Multiple linear regression

In [1]:

```python
# import pandas library
import pandas as pd
```

In [2]:

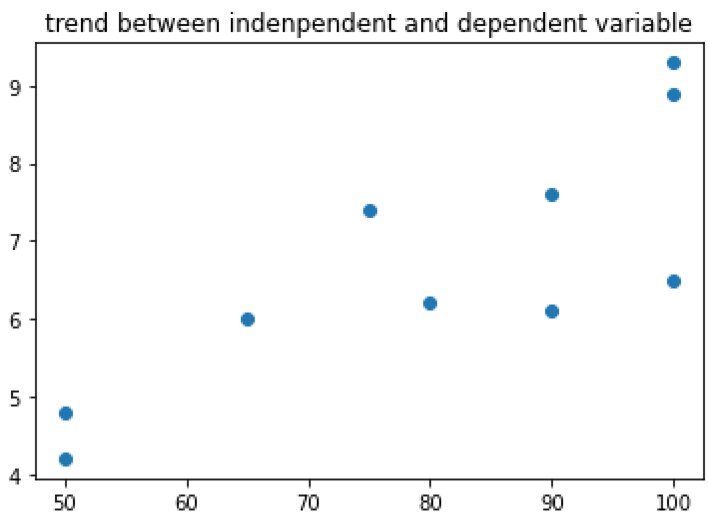
```python
#import data
data = pd.read_excel('C:/Users/subbu/OneDrive/Desktop/\
machine Learning/experiments/TRUCKING.xlsx')
data
```

Out[2]:

	Driving Assignmnet	miles_travelled	n_of_deliveries	travel_time
0	1	100	4	9.3
1	2	50	3	4.8
2	3	100	4	8.9
3	4	100	2	6.5
4	5	50	2	4.2
5	6	80	2	6.2
6	7	75	3	7.4
7	8	65	4	6.0
8	9	90	3	7.6
9	10	90	2	6.1

In [3]:

```python
# plot a graph between any input attribute and output label
import matplotlib.pyplot as plt
plt.scatter(data['miles_travelled'], data['travel_time'])
plt.ylabel = ('travel time')
plt.title('trend between indenpendent and dependent variable')
plt.show()
```



In [4]:

```
#Extract the input features in X and the output label in Y
X = data[['miles_travelled', 'n_of_deliveries']]
Y = data['travel_time']
print(X)
print(Y)
```

```
   miles_travelled  n_of_deliveries
0              100                4
1               50                3
2              100                4
3              100                2
4               50                2
5               80                2
6               75                3
7               65                4
8               90                3
9               90                2
0    9.3
1    4.8
2    8.9
3    6.5
4    4.2
5    6.2
6    7.4
7    6.0
8    7.6
9    6.1
Name: travel_time, dtype: float64
```

What is sklearn Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

In [5]:

```
# import LinearRegression class from sklearn.linear_model
from sklearn.linear_model import LinearRegression
# import train_test_split function from sklearn.model_selection
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = \
train_test_split( X, Y, test_size=0.2, random_state=45)
#random_state controls the shuffling applied to the data
#before applying the split.
```

In [6]:
```python
X_train, X_test, Y_train, Y_test
```

Out[6]:
```
(   miles_travelled  n_of_deliveries
 1               50                3
 8               90                3
 9               90                2
 6               75                3
 5               80                2
 7               65                4
 0              100                4
 3              100                2,
    miles_travelled  n_of_deliveries
 2              100                4
 4               50                2,
 1    4.8
 8    7.6
 9    6.1
 6    7.4
 5    6.2
 7    6.0
 0    9.3
 3    6.5
 Name: travel_time, dtype: float64,
 2    8.9
 4    4.2
 Name: travel_time, dtype: float64)
```

In [7]:
```python
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[7]:
```
((8, 2), (2, 2), (8,), (2,))
```

Create a model and fit it

The next step is to create a linear regression model and fit it using the existing data. With .fit(), you calculate the optimal values of the weights b_0 and b_1, using the existing input and output (x and y) as the arguments. In other words, .fit() fits the model.

Let’s create an instance of the class LinearRegression, which will represent the regression model:

In [8]:
```python
#Create the model by creating the object of class LinearRegression
regr = LinearRegression()
#use the object to fit the model on the data
#First, you need to call .fit() on model:
regr.fit(X_train,Y_train)
```

Out[8]:
```
LinearRegression()
```

In [9]:
```python
# find intercept of the linear equation
regr.intercept_
```

Out[9]:
```
-1.3154407102092618
```

In [10]:
```python
# find slope for 2 independent variables
regr.coef_
```
Out[10]:
```
array([0.06449588, 0.97831325])
```

In [11]:
```python
# Do the prediction for test data set
y_pred= regr.predict(X_test)
y_pred
```
Out[11]:
```
array([9.04740013, 3.86597971])
```

In [12]:
```python
# create a dataframe for actual and predicted values
d = pd.DataFrame({"Actual value": Y_test, "predicted values": y_pred})
d
```
Out[12]:

	Actual value	predicted values
2	8.9	9.04740
4	4.2	3.86598

Prediction on a particular data points suppose miles_travelled= 200 n_of_deliveries=5

In [13]:
```python
predicted = regr.predict([[200, 5]])
print(predicted)
```
```
[16.4753012]
```

The sklearn. metrics module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values.

In [14]:
```python
# calculate mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(Y_test, y_pred)
```
Out[14]:
```
0.06664817632509885
```

R sqaured-coefficient of determination--goodness of fit R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable

In [15]:
```python
# Adjusted  R squared -coefficient of determination--goodness of fit
from sklearn.metrics import r2_score
r2_score(Y_test, y_pred)
```
Out[15]:
```
0.9879315208103036
```

In [ ]:
```python

```

Linear regresssion

In [1]:
```python
import pandas as pd
import numpy as np
from sklearn import linear_model
from scipy import stats
import seaborn as sns
import matplotlib.pyplot as plt
```

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. Please note that sklearn is used to build machine learning models.

linear_model is a module in sklearn library:The linear_model module implements generalized linear models. It includes Ridge regression, Bayesian Regression, Lasso regresssion etc.

SciPy is an open-source Python library which is used to solve scientific and mathematical problems. It is built on the NumPy extension and allows the user to manipulate and visualize data with a wide range of high-level commands. As mentioned earlier, SciPy builds on NumPy and therefore if you import SciPy, there is no need to import NumPy.

Python Seaborn library is used to ease the challenging task of data visualization and it’s based on Matplotlib. Seaborn allows the creation of statistical graphics through the following functionalities:

Seaborn supports multi-plot grids that in turn ease building complex visualizations

Availability of different color palettes to reveal various kinds of patterns

Estimates and plots linear regression automatically

Matplotlib can be personalized but it’s difficult to figure out what settings are required to make plots more attractive. On the other hand, Seaborn comes with numerous customized themes and high-level interfaces to solve this issue.

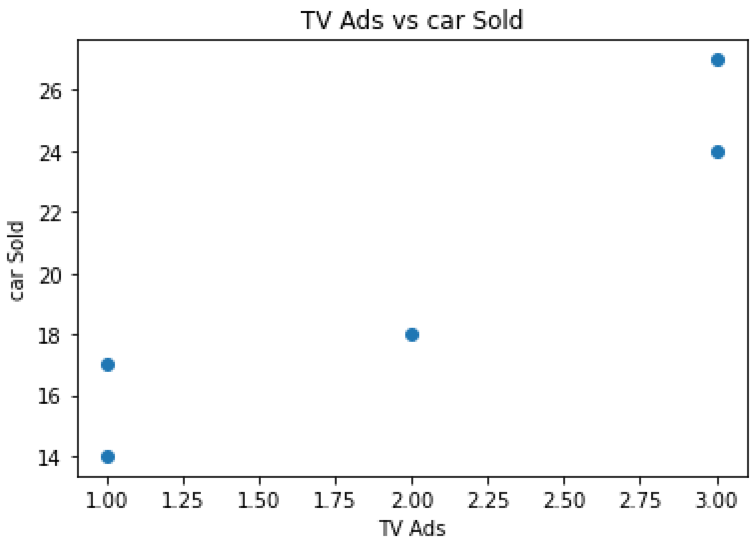
In [2]:
```python
# read the data into python notebook: the data is in excel format
data = pd.read_excel \
("C:/Users/subbu/Downloads/datasets for ML lab/linear_reg.xlsx")
data
```

Out[2]:

	TV Ads	car Sold
0	1	14
1	3	24
2	2	18
3	1	17
4	3	27

In [3]:

```python
# plot a scatter plot between car sold and TV adds
import matplotlib.pyplot as plt
plt.scatter(data['TV Ads'], data['car Sold'])
plt.title('TV Ads vs car Sold')
plt.xlabel('TV Ads')
plt.ylabel('car Sold')
plt.show()
```



In [4]:

```python
x = data['TV Ads'].values
y = data['car Sold'].values
```

In [5]:

```python
x
```

Out[5]:

```
array([1, 3, 2, 1, 3], dtype=int64)
```

In [6]:

```python
x.ndim
```

Out[6]:

```
1
```

reshape(-1, 1) results in an array with a single column and multiple rows (a column vector)

In [7]:

```python
x = data['TV Ads'].values.reshape(-1,1)
y = data['car Sold'].values.reshape(-1,1)
```

In [8]:

```python
x.ndim
```

Out[8]:

```
2
```

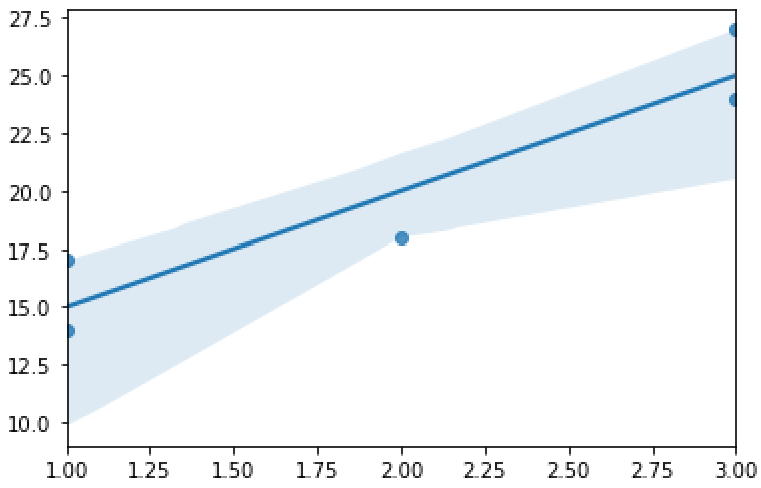
In [9]:

```python
x
```

Out[9]:

```
array([[1],
       [3],
       [2],
       [1],
       [3]], dtype=int64)
```

In [10]:
```
C:\Users\subbu\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. 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(
```



model_selection module has a function train_test_split():Split arrays or matrices into random train and test subsets

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = \
train_test_split(x, y, test_size=0.2, random_state=0)
```

Random state ensures that the splits that you generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order

In [11]:
```
# regplot: Plot data and a linear regression model fit.
sns.regplot(x, y)
# show the plot
plt.show()
```

In [12]:
```
x_train
```

Out[12]:
```
array([[1],
       [3],
       [1],
       [3]], dtype=int64)
```

In [13]:
```
y_train
```

Out[13]:
```
array([[14],
       [24],
       [17],
       [27]], dtype=int64)
```

In [14]:
```
x_test
```
Out[14]: array([[2]], dtype=int64)

In [15]:
```
y_test
```
Out[15]: array([[18]], dtype=int64)

In [16]:
```
len(x_train)
```
Out[16]: 4

In [17]:
```
len(x_test)
```
Out[17]: 1

In [18]:
```
# LinearRegresssion is a class that helps to implement  Ordinary least
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
# Fit the model to the data
regressor.fit(x_train, y_train)
```
Out[18]: LinearRegression()

With Scikit-Learn it is extremely straight forward to implement linear regression models, as all you really need to do is import the LinearRegression class, instantiate it, and call the fit() method along with our training data. This is about as simple as it gets when using a machine learning library to train on your data.

In [19]:
```
# find the intercept of the regresssion line
regressor.intercept_
```
Out[19]: array([10.5])

In [20]:
```
# find the slope of the regresssion line
regressor.coef_
```
Out[20]: array([[5.]])

Making Predictions Now that we have trained our algorithm, it's time to make some predictions

In [21]:
```
#Predict using the linear model
y_predict = regressor.predict(x_test)
```

In [22]:
```
y_predict
```
Out[22]: array([[20.5]])

In [23]:
```
y_test
```
Out[23]: array([[18]], dtype=int64)

Evaluating the Algorithm

1. Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|$$

2. Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2$$

3. Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}|Actual - Predicted|^2}$$

The final step is to evaluate the performance of algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

In [24]:
```
# metrics module is used to assess performance on different tasks

from sklearn import metrics
print('Mean Absolute Error:',\
      metrics.mean_absolute_error(y_test, y_predict))
print('Mean Squared Error:', \
      metrics.mean_squared_error \
      (y_test, y_predict))
print('Root Mean Squared Error:', \
      np.sqrt(metrics.mean_squared_error(y_test, y_predict)))
```

```
Mean Absolute Error: 2.5
Mean Squared Error: 6.25
Root Mean Squared Error: 2.5
```

R squared - coefficient of determination gives you goodness of fit, SSR/SST R-squared values range from 0 to 1 and are commonly stated as percentages from 0% to 100%

In [25]:
```
regressor.score(x_train, y_train)
```

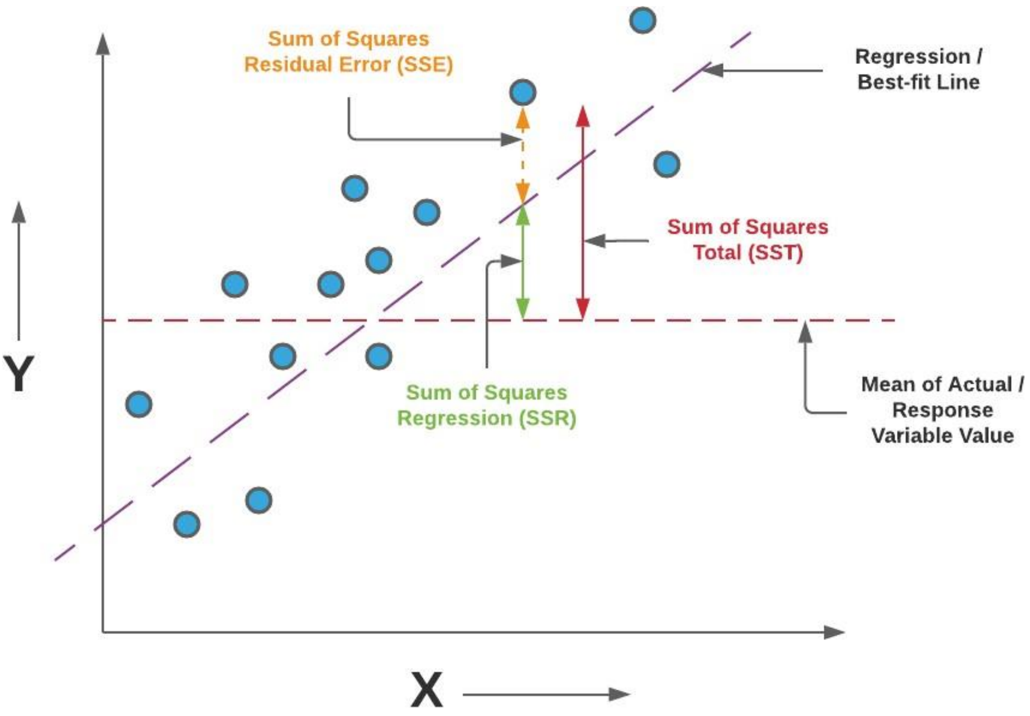
Out[25]: 0.9174311926605505

91%of variability in Y is expressed by the independent variable X

In [26]:

```
from IPython import display
display.Image("https://vitalflux.com/wp-content/uploads/2020/09/Regress

```

Out[26]:



In [27]:

```
from IPython import display
display.Image("https://vitalflux.com/wp-content/uploads/2019/07/R-squar

```

Out[27]:

$$R^2 = \frac{SSR}{SST} = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

In [ ]:

```


```

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [2]:

```
#Load and return the boston house-prices dataset (regression).
from sklearn.datasets import load_boston
data=load_boston()
```

In [3]:

```
data.feature_names
```

Out[3]:

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [4]:

```
x = data.data
x
```

Out[4]:

```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]])
```

In [5]:
```python
y=data.target
y
```

Out[5]:
```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
       18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
       15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
       13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
       21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
       35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
       19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
       20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
       23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
       33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
       21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
       20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
       23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
       15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
       17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
       25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
       23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
       32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
       34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
       20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
       26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
       31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
       22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
       42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
       36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
       32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
       20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
       20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
       22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
       21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
       19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
       32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
       18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
       16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
       13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
        7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
       12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
       27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
        8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
        9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
       10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
       15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
       19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
       29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
       20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
       23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9])
```

In [6]:
```python
#Split the training data
x_train, x_test, y_train, y_test = train_test_split\
(x,y,test_size=0.3, random_state=43)
```

In [7]:
```python
print(x_train.shape)
```

```
(354, 13)
```

In [8]:
```
print(x_test.shape)
```

```
(152, 13)
```

In [9]:
```
#Apply normal linear regression
from sklearn.linear_model import LinearRegression
linearreg =LinearRegression()
linearreg.fit(x_train, y_train)
```

Out[9]:
```
LinearRegression()
```

In [10]:
```
# predicting on test
linearreg_prediction = linearreg.predict(x_test)
linearreg_prediction
```

Out[10]:
```
array([23.49502283, 16.64276102, 20.89086257, 34.10893059, 16.99684444,
       12.37651684, 13.23003113, 25.10420394, 22.52995858,  7.35571615,
       27.9213835 , 26.67641743, 16.5561431 , 20.90705209, 22.21161678,
       12.02679628, 31.82627493, 33.10034681, 14.52291086, 41.17452207,
       20.72250171,  8.22482989, 21.00552944, 31.66548137,  6.38626137,
       25.47222083, 20.14001182, 19.75581079, 19.23289073,  6.55277632,
       21.47501184, 23.54992827, 15.5442664 ,  8.8745995 , 24.34972669,
       18.87328807, 16.18536356, 20.56369472, 28.67823077, 26.60884989,
       20.7244171 ,  5.28560207, 25.65967531, 17.89886503, 38.73954273,
       18.18256383, 24.13572711, 22.3127956 , 35.48734863, 18.08937758,
       27.95173764, 37.94797743, 35.08152847, 22.0344038 , 11.65601124,
       38.38853458, 33.85752175, 25.83803957, 28.78726471, 28.76691049,
       22.65628446, 14.08134402, 25.95352147, 27.7537849 , 27.18779307,
       13.00127534, 22.71248911, 36.66874059,  7.55607599, 24.33696695,
       12.88762168,  2.40023681,  8.27969373, -1.01789771,  5.99890289,
       27.30905412, 25.05994967, 19.66344897, 22.69592181, 20.34334117,
       12.43024004, 19.74084804, 11.62586045,  7.01841126, 17.09966975,
       30.31436365, 13.81473038, 37.18909581, 12.30324775, 10.7785622 ,
       15.34811923, 19.14486597, 21.3660069 , 30.15858641, 23.6873954 ,
       19.94724806, 34.27532188, 29.55575686, 24.52831471, 34.93895373,
       14.20250874, 21.39881629,  7.23549177, 26.32745162, 20.04234146,
        0.70444176,  1.02750827, 20.03437141, 20.62712589, 29.41560366,
       -0.26523122, 30.42356138, 15.00853848, 11.43685914, 15.69049156,
       18.29956041, 14.93771572, 18.81330867, 10.35226161, 17.91140174,
       24.19908289, 22.70563824, 18.64476024,  2.16848497, 20.95367378,
       25.21829799, 29.48482413, 31.77081071, 12.05039643, 29.58990815,
       30.48179728, 23.35366316, 22.85172393, 22.2008591 , 18.71575615,
       22.22570666, 21.09880979, 44.00740549, 20.42379073, 27.69131371,
       17.65292798, 24.07122638, 14.71832381, 15.98130109, 31.71573884,
       27.14148715, 10.36356238, 30.44732014, 25.0519413 , 20.0284205 ,
       21.93968285, 34.08257599])
```

In [11]:

```
from sklearn import metrics
from sklearn.metrics import r2_score
from numpy import sqrt
print('Mean absolute error : \
',metrics.mean_absolute_error(linearreg_prediction, y_test))
print('Mean square error : \
', metrics.mean_squared_error(linearreg_prediction, y_test))
print('R squared error', \
      r2_score(linearreg_prediction, y_test))
print('RMSE', \
      sqrt(metrics.mean_squared_error(linearreg_prediction, y_test)))
```

```
Mean absolute error :3.588546961776265
Mean square error :  25.40889031475487
R squared error 0.682307898477743
RMSE 5.04072319362558
```

In [12]:

```
linearreg.intercept_
```

Out[12]:

```
35.815750124268575
```

In [13]:

```
linearreg.coef_
```

Out[13]:

```
array([-3.14806846e-02,  4.35491340e-02,  3.72218331e-02,  2.59144276e+00,
       -1.76489873e+01,  3.78370328e+00,  2.20848708e-02, -1.24099405e+00,
        3.45873260e-01, -1.42131333e-02, -1.00537656e+00,  1.13001767e-02,
       -6.47916500e-01])
```

In [14]:

```
coefficient_df = pd.DataFrame()
coefficient_df["Column_Name"] = data.feature_names
coefficient_df['Coefficient_Value'] = pd.Series(linearreg.coef_)
print(coefficient_df.head(15))
```

```
   Column_Name  Coefficient_Value
0         CRIM          -0.031481
1           ZN           0.043549
2        INDUS           0.037222
3         CHAS           2.591443
4          NOX         -17.648987
5           RM           3.783703
6          AGE           0.022085
7          DIS          -1.240994
8          RAD           0.345873
9          TAX          -0.014213
10     PTRATIO          -1.005377
11           B           0.011300
12       LSTAT          -0.647916
```

Ridge Regression: Performs L2 regularization, i.e., adds penalty equivalent to the square of the magnitude of coefficients alpha: default value is 1.0. Constant that multiplies the L2 term, controlling regularization strength. alpha must be a non-negative float i.e. in [0, inf).

In [15]:

```
# Ridge regresssion

from sklearn.linear_model import Ridge

ridgeRegressor = Ridge(alpha =1)
ridgeRegressor.fit(x_train, y_train)
y_predict_ridge = ridgeRegressor.predict(x_test)
```

In [16]:

```
from sklearn import metrics
from sklearn.metrics import r2_score
from numpy import sqrt
print('Mean absolute error : \
', metrics.mean_absolute_error(y_predict_ridge, y_test))
print('Mean square error : \
', metrics.mean_squared_error(y_predict_ridge, y_test))
print('R squared error',\
      r2_score(y_predict_ridge, y_test))
print('RMSE', \
      sqrt(metrics.mean_squared_error(y_predict_ridge, y_test)))
```

```
Mean absolute error :3.587459850679014
Mean square error :  25.93135076792913
R squared error 0.6763150257666342
RMSE 5.092283453219109
```

In [17]:

```

# putting together the coefficients and their corresponding variable nc

coefficient_df = pd.DataFrame()
coefficient_df["Column_Name"] =  data.feature_names
coefficient_df['Coefficient_Value'] = pd.Series(ridgeRegressor.coef_)
print(coefficient_df.head(15))
```

	Column_Name	Coefficient_Value
0	CRIM	-0.021845
1	ZN	0.044846
2	INDUS	0.000143
3	CHAS	2.357306
4	NOX	-8.984993
5	RM	3.803084
6	AGE	0.014380
7	DIS	-1.121314
8	RAD	0.334100
9	TAX	-0.015235
10	PTRATIO	-0.924468
11	B	0.012155
12	LSTAT	-0.663183

Lasso regresssion; Performs L1 regularization, i.e., adds penalty equivalent to the absolute value of the magnitude of coefficients

In [18]:

```python
# Lasso regresssion

from sklearn.linear_model import Lasso

lassoRegressor = Lasso(alpha = 1)
lassoRegressor.fit(x_train, y_train)

# predicting on test
y_predicted_lasso = lassoRegressor.predict(x_test)

from sklearn import metrics
from sklearn.metrics import r2_score
from numpy import sqrt
print('Mean absolute error : \
', metrics.mean_absolute_error(y_predicted_lasso, y_test))
print('Mean square error : \
', metrics.mean_squared_error(y_predicted_lasso, y_test))
print('R squared error', \
      r2_score(y_predicted_lasso, y_test))
print('RMSE',\
      sqrt(metrics.mean_squared_error(y_predicted_lasso, y_test)))

# putting together the coeffcient and their corresponsing variable nam
coefficient_df = pd.DataFrame()
coefficient_df["Column_Name"] = data.feature_names
coefficient_df['Coefficient_Value'] = pd.Series(lassoRegressor.coef_)
print(coefficient_df.head(15))
```

```
Mean absolute error :4.104350191942339
Mean square error :  32.26413511588532
R squared error 0.5897220417210223
RMSE 5.680152737020838
   Column_Name  Coefficient_Value
0         CRIM          -0.000000
1           ZN           0.051067
2        INDUS          -0.000000
3         CHAS           0.000000
4          NOX          -0.000000
5           RM           0.743436
6          AGE           0.045691
7          DIS          -0.483200
8          RAD           0.350588
9          TAX          -0.018205
10     PTRATIO          -0.809085
11           B           0.011211
12       LSTAT          -0.915162
```

In [19]:

```python
#From the above output we see some attributes parameters became 0.
```

In [ ]:

```python

```

In [1]:
```python
# import the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```python
data=pd.read_csv \
('C:/Users/subbu/OneDrive/Desktop/machine Learning/experiments\
/diabetes.csv')
data
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [3]:
```python
print(len(data))
```

```
768
```

In [4]:
```python
data.columns
```

Out[4]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

There are certain columns where 0 values are there like in SkinThickness,Insulin,glucose

Replace zeros with mean of those column

In [5]:
```python
(data == 0).sum()
```

Out[5]:
```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

In [6]:

```python
non_zero =['Glucose', 'BloodPressure','SkinThickness','BMI', 'Insulin']
#Iterating all columns wherever 0 is there & substituting with NaN
#NaN defined in NUMPY LIBRARY
# Then we are replacing NaN with mean of the column
for column in non_zero:
    data[column] = data[column].replace(0, np.NaN)
    mean = int(data[column].mean(skipna = True))
    data[column] = data[column].replace(np.NaN, mean)
```

In [7]:

```python
#NaN is short for Not a number.
#It is used to represent entries that are undefined or
#missing values
#example
v= np.array([1, np.NaN, 3, 4])
np.sum(v)
```

Out[7]: nan

In [8]:

```python
np.nansum(v)
```

Out[8]: 8.0

In [9]:

```python
print(data['SkinThickness'])
```

```
0      35.0
1      29.0
2      29.0
3      23.0
4      35.0
       ...
763    48.0
764    27.0
765    23.0
766    29.0
767    31.0
Name  SkinThickness, Length: 768, dtype: float64
```

In [10]:

```python
x = data.iloc[:, 0:8]
x
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.0	155.0	33.6	0.627	50
1	1	85.0	66.0	29.0	155.0	26.6	0.351	31
2	8	183.0	64.0	29.0	155.0	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63
764	2	122.0	70.0	27.0	155.0	36.8	0.340	27
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30
766	1	126.0	60.0	29.0	155.0	30.1	0.349	47
767	1	93.0	70.0	31.0	155.0	30.4	0.315	23

768 rows × 8 columns

-sklearn Library: Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.

•Supervised learning algorithms •Cross-validation •Unsupervised learning algorithms •Various toy datasets: (e.g. IRIS dataset, Boston House prices dataset). •Feature extraction: Scikit-learn for extracting features from images

In [11]:
```
y = data.iloc[:, 8]
y
```

Out[11]:
```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

-model_selection: It is a method for setting a blueprint to analyze data and then using it to measure new data. Selecting a proper model allows you to generate accurate results when making a prediction.

-train_test_split is a function in Sklearn model selection for splitting data arrays into two subsets: for training data and for testing data. With this function, you don't need to divide the dataset manually.

In [27]:
```
#Split the data into train and test
from sklearn.model_selection import train_test_split
x = data.iloc[:, 0:8]
y = data.iloc[:, 8]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,\
                                               random_state=89)
```

In [28]:
```
print(x)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6    148.0           72.0           35.0    155.0  33.6
1              1     85.0           66.0           29.0    155.0  26.6
2              8    183.0           64.0           29.0    155.0  23.3
3              1     89.0           66.0           23.0     94.0  28.1
4              0    137.0           40.0           35.0    168.0  43.1
..           ...      ...            ...            ...      ...   ...
763           10    101.0           76.0           48.0    180.0  32.9
764            2    122.0           70.0           27.0    155.0  36.8
765            5    121.0           72.0           23.0    112.0  26.2
766            1    126.0           60.0           29.0    155.0  30.1
767            1     93.0           70.0           31.0    155.0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        ...  ...
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

In [14]:
```python
print(y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

```
Data standardization is the process of rescaling the attributes so that they have mean as 0 and variance as 1.
The ultimate goal to perform standardization is to bring down all the features to a common scale without distorting the differences in the range of the values.
In sklearn.preprocessing.StandardScaler(), centering and scaling happens independently on each feature.
The idea behind StandardScaler is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1.
python sklearn library offers us with StandardScaler() function to standardize the data values into a standard format with mean 0 and standard deviation 1
```

```
feature scaling is done by making mean 0 and standard deviation 1 of every input column
fit_tranform function is used on x_train to learn paramters(mean and standard deviation)
so that standardization can happpen use the tranform function on x_text to use the paramters(mean and standard deviation)
of train set on set test
```

In [30]:
```python

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x_train = sc.fit_transform(x_train)

x_test = sc.transform(x_test)
```

```
object = StandardScaler()
object.fit_transform(data)
According to the above syntax, we initially create an object of the StandardScaler() function. Further, we use fit_transform()
fit_transform() is used on the training data so that we can scale the training data and also learn the scaling parameters of that data (mean and standard deviation) . Here, the model built by us will learn the mean and standard deviation of the features of the training set. These learned parameters are then used to scale our test data also.
Using the transform method we can use the same mean and standard deviation as it is calculated from our training data to transform our test data. Thus, the parameters learned by our model using the training data will help us to transform our test data also.
```

In [31]:
```
x_train
```

Out[31]:
```
array([[-0.83444704,  2.4607925 ,  0.2729567 , ...,  1.49054148,
         2.83104312, -0.95428911],
       [ 0.6549439 ,  0.46408671, -0.98652789, ..., -1.1976073 ,
        -0.97626322,  1.81271231],
       [-0.83444704,  0.23865218, -0.23083713, ..., -0.95323013,
         0.00695909, -0.95428911],
       ...,
       [-0.83444704, -1.46820922, -1.40635608, ...,  0.11052928,
         2.39303443, -0.78659205],
       [-0.23869066,  1.52684947, -0.73463097, ...,  0.2830308 ,
        -0.34834866, -0.28350089],
       [ 0.35706571,  1.04377549,  0.94468182, ...,  0.88678614,
         0.45721978,  0.05189323]])
```

Generating Model Let's build KNN classifier model.

First, import the KNeighborsClassifier module and create KNN classifier object by passing argument number of neighbors in KNeighborsClassifier() function.

In [32]:
```
# define the model
from sklearn.neighbors import KNeighborsClassifier
#classifier =  KNeighborsClassifier(n_neighbors = 11, p =2, metric = 'euclidean')
classifier =  KNeighborsClassifier(n_neighbors = 11, \
                                   metric = 'euclidean')
# p= power paramter 1 for manhatten and 2 for euclidian
```

Why k is odd: Let's think for a while: The k, in the KNN algorithm, represent the number of closest neighbors that you are comparing, right? So, no matter if you have 2 or n classes, if you choose an even k, there is a risk of a tie in the decision of which class you should set a new instance. This is why the k is usually odd - no ties.

Fit the model on the train set using fit() and perform prediction on the test set using predict().

In [18]:
```
classifier.fit(x_train, y_train)
```

Out[18]:
```
KNeighborsClassifier(metric='euclidean', n_neighbors=11)
```

In [19]:
```
# prediction on test set
y_pred = classifier.predict(x_test)
y_pred
```

Out[19]:
```
array([0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0],
      dtype=int64)
```

In [20]:
```
# Generate the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm);

```

```
[[82 21]
 [23 28]]
```

In [21]: 0.7142857142857143

```
# Find F1 score
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred))
```

In [22]: 0.5599999999999999

```
# Find accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

Elbow method: helps to find optimal value of k Elbow method helps data scientists to select the optimal number of neighbors for KNN. As K increases, the error usually goes down, then stabilizes, and then raises again. Pick the optimum K at the beginning of the stable zone. This is also called Elbow Method.

In [23]:

```
from sklearn.neighbors import KNeighborsClassifier
error_rate = []
for i in range(1,40):

 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(x_train,y_train)
 pred_i = knn.predict(x_test)
 error_rate.append(np.mean(pred_i != y_test))
```

In [24]:

```
error_rate
```

Out[24]:

```
[0.3246753246753247,
 0.2922077922077922,
 0.2792207792207792,
 0.2792207792207792,
 0.2922077922077922,
 0.2727272727272727,
 0.2727272727272727,
 0.2662337662337662,
 0.2792207792207792,
 0.2662337662337662,
 0.2857142857142857,
 0.2597402597402597,
 0.2792207792207792,
 0.2792207792207792,
 0.2597402597402597,
 0.2727272727272727,
 0.2532467532467532,
 0.2597402597402597,
 0.24675324675324675,
 0.24025974025974026,
 0.2532467532467532,
 0.2532467532467532,
 0.2532467532467532,
 0.2532467532467532,
 0.2532467532467532,
 0.24675324675324675,
 0.24675324675324675,
 0.2532467532467532,
 0.2597402597402597,
 0.2727272727272727,
 0.2792207792207792,
 0.2727272727272727,
 0.2662337662337662,
 0.2597402597402597,
 0.2597402597402597,
 0.2532467532467532,
 0.2792207792207792,
 0.2597402597402597,
 0.2662337662337662]
```

In [25]:

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color="blue", linestyle="dashed", marker="o",
 markerfacecolor="red", markersize=10)
plt.title("Error Rate vs. K Value")
plt.xlabel("K")
plt.ylabel("Error Rate")
```

Out[25]:

```
Text(0, 0.5, 'Error Rate')
```


```
Use the confusion_matrix method from sklearn.metrics to compute the confusion matrix.
 classification_report: Gives a text report showing the main classification metrics.
```

In [26]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
# NOW WITH K=20
knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(x_train,y_train)
pred = knn.predict(x_test)

print(confusion_matrix(y_test,pred))

print(classification_report(y_test,pred))
```

```
[[90 13]
 [24 27]]
              precision    recall  f1-score   support

           0       0.79      0.87      0.83       103
           1       0.68      0.53      0.59        51

    accuracy                           0.76       154
   macro avg       0.73      0.70      0.71       154
weighted avg       0.75      0.76      0.75       154
```

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
data=pd.read_excel('C:/Users/subbu/OneDrive/Desktop/machine Learning/experiments/dt.
```

In [3]:

```
print(data)
```

```
    RID          age  income student credit_rating buys_computer
0     1        youth    high      no          fair            no
1     2        youth    high      no     excellent            no
2     3  middle_aged    high      no          fair           yes
3     4       senior  medium      no          fair           yes
4     5       senior     low     yes          fair           yes
5     6       senior     low     yes     excellent            no
6     7  middle_aged     low     yes     excellent           yes
7     8        youth  medium      no          fair            no
8     9        youth     low     yes          fair           yes
9    10       senior  medium     yes          fair           yes
10   11        youth  medium     yes     excellent           yes
11   12  middle_aged  medium      no     excellent           yes
12   13  middle_aged    high     yes          fair           yes
13   14       senior  medium      no     excellent            no
```

In [4]:

```
#Encode the text or non numerical data into numerical value
```

In [5]:

```
from sklearn.preprocessing import LabelEncoder
```

In [6]:

```
# created instances for class LableEncoder
le_age = LabelEncoder()
le_income = LabelEncoder()
le_student = LabelEncoder()
le_credit_rating = LabelEncoder()
le_buys_computer = LabelEncoder()
```

In [7]:

```python
# fit_tranform
data['age_n']=le_age.fit_transform(data['age'])
data['income_n']=le_income.fit_transform(data['income'])
data['student_n']=le_student.fit_transform(data['student'])
data['credit_rating_n']=le_credit_rating.fit_transform(data['credit_rating'])
data['buys_computer_n']=le_buys_computer.fit_transform(data['buys_computer'])

```

In [8]:

```python
data.head()
```

Out[8]:

	RID	age	income	student	credit_rating	buys_computer	age_n	income_n	stud
0	1	youth	high	no	fair	no	2	0	
1	2	youth	high	no	excellent	no	2	0	
2	3	middle_aged	high	no	fair	yes	0	0	
3	4	senior	medium	no	fair	yes	1	2	
4	5	senior	low	yes	fair	yes	1	1	

In [9]:

```python
data_new=data.drop(['age','income','student','credit_rating','buys_computer'],axis=1
data_new
```

Out[9]:

	RID	age_n	income_n	student_n	credit_rating_n	buys_computer_n
0	1	2	0	0	1	0
1	2	2	0	0	0	0
2	3	0	0	0	1	1
3	4	1	2	0	1	1
4	5	1	1	1	1	1
5	6	1	1	1	0	0
6	7	0	1	1	0	1
7	8	2	2	0	1	0
8	9	2	1	1	1	1
9	10	1	2	1	1	1
10	11	2	2	1	0	1
11	12	0	2	0	0	1
12	13	0	0	1	1	1
13	14	1	2	0	0	0

In [10]:

```
feature_cols=['age_n', 'income_n', 'student_n', 'credit_rating_n']
x = data_new.drop(['buys_computer_n', 'RID'],axis = 'columns')
y = data_new['buys_computer_n']
```

In [11]:

```
x
```

Out[11]:

	age_n	income_n	student_n	credit_rating_n
0	2	0	0	1
1	2	0	0	0
2	0	0	0	1
3	1	2	0	1
4	1	1	1	1
5	1	1	1	0
6	0	1	1	0
7	2	2	0	1
8	2	1	1	1
9	1	2	1	1
10	2	2	1	0
11	0	2	0	0
12	0	0	1	1
13	1	2	0	0

In [12]:

```
y
```

Out[12]:

```
0     0
1     0
2     1
3     1
4     1
5     0
6     1
7     0
8     1
9     1
10    1
11    1
12    1
13    0
Name: buys_computer_n, dtype: int32
```

In [13]:

```python
# for splitting
from sklearn.model_selection import train_test_split
```

In [14]:

```python
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size = 0.25,random_state=
```

In [15]:

```python
x_train
```

Out[15]:

	age_n	income_n	student_n	credit_rating_n
5	1	1	1	0
8	2	1	1	1
2	0	0	0	1
1	2	0	0	0
13	1	2	0	0
4	1	1	1	1
7	2	2	0	1
10	2	2	1	0
3	1	2	0	1
6	0	1	1	0

In [16]:

```python
y_train
```

Out[16]:

```
5     0
8     1
2     1
1     0
13    0
4     1
7     0
10    1
3     1
6     1
Name: buys_computer_n, dtype: int32
```

In [17]:

```
x_test
```

Out[17]:

	age_n	income_n	student_n	credit_rating_n
9	1	2	1	1
11	0	2	0	0
0	2	0	0	1
12	0	0	1	1

In [18]:

```
y_test
```

Out[18]:

```
9     1
11    1
0     0
12    1
Name: buys_computer_n, dtype: int32
```

In [19]:

```
# concatenating the training dataset
pd.concat([x_train, y_train], axis = 1)
```

Out[19]:

	age_n	income_n	student_n	credit_rating_n	buys_computer_n
5	1	1	1	0	0
8	2	1	1	1	1
2	0	0	0	1	1
1	2	0	0	0	0
13	1	2	0	0	0
4	1	1	1	1	1
7	2	2	0	1	0
10	2	2	1	0	1
3	1	2	0	1	1
6	0	1	1	0	1

In [20]:

```
pd.concat([x_test, y_test], axis = 1)
```

Out[20]:

	age_n	income_n	student_n	credit_rating_n	buys_computer_n
9	1	2	1	1	1
11	0	2	0	0	1
0	2	0	0	1	0
12	0	0	1	1	1

In [21]:

```
# towards building our Decision Tree model
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(criterion = 'entropy')
dt = clf.fit(x_train, y_train)
dt
```

Out[21]:

```
DecisionTreeClassifier(criterion='entropy')
```

In [22]:

```
y_pred = dt.predict(x_test)
y_pred

```

Out[22]:

```
array([1, 1, 0, 1])
```

In [23]:

```
# metric
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[23]:

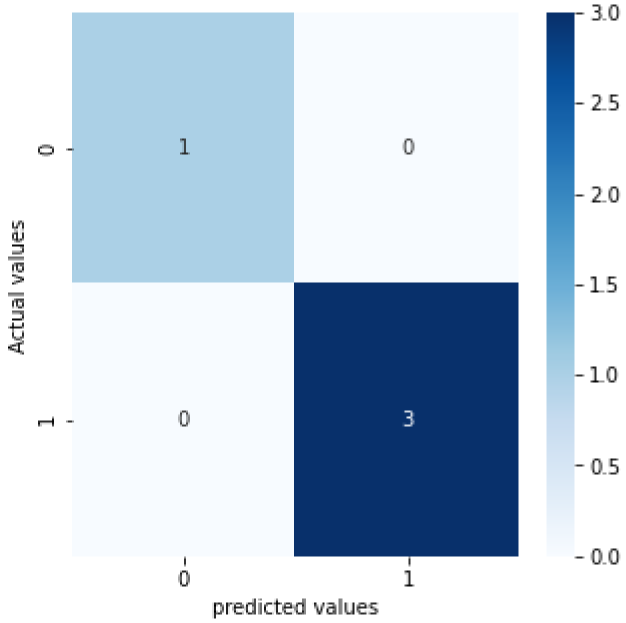
```
1.0
```

In [24]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

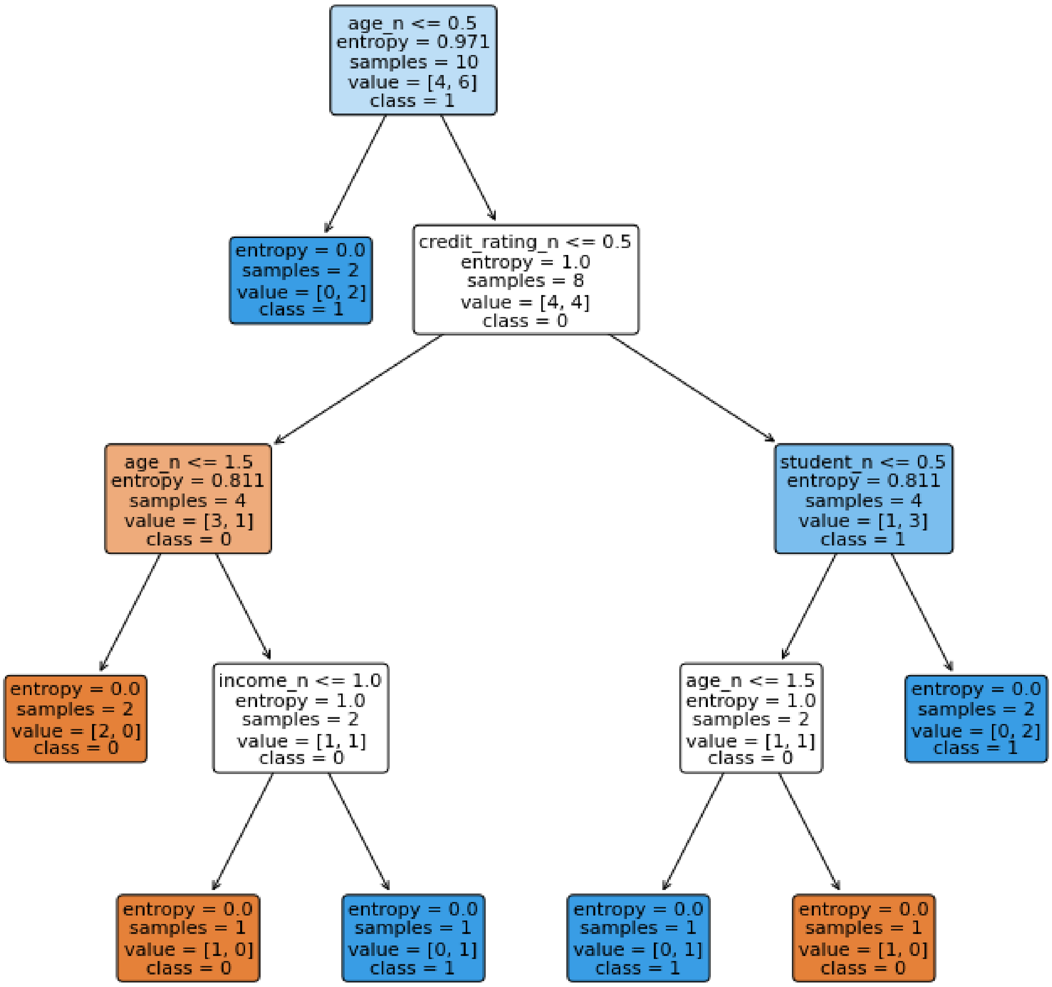
In [25]:

```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm, annot = True, cmap = 'Blues')
plt.ylabel('Actual values')
plt.xlabel('predicted values')
plt.show()

```



In [26]:

```
#graphical visualization of tree
from sklearn.tree import plot_tree
# help you to produce the figure of tree
plt.figure(figsize=(12,12))
dec_tree=plot_tree(decision_tree=dt,feature_names=feature_cols,class_names=["0","1"]
                   filled=True,rounded=True)
```



In [ ]:

```

```

In [1]:
```
# PCA: Principal component Analysis
import pandas as pd
import numpy as np
df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)
```

In [2]:
```
df_wine
```

Out[2]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows × 14 columns

In [3]:
```
df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue',
                   'OD280/OD315 of diluted wines', 'Proline']

df_wine.head()
```

Out[3]:

	Class label	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Pro
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	

In [4]:
```
np.unique(df_wine["Class label"])
```

Out[4]:
```
array([1, 2, 3], dtype=int64)
```

In [5]:
```
df_wine.shape
```

Out[5]:
```
(178, 14)
```

Splitting the data into 70% training and 30% test subsets.

In [6]:
```
from sklearn.model_selection import train_test_split

X  = df_wine.iloc[:, 1:].values
y = df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3,
                     stratify=y,
                     random_state=0)
```

Standardizing the data.

In [7]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

Eigen decomposition of the covariance matrix.

In [8]:
```
import numpy as np
cov_mat = np.cov(X_train_std.T)
eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)
print('\nEigenvalues \n%s' % eigen_vals)
```

```
Eigenvalues
[4.84274532 2.41602459 1.54845825 0.96120438 0.84166161 0.6620634
 0.51828472 0.34650377 0.3131368  0.10754642 0.21357215 0.15362835
 0.1808613 ]
```

In [9]:
```
print('\nEigenvectors \n%s' % eigen_vecs)
```

```
Eigenvectors
[[-1.37242175e-01  5.03034778e-01 -1.37748734e-01 -3.29610003e-03
   2.90625226e-01 -2.99096847e-01 -7.90529293e-02  3.68176414e-01
   3.98377017e-01 -9.44869777e-02  3.74638877e-01 -1.27834515e-01
   2.62834263e-01]
 [ 2.47243265e-01  1.64871190e-01  9.61503863e-02  5.62646692e-01
  -8.95378697e-02 -6.27036396e-01  2.74002014e-01  1.25775752e-02
  -1.10458230e-01  2.63652406e-02 -1.37405597e-01  8.06401578e-02
  -2.66769211e-01]
 [-2.54515927e-02  2.44564761e-01  6.77775667e-01 -1.08977111e-01
   1.60834991e-01 -3.89128239e-04 -1.32328045e-01 -1.77578177e-01
  -3.82496856e-01  1.42747511e-01  4.61583035e-01  1.67924873e-02
  -1.15542548e-01]
 [ 2.06945084e-01 -1.13529045e-01  6.25040550e-01  3.38187002e-02
  -5.15873402e-02  4.05836452e-02 -2.23999097e-01  4.40592110e-01
   2.43373853e-01 -1.30485780e-01 -4.18953989e-01 -1.10845657e-01
   1.99483410e-01]
 [-1.54365821e-01  2.89745182e-01  1.96135481e-01 -3.67511070e-01
  -6.76487073e-01 -6.57772614e-02  4.05268966e-01 -1.16617503e-01
   2.58982359e-01 -6.76080782e-02  1.00470630e-02  7.93879562e-02
   2.89018810e-02]
 [-3.93769523e-01  5.08010391e-02  1.40310572e-01  2.40245127e-01
   1.18511144e-01  5.89776247e-02  3.47419412e-02 -3.50192127e-01
   3.42312860e-01  4.59917661e-01 -2.21254241e-01 -4.91459313e-01
  -6.63868598e-02]
 [-4.17351064e-01 -2.28733792e-02  1.17053859e-01  1.87053299e-01
   1.07100349e-01  3.01103180e-02 -4.17835724e-02 -2.18718183e-01
   3.61231642e-02 -8.14583947e-01 -4.17513600e-02 -5.03074004e-02
  -2.13349079e-01]
 [ 3.05728961e-01  9.04888470e-02  1.31217777e-01 -2.29262234e-02
   5.07581610e-01  2.71728086e-01  6.31145686e-01 -1.97129425e-01
   1.71436883e-01 -9.57480885e-02 -8.87569452e-02  1.75328030e-01
   1.86391279e-01]
 [-3.06683469e-01  8.35232677e-03  3.04309008e-02  4.96262330e-01
  -2.01634619e-01  4.39997519e-01  3.23122775e-01  4.33055871e-01
  -2.44370210e-01  6.72468934e-02  1.99921861e-01 -3.67595797e-03
   1.68082985e-01]
 [ 7.55406578e-02  5.49775805e-01 -7.99299713e-02  1.06482939e-01
  -5.73607091e-03  4.11743459e-01 -2.69082623e-01  6.68411823e-02
   1.55514919e-01  8.73336218e-02 -2.21668868e-01  3.59756535e-01
  -4.66369031e-01]
 [-3.26132628e-01 -2.07164328e-01  5.30591506e-02 -3.69053747e-01
   2.76914216e-01 -1.41673377e-01  3.02640661e-01  4.59762295e-01
  -2.11961247e-02  1.29061125e-01 -9.84694573e-02  4.04669797e-02
  -5.32483880e-01]
 [-3.68610222e-01 -2.49025357e-01  1.32391030e-01  1.42016088e-01
   6.66275572e-02 -1.75842384e-01 -1.30540143e-01 -1.10827548e-01
   2.38089559e-01  1.87646268e-01  1.91205783e-02  7.42229543e-01
   2.37835283e-01]
 [-2.96696514e-01  3.80229423e-01 -7.06502178e-02 -1.67682173e-01
   1.28029045e-01 -1.38018388e-01 -8.11335043e-04 -5.60817288e-03
  -5.17278463e-01  1.21112574e-02 -5.42532072e-01  3.87395209e-02
   3.67763359e-01]]
```

In [10]:
```
print('\nEigenvalues \n%s' % eigen_vals)
```

```
Eigenvalues
[4.84274532 2.41602459 1.54845825 0.96120438 0.84166161 0.6620634
 0.51828472 0.34650377 0.3131368  0.10754642 0.21357215 0.15362835
 0.1808613 ]
```

Total and explained variance

In [11]:
```
tot = sum(eigen_vals)
var_exp = [(i / tot) for i in sorted(eigen_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
```

In [12]:
```
tot
```

Out[12]:
```
13.105691056910572
```

In [13]:
```
4.84274532/13.105691056910569
```

Out[13]:
```
0.3695146863275434
```

In [14]:
```
2.41602459/13.105691056910569
```

Out[14]:
```
0.18434927082506206
```

In [15]:
```
var_exp
```

Out[15]:
```
[0.36951468599607634,
 0.1843492705988419,
 0.11815159094596984,
 0.07334251763785465,
 0.06422107821731665,
 0.050517244849076534,
 0.03954653891241441,
 0.026439183169220004,
 0.023893192591852935,
 0.01629613773725101,
 0.013800211221948404,
 0.01172226244308595,
 0.008206085679091375]
```

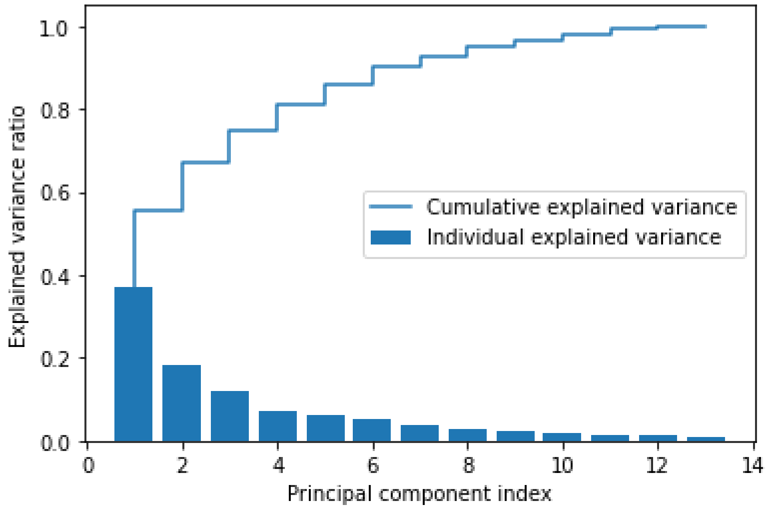
In [16]:
```
cum_var_exp
```

Out[16]:
```
array([0.36951469, 0.55386396, 0.67201555, 0.74535807, 0.80957914,
       0.86009639, 0.89964293, 0.92608211, 0.9499753 , 0.96627144,
       0.98007165, 0.99179391, 1.        ])
```

In [17]:
```
0.18434927082506206+0.3695146863275434
```

Out[17]:
```
0.5538639571526055
```

In [18]:
```
import matplotlib.pyplot as plt
plt.bar(range(1, 14), var_exp,
        label='Individual explained variance')
plt.step(range(1, 14), cum_var_exp,
         label='Cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal component index')
plt.legend()
plt.show()
```



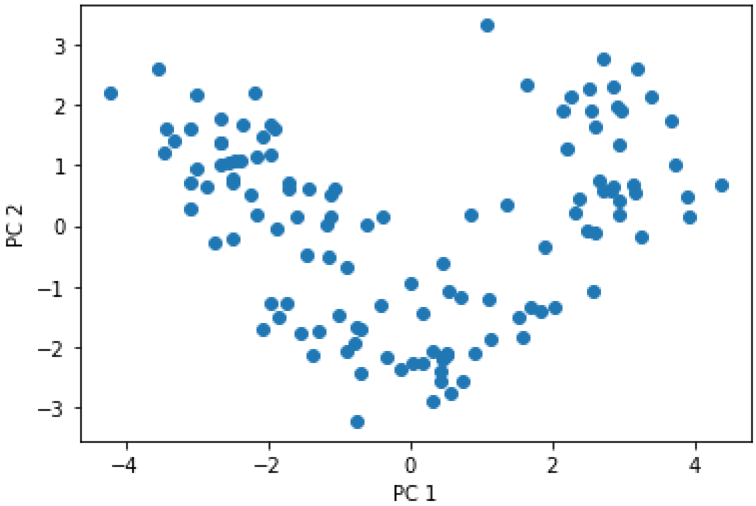
Principal component analysis in scikit-learn

In [19]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
```

In [20]:
```
X_train_pca
```

```
       [-1.12276518,  0.13877   ],
       [ 2.85996853,  2.28819559],
       [-0.74717125, -3.21746061],
       [-1.58427878,  0.16048055],
       [ 3.38887101,  2.11550689],
       [ 3.15405473,  0.54233966],
       [-1.28036506, -1.72926871],
       [-1.71438911,  0.71745249],
       [-1.55040291, -1.7580591 ],
       [ 1.10984489, -1.20480693],
       [-0.69108418, -1.71385374],
       [-2.086036  , -1.68453671],
       [ 2.90393456,  1.95258805],
       [-2.07635784,  1.47183304],
       [-1.74756185, -1.25842546],
       [ 2.59424456, -0.1056037 ],
       [-2.50372355,  0.70412212],
       [-2.19448402,  2.18657552],
       [ 3.91634534,  0.16136475],
       [-1.11739618,  0.51921086],
```

In [34]:
```
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1])
plt.xlabel('PC 1')
plt.ylabel('PC 2')
plt.show()
```



In [21]:
```
print(pca.explained_variance_ratio_)
```

```
[0.36951469 0.18434927]
```

Train a logistic model over the extracted features.

In [22]:
```
from sklearn.linear_model import LogisticRegression

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)

lr = LogisticRegression(multi_class='ovr', random_state=1)
lr = lr.fit(X_train_pca, y_train)
```

In [23]:
```
lr
```

Out[23]:
```
LogisticRegression(multi_class='ovr', random_state=1)
```

In [24]:
```
y_pred = lr.predict(X_test_pca)
y_pred
```

Out[24]:
```
array([1, 2, 1, 1, 2, 3, 2, 3, 1, 3, 1, 2, 3, 1, 3, 3, 2, 1, 3, 1, 1, 3,
       2, 2, 2, 2, 1, 3, 3, 2, 1, 3, 1, 2, 2, 1, 2, 2, 1, 3, 3, 2, 2, 2,
       1, 2, 2, 3, 2, 3, 2, 2, 1, 2], dtype=int64)
```

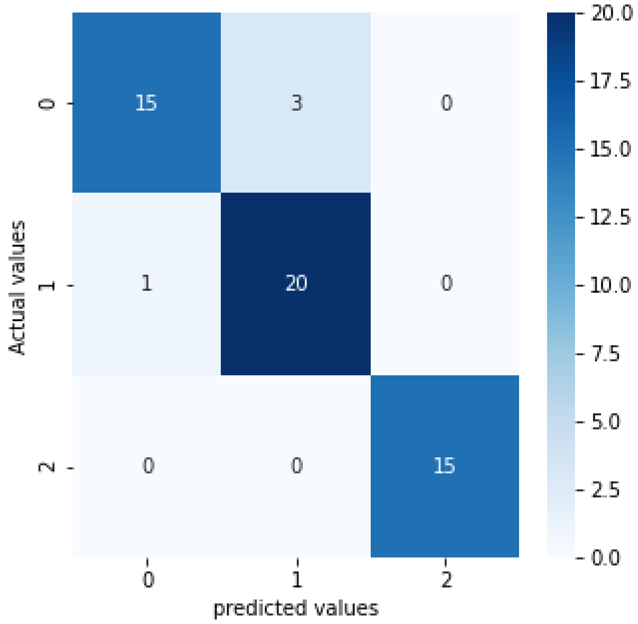
In [25]:
```
# metric
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[25]:
```
0.9259259259259259
```

Mlxtend stands for Machine Learning Extensions. It is a third-party Python library which contains many utilities and tools for machine learning and Data Science tasks, including feature selection, ensemble methods, visualization, and model evaluation.

In [32]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

In [33]:
```
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm, annot = True, cmap = 'Blues')
plt.ylabel('Actual values')
plt.xlabel('predicted values')
plt.show()
```



In [ ]:

LOGISTIC REGRESSION 1)What is logistic regresssion 2) What is sigmoid function - what it does in logistic regresssion 3) Why cant we use linear regresssion in problems where the output label is discrete

In [26]:

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

In [27]:

```
df=pd.read_excel\
('C:/Users/subbu/OneDrive/Desktop/machine Learning/experiments/Simmons.xls')
df
```

Out[27]:

	Customer	Spending	Card	Coupon
0	1	2.291	1	0
1	2	3.215	1	0
2	3	2.135	1	0
3	4	3.924	0	0
4	5	2.528	1	0
...	...	...	...	...
95	96	3.318	0	0
96	97	2.421	1	0
97	98	6.073	0	0
98	99	2.630	1	0
99	100	3.411	0	1

100 rows × 4 columns

In [28]:

```
#Find the length of the dataset
len(df)
```

Out[28]:

100

In [4]:

```python
#print the first five records
print(df.head(5))

```

```
   Customer  Spending  Card  Coupon
0         1     2.291     1       0
1         2     3.215     1       0
2         3     2.135     1       0
3         4     3.924     0       0
4         5     2.528     1       0
```

In [5]:

```python
#Describe your dataset
df.describe()
```

Out[5]:

	Customer	Spending	Card	Coupon
count	100.000000	100.000000	100.000000	100.000000
mean	50.500000	3.333790	0.500000	0.400000
std	29.011492	1.741298	0.502519	0.492366
min	1.000000	1.058000	0.000000	0.000000
25%	25.750000	2.059000	0.000000	0.000000
50%	50.500000	2.805500	0.500000	0.000000
75%	75.250000	4.468250	1.000000	1.000000
max	100.000000	7.076000	1.000000	1.000000

In [6]:

```python
# Find the shape of the dataset
df.shape
```

Out[6]:

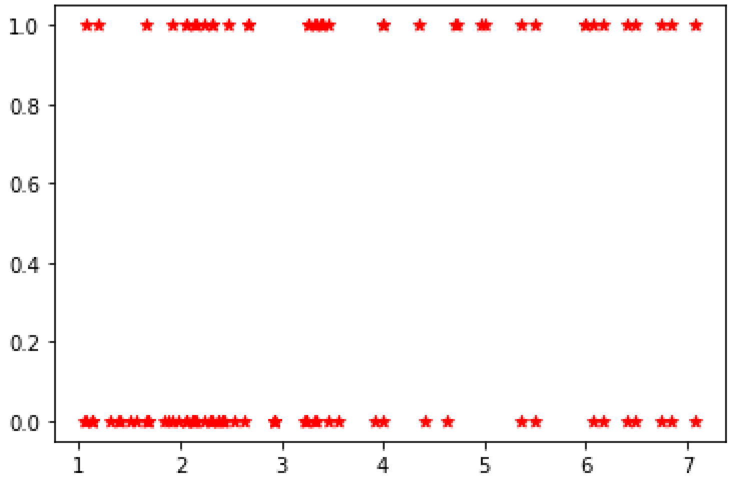
```
(100, 4)
```

In [7]:

```python
#Plot the scatter plot of spending and coupon and justify the graph
plt.scatter(df['Spending'], df['Coupon'], marker = '*', color = 'red')

```

Out[7]:

```
<matplotlib.collections.PathCollection at 0x263c855f820>
```



In [8]:

```python
#  x determines the independent variables
#x=df.drop(['Coupon', 'Customer'], axis=1 )
x = df[['Spending', 'Card']]
# y represents dependent variable
y=df['Coupon']

```

OR x = df.iloc[:,1:3] y = df.iloc[:,3]

In [9]:

```
x
```

Out[9]:

	Spending	Card
0	2.291	1
1	3.215	1
2	2.135	1
3	3.924	0
4	2.528	1
...	...	...
95	3.318	0
96	2.421	1
97	6.073	0
98	2.630	1
99	3.411	0

100 rows × 2 columns

In [10]:

```
y
```

Out[10]:

```
0     0
1     0
2     0
3     0
4     0
     ..
95    0
96    0
97    0
98    0
99    1
Name: Coupon, Length: 100, dtype: int64
```

In [11]:

```
# from sklearn library import model_selection module and train_test_split function
from sklearn.model_selection import train_test_split
#Split the data into random train and test subsets
#random_state is the seed used by the random number generator;
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_sta

```

In [12]:

```
#Logistic Regression (aka logit) classifier. Import LogisticRegression from sklearn.
from sklearn.linear_model import LogisticRegression
# create the object of class LogisticRegression
logmodel = LogisticRegression()
#fit-Fit the model according to the given training data.
logmodel.fit(x_train, y_train)
```

Out[12]:

```
LogisticRegression()
```

In [13]:

```
# do the predictions on x_test
predictions = logmodel.predict(x_test)
predictions
```

Out[13]:

```
array([1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0], dtype=int64)
```

In [14]:

```
#predict Probability estimates using predict_proba
prob = logmodel.predict_proba(x_test)
prob
```

Out[14]:

```
array([[0.34205601, 0.65794399],
       [0.76107428, 0.23892572],
       [0.58701304, 0.41298696],
       [0.31506144, 0.68493856],
       [0.31065746, 0.68934254],
       [0.46337996, 0.53662004],
       [0.61202787, 0.38797213],
       [0.53184912, 0.46815088],
       [0.76450974, 0.23549026],
       [0.49509649, 0.50490351],
       [0.37314736, 0.62685264],
       [0.77110125, 0.22889875],
       [0.53489588, 0.46510412],
       [0.64318576, 0.35681424],
       [0.74784036, 0.25215964],
       [0.34974311, 0.65025689],
       [0.49685384, 0.50314616],
       [0.69020399, 0.30979601],
       [0.47675934, 0.52324066],
       [0.54624978, 0.45375022],
       [0.54655938, 0.45344062],
       [0.51277761, 0.48722239],
       [0.5435772 , 0.4564228 ],
       [0.49666648, 0.50333352],
       [0.52299847, 0.47700153],
       [0.55613784, 0.44386216],
       [0.59137521, 0.40862479],
       [0.76450974, 0.23549026],
       [0.58222871, 0.41777129],
       [0.80394809, 0.19605191],
       [0.46337996, 0.53662004],
       [0.69842238, 0.30157762],
       [0.5747359 , 0.4252641 ]])
```

In [15]:

```
# create a datafreme for actual and predicted values
d = pd.DataFrame({"Actual value": y_test, "predicted values": predictions})
```

In [16]:

```
d
```

Out[16]:

	Actual value	predicted values
80	1	1
84	1	0
33	0	0
81	1	1
93	1	1
17	0	1
36	0	0
82	0	0
69	0	0
65	0	1
92	1	1
39	0	0
56	1	0
52	0	0
51	0	0
32	1	1
31	0	1
44	0	0
78	1	1
10	0	0
2	0	0
73	1	0
97	0	0
62	0	1
19	1	0
35	0	0
94	0	0
27	1	0
46	0	0
38	0	0
67	1	1
99	1	0
54	0	0

In [17]:

```python
# from sklearn.meterics import a function called classification_report to see the pe
from sklearn.metrics import classification_report
print(classification_report(y_test, predictions))

```

```
              precision    recall  f1-score   support

           0       0.73      0.80      0.76        20
           1       0.64      0.54      0.58        13

    accuracy                           0.70        33
   macro avg       0.68      0.67      0.67        33
weighted avg       0.69      0.70      0.69        33
```

Confusion matrix is a table which describes the performance of a prediction model. A confusion matrix contains the actual values and predicted values. we can use these values to calculate the accuracy score of the model.

In [18]:

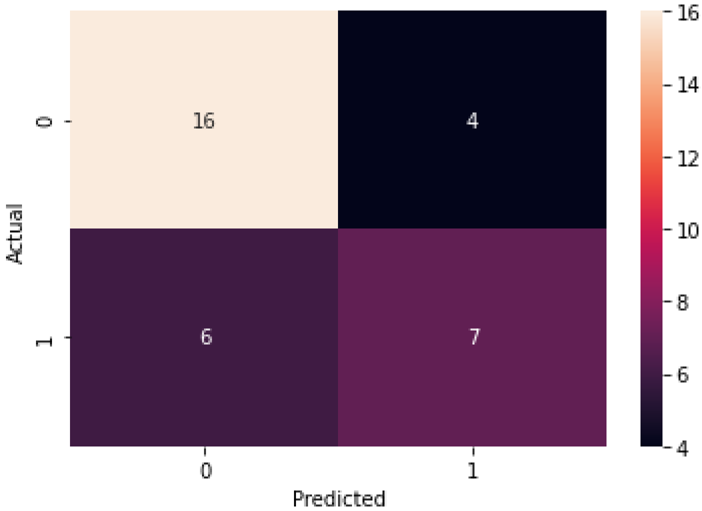
```python
#Create a confusion matrix by importing the function confusion_matrix
from sklearn.metrics import confusion_matrix,accuracy_score
confusion= confusion_matrix(y_test, predictions)
confusion
```

Out[18]:

```
array([[16,  4],
       [ 6,  7]], dtype=int64)
```

In [19]:

```
# Make a nice graphical confusion matrix
import matplotlib.pyplot as plt
import seaborn as sns

sns.heatmap(confusion, annot = True)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

```



In [20]:

```
#Accuracy score is the percentage of correct accuracy of the predictions made by the
print(accuracy_score(y_test, predictions))
```

```
0.696969696969697
```

In [21]:

```
# Find the slope coefficients for logistic regression for the two independent variab
logmodel.coef_
```

Out[21]:

```
array([[0.24982886, 0.9717996 ]])
```

In [22]:

```
# Find the intercept
logmodel.intercept_
```

Out[22]:

```
array([-1.69196285])
```

L1-regularized Logistic Regression

In [10]:
```
import pandas as pd
import numpy as np
```

In [11]:
```
df_wine = pd.read_csv('https://archive.ics.uci.edu/'
                      'ml/machine-learning-databases/wine/wine.data',
                      header=None)
```

In [12]:
```
df_wine
```

Out[12]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

178 rows × 14 columns

In [13]:
```
df_wine.shape
```

Out[13]:
```
(178, 14)
```

In [14]:
```
df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                   'Proline']
```

In [15]:
```
df_wine.columns
```

Out[15]:
```
Index(['Class label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash',
       'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',
       'Proanthocyanins', 'Color intensity', 'Hue',
       'OD280/OD315 of diluted wines', 'Proline'],
      dtype='object')
```

In [16]:
```
print('Class labels', np.unique(df_wine['Class label']))
```

```
Class labels [1 2 3]
```

In [19]:
```
from sklearn.model_selection import train_test_split

X, y = df_wine.iloc[:, 1:].values, df_wine.iloc[:, 0].values

X_train, X_test, y_train, y_test =\
    train_test_split(X, y,
                     test_size=0.3,
                     random_state=0,
                     stratify=y)
```

In [26]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

In [27]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(penalty='l1', C=1.0, solver='liblinear', multi_class
# Note that C=1.0 is the default. You can increase
# or decrease it to make the regulariztion effect
# C is the inverse regularization strength
# stronger or weaker, respectively.
lr.fit(X_train_std, y_train)
```

Out[27]:
```
LogisticRegression(multi_class='ovr', penalty='l1', solver='liblinear')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [28]:
```
y_pred = lr.predict(X_test_std)
y_pred
```

Out[28]:
```
array([1, 2, 1, 1, 2, 3, 2, 3, 1, 3, 1, 2, 3, 1, 3, 3, 1, 1, 3, 2, 1, 3,
       2, 2, 2, 2, 1, 3, 3, 2, 1, 3, 1, 2, 2, 1, 2, 1, 1, 3, 3, 2, 2, 2,
       1, 2, 2, 3, 2, 3, 2, 2, 1, 1], dtype=int64)
```

In [31]:
```
num_correct_predictions = (y_pred == y_test).sum()
accuracy = (num_correct_predictions / y_test.shape[0]) * 100

# print('Test set accuracy: %.2f%%' % accuracy)

print(f'Test set accuracy: {accuracy:}%')
```

```
Test set accuracy: 100.0%
```

In [32]:
```
lr.intercept_
```

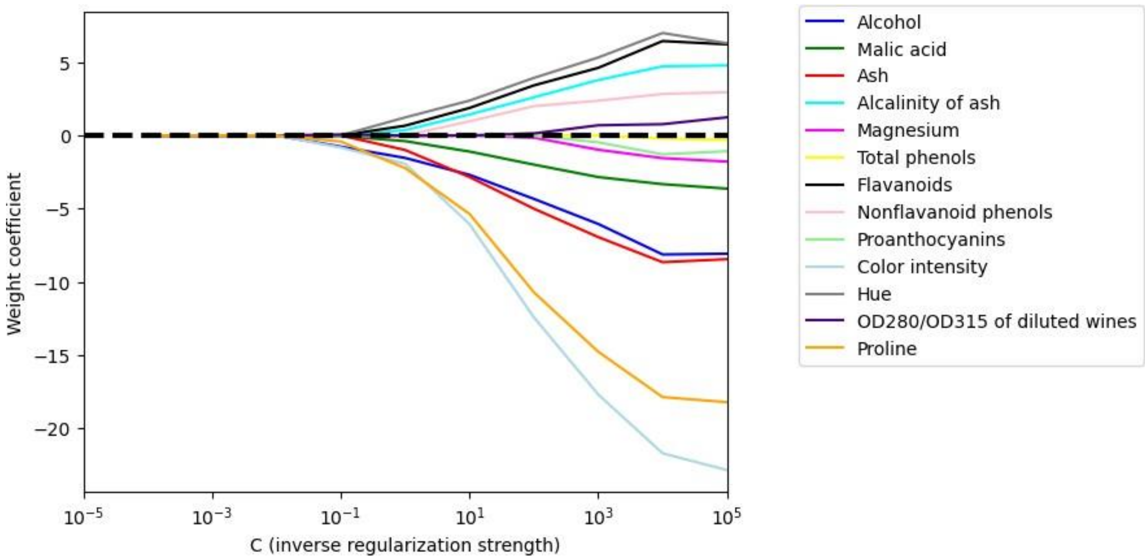
Out[32]:
```
array([-1.26351558, -1.21603898, -2.37064454])
```

In [33]:
```
lr.coef_
```

Out[33]:
```
array([[ 1.24573719,  0.18049256,  0.74478541, -1.16245516,  0.        ,
         0.        ,  1.16541599,  0.        ,  0.        ,  0.        ,
         0.        ,  0.55199904,  2.50984359],
       [-1.53682207, -0.387098  , -0.9952336 ,  0.36504426, -0.0594492 ,
         0.        ,  0.66834454,  0.        ,  0.        , -1.93454126,
         1.23326457,  0.        , -2.23122546],
       [ 0.13559451,  0.16849298,  0.35731636,  0.        ,  0.        ,
         0.        , -2.43768996,  0.        ,  0.        ,  1.56328995,
        -0.81877966, -0.49312847,  0.        ]])
```

Lasso path

In [34]:
```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = plt.subplot(111)

colors = ['blue', 'green', 'red', 'cyan',
          'magenta', 'yellow', 'black',
          'pink', 'lightgreen', 'lightblue',
          'gray', 'indigo', 'orange']

weights, params = [], []
for c in np.arange(-4., 6.):
    lr = LogisticRegression(penalty='l1', C=10.**c, solver='liblinear',
                            multi_class='ovr', random_state=0)
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)

for column, color in zip(range(weights.shape[1]), colors):
    plt.plot(params, weights[:, column],
             label=df_wine.columns[column + 1],
             color=color)
plt.axhline(0, color='black', linestyle='--', linewidth=3)
plt.xlim([10**(-5), 10**5])
plt.ylabel('Weight coefficient')
plt.xlabel('C (inverse regularization strength)')
plt.xscale('log')
#plt.legend(loc='upper left')
ax.legend(loc='upper center',
          bbox_to_anchor=(1.38, 1.03),
          ncol=1)
plt.show()
```



In [55]:
```
weights = np.array(weights)
weights.shape
```

Out[55]: (10, 13)

Expansion of commands

In [39]:
```
np.arange(-4., 6.)
```

Out[39]:
```
array([-4., -3., -2., -1.,  0.,  1.,  2.,  3.,  4.,  5.])
```

In [40]:
```
lr.coef_
```

Out[40]:
```
array([[  8.85827793,   1.78146388,   4.40003764,  -6.56912132,
          0.68139396,   0.03436769,   6.44866725,   0.45955986,
          1.35116061,  -2.51411858,  -2.70290329,   3.41110881,
         10.65447171],
       [ -8.0856239 ,  -3.63416028,  -8.4593189 ,   4.78819433,
         -1.78636854,  -0.30252817,   6.23457801,   2.95908431,
         -1.0768496 , -22.87914253,   6.30921537,   1.23987802,
        -18.23842479],
       [  4.93026192,   0.79248319,   4.22366488,   0.98185211,
         -0.41845062,   1.4302237 , -10.93262069,  -2.86111513,
         -2.67133407,   9.86612651,  -7.5464425 ,  -4.03532163,
          1.38476442]])
```

In [41]:
```
lr.coef_[1]
```

Out[41]:
```
array([ -8.0856239 ,  -3.63416028,  -8.4593189 ,   4.78819433,
        -1.78636854,  -0.30252817,   6.23457801,   2.95908431,
        -1.0768496 , -22.87914253,   6.30921537,   1.23987802,
       -18.23842479])
```

In [42]:
```
10**-4
```

Out[42]:
```
0.0001
```

In [57]:
```
weights = np.array(weights)
weights
```

Out[57]:
```
array([[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
       [-7.66115274e-01, -4.06314474e-02,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        -8.23987196e-01,  6.97554422e-03,  6.46568870e-02,
        -4.15802417e-01],
       [-1.53722377e+00, -3.87032209e-01, -9.95137081e-01,
         3.64931391e-01, -5.95585137e-02,  0.00000000e+00,
         6.67956118e-01,  0.00000000e+00,  0.00000000e+00,
        -1.93393651e+00,  1.23386892e+00,  0.00000000e+00,
        -2.23181947e+00],
       [-2.69263309e+00, -1.09953732e+00, -2.84331945e+00,
         1.43730115e+00,  0.00000000e+00,  0.00000000e+00,
         1.88048254e+00,  9.69718983e-01,  0.00000000e+00,
        -6.06540983e+00,  2.38415618e+00,  0.00000000e+00,
        -5.36610108e+00],
       [-4.33653058e+00, -1.99295502e+00, -5.00694069e+00,
         2.60767527e+00, -1.48022882e-01,  0.00000000e+00,
         3.42348514e+00,  2.00245197e+00,  0.00000000e+00,
        -1.24369473e+01,  3.93535103e+00,  1.37876883e-01,
        -1.07231482e+01],
       [-6.05087649e+00, -2.84435070e+00, -6.94834589e+00,
         3.78883950e+00, -9.75913259e-01,  0.00000000e+00,
         4.62330715e+00,  2.37537588e+00, -4.72466879e-01,
        -1.77063485e+01,  5.33204520e+00,  6.96546239e-01,
        -1.47879004e+01],
       [-8.13653733e+00, -3.32986829e+00, -8.66456824e+00,
         4.72474882e+00, -1.55129989e+00, -2.15328210e-01,
         6.45505988e+00,  2.83854129e+00, -1.29033708e+00,
        -2.17321071e+01,  7.00438310e+00,  7.70742502e-01,
        -1.78885558e+01],
       [-8.08562390e+00, -3.63416028e+00, -8.45931890e+00,
         4.78819433e+00, -1.78636854e+00, -3.02528168e-01,
         6.23457801e+00,  2.95908431e+00, -1.07684960e+00,
        -2.28791425e+01,  6.30921537e+00,  1.23987802e+00,
        -1.82384248e+01]])
```

In [56]:
```
weights = np.array(weights)
weights.shape
```

Out[56]:
```
(10, 13)
```

In [45]:
```
weights.shape[1]
```
Out[45]:
```
13
```

In [46]:
```
weights.shape[0]
```
Out[46]:
```
10
```

In [48]:
```
params
```
Out[48]:
```
[0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0, 100000.0]
```

In [49]:
```
label=df_wine.columns[column + 1]
```

In [50]:
```
label
```
Out[50]:
```
'Proline'
```

In [51]:
```
label=df_wine.columns
```

In [52]:
```
label
```
Out[52]:
```
Index(['Class label', 'Alcohol', 'Malic acid', 'Ash', 'Alcalinity of ash',
       'Magnesium', 'Total phenols', 'Flavanoids', 'Nonflavanoid phenols',
       'Proanthocyanins', 'Color intensity', 'Hue',
       'OD280/OD315 of diluted wines', 'Proline'],
      dtype='object')
```

In [ ]:

In [44]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy
```

In [45]:

```
data = pd.read_excel \
("C:/Users/subbu/OneDrive/Desktop/machine     Learning/experiments/hierarchical_clustering.xlsx")
```

In [46]:

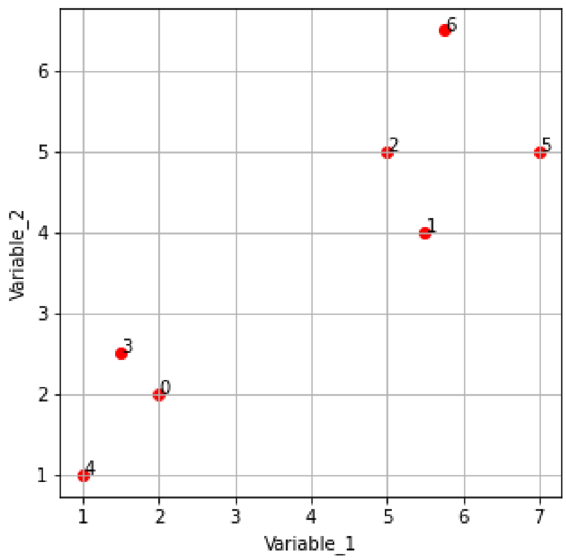
```
data
```

Out[46]:

	Variable 1	Variable 2
0	2.00	2.0
1	5.50	4.0
2	5.00	5.0
3	1.50	2.5
4	1.00	1.0
5	7.00	5.0
6	5.75	6.5

In [65]:

```
fig = plt.figure(figsize = (5,5))
x = data["Variable 1"]
y = data["Variable 2"]
n = range(0,7)
plt.grid()
plt.scatter(x, y, marker = 'o', c = 'red' )
plt.xlabel('Variable_1')
plt.ylabel('Variable_2')
for i, txt in enumerate(n):
    plt.annotate(txt, (x[i], y[i]))
```



In [4]:

```
x = data["Variable 1"]
x
```

Out[4]:

```
0    2.00
1    5.50
2    5.00
3    1.50
4    1.00
5    7.00
6    5.75
Name: Variable 1, dtype: float64
```

In [12]:

```
x[1]
```

Out[12]:

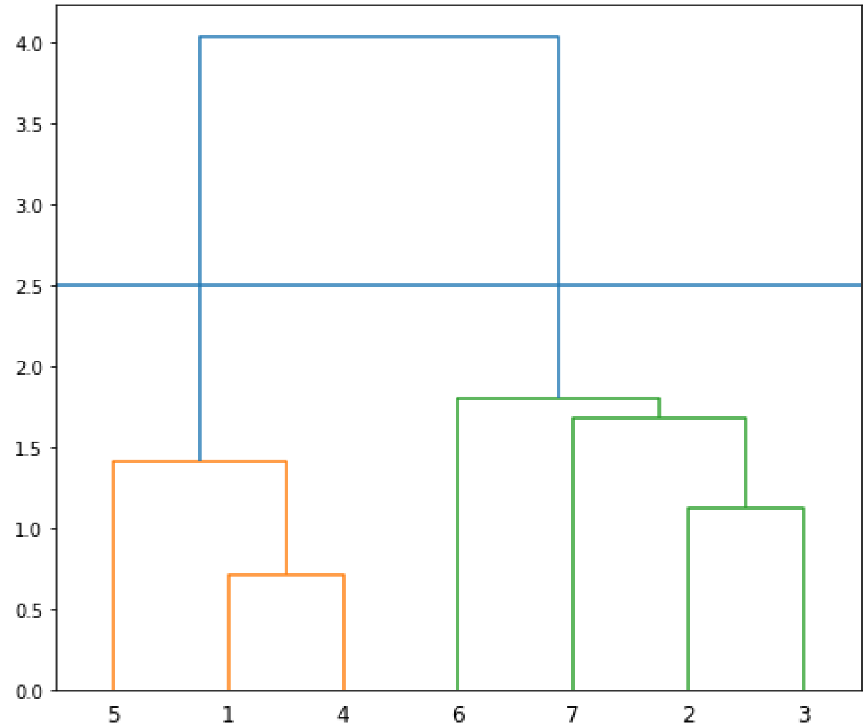
```
5.5
```

In [12]:

```
# how many clusters, draw a dendrogram
#linkage-single

from scipy.cluster.hierarchy import dendrogram, linkage
```

In [15]:

```
linked = linkage(data, 'single')
labelList =range(1,8)

plt.figure(figsize=(8,7))
dendrogram(linked, orientation = 'top',
           labels = labelList,
           distance_sort = 'descending')
plt.axhline(y = 2.5)
plt.show()
```



In [ ]:

```
# check for largest distance vertically without crossing any horizontal  line
```

In [16]:

```
import sklearn
from sklearn.cluster import AgglomerativeClustering
#k=2
#fit on my data
#Hclustering = AgglomerativeClustering(n_clusters = k, affinity = 'euclidean', linkage ='single')
Hclustering = AgglomerativeClustering(n_clusters = None, affinity = 'euclidean', linkage ='single
Hclustering.fit(data)


```

Out[16]:

```
AgglomerativeClustering(distance_threshold=3,   linkage='single',   n_clusters=None)
```

In [17]:

```
result=Hclustering.fit_predict(data)
```

In [18]:

```
result
```

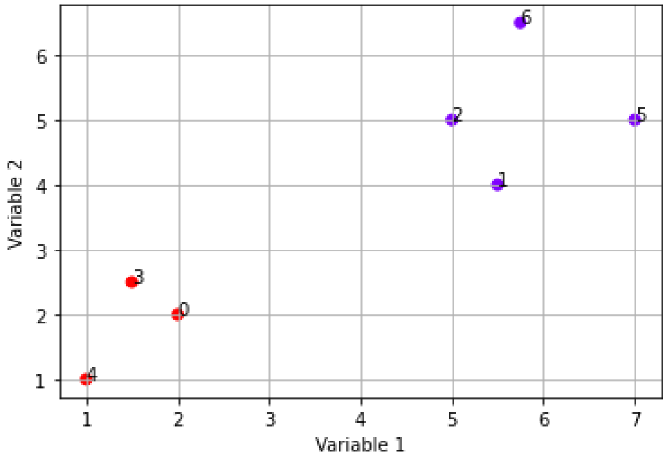
Out[18]:

```
array([1, 0, 0, 1, 1, 0, 0], dtype=int64)
```

In [40]:

```
fig = plt.figure(figsize = (5,5))
x = data["Variable 1"]
y = data["Variable 2"]
n = range(0,7)
fig, ax = plt.subplots()
ax.scatter(x, y, c = Hclustering.labels_, cmap ='rainbow')
plt.grid()
plt.xlabel('Variable 1')
plt.ylabel('Variable 2')
for i, txt in enumerate(n):
    plt.annotate(txt, (x[i], y[i]))
```

```
<Figure size 360x360 with 0 Axes>
```



Silhouette score is used to evaluate the quality of clusters created using clustering algorithms: The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar

In [111]:

```
from sklearn.metrics import silhouette_score
silhouette_score(data, result)
```

Out[111]:

```
0.7099601911940211
```

In [41]:

```
Hclustering.labels_
```

Out[41]:

```
array([1, 0, 0, 1, 1, 0, 0], dtype=int64)
```