K J Somaiya Institute of Management

Vidyavihar, Mumbai – 400 077

Masters in Computer Applications

Trimester III (2021 – 23)

This is to certify that Mr Mrinaal Paliwal Roll No. 2 5 ofMCA has completed his/her Artificial Intelligence and Machine Learning Journal as per the Syllabus for the Academic year 2021 – 2022. The Journal has also been evaluated by the concerned faculty of K J Somaiya Institute of Management.

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Signature of the Faculty-In charge

_ _ _ _ _ _ _ _ _ _ _ _ _

Signature of the Course Coordinator

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Date:_____

Signature of the External Examiner

Index

23.	K Nearest Neighbor – Wine Quality Dataset	21/6/22		
24.	Ensemble Learning – Random Forest Classifier	22/6/22		
	Unsupervised Learning			
25.	K Means Clustering - Digit dataset	22/6/22		
26.	K Means Clustering – Mall Data	22/6/22		
	Improving the Performance			
27.	Feature Selection	22/6/22		
28.	Principle Component Analysis	22/6/22		
29.	Cross Validation Techniques – Iris Dataset	21/6/22		
30.	**Project**	27/6/22		

Practical 1 - Numpy

6 June 2022

In [1]:
```
import numpy as np
```

In [2]:
```
x=[1,2,3,4,5,6]
```

In [3]:
```
x
```

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

In [4]:
```
type(x)
```

Out[4]:
```
list
```

In [5]:
```
y=np.array([1,2,3,4,5,6])
```

In [6]:
```
type(y)
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
y=np.array((1,2,3,4,5,6))
```

In [8]:
```
type(y)
```

Out[8]:
```
numpy.ndarray
```

In [9]:
```
y
```

Out[9]:
```
array([1, 2, 3, 4, 5, 6])
```

In [10]:
```
x=[1,2,'gaurav',18]
x
```

Out[10]:
```
[1, 2, 'gaurav', 18]
```

In [11]:
```
y=np.array(x)
y
```

Out[11]:
```
array(['1', '2', 'gaurav', '18'], dtype='<U11')
```

In [12]:
```
y=np.linspace(start=0,stop=20,num=10)
```

In [13]:
```
y
```

Out[13]:
```
array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
       11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ])
```

In [14]:
```
y=np.linspace(start=0,stop=20,num=10,endpoint=True)
y
```

Out[14]:
```
array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
       11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ])
```

In [15]:
```
y=np.linspace(start=0,stop=20,num=10,endpoint=False)
y
```

Out[15]:
```
array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14., 16., 18.])
```

In [16]:
```
y=np.linspace(start=0,stop=20,num=10,endpoint=True,retstep=True) #retstrp gives th
y
```

Out[16]:
```
(array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
        11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ]),
 2.2222222222222223)
```

In [17]:
```
y=np.linspace(start=0,stop=20,num=10,endpoint=True,retstep=False)
y
```

Out[17]:
```
array([ 0.        ,  2.22222222,  4.44444444,  6.66666667,  8.88888889,
       11.11111111, 13.33333333, 15.55555556, 17.77777778, 20.        ])
```

In [18]:
```
d=np.arange(start=1,stop=10,step=2) #arange is to create array with conditions
d
```

Out[18]:
```
array([1, 3, 5, 7, 9])
```

In [19]:
```
# 3D Array
#it is giving a normal nested list
z=[[1,2,3],[4,5,6],[7,8,9]]
z
```

Out[19]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

In [20]:
```
z=np.array([[1,2,3],[4,5,6],[7,8,9]]) #if we use np.array it will give 3D array
z
```

Out[20]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [21]:
```
# array of 1's
# give the parameter
np.ones((3,3)) #means 3x3 matrix of one's
```

Out[21]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [22]:
```
# now with zero array
np.zeros((4,5)) # means 4x5 matrix array of zeroes
```

Out[22]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [23]:
```
 np.eye(4) # for identity matrix , means diagonal is 1
```

Out[23]:
```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

RESHAPING

In [24]:
```
x=np.array([0,1,2,3,4,5,6,7,8,9])
x
```

Out[24]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [25]:
```
x.reshape((5,2)) # change the shape of 1D matrix to given parameters
```

Out[25]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
```

SLICING

In [26]:
```
a=np.arange(10) # it will create array
```

In [27]:
```
a
```

Out[27]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [28]:
```
s= slice(2,7,2)
# start , end , step
```

In [29]:
```
a[s]
```

Out[29]:
```
array([2, 4, 6])
```

In [30]:
```
s=slice(1,6,2)
a[s]
```

Out[30]:
```
array([1, 3, 5])
```

In [31]:
```
x[2:7]
```

Out[31]:
```
array([2, 3, 4, 5, 6])
```

In [32]:
```
x[-6:-2]
```

Out[32]:
```
array([4, 5, 6, 7])
```

random number generation

In [33]:
```
np.random.rand(5) #gives random numbers
```

Out[33]:
```
array([0.98573456, 0.11915051, 0.01743693, 0.01964315, 0.93730031])
```

In [34]:
```
np.random.rand(5,4)  #gives random numbers in 2D format if parameters are given
```

Out[34]:
```
array([[0.62770376, 0.86662857, 0.10921151, 0.21229802],
       [0.99984639, 0.99483125, 0.33217128, 0.10036316],
       [0.07608809, 0.5998778 , 0.44511185, 0.83293409],
       [0.93563019, 0.80366581, 0.23843873, 0.66867432],
       [0.81839422, 0.3003223 , 0.41237769, 0.85932887]])
```

In [35]:
```
np.random.randn(5) #
```

Out[35]:
```
array([ 1.15278608,  0.50173782,  0.03903805, -0.41357126, -0.94009669])
```

In [36]:
```
np.random.randn(5,5)
```

Out[36]:
```
array([[-1.47407791, -1.68452754, -0.21566416,  1.35627454, -1.03834738],
       [ 1.10318554, -0.04085744, -0.02478691, -0.70846019, -0.71480808],
       [-0.70695526, -1.01261306, -0.54574004, -0.12710284, -1.69118457],
       [ 0.9289106 , -0.71109946,  1.19214981, -0.88550132,  0.884397  ],
       [-0.09981517,  0.53211564,  0.05890128,  1.70995199,  0.36538546]])
```

randint

In [37]:
```
np.random.randint(low=1,high=100)
```

Out[37]:
```
7
```

In [38]:
```
np.random.randint(low=1,high=100,size=20)
```

Out[38]:
```
array([ 6, 94, 76, 44, 68, 57, 39, 85, 45, 25, 54, 55, 24, 62, 22,  5, 72,
       41, 35, 32])
```

seed

In [39]:
```
np.random.seed(69) # seed is for picking same value of random data
#69 is the starting point
# 4 is the random 4 numbers from starting point
np.random.rand(4)
```

Out[39]:
```
array([0.29624916, 0.80906772, 0.35025253, 0.78940926])
```

In [40]:
```
np.random.seed(69)
np.random.rand(5)
```

Out[40]:
```
array([0.29624916, 0.80906772, 0.35025253, 0.78940926, 0.56134898])
```

Broadcasting

In [41]:
```
#broadcasting means the things we will do to the array it will be reflected to who
arr=np.arange(0,10,1)
arr
```

Out[41]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [42]:
```
arr1=arr/10 #here we are div the whole array by 10
arr1
```

Out[42]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

In [43]:
```
arr*5 # here we are multiplying it by 5
```

Out[43]:
```
array([ 0,  5, 10, 15, 20, 25, 30, 35, 40, 45])
```

In [44]:
```
slice_arr=arr[0:6]
slice_arr
```

Out[44]:
```
array([0, 1, 2, 3, 4, 5])
```

In [45]:
```
slice_arr[:]=555
slice_arr
```

Out[45]:
```
array([555, 555, 555, 555, 555, 555])
```

In [46]:
```
arr_copy = arr.copy()
```

In [47]:
```
arr_copy[:]=1000
arr_copy
```

Out[47]:
```
array([1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000])
```

In [48]:
```
arr
```

Out[48]:
```
array([555, 555, 555, 555, 555, 555,   6,   7,   8,   9])
```

Conditional Selection

In [49]:
```
arr=np.arange(0,10,1)
arr
```

Out[49]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [50]:
```
boo_arr=arr>4 # it is checking the bool value with the condtion provided if the co
#will return True else False
boo_arr
```

Out[50]:
```
array([False, False, False, False, False,  True,  True,  True,  True,
        True])
```

In [51]:
```
arr[arr>4]
```

Out[51]:
```
array([5, 6, 7, 8, 9])
```

In [52]:
```
arr
```

Out[52]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [53]:
```
arr+arr # array + array it will return the point wise array
```

Out[53]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [54]:
```
arr * arr
```

Out[54]:
```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

In [55]:
```
arr/arr # warning becoz of the zero , that's why the first value is Nan (not a num
```

```
C:\Users\gaurav\AppData\Local\Temp\ipykernel_16372\3965243577.py:1: RuntimeWarning:
invalid value encountered in true_divide
  arr/arr # warning becoz of the zero , that's why the first value is Nan (not a n
umber)
```

Out[55]:
```
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

In [56]:
```
arr**9 # ** is raise
```

Out[56]:
```
array([        0,         1,       512,     19683,    262144,   1953125,
        10077696,  40353607, 134217728, 387420489], dtype=int32)
```

Universal Array Functions

In [57]:
```
arr
```

Out[57]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [58]:
```
np.sqrt(arr) # sq root of each number
```

Out[58]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

In [59]:
```
np.exp(arr)
```

Out[59]:
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

In [60]:
```
np.sin(arr)
```

Out[60]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

In [61]:
```
size = 100
```

In [62]:
```
x1=range(size)
y1=range(size)
```

In [63]:
```
x1
```

Out[63]:
```
range(0, 100)
```

In [64]:
```
y1
```

Out[64]:
```
range(0, 100)
```

In [65]:
```
%%timeit # it will give the time to execute this statement
[x1[i]*y1[i] for i in range(size)]
```

```
12.6 µs ± 543 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

In [66]:
```
x2=np.arange(size)
y2=np.arange(size)
```

In [67]:
```
%%timeit
x2*y2
```

```
418 ns ± 42.4 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

Append

In [68]:
```
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
a
```

Out[68]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [69]:
```
a_row=np.append(a,[[21,22,69]],axis=0) #axis = 0 means adding a row
a_row                                  # axis = 1 means adding a coloum
```

Out[69]:
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [21, 22, 69]])
```

In [70]:
```
a_col=np.array([34,23,55]).reshape(3,1)
a_col
```

Out[70]:
```
array([[34],
       [23],
       [55]])
```

In [71]:
```
a_col=np.append(a,a_col,axis=1)
a_col
```

Out[71]:
```
array([[ 1,  2,  3, 34],
       [ 4,  5,  6, 23],
       [ 7,  8,  9, 55]])
```

In [ ]:

Practical 2 - Pandas

7 June 2022

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
# create vector as a row
vector_row=np.array([1,2,3])
```

In [3]:

```
# create vector as a column
vector_column=np.array([[1],[2],[3]])
```

In [4]:

```
vector_row
```

Out[4]:

```
array([1, 2, 3])
```

In [5]:

```
vector_column
```

Out[5]:

```
array([[1],
       [2],
       [3]])
```

Creating a matrix

In [6]:

```
from scipy import sparse
```

only non zero elements will be stored in sparse matrix

In [7]:

```
matrix=np.array([[0,0],
                 [0,1],
                 [3,0]])

#create compressed sparse row (CSR) Matrix
matrix_sparse = sparse.csr_matrix(matrix)
```

In [8]:

```
matrix_sparse
```

Out[8]:

```
<3x2 sparse matrix of type '<class 'numpy.intc'>'
        with 2 stored elements in Compressed Sparse Row format>
```

In [9]:

```
matrix
```

Out[9]:

```
array([[0, 0],
       [0, 1],
       [3, 0]])
```

In [10]:

```
print(matrix_sparse) # it is giving the position , means 1 is it location 1st row 1st col ,
# and 3 is in locaiton 2nd row 1st col
```

```
  (1, 1)        1
  (2, 0)        3
```

In [11]:

```
#create a large matrix
martix_large=np.array([[0,0,0,0,0,0,0,0],
                       [5,3,0,0,0,0,0,0],
                       [0,0,0,8,0,0,0,0],
                       [0,4,0,0,0,1,0,69]])
```

In [12]:

```
#create compressed sparse row (CSR) matrix
matrix_large_sparse = sparse.csr_matrix(martix_large)
```

In [13]:

```
#view larger sparse matrix
print(matrix_large_sparse)
```

```
  (1, 0)        5
  (1, 1)        3
  (2, 3)        8
  (3, 1)        4
  (3, 5)        1
  (3, 7)        69
```

In [14]:

```
#create 3x3 matrix
matrix=np.array([[1,2,3],
                 [4,5,6],
                  [7,6,9]])
```

In [15]:

```
#finding maximum element in each column
np.max(matrix,axis=0)
```

Out[15]:

```
array([7, 6, 9])
```

In [16]:

```
#finding maximum element in each row
np.max(matrix,axis=1)
```

Out[16]:

```
array([3, 6, 9])
```

Type *Markdown* and LaTeX: $\square^2$

In [17]:

```
#return mean
np.mean(matrix)
```

Out[17]:

```
4.777777777777778
```

In [18]:

```
#finding variance
np.var(matrix)
```

Out[18]:

```
5.728395061728396
```

In [19]:

```
#finding standard deviation
np.std(matrix)
```

Out[19]:

```
2.3934065809486684
```

Reshaping of Array

In [20]:

```
#create 4x3 matrix
matrix=np.array([[1,2,3],
                 [4,5,6],
                 [6,7,8,],
                 [6,54,7]])
```

In [21]:

```
#reshaping matrix in 2x6 matrix
matrix.reshape(2,6)
```

Out[21]:

```
array([[ 1,  2,  3,  4,  5,  6],
       [ 6,  7,  8,  6, 54,  7]])
```

Type *Markdown* and LaTeX: $\square^2$

In [22]:

```
matrix.reshape(1,-1)
```

Out[22]:

```
array([[ 1,  2,  3,  4,  5,  6,  6,  7,  8,  6, 54,  7]])
```

In [23]:

```
matrix.size
```

Out[23]:

```
12
```

In [24]:

```
matrix.reshape(matrix.size)
```

Out[24]:

```
array([ 1,  2,  3,  4,  5,  6,  6,  7,  8,  6, 54,  7])
```

In [25]:

```
#TRANSPOSE OF MATRIX
matrix.T
```

Out[25]:

```
array([[ 1,  4,  6,  6],
       [ 2,  5,  7, 54],
       [ 3,  6,  8,  7]])
```

In [26]:

```
#TRANSPOSE A VECTOR
np.array([1,2,3,4,5,6,7]).T
```

Out[26]:

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

In [27]:

```
#TRANSPOSE ROW VECTOR
np.array([[1,2,3,4,5,6,7]]).T
```

Out[27]:

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

In [28]:

```
#FLATTEN MATRIX
matrix.flatten()
```

Out[28]:

```
array([ 1,  2,  3,  4,  5,  6,  6,  7,  8,  6, 54,  7])
```

In [29]:

```
#Return Diagonal Elements
# need a NxN matrix to show result properly
#here we have 4x3 matrix that's why we are getting 3 elements only
matrix.diagonal()
```

Out[29]:

```
array([1, 5, 8])
```

In [30]:

```
c= np.array([1,2,np.nan,4,5])
c # if we take nan value it will automaticly take floating point number
```

Out[30]:

```
array([ 1.,  2., nan,  4.,  5.])
```

In [31]:

```
np.isnan(c) # it will give nan value if there as TRUE other FALSE
```

Out[31]:

```
array([False, False,  True, False, False])
```

In [32]:

```
np.mean(c)
```

Out[32]:

```
nan
```

In [33]:

```
# if any data contains nan value and we need MEAN VALUE we need to use this
a=np.mean(c[~np.isnan(c)])
```

In [34]:

```
a
```

Out[34]:

```
3.0
```

In [35]:

```
matrix
```

Out[35]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 6,  7,  8],
       [ 6, 54,  7]])
```

In [36]:

```
# astype() is to convert any datatype to another
a=matrix.astype('float')
```

In [37]:

```
a
```

Out[37]:

```
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 6.,  7.,  8.],
       [ 6., 54.,  7.]])
```

Important **

In [38]:

```
a[0][0]=np.nan
```

In [39]:

```
a[1][1]=np.inf # infinity value
```

In [40]:

```
a
```

Out[40]:

```
array([[nan,  2.,  3.],
       [ 4., inf,  6.],
       [ 6.,  7.,  8.],
       [ 6., 54.,  7.]])
```

In [41]:

```
np.isinf(a) # checking for a infinity value
```

Out[41]:

```
array([[False, False, False],
       [False,  True, False],
       [False, False, False],
       [False, False, False]])
```

In [42]:

```
np.isnan(a)
```

Out[42]:

```
array([[ True, False, False],
       [False, False, False],
       [False, False, False],
       [False, False, False]])
```

In [44]:

```
flag=np.isinf(a)|np.isnan(a) # pipe fn is to check multiple conditions at a time
```

In [45]:

```
flag
```

Out[45]:

```
array([[ True, False, False],
       [False,  True, False],
       [False, False, False],
       [False, False, False]])
```

In [57]:

```
flag
```

Out[57]:

```
array([[ True, False, False],
       [False,  True, False],
       [False, False, False],
       [False, False, False]])
```

DATA CLEANING

In [46]:

```
a[flag]
```

Out[46]:

```
array([nan, inf])
```

In [47]:

```
a[flag]=0 #replacing flags value to zero
# not a right way to do
```

In [48]:

```
a
```

Out[48]:

```
array([[ 0.,  2.,  3.],
       [ 4.,  0.,  6.],
       [ 6.,  7.,  8.],
       [ 6., 54.,  7.]])
```

In [50]:

```
dict={'A':100,'B':200,'C':300,'D':400}
z1=pd.Series(dict)
z1
```

Out[50]:

```
A    100
B    200
C    300
D    400
dtype: int64
```

In [54]:

```
z2=pd.Series([10,20,30,40],'A B D E'.split())
```

In [55]:

```
'A B C D'.split()
```

Out[55]:

```
['A', 'B', 'C', 'D']
```

In [56]:

```
z2
```

Out[56]:

```
A    10
B    20
D    30
E    40
dtype: int64
```

In [57]:

```
z1+z2
```

Out[57]:

```
A    110.0
B    220.0
C      NaN
D    430.0
E      NaN
dtype: float64
```

Slicing in Series

In [58]:

```
z1
```

Out[58]:

```
A    100
B    200
C    300
D    400
dtype: int64
```

In [59]:

```
z1['A':'C']
```

Out[59]:

```
A    100
B    200
C    300
dtype: int64
```

In [60]:

```
type(z1['A':'C'])
```

Out[60]:

```
pandas.core.series.Series
```

In [61]:

```
z1[['A','C']]
```

Out[61]:

```
A    100
C    300
dtype: int64
```

In [62]:

```
#using default index values
z1[[0,1]]
```

Out[62]:

```
A    100
B    200
dtype: int64
```

Indexing

In [63]:

```
# .iloc does integer based indexing
z1.iloc[2] # need to specify the integer
```

Out[63]:

```
300
```

In [64]:

```
#loc does label based indexing
z1.loc['A'] # here we need to specify the label instead of integer
```

Out[64]:

```
100
```

OPERATIONS

In [65]:

```
z1>100
```

Out[65]:

```
A    False
B     True
C     True
D     True
dtype: bool
```

In [66]:

```
z1[z1>100] #condition
```

Out[66]:

```
B    200
C    300
D    400
dtype: int64
```

In [67]:

```
b=[True,False,True,False]
z1[b]
# here where ever there is true it will return
```

Out[67]:

```
A    100
C    300
dtype: int64
```

BROADCASTING

In [68]:

```
z1+200
```

Out[68]:

```
A    300
B    400
C    500
D    600
dtype: int64
```

Ordering on Series

In [69]:

```
#sort_values()  func to perform sorting
z1.sort_values(ascending=False)
```

Out[69]:

```
D    400
C    300
B    200
A    100
dtype: int64
```

In [70]:

```
z1.sort_values(ascending=False).index[1]
```

Out[70]:

```
'C'
```

In [71]:

```
z1 # no change in data
```

Out[71]:

```
A    100
B    200
C    300
D    400
dtype: int64
```

In [72]:

```
# if we want to change the data permanently use
# inplace=True
z1.sort_values(ascending=False,inplace=True)
```

In [73]:

```
z1
```

Out[73]:

```
D    400
C    300
B    200
A    100
dtype: int64
```

aggregation on series

In [74]:

```
z1.mean()
```

Out[74]:

```
250.0
```

In [75]:

```
z1.sum()
```

Out[75]:

```
1000
```

In [76]:

```
z1.max()
```

Out[76]:

```
400
```

In [77]:

```
z1.min()
```

Out[77]:

```
100
```

In [79]:

```
z1.idxmax()
```

Out[79]:

```
'D'
```

Series with two Column

In [80]:

```
data_1={'Column1':pd.Series([2,3,4,5],['a b c d'.split()]),
       'Column2':pd.Series([20,30,40,50],['A B C D'.split()])}
```

In [81]:

```
data_1
```

Out[81]:

```
{'Column1': a    2
 b    3
 c    4
 d    5
 dtype: int64,
 'Column2': A    20
 B    30
 C    40
 D    50
 dtype: int64}
```

Practical 3 - Data Frame

7 June 2022

In [82]:

```
rand_mat=np.random.randn(5,4)
rand_mat
```

Out[82]:

```
array([[-1.74540593,  0.07421877,  0.33358931, -0.47758962],
       [ 0.28027289, -0.82350207,  0.39630927, -0.8463551 ],
       [ 1.15484762, -2.49408557,  0.21734834, -0.85388345],
       [-0.32731119, -1.51705037,  0.05424037,  1.12742118],
       [-1.67727736,  0.07928247,  0.45841675, -0.63848852]])
```

In [83]:

```
df = pd.DataFrame(rand_mat,index='A B C D E'.split(),
columns = 'W X Y Z'.split())
```

In [84]:

```
df
```

Out[84]:

	W	X	Y	Z
A	-1.745406	0.074219	0.333589	-0.477590
B	0.280273	-0.823502	0.396309	-0.846355
C	1.154848	-2.494086	0.217348	-0.853883
D	-0.327311	-1.517050	0.054240	1.127421
E	-1.677277	0.079282	0.458417	-0.638489

Selection and Indexing

In [85]:

```
df['W']
```

Out[85]:

```
A   -1.745406
B    0.280273
C    1.154848
D   -0.327311
E   -1.677277
Name: W, dtype: float64
```

In [86]:

```
df[['W','Z']]
```

Out[86]:

	W	Z
A	-1.745406	-0.477590
B	0.280273	-0.846355
C	1.154848	-0.853883
D	-0.327311	1.127421
E	-1.677277	-0.638489

In [88]:

```
df[['W','X']]
```

Out[88]:

	W	X
A	-1.745406	0.074219
B	0.280273	-0.823502
C	1.154848	-2.494086
D	-0.327311	-1.517050
E	-1.677277	0.079282

In [89]:

```
df.W #NOT RECOMENDDED
```

Out[89]:

```
A   -1.745406
B    0.280273
C    1.154848
D   -0.327311
E   -1.677277
Name: W, dtype: float64
```

In [91]:

```
df.iloc[2]
```

Out[91]:

```
W    1.154848
X   -2.494086
Y    0.217348
Z   -0.853883
Name: C, dtype: float64
```

ADDING A ROW

In [92]:

```
df.loc['F']=df.loc['A']+df.loc['B']
```

In [93]:

```
df
```

Out[93]:

	W	X	Y	Z
A	-1.745406	0.074219	0.333589	-0.477590
B	0.280273	-0.823502	0.396309	-0.846355
C	1.154848	-2.494086	0.217348	-0.853883
D	-0.327311	-1.517050	0.054240	1.127421
E	-1.677277	0.079282	0.458417	-0.638489
F	-1.465133	-0.749283	0.729899	-1.323945

In [94]:

```
# if we want to reset the indexes as Default
df.reset_index()
```

Out[94]:

	index	W	X	Y	Z
0	A	-1.745406	0.074219	0.333589	-0.477590
1	B	0.280273	-0.823502	0.396309	-0.846355
2	C	1.154848	-2.494086	0.217348	-0.853883
3	D	-0.327311	-1.517050	0.054240	1.127421
4	E	-1.677277	0.079282	0.458417	-0.638489
5	F	-1.465133	-0.749283	0.729899	-1.323945

In [95]:

```
#for changing the index names
newindex = 'DEL UP UK TN AP KL'.split()
```

In [96]:

```
df['States']=newindex # added a column
```

In [97]:

```
df
```

Out[97]:

	W	X	Y	Z	States
A	-1.745406	0.074219	0.333589	-0.477590	DEL
B	0.280273	-0.823502	0.396309	-0.846355	UP
C	1.154848	-2.494086	0.217348	-0.853883	UK
D	-0.327311	-1.517050	0.054240	1.127421	TN
E	-1.677277	0.079282	0.458417	-0.638489	AP
F	-1.465133	-0.749283	0.729899	-1.323945	KL

In [98]:

```
# now we have to replace abcdef to states values
df.set_index('States',inplace=True)
```

In [99]:

```
df
```

Out[99]:

States	W	X	Y	Z
DEL	-1.745406	0.074219	0.333589	-0.477590
UP	0.280273	-0.823502	0.396309	-0.846355
UK	1.154848	-2.494086	0.217348	-0.853883
TN	-0.327311	-1.517050	0.054240	1.127421
AP	-1.677277	0.079282	0.458417	-0.638489
KL	-1.465133	-0.749283	0.729899	-1.323945

Multi -Index and Index Hierarchy

In [103]:

```
# Index levels
outside=['North','North','North','South','South','South']
inside=newindex
```

In [104]:

```
hier_index=list(zip(outside,inside))
```

In [105]:

```
hier_index
```

Out[105]:

```
[('North', 'DEL'),
 ('North', 'UP'),
 ('North', 'UK'),
 ('South', 'TN'),
 ('South', 'AP'),
 ('South', 'KL')]
```

In [106]:

```
hier_index=pd.MultiIndex.from_tuples(hier_index)
```

In [107]:

```
hier_index
```

Out[107]:

```
MultiIndex([('North', 'DEL'),
            ('North',  'UP'),
            ('North',  'UK'),
            ('South',  'TN'),
            ('South',  'AP'),
            ('South',  'KL')],
           )
```

In [108]:

```
df = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B'])
df
```

Out[108]:

		A	B
North	DEL	-1.558007	0.849948
	UP	1.059091	0.575323
	UK	1.727948	0.269926
South	TN	0.348608	1.617945
	AP	-0.333769	-0.129678
	KL	-0.484702	-0.267028

In [109]:

```
# now we can fetch the data according to hierarcy
df.loc['North']
```

Out[109]:

	A	B
DEL	-1.558007	0.849948
UP	1.059091	0.575323
UK	1.727948	0.269926

In [110]:

```
df.loc['North'].loc['DEL']
```

Out[110]:

```
A   -1.558007
B    0.849948
Name: DEL, dtype: float64
```

In [111]:

```
#index not been assigned to any names
df.index.names
```

Out[111]:

```
FrozenList([None, None])
```

In [112]:

```
df.index.names=['Region','States']
```

In [113]:

```
#Return Cross-section from the series/dataframe
df.xs('North')
```

Out[113]:

	A	B
States		
DEL	-1.558007	0.849948
UP	1.059091	0.575323
UK	1.727948	0.269926

In [115]:

```
outside=['North','North','North','South','South','South']
inside=[1,2,3,1,2,3]
hier_index=list(zip(outside,inside))
hier_index=pd.MultiIndex.from_tuples(hier_index)
```

In [116]:

```
df=pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B'])
df.index.names=['Region','Num']
df
```

Out[116]:

		A	B
Region	**Num**		
	1	0.751418	1.308372
North	**2**	0.778414	-0.481113
	3	1.521295	0.002127
	1	-0.917190	-2.664525
South	**2**	-0.025975	-1.221268
	3	1.980576	-1.036305

In [118]:

```
df.xs(['North',1])
```

```
C:\Users\studentadmin\AppData\Local\Temp\ipykernel_9256\453744059.py:1: Futu
reWarning: Passing lists as key for xs is deprecated and will be removed in
a future version. Pass key as a tuple instead.
  df.xs(['North',1])
```

Out[118]:

```
A    0.751418
B    1.308372
Name: (North, 1), dtype: float64
```

In [119]:

```
df.xs(1,level='Num')
```

Out[119]:

	A	B
Region		
North	0.751418	1.308372
South	-0.917190	-2.664525

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Practical 4 - Matplotlib and Operations - Pandas

8 June 2022

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
df1=pd.read_csv('D:\\ML\\df31.csv',index_col=0)
df2=pd.read_csv('D:\\ML\\df32.csv')
```

In [3]:
```
df1
```

Out[3]:

	A	B	C	D
2000-01-01	1.339091	-0.163643	-0.646443	1.041233
2000-01-02	-0.774984	0.137034	-0.882716	-2.253382
2000-01-03	-0.921037	-0.482943	-0.417100	0.478638
2000-01-04	-1.738808	-0.072973	0.056517	0.015085
2000-01-05	-0.905980	1.778576	0.381918	0.291436
...	...	...	...	...
2002-09-22	1.013897	-0.288680	-0.342295	-0.638537
2002-09-23	-0.642659	-0.104725	-0.631829	-0.909483
2002-09-24	0.370136	0.233219	0.535897	-1.552605
2002-09-25	0.183339	1.285783	-1.052593	-2.565844
2002-09-26	0.775133	-0.850374	0.486728	-1.053427

1000 rows × 4 columns

In [4]:
```
df2
```

Out[4]:

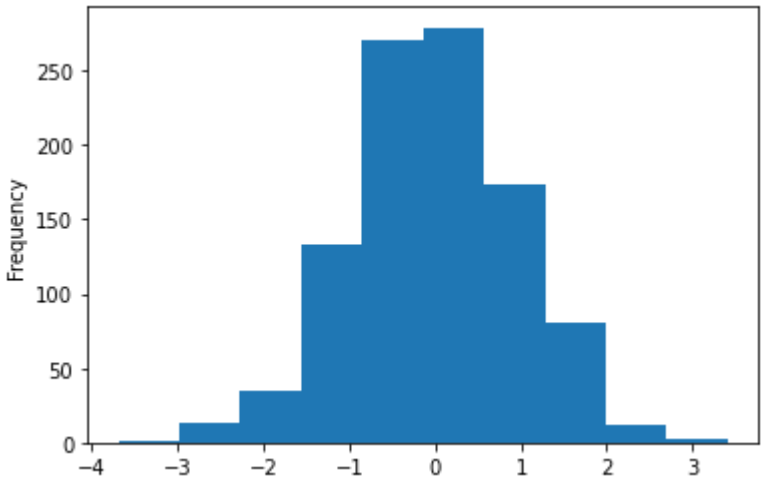
	a	b	c	d	e
0	0.039762	0.218517	0.103423	0.957904	x
1	0.937288	0.041567	0.899125	0.977680	y
2	0.780504	0.008948	0.557808	0.797510	x
3	0.672717	0.247870	0.264071	0.444358	z
4	0.053829	0.520124	0.552264	0.190008	y
5	0.286043	0.593465	0.907307	0.637898	x
6	0.430436	0.166230	0.469383	0.497701	z
7	0.312296	0.502823	0.806609	0.850519	z
8	0.187765	0.997075	0.895955	0.530390	x
9	0.908162	0.232726	0.414138	0.432007	y

PLOT

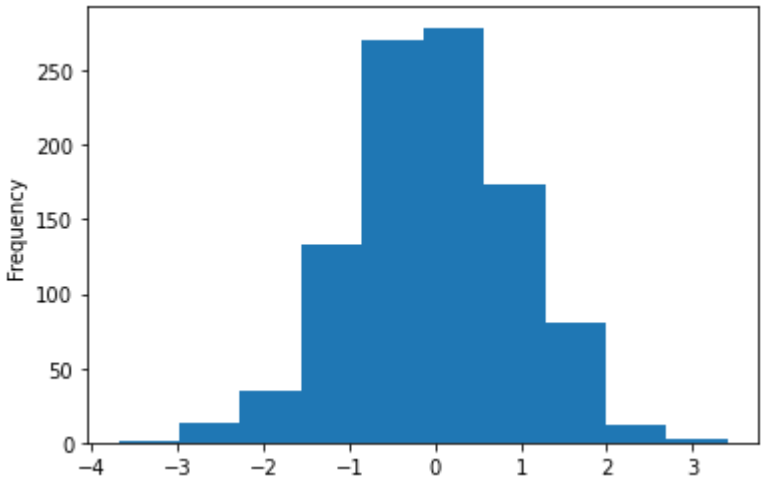
Histograms

In [5]:
```
df1['A'].plot.hist() #if we dont need
#this <AxesSubplot:ylabel='Frequency'>
#then add semicolon at end
```

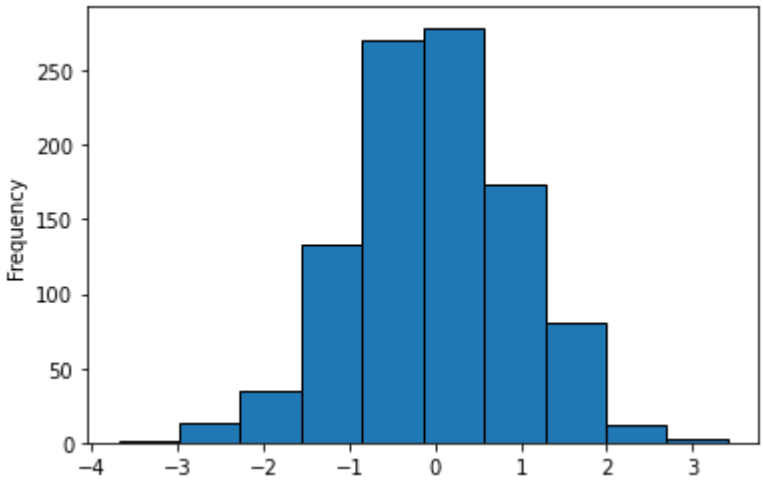
Out[5]:
```
<AxesSubplot:ylabel='Frequency'>
```



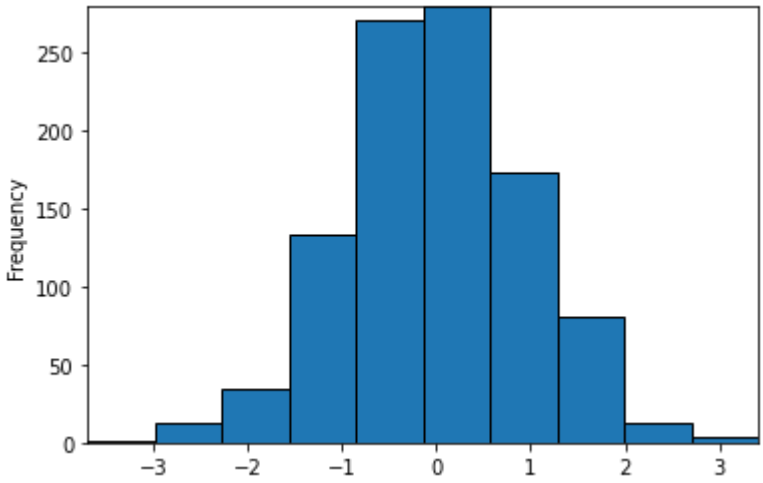
In [6]:
```
df1['A'].plot.hist();
```



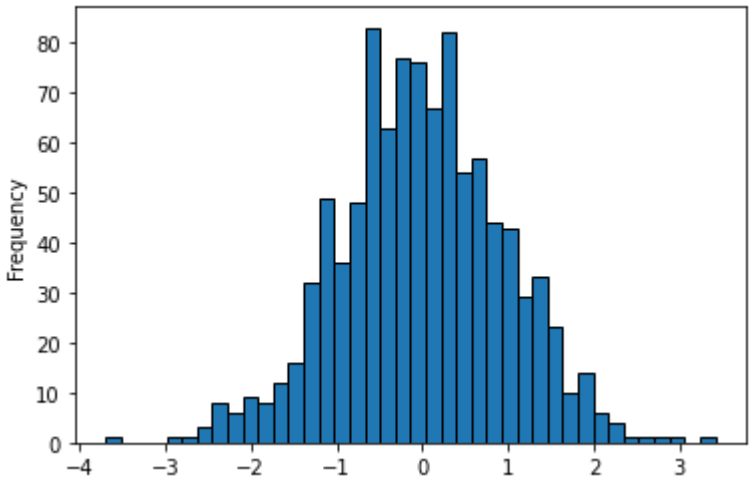
In [7]:
```
df1['A'].plot.hist(edgecolor='k').autoscale(enable=True,axis='both') # k is for bl
```



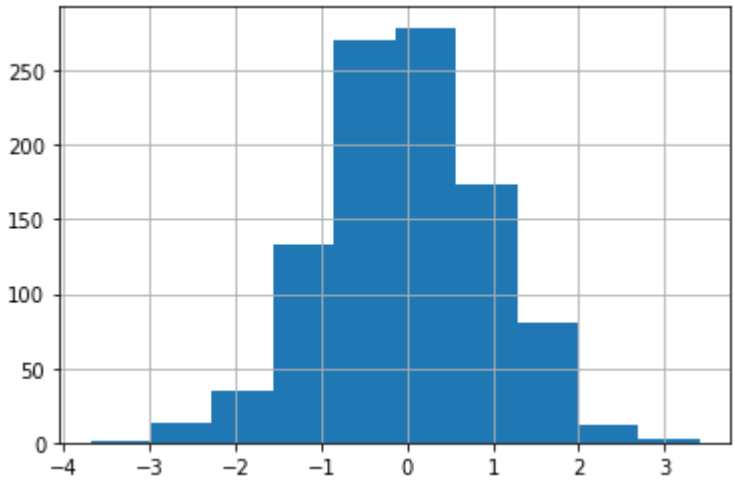
In [8]:
```
df1['A'].plot.hist(edgecolor='k').autoscale(enable=True,axis='both',tight=True)
# k is for black
#tight is for removing extra space
```



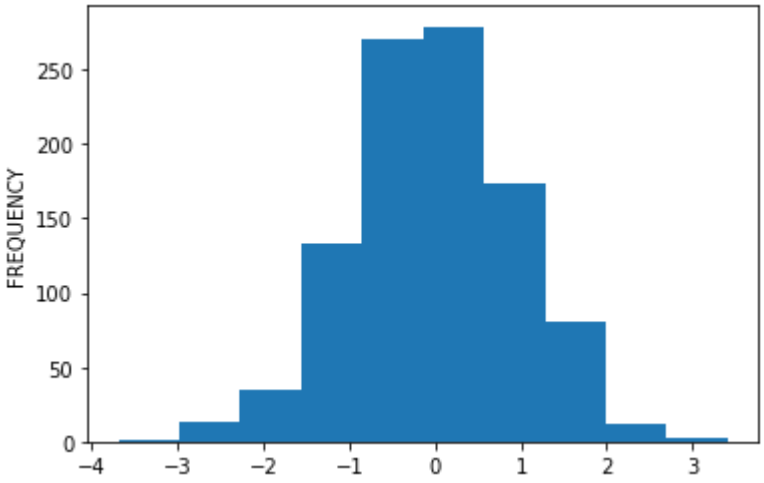
In [9]:
```
df1['A'].plot.hist(bins=40,edgecolor='k').autoscale(enable=True,axis='both')
#we can set bins also
```



In [10]:
```
df1['A'].hist();
```

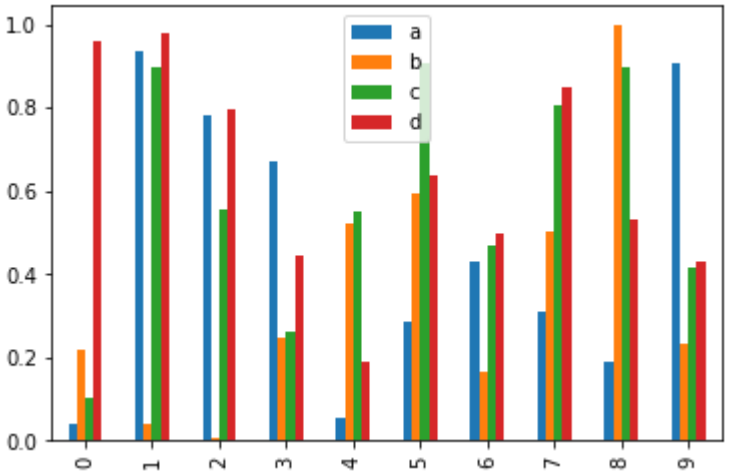


In [11]:
```
df1['A'].hist(grid=False).set_ylabel("FREQUENCY"); #remove the grid and add y labe
```

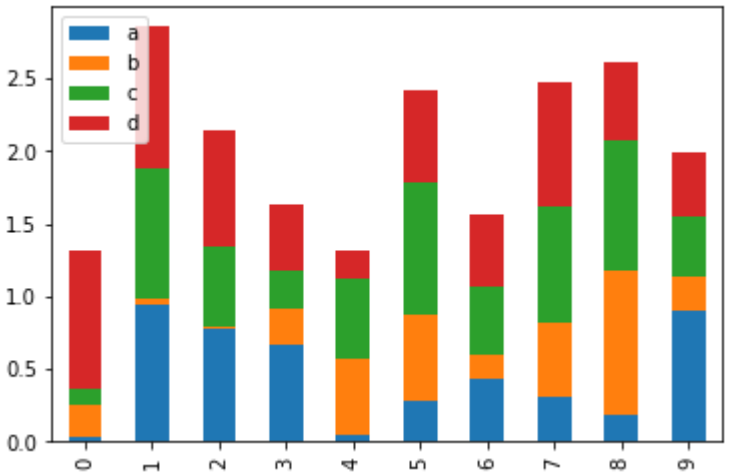


Bar Plots

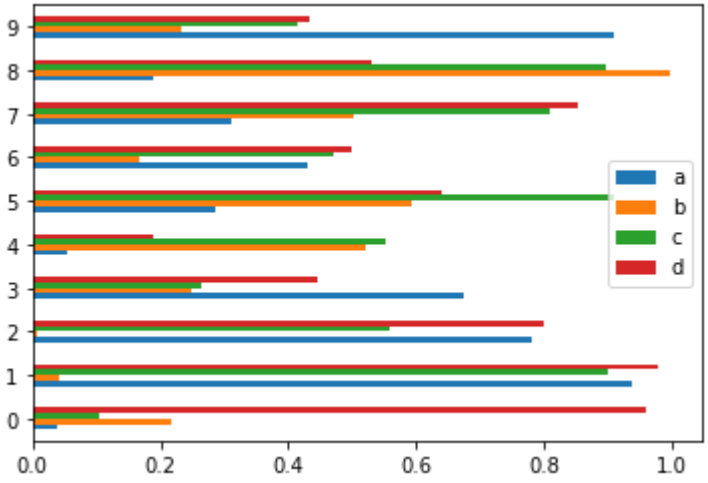
In [12]:
```
df2.plot.bar();
```



In [13]:
```
df2.plot.bar(stacked=True);
```

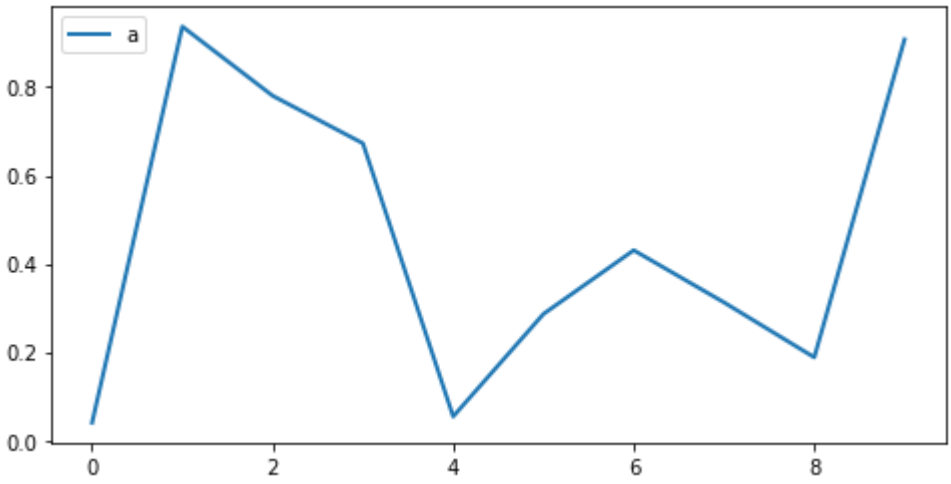


In [14]:
```
df2.plot.barh(); # barh for horizontal bar plot
```

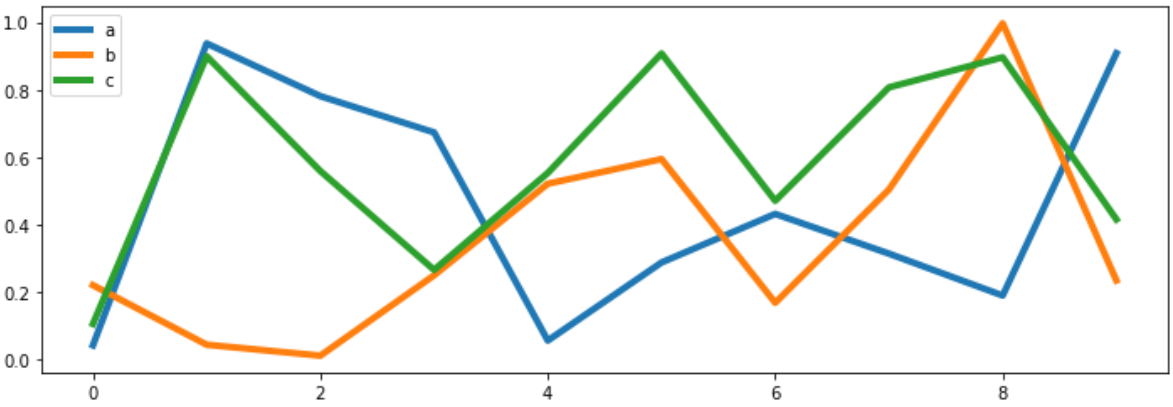


Line Plot

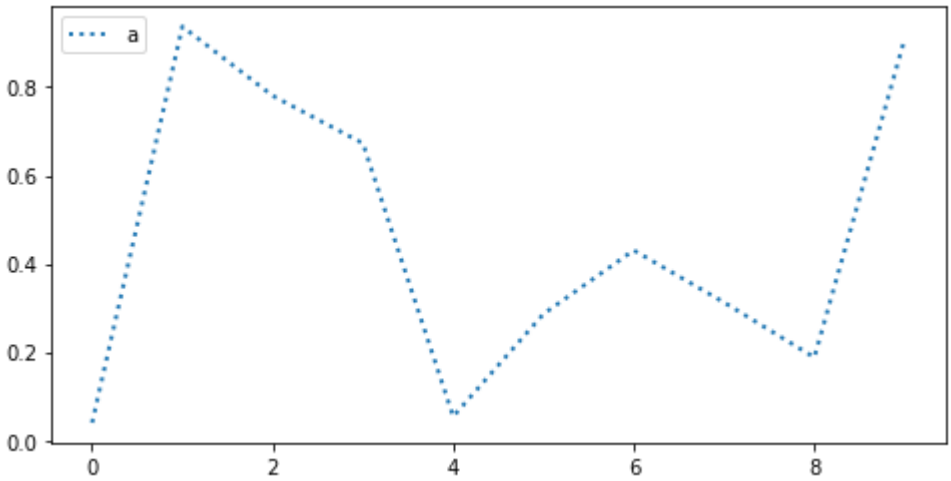
In [15]:
```
df2.plot.line(y='a',figsize=(8,4),lw=2); #figsize is the size of figure
#lw line width
```



In [16]:
```
df2.plot.line(y=['a','b','c'],figsize=(12,4),lw=4); #figsize is the size of figure
```

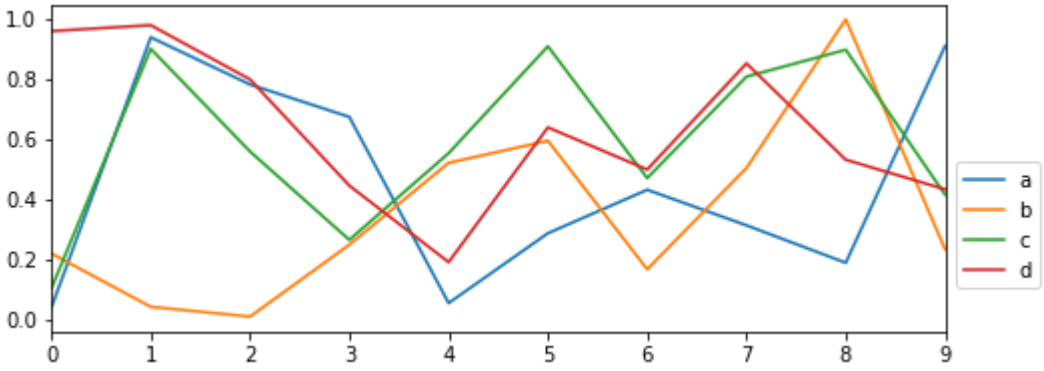


In [17]:
```
df2.plot.line(y='a',figsize=(8,4),lw=2,ls=':'); #ls can be change the dotted line
```



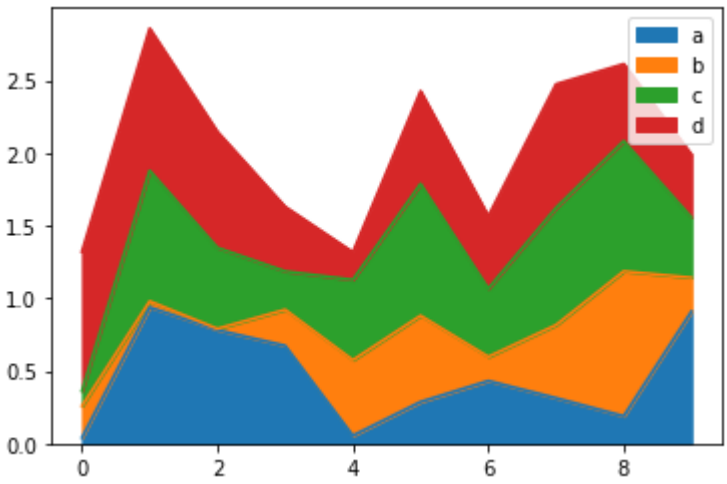
In [18]:
```
ax=df2.plot(figsize=(8,3))
ax.autoscale(axis='x',tight=True)
ax.legend(loc=3,bbox_to_anchor=(1.0,0.1))
```

Out[18]:
```
<matplotlib.legend.Legend at 0x137bc145130>
```

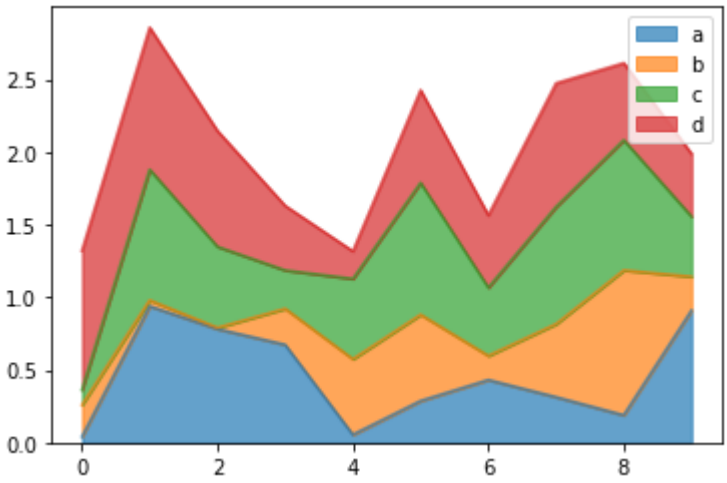


Area plot

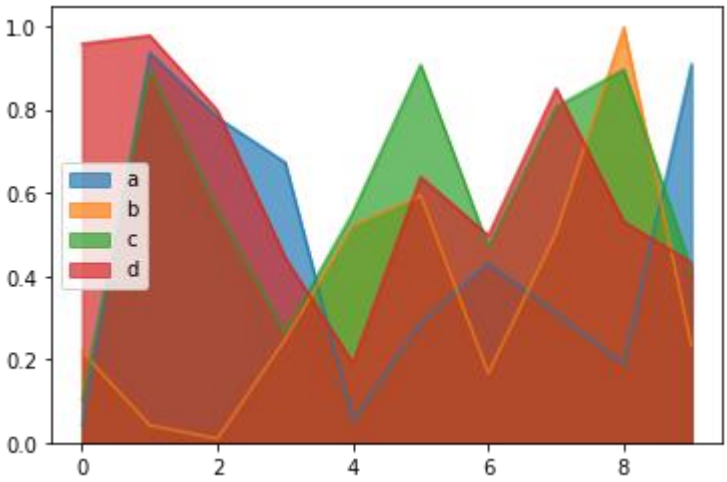
In [19]:
```
df2.plot.area();
```



In [20]:
```
df2.plot.area(alpha=0.69); #alpha is for channging the transparency
```

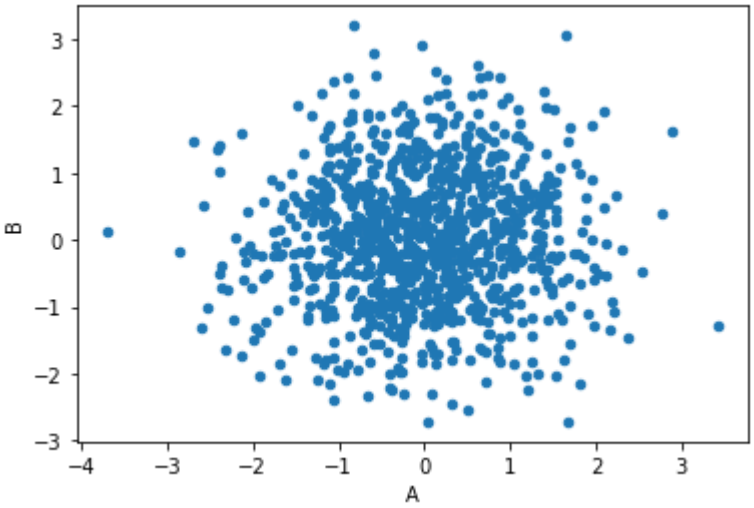


In [21]:
```
df2.plot.area(stacked=False,alpha=0.69);
```

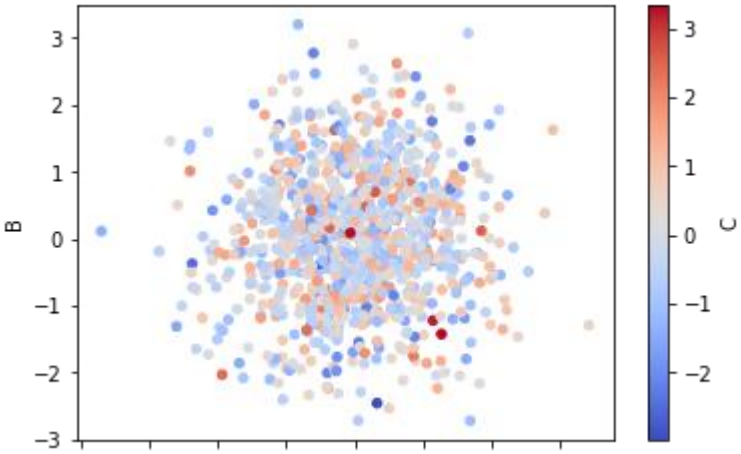


scatter plot

In [22]:
```
df1.plot.scatter(x='A',y='B');
```



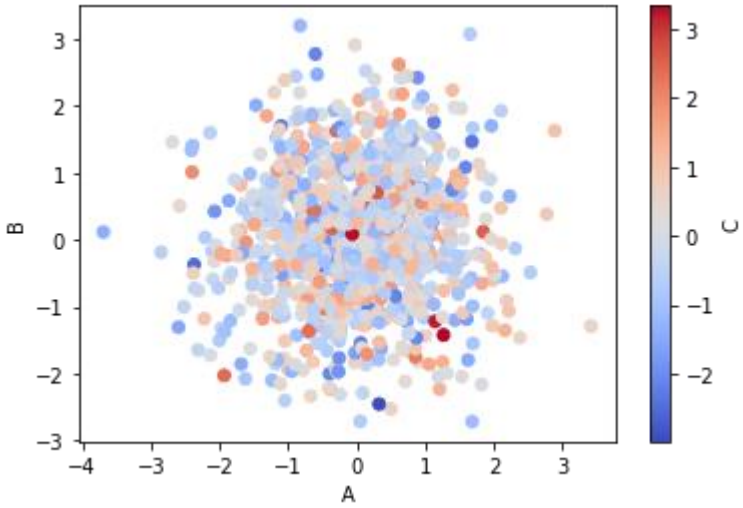
In [23]:
```
df1.plot.scatter(x='A',y='B',c='C',cmap='coolwarm');
```



matplotlib

In [24]:
```
import matplotlib.pyplot as plt
plt.scatter(df1['A'],df1['B'],c=df1['C'],cmap='coolwarm')
```

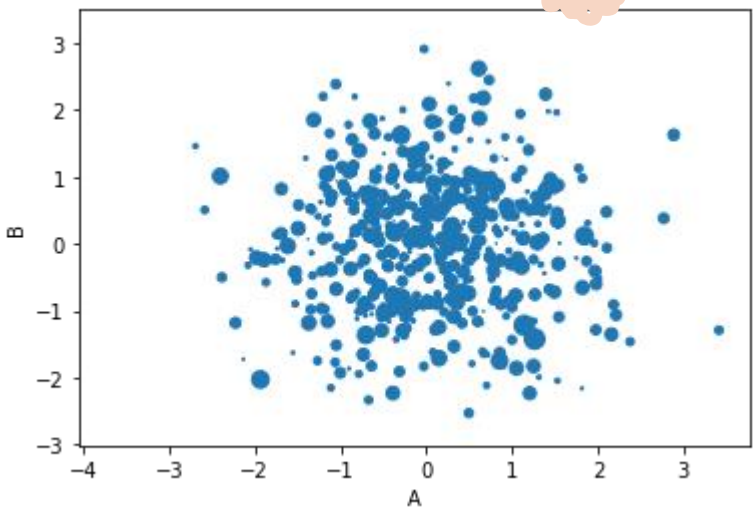
```
plt.colorbar().set_label('C')
plt.xlabel('A')
plt.ylabel('B')
plt.show()
```



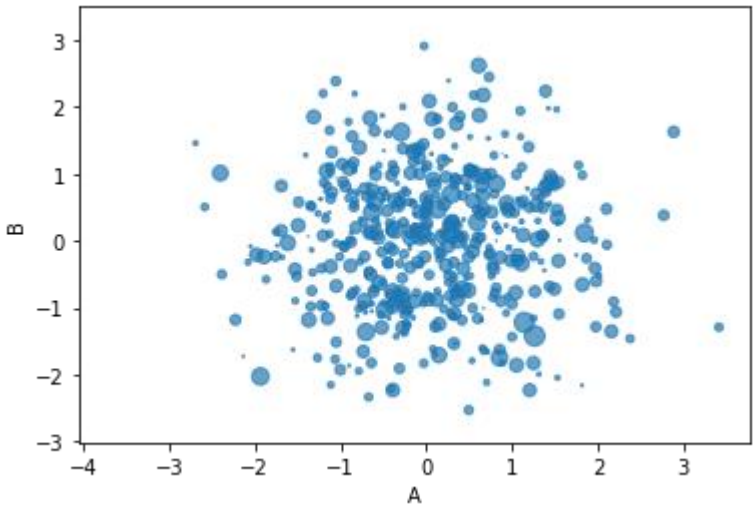
scatter plot with size marker

In [25]:
```
df1.plot.scatter(x='A',y='B',s=df1['C']*30);
```

```
C:\Users\divya\anaconda3\lib\site-packages\matplotlib\collections.py:982: RuntimeW
arning: invalid value encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```



In [26]:
```
df1.plot.scatter(x='A',y='B',s=df1['C']*30,alpha=0.69);
```

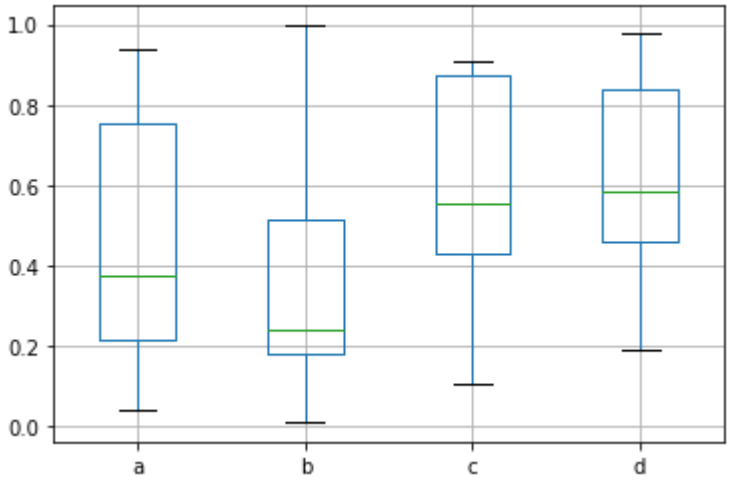


Box Plot

In [27]:
```
df2.describe
```

Out[27]:
```
<bound method NDFrame.describe of          a         b         c         d  e
0  0.039762  0.218517  0.103423  0.957904  x
1  0.937288  0.041567  0.899125  0.977680  y
2  0.780504  0.008948  0.557808  0.797510  x
3  0.672717  0.247870  0.264071  0.444358  z
4  0.053829  0.520124  0.552264  0.190008  y
5  0.286043  0.593465  0.907307  0.637898  x
6  0.430436  0.166230  0.469383  0.497701  z
7  0.312296  0.502823  0.806609  0.850519  z
8  0.187765  0.997075  0.895955  0.530390  x
9  0.908162  0.232726  0.414138  0.432007  y>
```

In [28]:
```
df2.boxplot();
```



Practical 5 - Data cleaning

8 June 2022

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

the data

In [2]:
```
titanic=pd.read_csv('D:\\ML\\titanic.csv')
```

In [3]:
```
titanic.head(10)
```

Out[3]:

	PassengerId	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embark
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	3	Wil es, Mrs. Jai es (E len Ne ds)	female	47.0	1	0	363272	7.0000	NaN	
2	894	2	M les, Mr. Tho as Fra cis	male	62.0	0	0	240276	9.6875	NaN	
3	895	3	Wirz, Mr. Al ert	male	27.0	0	0	315154	8.6625	NaN	
4	896	3	Hirvonen, Mrs. Alexa der (Helg a E Lindq ist)	female	22.0	1	1	3101298	12.2875	NaN	
5	897	3	Svens on, Mr. Jo an Ce vin	male	14.0	0	0	7538	9.2250	NaN	
6	898	3	Conn lly, M ss. Kate	female	30.0	0	0	330972	7.6292	NaN	
7	899	2	Cald ell, Mr. Albert Francis	male	26.0	1	1	248738	29.0000	NaN	
8	900	3	Abrahim, Mrs. Joseph (Sophie Halaut Easu)	female	18.0	0	0	2657	7.2292	NaN	
9	901	3	Davies, Mr. John Samuel	male	21.0	2	0	48 A/4 71	24.1500	NaN	

In [4]:
```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Gender       418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

Missing data

In [5]:
```
titanic.isnull()
```

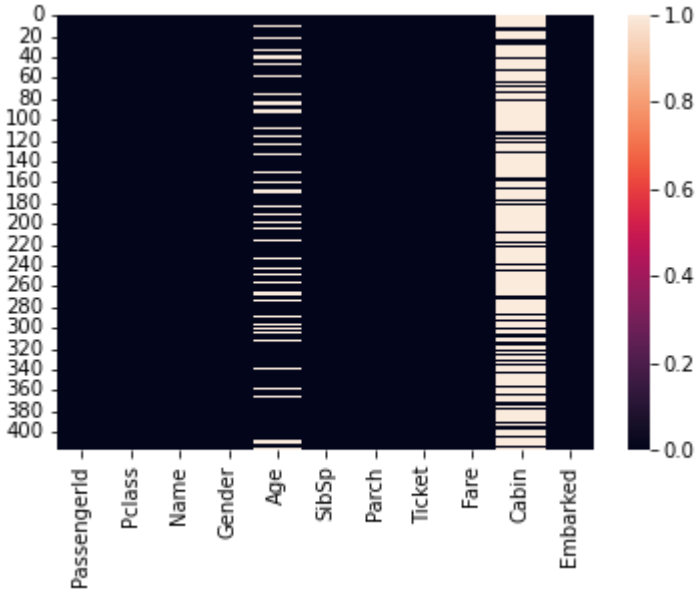
Out[5]:

	PassengerId	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	True	False
2	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	True	False
4	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...
413	False	False	False	False	True	False	False	False	False	True	False
414	False	False	False	False	False	False	False	False	False	False	False
415	False	False	False	False	False	False	False	False	False	True	False
416	False	False	False	False	True	False	False	False	False	True	False
417	False	False	False	False	True	False	False	False	False	True	False

418 rows × 11 columns

In [6]:
```
sns.heatmap(titanic.isnull())
```

Out[6]:
```
<AxesSubplot:>
```

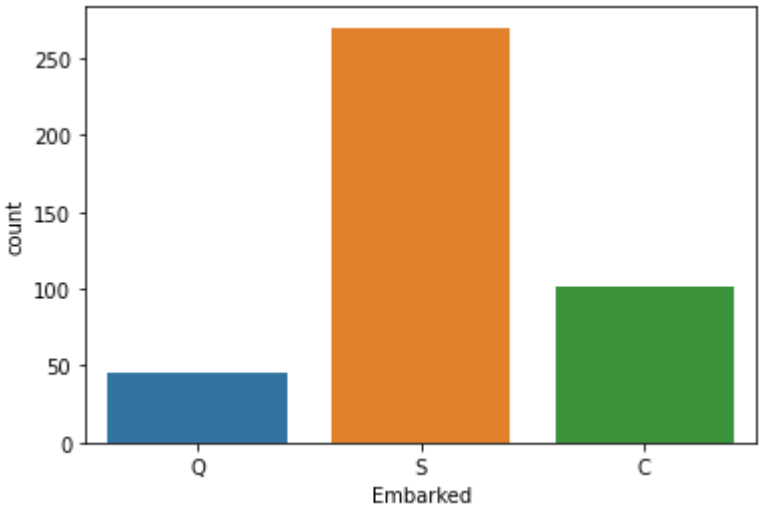


now we have to replace the age null value to some form of imputation

and drop the cabin coloumn

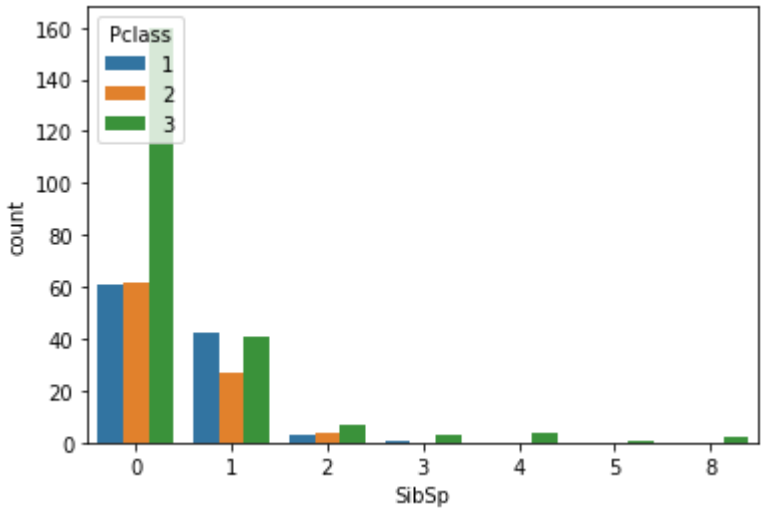
In [7]:
```
sns.countplot(x='Embarked',data=titanic)
```

Out[7]:
```
<AxesSubplot:xlabel='Embarked',  ylabel='count'>
```



From passenger list most of the passengers are in Embarked "S" class

In [8]:
```
sns.countplot(x='SibSp',hue='Pclass',data=titanic);
```



Color diff is the passengers class and x axis is siblings spouse

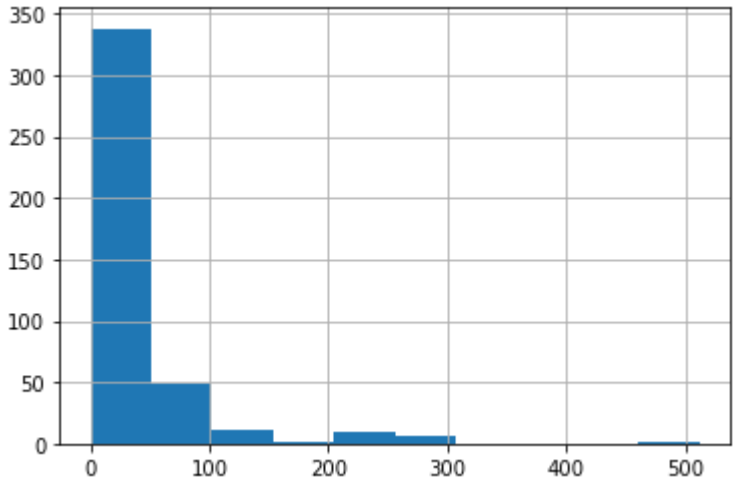
maximum passenegers are not dependent or zero dependance

In [9]:
```
titanic['Fare']
```

Out[9]:
```
0        7.8292
1        7.0000
2        9.6875
3        8.6625
4       12.2875
         ...
413      8.0500
414    108.9000
415      7.2500
416      8.0500
417     22.3583
Name: Fare, Length: 418, dtype: float64
```

In [10]:
```
titanic['Fare'].hist()
```

Out[10]:
```
<AxesSubplot:>
```



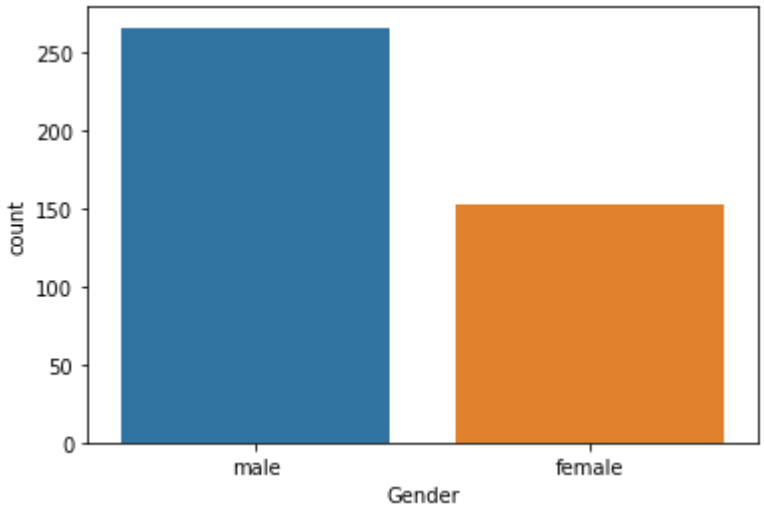
Max count comes in range 0-50

In [11]:
```
titanic['Gender']
```

Out[11]:
```
0        male
1      female
2        male
3        male
4      female
        ...
413      male
414    female
415      male
416      male
417      male
Name: Gender, Length: 418, dtype: object
```

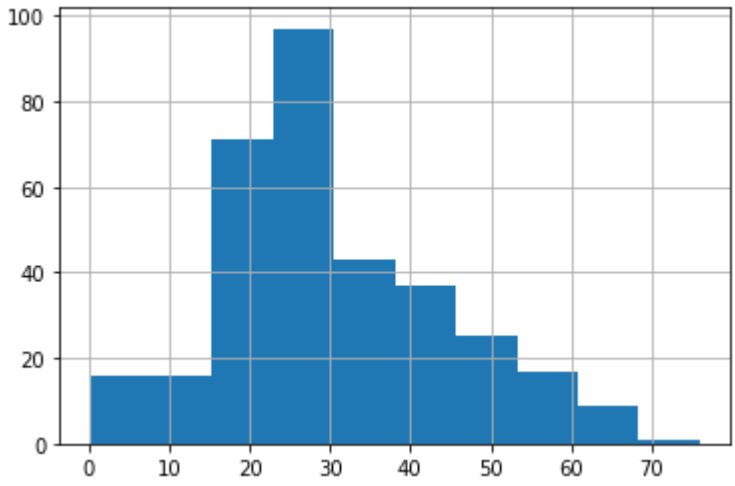
In [12]:
```
sns.countplot(x='Gender',data=titanic)
```

Out[12]:
```
<AxesSubplot:xlabel='Gender',  ylabel='count'>
```



Males are more than Females

In [13]:
```
titanic['Age'].hist();
```



most of the Passengers are of age between 20-30

Data Cleaning

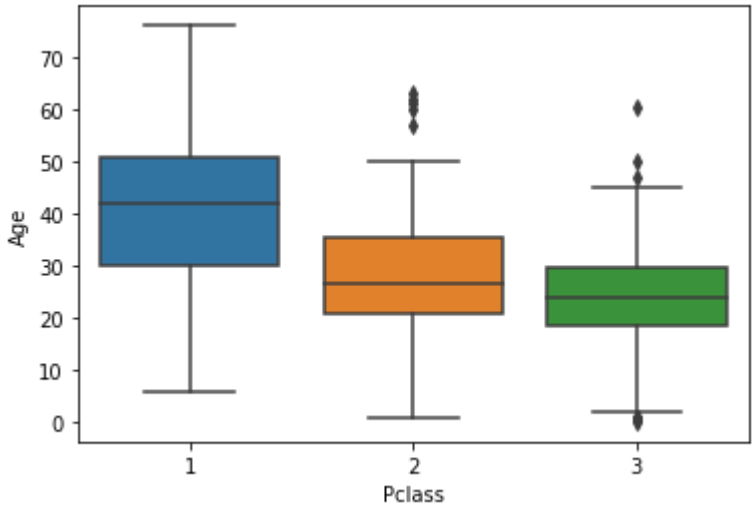
In [14]:
```
titanic.describe()
```

Out[14]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200

In [15]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x='Pclass',y='Age',data=titanic)
```

Out[15]:
```
<AxesSubplot:xlabel='Pclass',  ylabel='Age'>
```



In [16]:
```
titanic.groupby('Pclass').mean()['Age']
```

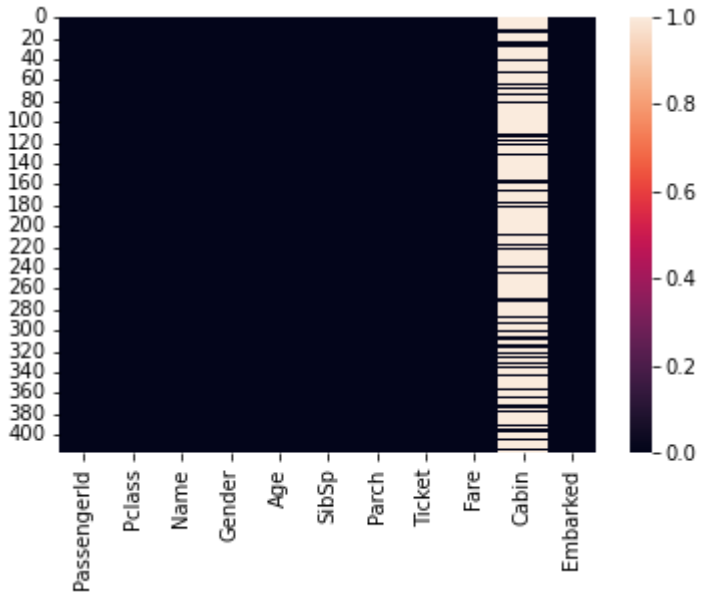
Out[16]:
```
Pclass
1    40.918367
2    28.777500
3    24.027945
Name: Age, dtype: float64
```

In [17]:
```
def  impute_age(cols):
    Age=cols[0]
    Pclass=cols[1]
    if pd.isnull(Age):
        if Pclass == 1:
            return 41
        elif Pclass ==2:
            return 29
        else:
            return 24
    else:
        return Age
```

In [18]:
```
titanic['Age']=titanic[['Age','Pclass']].apply(impute_age,axis=1)
```

In [19]:
```
sns.heatmap(titanic.isnull())
```

Out[19]: <AxesSubplot:>



Age has no missing value

In [20]:
```
titanic['Age']
```

Out[20]:
```
0      34.5
1      47.0
2      62.0
3      27.0
4      22.0
       ...
413    24.0
414    39.0
415    38.5
416    24.0
417    24.0
Name: Age, Length: 418, dtype: float64
```

In [21]:
```
titanic.drop('Cabin',axis=1,inplace=True)
```

In [22]:
```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Gender       418 non-null    object 
 4   Age          418 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(4)
memory usage: 32.8+ KB
```

Practical 6 - Data Cleaning - Youtube Dataset

9 June 2022

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [2]:
```
ytube=pd.read_csv('D:\\ML\\top-5000-youtube-channels.csv')
```

In [3]:
```
ytube
```

Out[3]:

	Rank	Grade	Channel name	Video Uploads	Subscribers	Video views
0	1st	A++	Zee TV	82757	18752951	20869786591
1	2nd	A++	T-Series	12661	61196302	47548839843
2	3rd	A++	Cocomelon - Nursery Rhymes	373	19238251	9793305082
3	4th	A++	SET India	27323	31180559	22675948293
4	5th	A++	WWE	36756	32852346	26273668433
...	...	...	...	...	...	...
4995	4,996th	B+	Uras Benlioğlu	706	2072942	441202795
4996	4,997th	B+	HI-TECH MUSIC LTD	797	1055091	377331722
4997	4,998th	B+	Mastersaint	110	3265735	311758426
4998	4,999th	B+	Bruce McIntosh	3475	32990	14563764
4999	5,000th	B+	SehatAQUA	254	21172	73312511

5000 rows × 6 columns

In [4]:
```
ytube.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Rank           5000 non-null   object
 1   Grade          5000 non-null   object
 2   Channel name   5000 non-null   object
 3   Video Uploads  5000 non-null   object
 4   Subscribers    5000 non-null   object
 5   Video views    5000 non-null   int64 
dtypes: int64(1), object(5)
memory usage: 234.5+ KB
```

In [5]:
```
ytube['Rank']=ytube['Rank'].str[0:-2].str.replace(',','').astype('int')
```

In [6]:
```
ytube
```

Out[6]:

	Rank	Grade	Channel name	Video Uploads	Subscribers	Video views
0	1	A++	Zee TV	82757	18752951	20869786591
1	2	A++	T-Series	12661	61196302	47548839843
2	3	A++	Cocomelon - Nursery Rhymes	373	19238251	9793305082
3	4	A++	SET India	27323	31180559	22675948293
4	5	A++	WWE	36756	32852346	26273668433
...	...	...	...	...	...	...
4995	4996	B+	Uras Benlioğlu	706	2072942	441202795
4996	4997	B+	HI-TECH MUSIC LTD	797	1055091	377331722
4997	4998	B+	Mastersaint	110	3265735	311758426
4998	4999	B+	Bruce McIntosh	3475	32990	14563764
4999	5000	B+	SehatAQUA	254	21172	73312511

5000 rows × 6 columns

In [7]:
```
ytube.dropna()
```

Out[7]:

	Rank	Grade	Channel name	Video Uploads	Subscribers	Video views
0	1	A++	Zee TV	82757	18752951	20869786591
1	2	A++	T-Series	12661	61196302	47548839843
2	3	A++	Cocomelon - Nursery Rhymes	373	19238251	9793305082
3	4	A++	SET India	27323	31180559	22675948293
4	5	A++	WWE	36756	32852346	26273668433
...	...	...	...	...	...	...
4995	4996	B+	Uras Benlioğlu	706	2072942	441202795
4996	4997	B+	HI-TECH MUSIC LTD	797	1055091	377331722
4997	4998	B+	Mastersaint	110	3265735	311758426
4998	4999	B+	Bruce McIntosh	3475	32990	14563764
4999	5000	B+	SehatAQUA	254	21172	73312511

5000 rows × 6 columns

In [8]:
```
ytube.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Rank           5000 non-null   int32 
 1   Grade          5000 non-null   object
 2   Channel name   5000 non-null   object
 3   Video Uploads  5000 non-null   object
 4   Subscribers    5000 non-null   object
 5   Video views    5000 non-null   int64 
dtypes: int32(1), int64(1), object(4)
memory usage: 215.0+ KB
```

In [9]:
```
#we need to remove the - from the data
mask1=ytube[ytube['Subscribers'].str.contains('-')].index
ytube.drop(labels=mask1,axis=0,inplace=True)
```

In [10]:
```
#we have to convert the object datatype as INTEGER !
ytube['Subscribers']=ytube['Subscribers'].astype('int')
```

In [11]:
```
mask1
```

Out[11]:
```
Int64Index([  17,  108,  115,  142,  143,  152,  156,  175,  180,  189,
            ...
            4892, 4893, 4895, 4912, 4936, 4941, 4948, 4956, 4961, 4990],
           dtype='int64',  length=387)
```

In [12]:
```
ytube.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4613 entries, 0 to 4999
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Rank           4613 non-null   int32 
 1   Grade          4613 non-null   object
 2   Channel name   4613 non-null   object
 3   Video Uploads  4613 non-null   object
 4   Subscribers    4613 non-null   int32 
 5   Video views    4613 non-null   int64 
dtypes: int32(2), int64(1), object(3)
memory usage: 216.2+ KB
```

In [13]:
```
mask2=ytube[ytube['Video Uploads'].str.contains('-')].index
ytube.drop(labels=mask2,axis=0,inplace=True)
```

In [14]:
```
ytube['Video  Uploads']=ytube['Video  Uploads'].astype('int')
```

In [15]:
```
mask2
```

Out[15]:
```
Int64Index([2323,  3072,  4898], dtype='int64')
```

In [16]:
```
ytube.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4610 entries, 0 to 4999
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Rank           4610 non-null   int32 
 1   Grade          4610 non-null   object
 2   Channel name   4610 non-null   object
 3   Video Uploads  4610 non-null   int32 
 4   Subscribers    4610 non-null   int32 
 5   Video views    4610 non-null   int64 
dtypes: int32(3), int64(1), object(2)
memory usage: 198.1+ KB
```

In [17]:
```
ytube['Grade'].unique()
```

Out[17]:
```
array(['A++ ', 'A+ ', 'A ', 'A- ', 'B+ '], dtype=object)
```

In [18]:
```
channel_map={'A++ ':5,'A+ ':4,'A ':3,'A- ':2,'B+ ':1}
```

In [19]:
```
ytube['Grade']=ytube['Grade'].map(channel_map)
```

In [20]:
```
ytube
```

Out[20]:

	Rank	Grade	Channel name	Video Uploads	Subscribers	Video views
0	1	5	Zee TV	82757	18752951	20869786591
1	2	5	T-Series	12661	61196302	47548839843
2	3	5	Cocomelon - Nursery Rhymes	373	19238251	9793305082
3	4	5	SET India	27323	31180559	22675948293
4	5	5	WWE	36756	32852346	26273668433
...	...	...	...	...	...	...
4995	4996	1	Uras Benlioğlu	706	2072942	441202795
4996	4997	1	HI-TECH MUSIC LTD	797	1055091	377331722
4997	4998	1	Mastersaint	110	3265735	311758426
4998	4999	1	Bruce McIntosh	3475	32990	14563764
4999	5000	1	SehatAQUA	254	21172	73312511

4610 rows × 6 columns

Practical 7 - Categorical Data Cleaning

9 June 2022

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [3]:
```
titanic  =  pd.read_csv('D:\\ML\\titanic.csv')
```

In [4]:
```
titanic.head(10) #to see starting few records
```

Out[4]:

	PassengerId	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embark
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	
1	893	3	Wil es, Mrs. Jai es (E len Ne ds)	female	47.0	1	0	363272	7.0000	NaN	
2	894	2	M les, Mr. Tho as Fra cis	male	62.0	0	0	240276	9.6875	NaN	
3	895	3	Wirz, Mr. Al ert	male	27.0	0	0	315154	8.6625	NaN	
4	896	3	Hirvonen, Mrs. Alexa der (Helg a E Lindq ist)	female	22.0	1	1	3101298	12.2875	NaN	
5	897	3	Svens on, Mr. Jo an Ce vin	male	14.0	0	0	7538	9.2250	NaN	
6	898	3	Conn lly, M ss. Kate	female	30.0	0	0	330972	7.6292	NaN	
7	899	2	Cald ell, Mr. Albert Francis	male	26.0	1	1	248738	29.0000	NaN	
8	900	3	Abrahim, Mrs. Joseph (Sophie Halaut Easu)	female	18.0	0	0	2657	7.2292	NaN	
9	901	3	Davies, Mr. John Samuel	male	21.0	2	0	48 A/4 71	24.1500	NaN	

In [5]:
```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Gender       418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

In [6]:
```
#convert categorical variable into dummy variables
pd.get_dummies(titanic['Gender'])
```

Out[6]:

	female	male
0	0	1
1	1	0
2	0	1
3	0	1
4	1	0
...	...	...
413	0	1
414	1	0
415	0	1
416	0	1
417	0	1

418 rows × 2 columns

In [7]:
```
pd.get_dummies(titanic['Gender'],drop_first=True)
```

Out[7]:

	male
0	1
1	0
2	1
3	1
4	0
...	...
413	1
414	0
415	1
416	1
417	1

418 rows × 1 columns

In [8]:
```
gender = pd.get_dummies(titanic['Gender'],drop_first=False)
gender
```

Out[8]:

	female	male
0	0	1
1	1	0
2	0	1
3	0	1
4	1	0
...	...	...
413	0	1
414	1	0
415	0	1
416	0	1
417	0	1

418 rows × 2 columns

In [9]:
```
pd.get_dummies(titanic['Embarked'])
```

Out[9]:

	C	Q	S
0	0	1	0
1	0	0	1
2	0	1	0
3	0	0	1
4	0	0	1
...	...	...	...
413	0	0	1
414	1	0	0
415	0	0	1
416	0	0	1
417	1	0	0

418 rows × 3 columns

In [10]:
```
pd.get_dummies(titanic['Embarked'],drop_first=True)
```

Out[10]:

	Q	S
0	1	0
1	0	1
2	1	0
3	0	1
4	0	1
...	...	...
413	0	1
414	0	0
415	0	1
416	0	1
417	0	0

418 rows × 2 columns

In [11]:
```
embarked  =  pd.get_dummies(titanic['Embarked'],drop_first=False)
embarked
```

Out[11]:

	C	Q	S
0	0	1	0
1	0	0	1
2	0	1	0
3	0	0	1
4	0	0	1
...	...	...	...
413	0	0	1
414	1	0	0
415	0	0	1
416	0	0	1
417	1	0	0

418 rows × 3 columns

In [12]:
```
titanic.drop(['Gender','Embarked','Name','Ticket'],axis=1,inplace=True)
```

In [13]:
```
titanic.head()
```

Out[13]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Cabin
0	892	3	34.5	0	0	7.8292	NaN
1	893	3	47.0	1	0	7.0000	NaN
2	894	2	62.0	0	0	9.6875	NaN
3	895	3	27.0	0	0	8.6625	NaN
4	896	3	22.0	1	1	12.2875	NaN

In [14]:
```
titanic  =  pd.concat([titanic,gender,embarked],axis=1)
```

In [15]:
```
titanic.head()
```

Out[15]:

	PassengerId	Pclass	Age	SibSp	Parch	Fare	Cabin	female	male	C	Q	S
0	892	3	34.5	0	0	7.8292	NaN	0	1	0	1	0
1	893	3	47.0	1	0	7.0000	NaN	1	0	0	0	1
2	894	2	62.0	0	0	9.6875	NaN	0	1	0	1	0
3	895	3	27.0	0	0	8.6625	NaN	0	1	0	0	1
4	896	3	22.0	1	1	12.2875	NaN	1	0	0	0	1

In [ ]:
```

```

Practical 8 - Linear Regression

9 June 2022

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.model_selection import train_test_split
```

In [2]:
```
dfgpa=pd.read_csv('D:\\ML\\LR1.csv')
```

In [3]:
```
dfgpa.head(10)
```

Out[3]:

	SAT	GPA
0	1714	2.40
1	1664	2.52
2	1760	2.54
3	1685	2.74
4	1693	2.83
5	1670	2.91
6	1764	3.00
7	1764	3.00
8	1792	3.01
9	1850	3.01

In [4]:
```
dfgpa.describe()
```

Out[4]:

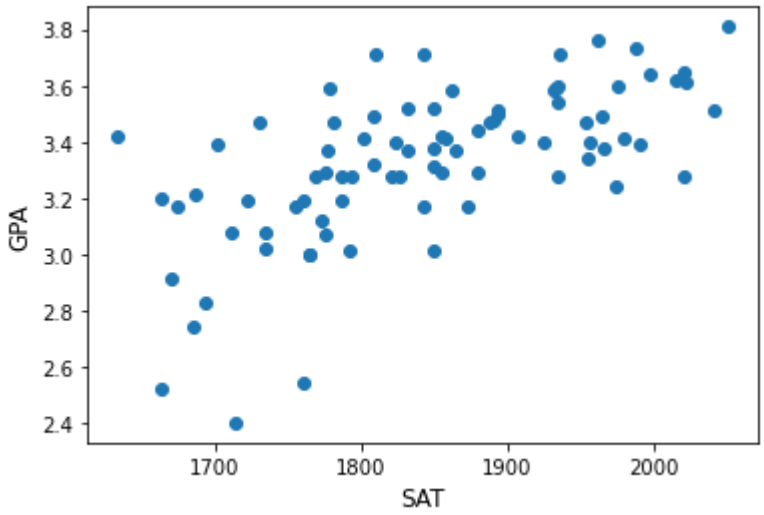
	SAT	GPA
count	84.000000	84.000000
mean	1845.273810	3.330238
std	104.530661	0.271617
min	1634.000000	2.400000
25%	1772.000000	3.190000
50%	1846.000000	3.380000
75%	1934.000000	3.502500
max	2050.000000	3.810000

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

Out[5]:
```
(84, 2)
```

In [6]:
```
x=dfgpa['SAT']
y=dfgpa['GPA']
```

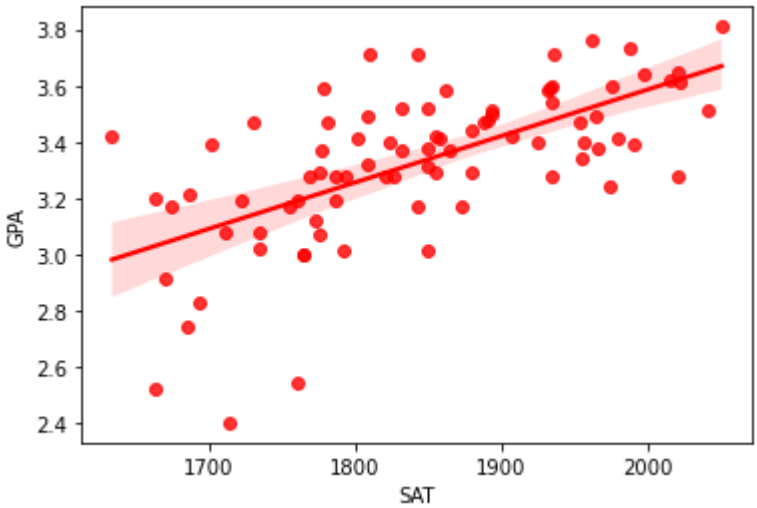
In [7]:
```
plt.scatter(x,y)
plt.xlabel('SAT',fontsize=12)
plt.ylabel('GPA',fontsize=12)
plt.show()
```



In [8]:
```
sns.regplot(x,y,color='red')
```

```
C:\Users\gaurav\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(
```

Out[8]:
```
<AxesSubplot:xlabel='SAT', ylabel='GPA'>
```



In [9]:
```
x.head()
```

Out[9]:
```
0    1714
1    1664
2    1760
3    1685
4    1693
Name: SAT, dtype: int64
```

In [10]:
```
X_=x.values.reshape(-1,1) # regression model only accepts the array value
```

In [11]:
```
x.shape
```

Out[11]:
```
(84,)
```

In [12]:
```
X_.shape
```

Out[12]:
```
(84, 1)
```

In [13]:
```
x
```

Out[13]:
```
0     1714
1     1664
2     1760
3     1685
4     1693
      ...
79    1936
80    1810
81    1987
82    1962
83    2050
Name: SAT, Length: 84, dtype: int64
```

In [14]:
```
X_
```

Out[14]:
```
array([[1714],
       [1664],
       [1760],
       [1685],
       [1693],
       [1670],
       [1764],
       [1764],
       [1792],
       [1850],
       [1735],
       [1775],
       [1735],
       [1712],
       [1773],
       [1872],
       [1755],
       [1674],
       [1842],
       [1786],
       [1761],
       [1722],
       [1663],
       [1687],
       [1974],
       [1826],
       [1787],
       [1821],
       [2020],
       [1794],
       [1769],
       [1934],
       [1775],
       [1855],
       [1880],
       [1849],
       [1808],
       [1954],
       [1777],
       [1831],
       [1865],
       [1850],
       [1966],
       [1702],
       [1990],
       [1925],
       [1824],
       [1956],
       [1857],
       [1979],
       [1802],
       [1855],
       [1907],
       [1634],
       [1879],
       [1887],
       [1730],
       [1953],
       [1781],
       [1891],
       [1964],
       [1808],
       [1893],
       [2041],
```

```
       [1893],
       [1832],
       [1850],
       [1934],
       [1861],
       [1931],
       [1933],
       [1778],
       [1975],
       [1934],
       [2021],
       [2015],
       [1997],
       [2020],
       [1843],
       [1936],
       [1810],
       [1987],
       [1962],
       [2050]], dtype=int64)
```

data cleaning done

Model

In [15]:
```
#this is important step for model devloping
#dividing the data into trainning and Testting
```

dividing

In [16]:
```
X_train,X_test,y_train,y_test=train_test_split(X_,y,test_size=0.1,random_state=69)
```

In [17]:
```
X_train.shape
```

Out[17]:
```
(75, 1)
```

In [18]:
```
X_test.shape
```

Out[18]:
```
(9, 1)
```

MODEL IS READY

NOW START THE *TRAINING*

In [19]:
```
#this is supervised traning technique
#that's why we are giving the input and label together
LR=LinearRegression()
LR.fit(X_train,y_train)
```

Out[19]:
```
LinearRegression()
```

In [20]:
```
# predict() is used for predicting
y_pred=LR.predict(X_test)
```

In [21]:
```
y_test
```

Out[21]:
```
67    3.54
15    3.17
36    3.32
25    3.28
10    3.02
50    3.41
51    3.42
31    3.28
40    3.37
Name: GPA, dtype: float64
```

In [22]:
```
y_pred
```

Out[22]:
```
array([3.48090482, 3.3774985 , 3.27075649, 3.30077768, 3.14900389,
       3.26074943, 3.34914515, 3.48090482, 3.36582359])
```

error

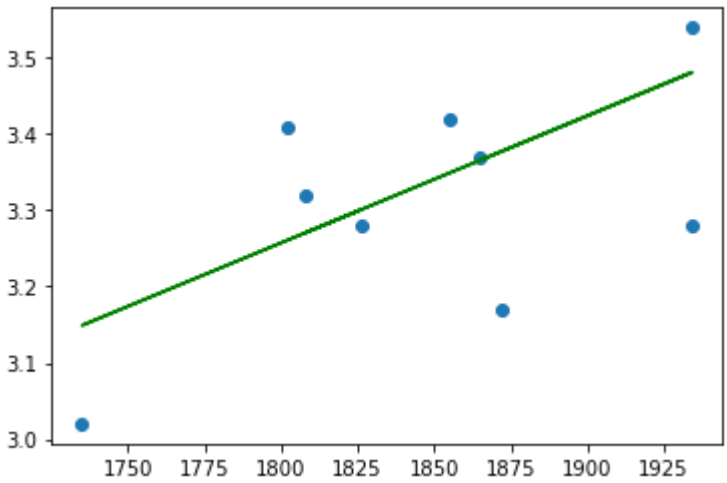
In [23]:
```
acc=mean_squared_error(y_test,y_pred)
acc
```

Out[23]:
```
0.014858092961980589
```

In [24]:
```
weights = LR.coef_
intercept = LR.intercept_
print(weights,intercept)
```

```
[0.00166784] 0.2552947901783016
```

In [25]:
```
plt.scatter(X_test, y_test)
plt.plot(X_test,y_pred, color='green')
plt.show()
```

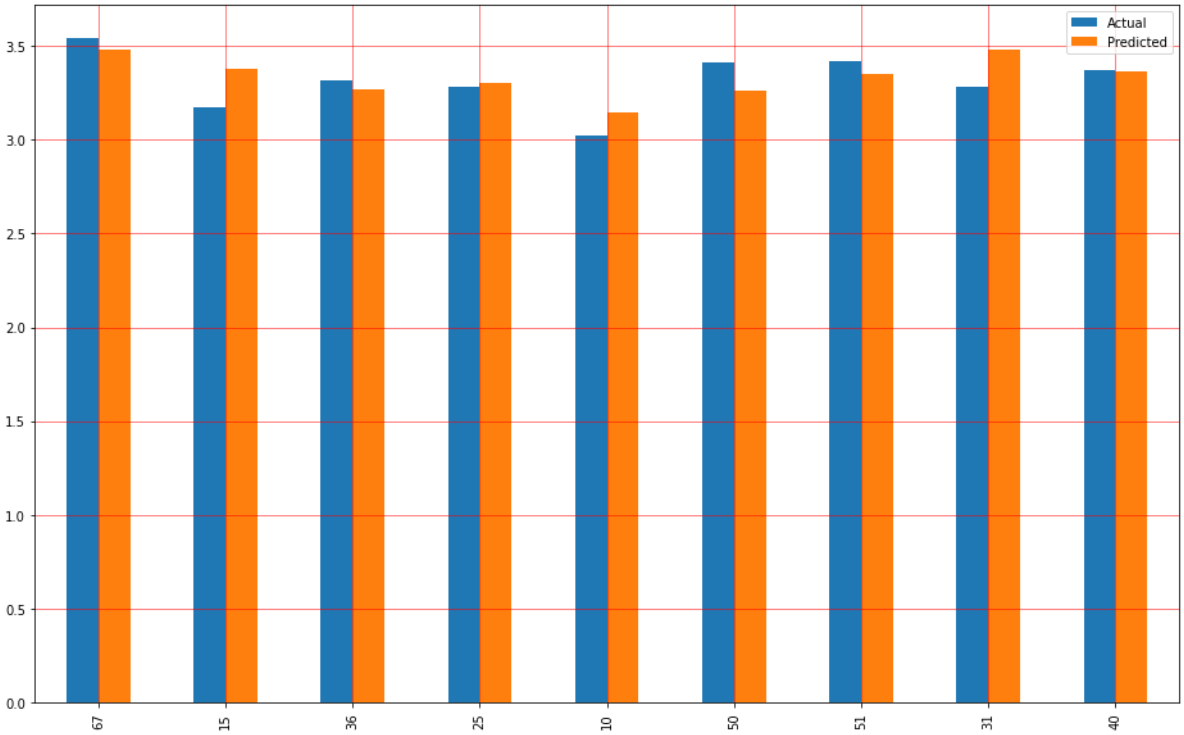


In [26]:
```
df=pd.DataFrame({'Actual':y_test,'Predicted':y_pred})
df
```

Out[26]:

	Actual	Predicted
67	3.54	3.480905
15	3.17	3.377498
36	3.32	3.270756
25	3.28	3.300778
10	3.02	3.149004
50	3.41	3.260749
51	3.42	3.349145
31	3.28	3.480905
40	3.37	3.365824

In [27]:
```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='red')
plt.grid(which='minor',linestyle=':',linewidth='0.5',color='green')
plt.show()
```



almost all the predicted and actual value are similar except some of the values

now we are changing the data or we can say giving new data to predict

this can be done only if the model is acceptable

In [28]:
```
new_data=pd.DataFrame([2115,900])
new_data
```

Out[28]:

	0
0	2115
1	900

In [29]:
```
LR.predict(new_data)
```

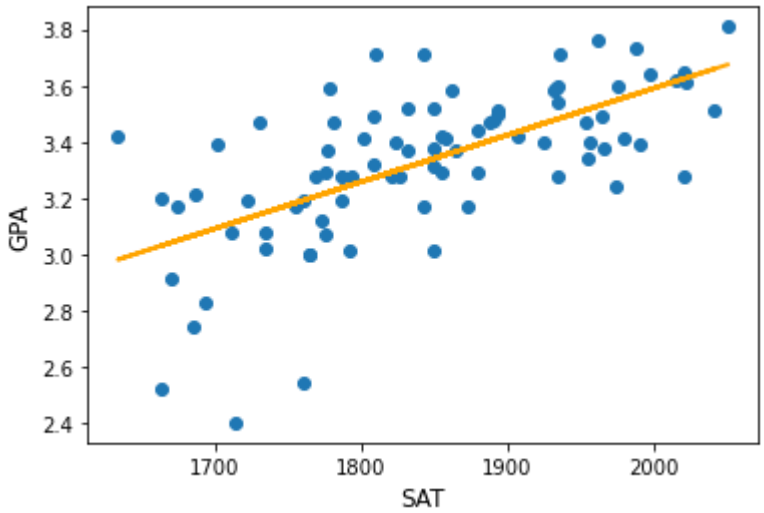
Out[29]:
```
array([3.78278456, 1.75635427])
```

preformance measure

In [30]:
```
print('Mean Absolute Error',mean_absolute_error(y_test,y_pred))
print('Mean Squared Error',mean_squared_error(y_test,y_pred))
print('Root Mean Sqaured Error',np.sqrt(mean_squared_error(y_test,y_pred)))
```

```
Mean Absolute Error 0.0989783783858536
Mean Squared Error 0.014858092961980589
Root Mean Sqaured Error 0.12189377737186008
```

In [31]:
```
plt.scatter(x,y)
yhat=LR.coef_*x+LR.intercept_  #yhat=0.275+0.0017x1 regression line
fig=plt.plot(x,yhat,lw=2,c='orange',label='Regression Line')
plt.xlabel('SAT',fontsize='12')
plt.ylabel('GPA',fontsize='12')
plt.show()
```



80 : 20------Mean Absolute Error 0.14738307729471656 Mean Squared Error 0.039186315582133514 Root Mean Sqaured Error 0.1979553373418699

70:30 -- Mean Absolute Error 0.1469412968249975 Mean Squared Error 0.03697034973833757 Root Mean Sqaured Error 0.1922767529846954

90:10 -- Mean Absolute Error 0.09897837838585355 Mean Squared Error 0.014858092961980589 Root Mean Sqaured Error 0.12189377737186008

In [ ]:

Practical 9 - Multivariate Linear Regression - auto mpg dataset

10 June 2022

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,OneHotEncoder
```

In [2]:
```
data=pd.read_fwf("D:\\ML\\auto-mpg.data")
data
```

Out[2]:

	18.0	8	307.0	130.0	3504.	12.0	70	1	"chevrolet chevelle malibu"
0	15.0	8	350.0	165.0	3693.0	11.5	70	1	"buick skylark 320"
1	18.0	8	318.0	150.0	3436.0	11.0	70	1	"plymouth satellite"
2	16.0	8	304.0	150.0	3433.0	12.0	70	1	"amc rebel sst"
3	17.0	8	302.0	140.0	3449.0	10.5	70	1	"ford torino"
4	15.0	8	429.0	198.0	4341.0	10.0	70	1	"ford galaxie 500"
...	...	...	...	...	...	...	...	...	...
392	27.0	4	140.0	86.00	2790.0	15.6	82	1	"ford mustang gl"
393	44.0	4	97.0	52.00	2130.0	24.6	82	2	"vw pickup"
394	32.0	4	135.0	84.00	2295.0	11.6	82	1	"dodge rampage"
395	28.0	4	120.0	79.00	2625.0	18.6	82	1	"ford ranger"
396	31.0	4	119.0	82.00	2720.0	19.4	82	1	"chevy s-10"

397 rows × 9 columns

In [3]:
```
col_names=['mpg','cylinder','displacement','horsepower','weight','acceleration','m
```

In [4]:
```
data.columns=col_names
```

In [5]:
```
data
```

Out[5]:

	mpg	cylinder	displacement	horsepower	weight	acceleration	modelyear	origin	carna
0	15.0	8	350.0	165.0	3693.0	11.5	70	1	"bui skyla 32
1	18.0	8	318.0	150.0	3436.0	11.0	70	1	"plymou satellit
2	16.0	8	304.0	150.0	3433.0	12.0	70	1	"a rebel s
3	17.0	8	302.0	140.0	3449.0	10.5	70	1	"fo torin
4	15.0	8	429.0	198.0	4341.0	10.0	70	1	"fo gala 50
...	...	...	...	...	...	...	...	...	
392	27.0	4	140.0	86.00	2790.0	15.6	82	1	"fo musta
393	44.0	4	97.0	52.00	2130.0	24.6	82	2	" picku
394	32.0	4	135.0	84.00	2295.0	11.6	82	1	"dod rampag
395	28.0	4	120.0	79.00	2625.0	18.6	82	1	"fo range
396	31.0	4	119.0	82.00	2720.0	19.4	82	1	"chevy 1

397 rows × 9 columns

In [6]:
```
data.head()
```

Out[6]:

	mpg	cylinder	displacement	horsepower	weight	acceleration	modelyear	origin	carname
0	15.0	8	350.0	165.0	3693.0	11.5	70	1	"buick skylark 320"
1	18.0	8	318.0	150.0	3436.0	11.0	70	1	"plymouth satellite"
2	16.0	8	304.0	150.0	3433.0	12.0	70	1	"amc rebel sst"
3	17.0	8	302.0	140.0	3449.0	10.5	70	1	"ford torino"
4	15.0	8	429.0	198.0	4341.0	10.0	70	1	"ford galaxie 500"

In [7]:
```
data.describe()
```

Out[7]:

	mpg	cylinder	displacement	weight	acceleration	modelyear	origin
count	397.000000	397.000000	397.000000	397.000000	397.000000	397.000000	397.000000
mean	23.528463	5.448363	193.139798	2969.080605	15.577078	76.025189	1.574307
std	7.820926	1.698329	104.244898	847.485218	2.755326	3.689922	0.802549
min	9.000000	3.000000	68.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.000000	2223.000000	13.900000	73.000000	1.000000
50%	23.000000	4.000000	146.000000	2800.000000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	3609.000000	17.200000	79.000000	2.000000
max	46.600000	8.000000	455.000000	5140.000000	24.800000	82.000000	3.000000

In [8]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 397 entries, 0 to 396
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------------------------------  -----  
 0   mpg           397 non-null    float64
 1   cylinder      397 non-null    int64  
 2   displacement  397 non-null    float64
 3   horsepower    397 non-null    object 
 4   weight        397 non-null    float64
 5   acceleration  397 non-null    float64
 6   modelyear     397 non-null    int64  
 7   origin        397 non-null    int64  
 8   carname       397 non-null    object 
dtypes: float64(4), int64(3), object(2)
memory usage: 28.0+ KB
```

In [9]:
```
data.shape
```

Out[9]:
```
(397, 9)
```

In [10]:
```
data.isnull()
```

Out[10]:

	mpg	cylinder	displacement	horsepower	weight	acceleration	modelyear	origin	carnam
0	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	Fals
3	False	False	False	False	False	False	False	False	Fals
4	False	False	False	False	False	False	False	False	Fals
...	...	...	...	...	...	...	...	...	.
392	False	False	False	False	False	False	False	False	Fals
393	False	False	False	False	False	False	False	False	Fals
394	False	False	False	False	False	False	False	False	Fals
395	False	False	False	False	False	False	False	False	Fals
396	False	False	False	False	False	False	False	False	Fals

397 rows × 9 columns

In [11]:
```
data['carname'].value_counts()
```

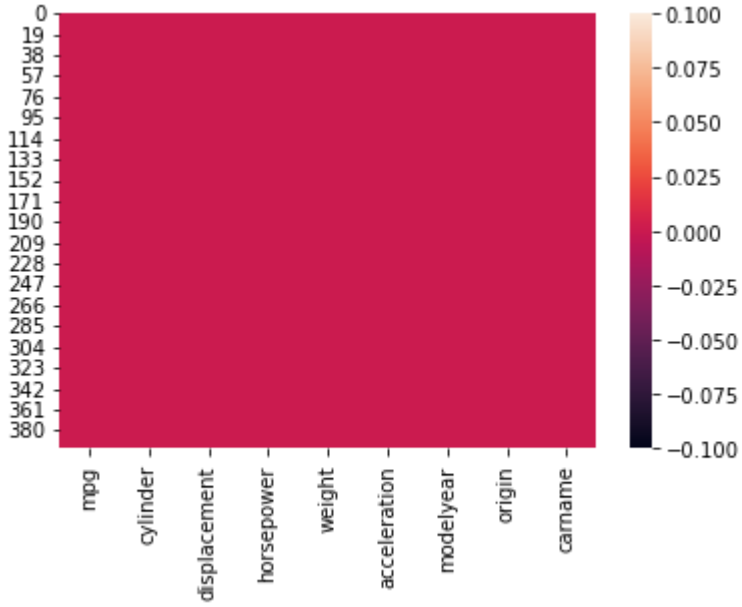
Out[11]:
```
"ford pinto"            6
"amc matador"           5
"toyota corolla"        5
"ford maverick"         5
"amc hornet"            4
                       ..
"chevrolet monza 2+2"   1
"ford mustang ii"       1
"pontiac astro"         1
"amc pacer"             1
"chevy s-10"            1
Name: carname, Length: 305, dtype: int64
```

In [12]:
```
type(data)
```

Out[12]:
```
pandas.core.frame.DataFrame
```

In [13]:
```
sns.heatmap(data.isnull())
```

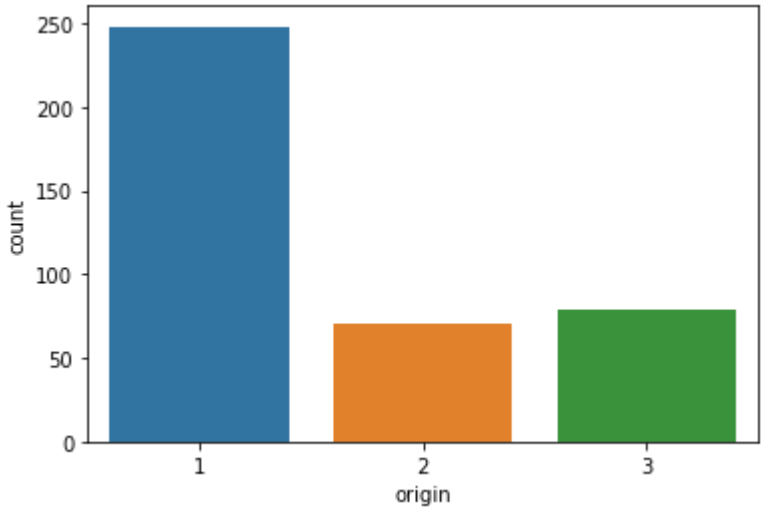
Out[13]:
```
<AxesSubplot:>
```



no null value is there

In [14]:
```
sns.countplot(x='origin',data=data)
```

Out[14]:
```
<AxesSubplot:xlabel='origin',  ylabel='count'>
```



In [15]:
```
data['carname'].unique()
```

Out[15]:
```
array(['"buick skylark 320"', '"plymouth satellite"', '"amc rebel sst"',
       '"ford torino"', '"ford galaxie 500"', '"chevrolet impala"',
       '"plymouth fury iii"', '"pontiac catalina"',
       '"amc ambassador dpl"', '"dodge challenger se"',
       '"plymouth \'cuda 340"', '"chevrolet monte carlo"',
       '"buick estate wagon (sw)"', '"toyota corona mark ii"',
       '"plymouth duster"', '"amc hornet"', '"ford maverick"',
       '"datsun pl510"', '"volkswagen 1131 deluxe sedan"',
       '"peugeot 504"', '"audi 100 ls"', '"saab 99e"', '"bmw 2002"',
       '"amc gremlin"', '"ford f250"', '"chevy c20"', '"dodge d200"',
       '"hi 1200d"', '"chevrolet vega 2300"', '"toyota corona"',
       '"ford pinto"', '"plymouth satellite custom"',
       '"chevrolet chevelle malibu"', '"ford torino 500"',
       '"amc matador"', '"pontiac catalina brougham"',
       '"dodge monaco (sw)"', '"ford country squire (sw)"',
       '"pontiac safari (sw)"', '"amc hornet sportabout (sw)"',
       '"chevrolet vega (sw)"', '"pontiac firebird"', '"ford mustang"',
       '"mercury capri 2000"', '"opel 1900"', '"peugeot 304"',
       '"fiat 124b"', '"toyota corolla 1200"', '"datsun 1200"',
       '"volkswagen model 111"', '"plymouth cricket"',
       '"toyota corona hardtop"', '"dodge colt hardtop"',
       '"volkswagen type 3"', '"chevrolet vega"', '"ford pinto runabout"',
       '"amc ambassador sst"', '"mercury marquis"',
       '"buick lesabre custom"', '"oldsmobile delta 88 royale"',
       '"chrysler newport royal"', '"mazda rx2 coupe"',
       '"amc matador (sw)"', '"chevrolet chevelle concours (sw)"',
       '"ford gran torino (sw)"', '"plymouth satellite custom (sw)"',
       '"volvo 145e (sw)"', '"volkswagen 411 (sw)"', '"peugeot 504 (sw)"',
       '"renault 12 (sw)"', '"ford pinto (sw)"', '"datsun 510 (sw)"',
       '"toyouta corona mark ii (sw)"', '"dodge colt (sw)"',
       '"toyota corolla 1600 (sw)"', '"buick century 350"',
       '"chevrolet malibu"', '"ford gran torino"',
       '"dodge coronet custom"', '"mercury marquis brougham"',
       '"chevrolet caprice classic"', '"ford ltd"',
       '"plymouth fury gran sedan"', '"chrysler new yorker brougham"',
       '"buick electra 225 custom"', '"amc ambassador brougham"',
       '"plymouth valiant"', '"chevrolet nova custom"',
       '"volkswagen super beetle"', '"ford country"',
       '"plymouth custom suburb"', '"oldsmobile vista cruiser"',
       '"toyota carina"', '"datsun 610"', '"maxda rx3"',
       '"mercury capri v6"', '"fiat 124 sport coupe"',
       '"chevrolet monte carlo s"', '"pontiac grand prix"', '"fiat 128"',
       '"opel manta"', '"audi 100ls"', '"volvo 144ea"',
       '"dodge dart custom"', '"saab 99le"', '"toyota mark ii"',
       '"oldsmobile omega"', '"chevrolet nova"', '"datsun b210"',
       '"chevrolet chevelle malibu classic',
       '"plymouth satellite sebring"', '"buick century luxus (sw)"',
       '"dodge coronet custom (sw)"', '"audi fox"', '"volkswagen dasher"',
       '"datsun 710"', '"dodge colt"', '"fiat 124 tc"', '"honda civic"',
       '"subaru"', '"fiat x1.9"', '"plymouth valiant custom"',
       '"mercury monarch"', '"chevrolet bel air"',
       '"plymouth grand fury"', '"buick century"',
       '"chevroelt chevelle malibu"', '"plymouth fury"',
       '"buick skyhawk"', '"chevrolet monza 2+2"', '"ford mustang ii"',
       '"toyota corolla"', '"pontiac astro"', '"volkswagen rabbit"',
       '"amc pacer"', '"volvo 244dl"', '"honda civic cvcc"', '"fiat 131"',
       '"capri ii"', '"renault 12tl"', '"dodge coronet brougham"',
       '"chevrolet chevette"', '"chevrolet woody"', '"vw rabbit"',
       '"dodge aspen se"', '"ford granada ghia"', '"pontiac ventura sj"',
       '"amc pacer d/l"', '"datsun b-210"', '"volvo 245"',
       '"plymouth volare premier v8"', '"mercedes-benz 280s"',
       '"cadillac seville"', '"chevy c10"', '"ford f108"', '"dodge d100"',
       '"honda accord cvcc"', '"buick opel isuzu deluxe"',
       '"renault 5 gtl"', '"plymouth arrow gs"',
```

```
'"datsun f-10 hatchback"', '"oldsmobile cutlass supreme"',
'"dodge monaco brougham"', '"mercury cougar brougham"',
'"chevrolet concours"', '"buick skylark"',
'"plymouth volare custom"', '"ford granada"',
'"pontiac grand prix lj"', '"chevrolet monte carlo landau"',
'"chrysler cordoba"', '"ford thunderbird"',
'"volkswagen rabbit custom"', '"pontiac sunbird coupe"',
'"toyota corolla liftback"', '"ford mustang ii 2+2"',
'"dodge colt m/m"', '"subaru dl"', '"datsun 810"', '"bmw 320i"',
'"mazda rx-4"', '"volkswagen rabbit custom diesel"',
'"ford fiesta"', '"mazda glc deluxe"', '"datsun b210 gx"',
'"oldsmobile cutlass salon brougham', '"dodge diplomat"',
'"mercury monarch ghia"', '"pontiac phoenix lj"',
'"ford fairmont (auto)"', '"ford fairmont (man)"',
'"plymouth volare"', '"amc concord"', '"buick century special"',
'"mercury zephyr"', '"dodge aspen"', '"amc concord d/l"',
'"buick regal sport coupe (turbo)"', '"ford futura"',
'"dodge magnum xe"', '"datsun 510"', '"dodge omni"',
'"toyota celica gt liftback"', '"plymouth sapporo"',
'"oldsmobile starfire sx"', '"datsun 200-sx"', '"audi 5000"',
'"volvo 264gl"', '"saab 99gle"', '"peugeot 604sl"',
'"volkswagen scirocco"', '"honda accord lx"',
'"pontiac lemans v6"', '"mercury zephyr 6"', '"ford fairmont 4"',
'"amc concord dl 6"', '"dodge aspen 6"', '"ford ltd landau"',
'"mercury grand marquis"', '"dodge st. regis"',
'"chevrolet malibu classic (sw)"',
'"chrysler lebaron town @ country (', '"vw rabbit custom"',
'"maxda glc deluxe"', '"dodge colt hatchback custom"',
'"amc spirit dl"', '"mercedes benz 300d"', '"cadillac eldorado"',
'"plymouth horizon"', '"plymouth horizon tc3"', '"datsun 210"',
'"fiat strada custom"', '"buick skylark limited"',
'"chevrolet citation"', '"oldsmobile omega brougham"',
'"pontiac phoenix"', '"toyota corolla tercel"', '"datsun 310"',
'"ford fairmont"', '"audi 4000"', '"toyota corona liftback"',
'"mazda 626"', '"datsun 510 hatchback"', '"mazda glc"',
'"vw rabbit c (diesel)"', '"vw dasher (diesel)"',
'"audi 5000s (diesel)"', '"mercedes-benz 240d"',
'"honda civic 1500 gl"', '"renault lecar deluxe"',
'"vokswagen rabbit"', '"datsun 280-zx"', '"mazda rx-7 gs"',
'"triumph tr7 coupe"', '"ford mustang cobra"', '"honda accord"',
'"plymouth reliant"', '"dodge aries wagon (sw)"',
'"toyota starlet"', '"plymouth champ"', '"honda civic 1300"',
'"datsun 210 mpg"', '"toyota tercel"', '"mazda glc 4"',
'"plymouth horizon 4"', '"ford escort 4w"', '"ford escort 2h"',
'"volkswagen jetta"', '"renault 18i"', '"honda prelude"',
'"datsun 200sx"', '"peugeot 505s turbo diesel"', '"volvo diesel"',
'"toyota cressida"', '"datsun 810 maxima"',
'"oldsmobile cutlass ls"', '"ford granada gl"',
'"chrysler lebaron salon"', '"chevrolet cavalier"',
'"chevrolet cavalier wagon"', '"chevrolet cavalier 2-door"',
'"pontiac j2000 se hatchback"', '"dodge aries se"',
'"ford fairmont futura"', '"amc concord dl"',
'"volkswagen rabbit l"', '"mazda glc custom l"',
'"mazda glc custom"', '"plymouth horizon miser"',
'"mercury lynx l"', '"nissan stanza xe"', '"honda civic (auto)"',
'"datsun 310 gx"', '"buick century limited"',
'"oldsmobile cutlass ciera (diesel)',
'"chrysler lebaron medallion"', '"ford granada l"',
'"toyota celica gt"', '"dodge charger 2.2"', '"chevrolet camaro"',
'"ford mustang gl"', '"vw pickup"', '"dodge rampage"',
'"ford ranger"', '"chevy s-10"'], dtype=object)
```

In [16]:
```
data['carname']=[i[0]for i in data['carname'].str.split(' ')]
```

In [17]:
```
data['carname'].unique()
```

Out[17]:
```
array(['"buick', '"plymouth', '"amc', '"ford', '"chevrolet', '"pontiac',
       '"dodge', '"toyota', '"datsun', '"volkswagen', '"peugeot', '"audi',
       '"saab', '"bmw', '"chevy', '"hi', '"mercury', '"opel', '"fiat',
       '"oldsmobile', '"chrysler', '"mazda', '"volvo', '"renault',
       '"toyouta', '"maxda', '"honda', '"subaru"', '"chevroelt', '"capri',
       '"vw', '"mercedes-benz', '"cadillac', '"subaru', '"mercedes',
       '"vokswagen', '"triumph', '"nissan'], dtype=object)
```

In [18]:
```
data['carname']=data['carname'].replace(['"chevrolet','"chevy','"chevroelt'],'chev
data['carname']=data['carname'].replace(['"volkswagen','"vokswagen','"vw'],'volksw
data['carname']=data['carname'].replace('"maxda','mazda')
data['carname']=data['carname'].replace('"toyouta','toyota')
data['carname']=data['carname'].replace('"mercedes','mercedes-benz')
data['carname']=data['carname'].replace('"nissan','datsun')
data['carname']=data['carname'].replace('"capri','ford')
```

In [19]:
```
data['carname'].unique()
```

Out[19]:
```
array(['"buick', '"plymouth', '"amc', '"ford', 'chevrolet', '"pontiac',
       '"dodge', '"toyota', '"datsun', 'volkswagen', '"peugeot', '"audi',
       '"saab', '"bmw', '"hi', '"mercury', '"opel', '"fiat',
       '"oldsmobile', '"chrysler', '"mazda', '"volvo', '"renault',
       'toyota', 'mazda', '"honda', '"subaru"', 'ford', '"mercedes-benz',
       '"cadillac', '"subaru', 'mercedes-benz', '"triumph', 'datsun'],
      dtype=object)
```

In [20]:
```
org=pd.get_dummies(data.origin,prefix='org')
org
```

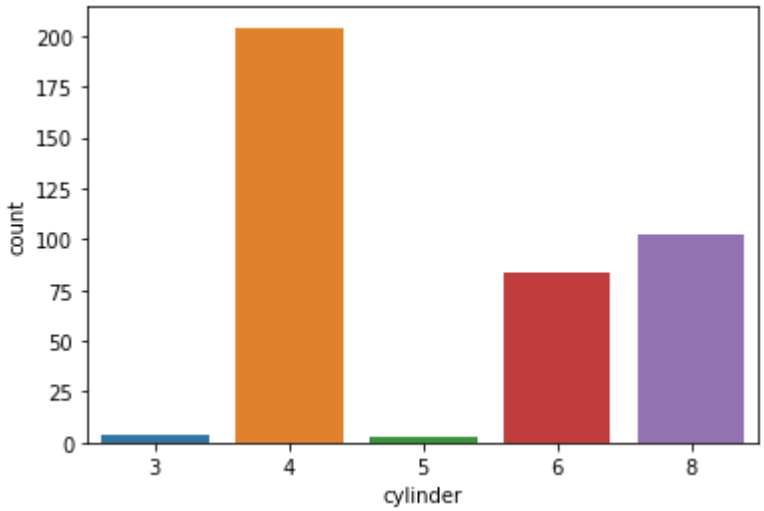
Out[20]:

	org_1	org_2	org_3
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
...	...	...	...
392	1	0	0
393	0	1	0
394	1	0	0
395	1	0	0
396	1	0	0

397 rows × 3 columns

In [21]:
```
sns.countplot(x='cylinder',data=data)
```

Out[21]:
```
<AxesSubplot:xlabel='cylinder',  ylabel='count'>
```



In [22]:
```
cyl=pd.get_dummies(data.cylinder,prefix='cyl')
cyl
```

Out[22]:

	cyl_3	cyl_4	cyl_5	cyl_6	cyl_8
0	0	0	0	0	1
1	0	0	0	0	1
2	0	0	0	0	1
3	0	0	0	0	1
4	0	0	0	0	1
...	...	...	...	...	...
392	0	1	0	0	0
393	0	1	0	0	0
394	0	1	0	0	0
395	0	1	0	0	0
396	0	1	0	0	0

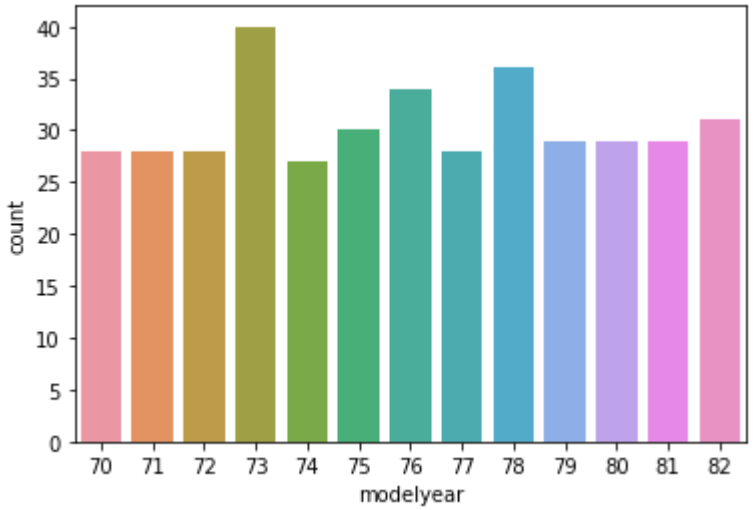
397 rows × 5 columns

In [23]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 397 entries, 0 to 396
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------------  --------------  -----
 0   mpg           397 non-null    float64
 1   cylinder      397 non-null    int64
 2   displacement  397 non-null    float64
 3   horsepower    397 non-null    object
 4   weight        397 non-null    float64
 5   acceleration  397 non-null    float64
 6   modelyear     397 non-null    int64
 7   origin        397 non-null    int64
 8   carname       397 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.0+ KB
```

In [24]:
```
sns.countplot(x='modelyear',data=data)
```

Out[24]:
```
<AxesSubplot:xlabel='modelyear',  ylabel='count'>
```



In [25]:
```
year=pd.get_dummies(data.modelyear,prefix='year')
year
```

Out[25]:

	year_70	year_71	year_72	year_73	year_74	year_75	year_76	year_77	year_78	year_79	y
0	1	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	
4	1	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...
392	0	0	0	0	0	0	0	0	0	0	
393	0	0	0	0	0	0	0	0	0	0	
394	0	0	0	0	0	0	0	0	0	0	
395	0	0	0	0	0	0	0	0	0	0	
396	0	0	0	0	0	0	0	0	0	0	

397 rows × 13 columns

In [26]:
```
cn=pd.get_dummies(data['carname'],prefix='cn')
cn
```

Out[26]:

	cn_"amc	cn_"audi	cn_"bmw	cn_"buick	cn_"cadillac	cn_"chrysler	cn_"datsun	cn_"dodge
0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
392	0	0	0	0	0	0	0	0
393	0	0	0	0	0	0	0	0
394	0	0	0	0	0	0	0	1
395	0	0	0	0	0	0	0	0
396	0	0	0	0	0	0	0	0

397 rows × 34 columns

In [27]:
```
data
```

Out[27]:

	mpg	cylinder	displacement	horsepower	weight	acceleration	modelyear	origin	carna
0	15.0	8	350.0	165.0	3693.0	11.5	70	1	"b
1	18.0	8	318.0	150.0	3436.0	11.0	70	1	"plymo
2	16.0	8	304.0	150.0	3433.0	12.0	70	1	"a
3	17.0	8	302.0	140.0	3449.0	10.5	70	1	"f
4	15.0	8	429.0	198.0	4341.0	10.0	70	1	"f
...	...	...	...	...	...	...	...	...	
392	27.0	4	140.0	86.00	2790.0	15.6	82	1	"f
393	44.0	4	97.0	52.00	2130.0	24.6	82	2	volkswa
394	32.0	4	135.0	84.00	2295.0	11.6	82	1	"do
395	28.0	4	120.0	79.00	2625.0	18.6	82	1	"f
396	31.0	4	119.0	82.00	2720.0	19.4	82	1	chevr

397 rows × 9 columns

In [28]:
```
data.drop(['origin','cylinder','modelyear','carname'],axis=1,inplace=True)
```

In [29]:
```
data
```

Out[29]:

	mpg	displacement	horsepower	weight	acceleration
0	15.0	350.0	165.0	3693.0	11.5
1	18.0	318.0	150.0	3436.0	11.0
2	16.0	304.0	150.0	3433.0	12.0
3	17.0	302.0	140.0	3449.0	10.5
4	15.0	429.0	198.0	4341.0	10.0
...	...	...	...	...	...
392	27.0	140.0	86.00	2790.0	15.6
393	44.0	97.0	52.00	2130.0	24.6
394	32.0	135.0	84.00	2295.0	11.6
395	28.0	120.0	79.00	2625.0	18.6
396	31.0	119.0	82.00	2720.0	19.4

397 rows × 5 columns

In [30]:
```
data=pd.concat([data,cn,year,cyl,org],axis=1)
data
```

Out[30]:

	mpg	displacement	horsepower	weight	acceleration	cn_"amc	cn_"audi	cn_"bmw	cn_"b
0	15.0	350.0	165.0	3693.0	11.5	0	0	0	
1	18.0	318.0	150.0	3436.0	11.0	0	0	0	
2	16.0	304.0	150.0	3433.0	12.0	1	0	0	
3	17.0	302.0	140.0	3449.0	10.5	0	0	0	
4	15.0	429.0	198.0	4341.0	10.0	0	0	0	
...	...	...	...	...	...	...	...	...	
392	27.0	140.0	86.00	2790.0	15.6	0	0	0	
393	44.0	97.0	52.00	2130.0	24.6	0	0	0	
394	32.0	135.0	84.00	2295.0	11.6	0	0	0	
395	28.0	120.0	79.00	2625.0	18.6	0	0	0	
396	31.0	119.0	82.00	2720.0	19.4	0	0	0	

397 rows × 60 columns

In [31]:
```
data.shape
```

Out[31]:
```
(397, 60)
```

In [58]:
```
#data[['displacement','horsepower','weight','acceleration']]=StandardScaler().fit_
```

In [33]:
```
sum(data.horsepower=='?')
```

Out[33]:
```
6
```

In [34]:
```
data=data[data.horsepower!='?']
```

In [35]:
```
data
```

Out[35]:

	mpg	displacement	horsepower	weight	acceleration	cn_"amc	cn_"audi	cn_"bmw	cn_"b
0	15.0	350.0	165.0	3693.0	11.5	0	0	0	
1	18.0	318.0	150.0	3436.0	11.0	0	0	0	
2	16.0	304.0	150.0	3433.0	12.0	1	0	0	
3	17.0	302.0	140.0	3449.0	10.5	0	0	0	
4	15.0	429.0	198.0	4341.0	10.0	0	0	0	
...	...	...	...	...	...	...	...	...	
392	27.0	140.0	86.00	2790.0	15.6	0	0	0	
393	44.0	97.0	52.00	2130.0	24.6	0	0	0	
394	32.0	135.0	84.00	2295.0	11.6	0	0	0	
395	28.0	120.0	79.00	2625.0	18.6	0	0	0	
396	31.0	119.0	82.00	2720.0	19.4	0	0	0	

391 rows × 60 columns

In [36]:
```
y=data.pop('mpg')
```

In [37]:
```
y
```

Out[37]:
```
0      15.0
1      18.0
2      16.0
3      17.0
4      15.0
       ...
392    27.0
393    44.0
394    32.0
395    28.0
396    31.0
Name: mpg, Length: 391, dtype: float64
```

In [38]:
```
x=data
```

In [39]:
```
x.head(269)
```

Out[39]:

	displacement	horsepower	weight	acceleration	cn_"amc	cn_"audi	cn_"bmw	cn_"buick	c
0	350.0	165.0	3693.0	11.5	0	0	0	1	
1	318.0	150.0	3436.0	11.0	0	0	0	0	
2	304.0	150.0	3433.0	12.0	1	0	0	0	
3	302.0	140.0	3449.0	10.5	0	0	0	0	
4	429.0	198.0	4341.0	10.0	0	0	0	0	
...	...	...	...	...	...	...	...	...	
266	134.0	95.00	2560.0	14.2	0	0	0	0	
267	119.0	97.00	2300.0	14.7	0	0	0	0	
268	105.0	75.00	2230.0	14.5	0	0	0	0	
269	134.0	95.00	2515.0	14.8	0	0	0	0	
270	156.0	105.0	2745.0	16.7	0	0	0	0	

269 rows × 59 columns

dividing 80:20

In [40]:
```
trainx,testx,trainy,testy=train_test_split(x,y,test_size=0.2,random_state=69)
```

In [41]:
```
LR=LinearRegression()
LR.fit(trainx,trainy)
```

Out[41]:
```
LinearRegression()
```

In [42]:
```
pred=LR.predict(testx)
mean_squared_error(pred,testy)
```

Out[42]:
```
6.527810934720171
```

In [43]:
```
df=pd.DataFrame({'Actual':testy,'Predicted':pred})
```

In [44]:
```
df
```

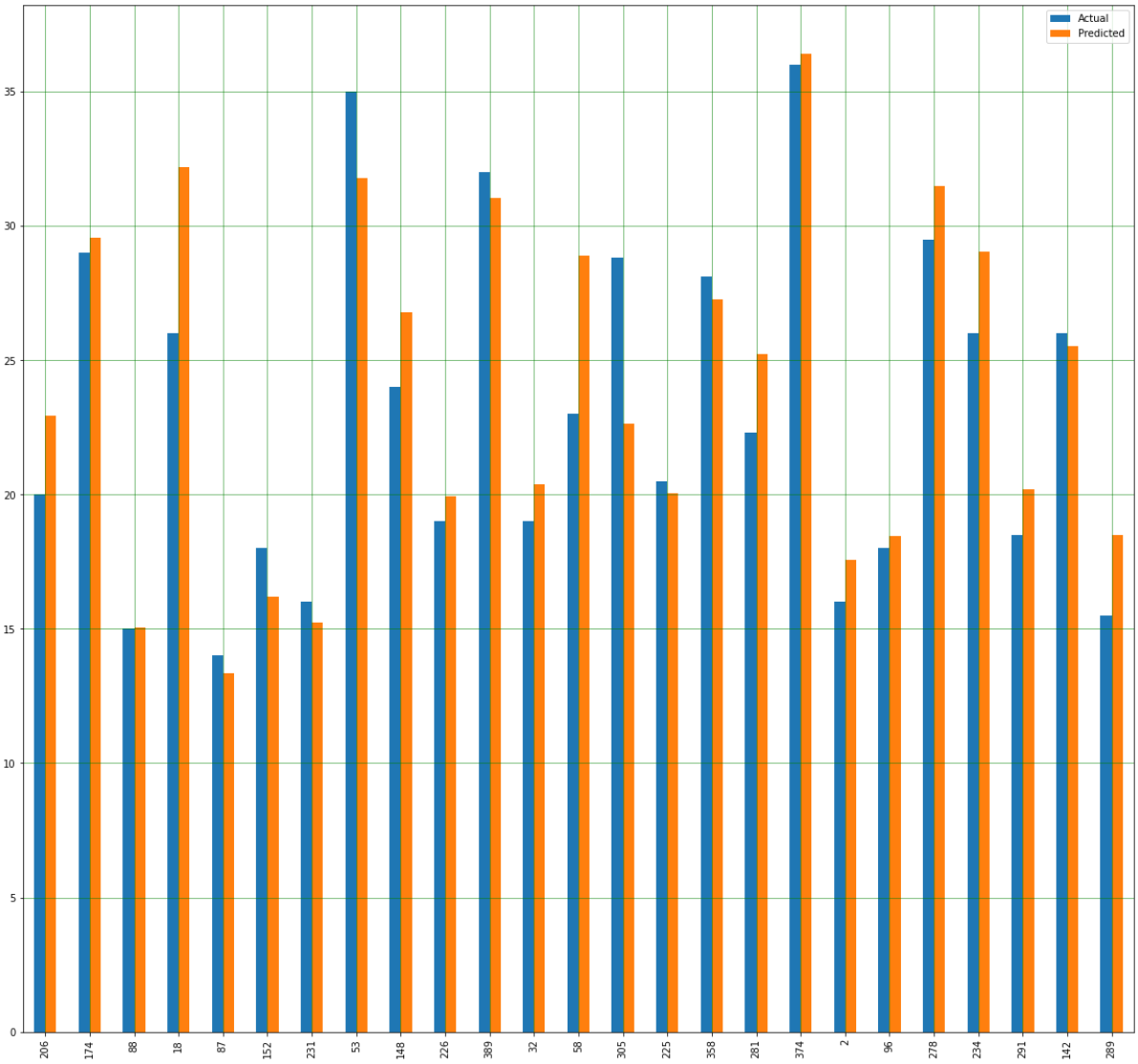
Out[44]:

	Actual	Predicted
206	20.0	22.939385
174	29.0	29.567980
88	15.0	15.039357
18	26.0	32.177102
87	14.0	13.334395
...	...	...
105	12.0	12.299412
211	16.5	14.440355
61	13.0	12.059697
199	18.0	17.634573
21	25.0	27.754687

79 rows × 2 columns

In [45]:

```
df1=df.head(25)
df1.plot(y=['Actual','Predicted'],kind='bar',figsize=(20,19));
plt.grid(which='major',ls='-',linewidth=0.5,color='g')
plt.grid(which='minor',ls=':',linewidth=0.5,color='b')
plt.show()
```



In [46]:
```
print('Mean  Absolute  Error',mean_absolute_error(testy,pred))
print('Mean Squared Error',mean_squared_error(testy,pred))
print('Root  Mean  Sqaured  Error',np.sqrt(mean_squared_error(testy,pred)))
```

```
Mean  Absolute  Error 2.011081468053784
Mean Squared Error 6.527810934720171
Root Mean Sqaured Error 2.5549581082123773
```

divding 70:30

In [47]:
```
trainx,testx,trainy,testy=train_test_split(x,y,test_size=0.3,random_state=69)
```

In [48]:
```
LR=LinearRegression()
LR.fit(trainx,trainy)
```

Out[48]:
```
LinearRegression()
```

In [49]:
```
pred=LR.predict(testx)
mean_squared_error(pred,testy)
```

Out[49]:
```
6.431982984501985
```

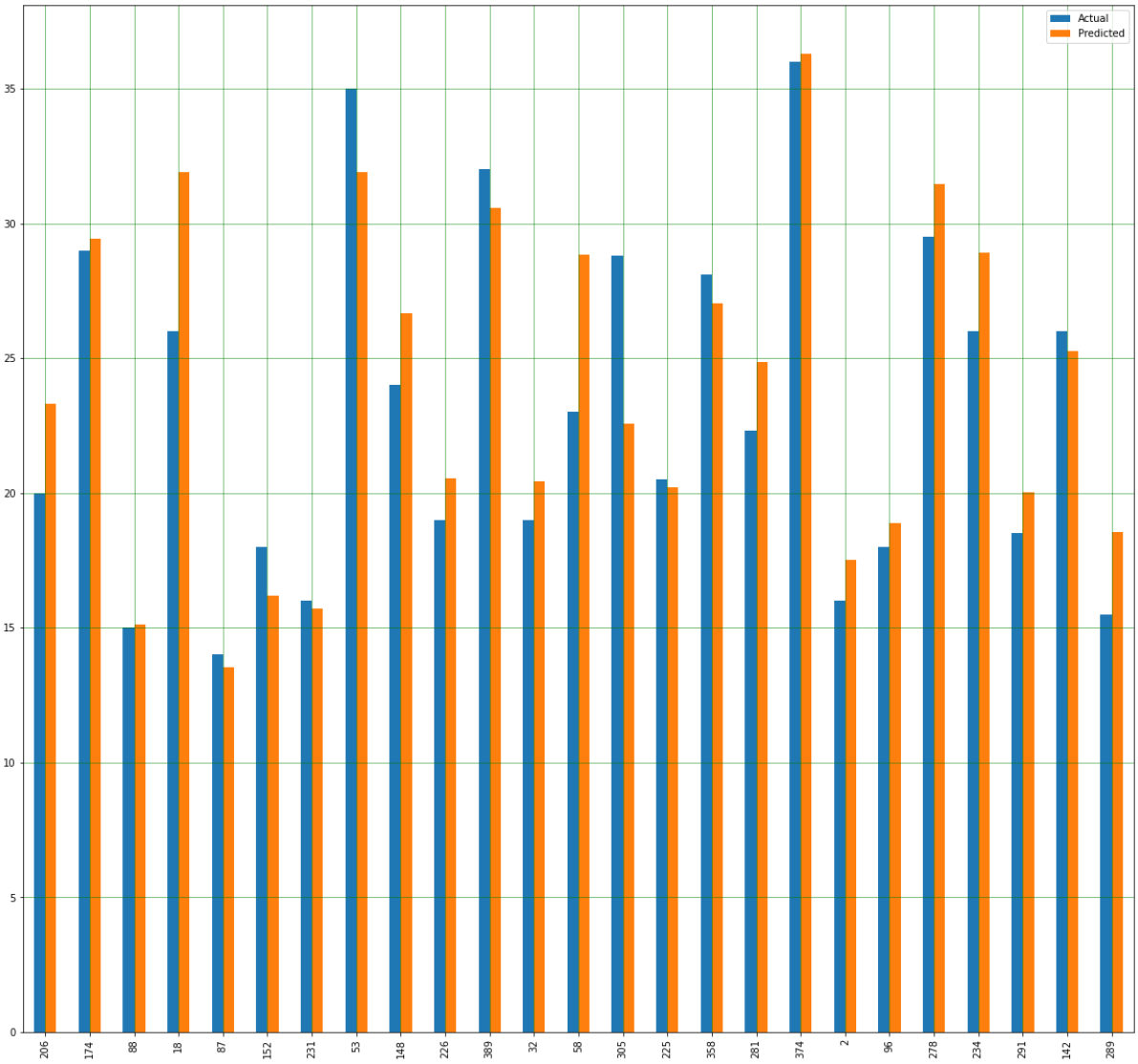
In [50]:
```
df=pd.DataFrame({'Actual':testy,'Predicted':pred})
df
```

Out[50]:

	Actual	Predicted
206	20.0	23.294768
174	29.0	29.441101
88	15.0	15.113378
18	26.0	31.893021
87	14.0	13.529668
...	...	...
136	13.0	13.382508
380	36.0	35.524091
240	22.0	25.303225
64	14.0	13.134841
77	21.0	21.874239

118 rows × 2 columns

In [51]:
```
df1=df.head(25)
df1.plot(y=['Actual','Predicted'],kind='bar',figsize=(20,19));
plt.grid(which='major',ls='-',linewidth=0.5,color='g')
plt.grid(which='minor',ls=':',linewidth=0.5,color='b')
plt.show()
```



In [52]:
```
print('Mean  Absolute  Error',mean_absolute_error(testy,pred))
print('Mean Squared Error',mean_squared_error(testy,pred))
print('Root  Mean  Sqaured  Error',np.sqrt(mean_squared_error(testy,pred)))
```

```
Mean  Absolute  Error 1.9451415688700682
Mean Squared Error 6.431982984501985
Root Mean Sqaured Error 2.536135442854341
```

dividing 90:10

In [53]:
```
trainx,testx,trainy,testy=train_test_split(x,y,test_size=0.1,random_state=69)
```

In [54]:
```
LR=LinearRegression()
LR.fit(trainx,trainy)
```

Out[54]:
```
LinearRegression()
```

In [55]:
```
pred=LR.predict(testx)
mean_squared_error(pred,testy)
```

Out[55]:
```
6.15688822298868
```

In [56]:
```
df=pd.DataFrame({'Actual':testy,'Predicted':pred})
df
```

Out[56]:

	Actual	Predicted
206	20.0	22.977681
174	29.0	29.634569
88	15.0	15.225041
18	26.0	31.670804
87	14.0	13.435147
152	18.0	16.401986
231	16.0	15.389032
53	35.0	31.704099
148	24.0	27.135909
226	19.0	20.063201
389	32.0	31.126836
32	19.0	20.326237
58	23.0	28.214223
305	28.8	22.769963
225	20.5	19.913113
358	28.1	27.360319
281	22.3	25.112825
374	36.0	36.206064
2	16.0	17.130518
96	18.0	18.759694
278	29.5	31.454056
234	26.0	28.947227
291	18.5	19.026123
142	26.0	25.685029
289	15.5	18.534911
119	19.0	21.799740
149	26.0	24.800204
218	25.5	28.050080
11	15.0	16.814223
75	18.0	21.722333
268	30.9	27.673996
10	14.0	18.412750
3	17.0	17.296470
17	27.0	28.924835
25	10.0	10.217065
304	28.4	27.673311

	Actual	Predicted
188	15.5	16.835072
15	18.0	19.636233
5	14.0	11.060353
252	20.5	19.735620

In [57]:
```
print('Mean  Absolute  Error',mean_absolute_error(testy,pred))
print('Mean Squared Error',mean_squared_error(testy,pred))
print('Root  Mean  Sqaured  Error',np.sqrt(mean_squared_error(testy,pred)))
```

```
Mean  Absolute  Error 1.9449719044323026
Mean Squared Error 6.15688822298868
Root  Mean  Sqaured  Error 2.481307764665375
```

80:20 ::: Mean Absolute Error 2.012801621835443 Mean Squared Error 6.538767175795156 Root Mean Sqaured Error 2.5571013229426707

90:10 :: Mean Absolute Error 1.8791406249999998 Mean Squared Error 6.1197006835937495 Root Mean Sqaured Error 2.473802878887837

70:30 Mean Absolute Error 1054299011303.7745 Mean Squared Error 3.0270984824050786e+25 Root Mean Sqaured Error 5501907380540.933

most least error is in 90 :10 split

In [ ]:

Practical 10 - Multivariate Linear Regression - Weather dataset

10 June 2022

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.model_selection import train_test_split
```

In [2]:
```
dfw=pd.read_csv('D:\\ML\\Weather.csv')
```

```
C:\Users\divya\AppData\Local\Temp\ipykernel_11016\135338299.py:1: DtypeWarning: Columns (7,8,18,25) have mixed types. Specify dtype option on import or set low_memory=False.
  dfw=pd.read_csv('D:\\ML\\Weather.csv')
```

In [3]:
```
dfw
```

Out[3]:

	STA	Date	Precip	WindGustSpd	MaxTemp	MinTemp	MeanTemp	Snowfall	PoorW
0	10001	1942-7-1	1.016	NaN	25.555556	22.222222	23.888889	0.0	
1	10001	1942-7-2	0	NaN	28.888889	21.666667	25.555556	0.0	
2	10001	1942-7-3	2.54	NaN	26.111111	22.222222	24.444444	0.0	
3	10001	1942-7-4	2.54	NaN	26.666667	22.222222	24.444444	0.0	
4	10001	1942-7-5	0	NaN	26.666667	21.666667	24.444444	0.0	
...	...	...	...	...	...	...	...	...	
119035	82506	1945-12-27	0	NaN	28.333333	18.333333	23.333333	0.0	
119036	82506	1945-12-28	9.906	NaN	29.444444	18.333333	23.888889	0.0	
119037	82506	1945-12-29	0	NaN	28.333333	18.333333	23.333333	0.0	
119038	82506	1945-12-30	0	NaN	28.333333	18.333333	23.333333	0.0	
119039	82506	1945-12-31	0	NaN	29.444444	17.222222	23.333333	0.0	

119040 rows × 31 columns

In [4]:
```
dfw.head()
```

Out[4]:

	STA	Date	Precip	WindGustSpd	MaxTemp	MinTemp	MeanTemp	Snowfall	PoorWeathe
0	10001	1942-7-1	1.016	NaN	25.555556	22.222222	23.888889	0.0	NaN
1	10001	1942-	0	NaN	28.888889	21.666667	25.555556	0.0	NaN
2	10001	1942-7-3	2.54	NaN	26.111111	22.222222	24.444444	0.0	NaN
3	10001	1942-	2.54	NaN	26.666667	22.222222	24.444444	0.0	NaN
4	10001	1942-7-5	0	NaN	26.666667	21.666667	24.444444	0.0	NaN

5 rows × 31 columns

In [5]:
```
dfw.describe()
```

Out[5]:

	STA	WindGustSpd	MaxTemp	MinTemp	MeanTemp	YR
count	119040.000000	532.000000	119040.000000	119040.000000	119040.000000	119040.000000
mean	29659.435795	37.774534	27.045111	17.789511	22.411631	43.805284
std	20953.209402	10.297808	8.717817	8.334572	8.297982	1.136718
min	10001.000000	18.520000	-33.333333	-38.333333	-35.555556	40.000000
25%	11801.000000	29.632000	25.555556	15.000000	20.555556	43.000000
50%	22508.000000	37.040000	29.444444	21.111111	25.555556	44.000000
75%	33501.000000	43.059000	31.666667	23.333333	27.222222	45.000000
max	82506.000000	75.932000	50.000000	34.444444	40.000000	45.000000

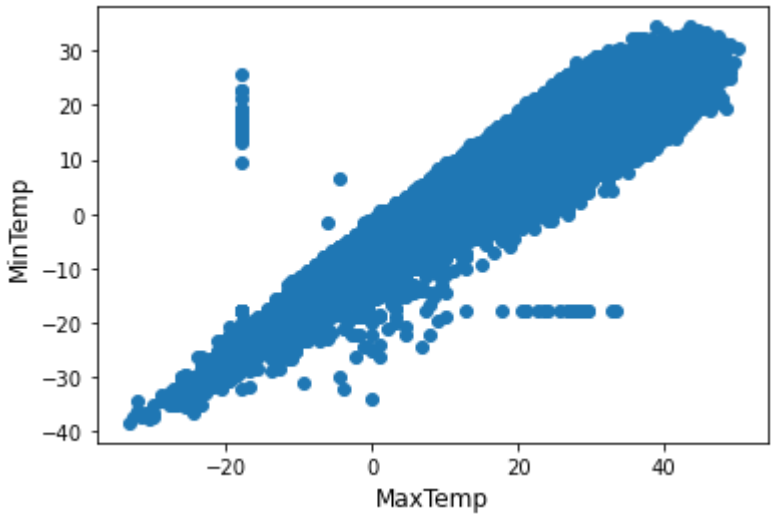
8 rows × 24 columns

In [6]:
```
dfw.shape
```

Out[6]:
```
(119040, 31)
```

In [7]:
```
x=dfw['MaxTemp']
y=dfw['MinTemp']
```

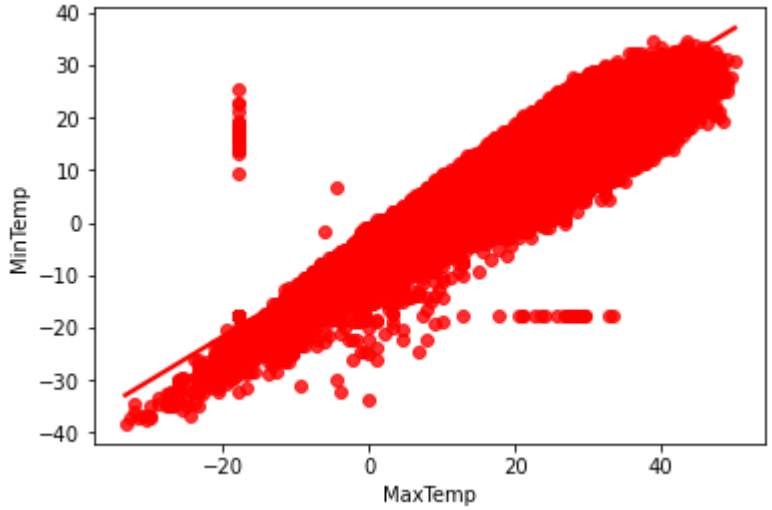
In [8]:
```
plt.scatter(x,y)
plt.xlabel('MaxTemp',fontsize='12')
plt.ylabel('MinTemp',fontsize='12')
plt.show()
```



In [9]:
```
sns.regplot(x,y,color='red')
```

```
C:\Users\[redacted]\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(
```

Out[9]:
```
<AxesSubplot:xlabel='MaxTemp', ylabel='MinTemp'>
```



In [10]:
```
x.head()
```

Out[10]:
```
0    25.555556
1    28.888889
2    26.111111
3    26.666667
4    26.666667
Name: MaxTemp, dtype: float64
```

In [11]:
```
y.head()
```

Out[11]:
```
0    22.222222
1    21.666667
2    22.222222
3    22.222222
4    21.666667
Name: MinTemp, dtype: float64
```

In [12]:
```
x.shape
```

Out[12]:
```
(119040,)
```

In [13]:
```
X_=x.values.reshape(-1,1)
```

In [14]:
```
X_.shape
```

Out[14]:
```
(119040, 1)
```

In [15]:
```
x
```

Out[15]:
```
0         25.555556
1         28.888889
2         26.111111
3         26.666667
4         26.666667
              ...
119035    28.333333
119036    29.444444
119037    28.333333
119038    28.333333
119039    29.444444
Name: MaxTemp, Length: 119040, dtype: float64
```

In [16]:
```
X_
```

Out[16]:
```
array([[25.55555556],
       [28.88888889],
       [26.11111111],
       ...,
       [28.33333333],
       [28.33333333],
       [29.44444444]])
```

Model

In [17]:
```
X_train,X_test,y_train,y_test=train_test_split(X_,y,test_size=0.2,random_state=30)
```

In [18]:
```
X_train.shape
```

Out[18]:
```
(95232, 1)
```

In [19]:
```
X_test.shape
```

Out[19]:
```
(23808, 1)
```

In [20]:
```
LR=LinearRegression()
LR.fit(X_train,y_train)
```

Out[20]:
```
LinearRegression()
```

In [21]:
```
y_pred=LR.predict(X_test)
```

In [22]:
```
y_test
```

Out[22]:
```
39071     12.222222
5109      22.222222
1113      22.222222
117003     7.777778
106549    26.111111
            ...
83309     12.222222
75290      6.111111
62785      8.888889
80737      6.111111
25569     22.777778
Name: MinTemp, Length: 23808, dtype: float64
```

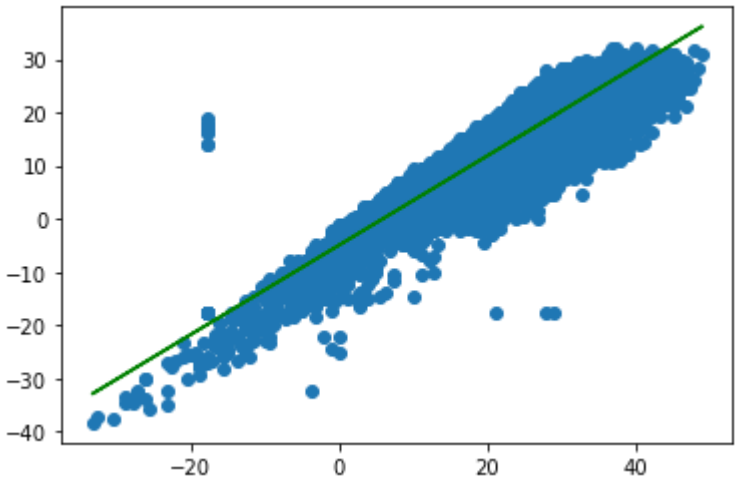
In [23]:
```
y_pred
```

Out[23]:
```
array([14.66774639, 22.13310546, 18.86701087, ..., 15.13433134,
        9.53531203, 20.2667657 ])
```

In [24]:
```
weights = LR.coef_
intercept = LR.intercept_
print(weights,intercept)
```

```
[0.8398529] -4.928821159365992
```

In [25]:
```
plt.scatter(X_test, y_test)
plt.plot(X_test,y_pred, color='green')
plt.show()
```



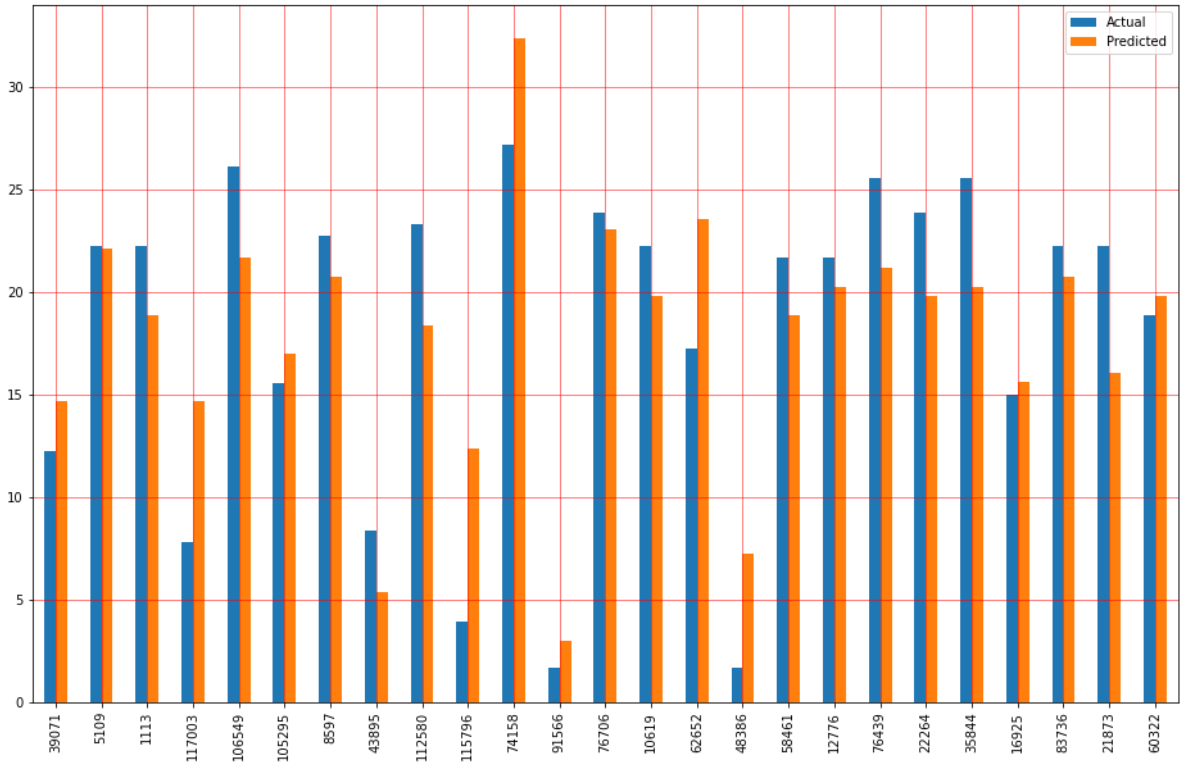
In [26]:
```
df=pd.DataFrame({'Actual':y_test,'Predicted':y_pred})
df
```

Out[26]:

	Actual	Predicted
39071	12.222222	14.667746
5109	22.222222	22.133105
1113	22.222222	18.867011
117003	7.777778	14.667746
106549	26.111111	21.666521
...	...	...
83309	12.222222	11.868237
75290	6.111111	11.401652
62785	8.888889	15.134331
80737	6.111111	9.535312
25569	22.777778	20.266766

23808 rows × 2 columns

In [27]:
```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='red')
plt.grid(which='minor',linestyle=':',linewidth='0.5',color='green')
plt.show()
```



In [28]:
```
print('Mean  Absolute  Error',mean_absolute_error(y_test,y_pred))
print('Mean  Squared  Error',mean_squared_error(y_test,y_pred))
print('Root  Mean  Sqaured  Error',np.sqrt(mean_squared_error(y_test,y_pred)))
```

```
Mean  Absolute  Error 3.0860711477073064
Mean  Squared  Error 15.642509497194942
Root  Mean  Sqaured  Error 3.955061250751364
```

80:20 ::: Mean Absolute Error 3.086071147707306 Mean Squared Error 15.642509497194942 Root Mean Sqaured Error 3.955061250751364

90:10 :: Mean Absolute Error 3.1047256786542192 Mean Squared Error 15.83161054325677 Root Mean Sqaured Error 3.97889564367511

70:30 Mean Absolute Error 3.1013916713826886 Mean Squared Error 15.768060731561533 Root Mean Sqaured Error 3.970901752947501

80:20 split has most least error compared to others

In [ ]:

Practical 11 - Logistic Regression

13 June 2022

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
```

In [2]:
```
df=pd.read_csv('D:\\ML\\insurance_data.csv')
```

In [3]:
```
df
```

Out[3]:

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1
8	62	1
9	61	1
10	18	0
11	28	0
12	27	0
13	29	0
14	49	1
15	55	1
16	25	1
17	58	1
18	19	0
19	18	0
20	21	0
21	26	0
22	40	1
23	45	1
24	50	1
25	54	1
26	23	0

In [4]:
```
df.describe()
```

Out[4]:

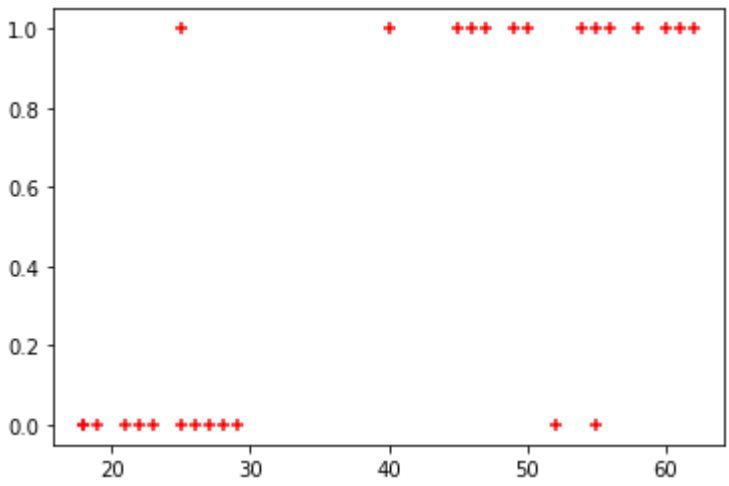
	age	bought_insurance
count	27.000000	27.000000
mean	39.666667	0.518519
std	15.745573	0.509175
min	18.000000	0.000000
25%	25.000000	0.000000
50%	45.000000	1.000000
75%	54.500000	1.000000
max	62.000000	1.000000

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

Out[5]:
```
(27, 2)
```

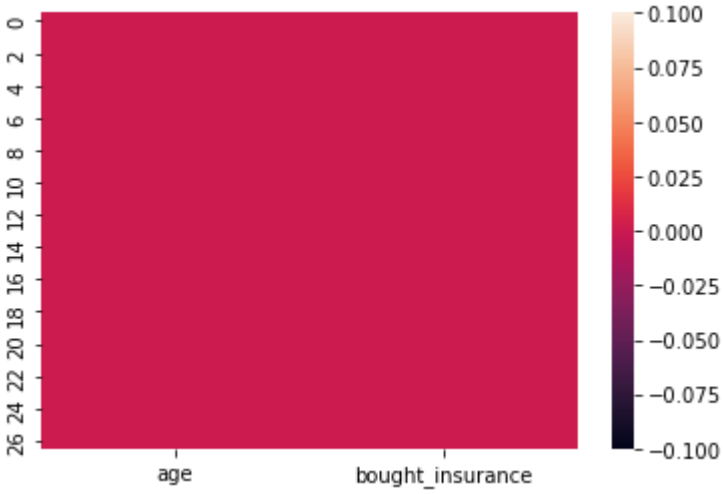
In [6]:
```
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
```

Out[6]:
```
<matplotlib.collections.PathCollection  at  0x19a00b08fd0>
```



In [7]:
```
sns.heatmap(df.isnull())
```

Out[7]:
```
<AxesSubplot:>
```



No null values are there

In [8]:
```
X=df.drop('bought_insurance',axis='columns')
X
```

Out[8]:

	age
0	22
1	25
2	47
3	52
4	46
5	56
6	55
7	60
8	62
9	61
10	18
11	28
12	27
13	29
14	49
15	55
16	25
17	58
18	19
19	18
20	21
21	26
22	40
23	45
24	50
25	54
26	23

In [9]:
```
X.shape
```

Out[9]:
```
(27, 1)
```

In [10]:
```
Y=df.drop('age',axis='columns')
Y
```

Out[10]:

	bought_insurance
0	0
1	0
2	1
3	0
4	1
5	1
6	0
7	1
8	1
9	1
10	0
11	0
12	0
13	0
14	1
15	1
16	1
17	1
18	0
19	0
20	0
21	0
22	1
23	1
24	1
25	1
26	0

In [11]:
```
Y.shape
```

Out[11]:
```
(27, 1)
```

Dividing

In [12]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=40)
```

In [13]:
```
model=LogisticRegression()
model.fit(X_train,Y_train)
```

```
C:\Users\gaurav\anaconda3\lib\site-packages\sklearn\utils\validation.py:993:  DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[13]:
```
LogisticRegression()
```

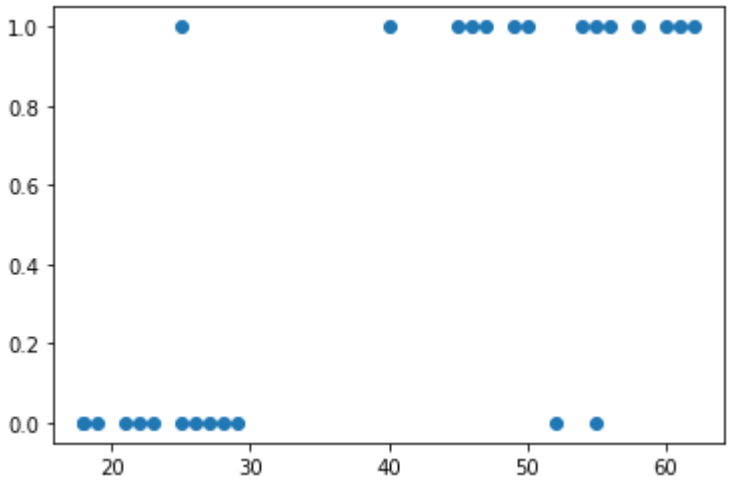
In [14]:
```
y_pred=model.predict(X_test)
```

In [15]:
```
y_pred
```

Out[15]:
```
array([1, 0, 0, 1, 1, 0], dtype=int64)
```

In [16]:
```
plt.scatter(df.age,df.bought_insurance)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x19a00d3a940>
```



In [18]:
```
y_pred.shape
```

Out[18]:
```
(6,)
```

In [19]:
```
y_pred
```

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

In [20]:
```
print(classification_report(Y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.67      1.00      0.80         2
           1       1.00      0.75      0.86         4

    accuracy                           0.83         6
   macro avg       0.83      0.88      0.83         6
weighted avg       0.89      0.83      0.84         6
```

In [21]:
```
print(confusion_matrix(Y_test,y_pred))
```

```
[[2 0]
 [1 3]]
```

In [ ]:

Practical 12 - Logistic Regression - Diabetes dataset

13 June 2022

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
```

In [4]:
```
df=pd.read_csv('D:\\Ml\\diabetes.csv')
```

In [5]:
```
df
```

Out[5]:

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

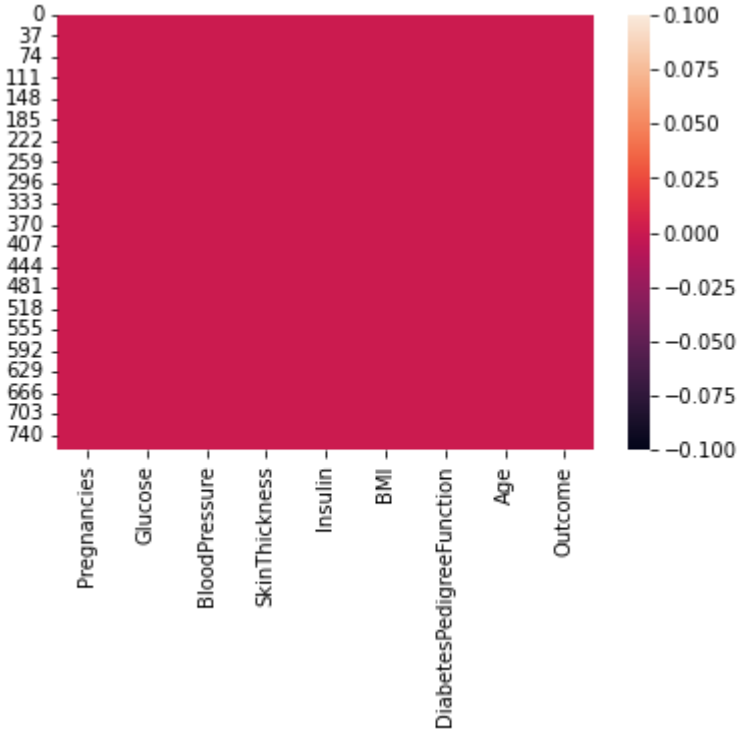
768 rows × 9 columns

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---                            ------------------
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [7]:
```
sns.heatmap(df.isnull())
```

Out[7]:
```
<AxesSubplot:>
```



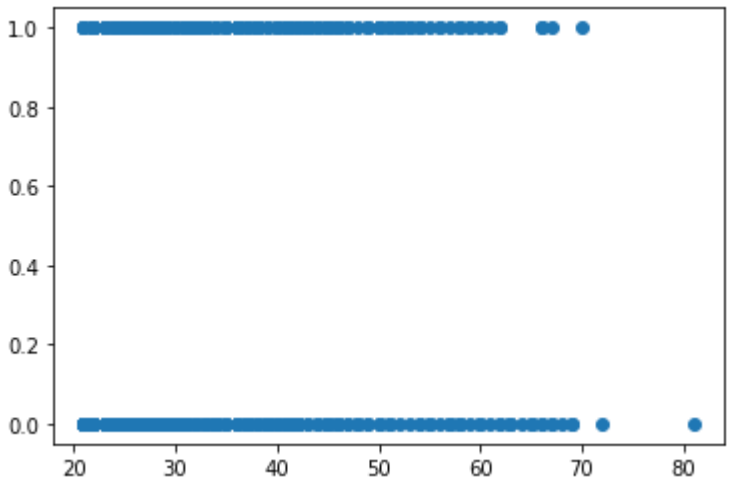
In [8]:
```
df.describe()
```

Out[8]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [9]:
```
plt.scatter(df.Age,df.Outcome)
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x1abf2598280>
```



In [10]:
```
Y=df.Outcome
```

In [11]:
```
X=df.drop(['Outcome'],axis=1)
```

In [12]:
```
X
```

Out[12]:

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

768 rows × 8 columns

In [13]:
```
Y
```

Out[13]:
```
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
```

80:20

In [14]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

In [15]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

```
C:\Users\gaurav\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814:
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 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(
```

Out[15]:
```
LogisticRegression()
```

In [16]:
```
y_pred=model.predict(X_test)
```

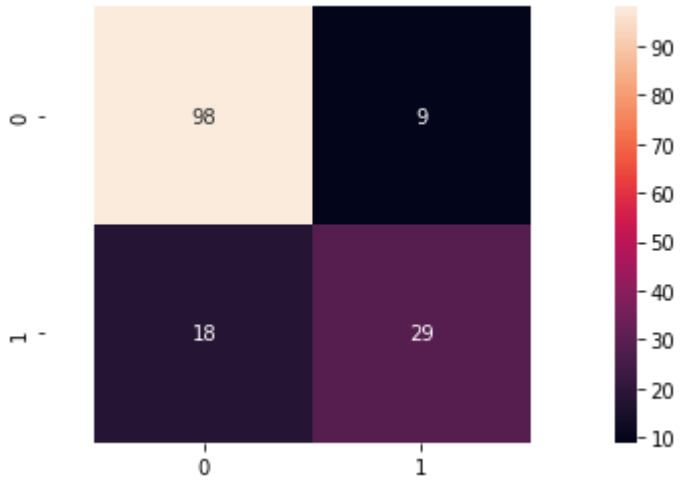
In [17]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       107
           1       0.76      0.62      0.68        47

    accuracy                           0.82       154
   macro avg       0.80      0.77      0.78       154
weighted avg       0.82      0.82      0.82       154
```

In [18]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[18]:
```
array([[98,  9],
       [18, 29]], dtype=int64)
```

In [19]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



70:30

In [20]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=0)
```

In [21]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

```
C:\Users\gaurav\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814:
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 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(
```

Out[21]:
```
LogisticRegression()
```

In [22]:
```
y_pred=model.predict(X_test)
```

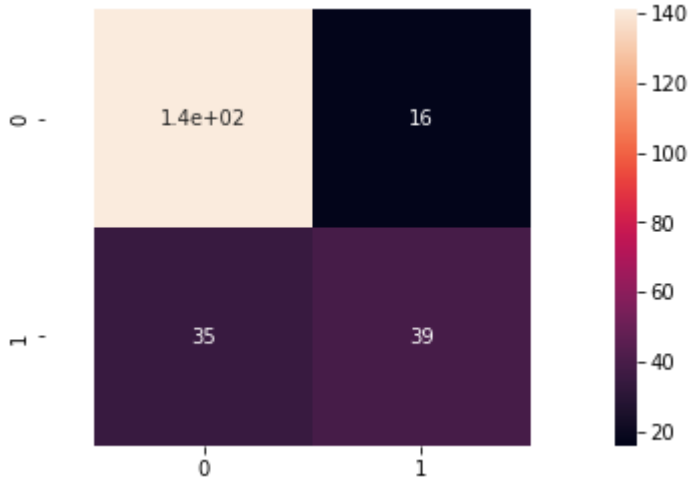
In [23]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       157
           1       0.71      0.53      0.60        74

    accuracy                           0.78       231
   macro avg       0.76      0.71      0.73       231
weighted avg       0.77      0.78      0.77       231
```

In [24]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[24]:
```
array([[141,  16],
       [ 35,  39]], dtype=int64)
```

In [25]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



90:10

In [26]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.1,random_state=0)
```

In [27]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

```
C:\Users\gaurav\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814:
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 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(
```

Out[27]:
```
LogisticRegression()
```

In [28]:
```
y_pred=model.predict(X_test)
```

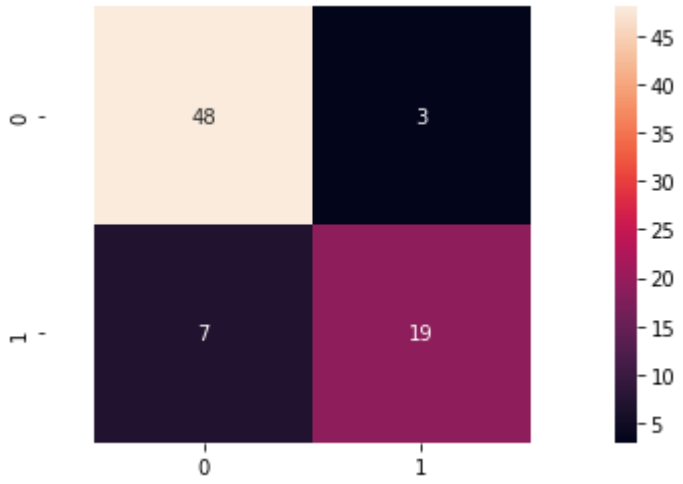
In [29]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.94      0.91        51
           1       0.86      0.73      0.79        26

    accuracy                           0.87        77
   macro avg       0.87      0.84      0.85        77
weighted avg       0.87      0.87      0.87        77
```

In [30]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[30]:
```
array([[48,  3],
       [ 7, 19]], dtype=int64)
```

In [31]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



60:40

In [32]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.4,random_state=0)
```

In [33]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

```
C:\Users\gaurav\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814:
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 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(
```

Out[33]:
```
LogisticRegression()
```

In [34]:
```
y_pred=model.predict(X_test)
```

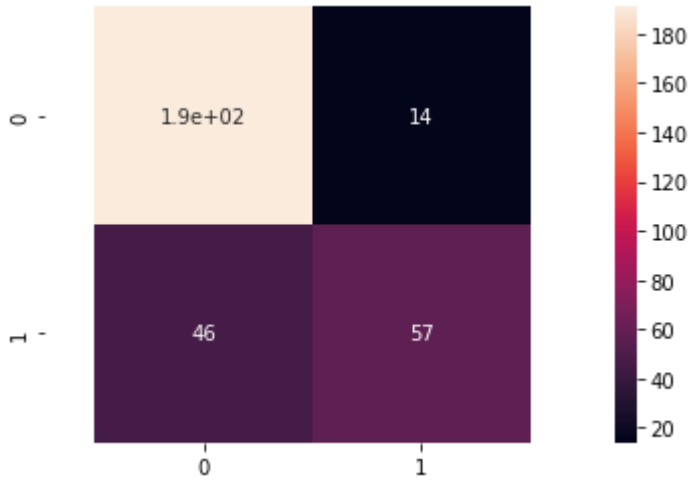
In [35]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.93      0.86       205
           1       0.80      0.55      0.66       103

    accuracy                           0.81       308
   macro avg       0.80      0.74      0.76       308
weighted avg       0.80      0.81      0.79       308
```

In [36]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[36]:
```
array([[191,  14],
       [ 46,  57]], dtype=int64)
```

In [37]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



50:50

In [38]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.5,random_state=0)
```

In [39]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

```
C:\Users\gaurav\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814:
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 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(
```

Out[39]:
```
LogisticRegression()
```

In [40]:
```
y_pred=model.predict(X_test)
```

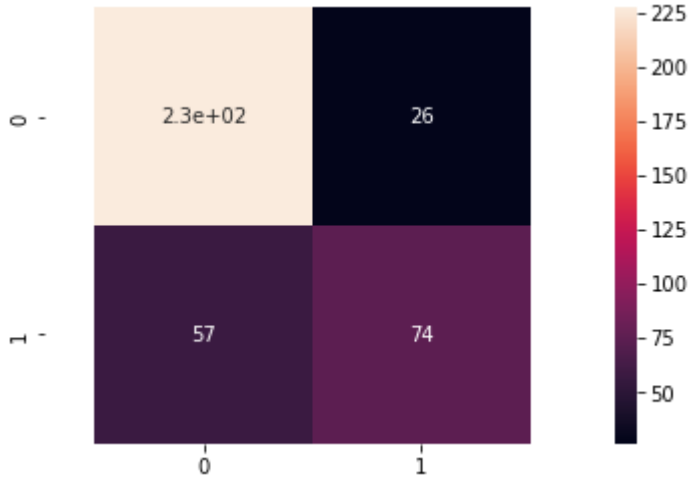
In [41]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       253
           1       0.74      0.56      0.64       131

    accuracy                           0.78       384
   macro avg       0.77      0.73      0.74       384
weighted avg       0.78      0.78      0.78       384
```

In [42]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[42]:
```
array([[227,  26],
       [ 57,  74]], dtype=int64)
```

In [43]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



Standardized

In [44]:
```
from sklearn import preprocessing
#get col names first
names=X.columns
#create the Scalar Object
scaler=preprocessing.StandardScaler()
#fit your data on the scaler obj
scaled_df=scaler.fit_transform(X)
X=pd.DataFrame(scaled_df,columns=names)
```

In [45]:
```
X
```

Out[45]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	0.639947	0.848324	0.149641	0.907270	-0.692891	0.204013	
1	-0.844885	-1.123396	-0.160546	0.530902	-0.692891	-0.684422	
2	1.233880	1.943724	-0.263941	-1.288212	-0.692891	-1.103255	
3	-0.844885	-0.998208	-0.160546	0.154533	0.123302	-0.494043	
4	-1.141852	0.504055	-1.504687	0.907270	0.765836	1.409746	
...	...	...	...	...	...	...	
763	1.827813	-0.622642	0.356432	1.722735	0.870031	0.115169	
764	-0.547919	0.034598	0.046245	0.405445	-0.692891	0.610154	
765	0.342981	0.003301	0.149641	0.154533	0.279594	-0.735190	
766	-0.844885	0.159787	-0.470732	-1.288212	-0.692891	-0.240205	
767	-0.844885	-0.873019	0.046245	0.656358	-0.692891	-0.202129	

768 rows × 8 columns

80:20

In [46]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=0)
```

In [47]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

Out[47]:
```
LogisticRegression()
```

In [48]:
```
y_pred=model.predict(X_test)
```

In [49]:
```
y_pred
```

Out[49]:
```
array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
      dtype=int64)
```

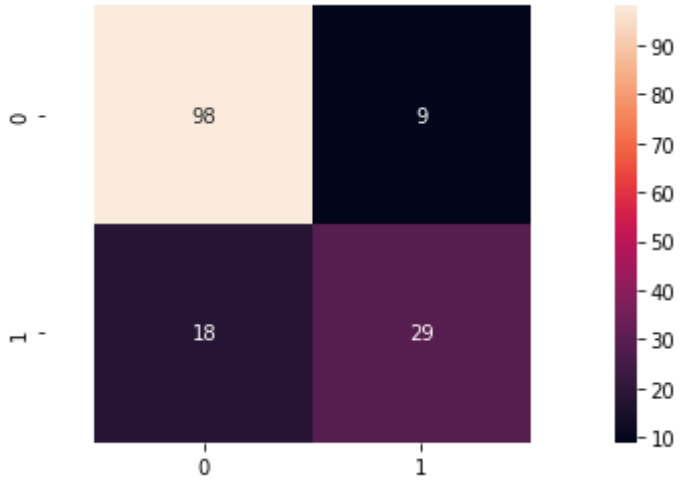
In [50]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.92      0.88       107
           1       0.76      0.62      0.68        47

    accuracy                           0.82       154
   macro avg       0.80      0.77      0.78       154
weighted avg       0.82      0.82      0.82       154
```

In [51]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[51]:
```
array([[98,  9],
       [18, 29]], dtype=int64)
```

In [52]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



70:30

In [53]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=0)
```

In [54]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

Out[54]:
```
LogisticRegression()
```

In [55]:
```
y_pred=model.predict(X_test)
```

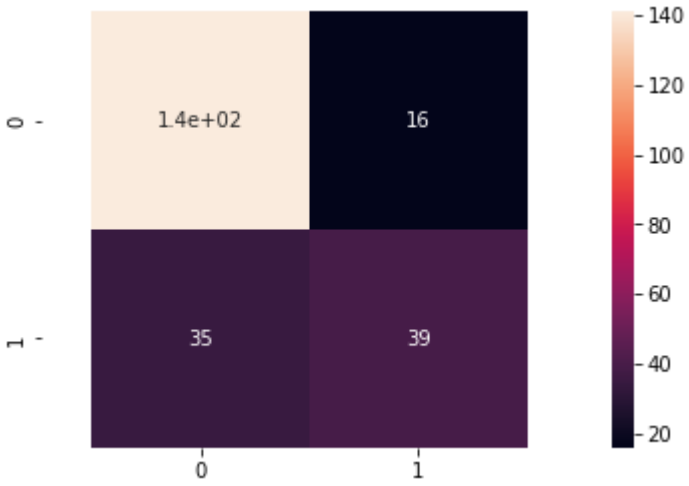
In [56]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       157
           1       0.71      0.53      0.60        74

    accuracy                           0.78       231
   macro avg       0.76      0.71      0.73       231
weighted avg       0.77      0.78      0.77       231
```

In [57]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[57]:
```
array([[141,  16],
       [ 35,  39]], dtype=int64)
```

In [58]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



90:10

In [59]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.1,random_state=0)
```

In [60]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

Out[60]:
```
LogisticRegression()
```

In [61]:
```
y_pred=model.predict(X_test)
```

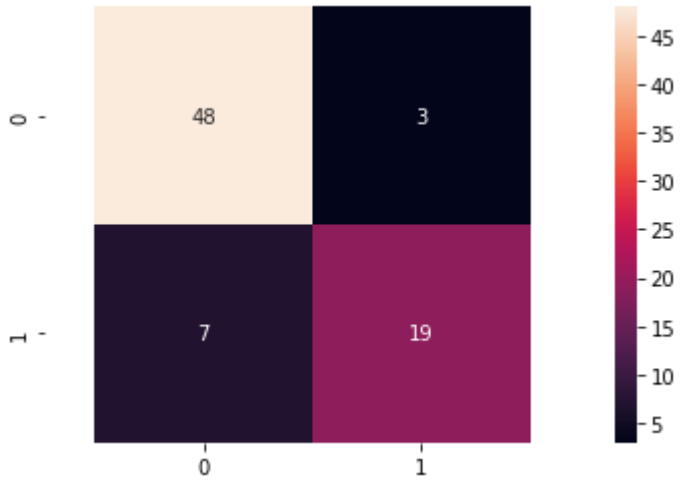
In [62]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.94      0.91        51
           1       0.86      0.73      0.79        26

    accuracy                           0.87        77
   macro avg       0.87      0.84      0.85        77
weighted avg       0.87      0.87      0.87        77
```

In [63]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[63]:
```
array([[48,  3],
       [ 7, 19]], dtype=int64)
```

In [64]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



50 :50

In [65]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.5,random_state=0)
```

In [66]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

Out[66]:
```
LogisticRegression()
```

In [67]:
```
y_pred=model.predict(X_test)
```

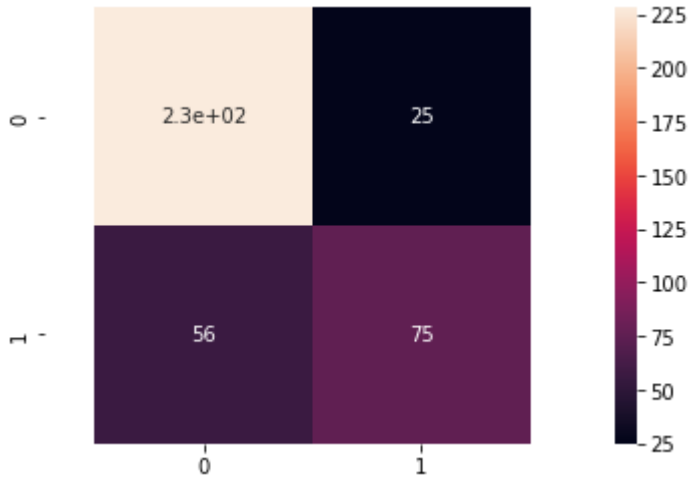
In [68]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.90      0.85       253
           1       0.75      0.57      0.65       131

    accuracy                           0.79       384
   macro avg       0.78      0.74      0.75       384
weighted avg       0.78      0.79      0.78       384
```

In [69]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[69]:
```
array([[228,  25],
       [ 56,  75]], dtype=int64)
```

In [70]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



60:40

In [71]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.4,random_state=0)
```

In [72]:
```
model=LogisticRegression()
model.fit(X_train,y_train)
```

Out[72]:
```
LogisticRegression()
```

In [73]:
```
y_pred=model.predict(X_test)
```

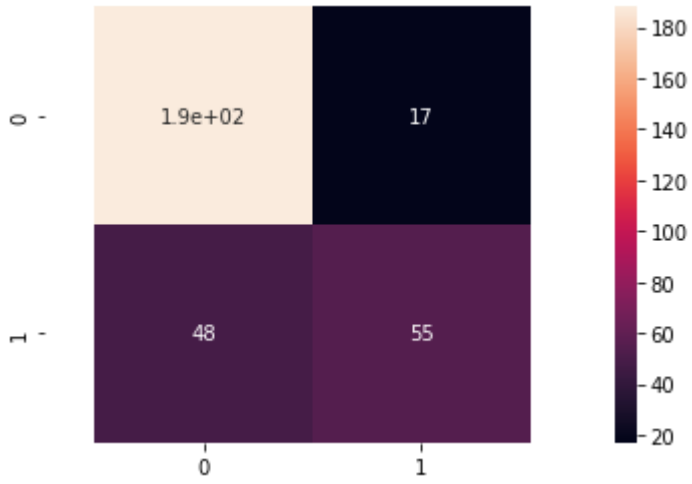
In [74]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.92      0.85       205
           1       0.76      0.53      0.63       103

    accuracy                           0.79       308
   macro avg       0.78      0.73      0.74       308
weighted avg       0.79      0.79      0.78       308
```

In [75]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[75]:
```
array([[188,  17],
       [ 48,  55]], dtype=int64)
```

In [76]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



Most accurate model is of split 90:10 with accuracy of 87%

In [ ]:

Practical 13 - Logistic Regression - Titanic dataset

13 June 2022

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
```

In [2]:
```
df=pd.read_csv('D:\\ML\\titanic_train.csv')
```

In [3]:
```
df
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Gender	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, M ss. L ina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futr lle, Mrs. Jacques eath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. Wil iam H nry	male	35.0	0	0	373450	8.0500
...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Mont ila, Rev. Ju zas	male	27.0	0	0	211536	13.0000
887	888	1	1	Grah am, M ss. Marg ret Edith	female	19.0	0	0	112053	30.0000
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500

891 rows × 12 columns

In [4]:
```
df.describe()
```

Out[4]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

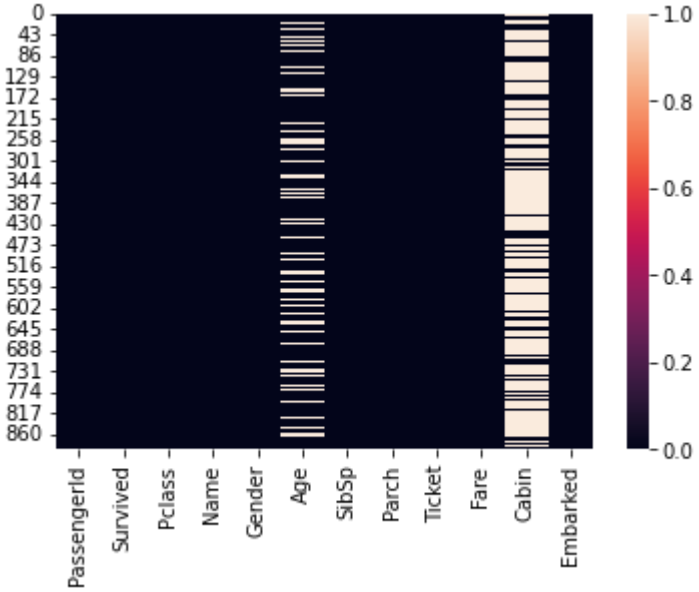
In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Gender       891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

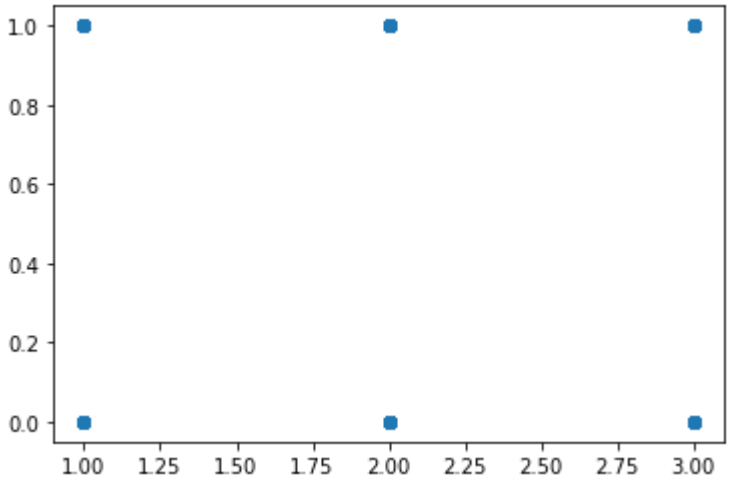
In [6]:
```
sns.heatmap(df.isnull())
```

Out[6]:

```
<AxesSubplot:>
```



Out[7]: <matplotlib.collections.PathCollection at 0x2728733dd90>



In [8]:
```
percnan=pd.DataFrame([(col,df[col].isna().mean()*100) for col in df],columns=["Fea
percnan
```

Out[8]:

	Feature	Percentage
0	PassengerId	0.000000
1	Survived	0.000000
2	Pclass	0.000000
3	Name	0.000000
4	Gender	0.000000
5	Age	19.865320
6	SibSp	0.000000
7	Parch	0.000000
8	Ticket	0.000000
9	Fare	0.000000
10	Cabin	77.104377
11	Embarked	0.224467

Age , Cabin and Emabrked contains Nan Values

Age approx 20% Nan values and Embarked contains 0.22% of Nan so we can replace those Nan values with some values *CABIN will be dropped*

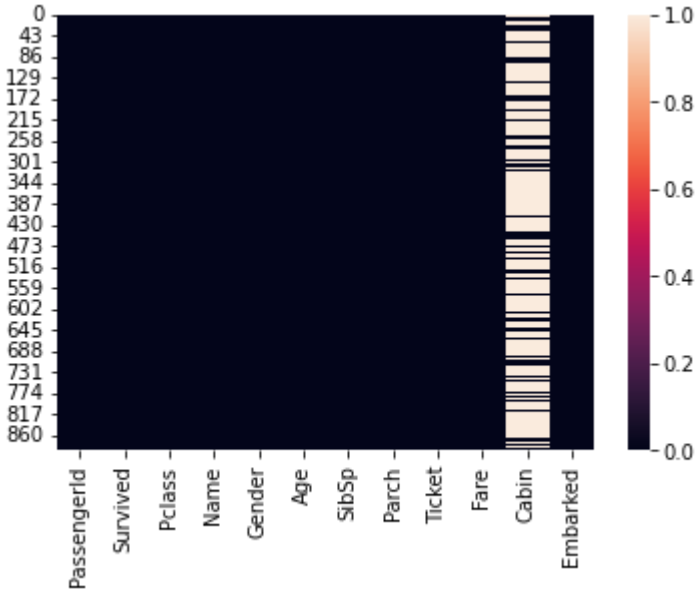
In [9]:
```
df['Age']=df['Age'].fillna(df['Age'].mean()) ## fillna will replace the age nan va
```

In [10]:
```
df['Embarked'].mode()
```

Out[10]:
```
0    S
Name: Embarked, dtype: object
```

In [11]:
```
df['Embarked']=df['Embarked'].fillna('S')
```

In [12]:
```
sns.heatmap(df.isnull())
```



No Nan values in AGE and Embarked

In [13]:
```
gd=pd.get_dummies(df['Gender'])
```

In [14]:
```
gd
```

Out[14]:

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1
...	...	...
886	0	1
887	1	0
888	1	0
889	0	1
890	0	1

891 rows × 2 columns

In [15]:
```
emb=pd.get_dummies(df['Embarked'])
```

In [16]:
```
emb
```

Out[16]:

	C	Q	S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
886	0	0	1
887	0	0	1
888	0	0	1
889	1	0	0
890	0	1	0

891 rows × 3 columns

In [17]:
```
df=df.drop(['PassengerId','Name','Gender','Ticket','Embarked'],axis=1)
df
```

Out[17]:

	Survived	Pclass	Age	SibSp	Parch	Fare	Cabin
0	0	3	22.000000	1	0	7.2500	NaN
1	1	1	38.000000	1	0	71.2833	C85
2	1	3	26.000000	0	0	7.9250	NaN
3	1	1	35.000000	1	0	53.1000	C123
4	0	3	35.000000	0	0	8.0500	NaN
...	...	...	...	...	...	...	...
886	0	2	27.000000	0	0	13.0000	NaN
887	1	1	19.000000	0	0	30.0000	B42
888	0	3	29.699118	1	2	23.4500	NaN
889	1	1	26.000000	0	0	30.0000	C148
890	0	3	32.000000	0	0	7.7500	NaN

891 rows × 7 columns

In [18]:
```
df=df.drop(['Cabin'],axis=1)
df
```

Out[18]:

	Survived	Pclass	Age	SibSp	Parch	Fare
0	0	3	22.000000	1	0	7.2500
1	1	1	38.000000	1	0	71.2833
2	1	3	26.000000	0	0	7.9250
3	1	1	35.000000	1	0	53.1000
4	0	3	35.000000	0	0	8.0500
...	...	...	...	...	...	...
886	0	2	27.000000	0	0	13.0000
887	1	1	19.000000	0	0	30.0000
888	0	3	29.699118	1	2	23.4500
889	1	1	26.000000	0	0	30.0000
890	0	3	32.000000	0	0	7.7500

891 rows × 6 columns

In [19]:
```
titanic=pd.concat([df,gd,emb],axis=1)
```

In [20]:
```
titanic
```

Out[20]:

	Survived	Pclass	Age	SibSp	Parch	Fare	female	male	C	Q	S
0	0	3	22.000000	1	0	7.2500	0	1	0	0	1
1	1	1	38.000000	1	0	71.2833	1	0	1	0	0
2	1	3	26.000000	0	0	7.9250	1	0	0	0	1
3	1	1	35.000000	1	0	53.1000	1	0	0	0	1
4	0	3	35.000000	0	0	8.0500	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	27.000000	0	0	13.0000	0	1	0	0	1
887	1	1	19.000000	0	0	30.0000	1	0	0	0	1
888	0	3	29.699118	1	2	23.4500	1	0	0	0	1
889	1	1	26.000000	0	0	30.0000	0	1	1	0	0
890	0	3	32.000000	0	0	7.7500	0	1	0	1	0

891 rows × 11 columns

In [21]:
```
Y=titanic.Survived
```

In [22]:
```
X=titanic.drop(['Survived'],axis=1)
```

In [23]:
```
X
```

Out[23]:

	Pclass	Age	SibSp	Parch	Fare	female	male	C	Q	S
0	3	22.000000	1	0	7.2500	0	1	0	0	1
1	1	38.000000	1	0	71.2833	1	0	1	0	0
2	3	26.000000	0	0	7.9250	1	0	0	0	1
3	1	35.000000	1	0	53.1000	1	0	0	0	1
4	3	35.000000	0	0	8.0500	0	1	0	0	1
...	...	...	...	...	...	...	...	...	...	...
886	2	27.000000	0	0	13.0000	0	1	0	0	1
887	1	19.000000	0	0	30.0000	1	0	0	0	1
888	3	29.699118	1	2	23.4500	1	0	0	0	1
889	1	26.000000	0	0	30.0000	0	1	1	0	0
890	3	32.000000	0	0	7.7500	0	1	0	1	0

891 rows × 10 columns

In [24]:
```
Y
```

Out[24]:
```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

we need to standarized the values

In [25]:
```
from sklearn import preprocessing
#get col names first
names=X.columns
#create the Scalar Object
scaler=preprocessing.StandardScaler()
#fit your data on the scaler obj
scaled_df=scaler.fit_transform(X)
X=pd.DataFrame(scaled_df,columns=names)
```

In [26]:
```
X
```

Out[26]:

	Pclass	Age	SibSp	Parch	Fare	female	male	C	
0	0.827377	-0.592481	0.432793	-0.473674	-0.502445	-0.737695	0.737695	-0.482043	-0.307
1	-1.566107	0.638789	0.432793	-0.473674	0.786845	1.355574	-1.355574	2.074505	-0.307
2	0.827377	-0.284663	-0.474545	-0.473674	-0.488854	1.355574	-1.355574	-0.482043	-0.307
3	-1.566107	0.407926	0.432793	-0.473674	0.420730	1.355574	-1.355574	-0.482043	-0.307
4	0.827377	0.407926	-0.474545	-0.473674	-0.486337	-0.737695	0.737695	-0.482043	-0.307
...	...	...	...	...	...	...	...	...	
886	-0.369365	-0.207709	-0.474545	-0.473674	-0.386671	-0.737695	0.737695	-0.482043	-0.307
887	-1.566107	-0.823344	-0.474545	-0.473674	-0.044381	1.355574	-1.355574	-0.482043	-0.307
888	0.827377	0.000000	0.432793	2.008933	-0.176263	1.355574	-1.355574	-0.482043	-0.307
889	-1.566107	-0.284663	-0.474545	-0.473674	-0.044381	-0.737695	0.737695	2.074505	-0.307
890	0.827377	0.177063	-0.474545	-0.473674	-0.492378	-0.737695	0.737695	-0.482043	3.251

891 rows × 10 columns

80:20

In [27]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=69)
```

In [28]:
```
model=LogisticRegression()
model.fit(X_train,y_train);
```

In [29]:
```
y_pred=model.predict(X_test)
```

In [30]:
```
y_pred
```

Out[30]:
```
array([1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
```

In [31]:

In [32]:
```
print(confusion_matrix(y_test,y_pred))
```

```
array([1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
[[95 17]
 [18 49]]
```

```
0, 0,           1       0.74      0.73      0.74        67
0],
dtyp     accuracy                           0.80       179
e=in    macro avg       0.79      0.79      0.79       179
t64) weighted avg       0.80      0.80      0.80       179
```

```
print(classification_report(y_test,y_pred))
```

```
    0       0.84      0.85      0.84       112
```

90:10

In [33]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.1,random_state=69)
```

In [34]:
```
model=LogisticRegression()
model.fit(X_train,y_train);
```

In [35]:
```
y_pred=model.predict(X_test)
```

In [36]:
```
y_pred
```

Out[36]:
```
       0, 0], dtype=int64)
```

In [37]:
```
print(classification_report(y_test,y_pred))
```

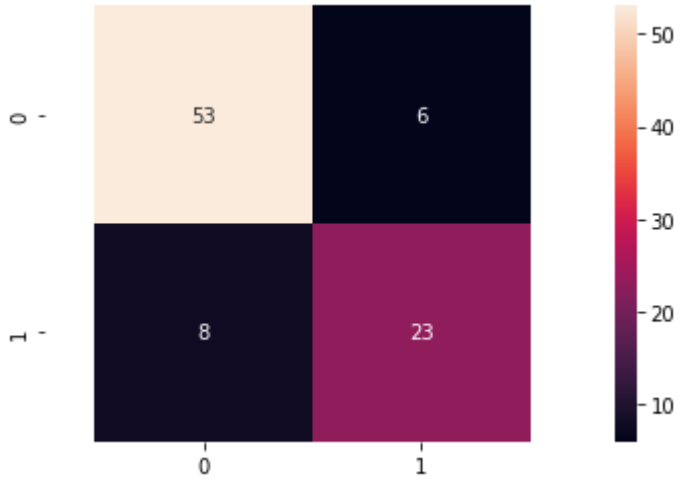
```
              precision    recall  f1-score   support

           0       0.87      0.90      0.88        59
           1       0.79      0.74      0.77        31

    accuracy                           0.84        90
   macro avg       0.83      0.82      0.82        90
weighted avg       0.84      0.84      0.84        90
```

In [38]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

```
array([1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 0,
```

Out[38]:
```
array([[53,  6],
       [ 8, 23]], dtype=int64)
```

In [39]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



50 : 50

In [40]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.5,random_state=69)
```

In [41]:
```
model=LogisticRegression()
```

In [42]:
```
y_pred=model.predict(X_test)
```

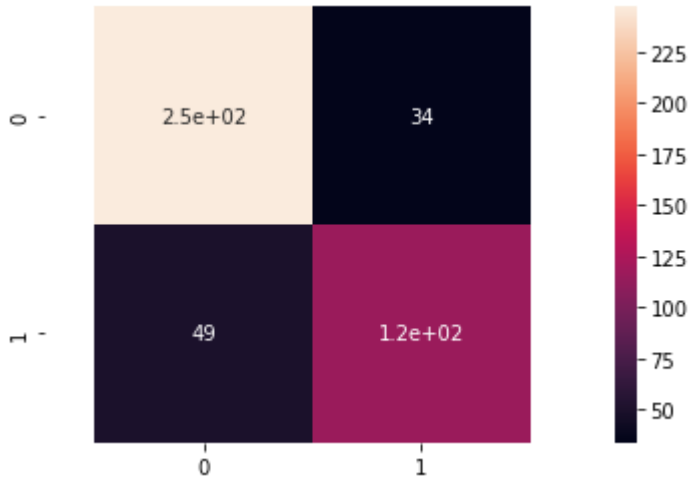
In [43]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.88      0.86       281
           1       0.77      0.70      0.74       165

    accuracy                           0.81       446
   macro avg       0.80      0.79      0.80       446
weighted avg       0.81      0.81      0.81       446
```

In [44]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[44]:
```
array([[247,  34],
       [ 49, 116]], dtype=int64)
```

In [45]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



70:30

In [46]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=69)
```

In [47]:
```
model=LogisticRegression()
model.fit(X_train,y_train);
```

In [48]:
```
y_pred=model.predict(X_test)
```

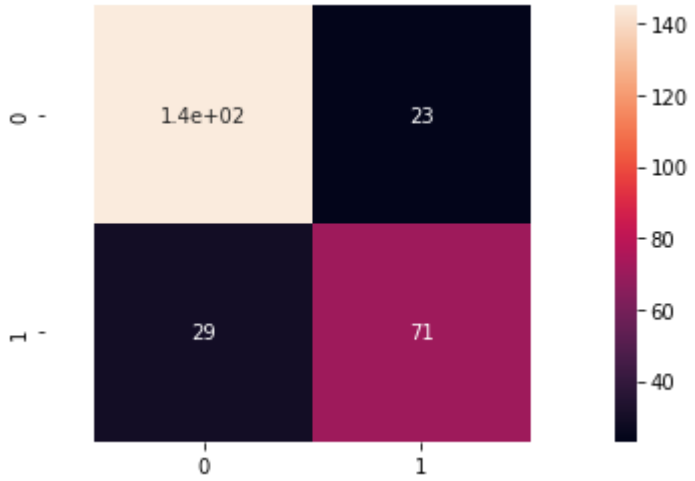
In [49]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.86      0.85       168
           1       0.76      0.71      0.73       100

    accuracy                           0.81       268
   macro avg       0.79      0.79      0.79       268
weighted avg       0.80      0.81      0.80       268
```

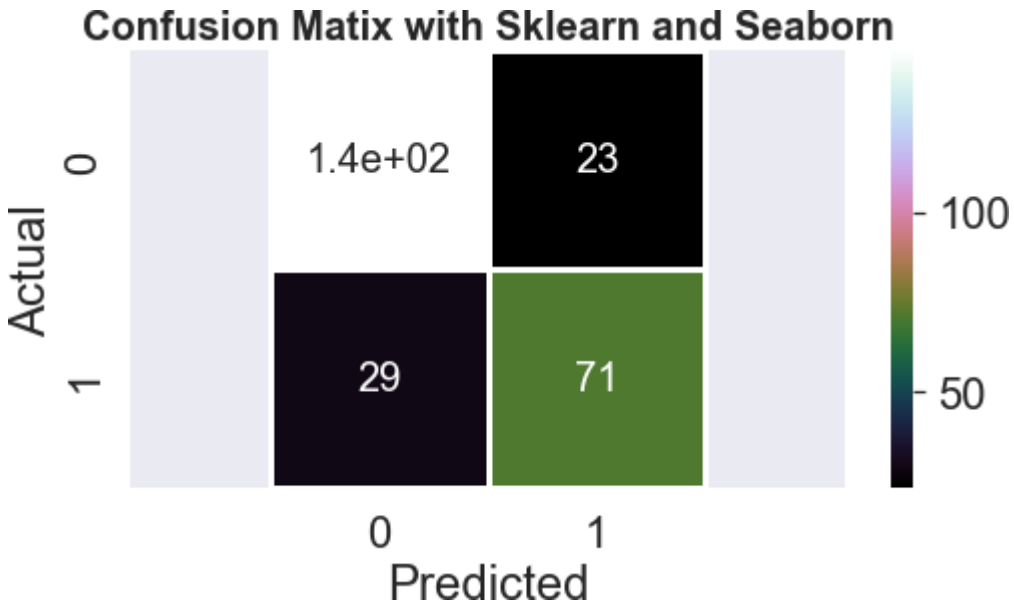
In [50]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[50]:
```
array([[145,  23],
       [ 29,  71]], dtype=int64)
```

In [51]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



In [52]:
```
with  plt.style.context('seaborn'):
    plt.figure(figsize=(8,4))
    sns.set(font_scale=2)
    sns.heatmap(cf,annot=True,square=True,annot_kws={"size":20},linewidth=3,cmap='
    plt.xlabel('Predicted');
    plt.ylabel('Actual');
    plt.axis('Equal');
    plt.title('Confusion Matix with Sklearn and Seaborn',fontweight='bold',fontsiz
    plt.show()
```



In [ ]:
```
```

Practical 14 - Support Vector Machines - Bank Marketing preprocessing dataset

14 June 2022

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [2]:
```
bank=pd.read_csv('D:\\ML\\bank.csv',sep=';')
```

In [3]:
```
bank
```

Out[3]:

	age	job	marital	education	default	balance	housing	loan	contact	day	m
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	
...	...	...	...	...	...	...	...	...	...	...	
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	
4518	57	technician	married	secondary	no	295	no	no	cellular	19	
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	

4521 rows × 17 columns

In [4]:
```
bank.shape
```

Out[4]:
```
(4521, 17)
```

In [5]:
```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        4521 non-null   int64 
 1   job        4521 non-null   object
 2   marital    4521 non-null   object
 3   education  4521 non-null   object
 4   default    4521 non-null   object
 5   balance    4521 non-null   int64 
 6   housing    4521 non-null   object
 7   loan       4521 non-null   object
 8   contact    4521 non-null   object
 9   day        4521 non-null   int64 
 10  month      4521 non-null   object
 11  duration   4521 non-null   int64 
 12  campaign   4521 non-null   int64 
 13  pdays      4521 non-null   int64 
 14  previous   4521 non-null   int64 
 15  poutcome   4521 non-null   object
 16  y          4521 non-null   object
dtypes: int64(7), object(10)
memory usage: 600.6+ KB
```

In [6]:
```
bank.head()
```

Out[6]:

	age	job	marital	education	default	balance	housing	loan	contact	day	mont
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oc
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	ma
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	ap
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	ju
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	ma

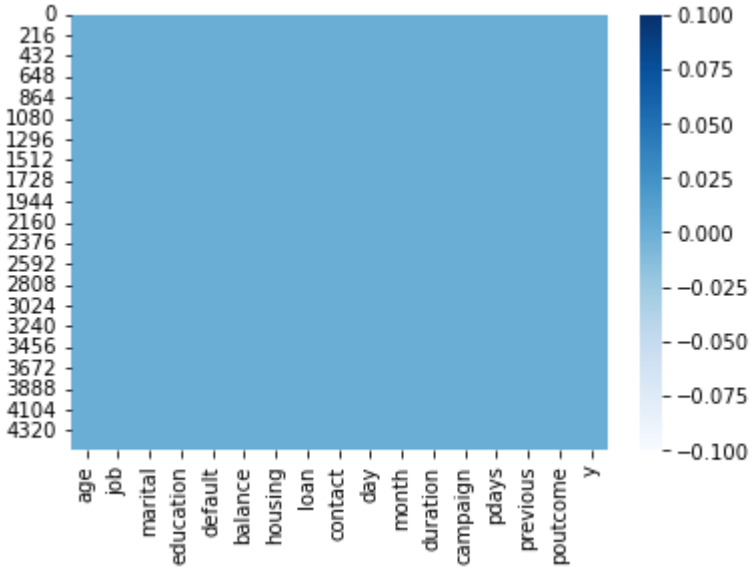
In [7]:
```
bank.describe()
```

Out[7]:

	age	balance	day	duration	campaign	pdays	previo
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.0000
mean	41.170095	1422.657819	15.915284	263.961292	2.793630	39.766645	0.5425
std	10.576211	3009.638142	8.247667	259.856633	3.109807	100.121124	1.6935
min	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.0000
25%	33.000000	69.000000	9.000000	104.000000	1.000000	-1.000000	0.0000
50%	39.000000	444.000000	16.000000	185.000000	2.000000	-1.000000	0.0000
75%	49.000000	1480.000000	21.000000	329.000000	3.000000	-1.000000	0.0000
max	87.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.0000

In [8]:
```
sns.heatmap(bank.isnull(),cmap='Blues')
```

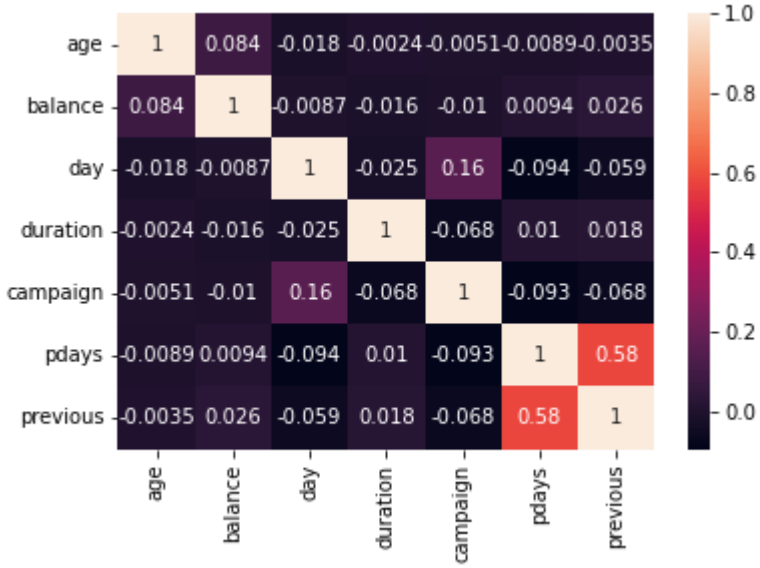
Out[8]:
```
<AxesSubplot:>
```



In [9]:
```
#to find corelattion
cor=bank.corr()
```

In [10]:
```
sns.heatmap(cor,annot=True)
```

Out[10]:
```
<AxesSubplot:>
```



pdays and previous are highly corelated with value 0.58

day and campaign are also corelated with 0.16

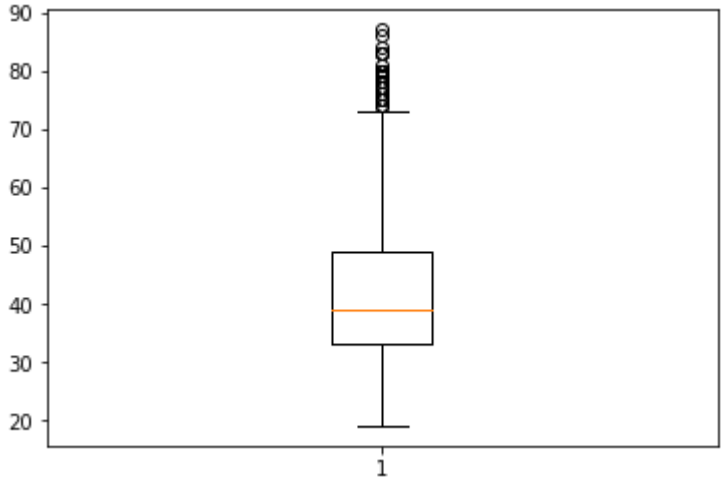
Numeric : age,balance,day,duration,campaign,pdays

categorical:

Oridinal:poutcome,default,housing,loan (using map)

nominal : job,marital,education ,contact,month (get dummies)

In [11]:
```
plt.boxplot(bank['age']);
```

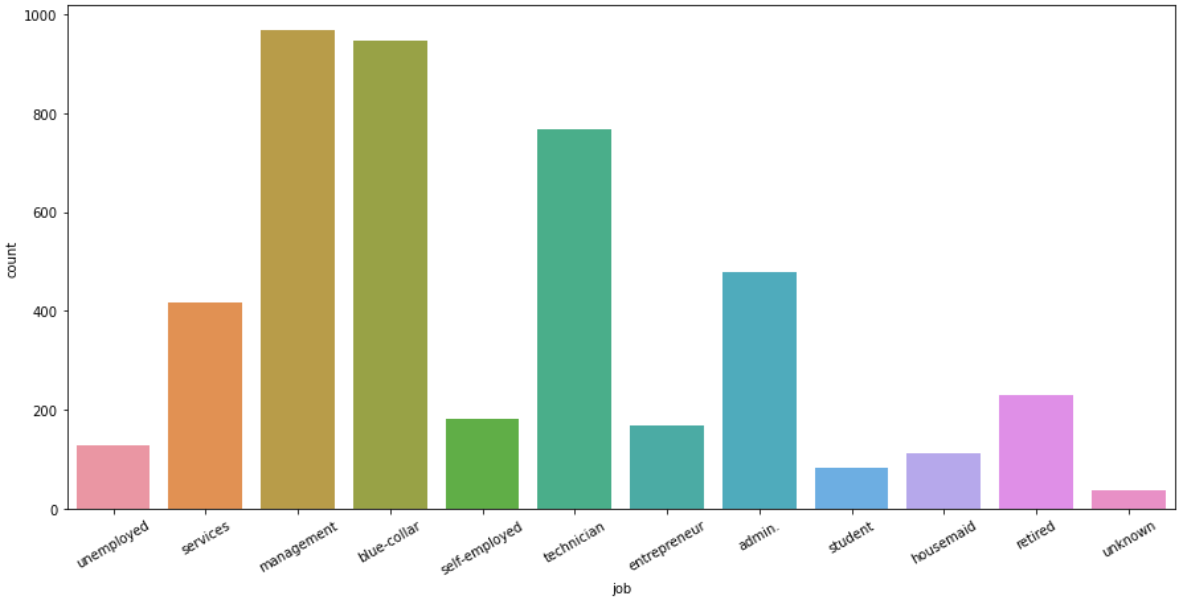


min age is 20 and max age is 70-90

In [12]:
```
bank['job'].unique()
```

Out[12]:
```
array(['unemployed', 'services', 'management', 'blue-collar',
       'self-employed', 'technician', 'entrepreneur', 'admin.', 'student',
       'housemaid', 'retired', 'unknown'], dtype=object)
```

In [13]:
```
plt.figure(figsize=(15,7))
ax=sns.countplot(x='job',data=bank)
ax.set_xticklabels(ax.get_xticklabels(),rotation=30);
```



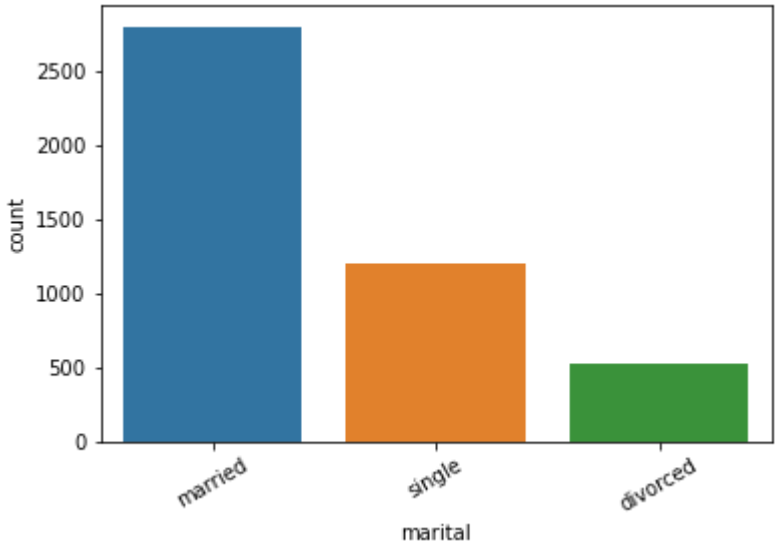
most of the customers are from managment,blue collar and technician

less from hosemain, students ,unknown and unemployed

In [14]:
```
bank['marital'].unique()
```

Out[14]:
```
array(['married', 'single', 'divorced'], dtype=object)
```

In [15]:
```
ax=sns.countplot(x='marital',data=bank)
ax.set_xticklabels(ax.get_xticklabels(),rotation  =  30);
```

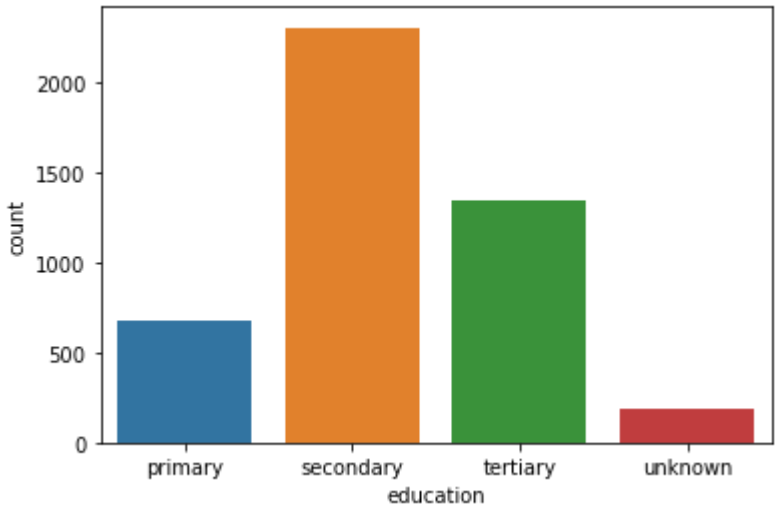


most are married

In [16]:
```
bank['education'].unique()
```

Out[16]:
```
array(['primary', 'secondary', 'tertiary', 'unknown'], dtype=object)
```

In [17]:
```
ax=sns.countplot(x='education',data=bank)
```

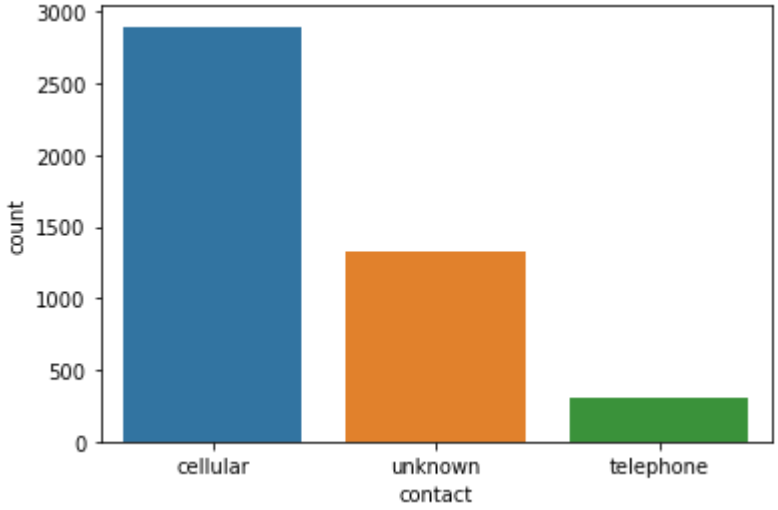


most of them are educated secondary and teriary

In [18]:
```
bank['contact'].unique()
```

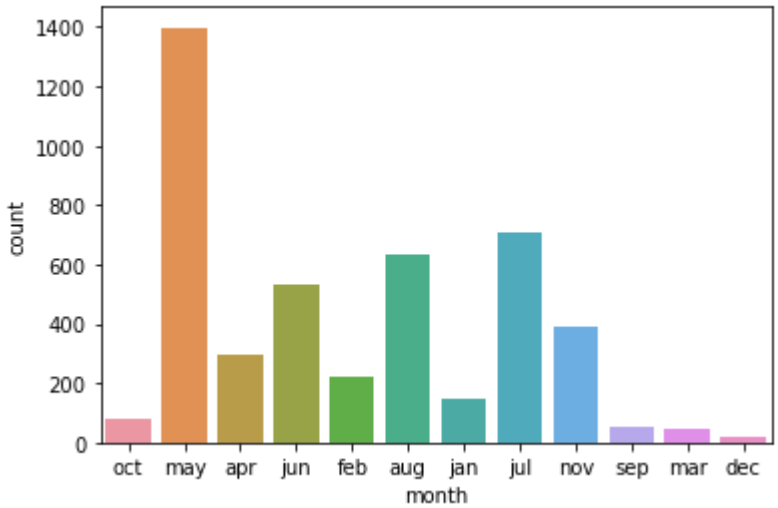
Out[18]:
```
array(['cellular', 'unknown', 'telephone'], dtype=object)
```

In [19]:
```
ax=sns.countplot(x='contact',data=bank)
```



most contact are made on cellular network and less on telephone

In [20]:
```
ax=sns.countplot(x='month',data=bank)
```

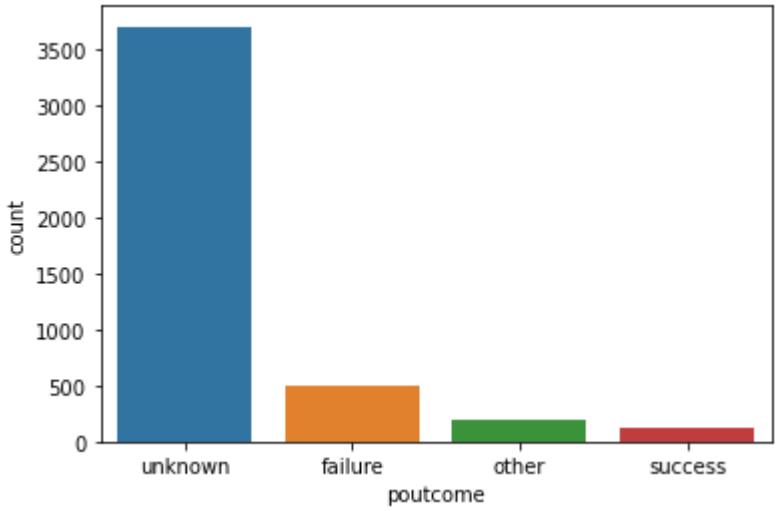


may month has most count and december has less count

In [21]:
```
bank['poutcome'].unique()
```

Out[21]:
```
array(['unknown', 'failure', 'other', 'success'], dtype=object)
```

In [22]:
```
ax=sns.countplot(x='poutcome',data=bank)
```



most of they are unknown

In [23]:
```
bank['poutcome']=bank['poutcome'].map({'failure':-1,'unknown':0,'success':1,'other
```

In [24]:
```
bank['default'].unique()
```

Out[24]:
```
array(['no', 'yes'], dtype=object)
```

In [25]:
```
bank['housing'].unique()
```

Out[25]:
```
array(['no', 'yes'], dtype=object)
```

In [26]:
```
bank['loan'].unique()
```

Out[26]:
```
array(['no', 'yes'], dtype=object)
```

In [27]:
```
bank['default']=bank['default'].map({'yes':0,'no':1})
bank['housing']=bank['housing'].map({'yes':0,'no':1})
bank['loan']=bank['loan'].map({'yes':0,'no':1})
```

In [28]:
```
bank
```

Out[28]:

	age	job	marital	education	default	balance	housing	loan	contact	day	m
0	30	unemployed	married	primary	1	1787	1	1	cellular	19	
1	33	services	married	secondary	1	4789	0	0	cellular	11	
2	35	management	single	tertiary	1	1350	0	1	cellular	16	
3	30	management	married	tertiary	1	1476	0	0	unknown	3	
4	59	blue-collar	married	secondary	1	0	0	1	unknown	5	
...	...	...	...	...	...	...	...	...	...	...	
4516	33	services	married	secondary	1	-333	0	1	cellular	30	
4517	57	self-employed	married	tertiary	0	-3313	0	0	unknown	9	
4518	57	technician	married	secondary	1	295	1	1	cellular	19	
4519	28	blue-collar	married	secondary	1	1137	1	1	cellular	6	
4520	44	entrepreneur	single	tertiary	1	1136	0	0	cellular	3	

4521 rows × 17 columns

In [29]:
```
nominal=['job','marital','education','contact','month']
data=pd.get_dummies(bank,columns=nominal)
```

In [30]:
```
data.shape
```

Out[30]:
```
(4521, 46)
```

In [31]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 46 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  4521 non-null   int64
 1   default              4521 non-null   int64
 2   balance              4521 non-null   int64
 3   housing              4521 non-null   int64
 4   loan                 4521 non-null   int64
 5   day                  4521 non-null   int64
 6   duration             4521 non-null   int64
 7   campaign             4521 non-null   int64
 8   pdays                4521 non-null   int64
 9   previous             4521 non-null   int64
 10  poutcome             4521 non-null   int64
 11  y                    4521 non-null   object
 12  job_admin.           4521 non-null   uint8
 13  job_blue-collar      4521 non-null   uint8
 14  job_entrepreneur     4521 non-null   uint8
 15  job_housemaid        4521 non-null   uint8
 16  job_management       4521 non-null   uint8
 17  job_retired          4521 non-null   uint8
 18  job_self-employed    4521 non-null   uint8
 19  job_services         4521 non-null   uint8
 20  job_student          4521 non-null   uint8
 21  job_technician       4521 non-null   uint8
 22  job_unemployed       4521 non-null   uint8
 23  job_unknown          4521 non-null   uint8
 24  marital_divorced     4521 non-null   uint8
 25  marital_married      4521 non-null   uint8
 26  marital_single       4521 non-null   uint8
 27  education_primary    4521 non-null   uint8
 28  education_secondary  4521 non-null   uint8
 29  education_tertiary   4521 non-null   uint8
 30  education_unknown    4521 non-null   uint8
 31  contact_cellular     4521 non-null   uint8
 32  contact_telephone    4521 non-null   uint8
 33  contact_unknown      4521 non-null   uint8
 34  month_apr            4521 non-null   uint8
 35  month_aug            4521 non-null   uint8
 36  month_dec            4521 non-null   uint8
 37  month_feb            4521 non-null   uint8
 38  month_jan            4521 non-null   uint8
 39  month_jul            4521 non-null   uint8
 40  month_jun            4521 non-null   uint8
 41  month_mar            4521 non-null   uint8
 42  month_may            4521 non-null   uint8
 43  month_nov            4521 non-null   uint8
 44  month_oct            4521 non-null   uint8
 45  month_sep            4521 non-null   uint8
dtypes: int64(11), object(1), uint8(34)
memory usage: 574.1+ KB
```

In [32]:
```
data['y']=data['y'].map({'yes':1,'no':0})
```

In [33]:
```
data.head()
```

Out[33]:

	age	default	balance	housing	loan	day	duration	campaign	pdays	previous	...	month_
0	30	1	1787	1	1	19	79	1	-1	0	...	
1	33	1	4789	0	0	11	220	1	339	4	...	
2	35	1	1350	0	1	16	185	1	330	1	...	
3	30	1	1476	0	0	3	199	4	-1	0	...	
4	59	1	0	0	1	5	226	1	-1	0	...	

5 rows × 46 columns

In [34]:
```
data[['age','balance','day','campaign','pdays']]=StandardScaler().fit_transform(da
```

In [35]:
```
data.shape
```

Out[35]:
```
(4521, 46)
```

In [36]:
```
data.drop('duration',axis=1,inplace=True)
```

In [37]:
```
data.shape
```

Out[37]:
```
(4521, 45)
```

In [38]:
```
op=data.pop('y')
```

In [39]:
```
op.shape
```

Out[39]:
```
(4521,)
```

In [40]:
```
xtr,xts,ytr,yts=train_test_split(data,op,train_size=0.8,random_state=100)
```

In [41]:
```
xtr.shape
```

Out[41]:
```
(3616, 44)
```

In [42]:
```
ytr.shape
```

Out[42]:
```
(3616,)
```

In [43]:
```
yts.shape
```

Out[43]:
```
(905,)
```

In [44]:
```
xts.shape
```

Out[44]:
```
(905, 44)
```

In [45]:
```
#svclassifier=SVC(kernel='linear')
 #svclassifier=SVC(kernel='rbf') #radial basis function
svclassifier=SVC(kernel='poly',degree=4)
```

In [46]:
```
svclassifier.fit(xtr,ytr)
```

Out[46]:

```
SVC(degree=4,  kernel='poly')
```

In [47]:

```python
pred=svclassifier.predict(xts)
```

In [48]:

```python
pred
```

Out[48]:

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

In [49]:

```python
print(classification_report(yts,pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       810
           1       0.60      0.13      0.21        95

    accuracy                           0.90       905
   macro avg       0.75      0.56      0.58       905
weighted avg       0.87      0.90      0.87       905
```

In [50]:

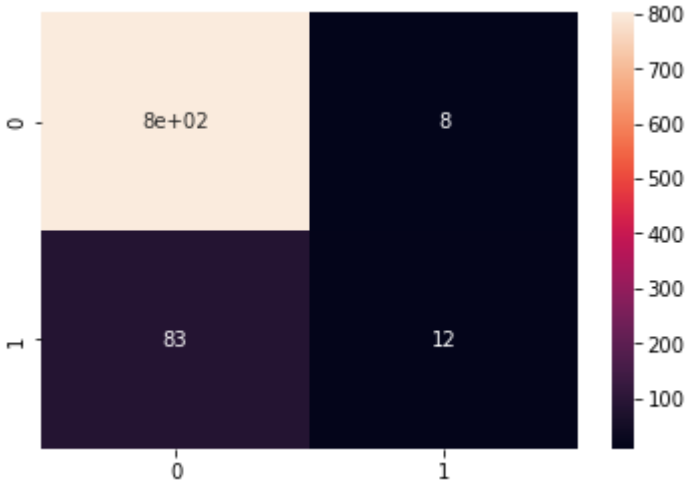
```python
cv=confusion_matrix(yts,pred)
```

In [51]:
```
cv
```

Out[51]:
```
array([[802,   8],
       [ 83,  12]], dtype=int64)
```

In [52]:
```
sns.heatmap(cv,annot=True)
```

Out[52]:
```
<AxesSubplot:>
```



In [53]:
```
kernels=['linear','rbf','poly']
for kernel in kernels:
    sv=SVC(kernel=kernel).fit(xtr,ytr)
    pred=sv.predict(xts)
    print("accuracy  :("+kernel+")",accuracy_score(yts,pred))
```

```
accuracy  :(linear) 0.8950276243093923
accuracy  :(rbf) 0.8994475138121547
accuracy  :(poly) 0.8983425414364641
```

In [54]:
```
# for non linear data
gammas=[0.1,1,10,100]
for gamma in gammas:
    sv=SVC(kernel='rbf',gamma=gamma).fit(xtr,ytr)
    pred=sv.predict(xts)
    print("Accuracy :(", gamma ,")",accuracy_score(yts,pred))
```

```
Accuracy :( 0.1 ) 0.9005524861878453
Accuracy :( 1 ) 0.8928176795580111
Accuracy :( 10 ) 0.8939226519337017
Accuracy :( 100 ) 0.8939226519337017
```

In [55]:
```
degrees=[0,1,2,3,4,5,20]
for degree in degrees:
    sv=SVC(kernel='poly',degree=degree).fit(xtr,ytr)
    pred=sv.predict(xts)
    print("Accuracy :(", degree ,")",accuracy_score(yts,pred))
```

```
Accuracy :( 0 ) 0.8950276243093923
Accuracy :( 1 ) 0.8950276243093923
Accuracy :( 2 ) 0.8972375690607735
Accuracy :( 3 ) 0.8983425414364641
Accuracy :( 4 ) 0.8994475138121547
Accuracy :( 5 ) 0.8961325966850828
Accuracy :( 20 ) 0.8718232044198895
```

Practical 15 - Support Vector Machines - bill authentication dataset

14 June 2022

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [3]:
```
df=pd.read_csv('D:\\ML\\bill_authentication.csv')
```

In [3]:
```
df
```

Out[3]:

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.66610	-2.8073	-0.44699	0
1	4.54590	8.16740	-2.4586	-1.46210	0
2	3.86600	-2.63830	1.9242	0.10645	0
3	3.45660	9.52280	-4.0112	-3.59440	0
4	0.32924	-4.45520	4.5718	-0.98880	0
...	...	...	...	...	...
1367	0.40614	1.34920	-1.4501	-0.55949	1
1368	-1.38870	-4.87730	6.4774	0.34179	1
1369	-3.75030	-13.45860	17.5932	-2.77710	1
1370	-3.56370	-8.38270	12.3930	-1.28230	1
1371	-2.54190	-0.65804	2.6842	1.19520	1

1372 rows × 5 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1372 entries, 0 to 1371
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Variance  1372 non-null   float64
 1   Skewness  1372 non-null   float64
 2   Curtosis  1372 non-null   float64
 3   Entropy   1372 non-null   float64
 4   Class     1372 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 53.7 KB
```

In [5]:
```
df.head()
```

Out[5]:

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

In [6]:
```
df.describe()
```

Out[6]:

	Variance	Skewness	Curtosis	Entropy	Class
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

In [7]:
```
df.tail()
```

Out[7]:

	Variance	Skewness	Curtosis	Entropy	Class
1367	0.40614	1.34920	-1.4501	-0.55949	1
1368	-1.38870	-4.87730	6.4774	0.34179	1
1369	-3.75030	-13.45860	17.5932	-2.77710	1
1370	-3.56370	-8.38270	12.3930	-1.28230	1
1371	-2.54190	-0.65804	2.6842	1.19520	1

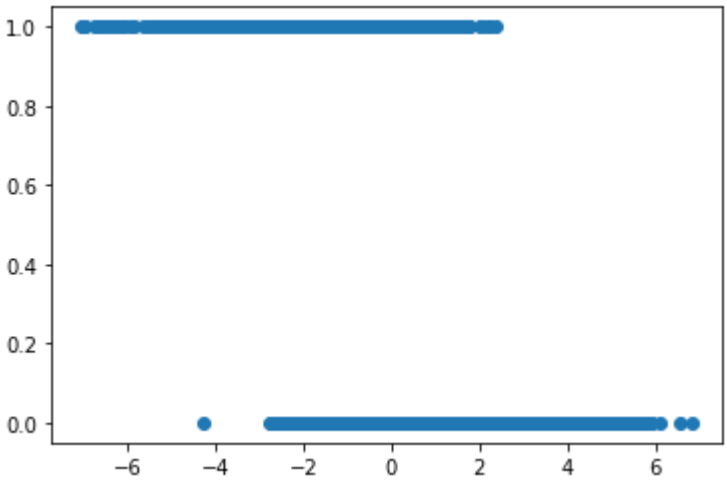
In [8]:
```
sns.heatmap(df.isnull())
```

Out[8]:
```
<AxesSubplot:>
```

NO NULL VALUES

In [9]:
```
plt.scatter(df.Variance,df.Class)
```

Out[9]:
```
<matplotlib.collections.PathCollection at 0x1fd9f345fd0>
```



In [10]:
```
X=df.drop('Class',axis=1)
```

In [11]:
```
Y=df['Class']
```

In [12]:
```
X
```

Out[12]:

	Variance	Skewness	Curtosis	Entropy
0	3.62160	8.66610	-2.8073	-0.44699
1	4.54590	8.16740	-2.4586	-1.46210
2	3.86600	-2.63830	1.9242	0.10645
3	3.45660	9.52280	-4.0112	-3.59440
4	0.32924	-4.45520	4.5718	-0.98880
...	...	...	...	...
1367	0.40614	1.34920	-1.4501	-0.55949
1368	-1.38870	-4.87730	6.4774	0.34179
1369	-3.75030	-13.45860	17.5932	-2.77710
1370	-3.56370	-8.38270	12.3930	-1.28230
1371	-2.54190	-0.65804	2.6842	1.19520

1372 rows × 4 columns

In [13]:
```
Y
```

Out[13]:
```
0       0
1       0
2       0
3       0
4       0
       ..
1367    1
1368    1
1369    1
1370    1
1371    1
Name: Class, Length: 1372, dtype: int64
```

In [14]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.20)
```

In [15]:
```
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train,y_train)
```

Out[15]:
```
SVC(kernel='linear')
```

In [16]:
```
y_pred=svclassifier.predict(X_test)
```

In [17]:
```
y_pred
```

Out[17]:
```
array([0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1], dtype=int64)
```

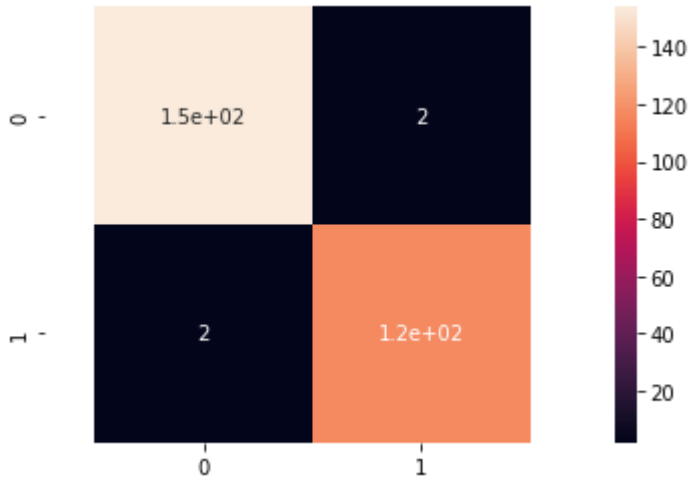
In [18]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       156
           1       0.98      0.98      0.98       119

    accuracy                           0.99       275
   macro avg       0.99      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275
```

In [21]:
```
cf=confusion_matrix(y_test,y_pred)
cf
```

Out[21]:
```
array([[154,   2],
       [  2, 117]], dtype=int64)
```

In [22]:
```
sns.heatmap(cf,annot=True)
plt.axis('equal')
plt.show()
```



In [ ]:

Practical 16 - Support Vector Machines - Wine dataset

15 June 2022

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [2]:
```
wine=pd.read_csv('D:\\ML\\winequality-red.csv',sep=';')
```

In [3]:
```
wine
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	

1599 rows × 12 columns

In [4]:
```
wine.describe()
```

Out[4]:

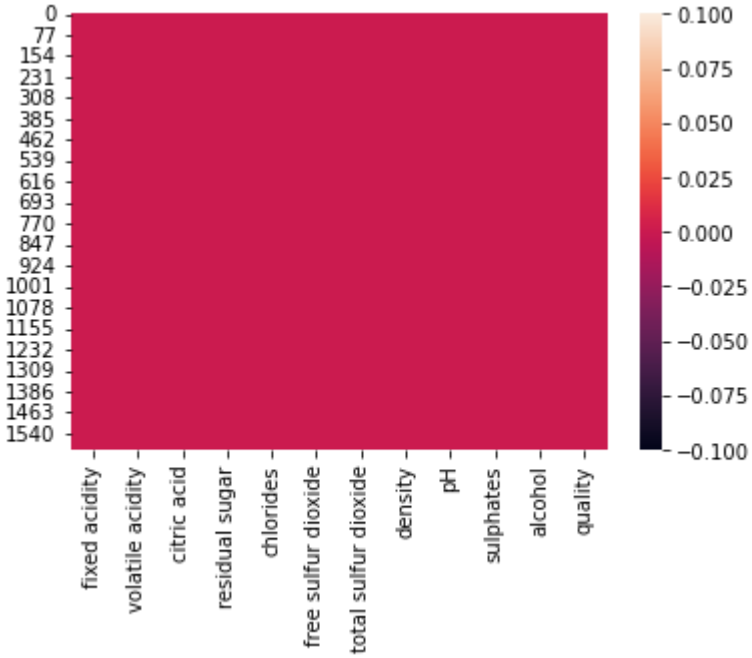
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfu dioxid
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.00000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.46779
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.89532
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.00000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.00000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.00000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.00000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.00000

In [5]:
```
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [6]:
```
sns.heatmap(wine.isnull())
```

Out[6]:
```
<AxesSubplot:>
```

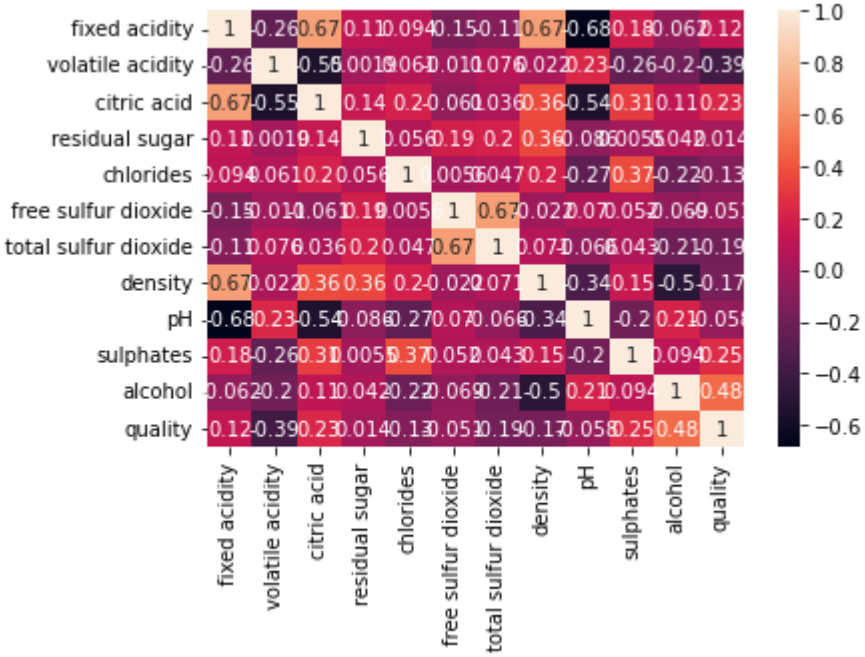


No null value

In [7]:
```
cor=wine.corr()
```

In [8]:
```
sns.heatmap(cor,annot=True)
```

Out[8]:
```
<AxesSubplot:>
```



citric acid and fixed acid are highly corelatted density and fixed acid are highly corelatted total sulfur dioxide and free sulfur dioxide are highly corelatted

In [9]:
```
wine['quality'].unique()
```

Out[9]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [10]:
```
X=wine.drop('quality',axis=1)
```

In [11]:
```
X
```

Out[11]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	

1599 rows × 11 columns

In [12]:
```
Y=wine.quality
```

In [13]:
```
Y
```

Out[13]:
```
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1599, dtype: int64
```

In [14]:
```
X.shape
```

Out[14]:
```
(1599, 11)
```

In [15]:
```
Y.shape
```

Out[15]:
```
(1599,)
```

In [16]:
```
xtr,xts,ytr,yts=train_test_split(X,Y,train_size=0.8,random_state=100)
```

In [17]:
```
xtr.shape
```

Out[17]:
```
(1279, 11)
```

In [18]:
```
ytr.shape
```

Out[18]:
```
(1279,)
```

In [19]:
```
xts.shape
```

Out[19]:
```
(320, 11)
```

In [20]:
```
yts.shape
```

Out[20]:
```
(320,)
```

In [42]:
```
#svclassifier=SVC(kernel='linear')  #0.61
#svclassifier=SVC(kernel='rbf')  #0.54
#svclassifier=SVC(kernel='poly',degree=4)  #0.52
#svclassifier=SVC(kernel='poly',degree=2)  #0.52
svclassifier=SVC(kernel='poly',degree=3)
```

In [43]:
```
svclassifier.fit(xtr,ytr)
```

Out[43]:
```
SVC(kernel='poly')
```

In [44]:
```
pred=svclassifier.predict(xts)
```

In [45]:
```
pred
```

Out[45]:
```
array([6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 5, 6, 6,
       6, 5, 6, 6, 6, 6, 6, 6, 5, 5, 5, 6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 6,
       5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 5, 6, 5, 6,
       6, 6, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 6, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 6,
       6, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 5, 6, 6, 6, 6, 5, 5, 6, 6,
       6, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 5, 6, 6, 6,
       5, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 5, 6, 6, 6, 5, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       5, 6, 5, 5, 6, 5, 6, 6, 6, 6, 5, 5, 6, 5, 5, 5, 6, 6, 6, 6, 5, 6,
       6, 6, 5, 6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 6, 6, 6, 6, 6, 5, 5, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 5, 6, 6, 5, 6, 6, 6, 6, 6, 6, 5,
       6, 6, 6, 6, 6, 5, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, 6,
       5, 6, 6, 6, 5, 6, 6, 6, 6, 6, 6, 5], dtype=int64)
```

In [46]:
```
print(classification_report(yts,pred))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         3
           4       0.00      0.00      0.00        12
           5       0.70      0.32      0.44       133
           6       0.47      0.89      0.62       137
           7       0.00      0.00      0.00        32
           8       0.00      0.00      0.00         3

    accuracy                           0.52       320
   macro avg       0.20      0.20      0.18       320
weighted avg       0.49      0.52      0.45       320
```

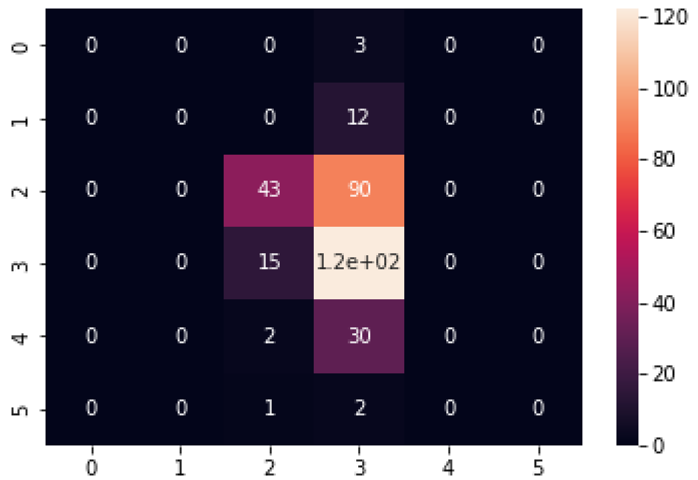
```
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [47]:
```
cv=confusion_matrix(yts,pred)
cv
```

Out[47]:
```
array([[  0,   0,   0,   3,   0,   0],
       [  0,   0,   0,  12,   0,   0],
       [  0,   0,  43,  90,   0,   0],
       [  0,   0,  15, 122,   0,   0],
       [  0,   0,   2,  30,   0,   0],
       [  0,   0,   1,   2,   0,   0]], dtype=int64)
```

In [48]:
```
sns.heatmap(cv,annot=True)
```

Out[48]: <AxesSubplot:>



In [ ]:

In [ ]:

Practical 17 - Support Vector Regressor

15 June 2022

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

In [2]:
```
data=pd.read_csv('D:\\ML\\machine.data',sep=',')
```

In [3]:
```
data
```

Out[3]:

	adviser	32/60	125	256	6000	256.1	16	128	198	199
0	amdahl	470v/7	29	8000	32000	32	8	32	269	253
1	amdahl	470v/7a	29	8000	32000	32	8	32	220	253
2	amdahl	470v/7b	29	8000	32000	32	8	32	172	253
3	amdahl	470v/7c	29	8000	16000	32	8	16	132	132
4	amdahl	470v/b	26	8000	32000	64	8	32	318	290
...	...	...	...	...	...	...	...	...	...	...
203	sperry	80/8	124	1000	8000	0	1	8	42	37
204	sperry	90/80-model-3	98	1000	8000	32	2	8	46	50
205	sratus	32	125	2000	8000	0	2	14	52	41
206	wang	vs-100	480	512	8000	32	0	0	67	47
207	wang	vs-90	480	1000	4000	0	0	0	45	25

208 rows × 10 columns

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 208 entries, 0 to 207
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   adviser  208 non-null    object
 1   32/60    208 non-null    object
 2   125      208 non-null    int64 
 3   256      208 non-null    int64 
 4   6000     208 non-null    int64 
 5   256.1    208 non-null    int64 
 6   16       208 non-null    int64 
 7   128      208 non-null    int64 
 8   198      208 non-null    int64 
 9   199      208 non-null    int64 
dtypes: int64(8), object(2)
memory usage: 16.4+ KB
```

In [5]:
```
data.describe()
```

Out[5]:

	125	256	6000	256.1	16	128	198
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000
mean	204.201923	2880.538462	11824.019231	24.096154	4.644231	17.740385	105.177885
std	260.833016	3883.839300	11747.916663	37.417999	6.787198	24.913375	161.090223
min	17.000000	64.000000	64.000000	0.000000	0.000000	0.000000	6.000000
25%	50.000000	768.000000	4000.000000	0.000000	1.000000	5.000000	27.000000
50%	110.000000	2000.000000	8000.000000	8.000000	2.000000	8.000000	49.500000
75%	225.000000	4000.000000	16000.000000	32.000000	6.000000	24.000000	111.500000
max	1500.000000	32000.000000	64000.000000	256.000000	52.000000	176.000000	1150.000000

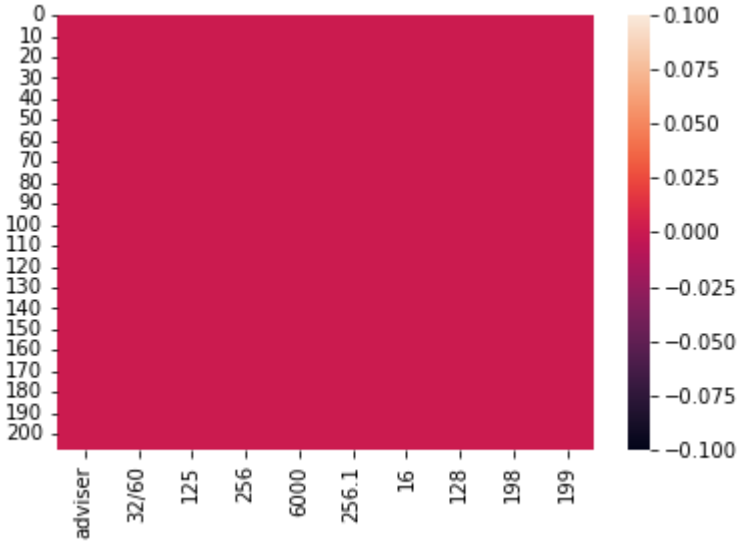
In [6]:
```
data.head()
```

Out[6]:

	adviser	32/60	125	256	6000	256.1	16	128	198	199
0	amdahl	470v/7	29	8000	32000	32	8	32	269	253
1	amdahl	470v/7a	29	8000	32000	32	8	32	220	253
2	amdahl	470v/7b	29	8000	32000	32	8	32	172	253
3	amdahl	470v/7c	29	8000	16000	32	8	16	132	132
4	amdahl	470v/b	26	8000	32000	64	8	32	318	290

In [7]:
```
sns.heatmap(data.isna())
```

Out[7]:
```
<AxesSubplot:>
```



no null values

In [8]:
```
data.shape
```

Out[8]:
```
(208, 10)
```

In [9]:
```
col_names=['vendorname','modelname','MYCT','MMIN','MMAX','CACH','CHMIN','CHMAX','P
```

In [10]:
```
data.columns=col_names
```

In [11]:
```
data
```

Out[11]:

	vendorname	modelname	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP	ERP
0	amdahl	470v/7	29	8000	32000	32	8	32	269	253
1	amdahl	470v/7a	29	8000	32000	32	8	32	220	253
2	amdahl	470v/7b	29	8000	32000	32	8	32	172	253
3	amdahl	470v/7c	29	8000	16000	32	8	16	132	132
4	amdahl	470v/b	26	8000	32000	64	8	32	318	290
...	...	...	...	...	...	...	...	...	...	...
203	sperry	80/8	124	1000	8000	0	1	8	42	37
204	sperry	90/80-model-3	98	1000	8000	32	2	8	46	50
205	sratus	32	125	2000	8000	0	2	14	52	41
206	wang	vs-100	480	512	8000	32	0	0	67	47
207	wang	vs-90	480	1000	4000	0	0	0	45	25

208 rows × 10 columns

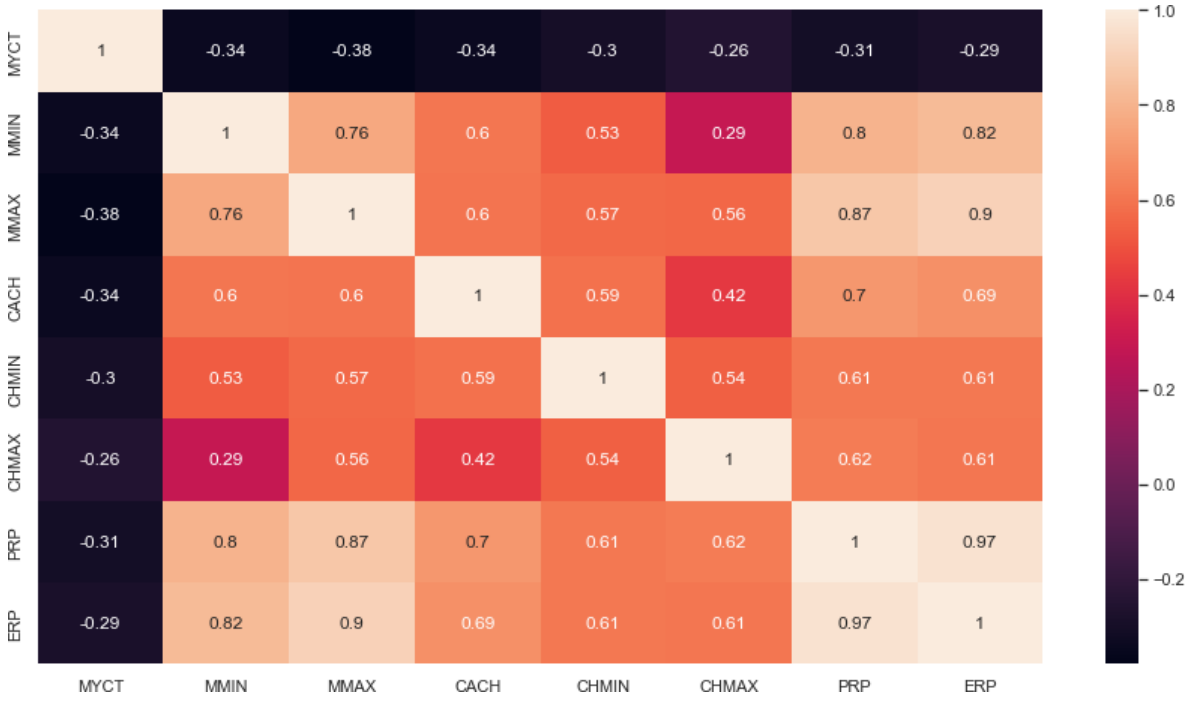
In [12]:
```
data['ERP'].unique()
```

Out[12]:
```
array([ 253,  132,  290,  381,  749, 1238,   23,   24,   70,  117,   15,
         64,   29,   22,  124,   35,   39,   40,   45,   28,   21,   27,
        102,   74,  138,  136,   44,   30,   41,   54,   18,   36,   38,
         34,   19,   72,   56,   42,   75,  113,  157,   20,   33,   47,
         25,   52,   50,   53,   73,   32,  175,   57,  181,   82,  171,
        361,  350,  220,   17,   26,   31,   76,   59,   65,  101,  116,
        128,   37,   46,   80,   88,   86,   95,  107,  119,  120,   48,
        126,  266,  270,  426,  151,  267,  603,   62,   78,  142,  281,
        190,   67,   43,   99,   81,  149,  183,  275,  382,  182,  227,
        341,  360,  919,  978], dtype=int64)
```

In [13]:
```
cor=data.corr()
```

In [37]:
```
sns.set(rc  = {'figure.figsize':(15,8)})

sns.heatmap(cor,annot=True)
```

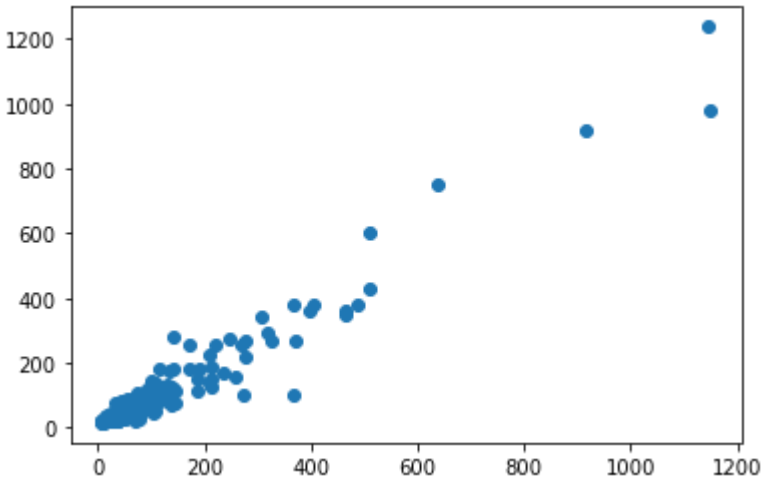
Out[37]:
```
<AxesSubplot:>
```



all features are highly corelated with ERP,whereas MYCT machine cycle time in nanoseconds is negativly corelated

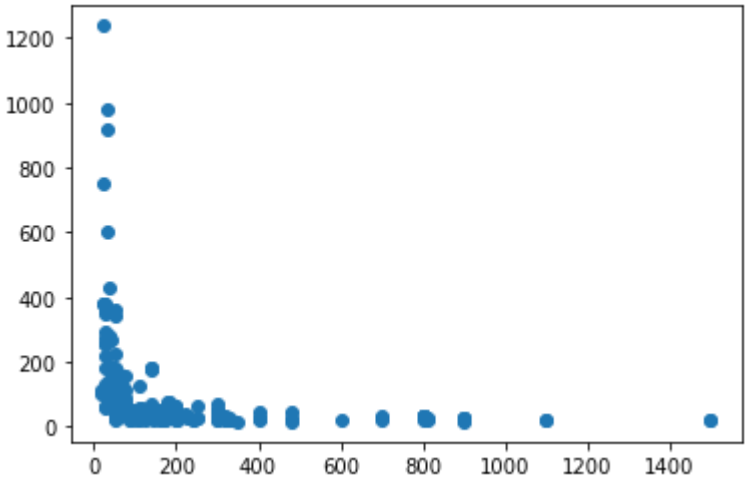
In [15]:
```
plt.scatter(data.PRP,data.ERP)
```

Out[15]:
```
<matplotlib.collections.PathCollection  at  0x1e6b55eab50>
```



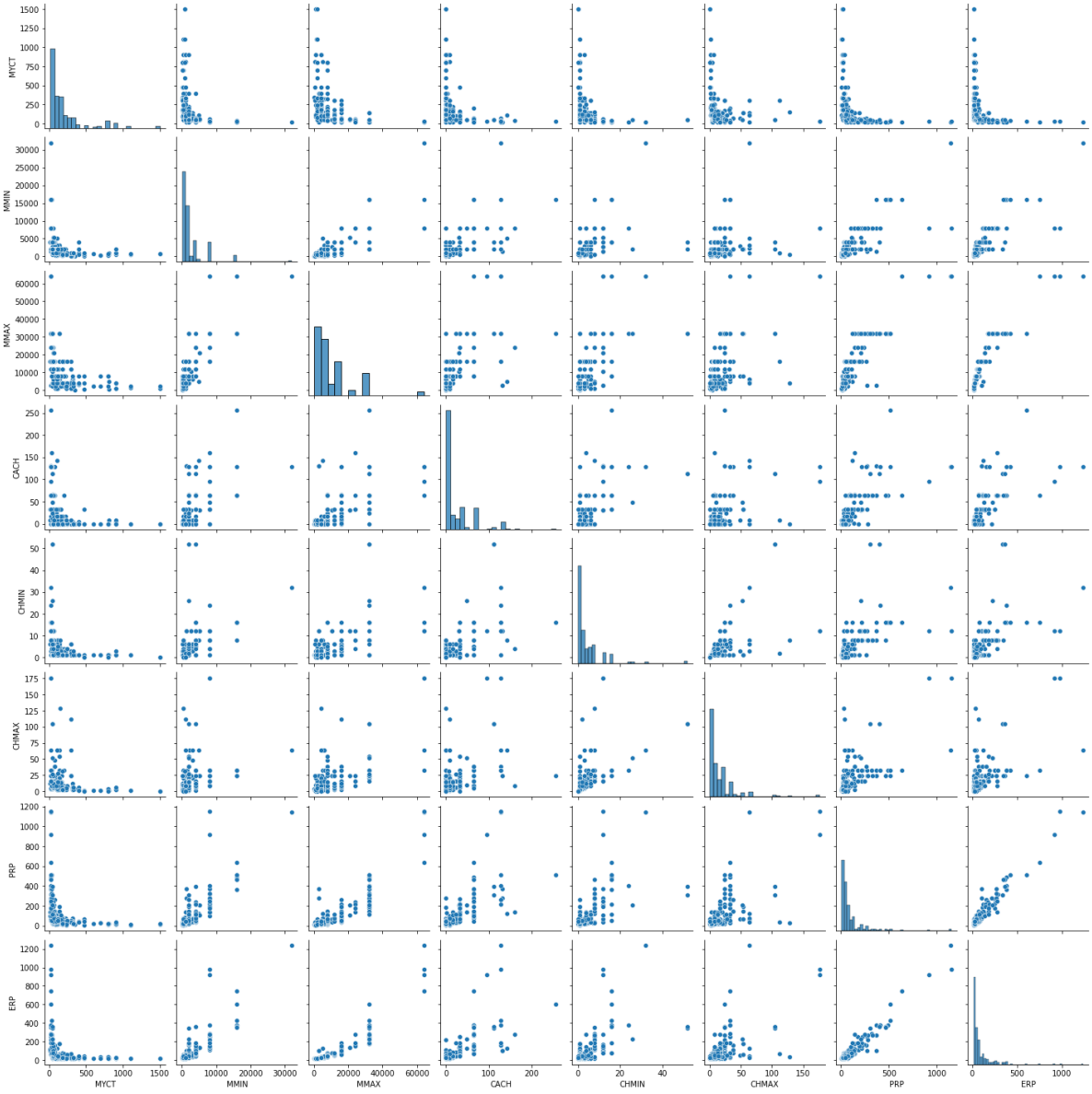
In [16]:
```
plt.scatter(data.MYCT,data.ERP)
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x1e6b566cc70>
```



In [17]:
```
sns.pairplot(data)
```

Out[17]:
```
<seaborn.axisgrid.PairGrid at 0x1e6b56a0520>
```



In [18]:
```
data.vendorname.unique()
```

Out[18]:
```
array(['amdahl', 'apollo', 'basf', 'bti', 'burroughs', 'c.r.d', 'cdc',
       'cambex', 'dec', 'dg', 'formation', 'four-phase', 'gould', 'hp',
       'harris', 'honeywell', 'ibm', 'ipl', 'magnuson', 'microdata',
       'nas', 'ncr', 'nixdorf', 'perkin-elmer', 'prime', 'siemens',
       'sperry', 'sratus', 'wang'], dtype=object)
```

In [19]:
```
data.modelname.unique()
```

Out[19]:
```
array(['470v/7', '470v/7a', '470v/7b', '470v/7c', '470v/b', '580-5840',
       '580-5850', '580-5860', '580-5880', 'dn320', 'dn420', '7/65',
       '7/68', '5000', '8000', 'b1955', 'b2900', 'b2925', 'b4955',
       'b5900', 'b5920', 'b6900', 'b6925', '68/10-80', 'universe:2203t',
       'universe:68', 'universe:68/05', 'universe:68/137',
       'universe:68/37', 'cyber:170/750', 'cyber:170/760',
       'cyber:170/815', 'cyber:170/825', 'cyber:170/835', 'cyber:170/845',
       'omega:480-i', 'omega:480-ii', 'omega:480-iii', '1636-1',
       '1636-10', '1641-1', '1641-11', '1651-1', 'decsys:10:1091',
       'decsys:20:2060', 'microvax-1', 'vax:11/730', 'vax:11/750',
       'vax:11/780', 'eclipse:c/350', 'eclipse:m/600', 'eclipse:mv/10000',
       'eclipse:mv/4000', 'eclipse:mv/6000', 'eclipse:mv/8000',
       'eclipse:mv/8000-ii', 'f4000/100', 'f4000/200', 'f4000/200ap',
       'f4000/300', 'f4000/300ap', '2000/260', 'concept:32/8705',
       'concept:32/8750', 'concept:32/8780', '3000/30', '3000/40',
       '3000/44', '3000/48', '3000/64', '3000/88', '3000/iii', '100',
       '300', '500', '600', '700', '80', '800', 'dps:6/35', 'dps:6/92',
       'dps:6/96', 'dps:7/35', 'dps:7/45', 'dps:7/55', 'dps:7/65',
       'dps:8/44', 'dps:8/49', 'dps:8/50', 'dps:8/52', 'dps:8/62',
       'dps:8/20', '3033:s', '3033:u', '3081', '3081:d', '3083:b',
       '3083:e', '370/125-2', '370/148', '370/158-3', '38/3', '38/4',
       '38/5', '38/7', '38/8', '4321', '4331-1', '4331-11', '4331-2',
       '4341', '4341-1', '4341-10', '4341-11', '4341-12', '4341-2',
       '4341-9', '4361-4', '4361-5', '4381-1', '4381-2', '8130-a',
       '8130-b', '8140', '4436', '4443', '4445', '4446', '4460', '4480',
       'm80/30', 'm80/31', 'm80/32', 'm80/42', 'm80/43', 'm80/44',
       'seq.ms/3200', 'as/3000', 'as/3000-n', 'as/5000', 'as/5000-e',
       'as/5000-n', 'as/6130', 'as/6150', 'as/6620', 'as/6630', 'as/6650',
       'as/7000', 'as/7000-n', 'as/8040', 'as/8050', 'as/8060',
       'as/9000-dpc', 'as/9000-n', 'as/9040', 'as/9060', 'v8535:ii',
       'v8545:ii', 'v8555:ii', 'v8565:ii', 'v8565:ii-e', 'v8575:ii',
       'v8585:ii', 'v8595:ii', 'v8635', 'v8650', 'v8655', 'v8665',
       'v8670', '8890/30', '8890/50', '8890/70', '3205', '3210', '3230',
       '50-2250', '50-250-ii', '50-550-ii', '50-750-ii', '50-850-ii',
       '7.521', '7.531', '7.536', '7.541', '7.551', '7.561', '7.865-2',
       '7.870-2', '7.872-2', '7.875-2', '7.880-2', '7.881-2',
       '1100/61-h1', '1100/81', '1100/82', '1100/83', '1100/84',
       '1100/93', '1100/94', '80/3', '80/4', '80/5', '80/6', '80/8',
       '90/80-model-3', '32', 'vs-100', 'vs-90'], dtype=object)
```

In [20]:
```
y=data.pop('ERP')
```

In [21]:
```
x=data.drop(['vendorname','modelname'],axis=1)
```

In [22]:
```
x.shape
```

Out[22]:
```
(208, 7)
```

In [23]:
```
y.shape
```

Out[23]:
```
(208,)
```

In [24]:
```
x
```

Out[24]:

	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP
0	29	8000	32000	32	8	32	269
1	29	8000	32000	32	8	32	220
2	29	8000	32000	32	8	32	172
3	29	8000	16000	32	8	16	132
4	26	8000	32000	64	8	32	318
...	...	...	...	...	...	...	...
203	124	1000	8000	0	1	8	42
204	98	1000	8000	32	2	8	46
205	125	2000	8000	0	2	14	52
206	480	512	8000	32	0	0	67
207	480	1000	4000	0	0	0	45

208 rows × 7 columns

In [25]:
```
y
```

Out[25]:
```
0      253
1      253
2      253
3      132
4      290
      ...
203     37
204     50
205     41
206     47
207     25
Name: ERP, Length: 208, dtype: int64
```

perform scalling on x

In [26]:
```
data[['MYCT','MMIN','MMAX','CACH','CHMIN','CHMAX','PRP']]=StandardScaler().fit_tra
```

In [27]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2)
```

In [28]:
```
reg=SVR(kernel='rbf')
reg.fit(xtrain,ytrain)
```

Out[28]:
```
SVR()
```

In [29]:
```
pred=reg.predict(xtest)
mean_squared_error(pred,ytest)
```

Out[29]:
```
52581.63770438957
```

In [30]:
```
pred
```

Out[30]:
```
array([42.46287195, 76.34858199, 31.44508081, 32.33623693, 45.1471775 ,
       27.66039618, 42.45517321, 56.73797856, 96.01668616, 31.4316104 ,
       27.50948139, 30.54559369, 41.42460478, 30.2408295 , 76.35024458,
       31.40444177, 60.51628985, 73.90030619, 27.29879927, 42.46952107,
       42.45665773, 34.85784104, 42.46726195, 32.31687593, 30.57047553,
       57.91590675, 42.44821687, 28.62863608, 26.48284039, 28.71472567,
       70.43650323, 68.8001756 , 30.53492263, 73.899844  , 31.43665302,
       40.75004427, 60.51950061, 73.8876442 , 26.67242339, 42.46839197,
       41.41610857, 52.52797342])
```

In [31]:
```
xtrain.shape
```

Out[31]:
```
(166, 7)
```

In [32]:
```
ytrain.shape
```

Out[32]:
```
(166,)
```

In [34]:
```
svclassifier=SVR(kernel='linear')
svclassifier.fit(xtrain,ytrain)
```

Out[34]:
```
SVR(kernel='linear')
```

In [35]:
```
pred=svclassifier.predict(xtest)
```

In [36]:
```
pred
```

Out[36]:
```
array([ 56.58539712,  79.12163769,  31.07734524,  29.25343394,
       103.10757314,  16.36098724,  70.61143624,  46.07327592,
       107.88796508,  35.22208642,  15.39071726,  20.16488611,
        42.19337355,  86.98142814,  98.00846087,  32.7938671 ,
        80.48070895,  57.443035  ,   8.76931786,  76.51198851,
        87.66016232,  25.86217987,  58.03392731,  26.90167355,
        21.89404344,  48.89939844,  66.08888518, 143.3386653 ,
        13.18263627,  28.00379931, 553.51555335, 783.03171431,
        19.96671235,  62.39014994,  43.66914713,  18.79148471,
        99.38499009,  63.56395137,  10.85526346,  43.27447545,
        43.04953515,  69.14209544])
```

In [ ]:
```

```

Practical 18 - Decision Tree Regressor – Air Quality Dataset

17 June 2022

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
import pandas as pd
from sklearn.metrics import mean_squared_error
import seaborn as sns
from sklearn.model_selection import train_test_split
```

In [2]:
```
df=pd.read_csv("D:\\ML\\AirQualityUCI.csv")
```

In [3]:
```
df
```

Out[3]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	P
0	10-03-2004	18:00:00	2.6	1360.0	150.0	11.9	1046.0	166.0	
1	10-03-2004	19:00:00	2.0	1292.0	112.0	9.4	955.0	103.0	
2	10-03-2004	20:00:00	2.2	1402.0	88.0	9.0	939.0	131.0	
3	10-03-2004	21:00:00	2.2	1376.0	80.0	9.2	948.0	172.0	
4	10-03-2004	22:00:00	1.6	1272.0	51.0	6.5	836.0	131.0	
...	...	...	...	...	...	...	...	...	...
9466	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9467	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9468	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9469	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
9470	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

9471 rows × 17 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9471 entries, 0 to 9470
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Date           9357 non-null   object
 1   Time           9357 non-null   object
 2   CO(GT)         9357 non-null   float64
 3   PT08.S1(CO)    9357 non-null   float64
 4   NMHC(GT)       9357 non-null   float64
 5   C6H6(GT)       9357 non-null   float64
 6   PT08.S2(NMHC)  9357 non-null   float64
 7   NOx(GT)        9357 non-null   float64
 8   PT08.S3(NOx)   9357 non-null   float64
 9   NO2(GT)        9357 non-null   float64
 10  PT08.S4(NO2)   9357 non-null   float64
 11  PT08.S5(O3)    9357 non-null   float64
 12  T              9357 non-null   float64
 13  RH             9357 non-null   float64
 14  AH             9357 non-null   float64
 15  Unnamed: 15    0 non-null      float64
 16  Unnamed: 16    0 non-null      float64
dtypes: float64(15), object(2)
memory usage: 1.2+ MB
```

In [5]:
```
df.describe()
```

Out[5]:

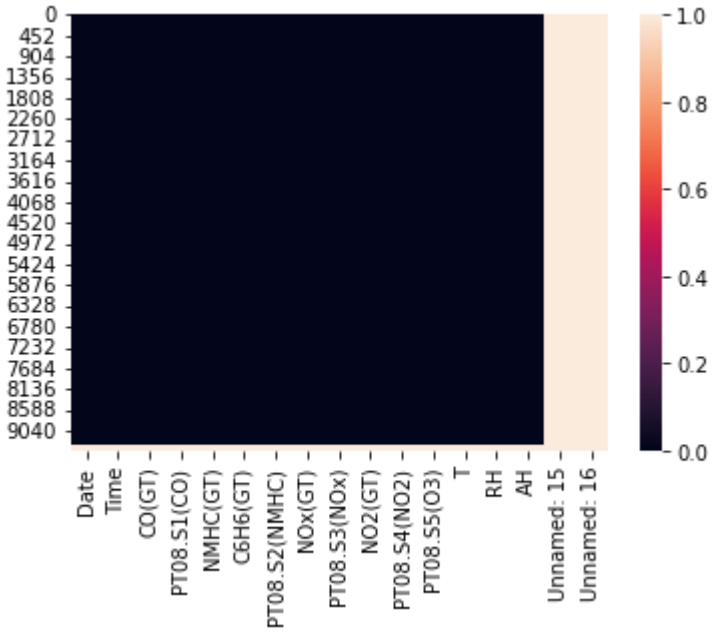
	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S
count	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357.000000	9357
mean	-34.207524	1048.990061	-159.090093	1.865683	894.595276	168.616971	794
std	77.657170	329.832710	139.789093	41.380206	342.333252	257.433866	321
min	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200.000000	-200
25%	0.600000	921.000000	-200.000000	4.000000	711.000000	50.000000	637
50%	1.500000	1053.000000	-200.000000	7.900000	895.000000	141.000000	794
75%	2.600000	1221.000000	-200.000000	13.600000	1105.000000	284.000000	960
max	11.900000	2040.000000	1189.000000	63.700000	2214.000000	1479.000000	2683

In [6]:
```
df.dtypes
```

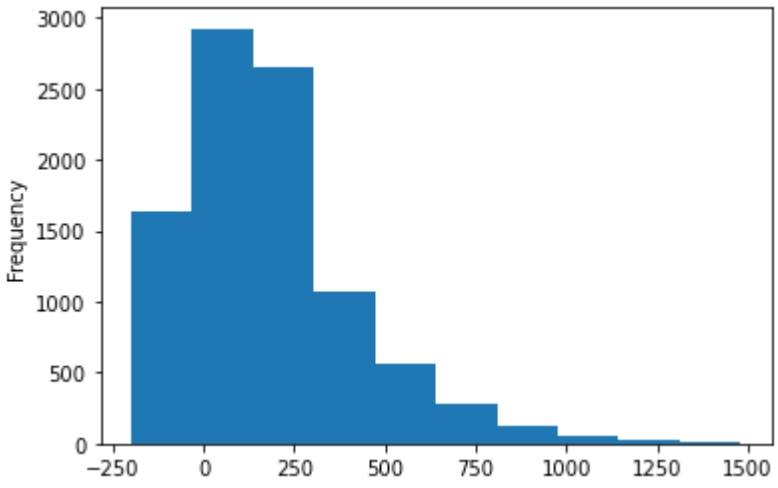
Out[6]:
```
Date             object
Time             object
CO(GT)          float64
PT08.S1(CO)     float64
NMHC(GT)        float64
C6H6(GT)        float64
PT08.S2(NMHC)   float64
NOx(GT)         float64
PT08.S3(NOx)    float64
NO2(GT)         float64
PT08.S4(NO2)    float64
PT08.S5(O3)     float64
T               float64
RH              float64
AH              float64
Unnamed: 15     float64
Unnamed: 16     float64
dtype: object
```

In [7]:
```
sns.heatmap(df.isnull())
```

Out[7]:
```
<AxesSubplot:>
```



In [13]:
```
df['NOx(GT)'].plot.hist()
plt.show()
```



In [14]:
```
df['Date'].head() # its object
```

Out[14]:
```
0    10-03-2004
1    10-03-2004
2    10-03-2004
3    10-03-2004
4    10-03-2004
Name: Date, dtype: object
```

In [1]:
```
#df['Date'].dropna()
#df['Date']=pd.to_datetime(df['Date'])
```

In [16]:
```
df['Date'].head() # now datatype is datetime
```

Out[16]:
```
0   2004-10-03
1   2004-10-03
2   2004-10-03
3   2004-10-03
4   2004-10-03
Name: Date, dtype: datetime64[ns]
```

In [17]:
```
df['Time'].head() # its object
```

Out[17]:
```
0    18:00:00
1    19:00:00
2    20:00:00
3    21:00:00
4    22:00:00
Name: Time, dtype: object
```

In [18]:
```
df['Time'].dropna()
df['Time']=pd.to_datetime(df['Time'])
```

In [19]:
```
df['Time'].head() # now datatype is datetime
```

Out[19]:
```
0   2022-06-17  18:00:00
1   2022-06-17  19:00:00
2   2022-06-17  20:00:00
3   2022-06-17  21:00:00
4   2022-06-17  22:00:00
Name: Time, dtype: datetime64[ns]
```

In [20]:
```
df['day']=df["Date"].apply(lambda  x:  x.day)
df['month']=df["Date"].apply(lambda  x:  x.month)
df['year']=df["Date"].apply(lambda x: x.year)
```

In [21]:
```
df.head()
```

Out[21]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08
0	2004-10-03	2022-06-17 18:00:00	2.6	1360.0	150.0	11.9	1046.0	166.0	
1	2004-10-03	2022-06-17	2.0	1292.0	112.0	9.4	955.0	103.0	
2	2004-10-03	2022-06-17 20:00:00	2.2	1402.0	88.0	9.0	939.0	131.0	
3	2004-10-03	2022-06-17	2.2	1376.0	80.0	9.2	948.0	172.0	
4	2004-10-03	2022-06-17 22:00:00	1.6	1272.0	51.0	6.5	836.0	131.0	

In [22]:
```
df['hour']=df["Time"].apply(lambda x: x.hour)
df['minute']=df["Time"].apply(lambda x: x.minute)
df['second']=df["Time"].apply(lambda x: x.second)
```

In [23]:
```
df.head()
```

Out[23]:

	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08
0	2004-10-03	2022-06-17 18:00:00	2.6	1360.0	150.0	11.9	1046.0	166.0	
1	2004-10-03	2022-06-17 19:00:00	2.0	1292.0	112.0	9.4	955.0	103.0	
2	2004-10-03	2022-06-17 20:00:00	2.2	1402.0	88.0	9.0	939.0	131.0	
3	2004-10-03	2022-06-17 21:00:00	2.2	1376.0	80.0	9.2	948.0	172.0	
4	2004-10-03	2022-06-17 22:00:00	1.6	1272.0	51.0	6.5	836.0	131.0	

5 rows × 23 columns

In [24]:
```
df=df.drop(['Date','Time'],axis=1)
```

In [25]:
```
print("Number of null values in each column:\n{}".format(df.isnull().sum()))
```

```
Number of null values in each column:
CO(GT)            114
PT08.S1(CO)       114
NMHC(GT)          114
C6H6(GT)          114
PT08.S2(NMHC)     114
NOx(GT)           114
PT08.S3(NOx)      114
NO2(GT)           114
PT08.S4(NO2)      114
PT08.S5(O3)       114
T                 114
RH                114
AH                114
Unnamed: 15      9471
Unnamed: 16      9471
day               114
month             114
year              114
hour              114
minute            114
second            114
dtype: int64
```

In [28]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9471 entries, 0 to 9470
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   CO(GT)         9471 non-null   float64
 1   PT08.S1(CO)    9471 non-null   float64
 2   NMHC(GT)       9471 non-null   float64
 3   C6H6(GT)       9471 non-null   float64
 4   PT08.S2(NMHC)  9471 non-null   float64
 5   NOx(GT)        9471 non-null   float64
 6   PT08.S3(NOx)   9471 non-null   float64
 7   NO2(GT)        9471 non-null   float64
 8   PT08.S4(NO2)   9471 non-null   float64
 9   PT08.S5(O3)    9471 non-null   float64
 10  T              9471 non-null   float64
 11  RH             9471 non-null   float64
 12  AH             9471 non-null   float64
 13  Unnamed: 15    0 non-null      float64
 14  Unnamed: 16    0 non-null      float64
 15  day            9471 non-null   float64
 16  month          9471 non-null   float64
 17  year           9471 non-null   float64
 18  hour           9471 non-null   float64
 19  minute         9471 non-null   float64
 20  second         9471 non-null   float64
dtypes: float64(21)
memory usage: 1.5 MB
```

In [26]:
```
for i in df.columns:
    df[i]=df[i].fillna(df[i].mean())
```

In [29]:
```
feat=df.drop(['AH','Unnamed:  15','Unnamed:  16'],axis=1)
val=df['AH'].values
```

In [30]:
```
feat.head()
```

Out[30]:

	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(G
0	2.6	1360.0	150.0	11.9	1046.0	166.0	1056.0	11
1	2.0	1292.0	112.0	9.4	955.0	103.0	1174.0	9
2	2.2	1402.0	88.0	9.0	939.0	131.0	1140.0	11
3	2.2	1376.0	80.0	9.2	948.0	172.0	1092.0	12
4	1.6	1272.0	51.0	6.5	836.0	131.0	1205.0	11

In [33]:
```
(train_feat,test_feat,train_classes,test_classes)=train_test_split(feat,val,random
```

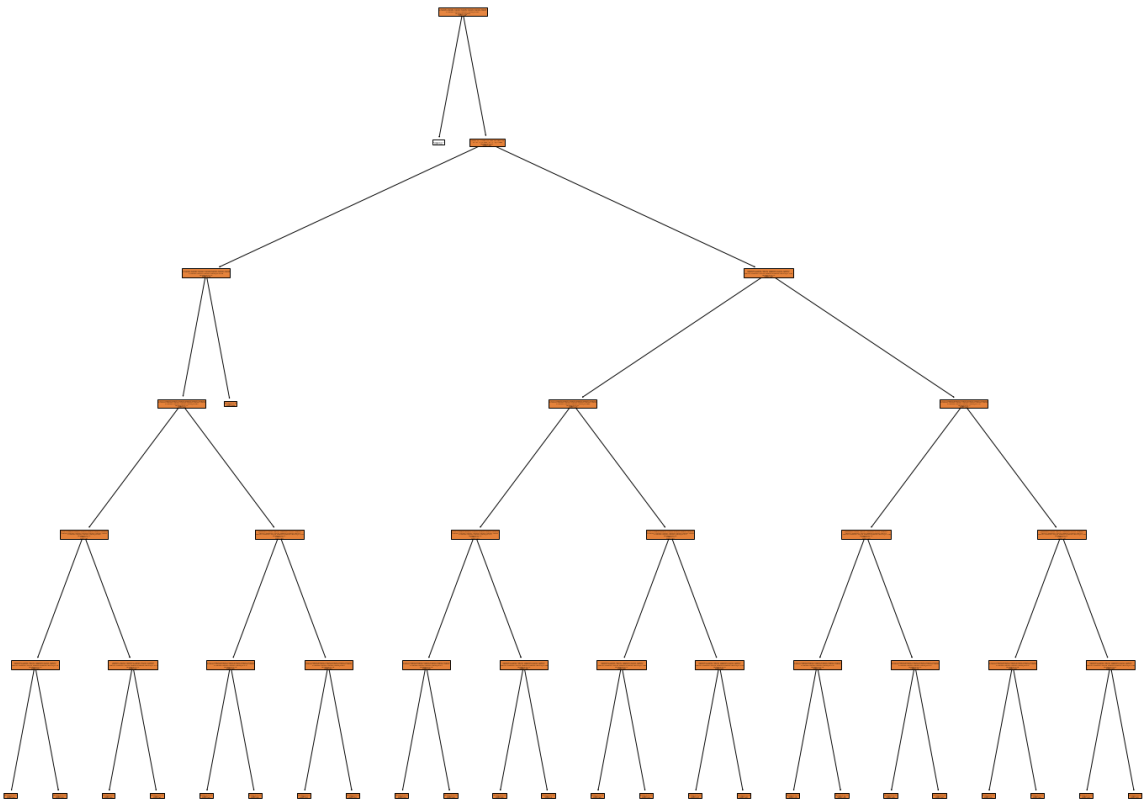
In [57]:
```
m=DecisionTreeRegressor(max_depth=6).fit(train_feat,train_classes)  #for  max  depth
#m=DecisionTreeRegressor().fit(train_feat,train_classes)
```

In [58]:
```
ypred=m.predict(test_feat)
```

In [59]:
```
from sklearn import metrics

pred=m.predict(test_feat)
print("MSE  ",metrics.mean_squared_error(test_classes,ypred))
```

```
MSE  0.029433227485455835
```

In [60]:
```
f=feat.values
from sklearn import tree


fig=plt.figure(figsize=(25,20))
_=tree.plot_tree(m,feature_names=f,class_names=val,filled=True)
```



In [61]:
```
text_representation= tree.export_text(m)
print(text_representation)
```

```
|--- feature_1 <= 223.50
|   |--- value: [-200.00]
|--- feature_1 >  223.50
|   |--- feature_3 <= 1.88
|   |   |--- feature_4 <= 732.80
|   |   |   |--- feature_8 <= 928.00
|   |   |   |   |--- feature_8 <= 826.50
|   |   |   |   |   |--- feature_10 <= 2.90
|   |   |   |   |   |   |--- value: [0.30]
|   |   |   |   |   |--- feature_10 >  2.90
|   |   |   |   |   |   |--- value: [0.42]
|   |   |   |   |--- feature_8 >  826.50
|   |   |   |   |   |--- feature_9 <= 362.00
|   |   |   |   |   |   |--- value: [0.70]
|   |   |   |   |   |--- feature_9 >  362.00
|   |   |   |   |   |   |--- value: [0.56]
|   |   |   |--- feature_8 >  928.00
|   |   |   |   |--- feature_10 <= 19.25
|   |   |   |   |   |--- feature_11 <= 43.75
|   |   |   |   |   |   |--- value: [0.61]
|   |   |   |   |   |--- feature_11 >  43.75
|   |   |   |   |   |   |--- value: [0.88]
|   |   |   |   |--- feature_10 >  19.25
|   |   |   |   |   |--- feature_11 <= 55.20
|   |   |   |   |   |   |--- value: [1.23]
|   |   |   |   |   |--- feature_11 >  55.20
|   |   |   |   |   |   |--- value: [1.77]
|   |   |--- feature_4 >  732.80
|   |   |   |--- value: [-6.84]
|   |--- feature_3 >  1.88
|   |   |--- feature_10 <= 16.85
|   |   |   |--- feature_8 <= 1088.50
|   |   |   |   |--- feature_11 <= 40.05
|   |   |   |   |   |--- feature_11 <= 27.90
|   |   |   |   |   |   |--- value: [0.29]
|   |   |   |   |   |--- feature_11 >  27.90
|   |   |   |   |   |   |--- value: [0.40]
|   |   |   |   |--- feature_11 >  40.05
|   |   |   |   |   |--- feature_10 <= 5.55
|   |   |   |   |   |   |--- value: [0.46]
|   |   |   |   |   |--- feature_10 >  5.55
|   |   |   |   |   |   |--- value: [0.60]
|   |   |   |--- feature_8 >  1088.50
|   |   |   |   |--- feature_11 <= 49.45
|   |   |   |   |   |--- feature_10 <= 13.55
|   |   |   |   |   |   |--- value: [0.50]
|   |   |   |   |   |--- feature_10 >  13.55
|   |   |   |   |   |   |--- value: [0.73]
|   |   |   |   |--- feature_11 >  49.45
|   |   |   |   |   |--- feature_10 <= 11.25
|   |   |   |   |   |   |--- value: [0.77]
|   |   |   |   |   |--- feature_10 >  11.25
|   |   |   |   |   |   |--- value: [1.04]
|   |   |--- feature_10 >  16.85
|   |   |   |--- feature_11 <= 47.95
|   |   |   |   |--- feature_10 <= 25.95
|   |   |   |   |   |--- feature_11 <= 37.95
|   |   |   |   |   |   |--- value: [0.79]
|   |   |   |   |   |--- feature_11 >  37.95
|   |   |   |   |   |   |--- value: [1.13]
|   |   |   |   |--- feature_10 >  25.95
|   |   |   |   |   |--- feature_11 <= 24.25
|   |   |   |   |   |   |--- value: [1.04]
|   |   |   |   |   |--- feature_11 >  24.25
```

```
|   |   |   |   |   |   |--- value: [1.44]
|   |   |   |--- feature_11 >  47.95
|   |   |   |   |--- feature_10 <= 21.05
|   |   |   |   |   |--- feature_11 <= 62.65
|   |   |   |   |   |   |--- value: [1.19]
|   |   |   |   |   |--- feature_11 >  62.65
|   |   |   |   |   |   |--- value: [1.53]
|   |   |   |   |--- feature_10 >  21.05
|   |   |   |   |   |--- feature_10 <= 26.05
|   |   |   |   |   |   |--- value: [1.62]
|   |   |   |   |   |--- feature_10 >  26.05
|   |   |   |   |   |   |--- value: [1.91]
```

Practical 19 - Decision Tree Classifier – Iris Dataset

17 June 2022

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.tree import DecisionTreeClassifier
```

In [2]:
```
data=pd.read_csv("D:\\ML\\iris.csv")
```

In [3]:
```
data
```

Out[3]:

	Sepal length	Sepal width	Petal length	Petal width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Sepal length  150 non-null    float64
 1   Sepal width   150 non-null    float64
 2   Petal length  150 non-null    float64
 3   Petal width   150 non-null    float64
 4   Class         150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [5]:
```
data.describe()
```

Out[5]:

	Sepal length	Sepal width	Petal length	Petal width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

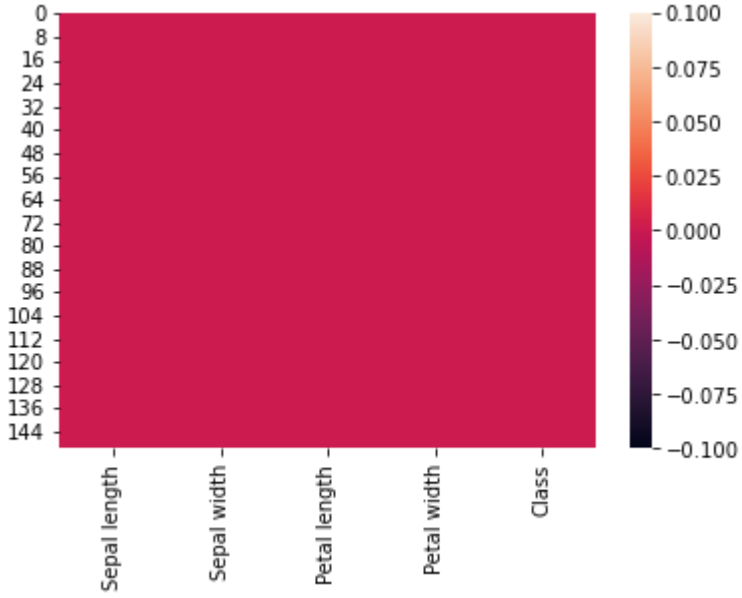
In [6]:
```
data.head()
```

Out[6]:

	Sepal length	Sepal width	Petal length	Petal width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

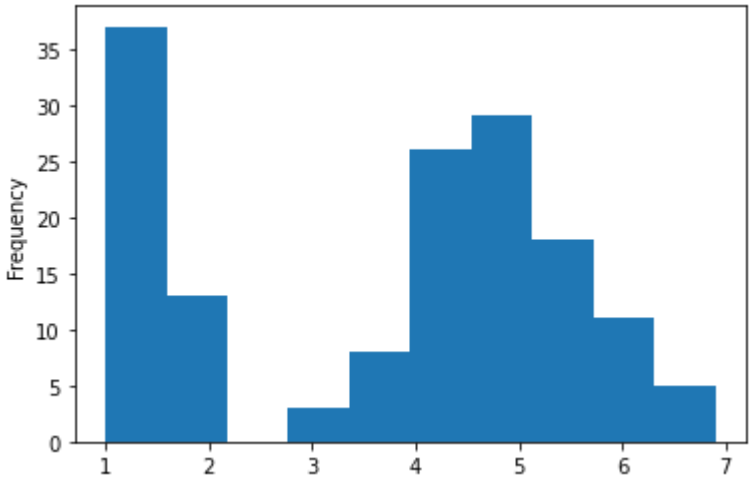
In [7]:
```
sns.heatmap(data.isnull())
```

Out[7]:
```
<AxesSubplot:>
```



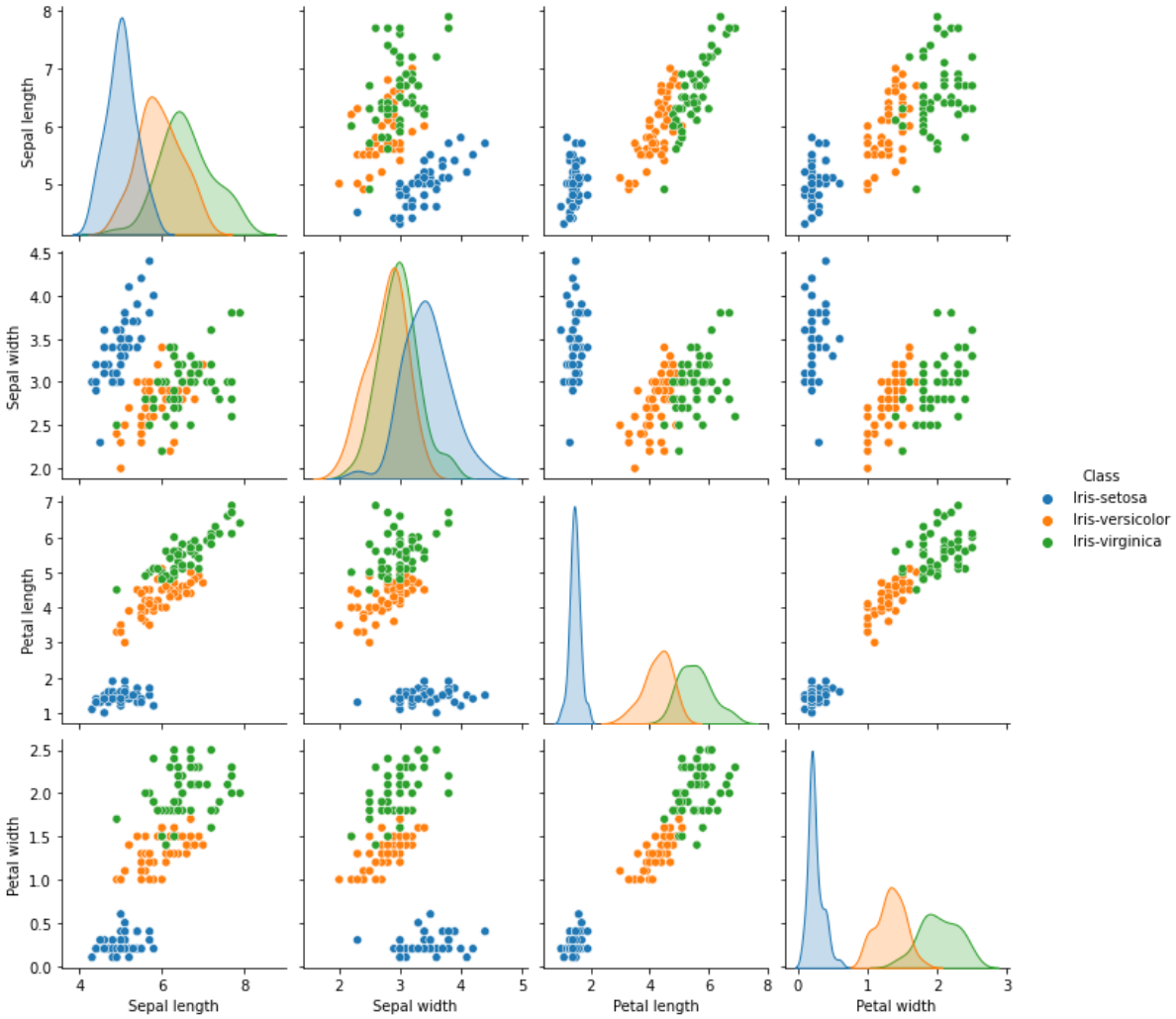
No null values

In [8]:
```
data['Petal  length'].plot.hist()
plt.show()
```



In [9]:
```
sns.pairplot(data,hue='Class')
```

Out[9]:
```
<seaborn.axisgrid.PairGrid  at  0x1e970f379a0>
```



In [10]:
```
features = data[['Sepal length','Sepal width','Petal length','Petal width']].value
classes = data['Class'].values
```

In [11]:
```
(train_feat,test_feat,train_classes,test_classes)=train_test_split(features,classe
```

In [12]:
```
#Training
dectree=DecisionTreeClassifier(criterion='gini')
dectree.fit(train_feat,train_classes)
```

Out[12]:
```
DecisionTreeClassifier()
```

In [13]:
```
from sklearn import metrics

pred=dectree.predict(test_feat)
print("accuracy",metrics.accuracy_score(test_classes,pred))
```

```
accuracy 0.8083333333333333
```

In [14]:
```
print(classification_report(test_classes,pred))
```

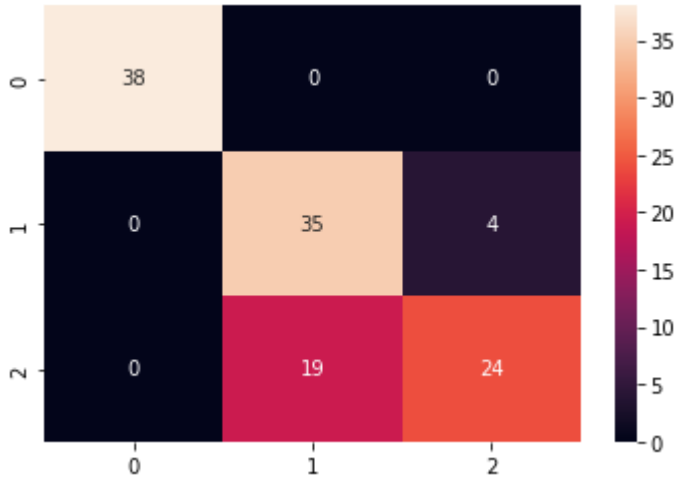
```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        38
Iris-versicolor       0.65      0.90      0.75        39
 Iris-virginica       0.86      0.56      0.68        43

       accuracy                           0.81       120
      macro avg       0.84      0.82      0.81       120
   weighted avg       0.83      0.81      0.80       120
```

In [15]:
```
cf=confusion_matrix(test_classes,pred)
cf
```

Out[15]:
```
array([[38,  0,  0],
       [ 0, 35,  4],
       [ 0, 19, 24]], dtype=int64)
```

In [16]:
```
sns.heatmap(cf,annot=True)
```

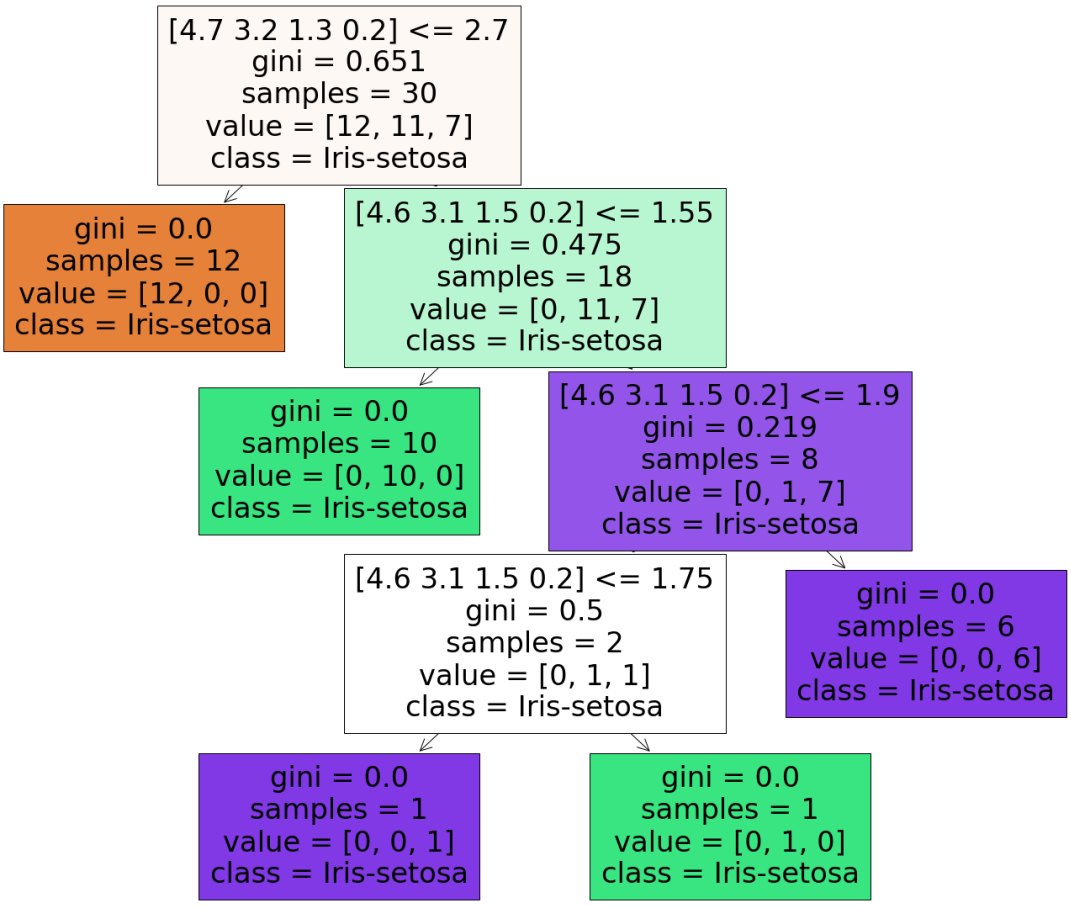
Out[16]:
```
<AxesSubplot:>
```



In [17]:
```
#predicting a single input feature
sepl=input("Sepal  length:")
sepw=input("Sepal width:")
petl=input("Petal  length:")
petw=input("Petal width:")
print(sepl,sepw,petl,petw)
pr=dectree.predict(np.column_stack([sepl,sepw,petl,petw]))
print("Predicted Species is ",pr)
```

```
Sepal  length:4
Sepal width:4
Petal  length:4
Petal width:4
4 4 4 4
Predicted Species is  ['Iris-virginica']
```

In [18]:
```
from sklearn import tree

fig=plt.figure(figsize=(25,20))
_=tree.plot_tree(dectree,feature_names=features,
                 class_  names=classes,filled=True)
```



In [19]:
```
text_representation= tree.export_text(dectree)
print(text_representation)
```

```
|--- feature_2 <= 2.70
|   |--- class: Iris-setosa
|--- feature_2 >  2.70
|   |--- feature_3 <= 1.55
|   |   |--- class: Iris-versicolor
|   |--- feature_3 >  1.55
|   |   |--- feature_3 <= 1.90
|   |   |   |--- feature_3 <= 1.75
|   |   |   |   |--- class: Iris-virginica
|   |   |   |--- feature_3 >  1.75
|   |   |   |   |--- class: Iris-versicolor
|   |   |--- feature_3 >  1.90
|   |   |   |--- class: Iris-virginica
```

Entropy

In [20]:
```
(train_feat,test_feat,train_classes,test_classes)=train_test_split(features,classe
```

In [21]:
```
#Training
dectree=DecisionTreeClassifier(criterion='entropy')
dectree.fit(train_feat,train_classes)
```

Out[21]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [22]:
```
from sklearn import metrics
```

```
pred=dectree.predict(test_feat)
print("accuracy",metrics.accuracy_score(test_classes,pred))
```

```
accuracy 0.95
```

In [23]:
```
print(classification_report(test_classes,pred))
```

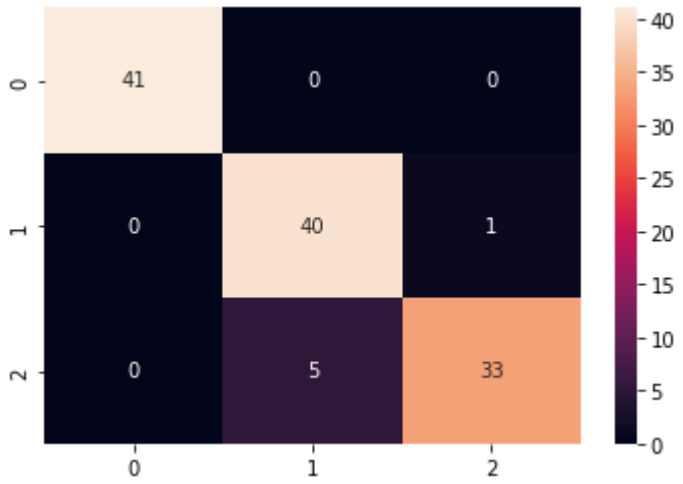
```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        41
Iris-versicolor       0.89      0.98      0.93        41
 Iris-virginica       0.97      0.87      0.92        38

      accuracy                           0.95       120
     macro avg       0.95      0.95      0.95       120
  weighted avg       0.95      0.95      0.95       120
```

In [24]:
```
cf=confusion_matrix(test_classes,pred)
cf
```

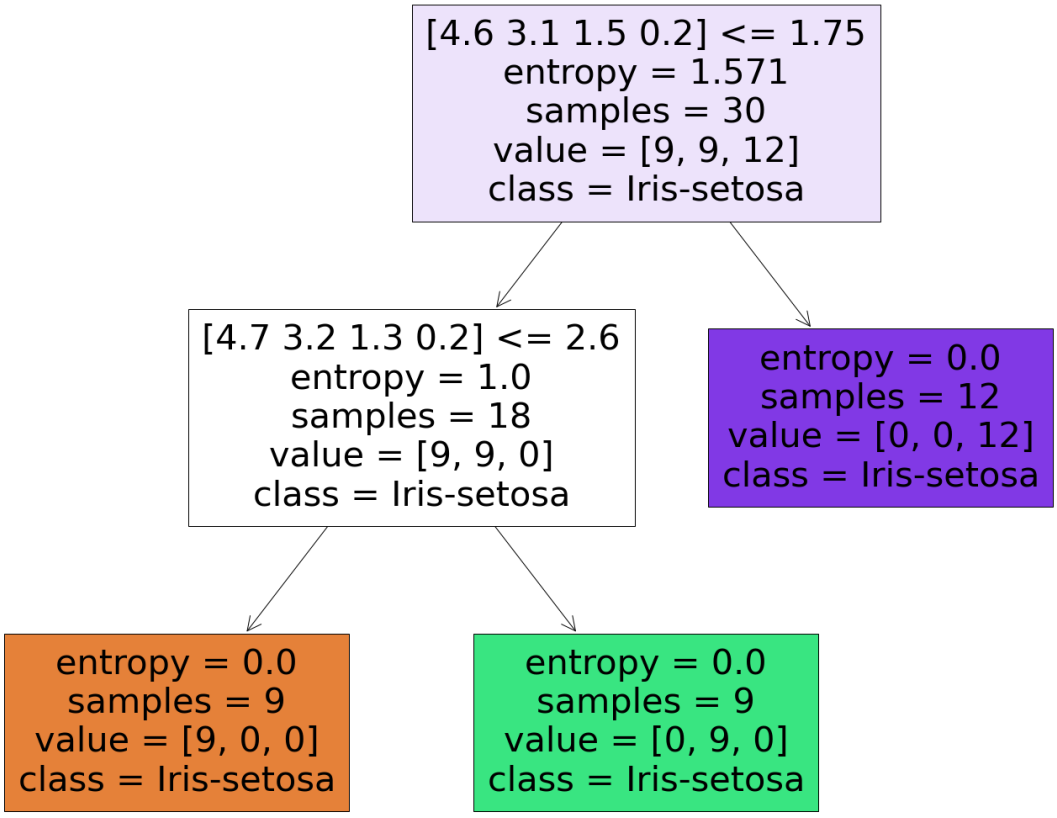
Out[24]:
```
array([[41,  0,  0],
       [ 0, 40,  1],
       [ 0,  5, 33]], dtype=int64)
```

In [25]:
```
sns.heatmap(cf,annot=True)
```

Out[25]:
```
<AxesSubplot:>
```



In [26]:
```
from sklearn import tree

fig=plt.figure(figsize=(25,20))
_=tree.plot_tree(dectree,feature_names=features,
                 class_names=classes,filled=True)
```



Specifing the Comparison between Gini and Entropy , **Entropy is 95% accurate** and *Gini is 81% accurate*

In [28]:
```
import os
os.environ["PATH"]='C:/Program  Files(x86)/Graphviz2.38/bin/'
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data=StringIO()
feat_col=['Sepal  length','Sepal  width','Petal  length','Petal  width']
export_graphviz(dectree,out_file=dot_data,
                filled=True,rounded=True,
                 special_characters=True,feature_names=
feat_col,class_names=['Setosa','Versicolor','Virginica'])
graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('iris.png')
Image(graph.create_png())
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Input In [28], in <cell line: 3>()
      1 import os
      2 os.environ["PATH"]='C:/Program  Files(x86)/Graphviz2.38/bin/'
----> 3 from sklearn.externals.six import StringIO
      4 from IPython.display import Image
      5 from sklearn.tree import export_graphviz

ModuleNotFoundError: No module named 'sklearn.externals.six'
```

In [ ]:

Practical 20 - Decision Tree Classifier – Wine Dataset

17 June 2022

In [2]:
```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [3]:
```
wine=pd.read_csv("D:\\ML\\winequality-red.csv",sep=';')
```

In [4]:
```
wine.head()
```

Out[4]:

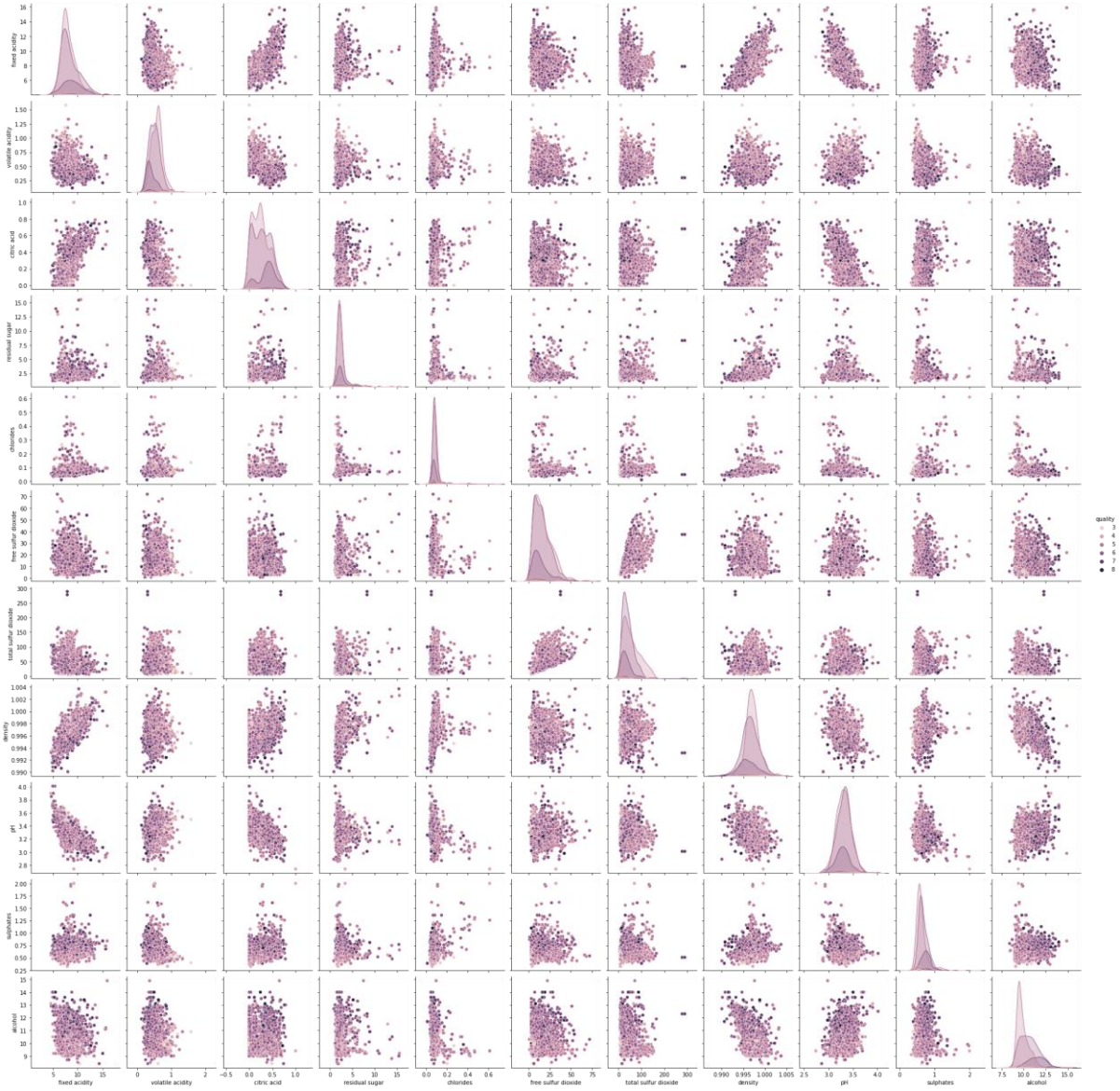
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [5]:
```
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [6]:
```
sns.pairplot(wine,hue='quality')
```

Out[6]:
```
<seaborn.axisgrid.PairGrid at 0x2149bc9e6a0>
```



In [8]:
```
x=wine[['fixed  acidity','volatile  acidity','citric  acid','residual  sugar','chlorid
```

In [9]:
```
y=wine['quality'].values
```

In [10]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=34)
```

In [11]:
```
dectree=DecisionTreeClassifier(criterion='entropy')
dectree.fit(x_train,y_train)
```

Out[11]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [12]:
```
dectree.score(x_test,y_test)
```

Out[12]:
```
1.0
```

In [13]:
```
from sklearn import metrics
pred = dectree.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test,pred))
```

```
Accuracy: 1.0
```

In [14]:
```
cf = confusion_matrix(y_test,pred)
```

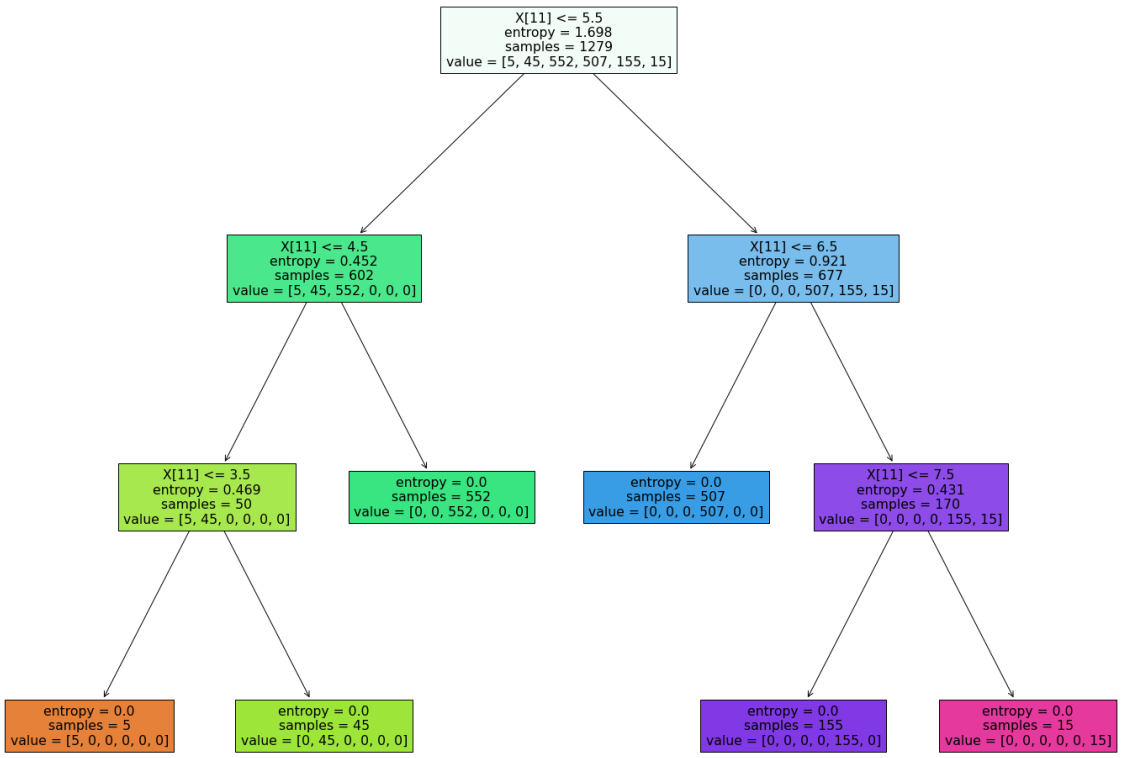
In [15]:
```
cf
```

Out[15]:
```
array([[  5,   0,   0,   0,   0,   0],
       [  0,   8,   0,   0,   0,   0],
       [  0,   0, 129,   0,   0,   0],
       [  0,   0,   0, 131,   0,   0],
       [  0,   0,   0,   0,  44,   0],
       [  0,   0,   0,   0,   0,   3]], dtype=int64)
```

In [16]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           3       1.00      1.00      1.00         5
           4       1.00      1.00      1.00         8
           5       1.00      1.00      1.00       129
           6       1.00      1.00      1.00       131
           7       1.00      1.00      1.00        44
           8       1.00      1.00      1.00         3

    accuracy                           1.00       320
   macro avg       1.00      1.00      1.00       320
weighted avg       1.00      1.00      1.00       320
```

In [17]:
```
from sklearn import tree
fig=plt.figure(figsize=(25,20))
_=tree.plot_tree(dectree,filled=True)
```



In [18]:
```
text_representation=tree.export_text(dectree)
print(text_representation)
```

```
|--- feature_11 <= 5.50
|   |--- feature_11 <= 4.50
|   |   |--- feature_11 <= 3.50
|   |   |   |--- class: 3
|   |   |--- feature_11 >  3.50
|   |   |   |--- class: 4
|   |--- feature_11 >  4.50
|   |   |--- class: 5
|--- feature_11 >  5.50
|   |--- feature_11 <= 6.50
|   |   |--- class: 6
|   |--- feature_11 >  6.50
|   |   |--- feature_11 <= 7.50
|   |   |   |--- class: 7
|   |   |--- feature_11 >  7.50
|   |   |   |--- class: 8
```

In [ ]:

Practical 21 - Naive Bayes Classification - Glass Identification

20 June 2022

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.naive_bayes import GaussianNB,MultinomialNB,BernoulliNB
from sklearn.model_selection import train_test_split
import warnings
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [2]:
```
df=pd.read_csv("D:\\ML\\glass.data",sep=",")
```

In [3]:
```
df
```

Out[3]:

	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.00.1	1.1
0	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.00	1
1	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.00	1
2	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.00	1
3	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.00	1
4	6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.00	0.26	1
...	...	...	...	...	...	...	...	...	...	...	...
208	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.00	7
209	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.00	7
210	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.00	7
211	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.00	7
212	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.00	7

213 rows × 11 columns

In [4]:
```
col_names=["Id","RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type of glass"]
```

In [5]:
```
df.columns=col_names
```

In [6]:
```
df
```

Out[6]:

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type of glass
0	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.00	1
1	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.00	1
2	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.00	1
3	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.00	1
4	6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.00	0.26	1
...	...	...	...	...	...	...	...	...	...	...	...
208	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.00	7
209	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.00	7
210	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.00	7
211	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.00	7
212	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.00	7

213 rows × 11 columns

In [7]:
```
#create a naive bayes obj
nb=BernoulliNB()
gnb=GaussianNB()
mnb=MultinomialNB()
```

In [8]:
```
x=df.drop(columns=['Type of glass'])
y=df['Type of glass']
```

In [9]:
```
x
```

Out[9]:

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
0	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.00
1	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.00
2	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.00
3	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.00
4	6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.00	0.26
...	...	...	...	...	...	...	...	...	...	...
208	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.00
209	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.00
210	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.00
211	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.00
212	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.00

213 rows × 10 columns

In [10]:
```
y
```

Out[10]:
```
0      1
1      1
2      1
3      1
4      1
      ..
208    7
209    7
210    7
211    7
212    7
Name: Type of glass, Length: 213, dtype: int64
```

In [11]:
```
df.describe()
```

Out[11]:

	Id	RI	Na	Mg	Al	Si	K	
count	213.000000	213.000000	213.000000	213.000000	213.000000	213.000000	213.000000	213.00
mean	108.000000	1.518353	13.406761	2.676056	1.446526	72.655023	0.499108	8.95
std	61.631972	0.003039	0.818371	1.440453	0.499882	0.774052	0.653035	1.42
min	2.000000	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.43
25%	55.000000	1.516520	12.900000	2.090000	1.190000	72.280000	0.130000	8.24
50%	108.000000	1.517680	13.300000	3.480000	1.360000	72.790000	0.560000	8.60
75%	161.000000	1.519150	13.830000	3.600000	1.630000	73.090000	0.610000	9.18
max	214.000000	1.533930	17.380000	3.980000	3.500000	75.410000	6.210000	16.19

In [12]:
```
df.head()
```

Out[12]:

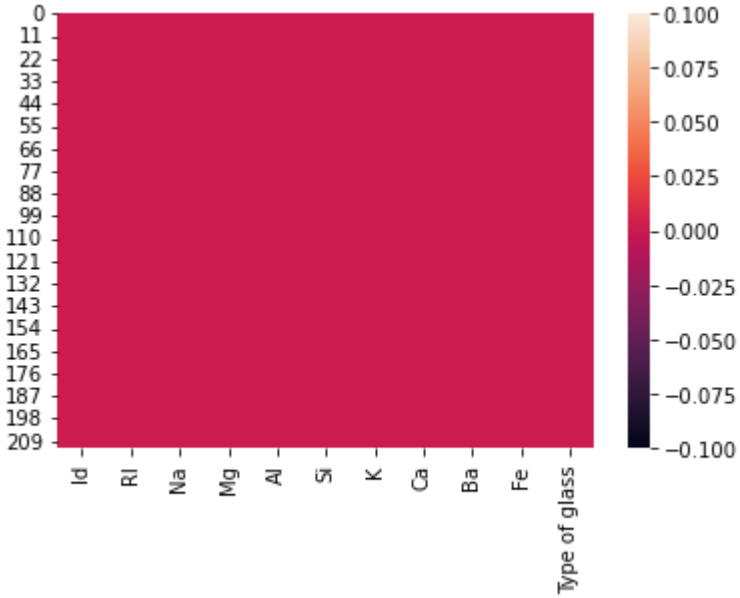
	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type of glass
0	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.00	1
1	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.00	1
2	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.00	1
3	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.00	1
4	6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.0	0.26	1

In [13]:
```
df.info(
)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213 entries, 0 to 212
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             213 non-null    int64  
 1   RI             213 non-null    float64
 2   Na             213 non-null    float64
 3   Mg             213 non-null    float64
 4   Al             213 non-null    float64
 5   Si             213 non-null    float64
 6   K              213 non-null    float64
 7   Ca             213 non-null    float64
 8   Ba             213 non-null    float64
 9   Fe             213 non-null    float64
 10  Type of glass  213 non-null    int64  
dtypes: float64(9), int64(2)
memory usage: 18.4 KB
```

In [14]:
```
sns.heatmap(df.isnull())
```

Out[14]:
```
<AxesSubplot:>
```



NO NULL VALUES

In [15]:
```
#split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=4)
```

In [16]:
```
nb.fit(x_train,y_train)
gnb.fit(x_train,y_train)
mnb.fit(x_train,y_train)
```

Out[16]:
```
MultinomialNB()
```

In [17]:
```
y_pred=nb.predict(x_test)
y_predg=gnb.predict(x_test)
y_predm=mnb.predict(x_test)
```

In [18]:
```
print(accuracy_score(y_test,y_pred))
print(accuracy_score(y_test,y_predg))
print(accuracy_score(y_test,y_predm))
```

```
0.4883720930232558
0.8604651162790697
0.7906976744186046
```

Gaussian has most accuracy

In [19]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           1       0.39      0.92      0.55        13
           2       0.67      0.11      0.19        18
           5       0.00      0.00      0.00         4
           6       0.50      1.00      0.67         1
           7       0.86      0.86      0.86         7

    accuracy                           0.49        43
   macro avg       0.48      0.58      0.45        43
weighted avg       0.55      0.49      0.40        43
```

```
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMe
tricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMe
tricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMe
tricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [20]:
```
print(classification_report(y_test,y_predg))
```

```
              precision    recall  f1-score   support

           1       0.87      1.00      0.93        13
           2       0.88      0.83      0.86        18
           5       1.00      0.25      0.40         4
           6       1.00      1.00      1.00         1
           7       0.78      1.00      0.88         7

    accuracy                           0.86        43
   macro avg       0.91      0.82      0.81        43
weighted avg       0.87      0.86      0.84        43
```

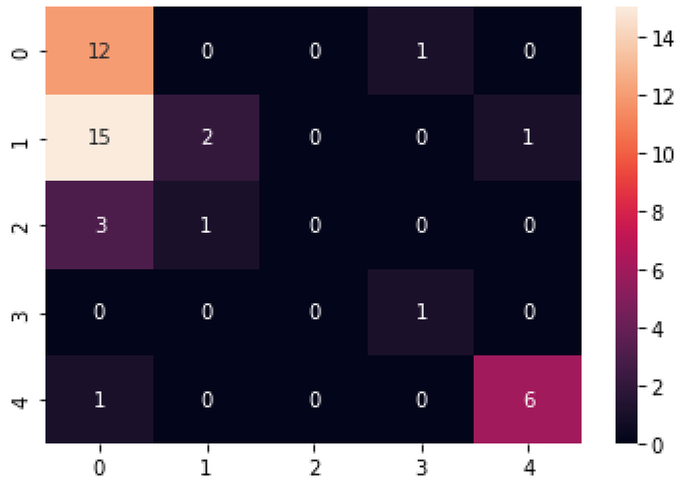
In [21]:
```
print(classification_report(y_test,y_predm))
```

```
              precision    recall  f1-score   support

           1       1.00      0.85      0.92        13
           2       0.88      0.83      0.86        18
           3       0.00      0.00      0.00         0
           5       1.00      0.25      0.40         4
           6       0.00      0.00      0.00         1
           7       0.78      1.00      0.88         7

    accuracy                           0.79        43
   macro avg       0.61      0.49      0.51        43
weighted avg       0.89      0.79      0.82        43
```

```
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
D:\anaconda\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

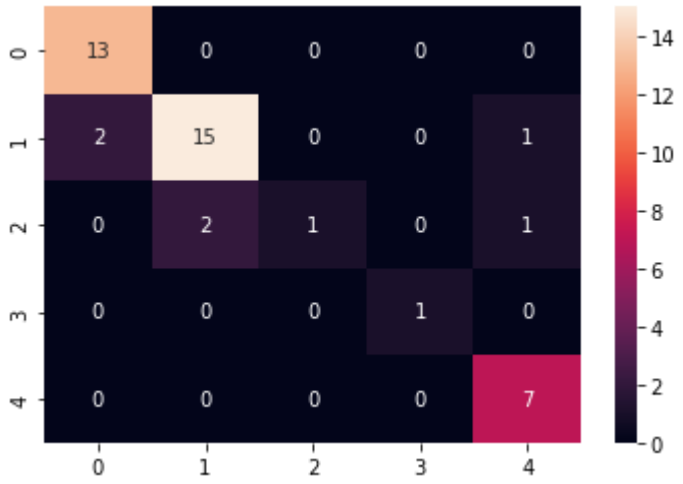
In [22]:
```
cv1=confusion_matrix(y_test,y_pred)
cv1
sns.heatmap(cv1,annot=True)
```

Out[22]: <AxesSubplot:>



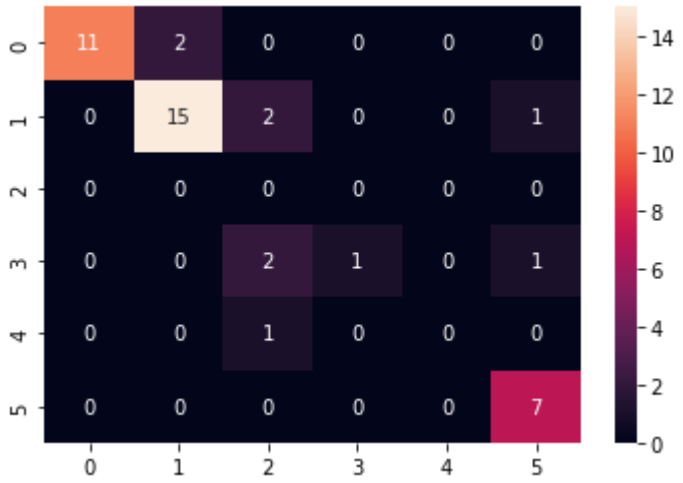
In [23]:
```
cv2=confusion_matrix(y_test,y_predg)
cv2
sns.heatmap(cv2,annot=True)
```

Out[23]: <AxesSubplot:>



In [24]:
```
cv3=confusion_matrix(y_test,y_predm)
cv3
sns.heatmap(cv3,annot=True)
```

Out[24]: <AxesSubplot:>



In [ ]:

Practical 22 - K Nearest Neighbor – Bank Dataset

21 June 2022

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
```

In [2]:
```
bank=pd.read_csv('D:\\ML\\bank.csv',sep=';')
```

In [5]:
```
bank
```

Out[5]:

	age	job	marital	education	default	balance	housing	loan	contact	day	m
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	
...	...	...	...	...	...	...	...	...	...	...	
4516	33	services	married	secondary	no	-333	yes	no	cellular	30	
4517	57	self-employed	married	tertiary	yes	-3313	yes	yes	unknown	9	
4518	57	technician	married	secondary	no	295	no	no	cellular	19	
4519	28	blue-collar	married	secondary	no	1137	no	no	cellular	6	
4520	44	entrepreneur	single	tertiary	no	1136	yes	yes	cellular	3	

4521 rows × 17 columns

In [6]:
```
bank['poutcome']=bank['poutcome'].map({'failure':-1,'unknown':0,'success':1,'other
```

In [7]:
```
bank['default'].unique()
```

Out[7]:
```
array(['no', 'yes'], dtype=object)
```

In [8]:
```
bank['housing'].unique()
```

Out[8]:
```
array(['no', 'yes'], dtype=object)
```

In [9]:
```
bank['loan'].unique()
```

Out[9]:
```
array(['no', 'yes'], dtype=object)
```

In [10]:
```
bank['default']=bank['default'].map({'yes':0,'no':1})
bank['housing']=bank['housing'].map({'yes':0,'no':1})
bank['loan']=bank['loan'].map({'yes':0,'no':1})
```

In [11]:
```
nominal=['job','marital','education','contact','month']
data=pd.get_dummies(bank,columns=nominal)
```

In [12]:
```
data.shape
```

Out[12]:
```
(4521, 46)
```

In [13]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 46 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  4521 non-null   int64
 1   default              4521 non-null   int64
 2   balance              4521 non-null   int64
 3   housing              4521 non-null   int64
 4   loan                 4521 non-null   int64
 5   day                  4521 non-null   int64
 6   duration             4521 non-null   int64
 7   campaign             4521 non-null   int64
 8   pdays                4521 non-null   int64
 9   previous             4521 non-null   int64
 10  poutcome             4521 non-null   int64
 11  y                    4521 non-null   object
 12  job_admin.           4521 non-null   uint8
 13  job_blue-collar      4521 non-null   uint8
 14  job_entrepreneur     4521 non-null   uint8
 15  job_housemaid        4521 non-null   uint8
 16  job_management       4521 non-null   uint8
 17  job_retired          4521 non-null   uint8
 18  job_self-employed    4521 non-null   uint8
 19  job_services         4521 non-null   uint8
 20  job_student          4521 non-null   uint8
 21  job_technician       4521 non-null   uint8
 22  job_unemployed       4521 non-null   uint8
 23  job_unknown          4521 non-null   uint8
 24  marital_divorced     4521 non-null   uint8
 25  marital_married      4521 non-null   uint8
 26  marital_single       4521 non-null   uint8
 27  education_primary    4521 non-null   uint8
 28  education_secondary  4521 non-null   uint8
 29  education_tertiary   4521 non-null   uint8
 30  education_unknown    4521 non-null   uint8
 31  contact_cellular     4521 non-null   uint8
 32  contact_telephone    4521 non-null   uint8
 33  contact_unknown      4521 non-null   uint8
 34  month_apr            4521 non-null   uint8
 35  month_aug            4521 non-null   uint8
 36  month_dec            4521 non-null   uint8
 37  month_feb            4521 non-null   uint8
 38  month_jan            4521 non-null   uint8
 39  month_jul            4521 non-null   uint8
 40  month_jun            4521 non-null   uint8
 41  month_mar            4521 non-null   uint8
 42  month_may            4521 non-null   uint8
 43  month_nov            4521 non-null   uint8
 44  month_oct            4521 non-null   uint8
 45  month_sep            4521 non-null   uint8
dtypes: int64(11), object(1), uint8(34)
memory usage: 574.1+ KB
```

In [14]:
```
data['y']=data['y'].map({'yes':1,'no':0})
```

In [16]:
```
from sklearn.preprocessing import StandardScaler

data[['age','balance','day','campaign','pdays']]=StandardScaler().fit_transform(da
```

In [17]:
```
data.drop('duration',axis=1,inplace=True)
```

In [18]:
```
op=data.pop('y')
```

In [21]:
```
train_feat,test_feat,train_classes,test_classes=train_test_split(data,op,train_siz
```

In [78]:
```
knn=KNeighborsClassifier(n_neighbors=15)
```

In [79]:
```
knn.fit(train_feat,train_classes)
```

Out[79]:
```
KNeighborsClassifier(n_neighbors=15)
```

In [80]:
```
pred=knn.predict(test_feat)
```

In [81]:
```
print("accuracy",metrics.accuracy_score(test_classes,pred))
```

```
accuracy  0.8928176795580111
```

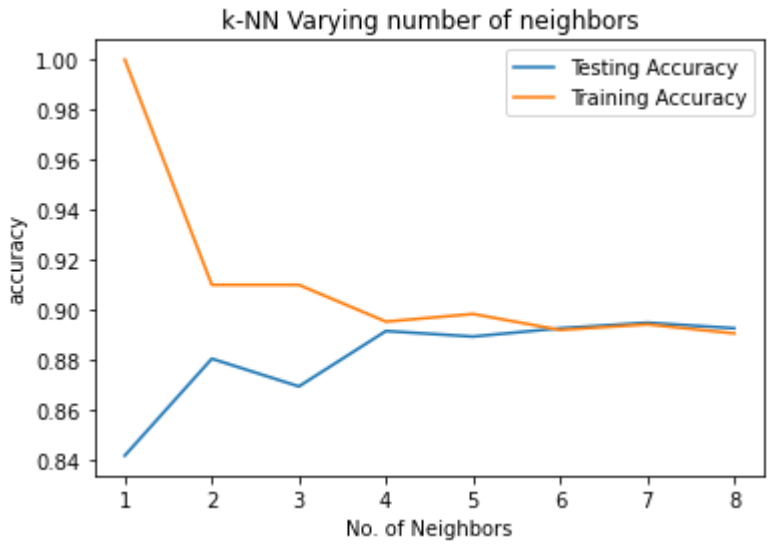
0.8928176795580111 highest for k =15

In [83]:
```
neighbors=np.arange(1,9)
```

In [84]:
```
train_accuracy=np.empty(len(neighbors))
test_accuracy=np.empty(len(neighbors))
```

In [85]:
```
for i,k in enumerate(neighbors):
    #setup a knn classifier with k neighbors
    knn=KNeighborsClassifier(n_neighbors=k)
    #fit the model
    knn.fit(train_feat,train_classes)
    #compute accuracy on tranning set
    train_accuracy[i]=knn.score(train_feat,train_classes)
    #compute accuracy on the test set
    test_accuracy[i]=knn.score(test_feat,test_classes)
```

In [86]:
```
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors,test_accuracy,label='Testing  Accuracy')
plt.plot(neighbors,train_accuracy,label='Training  Accuracy')
plt.legend()
plt.xlabel("No. of Neighbors")
plt.ylabel("accuracy")
plt.show()
```



predicting

In [ ]:

Practical 23 - K Nearest Neighbor – Wine Dataset

21 June 2022

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns
```

In [2]:
```
df=pd.read_csv("D:\\ML\\winequality-red.csv",sep=";")
```

In [3]:
```
df
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	

1599 rows × 12 columns

In [4]:
```
df.describe()
```

Out[4]:

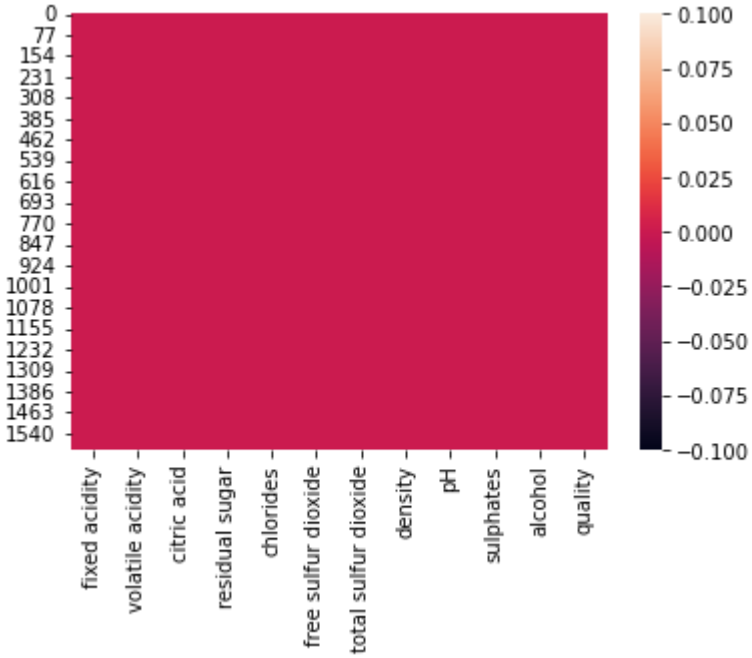
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfu dioxid
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.00000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.46779
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.89532
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.00000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.00000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.00000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.00000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.00000

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [6]:
```
sns.heatmap(df.isnull())
```

Out[6]:
```
<AxesSubplot:>
```



In [7]:
```
features=df.drop('quality',axis=1).values
```

In [8]:
```
features
```

Out[8]:
```
array([[ 7.4  ,  0.7  ,  0.   , ...,  3.51 ,  0.56 ,  9.4  ],
       [ 7.8  ,  0.88 ,  0.   , ...,  3.2  ,  0.68 ,  9.8  ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  3.26 ,  0.65 ,  9.8  ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  3.42 ,  0.75 , 11.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  3.57 ,  0.71 , 10.2  ],
       [ 6.   ,  0.31 ,  0.47 , ...,  3.39 ,  0.66 , 11.   ]])
```

In [9]:
```
classes=df['quality'].values
```

In [10]:
```
classes
```

Out[10]:
```
array([5, 5, 5, ..., 6, 5, 6], dtype=int64)
```

In [11]:
```
train_feat,test_feat,train_classes,test_classes=train_test_split(features,classes,
```

In [12]:
```
knn=KNeighborsClassifier(n_neighbors=1)
```

In [13]:
```
knn.fit(train_feat,train_classes)
```

Out[13]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [14]:
```
pred=knn.predict(test_feat)
```

In [15]:
```
print("accuracy",metrics.accuracy_score(test_classes,pred))
```

```
accuracy 0.584375
```

In [16]:
```
#loop
```

In [17]:
```
neighbors=np.arange(1,9)
```

In [18]:
```
train_accuracy=np.empty(len(neighbors))
test_accuracy=np.empty(len(neighbors))
```

In [19]:
```
for i,k in enumerate(neighbors):
    #setup a knn classifier with k neighbors
    knn=KNeighborsClassifier(n_neighbors=k)
    #fit the model
    knn.fit(train_feat,train_classes)
    #compute accuracy on tranning set
    train_accuracy[i]=knn.score(train_feat,train_classes)
    #compute accuracy on the test set
    test_accuracy[i]=knn.score(test_feat,test_classes)
```

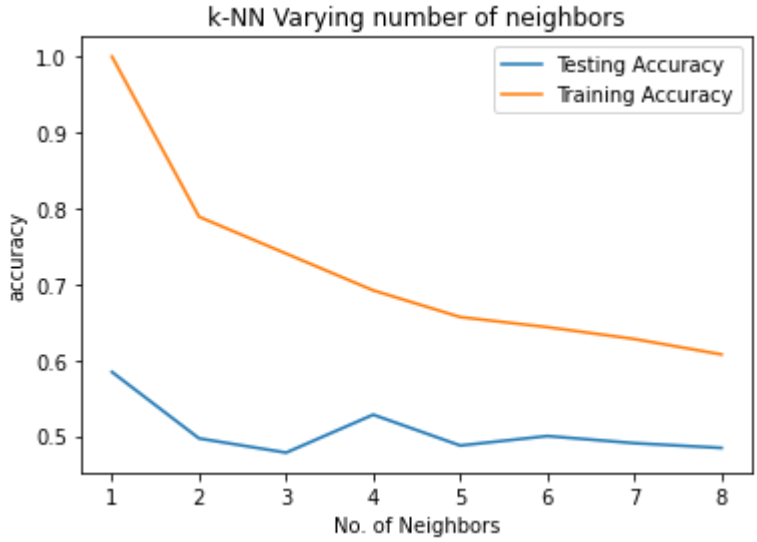
In [20]:
```
train_accuracy[i]
```

Out[20]:
```
0.6075058639562158
```

In [21]:
```
test_accuracy[i]
```

Out[21]:
```
0.484375
```

In [22]:
```
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors,test_accuracy,label='Testing  Accuracy')
plt.plot(neighbors,train_accuracy,label='Training  Accuracy')
plt.legend()
plt.xlabel("No. of Neighbors")
plt.ylabel("accuracy")
plt.show()
```



In [24]:
```
#predicting new value
df
```

Out[24]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	

1599 rows × 12 columns

In [25]:
```
#predicting a single input feature
fa=input("fixed acidity:")
va=input("volatile acidity:")
ca=input("citric acid:")
rs=input("residual sugar:")
c=input("chlorides:")
fso=input("free sulfur dioxide:")
tso=input("total sulfur dioxide:")
d=input("density:")
pH=input("pH:")
sl=input("sulphates:")
ac=input("alcohol:")

#all the inputs
print(fa,va,ca,rs,c,fso,tso,d,pH,sl,ac)

#predict with kNN
pr=knn.predict(np.column_stack([fa,va,ca,rs,c,fso,tso,d,pH,sl,ac]))

#print
print("Predicted quality is ",pr)
```

```
fixed acidity:5
volatile acidity:6
citric acid:7
residual sugar:8
chlorides:4
free sulfur dioxide:5
total sulfur dioxide:6
density:2
pH:4
sulphates:5
alcohol:6
5 6 7 8 4 5 6 2 4 5 6
Predicted quality is  [5]
```

```
D:\anaconda\lib\site-packages\sklearn\base.py:566: FutureWarning: Arrays of byt
es/strings is being converted to decimal numbers if dtype='numeric'. This behav
ior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please
convert your data to numeric values explicitly instead.
  X = check_array(X, **check_params)
```

In [ ]:

Practical 24 - Ensemble Learning – Random Forest Classifier

24 June 2022

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
import seaborn as sns
```

In [3]:
```
data=pd.read_csv('D:\\ML\\iris.csv')
```

In [4]:
```
data
```

Out[4]:

	Sepal length	Sepal width	Petal length	Petal width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [5]:
```
data.head(10)
```

Out[5]:

	Sepal length	Sepal width	Petal length	Petal width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Sepal length  150 non-null    float64
 1   Sepal width   150 non-null    float64
 2   Petal length  150 non-null    float64
 3   Petal width   150 non-null    float64
 4   Class         150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [7]:
```
data.describe()
```

Out[7]:

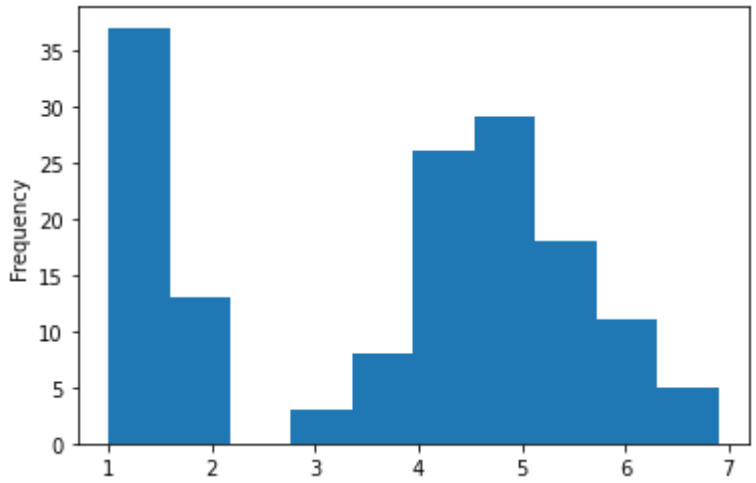
	Sepal length	Sepal width	Petal length	Petal width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [8]:
```
#returns the data types
data.dtypes
```

Out[8]:

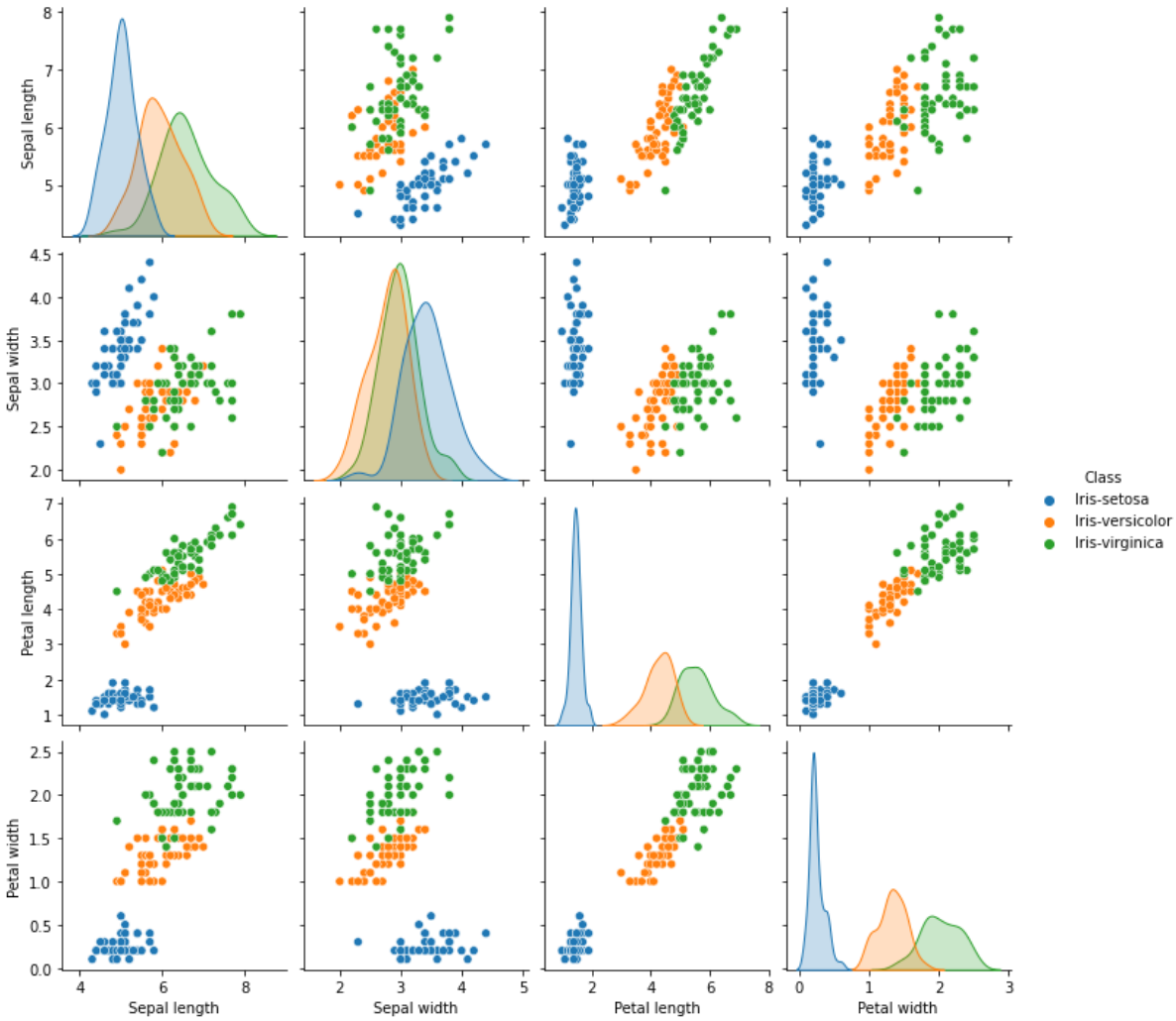
```
Sepal  length  float64
Sepal   width  float64
Petal  length  float64
Petal width    float64
Class           object
dtype: object
```

In [9]:
```
data['Petal  length'].plot.hist()
plt.show()
```



In [10]:
```
#pairplor that plot pairwise relationship in a data set.
sns.pairplot(data,hue='Class')
```

Out[10]:
```
<seaborn.axisgrid.PairGrid  at  0x1d3b5a53190>
```



In [11]:
```
#split dataset into training and testoing dataset
features=data[['Sepal  length','Sepal  width','Petal  length','Petal  width']].values
classes=data['Class'].values
features
classes
```

Out[11]:
```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica'], dtype=object)
```

In [12]:
```
(train_feat,test_feat,train_classes,test_classes)=train_test_split(features,classe
```

In [13]:
```
#Training
dectree=DecisionTreeClassifier(criterion='entropy')
dectree.fit(train_feat,train_classes)
```

Out[13]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [14]:
```
y_predict=dectree.predict(test_feat)
```

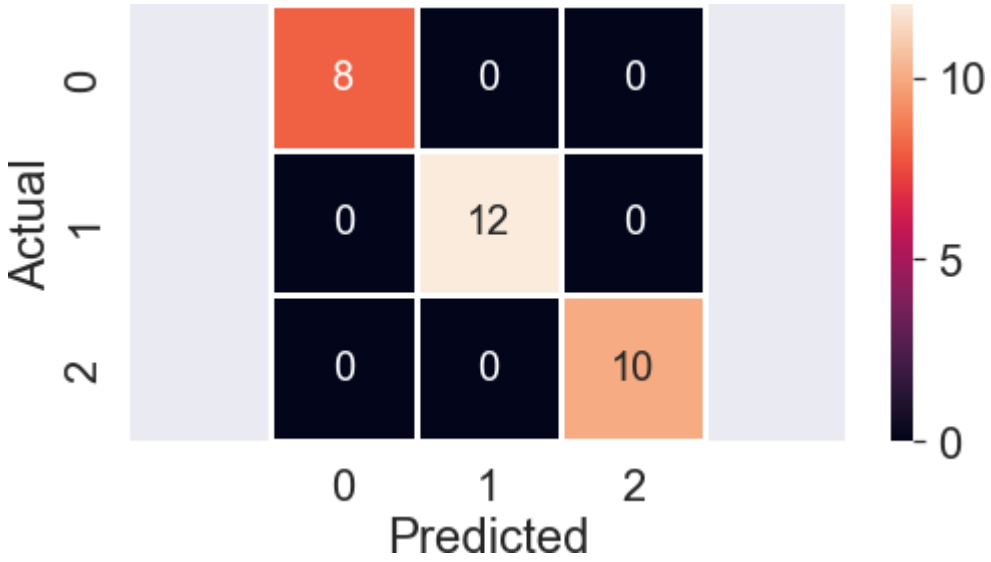
In [15]:
```
print(classification_report(test_classes,y_predict))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         8
Iris-versicolor       1.00      1.00      1.00        12
 Iris-virginica       1.00      1.00      1.00        10

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

In [16]:
```
cf=confusion_matrix(test_classes,y_predict)
cf
```

Out[16]:
```
array([[ 8,  0,  0],
       [ 0, 12,  0],
       [ 0,  0, 10]], dtype=int64)
```

In [20]:
```
with plt.style.context('seaborn'):
 plt.figure(figsize=(8,4))
 sns.set(font_scale=2)
 sns.heatmap(cf,annot=True,square=True,annot_kws={"size":20},linewidth=3)
 plt.xlabel('Predicted')
 plt.ylabel('Actual')
 plt.axis('equal')
 plt.show()
```



In [21]:
```
#Testing
print("Accuracy:  ",metrics.accuracy_score(test_classes,y_predict))
```

```
Accuracy:  1.0
```

In [22]:
```
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier(n_estimators=25)
```

In [23]:
```
model.fit(features,classes)
```

Out[23]:
```
RandomForestClassifier(n_estimators=25)
```

In [24]:
```
from sklearn import metrics
pred=model.predict(test_feat)
print("Accuracy:",metrics.accuracy_score(test_classes,pred))
```

```
Accuracy: 1.0
```

In [ ]:

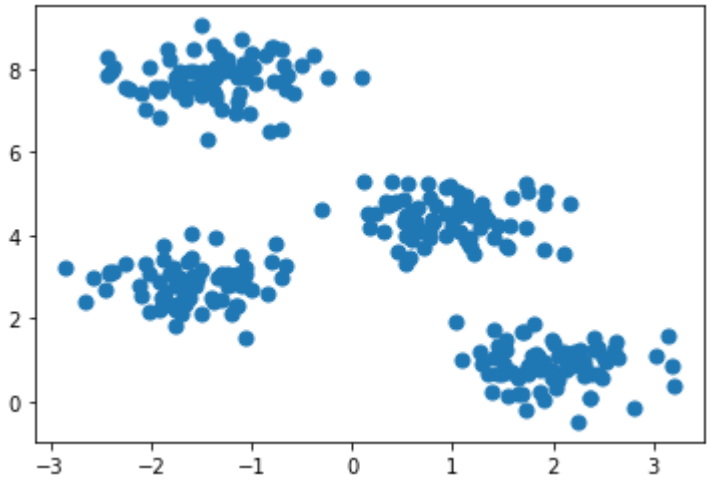
Practical 25 - K Means Clustering - Digit dataset

22 June 2022

In [1]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
from sklearn.datasets import make_blobs #to create data
X,y_true=make_blobs(n_samples=300,centers=4,
                    cluster_std=0.5,random_state=0)
plt.scatter(X[:,0],X[:,1],s=50)
```

Out[2]:
```
<matplotlib.collections.PathCollection at 0x1ce73ab0760>
```



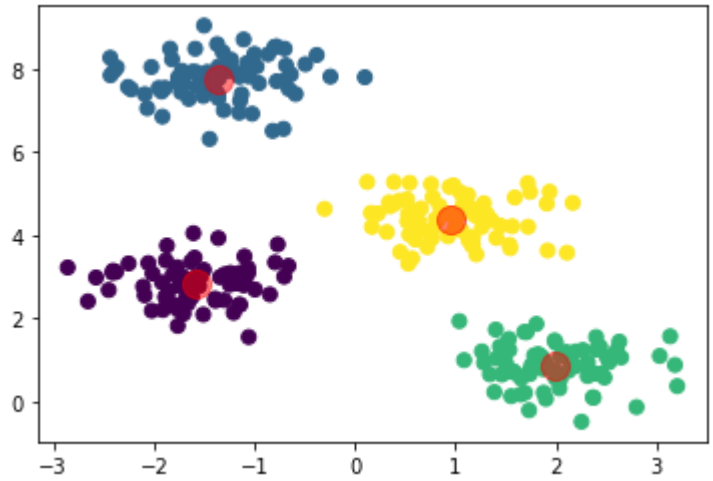
In [3]:
```
kmeans=KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans=kmeans.predict(X)
```

In [4]:
```
y_kmeans
```

Out[4]:
```
array([2, 1, 3, 1, 2, 2, 0, 3, 1, 1, 0, 1, 3, 1, 2, 3, 3, 2, 0, 0, 2, 2,
       3, 0, 0, 3, 2, 3, 0, 3, 1, 1, 3, 1, 1, 1, 1, 1, 0, 2, 3, 0, 3, 3,
       0, 0, 1, 0, 1, 2, 0, 2, 1, 2, 2, 0, 1, 0, 1, 2, 1, 3, 1, 0, 0, 0,
       1, 2, 1, 0, 3, 0, 1, 0, 0, 1, 0, 3, 2, 1, 2, 3, 2, 2, 1, 3, 2, 3,
       1, 1, 3, 2, 1, 0, 0, 3, 2, 2, 3, 0, 1, 2, 1, 2, 3, 2, 2, 3, 1, 3,
       0, 0, 2, 1, 2, 3, 1, 2, 2, 3, 0, 2, 0, 2, 2, 2, 2, 0, 2, 0, 1, 0,
       0, 2, 1, 0, 0, 1, 3, 1, 1, 0, 3, 0, 3, 0, 1, 3, 1, 1, 1, 3, 1, 3,
       2, 0, 1, 0, 2, 3, 1, 3, 3, 2, 3, 0, 0, 3, 2, 3, 3, 1, 2, 3, 0, 1,
       2, 2, 3, 0, 2, 3, 0, 0, 3, 3, 3, 3, 2, 1, 3, 0, 3, 3, 0, 0, 0, 3,
       0, 1, 3, 0, 2, 0, 3, 1, 0, 1, 3, 1, 3, 0, 3, 3, 1, 0, 0, 2, 2, 3,
       1, 2, 2, 0, 2, 0, 3, 1, 1, 3, 3, 1, 3, 2, 0, 3, 2, 0, 1, 0, 2, 3,
       2, 1, 1, 1, 1, 0, 0, 1, 3, 0, 2, 3, 0, 0, 0, 2, 2, 1, 3, 3, 0, 2,
       1, 0, 3, 1, 3, 2, 2, 0, 0, 3, 2, 2, 2, 3, 1, 1, 2, 2, 3, 2, 2, 2,
       1, 0, 1, 3, 2, 2, 1, 1, 1, 2, 2, 3, 1, 0])
```

In [5]:
```
plt.scatter(X[:,0],X[:,1],c=y_kmeans,s=50,cmap='viridis')
centers=kmeans.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='red',s=200,alpha=0.5)
```

Out[5]:
```
<matplotlib.collections.PathCollection at 0x1ce7400f7c0>
```



In [6]:
```
from sklearn.datasets import load_digits
digits=load_digits()
digits.data.shape #64 pixel and 1797 images
```

Out[6]:
```
(1797, 64)
```

In [7]:
```
digits.data
```

Out[7]:
```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

In [8]:
```
kmeans=KMeans(n_clusters=10,random_state=42)
cluster=kmeans.fit_predict(digits.data)
kmeans.cluster_centers_.shape
```

Out[8]:
```
(10, 64)
```

In [9]:
```
cluster
```

Out[9]:
```
array([0, 5, 5, ..., 5, 1, 1])
```

In [10]:
```
kmeans.cluster_centers_
```

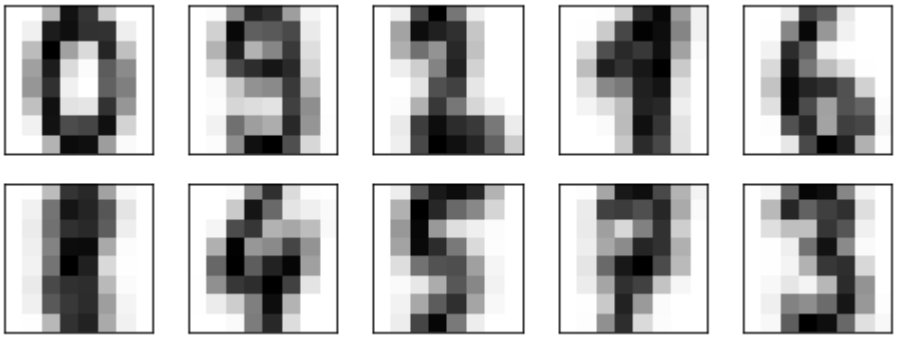
Out[10]:
```
array([[ 0.00000000e+00,  2.23463687e-02,  4.22905028e+00,
         1.31396648e+01,  1.12681564e+01,  2.93854749e+00,
         3.35195531e-02, -5.55111512e-17,   2.60208521e-18,
         8.82681564e-01,  1.26201117e+01,  1.33687151e+01,
         1.14078212e+01,  1.13687151e+01,  9.60893855e-01,
        -1.38777878e-17,  1.30104261e-18,  3.72625698e+00,
         1.42122905e+01,  5.25139665e+00,  2.10614525e+00,
         1.21173184e+01,   3.53072626e+00, -1.38777878e-17,
        -2.16840434e-19,  5.29608939e+00,  1.26424581e+01,
         2.03351955e+00,  2.29050279e-01,  9.07821229e+00,
         6.47486034e+00, -4.33680869e-19,   0.00000000e+00,
         5.88268156e+00,  1.14916201e+01,  8.65921788e-01,
         3.35195531e-02,  8.81005587e+00,  7.15083799e+00,
         0.00000000e+00, -1.73472348e-18,   3.51396648e+00,
         1.32849162e+01,  1.65921788e+00,  1.49162011e+00,
         1.13519553e+01,  5.84357542e+00,  3.46944695e-18,
         8.67361738e-19,  8.04469274e-01,  1.31117318e+01,
         9.96089385e+00,  1.03519553e+01,  1.32960894e+01,
         2.47486034e+00,   2.23463687e-02, -1.08420217e-19,
         5.58659218e-03,  4.19553073e+00,  1.35865922e+01,
         1.33407821e+01,  5.48044693e+00,  3.18435754e-01,
         1.67597765e-02],
       [ 0.00000000e+00,  2.00819672e-01,  6.55737705e+00,
         1.25614754e+01,  1.17950820e+01,  5.55327869e+00,
         5.98360656e-01,  8.19672131e-03,  4.09836066e-03,
         2.63524590e+00,  1.39959016e+01,  9.10655738e+00,
         9.40163934e+00,  1.03524590e+01,  1.21311475e+00,
         4.09836066e-03,  1.30104261e-18,  4.33196721e+00,
         1.27991803e+01,  4.36885246e+00,  6.86475410e+00,
         1.11721311e+01,   1.88524590e+00, -6.93889390e-18,
         2.16840434e-19,  2.30737705e+00,  1.04672131e+01,
         1.19262295e+01,  1.32336066e+01,  1.20860656e+01,
         2.49180328e+00,  4.33680869e-19,  0.00000000e+00,
         3.07377049e-01,  3.22131148e+00,  6.22950820e+00,
         6.68032787e+00,  1.12090164e+01,  4.30737705e+00,
         0.00000000e+00,  1.73472348e-18,  2.21311475e-01,
         2.37295082e+00,  1.93852459e+00,  1.64344262e+00,
         1.08606557e+01,  6.44672131e+00,  1.63934426e-02,
         3.46944695e-18,  7.54098361e-01,  8.11475410e+00,
         5.51639344e+00,  4.65983607e+00,  1.22172131e+01,
         6.02049180e+00,  1.14754098e-01,  1.08420217e-19,
         1.72131148e-01,  6.47950820e+00,  1.35245902e+01,
         1.45163934e+01,  9.98360656e+00,  2.32377049e+00,
         1.14754098e-01],
       [ 0.00000000e+00,  9.42857143e-01,  1.01885714e+01,
         1.44400000e+01,  7.77142857e+00,  9.82857143e-01,
        -6.66133815e-16, -2.77555756e-17,  2.28571429e-02,
         5.24000000e+00,  1.37200000e+01,  1.26228571e+01,
         1.16914286e+01,  3.23428571e+00,  1.71428571e-02,
        -1.38777878e-17,  1.14285714e-02,  4.56000000e+00,
         8.11428571e+00,  6.13714286e+00,  1.21600000e+01,
         3.56000000e+00,   1.71428571e-02, -1.38777878e-17,
        -2.16840434e-19,  9.65714286e-01,  2.81714286e+00,
         7.00571429e+00,  1.25371429e+01,  2.56000000e+00,
         4.00000000e-02, -4.33680869e-19,   0.00000000e+00,
         4.57142857e-02,  1.57142857e+00,  9.89714286e+00,
         1.06971429e+01,  1.45142857e+00,  0.00000000e+00,
         0.00000000e+00, -1.73472348e-18,   2.51428571e-01,
         4.45714286e+00,  1.12457143e+01,  7.74285714e+00,
         2.37142857e+00,  8.45714286e-01,  1.14285714e-02,
         8.67361738e-19,  1.19428571e+00,  1.09942857e+01,
         1.37314286e+01,  1.19257143e+01,  1.11600000e+01,
         7.66857143e+00,   1.10285714e+00, -1.08420217e-19,
         9.31428571e-01,  1.03885714e+01,  1.44685714e+01,
```

```
  1.35028571e+01,  1.23542857e+01,  8.96571429e+00,
  2.95428571e+00],
[ 0.00000000e+00,  1.66533454e-16,  1.53061224e-01,
  2.57142857e+00,  1.11530612e+01,  1.31836735e+01,
  5.31632653e+00,  7.44897959e-01,  1.73472348e-18,
  9.18367347e-02,  2.80612245e+00,  9.80612245e+00,
  1.33775510e+01,  1.28367347e+01,  6.46938776e+00,
  7.55102041e-01,  8.67361738e-19,  1.64285714e+00,
  9.33673469e+00,  1.13367347e+01,  1.05714286e+01,
  1.26428571e+01,  4.97959184e+00,  2.55102041e-01,
 -2.16840434e-19,  3.53061224e+00,  1.19693878e+01,
  1.08367347e+01,  1.23469388e+01,  1.35714286e+01,
  2.82653061e+00, -4.33680869e-19,  0.00000000e+00,
  1.66326531e+00,  6.34693878e+00,  7.07142857e+00,
  1.18367347e+01,  1.24591837e+01,  1.51020408e+00,
  0.00000000e+00, -1.73472348e-18,  6.22448980e-01,
  1.71428571e+00,  3.55102041e+00,  1.17040816e+01,
  1.14285714e+01,  9.38775510e-01,  0.00000000e+00,
  8.67361738e-19,  3.06122449e-02,  2.95918367e-01,
  3.64285714e+00,  1.26428571e+01,  1.06938776e+01,
  1.50000000e+00, -8.32667268e-17, -1.08420217e-19,
  5.55111512e-17,  0.00000000e+00,  2.95918367e+00,
  1.12551020e+01,  9.70408163e+00,  1.55102041e+00,
  0.00000000e+00],
[ 0.00000000e+00,  1.11022302e-16,  1.15934066e+00,
  1.12252747e+01,  9.53296703e+00,  1.41758242e+00,
  5.49450549e-03, -5.55111512e-17,  2.60208521e-18,
  6.04395604e-02,  7.18131868e+00,  1.45604396e+01,
  6.19230769e+00,  8.29670330e-01,  2.74725275e-02,
 -2.77555756e-17,  1.30104261e-18,  7.69230769e-01,
  1.24560440e+01,  9.47252747e+00,  9.34065934e-01,
  1.09890110e-01,  0.00000000e+00, -2.08166817e-17,
 -4.33680869e-19,  2.29670330e+00,  1.36208791e+01,
  8.09340659e+00,  3.87362637e+00,  1.92857143e+00,
  1.04395604e-01, -8.67361738e-19,  0.00000000e+00,
  3.52747253e+00,  1.46758242e+01,  1.29175824e+01,
  1.22527473e+01,  1.02857143e+01,  2.71978022e+00,
  0.00000000e+00, -3.46944695e-18,  1.86813187e+00,
  1.45164835e+01,  1.06538462e+01,  5.57692308e+00,
  1.01923077e+01,  9.13186813e+00,  2.30769231e-01,
  1.73472348e-18,  1.75824176e-01,  1.02857143e+01,
  1.26263736e+01,  5.41758242e+00,  1.13241758e+01,
  1.08956044e+01,  6.26373626e-01, -2.16840434e-19,
 -5.55111512e-17,  1.44505495e+00,  1.07362637e+01,
  1.50989011e+01,  1.31318681e+01,  4.62087912e+00,
  1.70329670e-01],
[ 0.00000000e+00,  1.12107623e-01,  3.91479821e+00,
  1.17668161e+01,  1.24484305e+01,  5.38565022e+00,
  4.34977578e-01,  0.00000000e+00,  8.96860987e-03,
  8.29596413e-01,  8.08520179e+00,  1.35470852e+01,
  1.26816143e+01,  9.82062780e+00,  1.56502242e+00,
 -2.77555756e-17,  1.30104261e-18,  1.18834081e+00,
  8.19730942e+00,  1.19686099e+01,  1.23408072e+01,
  9.37668161e+00,  1.02242152e+00,  0.00000000e+00,
  0.00000000e+00,  9.77578475e-01,  7.20627803e+00,
  1.40941704e+01,  1.41748879e+01,  5.00448430e+00,
  2.06278027e-01,  0.00000000e+00,  0.00000000e+00,
  7.89237668e-01,  8.08071749e+00,  1.48161435e+01,
  1.29013453e+01,  2.25112108e+00,  6.27802691e-02,
  0.00000000e+00,  0.00000000e+00,  1.23318386e+00,
  1.05291480e+01,  1.19865471e+01,  1.21748879e+01,
  4.05381166e+00,  2.69058296e-01,  1.04083409e-17,
  1.34529148e-02,  8.78923767e-01,  9.58744395e+00,
  1.14932735e+01,  1.22152466e+01,  5.66816143e+00,
```

```
  6.81614350e-01,  4.48430493e-03,  4.48430493e-03,
  1.12107623e-01,  4.11659193e+00,  1.19013453e+01,
  1.27533632e+01,  4.97309417e+00,  8.56502242e-01,
  8.96860987e-03],
[ 0.00000000e+00,  1.11022302e-16,  2.60355030e-01,
  6.85798817e+00,  1.20177515e+01,  2.08875740e+00,
  1.47928994e-01,  5.32544379e-02,  2.60208521e-18,
  1.18343195e-02,  3.10059172e+00,  1.35621302e+01,
  8.72781065e+00,  1.56804734e+00,  9.52662722e-01,
  3.13609467e-01,  1.30104261e-18,  5.97633136e-01,
  1.03905325e+01,  1.16804734e+01,  4.46153846e+00,
  5.13609467e+00,  3.84615385e+00,  3.49112426e-01,
  5.91715976e-03,  4.59171598e+00,  1.45976331e+01,
  6.07100592e+00,  6.79289941e+00,  1.07810651e+01,
  6.25443787e+00,  1.77514793e-02,  0.00000000e+00,
  8.80473373e+00,  1.48284024e+01,  9.40236686e+00,
  1.27692308e+01,  1.44674556e+01,  5.52662722e+00,
  0.00000000e+00,  9.46745562e-02,  6.47928994e+00,
  1.15976331e+01,  1.22189349e+01,  1.47751479e+01,
  1.09822485e+01,  1.62130178e+00,  0.00000000e+00,
  5.91715976e-02,  1.11242604e+00,  2.91124260e+00,
  7.44970414e+00,  1.40295858e+01,  4.43195266e+00,
  1.77514793e-02, -8.32667268e-17, -1.08420217e-19,
  2.36686391e-02,  3.13609467e-01,  7.64497041e+00,
  1.24023669e+01,   1.98816568e+00, -8.88178420e-16,
  5.55111512e-17],
[ 0.00000000e+00,  1.10810811e+00,  1.00135135e+01,
  1.33986486e+01,  1.41891892e+01,  1.26081081e+01,
  4.40540541e+00,  4.05405405e-02,  6.75675676e-03,
  4.56081081e+00,  1.49391892e+01,  1.26418919e+01,
  8.72297297e+00,  7.00000000e+00,  2.47972973e+00,
  3.37837838e-02,  1.35135135e-02,  6.06756757e+00,
  1.45270270e+01,  5.99324324e+00,  1.97972973e+00,
  9.39189189e-01,   1.75675676e-01, -1.38777878e-17,
  6.75675676e-03,  5.31756757e+00,  1.43310811e+01,
  1.23648649e+01,  7.81081081e+00,  2.20945946e+00,
  1.48648649e-01, -4.33680869e-19,   0.00000000e+00,
  1.95945946e+00,  8.18918919e+00,  1.00608108e+01,
  1.03310811e+01,  5.54729730e+00,  6.41891892e-01,
  0.00000000e+00, -1.73472348e-18,   3.04054054e-01,
  1.40540541e+00,  4.88513514e+00,  9.85135135e+00,
  7.06081081e+00,   7.83783784e-01, -3.46944695e-18,
  8.67361738e-19,  8.10810811e-01,  5.09459459e+00,
  9.54054054e+00,  1.21418919e+01,  5.27027027e+00,
  4.45945946e-01, -1.11022302e-16, -1.08420217e-19,
  1.06081081e+00,  1.08918919e+01,  1.45540541e+01,
  7.80405405e+00,  1.06756757e+00,  2.02702703e-02,
  0.00000000e+00],
[ 0.00000000e+00,  1.66666667e-01,  5.15151515e+00,
  1.33080808e+01,  1.39444444e+01,  1.05202020e+01,
  4.44444444e+00,  7.22222222e-01,  1.73472348e-18,
  1.18181818e+00,  1.09646465e+01,  1.13181818e+01,
  1.02121212e+01,  1.25555556e+01,  4.93434343e+00,
  2.97979798e-01,  8.67361738e-19,  1.24747475e+00,
  5.60606061e+00,  2.29292929e+00,  7.29292929e+00,
  1.18737374e+01,  2.84848485e+00,  3.03030303e-02,
 -4.33680869e-19,  1.02020202e+00,  5.07070707e+00,
  6.75757576e+00,  1.26060606e+01,  1.19090909e+01,
  4.59595960e+00,  5.05050505e-03,  0.00000000e+00,
  1.50505051e+00,  8.71717172e+00,  1.33737374e+01,
  1.48030303e+01,  1.04595960e+01,  4.01515152e+00,
  0.00000000e+00, -3.46944695e-18,   1.11111111e+00,
  5.22727273e+00,  1.20151515e+01,  1.09343434e+01,
  3.43434343e+00,  5.55555556e-01,  6.93889390e-18,
```

```
  1.73472348e-18,  1.06060606e-01,  3.15656566e+00,
  1.27525253e+01,  5.85353535e+00,  2.82828283e-01,
 -3.10862447e-15, -1.11022302e-16, -2.16840434e-19,
  1.31313131e-01,  6.46464646e+00,  1.21969697e+01,
  2.18686869e+00,  1.86868687e-01,  1.01010101e-02,
  5.55111512e-17],
[ 0.00000000e+00,  5.85635359e-01,  8.66298343e+00,
  1.45248619e+01,  1.40331492e+01,  7.09392265e+00,
  6.62983425e-01, -5.55111512e-17,  1.10497238e-02,
  4.13812155e+00,  1.26574586e+01,  9.20994475e+00,
  1.12486188e+01,  1.20386740e+01,  1.95580110e+00,
  1.10497238e-02,  5.52486188e-03,  1.88397790e+00,
  3.77348066e+00,  3.68508287e+00,  1.17458564e+01,
  9.95580110e+00,  9.00552486e-01, -6.93889390e-18,
 -2.16840434e-19,  6.07734807e-02,  9.72375691e-01,
  8.17679558e+00,  1.38066298e+01,  6.87292818e+00,
  3.31491713e-01, -4.33680869e-19,  0.00000000e+00,
  6.07734807e-02,  6.68508287e-01,  4.53591160e+00,
  1.17071823e+01,  1.22375691e+01,  2.29834254e+00,
  0.00000000e+00, -1.73472348e-18,  4.58563536e-01,
  1.48066298e+00,  6.85082873e-01,  4.23756906e+00,
  1.23977901e+01,  6.24309392e+00,  5.52486188e-03,
  1.73472348e-18,  9.33701657e-01,  7.34254144e+00,
  6.60773481e+00,  8.66298343e+00,  1.36298343e+01,
  6.01657459e+00,  1.71270718e-01, -1.08420217e-19,
  4.64088398e-01,  9.45303867e+00,  1.49226519e+01,
  1.40828729e+01,  8.81215470e+00,  1.84530387e+00,
  4.08839779e-01]])
```

In [11]:
```
fig,ax=plt.subplots(2,5,figsize=(8,3))
centers=kmeans.cluster_centers_.reshape(10,8,8)
for axi,center in zip(ax.flat,centers):
    axi.set(xticks=[],yticks=[])
    axi.imshow(center,interpolation='nearest',cmap=plt.cm.binary)
```



In [12]:
```
import seaborn as sns;sns.set()#for plot styling
from sklearn.metrics import confusion_matrix
mat=confusion_matrix(digits.target,cluster)
mat
```

Out[12]:
```
array([[177,   0,   0,   0,   0,   0,   1,   0,   0,   0],
       [  0,   0,  24,  54,   2, 100,   0,   1,   0,   1],
       [  1,   2, 148,   3,   0,   8,   0,   0,   2,  13],
       [  0,  12,   0,   0,   0,   7,   0,   2,   6, 156],
       [  0,   0,   0,   2,   0,   2, 166,   0,  11,   0],
       [  0,  41,   0,   0,   1,   0,   2, 136,   0,   2],
       [  1,   0,   0,   0, 177,   3,   0,   0,   0,   0],
       [  0,   0,   0,  10,   0,   2,   0,   0, 167,   0],
       [  0,  50,   3,   9,   2, 100,   0,   4,   4,   2],
       [  0, 139,   0,  20,   0,   1,   0,   5,   8,   7]], dtype=int64)
```

Using Clustering on iris dataset

In [13]:
```
df=pd.read_csv('D:\\ML\\iris.csv')
df
```

Out[13]:

	Sepal length	Sepal width	Petal length	Petal width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [14]:
```
X=df[['Sepal length','Sepal width','Petal length','Petal width']].values
```

In [15]:
```
kmeans=KMeans(n_clusters=3)
kmeans.fit(X)
y_kmeans=kmeans.predict(X)
```

In [16]:
```
print(y_kmeans)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0 0 0 2 0 0
 0 2]
```

In [17]:
```
kmeans.cluster_centers_
```

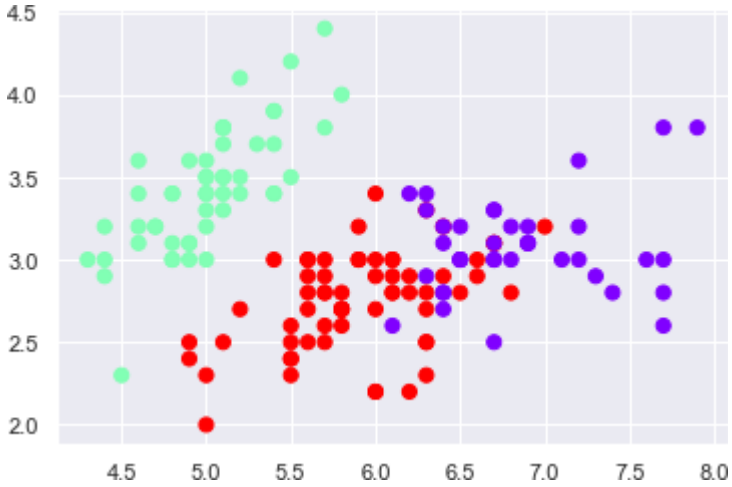
Out[17]:
```
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097]])
```

In [18]:
```
n_cluster=3
silhouette_avg=silhouette_score(X,y_kmeans)
print("For n_cluster= ",n_cluster,"average silhouette_score is :",silhouette_avg)
```

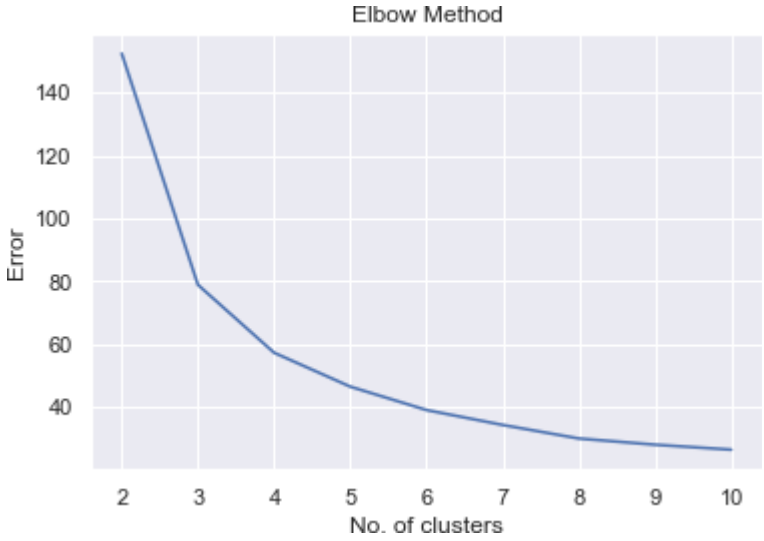
```
For n_cluster=  3 average silhouette_score is : 0.5528190123564102
```

In [19]:
```
plt.scatter(X[:,0],X[:,1],c=y_kmeans,s=50,cmap='rainbow')
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x1ce75bf0dc0>
```



In [20]:
```
Error=[]                              #for selecting no. of clusters
for i in range(2,11):
    kmeans=KMeans(n_clusters=i).fit(X)
    Error.append(kmeans.inertia_)
import matplotlib.pyplot as plt
plt.plot(range(2,11),Error)
plt.title('Elbow Method')
plt.xlabel('No. of clusters')
plt.ylabel('Error')
plt.show()
```



Practical 26 - K Means Clustering - Mall dataset

22 June 2022

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
df=pd.read_csv("D:\\ML\\Mall_Customers.csv")
```

In [3]:
```
df
```

Out[3]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype 
---  ------                  --------------  ----- 
 0   CustomerID              200 non-null    int64 
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64 
 3   Annual Income (k$)      200 non-null    int64 
 4   Spending Score (1-100)  200 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

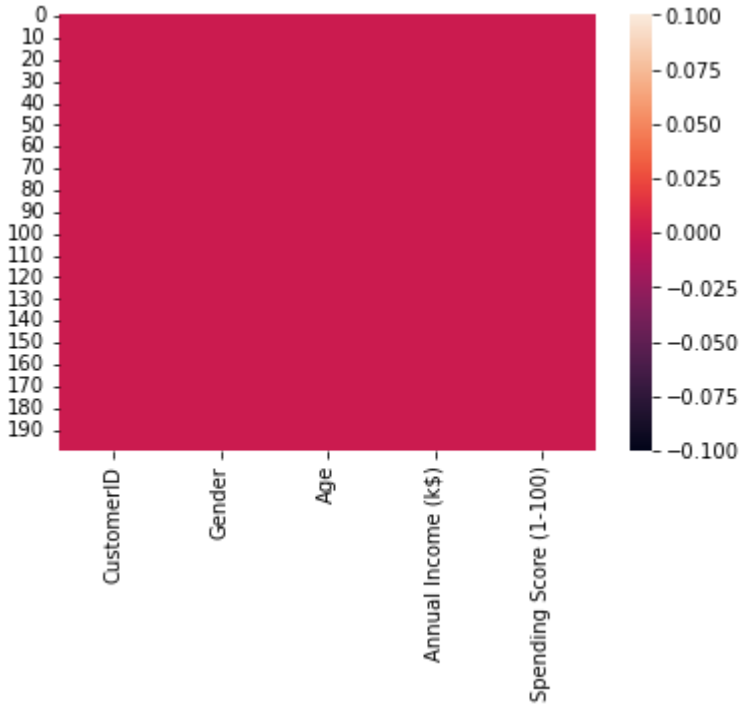
In [5]:
```
df.describe()
```

Out[5]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

In [6]:
```
sns.heatmap(df.isnull())
```

Out[6]:
```
<AxesSubplot:>
```



we need to convert Gender into int

In [7]:
```
df["Gender"].dtypes
```

Out[7]:
```
dtype('O')
```

In [8]:
```
df['Gender'] = df['Gender'].map({'Male':1,'Female':0})
```

In [9]:
```
df
```

Out[9]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15	39
1	2	1	21	15	81
2	3	0	20	16	6
3	4	0	23	16	77
4	5	0	31	17	40
...	...	...	...	...	...
195	196	0	35	120	79
196	197	0	45	126	28
197	198	1	32	126	74
198	199	1	32	137	18
199	200	1	30	137	83

200 rows × 5 columns

In [10]:
```
X=df[['Gender','Age','Annual Income (k$)','Spending Score (1-100)']].values
```

In [11]:
```
X
```

Out[11]:
```
array([[  1,  19,  15,  39],
       [  1,  21,  15,  81],
       [  0,  20,  16,   6],
       [  0,  23,  16,  77],
       [  0,  31,  17,  40],
       [  0,  22,  17,  76],
       [  0,  35,  18,   6],
       [  0,  23,  18,  94],
       [  1,  64,  19,   3],
       [  0,  30,  19,  72],
       [  1,  67,  19,  14],
       [  0,  35,  19,  99],
       [  0,  58,  20,  15],
       [  0,  24,  20,  77],
       [  1,  37,  20,  13],
       [  1,  22,  20,  79],
       [  0,  35,  21,  35],
       [  1,  20,  21,  66],
       [  1,  52,  23,  29],
       [  0,  35,  23,  98],
       [  1,  35,  24,  35],
       [  1,  25,  24,  73],
       [  0,  46,  25,   5],
       [  1,  31,  25,  73],
       [  0,  54,  28,  14],
       [  1,  29,  28,  82],
       [  0,  45,  28,  32],
       [  1,  35,  28,  61],
       [  0,  40,  29,  31],
       [  0,  23,  29,  87],
       [  1,  60,  30,   4],
       [  0,  21,  30,  73],
       [  1,  53,  33,   4],
       [  1,  18,  33,  92],
       [  0,  49,  33,  14],
       [  0,  21,  33,  81],
       [  0,  42,  34,  17],
       [  0,  30,  34,  73],
       [  0,  36,  37,  26],
       [  0,  20,  37,  75],
       [  0,  65,  38,  35],
       [  1,  24,  38,  92],
       [  1,  48,  39,  36],
       [  0,  31,  39,  61],
       [  0,  49,  39,  28],
       [  0,  24,  39,  65],
       [  0,  50,  40,  55],
       [  0,  27,  40,  47],
       [  0,  29,  40,  42],
       [  0,  31,  40,  42],
       [  0,  49,  42,  52],
       [  1,  33,  42,  60],
       [  0,  31,  43,  54],
       [  1,  59,  43,  60],
       [  0,  50,  43,  45],
       [  1,  47,  43,  41],
       [  0,  51,  44,  50],
       [  1,  69,  44,  46],
       [  0,  27,  46,  51],
       [  1,  53,  46,  46],
       [  1,  70,  46,  56],
       [  1,  19,  46,  55],
       [  0,  67,  47,  52],
       [  0,  54,  47,  59],
```

```
[  1,  63,  48,  51],
[  1,  18,  48,  59],
[  0,  43,  48,  50],
[  0,  68,  48,  48],
[  1,  19,  48,  59],
[  0,  32,  48,  47],
[  1,  70,  49,  55],
[  0,  47,  49,  42],
[  0,  60,  50,  49],
[  0,  60,  50,  56],
[  1,  59,  54,  47],
[  1,  26,  54,  54],
[  0,  45,  54,  53],
[  1,  40,  54,  48],
[  0,  23,  54,  52],
[  0,  49,  54,  42],
[  1,  57,  54,  51],
[  1,  38,  54,  55],
[  1,  67,  54,  41],
[  0,  46,  54,  44],
[  0,  21,  54,  57],
[  1,  48,  54,  46],
[  0,  55,  57,  58],
[  0,  22,  57,  55],
[  0,  34,  58,  60],
[  0,  50,  58,  46],
[  0,  68,  59,  55],
[  1,  18,  59,  41],
[  1,  48,  60,  49],
[  0,  40,  60,  40],
[  0,  32,  60,  42],
[  1,  24,  60,  52],
[  0,  47,  60,  47],
[  0,  27,  60,  50],
[  1,  48,  61,  42],
[  1,  20,  61,  49],
[  0,  23,  62,  41],
[  0,  49,  62,  48],
[  1,  67,  62,  59],
[  1,  26,  62,  55],
[  1,  49,  62,  56],
[  0,  21,  62,  42],
[  0,  66,  63,  50],
[  1,  54,  63,  46],
[  1,  68,  63,  43],
[  1,  66,  63,  48],
[  1,  65,  63,  52],
[  0,  19,  63,  54],
[  0,  38,  64,  42],
[  1,  19,  64,  46],
[  0,  18,  65,  48],
[  0,  19,  65,  50],
[  0,  63,  65,  43],
[  0,  49,  65,  59],
[  0,  51,  67,  43],
[  0,  50,  67,  57],
[  1,  27,  67,  56],
[  0,  38,  67,  40],
[  0,  40,  69,  58],
[  1,  39,  69,  91],
[  0,  23,  70,  29],
[  0,  31,  70,  77],
[  1,  43,  71,  35],
[  1,  40,  71,  95],
```

```
[  1,  59,  71,  11],
[  1,  38,  71,  75],
[  1,  47,  71,   9],
[  1,  39,  71,  75],
[  0,  25,  72,  34],
[  0,  31,  72,  71],
[  1,  20,  73,   5],
[  0,  29,  73,  88],
[  0,  44,  73,   7],
[  1,  32,  73,  73],
[  1,  19,  74,  10],
[  0,  35,  74,  72],
[  0,  57,  75,   5],
[  1,  32,  75,  93],
[  0,  28,  76,  40],
[  0,  32,  76,  87],
[  1,  25,  77,  12],
[  1,  28,  77,  97],
[  1,  48,  77,  36],
[  0,  32,  77,  74],
[  0,  34,  78,  22],
[  1,  34,  78,  90],
[  1,  43,  78,  17],
[  1,  39,  78,  88],
[  0,  44,  78,  20],
[  0,  38,  78,  76],
[  0,  47,  78,  16],
[  0,  27,  78,  89],
[  1,  37,  78,   1],
[  0,  30,  78,  78],
[  1,  34,  78,   1],
[  0,  30,  78,  73],
[  0,  56,  79,  35],
[  0,  29,  79,  83],
[  1,  19,  81,   5],
[  0,  31,  81,  93],
[  1,  50,  85,  26],
[  0,  36,  85,  75],
[  1,  42,  86,  20],
[  0,  33,  86,  95],
[  0,  36,  87,  27],
[  1,  32,  87,  63],
[  1,  40,  87,  13],
[  1,  28,  87,  75],
[  1,  36,  87,  10],
[  1,  36,  87,  92],
[  0,  52,  88,  13],
[  0,  30,  88,  86],
[  1,  58,  88,  15],
[  1,  27,  88,  69],
[  1,  59,  93,  14],
[  1,  35,  93,  90],
[  0,  37,  97,  32],
[  0,  32,  97,  86],
[  1,  46,  98,  15],
[  0,  29,  98,  88],
[  0,  41,  99,  39],
[  1,  30,  99,  97],
[  0,  54, 101,  24],
[  1,  28, 101,  68],
[  0,  41, 103,  17],
[  0,  36, 103,  85],
[  0,  34, 103,  23],
[  0,  32, 103,  69],
```

```
       [  1,  33, 113,   8],
       [  0,  38, 113,  91],
       [  0,  47, 120,  16],
       [  0,  35, 120,  79],
       [  0,  45, 126,  28],
       [  1,  32, 126,  74],
       [  1,  32, 137,  18],
       [  1,  30, 137,  83]], dtype=int64)
```

In [12]:
```
kmeans=KMeans(n_clusters=2)
kmeans.fit(X)
y_kmeans=kmeans.predict(X)
```

In [13]:
```
print(y_kmeans)
```

```
[1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
 0 1 0 1 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1
 1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1
 0 1 0 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0]
```

In [14]:
```
kmeans.cluster_centers_
```

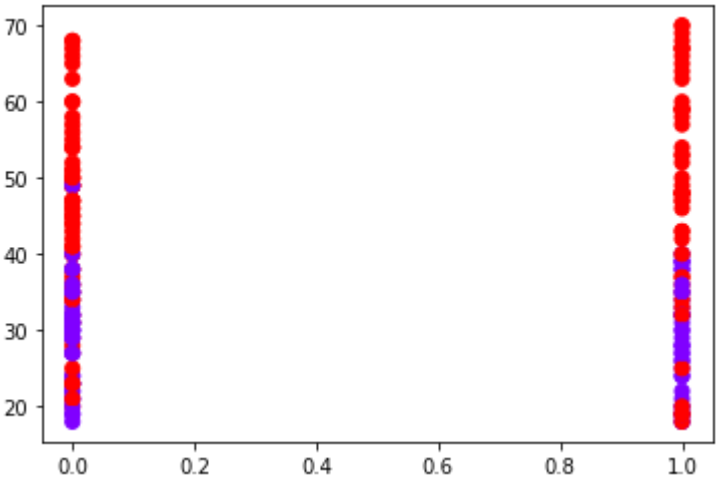
Out[14]:
```
array([[ 0.44705882, 28.95294118, 62.17647059, 73.62352941],
       [ 0.43478261, 46.16521739, 59.36521739, 32.88695652]])
```

In [15]:
```
n_cluster=6
silhouette_avg=silhouette_score(X,y_kmeans)
print("For n_cluster= ",n_cluster,"average silhouette_score is :",silhouette_avg)
```

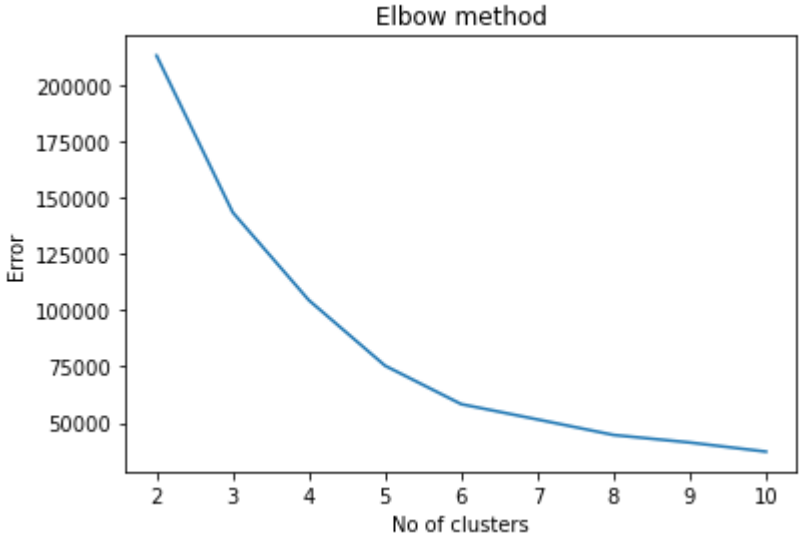
```
For n_cluster=  6 average silhouette_score is : 0.29307334005502633
```

In [16]:
```
plt.scatter(X[:,0],X[:,1],c=y_kmeans,s=50,cmap='rainbow')
```

Out[16]:
```
<matplotlib.collections.PathCollection at 0x215e6aeffd0>
```



In [17]:
```
Error =[]
for i in range (2,11):
     kmeans = KMeans(n_clusters = i).fit(X)
#     kmeans.fit(x)
     Error.append(kmeans.inertia_)

plt.plot(range(2, 11), Error)
plt.title( 'Elbow method')
plt.xlabel('No of clusters')
plt.ylabel('Error')
plt.show()
```



6 clusters

In [ ]:

In [ ]:

Practical 27 - Feature Selection - Univariate

22 June 2022

In [1]:
```
from sklearn. datasets import load_breast_cancer
from sklearn. feature_selection import SelectPercentile, SelectKBest
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
cancer=load_breast_cancer()
```

In [3]:
```
print(cancer.data.shape)
```

```
(569, 30)
```

In [4]:
```
# get deterministic random numbers
rng=np.random.RandomState(42)
noise=rng.normal(size=(len(cancer.data),50))
```

In [5]:
```
print(noise.data.shape)
```

```
(569, 50)
```

In [6]:
```
#using horizontal stack we will add these data
# add noise features to the data
# the first 30 features are from the dataset, the next 50 are noise
X_w_noise=np.hstack([cancer.data,  noise])
```

In [7]:
```
print(X_w_noise.data.shape)
```

```
(569, 80)
```

In [8]:
```
X_train,X_test,y_train,y_test=train_test_split(X_w_noise,  cancer.target,  random_st
```

In [13]:
```
#select = SelectPercentile(percentile=30) # use this line for one time and see acc
select=SelectKBest(k=10)
select.fit(X_train,  y_train)
```

Out[13]:
```
SelectKBest()
```

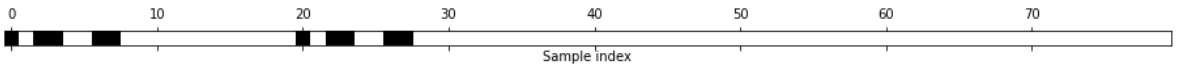
In [10]:
```
#transform training set
X_train_selected  =  select.transform(X_train)
print ("X_train.shape{} ".format (X_train.shape))
print("X_train_selected.  shape{}".format(X_train_selected.shape)  )
```

```
X_train.shape(398,  80)
X_train_selected.  shape(398,  10)
```

In [11]:
```
#now we will see that how many feature this algo choose
mask  =  select.get_support()
print(mask)
# visualize the mask -- black is True, white is False
plt.matshow(mask.  reshape(1,  -1),  cmap='gray_r')
plt.xlabel(("Sample index"))
plt.yticks(())
```

```
[ True False True True False False True True False False False False
 False False False False False False False False True False True True
 False False True True False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False]
```

Out[11]: ([], [])



In [12]:
```
from sklearn. tree import DecisionTreeClassifier
# transform test data
X_test_selected = select. transform(X_test)
lr = DecisionTreeClassifier()
lr.fit(X_train, y_train)
print("Score with all features: {:.3f}" .format (lr.score(X_test, y_test)))
lr.fit(X_train_selected, y_train)
print("Score with only selected features: {:.3f}".format(
lr.score(X_test_selected, y_test)))
```

```
Score with all features: 0.924
Score with only selected features: 0.947
```

feature selection - Model Based

In [14]:
```
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
select=SelectFromModel(RandomForestClassifier(n_estimators=10,random_state=42),thr
```

Select from model

class selects all the features that have an impoertance measure of the features (as provided by the supervised model) greater than the provided threshold

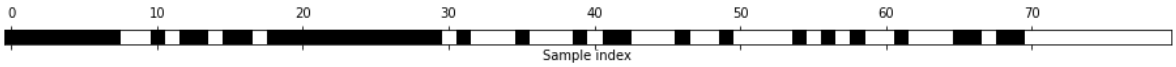
In [15]:
```
select.fit(X_train,y_train)
X_train_l1 = select.transform(X_train)
print("X_train.shape{}".format(X_train.shape))
print("X_train_l1.shape{}".format(X_train_l1.shape))
```

```
X_train.shape(398, 80)
X_train_l1.shape(398, 40)
```

In [16]:
```
#now we will see that how many feature this algo choose
mask = select.get_support()
print(mask)
# visualize the mask -- black is True, white is False
plt.matshow(mask. reshape(1, -1), cmap='gray_r')
plt.xlabel(("Sample index"))
plt.yticks(())
```

```
[ True  True  True  True  True  True  True  True False False  True False
  True  True False  True  True False  True  True  True  True  True  True
  True  True  True  True  True  True False  True False False False  True
  False False False True  False True True  False False False True  False
  False True  False False False False True  False True  False True  False
  False True  False False False True  True  False True  True  False False
  False False False False False False False False]
```

Out[16]:
```
([], [])
```



In [17]:
```
X_test_l1 = select.transform(X_test)
score=SVC().fit(X_train,y_train).score(X_test,y_test)
print("Test Score:{:.3f}".format(score))
score=SVC().fit(X_train_l1,y_train).score(X_test_l1,y_test)
print("Test  Score:{:.3f}".format(score))
```

```
Test Score:0.924
Test Score:0.924
```

Iterative Feature Selection

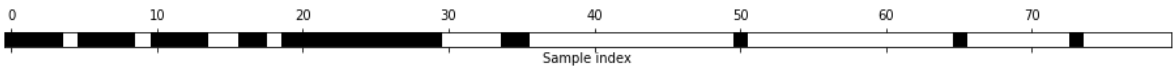
In [18]:
```
from sklearn.feature_selection import RFE
select=RFE(RandomForestClassifier(n_estimators=100,random_state=42),n_features_to_
select.fit(X_train,y_train)
```

Out[18]:
```
RFE(estimator=RandomForestClassifier(random_state=42),   n_features_to_select=30)
```

In [19]:
```
#now we will see that how many feature this algo choose
mask = select.get_support()
print(mask)
# visualize the mask -- black is True, white is False
plt.matshow(mask. reshape(1, -1), cmap='gray_r')
plt.xlabel(("Sample index"))
plt.yticks(())
```

```
[ True  True  True  True False  True  True  True  True False  True  True
  True  True  True False False  True  True False  True  True  True  True
  True  True  True  True  True  True False False False False  True  True
  False False False False False False False False False False False False
  False False  True False False False False False False False False False
  False False False False False  True False False False False False False
  False  True False False False False False False]
```

Out[19]:
```
([], [])
```



In [ ]:
```
from sklearn.feature_selection import LogisticRegression
X_train_rfe = select.transform(X_train)
X_test_rfe = select.transform(X_test)
score=LogisticRegression
```

Practical 28 - Principle Component Analysis

22 June 2022

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import decomposition
from sklearn import datasets
from  sklearn.tree  import  DecisionTreeClassifier
```

In [2]:
```
iris=datasets.load_iris()
X=iris.data
Y=iris.target
```

In [3]:
```
pca  =  decomposition.PCA(n_components=3)
pca.fit(X)
X1=pca.transform(X)
```

In [4]:
```
print(iris.data.shape)
print(X.shape)
print(X1.shape)
```

```
(150, 4)
(150, 4)
(150, 3)
```

In [5]:
```
X1
```

Out[5]:
```
array([[-2.68412563,  0.31939725, -0.02791483],
       [-2.71414169, -0.17700123, -0.21046427],
       [-2.88899057, -0.14494943,  0.01790026],
       [-2.74534286, -0.31829898,  0.03155937],
       [-2.72871654,  0.32675451,  0.09007924],
       [-2.28085963,  0.74133045,  0.16867766],
       [-2.82053775, -0.08946138,  0.25789216],
       [-2.62614497,  0.16338496, -0.02187932],
       [-2.88638273, -0.57831175,  0.02075957],
       [-2.6727558 , -0.11377425, -0.19763272],
       [-2.50694709,  0.6450689 , -0.07531801],
       [-2.61275523,  0.01472994,  0.10215026],
       [-2.78610927, -0.235112  , -0.20684443],
       [-3.22380374, -0.51139459,  0.06129967],
       [-2.64475039,  1.17876464, -0.15162752],
       [-2.38603903,  1.33806233,  0.2777769 ],
       [-2.62352788,  0.81067951,  0.13818323],
       [-2.64829671,  0.31184914,  0.02666832],
       [-2.19982032,  0.87283904, -0.12030552],
       [-2.5879864 ,  0.51356031,  0.21366517],
       [-2.31025622,  0.39134594, -0.23944404],
       [-2.54370523,  0.43299606,  0.20845723],
       [-3.21593942,  0.13346807,  0.29239675],
       [-2.30273318,  0.09870885,  0.03912326],
       [-2.35575405, -0.03728186,  0.12502108],
       [-2.50666891, -0.14601688, -0.25342004],
       [-2.46882007,  0.13095149,  0.09491058],
       [-2.56231991,  0.36771886, -0.07849421],
       [-2.63953472,  0.31203998, -0.1459089 ],
       [-2.63198939, -0.19696122,  0.04077108],
       [-2.58739848, -0.20431849, -0.07722299],
       [-2.4099325 ,  0.41092426, -0.14552497],
       [-2.64886233,  0.81336382,  0.22566915],
       [-2.59873675,  1.09314576,  0.15781081],
       [-2.63692688, -0.12132235, -0.14304958],
       [-2.86624165,  0.06936447, -0.16433231],
       [-2.62523805,  0.59937002, -0.26835038],
       [-2.80068412,  0.26864374,  0.09369908],
       [-2.98050204, -0.48795834,  0.07292705],
       [-2.59000631,  0.22904384, -0.0800823 ],
       [-2.77010243,  0.26352753,  0.07724769],
       [-2.84936871, -0.94096057, -0.34923038],
       [-2.99740655, -0.34192606,  0.19250921],
       [-2.40561449,  0.18887143,  0.26386795],
       [-2.20948924,  0.43666314,  0.29874275],
       [-2.71445143, -0.2502082 , -0.09767814],
       [-2.53814826,  0.50377114,  0.16670564],
       [-2.83946217, -0.22794557,  0.08372685],
       [-2.54308575,  0.57941002, -0.01711502],
       [-2.70335978,  0.10770608, -0.08929401],
       [ 1.28482569,  0.68516047, -0.40656803],
       [ 0.93248853,  0.31833364, -0.01801419],
       [ 1.46430232,  0.50426282, -0.33832576],
       [ 0.18331772, -0.82795901, -0.17959139],
       [ 1.08810326,  0.07459068, -0.3077579 ],
       [ 0.64166908, -0.41824687,  0.04107609],
       [ 1.09506066,  0.28346827,  0.16981024],
       [-0.74912267, -1.00489096,  0.01230292],
       [ 1.04413183,  0.2283619 , -0.41533608],
       [-0.0087454 , -0.72308191,  0.28114143],
       [-0.50784088, -1.26597119, -0.26981718],
       [ 0.51169856, -0.10398124,  0.13054775],
       [ 0.26497651, -0.55003646, -0.69414683],
       [ 0.98493451, -0.12481785, -0.06211441],
```

```
[-0.17392537, -0.25485421,  0.09045769],
 [ 0.92786078,  0.46717949, -0.31462098],
 [ 0.66028376, -0.35296967,  0.32802753],
 [ 0.23610499, -0.33361077, -0.27116184],
 [ 0.94473373, -0.54314555, -0.49951905],
 [ 0.04522698, -0.58383438, -0.2350021 ],
 [ 1.11628318, -0.08461685,  0.45962099],
 [ 0.35788842, -0.06892503, -0.22985389],
 [ 1.29818388, -0.32778731, -0.34785435],
 [ 0.92172892, -0.18273779, -0.23107178],
 [ 0.71485333,  0.14905594, -0.32180094],
 [ 0.90017437,  0.32850447, -0.31620907],
 [ 1.33202444,  0.24444088, -0.52170278],
 [ 1.55780216,  0.26749545, -0.16492098],
 [ 0.81329065, -0.1633503 ,  0.0354245 ],
[-0.30558378, -0.36826219, -0.31849158],
[-0.06812649, -0.70517213, -0.24421381],
[-0.18962247, -0.68028676, -0.30642056],
 [ 0.13642871, -0.31403244, -0.17724277],
 [ 1.38002644, -0.42095429,  0.01616713],
 [ 0.58800644, -0.48428742,  0.4444335 ],
 [ 0.80685831,  0.19418231,  0.38896306],
 [ 1.22069088,  0.40761959, -0.23716701],
 [ 0.81509524, -0.37203706, -0.61472084],
 [ 0.24595768, -0.2685244 ,  0.18836681],
 [ 0.16641322, -0.68192672, -0.06000923],
 [ 0.46480029, -0.67071154, -0.02430686],
 [ 0.8908152 , -0.03446444, -0.00994693],
 [ 0.23054802, -0.40438585, -0.22941024],
[-0.70453176, -1.01224823, -0.10569115],
 [ 0.35698149, -0.50491009,  0.01661717],
 [ 0.33193448, -0.21265468,  0.08320429],
 [ 0.37621565, -0.29321893,  0.07799635],
 [ 0.64257601,  0.01773819, -0.20539497],
[-0.90646986, -0.75609337, -0.01259965],
 [ 0.29900084, -0.34889781,  0.01058166],
 [ 2.53119273, -0.00984911,  0.76016543],
 [ 1.41523588, -0.57491635,  0.29632253],
 [ 2.61667602,  0.34390315, -0.11078788],
 [ 1.97153105, -0.1797279 ,  0.10842466],
 [ 2.35000592, -0.04026095,  0.28538956],
 [ 3.39703874,  0.55083667, -0.34843756],
 [ 0.52123224, -1.19275873,  0.5456593 ],
 [ 2.93258707,  0.3555    , -0.42023994],
 [ 2.32122882, -0.2438315 , -0.34830439],
 [ 2.91675097,  0.78279195,  0.42333542],
 [ 1.66177415,  0.24222841,  0.24244019],
 [ 1.80340195, -0.21563762, -0.03764817],
 [ 2.1655918 ,  0.21627559,  0.03332664],
 [ 1.34616358, -0.77681835,  0.28190288],
 [ 1.58592822, -0.53964071,  0.62902933],
 [ 1.90445637,  0.11925069,  0.47963982],
 [ 1.94968906,  0.04194326,  0.04418617],
 [ 3.48705536,  1.17573933,  0.13389487],
 [ 3.79564542,  0.25732297, -0.51376776],
 [ 1.30079171, -0.76114964, -0.34499504],
 [ 2.42781791,  0.37819601,  0.21911932],
 [ 1.19900111, -0.60609153,  0.51185551],
 [ 3.49992004,  0.4606741 , -0.57318224],
 [ 1.38876613, -0.20439933, -0.06452276],
 [ 2.2754305 ,  0.33499061,  0.28615009],
 [ 2.61409047,  0.56090136, -0.20553452],
 [ 1.25850816, -0.17970479,  0.0458477 ],
 [ 1.29113206, -0.11666865,  0.23125646],
```

```
       [ 2.12360872, -0.20972948,  0.15418002],
       [ 2.38800302,  0.4646398 , -0.44953019],
       [ 2.84167278,  0.37526917, -0.49889808],
       [ 3.23067366,  1.37416509, -0.11454821],
       [ 2.15943764, -0.21727758,  0.20876317],
       [ 1.44416124, -0.14341341, -0.15323389],
       [ 1.78129481, -0.49990168, -0.17287519],
       [ 3.07649993,  0.68808568, -0.33559229],
       [ 2.14424331,  0.1400642 ,  0.73487894],
       [ 1.90509815,  0.04930053,  0.16218024],
       [ 1.16932634, -0.16499026,  0.28183584],
       [ 2.10761114,  0.37228787,  0.02729113],
       [ 2.31415471,  0.18365128,  0.32269375],
       [ 1.9222678 ,  0.40920347,  0.1135866 ],
       [ 1.41523588, -0.57491635,  0.29632253],
       [ 2.56301338,  0.2778626 ,  0.29256952],
       [ 2.41874618,  0.3047982 ,  0.50448266],
       [ 1.94410979,  0.1875323 ,  0.17782509],
       [ 1.52716661, -0.37531698, -0.12189817],
       [ 1.76434572,  0.07885885,  0.13048163],
       [ 1.90094161,  0.11662796,  0.72325156],
       [ 1.39018886, -0.28266094,  0.36290965]])
```

In [6]:
```
from sklearn.model_selection import train_test_split
(train_feat,test_feat,train_classes,test_classes)=train_test_split(X,Y,train_size=
dectree=DecisionTreeClassifier()
dectree.fit(train_feat,train_classes)
```

Out[6]:
```
DecisionTreeClassifier()
```

In [7]:
```
from sklearn import metrics
from sklearn.metrics import accuracy_score
pred=dectree.predict(test_feat)
print("accuracy",accuracy_score(test_classes,pred))
```

```
accuracy 0.9777777777777777
```

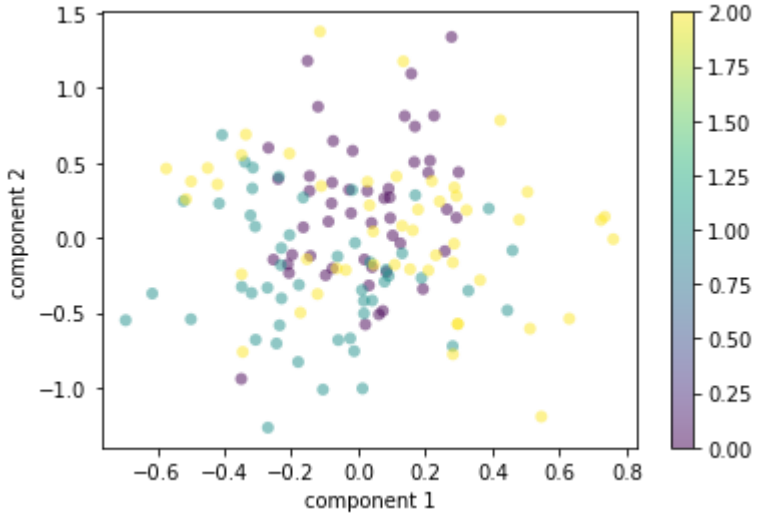
In [8]:
```
from sklearn.model_selection import train_test_split
(train_feat,test_feat,train_classes,test_classes)=train_test_split(X1,Y,train_size
dectree=DecisionTreeClassifier()
dectree.fit(train_feat,train_classes)
```

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

In [9]:
```
from sklearn import metrics
from sklearn.metrics import accuracy_score
pred=dectree.predict(test_feat)
print("accuracy",accuracy_score(test_classes,pred))
```

```
accuracy 0.9777777777777777
```

In [12]:
```
plt.scatter(X1[:,2],X1[:,1],c=iris.target,edgecolor='none',alpha=0.5)
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```


component 2
component 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

In [ ]:

Practical 29 - Cross Validation Techniques – Iris Dataset

21 June 2022

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold,StratifiedKFold,LeaveOneOut,ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [2]:
```
iris=pd.read_csv("D:\\ML\\iris.csv")
```

In [3]:
```
iris
```

Out[3]:

	Sepal length	Sepal width	Petal length	Petal width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [4]:
```
iris.dtypes
```

Out[4]:
```
Sepal  length  float64
Sepal   width  float64
Petal  length  float64
Petal width    float64
Class           object
dtype: object
```

In [5]:
```
features = iris[['Sepal length','Sepal width','Petal length','Petal width']].value
```

```
classes = iris['Class'].values
```

In [6]:
```
#kfold cross validation technique
kf=KFold(n_splits=5,random_state=1,shuffle=True)
#stratified kfold cross validation technique
skf=StratifiedKFold(n_splits=5)
#Leaveoneout cross validation technique
loocv=LeaveOneOut()
#shuffle split cross validation technique
shcv=ShuffleSplit()
```

In [7]:
```
#evaluating the data sets with kfold and Decision tree classifier
dst=DecisionTreeClassifier()
scores=cross_val_score(dst,features,classes,scoring='accuracy',cv=kf)  #kf  is  kFold
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))#this will give a
print(scores) # this will give accuracy for 5 splits
```

```
Accuarcy using Decision Tree:94.00 %
[0.96666667  0.96666667  0.96666667  0.93333333  0.86666667]
```

In [8]:
```
#evaluating the data sets with kfold and Decision tree classifier
nb=GaussianNB()
scores=cross_val_score(nb,features,classes,scoring='accuracy',cv=kf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))#this will give a
print(scores) # this will give accuracy for 5 splits
```

```
Accuarcy using Decision Tree:95.33 %
[0.96666667 0.96666667 1.         0.9        0.93333333]
```

In [9]:
```
#evaluating the data sets with kfold and Decision tree classifier
svm=SVC()
scores=cross_val_score(svm,features,classes,scoring='accuracy',cv=kf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))#this will give a
print(scores) # this will give accuracy for 5 splits
```

```
Accuarcy using Decision Tree:94.67 %
[0.96666667 0.93333333 0.9        0.96666667  0.96666667]
```

In [10]:
```
#evaluating the data sets with kfold and Decision tree classifier
knn=KNeighborsClassifier()
scores=cross_val_score(knn,features,classes,scoring='accuracy',cv=kf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))#this will give a
print(scores) # this will give accuracy for 5 splits
```

```
Accuarcy using Decision Tree:95.33 %
[1.         0.96666667 0.9        0.93333333  0.96666667]
```

Evaluating Dataset using Stratified Kfold

In [11]:
```
#evaluating the data sets with stratified kfold and Decision tree classifier
scores=cross_val_score(dst,features,classes,scoring='accuracy',cv=skf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:96.67 %
[0.96666667 0.96666667 0.9        1.         1.        ]
```

In [12]:
```
#evaluating the data sets with stratified kfold and Naive bayes
scores=cross_val_score(nb,features,classes,scoring='accuracy',cv=skf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:95.33 %
[0.93333333 0.96666667 0.93333333 0.93333333 1.        ]
```

In [13]:
```
#evaluating the data sets with stratified kfold and SVM classifier
scores=cross_val_score(svm,features,classes,scoring='accuracy',cv=skf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:96.67 %
[0.96666667 0.96666667 0.96666667 0.93333333 1.        ]
```

In [14]:
```
#evaluating the data sets with stratified kfold and knn classifier
scores=cross_val_score(knn,features,classes,scoring='accuracy',cv=skf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:97.33 %
[0.96666667 1.         0.93333333 0.96666667 1.        ]
```

leave one out

In [15]:
```
#evaluating the data sets with stratified kfold and Decision tree classifier
scores=cross_val_score(dst,features,classes,scoring='accuracy',cv=loocv)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:96.00 %
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```

In [16]:
```
#evaluating the data sets with stratified kfold and Naive bayes
scores=cross_val_score(nb,features,classes,scoring='accuracy',cv=loocv)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:95.33 %
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```

In [17]:
```
#evaluating the data sets with stratified kfold and SVM classifier
scores=cross_val_score(svm,features,classes,scoring='accuracy',cv=loocv)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:96.67 %
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```

In [18]:
```
#evaluating the data sets with stratified kfold and knn classifier
scores=cross_val_score(knn,features,classes,scoring='accuracy',cv=loocv)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:96.67 %
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```

shuffle

In [19]:
```
#evaluating the data sets with stratified kfold and Decision tree classifier
scores=cross_val_score(dst,features,classes,scoring='accuracy',cv=skf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:96.00 %
[0.96666667 0.96666667 0.9        0.96666667 1.        ]
```

In [20]:
```
#evaluating the data sets with stratified kfold and Naive bayes
scores=cross_val_score(nb,features,classes,scoring='accuracy',cv=skf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:95.33 %
[0.93333333 0.96666667 0.93333333 0.93333333 1.        ]
```

In [21]:
```
#evaluating the data sets with stratified kfold and SVM classifier
scores=cross_val_score(svm,features,classes,scoring='accuracy',cv=skf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:96.67 %
[0.96666667 0.96666667 0.96666667 0.93333333 1.        ]
```

In [22]:
```
#evaluating the data sets with stratified kfold and knn classifier
scores=cross_val_score(knn,features,classes,scoring='accuracy',cv=skf)
print("Accuarcy using Decision Tree:%.2f %%"%(scores.mean()*100))
print(scores)
```

```
Accuarcy using Decision Tree:97.33 %
[0.96666667 1.         0.93333333 0.96666667 1.        ]
```

In [ ]:

AIML
Project

GROUP 6:

9 - JAIDEEP SINGH HUNDAL

10 - SHREYA JAGADALE

25 - MRINAAL PALIWAL

41 - PALLAVI SAWANT

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import  metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold, LeaveOneOut, ShuffleSplit, StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
from sklearn import datasets
from sklearn import decomposition
from sklearn.feature_selection import SelectPercentile,SelectKBest
import warnings
warnings.filterwarnings('ignore') #ignoring warmings
```

In [2]:

```
data = pd.read_csv('C:\\Users\\Shreyash\\Desktop\\data.csv')
data
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

569 rows × 33 columns

Data Cleaning

In [3]:

```
data.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.1

5 rows × 33 columns

In [4]:

```
data.head(10)
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.1
5	843786	M	12.45	15.70	82.57	477.1	0.1
6	844359	M	18.25	19.98	119.60	1040.0	0.0
7	84458202	M	13.71	20.83	90.20	577.9	0.
8	844981	M	13.00	21.82	87.50	519.8	0.1
9	84501001	M	12.46	24.04	83.97	475.9	0.

10 rows × 33 columns

In [5]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---                           ------------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [6]:

```
data.shape
```

Out[6]:

```
(569, 33)
```

In [7]:

```
data.isnull()
```

Out[7]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
0	False	False	False	False	False	False	F
1	False	False	False	False	False	False	F
2	False	False	False	False	False	False	F
3	False	False	False	False	False	False	F
4	False	False	False	False	False	False	F
...	...	...	...	...	...	...	
564	False	False	False	False	False	False	F
565	False	False	False	False	False	False	F
566	False	False	False	False	False	False	F
567	False	False	False	False	False	False	F
568	False	False	False	False	False	False	F

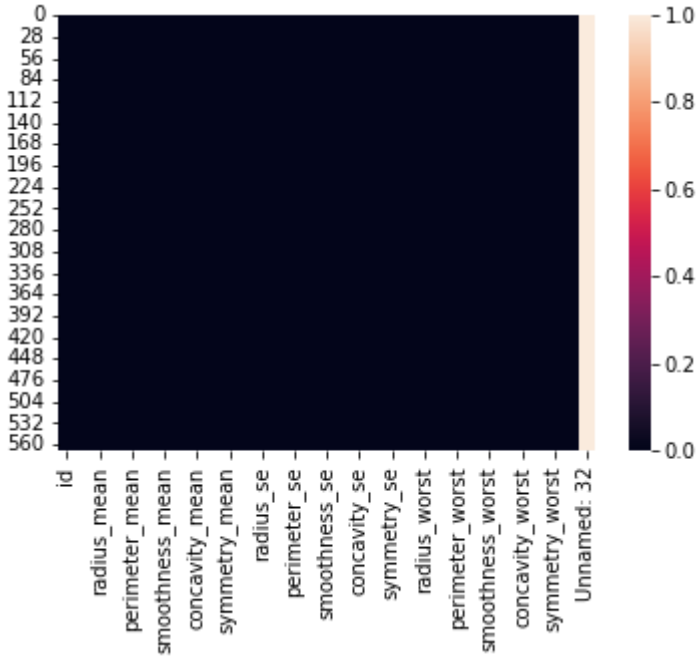
569 rows × 33 columns

In [8]:

```
sns.heatmap(data.isnull())
```

Out[8]:

```
<AxesSubplot:>
```

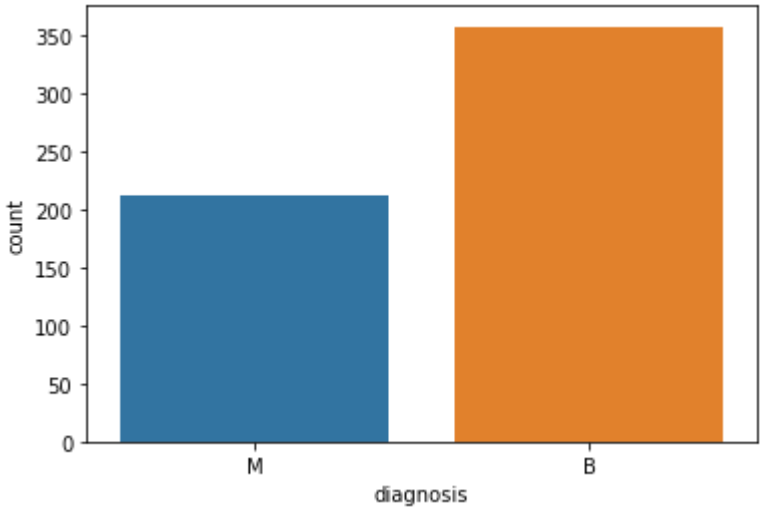


In [9]:

```
sns.countplot(x='diagnosis', data=data)
```

Out[9]:

```
<AxesSubplot:xlabel='diagnosis', ylabel='count'>
```



In [10]:

```
data.diagnosis.value_counts()
```

Out[10]:

```
B     357
M     212
Name: diagnosis, dtype: int64
```

In [11]:

```
data.dtypes
```

Out[11]:

```
id                         int64
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
Unnamed: 32              float64
dtype: object
```

In [12]:

```
data['diagnosis'].unique()
```

Out[12]:

```
array(['M', 'B'], dtype=object)
```

In [13]:

```
channel_map = {'M':0, 'B':1}
```

In [14]:

```
data['diagnosis'] = data['diagnosis'].map(channel_map)
```

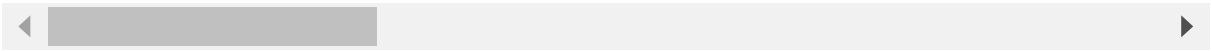
In [15]:

```
data
```

Out[15]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	0	17.99	10.38	122.80	1001.0	
1	842517	0	20.57	17.77	132.90	1326.0	
2	84300903	0	19.69	21.25	130.00	1203.0	
3	84348301	0	11.42	20.38	77.58	386.1	
4	84358402	0	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	0	21.56	22.39	142.00	1479.0	
565	926682	0	20.13	28.25	131.20	1261.0	
566	926954	0	16.60	28.08	108.30	858.1	
567	927241	0	20.60	29.33	140.10	1265.0	
568	92751	1	7.76	24.54	47.92	181.0	

569 rows × 33 columns



In [16]:

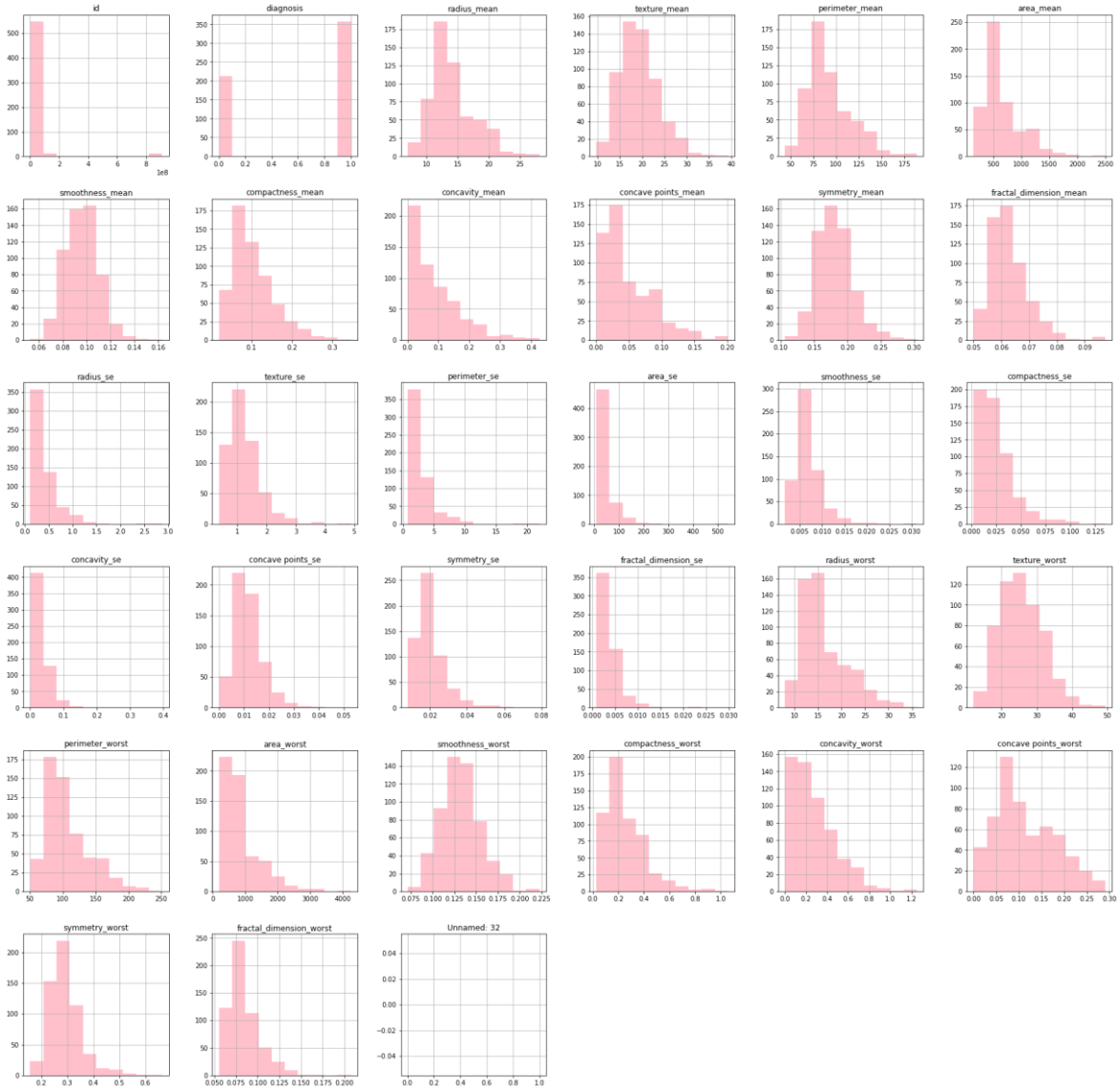
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---                           --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    int64  
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(2)
memory usage: 146.8 KB
```

Data Visualization

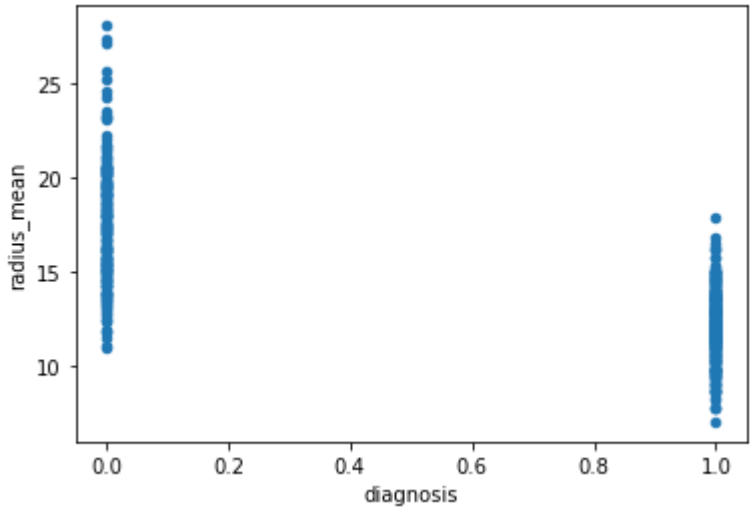
In [17]:

```
#plot the histograms for each feature:
data.hist(figsize = (30,30), color = 'pink')
plt.show()
```



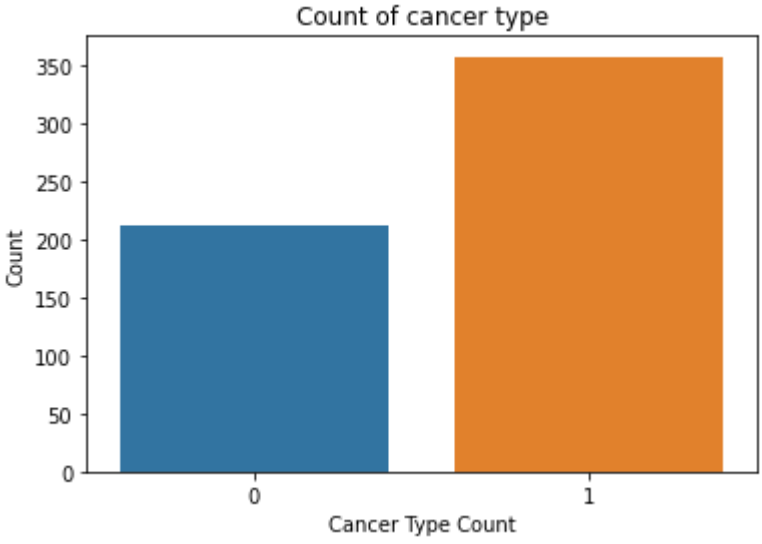
In [18]:

```
#scatterplot
data.plot.scatter(x='diagnosis',y='radius_mean');
```



In [19]:

```
# Analyzing the target variable

plt.title('Count of cancer type')
sns.countplot(data['diagnosis'])
plt.xlabel('Cancer Type Count')
plt.ylabel('Count')
plt.show()
```

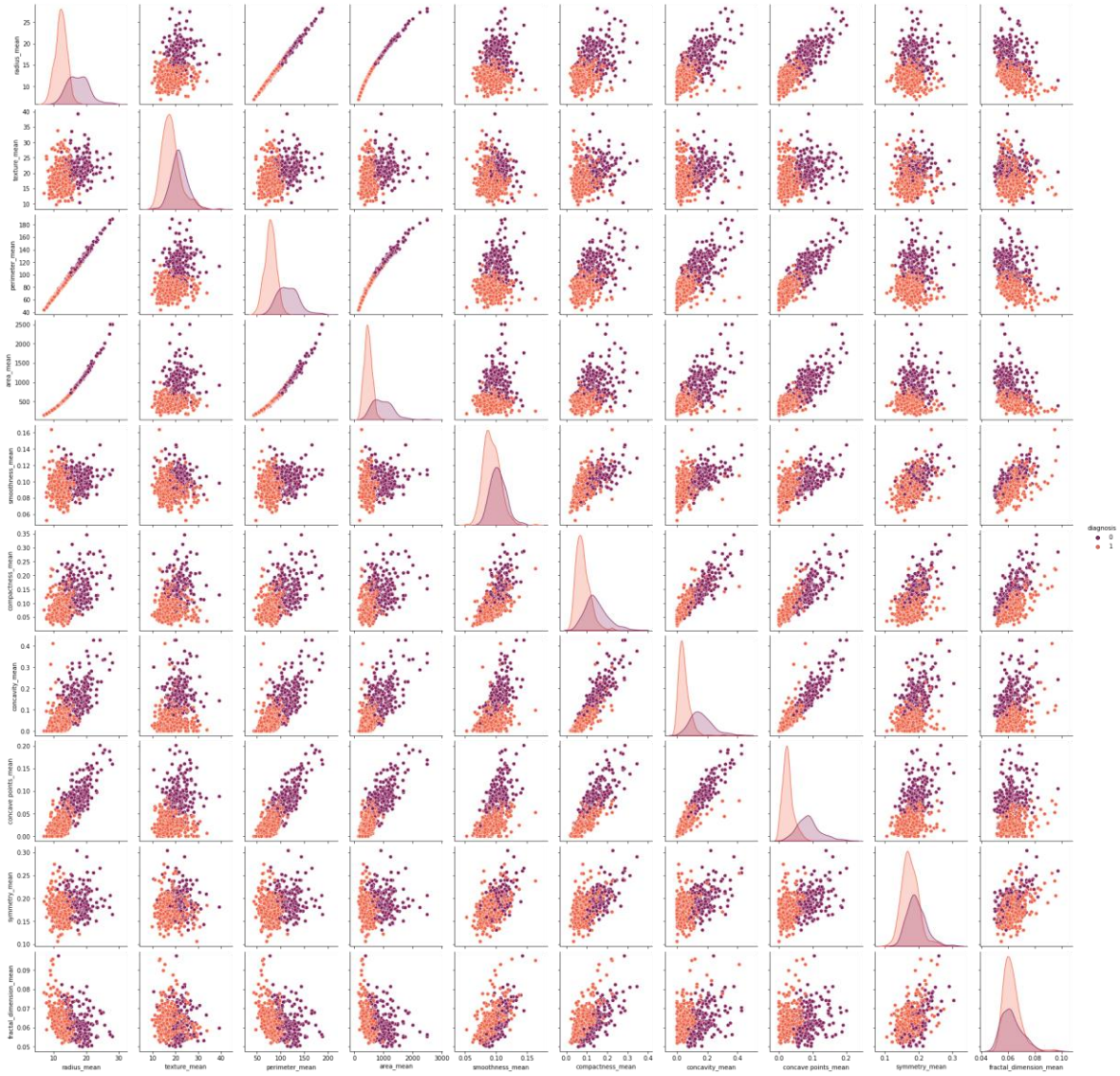


In [20]:

```
#generate a scatter plot with the following columns:

columns = ['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoo
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']

sns.pairplot(data=data[columns], hue="diagnosis", palette='rocket')
```

Out[20]:

<seaborn.axisgrid.PairGrid at 0x26220609ee0>



In [21]:

```
cor = data.corr()
cor
```

Out[21]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	are
id	1.000000	-0.039769	0.074626	0.099770	0.073159	0
diagnosis	-0.039769	1.000000	-0.730029	-0.415185	-0.742636	-0
radius_mean	0.074626	-0.730029	1.000000	0.323782	0.997855	0
texture_mean	0.099770	-0.415185	0.323782	1.000000	0.329533	0
perimeter_mean	0.073159	-0.742636	0.997855	0.329533	1.000000	0
area_mean	0.096893	-0.708984	0.987357	0.321086	0.986507	1
smoothness_mean	-0.012968	-0.358560	0.170581	-0.023389	0.207278	0
compactness_mean	0.000096	-0.596534	0.506124	0.236702	0.556936	0
concavity_mean	0.050080	-0.696360	0.676764	0.302418	0.716136	0
concave points_mean	0.044158	-0.776614	0.822529	0.293464	0.850977	0
symmetry_mean	-0.022114	-0.330499	0.147741	0.071401	0.183027	0
fractal_dimension_mean	-0.052511	0.012838	-0.311631	-0.076437	-0.261477	-0
radius_se	0.143048	-0.567134	0.679090	0.275869	0.691765	0
texture_se	-0.007526	0.008303	-0.097317	0.386358	-0.086761	-0
perimeter_se	0.137331	-0.556141	0.674172	0.281673	0.693135	0
area_se	0.177742	-0.548236	0.735864	0.259845	0.744983	0
smoothness_se	0.096781	0.067016	-0.222600	0.006614	-0.202694	-0
compactness_se	0.033961	-0.292999	0.206000	0.191975	0.250744	0
concavity_se	0.055239	-0.253730	0.194204	0.143293	0.228082	0
concave points_se	0.078768	-0.408042	0.376169	0.163851	0.407217	0
symmetry_se	-0.017306	0.006522	-0.104321	0.009127	-0.081629	-0
fractal_dimension_se	0.025725	-0.077972	-0.042641	0.054458	-0.005523	-0
radius_worst	0.082405	-0.776454	0.969539	0.352573	0.969476	0
texture_worst	0.064720	-0.456903	0.297008	0.912045	0.303038	0
perimeter_worst	0.079986	-0.782914	0.965137	0.358040	0.970387	0
area_worst	0.107187	-0.733825	0.941082	0.343546	0.941550	0
smoothness_worst	0.010338	-0.421465	0.119616	0.077503	0.150549	0
compactness_worst	-0.002968	-0.590998	0.413463	0.277830	0.455774	0
concavity_worst	0.023203	-0.659610	0.526911	0.301025	0.563879	0
concave points_worst	0.035174	-0.793566	0.744214	0.295316	0.771241	0
symmetry_worst	-0.044224	-0.416294	0.163953	0.105008	0.189115	0
fractal_dimension_worst	-0.029866	-0.323872	0.007066	0.119205	0.051019	0
Unnamed: 32	NaN	NaN	NaN	NaN	NaN	

33 rows × 33 columns

In [22]:

```
cor.shape
```

Out[22]:

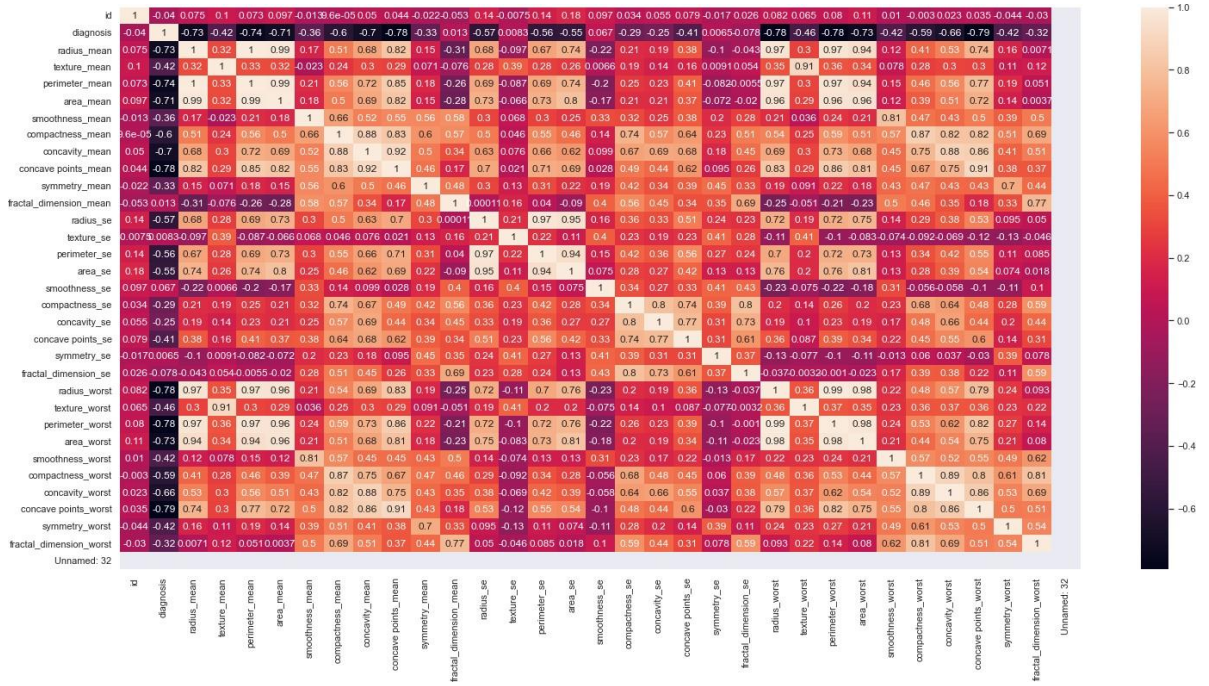
```
(33, 33)
```

In [23]:

```
sns.set(rc = {'figure.figsize':(25,12)})
sns.heatmap(cor,annot=True)
```

Out[23]:

```
<AxesSubplot:>
```



From above correlation we can see that 'concave points_worst' has the highest absolute correlation withdiagnosis

In [24]:

```
srx = data['concave points_worst']
srx.shape
```

Out[24]:

```
(569,)
```

In [25]:

```
srxx = srx.values.reshape(-1,1)
srxx.shape
```

Out[25]:

```
(569, 1)
```

We will use 'concave points_worst' as feature forsimple LR

In [26]:

```
y = data['diagnosis']
y.shape
```

Out[26]:

```
(569,)
```

In [27]:

```
ydt = y.values.reshape(-1,1)
ydt.shape
```

Out[27]:

```
(569, 1)
```

In [28]:

```
xtr, xts, ytr, yts = train_test_split(srxx,ydt, test_size = 0.1, random_state = 0)
print("Size of training set:", xtr.shape)
print("Size of testing set:", xts.shape)
```

```
Size of training set: (512, 1)
Size of testing set: (57, 1)
```

9 - JAIDEEP SINGH

Multivariate Logistic Regression

In [62]:

```
LR=LogisticRegression()
```

In [63]:

```
LR.fit(xtr,ytr)
```

Out[63]:

```
LogisticRegression()
```

In [64]:

```
y_pred = LR.predict(xts)
```

In [65]:

```
acc = mean_squared_error(yts, y_pred)
acc
```

Out[65]:

```
0.04678362573099415
```

In [66]:

```
LR.score(xts, yts)
```

Out[66]:

```
0.9532163742690059
```

In [67]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

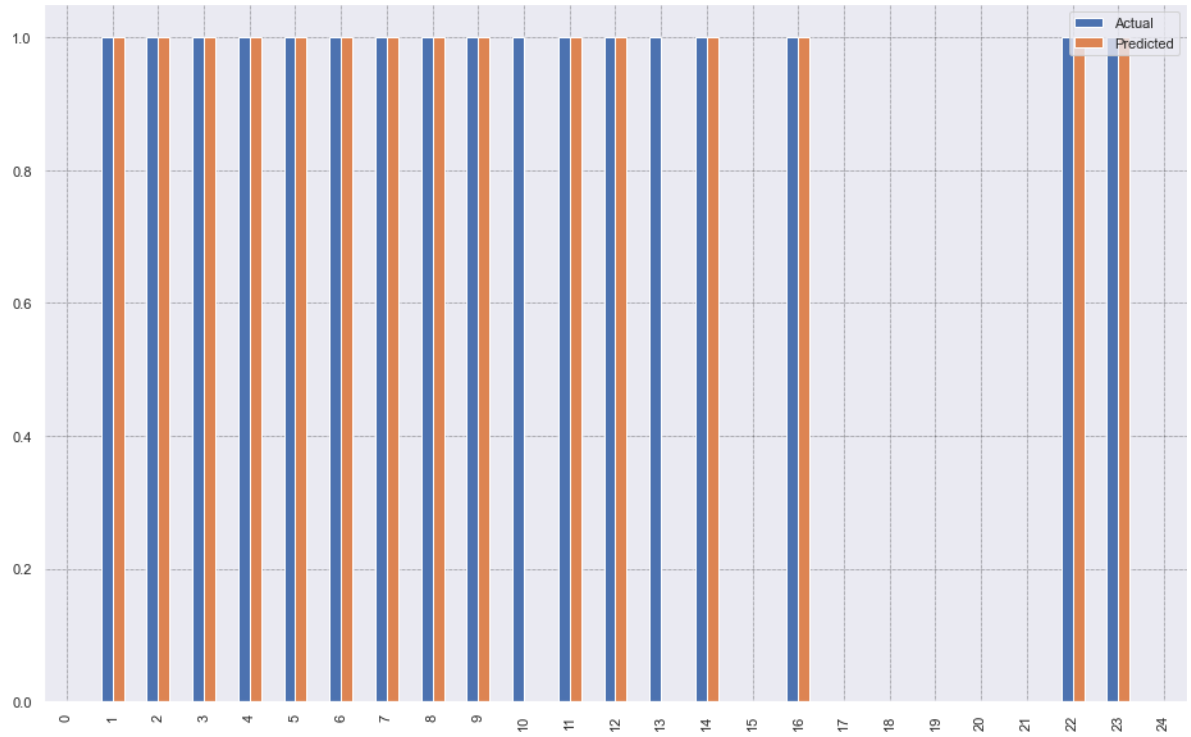
Out[67]:

	Actual	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
...	...	...
166	0	0
167	0	0
168	1	1
169	1	1
170	1	1

171 rows × 2 columns

In [68]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



In [69]:

```
print('Mean Absolute Error:',mean_absolute_error(yts,y_pred))
print('Mean Squared Error:',mean_squared_error(yts,y_pred))
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(yts,y_pred)))
```

```
Mean Absolute Error: 0.04678362573099415Mean
Squared Error: 0.04678362573099415
Root Mean Squared Error: 0.21629522817435004
```

25 - MRINAAL PALIWAL

Naive-Bayes

In [70]:

```
nb = BernoulliNB()
gnb = GaussianNB()
mnb = MultinomialNB()
```

In [71]:

```
nb.fit(xtr,ytr)
gnb.fit(xtr,ytr)
mnb.fit(xtr,ytr)
```

Out[71]:

```
MultinomialNB()
```

BernoulliNB

In [72]:

```
ypred = nb.predict(xts)
```

In [73]:

```
accuracy_score(yts,ypred)
```

Out[73]:

```
0.631578947368421
```

In [74]:

```
print(classification_report(yts,ypred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        63
           1       0.63      1.00      0.77       108

    accuracy                           0.63       171
   macro avg       0.32      0.50      0.39       171
weighted avg       0.40      0.63      0.49       171
```

In [75]:

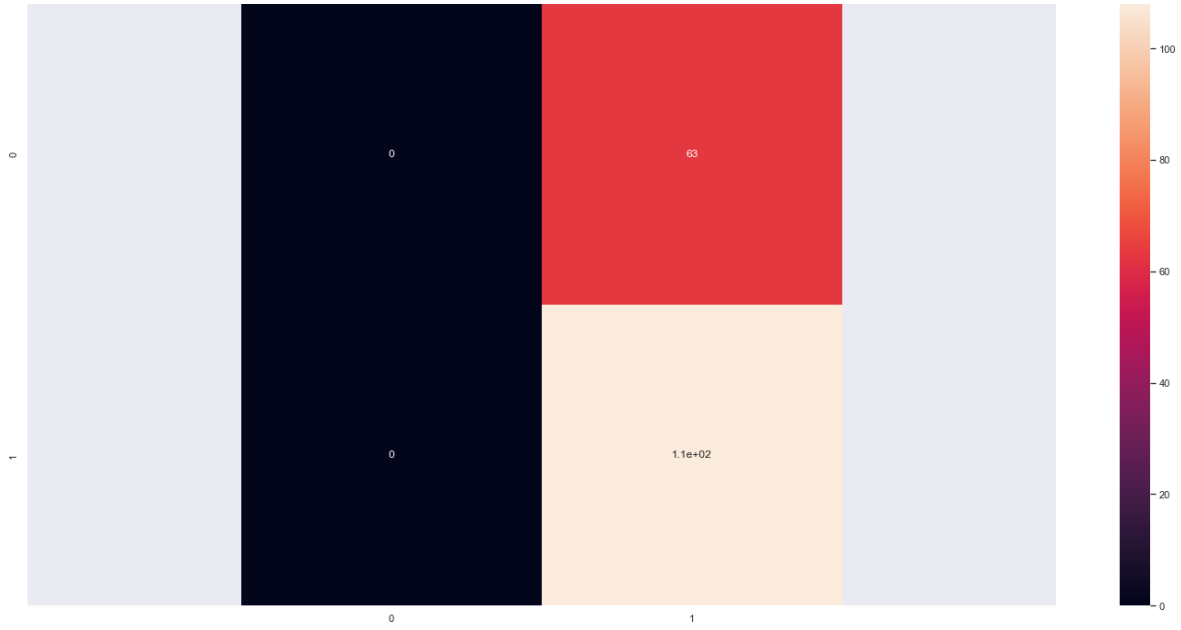
```
cf = confusion_matrix(yts,ypred)
cf
```

Out[75]:

```
array([[  0,  63],
       [  0, 108]], dtype=int64)
```

In [76]:

```
sns.heatmap (cf,annot=True)
plt.axis('equal')
plt.show()
```



In [77]:

```
nb.score(xts,yts)
```

Out[77]:

```
0.631578947368421
```

In [78]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

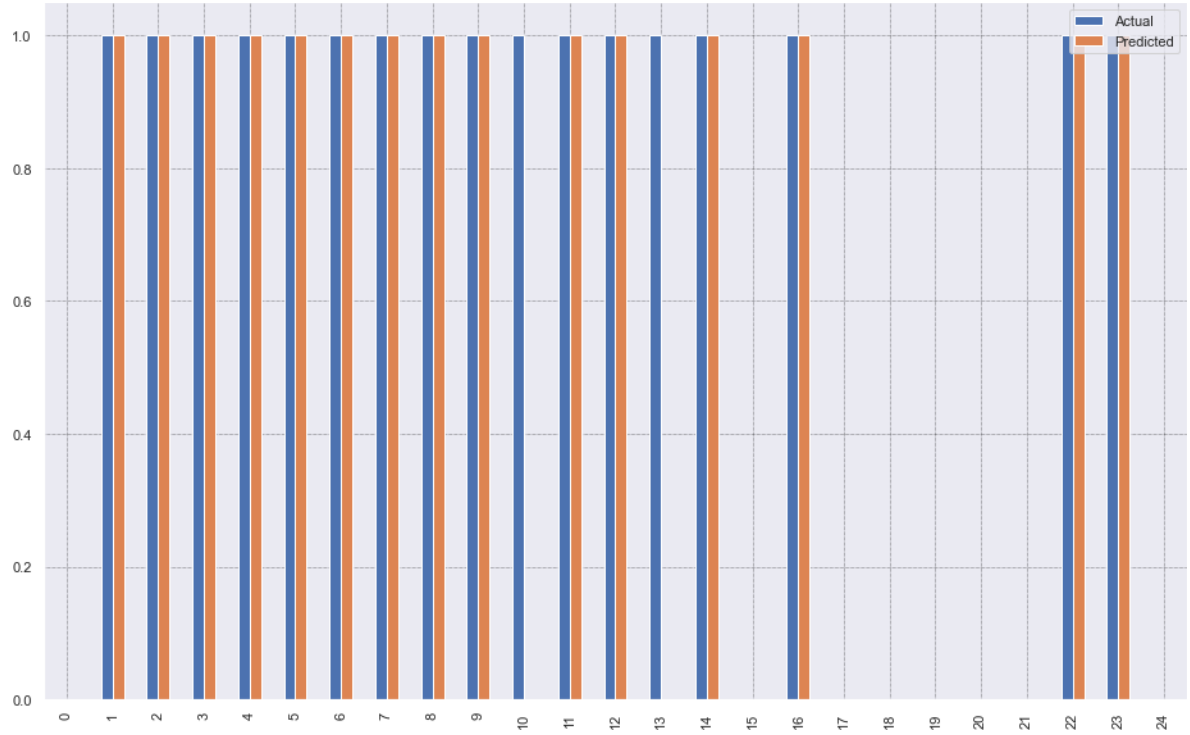
Out[78]:

	Actual	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
...	...	...
166	0	0
167	0	0
168	1	1
169	1	1
170	1	1

171 rows × 2 columns

In [79]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



GaussianNB

In [80]:

```
ypred = gnb.predict(xts)
```

In [81]:

```
accuracy_score(yts,ypred)
```

Out[81]:

```
0.9239766081871345
```

In [82]:

```
print(classification_report(yts,ypred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.90        63
           1       0.94      0.94      0.94       108

    accuracy                           0.92       171
   macro avg       0.92      0.92      0.92       171
weighted avg       0.92      0.92      0.92       171
```

In [83]:

```
cf = confusion_matrix(yts,ypred)
cf
```

Out[83]:

```
array([[ 57,   6],
       [  7, 101]], dtype=int64)
```

In [84]:

```
sns.heatmap (cf,annot=True)
plt.axis('equal')
plt.show()
```

In [85]:

```
gnb.score(xts,yts)
```

Out[85]:

```
0.9239766081871345
```

In [86]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

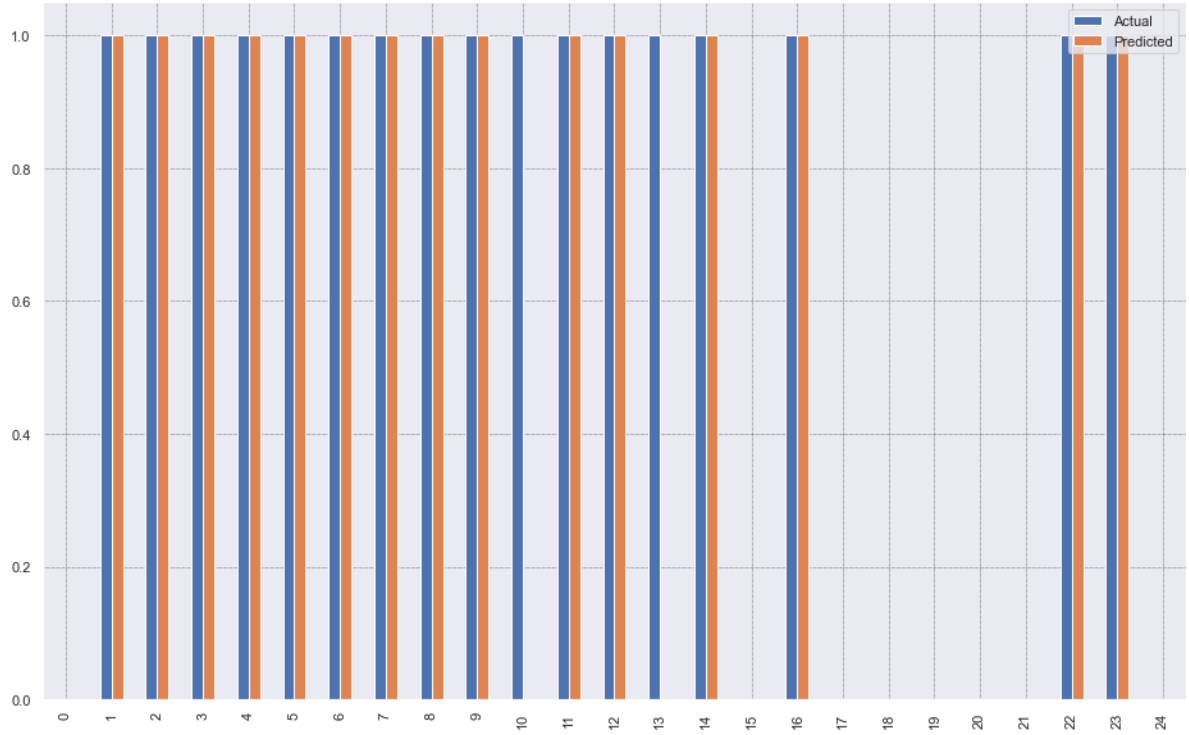
Out[86]:

	Actual	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
...	...	...
166	0	0
167	0	0
168	1	1
169	1	1
170	1	1

171 rows × 2 columns

In [87]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



MultinomialNB

In [88]:

```
ypred = mnb.predict(xts)
```

In [89]:

```
accuracy_score(yts,ypred)
```

Out[89]:

```
0.9005847953216374
```

In [90]:

```
print(classification_report(yts,ypred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.76      0.85        63
           1       0.88      0.98      0.93       108

    accuracy                           0.90       171
   macro avg       0.92      0.87      0.89       171
weighted avg       0.91      0.90      0.90       171
```

In [91]:

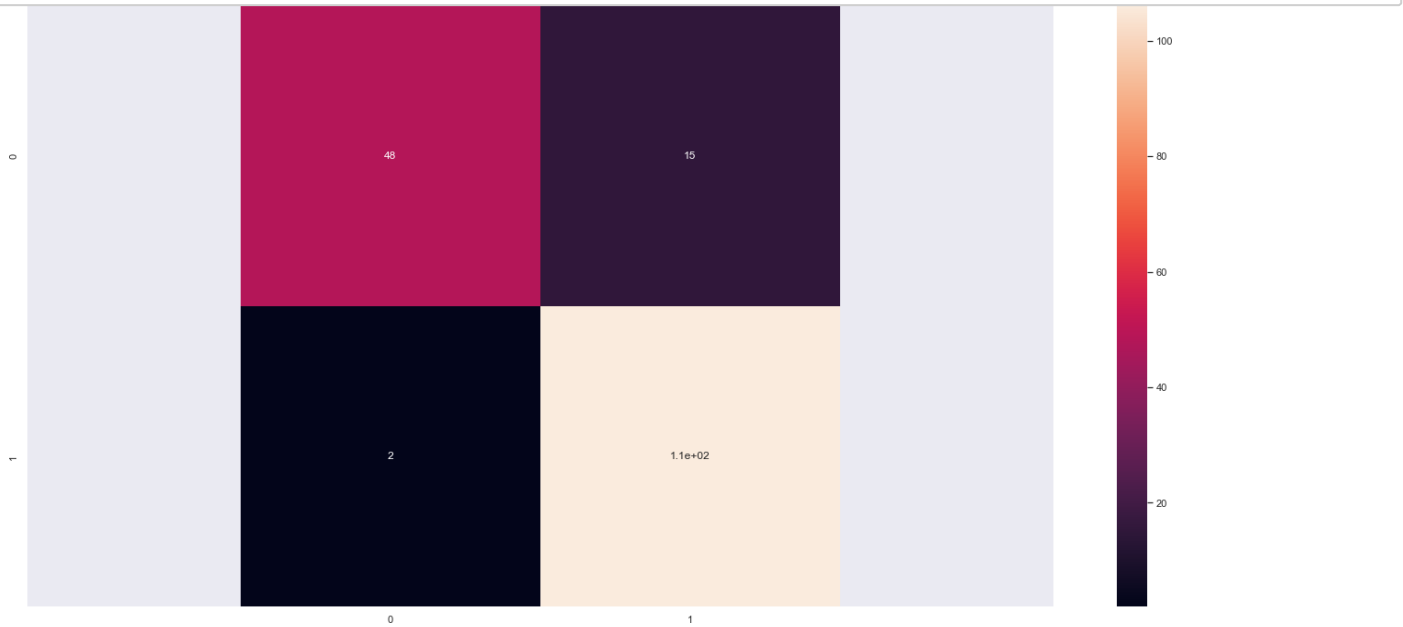
```
cf = confusion_matrix(yts,ypred)
cf
```

Out[91]:

```
array([[ 48,  15],
       [  2, 106]], dtype=int64)
```

In [92]:

```
sns.heatmap (cf,annot=True)
plt.axis('equal')
plt.show()
```



In [93]:

```
mnb.score(xts,yts)
```

Out[93]:

```
0.9005847953216374
```

In [94]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

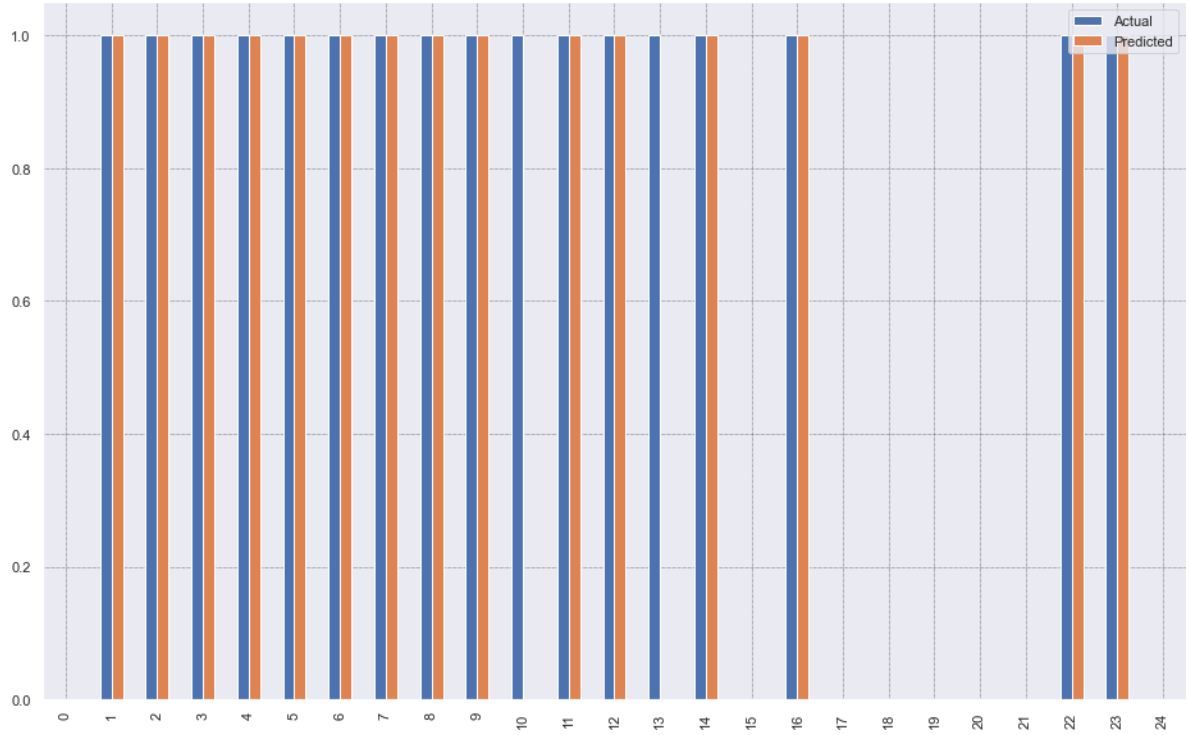
Out[94]:

	Actual	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
...	...	...
166	0	0
167	0	0
168	1	1
169	1	1
170	1	1

171 rows × 2 columns

In [95]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



Cross Validation

In [104]:

```
# k-fold cross validation technique
kf = KFold(n_splits=5, random_state=1,shuffle=True)
# stratified kfold cross validation technique
skf = StratifiedKFold(n_splits=5)
# LeaveOneOut cross validation technique
loocv = LeaveOneOut()
# shuffle split cross validation technique
shvc = ShuffleSplit()
```

In [105]:

```
# evaluating the data sets with kfold and DecisionTreeClassifier
dst = DecisionTreeClassifier()
scores = cross_val_score(dst,feat,classes,scoring='accuracy',cv=kf)
print('Accuracy using Decision Tree: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using Decision Tree: 93.147027%
[0.94736842 0.92982456 0.90350877 0.93859649 0.9380531 ]
```

In [96]:

```
# evaluating the data sets with kfold and Naive Bayes Classifier
nb = GaussianNB()
scores = cross_val_score(nb,feat,classes,scoring='accuracy',cv=kf)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 93.851886%
[0.94736842 0.93859649 0.9122807  0.93859649 0.95575221]
```

In [107]:

```
# evaluating the data sets with kfold and SVM
svm = SVC()
scores = cross_val_score(svm,feat,classes,scoring='accuracy',cv=kf)
print('Accuracy using SVC: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using SVC: 91.386431%
[0.90350877 0.92982456 0.88596491 0.94736842 0.90265487]
```

In [108]:

```
# evaluating the data sets with kfold and KNN Classifier
knn = KNeighborsClassifier()
scores = cross_val_score(knn,feat,classes,scoring='accuracy',cv=kf)
print('Accuracy using KNN: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using KNN: 92.268281%
[0.93859649 0.89473684 0.88596491 0.96491228 0.92920354]
```

In [109]:

```
# evaluating the data sets with Stratified KFold and DecisionTree
scores = cross_val_score(dst,feat,classes,scoring='accuracy',cv=skf)
print('Accuracy using Decision Tree: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using Decision Tree: 90.504580%
[0.90350877 0.92105263 0.90350877 0.92105263 0.87610619]
```

In [110]:

```
# evaluating the data sets Stratified KFold and Naive Bayes Classifier
scores = cross_val_score(nb,feat,classes,scoring='accuracy',cv=skf)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 93.851886%
[0.92105263 0.92105263 0.94736842 0.94736842 0.95575221]
```

In [111]:

```
# evaluating the data sets Stratified KFold and SVM
scores = cross_val_score(svm,feat,classes,scoring='accuracy',cv=skf)
print('Accuracy using SVM: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using SVM: 91.217202%
[0.85087719 0.89473684 0.92982456 0.94736842 0.9380531 ]
```

In [112]:

```
# evaluating the data sets Stratified KFold and KNN
scores = cross_val_score(knn,feat,classes,scoring='accuracy',cv=skf)
print('Accuracy using KNN: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using KNN: 92.794597%
[0.88596491 0.93859649 0.93859649 0.94736842 0.92920354]
```

In [113]:

```
# evaluating the data sets with LeaveOneOut and DecisionTree
scores = cross_val_score(dst,feat,classes,scoring='accuracy',cv=loocv)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 92.267135%
[1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [114]:

```
# evaluating the data sets LeaveOneOut and Naive Bayes Classifier
scores = cross_val_score(nb,feat,classes,scoring='accuracy',cv=loocv)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 93.848858%
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [115]:

```
# evaluating the data sets LeaveOneOut and SVM
scores = cross_val_score(svm,feat,classes,scoring='accuracy',cv=loocv)
print('Accuracy using SVM: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using SVM: 91.212654%
[1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.]
```

In [116]:

```
# evaluating the data sets LeaveOneOut and KNN
scores = cross_val_score(knn,feat,classes,scoring='accuracy',cv=loocv)
print('Accuracy using KNN: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using KNN: 93.321617%
[1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [117]:

```
# evaluating the data sets with ShuffleSplit and DecisionTree
scores = cross_val_score(dst,feat,classes,scoring='accuracy',cv=shvc)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 90.701754%
[0.94736842 0.84210526 0.92982456 0.96491228 0.89473684 0.89473684
 0.89473684 0.89473684 0.89473684 0.9122807 ]
```

In [118]:

```
# evaluating the data sets with ShuffleSplit and Naive Bayes Classifier
scores = cross_val_score(nb,feat,classes,scoring='accuracy',cv=shvc)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 95.087719%
[0.96491228 0.92982456 0.98245614 0.94736842 1.         0.94736842
 0.96491228 0.89473684 1.         0.87719298]
```

In [119]:

```
# evaluating the data sets with ShuffleSplit and SVM
scores = cross_val_score(svm,feat,classes,scoring='accuracy',cv=shvc)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 91.403509%
[0.94736842 0.87719298 0.94736842 0.85964912 0.9122807  0.87719298
 0.94736842 0.98245614 0.9122807  0.87719298]
```

In [120]:

```
# evaluating the data sets with ShuffleSplit and KNN
scores = cross_val_score(knn,feat,classes,scoring='accuracy',cv=shvc)
print('Accuracy using KNN: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using KNN: 93.859649%
[0.98245614 1.         0.87719298 0.94736842 0.98245614 0.9122807
 0.9122807  0.92982456 0.92982456 0.9122807 ]
```

41 - PALLAVI SAWANT

PreProcessing

Define Target Data

```
Diagnosis column will be our target data
  If the cancer is Benign, it will be 0
  If the cancer is Malignant, it will be 1
```

In [21]:

```
#split the dataset into training and testing sets
features = df.drop(['diagnosis'],axis=1).values
classes=df['diagnosis'].values
```

In [22]:

```
feat_train, feat_test, class_train, class_test = train_test_split(features, classes,
                                                  test_size=0.2, random_state=9)
```

In [23]:

```
print('features train shape: ', feat_train.shape)
print('classes train shape: ', class_train.shape)
print('features test shape: ', feat_test.shape)
print('classes test shape: ', class_test.shape)
```

```
features train shape:  (455, 30)
classes train shape:  (455,)
features test shape:  (114, 30)
classes test shape:  (114,)
```

Decision Tree Classifier

Criterion=Gini

In [24]:

```
#Training
dectree=DecisionTreeClassifier(criterion='gini')

dectree.fit(feat_train,class_train)
```

Out[24]:

```
DecisionTreeClassifier()
```

In [25]:

```
#predict target values
pred=dectree.predict(feat_test)
print(pred)
```

```
       ['M' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
        'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'M' 'M'
        'M' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B'
        'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
        'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B''B' 'B'
 'B' 'B' 'B' 'M']
```

In [26]:

```
#confusion matrix and accuracy
print("Accuracy",accuracy_score(class_test,pred))
print("Classification Report\n",classification_report(class_test,pred))
print("Confusion Matrix\n",confusion_matrix(class_test,pred))
```

```
Accuracy 0.956140350877193
Classification Report
               precision    recall  f1-score   support

           B       0.97      0.96      0.97        74
           M       0.93      0.95      0.94        40

    accuracy                           0.96       114
   macro avg       0.95      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

Confusion Matrix
 [[71  3]
 [ 2 38]]
```

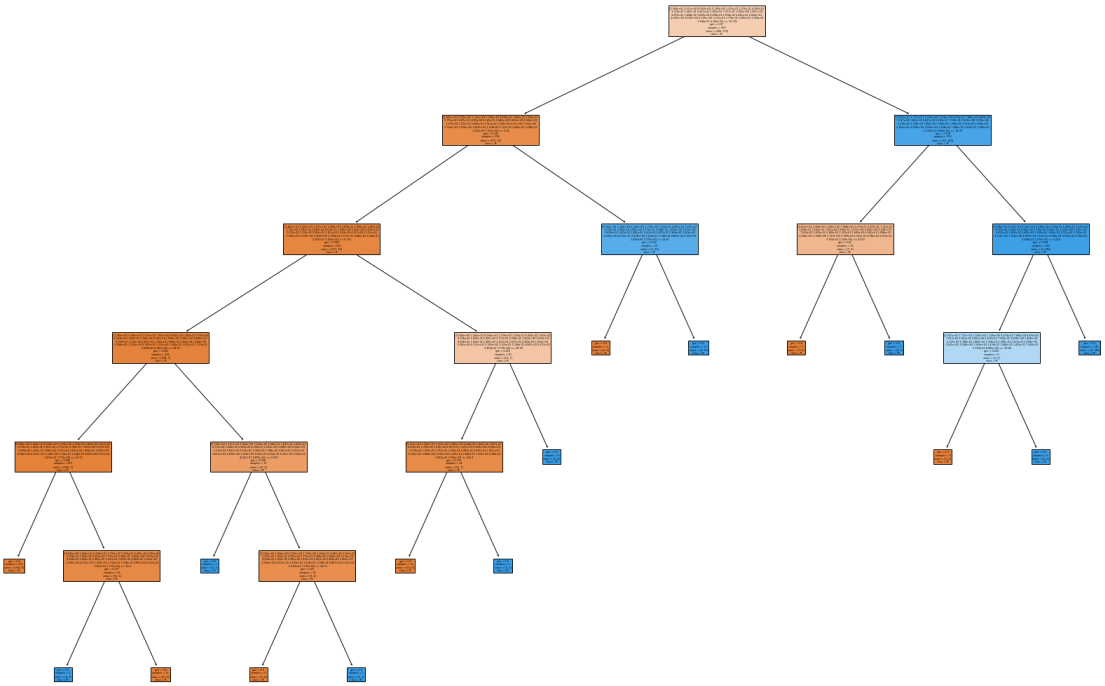
In [27]:

```
#Testing
pred=dectree.predict(feat_test)
print("Accuracy:",metrics.accuracy_score(class_test,pred))
```

```
Accuracy: 0.956140350877193
```

In [28]:

```
from sklearn import tree
fig=plt.figure(figsize=(30,20))
_=tree.plot_tree(dectree,feature_names=features,class_names=classes,filled=True)
```



In [29]:

```
text_representation=tree.export_text(dectree)
print(text_representation)
```

```
|--- feature_20 <= 16.80
|   |--- feature_27 <= 0.16
|   |   |--- feature_27 <= 0.13
|   |   |   |--- feature_13 <= 38.35
|   |   |   |   |--- feature_21 <= 33.27
|   |   |   |   |   |--- class: B
|   |   |   |   |--- feature_21 >  33.27
|   |   |   |   |   |--- feature_21 <= 33.80
|   |   |   |   |   |   |--- class: M
|   |   |   |   |   |--- feature_21 >  33.80
|   |   |   |   |   |   |--- class: B
|   |   |   |--- feature_13 >  38.35
|   |   |   |   |--- feature_28 <= 0.21
|   |   |   |   |   |--- class: M
|   |   |   |   |--- feature_28 >  0.21
|   |   |   |   |   |--- feature_21 <= 28.13
|   |   |   |   |   |   |--- class: B
|   |   |   |   |   |--- feature_21 >  28.13
|   |   |   |   |   |   |--- class: M
|   |   |--- feature_27 >  0.13
|   |   |   |--- feature_21 <= 28.78
|   |   |   |   |--- feature_23 <= 811.10
|   |   |   |   |   |--- class: B
|   |   |   |   |--- feature_23 >  811.10
|   |   |   |   |   |--- class: M
|   |   |   |--- feature_21 >  28.78
|   |   |   |   |--- class: M
|   |--- feature_27 >  0.16
|   |   |--- feature_21 <= 23.47
|   |   |   |--- class: B
|   |   |--- feature_21 >  23.47
|   |   |   |--- class: M
|--- feature_20 >  16.80
|   |--- feature_1 <= 14.99
|   |   |--- feature_17 <= 0.01
|   |   |   |--- class: B
|   |   |--- feature_17 >  0.01
|   |   |   |--- class: M
|   |--- feature_1 >  14.99
|   |   |--- feature_26 <= 0.22
|   |   |   |--- feature_1 <= 19.86
|   |   |   |   |--- class: B
|   |   |   |--- feature_1 >  19.86
|   |   |   |   |--- class: M
|   |   |--- feature_26 >  0.22
|   |   |   |--- class: M
```

```
Criterion=Entropy
```

In [30]:

```
#Training
dectree=DecisionTreeClassifier(criterion='entropy')
dectree.fit(feat_train,class_train)
```

Out[30]:

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

In [31]:

```
#confusion matrix and accuracy
pred=dectree.predict(feat_test)
print(pred)
print("Accuracy",accuracy_score(class_test,pred))
print("Classification Report\n",classification_report(class_test,pred))
print("Confusion Matrix\n",confusion_matrix(class_test,pred))
```

```
       ['B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
        'B' 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'M' 'M'
        'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B'
        'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
        'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B''B' 'B'
 'B' 'B' 'B' 'M']
Accuracy 0.9210526315789473
Classification Report
              precision    recall  f1-score   support

           B       0.92      0.96      0.94        74
           M       0.92      0.85      0.88        40

    accuracy                           0.92       114
   macro avg       0.92      0.90      0.91       114
weighted avg       0.92      0.92      0.92       114

Confusion Matrix
 [[71  3]
 [ 6 34]]
```

In [32]:

```
text_representation=tree.export_text(dectree)
print(text_representation)
```

```
|--- feature_22 <= 105.95
|    |--- feature_27 <= 0.13
|    |    |--- feature_10 <= 0.64
|    |    |    |--- feature_21 <= 33.27
|    |    |    |    |--- class: B
|    |    |    |--- feature_21 >  33.27
|    |    |    |    |--- feature_21 <= 33.80
|    |    |    |    |    |--- class: M
|    |    |    |    |--- feature_21 >  33.80
|    |    |    |    |    |--- class: B
|    |    |--- feature_10 >  0.64
|    |    |    |--- feature_24 <= 0.11
|    |    |    |    |--- class: B
|    |    |    |--- feature_24 >  0.11
|    |    |    |    |--- class: M
|    |--- feature_27 >  0.13
|    |    |--- feature_1 <= 20.30
|    |    |    |--- feature_28 <= 0.36
|    |    |    |    |--- class: B
|    |    |    |--- feature_28 >  0.36
|    |    |    |    |--- feature_22 <= 78.63
|    |    |    |    |    |--- class: B
|    |    |    |    |--- feature_22 >  78.63
|    |    |    |    |    |--- class: M
|    |    |--- feature_1 >  20.30
|    |    |    |--- class: M
|--- feature_22 >  105.95
|    |--- feature_22 <= 117.45
|    |    |--- feature_21 <= 27.46
|    |    |    |--- feature_24 <= 0.14
|    |    |    |    |--- class: B
|    |    |    |--- feature_24 >  0.14
|    |    |    |    |--- feature_24 <= 0.15
|    |    |    |    |    |--- class: M
|    |    |    |    |--- feature_24 >  0.15
|    |    |    |    |    |--- class: B
|    |    |--- feature_21 >  27.46
|    |    |    |--- feature_4 <= 0.09
|    |    |    |    |--- feature_9 <= 0.06
|    |    |    |    |    |--- class: M
|    |    |    |    |--- feature_9 >  0.06
|    |    |    |    |    |--- class: B
|    |    |    |--- feature_4 >  0.09
|    |    |    |    |--- class: M
|    |--- feature_22 >  117.45
|    |    |--- feature_27 <= 0.09
|    |    |    |--- feature_29 <= 0.06
|    |    |    |    |--- class: B
|    |    |    |--- feature_29 >  0.06
|    |    |    |    |--- class: M
|    |    |--- feature_27 >  0.09
|    |    |    |--- class: M
```

KNN Classifier

In [33]:

```
feat_train, feat_test, class_train, class_test = train_test_split(features,
                                    classes, test_size=0.2, random_state=9)
```

In [34]:

```
knn=KNeighborsClassifier(n_neighbors=4)
knn.fit(feat_train,class_train)
```

Out[34]:

```
KNeighborsClassifier(n_neighbors=4)
```

In [35]:

```
pred=knn.predict(feat_test)
print("Accuracy:",metrics.accuracy_score(class_test,pred))
```

```
Accuracy: 0.9210526315789473
```

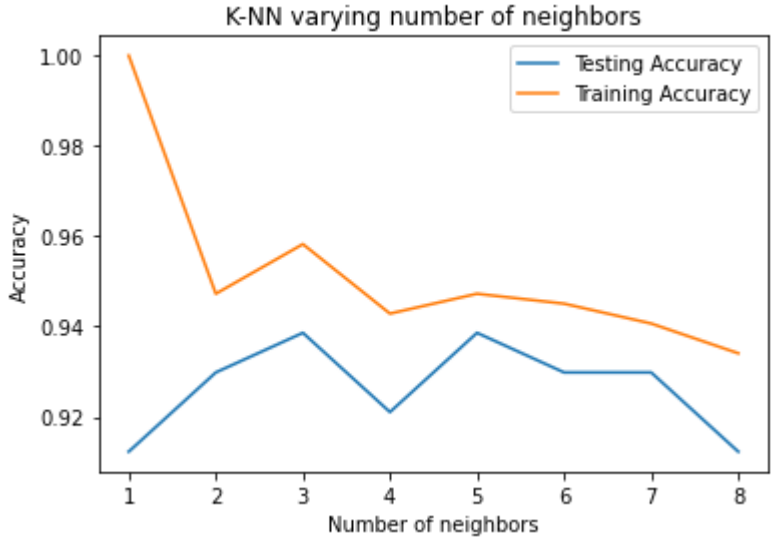
In [36]:

```
neighbors=np.arange(1,9)
train_accuracy=np.empty(len(neighbors))
test_accuracy=np.empty(len(neighbors))
for i,k in enumerate(neighbors):
    #setup as knn vlassifier with k neighbors
    knn1=KNeighborsClassifier(n_neighbors=k)
    #fit the model
    knn1.fit(feat_train,class_train)
    pred=knn1.predict(feat_test)
    #compute accuracy on the training set
    train_accuracy[i]=knn1.score(feat_train,class_train)
    #compute accuracy on the test set
    test_accuracy[i]=knn1.score(feat_test,class_test)
    print("Accuracy:",i,metrics.accuracy_score(class_test,pred))
print("train_accuracy\n",train_accuracy)
print("test_accuracy\n",test_accuracy)
```

```
Accuracy: 0 0.9122807017543859
Accuracy: 1 0.9298245614035088
Accuracy: 2 0.9385964912280702
Accuracy: 3 0.9210526315789473
Accuracy: 4 0.9385964912280702
Accuracy: 5 0.9298245614035088
Accuracy: 6 0.9298245614035088
Accuracy: 7 0.9122807017543859
train_accuracy
 [1.          0.94725275 0.95824176 0.94285714 0.94725275 0.94505495
 0.94065934 0.93406593]
test_accuracy
 [0.9122807  0.92982456 0.93859649 0.92105263 0.93859649 0.92982456
 0.92982456 0.9122807 ]
```

In [37]:

```
plt.title('K-NN varying number of neighbors')
plt.plot(neighbors,test_accuracy,label='Testing Accuracy')
plt.plot(neighbors,train_accuracy,label='Training Accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



Conclusion: From above graph we see training accuracy is more than that of testing accuracy

10 - SHREYA JAGADALE

Support Vector Machine

In [38]:

```
df.head()
```

Out[38]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	com
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 31 columns

In [39]:

```
df['diagnosis'].unique()
```

Out[39]:

```
array(['M', 'B'], dtype=object)
```

In [40]:

```
df['diagnosis'] = df['diagnosis'].map({'M':1,'B':0})
```

In [41]:

```
classess=df.diagnosis
classess
```

Out[41]:

```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

In [42]:

```
featuress = df.drop(['diagnosis'],axis=1)
featuress
```

Out[42]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0
...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0
565	20.13	28.25	131.20	1261.0	0.09780	0
566	16.60	28.08	108.30	858.1	0.08455	0
567	20.60	29.33	140.10	1265.0	0.11780	0
568	7.76	24.54	47.92	181.0	0.05263	0

569 rows × 30 columns

In [43]:

```
from sklearn import preprocessing
#get col names
names = featuress.columns
#create scaler object
scaler = preprocessing.StandardScaler()
#fit data on the scaler object
scaled_df = scaler.fit_transform(featuress)
featuress = pd.DataFrame(scaled_df,columns=names)
```

In [44]:

```
featuress
```

Out[44]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.2
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.4
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.0
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.4
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.5
...	...	...	...	...	...	
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.2
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.0
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.0
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.2
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.1

569 rows × 30 columns

In [45]:

```
feat_train, feat_test, class_train, class_test = train_test_split(featuress,
                                  classess, train_size=0.9, random_state=100)
```

In [46]:

```
svlassifier=SVC(kernel='linear')
svlassifier.fit(feat_train,class_train)
```

Out[46]:

```
SVC(kernel='linear')
```

In [47]:

```
print('features train shape: ', feat_train.shape)
print('classes train shape: ', class_train.shape)
print('features test shape: ', feat_test.shape)
print('classes test shape: ', class_test.shape)
```

```
features train shape: (512, 30)classes
train shape: (512,)
features test shape: (57, 30)
classes test shape: (57,)
```

In [48]:

```
pred=svlassifier.predict(feat_test)
```

In [49]:

```
accuracy_score(class_test,pred)
```

Out[49]:

```
0.9473684210526315
```

In [50]:

```
print(classification_report(class_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        35
           1       1.00      0.86      0.93        22

    accuracy                           0.95        57
   macro avg       0.96      0.93      0.94        57
weighted avg       0.95      0.95      0.95        57
```

In [51]:

```
cf=confusion_matrix(class_test,pred)
cf
```

Out[51]:

```
array([[35,  0],
       [ 3, 19]], dtype=int64)
```

In [52]:

```
kernels = ['linear','rbf','poly']
for kernel in kernels:
    sv = SVC(kernel=kernel).fit(feat_train,class_train)
    pred=sv.predict(feat_test)
    print("Accuracy:("+kernel+")", accuracy_score(class_test,pred))
```

```
Accuracy:(linear) 0.9473684210526315
Accuracy:(rbf) 0.9649122807017544
Accuracy:(poly) 0.8947368421052632
```

In [53]:

```
gammas = [0.1,1,10,100]
for gamma in gammas:
    sv = SVC(kernel='rbf', gamma=gamma).fit(feat_train,class_train)
    pred=sv.predict(feat_test)
    print("Accuracy:(", gamma , ")", accuracy_score(class_test,pred))
```

```
Accuracy:( 0.1 ) 0.9473684210526315
Accuracy:( 1 ) 0.6140350877192983
Accuracy:( 10 ) 0.6140350877192983
Accuracy:( 100 ) 0.6140350877192983
```

In [54]:

```
degrees = [0,1,2,3,4,5,20]
for degree in degrees:
    sv = SVC(kernel='poly', degree=degree).fit(feat_train,class_train)
    pred=sv.predict(feat_test)
    print("Accuracy:(", degree , "):", accuracy_score(class_test,pred))
```

```
Accuracy:( 0 ): 0.6140350877192983
Accuracy:( 1 ): 0.9473684210526315
Accuracy:( 2 ): 0.8771929824561403
Accuracy:( 3 ): 0.8947368421052632
Accuracy:( 4 ): 0.8596491228070176
Accuracy:( 5 ): 0.8596491228070176
Accuracy:( 20 ): 0.7543859649122807
```

Principal Component Analysis

In [57]:

```
pca = decomposition.PCA(n_components=3)
pca.fit(featuress)
X1 = pca.transform(featuress)
```

In [58]:

```
print(cancer.data.shape)
print(featuress.shape)
print(X1.shape)
```

```
(569, 30)
(569, 30)
(569, 3)
```

In [59]:

```
from sklearn.model_selection import train_test_split
(train_feat,test_feat,train_classes,test_classes)= train_test_split(featuress,
                                 classess,train_size=0.5,random_state=6)
dectree = DecisionTreeClassifier()
dectree.fit(train_feat,train_classes)
```

Out[59]:

```
DecisionTreeClassifier()
```

In [60]:

```
from sklearn import metrics
pred=dectree.predict(test_feat)
print("Accuracy:",metrics.accuracy_score(test_classes,pred))
```

```
Accuracy: 0.9157894736842105
```

In [61]:

```
from sklearn.model_selection import train_test_split
(train_feat,test_feat,train_classes,test_classes)= train_test_split(X1,classess,
                                    train_size=0.5,random_state=6)
dectree = DecisionTreeClassifier()
dectree.fit(train_feat,train_classes)
```

Out[61]:

```
DecisionTreeClassifier()
```

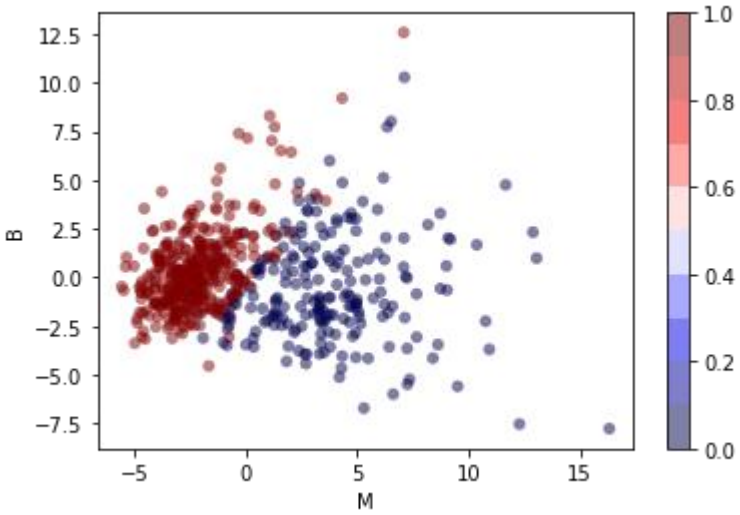
In [62]:

```
from sklearn import metrics
pred=dectree.predict(test_feat)
print("Accuracy:",metrics.accuracy_score(test_classes,pred))
```

```
Accuracy: 0.9298245614035088
```

In [63]:

```
plt.scatter(X1[:,0],X1[:,1],c=cancer.target,edgecolor='none',
            alpha=0.5,cmap=plt.cm.get_cmap('seismic',10))
plt.xlabel('M')
plt.ylabel('B')
plt.colorbar();
```



Select K Percentile and K Best

In [102]:

```
X_train,X_test,y_train,y_test = train_test_split(featuress,
                                classess,test_size=0.9,random_state=100)
```

In [103]:

```
select = SelectPercentile(percentile=20)
select.fit(X_train,y_train)
```

Out[103]:

```
SelectPercentile(percentile=20)
```

In [104]:

```
X_train_selected = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_selected.shape: {}".format(X_train_selected.shape))
```

```
X_train.shape: (56, 30)
X_train_selected.shape: (56, 6)
```

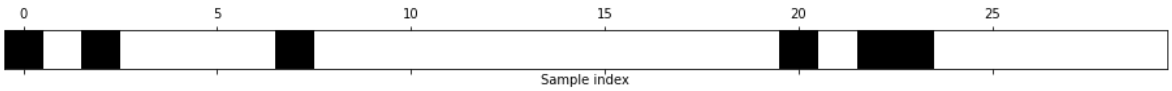
In [105]:

```
mask = select.get_support()
print(mask)
plt.matshow(mask.reshape(1,-1),cmap='gray_r')
plt.xlabel("Sample index")
plt.yticks(())
```

```
[ True False True False False False False True False False False False False
 False False False False False False False True False True True False False
 False False False False]
```

Out[105]:

```
([], [])
```



In [106]:

```
from sklearn.tree import DecisionTreeClassifier
X_test_selected = select.transform(X_test)
lr = DecisionTreeClassifier()
lr.fit(X_train,y_train)
print("Score with all features: {:.3f}".format(lr.score(X_test,y_test)))
lr.fit(X_train_selected,y_train)
print("Score with all features: {:.3f}".format(lr.score(X_test_selected,y_test)))
```

```
Score with all features: 0.854
Score with all features: 0.903
```

In [107]:

```
select=SelectKBest(k=2)
select.fit(X_train,y_train)
```

Out[107]:

```
SelectKBest(k=2)
```

In [108]:

```
X_train_selected = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_selected.shape: {}".format(X_train_selected.shape))
```

```
X_train.shape: (56, 30)
X_train_selected.shape: (56, 2)
```

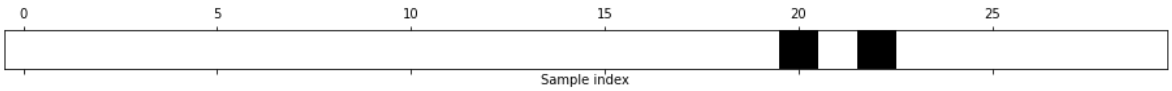
In [109]:

```
mask = select.get_support()
print(mask)
plt.matshow(mask.reshape(1,-1),cmap='gray_r')
plt.xlabel("Sample index")
plt.yticks(())
```

```
[False False False False False False False False False False False False False
 False False False False False False False True False True False False False
 False False False False]
```

Out[109]:

```
([], [])
```



Feature Seletion-Model Based

In [86]:

```
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
select = SelectFromModel(RandomForestClassifier(n_estimators=10,
                                                random_state=100),threshold="median")
#select = SelectFromModel(SVC(kernel='linear'))
```

In [87]:

```
select.fit(X_train,y_train)
X_train_l1 = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_l1.shape: {}".format(X_train_l1.shape))
```

```
X_train.shape: (56, 30)
X_train_l1.shape: (56, 15)
```

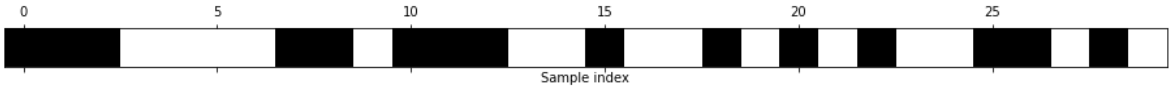
In [88]:

```
mask = select.get_support()
print(mask)
plt.matshow(mask.reshape(1,-1),cmap='gray_r')
plt.xlabel("Sample index")
plt.yticks(())
```

```
[ True True True False False False False True  True False  True  TrueTrue False
 False True False False True False True False  True False False True True False
 True False]
```

Out[88]:

```
([], [])
```



In [89]:

```
X_test_l1 = select.transform(X_test)
score = SVC().fit(X_train,y_train).score(X_test,y_test)
print("Test Score: {:.3f}".format(score))
score = SVC().fit(X_train_l1,y_train).score(X_test_l1,y_test)
print("Test Score: {:.3f}".format(score))
```

```
Test Score: 0.942
Test Score: 0.930
```

Iterative Feature Selection

In [120]:

```
from sklearn.feature_selection import RFE
select = RFE(RandomForestClassifier(n_estimators=20,
                    random_state=100),n_features_to_select=30)
select.fit(X_train,y_train)
```

Out[120]:

```
RFE(estimator=RandomForestClassifier(n_estimators=20, random_state=100),
    n_features_to_select=30)
```

mask = select.get_support() print(mask) plt.matshow(mask.reshape(1,-1),cmap='gray_r') plt.xlabel("Sampleindex") plt.yticks(())

In [88]:
```
from sklearn.linear_model import LogisticRegression
X_train_rfe =select.transform(X_train)
X_test_rfe = select.transform(X_test)
score = LogisticRegression().fit(X_train_rfe,y_train).score(X_test_rfe,y_test)
print("Test score: {:.3f}".format(score))
print("Test score: {:.3f}".format(select.score(X_test,y_test)))
```

```
Test score: 0.975
Test score: 0.932
```

Top 5 Accuracy:

Random Forest=0.97 ¶

SVM kernel(rbf)= 0.96

Multivariate Logistic Regression=0.95

Decision Tree Classifier entropy=0.95

KNN shvc=0.93