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

Code:

```
val1=2, val2=3

#using addition operator res1=val1+val2 print("First
        value-",val1,"\nSecond value-",val2)
        print("Addition-",res1)
#using multiplication operator
        res2=val1*val2
        print("Multiplication-",res2)
#using division operator
        res3=val1/val2 print("Division-
        ",res3)
#using exponential operator
        res4=val1 ** val2
        print("Exponent-",res4
```

Output:

```
====================== RESTART: C:/Users/Dell/Desktop/Q1.py
First value- 2
Second value- 3
Addition- 5
Multiplication- 6
Division- 0.6666666666666666
Exponent- 8
>>>
```

Ques 2: Perform elementary logical operations in Octave/MATLAB (like OR, AND, Checking for Equality, NOT, XOR).

Code:

```
#Logical and operator
        a=10
```

```
        b=5 c=-4 if a>0
        and b>0:
        print("The numbers are greater than 0")
        if a>0 and b>0 and c>0:
        print("The numbers are greater than 0")
        else:
        print("Atleast one number is not greater than 0")
#Logical or operator
        a=10 b=-10
        c=0 if a>0 or
        b>0:
        print("Either of the number is greater than 0")
        else:
        print("No number is greater than 0")
        if b>0 or c>0:
        print("Either of the number is greater than 0") else:
        print("No number is greater than 0")#Logical not
        operator a=10 if not a:
        print("Boolean value of a is true")
        if not(a%3==0 or a%6==0):
        print("10 is not divisible by either 3 and 6")
        else:
        print("10 is divisible by either 3 or 6")
#Logical xor operator
        a=6 b=3
        c=a^b
        print("XOR of a=6, b=3 is",c)
#Logical equality operator
        a=5 b=3 c=5 if
        a==b:
        print("Equal")
        else:
        print("Not Equal")
        if a==c:
        print("Equal")
        else: print("Not
        Equal”)
```

Output:

```
The numbers are greater than 0
Atleast one number is not greater than 0
Either of the number is greater than 0
No number is greater than 0
10 is not divisible by either 3 and 6
XOR of a=6, b=3 is 5
Not Equal
Equal
>>>
```

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

Code & Output:

```
s='apples'
print('I like to have {0} every day.'.format(s))
```

```
I like to have apples every day.
```

```
In [12]:
```

```
s='{0} is of type integer, {1} is of type float,{2} is of type string'
print(f'{6} is of type integer, {1.9} is of type float,{"hello"} is of type string')
```

```
6 is of type integer, 1.9 is of type float,hello is of type string
```

Ques 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.

Code & Output:

```
import numpy as py
```

In [23]:

```
arr=py.array([1,2,3,4,5,6])
print('dimension of array is',arr.ndim)
arr
```

```
dimension of array is 1
```

Out[23]:

```
array([1, 2, 3, 4, 5, 6])
```

In [26]:

```
arr1=arr.reshape((2,3))
arr1
```

Out[26]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```


```
import numpy as py
```

In [23]:

```
arr=py.array([1,2,3,4,5,6])
print('dimension of array is',arr.ndim)
arr
```

```
dimension of array is 1
```

Out[23]:

```
array([1, 2, 3, 4, 5, 6])
```

In [26]:

```
arr1=arr.reshape((2,3))
arr1
```

Out[26]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
print('dimension of array is',arr1.ndim)
```

```
dimension of array is 2
```

In [30]:

```
arr2=py.zeros((2,3))
arr2
```

Out[30]:

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

```
arr3=py.ones((3,3))
arr3
```

Out[32]:

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

In [34]:

```
arr2=py.zeros_like(arr3)
arr2
```

Out[34]:

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

```
py.eye(4)
```

Out[35]:

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

In [46]:

```
t=py.random.rand(12)
```

In [47]:

```
py.array(t).reshape(6,2)
```

Out[47]:

```
array([[0.09181146, 0.21945836],
       [0.87899698, 0.07955509],
       [0.81871293, 0.2883827 ],
       [0.78245522, 0.67570252],
       [0.18602969, 0.7805035 ],
       [0.86961489, 0.70818977]])
```

Ques 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.

Code & Output:

```
import numpy as py
```

```
In [2]:
```

```
mat=py.array([1,2,3,4,5,6]).reshape(2,3)
```

```
In [3]:
```

```
mat
```

```
Out[3]:
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
In [4]:
```

```
mat.size
```

```
Out[4]:
```

```
6
```

```
In [5]:
```

```
mat[0].size
```

```
Out[5]:
```

```
3
```

```
import pandas as pd
```

In [7]:

```
df=pd.read_csv('pro-5-text.txt',sep=" ",header=None)
```

In [8]:

```
df
```

Out[8]:

	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i

In [9]:

```
df.to_csv('text_file.txt')
```

```
f=open('text_file.txt')
```

In [15]:

```
print(f.read())
```

```
,0,1,2
0,a,b,c
1,d,e,f
2,g,h,i
```

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

Code & Output:

```
print('addition of two arrays \n',arr1+arr)
```

```
addition of two arrays
 [[12 14]
 [16 18]
 [20 22]
 [24 26]]
```

```
In [14]:
```

```
print('substraction of two arrays \n',arr1-arr)
```

```
substraction of two arrays
 [[10 10]
 [10 10]
 [10 10]
 [10 10]]
```

```
print('multiplication of two arrays \n',py.dot(arr1,arr))
```

```
multiplication of two arrays
 [[ 71  94 117 140]
 [ 83 110 137 164]
 [ 95 126 157 188]
 [107 142 177 212]]
```

Ques 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.

Code & Output:

```
print('Additing column')
py.append(arr,[['a'],['b'],['c'],['d']],axis=1)
```

```
Additing column
```

Out[21]:

```
array([['1', '2', 'a'],
       ['-3', '4', 'b'],
       ['-5', '6', 'c'],
       ['7', '8', 'd']], dtype='<U11')
```

In [23]:

```
print('Additing row')
py.insert(arr,2,[[88,88]],axis=0)
```

```
Additing row
```

Out[23]:

```
array([[ 1,  2],
       [-3,  4],
       [88, 88],
       [-5,  6],
       [ 7,  8]])
```

```
print('maximum of matrix ',py.max(arr))
```

```
maximum of matrix  8
```

In [30]:

```
print('minimum of matrix ',py.min(arr))
```

```
minimum of matrix  -5
```

In [31]:

```
print('sum of matrix ',py.sum(arr))
```

```
sum of matrix  20
```

In [35]:

```
print('sum of first row of matrix ',py.sum(arr[0]))
```

```
sum of first row of matrix  3
```

```
print('Absolute values of matrix')
py.absolute(arr)
```

```
Absolute values of matrix

Out[5]:

array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

```
In [6]:

print('Negative of matrix')
py.negative(arr)
```

```
Negative of matrix

Out[6]:

array([[-1, -2],
       [ 3, -4],
       [ 5, -6],
       [-7, -8]])
```

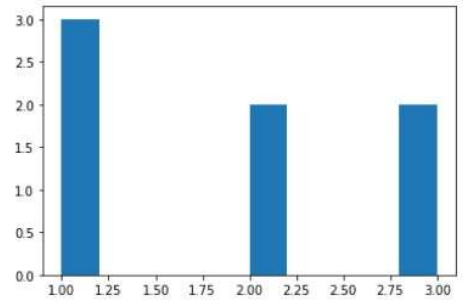
Ques 8: Create various type of plots/charts like histograms, plot based on sine/cosine function based on data from a matrix. Further label different axes in a plot and data in a plot.

Code & Output:

Histogram

```
import matplotlib.pyplot as plt
import numpy as py
import math
```

```
In [4]:

plt.hist([1,1,1,2,2,3,3])
plt.show()
```



```
arr=py.array([1,2,3,4,5,6]).reshape(3,2)
```

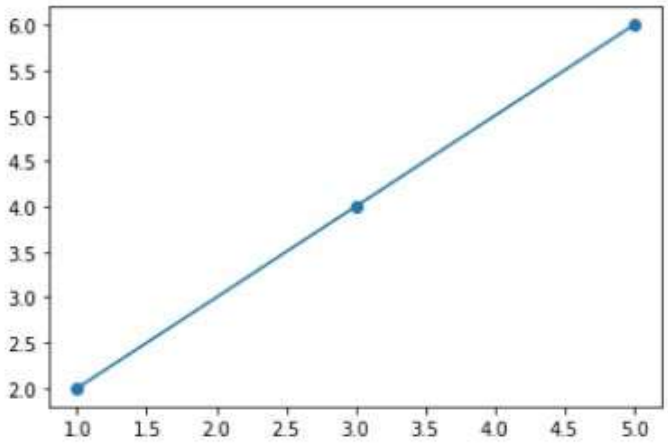
In [6]:

```
arr
```

Out[6]:

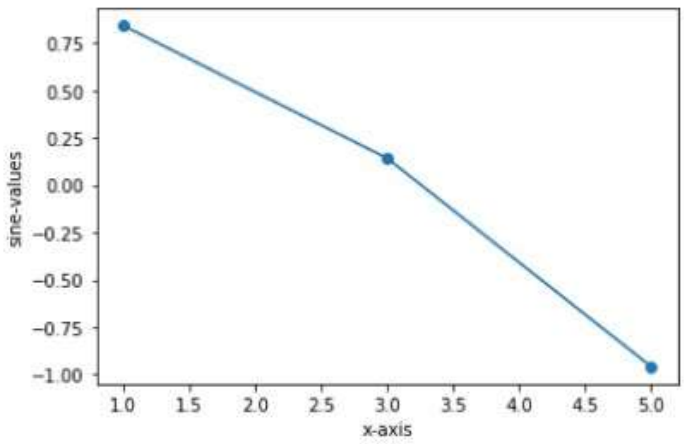
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
plt.plot(arr[:,0],arr[:,1],marker='o')
plt.show()
```




```
sin_value=[math.sin(i) for i in arr[:,0]]
```

```
plt.plot(arr[:,0],sin_value,marker='o')
plt.xlabel('x-axis')
plt.ylabel('sine-values')
plt.show()
```



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

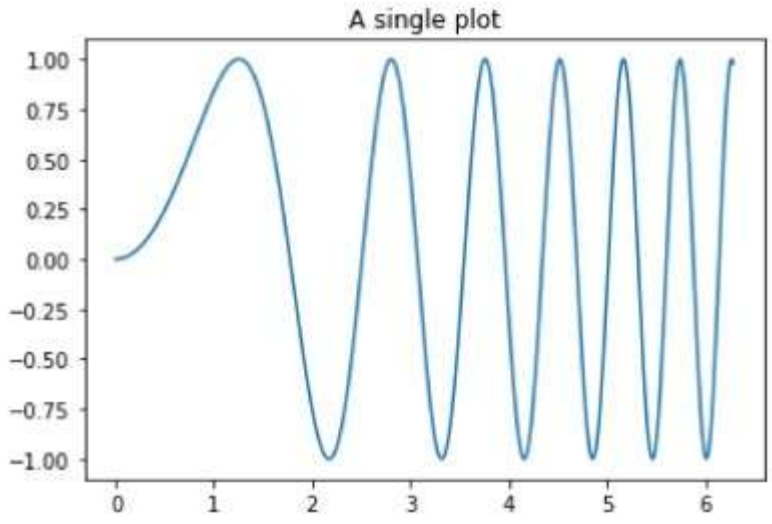
Code & Output:

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


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

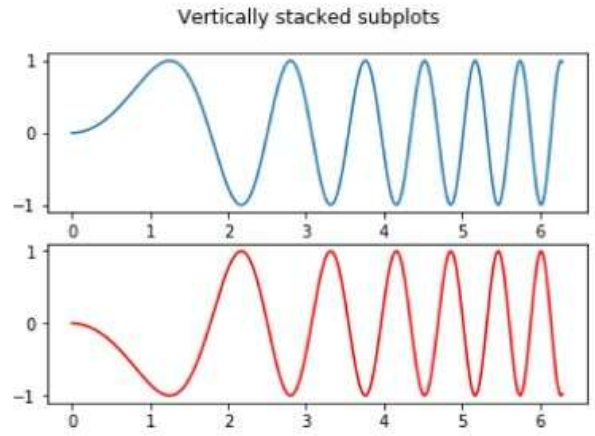
```
fig, ax = plt.subplots()
ax.plot(x, y)
ax.set_title('A single plot')
```

Out[5]:

```
Text(0.5, 1.0, 'A single plot')
```



```
fig, axs = plt.subplots(2)
fig.suptitle('Vertically stacked subplots')
axs[0].plot(x, y)
axs[1].plot(x, -y,'red')
plt.show()
```



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

Code & Output

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
```

```
apple
banana
```

In [3]:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

```
1
2
3
```

Ques 12: Implement Linear Regression problem. For example, based on a dataset comprising of existing set of prices and area/size of the houses, predict the estimated price of a given house.

Code & Output

In [57]:

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

In [56]:

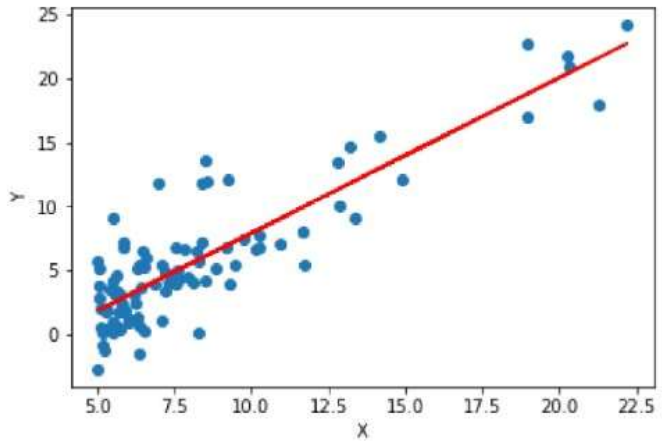
```
df= pd.read_csv('ex1data1 - ex1data1.csv')
df.columns=['X','Y']
df
```

Out[56]:

	X	Y
0	5.5277	9.13020
1	8.5186	13.66200
2	7.0032	11.85400
3	5.8598	6.82330
4	8.3829	11.88600

```
def leastSquareRegression(indep_val,dep_val,predict_indep_val):
    mean_dep_val=dep_val.mean()
    mean_indep_val=indep_val.mean()
    diff_dep_val=[]
    diff_indep_val=[]
    numerator=0.0
    denominator=0.0

    diff_dep_val=dep_val.map(lambda i:i-mean_dep_val)
    diff_indep_val=indep_val.map(lambda i:i-mean_indep_val)

    for i,j in zip(diff_indep_val,diff_dep_val):
        numerator = numerator+(i*j)

    for i in diff_indep_val:
        denominator=denominator+i**2

    beta1=numerator/denominator

    beta0=mean_dep_val - (beta1*mean_indep_val)

    predict_dep_val=[]
    for i in indep_val:
        predict_dep_val.append(beta0 + (beta1*i))
```

```
result=leastSquareRegression(df.X,df.Y, 15.67)
print(f'predicted value for x = {15.67} is: ',result['predict_val'])
```




```
predicted value for x = 15.67 is:  14.804781463319879
```

Ques 13: Based on multiple features/variables perform Linear Regression. For example, based on a number of additional features like number of bedrooms, servant room, number of balconies, number of houses of years a house has been built – predict the price of a house.

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

In [2]:

```
df= pd.read_csv('ex1data1 - ex1data1.csv')
df.columns=['X','Y']
df
```

Out[2]:

	X	Y
0	5.5277	9.13020
1	8.5186	13.66200
2	7.0032	11.85400
3	5.8598	6.82330
4	8.3829	11.88600

```
X = np.array(df['X']).reshape(-1, 1)
Y = np.array(df['Y']).reshape(-1, 1)
```

In [65]:

```
x_train, x_test, y_train, y_test = train_test_split(X,Y, test_size = 0.25)
```

In [66]:

```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[66]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=F
alse)
```

```
Y_pred=lr.predict(x_test)
```

In [68]:

```
print('mean square error: ',mean_squared_error(y_test, Y_pred))
print(r2_score(y_test, Y_pred))
```

```
mean square error:  4.431495395674081
0.8404885818633169
```

In [69]:

```
lr.score(x_test,y_test)
```

Out[69]:

```
0.8404885818633169
```

In [70]:

```
print('slope: ',lr.coef_)
print('Intercept: ',lr.intercept_)
```

```
slope:  [[1.22152923]]
Intercept:  [-4.27942216]
```

```
y=lr.predict([[15.67]])
```

```
In [72]:
```

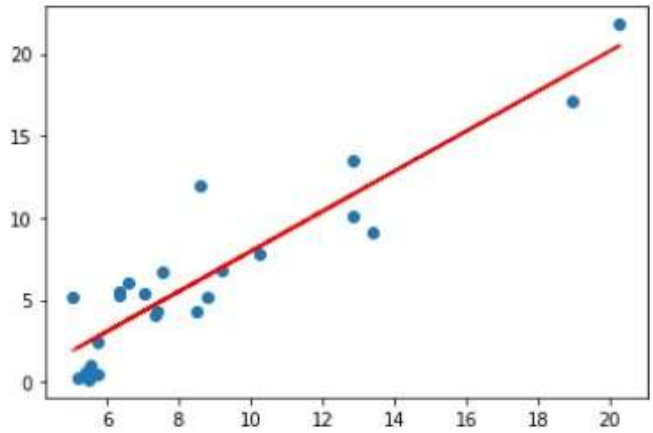
```
y[0][0]
```

```
Out[72]:
```

```
14.861940946088342
```

```
In [73]:
```

```
plt.scatter(x_test,y_test)
plt.plot(x_test,Y_pred,'red')
plt.show()
```



Ques 14: Implement a classification/ logistic regression problem. For example based on different features of students data, classify, whether a student is suitable for a particular activity. Based on the available dataset, a student can also implement another classification problem like checking whether an email is spam or not.

Code & Output:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
df= pd.read_csv('diabetes.csv')
```

In [3]:

```
df
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	

```
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age','Glucose','BloodPressure','Diabe
tesPedigreeFunction']
X = df[feature_cols]
y = df.Outcome
```

In [5]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
```

```
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.
py:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [7]:

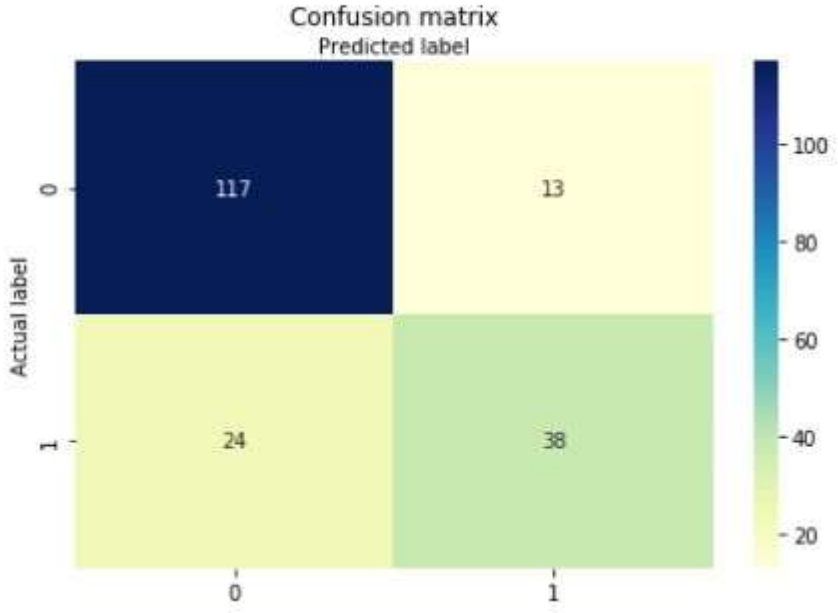
```
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

Out[7]:

```
array([[117,  13],
       [ 24,  38]], dtype=int64)
```

```
class_names=[0,1]
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')
```

```
Text(0.5, 257.44, 'Predicted label')
```




```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Accuracy: 0.8072916666666666
Precision: 0.7450980392156863
Recall: 0.6129032258064516
```

Ques 15: Use some function for regularization of dataset based on problem 14.

Code & Output:

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

In [3]:

```
from sklearn import datasets
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

In [4]:

```
boston_dataset= datasets.load_boston()
```

```
boston_dataset
```

```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.96
90e+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]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 1
8.9, 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,
```

```
boston_pd=pd.DataFrame(boston_dataset.data)
boston_pd_target=py.asarray(boston_dataset.target)
```

In [18]:

```
boston_pd['House Price']=pd.Series(boston_pd_target)
```

In [25]:

```
boston_dataset.feature_names
```

Out[25]:

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

```
X=boston_pd.iloc[:,:-1]
Y=boston_pd.iloc[:,-1]
print(boston_pd)
```

```
           0     1      2    3      4      5     6       7    8      9
10  \
0    0.00632  18.0   2.31  0.0  0.538  6.575  65.2  4.0900  1.0  296.0  1
5.3
1    0.02731   0.0   7.07  0.0  0.469  6.421  78.9  4.9671  2.0  242.0  1
7.8
2    0.02729   0.0   7.07  0.0  0.469  7.185  61.1  4.9671  2.0  242.0  1
7.8
3    0.03237   0.0   2.18  0.0  0.458  6.998  45.8  6.0622  3.0  222.0  1
8.7
4    0.06905   0.0   2.18  0.0  0.458  7.147  54.2  6.0622  3.0  222.0  1
8.7
..       ...   ...    ...  ...    ...    ...   ...     ...  ...    ...
...
501  0.06263   0.0  11.93  0.0  0.573  6.593  69.1  2.4786  1.0  273.0  2
1.0
502  0.04527   0.0  11.93  0.0  0.573  6.120  76.7  2.2875  1.0  273.0  2
1.0
503  0.06076   0.0  11.93  0.0  0.573  6.976  91.0  2.1675  1.0  273.0  2
1.0
```

```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=0)
```

In [21]:

```
lreg=LinearRegression()
lreg.fit(X_train,y_train)
```

Out[21]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=F
alse)
```

```
y_pred=lreg.predict(X_test)
```

In [23]:

```
lreg_coff=pd.DataFrame()
lreg_coff['Columns']=X_train.columns
lreg_coff['Cofficient']=pd.Series(lreg.coef_)
```

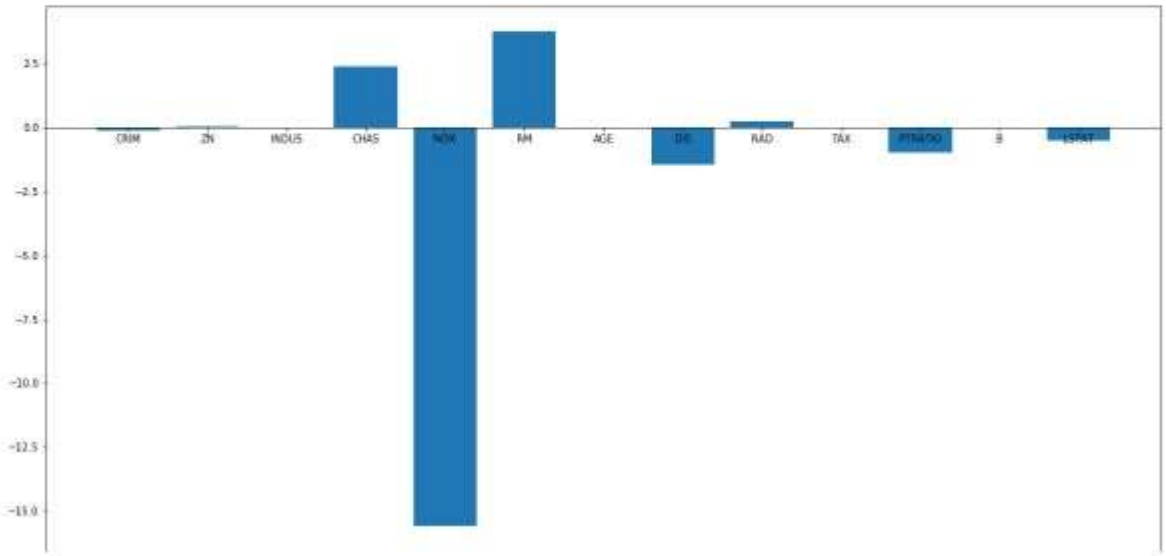
In [24]:

```
lreg_coff
```

Out[24]:

	Columns	Cofficient
0	0	-0.117735
1	1	0.044017
2	2	-0.005768
3	3	2.393416
4	4	-15.589421
5	5	3.768968
6	6	-0.007035
7	7	-1.434956

```
fig,ax=plt.subplots(figsize=(20,10))
ax.bar(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'],lreg_coff['Cofficient'])
ax.spines['bottom'].set_position('zero')
plt.show()
```

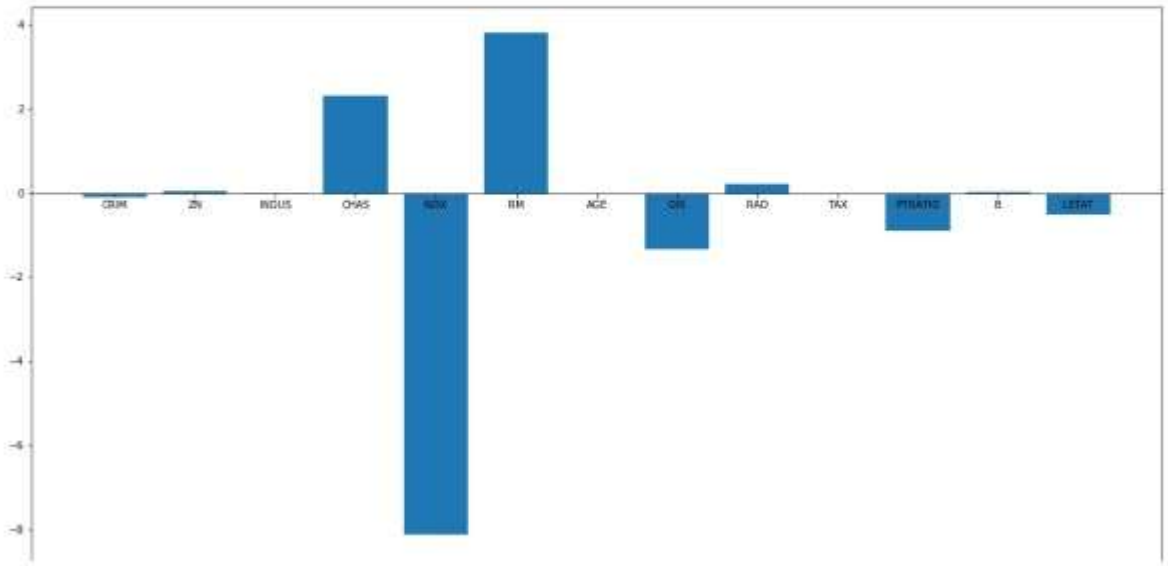


```
from sklearn.linear_model import Ridge

ri=Ridge()
ri.fit(X_train,y_train)
ri_y_pred=ri.predict(X_test)
```

In [30]:

```
ri_coff=pd.DataFrame()
ri_coff['Columns']=X_train.columns
ri_coff['Cofficient']=pd.Series(ri.coef_)
```

```
fig,ax=plt.subplots(figsize=(20,10))
ax.bar(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'],ri_coff['Cofficient'])
ax.spines['bottom'].set_position('zero')
plt.show()
```



Ques 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.

Code & Output:

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
```

In [2]:

```
df= pd.read_csv('diabetes.csv')
```

In [3]:

```
df
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	
...	...	...	...	...	...	...	
763	10	101	76	48	180	32.9	
764	2	122	70	27	0	36.8	

```
feature_cols = ['Pregnancies', 'Insulin', 'BMI', 'Age','Glucose','BloodPressure','Diabe
tesPedigreeFunction']
X = df[feature_cols]
y = df.Outcome
```

In [5]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
```

In [12]:

```
sgdc = SGDClassifier(max_iter=100,)
```

```
sgdc.fit(X_train,y_train)
y_pred=sgdc.predict(X_test)
```

In [14]:

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("F1-score:",metrics.f1_score(y_test, y_pred))
```

```
Accuracy: 0.6510416666666666
Precision: 0.463768115942029
Recall: 0.5161290322580645
F1-score: 0.4885496183206106
```

In [15]:

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

Out[15]:

```
array([[93, 37],
       [30, 32]], dtype=int64)
```