NAME-MEENAKSHI SHARMA

ROLL NO-2020327

MACHINE LEARNING PRACTICAL FILE

1. Perform elementary mathematical operations in Octave/MATLAB/R like addition, multiplication,division and exponentiation.

In [1]:
```
n1=int(input("ENTER THE FIRST NUMBER: "))
n2=int(input("ENTER THE SECOND NUMBER: "))
option='y'
while(option=='y' or option=='Y'):
    print("SELECT ANY ONE OPTION:\n\t1. ADDITION\n\t2. SUBTRACTION\n\t3. MULTIPLICATION\n\t4. DIVISION\n\t5. EX
    choice=int(input("ENTER YOUR CHOICE:"))
    if choice==1:
        s=n1+n2
        print("Addition:",s)
    elif choice==2:
        d=n1-n2
        print("Subtraction:",d)
    elif choice==3:
        m=n1*n2
        print("Multiplication:",m)
    elif choice==4:
        div=n1/n2
        print("Division:",div)
    elif choice==5:
        e=n1**n2
        print("Exponentiation:",e)
    else:
        print("WRONG CHOICE")

    print("DO YOU WANT TO CONTINUE? ('Y/N')")
    option=input()
print("END")
```

```
ENTER THE FIRST NUMBER: 5
ENTER THE SECOND NUMBER: 2
SELECT ANY ONE OPTION:
        1. ADDITION
        2. SUBTRACTION
        3. MULTIPLICATION
        4. DIVISION
        5. EXPONENTIATION
ENTER YOUR CHOICE:1
Addition: 7
DO YOU WANT TO CONTINUE? ('Y/N')
y
SELECT ANY ONE OPTION:
        1. ADDITION
        2. SUBTRACTION
        3. MULTIPLICATION
        4. DIVISION
        5. EXPONENTIATION
ENTER YOUR CHOICE:2
Subtraction: 3
DO YOU WANT TO CONTINUE? ('Y/N')
y
SELECT ANY ONE OPTION:
        1. ADDITION
        2. SUBTRACTION
        3. MULTIPLICATION
        4. DIVISION
        5. EXPONENTIATION
ENTER YOUR CHOICE:3
Multiplication: 10
DO YOU WANT TO CONTINUE? ('Y/N')
y
SELECT ANY ONE OPTION:
        1. ADDITION
        2. SUBTRACTION
        3. MULTIPLICATION
        4. DIVISION
        5. EXPONENTIATION
ENTER YOUR CHOICE:4
Division: 2.5
DO YOU WANT TO CONTINUE? ('Y/N')
y
SELECT ANY ONE OPTION:
        1. ADDITION
        2. SUBTRACTION
        3. MULTIPLICATION
        4. DIVISION
        5. EXPONENTIATION
ENTER YOUR CHOICE:5
Exponentiation: 25
DO YOU WANT TO CONTINUE? ('Y/N')
n
END
```

2.Perform elementary logical operations in Octave/MATLAB/R (like OR, AND, Checking forEquality, NOT, XOR).

In [2]:

```
A=True
B=False
print("A AND B is: ", A and B)
print("A OR B is: ", A or B)
print("NOT A is: ", not A)
print("NOT B is: ", not B)
print("A XOR B is: ", A ^ B)
print("A == B is: ", A==B)
```

```
A AND B is:  False
A OR B is:  True
NOT A is:  False
NOT B is:  True
A XOR B is:  True
A == B is:  False
```

3. Create, initialize and display simple variables and simple strings and use simple formatting for variable

In [3]:

```
x=50
y=60
z=x+y
```

```
print(z)
s="Hello"
print(s)
print(s+" World")
```

```
110
Hello
Hello World
```

4. Create/Define single dimension / multi-dimension arrays, and arrays with specific values like array of all ones, all zeros, array with random values within a range, or a diagonal matrix.

In [4]:

```
import numpy as np

arr1=np.array([1,2,3])
print("Matrix 1: ", arr1)

arr2=np.array([[4,5,6],[1,3,7]])
print("Matrix 2:\n", arr2,'\n')

arr3=np.ones((2,3)).astype('int32')
print("Ones Matrix:\n", arr3,'\n')

arr4=np.zeros((2,2)).astype('int32')
print("Zeroes Matrix:\n", arr4,'\n')

arr5=np.random.randint(1,7,size=(3,3))
print("Random Value Matrix:\n", arr5,'\n')

arr6=np.diag([1,2,3,4])
print("Diagonal Matrix:\n",arr6)
```

```
Matrix 1:  [1 2 3]
Matrix 2:
 [[4 5 6]
 [1 3 7]]

Ones Matrix:
 [[1 1 1]
 [1 1 1]]

Zeroes Matrix:
 [[0 0]
 [0 0]]

Random Value Matrix:
 [[2 6 5]
 [4 4 3]
 [5 3 2]]

Diagonal Matrix:
 [[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```

5. Use command to compute the size of a matrix, size/length of a particular row/column, load data from a text file, store matrix data to a text file, finding out variables and their features in the current scope

In [5]:

```
import numpy as np

x=np.array([[1,2,3,4],[5,6,7,8]])
print("SHAPE: ", x.shape,'\n')
print("SIZE: ", x.size,'\n')
print("ITEM SIZE: ", x.itemsize,'\n')

file=np.random.randint(1,20,size=(3,3))
np.savetxt('data1.txt',file)
np.genfromtxt('data1.txt',delimiter=' ')
file=file.astype('int32')
print("MATRIX:\n", file)
```

```
SHAPE:  (2, 4)

SIZE:  8

ITEM SIZE:  4

MATRIX:
 [[ 2 11 10]
 [ 5 17 11]
 [13 18  6]]
```

6. Perform basic operations on matrices (like addition, subtraction, multiplication) and display specific rows or columns of the matrix.

In [6]:

```python
import numpy as np

a=np.array([[1,2,3],[4,5,6]])
b=np.array([[4,5,6],[7,8,9]])

s=a+b
print("Matrix after ADDITION:\n", s,'\n')
print("3rd Column of Sum Matrix:\n", s[:,2],'\n')

d=b-a
print("Matrix after SUBTRACTION:\n", d,'\n')
print("2nd Row of Difference Matrix:\n", d[1,:],'\n')

m=a*b
print("Matrix after MULTIPLICATION:\n", m,'\n')
print("M[1,2]: ", m[1,2],'\n')
```

```
Matrix after ADDITION:
 [[ 5  7  9]
 [11 13 15]]

3rd Column of Sum Matrix:
 [ 9 15]

Matrix after SUBTRACTION:
 [[3 3 3]
 [3 3 3]]

2nd Row of Difference Matrix:
 [3 3 3]

Matrix after MULTIPLICATION:
 [[ 4 10 18]
 [28 40 54]]

M[1,2]:  54
```

7. Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, additing/removing rows/columns from a matrix, finding the maximum or minimum values in a matrix or in a row/column, and finding the sum of some/all elements in a matrix.

In [7]:

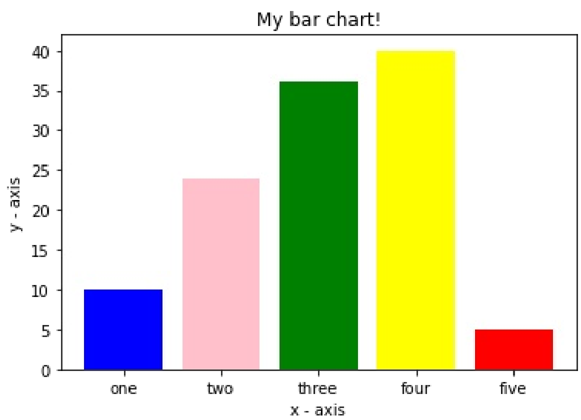
```python
import numpy as np

c=np.array([[1,2,4],[5,6,8]])
print("MATRIX:\n", c, '\n')

v=np.sin(c)
print("SINE MATRIX:\n", v,'\n')
print("ABSOLUTE MATRIX:\n", np.abs(v),'\n')

d=np.append(c,np.array([[9,10,12]]), axis=0)
print("APPENDED MATRIX:\n", d,'\n')

print("2nd ROW DELETED:\n", np.delete(d,1,0),'\n')
print("2nd COLUMN DELETED:\n", np.delete(d,1,1),'\n')

print("MAX. ELEMENT OF THE MATRIX:", np.max(d),'\n')
print("MAX. ELEMENT OF THE MATRIX:", np.min(d),'\n')
print("SUM OF ALL THE ELEMENTS OF THE MATRIX:",np.sum(d),'\n')
print("COLUMN-WISE SUM OF ELEMENTS\n", np.sum(d,axis=0))
```

```
MATRIX:
 [[1 2 4]
 [5 6 8]]

SINE MATRIX:
 [[ 0.84147098  0.90929743 -0.7568025 ]
 [-0.95892427 -0.2794155   0.98935825]]

ABSOLUTE MATRIX:
 [[0.84147098 0.90929743 0.7568025 ]
 [0.95892427 0.2794155  0.98935825]]

APPENDED MATRIX:
 [[ 1  2  4]
 [ 5  6  8]
 [ 9 10 12]]

2nd ROW DELETED:
 [[ 1  2  4]
 [ 9 10 12]]

2nd COLUMN DELETED:
 [[ 1  4]
 [ 5  8]
 [ 9 12]]

MAX. ELEMENT OF THE MATRIX: 12

MAX. ELEMENT OF THE MATRIX: 1

SUM OF ALL THE ELEMENTS OF THE MATRIX: 57

COLUMN-WISE SUM OF ELEMENTS
 [15 18 24]
```

8.Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

In [8]:

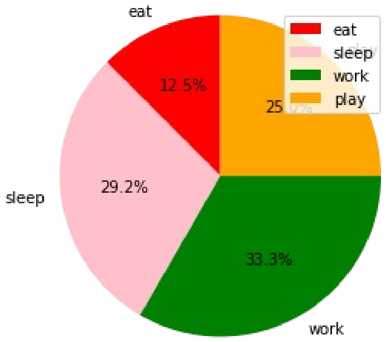
```
import matplotlib.pyplot as plt
import numpy as np

# x-coordinates of left sides of bars
left = [1, 2, 3, 4, 5]
# heights of bars
height = [10, 24, 36, 40, 5]
# labels for bars
tick_label = ['one', 'two', 'three', 'four', 'five']

# plotting a bar chart
plt.bar(left, height, tick_label = tick_label, width = 0.8, color = ['blue', 'pink','green','yellow','red'])

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')
# function to show the plot
plt.show()
```



In [9]:

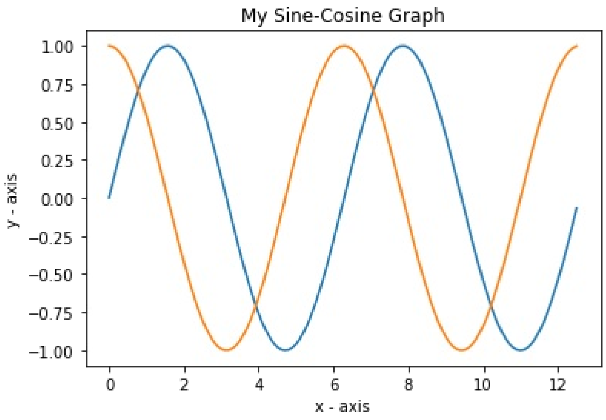
```
# defining labels
activities = ['eat', 'sleep', 'work', 'play']
# portion covered by each label
```

```
slices = [3, 7, 8, 6]
# color for each label
colors = ['red', 'pink', 'green', 'orange']

# plotting the pie chart
plt.pie(slices, labels = activities, colors=colors, startangle=90, radius = 1.2, autopct = '%1.1f%%')

# plotting legend
plt.legend()
# showing the plot
plt.show()
```



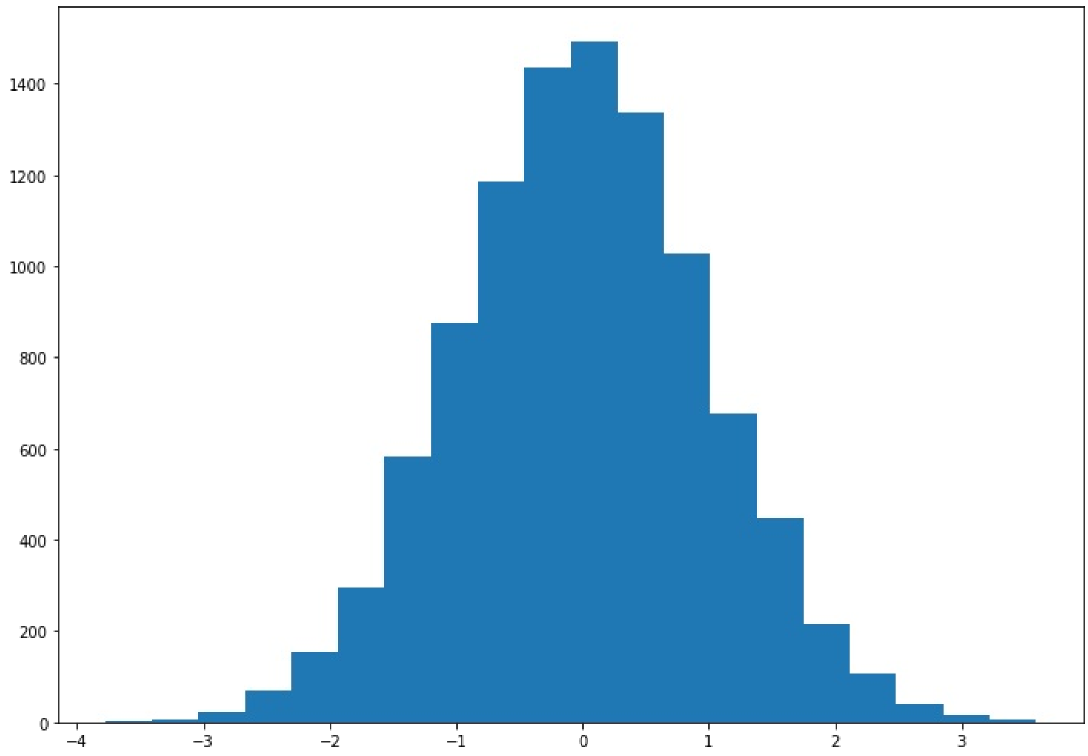
In [10]:

```
x = np.arange(0,4*np.pi,0.1)   # start,stop,step
y = np.sin(x)
z = np.cos(x)
plt.plot(x,y,x,z)
# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My Sine-Cosine Graph')
plt.show()
```



In [11]:

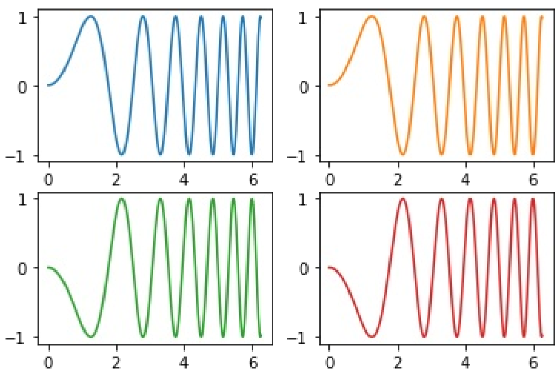
```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker import PercentFormatter

# Creating dataset
np.random.seed(23685752)
N_points = 10000
n_bins = 20

# Creating distribution
x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) + 25

# Creating histogram
fig, axs = plt.subplots(1, 1,
                                                figsize =(10, 7),
                                                tight_layout = True)

axs.hist(x, bins = n_bins)

# Show plot
plt.show()
```



9.Generate different subplots from a given plot and color plot data.

In [12]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x ** 2)

fig, axs = plt.subplots(2, 2)
axs[0, 0].plot(x, y)
axs[0, 1].plot(x, y, 'tab:orange')
axs[1, 0].plot(x, -y, 'tab:green')
axs[1, 1].plot(x, -y, 'tab:red')
```

Out[12]: [<matplotlib.lines.Line2D at 0x2975abc1d30>]



10.Use conditional statements and different type of loops based on simple example/s.

In [13]:
```
ch='y'
while(ch=='y'):
    s=input("Enter a String: ")
    s1=""
    for i in s:
        s1=i+s1
    if(s1==s):
        print(s, "is Palindrome")
    else:
        print(s, "is not Palindrome")
    ch=input("Do u want to continue(y/n)?")
```

```
Enter a String: abcdcba
abcdcba is Palindrome
Do u want to continue(y/n)?y
Enter a String: meenakshi
meenakshi is not Palindrome
Do u want to continue(y/n)?n
```

11.Perform vectorized implementation of simple matrix operation like finding the transpose of a matrix, adding, subtracting or multiplying two matrices.

In [14]:
```
import numpy as np
a=np.array([[1,2,3],[4,5,6],[1,1,1]])
b=np.array([[6,7,8],[9,10,11],[2,2,2]])
print("MATRIX 1:\n", a,'\n')
print("MATRIX 2:\n", b,'\n')
print("TRANSPOSE OF MATRIX 1:\n", np.transpose(a),'\n')
print("M1+M2:\n", a+b,'\n')
print("M1-M2:\n", b-a,'\n')
print("M1*M2:\n", np.matmul(a,b),'\n')
```

```
MATRIX 1:
 [[1 2 3]
 [4 5 6]
 [1 1 1]]

MATRIX 2:
 [[ 6  7  8]
 [ 9 10 11]
 [ 2  2  2]]

TRANSPOSE OF MATRIX 1:
 [[1 4 1]
 [2 5 1]
 [3 6 1]]

M1+M2:
 [[ 7  9 11]
 [13 15 17]
 [ 3  3  3]]

M1-M2:
 [[5 5 5]
 [5 5 5]
 [1 1 1]]

M1*M2:
 [[30 33 36]
 [81 90 99]
 [17 19 21]]
```

12.Implement Linear Regression problem. For example, based on the “Advertising” dataset comprising of budget of TV, Radio etc. and the sales data, predict the estimated sales for TV budget.

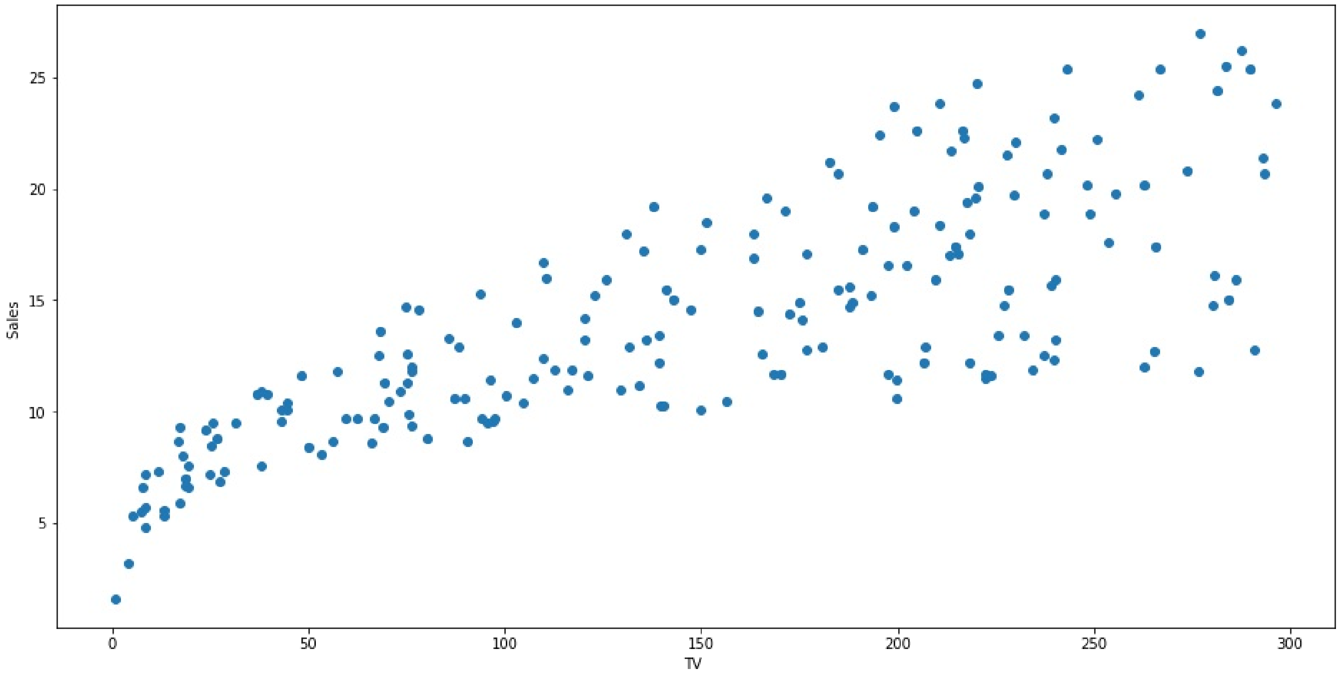
In [15]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
adv = pd.read_csv("Advertising.csv")
adv.head()
```

Out[15]:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

In [16]:
```
#check for nulls in the data
adv.isnull().sum()
```

```
plt.figure(figsize=(16, 8))
plt.scatter(adv['TV'], adv['sales'])
plt.xlabel("TV ")
plt.ylabel("Sales ")
plt.show()
```



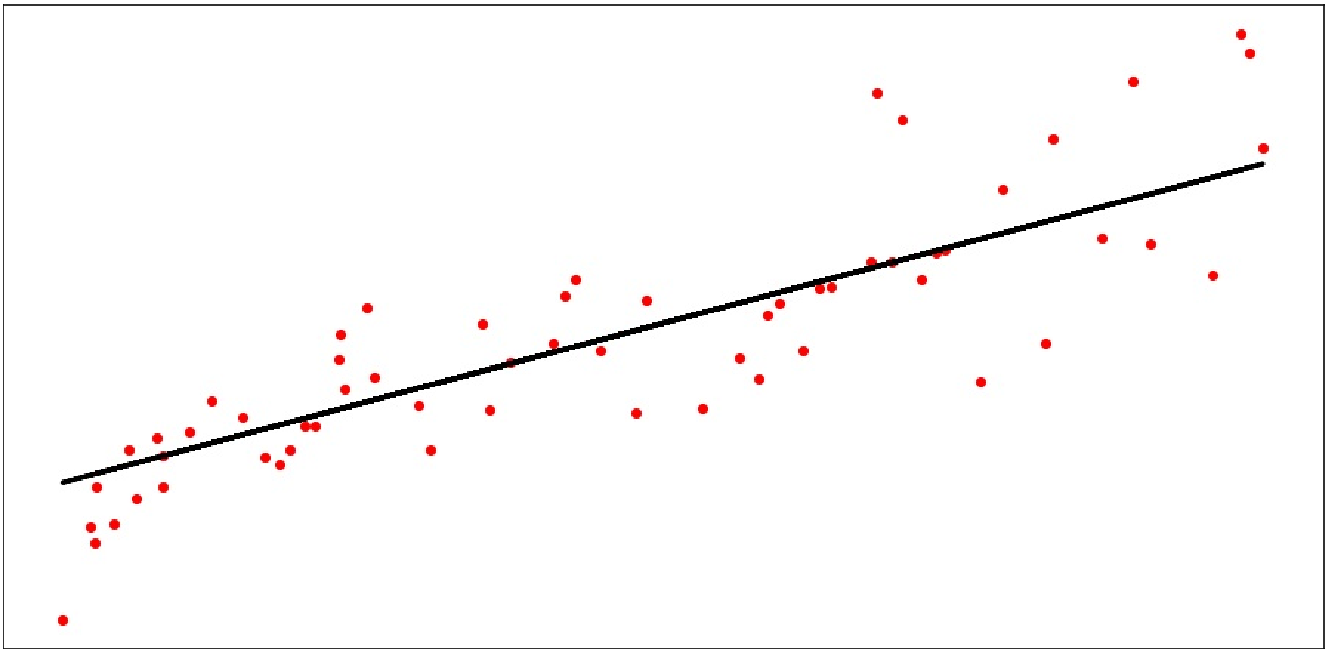
In [17]:
```
x = adv['TV'].values.reshape(-1,1)
y = adv['sales'].values.reshape(-1,1)
scaler = StandardScaler()
X_std = scaler.fit_transform(x)
# split data into train and test
x_train, x_test, y_train, y_test = train_test_split(X_std,y,test_size=0.3,random_state=0)

#fit the model using Linear Regression
linreg = LinearRegression()
linreg.fit(x_train, y_train)


print("INTERCEPT: ", linreg.intercept_[0])          #Intercept
print("\nCOEFFICIENT: ", linreg.coef_[0][0])          #Coefficient
print("\nThe linear model is: y = {:.5} + {:.5}TV".format(linreg.intercept_[0], linreg.coef_[0][0]))

# Make predictions using the testing set
y_pred = linreg.predict(x_test)
y=linreg.predict(np.array([1000]).reshape(1,-1))         #Prediction
print("\nPredicted Value for the SALES of TV: ", y)


#Accuracy Score
print("\nAccuracy Score: ", linreg.score(x_test,y_test))
print('Mean Squared Error :', metrics.mean_squared_error(y_test,y_pred))
print('Root Mean Squared Error :', np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

```
INTERCEPT:  14.047465574222732

COEFFICIENT:  3.9235096484782392

The linear model is: y = 14.047 + 3.9235TV

Predicted Value for the SALES of TV:  [[3937.55711405]]

Accuracy Score:  0.725606346597073
Mean Squared Error : 7.497479593464674
Root Mean Squared Error : 2.7381525876883988
```

In [18]:
```
print('Train Score :', linreg.score(x_train,y_train))
print('Test Score:', linreg.score(x_test,y_test))
```

```
Train Score : 0.5552336104251211
Test Score: 0.725606346597073
```

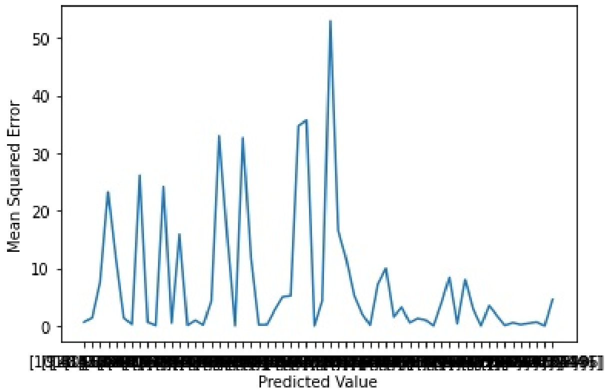
In [19]:
```
plt.figure(figsize=(16, 8))
plt.scatter(x_test, y_test, color="red")
plt.plot(x_test, y_pred, color="black", linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```



In [20]:

```
errors = list()
for i in range(len(y_test)):
    # calculate error
    err = (y_test[i] - y_pred[i])**2
    # store error
    errors.append(err)

# plot errors
plt.plot(errors)
plt.xticks(ticks=[i for i in range(len(errors))], labels=y_pred)
plt.xlabel('Predicted Value')
plt.ylabel('Mean Squared Error')
plt.show()
```

```
c:\users\91798\appdata\local\programs\python\python39\lib\site-packages\matplotlib\text.py:1223: FutureWarning:
elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
  if s != self._text:
```



13.Based on multiple features/variables perform Linear Regression on “Advertising” dataset. For example, based on the budget of TV, Radio and Newspaper, predict the overall sales.

In [21]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import StandardScaler

adv = pd.read_csv("Advertising.csv")
adv.head()
```

Out[21]:

	Unnamed: 0	TV	radio	newspaper	sales
0	1	230.1	37.8	69.2	22.1
1	2	44.5	39.3	45.1	10.4
2	3	17.2	45.9	69.3	9.3
3	4	151.5	41.3	58.5	18.5
4	5	180.8	10.8	58.4	12.9

In [22]:

```
x = adv.drop(['sales', 'Unnamed: 0'], axis=1)
y = adv['sales'].values.reshape(-1,1)
scaler = StandardScaler()
X_std = scaler.fit_transform(x)
# split data into train and test
x_train, x_test, y_train, y_test = train_test_split(X_std,y,test_size=0.3,random_state=0)

#fit the model using Linear Regression
linreg = LinearRegression()
linreg.fit(x_train, y_train)

print("INTERCEPT: ", linreg.intercept_[0])          #Intercept
print("\nCOEFFICIENT: ", linreg.coef_)          #Coefficient
print("The linear model is: Y = {:.5} + {:.5}*TV + {:.5}*radio + {:.5}*newspaper".format(linreg.intercept_[0], 

# Make predictions using the testing set
y_pred = linreg.predict(x_test)
y=linreg.predict(np.array([275,55.7,80.6]).reshape(1,-1))         #Prediction
print("\nPredicted Value for the SALES for given instance: ", y)

#Accuracy Score
print("\nAccuracy Score: ", linreg.score(x_test,y_test))
predictions = linreg.predict(x_test)
print('Mean Squared Error :', metrics.mean_squared_error(y_test,predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,predictions)))
```

```
INTERCEPT:  14.053309666438658

COEFFICIENT:  [[3.76087812 2.96607016 0.04005234]]
The linear model is: Y = 14.053 + 3.7609*TV + 2.9661*radio + 0.040052*newspaper

Predicted Value for the SALES for given instance:  [[1216.73311857]]

Accuracy Score:  0.8649018906637793
Mean Squared Error : 3.691394845698606
Root Mean Squared Error: 1.921300300759516
```

14.Implement a classification/ logistic regression problem. For example, based on different features of “diabetes” data, classify, whether a woman is diabetic or not.

In [23]:

```
# import libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

from sklearn.preprocessing import StandardScaler

# load dataset
data = pd.read_csv("diabetes.csv")
data.head()
```

Out[23]:

	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

In [24]:

```
# split dataset into features and target variable
X = data.drop(columns=["Outcome"])
y = data["Outcome"]

# split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=16)
```

```
# standardize the values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# instantiate the logistic regression model
lr = LogisticRegression()

# fit the model to the training data
lr.fit(X_train, y_train)

# make predictions on the test set
y_pred = lr.predict(X_test)

# calculate accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

print("Coefficients:", lr.coef_)
# print the accuracy
print("Accuracy:", accuracy)
```

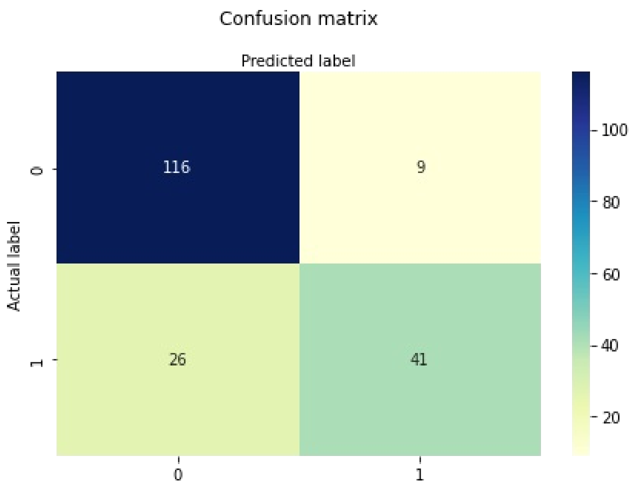
```
Coefficients: [[ 0.29873476  1.04748909 -0.26589863  0.05589826 -0.17356267  0.70132615
   0.28256485  0.15350615]]
Accuracy: 0.8177083333333334
```

In [25]:
```
# import the metrics class
from sklearn import metrics

cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[25]:
```
array([[116,   9],
       [ 26,  41]], dtype=int64)
```

In [26]:
```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[26]:
```
Text(0.5, 257.44, 'Predicted label')
```



In [27]:
```
from sklearn.metrics import classification_report
target_names = ['without diabetes', 'with diabetes']
print(classification_report(y_test, y_pred, target_names=target_names))
```

```
                  precision    recall  f1-score   support

without diabetes       0.82      0.93      0.87       125
   with diabetes       0.82      0.61      0.70        67

        accuracy                           0.82       192
       macro avg       0.82      0.77      0.78       192
    weighted avg       0.82      0.82      0.81       192
```

15.Use some function for regularization of “BOSTON” dataset available in ‘sklearn library’.

Lasso on some values

In [28]:

```
import numpy as np
from sklearn.linear_model import Lasso

# Creating a toy dataset
X = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
y = np.array([10, 11, 12])

# Creating Lasso model with alpha=0.1
lasso = Lasso(alpha=0.1)

# Fitting the model on the dataset
lasso.fit(X, y)

# Printing the coefficients and intercept
print("Coefficients:", lasso.coef_)
print("Intercept:", lasso.intercept_)
```

```
Coefficients: [3.16666667e-01 2.46716228e-17 0.00000000e+00]
Intercept: 9.733333333333333
```

On Dataset

In [29]:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston
boston = load_boston()
```

```
c:\users\91798\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: Future
Warning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np

        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.
  warnings.warn(msg, category=FutureWarning)
```

In [30]:

```python
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target, test_size=0.2, random_state=42)

# standardize the values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Creating a Lasso model with alpha=0.1
lasso = Lasso(alpha=0.1)

# Fitting the model on the training set
lasso.fit(X_train, y_train)

# Predicting on the testing set
y_pred = lasso.predict(X_test)

# Calculating the mean squared error of the predictions
mse = mean_squared_error(y_test, y_pred)

# Printing the mean squared error and the coefficients of the Lasso model
print("Mean Squared Error:", mse)
print("Coefficients:", lasso.coef_)
```

```
Mean Squared Error: 25.65673936716768
Coefficients: [-0.71836455  0.25962714 -0.          0.69822096 -1.56814243  3.27150693
 -0.         -2.28444944  0.67193802 -0.3566537  -1.89333519  1.03136581
 -3.60941047]
```

RIDGE ON SOME VALUES

In [31]:

```python
import numpy as np
from sklearn.linear_model import Ridge

# Creating a toy dataset
X = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
y = np.array([10, 11, 12])

# Creating Ridge model with alpha=0.1
ridge = Ridge(alpha=0.1)

# Fitting the model on the dataset
ridge.fit(X, y)

# Printing the coefficients and intercept
print("Coefficients:", ridge.coef_)
```

```
print("Intercept:", ridge.intercept_)
```

```
Coefficients: [0.11090573 0.11090573 0.11090573]
Intercept: 9.33641404805915
```

RIDGE ON DATASET

In [32]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston
boston = load_boston()
```

```
c:\users\91798\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: Future
Warning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np

        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.
  warnings.warn(msg, category=FutureWarning)
```

In [33]:
```
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(boston.data, boston.target, test_size=0.2, random_state=42)

# standardize the values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Creating a Ridge model with alpha=0.1
ridge = Ridge(alpha=0.1)

# Fitting the model on the training set
ridge.fit(X_train, y_train)

# Predicting on the testing set
y_pred = ridge.predict(X_test)

# Calculating the mean squared error of the predictions
mse = mean_squared_error(y_test, y_pred)

# Printing the mean squared error and the coefficients of the Ridge model
print("Mean Squared Error:", mse)
print("Coefficients:", ridge.coef_)
```

```
Mean Squared Error: 24.293294309665924
Coefficients: [-1.00111591  0.69436316  0.27539404  0.71912548 -2.01912122  3.14590087
 -0.17617627 -3.07816919  2.24333232 -1.75959591 -2.03674427  1.12933027
 -3.61037565]
```

16. Use some function for neural networks, like Stochastic Gradient Descent or

backpropagation - algorithm to predict the value of a variable based on the dataset of problem 14.

In [35]:
```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [36]:
```
bostan = load_boston()
```

```
c:\users\91798\appdata\local\programs\python\python39\lib\site-packages\sklearn\utils\deprecation.py:87: Future
Warning: Function load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np

        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.
  warnings.warn(msg, category=FutureWarning)
```

In [37]:
```
# Data shape
bostan.data.shape
```

Out[37]: (506, 13)

In [38]:
```
# Feature name
bostan.feature_names
```

Out[38]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [39]:
```
# This is y value i.e. target
bostan.target.shape
```

Out[39]: (506,)

In [40]:
```
# Convert it into pandas dataframe
data = pd.DataFrame(bostan.data, columns = bostan.feature_names)
data.head()
```

Out[40]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [41]:
```
# Statistical summary
data.describe()
```

Out[41]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000

In [42]:

```
#noramlization for fast convergence to minima
data = (data - data.mean())/data.std()
data.head()
```

Out[42]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	-0.419367	0.284548	-1.286636	-0.272329	-0.144075	0.413263	-0.119895	0.140075	-0.981871	-0.665949	-1.457558	0.440616	-1.074499
1	-0.416927	-0.487240	-0.592794	-0.272329	-0.739530	0.194082	0.366803	0.556609	-0.867024	-0.986353	-0.302794	0.440616	-0.491953
2	-0.416929	-0.487240	-0.592794	-0.272329	-0.739530	1.281446	-0.265549	0.556609	-0.867024	-0.986353	-0.302794	0.396035	-1.207532
3	-0.416338	-0.487240	-1.305586	-0.272329	-0.834458	1.015298	-0.809088	1.076671	-0.752178	-1.105022	0.112920	0.415751	-1.360171
4	-0.412074	-0.487240	-1.305586	-0.272329	-0.834458	1.227362	-0.510674	1.076671	-0.752178	-1.105022	0.112920	0.440616	-1.025487

In [43]:

```
data.mean()
```

Out[43]:

```
CRIM        9.983231e-17
ZN         -2.248970e-16
INDUS      -3.019488e-15
CHAS       -3.940634e-16
NOX         3.009012e-15
RM         -1.151736e-14
AGE        -1.145987e-15
DIS         7.073832e-16
RAD         1.664018e-15
TAX         3.918692e-16
PTRATIO    -9.475951e-15
B           8.115270e-15
LSTAT      -6.494585e-16
dtype: float64
```

In [44]:

```
# from sklearn.preprocessing import StandardScaler
# std = StandardScaler()
# data = std.fit_transform(data)
# data
# MEDV(median value is usually target), change it to price
data["PRICE"] = bostan.target
data.head()
```

Out[44]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	F
0	-0.419367	0.284548	-1.286636	-0.272329	-0.144075	0.413263	-0.119895	0.140075	-0.981871	-0.665949	-1.457558	0.440616	-1.074499	
1	-0.416927	-0.487240	-0.592794	-0.272329	-0.739530	0.194082	0.366803	0.556609	-0.867024	-0.986353	-0.302794	0.440616	-0.491953	
2	-0.416929	-0.487240	-0.592794	-0.272329	-0.739530	1.281446	-0.265549	0.556609	-0.867024	-0.986353	-0.302794	0.396035	-1.207532	
3	-0.416338	-0.487240	-1.305586	-0.272329	-0.834458	1.015298	-0.809088	1.076671	-0.752178	-1.105022	0.112920	0.415751	-1.360171	
4	-0.412074	-0.487240	-1.305586	-0.272329	-0.834458	1.227362	-0.510674	1.076671	-0.752178	-1.105022	0.112920	0.440616	-1.025487	

In [52]:

```
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3)


print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(354, 13) (152, 13) (354,) (152,)
```

In [54]:

```
x_train["PRICE"]=y_train
```

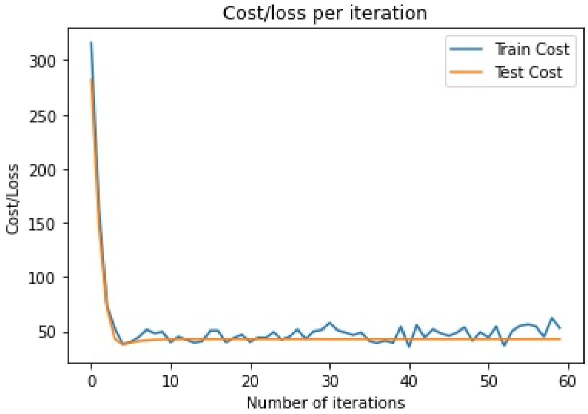
In [55]:

```
#x_train["PRICE"] = y_train
#x_test["PRICE"] = y_test
def cost_function(b, m, features, target):
    totalError = 0
```

```python
    for i in range(0, len(features)):
        x = features
        y = target
        totalError += (y[:,i] - (np.dot(x[i] , m) + b)) ** 2
    return totalError / len(x)


# In[31]:


# The total sum of squares (proportional to the variance of the data)i.e. ss_tot
# The sum of squares of residuals, also called the residual sum of squares i.e. ss_res
# the coefficient of determination i.e. r^2(r squared)
def r_sq_score(b, m, features, target):
    for i in range(0, len(features)):
        x = features
        y = target
        mean_y = np.mean(y)
        ss_tot = sum((y[:,i] - mean_y) ** 2)
        ss_res = sum(((y[:,i]) - (np.dot(x[i], m) + b)) ** 2)
        r2 = 1 - (ss_res / ss_tot)
    return r2
def gradient_decent(w0, b0, train_data, x_test, y_test, learning_rate):
    n_iter = 500
    partial_deriv_m = 0
    partial_deriv_b = 0
    cost_train = []
    cost_test = []
    for j in range(1, n_iter):

        # Train sample
        train_sample = train_data.sample(160)
        y = np.asmatrix(train_sample["PRICE"])
        x = np.asmatrix(train_sample.drop("PRICE", axis = 1))
        # Test sample
        #x_test["PRICE"] = [y_test]
        #test_data = x_test
        #test_sample = test_data.sample()
        #y_test = np.asmatrix(test_sample["PRICE"])
        #x_test = np.asmatrix(test_sample.drop("PRICE", axis = 1))

        for i in range(len(x)):
            partial_deriv_m += np.dot(-2*x[i].T , (y[:,i] - np.dot(x[i] , w0) + b0))
            partial_deriv_b += -2*(y[:,i] - (np.dot(x[i] , w0) + b0))

        w1 = w0 - learning_rate * partial_deriv_m
        b1 = b0 - learning_rate * partial_deriv_b

        if (w0==w1).all():
            #print("W0 are\n", w0)
            #print("\nW1 are\n", w1)
            #print("\n X are\n", x)
            #print("\n y are\n", y)
            break
        else:
            w0 = w1
            b0 = b1
            learning_rate = learning_rate/2


        error_train = cost_function(b0, w0, x, y)
        cost_train.append(error_train)
        error_test = cost_function(b0, w0, np.asmatrix(x_test), np.asmatrix(y_test))
        cost_test.append(error_test)

        #print("After {0} iteration error = {1}".format(j, error_train))
        #print("After {0} iteration error = {1}".format(j, error_test))

    return w0, b0, cost_train, cost_test
# Run our model
learning_rate = 0.001
w0_random = np.random.rand(13)
w0 = np.asmatrix(w0_random).T
b0 = np.random.rand()

optimal_w, optimal_b, cost_train, cost_test = gradient_decent(w0, b0, x_train, x_test, y_test, learning_rate)
print("Coefficient: {} \n y_intercept: {}".format(optimal_w, optimal_b))

'''
error = cost_function(optimal_b, optimal_w, np.asmatrix(x_test), np.asmatrix(y_test))
print("Mean squared error:",error)
'''
```

```
plt.figure()
plt.plot(range(len(cost_train)), np.reshape(cost_train,[len(cost_train), 1]), label = "Train Cost")
plt.plot(range(len(cost_test)), np.reshape(cost_test, [len(cost_test), 1]), label = "Test Cost")
plt.title("Cost/loss per iteration")
plt.xlabel("Number of iterations")
plt.ylabel("Cost/Loss")
plt.legend()
plt.show()

#error = cost_function(optimal_b, optimal_w, np.asmatrix(x_test), np.asmatrix(y_test))
#print("Mean squared error: %.2f" % error)
```

```
Coefficient: [[-1.34301263]
 [ 0.91287455]
 [-0.55469427]
 [ 3.14128584]
 [ 0.14691328]
 [ 5.37824045]
 [ 1.6204649 ]
 [-1.99062461]
 [ 1.10758615]
 [ 0.65315766]
 [-1.29669408]
 [ 1.0543267 ]
 [-2.18462248]]
 y_intercept: [[21.50663228]]
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js