Problem Statement: -

A logistics company recorded the time taken for delivery and the time taken for the sorting of the items for delivery. Build a Simple Linear Regression model to find the relationship between delivery time and sorting time with delivery time as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.

In [31]:

```
# Importing necessary libraries
import pandas as pd # deals with data frame
import numpy as np  # deals with numerical values
import seaborn as sns
import matplotlib.pyplot as plt

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

df.describe()

df.columns.values[0] = "DT"
df.columns.values[1] = "ST"
df.columns
```

Out[31]:

```
Index(['DT', 'ST'], dtype='object')
```

Exploratory data analysis:

In [32]:

```
# 1. Measures of central tendency
# 2. Measures of dispersion
# 3. Third moment business decision
# 4. Fourth moment business decision
# 5. Probability distributions of variables
# 6. Graphical representations (Histogram, Box plot, Dot plot, Stem & Leaf plot,




EDA ={"column ": df.columns,
      "mean": df.mean(),
      "median":df.median(),
      "mode":df.mode(),
      "standard deviation": df.std(),
      "variance":df.var(),
      "skewness":df.skew(),
      "kurtosis":df.kurt()}


EDA
```

Out[32]:

```
{'column ': Index(['DT', 'ST'], dtype='object'),
 'mean': DT    16.790952
 ST     6.190476
 dtype: float64,
 'median': DT    17.83
 ST     6.00
 dtype: float64,
 'mode':        DT   ST
 0    8.00  7.0
 1    9.50  NaN
 2   10.75  NaN
 3   11.50  NaN
 4   12.03  NaN
 5   13.50  NaN
 6   13.75  NaN
 7   14.88  NaN
 8   15.35  NaN
 9   16.68  NaN
 10  17.83  NaN
 11  17.90  NaN
 12  18.11  NaN
 13  18.75  NaN
 14  19.00  NaN
 15  19.75  NaN
 16  19.83  NaN
 17  21.00  NaN
 18  21.50  NaN
 19  24.00  NaN
 20  29.00  NaN,
 'standard deviation': DT    5.074901
 ST    2.542028
 dtype: float64,
 'variance': DT    25.754619
```

```
ST      6.461905
dtype: float64,
'skewness': DT    0.352390
ST    0.047115
dtype: float64,
'kurtosis': DT    0.317960
ST   -1.148455
dtype: float64}
```

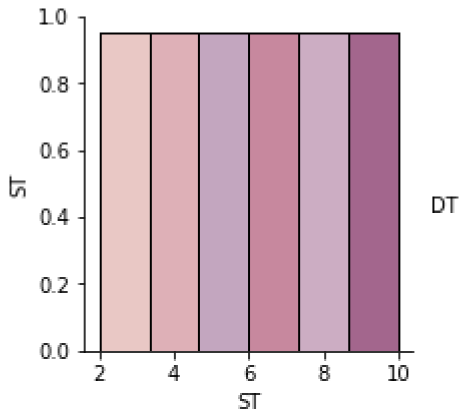
In [33]:

```
plt.figure(figsize=(30, 30))
sns.pairplot(df, hue='DT', height=3, diag_kind='hist')
```

Out[33]:

```
<seaborn.axisgrid.PairGrid at 0x20654bc4fd0>

<Figure size 2160x2160 with 0 Axes>
```

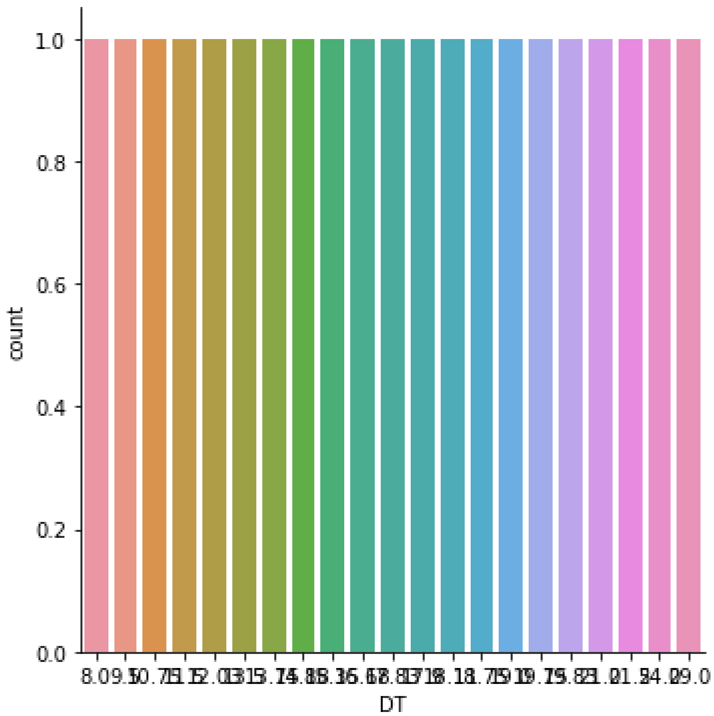


In [34]:

```
#yes or no count
sns.catplot('DT', data=df, kind='count')
```

```
D:\anconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the
following variable as a keyword arg: x. From version 0.12, the only valid posit
ional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```

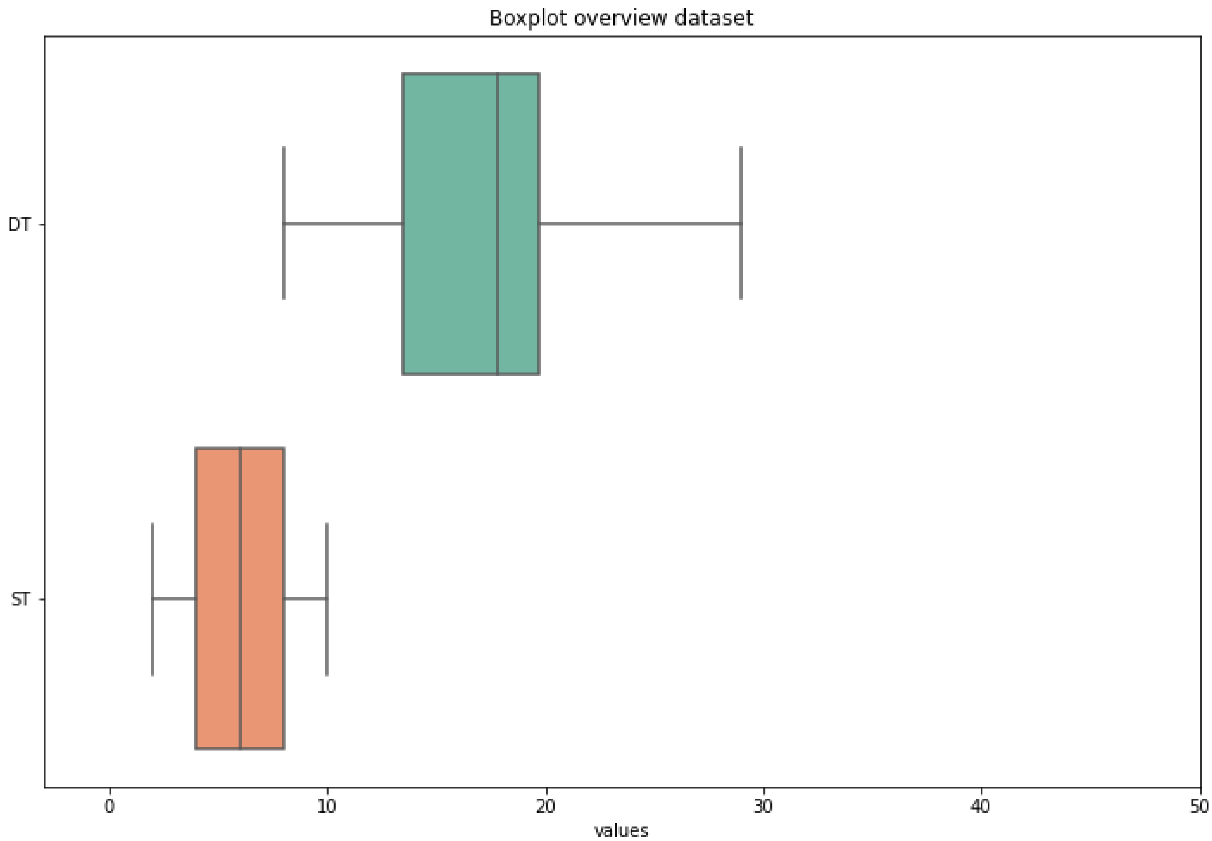
Out[34]: <seaborn.axisgrid.FacetGrid at 0x2065453df40>



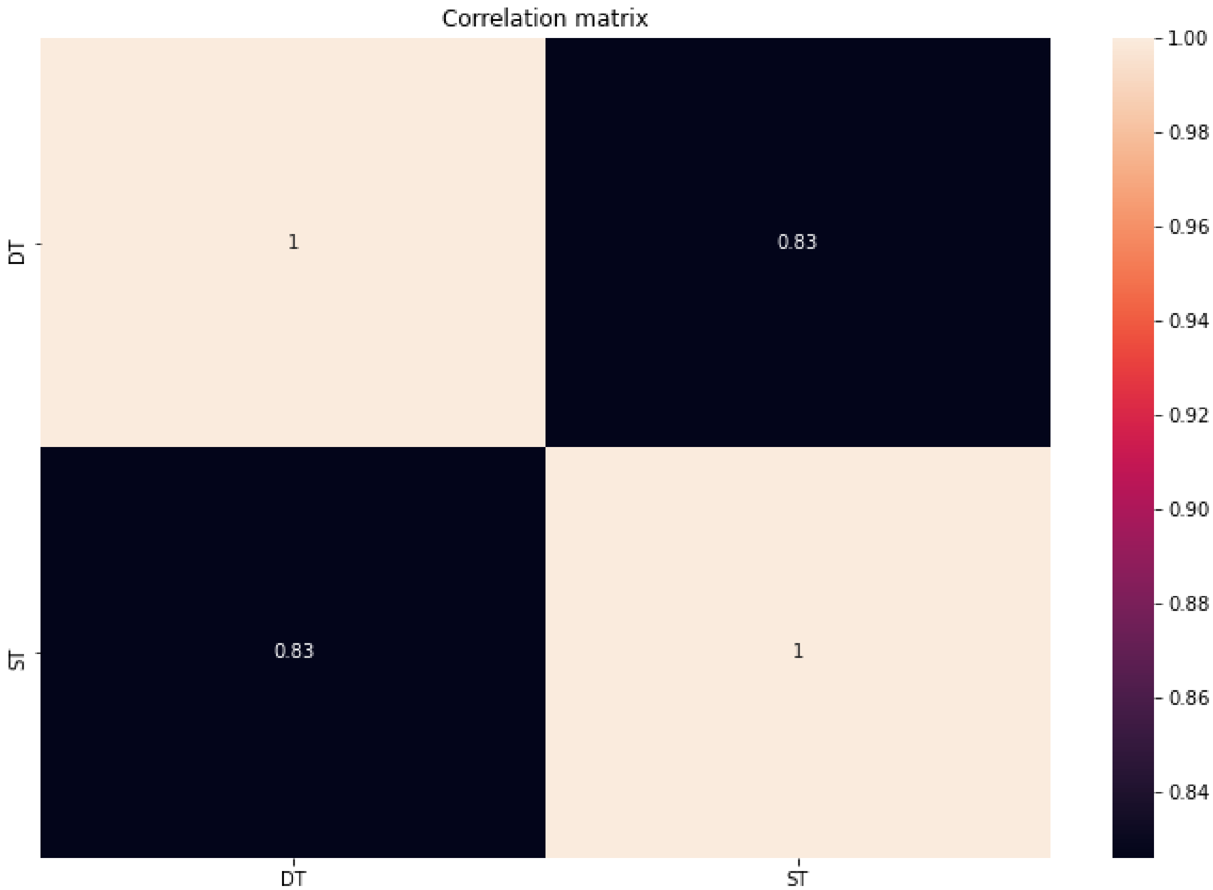
In [35]:

```
import matplotlib.pyplot as plt

plt.figure(figsize = (12, 8))
ax = sns.boxplot(data = df, orient = 'h', palette = 'Set2')
plt.title('Boxplot overview dataset')
plt.xlabel('values')
plt.xlim(-3, 50)
plt.show()
```

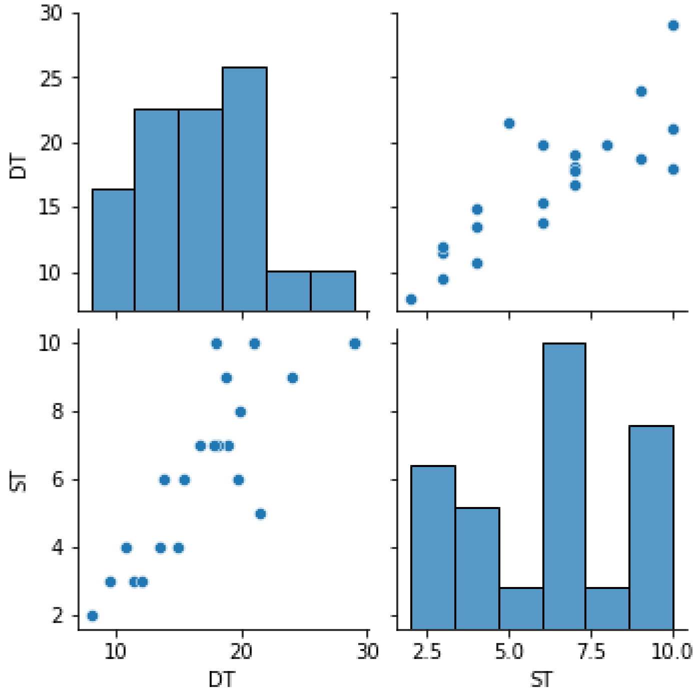


In [36]:
```
plt.figure(figsize = (12, 8))
sns.heatmap(df.corr(), annot = True)
plt.title('Correlation matrix')
plt.show()
```



In [37]:
```
sns.pairplot(df)
```

Out[37]: <seaborn.axisgrid.PairGrid at 0x20654bcfb20>



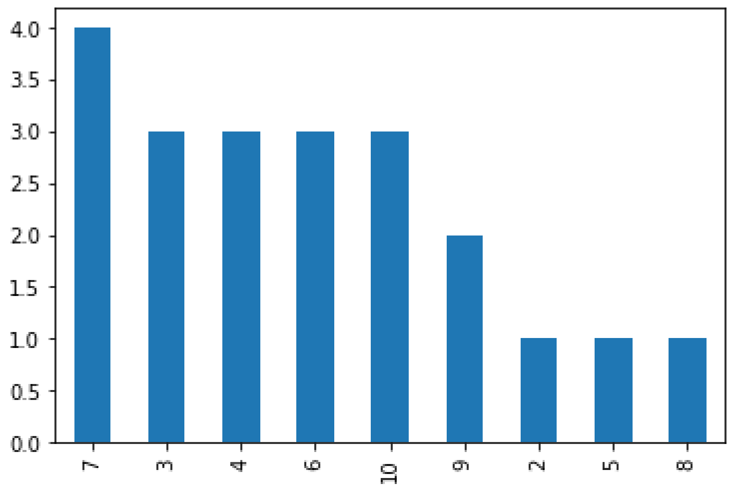
In [38]:

```
#Bar plot
df['ST'].value_counts().plot.bar()

# 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()

cal = df
```

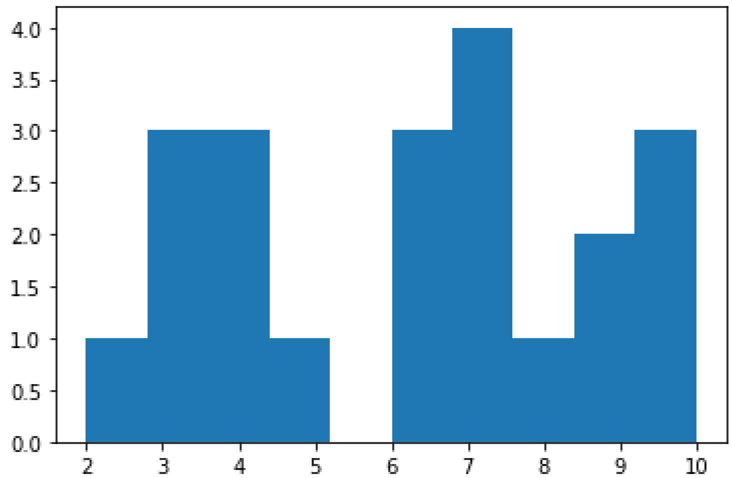


In [39]:

```
#Graphical Representation
import matplotlib.pyplot as plt # mostly used for visualization purposes
```

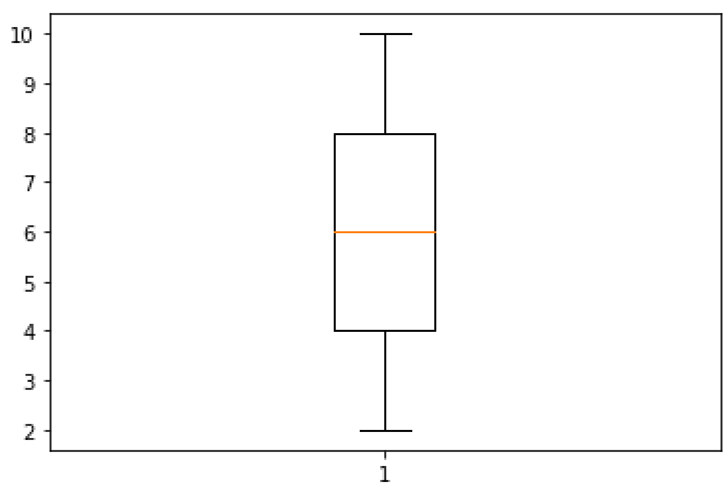
In [40]:
```
plt.hist(cal.ST) #histogram
```

Out[40]:
```
(array([1., 3., 3., 1., 0., 3., 4., 1., 2., 3.]),
 array([ 2. ,  2.8,  3.6,  4.4,  5.2,  6. ,  6.8,  7.6,  8.4,  9.2, 10. ]),
 <BarContainer object of 10 artists>)
```



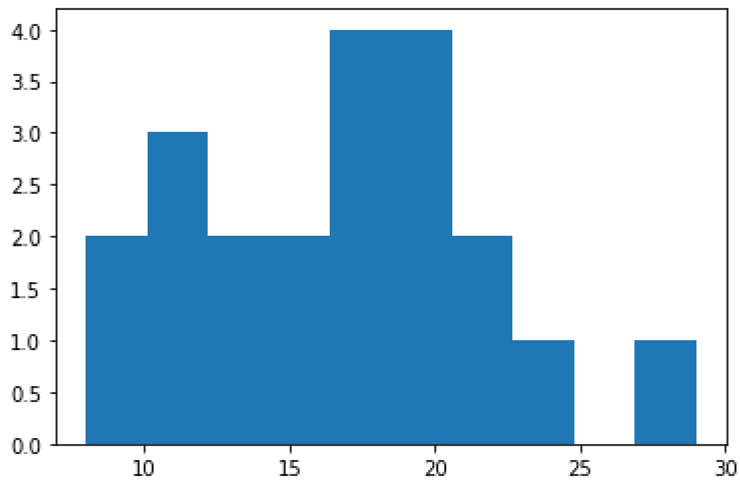
In [41]:
```
plt.boxplot(cal.ST) #boxplot
```

Out[41]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x206539fa910>,
  <matplotlib.lines.Line2D at 0x206539fac70>],
 'caps': [<matplotlib.lines.Line2D at 0x206539fafd0>,
  <matplotlib.lines.Line2D at 0x20653a07370>],
 'boxes': [<matplotlib.lines.Line2D at 0x206539fa5b0>],
 'medians': [<matplotlib.lines.Line2D at 0x20653a076d0>],
 'fliers': [<matplotlib.lines.Line2D at 0x20653a07a30>],
 'means': []}
```



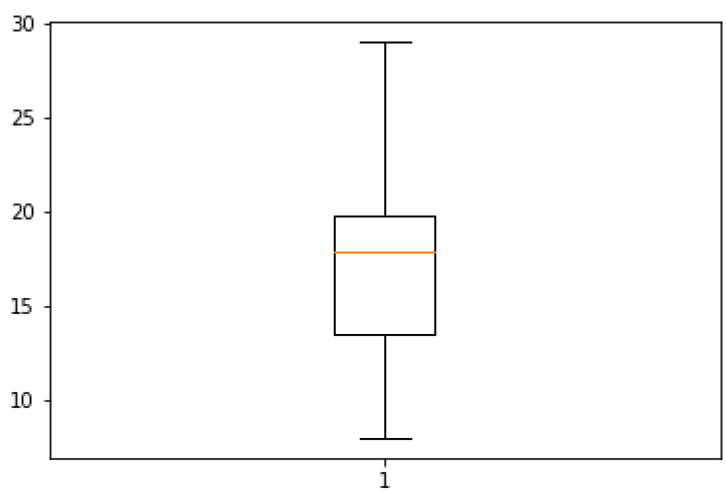
In [42]:
```
plt.hist(cal.DT) #histogram
```

Out[42]:
```
(array([2., 3., 2., 2., 4., 4., 2., 1., 0., 1.]),
 array([ 8. , 10.1, 12.2, 14.3, 16.4, 18.5, 20.6, 22.7, 24.8, 26.9, 29. ]),
 <BarContainer object of 10 artists>)
```



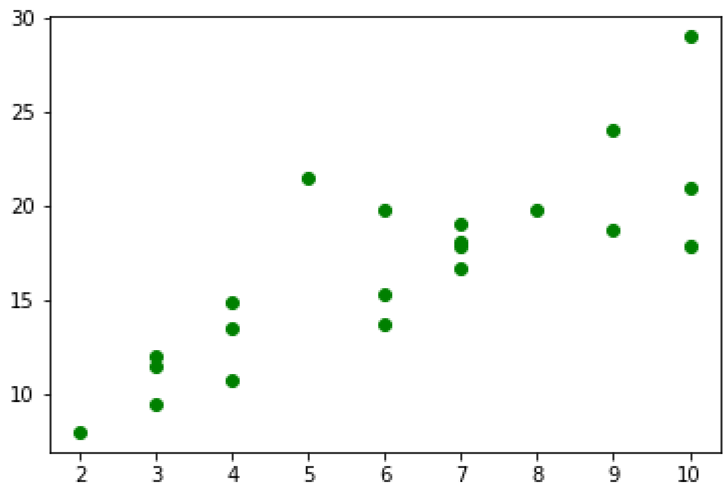
In [43]:
```
plt.boxplot(cal.DT) #boxplot
```

Out[43]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x20656023a90>,
  <matplotlib.lines.Line2D at 0x20656023df0>],
 'caps': [<matplotlib.lines.Line2D at 0x20656032190>,
  <matplotlib.lines.Line2D at 0x206560324f0>],
 'boxes': [<matplotlib.lines.Line2D at 0x20656023730>],
 'medians': [<matplotlib.lines.Line2D at 0x20656032850>],
 'fliers': [<matplotlib.lines.Line2D at 0x20656032bb0>],
 'means': []}
```



In [44]:
```
# Scatter plot
plt.scatter(x = cal.ST, y = cal.DT, color = 'green')
```

Out[44]:
```
<matplotlib.collections.PathCollection at 0x2065607d700>
```



In [45]:
```
# correlation
np.corrcoef(cal.ST, cal.DT)
```

Out[45]:
```
array([[1.        , 0.82599726],
       [0.82599726, 1.        ]])
```

In [46]:
```
# 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.ST, cal.DT)[0, 1]
cov_output
```

Out[46]:
```
10.655809523809523
```

In [ ]:

Data Modeling

In [47]:

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

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

Out[47]:

OLS Regression Results

Dep. Variable:	DT	**R-squared:**	0.682
Model:	OLS	**Adj. R-squared:**	0.666
Method:	Least Squares	**F-statistic:**	40.80
Date:	Fri, 18 Jun 2021	**Prob (F-statistic):**	3.98e-06
Time:	23:46:02	**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
ST	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

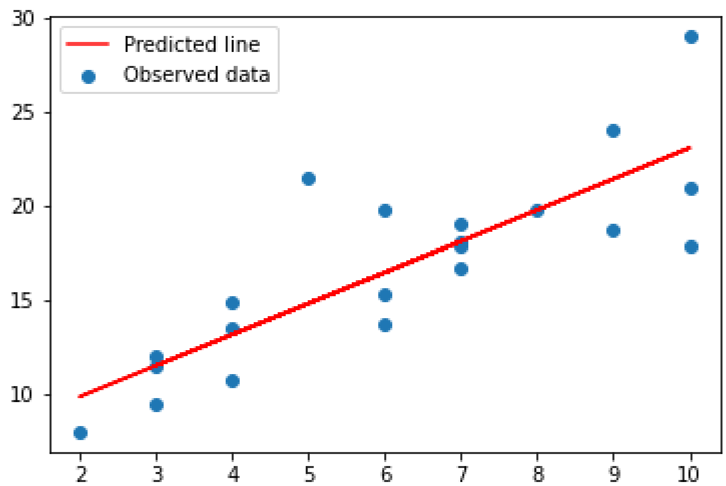
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [48]:

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

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

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



Out[48]: 2.7916503270617654

In [49]:

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

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

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

Out[49]:

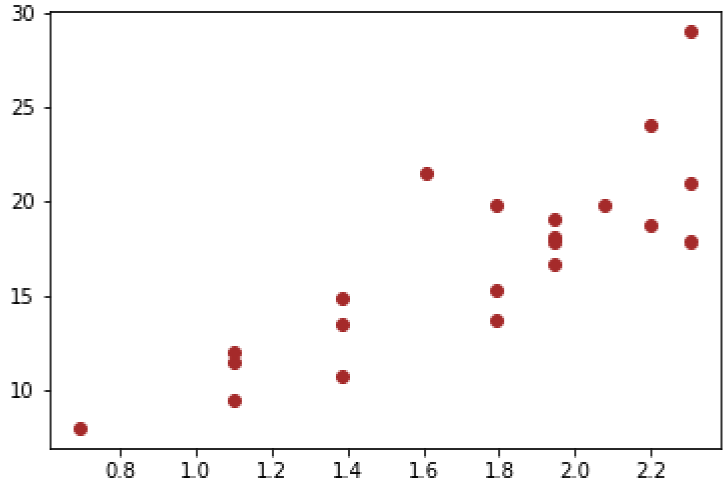
OLS Regression Results

Dep. Variable:	DT	**R-squared:**	0.695
Model:	OLS	**Adj. R-squared:**	0.679
Method:	Least Squares	**F-statistic:**	43.39
Date:	Fri, 18 Jun 2021	**Prob (F-statistic):**	2.64e-06
Time:	23:46:02	**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
np.log(ST)	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.

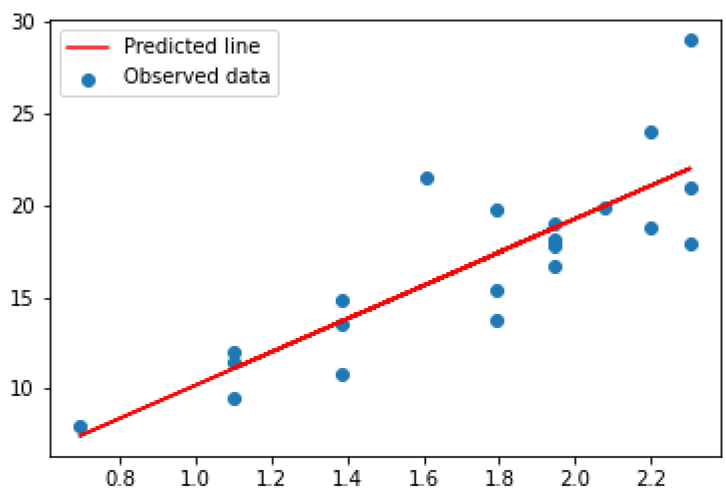


In [50]:

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

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

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



Out[50]: 2.7331714766820663

In [51]:

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

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

model3 = smf.ols('np.log(DT) ~ ST', data = cal).fit()
model3.summary()
```

Out[51]:

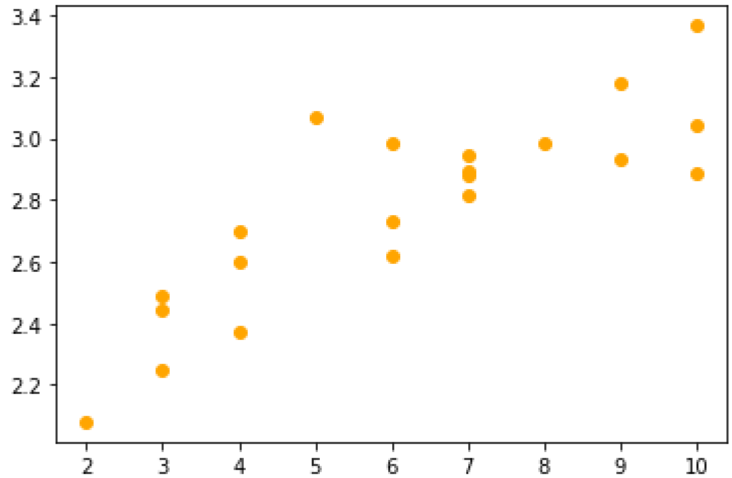
OLS Regression Results

Dep. Variable:	np.log(DT)	**R-squared:**	0.711
Model:	OLS	**Adj. R-squared:**	0.696
Method:	Least Squares	**F-statistic:**	46.73
Date:	Fri, 18 Jun 2021	**Prob (F-statistic):**	1.59e-06
Time:	23:46:03	**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
ST	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.

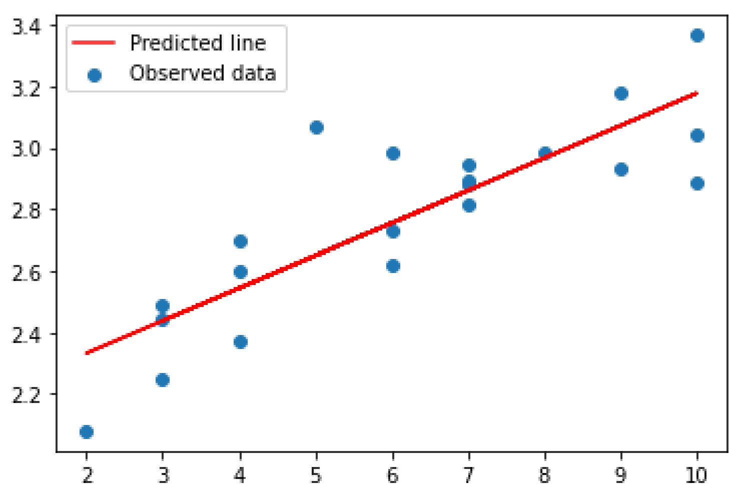


In [52]:

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

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

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



Out[52]: 2.9402503230562007

In [53]:

```python
#### 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()
```

Out[53]:

OLS Regression Results

Dep. Variable:	np.log(DT)	**R-squared:**	0.765
Model:	OLS	**Adj. R-squared:**	0.739
Method:	Least Squares	**F-statistic:**	29.28
Date:	Fri, 18 Jun 2021	**Prob (F-statistic):**	2.20e-06
Time:	23:46:04	**Log-Likelihood:**	9.9597
No. Observations:	21	**AIC:**	-13.92
Df Residuals:	18	**BIC:**	-10.79
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	1.6997	0.228	7.441	0.000	1.220	2.180
ST	0.2659	0.080	3.315	0.004	0.097	0.434
I(ST * ST)	-0.0128	0.006	-2.032	0.057	-0.026	0.000

Omnibus:	2.548	**Durbin-Watson:**	1.369
Prob(Omnibus):	0.280	**Jarque-Bera (JB):**	1.777
Skew:	0.708	**Prob(JB):**	0.411
Kurtosis:	2.846	**Cond. No.**	373.

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

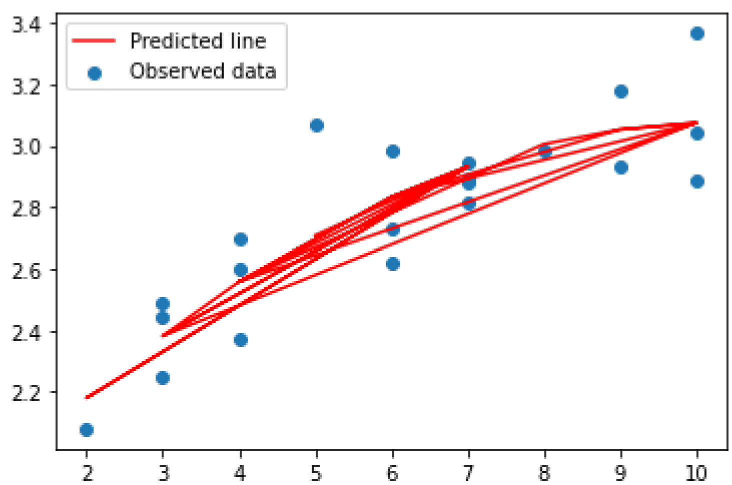
In [54]:

```
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
```

In [55]:

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



In [56]:

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

Out[56]: 2.799041988740927

In [57]:

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

Out[57]:

	MODEL	RMSE
0	SLR	2.791650
1	Log model	2.733171
2	Exp model	2.940250
3	Poly model	2.799042

In [58]:

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

from sklearn.model_selection import train_test_split

train, test = train_test_split(cal, test_size = 0.2)

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

```
D:\anconda\lib\site-packages\scipy\stats\stats.py:1603: UserWarning: kurtosiste
st only valid for n>=20 ... continuing anyway, n=16
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[58]:

OLS Regression Results

Dep. Variable:	DT	**R-squared:**	0.739
Model:	OLS	**Adj. R-squared:**	0.720
Method:	Least Squares	**F-statistic:**	39.59
Date:	Fri, 18 Jun 2021	**Prob (F-statistic):**	1.98e-05
Time:	23:46:05	**Log-Likelihood:**	-38.481
No. Observations:	16	**AIC:**	80.96
Df Residuals:	14	**BIC:**	82.51
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-0.1709	2.721	-0.063	0.951	-6.008	5.666
np.log(ST)	9.6950	1.541	6.292	0.000	6.390	13.000

Omnibus:	3.269	**Durbin-Watson:**	1.707
Prob(Omnibus):	0.195	**Jarque-Bera (JB):**	1.438
Skew:	0.694	**Prob(JB):**	0.487
Kurtosis:	3.479	**Cond. No.**	8.74

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

In [59]:
```
# 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.DT - pred_test_AT
test_sqrs = test_res * test_res
test_mse = np.mean(test_sqrs)
test_rmse = np.sqrt(test_mse)
test_rmse
```

Out[59]: 1880962736.4394288

In [60]:
```
# 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.DT - pred_train_AT
train_sqrs = train_res * train_res
train_mse = np.mean(train_sqrs)
train_rmse = np.sqrt(train_mse)
train_rmse
```

Out[60]: 1570467594.3581145

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 Log DT ~ np.log(ST) beacause the it is the best model.

In [ ]: