In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

In [2]:
```
df= pd.read_csv('Salary_dataset.csv')
```

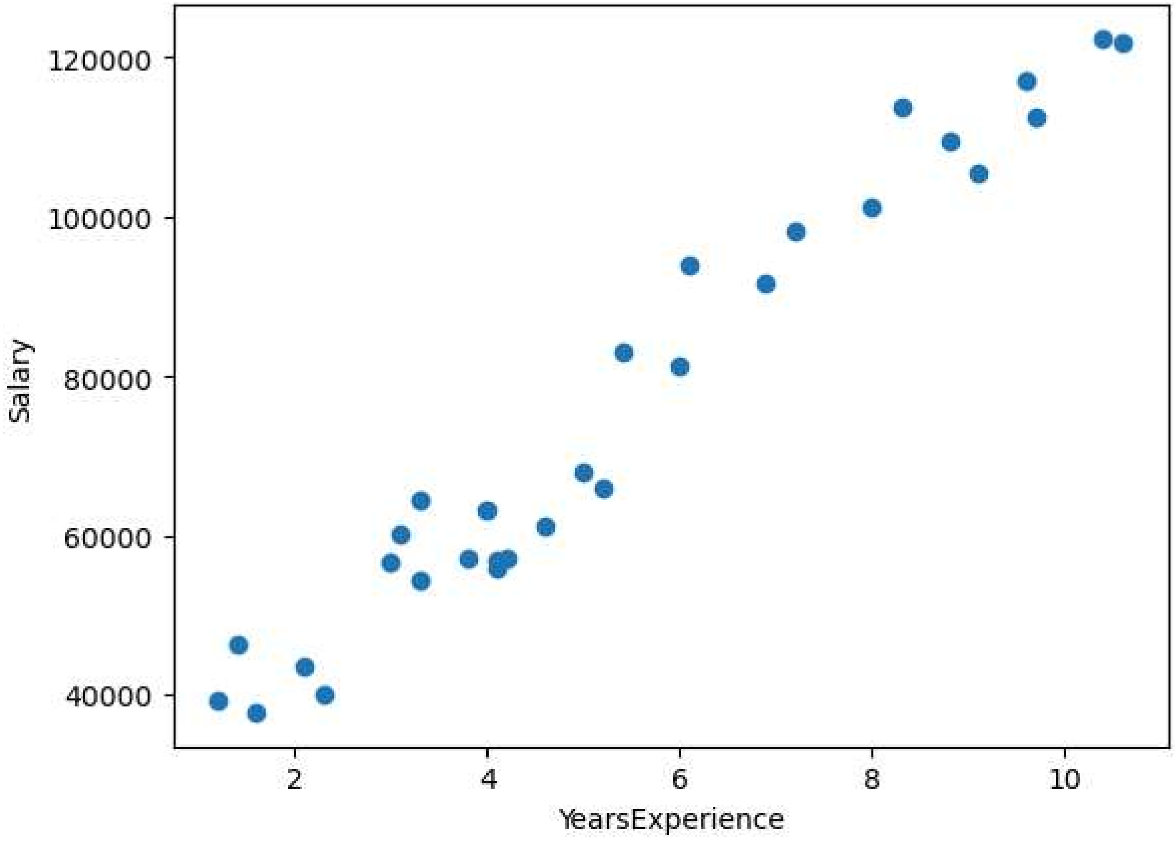
In [3]:
```
df.head()
```

Out[3]:

	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

In [4]:
```
# Scatter Plot
plt.scatter(df['YearsExperience'],df['Salary'])
plt.xlabel("YearsExperience")
plt.ylabel("Salary")
```

Out[4]:
```
Text(0, 0.5, 'Salary')
```



In [5]:
```
# Correlation
df.corr()
```

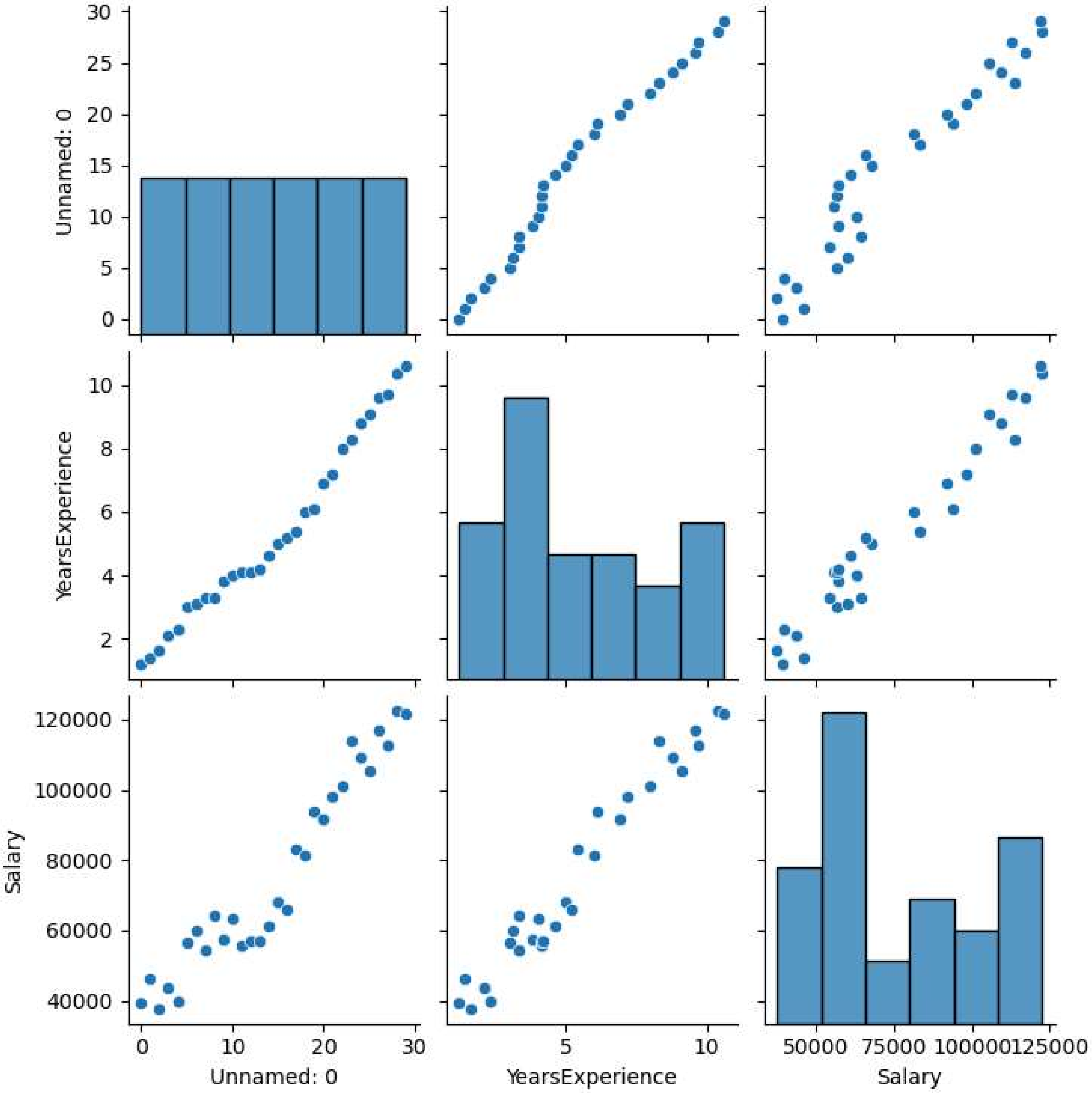
Out[5]:

	Unnamed: 0	YearsExperience	Salary
Unnamed: 0	1.000000	0.986460	0.960826
YearsExperience	0.986460	1.000000	0.978242
Salary	0.960826	0.978242	1.000000

In [7]:
```
# Seaborn for Visualization Plot
import seaborn as sns
sns.pairplot(df)
```

```
C:\Users\win 10\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarn
ing: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[7]:
```
<seaborn.axisgrid.PairGrid at 0x1ef59dc3210>
```



In [8]:

```
# Independent and dependent feature
df.head()
```

Out[8]:

	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

In [11]:

```
X=df[["YearsExperience"]] ## independent features should be data frame or 2 dim
np.array(X)
```

Out[11]:

```
array([[ 1.2],
       [ 1.4],
       [ 1.6],
       [ 2.1],
       [ 2.3],
       [ 3. ],
       [ 3.1],
       [ 3.3],
       [ 3.3],
       [ 3.8],
       [ 4. ],
       [ 4.1],
       [ 4.1],
       [ 4.2],
       [ 4.6],
       [ 5. ],
       [ 5.2],
       [ 5.4],
       [ 6. ],
       [ 6.1],
       [ 6.9],
       [ 7.2],
       [ 8. ],
       [ 8.3],
       [ 8.8],
       [ 9.1],
       [ 9.6],
       [ 9.7],
       [10.4],
       [10.6]])
```

In [12]:

```
X=df[["YearsExperience"]]
np.array(X).shape
```

Out[12]:

```
(30, 1)
```

In [13]:
```
X_series=df["YearsExperience"]
np.array(X_series).shape
```

Out[13]:
```
(30,)
```

In [14]:
```
X=df[["YearsExperience"]]
y=df["Salary"] # this variable can be in series or 1d array
```

In [15]:
```
y
```

Out[15]:
```
0      39344.0
1      46206.0
2      37732.0
3      43526.0
4      39892.0
5      56643.0
6      60151.0
7      54446.0
8      64446.0
9      57190.0
10     63219.0
11     55795.0
12     56958.0
13     57082.0
14     61112.0
15     67939.0
16     66030.0
17     83089.0
18     81364.0
19     93941.0
20     91739.0
21     98274.0
22    101303.0
23    113813.0
24    109432.0
25    105583.0
26    116970.0
27    112636.0
28    122392.0
29    121873.0
Name: Salary, dtype: float64
```

In [16]:
```
np.array(y).shape
```

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

In [18]:
```
# Train Test Split
from sklearn.model_selection import train_test_split
```

In [19]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=
```

In [20]:
```
X_train.shape
```

Out[20]:
```
(22, 1)
```

In [21]:
```
# Standardization
from sklearn.preprocessing import StandardScaler
```

In [23]:
```
scaler=StandardScaler()
scaler.fit_transform(X_train)
```

Out[23]:
```
array([[-0.29741739],
       [-1.38171623],
       [-0.97043047],
       [ 0.11386837],
       [-0.70870316],
       [-0.26002778],
       [-0.29741739],
       [ 1.1607776 ],
       [-1.306937  ],
       [-1.23215777],
       [ 1.57206337],
       [-1.0452097 ],
       [ 0.86166068],
       [ 1.75901144],
       [ 0.41298529],
       [ 2.13290759],
       [ 0.74949183],
       [-0.59653431],
       [-0.33480701],
       [-0.11046932],
       [ 0.45037491],
       [-0.67131355]])
```

In [24]:
```
X_test=scaler.transform(X_test)
```

In [25]:
```
X_test
```

Out[25]:
```
array([[ 1.79640106],
       [ 0.03908914],
       [ 1.27294644],
       [ 0.1886476 ],
       [-0.59653431],
       [-0.40958624],
       [ 2.05812836],
       [ 1.45989452]])
```

In [26]:
```
## Apply Simple Linear Regression
from sklearn.linear_model import LinearRegression
```

In [27]:
```
regression=LinearRegression(n_jobs=-1)
```

In [28]:
```
regression.fit(X_train,y_train)
```

Out[28]:
```
LinearRegression(n_jobs=-1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [30]:
```
regression.coef_
```

Out[30]:
```
array([9371.0160797])
```

In [31]:
```
regression.intercept_
```

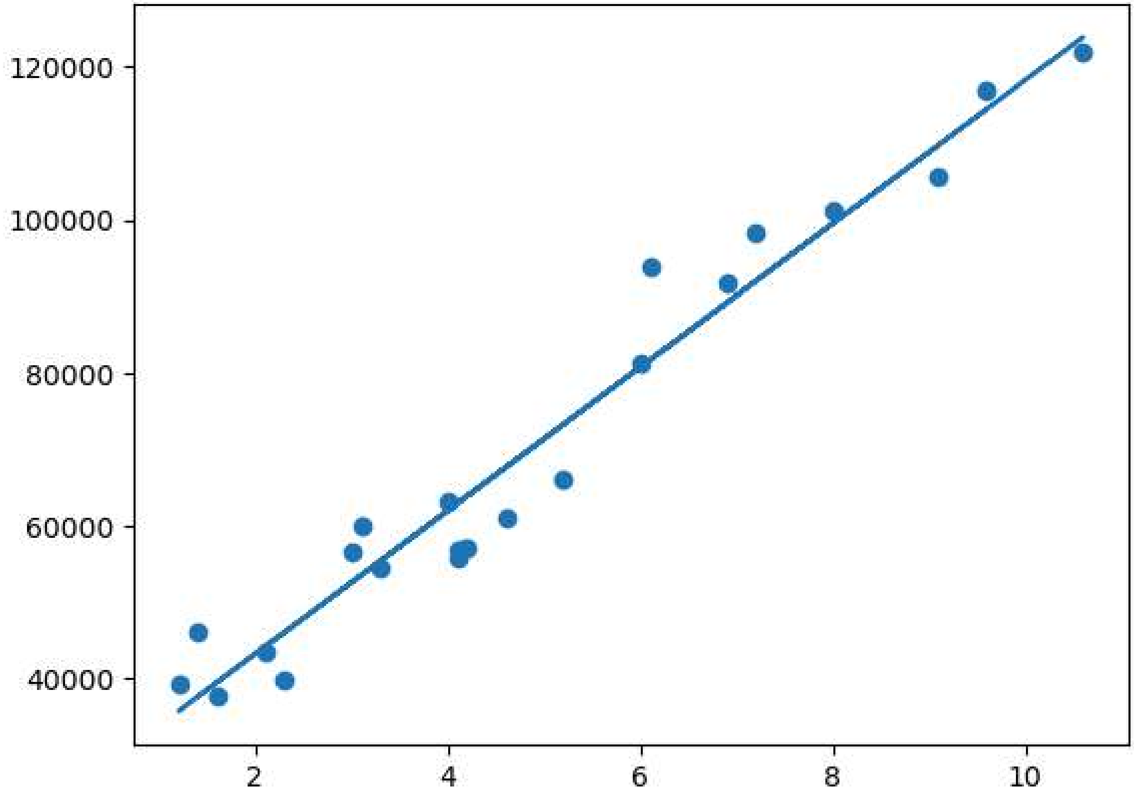
Out[31]:
```
24542.025828030746
```

In [32]:
```
print("Coefficient or slope:",regression.coef_)
print("Intercept:",regression.intercept_)
```

```
Coefficient or slope: [9371.0160797]
Intercept: 24542.025828030746
```

In [33]:
```
## plot Training data plot best fit line
plt.scatter(X_train,y_train)
plt.plot(X_train,regression.predict(X_train))
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x1ef5dd2ea50>]
```



prediction of test data

predicted height output= intercept +coef_(YearsExperience)

y_pred_test =24542.02 + 9371.01(X_test

```
In [34]: ## Predict for test data
         y_pred=regression.predict(X_test)
```

```
C:\Users\win 10\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning:
X does not have valid feature names, but LinearRegression was fitted with fea
ture names
  warnings.warn(
```

```
In [35]: ## Performance Matrics
         from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
In [36]: mse=mean_squared_error(y_test,y_pred)
         mae=mean_absolute_error(y_test,y_pred)
         rms= np.sqrt(mse)
         print(mse)
         print(mae)
         print(rms)
```

```
3847398784.0427666
60020.585847156995
62027.40349267222
```

R square

Formula

R^2 = 1 - SSR/SST

R^2 = coefficient of determination SSR = sum of squares of residuals SST = total sum of squares

```
In [37]: from sklearn.metrics import r2_score
```

In [38]:

```
score=r2_score(y_test,y_pred)
print(score)
```

```
-5.472618128428121
```

Adjusted R2 = 1 – [(1-R2)*(n-1)/(n-k-1)]

where:

R2: The R2 of the model n: The number of observations k: The number of predictor variables

In [39]:

```
#display adjusted R-squared
1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
```

Out[39]:

```
-6.551387816499474
```

In [40]:

```
## OLS Linear Regression
import statsmodels.api as sm
```

In [41]:

```
model=sm.OLS(X_train,y_train).fit()
```

In [42]:

```
prediction=model.predict(X_test)
print(prediction)
```

```
[ 1.31707258e-04  2.86591005e-06  9.33289839e-05  1.38311311e-05
 -4.37362795e-05 -3.00297532e-05  1.50896395e-04  1.07035510e-04]
```

In [43]:
```
print(model.summary())
```

```
                                 OLS Regression Results                                
=======================================================================================
==========
Dep. Variable:        YearsExperience   R-squared (uncentered):
0.970
Model:                            OLS   Adj. R-squared (uncentered):
0.969
Method:                 Least Squares   F-statistic:
681.9
Date:                Fri, 30 May 2025   Prob (F-statistic):
1.71e-17
Time:                        20:21:42   Log-Likelihood:
-30.415
No. Observations:                  22   AIC:
62.83
Df Residuals:                      21   BIC:
63.92
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=
                 coef    std err          t      P>|t|      [0.025      0.97
5]
------------------------------------------------------------------------------
-
Salary      7.332e-05   2.81e-06     26.114      0.000    6.75e-05    7.92e-0
5
==============================================================================
=
Omnibus:                        0.229   Durbin-Watson:                   1.89
6
Prob(Omnibus):                  0.892   Jarque-Bera (JB):                0.37
8
Skew:                           0.193   Prob(JB):                        0.82
8
Kurtosis:                       2.487   Cond. No.                         1.0
0
==============================================================================
=

Notes:
[1] R² is computed without centering (uncentered) since the model does not co
ntain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
```

In [45]:

```
# Prediction for new data
regression.predict(scaler.transform([[72]]))
```

```
C:\Users\win 10\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning:
X does not have valid feature names, but StandardScaler was fitted with featu
re names
  warnings.warn(
C:\Users\win 10\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning:
X does not have valid feature names, but LinearRegression was fitted with fea
ture names
  warnings.warn(
```

Out[45]:

```
array([259662.04973488])
```