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

```
df = pd.read_csv("/content/Salary_dataset.csv")
```

```
df.head()
```

	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

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

```
df.drop("Unnamed: 0", axis=1, inplace=True)
```

```
df.head()
```

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

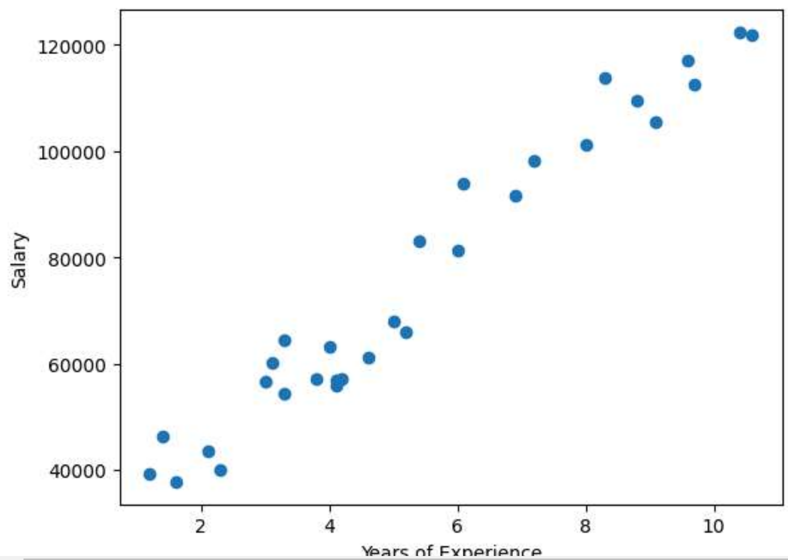
Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

```
## Scatter Plot
plt.scatter(df["YearsExperience"], df["Salary"])
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
```

```
Text(0, 0.5, 'Salary')
```

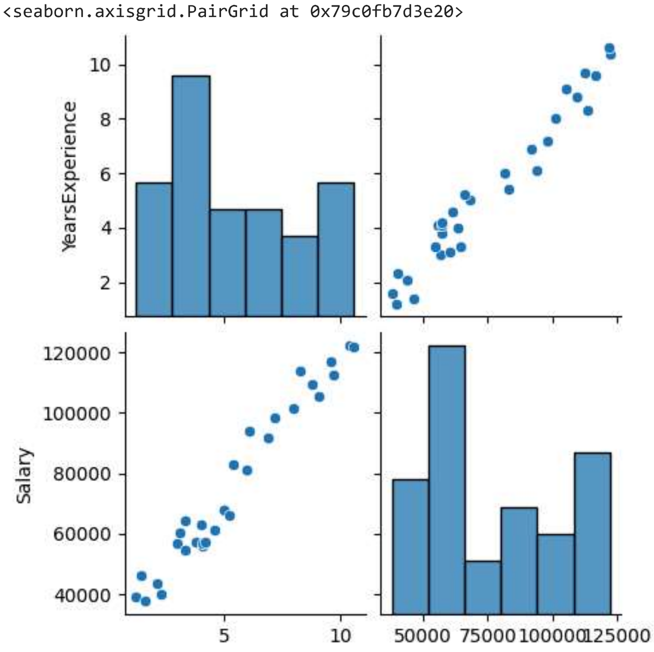



```
## Correlation
df.corr()
```

	YearsExperience	Salary
YearsExperience	1.000000	0.978242
Salary	0.978242	1.000000

```
## Seaborn for visualization
import seaborn as sns
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x79c0fb7d3e20>
```




```
## Independent and Depenedent features
X = df[['YearsExperience']] ### Independent features should be data frame or 2 dimension not series
Y = df['Salary'] ### Can be a series
np.array(X).shape
```

```
(30, 1)
```

```
X.head()
```

	YearsExperience
0	1.2
1	1.4
2	1.6
3	2.1
4	2.3

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

```
Y.head()
```

	Salary
0	39344.0
1	46206.0
2	37732.0
3	43526.0
4	39892.0

```
np.array(Y).shape
```

```
(30,)
```

```
### Train Test Split
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_state=0)
```

```
X_train.head()
```

	YearsExperience
17	5.4
22	8.0
5	3.0
16	5.2
8	3.3

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

```
Y_train.head()
```

	Salary
17	83089.0
22	101303.0
5	56643.0
16	66030.0
8	64446.0

Generate | create a dataframe with 2 columns and 10 rows | Close

```
## Standardization
# y -> rupees As the units are different, in Gradient Descent values to come to global minima will take long time
# x -> years So we use Z Score, it will convert all values with mean =0 and SD =1
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train) # (fit_transform ) calculates with mean and std dev from train data
X_test = sc.transform(X_test)
```

```
X_train
```

```
array([[ 1.59814143e-01],
       [ 1.19860607e+00],
       [-7.99070713e-01],
       [ 7.99070713e-02],
       [-6.79210106e-01],
       [-1.59814143e-01],
       [ 1.31846668e+00],
       [ 7.59117177e-01],
       [-1.43832728e+00],
       [ 2.23739800e+00],
       [-7.59117177e-01],
       [-1.07874546e+00],
       [ 3.99535356e-01],
       [ 4.39488892e-01],
       [-4.79442428e-01],
       [-6.79210106e-01],
       [ 1.63809496e+00],
       [-1.15865253e+00],
       [-1.51823435e+00],
       [ 8.78977784e-01],
       [ 3.69311789e-17],
       [-3.59581821e-01]])
```

```
# The same mean of Std Dev and Mean we use in test
# And the mean and Std Dev of train is used in test (transform)
X_test
```

```
array([[-1.35842021],
       [ 2.15749092],
       [-0.31962829],
       [-0.39953536],
       [ 1.83786264],
       [ 1.51823435],
       [ 1.87781617],
       [-0.35958182]])
```

```
## Apply Linear Regression
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
regression.fit(X_train, Y_train)

## Here in the fit all the train must be in a 2d array
```

```
▾ LinearRegression
LinearRegression()
```

```
regression.coef_ #Slope
```

```
array([23476.54680309])
```

```
# y = B0 + B1 * x1
# B0 = Intercept
# B1 = Slope
regression.intercept_
```

```
72948.27272727272
```

One unit movement in X axis it leads to 23476 movements in Y axis

When X =0 Y is 72948

```
## Plot training data plot best fit line
plt.scatter(X_train, Y_train, color="red")
plt.plot(X_train, regression.predict(X_train), color="blue")
plt.title("Salary vs Experience (Training Set)")
```

```
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
```

```
Text(0, 0.5, 'Salary')
```


```
## Prediction for test data
Y_pred = regression.predict(X_test)
```

y_pred_test = 72948.27 + 23476.54680309(X_test)

```
## Performance Metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error
MAE = mean_absolute_error(Y_test, Y_pred)
MSE = mean_squared_error(Y_test, Y_pred)
RMSE = np.sqrt(MSE)
```

```
print(MAE)
print(MSE)
print(RMSE)
```

```
3508.5455930660537
22407940.14334066
4733.702582898577
```

```
# R^2 = 1 - SSR / SST
from sklearn.metrics import r2_score
r2_score = r2_score(Y_test, Y_pred)
```

```
r2_score
```

```
0.9779208335417602
```

```
# Plot test
plt.scatter(X_test, Y_test, color="red")
plt.plot(X_test, regression.predict(X_test), color="blue")
plt.title("Salary vs Experience (Training Set)")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
```

```
Text(0, 0.5, 'Salary')
```


```
# Adjusted R2 = 1 - [(1 - R2)* (n-1)/ (n-k-1)]
# n = no of observations
# k = no of independent variables
1 - (1 - r2_score) * (len(Y_test) - 1) / (len(Y_test) - X_test.shape[1] - 1)
```

```
0.9742409724653869
```

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

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

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

```
prediction
```

```
array([-31891.01567262,  50650.43665651,  -7503.76839356,  -9379.71049195,
        43146.66826295,  35642.89986939,  44084.63931215,  -8441.73944275])
```

```
print(model.summary())
```

```
                                 OLS Regression Results                                
=======================================================================================
Dep. Variable:                 Salary   R-squared (uncentered):                   0.093
Model:                            OLS   Adj. R-squared (uncentered):              0.050
Method:                 Least Squares   F-statistic:                              2.161
Date:                Thu, 19 Sep 2024   Prob (F-statistic):                       0.156
Time:                        13:44:56   Log-Likelihood:                         -277.63
No. Observations:                  22   AIC:                                      557.3
Df Residuals:                      21   BIC:                                      558.4
Df Model:                           1                                                  
Covariance Type:            nonrobust                                                  
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1          2.348e+04    1.6e+04      1.470      0.156   -9738.165    5.67e+04
==============================================================================
Omnibus:                        3.210   Durbin-Watson:                   0.017
Prob(Omnibus):                  0.201   Jarque-Bera (JB):                1.397
Skew:                           0.188   Prob(JB):                        0.497
Kurtosis:                       1.824   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.
```

```
# Step 1: Make the prediction (which is still in the scaled form)
```

```
scaled_prediction = regression.predict(sc.transform([[10]]))

# Step 2: Inverse transform the scaled prediction to get the original value
original_prediction = sc.inverse_transform(scaled_prediction.reshape(-1, 1))

print(original_prediction)
```