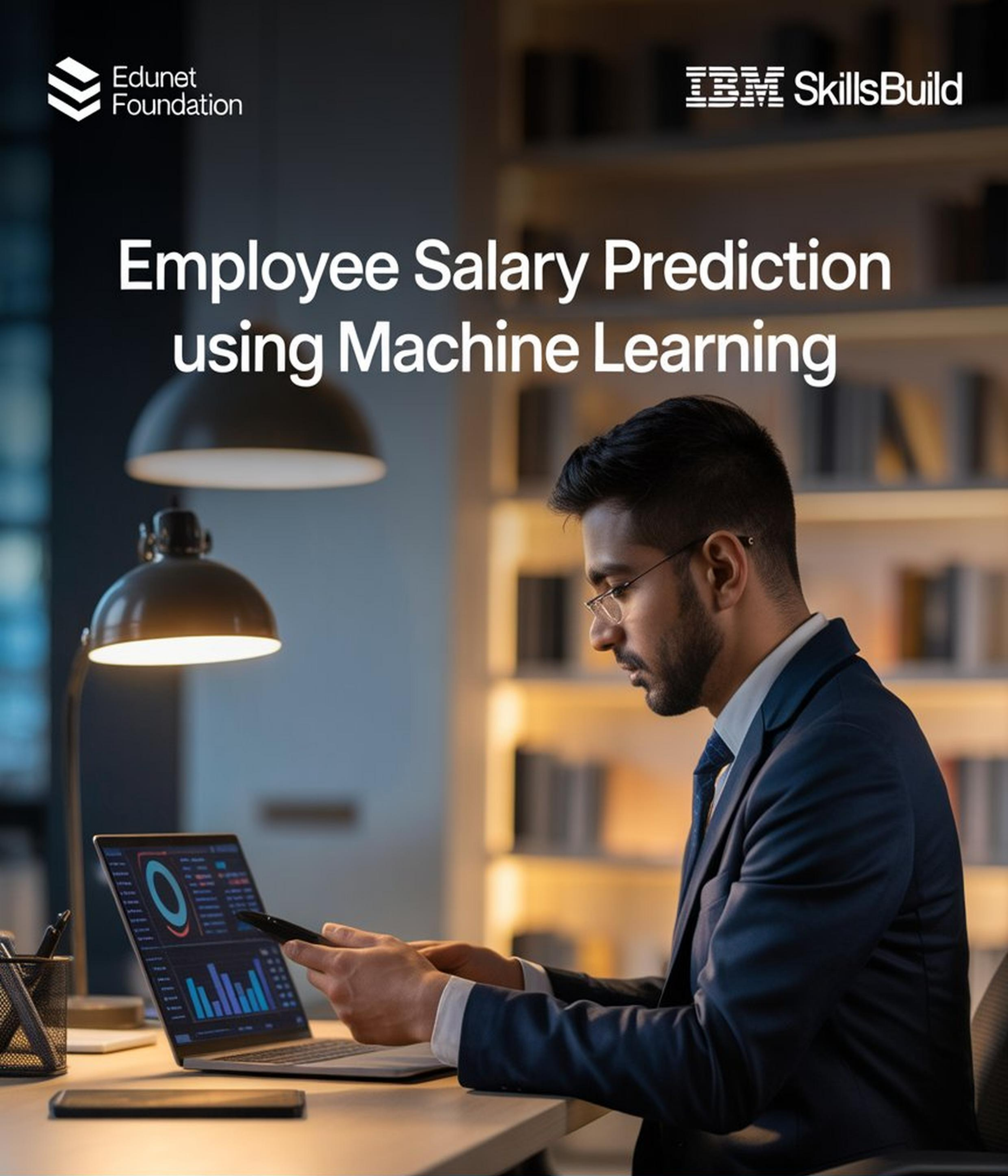
Edunet Foundation

IBM SkillsBuild

Employee Salary Prediction using Machine Learning



PRESENTED BY

NITIN RAJ

Trainer :- Dr. Nanthini Mohan Mam

Project :- Employee-salary-prediction-project

```
import pandas as pd
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("rkiattisak/salaly-prediction-for-beginer")

print("Path to dataset files:", path)
```

```
Path to dataset files: /kaggle/input/salaly-prediction-for-beginer
```

```
df=pd.read_csv("/kaggle/input/salaly-prediction-for-beginer/Salary Data.csv")
df.head()
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
0	32.0	Male	Bachelor's	Software Engineer	5.0	90000.0
1	28.0	Female	Master's	Data Analyst	3.0	65000.0
2	45.0	Male	PhD	Senior Manager	15.0	150000.0
3	36.0	Female	Bachelor's	Sales Associate	7.0	60000.0
4	52.0	Male	Master's	Director	20.0	200000.0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.shape
```

```
(375, 6)
```

```
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
```

```
df.tail()
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
370	35.0	Female	Bachelor's	Senior Marketing Analyst	8.0	85000.0
371	43.0	Male	Master's	Director of Operations	19.0	170000.0
372	29.0	Female	Bachelor's	Junior Project Manager	2.0	40000.0
373	34.0	Male	Bachelor's	Senior Operations Coordinator	7.0	90000.0
374	44.0	Female	PhD	Senior Business Analyst	15.0	150000.0

```
df.sample()
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
281	41.0	Female	Bachelor's	Senior Project Coordinator	11.0	95000.0

```
df.describe()
```

	Age	Years of Experience	Salary
count	373.000000	373.000000	373.000000
mean	37.431635	10.030831	100577.345845
std	7.069073	6.557007	48240.013482
min	23.000000	0.000000	350.000000
25%	31.000000	4.000000	55000.000000
50%	36.000000	9.000000	95000.000000
75%	44.000000	15.000000	140000.000000
max	53.000000	25.000000	250000.000000

```
df.duplicated().sum()
```

```
np.int64(50)
```

```
df.duplicated()
```

	0
0	False
1	False
2	False
3	False
4	False
...	...
370	True
371	False
372	True
373	True
374	True

375 rows × 1 columns

dtype: bool

```
df.isna().sum()
```

	0
Age	0
Gender	0
Education Level	0
Job Title	0
Years of Experience	0
Salary	0

dtype: int64

```
df.isnull().sum()
```

	0
Age	0
Gender	0
Education Level	0
Job Title	0
Years of Experience	0
Salary	0

dtype: int64

```
df["Education Level"].value_counts()
```

Education Level	count
Bachelor's	224
Master's	98
PhD	51

dtype: int64

```
df["Gender"].value_counts()
```

Gender	count
Male	194
Female	179

dtype: int64

```
df["Job Title"].value_counts()
```

Job Title	count
Director of Marketing	12
Director of Operations	11
Senior Business Analyst	10
Senior Marketing Manager	9
Senior Marketing Analyst	9
...	...
Junior Social Media Specialist	1
Junior Operations Coordinator	1
Senior HR Specialist	1
Director of HR	1
Junior Financial Advisor	1

174 rows × 1 columns

dtype: int64

```
df.dropna(subset=["Gender"], inplace=True)
```

```
df.fillna({"Age": 37, "Salary": 100577,"Education Level":"Bachelor's","Job Title":"Director of Marketing","Years
```

```
df.isnull().sum()
```

	0
Age	0
Gender	0
Education Level	0
Job Title	0
Years of Experience	0
Salary	0

dtype: int64

```
df["Age"].value_counts().sum()
```

```
np.int64(373)
```

```
df["Salary"].value_counts().unique().sum()
```

```
np.int64(198)
```

```
df["Age"].value_counts().unique()
```

```
array([24, 23, 22, 21, 20, 17, 15, 13, 12, 11, 10,  9,  8,  7,  5,  4,  3,
        1])
```

```
print(df.Age.value_counts())
```

```
Age
33.0    24
29.0    23
```

```
35.0    22
44.0    21
31.0    21
36.0    20
45.0    17
34.0    17
47.0    15
30.0    15
38.0    15
40.0    13
28.0    13
32.0    12
39.0    12
43.0    12
41.0    12
37.0    12
42.0    11
46.0    10
27.0     9
48.0     9
50.0     8
49.0     8
26.0     7
51.0     5
25.0     4
52.0     3
24.0     1
23.0     1
53.0     1
Name: count, dtype: int64
```

```
print(df.Gender.value_counts())
```

```
Gender
Male      194
Female    179
Name: count, dtype: int64
```

```
print(df.Salary.value_counts())
```

```
Salary
40000.0     31
50000.0     22
95000.0     22
120000.0    20
180000.0    20
45000.0     18
150000.0    18
90000.0     18
160000.0    17
60000.0     17
110000.0    17
170000.0    16
130000.0    14
100000.0    14
140000.0    14
35000.0     13
80000.0     12
55000.0     10
85000.0     10
65000.0      9
70000.0      9
105000.0     6
75000.0      4
190000.0     4
115000.0     3
200000.0     2
135000.0     2
250000.0     2
175000.0     2
125000.0     1
30000.0      1
220000.0     1
```

```
185000.0     1
145000.0     1
155000.0     1
350.0        1
Name: count, dtype: int64
```

```
print(df["Education Level"].value_counts())
```

```
Education Level
Bachelor's    224
Master's       98
PhD            51
Name: count, dtype: int64
```

```
print(df["Years of Experience"].value_counts().sum())
```

```
373
```

```
print(df["Years of Experience"].value_counts())
```

```
Years of Experience
2.0     31
3.0     30
8.0     25
9.0     22
4.0     20
16.0    18
10.0    18
7.0     18
5.0     17
15.0    16
19.0    15
12.0    15
21.0    13
14.0    13
18.0    13
20.0    13
1.5     12
6.0     12
13.0    11
11.0    10
22.0     9
1.0      7
17.0     5
0.0      3
25.0     3
23.0     2
24.0     1
0.5      1
Name: count, dtype: int64
```

```
print(df["Job Title"].value_counts().sum())
```

```
373
```

```
print(df["Job Title"].value_counts())
```

```
Job Title
Director of Marketing               12
Director of Operations              11
Senior Business Analyst             10
Senior Marketing Manager             9
Senior Marketing Analyst             9
                                    ..
Junior Social Media Specialist       1
Junior Operations Coordinator        1
Senior HR Specialist                 1
Director of HR                       1
Junior Financial Advisor             1
Name: count, Length: 174, dtype: int64
```

```
df = df[df["Years of Experience"] != 0.5]
df = df[df["Years of Experience"] != 24.0]
```

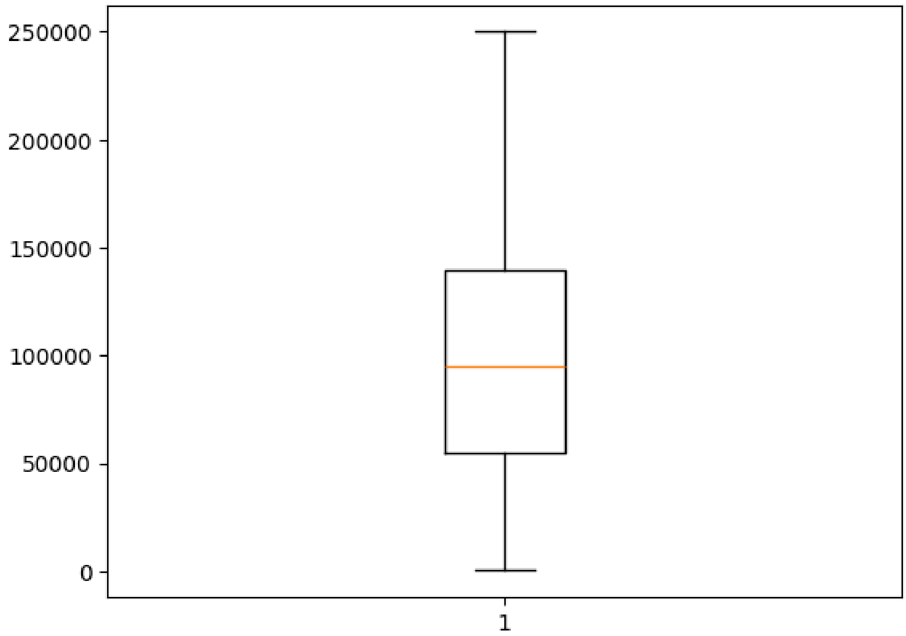
```
print(df["Years of Experience"].value_counts())
```

```
Years of Experience
2.0     31
3.0     30
8.0     25
9.0     22
4.0     20
10.0    18
7.0     18
16.0    18
5.0     17
15.0    16
12.0    15
19.0    15
14.0    13
20.0    13
21.0    13
18.0    13
1.5     12
6.0     12
13.0    11
11.0    10
22.0     9
1.0      7
17.0     5
0.0      3
25.0     3
23.0     2
Name: count, dtype: int64
```

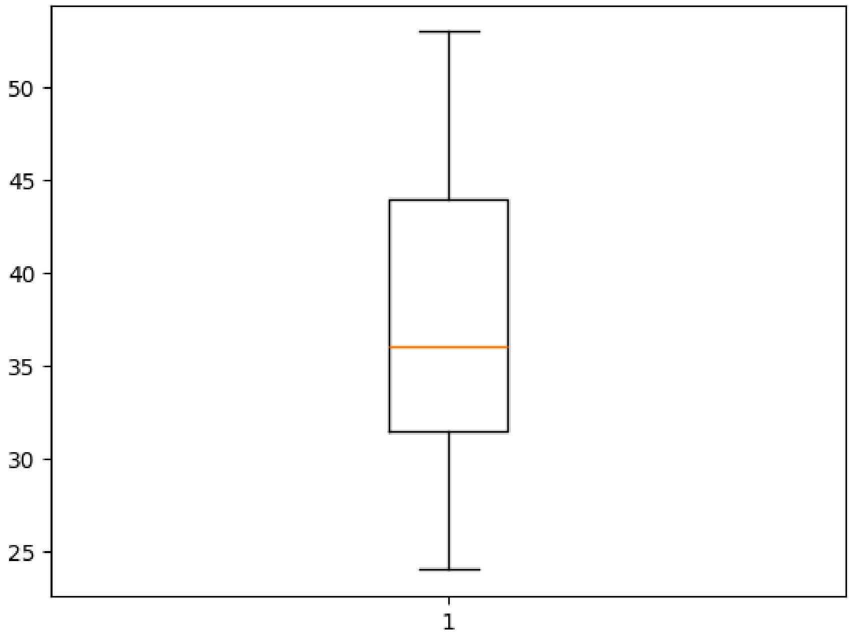
```
df.sample(5)
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
3	36.0	Female	Bachelor's	Sales Associate	7.0	60000.0
63	47.0	Male	PhD	Senior Data Scientist	21.0	180000.0
335	47.0	Male	Master's	Director of Operations	19.0	170000.0
8	26.0	Female	Bachelor's	Marketing Coordinator	1.0	45000.0
53	47.0	Male	Master's	VP of Finance	19.0	200000.0

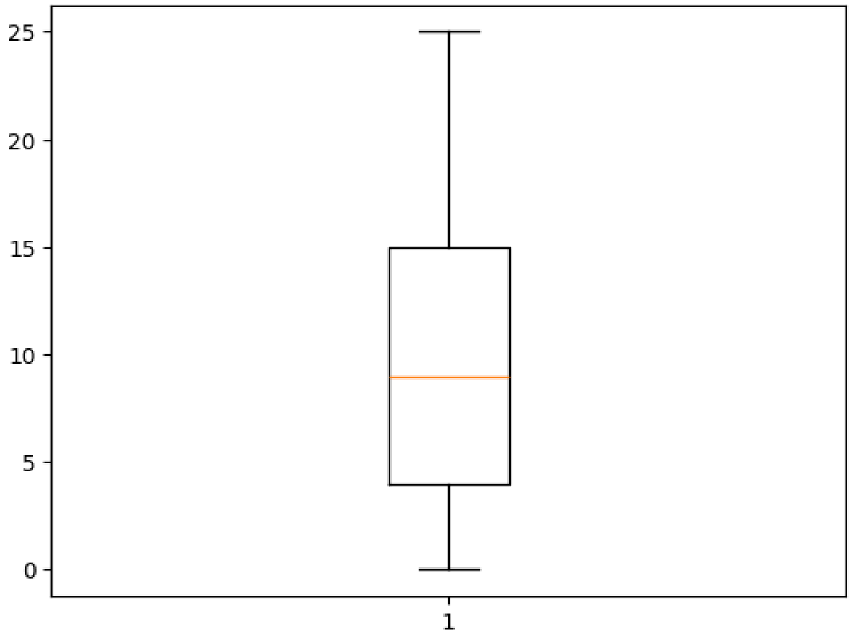
```
import matplotlib.pyplot as plt
import seaborn as sna
plt.boxplot(df["Salary"])
plt.show()
```



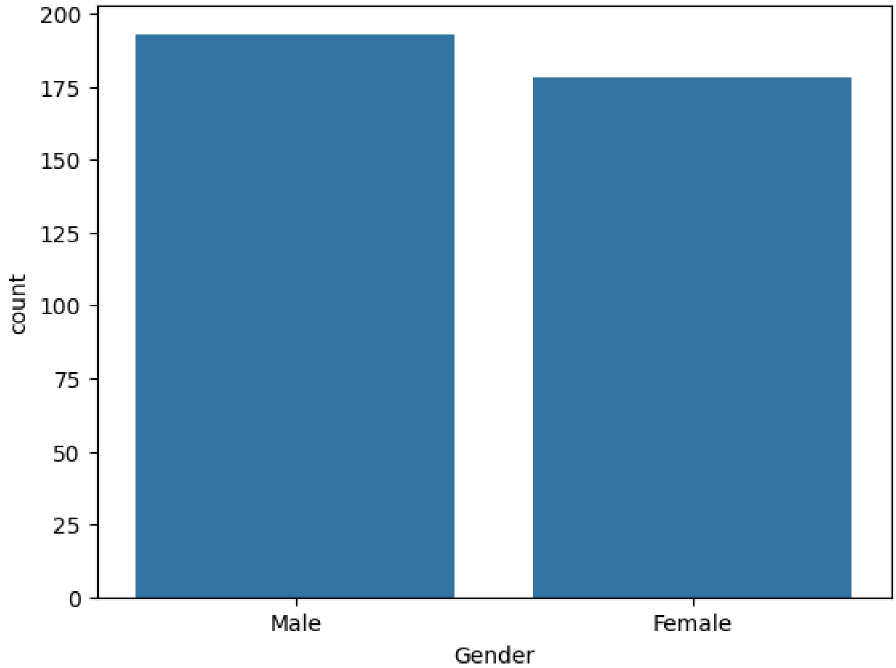

```
import matplotlib.pyplot as plt
import seaborn as sna
plt.boxplot(df["Age"])
plt.show()
```



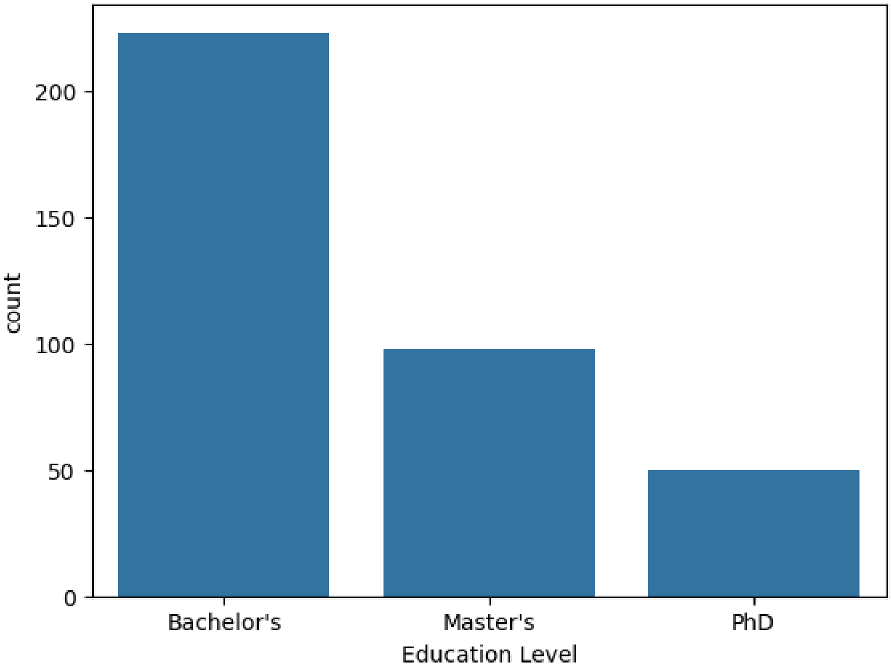

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.boxplot(df["Years of Experience"])
plt.show()
```




```
import matplotlib.pyplot as plt
import seaborn as sns
sns.countplot(x=df["Gender"])
plt.show()
```




```
sns.countplot(x=df["Education Level"])
plt.show()
```




```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df["Gender"]=le.fit_transform(df["Gender"])
df["Education Level"]=le.fit_transform(df["Education Level"])
df["Job Title"]=le.fit_transform(df["Job Title"])
df
```

	Age	Gender	Education Level	Job Title	Years of Experience	Salary
0	32.0	1	0	158	5.0	90000.0
1	28.0	0	1	16	3.0	65000.0
2	45.0	1	2	129	15.0	150000.0
3	36.0	0	0	100	7.0	60000.0
4	52.0	1	1	21	20.0	200000.0
...	...	...	...	...	...	...
370	35.0	0	0	130	8.0	85000.0
371	43.0	1	1	29	19.0	170000.0
372	29.0	0	0	69	2.0	40000.0
373	34.0	1	0	136	7.0	90000.0
374	44.0	0	2	109	15.0	150000.0

371 rows × 6 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
x=df.drop(columns=["Salary"]) # income
y=df["Salary"]      # output
x
```

	Age	Gender	Education Level	Job Title	Years of Experience
0	32.0	1	0	158	5.0
1	28.0	0	1	16	3.0
2	45.0	1	2	129	15.0
3	36.0	0	0	100	7.0
4	52.0	1	1	21	20.0
...	...	...	...	...	...
370	35.0	0	0	130	8.0
371	43.0	1	1	29	19.0
372	29.0	0	0	69	2.0
373	34.0	1	0	136	7.0
374	44.0	0	2	109	15.0

371 rows × 5 columns

Next steps: Generate code with x | View recommended plots | New interactive sheet

```
y
```

	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

371 rows × 1 columns

dtype: float64

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler() # all data convert randge 0 to 1
x=scaler.fit_transform(x) #x is input
x
```

```
array([[0.27586207, 1.        , 0.        , 0.91860465, 0.2       ],
       [0.13793103, 0.        , 0.5       , 0.09302326, 0.12      ],
       [0.72413793, 1.        , 1.        , 0.75      , 0.6       ],
       ...,
       [0.17241379, 0.        , 0.        , 0.40116279, 0.08      ],
       [0.34482759, 1.        , 0.        , 0.79069767, 0.28      ],
       [0.68965517, 0.        , 1.        , 0.63372093, 0.6       ]])
```

```
from sklearn.model_selection import train_test_split
train_X,test_X,train_y,test_y=train_test_split(x,y,test_size=0.2, random_state=23)
```

```
train_X
```

```
array([[0.27586207, 1.        , 0.        , 0.        , 0.2       ],
       [0.82758621, 1.        , 0.5       , 0.1627907 , 0.76      ],
       [0.24137931, 0.        , 0.        , 0.28488372, 0.12      ],
       ...,
       [0.34482759, 0.        , 0.5       , 0.97674419, 0.2       ],
       [0.37931034, 1.        , 0.        , 0.78488372, 0.32      ],
       [0.17241379, 0.        , 0.        , 0.31976744, 0.08      ]])
```

```
# Machine learning algorithm
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(train_X,train_y)
predict=knn.predict(test_X)
predict
```

```
array([110000.,  60000., 180000.,  85000., 180000.,  95000.,  50000.,
       120000.,  60000., 150000., 110000., 130000., 150000.,  85000.,
        45000.,  40000.,  35000.,  40000., 150000., 160000., 150000.,
       110000., 110000.,  80000., 150000.,  40000., 140000.,  40000.,
        90000.,  85000., 180000.,  50000.,  50000.,  80000.,  90000.,
        35000., 180000.,  95000., 150000.,  95000.,  40000., 110000.,
       150000., 110000., 160000.,  35000., 120000.,  40000.,  50000.,
       170000., 150000.,  40000.,  35000.,  40000.,  95000., 170000.,
        40000.,  95000., 150000.,  60000.,  40000., 130000.,  40000.,
        95000.,  45000.,  60000.,  50000., 120000.,  70000., 120000.,
       120000.,  95000.,  50000.,  60000., 110000.])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(test_y,predict)
```

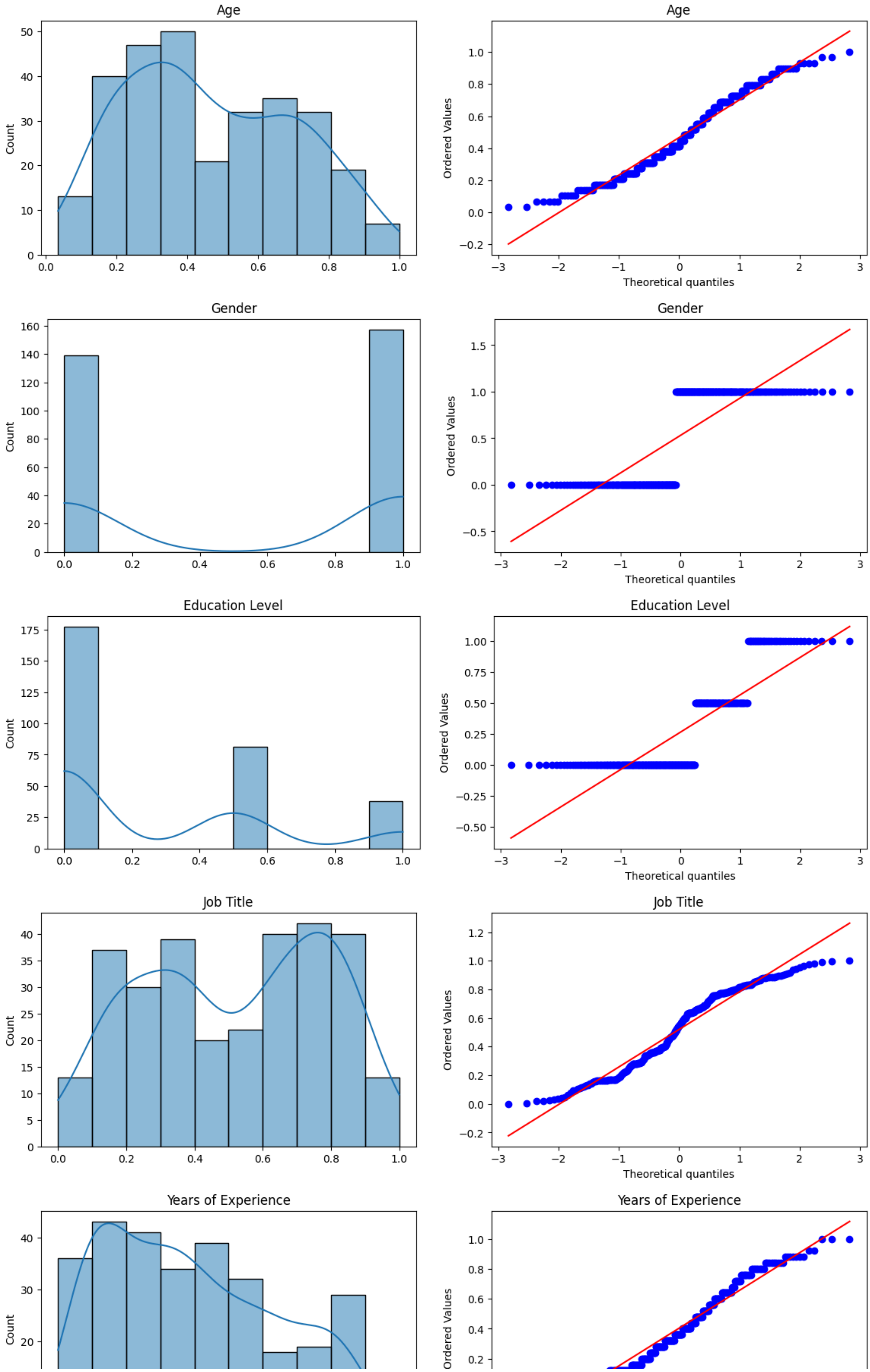
```
0.3333333333333333
```

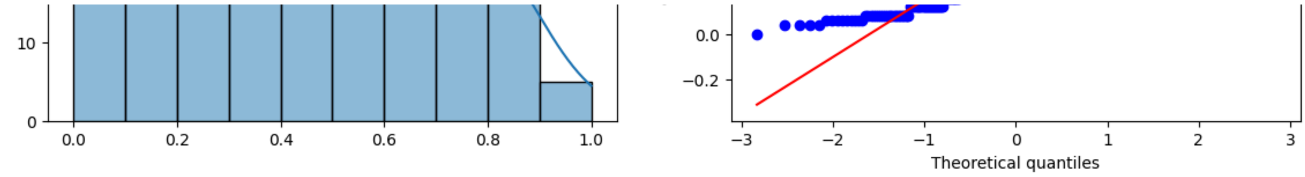
```
from sklearn.linear_model import LinearRegression
kr=LinearRegression()
kr.fit(train_X,train_y)
predict=kr.predict(test_X)
predict
```

```
array([163399.66959553,  69612.84529615, 124970.8938659 ,  80063.68526019,
       166701.50906871,  89241.10992043,  54403.80507276,  88850.57363891,
        60028.57959129, 154400.11737174, 107578.43867805, 140279.39126805,
       153296.05645863,  77155.07065112,  49104.957204  ,  38575.83111042,
        41596.08879032,  44474.03504069, 147693.50575137, 175027.40598431,
       188649.2596052 , 160215.33192407,  85451.26529068, 161434.5878072 ,
       151001.10118939,  40106.66322876, 165075.41214097,  32691.69860987,
        82657.17173485,  89149.10484434, 166670.84071002,  60733.95184133,
        26847.0739474 , 106272.43893691,  88811.46073278,  39822.20345305,
       160700.56186427,  84310.39830314, 151491.79492855, 101802.89500153,
        46015.60535711, 121694.14300536, 174046.01850599, 148080.01270127,
       162504.44164346,  40374.52607549,  89065.25214979,  40014.65815267,
        21030.13689514, 163907.70767877, 185035.85707786,  41199.98594379,
        31434.00373739,  42524.58409867,  95732.75086403, 149548.93695149,
        37870.45886038,  95300.52904089, 170923.01755193,  65723.13537464,
        42739.26260955, 164587.2910373 ,  43106.99074804,  89547.50134152,
        76046.13027083,  69275.49335048,  35357.94709784, 136824.91028026,
        57312.12751595, 137100.63334265, 112048.832749  ,  78409.60855634,
        48372.18950963,  90619.28734311, 125834.48737661])
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

from sklearn.preprocessing import PowerTransformer
```

```
# Plotting the distplots without any transformation
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import numpy as np

for i, col in enumerate(df.columns[:-1]):  # Iterate through original DataFrame columns (excluding target)
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.histplot(train_X[:, i], kde=True) # Use histplot for distribution and access column by index
    plt.title(col)

    plt.subplot(122)
    stats.probplot(train_X[:, i], dist="norm", plot=plt) # Access column by index
    plt.title(col)

    plt.show()
```


Age
Count
Age
Ordered Values
Theoretical quantiles
Gender
Count
Gender
Ordered Values
Theoretical quantiles
Education Level
Count
Education Level
Ordered Values
Theoretical quantiles
Job Title
Count
Job Title
Ordered Values
Theoretical quantiles
Years of Experience
Count
Years of Experience
Ordered Values


10
0
0.0
0.2
0.4
0.6
0.8
1.0
0.0
−0.2
−3
−2
−1
0
1
2
3
Theoretical quantiles

```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
train_X_transformed = pt.fit_transform(train_X + 0.0000001)
test_X_transformed = pt.transform(test_X + 0.0000001)

lr = LinearRegression()
lr.fit(train_X_transformed,train_y)

pred2_y = lr.predict(test_X_transformed)

r2_score(test_y,pred2_y)
```

```
0.9026625160006326
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

mse = mean_squared_error(test_y, predict)
mae = mean_absolute_error(test_y, predict)
r2 = r2_score(test_y, predict)

print("MSE:", mse)
print("MAE:", mae)
print("R2 Score:", r2)
```

```
MSE: 234511500.54412812
MAE: 11495.470321853129
R2 Score: 0.8983723273836117
```

```
from sklearn.model_selection import cross_val_score
scores=cross_val_score(kr,x,y,cv=5)
scores
```

```
array([0.87546405, 0.68049049, 0.95436488, 0.90664077, 0.90729046])
```

```
from sklearn.linear_model import LogisticRegression
lt=LogisticRegression()
lt.fit(train_X,train_y)
predict=lt.predict(test_X)
predict
```

```
array([160000.,  40000., 180000.,  95000., 180000.,  95000.,  50000.,
        95000.,  40000., 160000., 120000., 130000., 160000.,  95000.,
        95000.,  40000.,  40000.,  40000., 160000., 160000., 180000.,
       160000.,  95000., 180000., 160000.,  50000., 160000.,  40000.,
        95000.,  95000., 180000.,  40000.,  40000., 120000.,  95000.,
        40000., 180000.,  90000., 160000., 110000.,  50000., 130000.,
       160000., 170000., 160000.,  40000.,  95000.,  50000.,  40000.,
       170000., 180000.,  40000.,  40000.,  50000., 130000., 170000.,
        50000.,  90000., 160000.,  40000.,  50000., 130000.,  50000.,
        95000.,  50000.,  40000.,  50000., 160000.,  50000., 160000.,
       130000., 120000.,  50000.,  50000., 160000.])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(test_y,predict)
```

```
0.29333333333333333
```

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(train_X,train_y)
predict=rf.predict(test_X)
predict
```

```
array([160000.,  50000.,  80000.,  95000., 180000.,  95000.,  50000.,
       120000.,  50000., 160000.,  95000., 140000., 160000.,  85000.,
        45000.,  35000.,  45000.,  50000., 160000., 160000., 180000.,
```

```
       160000., 120000., 250000., 150000.,  40000., 150000.,  35000.,
        90000.,  95000., 180000.,  50000.,  35000., 120000., 100000.,
        35000., 180000.,  95000., 150000.,  90000.,  45000., 150000.,
       160000., 110000., 150000.,  35000., 120000.,  40000.,  35000.,
       170000., 180000.,  35000.,  35000.,  40000., 100000., 170000.,
        35000.,  90000., 155000.,  60000.,  40000., 135000.,  45000.,
        95000.,  60000.,  70000.,  40000., 120000.,  55000., 150000.,
       130000., 100000.,  50000.,  80000., 110000.])
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

mse = mean_squared_error(test_y, predict)
mae = mean_absolute_error(test_y, predict)
r2 = r2_score(test_y, predict)

print("MSE:", mse)
print("MAE:", mae)
print("R2 Score:", r2)
```

```
MSE: 295666666.6666667
MAE: 7400.0
R2 Score: 0.8718701848998459
```

```
import numpy as np

# Consider prediction correct if within 10,000 of true value
def regression_accuracy(y_true, y_pred, threshold=10000):
    return np.mean(np.abs(y_true - y_pred) < threshold)

acc = regression_accuracy(test_y, predict)
print("Custom Regression Accuracy:", acc)
```

```
Custom Regression Accuracy: 0.68
```

```
from sklearn.metrics import accuracy_score
accuracy_score(test_y,predict)
```

```
0.49333333333333335
```

```
from sklearn.ensemble import GradientBoostingRegressor
gbr=GradientBoostingRegressor()
gbr.fit(train_X,train_y)
predict=gbr.predict(test_X)
predict
```

```
array([175095.97164123,  60239.79307523, 121638.78405033,  89801.80212512,
       175032.02216667,  96388.16518115,  56756.33851675,  97334.99403492,
        52816.10390558, 156366.00637872, 109802.36968573, 115800.63864024,
       158814.45773161,  87460.80631442,  46463.42064661,  32848.63674156,
        43122.0910041 ,  43122.0910041 , 143395.4352571 , 155949.56622098,
       184547.65637449, 175095.97164123,  96545.720971  , 235857.18724474,
       150788.99818126,  39147.79297419, 159558.53204545,  55551.83180645,
        92535.10728885,  93179.50603544, 177134.09956239,  53313.00045598,
        36979.71655941, 108291.71840247, 100087.99565653,  41347.21276858,
       179489.52538523, 102373.62171966, 148374.18567806,  96517.02092457,
        45897.61897257, 135516.23227397, 157648.28886342, 126066.60166359,
       153388.02234373,  34294.26766892,  96413.02864398,  39964.21605127,
        39295.56180515, 173717.52269127, 179400.72879662,  35153.37616843,
        31105.5386774 ,  40872.11498731,  91677.02039579, 165350.82311546,
        34047.52219142,  95430.7982612 , 157783.84447112,  59052.19518782,
        40872.11498731, 152783.24906435,  46683.57984617,  93623.56864845,
        68925.43937043,  63124.4142376 ,  46542.83811203, 141328.11278395,
        59950.63159482, 136385.94317704, 112410.87271323,  94471.37403486,
        48316.99796492,  81895.8498847 , 120079.2813275 ])
```

```python
# Step 1: Import
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Step 2: Create Model
gbr = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)

# Step 3: Train Model
gbr.fit(train_X, train_y)

# Step 4: Predict
predict = gbr.predict(test_X)

# Step 5: Evaluate Model
mse = mean_squared_error(test_y, predict)
mae = mean_absolute_error(test_y, predict)
r2 = r2_score(test_y, predict)

print("MSE:", mse)
print("MAE:", mae)
print("R2 Score:", r2)
```

```
MSE: 269923034.90580136
MAE: 8650.10764571466
R2 Score: 0.8830264197731023
```

```python
import numpy as np

def regression_accuracy(y_true, y_pred, threshold=10000):
    return np.mean(np.abs(y_true - y_pred) < threshold)

acc = regression_accuracy(test_y, predict)
print("Custom Regression Accuracy (±₹10K):", acc)
```

```
Custom Regression Accuracy (±₹10K): 0.7466666666666667
```

```python
gbr = GradientBoostingRegressor(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=4,
    random_state=42
)
gbr.fit(train_X, train_y)
predict = gbr.predict(test_X)
```

```python
# Evaluation
print("R2:", r2_score(test_y, predict))
print("Custom Accuracy:", regression_accuracy(test_y, predict))
```

```
R2: 0.8788188649904828
Custom Accuracy: 0.7466666666666667
```

```python
# Step 1: Import
from sklearn.ensemble import VotingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Step 2: Define individual models
lr = LinearRegression()
rf = RandomForestRegressor(n_estimators=100, random_state=42)
gbr = GradientBoostingRegressor(n_estimators=200, learning_rate=0.05, max_depth=4, random_state=42)

# Step 3: Create VotingRegressor
voting_model = VotingRegressor(estimators=[
```

```
    ('lr', lr),
    ('rf', rf),
    ('gbr', gbr)
])

# Step 4: Train
voting_model.fit(train_X, train_y)

# Step 5: Predict
predict = voting_model.predict(test_X)

# Step 6: Evaluate
mse = mean_squared_error(test_y, predict)
mae = mean_absolute_error(test_y, predict)
r2 = r2_score(test_y, predict)

print("MSE:", mse)
print("MAE:", mae)
print("R2 Score:", r2)

# Optional: Custom Accuracy (within ±₹10K)
def regression_accuracy(y_true, y_pred, threshold=10000):
    return np.mean(np.abs(y_true - y_pred) < threshold)

acc = regression_accuracy(test_y, predict)
print("Custom Accuracy (±10K):", acc)
```

```
MSE: 192980814.77955002
MAE: 8449.27604992183
R2 Score: 0.9163700244117898
Custom Accuracy (±10K): 0.7333333333333333
```

```
from sklearn.neural_network import MLPClassifier
clf=MLPClassifier(solver="adam", hidden_layer_sizes=(5,2), random_state=2, max_iter=2000)
clf.fit(train_X,train_y)
predict2=clf.predict(test_X)
predict2
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWar
  warnings.warn(
array([160000.,  60000., 180000.,  90000., 180000.,  95000.,  50000.,
        95000.,  50000., 160000.,  95000., 130000., 160000.,  90000.,
        45000.,  40000.,  40000.,  40000., 160000., 160000., 160000.,
       160000.,  95000., 180000., 160000.,  40000., 160000.,  40000.,
        90000.,  95000., 180000.,  50000.,  40000., 120000.,  95000.,
        40000., 180000.,  90000., 130000., 120000.,  45000., 130000.,
       160000., 180000., 160000.,  40000.,  95000.,  40000.,  35000.,
       180000., 160000.,  40000.,  40000.,  40000., 120000., 180000.,
        40000.,  90000., 160000.,  60000.,  40000., 160000.,  40000.,
        95000.,  60000.,  60000.,  40000., 130000.,  50000., 130000.,
       120000.,  90000.,  40000.,  60000., 120000.])
```

```
from sklearn.metrics import accuracy_score
accuracy_score(test_y,predict2) # accuracy (100%)
```

```
0.37333333333333335
```

```
# Step 1: Import Libraries
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Step 2: Create and Configure the Model
model = MLPRegressor(hidden_layer_sizes=(100, 50), activation='relu', max_iter=500, random_state=42)

# Step 3: Train the Model
model.fit(train_X, train_y)
```

```
# Step 4: Predict
y_pred = model.predict(test_X)

# Step 5: Evaluate the Model
mse = mean_squared_error(test_y, y_pred)
mae = mean_absolute_error(test_y, y_pred)
r2 = r2_score(test_y, y_pred)

print("MSE:", mse)
print("MAE:", mae)
print("R2 Score:", r2)

# Optional: Custom Accuracy (±₹10K)
def regression_accuracy(y_true, y_pred, threshold=10000):
    return np.mean(np.abs(y_true - y_pred) < threshold)

acc = regression_accuracy(test_y, y_pred)
print("Custom Accuracy (±10K):", acc)
```

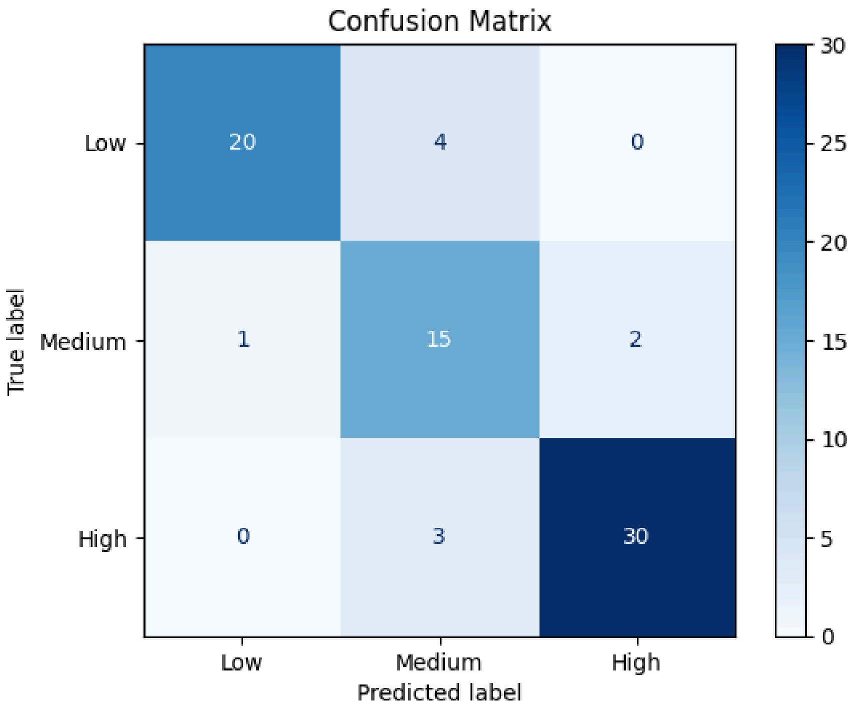
```
MSE: 4460792759.801487
MAE: 55629.636431259576
R2 Score: -0.9331247514548044
Custom Accuracy (±10K): 0.09333333333333334
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWar
  warnings.warn(
```

```
import pandas as pd

# Binning salary into categories
train_y_class = pd.cut(train_y, bins=[0, 50000, 100000, float('inf')], labels=[0, 1, 2])
test_y_class = pd.cut(test_y, bins=[0, 50000, 100000, float('inf')], labels=[0, 1, 2])
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Create Model
clf = MLPClassifier(solver="adam", hidden_layer_sizes=(5,2), random_state=2, max_iter=2000)

# Step 2: Train
clf.fit(train_X, train_y_class)

# Step 3: Predict
predict2 = clf.predict(test_X)

# Step 4: Accuracy
acc = accuracy_score(test_y_class, predict2)
print("Accuracy:", acc)

# Optional: Full classification report
print("\nClassification Report:\n", classification_report(test_y_class, predict2))
```

```
Accuracy: 0.8666666666666667

Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.83      0.89        24
           1       0.68      0.83      0.75        18
           2       0.94      0.91      0.92        33

    accuracy                           0.87        75
   macro avg       0.86      0.86      0.85        75
weighted avg       0.88      0.87      0.87        75
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

cm = confusion_matrix(test_y_class, predict2)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Low", "Medium", "High"])
disp.plot(cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```

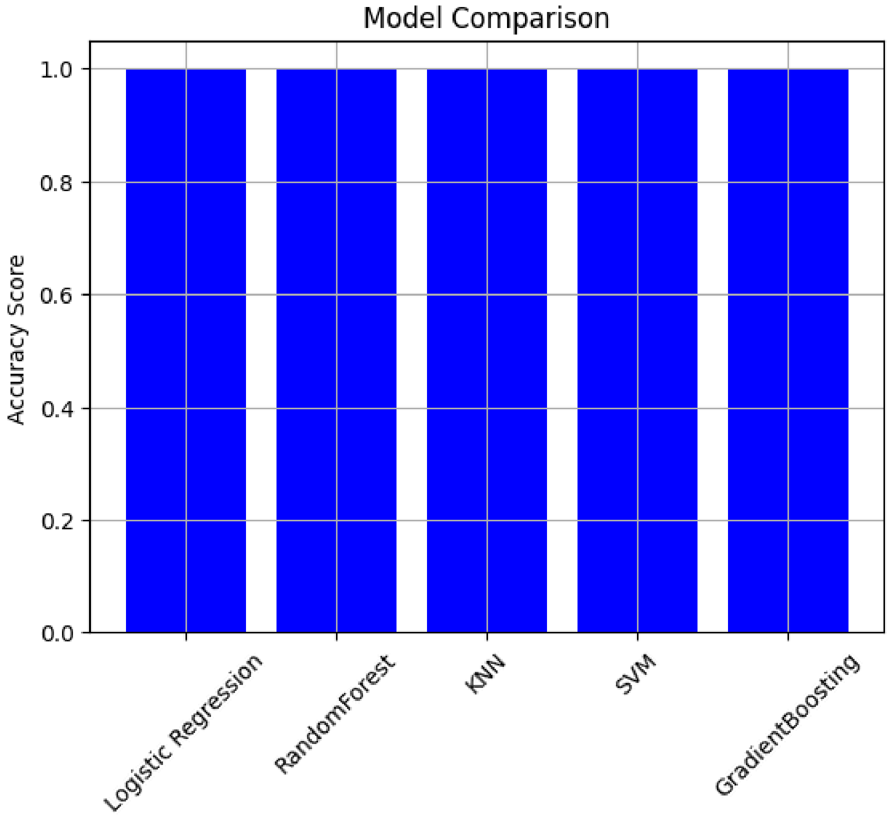



```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# Replace `x` and `y` with your actual feature matrix and target variable
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

models = {
    "Logistic Regression": LogisticRegression(),
    "RandomForest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(),
    "SVM": SVC(),
    "GradientBoosting": GradientBoostingClassifier()
}

results = {}

for name, model in models.items():
    pipe = Pipeline([
        ('scaler', StandardScaler()),
        ('model', model)
    ])

    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
```

```
    print(f"{name} Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred))
```

```
Logistic Regression Accuracy: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        39
           1       1.00      1.00      1.00        36

    accuracy                           1.00        75
   macro avg       1.00      1.00      1.00        75
weighted avg       1.00      1.00      1.00        75

RandomForest Accuracy: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        39
           1       1.00      1.00      1.00        36

    accuracy                           1.00        75
   macro avg       1.00      1.00      1.00        75
weighted avg       1.00      1.00      1.00        75

KNN Accuracy: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        39
           1       1.00      1.00      1.00        36

    accuracy                           1.00        75
   macro avg       1.00      1.00      1.00        75
weighted avg       1.00      1.00      1.00        75

SVM Accuracy: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        39
           1       1.00      1.00      1.00        36

    accuracy                           1.00        75
   macro avg       1.00      1.00      1.00        75
weighted avg       1.00      1.00      1.00        75

GradientBoosting Accuracy: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        39
           1       1.00      1.00      1.00        36

    accuracy                           1.00        75
   macro avg       1.00      1.00      1.00        75
weighted avg       1.00      1.00      1.00        75
```

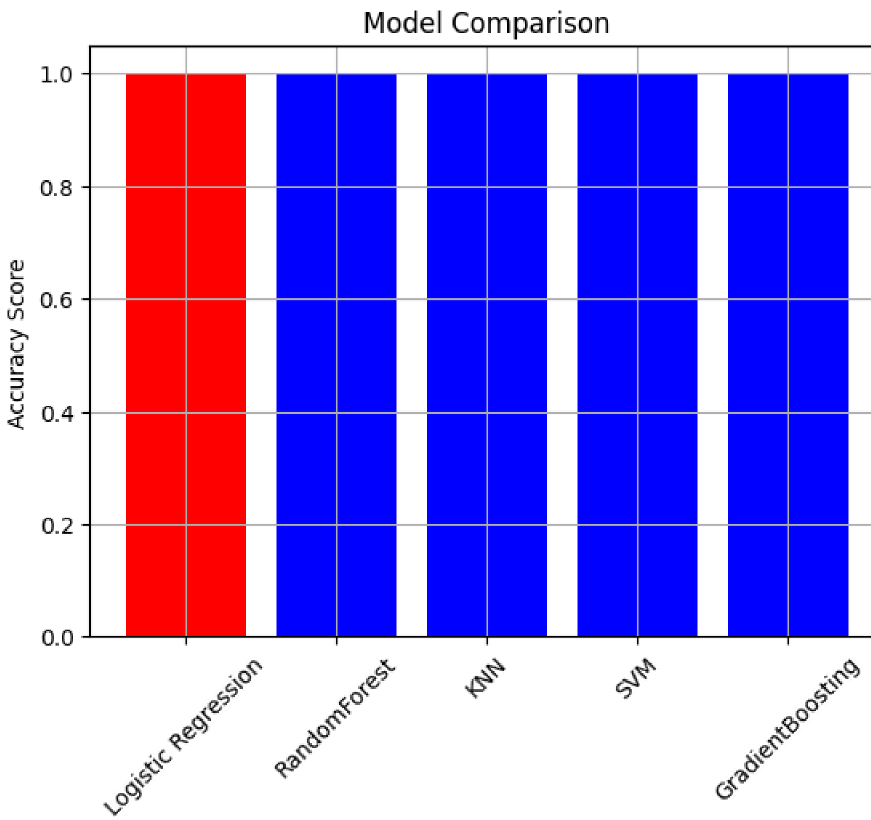
```
import matplotlib.pyplot as plt
plt.bar(results.keys(), results.values(), color='Blue')
plt.ylabel('Accuracy Score')
plt.title('Model Comparison')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```




```
best_model = max(results, key=results.get)
best_score = results[best_model]
print(f"Best model: {best_model} with accuracy {best_score:.4f}")
```

```
Best model: Logistic Regression with accuracy 1.0000
```

```
colors = ['red' if model == best_model else 'blue' for model in results.keys()]
plt.bar(results.keys(), results.values(), color=colors)
plt.ylabel('Accuracy Score')
plt.title('Model Comparison')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```




```
class MeraLR:

    def __init__(self):
        self.m = None
        self.b = None

    def fit(self,X_train,y_train):

        # Ensure X_train is a 1D array (or select a single feature if multi-dimensional
        if X_train.ndim > 1:
            # For this simple LR, we'll use the first feature as an example
            X_train_single = X_train[:, 0]
        else:
            X_train_single = X_train

        num = 0
        den = 0

        mean_X = X_train_single.mean()
        mean_y = y_train.mean()

        for i in range(X_train_single.shape[0]):

            num = num + ((X_train_single[i] - mean_X)*(y_train[i] - mean_y))
            den = den + ((X_train_single[i] - mean_X)*(X_train_single[i] - mean_X))

        self.m = num/den
        self.b = mean_y - (self.m * mean_X)
        print(f"Calculated m: {self.m}")
        print(f"Calculated b: {self.b}")


    def predict(self,X_test):

        # Ensure X_test is a 1D array (or select a single feature if multi-dimensional)
        if X_test.ndim > 1:
             # For this simple LR, we'll use the first feature as an example
            X_test_single = X_test[:, 0]
        else:
```

```
            X_test_single = X_test

        print(f"Input for prediction: {X_test_single}")



        return self.m * X_test_single + self.b
```

```
from sklearn.model_selection import train_test_split
# Use only the first feature (Age) for this simple linear regression
X_train_single, X_test_single, y_train_single, y_test_single = train_test_split(x[:, 0], y, test_size=0.2, rando
```

```
# Assuming 'x' contains your features and 'y' is your target
# If you want to use only the first feature (Age) for MeraLR, keep these lines.
# If you want to use a different single feature, change the index (e.g., x[:, 1] for Gender).
X = x[:,0]
y = y
```

```
X
```

```
array([32., 28., 45., 36., 52., 29., 42., 31., 26., 38., 29., 48., 35.,
       40., 27., 44., 33., 39., 25., 51., 34., 47., 30., 36., 41., 28.,
       37., 24., 43., 33., 50., 31., 29., 39., 46., 27., 35., 42., 26.,
       49., 34., 48., 30., 36., 41., 28., 32., 45., 38., 25., 51., 33.,
       40., 47., 29., 36., 27., 43., 30., 35., 51., 29., 40., 47., 26.,
       38., 46., 31., 34., 49., 33., 39., 45., 28., 42., 37., 50., 32.,
       48., 30., 36., 41., 25., 29., 34., 27., 40., 46., 31., 36., 29.,
       43., 52., 33., 39., 47., 26., 38., 45., 31., 35., 49., 33., 39.,
       44., 30., 36., 41., 28., 42., 37., 50., 32., 31., 40., 48., 29.,
       35., 42., 53., 33., 38., 44., 26., 37., 45., 32., 34., 50., 29.,
       40., 47., 27., 39., 46., 30., 36., 43., 28., 41., 33., 47., 25.,
       34., 42., 31., 38., 45., 29., 36., 43., 26., 37., 44., 32., 33.,
       51., 28., 39., 48., 30., 35., 41., 27., 40., 46., 31., 34., 50.,
       29., 43., 26., 35., 42., 31., 38., 46., 29., 37., 44., 27., 36.,
       43., 33., 34., 50., 28., 39., 47., 30., 34., 40., 28., 41., 45.,
       32., 35., 49., 30., 44., 27., 36., 41., 31., 39., 47., 30., 38.,
       45., 28., 35., 44., 34., 35., 50., 29., 40., 48., 31., 33., 42.,
```