Assignment-04-Simple Linear Regression-2

Salary_hike ->Build a prediction model for Salary_hike

Build a simple linear regression model by performing EDA and do necessary transformations and select the best model using R or Python.

Importing libraries

In [38]:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as smf
import statsmodels.formula.api as sm
import warnings
warnings.filterwarnings('ignore')
```

Step 1

Importing data

In [39]:
```
df = pd.read_csv('Salary_Data.csv')
df
```

Out[39]:

	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
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
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
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

Step 2

Performing EDA On Data

Checking Data Type

In [40]:
```
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
```

In [41]:
```
df.describe()
```

Out[41]:

	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 Null Values

In [42]:
```
df.isnull().sum()
```

Out[42]:
```
YearsExperience    0
Salary             0
dtype: int64
```

Checking for Duplicate Values

In [43]:
```
df[df.duplicated()].shape
```

Out[43]:
```
(0, 2)
```

In [44]:
```
df[df.duplicated()]
```

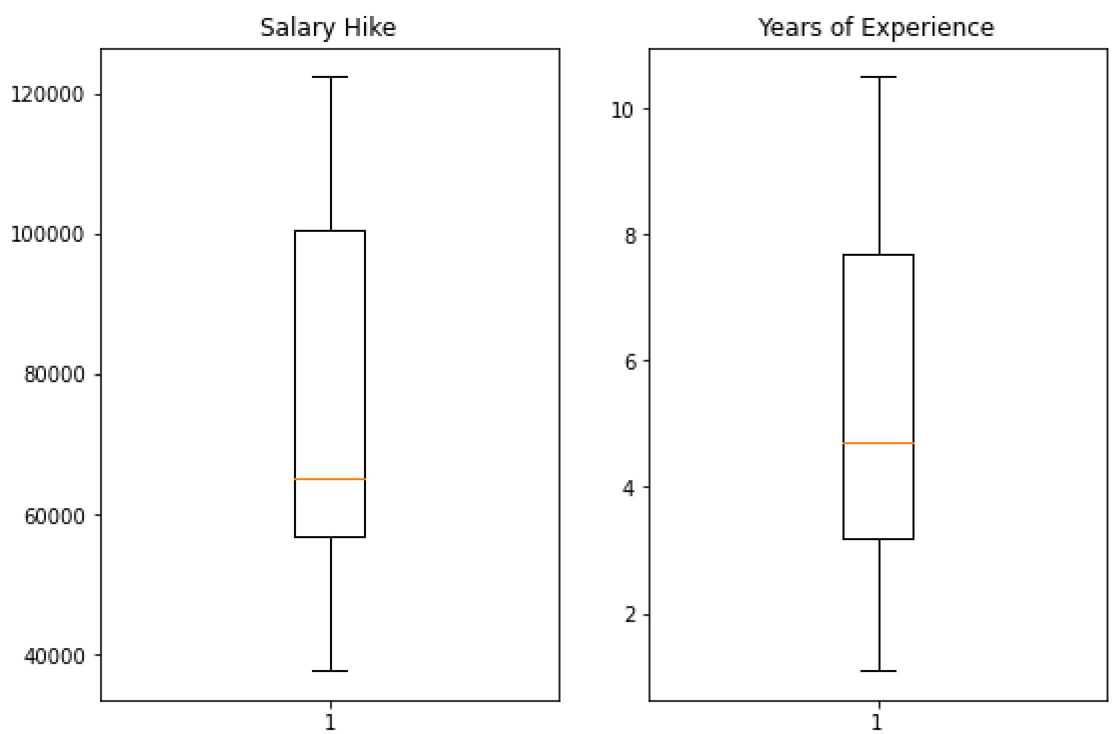
Out[44]:

YearsExperience	Salary

Step 3

Plotting the data to check for outliers

In [45]:
```
plt.subplots(figsize = (9,6))
plt.subplot(121)
plt.boxplot(df['Salary'])
plt.title('Salary Hike')
plt.subplot(122)
plt.boxplot(df['YearsExperience'])
plt.title('Years of Experience')
plt.show()
```



As you can see there are no Outliers in the data

Step 4

Checking the Correlation between variables

In [46]:
```
df.corr()
```

Out[46]:

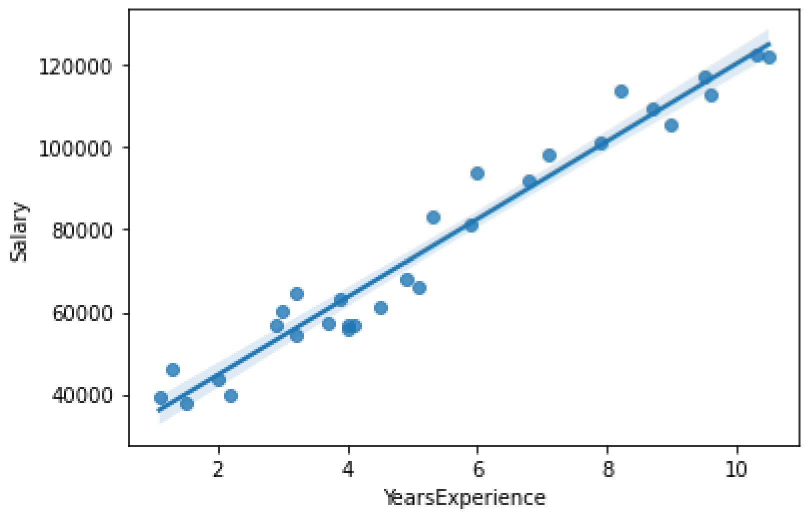
	YearsExperience	Salary
YearsExperience	1.000000	0.978242
Salary	0.978242	1.000000

Visualization of Correlation beteen x and y

regplot = regression plot

In [47]:
```
sns.regplot(x=df['YearsExperience'],y=df['Salary'])
```

Out[47]: <AxesSubplot:xlabel='YearsExperience', ylabel='Salary'>



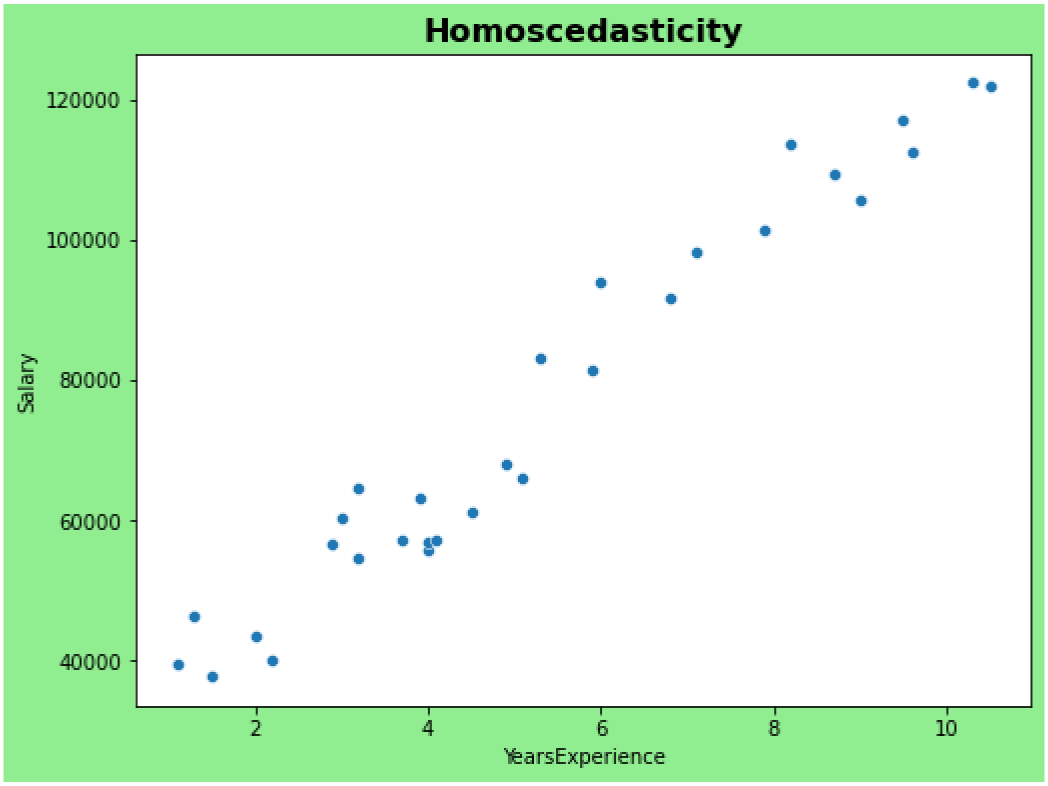
As you can see above

```
There is good correlation between the two variable.
The score is more than 0.8 which is a good sign
```

Step 5

Checking for Homoscedasticity or Hetroscedasticity

In [48]:
```
plt.figure(figsize = (8,6), facecolor = 'lightgreen')
sns.scatterplot(x = df['YearsExperience'], y = df['Salary'])
plt.title('Homoscedasticity', fontweight = 'bold', fontsize = 16)
plt.show()
```



As you can see in above graph

It shows as the Salary Increases the Years of Experience increases variation is ocnstant along the way in data
The data doesn't have any specific pattern in the variation. hence, we can say it's Homoscedasticity

In [49]:
```
df.var()
```

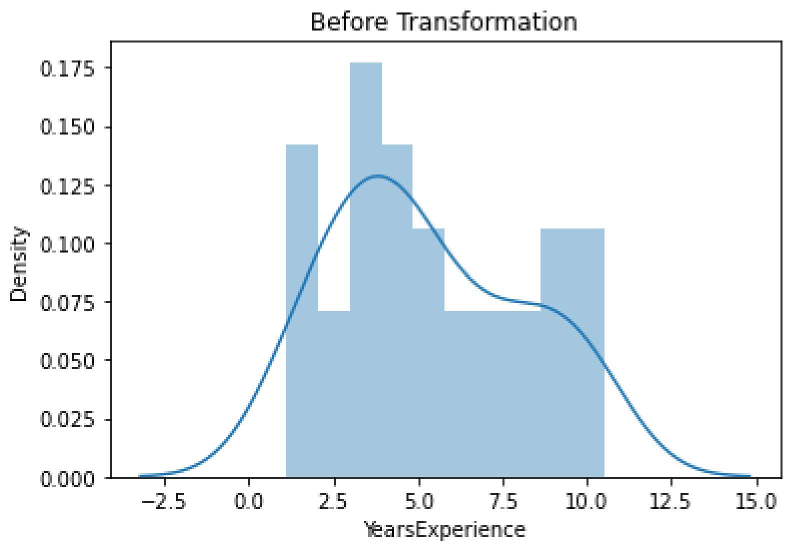
Out[49]:
```
YearsExperience    8.053609e+00
Salary             7.515510e+08
dtype: float64
```

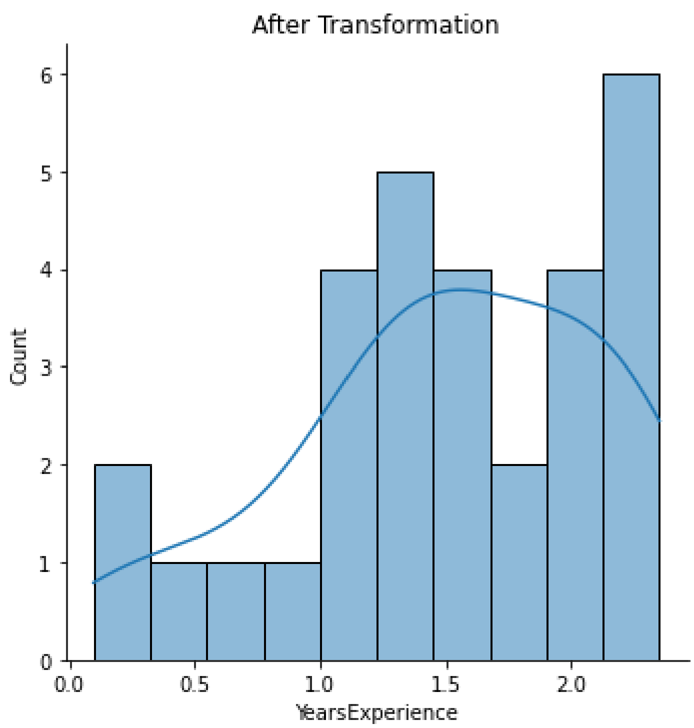
Step 6

Feature Engineering

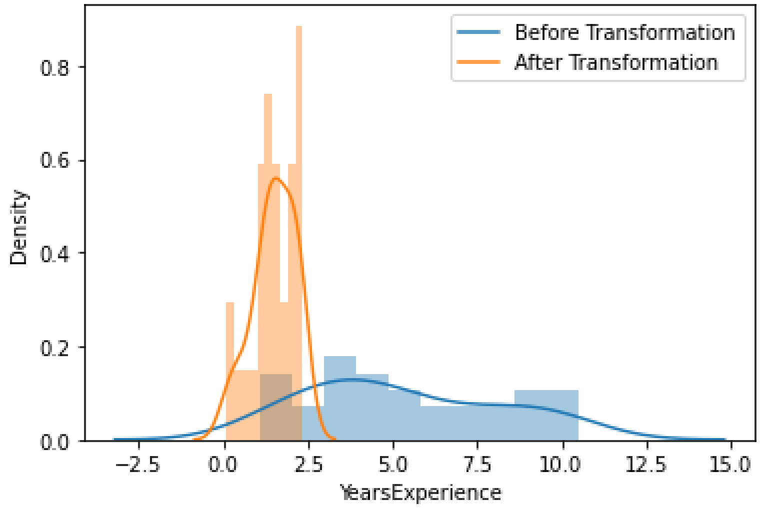
Trying different transformation of data to estimate normal distribution and remove any skewness

In [50]:
```
sns.distplot(df['YearsExperience'], bins = 10, kde = True)
plt.title('Before Transformation')
sns.displot(np.log(df['YearsExperience']), bins = 10, kde = True)
plt.title('After Transformation')
plt.show()
```





In [51]:
```
labels = ['Before Transformation','After Transformation']
sns.distplot(df['YearsExperience'], bins = 10, kde = True)
sns.distplot(np.log(df['YearsExperience']), bins = 10, kde = True)
plt.legend(labels)
plt.show()
```

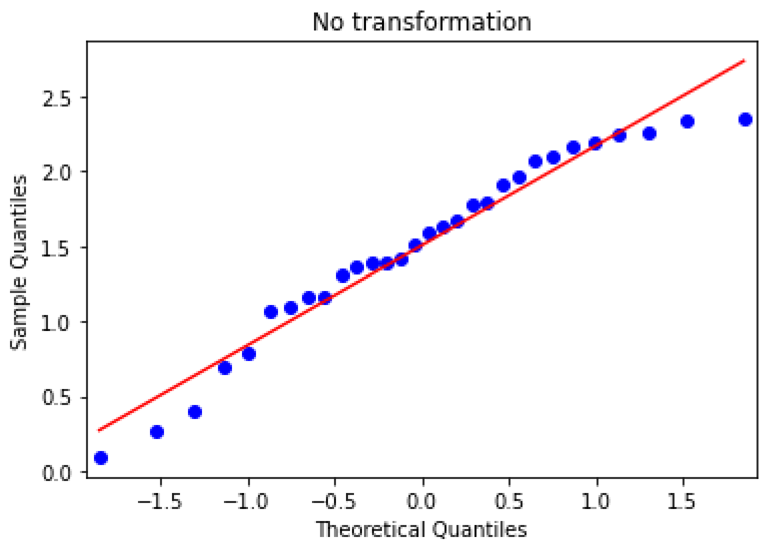


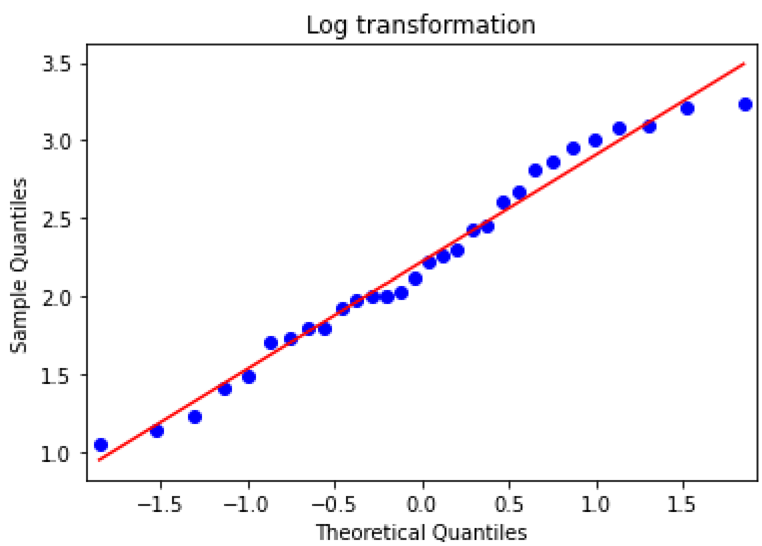
As you can see

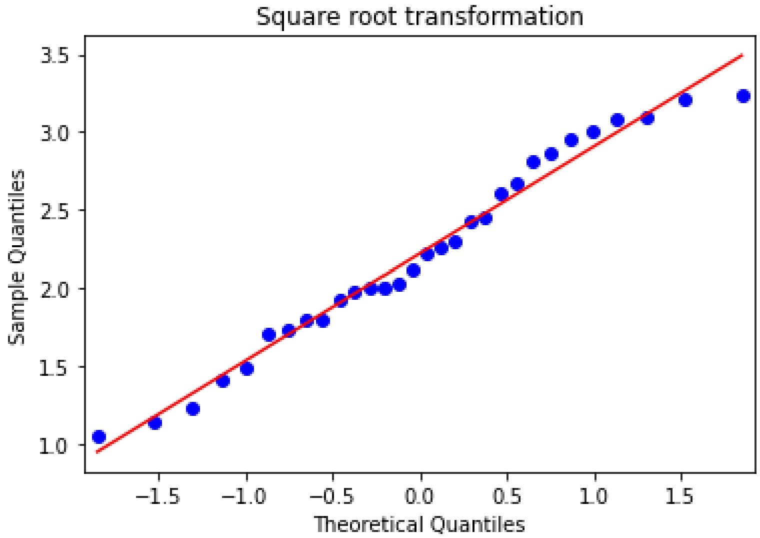
How log transformation affects the data and it scales the values down. Before prediction it is necessary to reverse scaled the values, even for calculating RMSE for the models.(Errors)

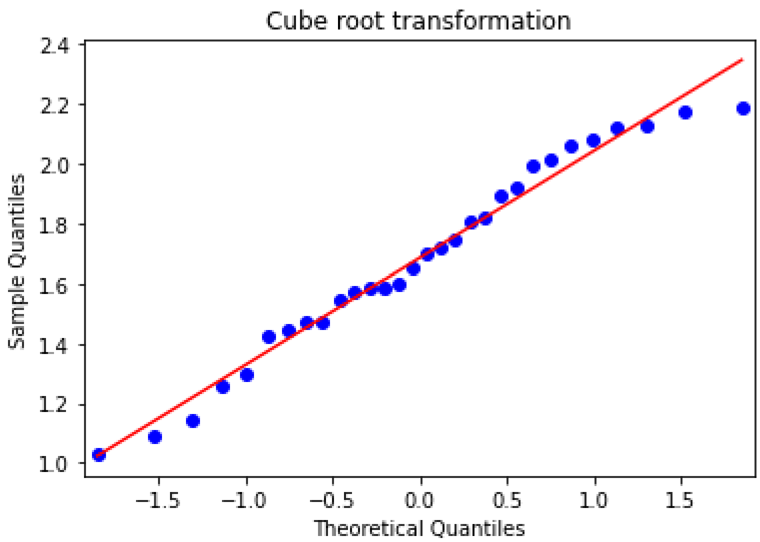
In [52]:

```
smf.qqplot(np.log(df['YearsExperience']), line = 'r')
plt.title('No transformation')
smf.qqplot(np.sqrt(df['YearsExperience']), line = 'r')
plt.title('Log transformation')
smf.qqplot(np.sqrt(df['YearsExperience']), line = 'r')
plt.title('Square root transformation')
smf.qqplot(np.cbrt(df['YearsExperience']), line = 'r')
plt.title('Cube root transformation')
plt.show()
```



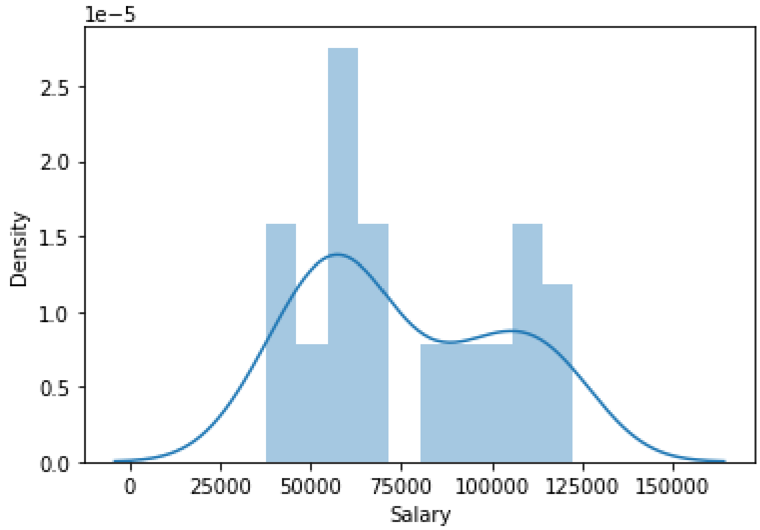


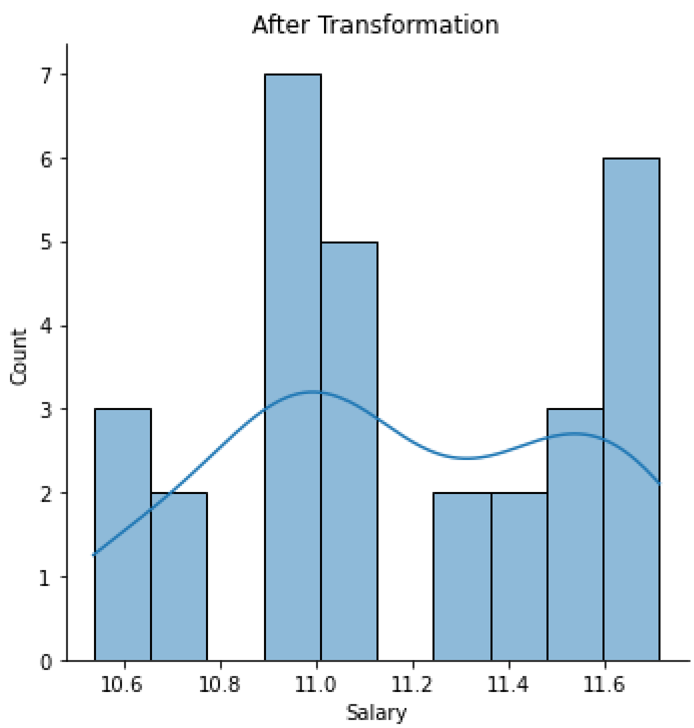

Square root transformation
Sample Quantiles
Theoretical Quantiles
3.5
3.0
2.5
2.0
1.5
1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Cube root transformation
Sample Quantiles
Theoretical Quantiles
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

In [53]:

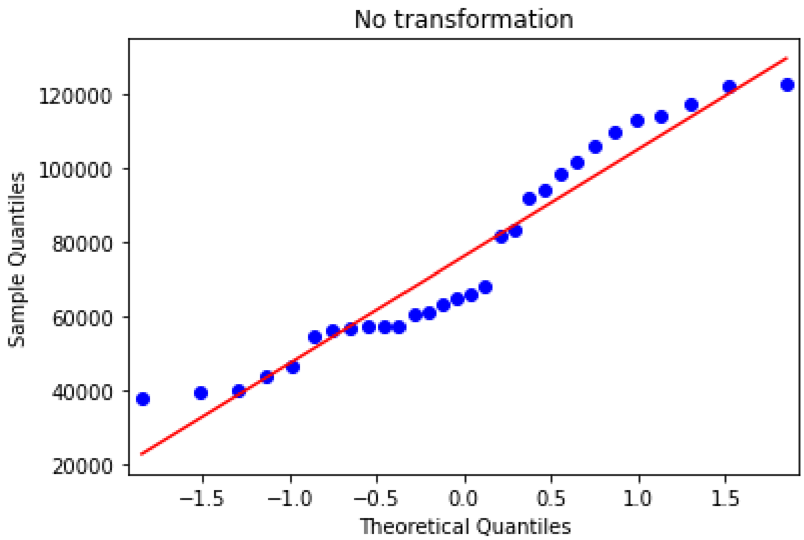
```
labels = ['Before Transformation','After Transformation']
sns.distplot(df['Salary'], bins = 10, kde = True)
sns.displot(np.log(df['Salary']), bins = 10, kde = True)
plt.title('After Transformation')
plt.show()
```

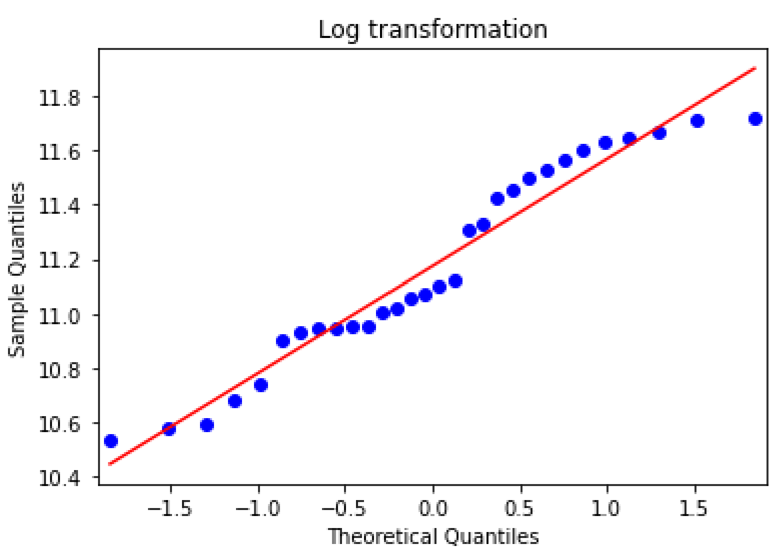


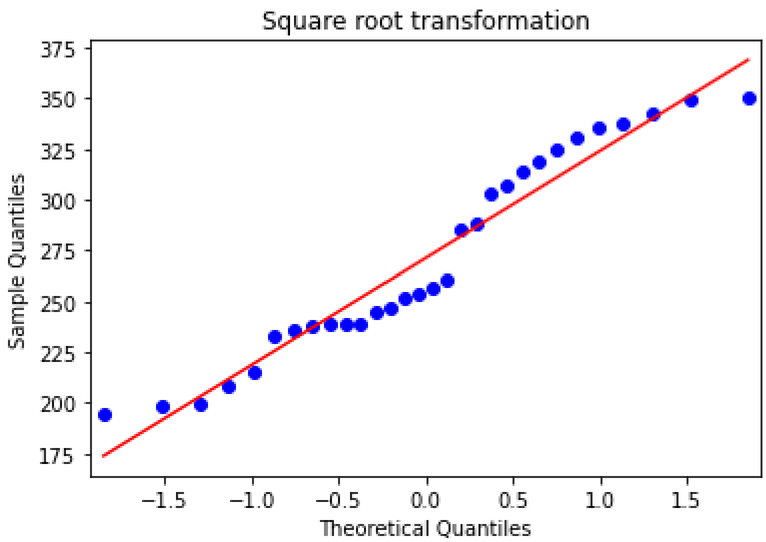


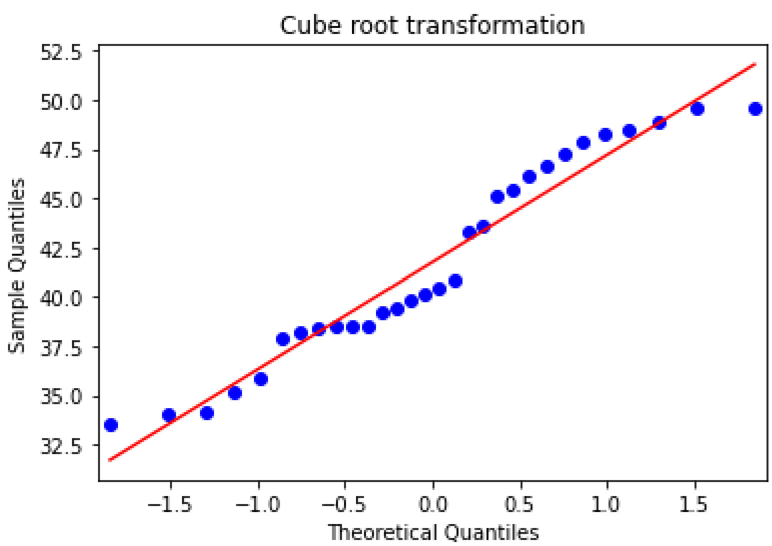
In [54]:

```
smf.qqplot(df['Salary'], line = 'r')
plt.title('No transformation')
smf.qqplot(np.log(df['Salary']), line = 'r')
plt.title('Log transformation')
smf.qqplot(np.sqrt(df['Salary']), line = 'r')
plt.title('Square root transformation')
smf.qqplot(np.cbrt(df['Salary']), line = 'r')
plt.title('Cube root transformation')
plt.show()
```






Square root transformation
Sample Quantiles
Theoretical Quantiles
375
350
325
300
275
250
225
200
175
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


Cube root transformation
Sample Quantiles
Theoretical Quantiles
52.5
50.0
47.5
45.0
42.5
40.0
37.5
35.0
32.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

Important Note:

```
We only Perform any data transformation when the data is skewed or not n
ormal distribution N(0,1)
```

Step 7

Fitting a Linear Regression Model

Using Ordinary least squares (OLS) regression

It is a statistical method of analysis that estimates the relationship between one or more independent variables and a dependent variable; the method estimates the relationship by minimizing the sum of the squares in the difference between the observed and predicted values of the dependent variable configured as a straight line

In [55]:
```
import statsmodels.formula.api as sm
model = sm.ols('Salary~YearsExperience', data = df).fit()
```

In [56]:
```
model.summary()
```

Out[56]:

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:	Wed, 08 Jun 2022	**Prob (F-statistic):**	1.14e-20
Time:	14:10:47	**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	3.04e+04
YearsExperience	9449.9623	378.755	24.950	0.000	8674.119	1.02e+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 correctly specified.

As you can notice in the above model

```
 The R-squared and Adjusted R-squared scores are above 0.85.
 (It is a thumb rule to consider Adjusted R-squared to be greater than
 0.8 for a good model for prediction)
 F-statitics is quite high as well and yes desire it to be higher
 But log-likelihood is quite very low far away from 0
 and AIC and BIC score are much higher for this model
 Lets Try some data transformation to check whether these scores can get
any better than this.
```

Square Root transformation on data

In [57]:

```
model1 = sm.ols('np.sqrt(Salary)~np.sqrt(YearsExperience)', data = df).fit()
model1.summary()
```

Out[57]:

OLS Regression Results

Dep. Variable:	np.sqrt(Salary)	**R-squared:**	0.942
Model:	OLS	**Adj. R-squared:**	0.940
Method:	Least Squares	**F-statistic:**	454.3
Date:	Wed, 08 Jun 2022	**Prob (F-statistic):**	7.58e-19
Time:	14:10:47	**Log-Likelihood:**	-116.52
No. Observations:	30	**AIC:**	237.0
Df Residuals:	28	**BIC:**	239.8
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	103.5680	8.178	12.663	0.000	86.815	120.321
np.sqrt(YearsExperience)	75.6269	3.548	21.315	0.000	68.359	82.895

Omnibus:	0.924	**Durbin-Watson:**	1.362
Prob(Omnibus):	0.630	**Jarque-Bera (JB):**	0.801
Skew:	0.087	**Prob(JB):**	0.670
Kurtosis:	2.219	**Cond. No.**	9.97

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

As you can notice in the above model

```
The R-squared and Adjusted R-squared scores are above 0.85. but its has
 gotten less than previous model
(It is a thumb rule to consider Adjusted R-squared to be greater than 0.
8 for a good model for prediction)
F-statitics has gotten a little lower for this model than previous.
But log-likelihood got better than before close to 0 higher than previou
s model
and AIC and BIC score are now much better for this model
Lets Try some data transformation to check whether these scores can get
 any better than this.
```

Cuberoot transformation on Data

In [58]:
```
model2 = sm.ols('np.cbrt(Salary)~np.cbrt(YearsExperience)', data = df).fit()
model2.summary()
```

Out[58]:

OLS Regression Results

Dep. Variable:	np.cbrt(Salary)	**R-squared:**	0.932
Model:	OLS	**Adj. R-squared:**	0.930
Method:	Least Squares	**F-statistic:**	386.5
Date:	Wed, 08 Jun 2022	**Prob (F-statistic):**	6.37e-18
Time:	14:10:48	**Log-Likelihood:**	-50.589
No. Observations:	30	**AIC:**	105.2
Df Residuals:	28	**BIC:**	108.0
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	16.6603	1.300	12.811	0.000	13.996	19.324
np.cbrt(YearsExperience)	14.8963	0.758	19.659	0.000	13.344	16.448

Omnibus:	0.386	**Durbin-Watson:**	1.229
Prob(Omnibus):	0.824	**Jarque-Bera (JB):**	0.535
Skew:	0.070	**Prob(JB):**	0.765
Kurtosis:	2.361	**Cond. No.**	12.0

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

Log transformation on Data

In [59]:
```
model3 = sm.ols('np.log(Salary)~np.log(YearsExperience)', data = df).fit()
model3.summary()
```

Out[59]:

OLS Regression Results

Dep. Variable:	np.log(Salary)	**R-squared:**	0.905
Model:	OLS	**Adj. R-squared:**	0.902
Method:	Least Squares	**F-statistic:**	267.4
Date:	Wed, 08 Jun 2022	**Prob (F-statistic):**	7.40e-16
Time:	14:10:48	**Log-Likelihood:**	23.209
No. Observations:	30	**AIC:**	-42.42
Df Residuals:	28	**BIC:**	-39.61
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	10.3280	0.056	184.868	0.000	10.214	10.442
np.log(YearsExperience)	0.5621	0.034	16.353	0.000	0.492	0.632

Omnibus:	0.102	**Durbin-Watson:**	0.988
Prob(Omnibus):	0.950	**Jarque-Bera (JB):**	0.297
Skew:	0.093	**Prob(JB):**	0.862
Kurtosis:	2.549	**Cond. No.**	5.76

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

Model Testing

As Y = Beta0 + Beta1*(X)

Finding Coefficient Parameters (Beta0 and Beta1 values)

In [60]:
```
model.params
```

Out[60]:
```
Intercept          25792.200199
YearsExperience     9449.962321
dtype: float64
```

Here, (Intercept) Beta0 value = 25792.20 & (YearsExperience) Beta1 value = 9449.96

Hypothesis testing of X variable by finding test_statistics and P_values for Beta1 i.e if (P_value < α=0.05 ; Reject Null)

Null Hypothesis as Beta1=0 (No Slope) and Alternate Hypthesis as Beta1≠0 (Some or significant Slope)

In [61]:
```
print(model.tvalues,'\n',model.pvalues)
```

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

(Intercept) Beta0: tvalue=11.34 , pvalue=5.511950e-12

(daily) Beta1: tvalue=24.95, pvalue= 1.143068e-20

As (pvalue=0)<(α=0.05); Reject Null hyp. Thus, X(YearsExperience) variable has good slope and variance w.r.t Y(Salary) variable.

R-squared measures the strength of the relationship between your model and the dependent variable on a 0 – 100% scale.

Measure goodness-of-fit by finding rsquared values (percentage of variance)

In [62]:
```
model.rsquared,model.rsquared_adj
```

Out[62]:
```
(0.9569566641435086, 0.9554194021486339)
```

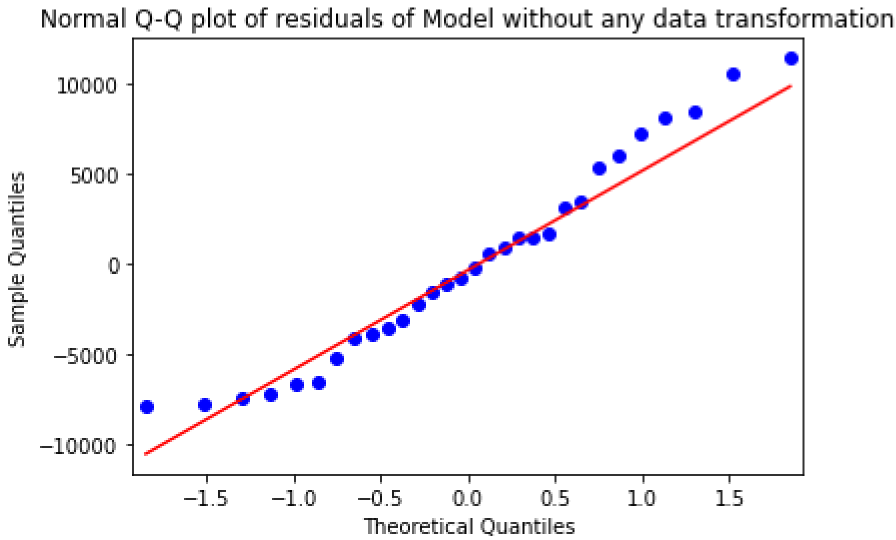
Determination Coefficient = rsquared value = 0.95 ; very good fit >= 85%

Step 8

Residual Analysis

Test for Normality of Residuals (Q-Q Plot)

In [63]:
```
import statsmodels.api as sm
sm.qqplot(model.resid, line = 'q')
plt.title('Normal Q-Q plot of residuals of Model without any data transformat
plt.show()
```



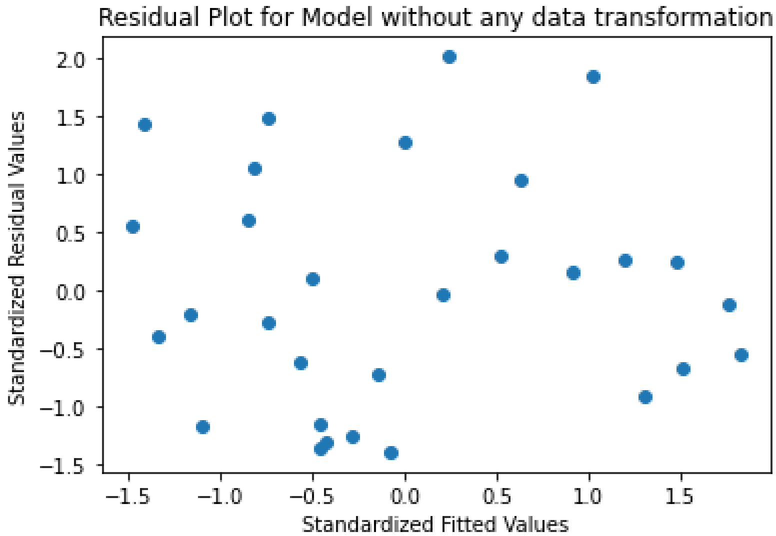
As you can notice in the above plot

    The first model follows normal distribution

Residual Plot to check Homoscedasticity or Hetroscedasticity

In [64]:
```
def get_standardized_values( vals ):
    return (vals - vals.mean())/vals.std()
```

In [65]:
```
plt.scatter(get_standardized_values(model.fittedvalues), get_standardized_val
plt.title('Residual Plot for Model without any data transformation')
plt.xlabel('Standardized Fitted Values')
plt.ylabel('Standardized Residual Values')
plt.show()
```



As you can notice in the above plots

```
The Model have Homoscedasciticity.
The Residual(i.e Residual = Actual Value - Predicted Value) and the Fitt
ed values do not share any Pattern.
Hence, there is no relation between the Residual and the Fitted Value. I
t is Randomly distributed
```

Step 9

Model Validation

We will analyze Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) — AKA the average distance (squared to get rid of negative numbers) between the model’s predicted target value and the actual target value.

Comparing different models with respect to the Root Mean Squared Errors

In [66]:
```
from sklearn.metrics import mean_squared_error
```

In [67]:
```
model1_pred_y =np.square(model1.predict(df['YearsExperience']))
model2_pred_y =pow(model2.predict(df['YearsExperience']),3)
model3_pred_y =np.exp(model3.predict(df['YearsExperience']))
```

In [68]:
```
model1_rmse =np.sqrt(mean_squared_error(df['Salary'], model1_pred_y))
model2_rmse =np.sqrt(mean_squared_error(df['Salary'], model2_pred_y))
model3_rmse =np.sqrt(mean_squared_error(df['Salary'], model3_pred_y))
print('model=', np.sqrt(model.mse_resid),'\n' 'model1=', model1_rmse,'\n' 'mo
```

```
model= 5788.315051119395
model1= 5960.647096174311
model2= 6232.8154558358565
model3= 7219.716974372802
```

In [69]:
```
rmse = {'model': np.sqrt(model.mse_resid), 'model1': model1_rmse, 'model2': m
min(rmse, key=rmse.get)
```

Out[69]:
```
'model'
```

As model has the minimum RMSE and highest Adjusted R-squared score. Hence, we are going to use model to predict our values

Model is that Simple Linear regression model where we did not perfrom any data transformation and got the highest Adjusted R-squared value

Step 10

Predicting values

In [72]:

```
# first model results without any transformation

predicted = pd.DataFrame()
predicted['YearsExperience'] =  df.YearsExperience
predicted['Salary'] = df.Salary
predicted['Predicted_Salary_Hike'] =  pd.DataFrame(model.predict(predicted.Ye
predicted
#.........................................................
```

Out[72]:

	YearsExperience	Salary	Predicted_Salary_Hike
0	1.1	39343.0	36187.158752
1	1.3	46205.0	38077.151217
2	1.5	37731.0	39967.143681
3	2.0	43525.0	44692.124842
4	2.2	39891.0	46582.117306
5	2.9	56642.0	53197.090931
6	3.0	60150.0	54142.087163
7	3.2	54445.0	56032.079627
8	3.2	64445.0	56032.079627
9	3.7	57189.0	60757.060788
10	3.9	63218.0	62647.053252
11	4.0	55794.0	63592.049484
12	4.0	56957.0	63592.049484
13	4.1	57081.0	64537.045717
14	4.5	61111.0	68317.030645
15	4.9	67938.0	72097.015574
16	5.1	66029.0	73987.008038
17	5.3	83088.0	75877.000502
18	5.9	81363.0	81546.977895
19	6.0	93940.0	82491.974127
20	6.8	91738.0	90051.943985
21	7.1	98273.0	92886.932681
22	7.9	101302.0	100446.902538
23	8.2	113812.0	103281.891235
24	8.7	109431.0	108006.872395
25	9.0	105582.0	110841.861092
26	9.5	116969.0	115566.842252
27	9.6	112635.0	116511.838485

	YearsExperience	Salary	Predicted_Salary_Hike
28	10.3	122391.0	123126.812110
29	10.5	121872.0	125016.804574

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: