Problem Statement: -

A certain organization wants an early estimate of their employee churn out rate. So the HR department gathered the data regarding the employee's salary hike and the churn out rate in a financial year. The analytics team will have to perform an analysis and predict an estimate of employee churn based on the salary hike. Build a Simple Linear Regression model with churn out rate as the target variable. Apply necessary transformations and record the RMSE and correlation coefficient values for different models.

Data Pre-processing.

In [29]:

```
# 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\\emp_data.csv")

df.describe()

df.columns.values[0] = "SH"
df.columns.values[1] = "CC"
df.columns


df.head()
```

Out[29]:

	SH	CC
0	1580	92
1	1600	85
2	1610	80
3	1640	75
4	1660	72

Exploratory data analysis:

In [3]:

```
# 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[3]:

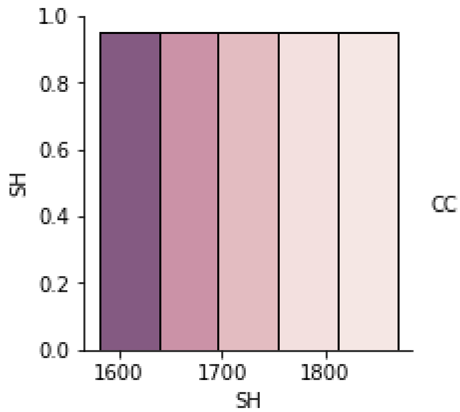
```
{'column ': Index(['SH', 'CC'], dtype='object'),
 'mean': SH    1688.6
 CC      72.9
 dtype: float64,
 'median': SH    1675.0
 CC      71.0
 dtype: float64,
 'mode':      SH  CC
 0  1580  60
 1  1600  62
 2  1610  65
 3  1640  68
 4  1660  70
 5  1690  72
 6  1706  75
 7  1730  80
 8  1800  85
 9  1870  92,
 'standard deviation': SH    92.096809
 CC    10.257247
 dtype: float64,
 'variance': SH    8481.822222
 CC     105.211111
 dtype: float64,
 'skewness': SH    0.858375
 CC    0.647237
 dtype: float64,
 'kurtosis': SH    0.165793
 CC   -0.328199
 dtype: float64}
```

In [4]:

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

Out[4]: <seaborn.axisgrid.PairGrid at 0x25a0aa35820>

<Figure size 2160x2160 with 0 Axes>

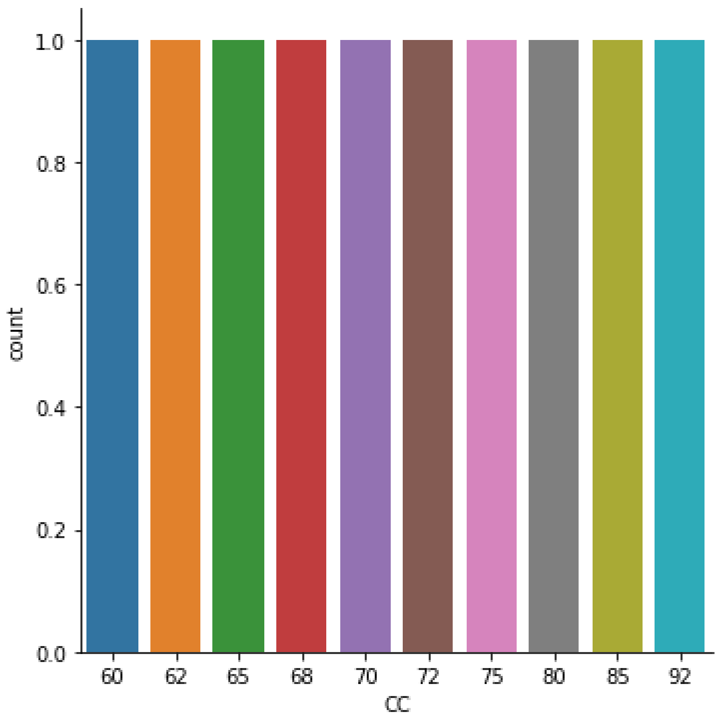


In [5]:

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

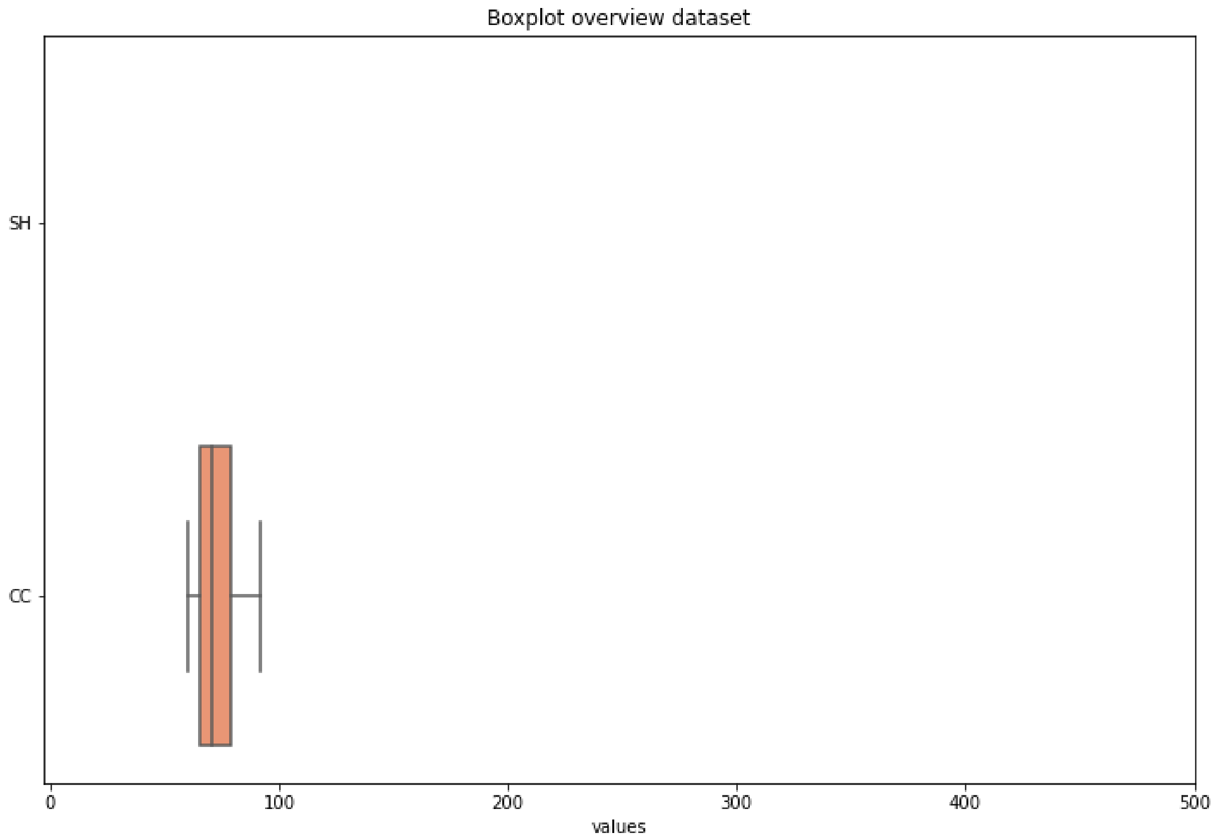
```
D:\anconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(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

Out[5]: <seaborn.axisgrid.FacetGrid at 0x25a0b250850>



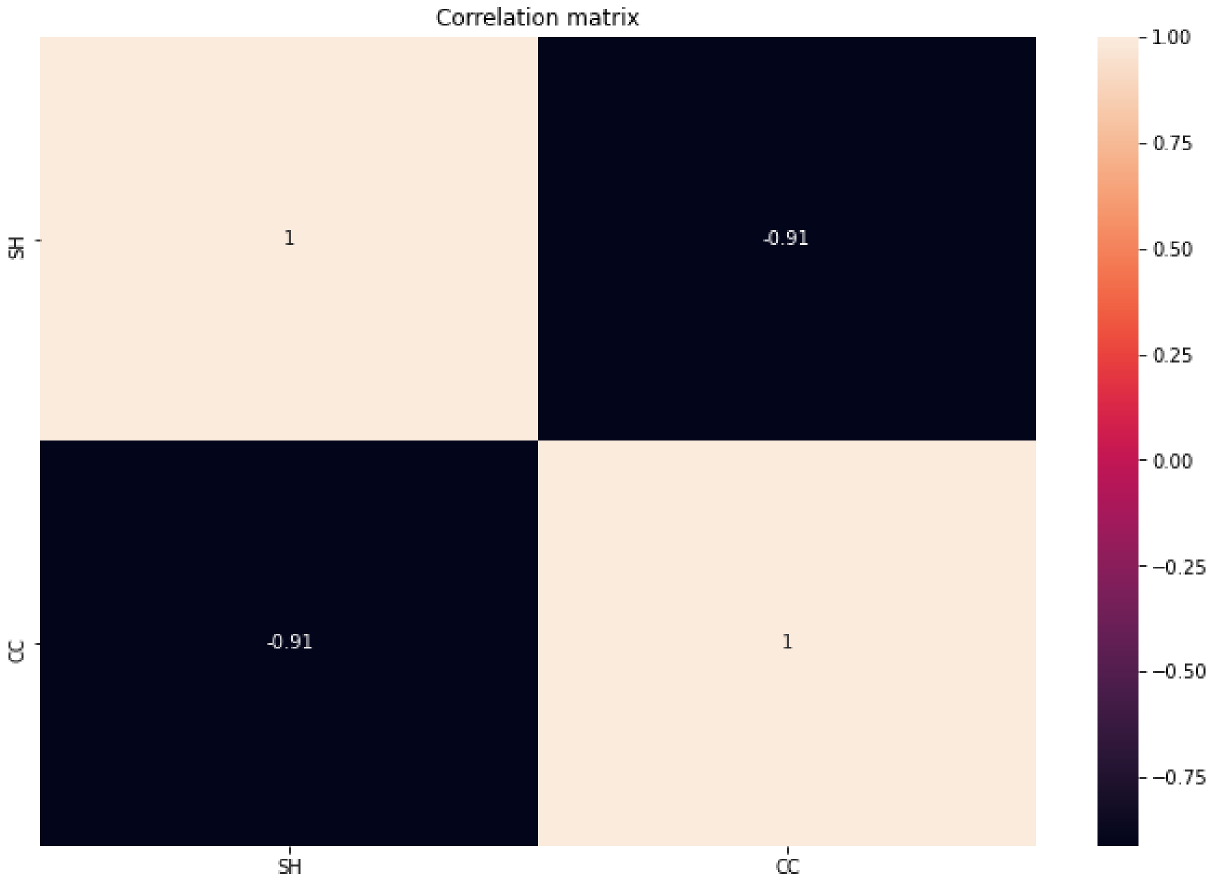
In [6]:

```
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, 500)
plt.show()
```



In [7]:

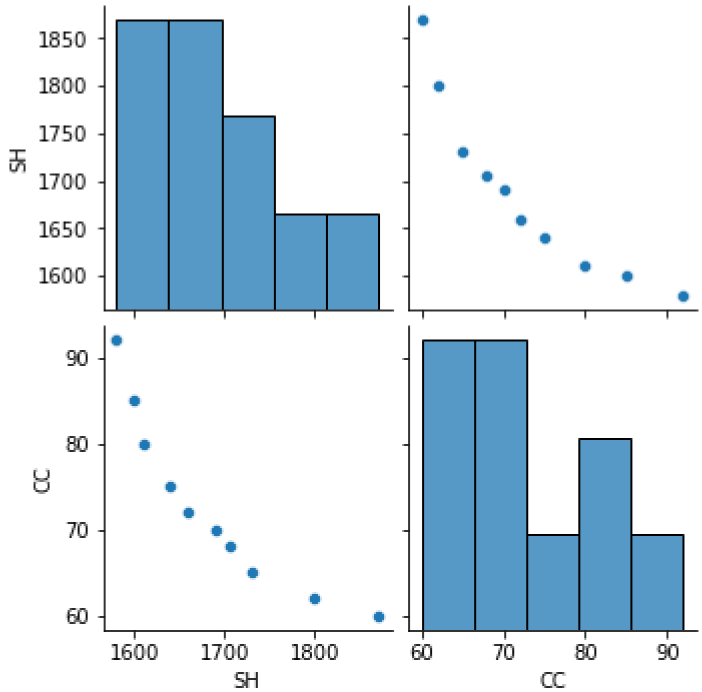
```
plt.figure(figsize = (12, 8))
sns.heatmap(df.corr(), annot = True)
plt.title('Correlation matrix')
plt.show()
```



In [8]:

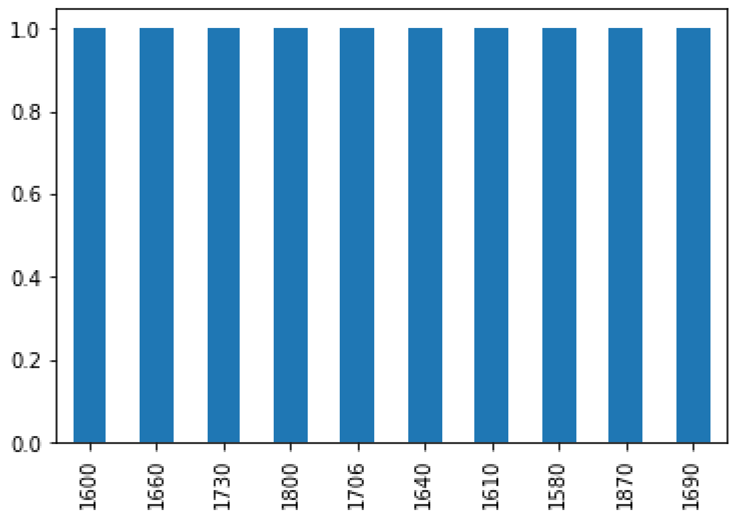
```
sns.pairplot(df)
```

Out[8]: <seaborn.axisgrid.PairGrid at 0x25a0c3dfe20>



In [9]:

```
#Bar plot
df['SH'].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
```

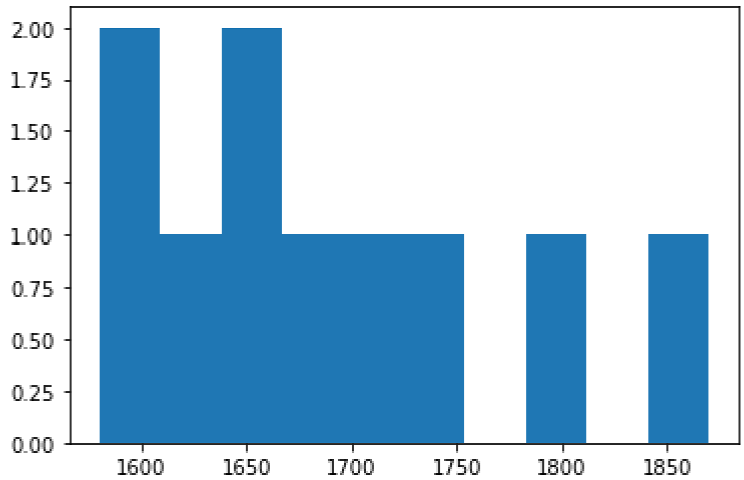


In [10]:

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

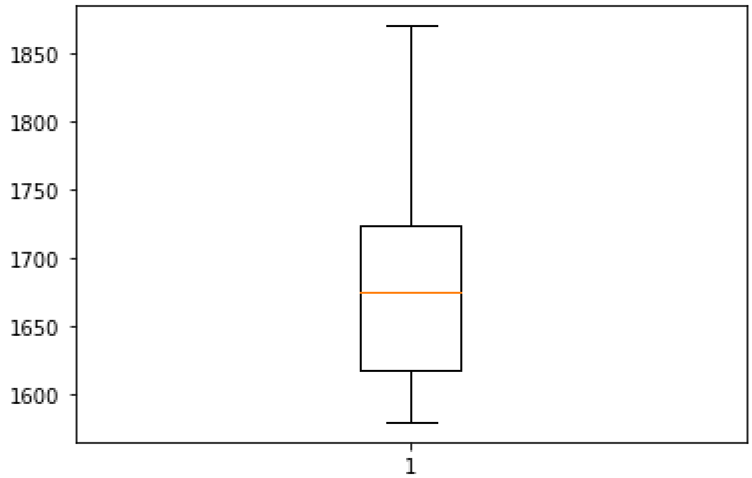
In [11]:
```
plt.hist(cal.SH) #histogram
```

Out[11]:
```
(array([2., 1., 2., 1., 1., 1., 0., 1., 0., 1.]),
 array([1580., 1609., 1638., 1667., 1696., 1725., 1754., 1783., 1812.,
        1841., 1870.]),
 <BarContainer object of 10 artists>)
```



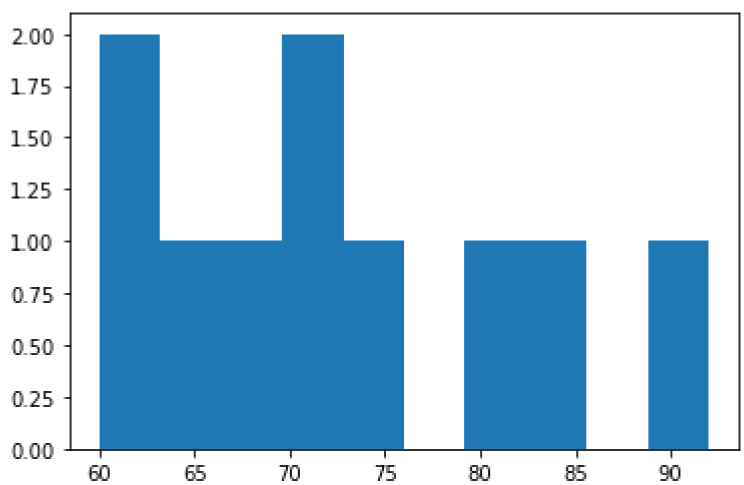
In [12]:
```
plt.boxplot(cal.SH) #boxplot
```

Out[12]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x25a0ca37e20>,
  <matplotlib.lines.Line2D at 0x25a0ca37d60>],
 'caps': [<matplotlib.lines.Line2D at 0x25a0ca37220>,
  <matplotlib.lines.Line2D at 0x25a0c9d4d90>],
 'boxes': [<matplotlib.lines.Line2D at 0x25a0ca50100>],
 'medians': [<matplotlib.lines.Line2D at 0x25a0c9d4e20>],
 'fliers': [<matplotlib.lines.Line2D at 0x25a0c9d4700>],
 'means': []}
```



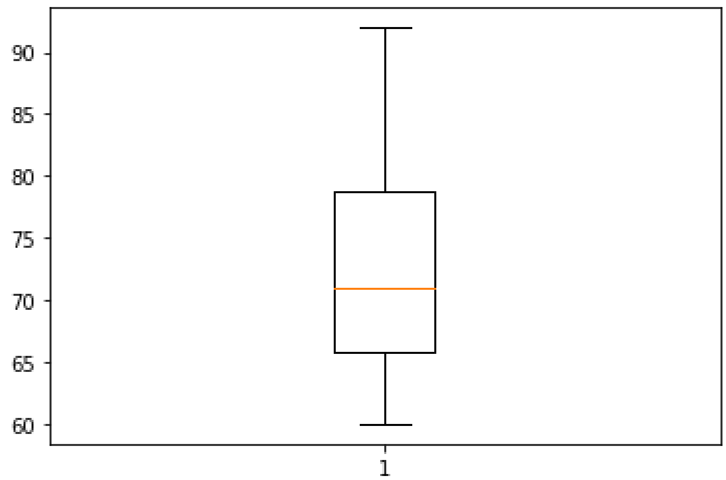
In [13]:
```
plt.hist(cal.CC) #histogram
```

Out[13]:
```
(array([2., 1., 1., 2., 1., 0., 1., 1., 0., 1.]),
 array([60. , 63.2, 66.4, 69.6, 72.8, 76. , 79.2, 82.4, 85.6, 88.8, 92. ]),
 <BarContainer object of 10 artists>)
```



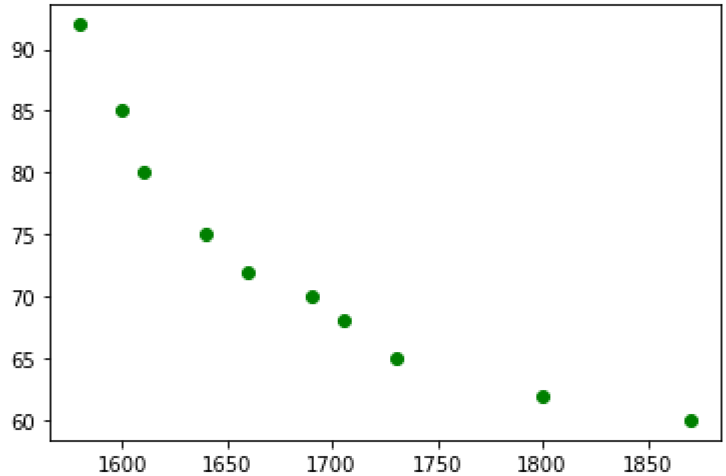
In [14]:
```
plt.boxplot(cal.CC) #boxplot
```

Out[14]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x25a0c502fd0>,
  <matplotlib.lines.Line2D at 0x25a0c510370>],
 'caps': [<matplotlib.lines.Line2D at 0x25a0c5106d0>,
  <matplotlib.lines.Line2D at 0x25a0c510a30>],
 'boxes': [<matplotlib.lines.Line2D at 0x25a0c502c70>],
 'medians': [<matplotlib.lines.Line2D at 0x25a0c510d90>],
 'fliers': [<matplotlib.lines.Line2D at 0x25a0c51d130>],
 'means': []}
```



In [15]:
```
# Scatter plot
plt.scatter(x = cal.SH, y = cal.CC, color = 'green')
```

Out[15]: <matplotlib.collections.PathCollection at 0x25a0c564700>



In [16]:
```
# correlation
np.corrcoef(cal.SH, cal.CC)
```

Out[16]:
```
array([[ 1.        , -0.91172162],
       [-0.91172162,  1.        ]])
```

In [17]:
```
# 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.SH, cal.CC)[0, 1]
cov_output
```

Out[17]: -861.2666666666667

DATA MODELING

In [18]:

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

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

```
D:\anconda\lib\site-packages\scipy\stats\stats.py:1603: UserWarning: kurtosiste
st only valid for n>=20 ... continuing anyway, n=10
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[18]:

OLS Regression Results

Dep. Variable:	CC	**R-squared:**	0.831
Model:	OLS	**Adj. R-squared:**	0.810
Method:	Least Squares	**F-statistic:**	39.40
Date:	Sat, 19 Jun 2021	**Prob (F-statistic):**	0.000239
Time:	00:25:46	**Log-Likelihood:**	-28.046
No. Observations:	10	**AIC:**	60.09
Df Residuals:	8	**BIC:**	60.70
Df Model:	1		
Covariance Type:	nonrobust		

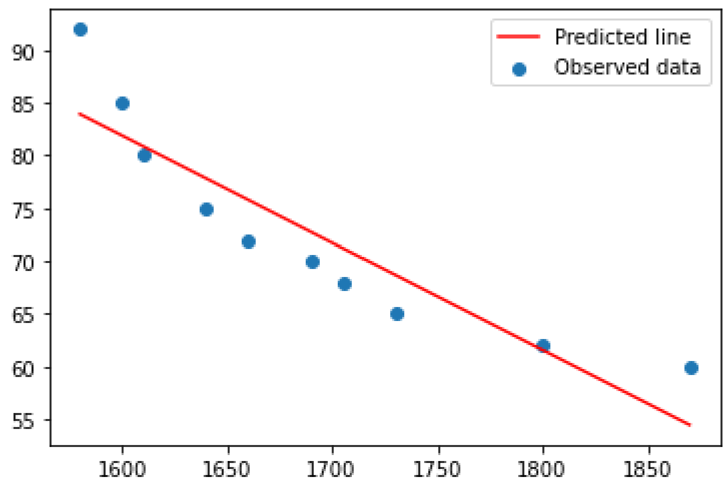
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	244.3649	27.352	8.934	0.000	181.291	307.439
SH	-0.1015	0.016	-6.277	0.000	-0.139	-0.064

Omnibus:	2.201	**Durbin-Watson:**	0.562
Prob(Omnibus):	0.333	**Jarque-Bera (JB):**	1.408
Skew:	0.851	**Prob(JB):**	0.495
Kurtosis:	2.304	**Cond. No.**	3.27e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.27e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [19]:

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

# Regression Line
plt.scatter(cal.SH, cal.CC)
plt.plot(cal.SH, 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[19]: 3.9975284623377902

In [20]:

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

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

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

```
D:\anconda\lib\site-packages\scipy\stats\stats.py:1603: UserWarning: kurtosiste
st only valid for n>=20 ... continuing anyway, n=10
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[20]:

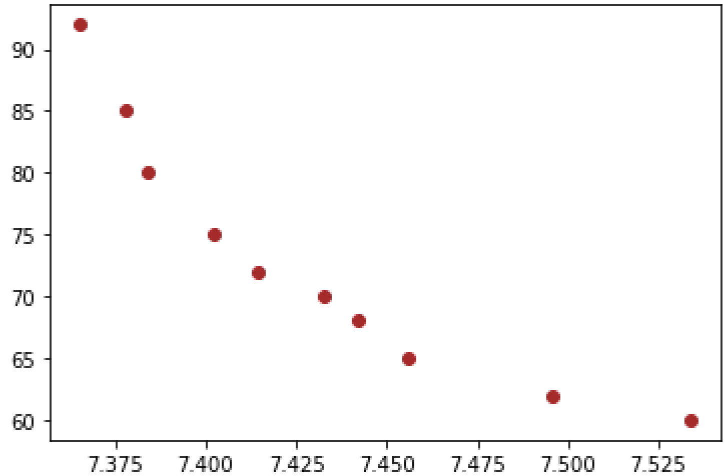
OLS Regression Results

Dep. Variable:	CC	**R-squared:**	0.849
Model:	OLS	**Adj. R-squared:**	0.830
Method:	Least Squares	**F-statistic:**	44.85
Date:	Sat, 19 Jun 2021	**Prob (F-statistic):**	0.000153
Time:	00:25:46	**Log-Likelihood:**	-27.502
No. Observations:	10	**AIC:**	59.00
Df Residuals:	8	**BIC:**	59.61
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	1381.4562	195.402	7.070	0.000	930.858	1832.054
np.log(SH)	-176.1097	26.297	-6.697	0.000	-236.751	-115.468

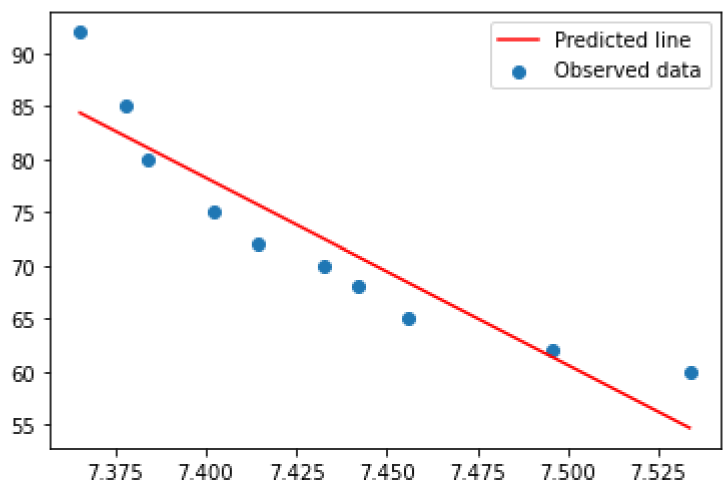
Omnibus:	2.213	**Durbin-Watson:**	0.571
Prob(Omnibus):	0.331	**Jarque-Bera (JB):**	1.418
Skew:	0.853	**Prob(JB):**	0.492
Kurtosis:	2.298	**Cond. No.**	1.10e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.1e+03. This might indicate that there are
strong multicollinearity or other numerical problems.



In [21]:

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

# Regression Line
plt.scatter(np.log(cal.SH), cal.CC)
plt.plot(np.log(cal.SH), 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
```



Out[21]: 3.786003613022774

In [22]:

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

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

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

```
D:\anconda\lib\site-packages\scipy\stats\stats.py:1603: UserWarning: kurtosiste
st only valid for n>=20 ... continuing anyway, n=10
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[22]:

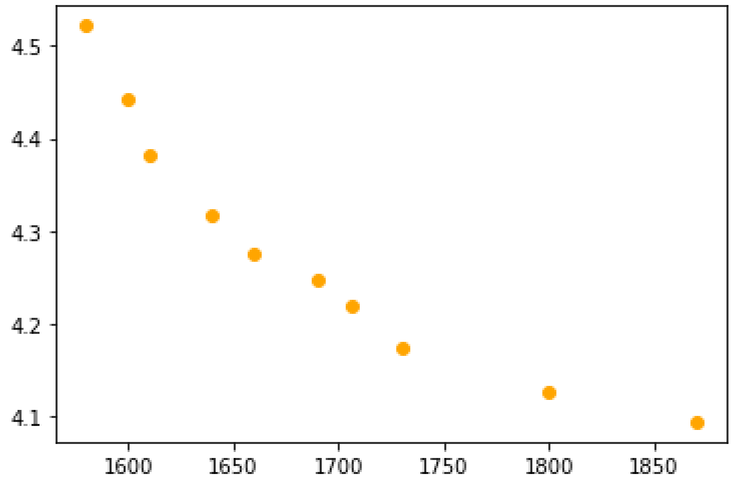
OLS Regression Results

Dep. Variable:	np.log(CC)	**R-squared:**	0.874
Model:	OLS	**Adj. R-squared:**	0.858
Method:	Least Squares	**F-statistic:**	55.26
Date:	Sat, 19 Jun 2021	**Prob (F-statistic):**	7.38e-05
Time:	00:25:46	**Log-Likelihood:**	16.511
No. Observations:	10	**AIC:**	-29.02
Df Residuals:	8	**BIC:**	-28.42
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	6.6383	0.318	20.902	0.000	5.906	7.371
SH	-0.0014	0.000	-7.434	0.000	-0.002	-0.001

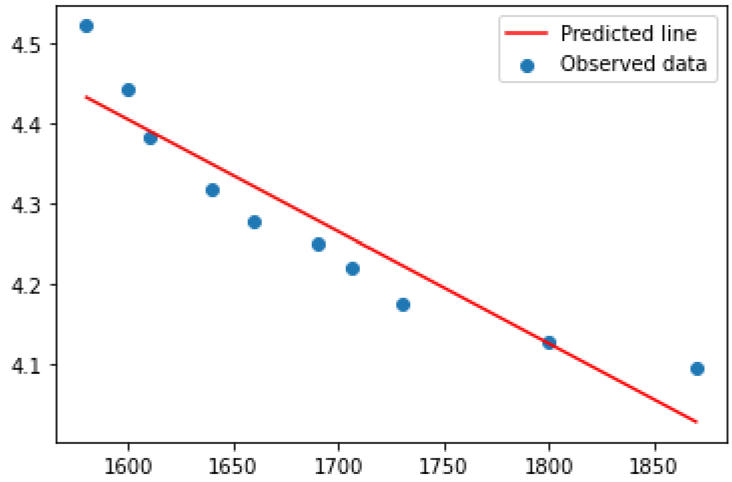
Omnibus:	1.935	**Durbin-Watson:**	0.585
Prob(Omnibus):	0.380	**Jarque-Bera (JB):**	1.314
Skew:	0.780	**Prob(JB):**	0.519
Kurtosis:	2.152	**Cond. No.**	3.27e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.27e+04. This might indicate that there are
strong multicollinearity or other numerical problems.


4.5
4.4
4.3
4.2
4.1
1600
1650
1700
1750
1800
1850

In [23]:

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

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

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


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

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



Out[23]:

```
"\n#### Polynomial transformation\n# x = waist; x^2 = waist*waist; y = log(at)
\n\nmodel4 = smf.ols('np.log(DT) ~ ST + I(ST*ST)', data = cal).fit()\nmodel4.su
mmary()\n\npred4 = model4.predict(pd.DataFrame(cal.ST))\npred4_at = np.exp(pred
4)\npred4_at\n\n# Regression line\nfrom sklearn.preprocessing import Polynomial
Features\npoly_reg = PolynomialFeatures(degree = 2)\nX = cal.iloc[:, 1:].values
\nX_poly = poly_reg.fit_transform(X)\n# y = wcat.iloc[:, 1].values\n\n\nplt.sca
tter(cal.ST, np.log(cal.DT))\nplt.plot(X, pred4, color = 'red')\nplt.legend(['P
redicted line', 'Observed data'])\nplt.show()\n\n# Error calculation\nres4 = ca
l.DT - pred4_at\nres_sqr4 = res4 * res4\nmse4 = np.mean(res_sqr4)\nrmse4 = np.s
qrt(mse4)\nrmse4\n"
```

In [24]:

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

Out[24]:

	MODEL	RMSE
0	SLR	3.997528
1	Log model	3.786004
2	Exp model	3.541549

In [25]:

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

from sklearn.model_selection import train_test_split

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

finalmodel = smf.ols('np.log(CC) ~ SH', 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=8
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[25]:

OLS Regression Results

Dep. Variable:	np.log(CC)	**R-squared:**	0.873
Model:	OLS	**Adj. R-squared:**	0.851
Method:	Least Squares	**F-statistic:**	41.12
Date:	Sat, 19 Jun 2021	**Prob (F-statistic):**	0.000679
Time:	00:25:51	**Log-Likelihood:**	12.638
No. Observations:	8	**AIC:**	-21.28
Df Residuals:	6	**BIC:**	-21.12
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	6.6524	0.371	17.919	0.000	5.744	7.561
SH	-0.0014	0.000	-6.412	0.001	-0.002	-0.001

Omnibus:	1.710	**Durbin-Watson:**	2.329
Prob(Omnibus):	0.425	**Jarque-Bera (JB):**	0.911
Skew:	0.494	**Prob(JB):**	0.634
Kurtosis:	1.674	**Cond. No.**	3.10e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.1e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [26]:
```
# 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)
test_rmse
```

Out[26]: 2.224957297246967

In [27]:
```

# 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)
train_rmse
```

Out[27]: 3.7983654142497283

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 Expo model np.log(CC) ~ SH beacause the it is the best model.

R-squared: 0.873

test_rmse 2.224957297246967

train_rmse 3.7983654142497283

In [ ]: