Problem statement

Build a simple linear regression model to predict the Salary Hike using Years of Experience.

Start by Importing necessary libraries

necessary libraries are pandas, NumPy to work with data frames, matplotlib, seaborn for visualizations, and sklearn, statsmodels to build regression models.

In [1]:

```
import pandas as pd
import numpy as np
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
```

Once, we are done with importing libraries, we create a pandas dataframe from CSV file

In [32]:

```
df = pd.read_csv (r"/Users/rith/Desktop/Python /SLR/Salary_Data.csv")
df.head(n=6)
```

Out[32]:

	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
5	2.9	56642.0

Perform EDA (Exploratory Data Analysis)

The basic steps of EDA are:

Understand the dataset

-Identifying the number of features or columns -Identifying the features or columns -Identify the size of the dataset -Identifying the data types of features -Checking if the dataset has empty cells -Identifying the number of empty cells by features or columns

-Handling Missing Values and Outliers -Encoding Categorical variables -Graphical Univariate Analysis, Bivariate -Normalization and Scaling

In [33]:
```
len(df.columns) # identify the number of features
```
Out[33]:
```
2
```

In [34]:
```
df.columns # idenfity the features
```
Out[34]:
```
Index(['YearsExperience', 'Salary'], dtype='object')
```

In [35]:
```
df.shape # identify the size of of the dataset
```
Out[35]:
```
(30, 2)
```

In [36]:
```
df.dtypes # identify the datatypes of the features
```
Out[36]:
```
YearsExperience    float64
Salary             float64
dtype: object
```

In [37]:
```
df.isnull().values.any() # checking if dataset has empty cells
```
Out[37]:
```
False
```

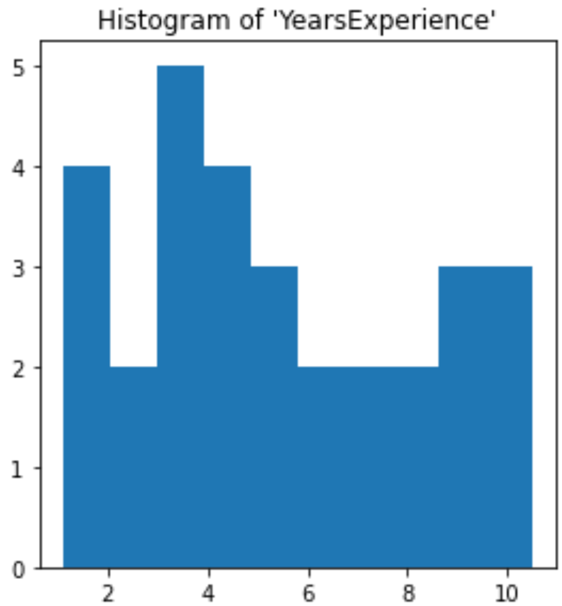
In [38]:
```
df.isnull().sum() # identify the number of empty cells
```
Out[38]:
```
YearsExperience    0
Salary             0
dtype: int64
```

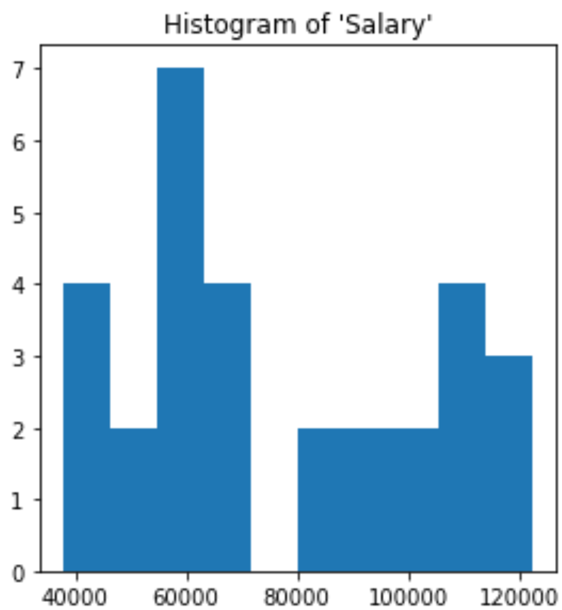
In [7]:

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

Out[7]:

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





In [ ]:

In [ ]:

Our dataset has two columns: YearsExperience, Salary. And both are of float datatype. We have 30 records and no null-values or outliers in our dataset.

Graphical Univariate analysis

For univariate analysis, we have Histogram, density plot, boxplot or violinplot, and Normal Q-Q plot. They help us understand the distribution of the data points and the presence of outliers.

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.

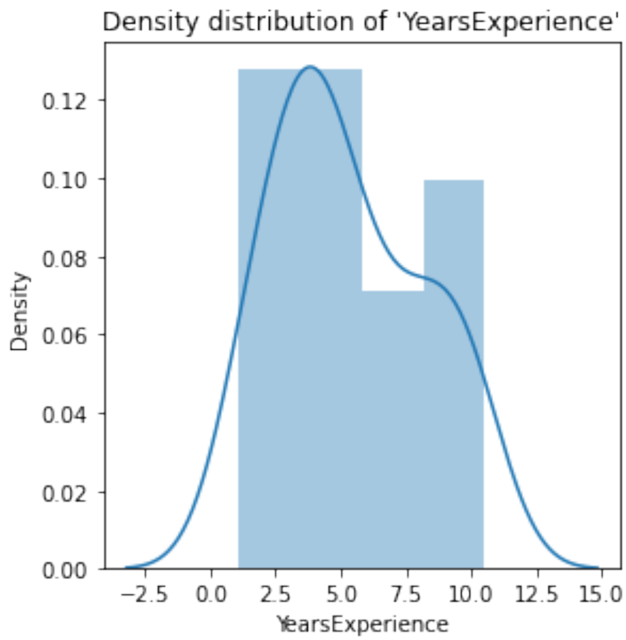
In [9]:

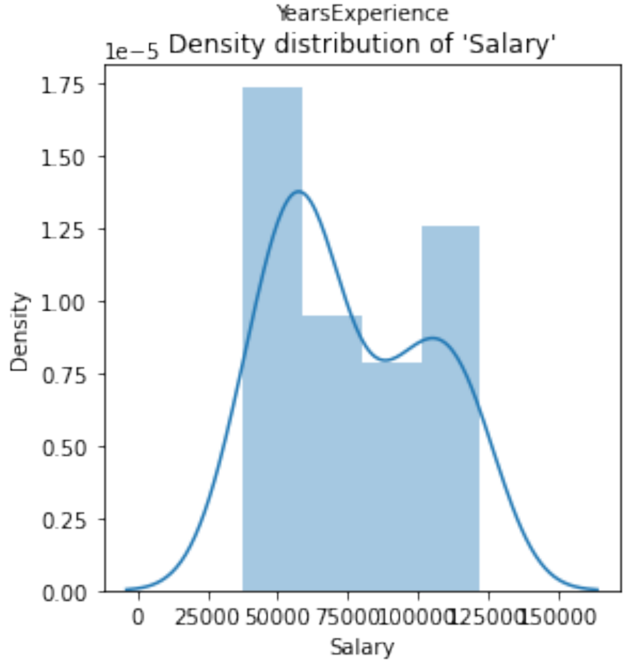
```
# Density plot
plt.figure(figsize=(20,10))
plt.subplot(2,4,2)
sns.distplot(df['YearsExperience'], kde=True)
plt.title("Density distribution of 'YearsExperience'")
plt.subplot(2,4,6)
sns.distplot(df['Salary'], kde=True)
plt.title("Density distribution of 'Salary'")
```

```
/Users/Code/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py
:2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figure-
level function with similar flexibility) or `histplot` (an axes-level function
for histograms).
  warnings.warn(msg, FutureWarning)
/Users/Code/opt/anaconda3/lib/python3.9/site-packages/seaborn/distributions.py
:2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figure-
level function with similar flexibility) or `histplot` (an axes-level function
for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[9]:

```
Text(0.5, 1.0, "Density distribution of 'Salary'")
```



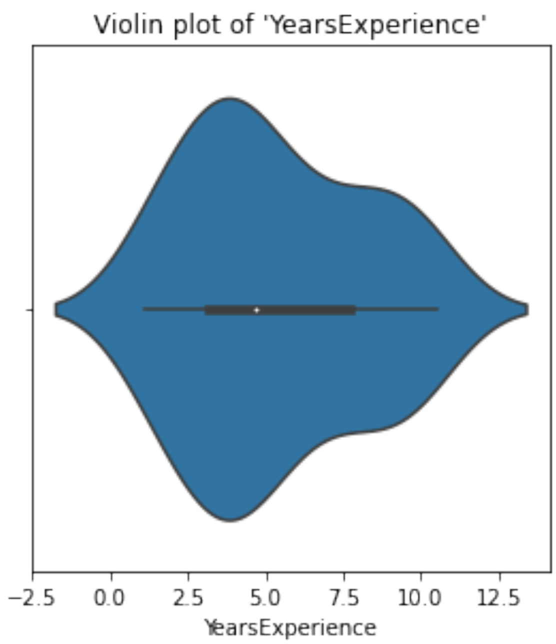


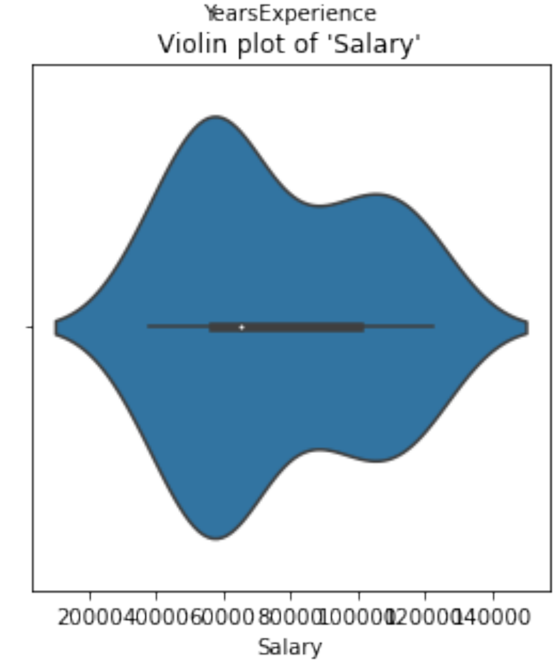
In [11]:
```
# boxplot or violin plot
# A violin plot is a method of plotting numeric data. It is similar to a box 
# with the addition of a rotated kernel density plot on each side
plt.figure(figsize=(20,10))
plt.subplot(2,4,3)
# plt.boxplot(df['YearsExperience'])
sns.violinplot(df['YearsExperience'])
# plt.title("Boxlpot of 'YearsExperience'")
plt.title("Violin plot of 'YearsExperience'")
plt.subplot(2,4,7)
# plt.boxplot(df['Salary'])
sns.violinplot(df['Salary'])
# plt.title("Boxlpot of 'Salary'")
plt.title("Violin plot of 'Salary'")
```

```
/Users/Code/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From versio
n 0.12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
/Users/Code/opt/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.py:3
6: FutureWarning: Pass the following variable as a keyword arg: x. From versio
n 0.12, the only valid positional argument will be `data`, and passing other a
rguments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
```

Out[11]:

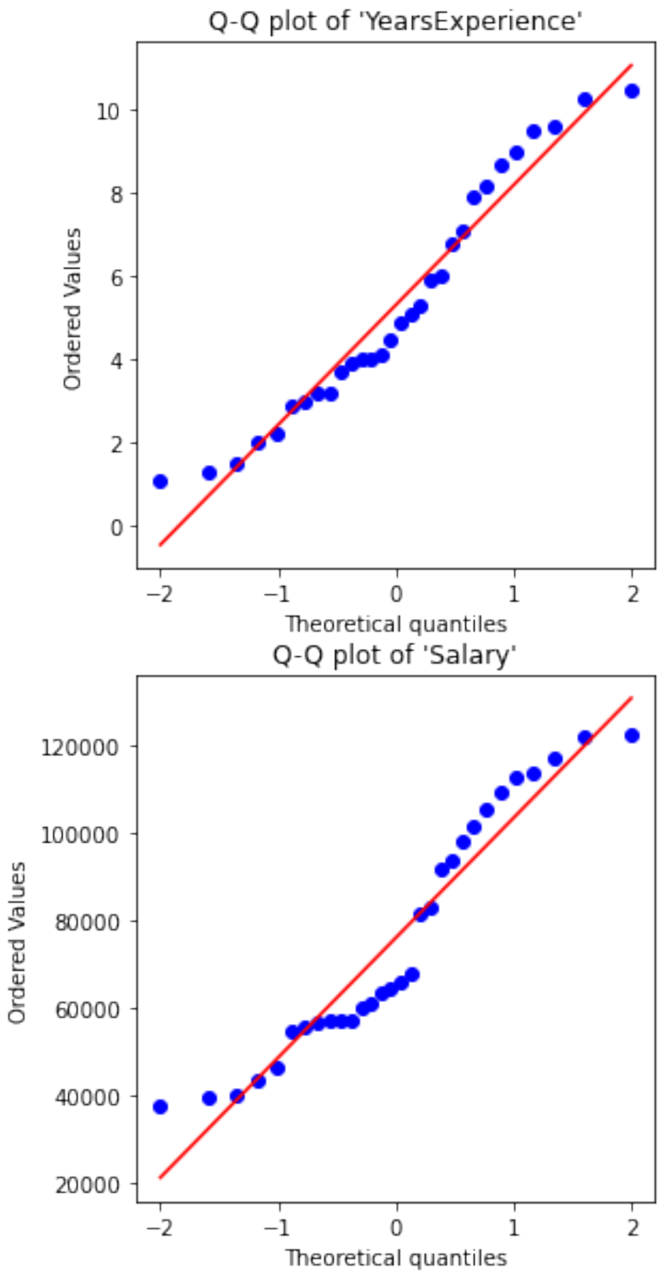
```
Text(0.5, 1.0, "Violin plot of 'Salary'")
```





In [13]:
```
# Normal Q-Q plot
plt.figure(figsize=(20,10))
plt.subplot(2,4,4)
probplot(df['YearsExperience'], plot=plt)
plt.title("Q-Q plot of 'YearsExperience'")
plt.subplot(2,4,8)
probplot(df['Salary'], plot=plt)
plt.title("Q-Q plot of 'Salary'")
```

Out[13]:
```
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.

Check if there is any correlation between the variables using df.corr()

In [16]:
```
print("Correlation: "+ 'n', df.corr()) # 0.978 which is high positive correla
# Draw a heatmap for correlation matrix
plt.subplot(1,1,1)
sns.heatmap(df.corr(), annot=True)
```

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

Out[16]:
```
<AxesSubplot:>
```

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 [19]:
```
#     defining the independent and dependent features
x= df.iloc[:, 1:2]
y= df.iloc[:, 0:1]
    # print(x,y)
```

In [21]:
```
# Instantiating the LinearRegression object
regressor = LinearRegression()
```

In [39]:
```
model = smf.ols('Salary ~ YearsExperience', data = df)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                 Salary   R-squared:                       0.957
Model:                            OLS   Adj. R-squared:                  0.955
Method:                 Least Squares   F-statistic:                     622.5
Date:                Fri, 15 Apr 2022   Prob (F-statistic):           1.14e-20
Time:                        16:19:17   Log-Likelihood:                -301.44
No. Observations:                  30   AIC:                             606.9
Df Residuals:                      28   BIC:                             609.7
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=====
                      coef    std err          t      P>|t|      [0.025      0
.975]
------------------------------------------------------------------------------
-----
Intercept        2.579e+04   2273.053     11.347      0.000    2.11e+04     3.0
4e+04
YearsExperience  9449.9623    378.755     24.950      0.000    8674.119     1.0
2e+04
==============================================================================
Omnibus:                        2.140   Durbin-Watson:                   1.648
Prob(Omnibus):                  0.343   Jarque-Bera (JB):                1.569
Skew:                           0.363   Prob(JB):                        0.456
Kurtosis:                       2.147   Cond. No.                         13.2
==============================================================================

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

In [22]:
```
# Training the model
regressor.fit(x,y)
```

Out[22]:
```
LinearRegression()
```

In [23]:
```
# Checking the coefficients for the prediction of each of the predictor
print('n'+"Coeff of the predictor: ",regressor.coef_)
```

```
nCoeff of the predictor:  [[0.00010127]]
```

In [24]:
```
# Checking the intercept
print("Intercept: ",regressor.intercept_)
```

```
Intercept:  [-2.38316056]
```

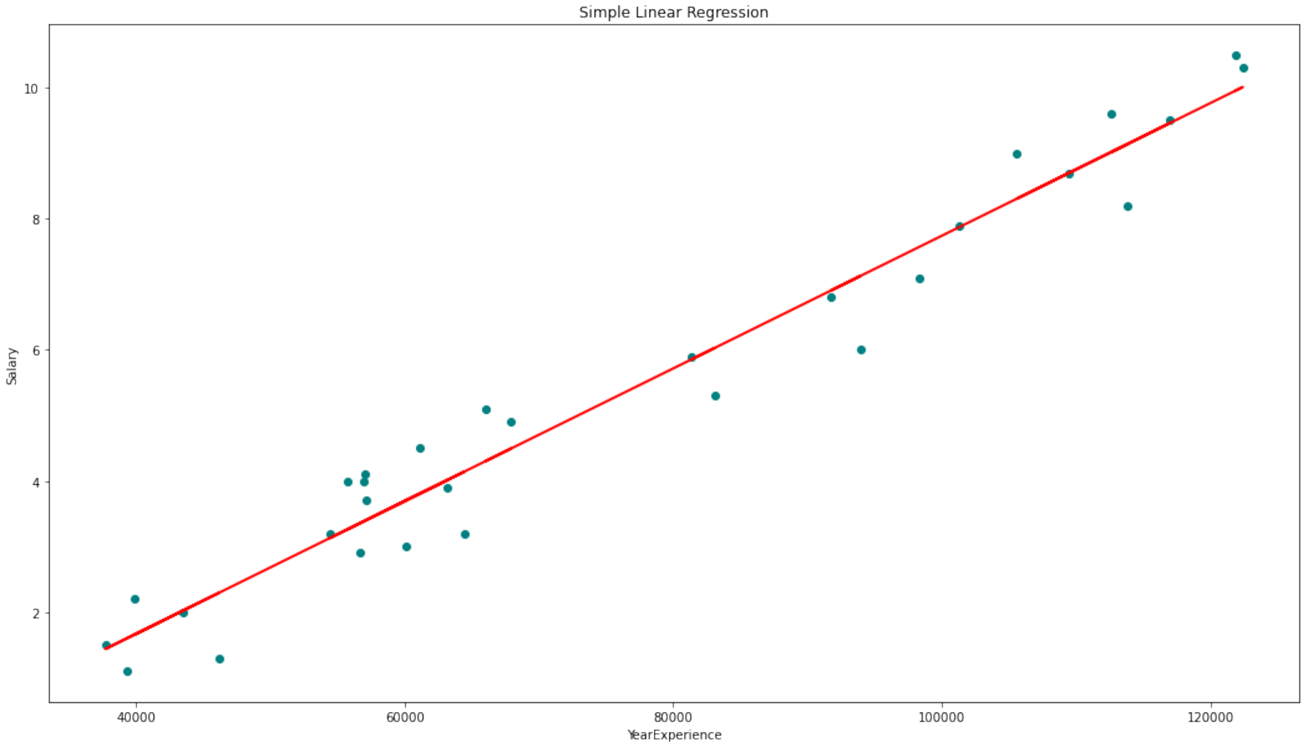
In [28]:
```
# Predicting the output
y_pred = regressor.predict(x)
#print(y_pred)
```

In [27]:
```
# 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
```

```
Mean squared error(MSE): 0.34
Coefficient of determination: 0.957
```

In [29]:
```
# 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')
```

Out[29]: Text(0, 0.5, 'Salary')



In [ ]:

In [ ]: