The University of Faisalabad

ML PROJECT MANUAL

Registration No:

2023-BS-AI-022

Submitted to:

SIR SAEED

Submitted by:

ABEER AWAIS

Section:

(A)

Degree program:

ARTIFICIAL INTELLIGENCE

Subject:

MACHINE LEARNING

Contents

PROJECT 1

The dataset contains **30 entries** and **3 columns**, structured as follows:

1. **Unnamed: 0**: An index column (probably auto-generated and not meaningful for analysis).
2. **YearsExperience**: A numerical column representing the number of years of professional experience.
3. **Salary**: A numerical column representing the corresponding salary (likely in USD or another currency).

Sample Data:

YearsExperience	Salary
1.2	39344.0
1.4	46206.0
1.6	37732.0
2.1	43526.0
2.3	39892.0

This dataset is likely used for analyzing or predicting salary based on years of experience, a typical use case for linear regression modeling.

Loading required libraries:

Importing libraries and loading dataset

```
[8]: import matplotlib.pyplot as plt
```

```
[9]: import seaborn as sns
     color =sns.color_palette()
```

```
[10]: import numpy as np
```

```
[11]: import pandas as pd
```

```
[12]: data = pd.read_csv("C://Users//ADMIN//MACHINE LEARNING//archive (1)//Salary_dataset.csv")
```

Displaying dataset head :

Preview of the Diamonds Dataset

[13]:
```
data.head()
```

[13]:

	Unnamed: 0	YearsExperience	Salary
0	0	1.2	39344.0
1	1	1.4	46206.0
2	2	1.6	37732.0
3	3	2.1	43526.0
4	4	2.3	39892.0

First 30 Rows

[14]:
```
data.head(30)
```

[14]:

	Unnamed: 0	YearsExperience	Salary
0	0	1.2	39344.0
1	1	1.4	46206.0
2	2	1.6	37732.0
3	3	2.1	43526.0
4	4	2.3	39892.0
5	5	3.0	56643.0
6	6	3.1	60151.0
7	7	3.3	54446.0
8	8	3.3	64446.0
9	9	3.8	57190.0
10	10	4.0	63219.0
11	11	4.1	55795.0
12	12	4.1	56958.0
13	13	4.2	57082.0
14	14	4.6	61112.0

15	15	5.0	67939.0
16	16	5.2	66030.0
17	17	5.4	83089.0
18	18	6.0	81364.0
19	19	6.1	93941.0
20	20	6.9	91739.0
21	21	7.2	98274.0
22	22	8.0	101303.0
23	23	8.3	113813.0
24	24	8.8	109432.0
25	25	9.1	105583.0
26	26	9.6	116970.0
27	27	9.7	112636.0
28	28	10.4	122392.0
29	29	10.6	121873.0

Preview of tail of dataset :

Preview of the Last Rows

[15]:
```
data.tail()
```

[15]:

	Unnamed: 0	YearsExperience	Salary
25	25	9.1	105583.0
26	26	9.6	116970.0
27	27	9.7	112636.0
28	28	10.4	122392.0
29	29	10.6	121873.0

Displaying shape:

Dimensions of the Diamonds Dataset

[16]:
```
data.shape
```

[16]:
```
(30, 3)
```

Preview of the Last 20 Rows

[17]:
```
data.tail(20)
```

[17]:

	Unnamed: 0	YearsExperience	Salary
10	10	4.0	63219.0
11	11	4.1	55795.0
12	12	4.1	56958.0
13	13	4.2	57082.0
14	14	4.6	61112.0
15	15	5.0	67939.0
16	16	5.2	66030.0
17	17	5.4	83089.0
18	18	6.0	81364.0
19	19	6.1	93941.0
20	20	6.9	91739.0
21	21	7.2	98274.0
22	22	8.0	101303.0
23	23	8.3	113813.0
24	24	8.8	109432.0
25	25	9.1	105583.0
26	26	9.6	116970.0
27	27	9.7	112636.0
28	28	10.4	122392.0
29	29	10.6	121873.0

Random Sample of 30 Rows

[18]:
```
data.sample(30)
```

[18]:

	Unnamed: 0	YearsExperience	Salary
16	16	5.2	66030.0
18	18	6.0	81364.0
5	5	3.0	56643.0
12	12	4.1	56958.0
0	0	1.2	39344.0
24	24	8.8	109432.0
11	11	4.1	55795.0
3	3	2.1	43526.0
29	29	10.6	121873.0
28	28	10.4	122392.0
9	9	3.8	57190.0
10	10	4.0	63219.0
2	2	1.6	37732.0
19	19	6.1	93941.0
13	13	4.2	57082.0
1	1	1.4	46206.0
8	8	3.3	64446.0
7	7	3.3	54446.0
6	6	3.1	60151.0
25	25	9.1	105583.0
14	14	4.6	61112.0
20	20	6.9	91739.0
27	27	9.7	112636.0
21	21	7.2	98274.0
4	4	2.3	39892.0
17	17	5.4	83089.0
23	23	8.3	113813.0
15	15	5.0	67939.0
22	22	8.0	101303.0
26	26	9.6	116970.0

Information of dataset :

DataFrame Info Summary

[19]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Unnamed: 0       30 non-null     int64  
 1   YearsExperience  30 non-null     float64
 2   Salary           30 non-null     float64
dtypes: float64(2), int64(1)
memory usage: 852.0 bytes
```

Describing the dataset :

Descriptive Statistics of the Diamonds Dataset

[20]:
```
data.describe()
```

[20]:

	Unnamed: 0	YearsExperience	Salary
count	30.000000	30.000000	30.000000
mean	14.500000	5.413333	76004.000000
std	8.803408	2.837888	27414.429785
min	0.000000	1.200000	37732.000000
25%	7.250000	3.300000	56721.750000
50%	14.500000	4.800000	65238.000000
75%	21.750000	7.800000	100545.750000
max	29.000000	10.600000	122392.000000

Counting missing values:

Count of Missing Values per Column in the Dataset

22]:
```
data.isnull().sum()
```

22]:
```
Unnamed: 0         0
YearsExperience    0
Salary             0
dtype: int64
```

23]:
```python
import pandas as pd
```

24]:
```python
import numpy as np
```

25]:
```python
numeric_cols =data.select_dtypes(include=[np.number])
```

26]:
```python
non_numeric_cols =data.select_dtypes(exclude=[np.number])
```

27]:
```python
numeric_cols.fillna(numeric_cols.mean(),inplace= True )
```

28]:
```python
for col in non_numeric_cols.columns:
    non_numeric_cols[col].fillna(non_numeric_cols[col].mode()[0], inplace=True)
```

29]:
```python
data = pd.concat([numeric_cols,non_numeric_cols], axis=1)
```

30]:
```python
missing_values=data.isnull().sum()
```

31]:
```python
print(missing_values)
```

```
Unnamed: 0         0
YearsExperience    0
Salary             0
dtype: int64
```

32]:
```python
data.isnull().sum()
```

32]:
```
Unnamed: 0         0
YearsExperience    0
Salary             0
dtype: int64
```

Dropping rows with null values:

Missing Values After Dropping Rows with Nulls

33]:
```python
data.dropna(inplace=True)
missing_values= data.isnull().sum()
print(missing_values)
```

```
Unnamed: 0         0
YearsExperience    0
Salary             0
dtype: int64
```

[34]:
```python
data.shape
```

[34]:
```
(30, 3)
```

[35]:
```python
data.drop_duplicates(inplace=True)
data.shape
```

[35]:
```
(30, 3)
```

[36]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

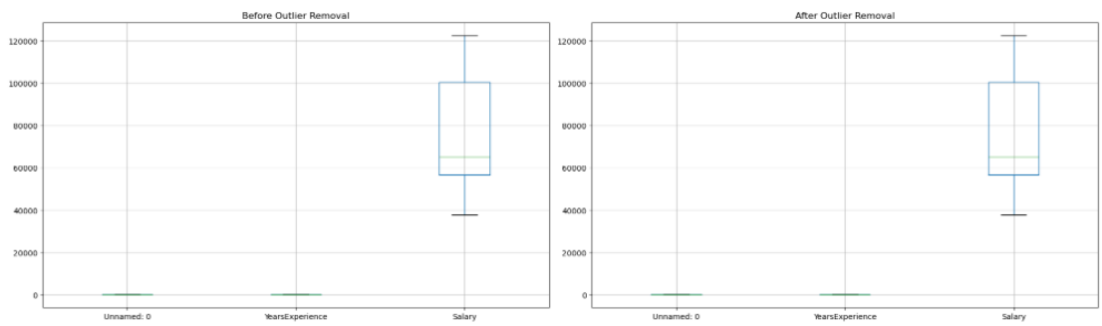
```
data.describe()
```

	Unnamed: 0	YearsExperience	Salary
count	30.000000	30.000000	30.000000
mean	14.500000	5.413333	76004.000000
std	8.803408	2.837888	27414.429785
min	0.000000	1.200000	37732.000000
25%	7.250000	3.300000	56721.750000
50%	14.500000	4.800000	65238.000000
75%	21.750000	7.800000	100545.750000
max	29.000000	10.600000	122392.000000

Comparison of numeric features after and before removing outliers and displaying it in graph form:

Comparison of Numeric Features Before and After Outlier Removal Using IQR Method

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Select numeric columns
numeric_cols = data.select_dtypes(include=[np.number])

# Step 2: Calculate IQR for each numeric column
Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1

# Step 3: Remove outliers using IQR method
data_cleaned = data[~((numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * IQR))).any(axis=1)]

# Step 4: Plot before outlier removal
plt.figure(figsize=(20, 6))

plt.subplot(1, 2, 1)
numeric_cols.boxplot()
plt.title("Before Outlier Removal")

# Optional: Plot after outlier removal
plt.subplot(1, 2, 2)
data_cleaned.select_dtypes(include=[np.number]).boxplot()
plt.title("After Outlier Removal")

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



Min max scaling :

```python
from sklearn.preprocessing import MinMaxScaler
```

Dataset After Min-Max Scaling of Numeric Features and Combining with Non-Numeric Features

```python
numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


scaler = MinMaxScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)


scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)


scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)


print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(3, 3)

************************************************************
```

	age	income	city
0	0.0	0.0	NY
1	0.5	0.5	LA
2	1.0	1.0	SF

Dataset after scaling:

Dataset After Standard Scaling of Numeric Features and Integration with Non-Numeric Columns

```python
numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


scaler = StandardScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)


scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)


scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
(133, 13)
```

```
(3, 3)

************************************************************

(133, 13)
```

PROJECT 2

Description :

diamonds dataset, which contains detailed information about 53,940 diamonds.

What the dataset is about:

This dataset provides characteristics of diamonds and their prices. It can be used to understand how different features (like size or quality) affect the price of a diamond.

What each column means:

Column	**Description:**
carat	Weight of the diamond (a key factor in pricing).
cut	Quality of the cut (e.g., Ideal, Premium, Good). Better cuts sparkle more.
color	Color grade of the diamond, from best (D) to worst (J).
clarity	Clarity grade — fewer flaws mean better clarity (e.g., VS1, SI2).
depth	Total depth of the diamond (percentage of height vs. width).
table	Width of the top of the diamond (as a percentage).
price	Price in US dollars 💵. This is what we may want to predict or analyze.
x, y, z	Physical dimensions (length, width, depth) in millimeters.
Unnamed: 0	Just an index column — not useful for analysis.

Example Row:

One diamond in the dataset:

0.23 carats, Ideal cut, E color, SI2 clarity

Depth: 61.5%, Table: 55%

Dimensions: 3.95mm x 3.98mm x 2.43mm

Price: $326

Importing required libraries :

Import Libraries and Load Dataset

```
[4]: import pandas as pd
     import numpy as np
     import plotly.express as px
     import plotly.graph_objects as go

     data = pd.read_csv("C:/Users/ADMIN/Downloads/diamonds.csv")
     print(data.head())
```

```
   Unnamed: 0  carat      cut color clarity  depth  table  price     x     y  \
0           1   0.23    Ideal     E     SI2   61.5   55.0    326  3.95  3.98
1           2   0.21  Premium     E     SI1   59.8   61.0    326  3.89  3.84
2           3   0.23     Good     E     VS1   56.9   65.0    327  4.05  4.07
3           4   0.29  Premium     I     VS2   62.4   58.0    334  4.20  4.23
4           5   0.31     Good     J     SI2   63.3   58.0    335  4.34  4.35

      z
0  2.43
1  2.31
2  2.31
3  2.63
4  2.75
```

Displaying tail of dataset :

View Last Few Rows

```
[5]: data.tail()
```

[5]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

Shape of dataset :

Check Dataset Dimensions

```
[6]: data.shape
```

```
[6]: (53940, 11)
```

View a Random Sample

7]:
```
data.sample
```

7]:
```
<bound method NDFrame.sample of        Unnamed: 0  carat        cut color clarity  depth  table  price     x  \
0               1   0.23      Ideal     E     SI2   61.5   55.0    326  3.95
1               2   0.21    Premium     E     SI1   59.8   61.0    326  3.89
2               3   0.23       Good     E     VS1   56.9   65.0    327  4.05
3               4   0.29    Premium     I     VS2   62.4   58.0    334  4.20
4               5   0.31       Good     J     SI2   63.3   58.0    335  4.34
...           ...    ...        ...   ...     ...    ...    ...    ...   ...
53935       53936   0.72      Ideal     D     SI1   60.8   57.0   2757  5.75
53936       53937   0.72       Good     D     SI1   63.1   55.0   2757  5.69
53937       53938   0.70  Very Good     D     SI1   62.8   60.0   2757  5.66
53938       53939   0.86    Premium     H     SI2   61.0   58.0   2757  6.15
53939       53940   0.75      Ideal     D     SI2   62.2   55.0   2757  5.83

          y     z
0      3.98  2.43
1      3.84  2.31
2      4.07  2.31
3      4.23  2.63
4      4.35  2.75
...     ...   ...
53935  5.76  3.50
53936  5.75  3.61
53937  5.68  3.56
53938  6.12  3.74
53939  5.87  3.64

[53940 rows x 11 columns]>
```

Displaying data summary :

Dataset Info Summary

8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53940 non-null  int64
 1   carat       53940 non-null  float64
 2   cut         53940 non-null  object
 3   color       53940 non-null  object
 4   clarity     53940 non-null  object
 5   depth       53940 non-null  float64
 6   table       53940 non-null  float64
 7   price       53940 non-null  int64
 8   x           53940 non-null  float64
 9   y           53940 non-null  float64
 10  z           53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

Describing dataset:

Descriptive Statistics

[9]:
```
data.describe()
```

[9]:

	Unnamed: 0	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	26970.500000	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	15571.281097	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	1.000000	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	13485.750000	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	26970.500000	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	40455.250000	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	53940.000000	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

Checking for null values:

Check for Missing Values

[10]:
```
data.isnull().sum()
```

[10]:
```
Unnamed: 0    0
carat         0
cut           0
color         0
clarity       0
depth         0
table         0
price         0
x             0
y             0
z             0
dtype: int64
```

removing unnecessary columns:

Remove Unnecessary Index Column

[11]:
```
if "Unnamed: 0" in data.columns:
    data = data.drop("Unnamed: 0", axis=1)
```

Clear Output in Jupyter Notebook

[12]:
```
from IPython.display import display, clear_output
clear_output(wait=True)
```

Create a New Feature – size

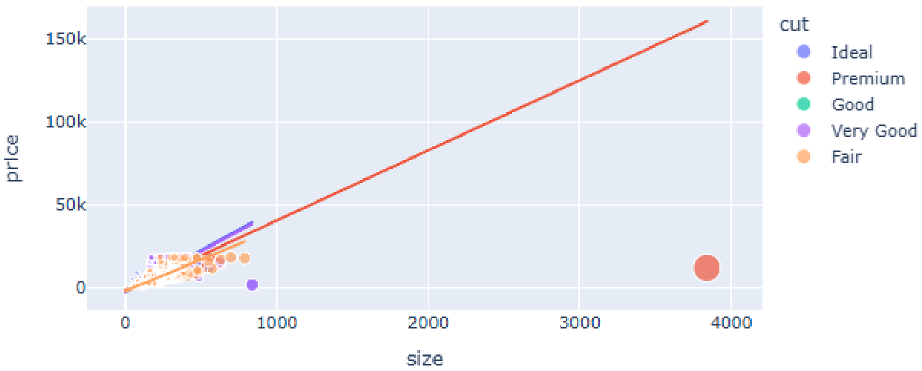
```
[13]: data["size"] = data["x"] * data["y"] * data["z"]
print(data)
```

```
       carat        cut color clarity  depth  table  price     x     y     z  \
0       0.23      Ideal     E     SI2   61.5   55.0    326  3.95  3.98  2.43
1       0.21    Premium     E     SI1   59.8   61.0    326  3.89  3.84  2.31
2       0.23       Good     E     VS1   56.9   65.0    327  4.05  4.07  2.31
3       0.29    Premium     I     VS2   62.4   58.0    334  4.20  4.23  2.63
4       0.31       Good     J     SI2   63.3   58.0    335  4.34  4.35  2.75
...      ...        ...   ...     ...    ...    ...    ...   ...   ...   ...
53935   0.72      Ideal     D     SI1   60.8   57.0   2757  5.75  5.76  3.50
53936   0.72       Good     D     SI1   63.1   55.0   2757  5.69  5.75  3.61
53937   0.70  Very Good     D     SI1   62.8   60.0   2757  5.66  5.68  3.56
53938   0.86    Premium     H     SI2   61.0   58.0   2757  6.15  6.12  3.74
53939   0.75      Ideal     D     SI2   62.2   55.0   2757  5.83  5.87  3.64

             size
0       38.202030
1       34.505856
2       38.076885
3       46.724580
4       51.917250
...           ...
53935  115.920000
53936  118.110175
53937  114.449728
53938  140.766120
53939  124.568444

[53940 rows x 11 columns]
```

Plotting dataset :

Scatter Plot – Diamond Size vs Price

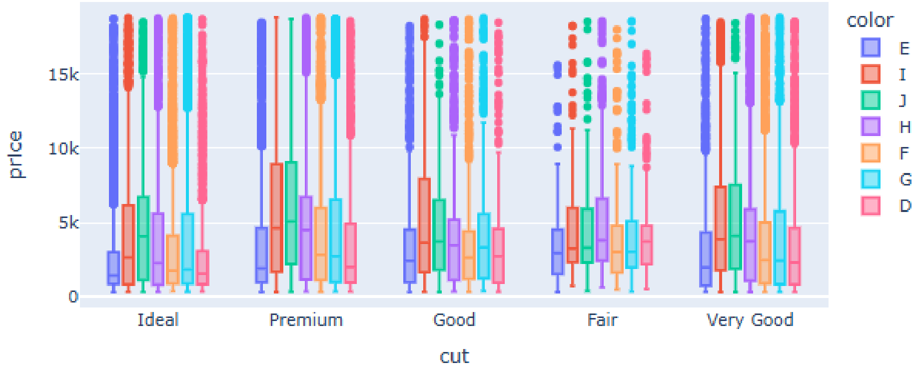
```
[14]: figure = px.scatter(data_frame = data, x="size",
                    y="price", size="size",
                    color= "cut", trendline="ols")
figure.show()
```



Plotting price distribution :

Box Plot – Price Distribution by Cut and Color

```
15]: fig = px.box(data, x="cut",
                 y="price",
                 color="color")
     fig.show()
```



Box Plot – Price Distribution by Cut and Clarity

```
[16]: fig = px.box(data,
                  x="cut",
                  y="price",
                  color="clarity")
      fig.show()
```

Encode cut Feature Numerically

```
[17]: data["cut"] = data["cut"].map({"Ideal": 1,
                                     "Premium": 2,
                                     "Good": 3,
                                     "Very Good": 4,
                                     "Fair": 5})
```

Sorted Correlation with Price

[18]:
```
correlation = data.corr(numeric_only=True)
print(correlation["price"].sort_values(ascending=False))
```

```
price     1.000000
carat     0.921591
size      0.902385
x         0.884435
y         0.865421
z         0.861249
table     0.127134
cut       0.049421
depth    -0.010647
Name: price, dtype: float64
```

Splitting data :

Splitting data

[19]:
```
from sklearn.model_selection import train_test_split
x = np.array(data[["carat", "cut", "size"]])
y = np.array(data[["price"]])

xtrain, xtest, ytrain, ytest = train_test_split(x, y,
                                                test_size=0.10,
                                                random_state=42)
```

Applying random forest :

Train Random Forest Model

[20]:
```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
model.fit(xtrain, ytrain)
```

```
C:\Users\ADMIN\anaconda3\Lib\site-packages\sklearn\base.py:1389: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
```

[20]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

Diamond Price Prediction Input

[21]:
```
print("Diamond Price Prediction")
a = float(input("Carat Size: "))
b = int(input("Cut Type (Ideal: 1, Premium: 2, Good: 3, Very Good: 4, Fair: 5): "))
c = float(input("Size: "))
features = np.array([[a, b, c]])
print("Predicted Diamond's Price = ", model.predict(features))
```

```
Diamond Price Prediction
Carat Size:  2
Cut Type (Ideal: 1, Premium: 2, Good: 3, Very Good: 4, Fair: 5):  3
Size:  2
Predicted Diamond's Price =  [15953.595]
```

ann and confusion matrix

```
df = df.drop(columns=["Unnamed: 0"])

threshold = df['Salary'].median()
df['SalaryClass'] = (df['Salary'] > threshold).astype(int)


X = df[['YearsExperience']].values
y = df['SalaryClass'].values


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


model = Sequential()
model.add(Dense(10, activation='relu', input_dim=1))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


model.fit(X_train_scaled, y_train, epochs=100, verbose=0)


y_pred_prob = model.predict(X_test_scaled)
```


```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


model = Sequential()
model.add(Dense(10, activation='relu', input_dim=1))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


model.fit(X_train_scaled, y_train, epochs=100, verbose=0)


y_pred_prob = model.predict(X_test_scaled)
```


```
y_pred = (y_pred_prob > 0.5).astype(int)


conf_matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

print("Confusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", report)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
Confusion Matrix:
 [[2 0]
 [1 3]]

Classification Report:
               precision    recall  f1-score   support

           0       0.67      1.00      0.80         2
           1       1.00      0.75      0.86         4

    accuracy                           0.83         6
   macro avg       0.83      0.88      0.83         6
weighted avg       0.89      0.83      0.84         6
```

confusion martix

In [4]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt


y_pred_prob = model.predict(X_test_scaled)
y_pred = (y_pred_prob > 0.5).astype(int)


conf_matrix = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=["Low Salary", "High Salary"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 61ms/step
```