Question -

1) Delivery_time -> Predict delivery time using sorting time 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.

In [11]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
```

PART A. - DELIVERY_TIME

In [42]:

```
delivery_data = pd.read_csv('delivery_time.csv')
delivery_data
```

Out[42]:

	Delivery Time	Sorting Time
0	21.00	10
1	13.50	4
2	19.75	6
3	24.00	9
4	29.00	10
5	15.35	6
6	19.00	7
7	9.50	3
8	17.90	10
9	18.75	9
10	19.83	8
11	10.75	4
12	16.68	7
13	11.50	3
14	12.03	3
15	14.88	4
16	13.75	6
17	18.11	7
18	8.00	2
19	17.83	7
20	21.50	5

In [44]:

```
#initial analysis
delivery_data.shape
```

Out[44]:

```
(21, 2)
```

In [45]:

```
delivery_data.dtypes
```

Out[45]:

```
Delivery Time    float64
Sorting Time       int64
dtype: object
```

In [46]:

```
delivery_data.isna().sum()
```

Out[46]:

```
Delivery Time    0
Sorting Time     0
dtype: int64
```

In [50]:

```
#correlation check
round(delivery_data.corr(),3)
```

Out[50]:

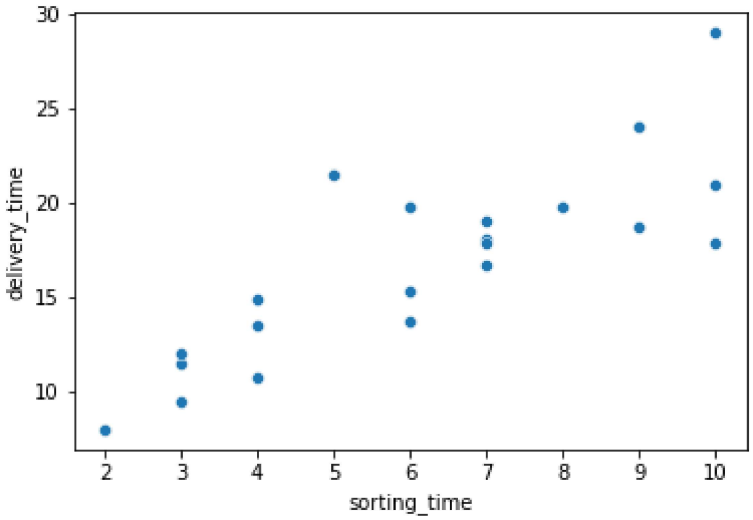
	Delivery Time	Sorting Time
Delivery Time	1.000	0.826
Sorting Time	0.826	1.000

In [9]:

```
#Linearity check
```

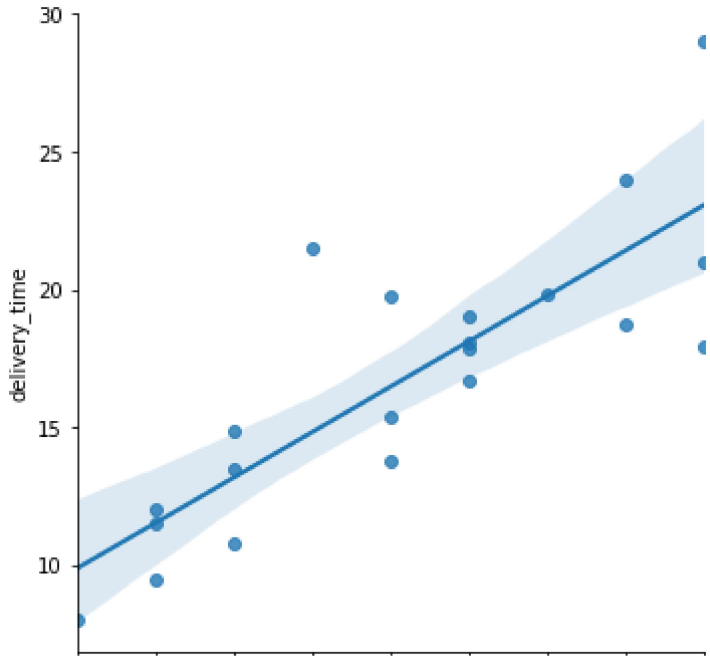
In [108]:

```
sns.scatterplot(x='sorting_time',y= 'delivery_time', data=delivery_data)
plt.show()
```



In [109]:

```
sns.lmplot(x='sorting_time', y= 'delivery_time', data= delivery_data)
plt.show()
```

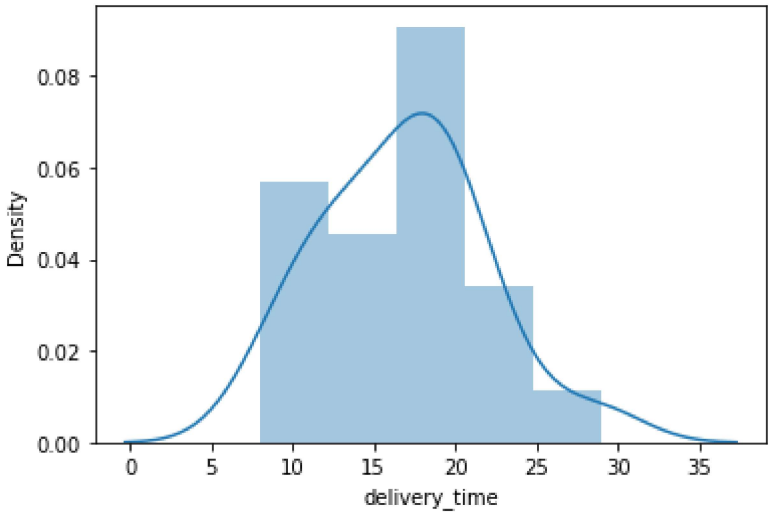


In [30]:

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

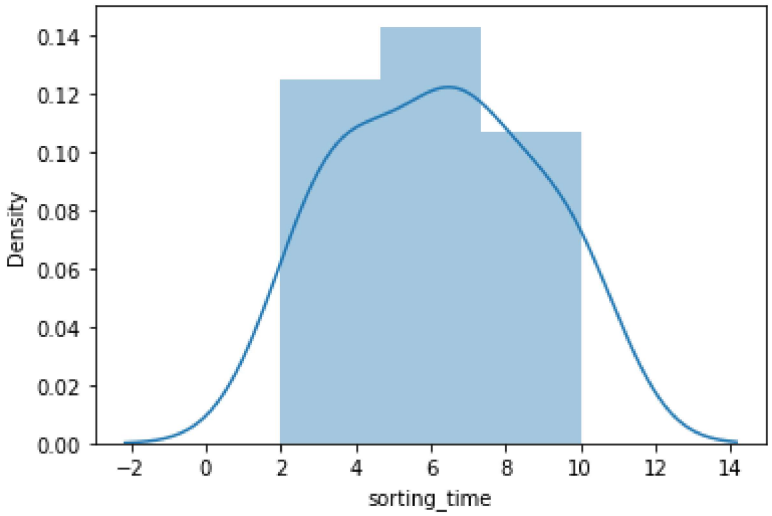
In [110]:

```
#Normality Check
sns.distplot(delivery_data['delivery_time'])
plt.show()
```



In [111]:

```
sns.distplot(delivery_data['sorting_time'])
plt.show()
```



In [55]:

```
#Renaming columns
delivery_data = delivery_data.rename({'Delivery Time' : 'delivery_time', 'Sorting Time': 's
delivery_data
```

Out[55]:

	delivery_time	sorting_time
0	21.00	10
1	13.50	4
2	19.75	6
3	24.00	9
4	29.00	10
5	15.35	6
6	19.00	7
7	9.50	3
8	17.90	10
9	18.75	9
10	19.83	8
11	10.75	4
12	16.68	7
13	11.50	3
14	12.03	3
15	14.88	4
16	13.75	6
17	18.11	7
18	8.00	2
19	17.83	7
20	21.50	5

Model Building

In [33]:

```
import statsmodels.formula.api as smf
```

In [56]:

```
linear_reg_model = smf.ols('delivery_time ~sorting_time', data = delivery_data).fit()
```

Model Testing

In [57]:

```
# finding coefficient parameters
linear_reg_model.params
```

Out[57]:

```
Intercept       6.582734
sorting_time    1.649020
dtype: float64
```

In [58]:

```
# Finding tvalues and pvalues
linear_reg_model.tvalues , linear_reg_model.pvalues
```

Out[58]:

```
(Intercept       3.823349
 sorting_time    6.387447
 dtype: float64,
 Intercept       0.001147
 sorting_time    0.000004
 dtype: float64)
```

In [59]:

```
linear_reg_model.rsquared , linear_reg_model.rsquared_adj
```

Out[59]:

```
(0.6822714748417231, 0.6655489208860244)
```

In [60]:

```
linear_reg_model.predict(delivery_data)
```

Out[60]:

```
0     23.072933
1     13.178814
2     16.476853
3     21.423913
4     23.072933
5     16.476853
6     18.125873
7     11.529794
8     23.072933
9     21.423913
10    19.774893
11    13.178814
12    18.125873
13    11.529794
14    11.529794
15    13.178814
16    16.476853
17    18.125873
18     9.880774
19    18.125873
20    14.827833
dtype: float64
```

In [79]:

```
from pickle import dump
```

In [80]:

```
dump(linear_reg_model, open('delivery_data.pkl','wb'))
```

In [81]:

```
from pickle import load
```

In [82]:

```
tested_linear_model = load(open('delivery_data.pkl','rb'))
```

In [83]:

```
tested_linear_model.predict(delivery_data)
```

Out[83]:

```
0     23.072933
1     13.178814
2     16.476853
3     21.423913
4     23.072933
5     16.476853
6     18.125873
7     11.529794
8     23.072933
9     21.423913
10    19.774893
11    13.178814
12    18.125873
13    11.529794
14    11.529794
15    13.178814
16    16.476853
17    18.125873
18     9.880774
19    18.125873
20    14.827833
dtype: float64
```

Transformation

1. Log Transformation

In [91]:

```
delivery_data
```

Out[91]:

	delivery_time	sorting_time
0	21.00	10
1	13.50	4
2	19.75	6
3	24.00	9
4	29.00	10
5	15.35	6
6	19.00	7
7	9.50	3
8	17.90	10
9	18.75	9
10	19.83	8
11	10.75	4
12	16.68	7
13	11.50	3
14	12.03	3
15	14.88	4
16	13.75	6
17	18.11	7
18	8.00	2
19	17.83	7
20	21.50	5

In [114]:

```
delivery_data['log_sorting_time']=np.log(delivery_data['sorting_time'])
delivery_data['log_delivery_time']=np.log(delivery_data['delivery_time'])
delivery_data
```

Out[114]:

	delivery_time	sorting_time	sorting_name	log_sorting_name	log_sorting_time	log_delivery_
0	21.00	10	2.302585	2.302585	2.302585	3.04
1	13.50	4	1.386294	1.386294	1.386294	2.60
2	19.75	6	1.791759	1.791759	1.791759	2.98
3	24.00	9	2.197225	2.197225	2.197225	3.17
4	29.00	10	2.302585	2.302585	2.302585	3.36
5	15.35	6	1.791759	1.791759	1.791759	2.73
6	19.00	7	1.945910	1.945910	1.945910	2.94
7	9.50	3	1.098612	1.098612	1.098612	2.25
8	17.90	10	2.302585	2.302585	2.302585	2.88
9	18.75	9	2.197225	2.197225	2.197225	2.93
10	19.83	8	2.079442	2.079442	2.079442	2.98
11	10.75	4	1.386294	1.386294	1.386294	2.37
12	16.68	7	1.945910	1.945910	1.945910	2.81
13	11.50	3	1.098612	1.098612	1.098612	2.44
14	12.03	3	1.098612	1.098612	1.098612	2.48
15	14.88	4	1.386294	1.386294	1.386294	2.70
16	13.75	6	1.791759	1.791759	1.791759	2.62
17	18.11	7	1.945910	1.945910	1.945910	2.89
18	8.00	2	0.693147	0.693147	0.693147	2.07
19	17.83	7	1.945910	1.945910	1.945910	2.88
20	21.50	5	1.609438	1.609438	1.609438	3.06

In [115]:

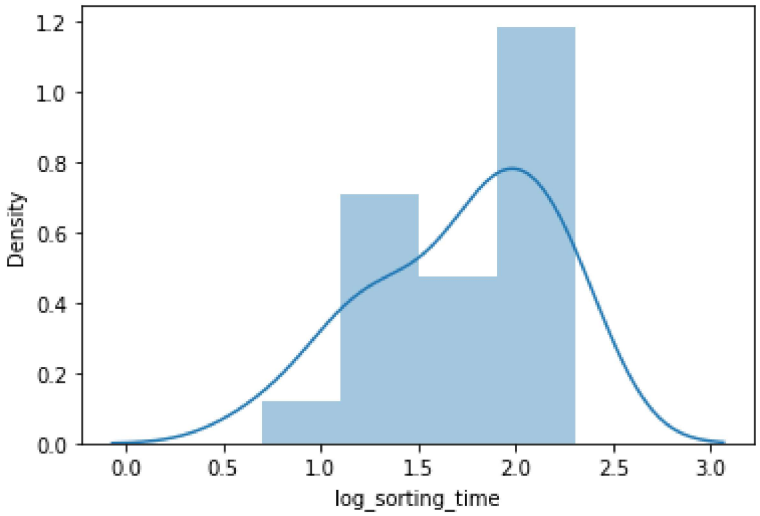
```
delivery_log_transformed = delivery_data.loc[:,['log_sorting_time', 'log_delivery_time']]
delivery_log_transformed
```

Out[115]:

	log_sorting_time	log_delivery_time
0	2.302585	3.044522
1	1.386294	2.602690
2	1.791759	2.983153
3	2.197225	3.178054
4	2.302585	3.367296
5	1.791759	2.731115
6	1.945910	2.944439
7	1.098612	2.251292
8	2.302585	2.884801
9	2.197225	2.931194
10	2.079442	2.987196
11	1.386294	2.374906
12	1.945910	2.814210
13	1.098612	2.442347
14	1.098612	2.487404
15	1.386294	2.700018
16	1.791759	2.621039
17	1.945910	2.896464
18	0.693147	2.079442
19	1.945910	2.880882
20	1.609438	3.068053

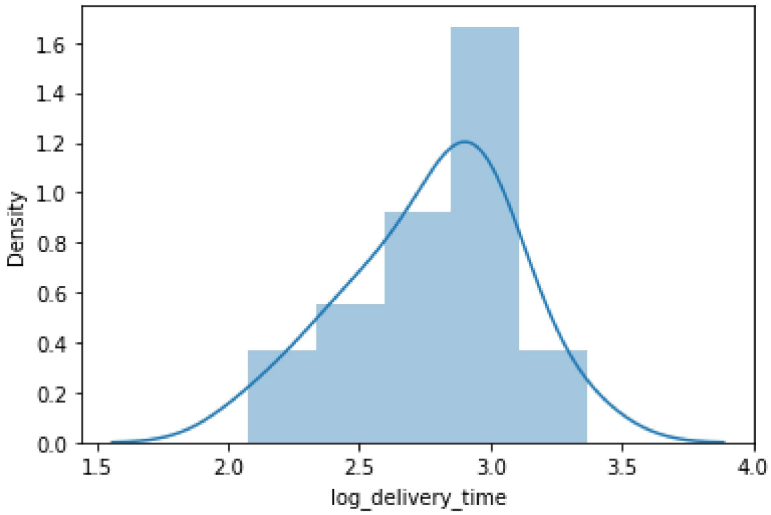
In [116]:

```
sns.distplot(delivery_data['log_sorting_time'])
plt.show()
```



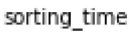
In [118]:

```
sns.distplot(delivery_data['log_delivery_time'])
plt.show()
```



In [119]:

```
sns.lmplot(x='sorting_time' , y = 'delivery_time', data =  delivery_time)
sns.lmplot(x= 'log_sorting_time' , y = 'log_delivery_time', data = delivery_log_transformed
plt.show()
```



Understanding R2 and Adjusted R2

In [121]:

```
linear_reg_model = smf.ols('delivery_time ~sorting_time', data = delivery_data).fit()
print('R2 score          : ', linear_reg_model.rsquared)
print('Adjusted R2 score :', linear_reg_model.rsquared_adj)
print('AIC value         : ', linear_reg_model.aic)
print('BIC value         :' , linear_reg_model.bic)
```

```
R2 score          :  0.6822714748417231
Adjusted R2 score : 0.6655489208860244
AIC value         :  106.71400170798609
BIC value         : 108.80304658343293
```

In [122]:

```
linear_reg_model_log = smf.ols('log_delivery_time ~ log_sorting_time', data = delivery_log_
print('R2 score          : ', linear_reg_model_log.rsquared)
print('Adjusted R2 score :', linear_reg_model_log.rsquared_adj)
print('AIC value         : ', linear_reg_model_log.aic)
print('BIC value         :' , linear_reg_model_log.bic)
```

```
R2 score          :  0.77216134926874
Adjusted R2 score : 0.7601698413355158
AIC value         :  -16.58128395971123
BIC value         : -14.492239084264384
```

2. SquareRoot Transformation

In [123]:

```
delivery_data['sqrt_sorting_time']=np.sqrt(delivery_data['sorting_time'])
delivery_data['sqrt_delivery_time']=np.sqrt(delivery_data['delivery_time'])
delivery_data
```

Out[123]:

	delivery_time	sorting_time	sorting_name	log_sorting_name	log_sorting_time	log_delivery_
0	21.00	10	2.302585	2.302585	2.302585	3.04
1	13.50	4	1.386294	1.386294	1.386294	2.60
2	19.75	6	1.791759	1.791759	1.791759	2.98
3	24.00	9	2.197225	2.197225	2.197225	3.17
4	29.00	10	2.302585	2.302585	2.302585	3.36
5	15.35	6	1.791759	1.791759	1.791759	2.73
6	19.00	7	1.945910	1.945910	1.945910	2.94
7	9.50	3	1.098612	1.098612	1.098612	2.25
8	17.90	10	2.302585	2.302585	2.302585	2.88
9	18.75	9	2.197225	2.197225	2.197225	2.93
10	19.83	8	2.079442	2.079442	2.079442	2.98
11	10.75	4	1.386294	1.386294	1.386294	2.37
12	16.68	7	1.945910	1.945910	1.945910	2.81
13	11.50	3	1.098612	1.098612	1.098612	2.44
14	12.03	3	1.098612	1.098612	1.098612	2.48
15	14.88	4	1.386294	1.386294	1.386294	2.70
16	13.75	6	1.791759	1.791759	1.791759	2.62
17	18.11	7	1.945910	1.945910	1.945910	2.89
18	8.00	2	0.693147	0.693147	0.693147	2.07
19	17.83	7	1.945910	1.945910	1.945910	2.88
20	21.50	5	1.609438	1.609438	1.609438	3.06

In [ ]:

```
```

In [124]:

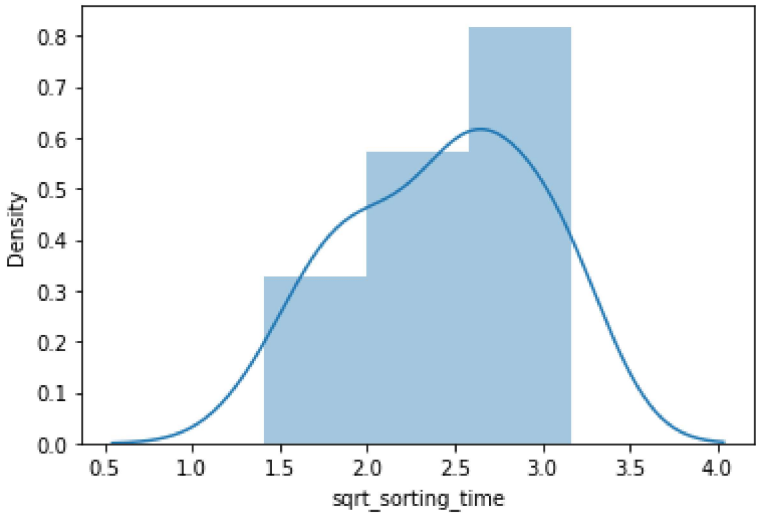
```
delivery_sqrt_transformed = delivery_data.loc[:,['sqrt_sorting_time', 'sqrt_delivery_time']]
delivery_sqrt_transformed
```

Out[124]:

	sqrt_sorting_time	sqrt_delivery_time
0	3.162278	4.582576
1	2.000000	3.674235
2	2.449490	4.444097
3	3.000000	4.898979
4	3.162278	5.385165
5	2.449490	3.917908
6	2.645751	4.358899
7	1.732051	3.082207
8	3.162278	4.230839
9	3.000000	4.330127
10	2.828427	4.453089
11	2.000000	3.278719
12	2.645751	4.084116
13	1.732051	3.391165
14	1.732051	3.468429
15	2.000000	3.857460
16	2.449490	3.708099
17	2.645751	4.255585
18	1.414214	2.828427
19	2.645751	4.222558
20	2.236068	4.636809

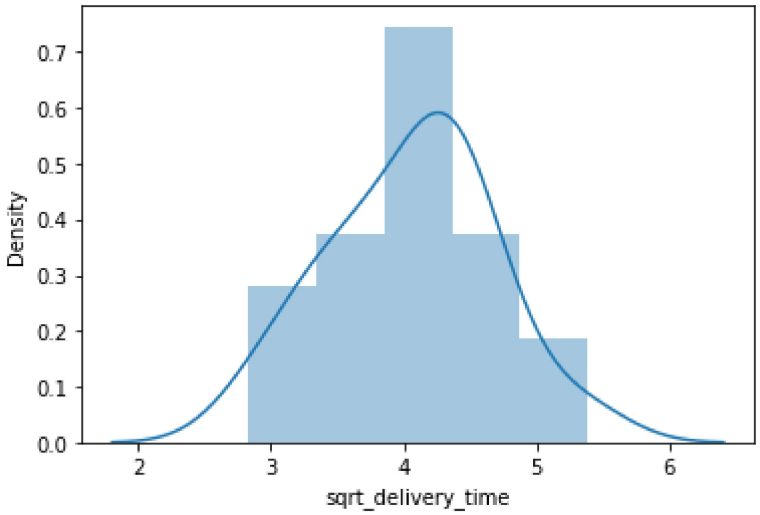
In [125]:

```
sns.distplot(delivery_data['sqrt_sorting_time'])
plt.show()
```



In [126]:

```
sns.distplot(delivery_data['sqrt_delivery_time'])
plt.show()
```



In [127]:

```
sns.lmplot(x='sorting_time' , y = 'delivery_time', data =  delivery_time)
sns.lmplot(x= 'sqrt_sorting_time' , y = 'sqrt_delivery_time', data = delivery_sqrt_transfor
plt.show()
```

Understanding R2 and Adjusted R2

In [121]:

```
linear_reg_model = smf.ols('delivery_time ~sorting_time', data = delivery_data).fit()
print('R2 score          : ', linear_reg_model.rsquared)
print('Adjusted R2 score :', linear_reg_model.rsquared_adj)
print('AIC value         : ', linear_reg_model.aic)
print('BIC value         :' , linear_reg_model.bic)
```

```
R2 score          :  0.6822714748417231
Adjusted R2 score : 0.6655489208860244
AIC value         :  106.71400170798609
BIC value         : 108.80304658343293
```

In [129]:

```
linear_reg_model_sqrt = smf.ols('sqrt_delivery_time ~ sqrt_sorting_time', data = delivery_s
print('R2 score          : ', linear_reg_model_sqrt.rsquared)
print('Adjusted R2 score :', linear_reg_model_sqrt.rsquared_adj)
print('AIC value         : ', linear_reg_model_sqrt.aic)
print('BIC value         :' , linear_reg_model_sqrt.bic)
```

```
R2 score          :  0.7292011987544664
Adjusted R2 score : 0.7149486302678594
AIC value         :  15.463994487107719
BIC value         : 17.553039362554564
```

In [139]:

```
linear_reg_model.summary()
```

Out[139]:

OLS Regression Results

Dep. Variable:	delivery_time	**R-squared:**	0.682
Model:	OLS	**Adj. R-squared:**	0.666
Method:	Least Squares	**F-statistic:**	40.80
Date:	Sat, 23 Oct 2021	**Prob (F-statistic):**	3.98e-06
Time:	21:35:39	**Log-Likelihood:**	-51.357
No. Observations:	21	**AIC:**	106.7
Df Residuals:	19	**BIC:**	108.8
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	6.5827	1.722	3.823	0.001	2.979	10.186
sorting_time	1.6490	0.258	6.387	0.000	1.109	2.189

Omnibus:	3.649	**Durbin-Watson:**	1.248
Prob(Omnibus):	0.161	**Jarque-Bera (JB):**	2.086
Skew:	0.750	**Prob(JB):**	0.352
Kurtosis:	3.367	**Cond. No.**	18.3

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

In [140]:

```
linear_reg_model_log.summary()
```

Out[140]:

OLS Regression Results

Dep. Variable:	log_delivery_time	**R-squared:**	0.772
Model:	OLS	**Adj. R-squared:**	0.760
Method:	Least Squares	**F-statistic:**	64.39
Date:	Sat, 23 Oct 2021	**Prob (F-statistic):**	1.60e-07
Time:	21:37:00	**Log-Likelihood:**	10.291
No. Observations:	21	**AIC:**	-16.58
Df Residuals:	19	**BIC:**	-14.49
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.7420	0.133	13.086	0.000	1.463	2.021
log_sorting_time	0.5975	0.074	8.024	0.000	0.442	0.753

Omnibus:	1.871	**Durbin-Watson:**	1.322
Prob(Omnibus):	0.392	**Jarque-Bera (JB):**	1.170
Skew:	0.577	**Prob(JB):**	0.557
Kurtosis:	2.916	**Cond. No.**	9.08

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

In [141]:

```
linear_reg_model_sqrt.summary()
```

Out[141]:

OLS Regression Results

Dep. Variable:	sqrt_delivery_time	**R-squared:**	0.729
Model:	OLS	**Adj. R-squared:**	0.715
Method:	Least Squares	**F-statistic:**	51.16
Date:	Sat, 23 Oct 2021	**Prob (F-statistic):**	8.48e-07
Time:	21:37:04	**Log-Likelihood:**	-5.7320
No. Observations:	21	**AIC:**	15.46
Df Residuals:	19	**BIC:**	17.55
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]

In [ ]:

```
```

PART B. - SALARY DATA

In [62]:

```
salary_data = pd.read_csv('Salary_Data.csv')
salary_data
```

Out[62]:

	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

In [ ]:

```

```

In [63]:

```
#initial analysis
salary_data.shape
```

Out[63]:

```
(30, 2)
```

In [64]:

```
salary_data.dtypes
```

Out[64]:

```
YearsExperience    float64
Salary             float64
dtype: object
```

In [65]:

```
salary_data.isna().sum()
```

Out[65]:

```
YearsExperience    0
Salary             0
dtype: int64
```

In [66]:

```
#correlation check
round(salary_data.corr(),3)
```

Out[66]:

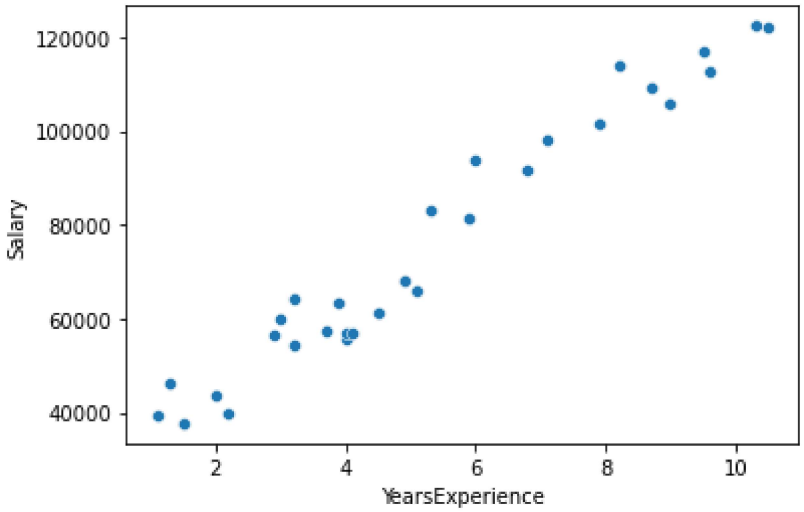
	YearsExperience	Salary
YearsExperience	1.000	0.978
Salary	0.978	1.000

In [9]:

```
#Linearity check
```

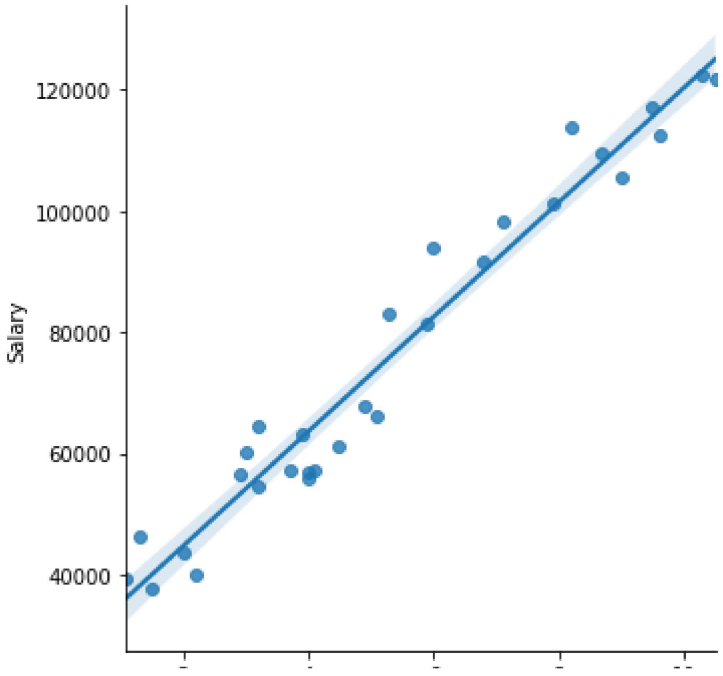
In [67]:

```
sns.scatterplot(x='YearsExperience',y= 'Salary', data=salary_data)
plt.show()
```



In [68]:

```
sns.lmplot(x='YearsExperience', y= 'Salary', data= salary_data)
plt.show()
```

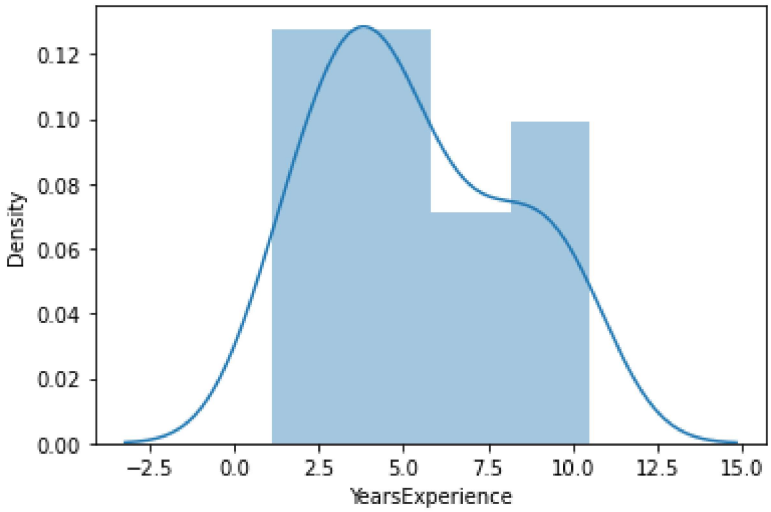


In [69]:

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

In [70]:

```
#Normality Check
sns.distplot(salary_data['YearsExperience'])
plt.show()
```



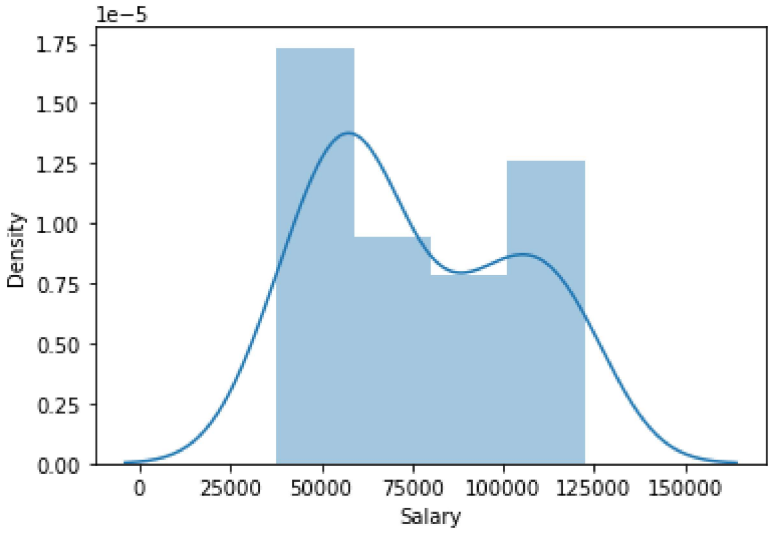
In [ ]:

```

```

In [71]:

```
sns.distplot(salary_data['Salary'])
plt.show()
```



Model Building

In [72]:

```
import statsmodels.formula.api as smf
```

In [74]:

```
linear_reg_model_1 = smf.ols('Salary~YearsExperience', data = salary_data).fit()
```

Model Testing

In [75]:

```
# finding coefficient parameters
linear_reg_model_1.params
```

Out[75]:

```
Intercept          25792.200199
YearsExperience     9449.962321
dtype: float64
```

In [76]:

```
# Finding tvalues and pvalues
linear_reg_model_1.tvalues , linear_reg_model_1.pvalues
```

Out[76]:

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

In [77]:

```
linear_reg_model_1.rsquared , linear_reg_model_1.rsquared_adj
```

Out[77]:

```
(0.9569566641435086, 0.9554194021486339)
```

In [78]:

```
linear_reg_model_1.predict(salary_data)
```

Out[78]:

```
0      36187.158752
1      38077.151217
2      39967.143681
3      44692.124842
4      46582.117306
5      53197.090931
6      54142.087163
7      56032.079627
8      56032.079627
9      60757.060788
10     62647.053252
11     63592.049484
12     63592.049484
13     64537.045717
14     68317.030645
15     72097.015574
16     73987.008038
17     75877.000502
18     81546.977895
19     82491.974127
20     90051.943985
21     92886.932681
22    100446.902538
23    103281.891235
24    108006.872395
25    110841.861092
26    115566.842252
27    116511.838485
28    123126.812110
29    125016.804574
dtype: float64
```

In [84]:

```
from pickle import dump
```

In [87]:

```
dump (linear_reg_model_1, open('salary_data.pkl','wb'))
```

In [88]:

```
from pickle import load
```

In [89]:

```
tested_linear_model_1 = load(open('salary_data.pkl','rb'))
```

In [90]:

```
tested_linear_model_1.predict(salary_data)
```

Out[90]:

```
0      36187.158752
1      38077.151217
2      39967.143681
3      44692.124842
4      46582.117306
5      53197.090931
6      54142.087163
7      56032.079627
8      56032.079627
9      60757.060788
10     62647.053252
11     63592.049484
12     63592.049484
13     64537.045717
14     68317.030645
15     72097.015574
16     73987.008038
17     75877.000502
18     81546.977895
19     82491.974127
20     90051.943985
21     92886.932681
22    100446.902538
23    103281.891235
24    108006.872395
25    110841.861092
26    115566.842252
27    116511.838485
28    123126.812110
29    125016.804574
dtype: float64
```

In [ ]:

```

```

Transformation

1. Log Transformation

In [130]:

```
salary_data
```

Out[130]:

	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

In [132]:

```
salary_data['log_YearsExperience']=np.log(salary_data['YearsExperience'])
salary_data['log_Salary']=np.log(salary_data['Salary'])
salary_data
```

Out[132]:

	YearsExperience	Salary	log_YearsExperience	log_Salary
0	1.1	39343.0	0.095310	10.580073
1	1.3	46205.0	0.262364	10.740843
2	1.5	37731.0	0.405465	10.538237
3	2.0	43525.0	0.693147	10.681091
4	2.2	39891.0	0.788457	10.593906
5	2.9	56642.0	1.064711	10.944506
6	3.0	60150.0	1.098612	11.004597
7	3.2	54445.0	1.163151	10.904946
8	3.2	64445.0	1.163151	11.073567
9	3.7	57189.0	1.308333	10.954117
10	3.9	63218.0	1.360977	11.054344
11	4.0	55794.0	1.386294	10.929422
12	4.0	56957.0	1.386294	10.950052
13	4.1	57081.0	1.410987	10.952227
14	4.5	61111.0	1.504077	11.020447
15	4.9	67938.0	1.589235	11.126351
16	5.1	66029.0	1.629241	11.097849
17	5.3	83088.0	1.667707	11.327656
18	5.9	81363.0	1.774952	11.306676
19	6.0	93940.0	1.791759	11.450412
20	6.8	91738.0	1.916923	11.426692
21	7.1	98273.0	1.960095	11.495505
22	7.9	101302.0	2.066863	11.525861
23	8.2	113812.0	2.104134	11.642303
24	8.7	109431.0	2.163323	11.603049
25	9.0	105582.0	2.197225	11.567243
26	9.5	116969.0	2.251292	11.669664
27	9.6	112635.0	2.261763	11.631908
28	10.3	122391.0	2.332144	11.714976
29	10.5	121872.0	2.351375	11.710727

In [133]:

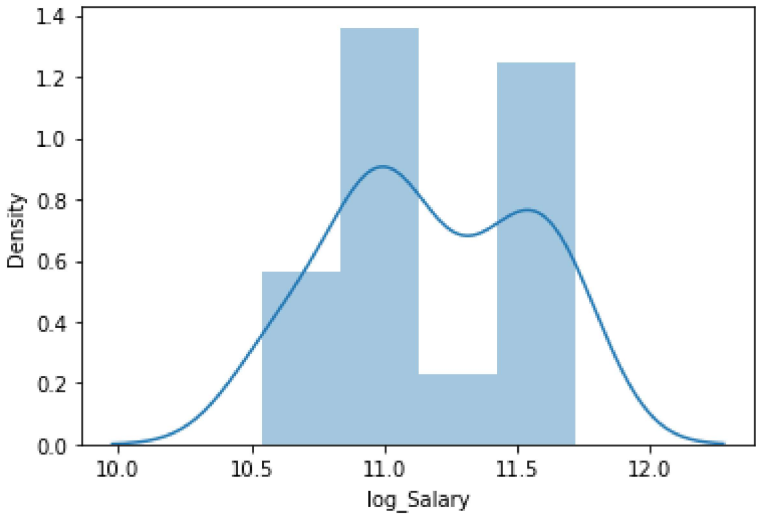
```
salary_log_transformed = salary_data.loc[:,['log_YearsExperience', 'log_Salary']]
salary_log_transformed
```

Out[133]:

	log_YearsExperience	log_Salary
0	0.095310	10.580073
1	0.262364	10.740843
2	0.405465	10.538237
3	0.693147	10.681091
4	0.788457	10.593906
5	1.064711	10.944506
6	1.098612	11.004597
7	1.163151	10.904946
8	1.163151	11.073567
9	1.308333	10.954117
10	1.360977	11.054344
11	1.386294	10.929422
12	1.386294	10.950052
13	1.410987	10.952227
14	1.504077	11.020447
15	1.589235	11.126351
16	1.629241	11.097849
17	1.667707	11.327656
18	1.774952	11.306676
19	1.791759	11.450412
20	1.916923	11.426692
21	1.960095	11.495505
22	2.066863	11.525861
23	2.104134	11.642303
24	2.163323	11.603049
25	2.197225	11.567243
26	2.251292	11.669664
27	2.261763	11.631908
28	2.332144	11.714976
29	2.351375	11.710727

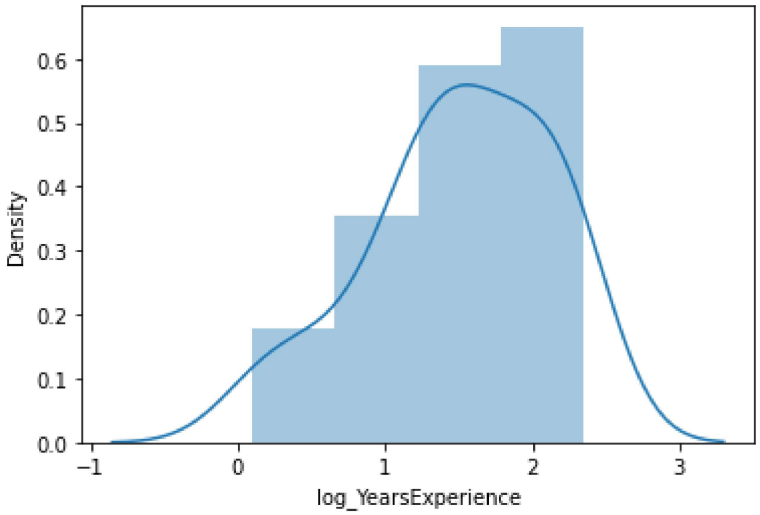
In [134]:

```
sns.distplot(salary_data['log_Salary'])
plt.show()
```



In [135]:

```
sns.distplot(salary_data['log_YearsExperience'])
plt.show()
```



In [136]:

```
sns.lmplot(x='YearsExperience' , y = 'Salary', data =  salary_data)
sns.lmplot(x= 'log_YearsExperience' , y = 'log_Salary', data = salary_log_transformed)
plt.show()
```

Understanding R2 and Adjusted R2

In [137]:

```
linear_reg_model_1 = smf.ols('Salary ~YearsExperience', data = salary_data).fit()
print('R2 score          : ', linear_reg_model_1.rsquared)
print('Adjusted R2 score :', linear_reg_model_1.rsquared_adj)
print('AIC value         : ', linear_reg_model_1.aic)
print('BIC value         :' , linear_reg_model_1.bic)
```

```
R2 score          :  0.9569566641435086
Adjusted R2 score : 0.9554194021486339
AIC value         :  606.882316930432
BIC value         : 609.6847116937563
```

In [138]:

```
linear_reg_model_1_log = smf.ols('log_Salary ~ log_YearsExperience', data = salary_log_tran
print('R2 score          : ', linear_reg_model_1_log.rsquared)
print('Adjusted R2 score :', linear_reg_model_1_log.rsquared_adj)
print('AIC value         : ', linear_reg_model_1_log.aic)
print('BIC value         :' , linear_reg_model_1_log.bic)
```

```
R2 score          :  0.905215072581715
Adjusted R2 score : 0.9018298966024905
AIC value         :  -42.41723686686487
BIC value         : -39.61484210354056
```

In [142]:

```
linear_reg_model_1.summary()
```

Out[142]:

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, 23 Oct 2021	**Prob (F-statistic):**	1.14e-20
Time:	21:37:38	**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.

In [143]:

```
linear_reg_model_1_log.summary()
```

Out[143]:

OLS Regression Results

Dep. Variable:	log_Salary	**R-squared:**	0.905
Model:	OLS	**Adj. R-squared:**	0.902
Method:	Least Squares	**F-statistic:**	267.4
Date:	Sat, 23 Oct 2021	**Prob (F-statistic):**	7.40e-16
Time:	21:37:49	**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
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.

In [ ]:

```

```