In [ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
```

EDA (Exploratory Data Analysis)

In [ ]:
```python
sal=pd.read_csv("C:\\Users\\Hi\\Desktop\\Python Datasets\\Salary_Data.csv")
sal.head()
```

Out[ ]:

	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

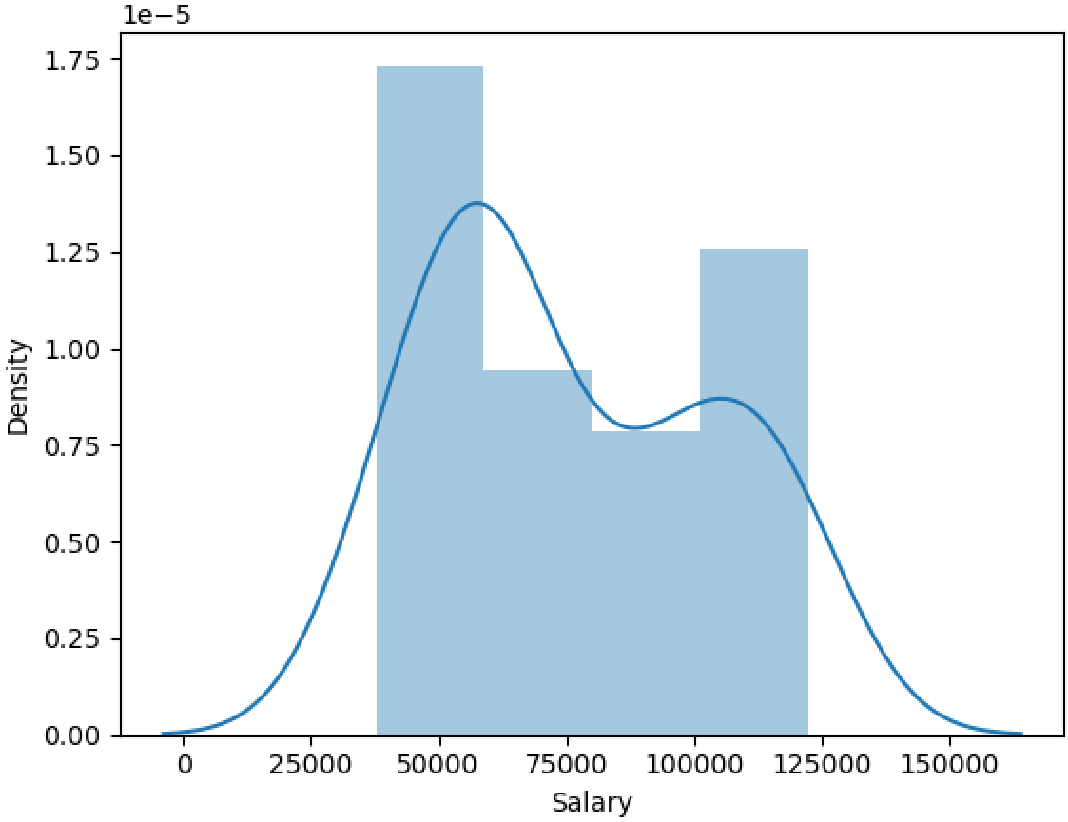
In [ ]:
```python
sal.info() #here we dont have null values or NA values
```

```
<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 [ ]:
```python
sns.distplot(sal['Salary']) #here we can see that Salary data Column is normally distrib
```

```
C:\Users\Hi\AppData\Local\Temp\ipykernel_18388\1423190555.py:1: UserWarning: 

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sal['Salary']) #here we can see that Salary data Column is normally distr
ibuted without any skewness
```

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



A skewness value greater than 1 or less than -1 indicates a highly skewed distribution. A value between 0.5 and 1 or -0.5 and -1 is moderately skewed. A value between -0.5 and 0.5 indicates that the distribution is fairly symmetrical.

In [ ]:

```
print('The skewness of the Salary Data Column is between -0.5 and 0.5 indicates that the
```

```
The skewness of the Salary Data Column is between -0.5 and 0.5 indicates that the distri
bution is fairly symmetrical.  0.35411967922959153
```

The pandas library function kurtosis() computes the Fisher's Kurtosis which is obtained by subtracting the Pearson's Kurtosis by three. With Fisher's Kurtosis, definition a normal distribution has a kurtosis of 0

Kurtosis number should be between 1 and - 1. If it is in this range that mean the data is normally distributed.

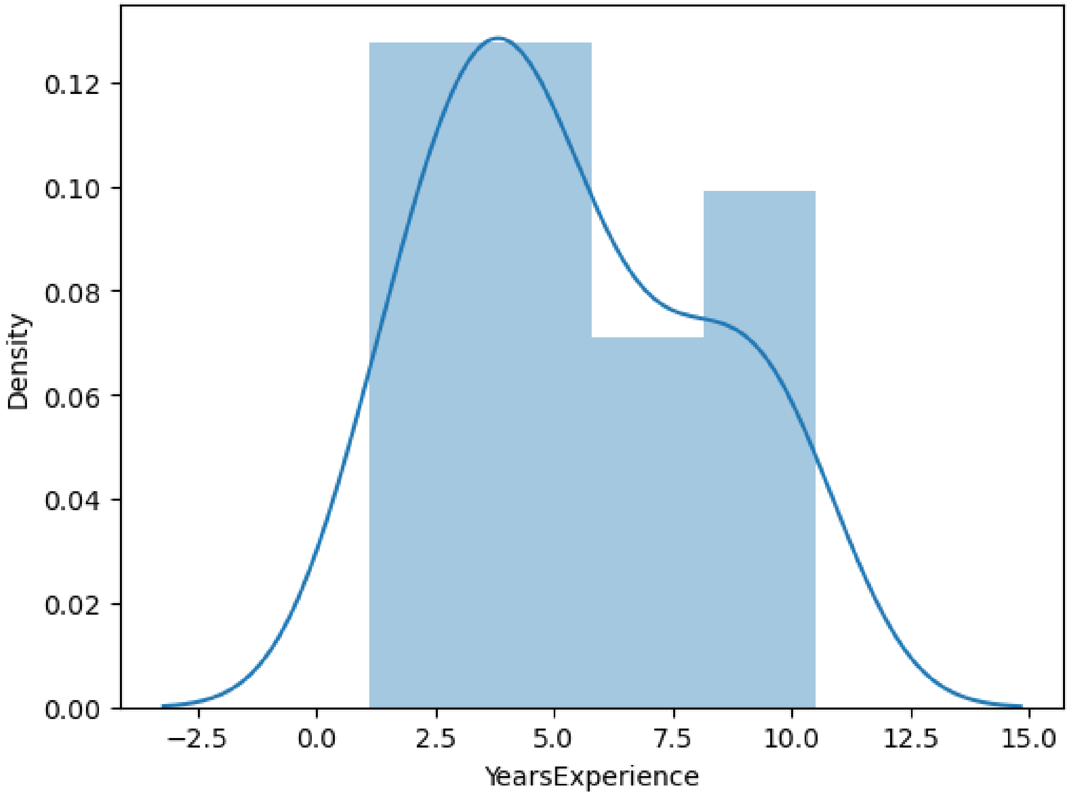
In [ ]:
```python
print('The Kurtosis of the Salary Data Column is :', sal.Salary.kurtosis())
```

```
The Kurtosis of the Salary Data Column is : -1.295421086394517
```

In [ ]:
```python
sns.distplot(sal['YearsExperience']) #here we can see that YearsExperience data column i
```

```
C:\Users\Hi\AppData\Local\Temp\ipykernel_18388\4093035387.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(sal['YearsExperience']) #here we can see that YearsExperience data column
is normally distributed and without any skewness
```

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



In [ ]:
```python
print('The skewness of the YearsExperience Data Column is between -0.5 and 0.5 indicates
```

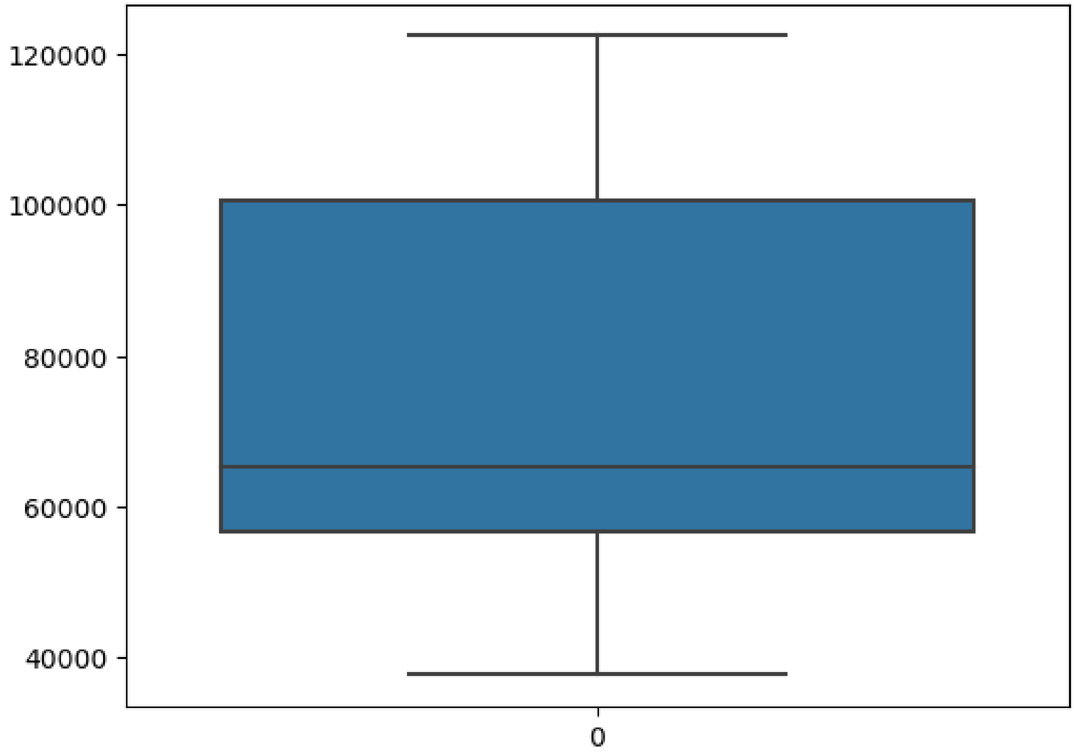
```
The skewness of the YearsExperience Data Column is between -0.5 and 0.5 indicates that t
he distribution is fairly symmetrical.  0.37956024064804106
```

In [ ]:
```python
print('The Kurtosis of the Salary Data Column is :', sal.YearsExperience.kurtosis())
```

```
The Kurtosis of the Salary Data Column is : -1.0122119403325072
```

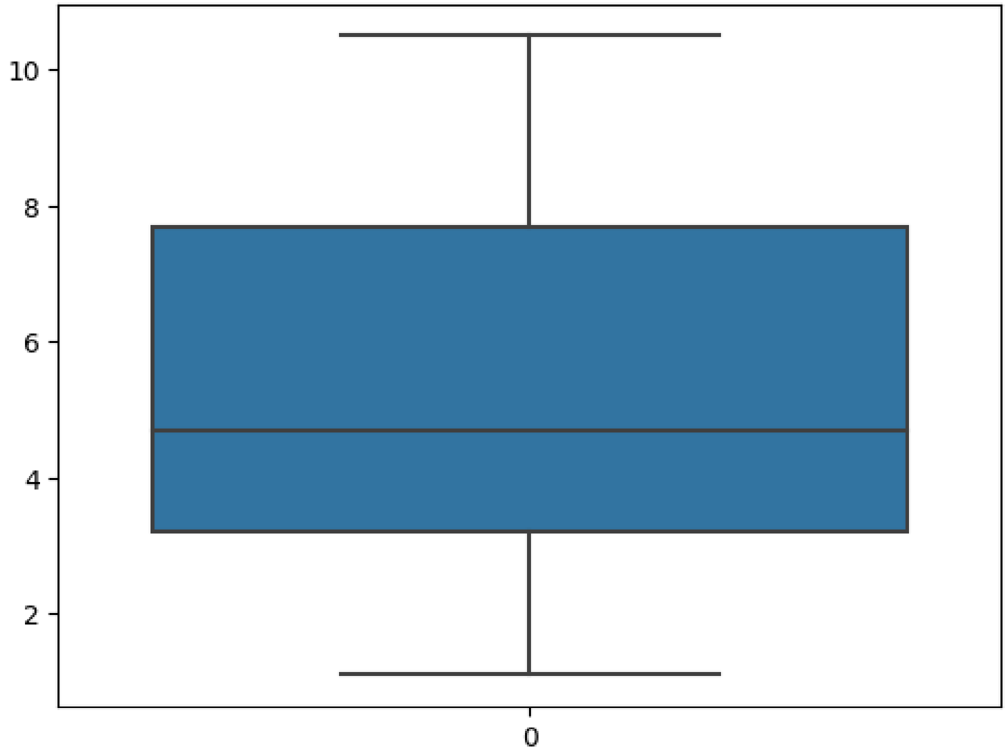
In [ ]:
```python
sns.boxplot(sal['Salary']) #here we can see that there are no outliers and the line in t
```

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



In [ ]:
```
sns.boxplot(sal['YearsExperience'])  #here we can see that there are no outliers and the
```

Out[ ]: <AxesSubplot:>



In [ ]:
```
sal.describe() #description of the whole dataset
```

Out[ ]:

	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

Checking for Correlation

In [ ]:

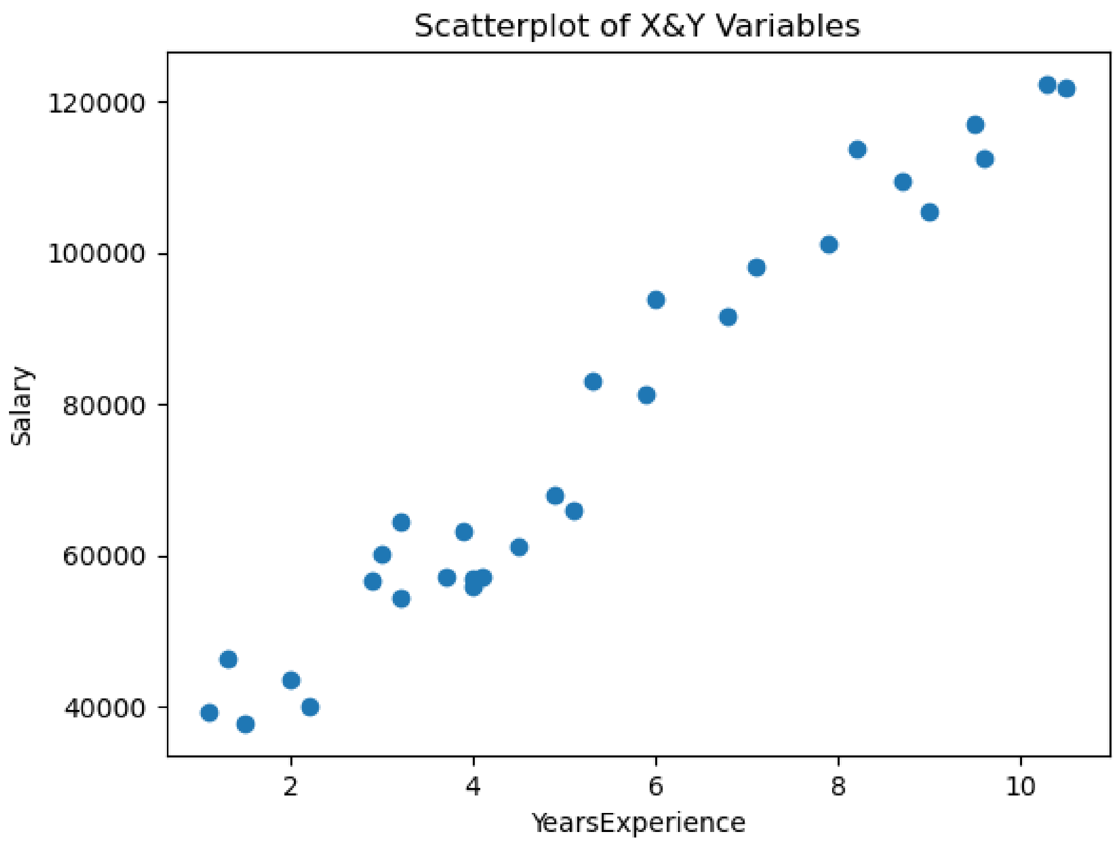
```
sal.corr() #there is very high correlation between target variable & Independent Variabl
```

Out[ ]:

	YearsExperience	Salary
YearsExperience	1.000000	0.978242
Salary	0.978242	1.000000

In [ ]:

```
plt.scatter(sal.YearsExperience,sal.Salary) # through this scatterplot we can see there
plt.title('Scatterplot of X&Y Variables')
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.show()
```



Feature Scaling

In [ ]:

```
#Normalization of Data , so that algorithm  does not get affected by the Salary column w
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
d=scaler.fit_transform(sal)
names=sal.columns
saldf=pd.DataFrame(d,columns=names)
saldf.head()
```

Out[ ]:

	YearsExperience	Salary
0	0.000000	0.019041
1	0.021277	0.100094
2	0.042553	0.000000
3	0.095745	0.068438
4	0.117021	0.025514

Transforming Dataset by applying square root on data columns

In [ ]:

```
saldf['T_YearsExperience']=np.sqrt(saldf['YearsExperience'])
print('Skewness of Years Expernc column without Feature Scaling---->',sal.YearsExperienc
print('Skewness of Years Expernc column with Feature Scaling And Square root Transformat
print('Kurtosis of Years Expernc column without Feature Scaling---->',sal.YearsExperienc
print('Kurtosis of Years Expernc column with Feature Scaling And Square root Transformat
saldf['T_Salary']=np.sqrt(saldf['Salary'])
print('Skewness of Salary column without Feature Scaling---->',sal.Salary.skew())
print('Skewness of Salary column with Feature Scaling And Square root Transformation----
print('Kurtosis of Salary column without Feature Scaling---->',sal.Salary.kurtosis())
print('Kurtosis of Salary column with Feature Scaling And Square root Transformation----
```

```
Skewness of Years Expernc column without Feature Scaling----> 0.37956024064804106
Skewness of Years Expernc column with Feature Scaling And Square root Transformation----
> -0.46589551648821814
Kurtosis of Years Expernc column without Feature Scaling----> -1.0122119403325072
Kurtosis of Years Expernc column with Feature Scaling And Square root Transformation----
> -0.2178604409043401
Skewness of Salary column without Feature Scaling----> 0.35411967922959153
Skewness of Salary column with Feature Scaling And Square root Transformation----> -0.36
321059439457953
Kurtosis of Salary column without Feature Scaling----> -1.295421086394517
Kurtosis of Salary column with Feature Scaling And Square root Transformation----> -0.60
99285962701315
```

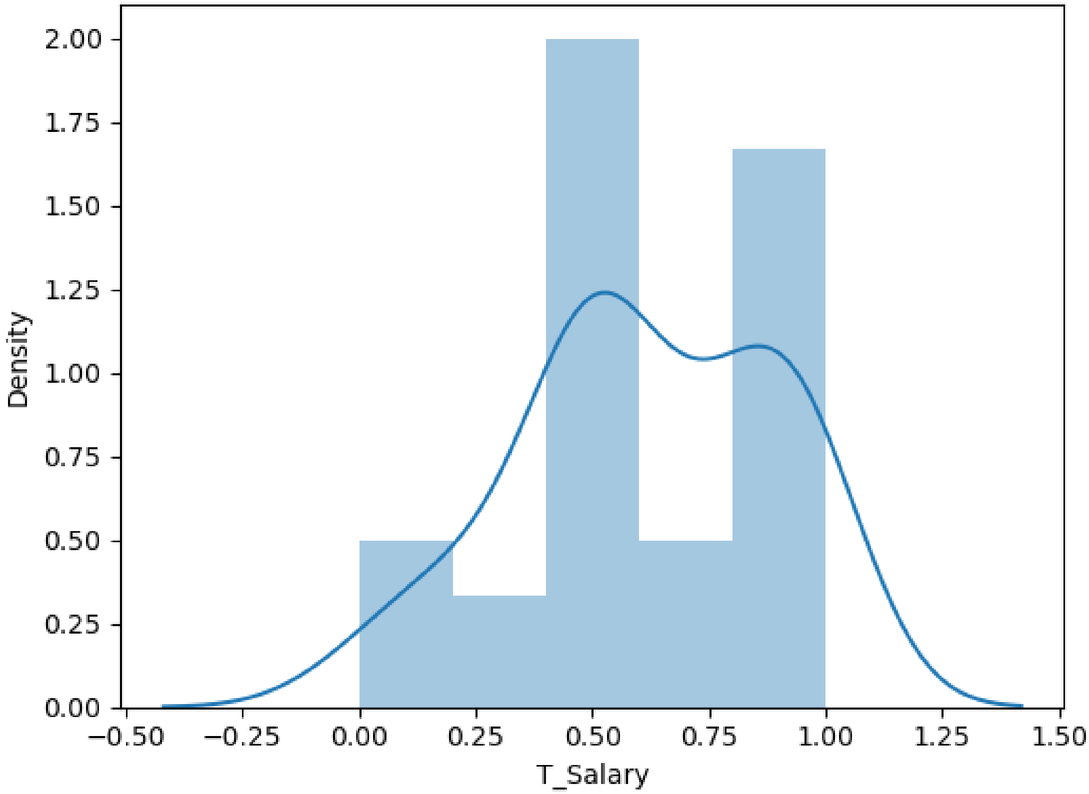
In [ ]:

```
sns.distplot(saldf.T_Salary)
```

```
C:\Users\Hi\AppData\Local\Temp\ipykernel_18388\3331751611.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(saldf.T_Salary)
```

Out[ ]:

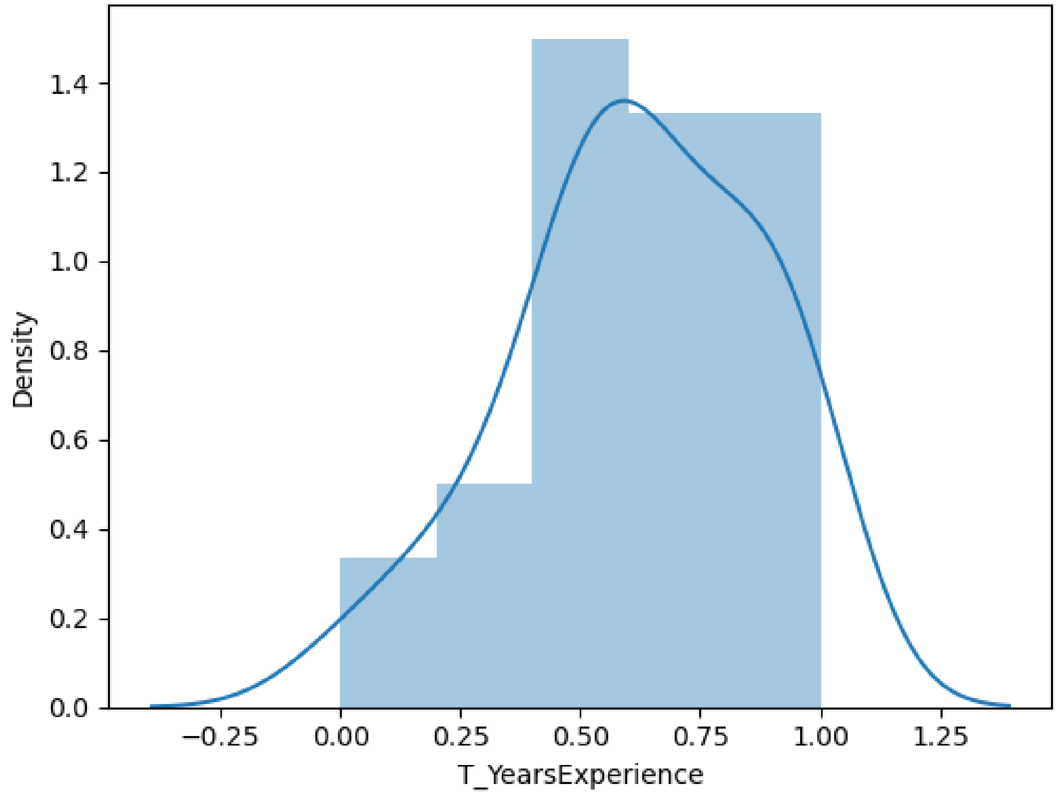
```
<AxesSubplot:xlabel='T_Salary', ylabel='Density'>
```



In [ ]:
```
sns.distplot(saldf.T_YearsExperience) # here we can see through Density plot that there
```

```
C:\Users\Hi\AppData\Local\Temp\ipykernel_18388\3041252967.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(saldf.T_YearsExperience) # here we can see through Density plot that ther
e is change in data distribution after tansforming data by square rooting the dataset
```

Out[ ]:
```
<AxesSubplot:xlabel='T_YearsExperience', ylabel='Density'>
```



Model1 Creation

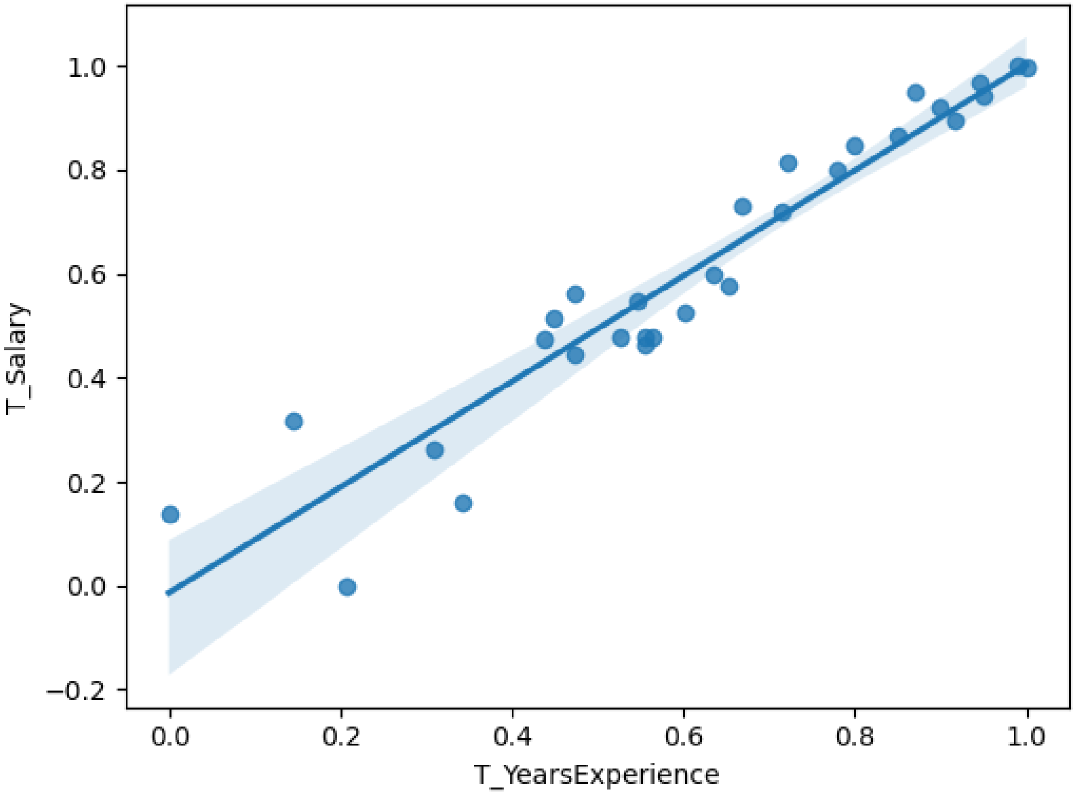
In [ ]:
```
model=smf.ols("T_Salary~T_YearsExperience",data=saldf).fit()
```

In [ ]:
```
model.params
```

Out[ ]:
```
Intercept           -0.013585
T_YearsExperience    1.015570
dtype: float64
```

In [ ]:
```
sns.regplot(x="T_YearsExperience",y="T_Salary",data=saldf)
```

Out[ ]:
```
<AxesSubplot:xlabel='T_YearsExperience', ylabel='T_Salary'>
```



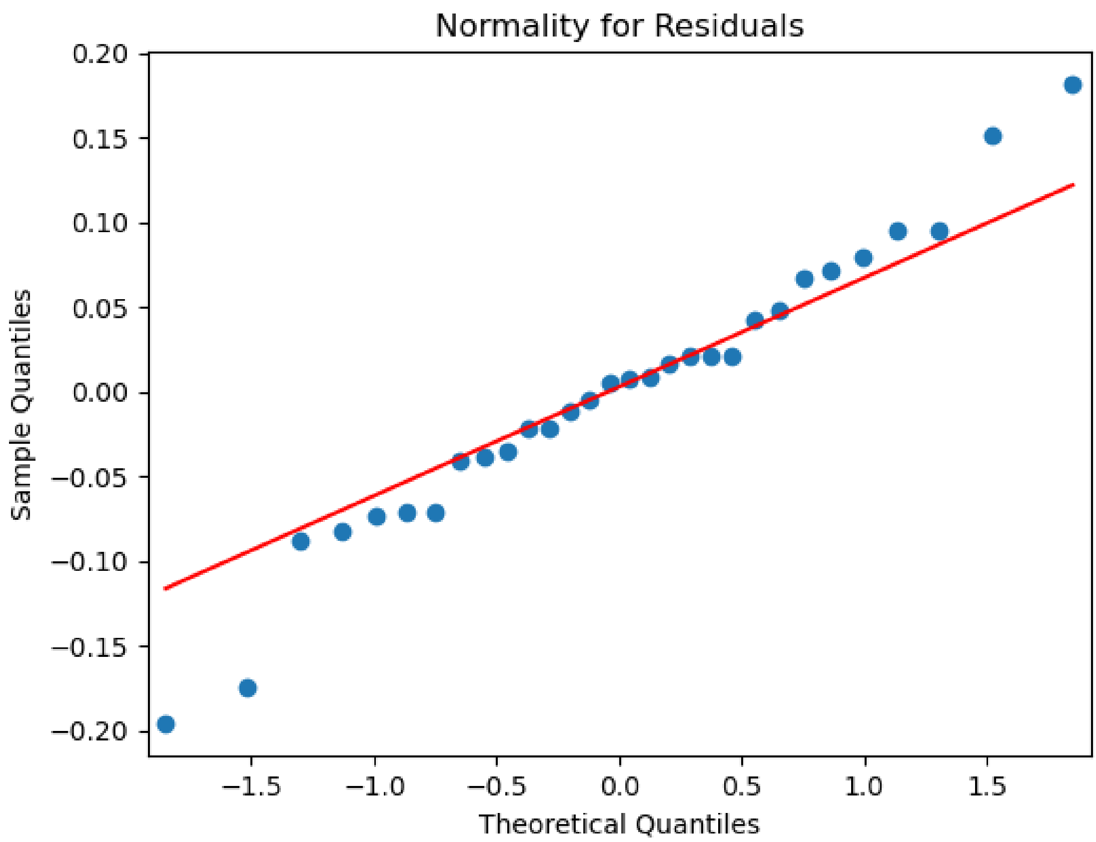
In [ ]:
```
model.tvalues,'/n',model.pvalues #pvalue is <0.05 hence rejecting H0 Hypothesis- (there
```

Out[ ]:
```
(Intercept            -0.334480
 T_YearsExperience    16.741112
 dtype: float64,
 '/n',
 Intercept            7.405098e-01
 T_YearsExperience    4.064467e-16
 dtype: float64)
```

In [ ]:
```
model.rsquared,model.rsquared_adj,model.aic
```

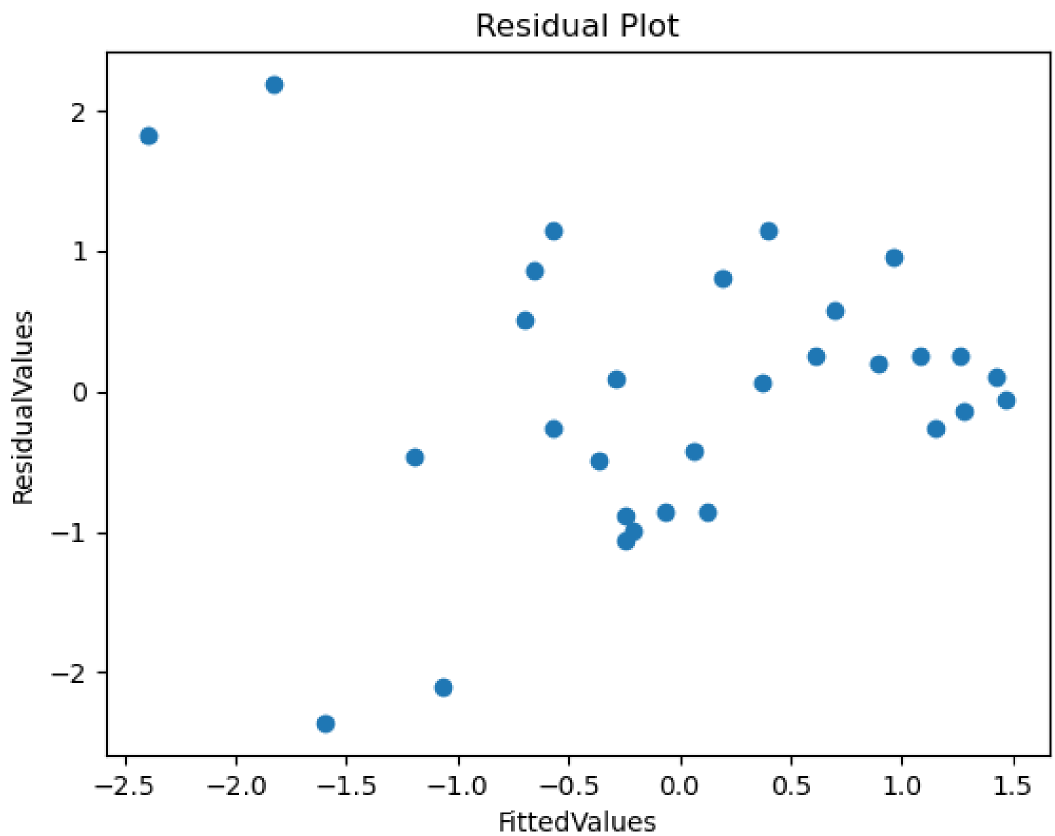
Out[ ]:
```
(0.909169005021548, 0.9059250409151747, -61.14601445430699)
```

In [ ]:
```
qqplot=sm.qqplot(model.resid,line='q')
plt.title("Normality for Residuals")
plt.show()
```



In [ ]:
```
def get_standardized_values(vals):
    return(vals-vals.mean())/vals.std()
```

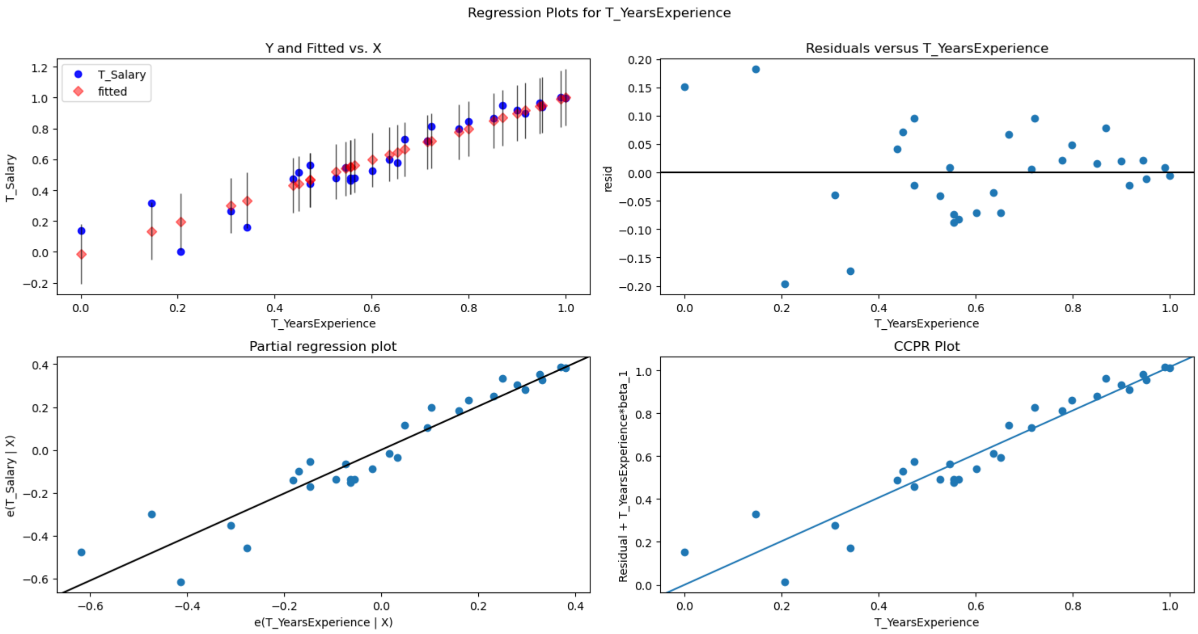
In [ ]:
```
plt.scatter(get_standardized_values(model.fittedvalues),
            get_standardized_values(model.resid))
plt.title("Residual Plot")
plt.xlabel("FittedValues")
plt.ylabel("ResidualValues")
plt.show() #There are no patterns in Residual vs Fitted Values ,hence No problem orelse
```



In [ ]:

```
fig=plt.figure(figsize=(15,8))
fig=sm.graphics.plot_regress_exog(model,"T_YearsExperience",fig=fig)
plt.show()
```

eval_env: 1



In [ ]:

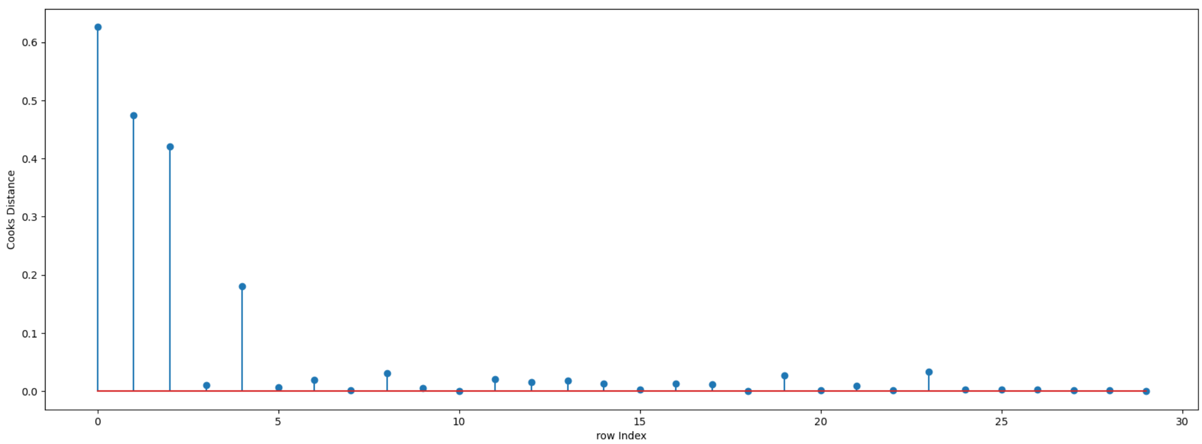
```
model_influence=model.get_influence()
(c, _)= model_influence.cooks_distance
summary_cooks=model_influence.summary_frame()
summary_cooks.head()
```

Out[ ]:

	dfb_Intercept	dfb_T_YearsExperience	cooks_d	standard_resid	hat_diag	dffits_internal	student_resid
0	1.191071	-1.101646	0.625899	2.043419	0.230646	1.118838	2.175344
1	1.059237	-0.937941	0.473995	2.330213	0.148637	0.973648	2.548653
2	-1.005122	0.867061	0.420598	-2.471064	0.121082	-0.917167	-2.744131
3	-0.136497	0.110324	0.010483	-0.482163	0.082722	-0.144795	-0.475452
4	-0.603758	0.473894	0.179608	-2.138019	0.072858	-0.599347	-2.295185

In [ ]:

```
fig=plt.subplots(figsize=(20,7))
plt.stem(np.arange(len(saldf)),np.round(c,3))
plt.xlabel("row Index")
plt.ylabel("Cooks Distance")
plt.show()
```



In [ ]:

```
(np.argmax(c),np.max(c)) # cooks distance threshold is 4/N so 4/30 = 0.1333 , hence data
```

Out[ ]: (0, 0.6258988217853959)

In [ ]:
```
(model.rsquared_adj,model.aic)
```

Out[ ]: (0.9059250409151747, -61.14601445430699)

In [ ]:
```
import math
from sklearn.metrics import mean_squared_error
```

In [ ]:
```
mse_m1= mean_squared_error(saldf.T_Salary ,model.fittedvalues) #checking for RMSE value
rmse_m1=math.sqrt(mse_m1)
print("the difference between actual and predicted values of model1 is :---",rmse_m1)
```

```
the difference between actual and predicted values of model1 is :--- 0.08169965982966364
```

In [ ]:
```
saldf.head()
```

Out[ ]:

	YearsExperience	Salary	T_YearsExperience	T_Salary
0	0.000000	0.019041	0.000000	0.137989
1	0.021277	0.100094	0.145865	0.316377
2	0.042553	0.000000	0.206284	0.000000
3	0.095745	0.068438	0.309426	0.261607
4	0.117021	0.025514	0.342084	0.159730

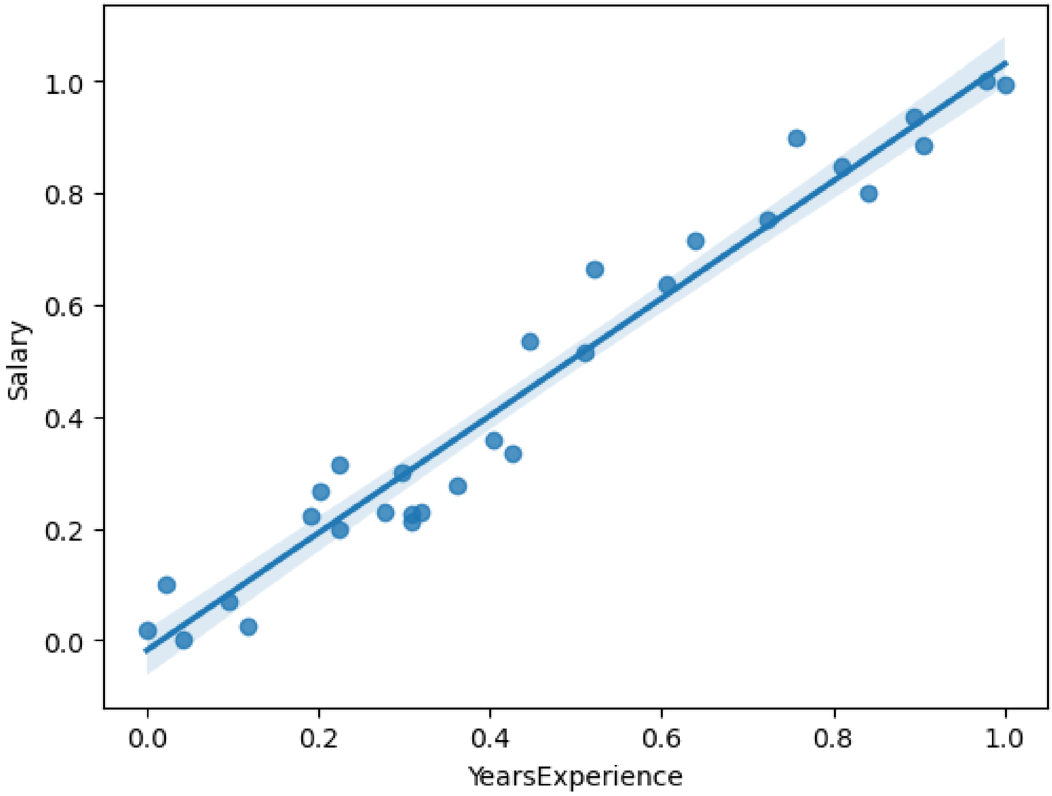
In [ ]:
```
model2=smf.ols("Salary~YearsExperience",data=saldf).fit()
```

In [ ]:
```
model2.params
```

Out[ ]:
```
Intercept         -0.018236
YearsExperience    1.049252
dtype: float64
```

In [ ]:
```
sns.regplot(x="YearsExperience",y="Salary",data=saldf)
```

Out[ ]: <AxesSubplot:xlabel='YearsExperience', ylabel='Salary'>



In [ ]:
```
model2.tvalues,'/n',model2.pvalues #pvalue is <0.05 hence rejecting H0 Hypothesis- (ther
```

Out[ ]:
```
(Intercept          -0.806598
 YearsExperience    24.950094
 dtype: float64,
 '/n',
 Intercept          4.266967e-01
 YearsExperience    1.143068e-20
 dtype: float64)
```

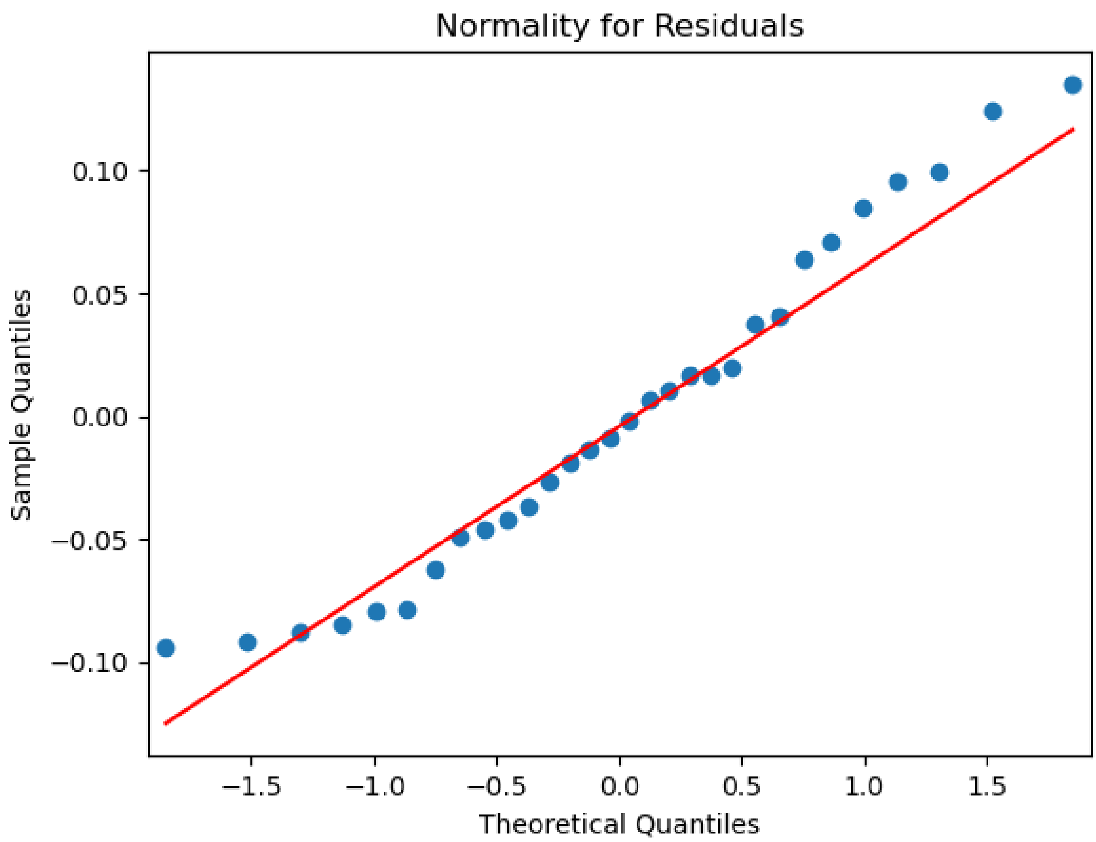
Model Validation using R-squared , R-squared adjusted, AIC(Akaike Information Criterion)

In [ ]:
```
model2.rsquared,model2.rsquared_adj,model2.aic
```

Out[ ]:
```
(0.9569566641435086, 0.9554194021486339, -73.90159391406291)
```

Residual Plot for Homoscedasticity

In [ ]:
```
qqplot1=sm.qqplot(model2.resid,line='q')
plt.title("Normality for Residuals")
plt.show()
```



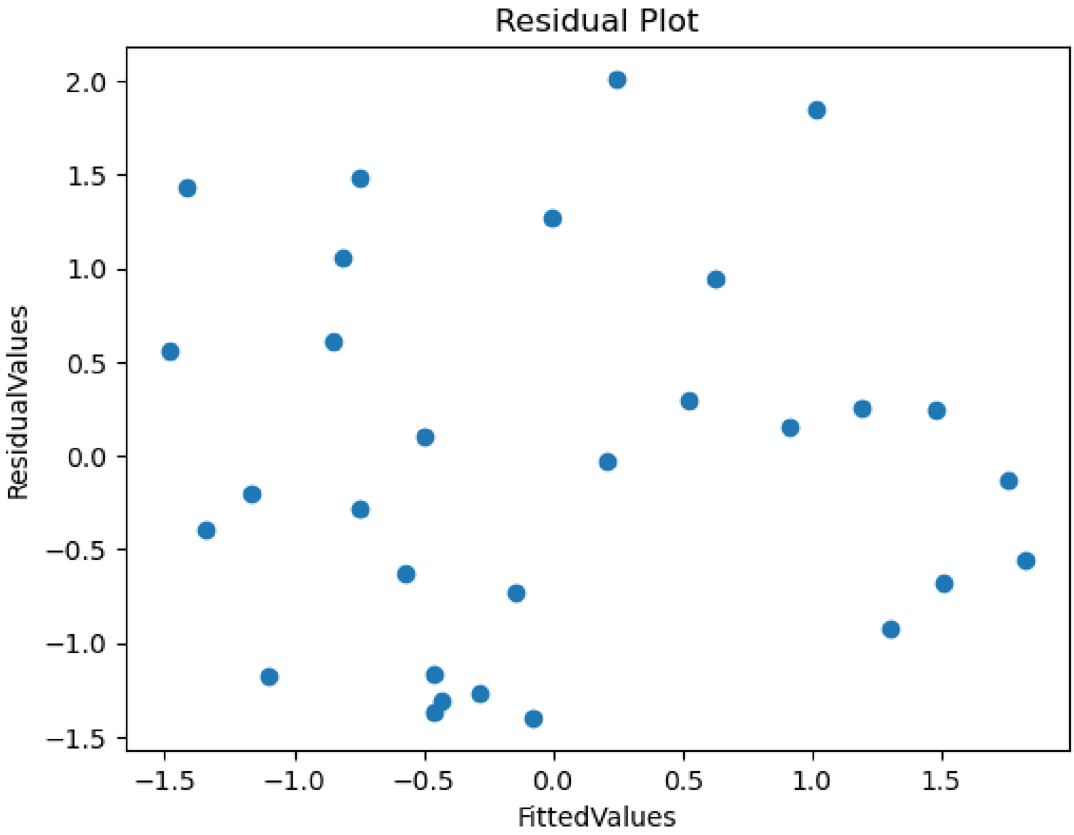
In [ ]:
```
list(np.where(model2.resid>10))
```

Out[ ]:
```
[array([], dtype=int64)]
```

Residual Analysis

In [ ]:
```
def get_standardized_values(vals):
    return(vals-vals.mean())/vals.std()
```

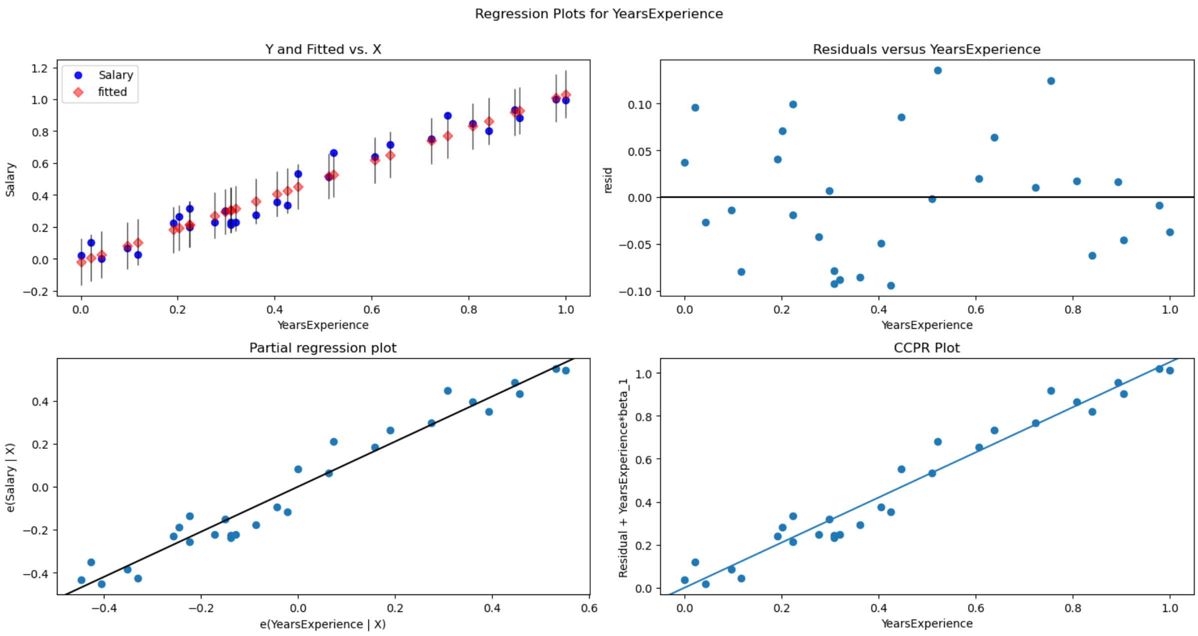
In [ ]:
```
plt.scatter(get_standardized_values(model2.fittedvalues),
            get_standardized_values(model2.resid))
plt.title("Residual Plot")
plt.xlabel("FittedValues")
plt.ylabel("ResidualValues")
plt.show() #There are no patterns in Residual vs Fitted Values ,hence No problem orelse
```



Residuals vs Regressors

In [ ]:
```
fig=plt.figure(figsize=(15,8))
fig=sm.graphics.plot_regress_exog(model2,"YearsExperience",fig=fig)
plt.show()
```

```
eval_env: 1
```



Cook's Distance to Detect High Influencing points & Outliers

In [ ]:
```
model_influence=model2.get_influence()
```

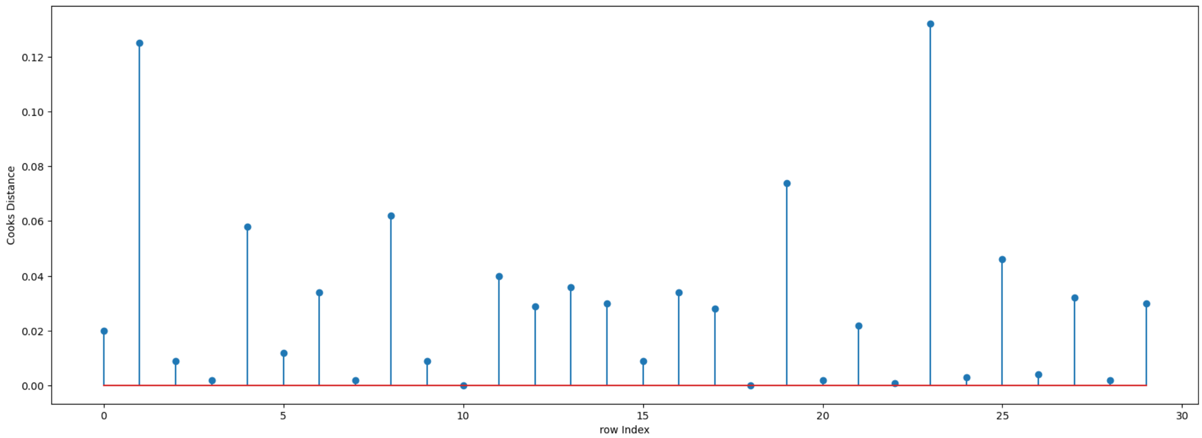
```python
(c2, _)= model_influence.cooks_distance
summary_cooks=model_influence.summary_frame()
summary_cooks.head()
```

Out[ ]:

	dfb_Intercept	dfb_YearsExperience	cooks_d	standard_resid	hat_diag	dffits_internal	student_resid
0	0.199964	-0.166721	0.020486	0.577707	0.109342	0.202416	0.570708
1	0.511627	-0.420188	0.125146	1.482031	0.102297	0.500291	1.515999
2	-0.129931	0.104974	0.008721	-0.406224	0.095595	-0.132069	-0.400085
3	-0.060669	0.046715	0.001931	-0.210257	0.080338	-0.062144	-0.206632
4	-0.340997	0.256662	0.058415	-1.201813	0.074835	-0.341805	-1.211826

In [ ]:

```python
fig=plt.subplots(figsize=(20,7))
plt.stem(np.arange(len(saldf)),np.round(c2,3))
plt.xlabel("row Index")
plt.ylabel("Cooks Distance")
plt.show()
```



In [ ]:

```python
(np.argmax(c2),np.max(c2)) # cooks distance threshold is 4/N so 4/30 = 0.1333 , hence da
```

Out[ ]:

```
(23, 0.13175452313135214)
```

In [ ]:

```python
(model2.rsquared_adj,model.rsquared_adj)
```

Out[ ]:

```
(0.9554194021486339, 0.9059250409151747)
```

MSE(Mean Squared Error), RMSE(Root Mean Squared Error), MAE(Mean Absolute Error), are the methods used to define loss function(actual-predicted values) , this measures error in our model , so that it give us to what extent the error rate is

here we are using RMSE which is the standard deviation of the Residuals(prediction errors), Residuals are a measure of how far from the regression line data points are . RMSE tells you how concentrated is the data around the BEST FIT LINE.

In [ ]:

```python
mse_m2= mean_squared_error(saldf.Salary,model2.fittedvalues) #checking for RMSE value
rmse_m2=math.sqrt(mse_m2)
print("the difference between actual and predicted values of model1 is :---",rmse_m2)
```

```
the difference between actual and predicted values of model1 is :--- 0.06605296017907701
```

In [ ]:

```python
print("RMSE-Model1---",rmse_m1,",", "RMSE-Model 2---",rmse_m2)
```

```
RMSE-Model1--- 0.08169965982966364 , RMSE-Model 2--- 0.06605296017907701
```

Based on a rule of thumb, it can be said that RMSE values between 0.2 and 0.5 shows that the model can relatively predict the data accurately.Lower values of RMSE indicate better fit. RMSE value should be Closer to ZERO '0'

Model2 is ready to predict with 95% accuracy as we got r-squared value as 0.955 and RMSE_M2 as 0.06

from sklearn.model_selection import train_test_split X=saldf.iloc[:,:-1] Y=saldf.iloc[:,-1:]
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.33,random_state=9)

from sklearn.linear_model import LinearRegression model2=LinearRegression()
model2.fit(x_train,y_train)

y_pred=model2.predict(x_test)

y_pred