Problem Statement: -

The head of HR of a certain organization wants to automate their salary hike estimation. The organization consulted an analytics service provider and asked them to build a basic prediction model by providing them with a dataset that contains the data about the number of years of experience and the salary hike given accordingly. Build a Simple Linear Regression model with salary as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.

In [34]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv("D:\\360Digi\Simple Resgression Ass\\Salary_Data.csv")
df.head()
```

Out[34]:

	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 [35]:
```
df.isnull().sum()
```

Out[35]:
```
YearsExperience    0
Salary             0
dtype: int64
```

EDA

In [21]:

```
Eda = {"columns": df.columns,
       "mean": df.mean(),
       "median":df.median(),
       "strdrand deviation":df.std(),
       "variance": df.var(),
       "kurtosis": df.kurt()
       }
Eda
```

Out[21]:

```
{'columns': Index(['YearsExperience', 'Salary'], dtype='object'),
 'mean': YearsExperience        5.313333
 Salary             76003.000000
 dtype: float64,
 'median': YearsExperience        4.7
 Salary             65237.0
 dtype: float64,
 'strdrand deviation': YearsExperience        2.837888
 Salary             27414.429785
 dtype: float64,
 'variance': YearsExperience    8.053609e+00
 Salary             7.515510e+08
 dtype: float64,
 'kurtosis': YearsExperience   -1.012212
 Salary            -1.295421
 dtype: float64}
```

In [22]:

```
''''
df.columns.values[0] = "Ye"
df.columns.values[1] = "Sal"
df.columns
''''
```

Out[22]:

```
Index(['Ye', 'Sal'], dtype='object')
```

In [36]:

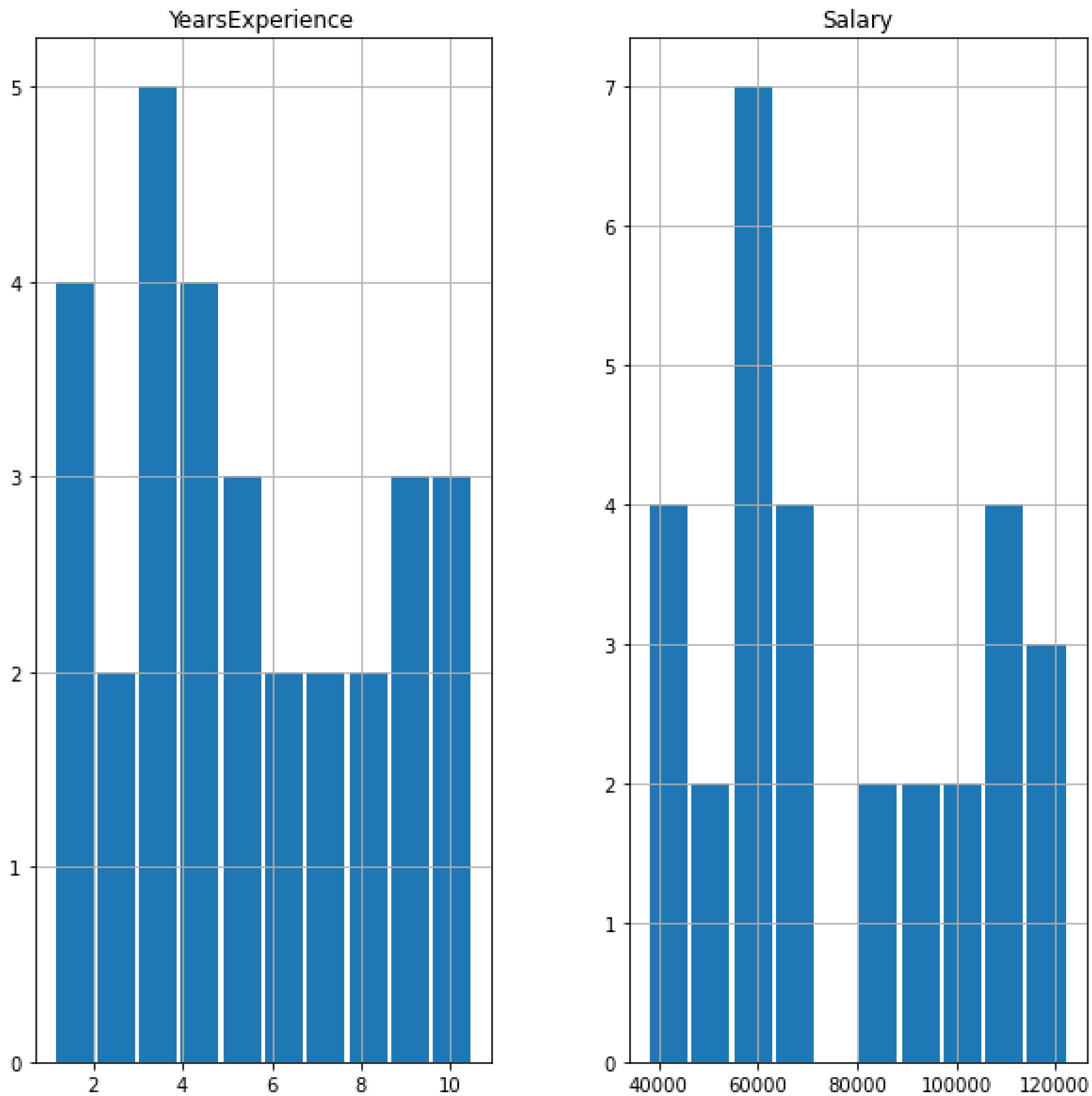
```
df.head()
```

Out[36]:

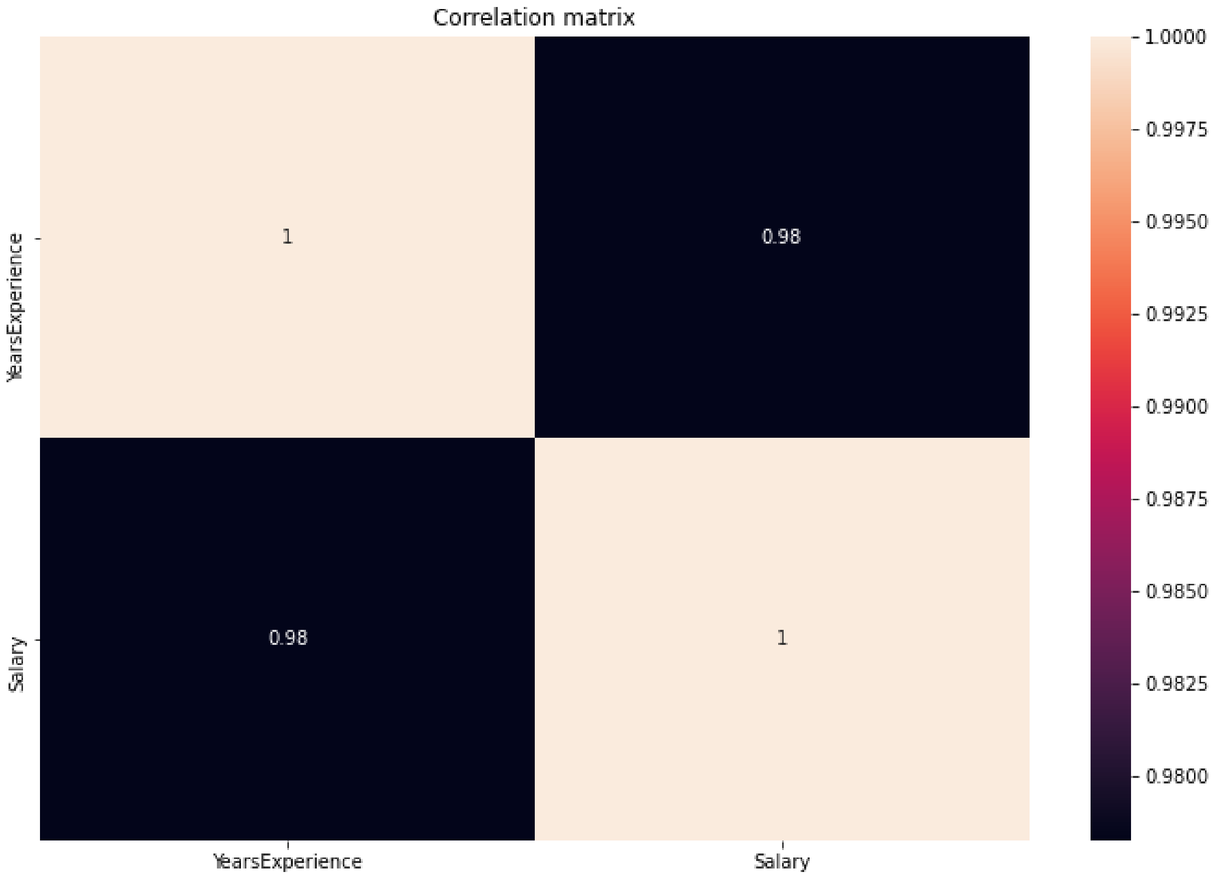
	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

In [37]:
```
df.hist(grid=True, rwidth=0.9, figsize=(10,10))
```

Out[37]:
```
array([[<AxesSubplot:title={'center':'YearsExperience'}>,
        <AxesSubplot:title={'center':'Salary'}>]], dtype=object)
```



In [38]:
```
plt.figure(figsize = (12, 8))
sns.heatmap(df.corr(), annot = True)
plt.title('Correlation matrix')
plt.show()
```



In [39]:

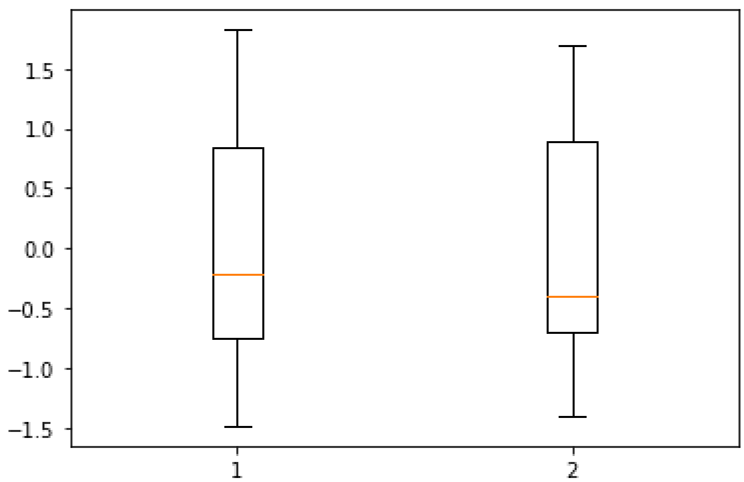
```
# Normalization function using z std. all are continuous data.
def std_func(i):
    x = (i-i.mean())/(i.std())
    return (x)

# Normalized data frame (considering the numerical part of data)
cal = std_func(df)
cal.describe()
```

Out[39]:

	YearsExperience	Salary
count	3.000000e+01	30.000000
mean	-1.480297e-17	0.000000
std	1.000000e+00	1.000000
min	-1.484672e+00	-1.396053
25%	-7.446852e-01	-0.703361
50%	-2.161232e-01	-0.392713
75%	8.410010e-01	0.895213
max	1.827650e+00	1.692102

In [40]:
```
plt.boxplot(cal) #boxplot
```

Out[40]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x20c8a0b45b0>,
  <matplotlib.lines.Line2D at 0x20c8a0b4910>,
  <matplotlib.lines.Line2D at 0x20c8a0c2d90>,
  <matplotlib.lines.Line2D at 0x20c8a0cc130>],
 'caps': [<matplotlib.lines.Line2D at 0x20c8a0b4c70>,
  <matplotlib.lines.Line2D at 0x20c8a0b4fd0>,
  <matplotlib.lines.Line2D at 0x20c8a0cc490>,
  <matplotlib.lines.Line2D at 0x20c8a0cc7f0>],
 'boxes': [<matplotlib.lines.Line2D at 0x20c8a0b42e0>,
  <matplotlib.lines.Line2D at 0x20c8a0c2a30>],
 'medians': [<matplotlib.lines.Line2D at 0x20c8a0c2370>,
  <matplotlib.lines.Line2D at 0x20c8a0ccb50>],
 'fliers': [<matplotlib.lines.Line2D at 0x20c8a0c26d0>,
  <matplotlib.lines.Line2D at 0x20c8a0cceb0>],
 'means': []}
```

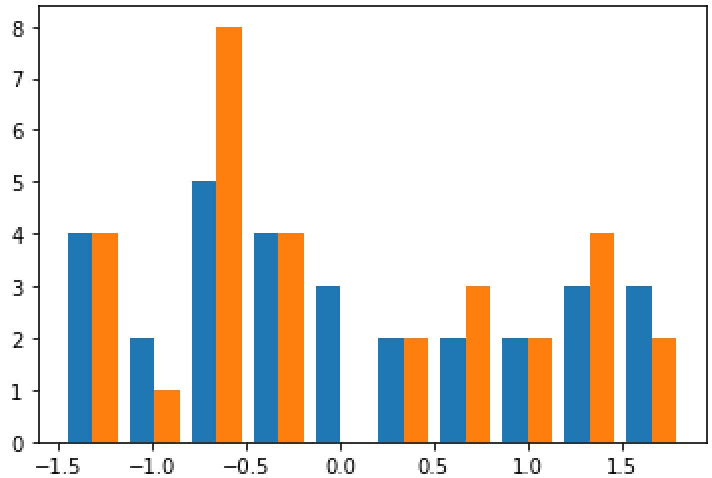


In [41]:

```
plt.hist(cal) #histogram
```

Out[41]:

```
(array([[4., 2., 5., 4., 3., 2., 2., 2., 3., 3.],
        [4., 1., 8., 4., 0., 2., 3., 2., 4., 2.]]),
 array([-1.48467209, -1.15343986, -0.82220764, -0.49097542, -0.1597432 ,
         0.17148902,  0.50272124,  0.83395347,  1.16518569,  1.49641791,
         1.82765013]),
 <a list of 2 BarContainer objects>)
```



In [29]:

```
# Scatter plot
#plt.scatter(x = cal.Ye, y = cal.Sal, color = 'green')
```

In [30]:
```
# correlation
np.corrcoef(cal)
```

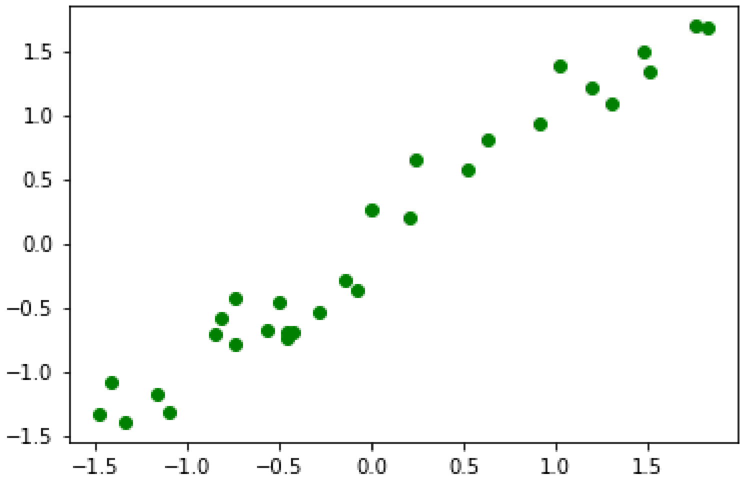
Out[30]:
```
array([[ 1.,  1., -1., -1., -1.,  1.,  1., -1.,  1., -1.,  1., -1., -1.,
        -1., -1., -1., -1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,
         1., -1., -1., -1.],
       [ 1.,  1., -1., -1., -1.,  1.,  1., -1.,  1., -1.,  1., -1., -1.,
        -1., -1., -1., -1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,
         1., -1., -1., -1.],
       [-1., -1.,  1.,  1.,  1., -1., -1.,  1., -1.,  1., -1.,  1.,  1.,
         1.,  1.,  1.,  1., -1.,  1., -1., -1., -1., -1., -1., -1.,  1.,
        -1.,  1.,  1.,  1.],
       [-1., -1.,  1.,  1.,  1., -1., -1.,  1., -1.,  1., -1.,  1.,  1.,
         1.,  1.,  1.,  1., -1.,  1., -1., -1., -1., -1., -1., -1.,  1.,
        -1.,  1.,  1.,  1.],
       [-1., -1.,  1.,  1.,  1., -1., -1.,  1., -1.,  1., -1.,  1.,  1.,
         1.,  1.,  1.,  1., -1.,  1., -1., -1., -1., -1., -1., -1.,  1.,
        -1.,  1.,  1.,  1.],
       [ 1.,  1., -1., -1., -1.,  1.,  1., -1.,  1., -1.,  1., -1., -1.,
        -1., -1., -1., -1.,  1., -1.,  1.,  1.,  1.,  1.,  1.,  1., -1.,
         1., -1., -1., -1.],
       [ 1.,  1., -1., -1., -1.,  1.,  1., -1.,  1., -1.,  1., -1., -1.,
```

In [43]:
```
# Covariance
# NumPy does not have a function to calculate the covariance between two variable
# Function for calculating a covariance matrix called cov()
# By default, the cov() function will calculate the unbiased or sample covariance

cov_output = np.cov(cal.YearsExperience, cal.Salary)[0, 1]
cov_output
```

Out[43]: 0.9782416184887599

In [68]:
```
# Scatter plot
plt.scatter(x = cal.YearsExperience, y = cal.Salary, color = 'green')
```

Out[68]: <matplotlib.collections.PathCollection at 0x20c8bf212e0>



In [44]:
```
cal.columns
```

Out[44]: Index(['YearsExperience', 'Salary'], dtype='object')

Data Modeling

In [45]:

```
# Import library
import statsmodels.formula.api as smf

# Simple Linear Regression
model = smf.ols('Salary ~ YearsExperience', data = cal).fit()
model.summary()
```

Out[45]:

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:	Sat, 19 Jun 2021	**Prob (F-statistic):**	1.14e-20
Time:	08:57:05	**Log-Likelihood:**	5.1236
No. Observations:	30	**AIC:**	-6.247
Df Residuals:	28	**BIC:**	-3.445
Df Model:	1		
Covariance Type:	nonrobust		

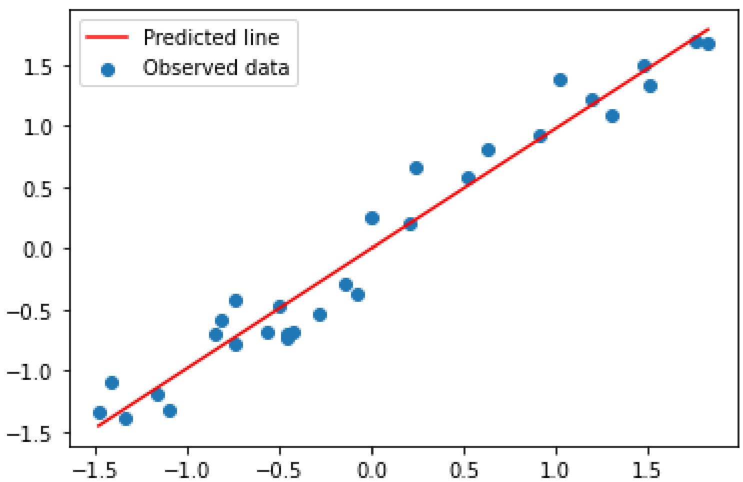
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.388e-17	0.039	3.6e-16	1.000	-0.079	0.079
YearsExperience	0.9782	0.039	24.950	0.000	0.898	1.059

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.**	1.02

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

In [48]:

```
pred1 = model.predict(pd.DataFrame(cal.YearsExperience))

# Regression Line
plt.scatter(cal.YearsExperience, cal.Salary)
plt.plot(cal.YearsExperience, pred1, "r")
plt.legend(['Predicted line', 'Observed data'])
plt.show()

# Error calculation
res1 = cal.Salary - pred1
res_sqr1 = res1 * res1
mse1 = np.mean(res_sqr1)
rmse1 = np.sqrt(mse1)
rmse1
```



Out[48]: 0.20398175897517998

In [51]:

```
######### Model building on Transformed Data
# Log Transformation
# x = log(waist); y = at

plt.scatter(x = np.log(cal.YearsExperience), y = cal.Salary, color = 'brown')
np.corrcoef(np.log(cal.YearsExperience), cal.Salary) #correlation

model2 = smf.ols('Salary ~ np.log(YearsExperience)', data = cal).fit()
model2.summary()
```

```
D:\anconda\lib\site-packages\scipy\stats\stats.py:1603: UserWarning: kurtosiste
st only valid for n>=20 ... continuing anyway, n=12
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[51]:

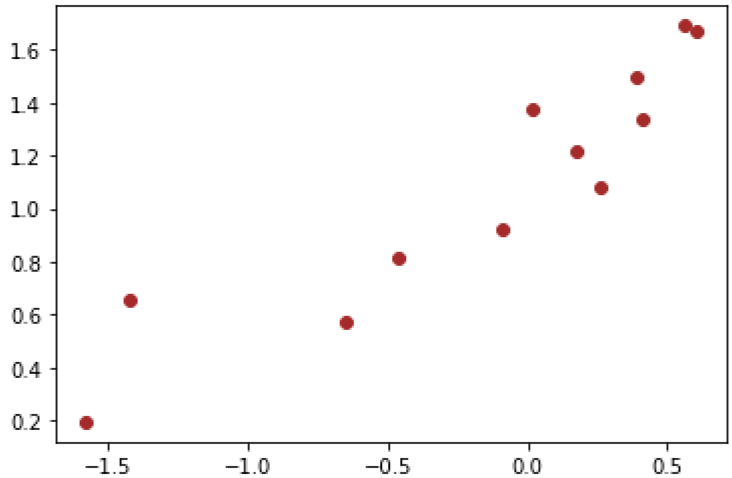
OLS Regression Results

Dep. Variable:	Salary	**R-squared:**	0.845
Model:	OLS	**Adj. R-squared:**	0.829
Method:	Least Squares	**F-statistic:**	54.47
Date:	Sat, 19 Jun 2021	**Prob (F-statistic):**	2.37e-05
Time:	09:00:39	**Log-Likelihood:**	3.8404
No. Observations:	12	**AIC:**	-3.681
Df Residuals:	10	**BIC:**	-2.711
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.1719	0.057	20.644	0.000	1.045	1.298
np.log(YearsExperience)	0.5809	0.079	7.380	0.000	0.406	0.756

Omnibus:	1.673	**Durbin-Watson:**	2.474
Prob(Omnibus):	0.433	**Jarque-Bera (JB):**	0.870
Skew:	0.205	**Prob(JB):**	0.647
Kurtosis:	1.746	**Cond. No.**	1.48

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

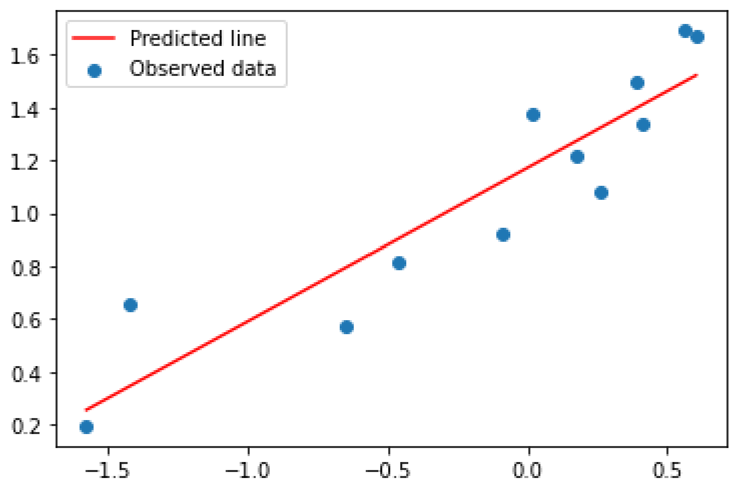

1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
−1.5
−1.0
−0.5
0.0
0.5

In [52]:

```
pred2 = model2.predict(pd.DataFrame(cal.YearsExperience))

# Regression Line
plt.scatter(np.log(cal.YearsExperience), cal.Salary)
plt.plot(np.log(cal.YearsExperience), pred2, "r")
plt.legend(['Predicted line', 'Observed data'])
plt.show()

# Error calculation
res2 = cal.Salary - pred2
res_sqr2 = res2 * res2
mse2 = np.mean(res_sqr2)
rmse2 = np.sqrt(mse2)
rmse2

'''
#### Exponential transformation
# x = waist; y = log(at)

plt.scatter(x = cal.Ye, y = np.log(cal.Sal), color = 'orange')
np.corrcoef(cal.Ye, np.log(cal.Sal)) #correlation

model3 = smf.ols('np.log(Sal) ~ Ye', data = cal).fit()
model3.summary()

pred3 = model3.predict(pd.DataFrame(cal.SH))
pred3_at = np.exp(pred3)
pred3_at

# Regression Line
plt.scatter(cal.SH, np.log(cal.CC))
plt.plot(cal.SH, pred3, "r")
plt.legend(['Predicted line', 'Observed data'])
plt.show()

# Error calculation
res3 = cal.CC - pred3_at
res_sqr3 = res3 * res3
mse3 = np.mean(res_sqr3)
rmse3 = np.sqrt(mse3)
rmse3

#### Polynomial transformation
# x = waist; x^2 = waist*waist; y = log(at)

model4 = smf.ols('np.log(DT) ~ ST + I(ST*ST)', data = cal).fit()
model4.summary()

pred4 = model4.predict(pd.DataFrame(cal.ST))
pred4_at = np.exp(pred4)
pred4_at

# Regression line
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 2)
```

```
X = cal.iloc[:, 1:].values
X_poly = poly_reg.fit_transform(X)
# y = wcat.iloc[:, 1].values


plt.scatter(cal.ST, np.log(cal.DT))
plt.plot(X, pred4, color = 'red')
plt.legend(['Predicted line', 'Observed data'])
plt.show()

# Error calculation
res4 = cal.DT - pred4_at
res_sqr4 = res4 * res4
mse4 = np.mean(res_sqr4)
rmse4 = np.sqrt(mse4)
rmse4
'''
```

```
D:\anconda\lib\site-packages\pandas\core\arraylike.py:358: RuntimeWarning: inva
lid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



Out[52]:

```
'\n#### Exponential transformation\n# x = waist; y = log(at)\n\nplt.scatter
(x = cal.Ye, y = np.log(cal.Sal), color = \'orange\')\nnp.corrcoef(cal.Ye, n
p.log(cal.Sal)) #correlation\n\nmodel3 = smf.ols(\'np.log(Sal) ~ Ye\', data
= cal).fit()\nmodel3.summary()\n\npred3 = model3.predict(pd.DataFrame(cal.S
H))\npred3_at = np.exp(pred3)\npred3_at\n\n# Regression Line\nplt.scatter(ca
l.SH, np.log(cal.CC))\nplt.plot(cal.SH, pred3, "r")\nplt.legend([\'Predicted
line\', \'Observed data\'])\nplt.show()\n\n# Error calculation\nres3 = cal.C
C - pred3_at\nres_sqr3 = res3 * res3\nmse3 = np.mean(res_sqr3)\nrmse3 = np.s
qrt(mse3)\nrmse3\n\n#### Polynomial transformation\n# x = waist; x^2 = waist
*waist; y = log(at)\n\nmodel4 = smf.ols(\'np.log(DT) ~ ST + I(ST*ST)\', data
= cal).fit()\nmodel4.summary()\n\npred4 = model4.predict(pd.DataFrame(cal.S
T))\npred4_at = np.exp(pred4)\npred4_at\n\n# Regression line\nfrom sklearn.p
reprocessing import PolynomialFeatures\npoly_reg = PolynomialFeatures(degree
= 2)\nX = cal.iloc[:, 1:].values\nX_poly = poly_reg.fit_transform(X)\n# y =
wcat.iloc[:, 1].values\n\n\nplt.scatter(cal.ST, np.log(cal.DT))\nplt.plot(X,
pred4, color = \'red\')\nplt.legend([\'Predicted line\', \'Observed data\'])
```

```
\nplt.show()\n\n# Error calculation\nres4 = cal.DT - pred4_at\nres_sqr4 = re
s4 * res4\nmse4 = np.mean(res_sqr4)\nrmse4 = np.sqrt(mse4)\nrmse4\n'
```

In [54]:

```
# Choose the best model using RMSE
data = {"MODEL":pd.Series(["SLR","Log"]), "RMSE":pd.Series([rmse1,rmse2])}
table_rmse = pd.DataFrame(data)
table_rmse
```

Out[54]:

	MODEL	RMSE
0	SLR	0.203982
1	Log	0.175701

In [65]:

```
###################
# The best model

from sklearn.model_selection import train_test_split

train, test = train_test_split(cal, test_size = 0.3)

finalmodel = smf.ols('Salary ~ np.log(YearsExperience)', data = train).fit()
finalmodel.summary()
```

```
D:\anconda\lib\site-packages\pandas\core\arraylike.py:358: RuntimeWarning: inva
lid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
D:\anconda\lib\site-packages\scipy\stats\stats.py:1603: UserWarning: kurtosiste
st only valid for n>=20 ... continuing anyway, n=8
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[65]:

OLS Regression Results

Dep. Variable:	Salary	**R-squared:**	0.943
Model:	OLS	**Adj. R-squared:**	0.934
Method:	Least Squares	**F-statistic:**	100.1
Date:	Sat, 19 Jun 2021	**Prob (F-statistic):**	5.78e-05
Time:	09:10:32	**Log-Likelihood:**	5.6448
No. Observations:	8	**AIC:**	-7.290
Df Residuals:	6	**BIC:**	-7.131
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.1533	0.049	23.631	0.000	1.034	1.273
np.log(YearsExperience)	0.6951	0.069	10.003	0.000	0.525	0.865

Omnibus:	3.751	**Durbin-Watson:**	1.018
Prob(Omnibus):	0.153	**Jarque-Bera (JB):**	0.964
Skew:	-0.065	**Prob(JB):**	0.618
Kurtosis:	1.305	**Cond. No.**	1.43

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

In [66]:

```
# Predict on test data
test_pred = finalmodel.predict(pd.DataFrame(test))
pred_test_AT = np.exp(test_pred)
pred_test_AT

# Model Evaluation on Test data
test_res = test.Salary - pred_test_AT
test_sqrs = test_res * test_res
test_mse = np.mean(test_sqrs)
test_rmse = np.sqrt(test_mse)
test_rmse
```

```
D:\anconda\lib\site-packages\pandas\core\arraylike.py:358: RuntimeWarning: inva
lid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[66]: 1.8185461734088717

In [67]:

```
# Prediction on train data
train_pred = finalmodel.predict(pd.DataFrame(train))
pred_train_AT = np.exp(train_pred)
pred_train_AT


# Model Evaluation on train data
train_res = train.Salary - pred_train_AT
train_sqrs = train_res * train_res
train_mse = np.mean(train_sqrs)
train_rmse = np.sqrt(train_mse)
train_rmse




# Model having highest R-Squared value is better i.e. (model=0.84 is better than
```

Out[67]: 2.4206644021338337

Summary

Model having highest R-Squared value is better. There has good relationship>0.85

RMSE- lower the RMSE incidcate better fit. RMSE is a good measure of how accuaracy the model predict the reponse. In Linear regression RMSE value between 0.2 to 0.5

But in final model trainig and traing we choose SLR Salary ~ YearsExperience beacause the it is the best fitted model and the RSME value is low.