INDRAPRASTHA COLLEGE FOR WOMEN

UNIVERSITY OF DELHI

COURSE: BSC. (HONS.) COMPUTER SCIENCE

PRACTICAL: MACHINE LEARNING

SUBMITTED TO: MISS DIKSHA JAIN

SUBMITTED BY: TASHI LAMO
ROLL NO.:21/CS/54

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

In [1]:

```
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)
```

```
First value- 2
Second value- 3
Addition- 5
Multiplication- 6
Division- 0.6666666666666666
Exponent- 8
```

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

In [2]:

```
#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")
```

```
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
```

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

In [3]:
```
import numpy as py
```

In [4]:
```
s='apples'
print('I like to have {0} every day . '.format(s))
```

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

In [5]:
```
s = '{0} is of type integer, {1} is of the type float, {2} is of the type string'
print(f'{6} is of the type integer, {1.9} is of the type float, {"hello"} is of the
```

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

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

```
dimension of array is : 1
```

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

In [7]:
```
arr1 = arr.reshape((2,3))
arr1
```

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

In [8]:
```
arr=py.array([1,2,3,4,5,6])
print('dimension of array is ',arr.ndim)
arr
```

```
dimension of array is  1
```

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

In [9]:
```
arr1=arr.reshape((2,3))
```

In [10]:
```
print('dimension of array is ',arr1.ndim)
```

```
dimension of array is  2
```

In [11]:
```
arr2=py.zeros((2,3))
arr2
```

Out[11]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [12]:
```
arr3=py.ones((3,3))
arr3
```

Out[12]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

arr2 = py.zeros_like((arr3)) arr2

In [13]:
```
py.eye(4)
```

Out[13]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

In [14]:
```
t = py.random.rand(12)
```

In [15]:
```
py.array(t).reshape(6,2)
```

Out[15]:
```
array([[0.41504095, 0.35564083],
       [0.50624062, 0.06586177],
       [0.2592188 , 0.34083234],
       [0.87612314, 0.10547914],
       [0.39187968, 0.69588613],
       [0.13092173, 0.41053725]])
```

Question 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 [16]:
```
mat=py.array([1,2,3,4,5,6]).reshape(2,3)
mat
```

Out[16]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [17]:
```
mat.size
mat
```

Out[17]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [18]:
```
mat[0].size
```

Out[18]:
```
3
```

In [19]:
```
import pandas as pd
```

In [20]:
```
df=pd.read_csv('pro-5-text.txt',sep=" ",header=None)
df
```

Out[20]:

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

In [21]:
```
df.to_csv('text_file.txt')
```

In [22]:
```
f= open('text_file.txt')
```

In [23]:
```
print(f.read())
```

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

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

In [28]:

```
a=py.array([[11,12],
            [13,14],
            [15,16],
            [17,18]])
b=py.array([[1,2],
            [3,4],
            [5,6],
            [7,8]])
print('addition of two arrays \n',a+b)
```

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

In [25]:

```
print('Subtraction of two arrays \n',a-b)
```

```
Subtraction of two arrays 
 [[10 10]
 [10 10]
 [10 10]
 [10 10]]
```

In [26]:

```
a=py.array([[1,2,5],
            [3,4,6],
            [4,5,6]])
b=py.array([[4,5,1],
            [4,5,7],
            [2,3,2]])
print('multiplication of two arrays \n',py.dot(a,b))
```

```
multiplication of two arrays 
 [[22 30 25]
 [40 53 43]
 [48 63 51]]
```

Question 7: Perform other matrix operations like converting matrix data to absolute values, taking the negative of matrix values, adting/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 [29]:

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

```
Adding column
```

Out[29]:

```
array([['11', '12', 'a'],
       ['13', '14', 'b'],
       ['15', '16', 'c'],
       ['17', '18', 'd']], dtype='<U11')
```

```
In [30]: print('Adding row')
         py.insert(a,2,[[88,88]],axis=0)
```

```
Adding row
```

```
Out[30]: array([[11, 12],
                [13, 14],
                [88, 88],
                [15, 16],
                [17, 18]])
```

```
In [31]: print('maximum of matrix',py.max(a))
```

```
maximum of matrix 18
```

```
In [32]: print('minimum of matrix',py.min(a))
```

```
minimum of matrix 11
```

```
In [33]: print('Sum of matrix',py.sum(a))
```

```
Sum of matrix 116
```

```
In [34]: print("Sum of first row of matrix",py.sum(a[0]))
```

```
Sum of first row of matrix 23
```

```
In [35]: print("Absolute values of matrix")
         py.absolute(a)
```

```
Absolute values of matrix
```

```
Out[35]: array([[11, 12],
                [13, 14],
                [15, 16],
                [17, 18]])
```

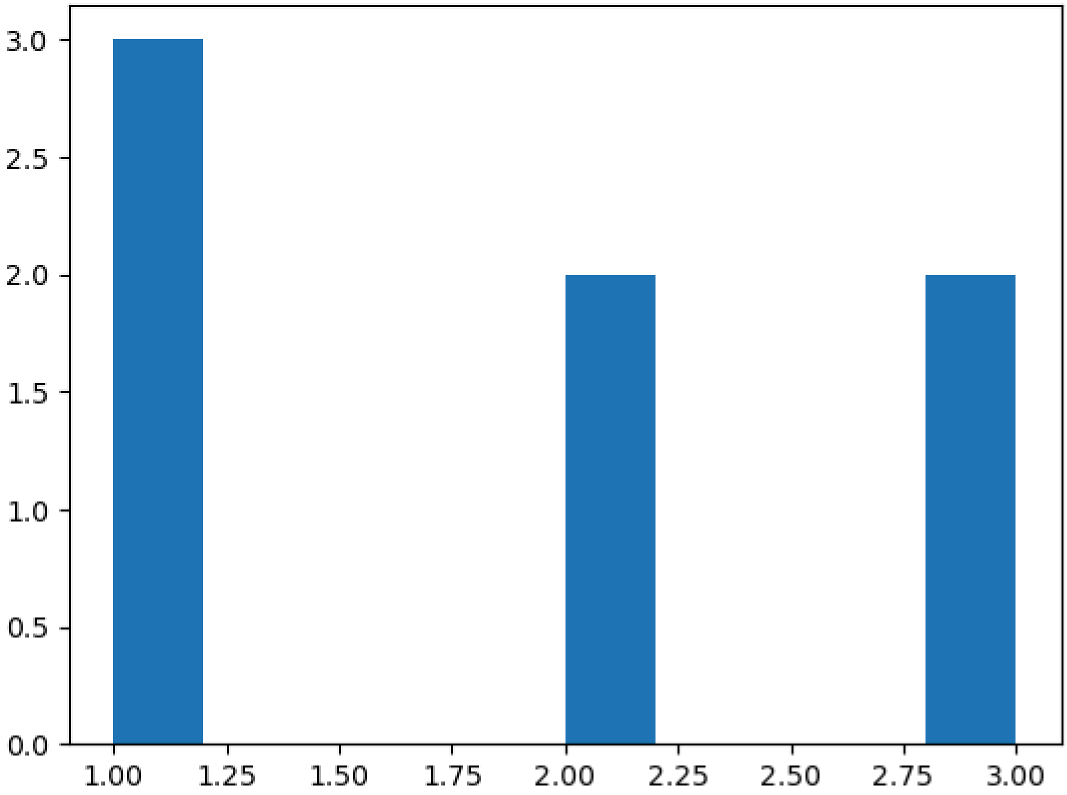
```
In [36]: print("Negative of matrix",py.negative(a))
```

```
Negative of matrix [[-11 -12]
 [-13 -14]
 [-15 -16]
 [-17 -18]]
```

Question 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 [37]: import matplotlib.pyplot as plt
         import math
```

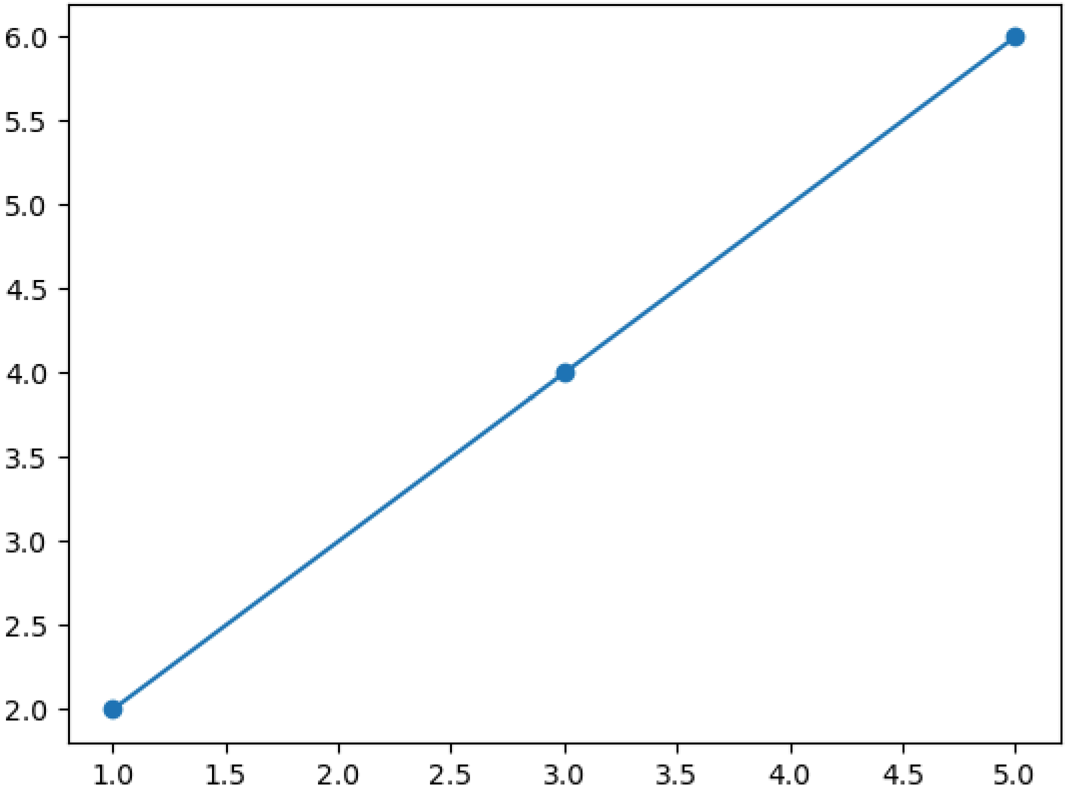
```
In [38]: plt.hist([1,1,1,2,2,3,3])
         plt.show()
```



In [39]:
```
arr=py.array([1,2,3,4,5,6]).reshape(3,2)
arr
```

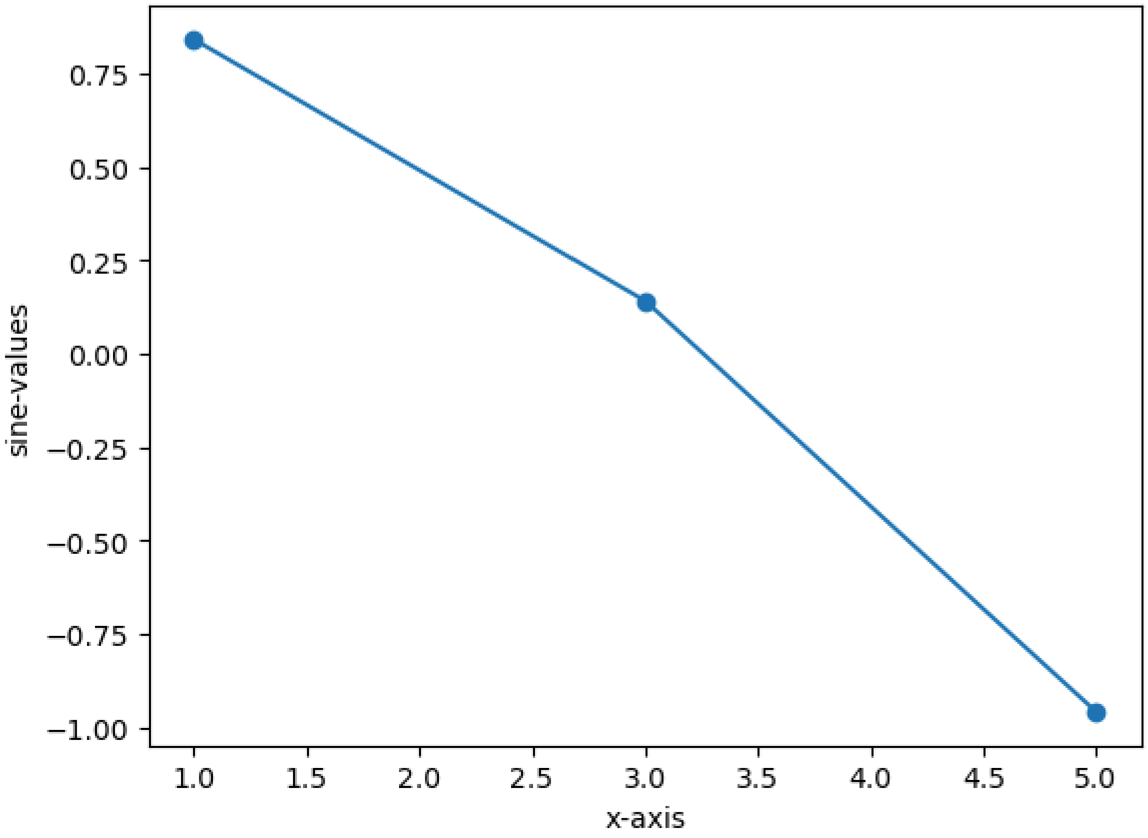
Out[39]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

In [40]:
```
plt.plot(arr[:,0],arr[:,1],marker='o')
plt.show()
```



```
In [41]: sin_value=[math.sin(i) for i in arr[:,0]]
```

```
In [42]: plt.plot(arr[:,0],sin_value,marker='o')
         plt.xlabel('x-axis')
         plt.ylabel('sine-values')
         plt.show()
```

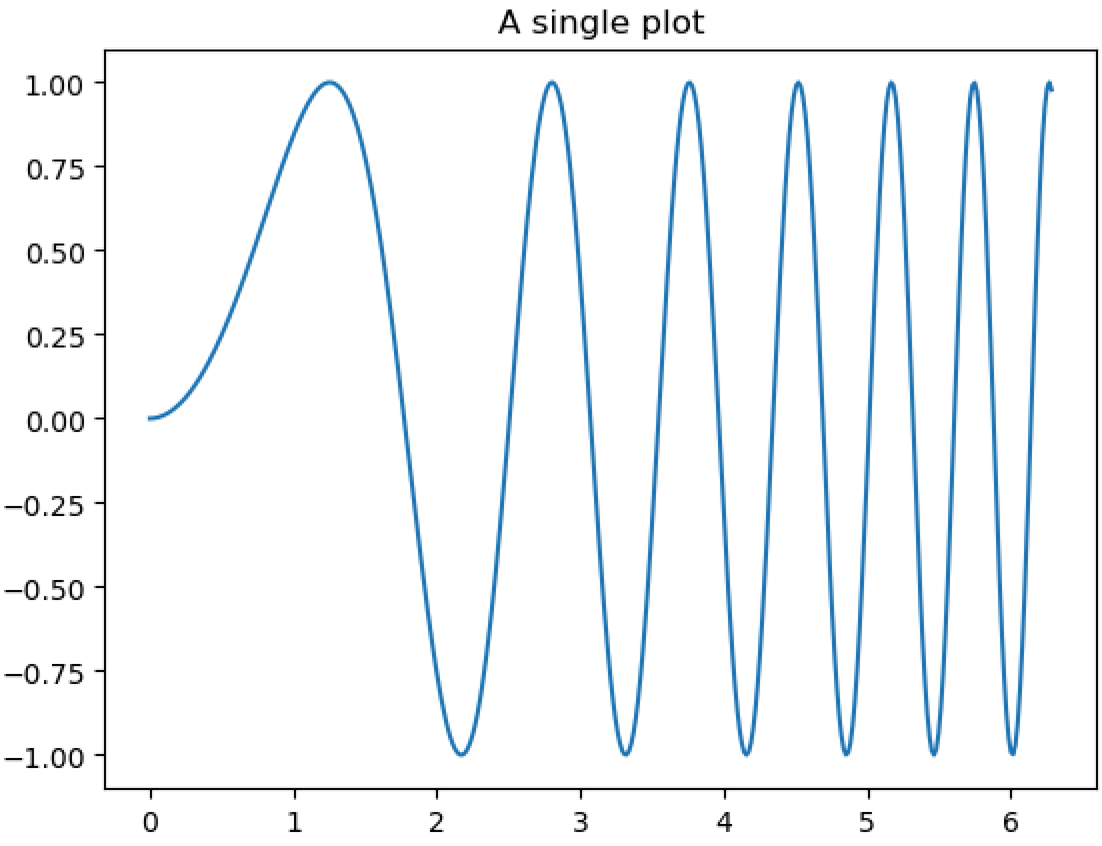


Question 9: Generate different subplots from a given plot and colour plot data.

```
In [44]: import numpy as np
         x=np.linspace(0,2*np.pi,400)
         y=np.sin(x**2)
```

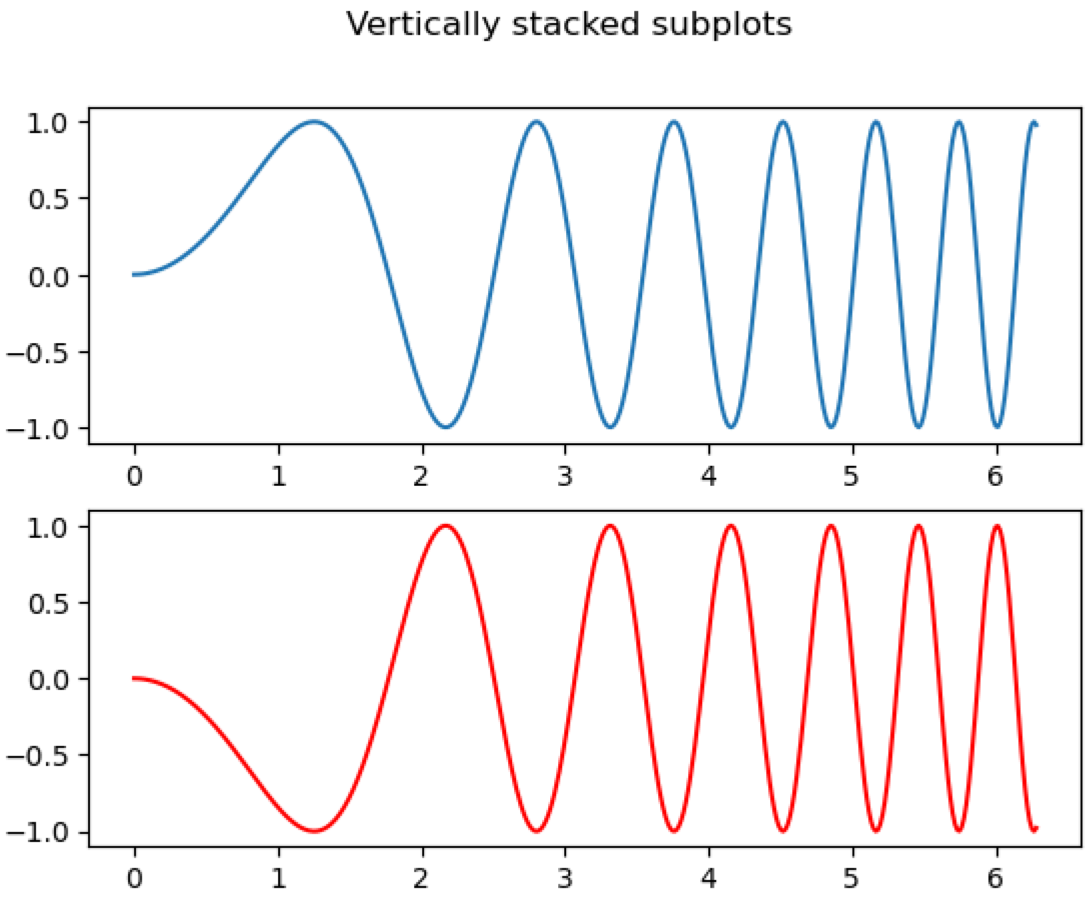
```
In [45]: fig,ax=plt.subplots()
         ax.plot(x,y)
         ax.set_title("A single plot")
```

```
Out[45]: Text(0.5, 1.0, 'A single plot')
```



In [46]:

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


Vertically stacked subplots

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

In [47]:
```
fruits =["apple","banana","cherry"]
for x in fruits:
    print(x)
    if x== "banana":
        break
```

```
apple
banana
```

In [48]:
```
i=1
while i<6:
    print(i)
    if i ==3:
        break
    i+=1
```

```
1
2
3
```

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

In [91]:

```
#import numpy as np

# Function to find the transpose of a matrix
def transpose(matrix):
    return np.transpose(matrix)

# Function to add two matrices
def add_matrices(matrix1, matrix2):
    return np.add(matrix1, matrix2)

# Function to subtract two matrices
def subtract_matrices(matrix1, matrix2):
    return np.subtract(matrix1, matrix2)

# Function to multiply two matrices
def multiply_matrices(matrix1, matrix2):
    return np.dot(matrix1, matrix2)

# Example usage:
# Define two matrices
matrix_a = np.array([[1, 2, 3],
                     [4, 5, 6]])

matrix_b = np.array([[7, 8, 9],
                     [10, 11, 12]])

# Find the transpose of matrix_a
transposed_matrix_a = transpose(matrix_a)
print("Transpose of matrix_a:")
print(transposed_matrix_a)

# Add matrix_a and matrix_b
added_matrices = add_matrices(matrix_a, matrix_b)
print("\nAddition of matrix_a and matrix_b:")
print(added_matrices)

# Subtract matrix_b from matrix_a
subtracted_matrices = subtract_matrices(matrix_a, matrix_b)
print("\nSubtraction of matrix_b from matrix_a:")
print(subtracted_matrices)

# Multiply matrix_a and matrix_b
multiplied_matrices = multiply_matrices(matrix_a, matrix_b.transpose())  # Multiply
print("\nMultiplication of matrix_a and the transpose of matrix_b:")
print(multiplied_matrices)
```

```
Transpose of matrix_a:
[[1 4]
 [2 5]
 [3 6]]

Addition of matrix_a and matrix_b:
[[ 8 10 12]
 [14 16 18]]

Subtraction of matrix_b from matrix_a:
[[-6 -6 -6]
 [-6 -6 -6]]

Multiplication of matrix_a and the transpose of matrix_b:
[[ 50  68]
 [122 167]]
```

In [1]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

Question 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.

In [10]:

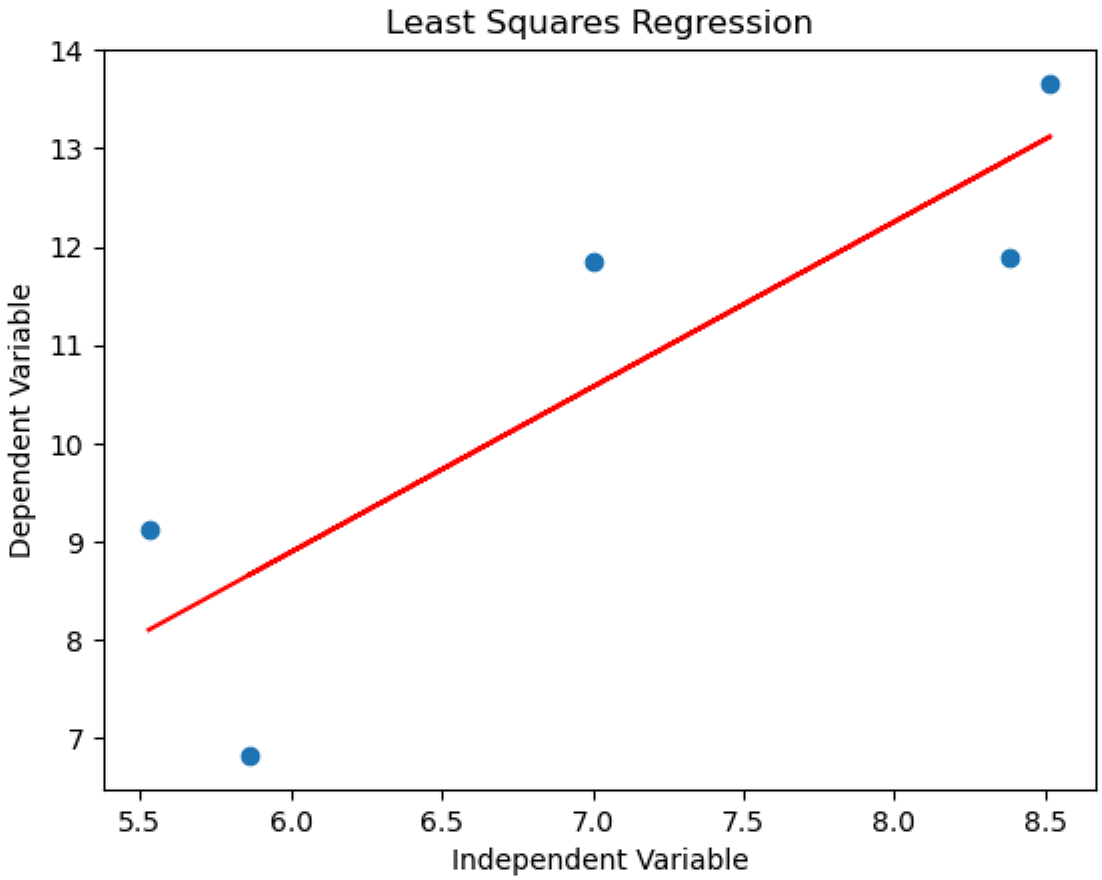
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv('lindata.csv')
df.columns=['X','Y']
df
```

Out[10]:

	X	Y
0	5.5277	9.1302
1	8.5186	13.6620
2	7.0032	11.8540
3	5.8598	6.8233
4	8.3829	11.8860

In [12]:

```
def leastSquareRegression(indep_val, dep_val, predict_indep_val):
    mean_dep_val = dep_val.mean()
    mean_indep_val = indep_val.mean()
    diff_dep_val = dep_val - mean_dep_val
    diff_indep_val = indep_val - mean_indep_val

    numerator = (diff_indep_val * diff_dep_val).sum()
    denominator = (diff_indep_val ** 2).sum()

    beta1 = numerator / denominator
    beta0 = mean_dep_val - (beta1 * mean_indep_val)
    predict_dep_val = beta0 + beta1 * predict_indep_val

    plt.scatter(indep_val, dep_val)
    plt.plot(indep_val, beta0 + beta1 * indep_val, 'red')
    plt.xlabel('Independent Variable')
    plt.ylabel('Dependent Variable')
    plt.title('Least Squares Regression')
    plt.show()

    return {'beta0': beta0, 'beta1': beta1, 'predict_val': predict_dep_val}

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 : 25.135710655002136
```

Question 13: Based on multiple features/variables perform Linear Regression. For example, based on several 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.

In [13]:

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

Out[13]:

	X	Y
0	5.5277	9.1302
1	8.5186	13.6620
2	7.0032	11.8540
3	5.8598	6.8233
4	8.3829	11.8860

In [14]:
```
X=np.array(df['X']).reshape(-1,1)
Y=np.array(df['Y']).reshape(-1,1)
```

In [15]:
```
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size = 0.25)
```

In [16]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[16]:
```
▾ LinearRegression
LinearRegression()
```

In [17]:
```
Y_pred=lr.predict(x_test)
```

In [24]:
```
print('mean square error :',mean_squared_error(y_test,Y_pred))
print(r2_score(y_test,Y_pred))
```

```
mean square error : 7.774748341791334
-3.191755092580558
```

In [25]:
```
lr.score(x_test, y_test)
```

Out[25]:
```
-3.191755092580558
```

In [20]:
```
print('slope:',lr.coef_)
print('Intercept:',lr.intercept_)
```

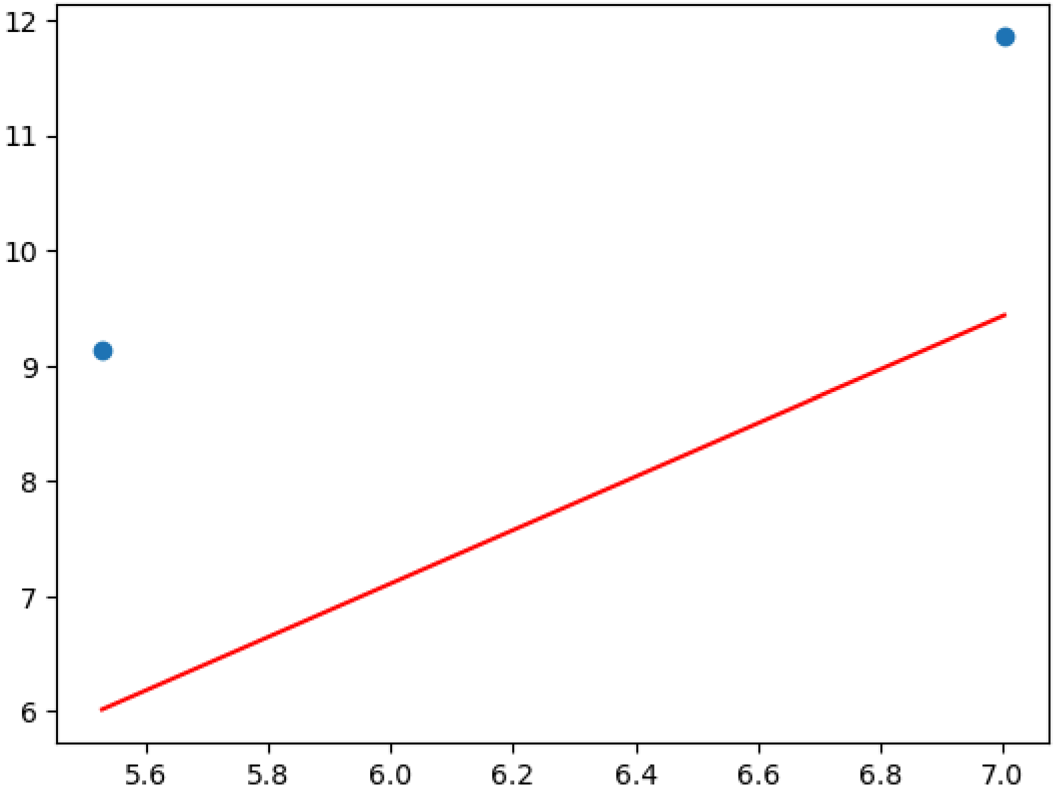
```
slope: [[2.31888009]]
Intercept: [-6.80314178]
```

In [21]:
```
y=lr.predict([[15.67]])
```

In [22]:
```
y[0][0]
```

Out[22]:
```
29.53370919583682
```

In [23]:
```
plt.scatter(x_test,y_test)
plt.plot(x_test,Y_pred,'red')
plt.show()
```



Question 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.

In [51]:
```
import seaborn as sns
#from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [31]:
```
df=pd.read_csv('diabetes.csv')
df
```

Out[31]:

	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	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 9 columns

In [45]:

```
feature_cols = ['Pregnancies','Insulin','BMI','Age','Glucose','BloodPressure','Diab
x=df[feature_cols]
y=df.Outcome
```

In [46]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
```

In [47]:

```
logreg=LogisticRegression()
logreg.fit(x_train,y_train)
y_pred=logreg.predict(x_test)
```

```
C:\Users\lamot\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:458: Co
nvergenceWarning: 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 in:
    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-regression
  n_iter_i = _check_optimize_result(
```

In [48]:

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

Out[48]:

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

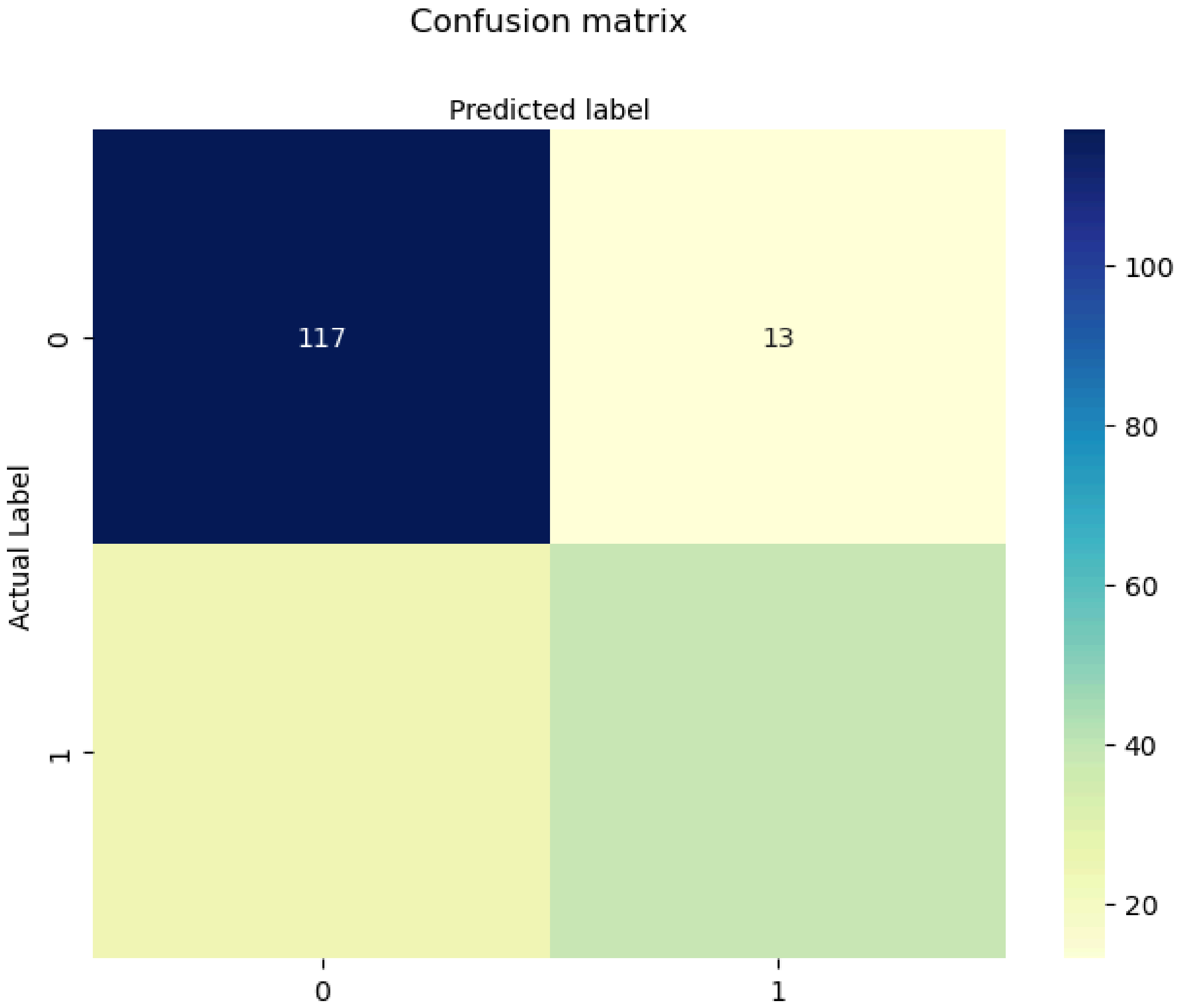
In [49]:

```
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')
```

Out[49]:

```
Text(0.5, 427.9555555555555, 'Predicted label')
```



In [50]:

```
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
```

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

In [53]:

```
#import pandas as pd
#import numpy as py
```

```
#import matplotlib.ppyplot as plt
from sklearn import datasets
#from sklearn.linear_model import LinearRegression
#from sklearn.model_selection import train_test_split
```

In [56]:
```
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]
```

In [59]:
```
boston_dataset=raw_df
boston_dataset
```

Out[59]:

	0	1	2	3	4	5	6	7	8	9	10
0	0.00632	18.00	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3
1	396.90000	4.98	24.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	0.02731	0.00	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8
3	396.90000	9.14	21.60	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	0.02729	0.00	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8
...	...	...	...	...	...	...	...	...	...	...	...
1007	396.90000	5.64	23.90	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1008	0.10959	0.00	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0
1009	393.45000	6.48	22.00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1010	0.04741	0.00	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0
1011	396.90000	7.88	11.90	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

1012 rows × 11 columns

In [60]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
```

In [61]:
```
lreg=LinearRegression()
lreg.fit(x_train,y_train)
```

Out[61]:
```
▾ LinearRegression
LinearRegression()
```

In [62]:
```
y_pred=lreg.predict(x_test)
```

In [67]:
```
lreg_coff=pd.DataFrame()
lreg_coff['Columns']=x_train.columns
lreg_coff['Cofficient']=pd.Series(lreg.coef_)
```

In [68]:

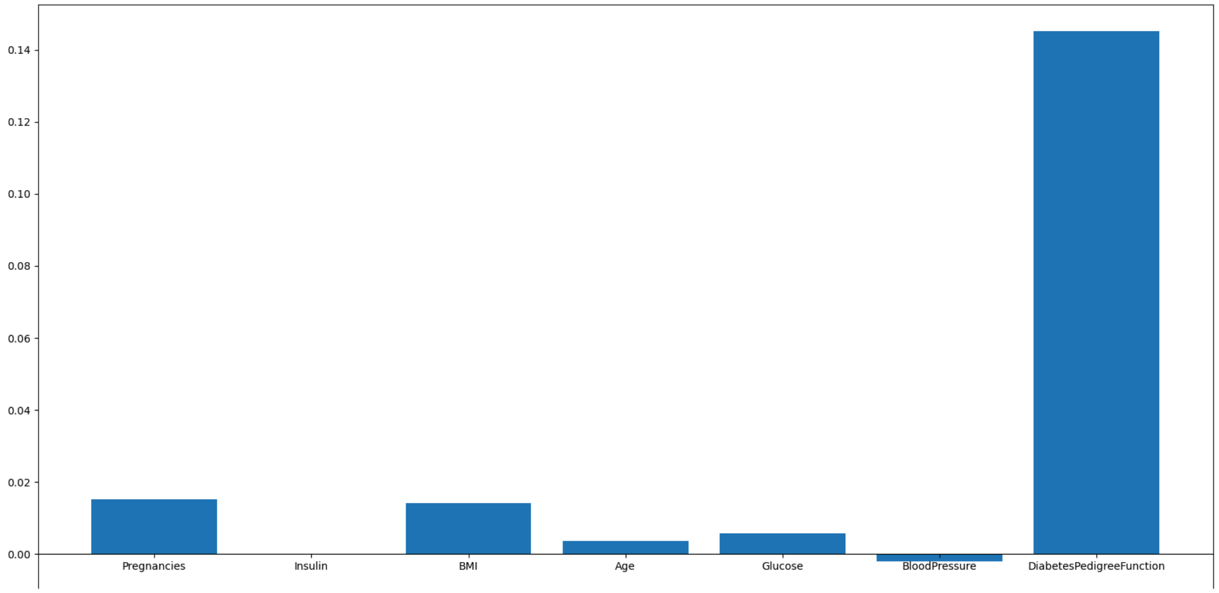
```
lreg_coff
```

Out[68]:

	Columns	Cofficient
0	Pregnancies	0.015115
1	Insulin	-0.000119
2	BMI	0.014168
3	Age	0.003623
4	Glucose	0.005652
5	BloodPressure	-0.002039
6	DiabetesPedigreeFunction	0.145175

In [83]:

```
fig,ax=plt.subplots(figsize=(20,10))
ax.bar(['Pregnancies','Insulin','BMI','Age','Glucose','BloodPressure','DiabetesPedi
ax.spines['bottom'].set_position('zero')
plt.show()
```



In [77]:

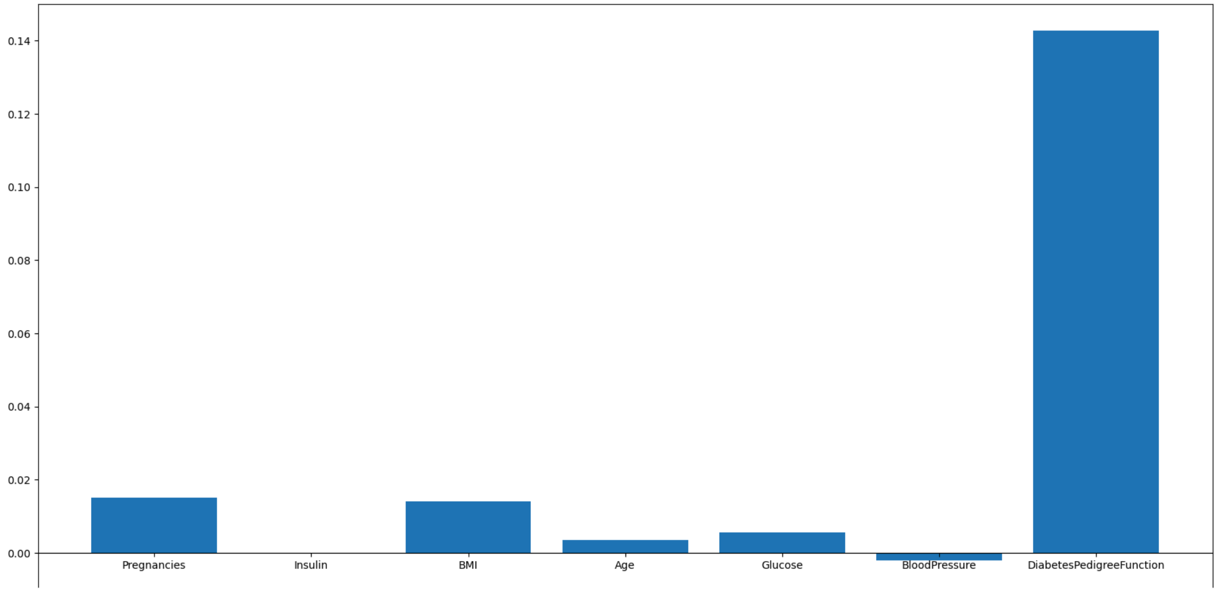
```
from sklearn.linear_model import Ridge

ri=Ridge()
ri.fit(x_train,y_train)
ri_y_pred=ri.predict(x_test)
```

In [78]:

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

In [82]:

```
fig,ax=plt.subplots(figsize=(20,10))
ax.bar(['Pregnancies','Insulin','BMI','Age','Glucose','BloodPressure','DiabetesPedi
ax.spines['bottom'].set_position('zero')
plt.show()
```



Question 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 [84]:
```
#import pandas as pd
#from sklearn.model_selection import train_test_split
#numpy as np
from sklearn.linear_model import SGDClassifier
#from sklearn import metrics
```

In [85]:
```
df=pd.read_csv('diabetes.csv')
df
```

Out[85]:

	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	
765	5	121	72	23	112	26.2	
766	1	126	60	0	0	30.1	
767	1	93	70	31	0	30.4	

768 rows × 9 columns

In [86]:

```
feature_cols=['Pregnancies','Insulin','BMI','Age','Glucose','BloodPressure','Diabet
x=df[feature_cols]
y=df.Outcome
```

In [87]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=0)
```

In [88]:

```
sgdc=SGDClassifier(max_iter=100,)
sgdc.fit(x_train,y_train)
y_pred=sgdc.predict(x_test)
```

In [89]:

```
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.accuracy_score(y_test,y_pred))
```

```
Accuracy: 0.359375
Precision: 0.33513513513513515
Recall: 1.0
F1-score 0.359375
```

In [90]:

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

Out[90]:

```
array([[  7, 123],
       [  0,  62]], dtype=int64)
```