SIMPLE LINEAR REGRESSION

ABOUT THE DATASET 'Salary Dataset'

Variable Notes

1. Columns
2. YearsExperience
3. Salary

DATA COLLECTION AND EXPLORATION

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

In [3]:
```
# LOAD THE DATASET
data = pd.read_csv('Salary_dataset.csv')
data
```

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
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

In [4]:

```
# INSPECT THE DATA
data.head()
```

Out[4]:

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

Out[5]:

	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

In [6]:
```
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
```

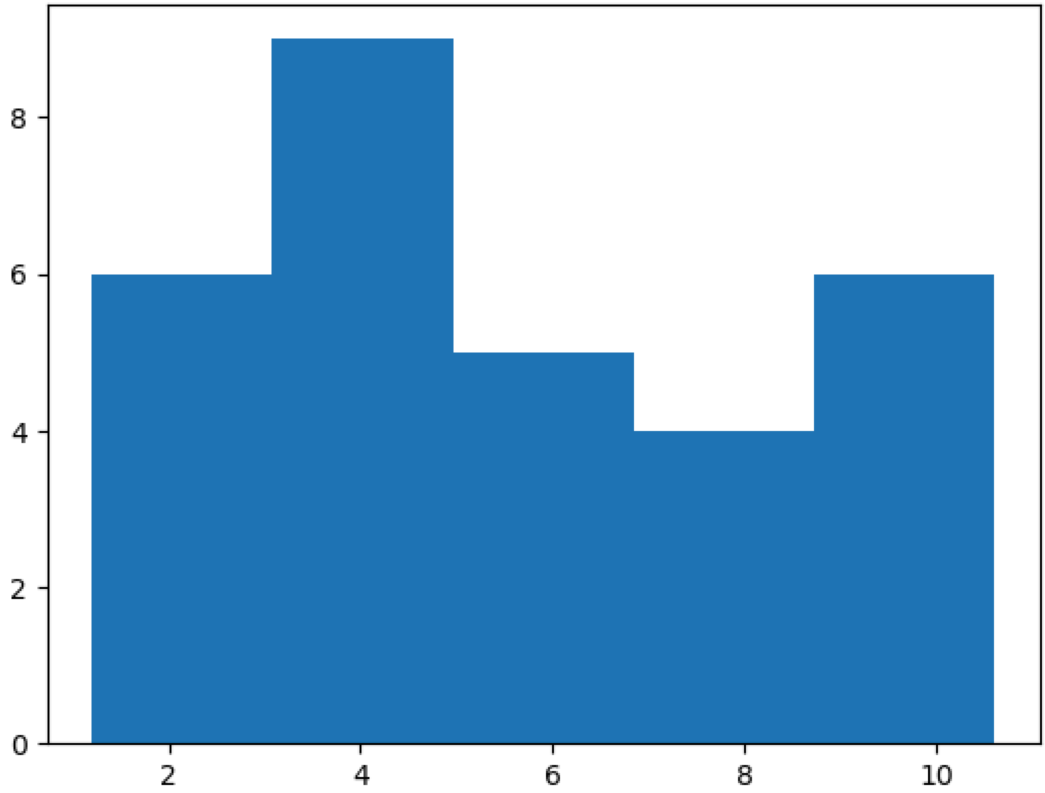
In [7]:
```
data.describe()
```

Out[7]:

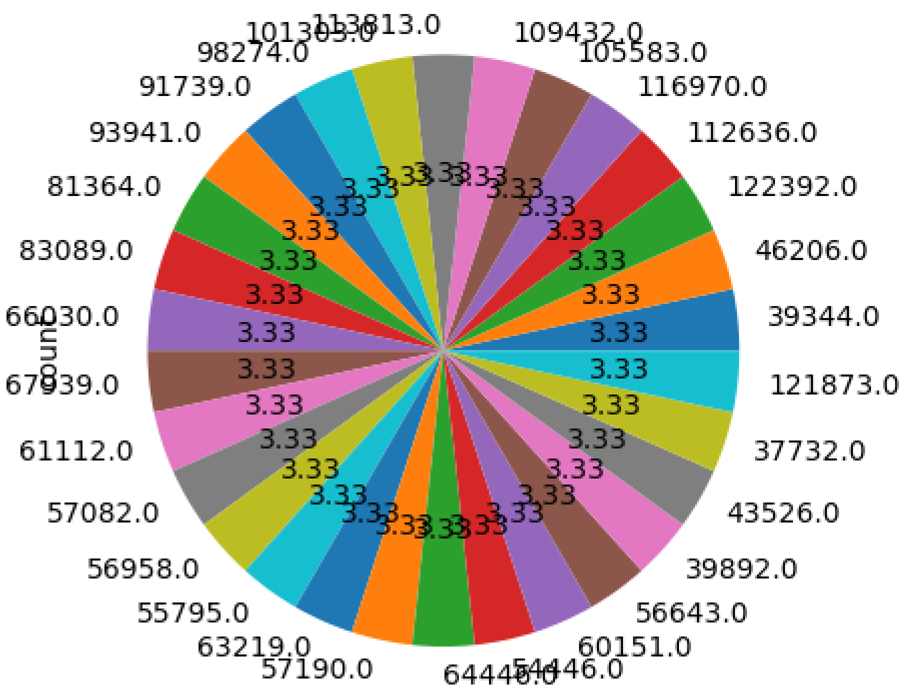
	Unnamed: 0	YearsExperience	Salary
count	30.000000	30.000000	30.000000
mean	14.500000	5.413333	76004.000000
std	8.803408	2.837888	27414.429785
min	0.000000	1.200000	37732.000000
25%	7.250000	3.300000	56721.750000
50%	14.500000	4.800000	65238.000000
75%	21.750000	7.800000	100545.750000
max	29.000000	10.600000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

In [58]:
```
plt.hist(data['YearsExperience'],bins=5)
```

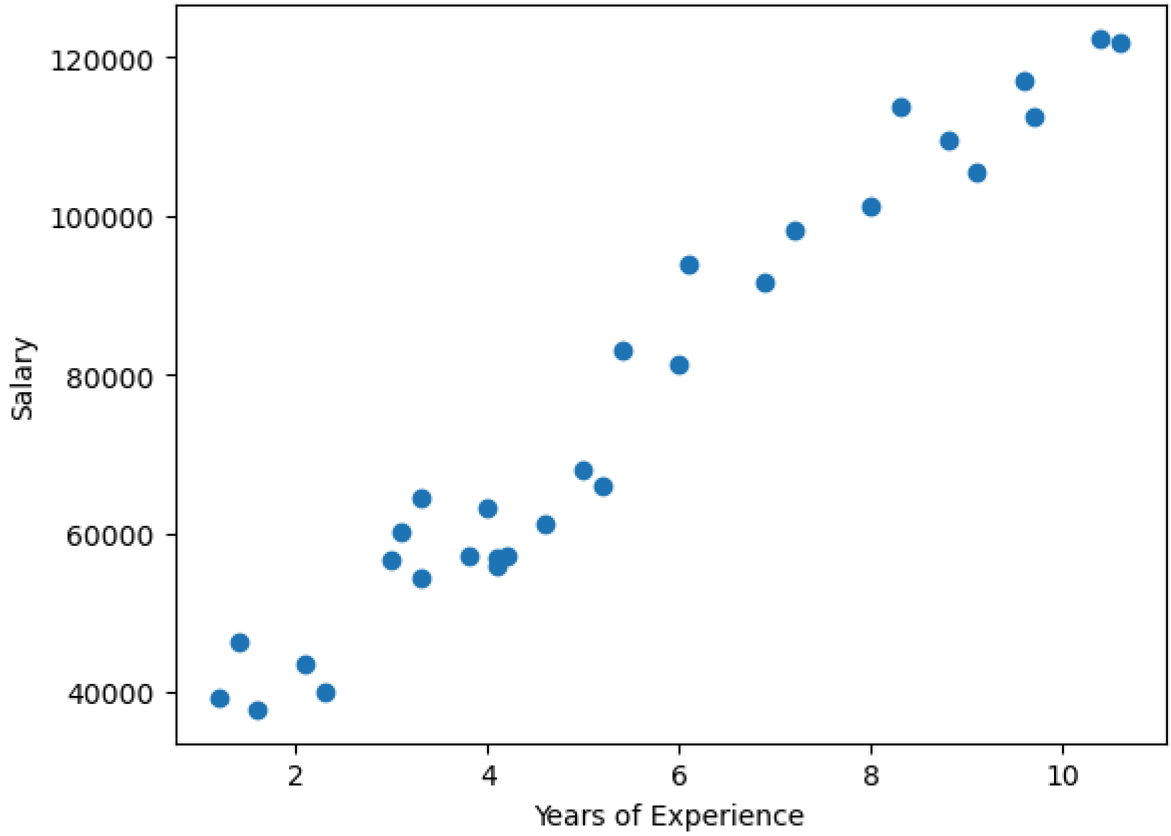
Out[58]:
```
(array([6., 9., 5., 4., 6.]),
 array([ 1.2 ,  3.08,  4.96,  6.84,  8.72, 10.6 ]),
 <BarContainer object of 5 artists>)
```



In [59]:
```python
data['Salary'].value_counts().plot(kind='pie', autopct='%.2f')
```
Out[59]:
```
<Axes: ylabel='count'>
```



In [10]:
```python
plt.scatter(data['YearsExperience'],data['Salary'])
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
```
Out[10]:
```
Text(0, 0.5, 'Salary')
```



CORRELATION

The code snippet removes the column 'Unnamed: 0' from the DataFrame 'data' and then computes the correlation matrix for the remaining columns using the data.corr() method.

In [13]:
```
data.drop(['Unnamed: 0'], axis=1, inplace=True)
data.corr()
```

Out[13]:

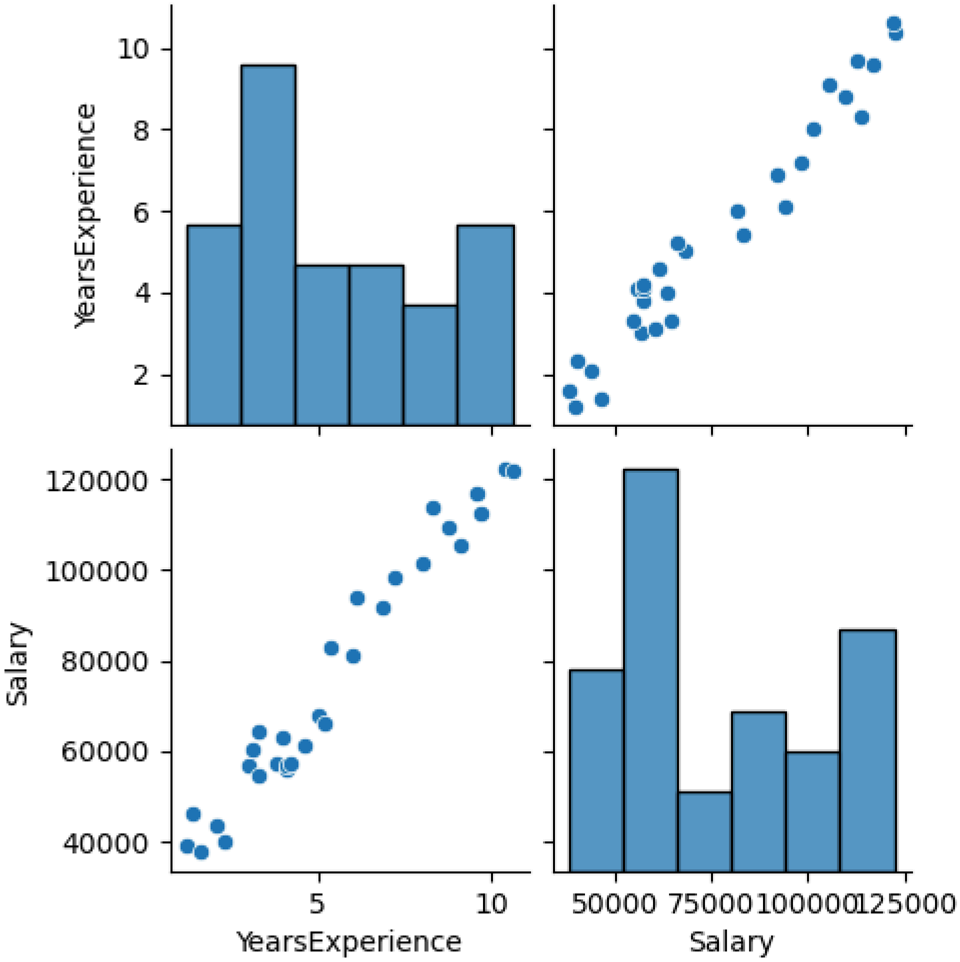
	YearsExperience	Salary
YearsExperience	1.000000	0.978242
Salary	0.978242	1.000000

SEABORN FOR VISUALIZATION

In [16]:
```
import seaborn as sns
sns.pairplot(data)
```

```
C:\Users\Nishita Bala\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWar
ning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[16]: <seaborn.axisgrid.PairGrid at 0x2485b497f90>



INDEPENDENT AND DEPENDENT FEATURES

1. Independent feature should be a datafarme or 2-D array
2. Dependent variable can be in series form or 1-D array

In [19]:
```
X = data[['YearsExperience']]
X
```

Out[19]:

	YearsExperience
0	1.2
1	1.4
2	1.6
3	2.1
4	2.3
5	3.0
6	3.1
7	3.3
8	3.3
9	3.8
10	4.0
11	4.1
12	4.1
13	4.2
14	4.6
15	5.0
16	5.2
17	5.4
18	6.0
19	6.1
20	6.9
21	7.2
22	8.0
23	8.3
24	8.8
25	9.1
26	9.6
27	9.7
28	10.4
29	10.6

In [22]:
```
np.array(X).shape
```

Out[22]:
```
(30, 1)
```

In [21]:
```
Y = data['Salary']
Y
```

Out[21]:
```
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 [23]:
```
np.array(Y).shape
```

Out[23]:
```
(30,)
```

TRAIN TEST SPLIT

The code uses train_test_split from scikit-learn to split dataset X and Y into training and testing sets. It assigns 25% to testing (test_size=0.25) and ensures reproducibility with random_state=42

In [24]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.25, random_stat
```

In [25]:
```
X_train.shape
```

Out[25]:
```
(22, 1)
```

In [26]:
```
X_test.shape
```

Out[26]:
```
(8, 1)
```

In [27]:
```
Y_train.shape
```

Out[27]:
```
(22,)
```

In [28]:
```
Y_test.shape
```

Out[28]:
```
(8,)
```

STANDARDIZATION

The code initializes a StandardScaler to normalize data. It fits and transforms X_train, scaling its features to zero mean and unit variance, and applies the same transformation to X_test.

In [30]:
```
from sklearn.preprocessing import StandardScaler
```

In [32]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

In [34]:
```
X_test = scaler.transform(X_test)
X_test
```

```
C:\Users\Nishita Bala\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarnin
g: X does not have valid feature names, but StandardScaler was fitted with feature
names
  warnings.warn(
```

Out[34]:
```
array([[-1.15872417],
       [-1.81577634],
       [-1.35444184],
       [-1.759857  ],
       [-2.0534335 ],
       [-1.98353433],
       [-1.06086534],
       [-1.28454267]])
```

APPLY LINEAR REGRESSION

The code imports the LinearRegression model from sklearn.linear_model, initializes it with n_jobs=-1 for parallel processing, and fits the model to training data X_train and Y_train.

In [35]:
```
from sklearn.linear_model import LinearRegression
```

In [38]:
```
regression = LinearRegression(n_jobs = -1)
```

In [39]:
```
regression.fit(X_train,Y_train)
```

Out[39]:
```
▾      LinearRegression
LinearRegression(n_jobs=-1)
```

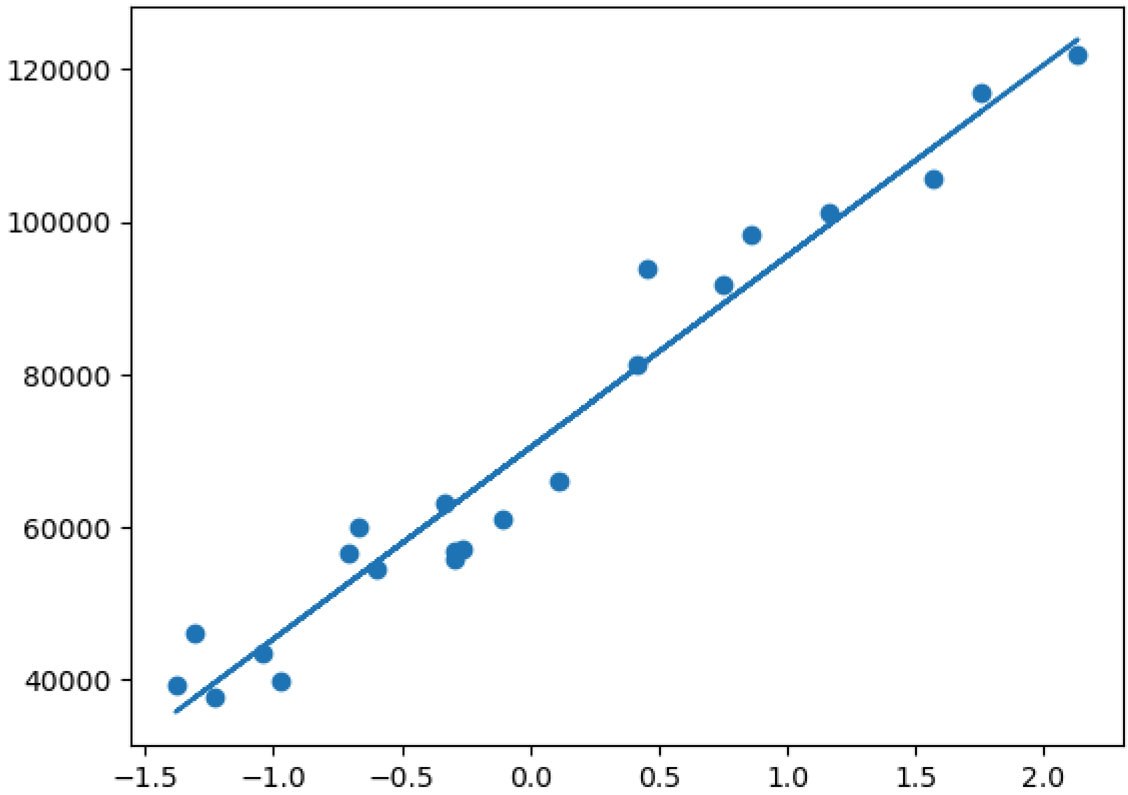
In [42]:
```
print("Coefficient or slope (Beta1):",regression.coef_)
print("Intercept (Beta0):",regression.intercept_)
```

```
Coefficient or slope (Beta1): [25063.1519945]
Intercept (Beta0): 70417.40909090909
```

PLOT TRAINING DATA PLOT BEST FIT LINE

In [44]:
```
plt.scatter(X_train,Y_train)
plt.plot(X_train,regression.predict(X_train))
```

Out[44]:
```
[<matplotlib.lines.Line2D at 0x2485d2d3590>]
```



PREDICTION FOR TEST DATA

In [71]:
```
Y_pred = regression.predict(X_test)
y_pred
```

Out[71]:
```
array([41376.1290096 , 24908.33081684, 36470.82742027, 26309.84555665,
       18951.89317266, 20703.78659742, 43828.77980427, 38222.72084503])
```

PERFORMANCE MATRICS

In [69]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [70]:
```
mse = mean_squared_error(Y_test,Y_pred)
mae = mean_absolute_error(Y_test,Y_pred)
rmse = np.sqrt(mse)

print("Mean Square Error",mse)
print("Mean Absolute",mae)
print("Root Mean Square Error",rmse)
```

```
Mean Square Error 3847398784.0427675
Mean Absolute 60020.58584715701
Root Mean Square Error 62027.40349267223
```