In [1]:
```
# Necessary imports
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv('Salary_Data.csv')  #dataset
df.head(7)
```

Out[2]:

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150

EDA

In [3]:
```
df.shape                                  #shape of dataset
```

Out[3]:
```
(30, 2)
```

In [4]:
```
df.info()                                 #no null value in dataset
```

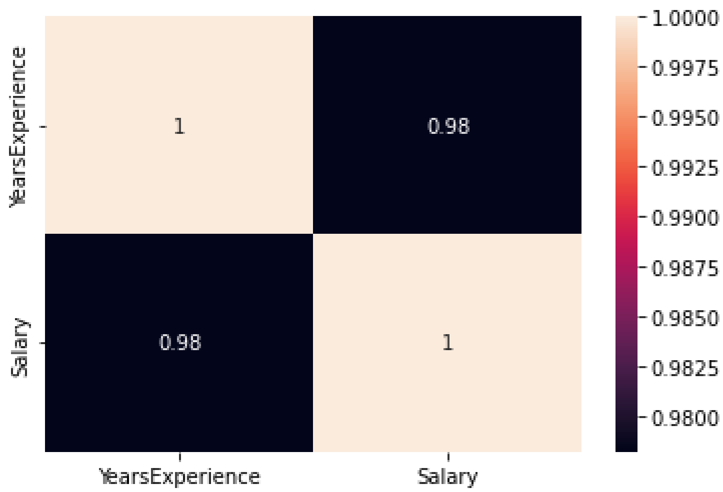
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64
dtypes: float64(1), int64(1)
memory usage: 608.0 bytes
```

In [5]:
```
df.skew()                                  #skewness of Feature and Target
```

Out[5]:
```
YearsExperience    0.37956
Salary             0.35412
dtype: float64
```

In [6]:
```
#Co-relation of Target and Feature is close to 1
sns.heatmap(df.corr(), annot = True)
```

Out[6]: <AxesSubplot:>



In [7]:
```
df.describe()
```

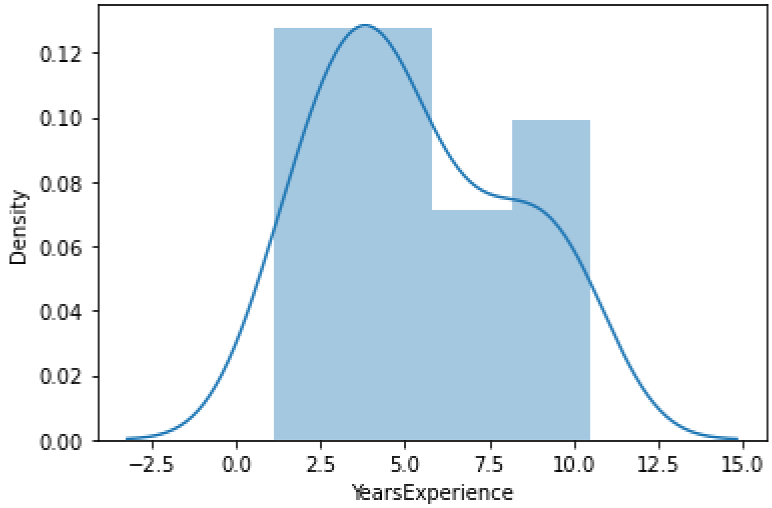
Out[7]:

	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

Graphical Univariate analysis on dataset

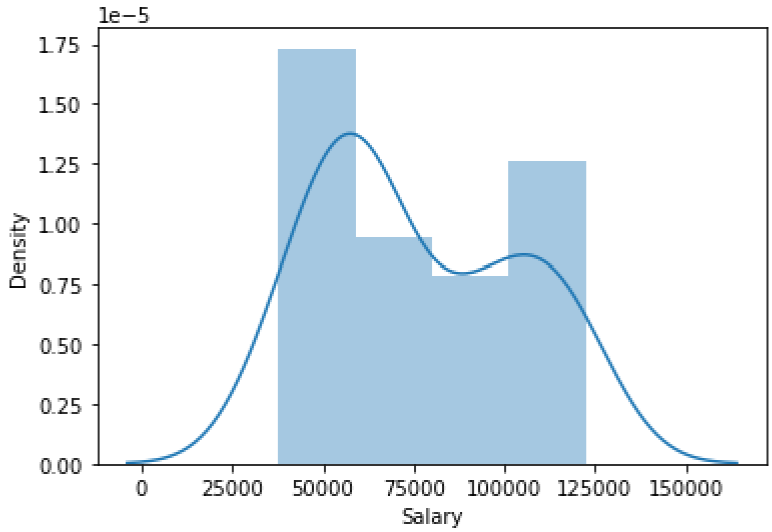
In [8]:
```
sns.distplot(df.YearsExperience)
```

Out[8]: <AxesSubplot:xlabel='YearsExperience', ylabel='Density'>



In [9]:
```
sns.distplot(df.Salary)
```

Out[9]: <AxesSubplot:xlabel='Salary', ylabel='Density'>



Transforming Target data - Log scale

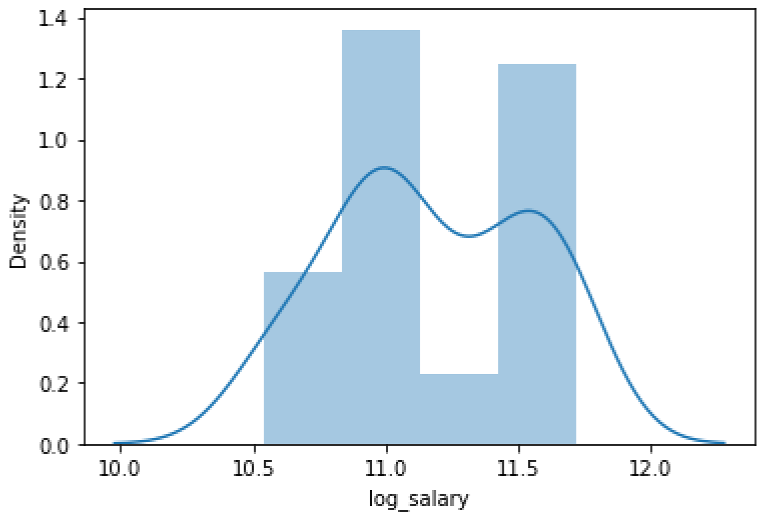
In [10]:
```
#using log for target data to scale target to feature range

df['log_salary'] = np.log(df.Salary)  #new target= log_salary
df.head()
```

Out[10]:

	YearsExperience	Salary	log_salary
0	1.1	39343	10.580073
1	1.3	46205	10.740843
2	1.5	37731	10.538237
3	2.0	43525	10.681091
4	2.2	39891	10.593906

In [11]:
```
sns.distplot(df.log_salary)
```

Out[11]:
```
<AxesSubplot:xlabel='log_salary', ylabel='Density'>
```



In [12]:
```
df['log_salary'].head()                    #scaled Target data scaled
```

Out[12]:
```
0    10.580073
1    10.740843
2    10.538237
3    10.681091
4    10.593906
Name: log_salary, dtype: float64
```

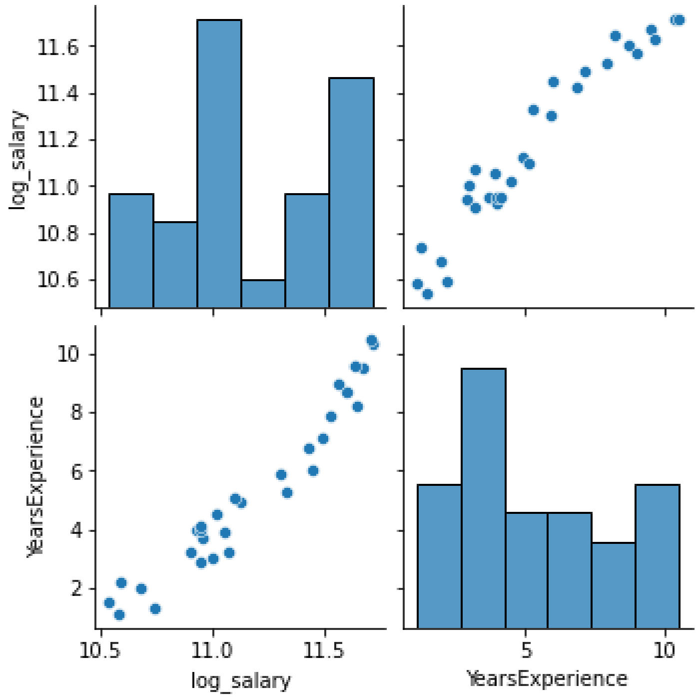
In [13]:
```
df.skew()                                  #skewness of target after scaling
```

Out[13]:
```
YearsExperience    0.379560
Salary             0.354120
log_salary        -0.044126
dtype: float64
```

Graphical Bi-variate analysis on dataset

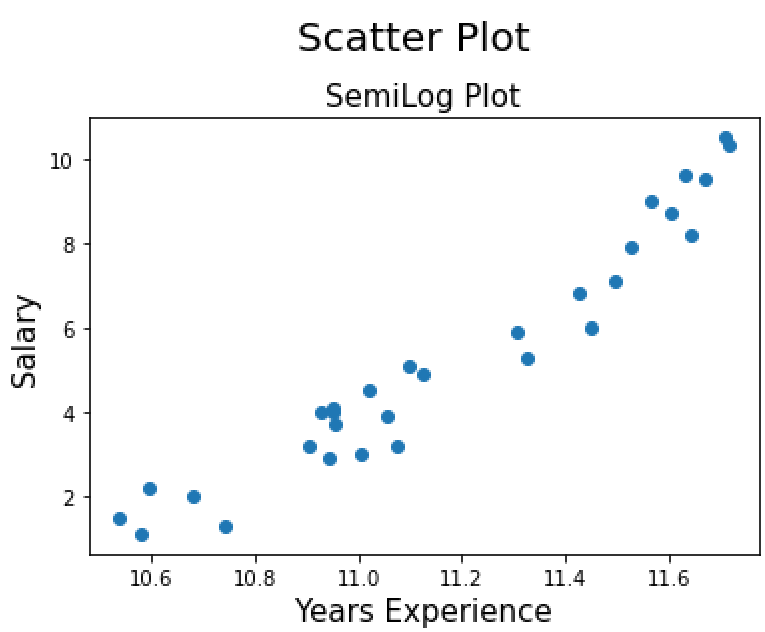
In [14]:
```
# Following the regression equation, our dependent variable (y) is the Salary
y = df['log_salary']

# Similarly, our independent variable (x) is the Year Experience
x = df['YearsExperience']

#Pairplot of predictor and target
sns.pairplot(df, vars =['log_salary','YearsExperience'])
```

Out[14]:
```
<seaborn.axisgrid.PairGrid at 0x271fa5469a0>
```



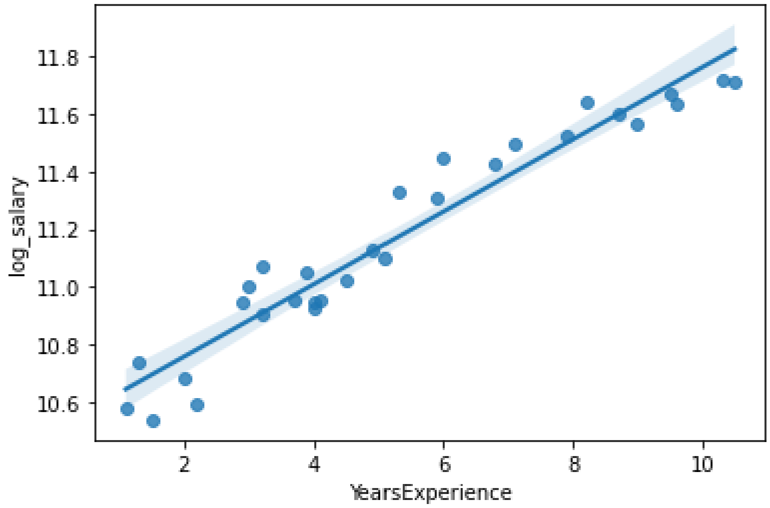
SemiLog Plot

In [15]:
```
plt.scatter(y,x)                                        # Scatter plot

plt.xlabel('Years Experience', fontsize = 15)           # Named the axes
plt.ylabel('Salary', fontsize = 15)
plt.title(label='SemiLog Plot', fontsize=15)
plt.suptitle('Scatter Plot', size=20, y=1.05)
plt.show()                                              # Show the plot
```



Fitting a Linear Regression Model

In [16]:
```
sns.regplot(x="YearsExperience", y="log_salary", data=df);
```



In [17]:
```
import statsmodels.formula.api as smf

model = smf.ols("log_salary~YearsExperience", data = df).fit()
model.summary()
```

Out[17]:

OLS Regression Results

Dep. Variable:	log_salary	**R-squared:**	0.932
Model:	OLS	**Adj. R-squared:**	0.930
Method:	Least Squares	**F-statistic:**	383.6
Date:	Fri, 21 Oct 2022	**Prob (F-statistic):**	7.03e-18
Time:	21:32:08	**Log-Likelihood:**	28.183
No. Observations:	30	**AIC:**	-52.37
Df Residuals:	28	**BIC:**	-49.56
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	10.5074	0.038	273.327	0.000	10.429	10.586
YearsExperience	0.1255	0.006	19.585	0.000	0.112	0.139

Omnibus:	0.826	**Durbin-Watson:**	1.438
Prob(Omnibus):	0.661	**Jarque-Bera (JB):**	0.812
Skew:	0.187	**Prob(JB):**	0.666
Kurtosis:	2.286	**Cond. No.**	13.2

Notes:

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

Summary:

That's one of the strong points of statsmodels Summary shows that beta values required for straight line for each feature as 'Years Experience' and 'constant'

Which feature is important for the Target is determined by P(t) i.e. hypothesis p-value

The feature Years Experience = p value(beta2) = 0.000 The feature Const = p value(beta1) = 0.000

Hence, we Reject null hypothesis in both cases. So, Years Experience and Constant both are important feature for the Target prediction

Intercept(Cosntant) of best fitted line is 10.5074 Feature(YearsExperience) coefficient is 0.1255

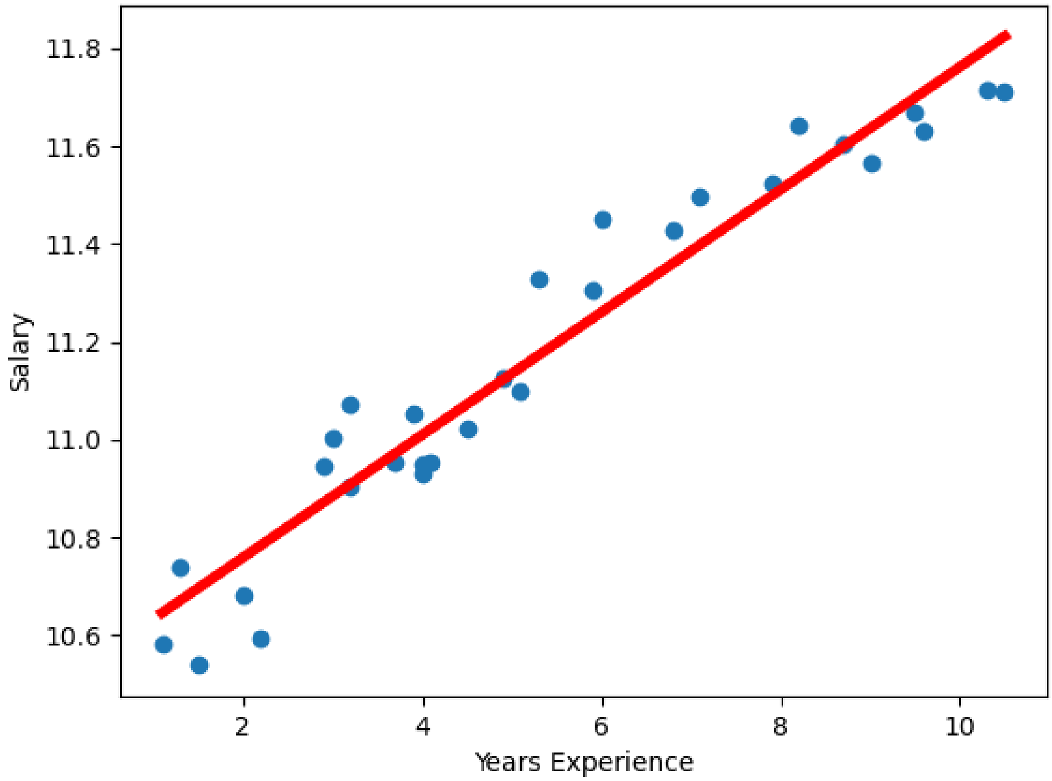
R-sqaured error value is closer to 1 that means the regression model covers most part of the variance of the values of the response variable and can be termed as a good model.

In [18]:

```
%matplotlib notebook
```

In [19]:

```
# Created a scatter plot
plt.scatter(x,y)

# Defined the estimated regression line, so we can plot it later
yhat = 0.1255*x + 10.5074  #predicted y

# Plotting the regression line against the independent variable (Years Experience)
fig = plt.plot(x,yhat, lw=4, c='red', label ='Regression line')

# Label the axes
plt.xlabel('Years Experience', fontsize = 10)
plt.ylabel('Salary', fontsize = 10)
plt.show()
```



In [20]:
```
#log values - predcited target value
yhat.head()
```

Out[20]:
```
0    10.64545
1    10.67055
2    10.69565
3    10.75840
4    10.78350
Name: YearsExperience, dtype: float64
```

In [21]:
```
#Anti-log - actual y values
y = np.exp(yhat)
y.head()
```

Out[21]:
```
0    42001.054328
1    43068.622727
2    44163.326214
3    47023.370416
4    48218.594324
Name: YearsExperience, dtype: float64
```

In [22]:
```
#Co-efficients values
#beta1 and bet0 values
model.params
```

Out[22]:
```
Intercept          10.507402
YearsExperience     0.125453
dtype: float64
```

In [23]:
```
#t and p-Values for intercept and Years Experience.
print(model.tvalues, '\n', model.pvalues)
```

```
Intercept          273.327166
YearsExperience     19.584833
dtype: float64
 Intercept          1.604634e-49
YearsExperience    7.027439e-18
dtype: float64
```