In [1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```python
path=r"C:\Users\kasho\OneDrive\Documents\Data Science\flask\Salary_Data - Sa
df=pd.read_csv(path)
df.head()
```

Out[2]:

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

In [3]:
```python
df.shape
```

Out[3]:
```
(30, 2)
```

In [4]:
```python
df.columns
```

Out[4]:
```
Index(['YearsExperience', 'Salary'], dtype='object')
```

In [5]:
```python
df.isnull().sum()
```

Out[5]:
```
YearsExperience    0
Salary             0
dtype: int64
```

In [6]:
```python
df.dtypes
```

Out[6]:
```
YearsExperience    float64
Salary               int64
dtype: object
```

In [7]:
```python
X=df.drop('YearsExperience',axis=1)
y=df['Salary']
```

In [8]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,  # Input data
                                                  y,  # output data
                                                  random_state=1234, # it s
                                                  test_size=0.30)
```

In [9]:
```
X_train.shape,X_test.shape
```

Out[9]:
```
((21, 1), (9, 1))
```

In [10]:
```
y_train.shape,y_test.shape
```

Out[10]:
```
((21,), (9,))
```

In [11]:
```
df.shape
```

Out[11]:
```
(30, 2)
```

In [12]:
```
X_train
```

Out[12]:

	Salary
13	57081
22	101302
24	109431
0	39343
2	37731
27	112635
26	116969
18	81363
5	56642
16	66029
25	105582
11	55794
9	57189
17	83088
29	121872
20	91738
12	56957
21	98273
6	60150
19	93940
15	67938

In [13]:

```
y_train
```

Out[13]:

```
13     57081
22    101302
24    109431
0      39343
2      37731
27    112635
26    116969
18     81363
5      56642
16     66029
25    105582
11     55794
9      57189
17     83088
29    121872
20     91738
12     56957
21     98273
6      60150
19     93940
15     67938
Name: Salary, dtype: int64
```

In [14]:

```
X_test
```

Out[14]:

	Salary
7	54445
10	63218
4	39891
1	46205
28	122391
8	64445
3	43525
23	113812
14	61111

In [15]:

```
y_test
```

Out[15]:

```
7      54445
10     63218
4      39891
1      46205
28    122391
8      64445
3      43525
23    113812
14     61111
Name: Salary, dtype: int64
```

In [16]:
```
X_train.ndim
# 1 dimension means 1 column only
# 2 dimension means 2 column only
# when you have only 1 coulmn, the shape will not show the coulumn
# (21,) it is only one column data having 21 observations
# (9,) it is one column data having 9 observation
# (30,2) it is 2 column data having 30 observation
# Reshape the data if you have only one column
```

Out[16]:
```
2
```

In [17]:
```
from sklearn.linear_model import LinearRegression
LR=LinearRegression()
LR.fit(X_train,y_train)
```

Out[17]:
```
▾ LinearRegression
LinearRegression()
```

In [18]:
```
# Model predictions happens X_test
y_predictions=LR.predict(X_test)
```

In [19]:
```
y_predictions
```

Out[19]:
```
array([ 54445.,  63218.,  39891.,  46205., 122391.,  64445.,  43525.,
       113812.,  61111.])
```

In [20]:
```
y_test.shape,y_predictions.shape
```

Out[20]:
```
((9,), (9,))
```

In [21]:
```
X_test
```

Out[21]:

	Salary
7	54445
10	63218
4	39891
1	46205
28	122391
8	64445
3	43525
23	113812
14	61111

In [22]:

```
X_test.iloc[0]  # series
# In order to pass a test sample to a model
# we need to pass a list of values
# or array of values
# tuple of values
X_test.iloc[0].values
```

Out[22]:

```
array([54445], dtype=int64)
```

In [23]:

```
LR.predict([X_test.iloc[0].values,
            X_test.iloc[1].values])
```

Out[23]:

```
array([54445., 63218.])
```

In [24]:

```
ip1=[5]
LR.predict([ip1])
```

Out[24]:

```
array([5.])
```

In [25]:

```
X_test.shape,y_test.shape,y_predictions.shape
```

Out[25]:

```
((9, 1), (9,), (9,))
```

In [26]:

```
test_data=X_test
test_data['y_actual']=y_test
test_data['y_predictions']=y_predictions
test_data
```

Out[26]:

	Salary	y_actual	y_predictions
7	54445	54445	54445.0
10	63218	63218	63218.0
4	39891	39891	39891.0
1	46205	46205	46205.0
28	122391	122391	122391.0
8	64445	64445	64445.0
3	43525	43525	43525.0
23	113812	113812	113812.0
14	61111	61111	61111.0

In [27]:

```
# y_test is series
# y_predictions is numpy array values
print(y_test.values[:5])  # float 5. means 5.0
print(y_predictions[:5])
```

```
[ 54445  63218  39891  46205 122391]
[ 54445.  63218.  39891.  46205. 122391.]
```

In [28]:

```
# RMSE
# MSE
# MAE
# R-square

from sklearn.metrics import r2_score,mean_squared_error
```

In [29]:

```
R2=r2_score(y_test,y_predictions)
MSE=mean_squared_error(y_test,y_predictions)
#MSE**(1/2)
RMSE=np.sqrt(MSE)
#accuracy_score(y_test,y_predictions) # it is a regression tech
print("R-sqaure:",R2)
print("MSE:",MSE)
print("RMSE:",RMSE)
```

```
R-sqaure: 1.0
MSE: 6.470390569303684e-23
RMSE: 8.043873798925294e-12
```

In [30]:

```
s=0
for i in range(len(y_test)):
    v1=y_test.values[i]-y_predictions[i]
    v2=v1**2
    s=s+v2
print(s/len(y_test))
```

```
6.470390569303684e-23
```

In [31]:

```
LR.coef_
print("The coeffiecnt of Years_of_experience is:",LR.coef_)
```

```
The coeffiecnt of Years_of_experience is: [1.]
```

In [32]:

```
LR.intercept_
```

Out[32]:

```
-1.4551915228366852e-11
```

In [33]:

```
X_train.columns
```

Out[33]:

```
Index(['Salary'], dtype='object')
```

In [34]:

```
#Regression_equation=LR.intercept_+LR.coef_ * col namee
#Regression_equation

y=-1.45+1-.*Salary
```

```
  Cell In[34], line 4
    y=-1.45+1-.*Salary
              ^
SyntaxError: invalid syntax
```

In [35]:

```
from sklearn.feature_selection import VarianceThreshold
vt=VarianceThreshold(threshold=0)
# Threshold variance value
# we want to drop the feaure based on threshold
vt.fit(df)
```

Out[35]:

```
▾       VarianceThreshold

VarianceThreshold(threshold=0)
```

In [36]:

```
dir(vt)
```

```
Out[36]: ['__abstractmethods__',
          '__annotations__',
          '__class__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getstate__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__le__',
          '__lt__',
          '__module__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__setstate__',
          '__sizeof__',
          '__sklearn_clone__',
          '__str__',
          '__subclasshook__',
          '__weakref__',
          '_abc_impl',
          '_build_request_for_signature',
          '_check_feature_names',
          '_check_n_features',
          '_get_default_requests',
          '_get_metadata_request',
          '_get_param_names',
          '_get_support_mask',
          '_get_tags',
          '_more_tags',
          '_parameter_constraints',
          '_repr_html_',
          '_repr_html_inner',
          '_repr_mimebundle_',
          '_sklearn_auto_wrap_output_keys',
          '_transform',
          '_validate_data',
          '_validate_params',
          'feature_names_in_',
          'fit',
          'fit_transform',
          'get_feature_names_out',
          'get_metadata_routing',
          'get_params',
          'get_support',
          'inverse_transform',
          'n_features_in_',
          'set_output',
          'set_params',
          'threshold',
          'transform',
          'variances_']
```

In [37]:
```
vt.variances_
# 300 is first column variance (T)
# 1.25 is second column variance (T)
# 30 is column varaince (T)
# 0  is fourth column variance (F)
```

Out[37]:
```
array([7.78515556e+00, 8.46600000e+04])
```

In [38]:
```
vt.get_support()
```

Out[38]:
```
array([ True,  True])
```

In [39]:
```
vt.get_params()
# Hyper parameter
# that we are providing inside the function
```

Out[39]:
```
{'threshold': 0}
```

In [40]:
```
vt.threshold
```

Out[40]:
```
0
```

In [41]:

```
cols=vt.get_feature_names_out()
# the above syntax gives the column names
# These fetaure only we want include
df[cols]
```

Out[41]:

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150
7	3.2	54445
8	3.2	64445
9	3.7	57189
10	3.9	63218
11	4.0	55794
12	4.0	56957
13	4.1	57081
14	4.5	61111
15	4.9	67938
16	5.1	66029
17	5.3	83088
18	5.9	81363
19	6.0	93940
20	6.8	91738
21	7.1	98273
22	7.9	101302
23	8.2	113812
24	8.7	109431
25	9.0	105582
26	9.5	116969
27	9.6	112635
28	10.3	122391
29	10.5	121872

In [42]:

```
path=r"C:\Users\kasho\OneDrive\Documents\Data Science\flask\Salary_Data - Sa
df=pd.read_csv(path)
df.head()
from sklearn.feature_selection import VarianceThreshold
vt=VarianceThreshold(threshold=0)
### Make sure before fitting the dataframe , do not include output column
X=df.drop('YearsExperience',axis=1)
# X it self a data frame
vt.fit(X)
vt.variances_
vt.get_support()
cols=vt.get_feature_names_out()
X[cols]
```

Out[42]:

	Salary
0	39343
1	46205
2	37731
3	43525
4	39891
5	56642
6	60150
7	54445
8	64445
9	57189
10	63218
11	55794
12	56957
13	57081
14	61111
15	67938
16	66029
17	83088
18	81363
19	93940
20	91738
21	98273
22	101302
23	113812
24	109431
25	105582
26	116969
27	112635
28	122391
29	121872

In [43]:

```
from statsmodels.api import OLS
OLS(y_train,X_train).fit().summary()
```

Out[43]:

```
OLS Regression Results

      Dep. Variable:          Salary      R-squared (uncentered):      1.000
              Model:             OLS  Adj. R-squared (uncentered):     1.000
             Method:   Least Squares                 F-statistic:  3.694e+32
               Date: Tue, 16 Apr 2024          Prob (F-statistic):  3.81e-314
               Time:        17:51:36              Log-Likelihood:     488.13
  No. Observations:              21                         AIC:      -974.3
       Df Residuals:             20                          BIC:      -973.2
           Df Model:              1
   Covariance Type:       nonrobust

             coef   std err         t    P>|t|   [0.025   0.975]
Salary     1.0000   5.2e-17  1.92e+16    0.000    1.000    1.000

        Omnibus:  3.403     Durbin-Watson:  0.280
 Prob(Omnibus):   0.182  Jarque-Bera (JB):  2.287
           Skew:  0.627          Prob(JB):  0.319
       Kurtosis:  1.979          Cond. No.   1.00


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

In [46]:

```
import pickle
pickle.dump(LR,
           open('YearsExperience_model.pkl','wb'))
```

In [47]:

```
# loading model to comapare the result
model=pickle.load(open('YearsExperience_model.pkl','rb'))
model
```

Out[47]:

```
▾ LinearRegression

LinearRegression()
```

In [ ]:

In [ ]: