Problem Statement: -

```
A certain food-based company conducted a survey with the help of a fitne
ss company to find the relationship between a person’s weight gain and t
he number of calories they consumed in order to come up with diet plans
 for these individuals. Build a Simple Linear Regression model with calo
ries consumed as the target variable. Apply necessary transformations an
d record the RMSE and correlation coefficient values for different model
s.
```

Data Pre-processing.

In [27]:

```python
# 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\\calories_consumed.csv")

df.describe()

df.columns.values[0] = "WT"
df.columns.values[1] = "CC"
df.columns
```

Out[27]: Index(['WT', 'CC'], dtype='object')

In [28]:

```python
df.head()
```

Out[28]:

	WT	CC
0	108	1500
1	200	2300
2	900	3400
3	200	2200
4	300	2500

In [ ]:

In [ ]:

Exploratory data analysis:

In [29]:
```
# 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
```

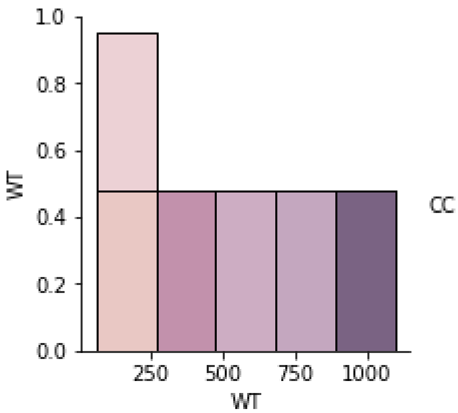
Out[29]:
```
{'column ': Index(['WT', 'CC'], dtype='object'),
 'mean': WT     357.714286
 CC    2340.714286
 dtype: float64,
 'median': WT     200.0
 CC    2250.0
 dtype: float64,
 'mode':      WT    CC
 0  200  1900,
 'standard deviation': WT    333.692495
 CC    752.109488
 dtype: float64,
 'variance': WT    111350.681319
 CC    565668.681319
 dtype: float64,
 'skewness': WT    1.255737
 CC    0.654930
 dtype: float64,
 'kurtosis': WT    0.431272
 CC   -0.290481
 dtype: float64}
```

In [30]:

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

Out[30]: <seaborn.axisgrid.PairGrid at 0x1f399cbcd60>

<Figure size 2160x2160 with 0 Axes>



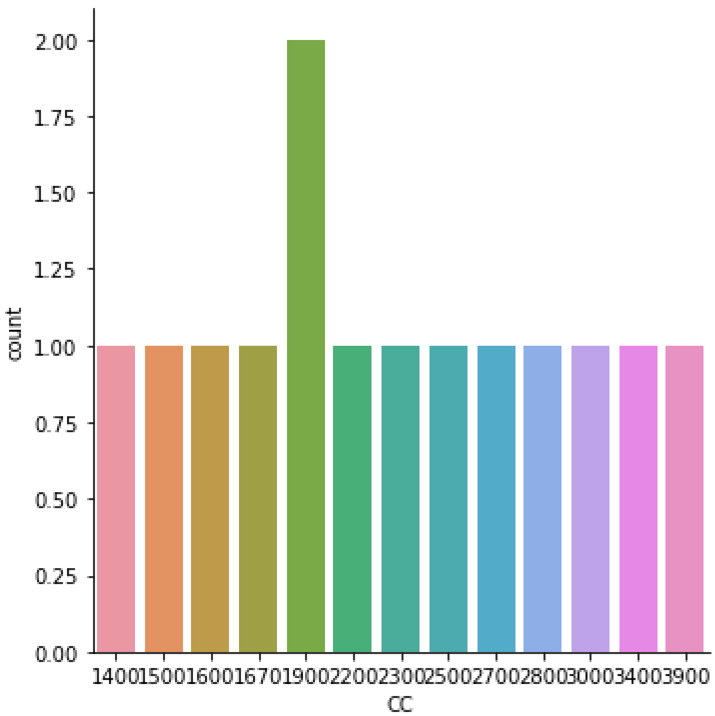
In [31]:

```
#yes or no count
sns.catplot('CC', 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(
```

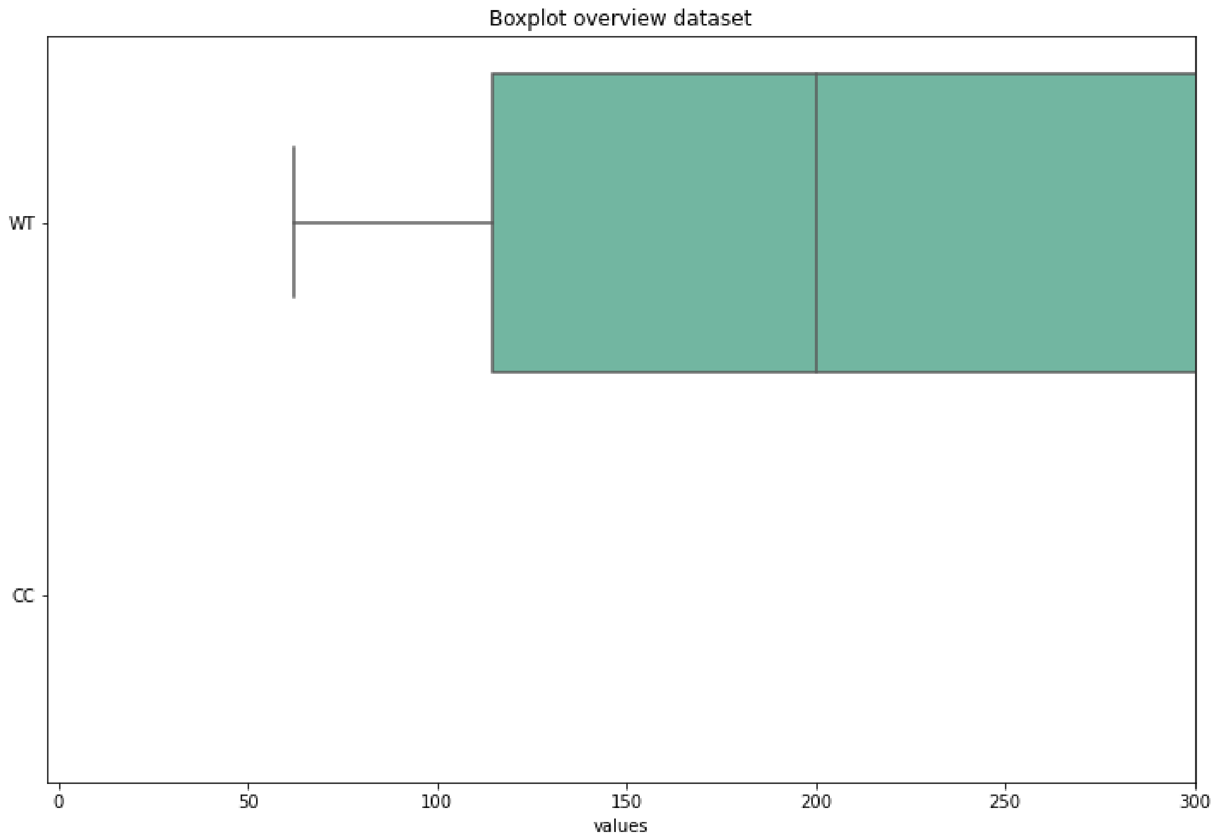
Out[31]:

```
<seaborn.axisgrid.FacetGrid at 0x1f399d7e2b0>
```



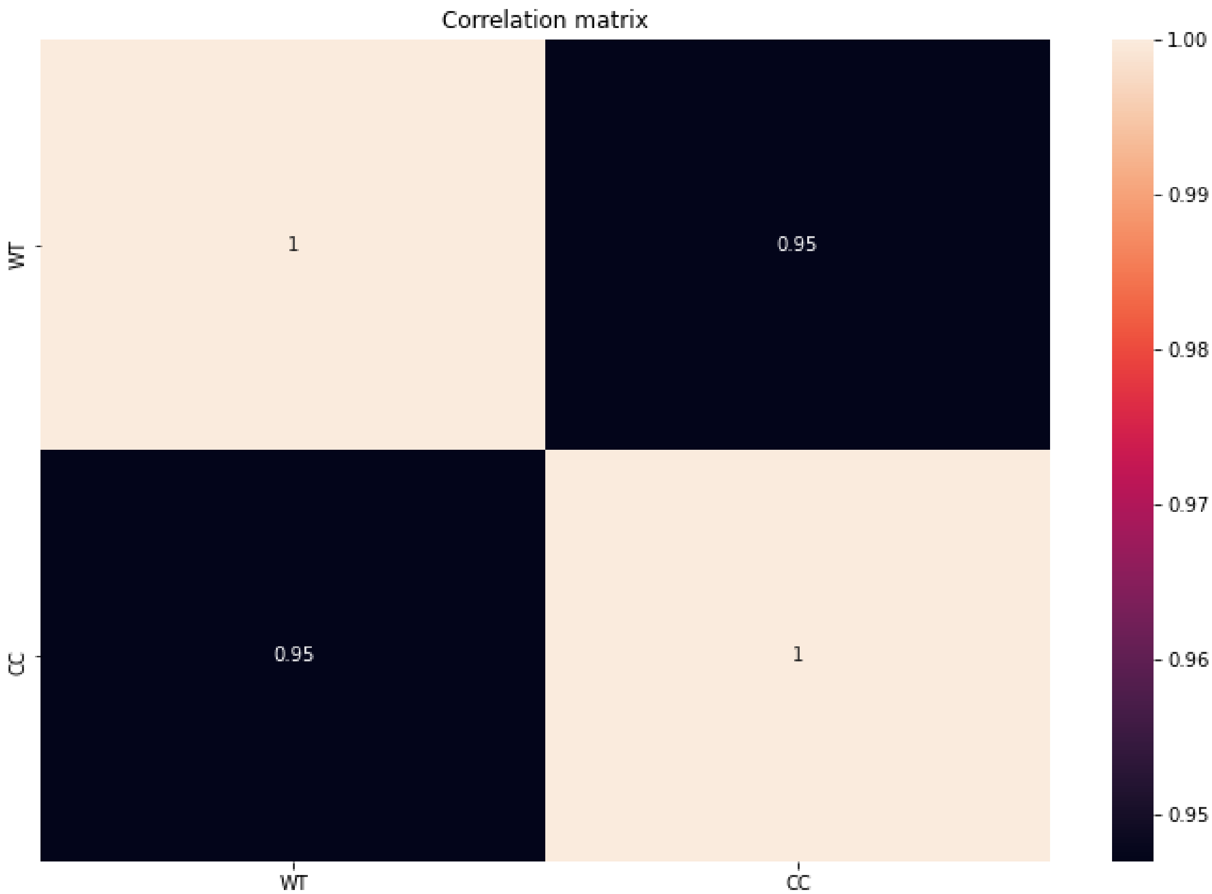
In [32]:

```
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, 300)
plt.show()
```



In [33]:

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



In [34]:

```
# 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[34]:

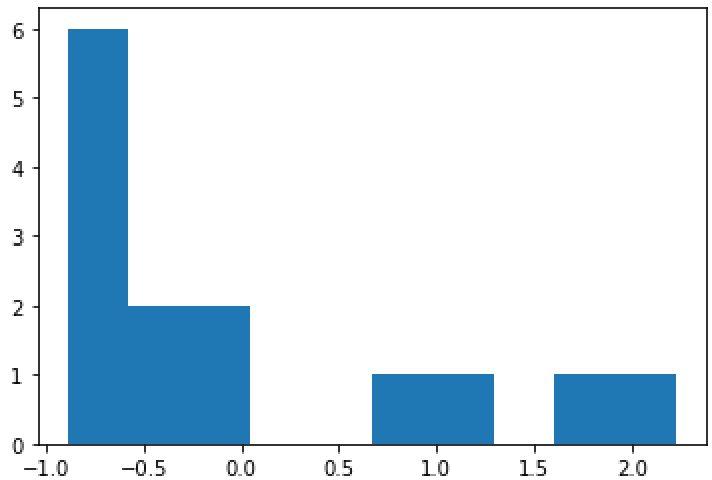
	WT	CC
count	1.400000e+01	1.400000e+01
mean	-6.344132e-17	-1.030921e-16
std	1.000000e+00	1.000000e+00
min	-8.861880e-01	-1.250768e+00
25%	-7.288575e-01	-8.153258e-01
50%	-4.726336e-01	-1.206131e-01
75%	5.387766e-01	5.774235e-01
max	2.224460e+00	2.073216e+00

In [35]:

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

#plt.bar(height = cal.WT, x = np.arange(1, 110, 1))
plt.hist(cal.WT) #histogram
```

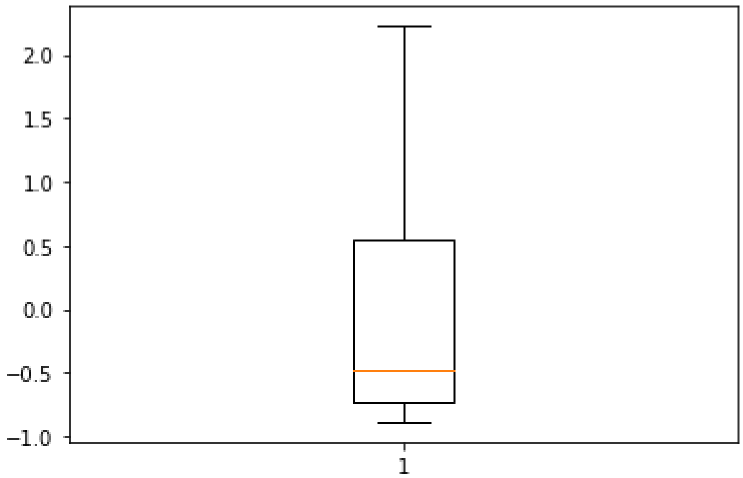
Out[35]:

```
(array([6., 2., 2., 0., 0., 1., 1., 0., 1., 1.]),
 array([-0.886188  , -0.57512317, -0.26405834,  0.04700649,  0.35807133,
         0.66913616,  0.98020099,  1.29126582,  1.60233065,  1.91339549,
         2.22446032]),
 <BarContainer object of 10 artists>)
```



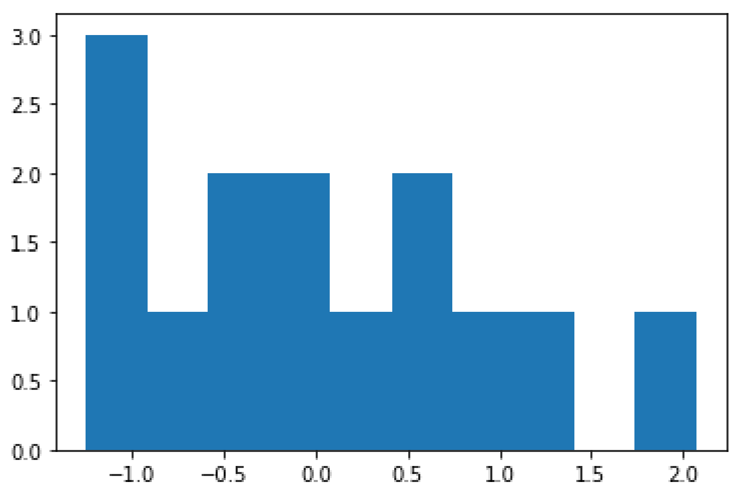
In [36]:
```
plt.boxplot(cal.WT) #boxplot
```

Out[36]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1f39a1e8580>,
  <matplotlib.lines.Line2D at 0x1f39a1e88e0>],
 'caps': [<matplotlib.lines.Line2D at 0x1f39a1e8c40>,
  <matplotlib.lines.Line2D at 0x1f39a1e8fa0>],
 'boxes': [<matplotlib.lines.Line2D at 0x1f39a1e8220>],
 'medians': [<matplotlib.lines.Line2D at 0x1f39a1f4340>],
 'fliers': [<matplotlib.lines.Line2D at 0x1f39a1f46a0>],
 'means': []}
```



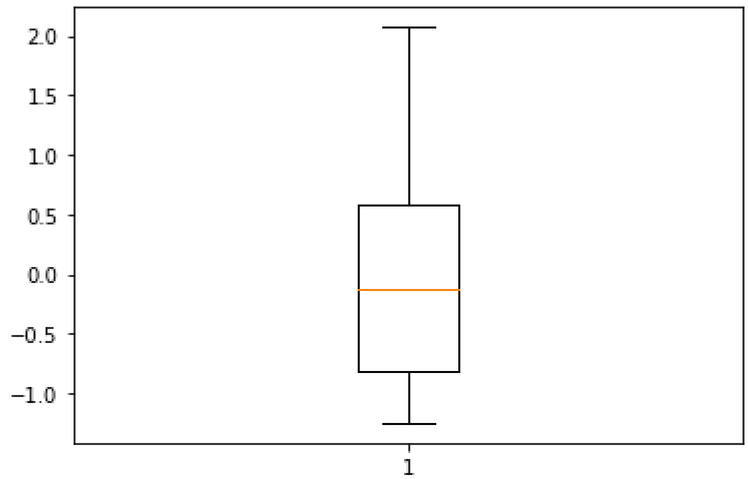
In [37]:
```
#plt.bar(height = cal.CC, x = np.arange(1, 110, 1))
plt.hist(cal.CC) #histogram
```

Out[37]:
```
(array([3., 1., 2., 2., 1., 2., 1., 1., 0., 1.]),
 array([-1.25076774, -0.91836933, -0.58597092, -0.2535725 ,  0.07882591,
         0.41122432,  0.74362274,  1.07602115,  1.40841956,  1.74081797,
         2.07321639]),
 <BarContainer object of 10 artists>)
```



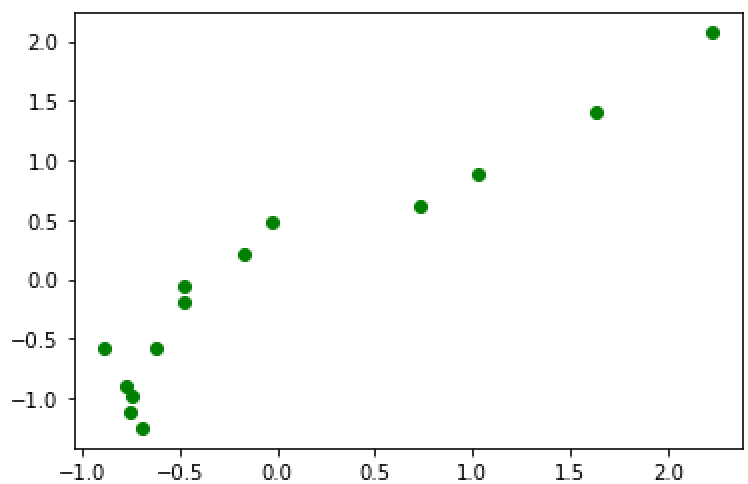
In [38]:
```
plt.boxplot(cal.CC) #boxplot
```

Out[38]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x1f39a2b3d90>,
  <matplotlib.lines.Line2D at 0x1f39a2c3130>],
 'caps': [<matplotlib.lines.Line2D at 0x1f39a2c3490>,
  <matplotlib.lines.Line2D at 0x1f39a2c37f0>],
 'boxes': [<matplotlib.lines.Line2D at 0x1f39a2b3a30>],
 'medians': [<matplotlib.lines.Line2D at 0x1f39a2c3b50>],
 'fliers': [<matplotlib.lines.Line2D at 0x1f39a2c3eb0>],
 'means': []}
```



In [39]:
```
# Scatter plot
plt.scatter(x = cal.WT, y = cal.CC, color = 'green')
```

Out[39]:
```
<matplotlib.collections.PathCollection at 0x1f39b2e8520>
```



In [40]:
```
# correlation
np.corrcoef(cal.WT, cal.CC)
```

Out[40]:
```
array([[1.        , 0.94699101],
       [0.94699101, 1.        ]])
```

In [41]:

```
# 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.WT, cal.CC)[0, 1]
cov_output
```

Out[41]: 0.9469910088554455

MODEL BUILDING

In [42]:

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

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

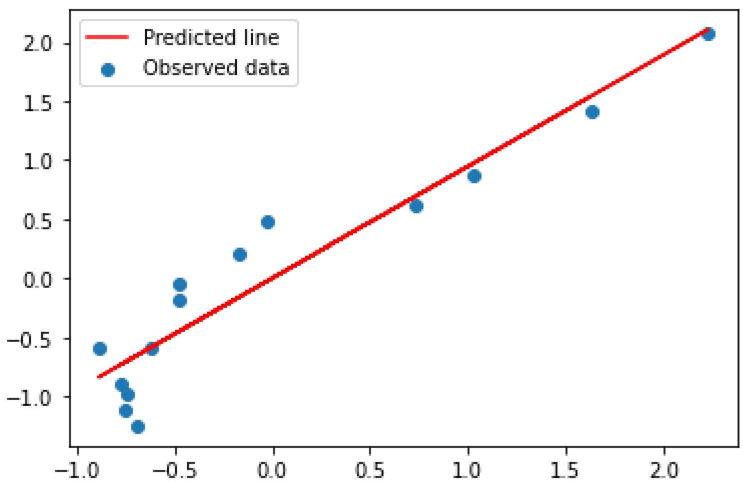
Out[42]:

OLS Regression Results

Dep. Variable:	CC	**R-squared:**	0.897
Model:	OLS	**Adj. R-squared:**	0.888
Method:	Least Squares	**F-statistic:**	104.3
Date:	Fri, 18 Jun 2021	**Prob (F-statistic):**	2.86e-07
Time:	23:13:29	**Log-Likelihood:**	-3.4493
No. Observations:	14	**AIC:**	10.90
Df Residuals:	12	**BIC:**	12.18
Df Model:	1		

In [43]:

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

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

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



Out[43]: 0.30957394442203984

In [44]:

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

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

model2 = smf.ols('CC ~ np.log(WT)', data = cal).fit()
model2.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\statsmodels\stats\stattools.py:74: ValueWarning: o
mni_normtest is not valid with less than 8 observations; 4 samples were given.
  warn("omni_normtest is not valid with less than 8 observations; %i "
```

Out[44]:

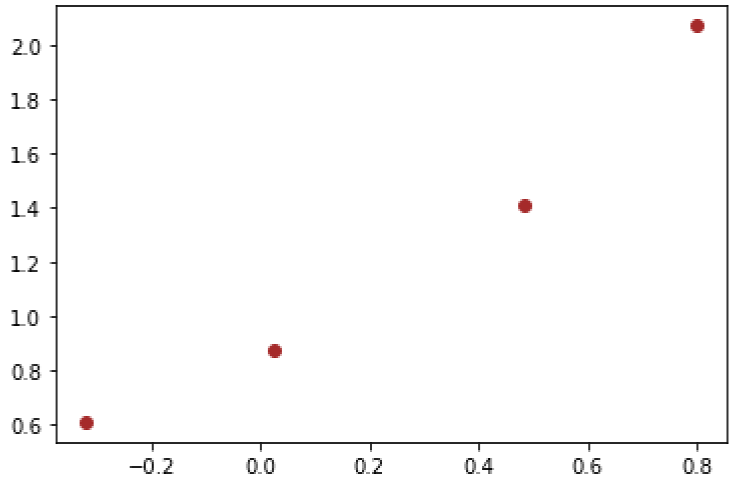
OLS Regression Results

Dep. Variable:	CC	**R-squared:**	0.960
Model:	OLS	**Adj. R-squared:**	0.940
Method:	Least Squares	**F-statistic:**	47.72
Date:	Fri, 18 Jun 2021	**Prob (F-statistic):**	0.0203
Time:	23:13:29	**Log-Likelihood:**	3.0757
No. Observations:	4	**AIC:**	-2.151
Df Residuals:	2	**BIC:**	-3.379
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.9253	0.092	10.100	0.010	0.531	1.320
np.log(WT)	1.2798	0.185	6.908	0.020	0.483	2.077

Omnibus:	nan	**Durbin-Watson:**	1.943
Prob(Omnibus):	nan	**Jarque-Bera (JB):**	0.569
Skew:	-0.069	**Prob(JB):**	0.753
Kurtosis:	1.158	**Cond. No.**	2.51

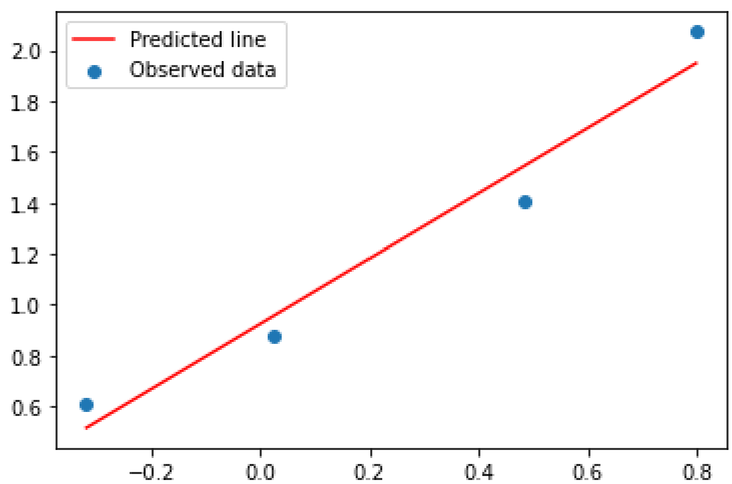
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.


2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
−0.2
0.0
0.2
0.4
0.6
0.8

In [45]:
```
pred2 = model2.predict(pd.DataFrame(cal.WT))

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

# Error calculation
res2 = cal.CC - 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.WT, y = np.log(cal.CC), color = 'orange')
np.corrcoef(cal.WT, np.log(cal.CC)) #correlation

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

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

# Regression Line
plt.scatter(cal.WT, np.log(cal.CC))
plt.plot(cal.WT, 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(CC) ~ WT + I(WT*WT)', data = cal).fit()
model4.summary()

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

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

```
X = cal.iloc[:, 0:1].values
X_poly = poly_reg.fit_transform(X)
# y = wcat.iloc[:, 1].values


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

# Error calculation
res4 = cal.CC - 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[45]:

```
'\n#### Exponential transformation\n# x = waist; y = log(at)\n\nplt.scatter(x =
cal.WT, y = np.log(cal.CC), color = \'orange\')\nnp.corrcoef(cal.WT, np.log(ca
l.CC)) #correlation\n\nmodel3 = smf.ols(\'np.log(CC) ~ WT\', data = cal).fit()
\nmodel3.summary()\n\npred3 = model3.predict(pd.DataFrame(cal.WT))\npred3_at =
np.exp(pred3)\npred3_at\n\n# Regression Line\nplt.scatter(cal.WT, np.log(cal.C
C))\nplt.plot(cal.WT, pred3, "r")\nplt.legend([\'Predicted line\', \'Observed d
ata\'])\nplt.show()\n\n# Error calculation\nres3 = cal.CC - pred3_at\nres_sqr3
= res3 * res3\nmse3 = np.mean(res_sqr3)\nrmse3 = np.sqrt(mse3)\nrmse3\n\n\n####
Polynomial transformation\n# x = waist; x^2 = waist*waist; y = log(at)\n\nmodel
4 = smf.ols(\'np.log(CC) ~ WT + I(WT*WT)\', data = cal).fit()\nmodel4.summary()
\n\npred4 = model4.predict(pd.DataFrame(cal))\npred4_at = np.exp(pred4)\npred4_
at\n\n# Regression line\nfrom sklearn.preprocessing import PolynomialFeatures\n
poly_reg = PolynomialFeatures(degree = 2)\nX = cal.iloc[:, 0:1].values\nX_poly
= poly_reg.fit_transform(X)\n# y = wcat.iloc[:, 1].values\n\n\nplt.scatter(cal.
WT, np.log(cal.CC))\nplt.plot(X, pred4, color = \'red\')\nplt.legend([\'Predict
ed line\', \'Observed data\'])\nplt.show()\n\n# Error calculation\nres4 = cal.C
C - pred4_at\nres_sqr4 = res4 * res4\nmse4 = np.mean(res_sqr4)\nrmse4 = np.sqrt
(mse4)\nrmse4\n'
```

In [46]:

```
# 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[46]:

	MODEL	RMSE
0	SLR	0.309574
1	Log	0.112155

In [47]:

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

from sklearn.model_selection import train_test_split

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

finalmodel = smf.ols('CC ~ WT', 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=11
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[47]:

OLS Regression Results

Dep. Variable:	CC	**R-squared:**	0.911
Model:	OLS	**Adj. R-squared:**	0.901
Method:	Least Squares	**F-statistic:**	92.27
Date:	Fri, 18 Jun 2021	**Prob (F-statistic):**	5.00e-06
Time:	23:13:30	**Log-Likelihood:**	-2.7884
No. Observations:	11	**AIC:**	9.577
Df Residuals:	9	**BIC:**	10.37
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-0.0387	0.104	-0.372	0.718	-0.274	0.196
WT	0.9525	0.099	9.606	0.000	0.728	1.177

Omnibus:	0.636	**Durbin-Watson:**	1.322
Prob(Omnibus):	0.728	**Jarque-Bera (JB):**	0.556
Skew:	-0.092	**Prob(JB):**	0.757
Kurtosis:	1.914	**Cond. No.**	1.05

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

In [48]:

```
# 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.CC - pred_test_AT
test_sqrs = test_res * test_res
test_mse = np.mean(test_sqrs)
test_rmse = np.sqrt(test_mse)
print(test_rmse)
```

```
1.0297464508770553
```

In [49]:

```
# 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.CC - pred_train_AT
train_sqrs = train_res * train_res
train_mse = np.mean(train_sqrs)
train_rmse = np.sqrt(train_mse)
print(train_rmse)
```

```
2.329778495018364
```

Summary

Model having highest R-Squared value is better i.e. (model=0.897 is not better than model1=0.960). 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 CC ~ WT beacause the training rmse was show good result in SLR rather than Log.

In [ ]: