Linear regression

Problem Statement:
Delivery_time -> Predict delivery time using sorting time
Build a simple linear regression model by performing EDA and do necessary transformations and select the best model using R or Python.

Delivery time prediction

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels

%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
data = pd.read_csv("delivery_time.csv")
data.head()
```

Out[1]:

	Delivery Time	Sorting Time
0	21.00	10
1	13.50	4
2	19.75	6
3	24.00	9
4	29.00	10

In [2]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Delivery Time  21 non-null     float64
 1   Sorting Time   21 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 464.0 bytes
```

In [3]:
```
data.describe()
```

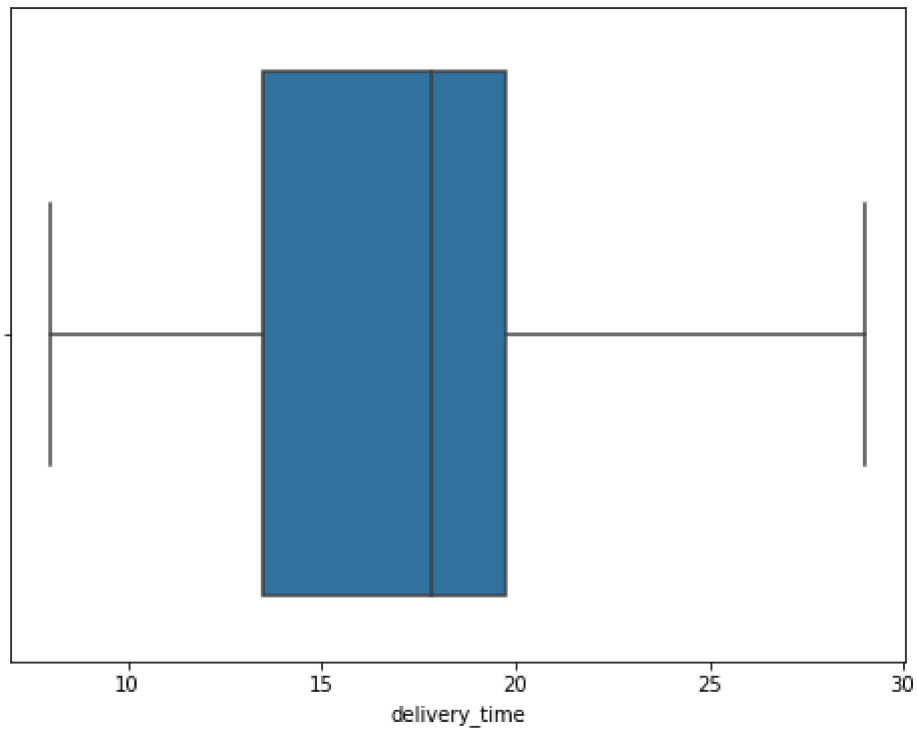
Out[3]:

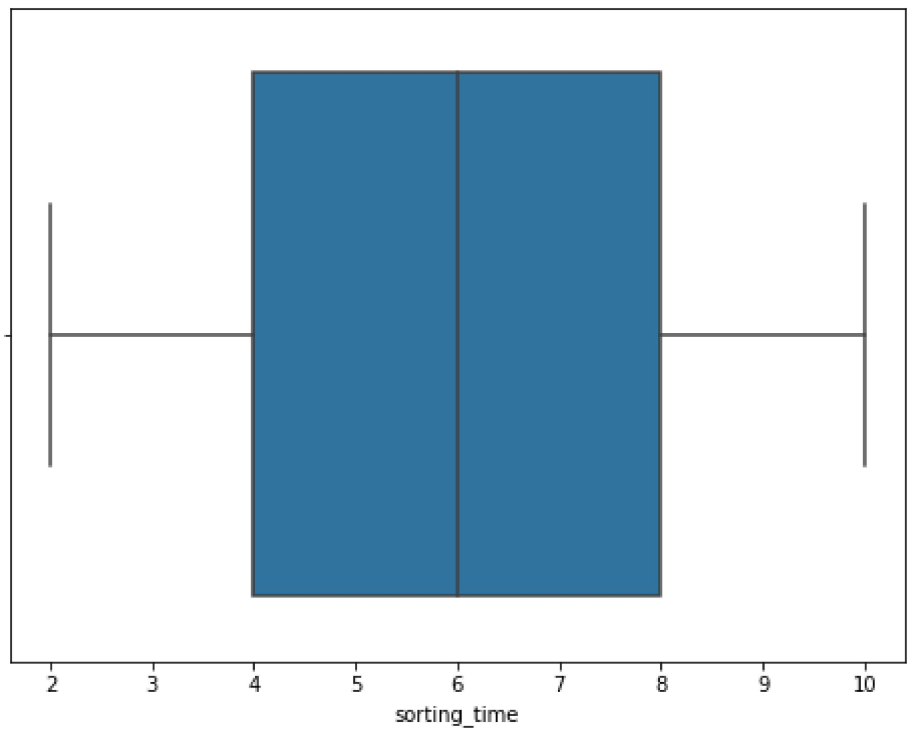
	Delivery Time	Sorting Time
count	21.000000	21.000000
mean	16.790952	6.190476
std	5.074901	2.542028
min	8.000000	2.000000
25%	13.500000	4.000000
50%	17.830000	6.000000
75%	19.750000	8.000000
max	29.000000	10.000000

In [5]:
```
data.isnull().any()
```

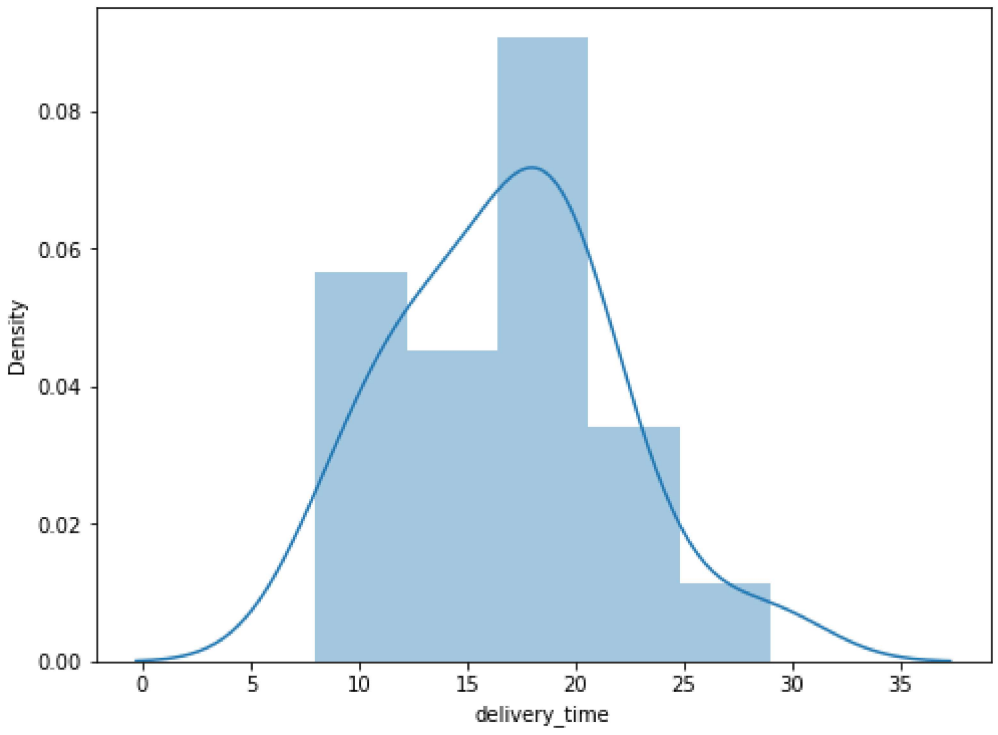
Out[5]:
```
Delivery Time    False
Sorting Time     False
dtype: bool
```

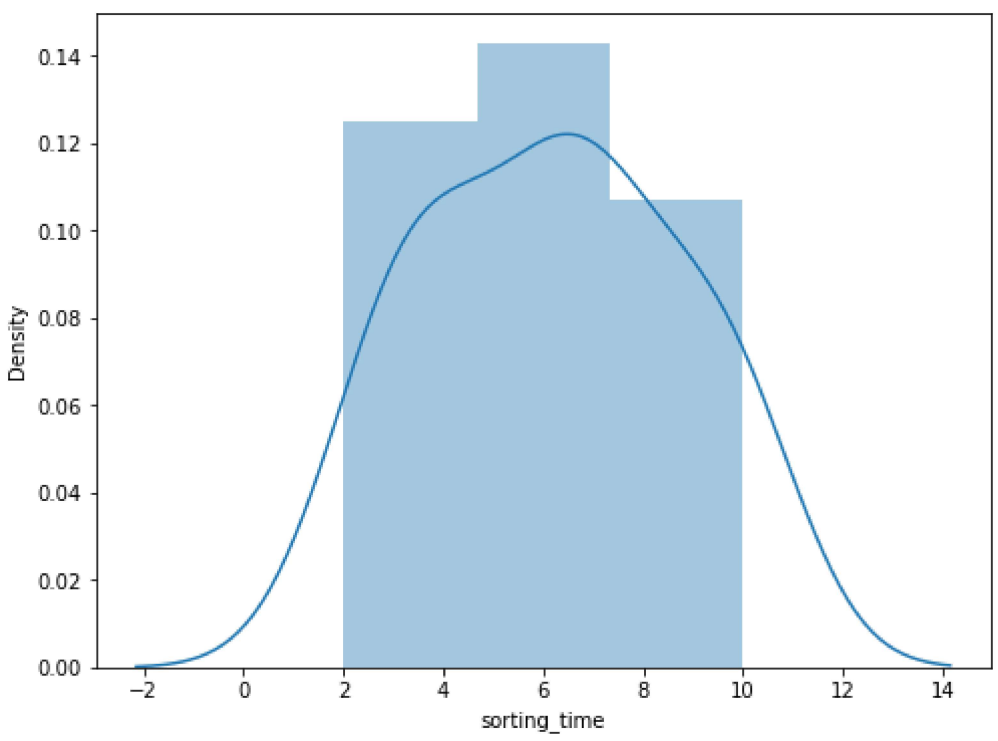
In [6]:
```
# Renaming Columns
data=data.rename({'Delivery Time':'delivery_time', 'Sorting Time':'sorting_time'},axis=1)

# Checking for outliers with help of boxplot
plt.figure(figsize = (8, 6))
plt.tight_layout()
sns.boxplot(data.delivery_time)
plt.show()

plt.figure(figsize = (8, 6))
plt.tight_layout()
sns.boxplot(data.sorting_time)
plt.show()
```





In [7]:
```
plt.figure(figsize = (8, 6))
plt.tight_layout()
sns.distplot(data.delivery_time)
plt.show()

plt.figure(figsize = (8, 6))
plt.tight_layout()
sns.distplot(data.sorting_time)
plt.show()
```

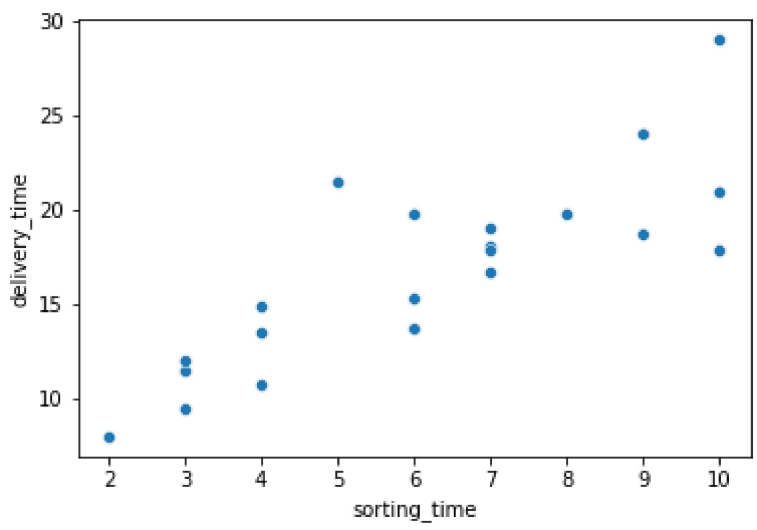




It looks like there are no outliers present in the datset

In [8]:
```
# scatterplot of input variable i.e., sorting time Vs. output variable i.e., delivery time
sns.scatterplot(x = data['sorting_time'], y = data['delivery_time'])
```

Out[8]: <AxesSubplot:xlabel='sorting_time', ylabel='delivery_time'>



From above scatter plot, increase in sorting time leads to increase delivery time

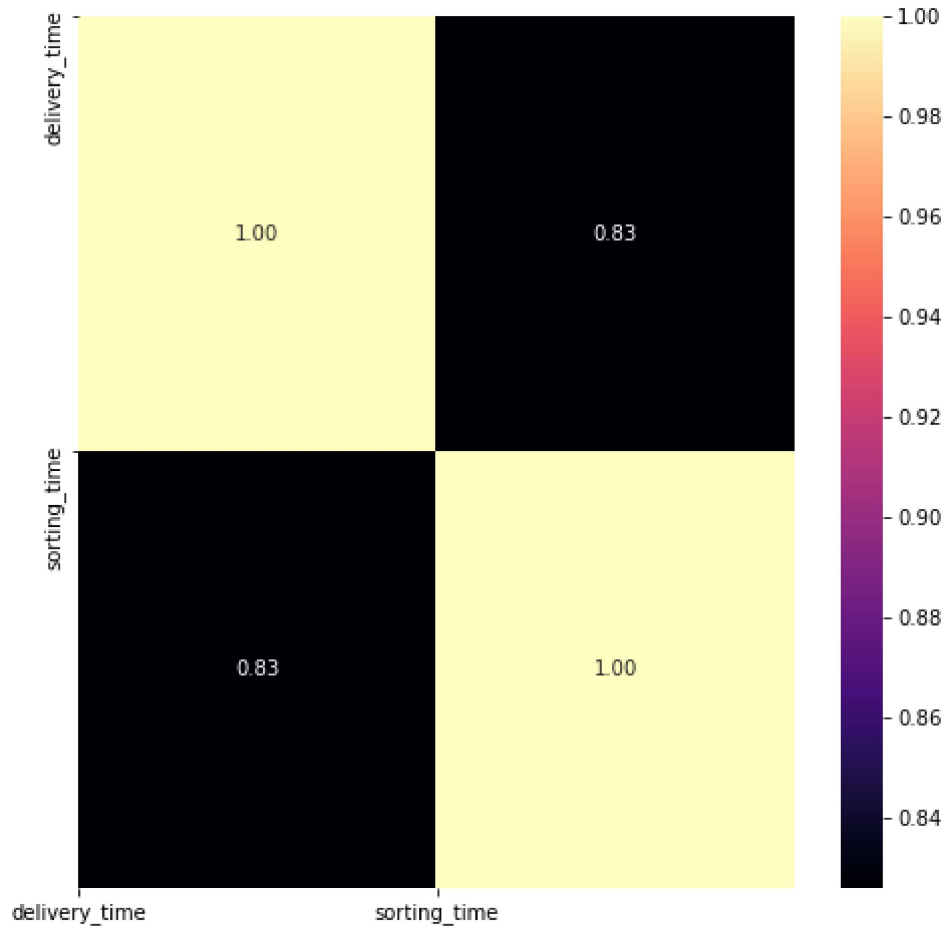
Correlation

In [9]:
```
data.corr()
```

Out[9]:

	delivery_time	sorting_time
delivery_time	1.000000	0.825997
sorting_time	0.825997	1.000000

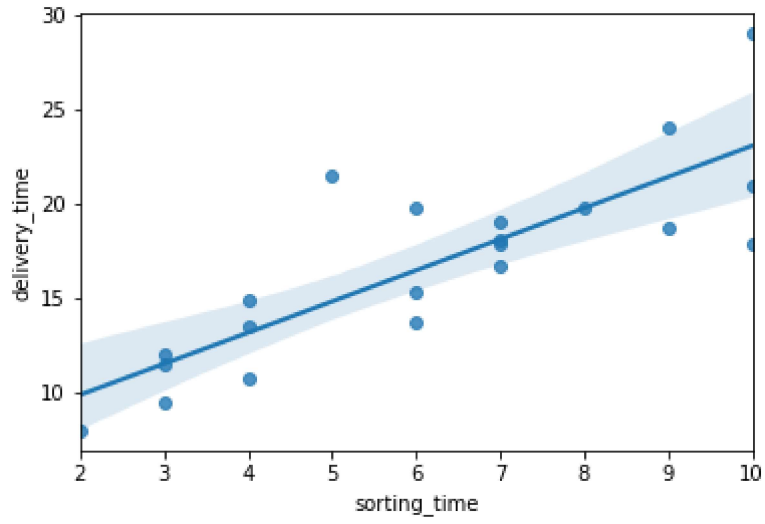
In [10]:
```
corr = data.corr()
#Plot figsize
fig, ax = plt.subplots(figsize=(8, 8))
#Generate Heat Map, allow annotations and place floats in map
sns.heatmap(corr, cmap='magma', annot=True, fmt=".2f")

plt.xticks(range(len(corr.columns)), corr.columns);

plt.yticks(range(len(corr.columns)), corr.columns)

plt.show()
```



Fitting a Linear Regression Model

In [15]:
```
import statsmodels.formula.api as smf
model = smf.ols("data['delivery_time']~data['sorting_time']",data = data).fit()
```

In [16]:
```
sns.regplot(y="delivery_time", x="sorting_time", data=data);
```



In [17]:
```
#Coefficients
model.params
```

Out[17]:
```
Intercept               6.582734
data['sorting_time']    1.649020
dtype: float64
```

In [18]:
```
#t and p-Values
print(model.tvalues, '\n', model.pvalues)
```

```
Intercept               3.823349
data['sorting_time']    6.387447
dtype: float64
 Intercept               0.001147
data['sorting_time']    0.000004
dtype: float64
```

In [19]:
```
#R squared values
(model.rsquared,model.rsquared_adj)
```

Out[19]: (0.6822714748417231, 0.6655489208860244)

In [20]:
```
model.summary()
```

Out[20]:

OLS Regression Results

Dep. Variable:	data['delivery_time']	**R-squared:**	0.682
Model:	OLS	**Adj. R-squared:**	0.666
Method:	Least Squares	**F-statistic:**	40.80
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	3.98e-06
Time:	17:41:27	**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
data['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.

Checking predictions of Model by equation

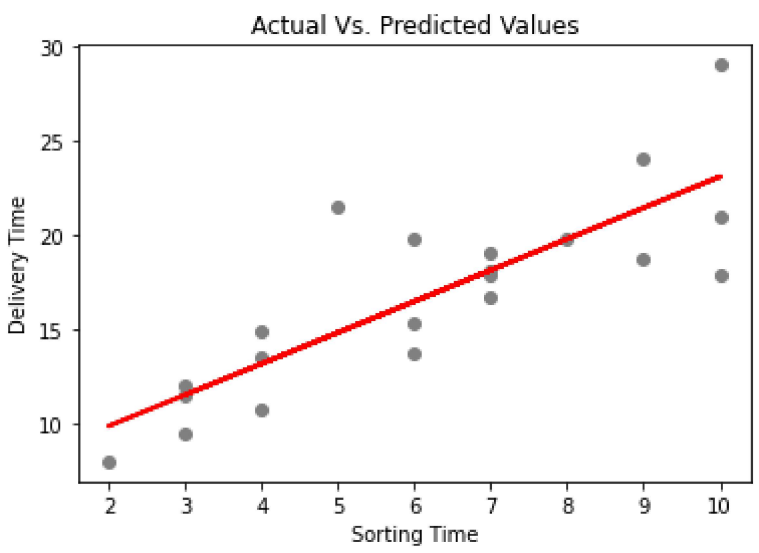
From the result B0 = 6.582734 and B1 = 1.649020, so our linear regression equation will become Y = B0 + B1X
Y = 6.582734 + 1.649020*X

In [25]:
```
predicted_time = model.predict(data.sorting_time)
pred_df = pd.DataFrame({'delivery_time_Actual' : data.delivery_time, 'delivery_time_Predicted' : predicted_t
pred_df
```

Out[25]:

	delivery_time_Actual	delivery_time_Predicted
0	21.00	23.072933
1	13.50	13.178814
2	19.75	16.476853
3	24.00	21.423913
4	29.00	23.072933
5	15.35	16.476853
6	19.00	18.125873
7	9.50	11.529794
8	17.90	23.072933
9	18.75	21.423913
10	19.83	19.774893
11	10.75	13.178814
12	16.68	18.125873
13	11.50	11.529794
14	12.03	11.529794
15	14.88	13.178814
16	13.75	16.476853
17	18.11	18.125873
18	8.00	9.880774
19	17.83	18.125873
20	21.50	14.827833

In [26]:
```
# Plotting Actual Vs. Predicted Values
plt.scatter(data.sorting_time, data.delivery_time,  color='gray')
plt.plot(data.sorting_time, predicted_time, color='red', linewidth=2)
plt.title('Actual Vs. Predicted Values')
plt.xlabel('Sorting Time')
plt.ylabel('Delivery Time')
plt.show()
```



Error calculation

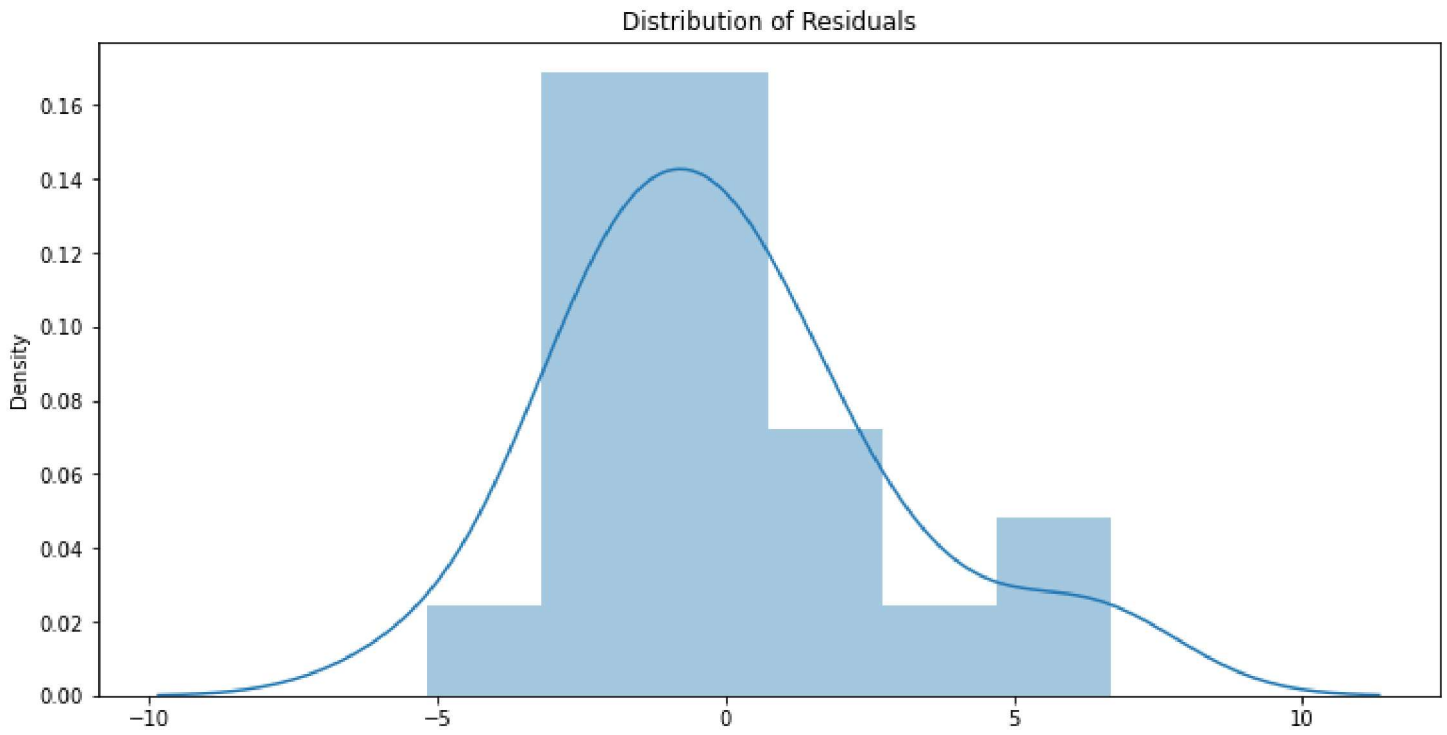
In [28]:
```
from sklearn import metrics
import numpy as np
print('Mean Absolute Error:', metrics.mean_absolute_error(data.delivery_time, predicted_time))
print('Mean Squared Error:', metrics.mean_squared_error(data.delivery_time, predicted_time))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(data.delivery_time, predicted_time)))
print("R^2 Score :          ", metrics.r2_score(data.delivery_time, predicted_time))
```

```
Mean Absolute Error: 2.085740955188266
Mean Squared Error: 7.793311548584062
Root Mean Squared Error: 2.7916503270617654
R^2 Score :           0.6822714748417231
```

Anderson-Darling test for normal distribution to check the normality of the residuals

In [29]:
```
from statsmodels.stats.diagnostic import normal_ad
residuals = data.delivery_time - predicted_time
p_value_thresh=0.05
# Performing the test on the residuals
p_value = normal_ad(residuals)[1]
print('p-value from the test: below 0.05 generally means non-normal:', p_value)

# Reporting the normality of the residuals
if p_value < p_value_thresh:
    print('Residuals are not normally distributed')
else:
    print('Residuals are normally distributed')

# Plotting the residuals distribution
plt.subplots(figsize=(12, 6))
plt.title('Distribution of Residuals')
sns.distplot(residuals)
plt.show()
```

```
p-value from the test: below 0.05 generally means non-normal: 0.1496009945346252
Residuals are normally distributed
```



Predict the new data points

In [30]:
```
newdata=pd.Series([7,8,9,3,4,9,10,5,6,3,2,9,8,6,6,4,3,6,8,7,9])
```

In [31]:
```
data_pred=pd.DataFrame(newdata,columns=['sorting_time'])
```

Delivery time prediction

In [32]:
```
model.predict(data_pred["sorting_time"])
```

Out[32]:
```
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
```

Prediction using other models/transformations

Exponential transformation

log(Y) =α + βX + ε

In [33]:
```
data1 = data.copy()
data1['log_delivery_time'] = np.log(data1.delivery_time)
data1.head()
```

Out[33]:

	delivery_time	sorting_time	log_delivery_time
0	21.00	10	3.044522
1	13.50	4	2.602690
2	19.75	6	2.983153
3	24.00	9	3.178054
4	29.00	10	3.367296

In [34]:
```
exp_model = smf.ols('log_delivery_time ~ sorting_time', data = data1).fit()
```

In [36]:
```
# Coefficients Bo and B1
Bo, B1 = exp_model.params
print("Intercept, Bo: ", Bo)
print("Sorting Time, B1: ", B1)

# Getting tvalue and pvalue
tvalue, pvalue = (exp_model.tvalues, exp_model.pvalues)
print("tvalue: ", tvalue)
print("pvalue: ", pvalue)

# Getting R squared values
r_sq, r_sq_adj = (exp_model.rsquared, exp_model.rsquared_adj)
print("R2: ", r_sq)
print("R2_Adj: ", r_sq_adj)
```

```
Intercept, Bo:  2.121371854893523
Sorting Time, B1:  0.1055515979434444
tvalue:  Intercept       20.601357
sorting_time     6.836088
dtype: float64
pvalue:  Intercept       1.857343e-14
sorting_time    1.592708e-06
dtype: float64
R2:  0.7109478980584187
R2_Adj:  0.6957346295351776
```

Checking predictions of Model by equation

From the result Bo = 2.121371854893523 and B1 = 0.1055515979434444, so our linear regression equation will become log(Y) = Bo + B1X + e

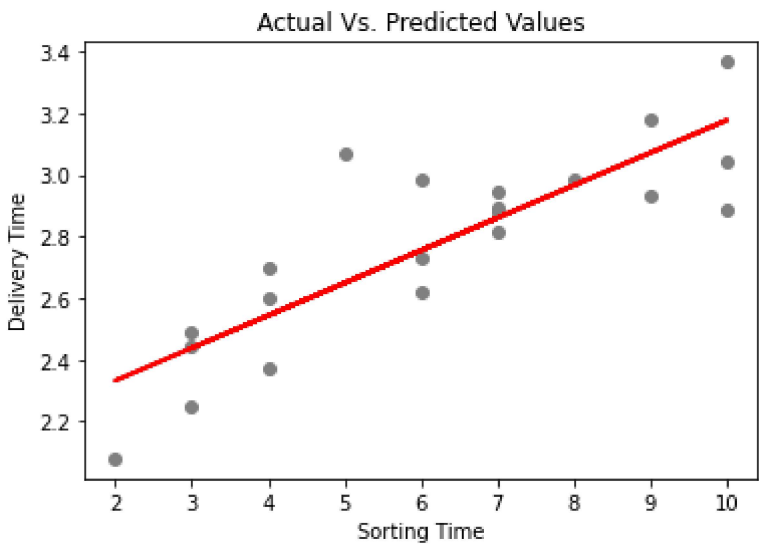
log(Y) = 2.121371854893523 + 0.1055515979434444*X + e

In [38]:
```
predicted_time = exp_model.predict(data1.sorting_time)
pred_df = pd.DataFrame({'delivery_time Actual' : data1.log_delivery_time, 'delivery_time Predicted' : predic
pred_df
```

Out[38]:

	delivery_time Actual	delivery_time Predicted
0	3.044522	3.176888
1	2.602690	2.543578
2	2.983153	2.754681
3	3.178054	3.071336
4	3.367296	3.176888
5	2.731115	2.754681
6	2.944439	2.860233
7	2.251292	2.438027
8	2.884801	3.176888
9	2.931194	3.071336
10	2.987196	2.965785
11	2.374906	2.543578
12	2.814210	2.860233
13	2.442347	2.438027
14	2.487404	2.438027
15	2.700018	2.543578
16	2.621039	2.754681
17	2.896464	2.860233
18	2.079442	2.332475
19	2.880882	2.860233
20	3.068053	2.649130

In [39]:
```
# Plotting Actual Vs. Predicted Values
plt.scatter(data1.sorting_time, data1.log_delivery_time,  color='gray')
plt.plot(data1.sorting_time, predicted_time, color='red', linewidth=2)
plt.title('Actual Vs. Predicted Values')
plt.xlabel('Sorting Time')
plt.ylabel('Delivery Time')
plt.show()
```



error calculation

In [40]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(data1.log_delivery_time, predicted_time))
print('Mean Squared Error:', metrics.mean_squared_error(data1.log_delivery_time, predicted_time))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(data1.log_delivery_time, predicted_time
print("R^2 Score :          ", metrics.r2_score(data1.log_delivery_time, predicted_time))
```

```
Mean Absolute Error: 0.1310730556090564
Mean Squared Error: 0.027876563581789988
Root Mean Squared Error: 0.16696276106302863
R^2 Score :           0.7109478980584187
```

In [41]:
```
pred_df = pd.DataFrame({'Actual': np.exp(data1.log_delivery_time), 'Predicted': np.exp(predicted_time)})
pred_df
```

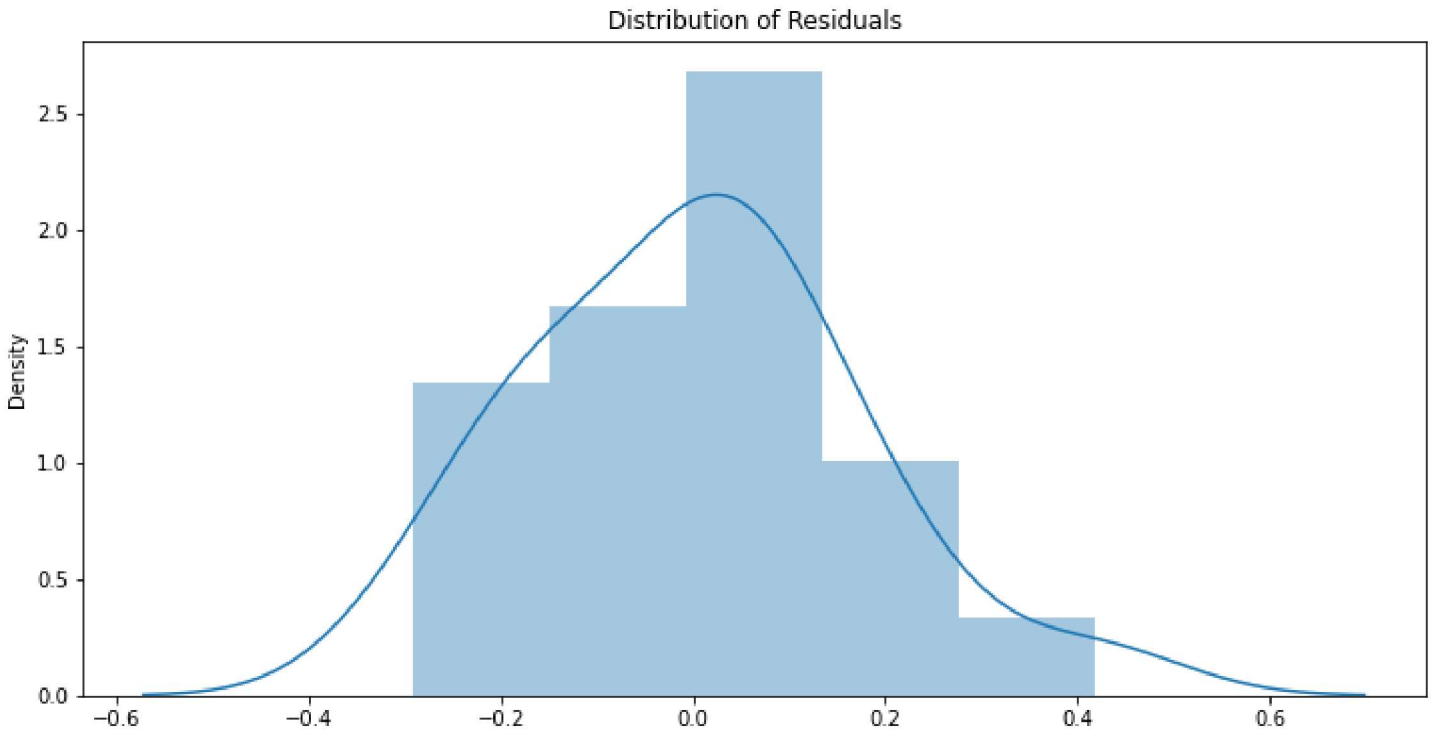
Out[41]:

	Actual	Predicted
0	21.00	23.972032
1	13.50	12.725123
2	19.75	15.716034
3	24.00	21.570707
4	29.00	23.972032
5	15.35	15.716034
6	19.00	17.465597
7	9.50	11.450423
8	17.90	23.972032
9	18.75	21.570707
10	19.83	19.409927
11	10.75	12.725123
12	16.68	17.465597
13	11.50	11.450423
14	12.03	11.450423
15	14.88	12.725123
16	13.75	15.716034
17	18.11	17.465597
18	8.00	10.303411
19	17.83	17.465597
20	21.50	14.141728

Checking Normality of Residuals Using the Anderson-Darling test

In [42]:
```
residuals = data1.log_delivery_time - predicted_time
p_value_thresh=0.05
# Performing the test on the residuals
p_value = normal_ad(residuals)[1]
print('p-value from the test: below 0.05 generally means non-normal:', p_value)

# Reporting the normality of the residuals
if p_value < p_value_thresh:
    print('Residuals are not normally distributed')
else:
    print('Residuals are normally distributed')

# Plotting the residuals distribution
plt.subplots(figsize=(12, 6))
plt.title('Distribution of Residuals')
sns.distplot(residuals)
plt.show()
```

```
p-value from the test: below 0.05 generally means non-normal: 0.8006026322922348
Residuals are normally distributed
```



In [43]:
```
exp_model.summary()
```

Out[43]:

OLS Regression Results

Dep. Variable:	log_delivery_time	**R-squared:**	0.711
Model:	OLS	**Adj. R-squared:**	0.696
Method:	Least Squares	**F-statistic:**	46.73
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	1.59e-06
Time:	18:42:57	**Log-Likelihood:**	7.7920
No. Observations:	21	**AIC:**	-11.58
Df Residuals:	19	**BIC:**	-9.495
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.1214	0.103	20.601	0.000	1.906	2.337
sorting_time	0.1056	0.015	6.836	0.000	0.073	0.138

Omnibus:	1.238	**Durbin-Watson:**	1.325
Prob(Omnibus):	0.538	**Jarque-Bera (JB):**	0.544
Skew:	0.393	**Prob(JB):**	0.762
Kurtosis:	3.067	**Cond. No.**	18.3

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

Square root transformation

$\sqrt{Y} = a + b * X + e$

In [44]:
```
# preparing data to build model
data1['sqrt_delivery_time'] = np.sqrt(data1.delivery_time)
data1.head()
```

Out[44]:

	delivery_time	sorting_time	log_delivery_time	sqrt_delivery_time
0	21.00	10	3.044522	4.582576
1	13.50	4	2.602690	3.674235
2	19.75	6	2.983153	4.444097
3	24.00	9	3.178054	4.898979
4	29.00	10	3.367296	5.385165

In [45]:
```
sqrt_model = smf.ols('sqrt_delivery_time ~ sorting_time', data = data1).fit()
```

In [46]:
```
Bo, B1 = sqrt_model.params
print("Intercept, Bo: ", Bo)
print("Sorting Time, B1: ", B1)

# Getting tvalue and pvalue
tvalue, pvalue = (sqrt_model.tvalues, sqrt_model.pvalues)
print("tvalue: ", tvalue)
print("pvalue: ", pvalue)

# Getting R squared values
r_sq, r_sq_adj = (sqrt_model.rsquared, sqrt_model.rsquared_adj)
print("R2: ", r_sq)
print("R2_Adj: ", r_sq_adj)
```

```
Intercept, Bo:  2.772731039345033
Sorting Time, B1:  0.20663181804577496
tvalue:  Intercept       13.527452
sorting_time     6.723095
dtype: float64
pvalue:  Intercept       3.339366e-11
sorting_time    2.001215e-06
dtype: float64
R2:  0.704049871541756
R2_Adj:  0.6884735489913221
```

Checking predictions of Model by equation

From the result Bo = 2.7727310393450324 and B1 = 0.20663181804577496, so our linear regression equation will become √Y = Bo + B1X + e

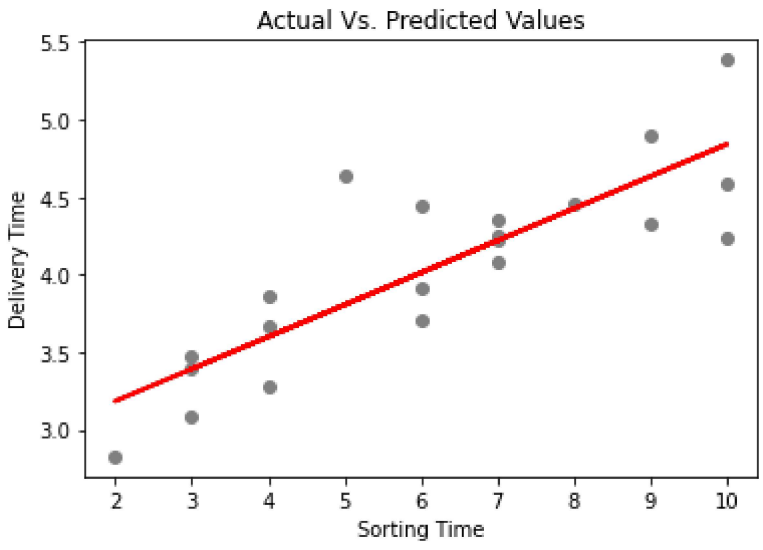
√Y = 2.7727310393450324 + 0.20663181804577496*X + e

In [47]:
```
predicted_time = sqrt_model.predict(data1.sorting_time)
pred_df = pd.DataFrame({'Delivery_Time_Actual' : data1.sqrt_delivery_time, 'Delivery_Time_Predicted' : predi
pred_df
```

Out[47]:

	Delivery_Time_Actual	Delivery_Time_Predicted
0	4.582576	4.839049
1	3.674235	3.599258
2	4.444097	4.012522
3	4.898979	4.632417
4	5.385165	4.839049
5	3.917908	4.012522
6	4.358899	4.219154
7	3.082207	3.392626
8	4.230839	4.839049
9	4.330127	4.632417
10	4.453089	4.425786
11	3.278719	3.599258
12	4.084116	4.219154
13	3.391165	3.392626
14	3.468429	3.392626
15	3.857460	3.599258
16	3.708099	4.012522
17	4.255585	4.219154
18	2.828427	3.185995
19	4.222558	4.219154
20	4.636809	3.805890

In [48]:
```
# Plotting Actual Vs. Predicted Values
plt.scatter(data1.sorting_time, data1.sqrt_delivery_time,  color='gray')
plt.plot(data1.sorting_time, predicted_time, color='red', linewidth=2)
plt.title('Actual Vs. Predicted Values')
plt.xlabel('Sorting Time')
plt.ylabel('Delivery Time')
plt.show()
```



Error calculation

In [49]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(data1.sqrt_delivery_time, predicted_time))
print('Mean Squared Error:', metrics.mean_squared_error(data1.sqrt_delivery_time, predicted_time))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(data1.sqrt_delivery_time, predicted_tim
print("R^2 Score :          ", metrics.r2_score(data1.sqrt_delivery_time, predicted_time))
```

```
Mean Absolute Error: 0.25628921927628917
Mean Squared Error: 0.11045382417324963
Root Mean Squared Error: 0.33234594050965877
R^2 Score :           0.7040498715417561
```

In [50]:
```
pred_df = pd.DataFrame({'Actual': (data1.sqrt_delivery_time)**2, 'Predicted': (predicted_time)**2})
pred_df
```

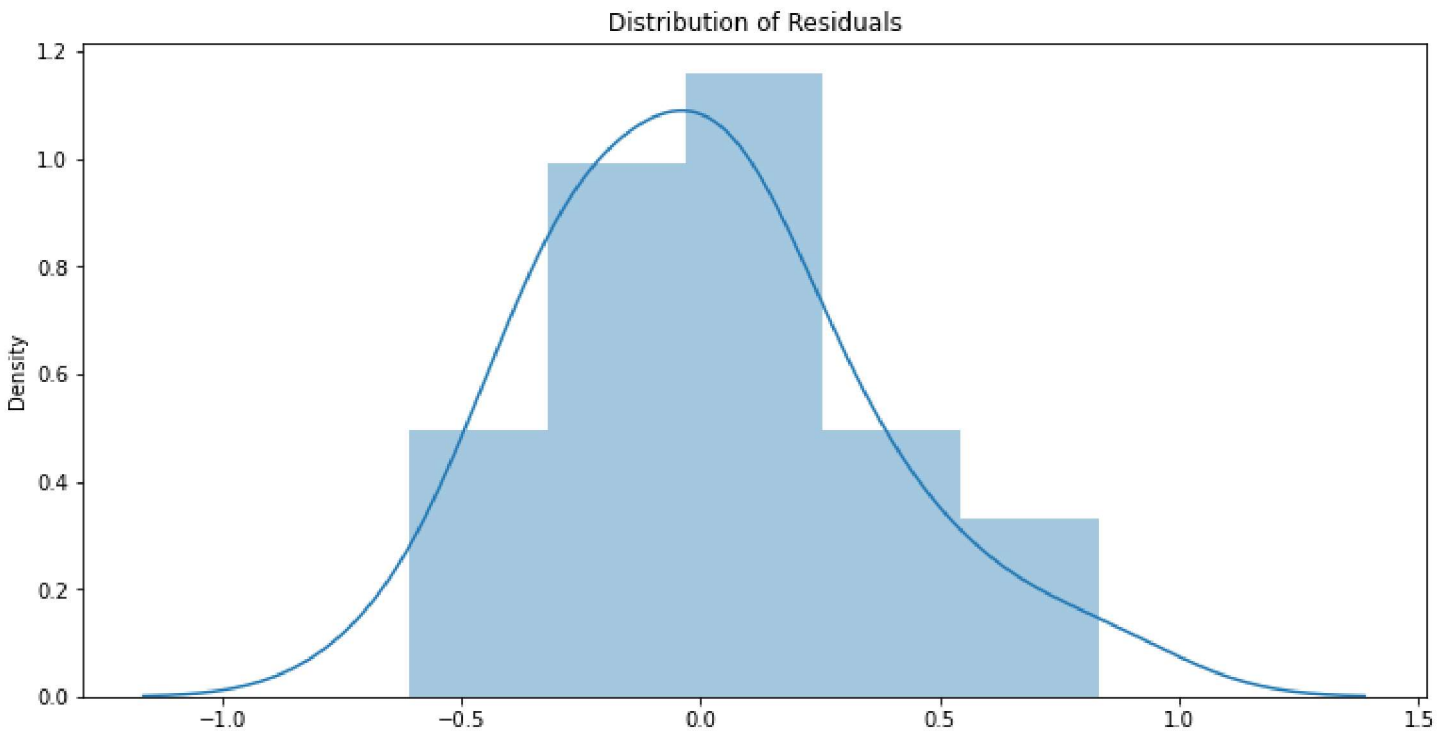
Out[50]:

	Actual	Predicted
0	21.00	23.416397
1	13.50	12.954660
2	19.75	16.100332
3	24.00	21.459291
4	29.00	23.416397
5	15.35	16.100332
6	19.00	17.801258
7	9.50	11.509915
8	17.90	23.416397
9	18.75	21.459291
10	19.83	19.587578
11	10.75	12.954660
12	16.68	17.801258
13	11.50	11.509915
14	12.03	11.509915
15	14.88	12.954660
16	13.75	16.100332
17	18.11	17.801258
18	8.00	10.150562
19	17.83	17.801258
20	21.50	14.484800

Anderson-Darling test

In [51]:
```
residuals = data1.sqrt_delivery_time - predicted_time
p_value_thresh=0.05
# Performing the test on the residuals
p_value = normal_ad(residuals)[1]
print('p-value from the test: below 0.05 generally means non-normal:', p_value)

# Reporting the normality of the residuals
if p_value < p_value_thresh:
    print('Residuals are not normally distributed')
else:
    print('Residuals are normally distributed')

# Plotting the residuals distribution
plt.subplots(figsize=(12, 6))
plt.title('Distribution of Residuals')
sns.distplot(residuals)
plt.show()
```

```
p-value from the test: below 0.05 generally means non-normal: 0.4177469231053469
Residuals are normally distributed
```



In [52]:
```
sqrt_model.summary()
```

Out[52]:

OLS Regression Results

Dep. Variable:	sqrt_delivery_time	**R-squared:**	0.704
Model:	OLS	**Adj. R-squared:**	0.688
Method:	Least Squares	**F-statistic:**	45.20
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	2.00e-06
Time:	20:05:56	**Log-Likelihood:**	-6.6646
No. Observations:	21	**AIC:**	17.33
Df Residuals:	19	**BIC:**	19.42
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.7727	0.205	13.527	0.000	2.344	3.202
sorting_time	0.2066	0.031	6.723	0.000	0.142	0.271

Omnibus:	2.228	**Durbin-Watson:**	1.258
Prob(Omnibus):	0.328	**Jarque-Bera (JB):**	1.195
Skew:	0.580	**Prob(JB):**	0.550
Kurtosis:	3.142	**Cond. No.**	18.3

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

Logarithmic transformation

Y = a + b * log(x)

In [53]:
```
data1 = pd.DataFrame.copy(data)
data1['log_sorting_time'] = np.log(data1.sorting_time)
data1.head()
```

Out[53]:

	delivery_time	sorting_time	log_sorting_time
0	21.00	10	2.302585
1	13.50	4	1.386294
2	19.75	6	1.791759
3	24.00	9	2.197225
4	29.00	10	2.302585

In [54]:
```
# Building model
log_model = smf.ols('delivery_time ~ log_sorting_time', data = data1).fit()
```

In [56]:
```
# Coefficients Bo and B1
B0, B1 = log_model.params
print("Intercept, B0: ", B0)
print("Sorting Time, B1: ", B1)

# Getting tvalue and pvalue
tvalue, pvalue = (log_model.tvalues, log_model.pvalues)
print("tvalue: ", tvalue)
print("pvalue: ", pvalue)

# Getting R squared values
r_sq, r_sq_adj = (log_model.rsquared, log_model.rsquared_adj)
print("R2: ", r_sq)
print("R2_Adj: ", r_sq_adj)
```

```
Intercept, B0:  1.1596835115465818
Sorting Time, B1:  9.04341345820576
tvalue:  Intercept           0.472460
log_sorting_time    6.586789
dtype: float64
pvalue:  Intercept           0.641980
log_sorting_time    0.000003
dtype: float64
R2:  0.6954434611324223
R2_Adj:  0.6794141696130761
```

Checking predictions of Model by equation

From the result B0 = 1.1596835115465813 and B1 = 9.043413458205762, so our linear regression equation will become Y = B0 + B1*log(X)*

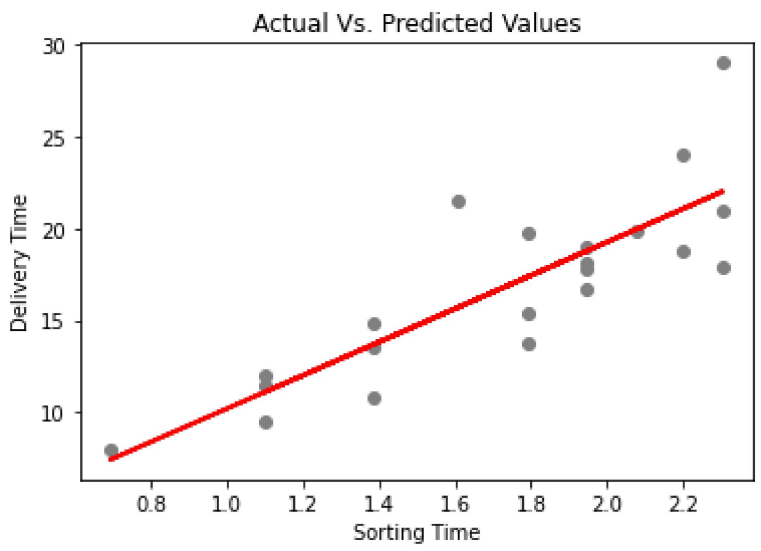
*Y = 1.1596835115465813 + 9.043413458205762*log(X)

In [57]:
```
predicted_time = log_model.predict(data1.log_sorting_time)
pred_df = pd.DataFrame({'delivery time actual' : data1.delivery_time, 'delivery time Predicted' : predicted_
pred_df
```

Out[57]:

	delivery time actual	delivery time Predicted
0	21.00	21.982913
1	13.50	13.696517
2	19.75	17.363305
3	24.00	21.030094
4	29.00	21.982913
5	15.35	17.363305
6	19.00	18.757354
7	9.50	11.094889
8	17.90	21.982913
9	18.75	21.030094
10	19.83	19.964933
11	10.75	13.696517
12	16.68	18.757354
13	11.50	11.094889
14	12.03	11.094889
15	14.88	13.696517
16	13.75	17.363305
17	18.11	18.757354
18	8.00	7.428100
19	17.83	18.757354
20	21.50	15.714496

In [58]:
```
# Plotting Actual Vs. Predicted Values
plt.scatter(data1.log_sorting_time, data1.delivery_time,  color='gray')
plt.plot(data1.log_sorting_time, predicted_time, color='red', linewidth=2)
plt.title('Actual Vs. Predicted Values')
plt.xlabel('Sorting Time')
plt.ylabel('Delivery Time')
plt.show()
```



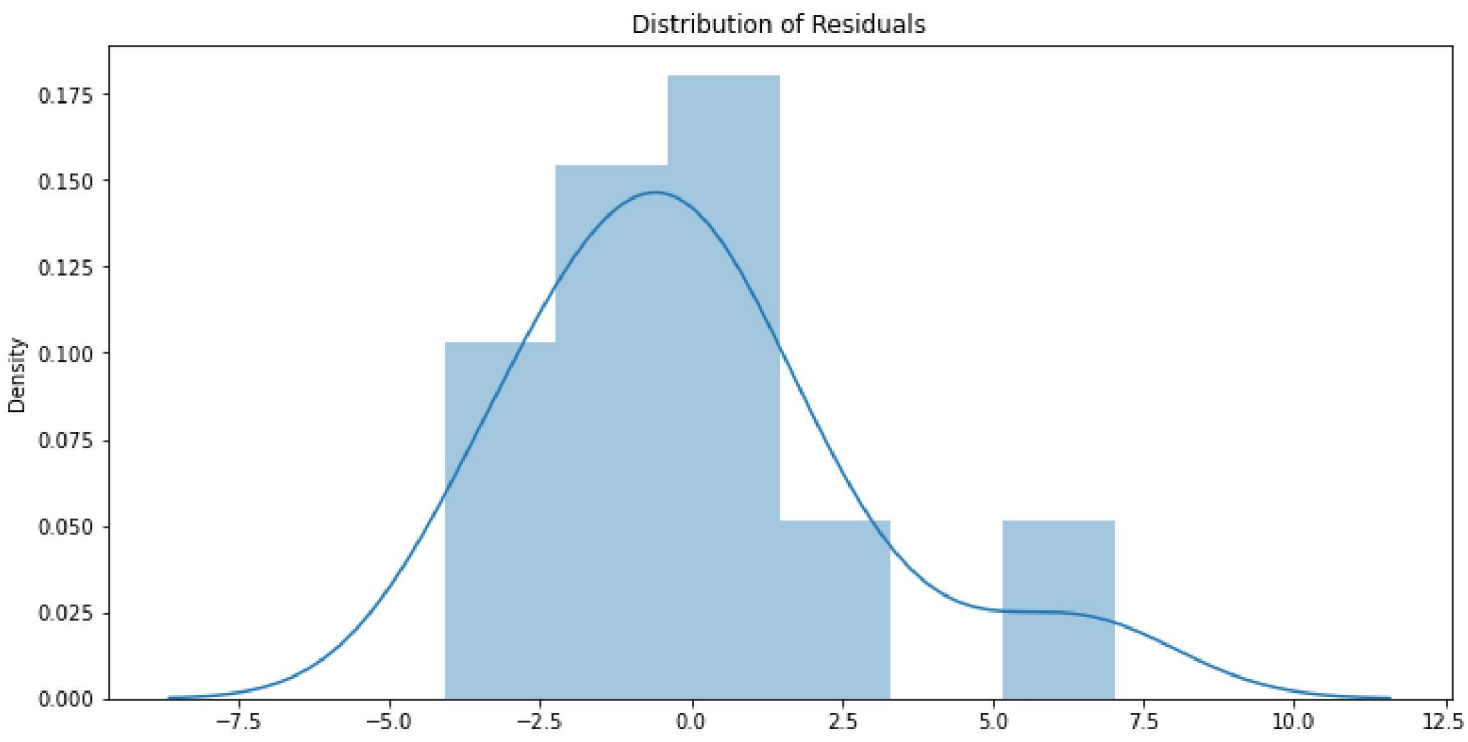
In [62]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(data1.delivery_time, predicted_time))
print('Mean Squared Error:', metrics.mean_squared_error(data1.delivery_time, predicted_time))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(data1.delivery_time, predicted_time)))
print("R^2 Score :          ", metrics.r2_score(data1.delivery_time, predicted_time))
```

```
Mean Absolute Error: 2.0473757067928986
Mean Squared Error: 7.470226320948427
Root Mean Squared Error: 2.7331714766820663
R^2 Score :           0.6954434611324223
```

Anderson-Darling test

In [60]:
```
residuals = data1.delivery_time - predicted_time
p_value_thresh=0.05
# Performing the test on the residuals
p_value = normal_ad(residuals)[1]
print('p-value from the test: below 0.05 generally means non-normal:', p_value)

# Reporting the normality of the residuals
if p_value < p_value_thresh:
    print('Residuals are not normally distributed')
else:
    print('Residuals are normally distributed')

# Plotting the residuals distribution
plt.subplots(figsize=(12, 6))
plt.title('Distribution of Residuals')
sns.distplot(residuals)
plt.show()
```

```
p-value from the test: below 0.05 generally means non-normal: 0.18201104321963033
Residuals are normally distributed
```



In [61]:
```
log_model.summary()
```

Out[61]:

OLS Regression Results

Dep. Variable:	delivery_time	**R-squared:**	0.695
Model:	OLS	**Adj. R-squared:**	0.679
Method:	Least Squares	**F-statistic:**	43.39
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	2.64e-06
Time:	20:19:39	**Log-Likelihood:**	-50.912
No. Observations:	21	**AIC:**	105.8
Df Residuals:	19	**BIC:**	107.9
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	1.1597	2.455	0.472	0.642	-3.978	6.297
log_sorting_time	9.0434	1.373	6.587	0.000	6.170	11.917

Omnibus:	5.552	**Durbin-Watson:**	1.427
Prob(Omnibus):	0.062	**Jarque-Bera (JB):**	3.481
Skew:	0.946	**Prob(JB):**	0.175
Kurtosis:	3.628	**Cond. No.**	9.08

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

Summary of various models

In [64]:
```
#Linear regression Model
model.summary()
```

Out[64]:

OLS Regression Results

Dep. Variable:	data['delivery_time']	**R-squared:**	0.682
Model:	OLS	**Adj. R-squared:**	0.666
Method:	Least Squares	**F-statistic:**	40.80
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	3.98e-06
Time:	20:23:15	**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
data['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 [65]:
```
# LOg transformation
log_model.summary()
```

Out[65]:

OLS Regression Results

Dep. Variable:	delivery_time	**R-squared:**	0.695
Model:	OLS	**Adj. R-squared:**	0.679
Method:	Least Squares	**F-statistic:**	43.39
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	2.64e-06
Time:	20:23:29	**Log-Likelihood:**	-50.912
No. Observations:	21	**AIC:**	105.8
Df Residuals:	19	**BIC:**	107.9
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.1597	2.455	0.472	0.642	-3.978	6.297
log_sorting_time	9.0434	1.373	6.587	0.000	6.170	11.917

Omnibus:	5.552	**Durbin-Watson:**	1.427
Prob(Omnibus):	0.062	**Jarque-Bera (JB):**	3.481
Skew:	0.946	**Prob(JB):**	0.175
Kurtosis:	3.628	**Cond. No.**	9.08

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

In [66]:
```
# Exponential
exp_model.summary()
```

Out[66]:

OLS Regression Results

Dep. Variable:	log_delivery_time	**R-squared:**	0.711
Model:	OLS	**Adj. R-squared:**	0.696
Method:	Least Squares	**F-statistic:**	46.73
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	1.59e-06
Time:	20:23:43	**Log-Likelihood:**	7.7920
No. Observations:	21	**AIC:**	-11.58
Df Residuals:	19	**BIC:**	-9.495
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.1214	0.103	20.601	0.000	1.906	2.337
sorting_time	0.1056	0.015	6.836	0.000	0.073	0.138

Omnibus:	1.238	**Durbin-Watson:**	1.325
Prob(Omnibus):	0.538	**Jarque-Bera (JB):**	0.544
Skew:	0.393	**Prob(JB):**	0.762
Kurtosis:	3.067	**Cond. No.**	18.3

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

In [67]:
```
# Square root
sqrt_model.summary()
```

Out[67]:

OLS Regression Results

Dep. Variable:	sqrt_delivery_time	**R-squared:**	0.704
Model:	OLS	**Adj. R-squared:**	0.688
Method:	Least Squares	**F-statistic:**	45.20
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	2.00e-06
Time:	20:24:07	**Log-Likelihood:**	-6.6646
No. Observations:	21	**AIC:**	17.33
Df Residuals:	19	**BIC:**	19.42
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	2.7727	0.205	13.527	0.000	2.344	3.202
sorting_time	0.2066	0.031	6.723	0.000	0.142	0.271

Omnibus:	2.228	**Durbin-Watson:**	1.258
Prob(Omnibus):	0.328	**Jarque-Bera (JB):**	1.195
Skew:	0.580	**Prob(JB):**	0.550
Kurtosis:	3.142	**Cond. No.**	18.3

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