In [18]:

```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
import statsmodels.formula.api as smf
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
from scipy.stats import probplot
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

1.Importing Data

In [19]:

```
salary_data = pd.read_csv('Salary_Data.csv')
salary_data
```

14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0

In [20]:

```
salary_data.head()
```

Out[20]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

In [21]:

```
salary_data.dtypes
```

Out[21]:

```
YearsExperience    float64
Salary             float64
dtype: object
```

In [22]:

```
salary_data.describe()
```

Out[22]:

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

2.EDA and Data Visualization

In [23]:

```
salary_data.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
```

In [24]:

```
len(salary_data .columns) # identify the number of features
```

Out[24]:

```
2
```

In [25]:

```
salary_data.columns # idenfity the features
```

Out[25]:

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

In [26]:

```
salary_data.shape # identify the size of of the dataset
```

Out[26]:

```
(30, 2)
```

In [27]:

```
salary_data.dtypes # identify the datatypes of the features
```

Out[27]:

```
YearsExperience    float64
Salary             float64
dtype: object
```

In [28]:

```
salary_data.isnull().values.any() # checking if dataset has empty cells
```

Out[28]:

```
False
```

In [29]:

```
salary_data.isnull().sum() # identify the number of empty cells
```

Out[29]:

```
YearsExperience    0
Salary             0
dtype: int64
```

3.Graphical Univariate analysis

Histogram

In [30]:

```
# We can use either plt.hist or sns.histplot
plt.figure(figsize=(20,10))
plt.subplot(2,4,1)
plt.hist(salary_data['YearsExperience'], density=False)
plt.title("Histogram of 'YearsExperience'")
plt.subplot(2,4,5)
plt.hist(salary_data['Salary'], density=False)
plt.title("Histogram of 'Salary'")
```

Out[30]:

```
Text(0.5, 1.0, "Histogram of 'Salary'")
```

In [ ]:

```

```

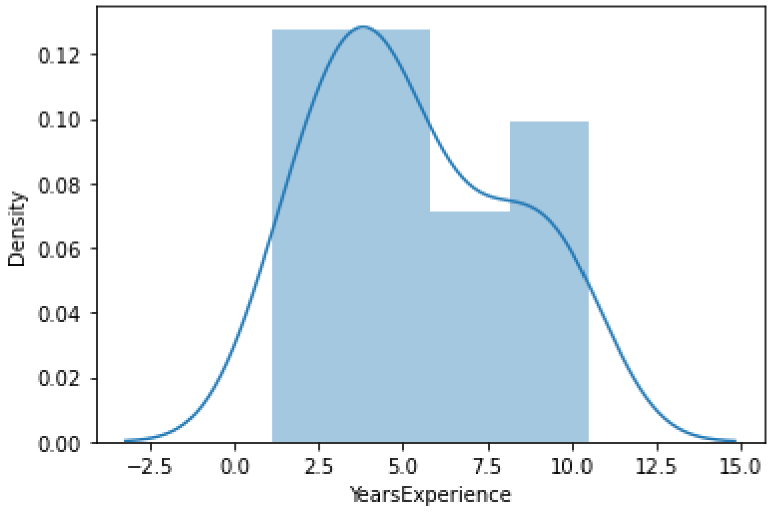
Density plot

In [31]:

```
sns.distplot(salary_data['YearsExperience'])
```

Out[31]:

```
<AxesSubplot:xlabel='YearsExperience', ylabel='Density'>
```

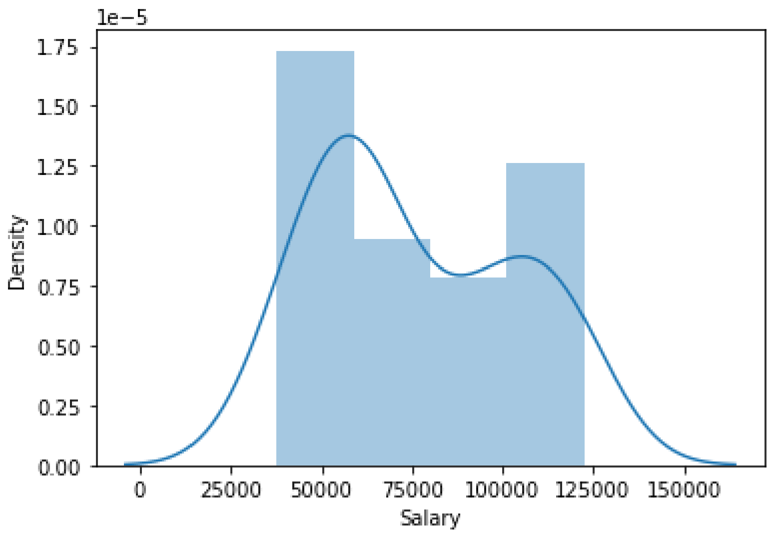


In [32]:

```
sns.distplot(salary_data['Salary'])
```

Out[32]:

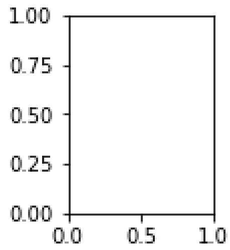
```
<AxesSubplot:xlabel='Salary', ylabel='Density'>
```

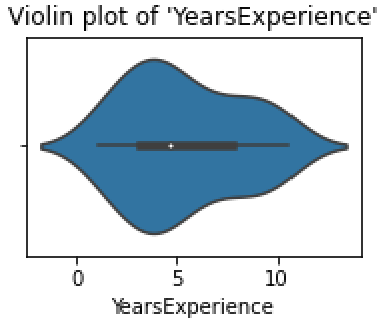


boxplot or violin plot

In [45]:

```
# A violin plot is a method of plotting numeric data. It is similar to a box plot,
# with the addition of a rotated kernel density plot on each side
plt.subplot(2,4,3)
plt.figure(figsize=(3,2))
# plt.boxplot(salary_data['YearsExperience'])
sns.violinplot(salary_data['YearsExperience'])
# plt.title("Boxlpot of 'YearsExperience'")
plt.title("Violin plot of 'YearsExperience'")
plt.show()
```



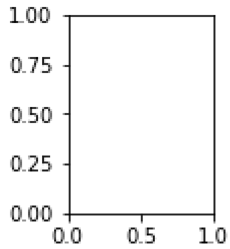


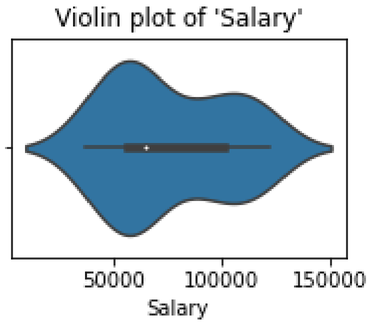
In [46]:

```
plt.subplot(2,4,7)
plt.figure(figsize=(3,2))
# plt.boxplot(salary_data['Salary'])
sns.violinplot(salary_data['Salary'])
# plt.title("Boxlpot of 'Salary'")
plt.title("Violin plot of 'Salary'")
```

Out[46]:

```
Text(0.5, 1.0, "Violin plot of 'Salary'")
```





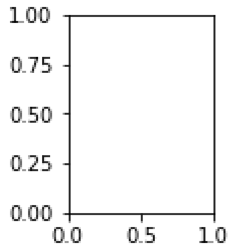
Normal Q-Q plot

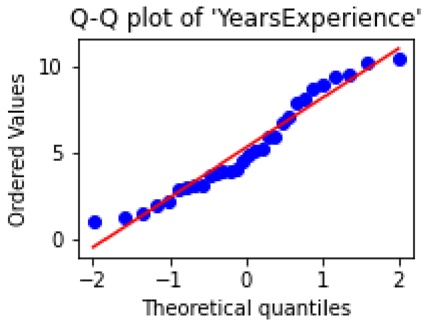
In [51]:

```
plt.subplot(2,4,4)
plt.figure(figsize=(3,2))
probplot(salary_data['YearsExperience'], plot=plt)
plt.title("Q-Q plot of 'YearsExperience'")
```

Out[51]:

```
Text(0.5, 1.0, "Q-Q plot of 'YearsExperience'")
```



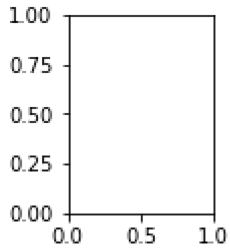


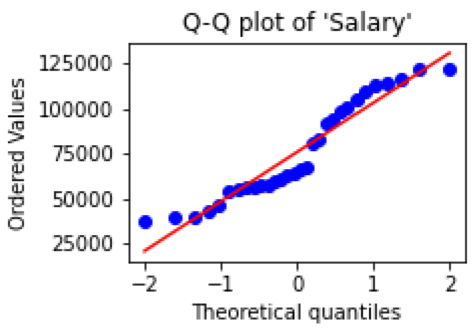
In [52]:

```
plt.subplot(2,4,8)
plt.figure(figsize=(3,2))
probplot(salary_data['Salary'], plot=plt)
plt.title("Q-Q plot of 'Salary'")
```

Out[52]:

```
Text(0.5, 1.0, "Q-Q plot of 'Salary'")
```





From the above graphical representations, we can say there are no outliers in our data, and YearsExperience looks like normally distributed, and Salary doesn't look normal. We can verify this using Shapiro Test.

In [83]:

```
# Def a function to run Shapiro test

# Defining our Null, Alternate Hypothesis
Ho = 'Data is Normal'
Ha = 'Data is not Normal'

# Defining a significance value
alpha = 0.05
def normality_check(salary_data):
    for columnName, columnData in salary_data.iteritems():
        print("Shapiro test for {columnName}".format(columnName=columnName))
        res = stats.shapiro(columnData)
#         print(res)
        pValue = round(res[1], 2)

        # Writing condition
        if pValue > alpha:
            print("pvalue = {pValue} > {alpha}. We fail to reject Null Hypothesis. {Ho}".fo
        else:
            print("pvalue = {pValue} <= {alpha}. We reject Null Hypothesis. {Ha}".format(pV


# Drive code
normality_check(salary_data)
```

```
Shapiro test for YearsExperience
pvalue = 0.1 > 0.05. We fail to reject Null Hypothesis. Data is Normal
Shapiro test for Salary
pvalue = 0.02 <= 0.05. We reject Null Hypothesis. Data is not Normal
Shapiro test for Norm_YearsExp
pvalue = 0.1 > 0.05. We fail to reject Null Hypothesis. Data is Normal
Shapiro test for Norm_Salary
pvalue = 0.02 <= 0.05. We reject Null Hypothesis. Data is not Normal
```

Our instinct from the graphs was correct. YearsExperience is normally distributed, and Salary isn’t normally distributed.

3. Bivariate visualization

for Numerical vs. Numerical data, we can plot the below graphs

1.Scatterplot

2.Line plot

3.Heatmap for correlation

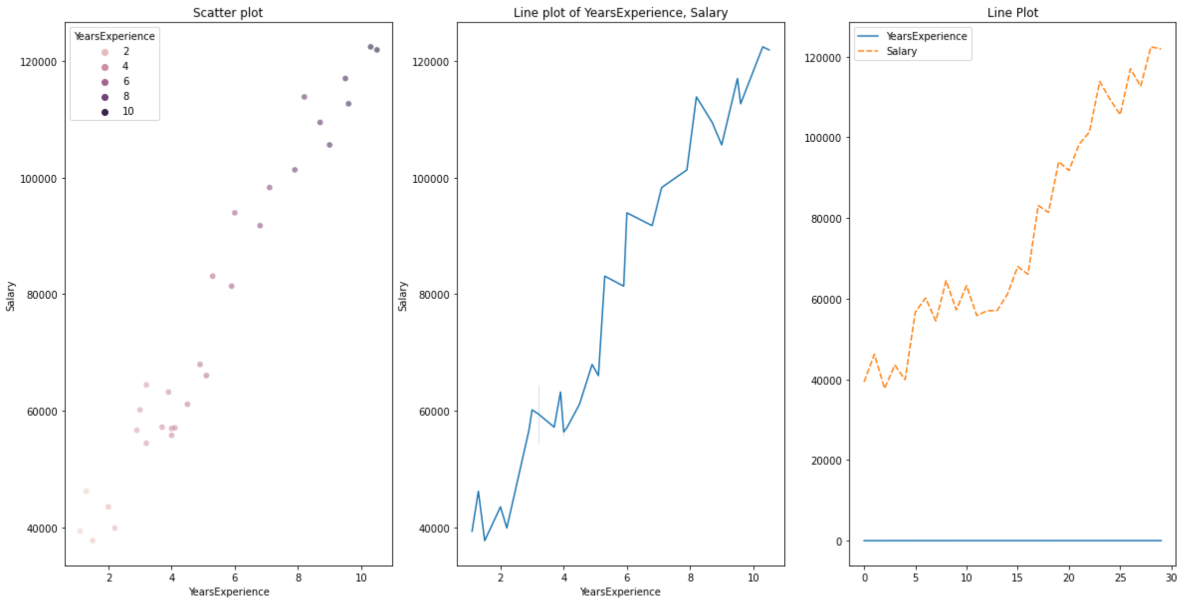
4.Joint plot

In [56]:

```
# 1. Scatterplot & Line plots
plt.figure(figsize=(20,10))
plt.subplot(1,3,1)
sns.scatterplot(data=salary_data, x="YearsExperience", y="Salary", hue="YearsExperience", a
plt.title("Scatter plot")
plt.subplot(1,3,2)
sns.lineplot(data=salary_data, x="YearsExperience", y="Salary")
plt.title("Line plot of YearsExperience, Salary")
plt.subplot(1,3,3)
sns.lineplot(data=salary_data)
plt.title('Line Plot')
```

Out[56]:

```
Text(0.5, 1.0, 'Line Plot')
```

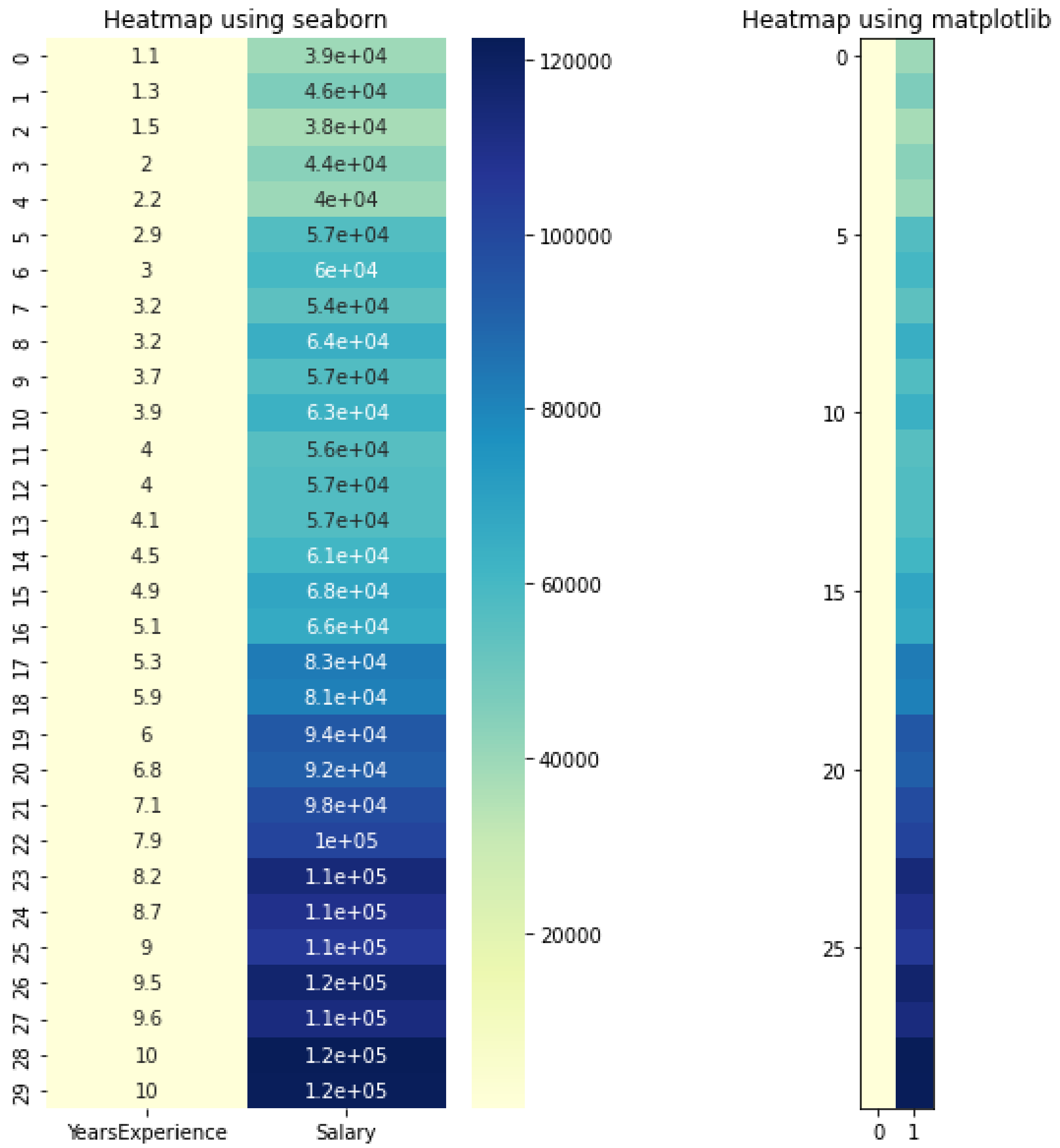


In [58]:

```
# heatmap
plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
sns.heatmap(data=salary_data, cmap="YlGnBu", annot = True)
plt.title("Heatmap using seaborn")
plt.subplot(1, 2, 2)
plt.imshow(salary_data, cmap ="YlGnBu")
plt.title("Heatmap using matplotlib")
```

Out[58]:

```
Text(0.5, 1.0, 'Heatmap using matplotlib')
```

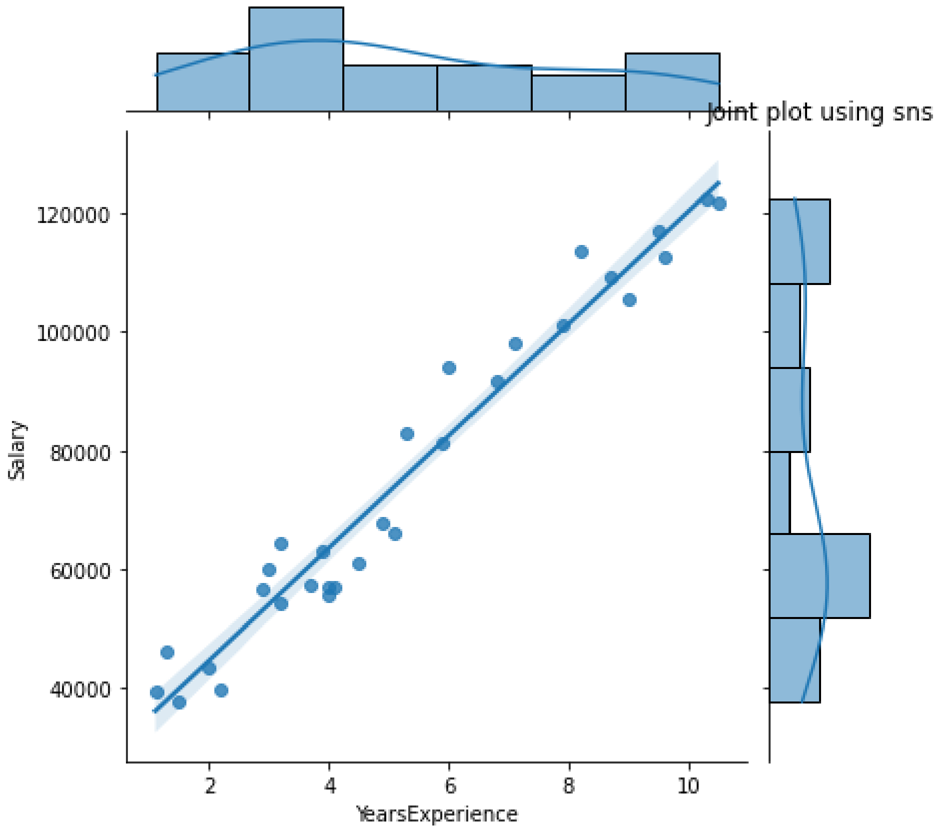


In [59]:

```
# Joint plot
sns.jointplot(x = "YearsExperience", y = "Salary", kind = "reg", data = salary_data)
plt.title("Joint plot using sns")
# kind can be hex, kde, scatter, reg, hist. When kind='reg' it shows the best fit line.
```

Out[59]:

```
Text(0.5, 1.0, 'Joint plot using sns')
```



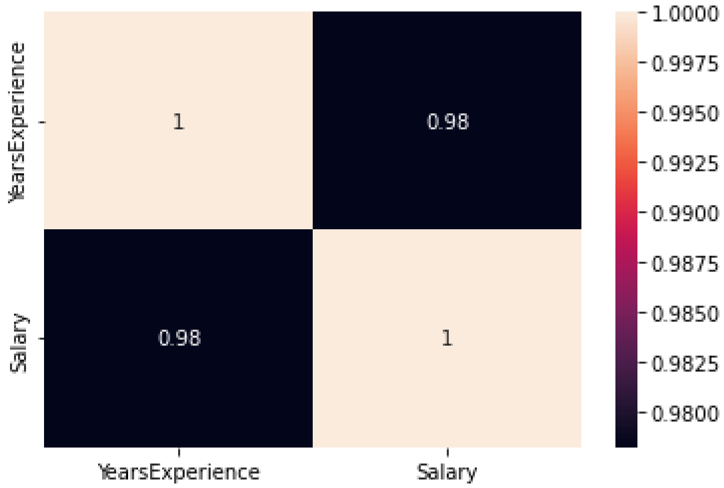
In [60]:

```
print("Correlation: "+ 'n', salary_data.corr()) # 0.978 which is high positive correlation
# Draw a heatmap for correlation matrix
plt.subplot(1,1,1)
sns.heatmap(salary_data.corr(), annot=True)
```

```
Correlation: n                 YearsExperience    Salary
YearsExperience         1.000000  0.978242
Salary                  0.978242  1.000000
```

Out[60]:

```
<AxesSubplot:>
```



correlation =0.98, which is a high positive correlation. This means the dependent variable increases as the independent variable increases.

4.Normalization

As we can see, there is a huge difference between the values of YearsExperience, Salary columns. We can use Normalization to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.

I use sklearn.preprocessing.Normalize to normalize our data. It returns values between 0 and 1.

In [62]:

```
# Create new columns for the normalized values
salary_data['Norm_YearsExp'] = preprocessing.normalize(salary_data[['YearsExperience']], ax
salary_data['Norm_Salary'] = preprocessing.normalize(salary_data[['Salary']], axis=0)
salary_data.head()
```

Out[62]:

	YearsExperience	Salary	Norm_YearsExp	Norm_Salary
0	1.1	39343.0	0.033464	0.089074
1	1.3	46205.0	0.039549	0.104610
2	1.5	37731.0	0.045633	0.085424
3	2.0	43525.0	0.060844	0.098542
4	2.2	39891.0	0.066928	0.090315

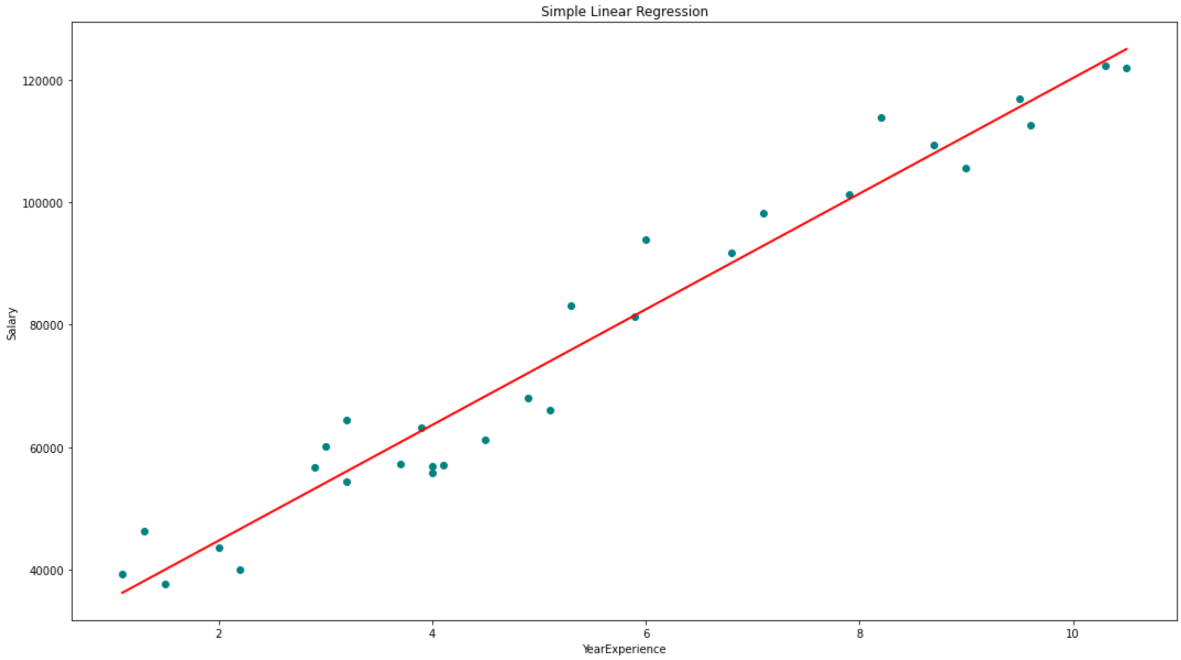
Linear Regression using scikit-learn

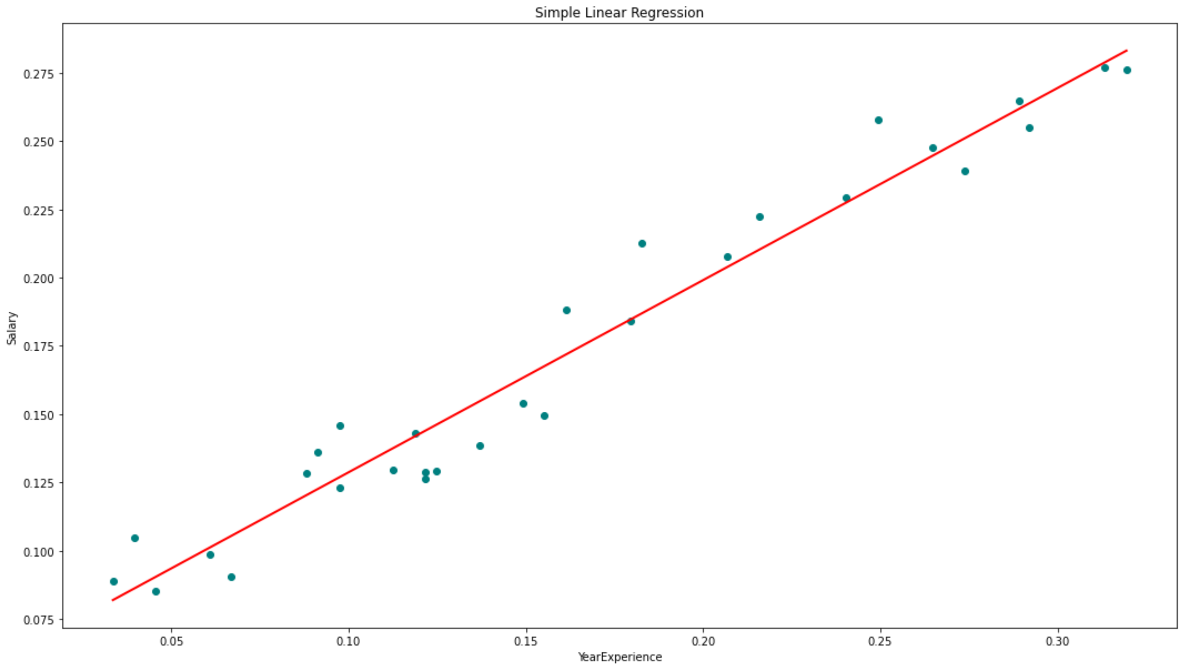
LinearRegression(): LinearRegression fits a linear model with coefficients β = (β1, …, βp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

In [68]:

```
def regression(salary_data):
#     defining the independent and dependent features
    x = salary_data.iloc[:, 1:2]
    y = salary_data.iloc[:, 0:1]
    # print(x,y)

    # Instantiating the LinearRegression object
    regressor = LinearRegression()

    # Training the model
    regressor.fit(x,y)

    # Checking the coefficients for the prediction of each of the predictor
    print('n'+"Coeff of the predictor: ",regressor.coef_)

    # Checking the intercept
    print("Intercept: ",regressor.intercept_)

    # Predicting the output
    y_pred = regressor.predict(x)
#     print(y_pred)

    # Checking the MSE
    print("Mean squared error(MSE): %.2f" % mean_squared_error(y, y_pred))
    # Checking the R2 value
    print("Coefficient of determination: %.3f" % r2_score(y, y_pred)) # Evaluates the perfo

    # visualizing the results.
    plt.figure(figsize=(18, 10))
    # Scatter plot of input and output values
    plt.scatter(x, y, color='teal')
    # plot of the input and predicted output values
    plt.plot(x, regressor.predict(x), color='Red', linewidth=2 )
    plt.title('Simple Linear Regression')
    plt.xlabel('YearExperience')
    plt.ylabel('Salary')


# Driver code
regression(salary_data[['Salary', 'YearsExperience']]) # 0.957 accuracy
regression(salary_data[['Norm_Salary', 'Norm_YearsExp']]) # 0.957 accuracy
```

```
nCoeff of the predictor:  [[9449.96232146]]
Intercept:  [25792.20019867]
Mean squared error(MSE): 31270951.72
Coefficient of determination: 0.957
nCoeff of the predictor:  [[0.70327706]]
Intercept:  [0.05839456]
Mean squared error(MSE): 0.00
Coefficient of determination: 0.957
```





We achieved 95.7% accuracy using scikit-learn but there is not much scope to understand the in-depth insights about the relevance of features from this model. So let’s build a model using statsmodels.api, statsmodels.formula.api

In [ ]:

```

```

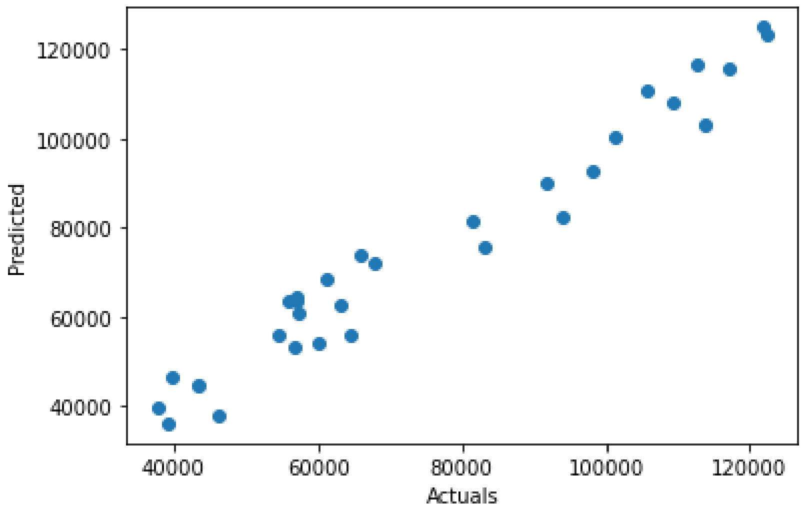
Linear Regression using statsmodel.formula.api (smf)

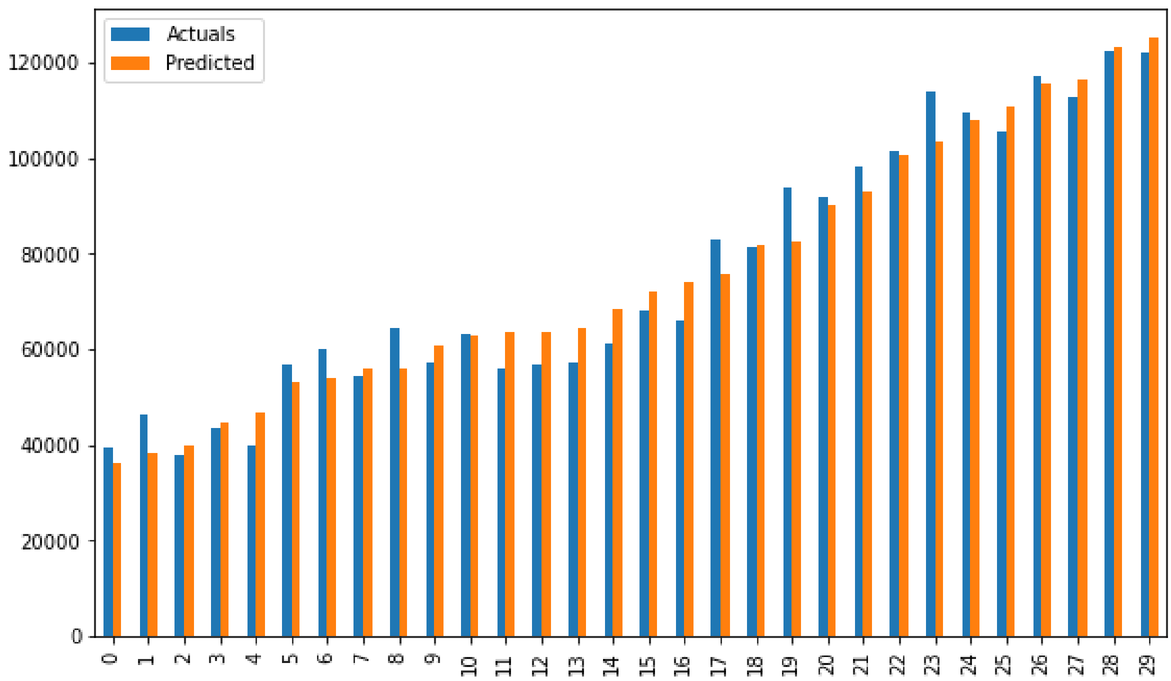
In [69]:

```
def smf_ols(salary_data):
    # defining the independent and dependent features
    x = salary_data.iloc[:, 1:2]
    y = salary_data.iloc[:, 0:1]
#     print(x)
    # train the model
    model = smf.ols('y~x', data=salary_data).fit()
    # print model summary
    print(model.summary())

    # Predict y
    y_pred = model.predict(x)
#     print(type(y), type(y_pred))
#     print(y, y_pred)

    y_lst = y.Salary.values.tolist()
#     y_lst = y.iloc[:, -1:].values.tolist()
    y_pred_lst = y_pred.tolist()

#     print(y_lst)

    data = [y_lst, y_pred_lst]
#     print(data)
    res = pd.DataFrame({'Actuals':data[0], 'Predicted':data[1]})
#     print(res)

    plt.scatter(x=res['Actuals'], y=res['Predicted'])
    plt.ylabel('Predicted')
    plt.xlabel('Actuals')

    res.plot(kind='bar',figsize=(10,6))

# Driver code
smf_ols(salary_data[['Salary', 'YearsExperience']]) # 0.957 accuracy
# smf_ols(df[['Norm_Salary', 'Norm_YearsExp']]) # 0.957 accuracy
```

```
                            OLS Regression Results                            
=============================================================================
==
Dep. Variable:                      y   R-squared:                       0.9
57
Model:                            OLS   Adj. R-squared:                  0.9
55
Method:                 Least Squares   F-statistic:                     62
2.5
Date:                Thu, 10 Feb 2022   Prob (F-statistic):           1.14e-
20
Time:                        22:21:28   Log-Likelihood:                -301.
44
No. Observations:                  30   AIC:                             60
6.9
Df Residuals:                      28   BIC:                             60
9.7
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
=============================================================================
==
                 coef    std err          t      P>|t|      [0.025      0.97
5]
```

```
------------------------------------------------------------------------------
--
Intercept    2.579e+04   2273.053     11.347      0.000    2.11e+04    3.04e+
04
x            9449.9623    378.755     24.950      0.000    8674.119    1.02e+
04
==============================================================================
==
Omnibus:                        2.140   Durbin-Watson:                   1.6
48
Prob(Omnibus):                  0.343   Jarque-Bera (JB):                1.5
69
Skew:                           0.363   Prob(JB):                        0.4
56
Kurtosis:                       2.147   Cond. No.                         1
3.2
==============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
```



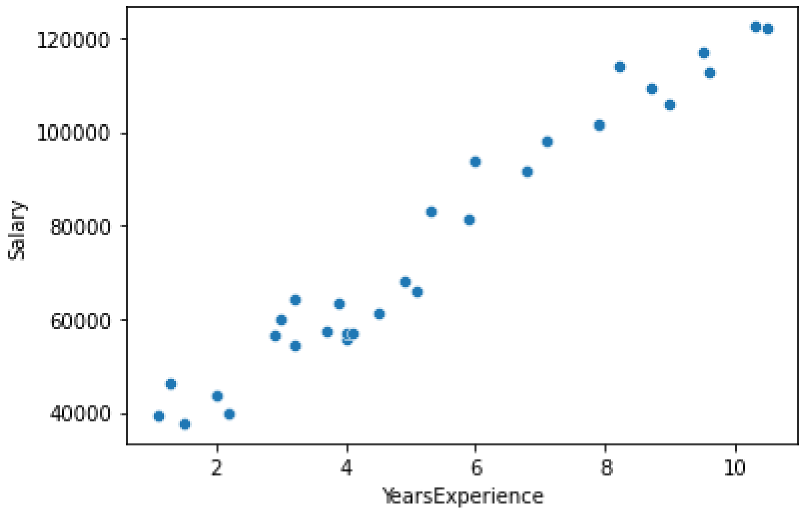


In othis case, the R-squared value (0.957) is close to Adj. R-squared value (0.955) is a good sign that the input features are contributing to the predictor model.

5.Check assumptions

In [70]:

```
#linearity check
sns.scatterplot(x='YearsExperience',y='Salary',data=salary_data)
plt.show()
```



In [72]:

```
salary_data.corr()
```

Out[72]:

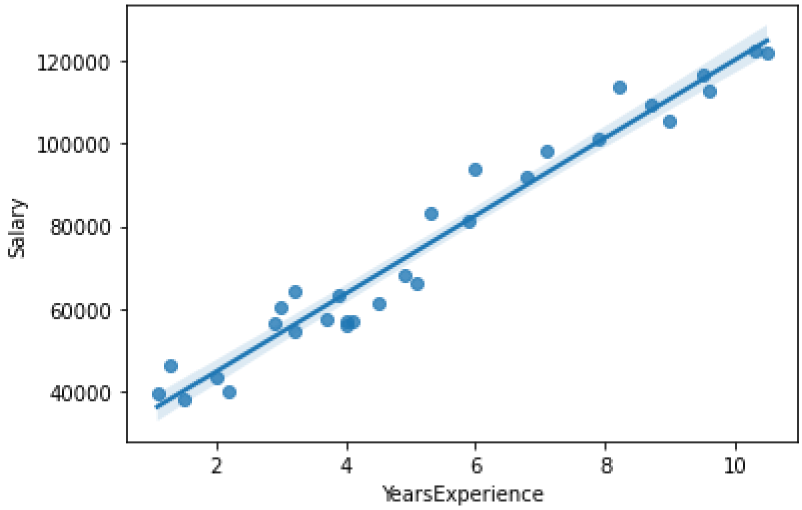
	YearsExperience	Salary	Norm_YearsExp	Norm_Salary
YearsExperience	1.000000	0.978242	1.000000	0.978242
Salary	0.978242	1.000000	0.978242	1.000000
Norm_YearsExp	1.000000	0.978242	1.000000	0.978242
Norm_Salary	0.978242	1.000000	0.978242	1.000000

In [73]:

```
sns.regplot(x='YearsExperience',y='Salary',data=salary_data)
plt.show
```

Out[73]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



6.Model Building || Model Training

In [74]:

```
lin_model=smf.ols(formula='Salary~YearsExperience',data=salary_data).fit()
lin_model
```

Out[74]:

```
<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x185083172
20>
```

7.Model Testing

In [75]:

```
lin_model.params
```

Out[75]:

```
Intercept          25792.200199
YearsExperience     9449.962321
dtype: float64
```

In [76]:

```
lin_model.tvalues,lin_model.pvalues
```

Out[76]:

```
(Intercept          11.346940
 YearsExperience    24.950094
 dtype: float64,
 Intercept          5.511950e-12
 YearsExperience    1.143068e-20
 dtype: float64)
```

In [77]:

```
lin_model.rsquared,lin_model.rsquared_adj
```

Out[77]:

```
(0.9569566641435086, 0.9554194021486339)
```

8. Model Prediction

In [78]:

```
pred_data={'YearsExperience':[2,4,6,8,10]}
pred_data
```

Out[78]:

```
{'YearsExperience': [2, 4, 6, 8, 10]}
```

In [79]:

```
test_data=pd.DataFrame(data=pred_data)
test_data
```

Out[79]:

	YearsExperience
0	2
1	4
2	6
3	8
4	10

In [81]:

```
lin_model.predict(test_data)
```

Out[81]:

```
0     44692.124842
1     63592.049484
2     82491.974127
3    101391.898770
4    120291.823413
dtype: float64
```

In [ ]:

```

```