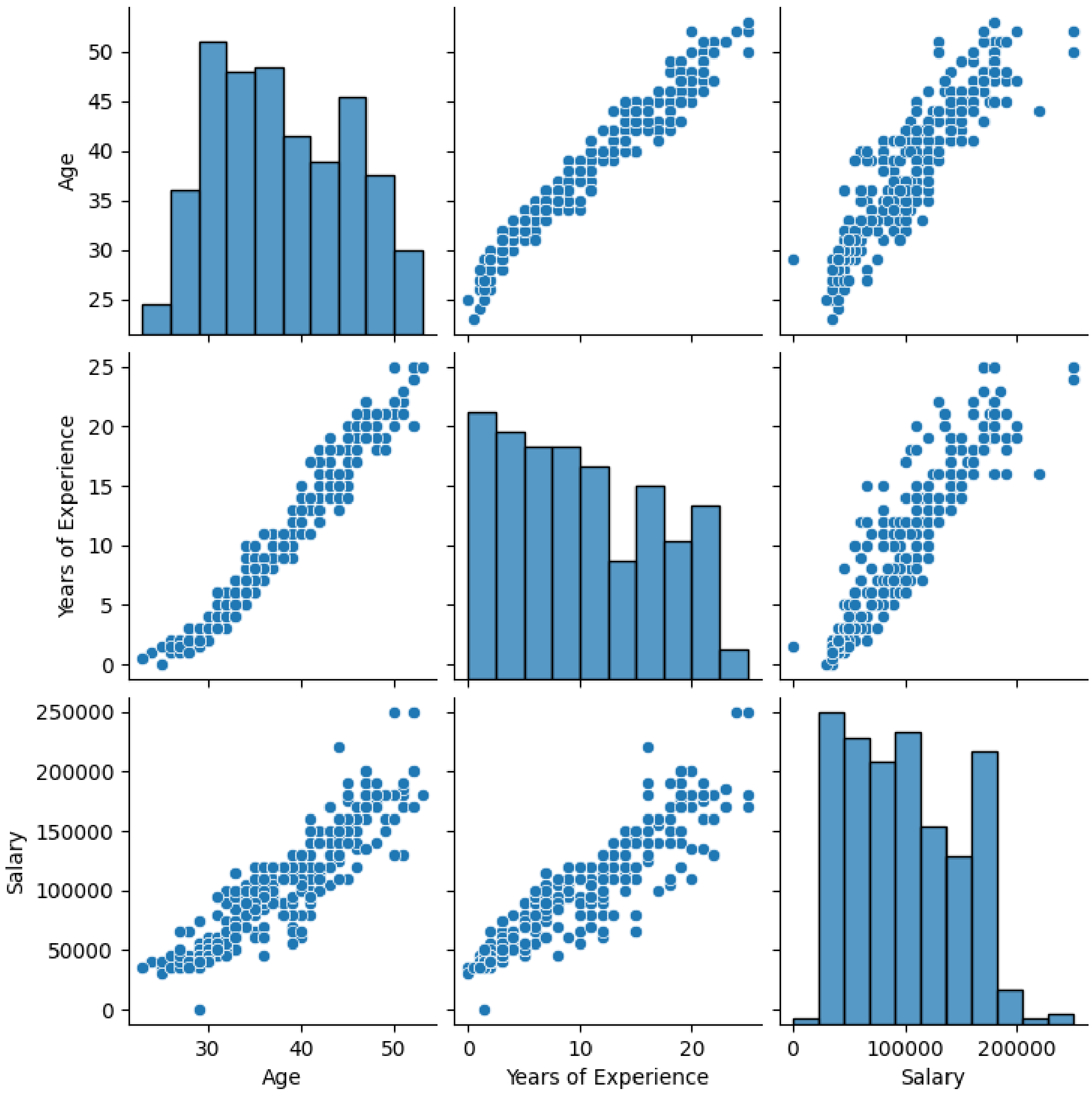

[ ]:
```python
## Correlation
df.corr()
```

```
<ipython-input-15-2d23776439fc>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  df.corr()
```

[ ]:
```
                          Age  Years of Experience    Salary
Age                  1.000000             0.979128  0.922335
Years of Experience  0.979128             1.000000  0.930338
Salary               0.922335             0.930338  1.000000
```

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

[ ]: <seaborn.axisgrid.PairGrid at 0x7f8b1432b220>




- Independent feature is 1-dimentional array, it should be a 2-dimentional array

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

[ ]:
```
X
```

[ ]:
```
0       90000.0
1       65000.0
2      150000.0
3       60000.0
4      200000.0
         …
370     85000.0
371    170000.0
372     40000.0
373     90000.0
374    150000.0
Name: Salary, Length: 375, dtype: float64
```

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

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

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

[ ]:
```
       Salary
0     90000.0
1     65000.0
2    150000.0
3     60000.0
4    200000.0
..        …
370   85000.0
371  170000.0
372   40000.0
373   90000.0
374  150000.0

[375 rows x 1 columns]
```

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

[ ]:
```
(375, 1)
```

[ ]:
```
## Independent and dependent features
X=df[['Salary']] ### independent features should be data frame or 2␣
 ↪dimesnionalarray
X
```

```
y=df['Years of Experience'] ## this variiable can be in series or 1d array
y
```

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

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

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

[ ]:
```
from sklearn.model_selection import train_test_split
```

[ ]:
```
## Standardization
from sklearn.preprocessing import StandardScaler
```

[ ]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
↪25,random_state=42)
```

[ ]:
```
scaler=StandardScaler()
X_train=scaler.fit_transform(X_train)
```

[ ]:
```
print(X_train)
```

```
[[ 0.81724175]
 [ 0.40474079]
 [ 0.19849031]
 [ 0.30161555]
 [ 0.40474079]
 [ 0.81724175]
 [-1.14213782]
 [-0.93588733]
 [-1.03901257]
 [-1.3483883 ]
 [-1.24526306]
 [-1.03901257]
 [ 1.64224367]
```

```
[-0.31713589]
[-0.11088541]
[-0.62651161]
[-0.21401065]
[ 0.81724175]
[-0.93588733]
[ 1.64224367]
[-1.24526306]
[-0.93588733]
[-1.03901257]
[-1.3483883 ]
[-0.83276209]
[-0.42026113]
[-0.83276209]
[-1.14213782]
[-0.42026113]
[-0.93588733]
[-0.83276209]
[-1.3483883 ]
[-1.24526306]
[-0.62651161]
[-0.83276209]
[ 0.40474079]
[-0.31713589]
[ 0.81724175]
[ 0.09536507]
[-0.11088541]
[-0.11088541]
[ 0.40474079]
[-0.83276209]
[ 1.64224367]
[-0.52338637]
[-1.24526306]
[-1.03901257]
[-0.42026113]
[-1.24526306]
[ 0.40474079]
[ 0.40474079]
[-1.3483883 ]
[-0.42026113]
[ 0.61099127]
[ 0.81724175]
[-0.83276209]
[-1.24526306]
[ 0.09536507]
[ 1.02349223]
[-1.24526306]
[ 1.22974271]
```

```
[ 0.71411651]
[ 1.43599319]
[-1.3483883 ]
[ 1.22974271]
[-0.21401065]
[-0.83276209]
[ 0.30161555]
[-1.24526306]
[-1.03901257]
[-1.14213782]
[ 1.64224367]
[ 1.64224367]
[-0.00776017]
[-0.62651161]
[-1.24526306]
[ 1.02349223]
[ 1.02349223]
[-1.14213782]
[-0.31713589]
[ 0.92036699]
[ 1.02349223]
[-1.24526306]
[-0.11088541]
[-1.24526306]
[ 1.22974271]
[-0.83276209]
[ 1.22974271]
[ 0.09536507]
[-1.24526306]
[ 0.81724175]
[ 1.64224367]
[ 0.61099127]
[-0.93588733]
[ 0.61099127]
[ 1.43599319]
[-0.11088541]
[-0.31713589]
[ 1.43599319]
[-0.31713589]
[-1.24526306]
[-0.21401065]
[ 1.02349223]
[ 0.40474079]
[ 0.09536507]
[ 0.81724175]
[-0.11088541]
[ 0.19849031]
[ 1.12661747]
```

```
[ 1.02349223]
[ 1.02349223]
[-0.21401065]
[-0.11088541]
[ 0.19849031]
[-1.24526306]
[ 0.19849031]
[-0.00776017]
[ 1.02349223]
[-1.03901257]
[-0.72963685]
[ 0.19849031]
[ 0.40474079]
[ 1.64224367]
[-0.21401065]
[ 3.08599703]
[-0.83276209]
[-0.83276209]
[-0.00776017]
[ 1.84849415]
[ 1.22974271]
[-0.31713589]
[-0.62651161]
[ 0.61099127]
[ 0.61099127]
[-0.21401065]
[ 0.19849031]
[-0.11088541]
[-0.11088541]
[-0.72963685]
[-1.24526306]
[-0.72963685]
[-1.14213782]
[ 0.81724175]
[ 1.64224367]
[-1.03901257]
[ 0.61099127]
[-0.11088541]
[-1.14213782]
[ 0.40474079]
[-1.14213782]
[-1.14213782]
[ 1.43599319]
[-0.31713589]
[ 1.22974271]
[-0.11088541]
[-0.72963685]
[-1.3483883 ]
```

```
[-0.42026113]
[-0.21401065]
[-0.42026113]
[ 1.84849415]
[-0.93588733]
[ 0.61099127]
[-1.03901257]
[-0.31713589]
[-0.00776017]
[ 2.05474463]
[-0.00776017]
[-0.11088541]
[-1.14213782]
[ 1.22974271]
[ 1.43599319]
[-0.31713589]
[-1.24526306]
[-0.21401065]
[-0.00776017]
[-1.03901257]
[ 1.64224367]
[-0.52338637]
[ 0.81724175]
[ 1.64224367]
[ 1.22974271]
[-1.24526306]
[ 0.19849031]
[-1.14213782]
[        nan]
[-1.14213782]
[-0.11088541]
[-0.83276209]
[-0.42026113]
[-0.93588733]
[-1.03901257]
[-1.14213782]
[-1.3483883 ]
[-1.24526306]
[-1.03901257]
[-0.42026113]
[-1.03901257]
[-0.93588733]
[-0.11088541]
[ 1.64224367]
[ 0.61099127]
[-0.00776017]
[ 0.40474079]
[-1.24526306]
```

```
[-0.83276209]
[-0.21401065]
[ 0.19849031]
[ 1.43599319]
[ 1.43599319]
[ 1.02349223]
[ 0.19849031]
[-1.14213782]
[ 1.53911843]
[-1.14213782]
[ 1.64224367]
[-0.11088541]
[ 1.43599319]
[-2.06304621]
[ 2.46724559]
[ 2.05474463]
[-0.72963685]
[-1.3483883 ]
[-0.62651161]
[ 1.22974271]
[ 1.43599319]
[-1.03901257]
[-1.24526306]
[-0.83276209]
[ 0.19849031]
[-1.24526306]
[ 0.61099127]
[ 0.61099127]
[ 0.71411651]
[ 1.64224367]
[ 0.19849031]
[ 0.19849031]
[ 1.22974271]
[-1.3483883 ]
[ 1.64224367]
[-0.21401065]
[ 0.40474079]
[ 1.02349223]
[-1.03901257]
[ 0.61099127]
[-1.14213782]
[-1.3483883 ]
[-1.24526306]
[-0.00776017]
[ 0.09536507]
[-0.93588733]
[ 0.40474079]
[ 1.22974271]
```

```
[-1.14213782]
[ 1.02349223]
[ 1.84849415]
[-0.83276209]
[-0.21401065]
[ 1.22974271]
[ 1.43599319]
[-0.21401065]
[ 1.02349223]
[-0.11088541]
[ 0.40474079]
[ 1.53911843]
[ 1.22974271]
[ 0.40474079]
[ 1.43599319]
[ 0.19849031]
[-0.42026113]
[-1.03901257]
[ 0.61099127]
[ 1.64224367]
[ 1.02349223]
[-0.42026113]
[ 1.64224367]
[-0.62651161]
[-1.03901257]
[ 1.02349223]
[-1.3483883 ]
[ 1.02349223]]
```

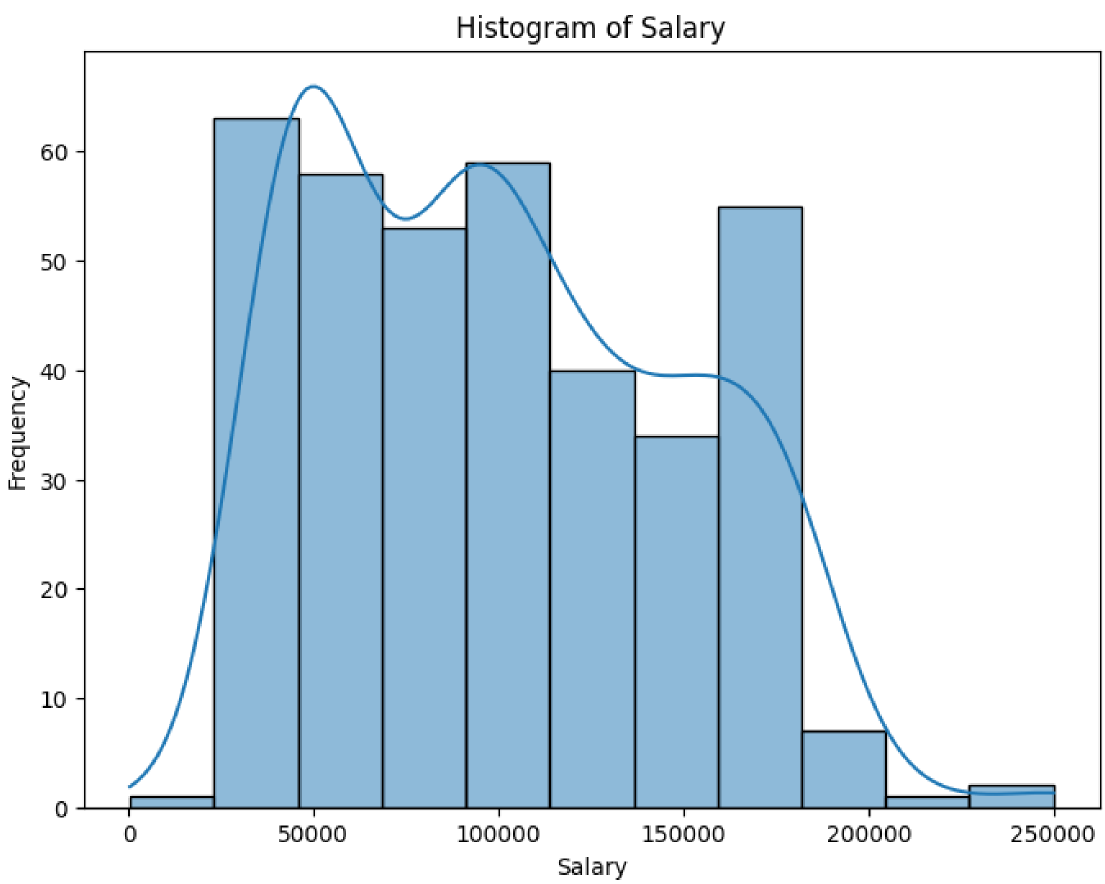
[ ]:
```
#Univariate Analysis:
#For numerical variables: a. Calculate basic descriptive statistics (mean,␣
↪median, mode, standard deviation,
#min, max, quartiles, etc.).

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


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

```
print(f"Max: {max_value}")
```

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

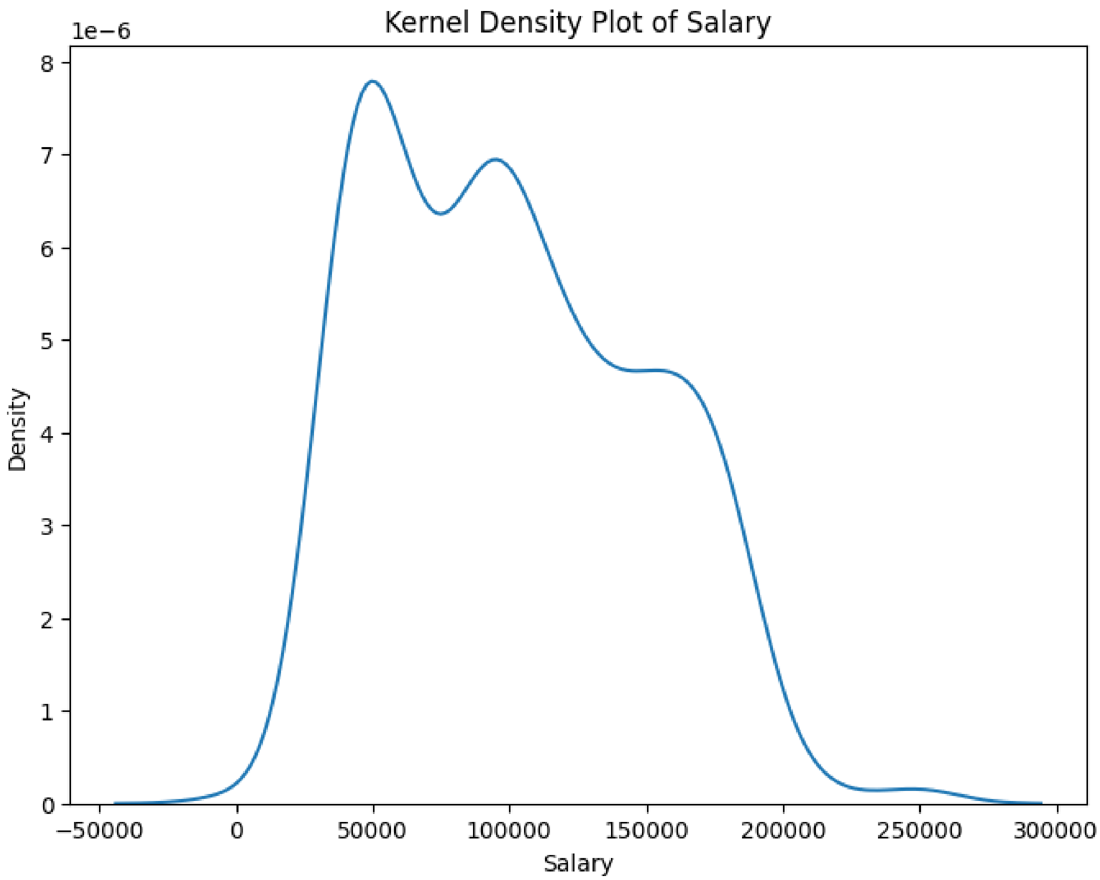
[ ]:

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

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

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



[ ]:

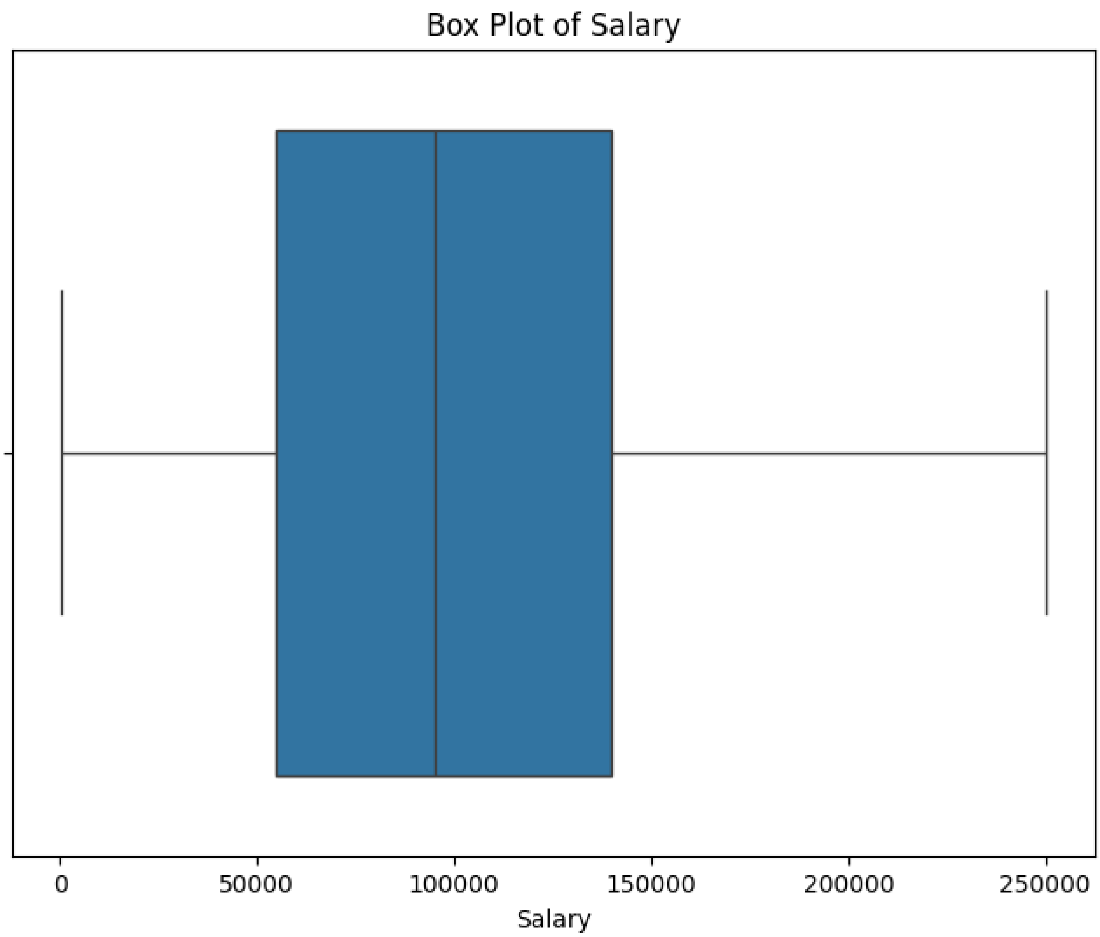
```python
# Plot a simple kernel density plot
plt.figure(figsize=(8, 6))
sns.kdeplot(df['Salary'])
plt.title('Kernel Density Plot of Salary')
plt.xlabel('Salary')
plt.ylabel('Density')

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




[ ]:

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

# Show the plot
```

```
plt.show()
```

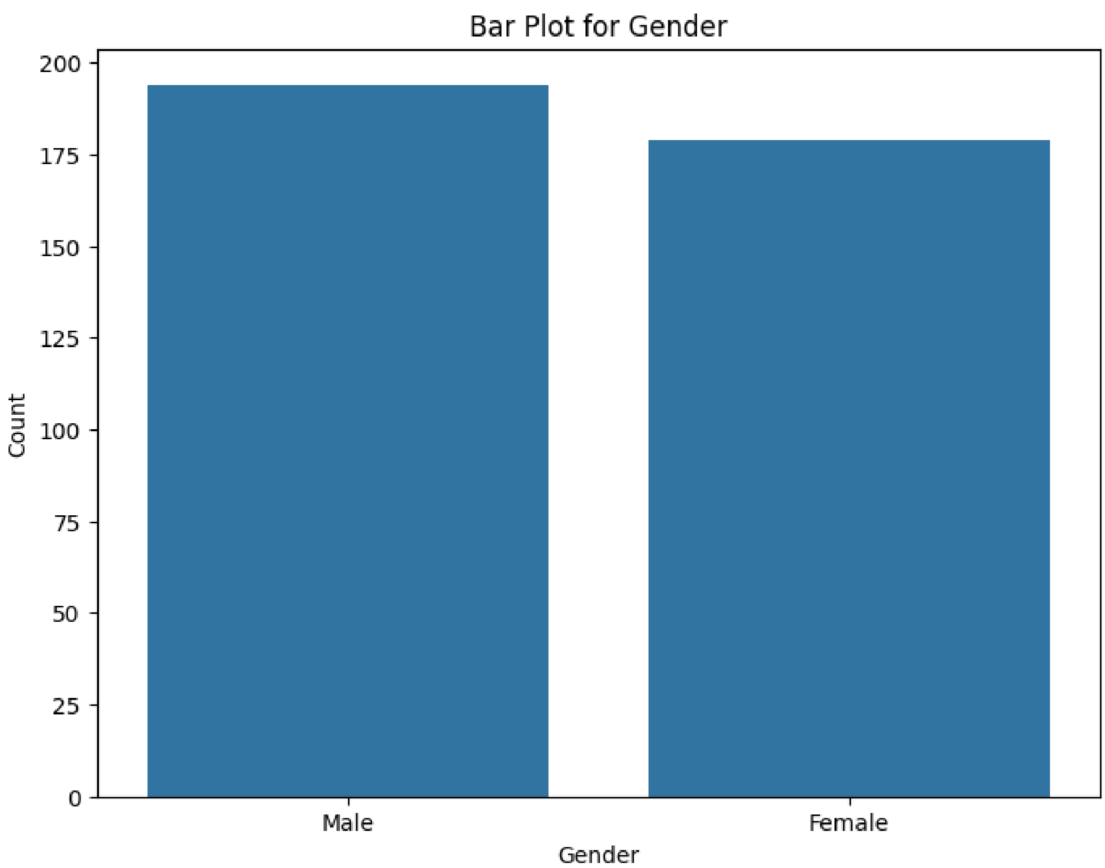



[ ]:
```
#For categorical variables: a. Display frequency tables showing counts and␣
↪percentages.

# Display frequency table for 'Gender'
gender_frequency = df['Gender'].value_counts()
gender_percentage = df['Gender'].value_counts(normalize=True) * 100

print("Frequency Table for 'Gender':")
print(gender_frequency)
print("\nPercentage Table for 'Gender':")
print(gender_percentage)
```

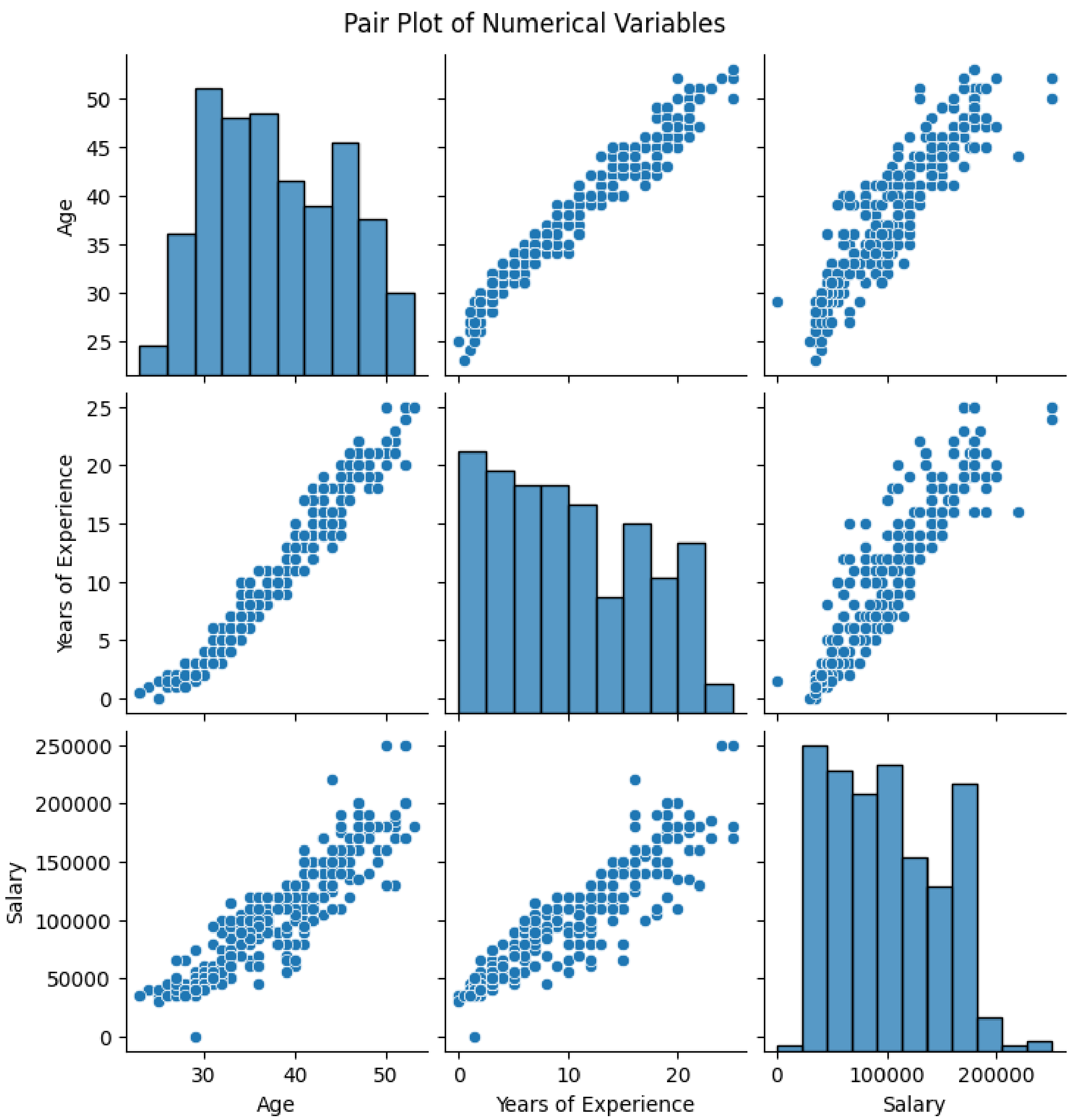
```
Frequency Table for 'Gender':
Male      194
Female    179
Name: Gender, dtype: int64
```

```
Percentage Table for 'Gender':
Male      52.010724
Female    47.989276
Name: Gender, dtype: float64
```

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

# Bar plot for 'Gender'
plt.figure(figsize=(8, 6))
sns.countplot(x='Gender', data=df)
plt.title('Bar Plot for Gender')
plt.xlabel('Gender')
plt.ylabel('Count')

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



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

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




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

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

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


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

# Calculate correlation coefficients
correlation_matrix = df.corr()

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

```
Correlation Coefficients:
```

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

```
<ipython-input-48-3a9b5b137c87>:4: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  correlation_matrix = df.corr()
```

[ ]:
```
#Perform Regression Analysis:i. Depending on the nature of your data and the␣
 ↪relationship between variables,choose an
 #appropriate regression model. Common choices include linear regression␣
 ↪(Simple Multiple), polynomial regression, etc.

# Drop rows with NaN values in either 'Years of Experience' or 'Salary'
df.dropna(subset=['Years of Experience', 'Salary'], inplace=True)

# Select independent (X) and dependent (y) variables
X = df[['Years of Experience']]
y = df['Salary']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Create a linear regression model
model = LinearRegression()

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

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Plot the regression line
plt.scatter(X_test, y_test, color='black')
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.title('Linear Regression - Years of Experience vs. Salary')
plt.xlabel('Years of Experience')
```

```
plt.ylabel('Salary')
plt.show()
```

```
Mean Squared Error: 241834883.8999349
R-squared: 0.8991338517367767
```


[ ]:
```
#Perform Regression Analysis: Train the chosen regression model using the
↪training data.

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
↪random_state=42)

# Create a linear regression model
model = LinearRegression()

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

[ ]:
```
LinearRegression()
```

```
[ ]: #Perform Regression Analysis: Evaluate the performance of the trained model␣
      ↪using appropriate metrics. For regression,
     #common metrics include Mean Squared Error (MSE), Root Mean Squared Error␣
      ↪(RMSE), R-squared, etc.

     # Make predictions on the test set
     y_pred = model.predict(X_test)

     # Evaluate the model
     mse = mean_squared_error(y_test, y_pred)
     rmse = np.sqrt(mse)
     r2 = r2_score(y_test, y_pred)

     print(f'Mean Squared Error (MSE): {mse:.2f}')
     print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
     print(f'R-squared (R2): {r2:.2f}')
```

```
Mean Squared Error (MSE): 241834883.90
Root Mean Squared Error (RMSE): 15551.04
R-squared (R2): 0.90
```

**R square**

- Formula
- R^2 = 1 - SSR/SST
- R^2 = coefficient of determination SSR = sum of squares of residuals SST = total sum of squares
- R-squared is a statistical measure that indicates how much of the variation of a dependent variable is explained by an independent variable in a regression model

```
[ ]: #Interpret the coefficients of the regression model to understand the␣
      ↪relationship between independent and dependent
      #variables. Identify significant predictors and assess the overall␣
      ↪goodness-of-fit of the model.

      # Select independent (X) and dependent (y) variables
     X = df[['Years of Experience']]
     y = df['Salary']

     # Split the dataset into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
      ↪random_state=42)

     # Create a linear regression model
     model = LinearRegression()
```

```python
# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Interpret the coefficients
coefficients = pd.DataFrame({'Variable': X.columns, 'Coefficient': model.coef_})
print("Coefficients:")
print(coefficients)

# Assess the overall goodness-of-fit
print(f'\nMean Squared Error (MSE): {mse:.2f}')
print(f'R-squared (R2): {r2:.2f}')
```

```
Coefficients:
              Variable  Coefficient
0  Years of Experience  6822.590175

Mean Squared Error (MSE): 241834883.90
R-squared (R2): 0.90
```

**conclusion:**

In conclusion, Regression Analysis proves to be a powerful tool in the realm of Machine Learning, particularly in the context of predicting salaries from a given dataset. Through the exploration of relationships between various factors such as education, experience, and job type, regression models enable us to make informed predictions about salary levels. The accuracy of these predictions relies heavily on the quality and diversity of the dataset, as well as the appropriateness of the chosen regression algorithm.

In the case of salary prediction, regression analysis helps uncover patterns and trends within the data, providing valuable insights for both individuals and organizations. By leveraging machine learning techniques, we can create models that not only capture the nuances of salary variations but also adapt to changing circumstances. This adaptability is crucial in dynamic job markets where factors influencing salaries are subject to constant evolution.

While regression analysis offers a robust foundation for salary prediction, it is important to acknowledge its limitations. Overfitting, multicollinearity, and outliers are challenges that require careful consideration and preprocessing. Additionally, the quality of predictions is only as good as the features included in the model, emphasizing the significance of feature engineering and domain knowledge.