1-numpy

November 11, 2023

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

[2]:
```python
np.arange(19)
```

[2]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18])
```

[3]:
```python
a = [1,2,3,4,5]
print(a)
print(type(a))
```

```
[1, 2, 3, 4, 5]
<class 'list'>
```

[4]:
```python
a = np.array(a)
print(type(a))
print(a.dtype)
```

```
<class 'numpy.ndarray'>
int64
```

[5]:
```python
np.arange(-3,3,0.5, dtype=int)
```

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

[6]:
```python
np.arange(-3,3,0.5, dtype=float)
```

[6]:
```
array([-3. , -2.5, -2. , -1.5, -1. , -0.5,  0. ,  0.5,  1. ,  1.5,  2. ,
        2.5])
```

[7]:
```python
l = [i**2 for i in range(10)]
```

[8]:
```python
l
```

[8]:
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

[9]:
```python
l = range(100000)
%timeit [i**2 for i in l]
```

```
23.6 ms ± 95.4 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[10]:
```
a = np.arange(10000000)
%timeit a**2
```

```
49.9 ms ± 13.2 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

[11]:
```
a = [1,2,3,4,5]
a = np.array(a)
```

[12]:
```
a.ndim
```

[12]:
```
1
```

[13]:
```
a % 2
```

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

[14]:
```
a.shape
```

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

[15]:
```
b = np.array([[1,2,3],[4,5,6]])
```

[16]:
```
b
```

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

[17]:
```
b.ndim
```

[17]:
```
2
```

[18]:
```
b.shape
```

[18]:
```
(2, 3)
```

[19]:
```
b.shape[0], b.shape[1]
```

[19]:
```
(2, 3)
```

[20]:
```
import pandas as pd
```

[21]:
```
pd.DataFrame(b)
```

[21]:

	0	1	2
0	1	2	3
1	4	5	6

[22]:
```
x = [[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]]
```

[23]:
```
x = np.array(x)
```

[24]:
```
x.ndim
```

[24]:
```
3
```

[25]:
```
d = list(map(int, input().split()))
```

```
5
```

[26]:
```
n, m = list(map(int, input("Enter the number of rows and column: ").split(",")))
A = np.array([input(f"Row{i+1}: ").split(",")[:m] for i in range(n)], int)
print(A)
type(A)
```

```
Enter the number of rows and column: 2,3
Row1: 1
Row2: 2
[[1]
 [2]]
```

[26]:
```
numpy.ndarray
```

[27]:
```
b = np.linspace(1,4,10)
```

[28]:
```
c = np.ones((4,5))
c
```

[28]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

[29]:
```
d = np.zeros((4,4))
```

[30]:
```
e = np.eye(3,3)
```

[31]:
```
np.diag(e)
```

[31]:
```
array([1., 1., 1.])
```

[32]:
```
n = int(input(f'Enter "nth" number: '))
g = np.random.rand(n)
print(g)
```

```
Enter "nth" number: 10
[0.96016601 0.49857984 0.93630255 0.22383969 0.67499168 0.92859701
 0.13253047 0.05318771 0.78824979 0.57813896]
```

[33]:
```python
h = np.random.rand(3)
h
h.dtype
```

[33]:
```
dtype('float64')
```

[34]:
```python
c = np.array([1+2j,3+4j])
print(c)
print(c.dtype)
```

```
[1.+2.j 3.+4.j]
complex128
```

[35]:
```python
b = np.array([True,True,False,False])
print(b)
print(b.dtype)
```

```
[ True  True False False]
bool
```

[36]:
```python
a = np.diag([1,2,3,4])
print(a)
a[2:4] = 5
print(a)
```

```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
[[1 0 0 0]
 [0 2 0 0]
 [5 5 5 5]
 [5 5 5 5]]
```

a[5:] = 1000

[37]:
```python
a
```

[37]:
```
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [5, 5, 5, 5],
       [5, 5, 5, 5]])
```

[38]:
```python
a[0] = 1000
```

[39]:
```
c = a[::2].copy()
```

[40]:
```
c
```

[40]:
```
array([[1000, 1000, 1000, 1000],
       [   5,    5,    5,    5]])
```

[41]:
```
np.shares_memory(a,c)
```

[41]:
```
False
```

[42]:
```
b = np.random.randint(20,30,10)
```

[ ]:
```

```

2-pandas

November 11, 2023

[8]:
```python
import pandas as pd
import matplotlib.pyplot as plt
```

[9]:
```python
p = pd.read_csv("Salary_Data.csv")
```

[10]:
```python
p
```

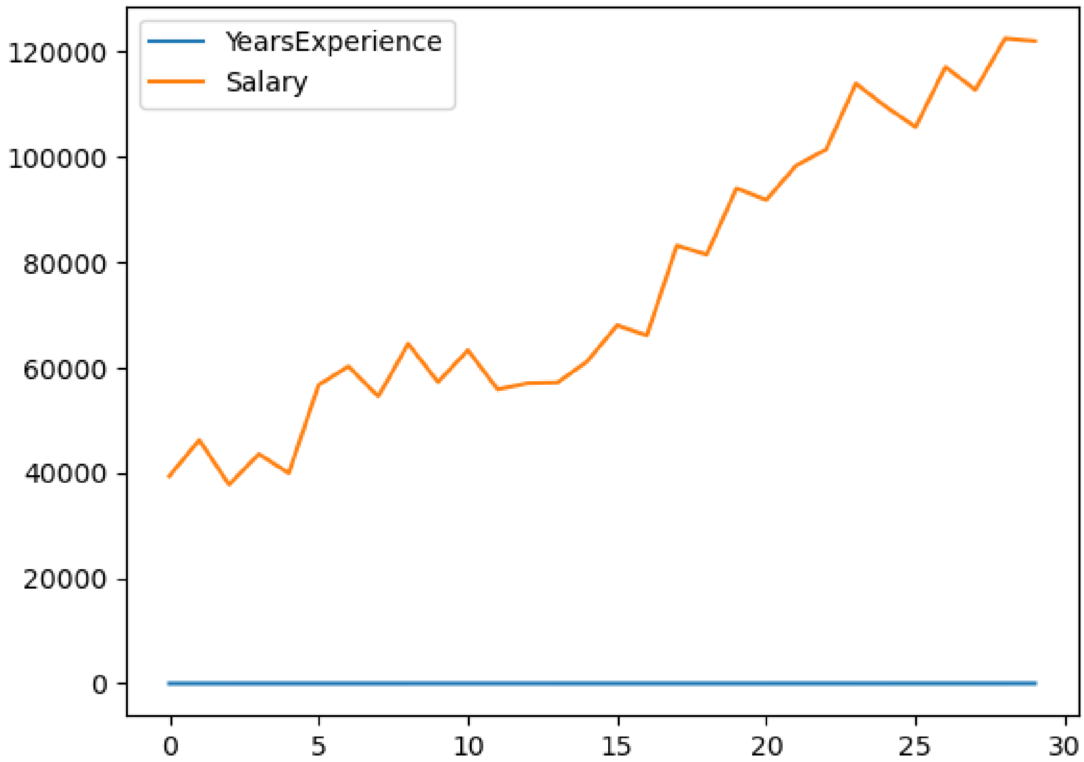
[10]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0

```
29                10.5   121872.0
```

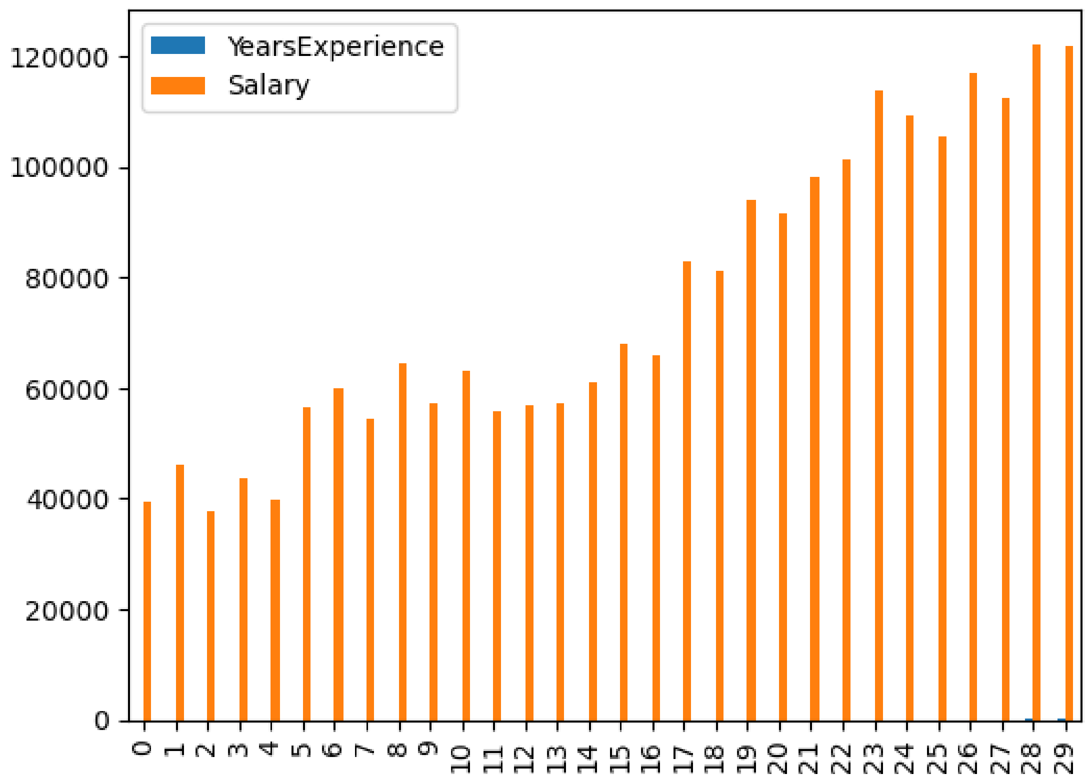
[11]:
```
p.plot()
```

[11]: <Axes: >



[15]:
```
p.plot.bar()
```

[15]: <Axes: >



[136]:
```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

[4]:
```
ser_1 = Series([1,1,2,-3,-5,8,13])
ser_1
```

[4]:
```
0     1
1     1
2     2
3    -3
4    -5
5     8
6    13
dtype: int64
```

[5]:
```
ser_1.values
```

[5]:
```
array([ 1,  1,  2, -3, -5,  8, 13], dtype=int64)
```

[6]:
```
ser_1.index
```

[6]: RangeIndex(start=0, stop=7, step=1)

[7]:
```
s2 = Series([1,1,2,-3,-5], index=['a','b','c','d','e'])
s2
```

[7]:
```
a    1
b    1
c    2
d   -3
e   -5
dtype: int64
```

[8]:
```
s2['a']
```

[8]: 1

[9]:
```
s2[4] == s2['e']
```

[9]: True

[10]:
```
s2[['c','a','b']]
```

[10]:
```
c    2
a    1
b    1
dtype: int64
```

[11]:
```
s2
```

[11]:
```
a    1
b    1
c    2
d   -3
e   -5
dtype: int64
```

[12]:
```
s2>0
```

[12]:
```
a     True
b     True
c     True
d    False
e    False
dtype: bool
```

[13]:
```
s2[s2>0]
```

[13]:
```
a    1
b    1
c    2
dtype: int64
```

[14]:
```
s2*2
```

[14]:
```
a     2
b     2
c     4
d    -6
e   -10
dtype: int64
```

[15]:
```
np.exp(s2)
```

[15]:
```
a    2.718282
b    2.718282
c    7.389056
d    0.049787
e    0.006738
dtype: float64
```

[16]:
```
d1 = {'foo':100,'bar':200,'baz':300}
s3 = Series(d1)
s3
```

[16]:
```
foo    100
bar    200
baz    300
dtype: int64
```

[18]:
```
index = ['foo', 'bar', 'baz','qux']
s4 = Series(d1, index=index)
s4
```

[18]:
```
foo    100.0
bar    200.0
baz    300.0
qux      NaN
dtype: float64
```

[19]:
```
pd.isnull(s4).sum()
```

[19]:
```
1
```

[20]:
```
s4.isnull()
```

[20]:
```
foo     False
bar     False
baz     False
qux      True
dtype: bool
```

[21]:
```python
s3 + s4
```

[21]:
```
bar    400.0
baz    600.0
foo    200.0
qux      NaN
dtype: float64
```

[23]:
```python
s4.name = 'foobarbazqux'
```

[24]:
```python
s4.index.name = 'label'
```

[25]:
```python
s4
```

[25]:
```
label
foo    100.0
bar    200.0
baz    300.0
qux      NaN
Name: foobarbazqux, dtype: float64
```

[26]:
```python
s4.index = ['fo','br','bz','qx']
```

[27]:
```python
s4
```

[27]:
```
fo    100.0
br    200.0
bz    300.0
qx      NaN
Name: foobarbazqux, dtype: float64
```

[28]:
```python
#DataFrame
```

[32]:
```python
da1 = {'state' : ['VA','VA','VA', 'MD', 'MD'],
                  'year' : [2012,2013,2014,2014,2015],
                  'pop' : [5.0,5.1,5.2,4.0,4.1]}
```

[34]:
```python
df1 = DataFrame(da1)
print(da1)
df1
```

```
{'state': ['VA', 'VA', 'VA', 'MD', 'MD'], 'year': [2012, 2013, 2014, 2014,
2015], 'pop': [5.0, 5.1, 5.2, 4.0, 4.1]}
```

[34]:

	state	year	pop
0	VA	2012	5.0
1	VA	2013	5.1
2	VA	2014	5.2
3	MD	2014	4.0
4	MD	2015	4.1

[35]:

```python
df1.describe()
```

[35]:

	year	pop
count	5.000000	5.000000
mean	2013.600000	4.680000
std	1.140175	0.580517
min	2012.000000	4.000000
25%	2013.000000	4.100000
50%	2014.000000	5.000000
75%	2014.000000	5.100000
max	2015.000000	5.200000

[137]:

```python
df2= DataFrame(da1, columns=['year','state','pop'])
df2
```

[137]:

	year	state	pop
0	2012	VA	5.0
1	2013	VA	5.1
2	2014	VA	5.2
3	2014	MD	4.0
4	2015	MD	4.1

[138]:

```python
df3 = DataFrame(da1, columns=['year','state','pop', 'unempl'])
df3
```

[138]:

	year	state	pop	unempl
0	2012	VA	5.0	NaN
1	2013	VA	5.1	NaN
2	2014	VA	5.2	NaN
3	2014	MD	4.0	NaN
4	2015	MD	4.1	NaN

[139]:

```python
df3['state']
```

[139]:

```
0    VA
1    VA
2    VA
```

```
3    MD
4    MD
Name: state, dtype: object
```

[141]:
```python
df3['year']
```

[141]:
```
0    2012
1    2013
2    2014
3    2014
4    2015
Name: year, dtype: int64
```

[142]:
```python
df3.iloc[0]
```

[142]:
```
year      2012
state       VA
pop        5.0
unempl     NaN
Name: 0, dtype: object
```

[143]:
```python
df3['unempl'] = np.arange(5)
df3
```

[143]:
```
   year state  pop  unempl
0  2012    VA  5.0       0
1  2013    VA  5.1       1
2  2014    VA  5.2       2
3  2014    MD  4.0       3
4  2015    MD  4.1       4
```

[62]:
```python
uempl = Series([6.0,6.0,6.1],index=[2,3,4])
df3['unempl'] = uempl
df3
```

[62]:
```
   year state  pop  unempl
0  2012    VA  5.0     NaN
1  2013    VA  5.1     NaN
2  2014    VA  5.2     6.0
3  2014    MD  4.0     6.0
4  2015    MD  4.1     6.1
```

[63]:
```python
df3['state_dp'] = df3['state']
df3
```

[63]:
```
   year state  pop  unempl state_dp
0  2012    VA  5.0     NaN       VA
```

```
1  2013     VA  5.1     NaN        VA
2  2014     VA  5.2     6.0        VA
3  2014     MD  4.0     6.0        MD
4  2015     MD  4.1     6.1        MD
```

[66]:
```python
del df3['state_dp']
df3
```

[66]:
```
   year state  pop  unempl
0  2012    VA  5.0     NaN
1  2013    VA  5.1     NaN
2  2014    VA  5.2     6.0
3  2014    MD  4.0     6.0
4  2015    MD  4.1     6.1
```

[68]:
```python
pop = {'VA' : {2013 : 5.1, 2014 :5.2},
       'MD' : {2014 :4.0, 2015 : 4.1}}
```

[69]:
```python
df4 = DataFrame(pop)
df4
```

[69]:
```
       VA   MD
2013  5.1  NaN
2014  5.2  4.0
2015  NaN  4.1
```

[70]:
```python
df4.T
```

[70]:
```
    2013  2014  2015
VA   5.1   5.2   NaN
MD   NaN   4.0   4.1
```

[73]:
```python
da2 = {'VA' : df4['VA'][1:],
      'MD' : df4['MD'][2:]}
df5 = DataFrame(da2)
df5
```

[73]:
```
       VA   MD
2014  5.2  NaN
2015  NaN  4.1
```

[75]:
```python
df5.index.name = 'year'
df5
```

[75]:
```
       VA   MD
year
2014  5.2  NaN
```

```
2015  NaN  4.1
```

[76]:
```python
df5.columns.name = 'state'
df5
```

[76]:
```
state   VA   MD
year
2014   5.2  NaN
2015   NaN  4.1
```

[77]:
```python
df5.values
```

[77]:
```
array([[5.2, nan],
       [nan, 4.1]])
```

[78]:
```python
df3.values
```

[78]:
```
array([[2012, 'VA', 5.0, nan],
       [2013, 'VA', 5.1, nan],
       [2014, 'VA', 5.2, 6.0],
       [2014, 'MD', 4.0, 6.0],
       [2015, 'MD', 4.1, 6.1]], dtype=object)
```

[79]:
```python
df3
```

[79]:

	year	state	pop	unempl
0	2012	VA	5.0	NaN
1	2013	VA	5.1	NaN
2	2014	VA	5.2	6.0
3	2014	MD	4.0	6.0
4	2015	MD	4.1	6.1

[81]:
```python
df3.reindex(list(reversed(range(0,7))), fill_value = 0)
```

[81]:

	year	state	pop	unempl
6	0	0	0.0	0.0
5	0	0	0.0	0.0
4	2015	MD	4.1	6.1
3	2014	MD	4.0	6.0
2	2014	VA	5.2	6.0
1	2013	VA	5.1	NaN
0	2012	VA	5.0	NaN

[82]:
```python
df3.reindex(range(6,0), fill_value=0)
```

[82]:
```
Empty DataFrame
Columns: [year, state, pop, unempl]
```

```
Index: []
```

[88]:
```python
s5 = Series(['foo','bar','baz'], index=[0,2,4])
s5
```

[88]:
```
0    foo
2    bar
4    baz
dtype: object
```

[89]:
```python
s5.reindex(range(5),method='ffill')
```

[89]:
```
0    foo
1    foo
2    bar
3    bar
4    baz
dtype: object
```

[90]:
```python
s5.reindex(range(5),method='bfill')
```

[90]:
```
0    foo
1    bar
2    bar
3    baz
4    baz
dtype: object
```

[91]:
```python
df3.reindex(columns=['state','pop','unempl','year'])
```

[91]:
```
  state  pop  unempl  year
0    VA  5.0     NaN  2012
1    VA  5.1     NaN  2013
2    VA  5.2     6.0  2014
3    MD  4.0     6.0  2014
4    MD  4.1     6.1  2015
```

[110]:
```python
df3.reindex(index = list(reversed(range(0,6))), fill_value =␣
↪0,columns=['state','pop','unempl','year'])
```

[110]:
```
  state  pop  unempl  year
5     0  0.0     0.0     0
4    MD  4.1     6.1  2015
3    MD  4.0     6.0  2014
2    VA  5.2     6.0  2014
1    VA  5.1     NaN  2013
0    VA  5.0     NaN  2012
```

[168]:
```python
df3.shape
```

[168]:
```
(5, 4)
```

[162]:
```python
#As dataframe have 5 rows, we will change the range from (0,7) to (0,5)
df6 = df3.loc[range(0,5),['state','pop','unempl','year']]
df6
```

[162]:

	state	pop	unempl	year
0	VA	5.0	0	2012
1	VA	5.1	1	2013
2	VA	5.2	2	2014
3	MD	4.0	3	2014
4	MD	4.1	4	2015

[170]:
```python
#We can use reindex() as an alternative
df6 = df3.reindex(index = list(range(0,7)),  columns= ␣
 ↪['state','pop','unempl','year'])
df6
```

[170]:

	state	pop	unempl	year
0	VA	5.0	0.0	2012.0
1	VA	5.1	1.0	2013.0
2	VA	5.2	2.0	2014.0
3	MD	4.0	3.0	2014.0
4	MD	4.1	4.0	2015.0
5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN

[112]:
```python
df7 = df6.drop(['unempl','pop'], axis=1)
df7
```

[112]:

	state	year
0	VA	2012.0
1	VA	2013.0
2	VA	2014.0
3	MD	2014.0
4	MD	2015.0
5	NaN	NaN
6	NaN	NaN

[115]:
```python
s2
```

[115]:
```
a    1
b    1
c    2
d   -3
```

```
e   -5
dtype: int64
```

[116]:
```
s2[0] ==s2['a']
```

[116]:
```
True
```

[117]:
```
s2[1:4]
```

[117]:
```
b    1
c    2
d   -3
dtype: int64
```

[119]:
```
s2[['b','c','d']]
```

[119]:
```
b    1
c    2
d   -3
dtype: int64
```

[120]:
```
s2[s2>0]
```

[120]:
```
a    1
b    1
c    2
dtype: int64
```

[121]:
```
s2['a':'b']
```

[121]:
```
a    1
b    1
dtype: int64
```

[122]:
```
s2['a':'b'] = 0
s2
```

[122]:
```
a    0
b    0
c    2
d   -3
e   -5
dtype: int64
```

[123]:
```
df6
```

[123]:

	state	pop	unempl	year
0	VA	5.0	NaN	2012.0
1	VA	5.1	NaN	2013.0
2	VA	5.2	6.0	2014.0
3	MD	4.0	6.0	2014.0
4	MD	4.1	6.1	2015.0
5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN

[124]:
```
df6[['pop','unempl']]
```

[124]:

	pop	unempl
0	5.0	NaN
1	5.1	NaN
2	5.2	6.0
3	4.0	6.0
4	4.1	6.1
5	NaN	NaN
6	NaN	NaN

[125]:
```
df6[:3]
```

[125]:

	state	pop	unempl	year
0	VA	5.0	NaN	2012.0
1	VA	5.1	NaN	2013.0
2	VA	5.2	6.0	2014.0

[126]:
```
df6[df6['pop']>5]
```

[126]:

	state	pop	unempl	year
1	VA	5.1	NaN	2013.0
2	VA	5.2	6.0	2014.0

[134]:
```
df6
```

[134]:

	state	pop	unempl	year
0	VA	5.0	NaN	2012.0
1	VA	5.1	NaN	2013.0
2	VA	5.2	6.0	2014.0
3	MD	4.0	6.0	2014.0
4	MD	4.1	6.1	2015.0
5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN

[148]:
```
df8 = df6.drop(['state'], axis = 1)
```

[149]:
```
df8
```

[149]:
```
   pop  unempl    year
0  5.0     NaN  2012.0
1  5.1     NaN  2013.0
2  5.2     6.0  2014.0
3  4.0     6.0  2014.0
4  4.1     6.1  2015.0
5  NaN     NaN     NaN
6  NaN     NaN     NaN
```

[150]:
```
df8 > 5
```

[150]:
```
     pop  unempl   year
0  False   False   True
1   True   False   True
2   True    True   True
3  False    True   True
4  False    True   True
5  False   False  False
6  False   False  False
```

[151]:
```
df6.iloc[2:6]
```

[151]:
```
  state  pop  unempl    year
2    VA  5.2     6.0  2014.0
3    MD  4.0     6.0  2014.0
4    MD  4.1     6.1  2015.0
5   NaN  NaN     NaN     NaN
```

[152]:
```
df6.loc[0:2,'pop']
```

[152]:
```
0    5.0
1    5.1
2    5.2
Name: pop, dtype: float64
```

[154]:
```
np.random.seed(1)
ser_7 = Series (np.random.randn(5),
index=['a', 'c', 'e', 'f', 'g'])
ser_7
```

[154]:
```
a    1.624345
c   -0.611756
e   -0.528172
f   -1.072969
g    0.865408
dtype: float64
```

[157]:
```
np.random.seed(0)
ser_6 = Series (np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
ser_6
```

[157]:
```
a    1.764052
b    0.400157
c    0.978738
d    2.240893
e    1.867558
dtype: float64
```

[158]:
```
ser_6 + ser_7
```

[158]:
```
a    3.388398
b         NaN
c    0.366982
d         NaN
e    1.339386
f         NaN
g         NaN
dtype: float64
```

[159]:
```
ser_6.add(ser_7, fill_value=0)
```

[159]:
```
a    3.388398
b    0.400157
c    0.366982
d    2.240893
e    1.339386
f   -1.072969
g    0.865408
dtype: float64
```

[ ]:
```

```

3-movie-rating-analysis

November 11, 2023

[ ]:
```
import numpy as np
import pandas as pd
movies = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/ML_LAB/Datasets/
↪movies.dat", delimiter='::')
print(movies.head())
```

```
<ipython-input-4-216480929c41>:3: ParserWarning: Falling back to the 'python'
engine because the 'c' engine does not support regex separators (separators > 1
char and different from '\s+' are interpreted as regex); you can avoid this
warning by specifying engine='python'.
  movies = pd.read_csv("/content/drive/MyDrive/Colab
Notebooks/ML_LAB/Datasets/movies.dat", delimiter='::')
```

```
   0000008      Edison Kinetoscopic Record of a Sneeze (1894)  \
0       10                  La sortie des usines Lumière (1895)
1       12                        The Arrival of a Train (1896)
2       25  The Oxford and Cambridge University Boat Race …
3       91                           Le manoir du diable (1896)
4      131                             Une nuit terrible (1896)

     Documentary|Short
0    Documentary|Short
1    Documentary|Short
2                  NaN
3         Short|Horror
4   Short|Comedy|Horror
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
movies.columns = ["ID", "Title", "Genre"]
print(movies.head())
```

```
    ID                                              Title              Genre
0   10                  La sortie des usines Lumière (1895)    Documentary|Short
1   12                        The Arrival of a Train (1896)    Documentary|Short
2   25  The Oxford and Cambridge University Boat Race …                  NaN
```

```
3   91                          Le manoir du diable (1896)         Short|Horror
4  131                            Une nuit terrible (1896)  Short|Comedy|Horror
```

[ ]:
```
ratings = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/ML_LAB/Datasets/
↪ratings.dat", delimiter='::')
print(ratings.head())
```

```
<ipython-input-6-e60cce7f4c75>:1: ParserWarning: Falling back to the 'python'
engine because the 'c' engine does not support regex separators (separators > 1
char and different from '\s+' are interpreted as regex); you can avoid this
warning by specifying engine='python'.
  ratings = pd.read_csv("/content/drive/MyDrive/Colab
Notebooks/ML_LAB/Datasets/ratings.dat", delimiter='::')
```

```
   1  0114508  8  1381006850
0  2   499549  9  1376753198
1  2  1305591  8  1376742507
2  2  1428538  1  1371307089
3  3    75314  1  1595468524
4  3   102926  9  1590148016
```

[ ]:
```
ratings.columns = ["User", "ID", "Ratings", "Timestamp"]
print(ratings.head())
```

```
   User       ID  Ratings   Timestamp
0     2   499549        9  1376753198
1     2  1305591        8  1376742507
2     2  1428538        1  1371307089
3     3    75314        1  1595468524
4     3   102926        9  1590148016
```

[ ]:
```
data = pd.merge(movies, ratings, on=["ID", "ID"])
print(data.head())
```

```
   ID                                              Title               Genre  \
0  10                 La sortie des usines Lumière (1895)  Documentary|Short
1  12                       The Arrival of a Train (1896)  Documentary|Short
2  25  The Oxford and Cambridge University Boat Race …                 NaN
3  91                          Le manoir du diable (1896)       Short|Horror
4  91                          Le manoir du diable (1896)       Short|Horror

    User  Ratings   Timestamp
0  70577       10  1412878553
1  69535       10  1439248579
2  37628        8  1488189899
3   5814        6  1385233195
4  37239        5  1532347349
```

[ ]:
```
ratings = data["Ratings"].value_counts()
numbers = ratings.index
quantity = ratings.values
import plotly.express as px
fig = px.pie(data, values=quantity, names=numbers)
fig.show()
```

[ ]:
```
print(data["Title"].value_counts().head(10))
```

```
Gravity (2013)                      3104
Interstellar (2014)                 2948
1917 (2019)                         2879
The Wolf of Wall Street (2013)      2836
Joker (2019)                        2753
Man of Steel (2013)                 2694
World War Z (2013)                  2429
Iron Man Three (2013)               2417
Now You See Me (2013)               2379
Gone Girl (2014)                    2284
Name: Title, dtype: int64
```

[ ]:
```
print(movies)
```

```
             ID                                              Title  \
0            10                  La sortie des usines Lumière (1895)
1            12                        The Arrival of a Train (1896)
2            25  The Oxford and Cambridge University Boat Race …
3            91                          Le manoir du diable (1896)
4           131                            Une nuit terrible (1896)
…           …                                                  …
37336  14499632                                   22 vs. Earth (2021)
37337  14527836                                       Recalled (2021)
37338  14544192                             Bo Burnham: Inside (2021)
37339  14735160                              Mum is Pregnant (2021)
37340  14740904                               Juanes: Origen (2021)

                          Genre
0            Documentary|Short
1            Documentary|Short
2                          NaN
3                 Short|Horror
4          Short|Comedy|Horror
…                          …
37336  Animation|Short|Adventure
37337     Drama|Mystery|Thriller
37338        Comedy|Drama|Music
37339                      NaN
37340                Documentary
```

```
[37341 rows x 3 columns]
```

[ ]:
```

```

[ ]:
```

```

4-decision-tree-on-iris

November 11, 2023

0.1 Prediction Using Decision Tree Algorithm

0.2 Create The Decision Tree Classifier and Visualze it Graphically

0.2.1 Link to Dataset:https://bit.ly/3kXTdox

Import the reqired libraries

[100]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[100]:
```

```

[101]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[102]:
```
df=pd.read_csv('/content/drive/MyDrive/ML 385/iris.csv')
```

[103]:
```
df.head()
```

[103]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[104]:
```
df.tail()
```

[104]:

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica

```
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica
```

[105]:
```python
df.shape
```

[105]:
```
(150, 5)
```

[106]:
```python
df.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   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Findiang missing value

[107]:
```python
df.isnull().sum()
```

[107]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

[108]:
```python
df.describe()
```

[108]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
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

[109]:
```python
df['species'].unique()
```

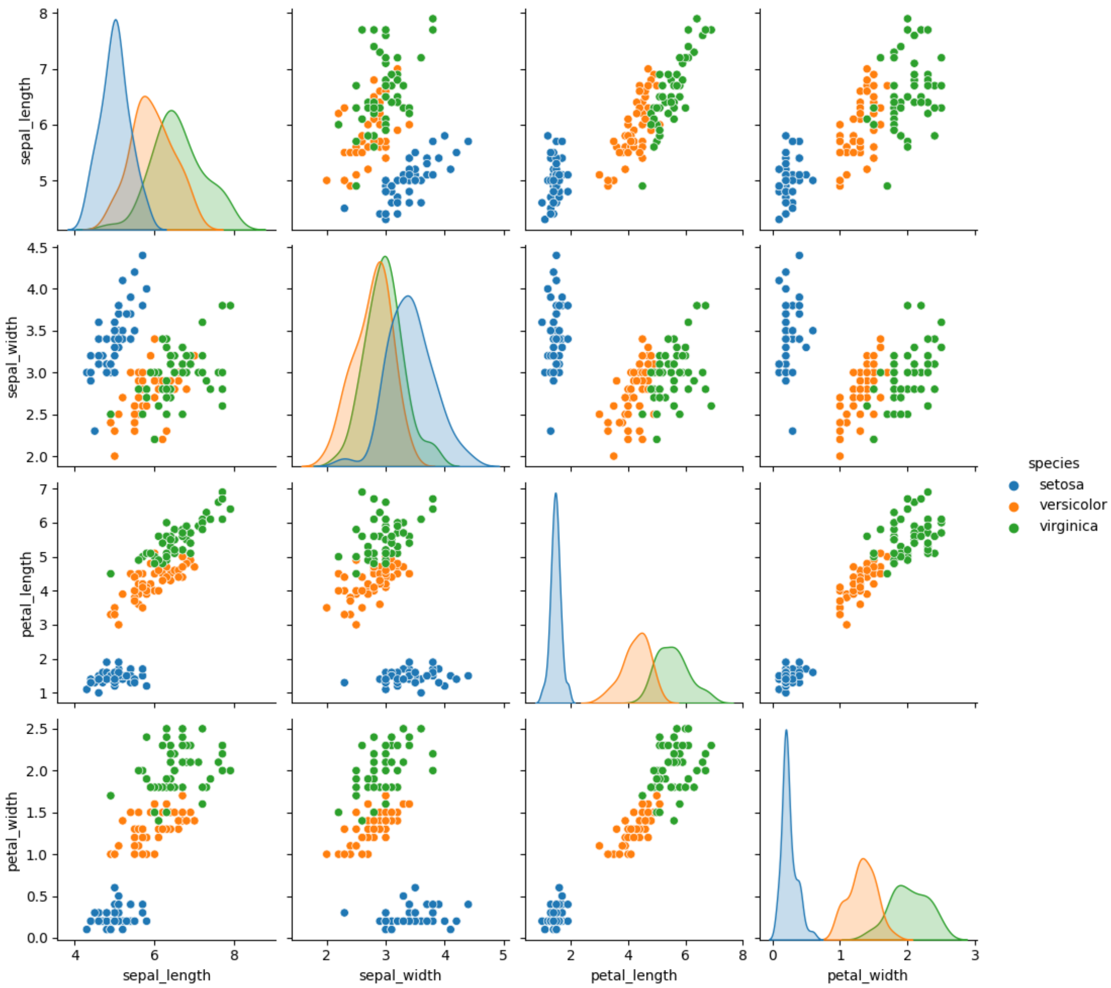
[109]:
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

[110]:
```
df['species'].value_counts()
```

[110]:
```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

[111]:
```
sns.pairplot(df,hue='species')
```

[111]: <seaborn.axisgrid.PairGrid at 0x7de723152a10>



[112]:
```
df.corr()
```

[112]:

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544

```
petal_length      0.871754    -0.420516      1.000000      0.962757
petal_width       0.817954    -0.356544      0.962757      1.000000
```

[113]:
```
X=df.drop(['species'],axis=1)
# y contain target column
y=df['species']
```

[114]:
```
X.shape
```

[114]:
```
(150, 4)
```

[115]:
```
y.shape
```

[115]:
```
(150,)
```

[116]:
```
from sklearn.model_selection import train_test_split
```

[117]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

[118]:
```
from sklearn.tree import DecisionTreeClassifier

DTC=DecisionTreeClassifier ()
```

[119]:
```
DTC.fit(X_train,y_train)
```

[119]:
```
DecisionTreeClassifier()
```

[120]:
```
prediction=DTC.predict(X_test)
```

[121]:
```
prediction
```

[121]:
```
array(['setosa', 'versicolor', 'virginica', 'versicolor', 'setosa',
       'virginica', 'versicolor', 'versicolor', 'setosa', 'versicolor',
       'versicolor', 'setosa', 'versicolor', 'setosa', 'virginica',
       'versicolor', 'virginica', 'setosa', 'setosa', 'setosa',
       'versicolor', 'virginica', 'setosa', 'virginica', 'versicolor',
       'versicolor', 'virginica', 'virginica', 'setosa', 'versicolor'],
      dtype=object)
```

[122]:
```
compare=pd.DataFrame({'Actual':y_test,'Prediction':prediction})
compare
```

[122]:

	Actual	Prediction
43	setosa	setosa
72	versicolor	versicolor
134	virginica	virginica
73	versicolor	versicolor

```
6        setosa      setosa
137   virginica   virginica
89   versicolor  versicolor
91   versicolor  versicolor
16       setosa      setosa
58   versicolor  versicolor
50   versicolor  versicolor
8        setosa      setosa
81   versicolor  versicolor
11       setosa      setosa
117   virginica   virginica
99   versicolor  versicolor
127   virginica   virginica
15       setosa      setosa
21       setosa      setosa
40       setosa      setosa
84   versicolor  versicolor
114   virginica   virginica
31       setosa      setosa
118   virginica   virginica
129   virginica  versicolor
74   versicolor  versicolor
107   virginica   virginica
131   virginica   virginica
20       setosa      setosa
55   versicolor  versicolor
```

[123]:

```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

[124]:

```
print(classification_report(y_test,prediction))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.92      1.00      0.96        11
   virginica       1.00      0.89      0.94         9

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

[125]:

```
Accuracy = accuracy_score(y_test,prediction)
# Precision = sklearn.metrics.precision_score(actual, predicted)
```

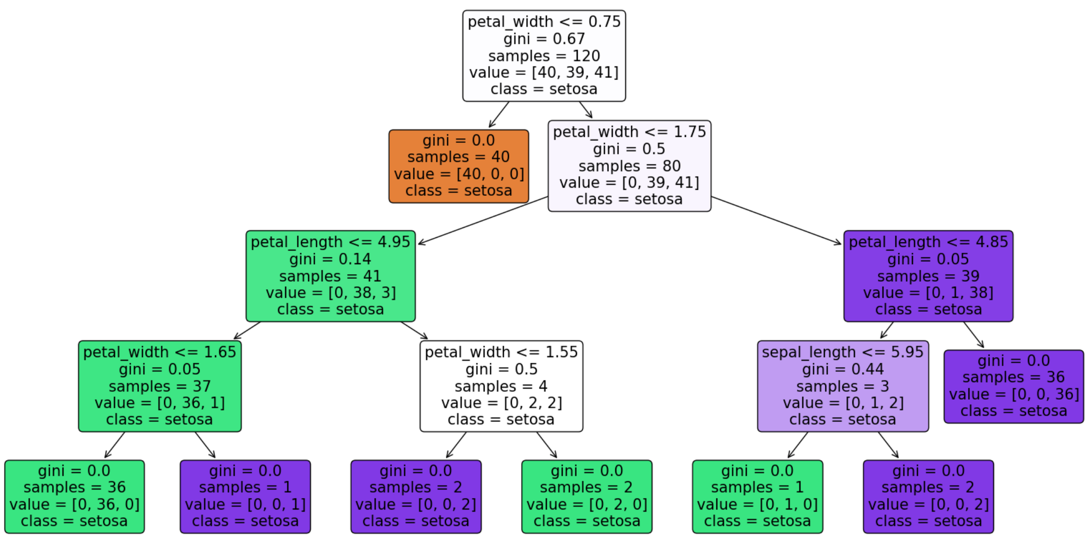
```
Accuracy
```

[125]: 0.9666666666666667

[126]:
```
# actual=(y_test==prediction).sum()
# prediction
Precision = precision_score(y_test, prediction,average='weighted')
Precision
```

[126]: 0.9694444444444444

[127]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(20,10))
tree=plot_tree(DTC,feature_names=X.
 ↪columns,precision=2,rounded=True,filled=True,class_names=y.values)
```



[127]:
```

```

5-dt-on-play-tennis

November 11, 2023

[1]:
```
import numpy as np
import pandas as pd
from pandas import Series,DataFrame
from sklearn import metrics
```

[3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[6]:
```
# reading the data
```

[4]:
```
data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/ML_LAB/Datasets/Play␣
 ↪Tennis.csv")
```

[5]:
```
data.head()
```

[5]:
```
  Day   Outlook Temprature Humidity    Wind Play_Tennis
0  D1     Sunny        Hot     High    Weak          No
1  D2     Sunny        Hot     High  Strong          No
2  D3  Overcast        Hot     High    Weak         Yes
3  D4      Rain       Mild     High    Weak         Yes
4  D5      Rain       Cool   Normal    Weak         Yes
```

[7]:
```
data.tail()
```

[7]:
```
    Day   Outlook Temprature Humidity    Wind Play_Tennis
9   D10      Rain       Mild   Normal    Weak         Yes
10  D11     Sunny       Mild   Normal  Strong         Yes
11  D12  Overcast       Mild     High  Strong         Yes
12  D13  Overcast        Hot   Normal    Weak         Yes
13  D14      Rain       Mild     High  Strong          No
```

[8]:
```
data.shape
```

[8]:
```
(14, 6)
```

[9]:
```
data.describe()
```

[9]:
```
        Day Outlook Temprature Humidity  Wind Play_Tennis
count    14      14         14       14    14          14
unique   14       3          3        2     2           2
top      D1   Sunny       Mild     High  Weak         Yes
freq      1       5          6        7     8           9
```

[10]:
```
# preparing the data
from sklearn import preprocessing
string_to_int = preprocessing.LabelEncoder()
data = data.apply(string_to_int.fit_transform)
```

[11]:
```
print(data)
```

```
    Day  Outlook  Temprature  Humidity  Wind  Play_Tennis
0     0        2           1         0     1            0
1     6        2           1         0     0            0
2     7        0           1         0     1            1
3     8        1           2         0     1            1
4     9        1           0         1     1            1
5    10        1           0         1     0            0
6    11        0           0         1     0            1
7    12        2           2         0     1            0
8    13        2           0         1     1            1
9     1        1           2         1     1            1
10    2        2           2         1     0            1
11    3        0           2         0     0            1
12    4        0           1         1     1            1
13    5        1           2         0     0            0
```

[12]:
```
# required imports
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

[13]:
```
features = ["Outlook","Temprature","Humidity","Wind"]
X = data[features]
y = data.Play_Tennis
```

[14]:
```
# splitting the data
train_X,val_X,train_y,val_y = train_test_split(X,y,train_size=0.7,test_size=0.
↪3,random_state=1)
```

[15]:
```
# model training
tennis_model = DecisionTreeClassifier(criterion="entropy",random_state=100)
tennis_model.fit(train_X,train_y)
```

[15]:
```
DecisionTreeClassifier(criterion='entropy', random_state=100)
```

[16]:
```
# prediciton of the data
prediction = tennis_model.predict(val_X)
```

[17]:
```
data_2 = {'Outlook' : ['2'], 'Temprature' : ['1'], 'Humidity' : ['0'], 'Wind' :␣
↪['1']}
df_2 = DataFrame(data_2)
df_2
```

[17]:

	Outlook	Temprature	Humidity	Wind
0	2	1	0	1

[18]:
```
y_pred = tennis_model.predict(df_2)
```

[19]:
```
y_pred
```

[19]:
```
array([0])
```

[20]:
```
# metrics calculation
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
```

[21]:
```
accur = accuracy_score(prediction,val_y)
print(accur)
```

```
0.4
```

[22]:
```
print( classification_report(prediction,val_y))
```

```
              precision    recall  f1-score   support

           0       1.00      0.25      0.40         4
           1       0.25      1.00      0.40         1

    accuracy                           0.40         5
   macro avg       0.62      0.62      0.40         5
weighted avg       0.85      0.40      0.40         5
```

[23]:
```
print(confusion_matrix(prediction,val_y))
```

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

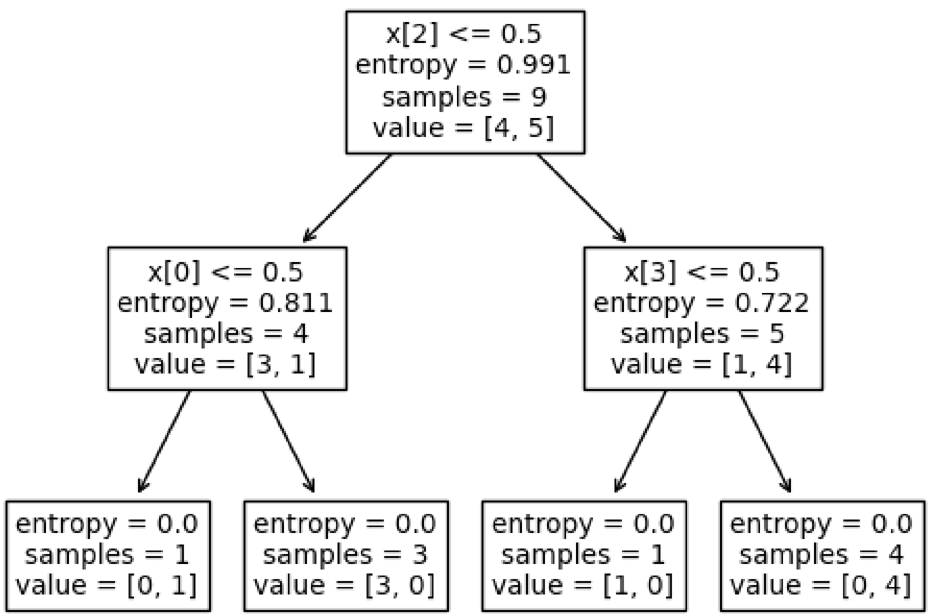
[24]:
```
# visualization of the tree
from sklearn.tree import export_graphviz
import sklearn.externals
from six import StringIO
```

```
from IPython.display import Image
import pydotplus
```

[25]:
```
dot_data = StringIO()
export_graphviz(tennis_model, out_file=dot_data,
filled=True, rounded=True,
special_characters=True,feature_names =features,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
#graph.write_png('Play Tennis.png')
#Image(graph.create_png())
```

[26]:
```
from sklearn.tree import plot_tree
```

[27]:
```
tre = plot_tree(tennis_model)
```



[ ]:

6-dt-for-movie-ratings

November 11, 2023

1 DT for Movie ratings

[1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

[2]:
```
# reading the csv file
movies = pd.read_csv('http://bit.ly/imdbratings')
```

[3]:
```
movies.head()
```

[3]:
```
   star_rating                     title content_rating   genre  duration  \
0          9.3  The Shawshank Redemption              R   Crime       142
1          9.2             The Godfather              R   Crime       175
2          9.1    The Godfather: Part II              R   Crime       200
3          9.0           The Dark Knight          PG-13  Action       152
4          8.9              Pulp Fiction              R   Crime       154

                                        actors_list
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…
4  [u'John Travolta', u'Uma Thurman', u'Samuel L…
```

[4]:
```
movies.columns
```

[4]:
```
Index(['star_rating', 'title', 'content_rating', 'genre', 'duration',
       'actors_list'],
      dtype='object')
```

[5]:
```
movies.isnull().sum()
```

[5]:
```
star_rating       0
title             0
content_rating    3
genre             0
duration          0
actors_list       0
dtype: int64
```

[6]:
```
content_rating_null_values = list(movies.content_rating.isnull())
for i in range(len(content_rating_null_values)):
  if content_rating_null_values[i] == True:
    print(i)
```

```
187
649
936
```

[7]:
```
movies.iloc[187,2]='PG13'
movies.iloc[649,2]='PG'
movies.iloc[936,2]='PG13'
```

[8]:
```
movies.drop(['actors_list'], axis=1, inplace=True)
```

[9]:
```
movies
```

[9]:
```
     star_rating                                          title  \
0            9.3                       The Shawshank Redemption
1            9.2                                  The Godfather
2            9.1                         The Godfather: Part II
3            9.0                                The Dark Knight
4            8.9                                   Pulp Fiction
..           …                                              …
974          7.4                                        Tootsie
975          7.4                    Back to the Future Part III
976          7.4  Master and Commander: The Far Side of the World
977          7.4                                    Poltergeist
978          7.4                                    Wall Street

    content_rating      genre  duration
0                R      Crime       142
1                R      Crime       175
2                R      Crime       200
3            PG-13     Action       152
4                R      Crime       154
..             …          …        …
```

```
974             PG     Comedy       116
975             PG  Adventure       118
976          PG-13     Action       138
977             PG     Horror       114
978              R      Crime       126

[979 rows x 5 columns]
```

[10]:
```python
categorical_features = [i for i in movies.select_dtypes(include=np.object)]
```

```
<ipython-input-10-97e24dbff0ba>:1: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  categorical_features = [i for i in movies.select_dtypes(include=np.object)]
```

[11]:
```python
dummy_df=pd.DataFrame()
```

[12]:
```python
dummy_df['duration']=movies.duration
```

[13]:
```python
for feature in categorical_features:
  df=pd.get_dummies(movies[feature])
```

[14]:
```python
train_df=pd.concat([df,dummy_df],axis=1)
```

[15]:
```python
train_df.head()
```

[15]:
```
   Action  Adventure  Animation  Biography  Comedy  Crime  Drama  Family  \
0       0          0          0          0       0      1      0       0
1       0          0          0          0       0      1      0       0
2       0          0          0          0       0      1      0       0
3       1          0          0          0       0      0      0       0
4       0          0          0          0       0      1      0       0

   Fantasy  Film-Noir  History  Horror  Mystery  Sci-Fi  Thriller  Western  \
0        0          0        0       0        0       0         0        0
1        0          0        0       0        0       0         0        0
2        0          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

   duration
0       142
1       175
2       200
3       152
```

```
4      154
```

[16]:
```
train_df=pd.concat([train_df,movies['star_rating']],axis=1)
```

[17]:
```
train_df.shape
```

[17]:
```
(979, 18)
```

[18]:
```
x = train_df.drop(['star_rating'], axis=1)
y=train_df['star_rating']
```

[19]:
```
X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=0.2,␣
↪random_state=42)
```

[20]:
```
LR=LinearRegression()
```

[21]:
```
LR.fit(X_train,y_train)
```

[21]:
```
LinearRegression()
```

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

[23]:
```
print('RMSE using Linear regression is', metrics.mean_squared_error(y_test,␣
↪y_pred,sample_weight=None))
```

```
RMSE using Linear regression is 0.0963980880321459
```

[24]:
```
sv=SVR()
```

[25]:
```
sv.fit(X_train, y_train)
```

[25]:
```
SVR()
```

[26]:
```
sv_pred = sv.predict(X_test)
```

[27]:
```
print('RMSE using SVR is', metrics.mean_squared_error(y_test,␣
↪sv_pred,sample_weight=None))
```

```
RMSE using SVR is 0.09749560506058148
```

[28]:
```
clf = tree.DecisionTreeRegressor()
```

[29]:
```
clf.fit(X_train, y_train)
```

[29]:
```
DecisionTreeRegressor()
```

[30]:
```
DT_pred = clf.predict(X_test)
```

[31]:
```
print('RMSE using DT is', metrics.mean_squared_error(y_test,␣
    ↪DT_pred,sample_weight=None))
```

RMSE using DT is 0.18113133503401357

7-dt-on-imdb-dataset

November 11, 2023

[ ]:
```
import pandas as pd
import numpy as np
import re
import string
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
```

[ ]:
```
#data = pd.read_csv("C:/Users/DSAI/Desktop/21STUCHH010385_lab/Datasets_385/IMDB/
 ↪IMDB_Dataset.csv",encoding='latin-1')
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[3]:
```
data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/ML_LAB/Datasets/IMDB␣
 ↪Dataset.csv",encoding='latin-1')
```

[5]:
```
data
```

[5]:
```
                                                  review sentiment
0      One of the other reviewers has mentioned that …  positive
1      A wonderful little production. <br /><br />The…  positive
2      I thought this was a wonderful way to spend ti…  positive
3      Basically there's a family where a little boy …  negative
4      Petter Mattei's "Love in the Time of Money" is…  positive
…                                                    …       …
49995  I thought this movie did a down right good job…  positive
49996  Bad plot, bad dialogue, bad acting, idiotic di…  negative
49997  I am a Catholic taught in parochial elementary…  negative
49998  I'm going to have to disagree with the previou…  negative
49999  No one expects the Star Trek movies to be high…  negative

[50000 rows x 2 columns]
```

[6]:
```python
data.shape
```

[6]:
```
(50000, 2)
```

[7]:
```python
data.head()
```

[7]:
```
                                              review sentiment
0  One of the other reviewers has mentioned that …  positive
1  A wonderful little production. <br /><br />The…  positive
2  I thought this was a wonderful way to spend ti…  positive
3  Basically there's a family where a little boy …  negative
4  Petter Mattei's "Love in the Time of Money" is…  positive
```

[8]:
```python
data["review"][1]
```

[8]:
```
'A wonderful little production. <br /><br />The filming technique is very
unassuming- very old-time-BBC fashion and gives a comforting, and sometimes
discomforting, sense of realism to the entire piece. <br /><br />The actors are
extremely well chosen- Michael Sheen not only "has got all the polari" but he
has all the voices down pat too! You can truly see the seamless editing guided
by the references to Williams\' diary entries, not only is it well worth the
watching but it is a terrificly written and performed piece. A masterful
production about one of the great master\'s of comedy and his life. <br /><br
/>The realism really comes home with the little things: the fantasy of the guard
which, rather than use the traditional \'dream\' techniques remains solid then
disappears. It plays on our knowledge and our senses, particularly with the
scenes concerning Orton and Halliwell and the sets (particularly of their flat
with Halliwell\'s murals decorating every surface) are terribly well done.'
```

[9]:
```python
review = data['review']
```

[10]:
```python
labels = data["sentiment"]
```

[11]:
```python
review
```

[11]:
```
0        One of the other reviewers has mentioned that …
1        A wonderful little production. <br /><br />The…
2        I thought this was a wonderful way to spend ti…
3        Basically there's a family where a little boy …
4        Petter Mattei's "Love in the Time of Money" is…
                              …
49995    I thought this movie did a down right good job…
49996    Bad plot, bad dialogue, bad acting, idiotic di…
49997    I am a Catholic taught in parochial elementary…
49998    I'm going to have to disagree with the previou…
49999    No one expects the Star Trek movies to be high…
Name: review, Length: 50000, dtype: object
```

[12]:
```
labels
```

[12]:
```
0        positive
1        positive
2        positive
3        negative
4        positive
           …
49995    positive
49996    negative
49997    negative
49998    negative
49999    negative
Name: sentiment, Length: 50000, dtype: object
```

1 Preprocessing the reviews

[13]:
```
# start replaceTwoorMore

def replaceTwoOrMore(s):
    #look for 2 or more repetitions of character and replace with the character␣
 ↪itself
    pattern = re.compile(r"(.)\1{1,}", re.DOTALL)
    return pattern.sub(r"\1\1", s)
```

[14]:
```
#start process_review

def processReview(review):
    # Removing numbers
    review = re.sub('[0-9]', '', review)
    #remove HTML tags
    cleanr=re.compile('<.*?>')
    review=re.sub(cleanr,' ',review)
    #Convert to lower case
    review = review.lower()
    review = review.translate(str.maketrans('', '', string.punctuation))
    #Remove additional white spaces
    review = re.sub('[\s]+', ' ', review)
    #Replace #word with word
    review = re.sub(r'#([^\s]+)', r'\1', review)
    #trim
    review = review.strip('\'"')
    review = review.strip('.,')
    review = replaceTwoOrMore(review)
    return review
```

[15]:
```python
processedReviews = []
for rev in review:
  processedReviews.append(processReview(rev))
```

[16]:
```python
processedReviews[1]
```

[16]:
```
'a wonderful little production the filming technique is very unassuming very
oldtimebbc fashion and gives a comforting and sometimes discomforting sense of
realism to the entire piece the actors are extremely well chosen michael sheen
not only has got all the polari but he has all the voices down pat too you can
truly see the seamless editing guided by the references to williams diary
entries not only is it well worth the watching but it is a terrificly written
and performed piece a masterful production about one of the great masters of
comedy and his life the realism really comes home with the little things the
fantasy of the guard which rather than use the traditional dream techniques
remains solid then disappears it plays on our knowledge and our senses
particularly with the scenes concerning orton and halliwell and the sets
particularly of their flat with halliwells murals decorating every surface are
terribly well done'
```

[17]:
```python
vectorizer = CountVectorizer(analyzer='word')
# Convert a collection of text documents to a matrix of token counts.
featurevector = vectorizer.fit_transform(processedReviews)
```

[18]:
```python
featurevector.shape
```

[18]:
```
(50000, 162851)
```

[19]:
```python
from sklearn.feature_extraction.text import CountVectorizer
```

[20]:
```python
corpus = [
     'This is the first document.',
     'This document is the second document.',
     'And this is the third one.',
     'Is this the first document?',
 ]
```

[21]:
```python
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
```

[22]:
```python
vectorizer.get_feature_names_out()
```

[22]:
```
array(['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third',
       'this'], dtype=object)
```

[23]:
```python
print(X.toarray())
```

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

[24]:
```python
vectorizer2 = CountVectorizer(analyzer='word', ngram_range=(2, 2))
```

[25]:
```python
X2 = vectorizer2.fit_transform(corpus)
```

[26]:
```python
vectorizer2.get_feature_names_out()
```

[26]:
```
array(['and this', 'document is', 'first document', 'is the', 'is this',
       'second document', 'the first', 'the second', 'the third',
       'third one', 'this document', 'this is', 'this the'], dtype=object)
```

[27]:
```python
print(X2.toarray())
```

```
[[0 0 1 1 0 0 1 0 0 0 0 1 0]
 [0 1 0 1 0 1 0 1 0 0 1 0 0]
 [1 0 0 1 0 0 0 0 1 1 0 1 0]
 [0 0 1 0 1 0 1 0 0 0 0 0 1]]
```

[28]:
```python
X_train, X_test, y_train, y_test = train_test_split(featurevector, labels,␣
 ↪test_size=0.30, random_state=42)
```

[29]:
```python
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[29]:
```
((35000, 162851), (15000, 162851), (35000,), (15000,))
```

[30]:
```python
print(X_train)
```

```
  (0, 101571)	1
  (0, 143397)	8
  (0, 62687)	1
  (0, 143293)	3
  (0, 11675)	1
  (0, 120152)	1
  (0, 7751)	4
  (0, 144246)	1
  (0, 72288)	2
  (0, 72831)	1
  (0, 4962)	6
  (0, 54091)	1
  (0, 145797)	4
  (0, 72594)	1
  (0, 101467)	2
  (0, 3495)	1
  (0, 30175)	1
```

```
  (0, 19479)	1
  (0, 56611)	1
  (0, 161742)	1
  (0, 82358)	1
  (0, 65351)	2
  (0, 156445)	1
  (0, 109223)	1
  (0, 83581)	1
  :	:
  (34999, 141212)	1
  (34999, 144791)	1
  (34999, 117035)	1
  (34999, 149586)	1
  (34999, 126164)	1
  (34999, 23648)	1
  (34999, 34329)	1
  (34999, 37364)	1
  (34999, 45847)	1
  (34999, 29333)	1
  (34999, 112099)	1
  (34999, 46952)	1
  (34999, 45861)	1
  (34999, 87569)	1
  (34999, 22719)	1
  (34999, 71309)	1
  (34999, 123798)	1
  (34999, 142751)	1
  (34999, 37090)	1
  (34999, 104292)	1
  (34999, 48315)	1
  (34999, 53258)	1
  (34999, 59882)	1
  (34999, 64744)	1
  (34999, 27663)	1
```

[31]:
```python
imdb_model = DecisionTreeClassifier(max_depth = 15)
```

[32]:
```python
imdb_model.fit(X_train, y_train)
```

[32]:
```
DecisionTreeClassifier(max_depth=15)
```

[33]:
```python
y_train_pred = imdb_model.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train, y_train_pred))

y_test_pred = imdb_model.predict(X_test)
print("Test Accuracy: ", accuracy_score(y_test, y_test_pred))
```

```
Train Accuracy:  0.8051428571428572
```

```
Test Accuracy:  0.7400666666666667
```

[34]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer # tf-idf method
```

[35]:
```python
#Convert a collection of raw documents to a matrix of TF-IDF features
tfidf = TfidfVectorizer(ngram_range=(1, 1))
tfidf_feature = tfidf.fit_transform(processedReviews)
```

[36]:
```python
tfidf_feature.get_shape()
```

[36]:
```
(50000, 162851)
```

[37]:
```python
feature_names = tfidf.get_feature_names_out()
len(feature_names)
```

[37]:
```
162851
```

[38]:
```python
feature_names[:10]
```

[38]:
```
array(['aa', 'aab', 'aachen', 'aada', 'aadha', 'aadmittedly', 'aag',
       'aage', 'aagghh', 'aagh'], dtype=object)
```

[39]:
```python
X_train, X_test, y_train, y_test = train_test_split(tfidf_feature, labels,␣
 ↪test_size=0.30, random_state=42)
```

[40]:
```python
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[40]:
```
((35000, 162851), (15000, 162851), (35000,), (15000,))
```

[41]:
```python
dt = DecisionTreeClassifier(max_depth = 15)
dt.fit(X_train, y_train)
```

[41]:
```
DecisionTreeClassifier(max_depth=15)
```

[42]:
```python
y_train_pred = dt.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train, y_train_pred))

y_test_pred = dt.predict(X_test)
print("Test Accuracy: ", accuracy_score(y_test, y_test_pred))
```

```
Train Accuracy:  0.8060857142857143
Test Accuracy:  0.7338666666666667
```

[43]:
```python
from sklearn.linear_model import LogisticRegression
```

[44]:
```python
logit = LogisticRegression()
```

[45]:
```python
logit.fit(X_train, y_train)
```

[45]: LogisticRegression()

[46]:
```
y_train_pred_logit = logit.predict(X_train)
print("Training Accuracy :", accuracy_score(y_train, y_train_pred_logit))

y_test_pred_logit = logit.predict(X_test)
print("Testing Accuracy :", accuracy_score(y_test, y_test_pred_logit))
```

```
Training Accuracy : 0.9311714285714285
Testing Accuracy : 0.8978
```

[46]:
```

```

8-perceptron-from-scratch

November 11, 2023

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

[2]:
```python
class Perceptron:
  def __init__(self,n,neta=0.1):
    self.w=np.random.randn(n+1)
    self.neta=neta
  def step(self,w_sum):
    if w_sum>0:
      return 1
    return 0
  def fit(self,X,y,epoch=5):
    X=np.c_[X,np.ones(X.shape[0])]
    for e in range(epoch):
      for (x,t) in zip(X,y):
        o=self.step(np.dot(x,self.w))
        if t!=o:
          er=t-o
          self.w+=self.neta*er*x
  def predict(self,X,addB=True):
    X=np.atleast_2d(X)
    if addB:
      X=np.c_[X,np.ones(X.shape[0])]
    return self.step(np.dot(X,self.w))
```

[3]:
```python
import numpy as np


X= np.array([[0,0],[1,0],[0,1],[1,1]])
Y= np.array([[0],[0],[0],[1]])
# p=Perceptron(X.shape[1],neta=0.01)
p=Perceptron(X.shape[1],neta=0.01)
```

[4]:
```python
p.fit(X,Y,epoch=50)
```

[5]:
```python
p.w
```

[5]: array([0.17463856, 0.00628511, -0.17592398])

[6]:
```
for (x,t) in zip(X,Y):
  pred=p.predict(x)
  print(f"Data :{x},target :{pred}")
```

```
Data :[0 0],target :0
Data :[1 0],target :0
Data :[0 1],target :0
Data :[1 1],target :1
```

[7]:
```
p.fit(X,Y,epoch=100)
```

[8]:
```
for (x,t) in zip(X,Y):
  pred=p.predict(x)
  print(f"Data :{x},target :{pred}")
```

```
Data :[0 0],target :0
Data :[1 0],target :0
Data :[0 1],target :0
Data :[1 1],target :1
```

[9]:
```
from sklearn.linear_model import Perceptron
from sklearn.datasets import load_digits
```

[10]:
```
X,y=load_digits(return_X_y=True)
y
```

[10]: array([0, 1, 2, …, 8, 9, 8])

[11]:
```
p=Perceptron()
```

[12]:
```
p.fit(X,y)
```

[12]: Perceptron()

[13]:
```
print(p.score(X,y))
```

```
0.9393433500278241
```

[14]:
```
x=np.arange(36).reshape(-1,9)

x[0]
x[0].shape
x[0].shape[0]
```

[14]: 9

[15]:
```
name=["manjeet","Nikhil","Shambhvi","Astha"]
r=[4,1,3,2]
mapped=zip(name,r)
print(set(mapped))
```

```
{('Nikhil', 1), ('Astha', 2), ('manjeet', 4), ('Shambhvi', 3)}
```

[16]:
```
in_num=10
print("INPUT number : ",in_num)

out_arr = np.atleast_2d(in_num)
print("output 2d array from imput number : ",out_arr)
```

```
INPUT number :  10
output 2d array from imput number :  [[10]]
```

[17]:
```
p1=Perceptron()

X= np.array([[0,0],[1,0],[0,1],[1,1]])
Y= np.array([0,0,0,1])
p1.fit(X,Y)
```

[17]:
```
Perceptron()
```

[18]:
```
for (x,t) in zip(X,Y):
  pred=p1.predict([x])
  print(f"Data :{x},target :{pred}")
```

```
Data :[0 0],target :[0]
Data :[1 0],target :[0]
Data :[0 1],target :[0]
Data :[1 1],target :[1]
```

[ ]:
```

```

9-gender-classification-percep

November 11, 2023

[1]:
```
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
import numpy as np
```

[2]:
```
data = [[1.81, 0.80, 0.44],
[1.77, 0.70, 0.43],
[1.60, 0.60, 0.38],
[1.54, 0.54, 0.37],
[1.66, 0.65, 0.40],
[1.90, 0.90, 0.47],
[1.75, 0.64, 0.39],
[1.77, 0.70, 0.40],
[1.59, 0.55, 0.37],
[1.71, 0.75, 0.42],
[1.81, 0.85, 0.43]]

results = ['male', 'male','female', 'female','male', 'male','female', 'female',␣
 ↪'female','male', 'male']
```

[3]:
```
from sklearn.utils import class_weight
model=Perceptron(alpha=0.0001,class_weight=None,random_state=0,eta0=1.
 ↪0,fit_intercept=True,max_iter=1000,shuffle=True)
```

[4]:
```
model.fit(data,results)
```

[4]:
```
Perceptron()
```

[5]:
```
ans=model.predict(data)
acc_per=accuracy_score(results,ans)
acc_per*=100
acc_per
```

[5]:
```
54.54545454545454
```

[6]:
```
pred=model.predict([[1.6,0.6,0.38]])
```

[7]:
```
print(pred)
```

```
['male']
```

[8]:
```python
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import NearestNeighbors


methods = ["Descicion TREE","SVM","Perceptron","KNN"]

X= [[181, 80, 44], [177, 70, 43], [160, 60, 38], [154, 54, 37], [166, 65, 48], [198, 90, 47], [175, 64, 39], [177, 78, 40], [159, 55, 37], [171, 75, 42], [181, 85, 43]]
Y = ['male', 'male', 'female', 'female', 'male', 'male', 'female', 'female', 'female', 'male', 'male']

clf_tree=DecisionTreeClassifier()
clf_SVM =SVC()
clf_Perceptron=Perceptron()
clf_KNN = NearestNeighbors()
```

[9]:
```python
clf_tree= clf_tree.fit(X,Y)
clf_SVM= clf_SVM.fit(X,Y)
clf_Perceptron= clf_Perceptron.fit(X,Y)
clf_KNN= clf_KNN.fit(X,Y)
```

[10]:
```python
t=clf_tree.predict(X)
t=accuracy_score(Y,t)*100
s=clf_SVM.predict(X)
s=accuracy_score(Y,s)*100
p=clf_Perceptron.predict(X)
p=accuracy_score(Y,p)*100
k,i=clf_KNN.kneighbors(X)
new_l=i[:,0]
k=[Y[i][:] for i in new_l]

k=accuracy_score(Y,k)*100
```

[11]:
```python
acc_all=[t,s,p,k]
```

[12]:
```python
score =np.max(acc_all)
```

[13]:
```python
best_method=np.argmax(acc_all)
```

[14]:
```python
print(methods[best_method],"is the best method of ",score)
```

```
Descicion TREE is the best method of  100.0
```

```
[ ]:
```

```
[ ]:
```

10-multilayer-nn-for-xor

November 11, 2023

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

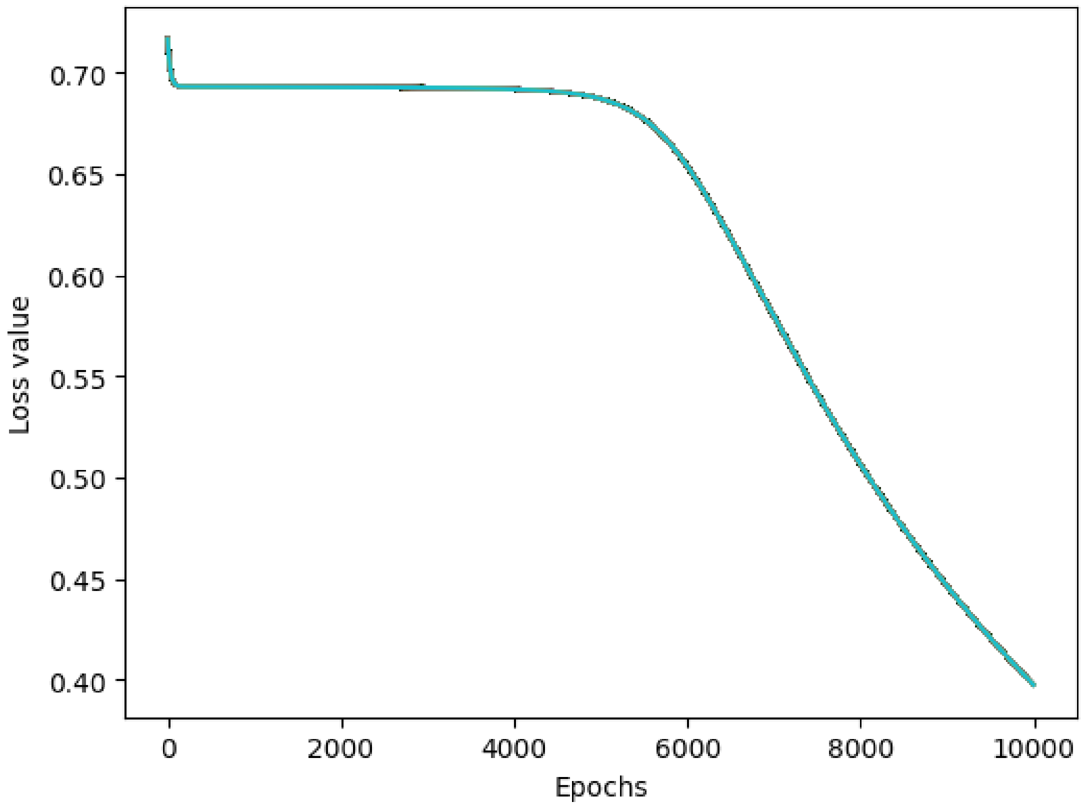
[2]:
```
X=np.array([[0,0,1,1],[0,1,0,1]])
Y=np.array([[0,1,1,0]])
```

[3]:
```
n_x=2
n_y=1
n_h=2
m=X.shape[1]
lr=0.1
np.random.seed(2)
w1=np.random.rand(n_h,n_x)
w2=np.random.rand(n_y,n_h)
loses=[]
```

[4]:
```
def sigmoid(z):
  return 1/(1+np.exp(-z))

def forw_prop(w1,w2,x):
  z1=np.dot(w1,x)
  a1=sigmoid(z1)
  z2=np.dot(w2,a1)
  a2=sigmoid(z2)
  return z1,a1,z2,a2
def back_prop(m,w1,w2,z1,a1,z2,a2,y):
  dz2 =a2-y
  dw2 = np.dot(dz2,a1.T)/m
  dz1=np.dot(w2.T,dz2)*a1*(1-a1)
  dw1=np.dot(dz1,X.T)/m
  dw1=np.reshape(dw1,w1.shape)
  dw2=np.reshape(dw2,w2.shape)
  return dz2,dw2,dz1,dw1
```

[5]:
```
ii=10000
for i in range(ii):
  z1,a1,z2,a2=forw_prop(w1,w2,X)
```

```
  loss= -(1/m)*np.sum(Y*np.log(a2)+(1-Y)*np.log(1-a2))
  loses.append(loss)
  da2,dw2,dz1,dw1=back_prop(m,w1,w2,z1,a1,z2,a2,Y)
  w2=w2-lr*dw2
  w1=w1-lr*dw1
  plt.plot(loses)
  plt.xlabel("Epochs")
  plt.ylabel("Loss value")
```



[6]:
```
def predict(w1,w2,input):
  z1,a1,z2,a2=forw_prop(w1,w2,test)
  a2=np.squeeze(a2)
  # print(a2)
  if a2>=0.5:
    print("For input",[i[0] for i in input],"output is 1")
  else:
    print("For input",[i[0] for i in input],"output is 0")
```

[7]:
```
test=np.array([[1],[0]])
predict(w1,w2,test)
test=np.array([[0],[0]])
```

```
predict(w1,w2,test)
test=np.array([[0],[1]])
predict(w1,w2,test)
test=np.array([[1],[1]])
predict(w1,w2,test)
```

```
For input [1, 0] output is 1
For input [0, 0] output is 0
For input [0, 1] output is 1
For input [1, 1] output is 0
```

[7]:

```

```

11-multilayer-nn-for-mnist-digits

November 11, 2023

[14]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten,Dense,Activation
import matplotlib.pyplot as plt
```

[15]:
```
(x_train, y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data()
```

[16]:
```
print("number of Training example: ",x_train.shape)
print("number of Training example target: ",y_train.shape)
print("number of testing example : ",x_test.shape)
print("number of testing example: ",y_test.shape)
```

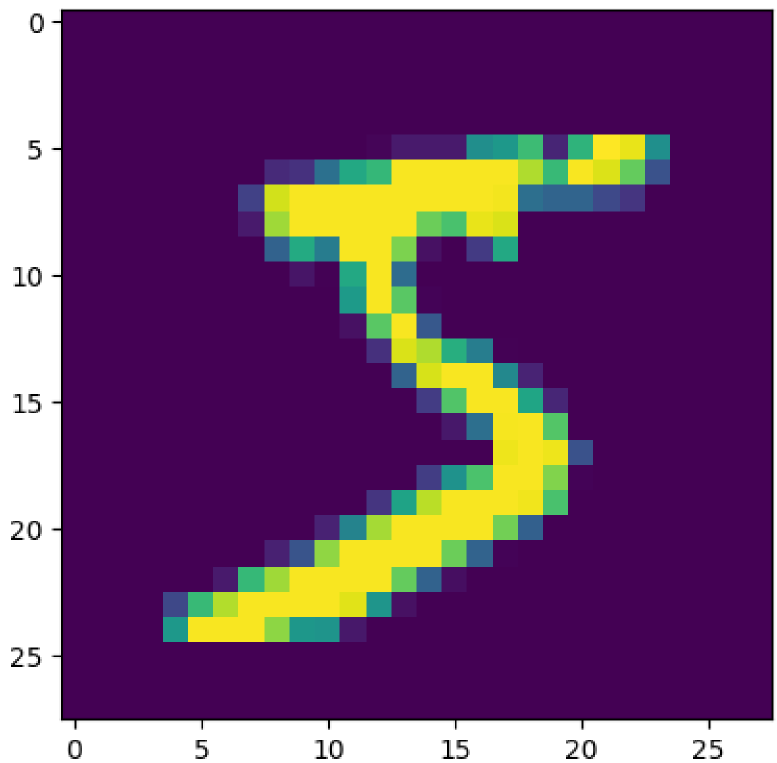
```
number of Training example:  (60000, 28, 28)
number of Training example target:  (60000,)
number of testing example :  (10000, 28, 28)
number of testing example:  (10000,)
```

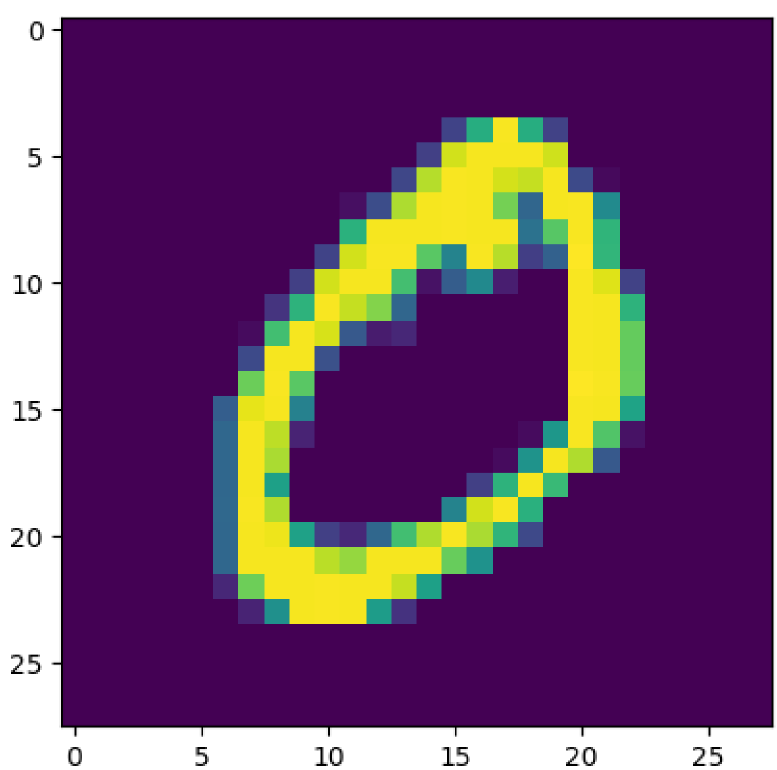
[17]:
```
print(x_train[0])
```

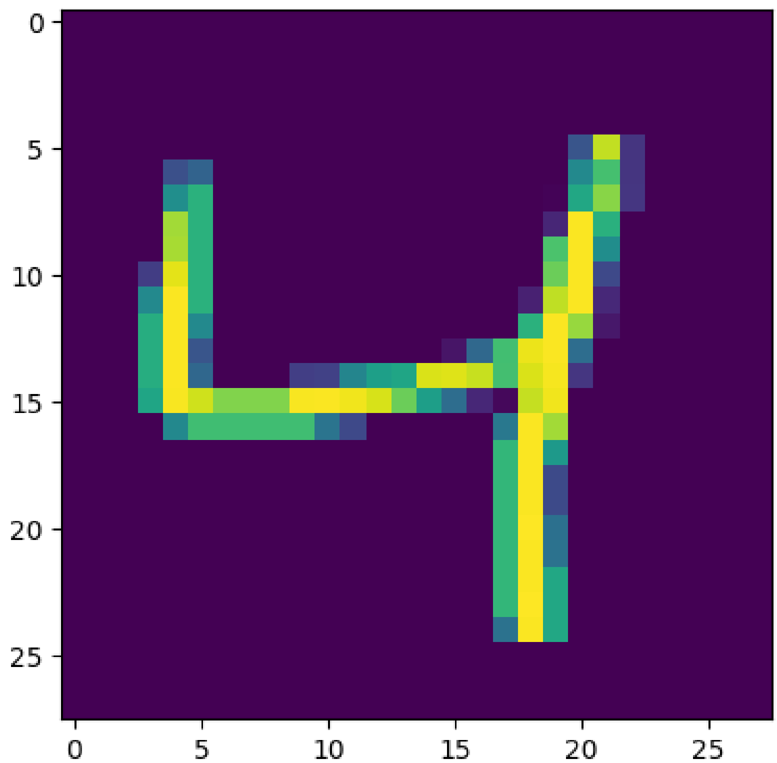
```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
```

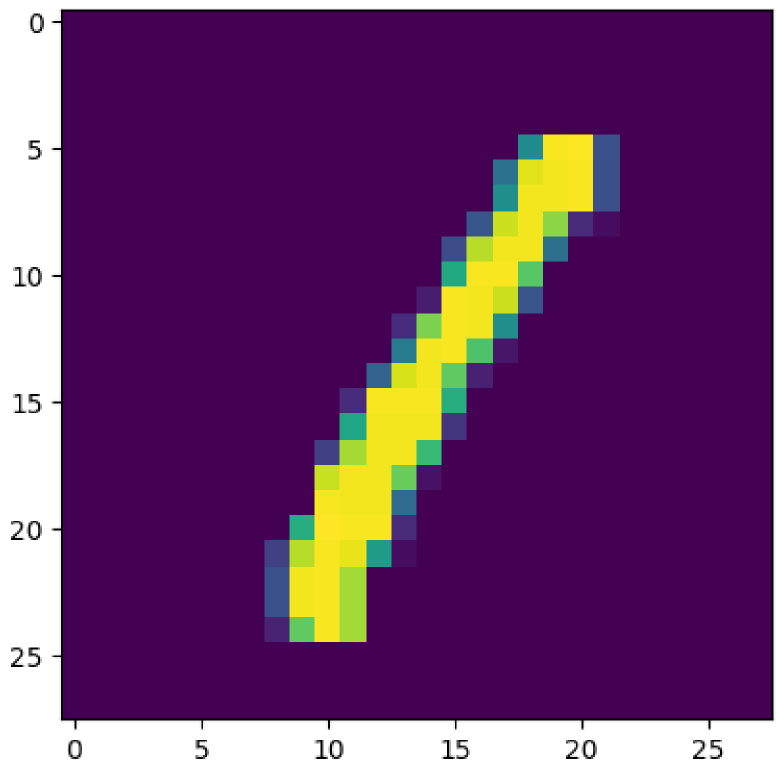
```
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
    25   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
   150  27   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
   253 187   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
   253 249  64   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
   253 207   2   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
   250 182   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
    78   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]]
```

[18]:
```
k=0
for i in range(2):
  for j in range(2):
    plt.imshow(x_train[k])
    k+=1
    plt.show()
```


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25



[19]:
```
y_train[0:4]
```

[19]:
```
array([5, 0, 4, 1], dtype=uint8)
```

[20]:
```
X_train=x_train/255
x_test=x_test/255
X_train[0]
```

[20]:
```
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.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.01176471, 0.07058824, 0.07058824,
 0.07058824, 0.49411765, 0.53333333, 0.68627451, 0.10196078,
 0.65098039, 1.        , 0.96862745, 0.49803922, 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.11764706, 0.14117647,
 0.36862745, 0.60392157, 0.66666667, 0.99215686, 0.99215686,
 0.99215686, 0.99215686, 0.99215686, 0.88235294, 0.6745098 ,
 0.99215686, 0.94901961, 0.76470588, 0.25098039, 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.19215686, 0.93333333, 0.99215686,
 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
 0.99215686, 0.99215686, 0.98431373, 0.36470588, 0.32156863,
 0.32156863, 0.21960784, 0.15294118, 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.07058824, 0.85882353, 0.99215686,
 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.77647059,
 0.71372549, 0.96862745, 0.94509804, 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.31372549, 0.61176471,
 0.41960784, 0.99215686, 0.99215686, 0.80392157, 0.04313725,
 0.        , 0.16862745, 0.60392157, 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
 0.        , 0.        , 0.        , 0.        , 0.05490196,
 0.00392157, 0.60392157, 0.99215686, 0.35294118, 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.54509804, 0.99215686, 0.74509804, 0.00784314,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.04313725, 0.74509804, 0.99215686, 0.2745098 ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.1372549 , 0.94509804, 0.88235294,
 0.62745098, 0.42352941, 0.00392157, 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.31764706, 0.94117647,
 0.99215686, 0.99215686, 0.46666667, 0.09803922, 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.17647059,
 0.72941176, 0.99215686, 0.99215686, 0.58823529, 0.10588235,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.0627451 , 0.36470588, 0.98823529, 0.99215686, 0.73333333,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.97647059, 0.99215686, 0.97647059,
 0.25098039, 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
```

```
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.18039216,
 0.50980392, 0.71764706, 0.99215686, 0.99215686, 0.81176471,
 0.00784314, 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.15294118, 0.58039216, 0.89803922,
 0.99215686, 0.99215686, 0.99215686, 0.98039216, 0.71372549,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.09411765, 0.44705882, 0.86666667, 0.99215686, 0.99215686,
 0.99215686, 0.99215686, 0.78823529, 0.30588235, 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.09019608, 0.25882353,
 0.83529412, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
 0.77647059, 0.31764706, 0.00784314, 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.07058824, 0.67058824, 0.85882353, 0.99215686,
 0.99215686, 0.99215686, 0.99215686, 0.76470588, 0.31372549,
 0.03529412, 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.21568627,
 0.6745098 , 0.88627451, 0.99215686, 0.99215686, 0.99215686,
 0.99215686, 0.95686275, 0.52156863, 0.04313725, 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.53333333,
 0.99215686, 0.99215686, 0.99215686, 0.83137255, 0.52941176,
 0.51764706, 0.0627451 , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
```

```
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ]])
```

[21]:
```
y_train = tf.keras.utils.to_categorical(y_train,10)
y_test = tf.keras.utils.to_categorical(y_test,10)

y_train.shape
```

[21]:
```
(60000, 10)
```

[22]:
```
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_4 (Dense)             (None, 256)               200960

 dense_5 (Dense)             (None, 128)               32896

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 10)                650

=================================================================
Total params: 242762 (948.29 KB)
Trainable params: 242762 (948.29 KB)
Non-trainable params: 0 (0.00 Byte)
```

----

[23]:
```
model.
 ↪compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
train_history = model.
 ↪fit(x_train,y_train,batch_size=64,epochs=10,verbose=1,validation_split=0.2)
```

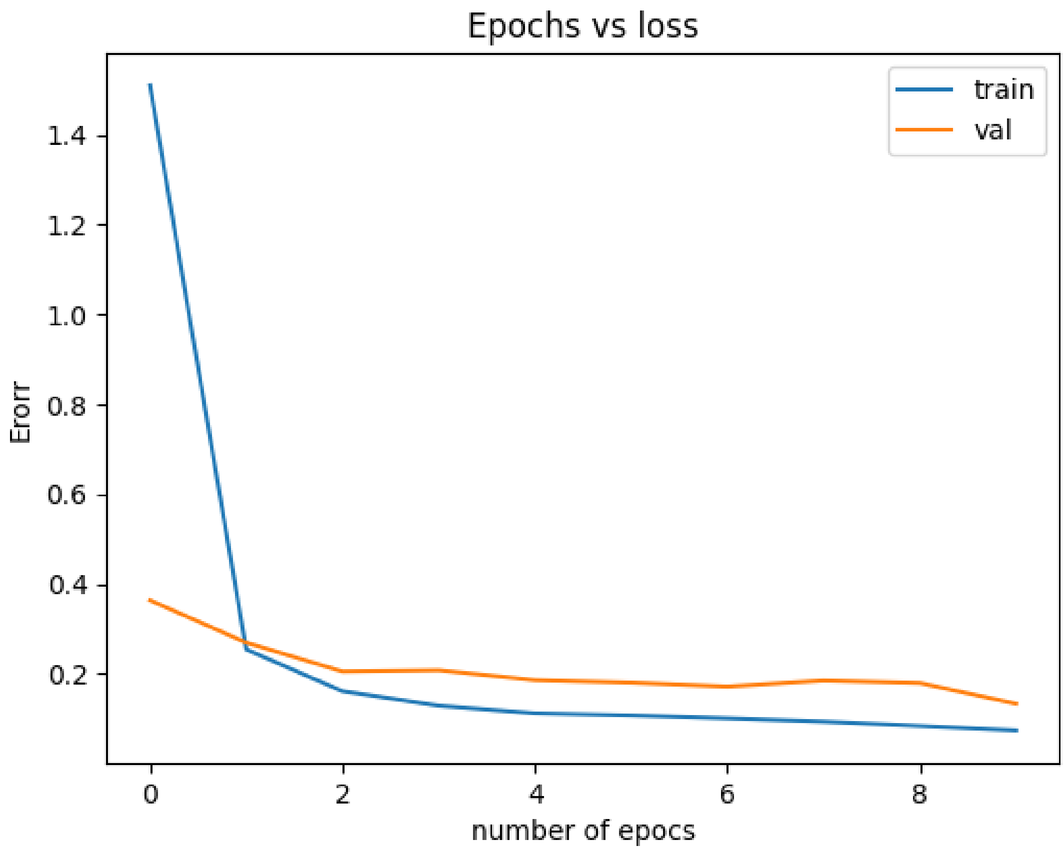
```
Epoch 1/10
750/750 [==============================] - 7s 9ms/step - loss: 1.5095 -
accuracy: 0.8668 - val_loss: 0.3636 - val_accuracy: 0.9237
Epoch 2/10
750/750 [==============================] - 11s 15ms/step - loss: 0.2543 -
accuracy: 0.9385 - val_loss: 0.2694 - val_accuracy: 0.9390
Epoch 3/10
750/750 [==============================] - 12s 16ms/step - loss: 0.1611 -
accuracy: 0.9567 - val_loss: 0.2051 - val_accuracy: 0.9507
Epoch 4/10
750/750 [==============================] - 11s 15ms/step - loss: 0.1290 -
accuracy: 0.9635 - val_loss: 0.2077 - val_accuracy: 0.9530
Epoch 5/10
750/750 [==============================] - 5s 7ms/step - loss: 0.1120 -
accuracy: 0.9680 - val_loss: 0.1859 - val_accuracy: 0.9574
Epoch 6/10
750/750 [==============================] - 6s 8ms/step - loss: 0.1073 -
accuracy: 0.9692 - val_loss: 0.1802 - val_accuracy: 0.9567
Epoch 7/10
750/750 [==============================] - 5s 7ms/step - loss: 0.1007 -
accuracy: 0.9712 - val_loss: 0.1715 - val_accuracy: 0.9647
Epoch 8/10
750/750 [==============================] - 7s 9ms/step - loss: 0.0935 -
accuracy: 0.9741 - val_loss: 0.1849 - val_accuracy: 0.9557
Epoch 9/10
750/750 [==============================] - 5s 7ms/step - loss: 0.0838 -
accuracy: 0.9758 - val_loss: 0.1793 - val_accuracy: 0.9606
Epoch 10/10
750/750 [==============================] - 6s 8ms/step - loss: 0.0739 -
accuracy: 0.9793 - val_loss: 0.1333 - val_accuracy: 0.9690
```

[24]:
```
train_history.history.keys()
```

[24]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[25]:
```
plt.plot(train_history.history['loss'])
plt.plot(train_history.history['val_loss'])
plt.title("Epochs vs loss")
plt.xlabel("number of epocs")
plt.ylabel("Erorr")
```

```
plt.legend(['train','val'])
plt.show()
```



[26]:
```
score=model.evaluate(x_test,y_test,batch_size=64)
print("testing accuray: ",score[1])
```

```
157/157 [==============================] - 1s 4ms/step - loss: 2.2479 -
accuracy: 0.1115
testing accuray:  0.11150000244379044
```

12-ml-nn-for-face-recog

November 11, 2023

[32]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.datasets import fetch_lfw_people
from   sklearn.model_selection import  train_test_split
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
```

[33]:
```
lfw =fetch_lfw_people(min_faces_per_person=100)
```

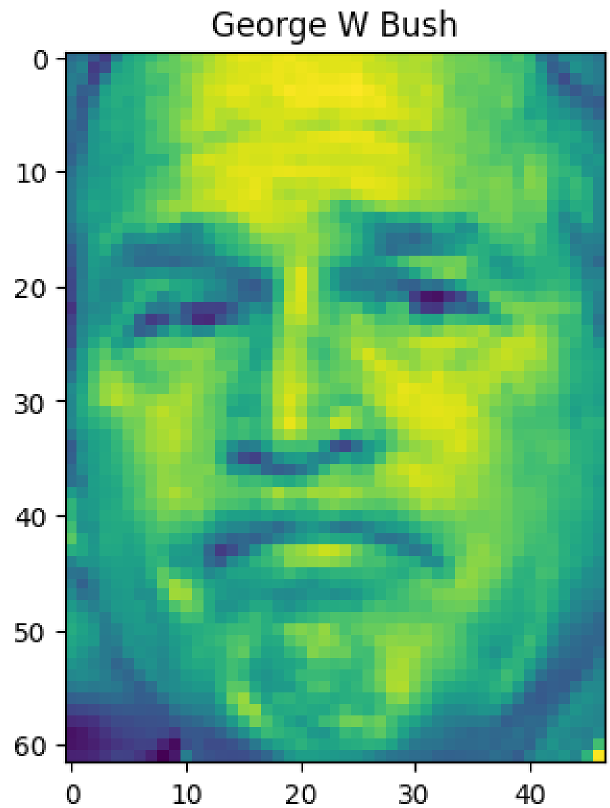
[34]:
```
n_samples, h, w=lfw.images.shape
X=lfw.data
y=lfw.target
target_names=lfw.target_names
print("input data shape",X.shape)
print("Target_names",target_names)
```

```
input data shape (1140, 2914)
Target_names ['Colin Powell' 'Donald Rumsfeld' 'George W Bush' 'Gerhard
Schroeder'
 'Tony Blair']
```

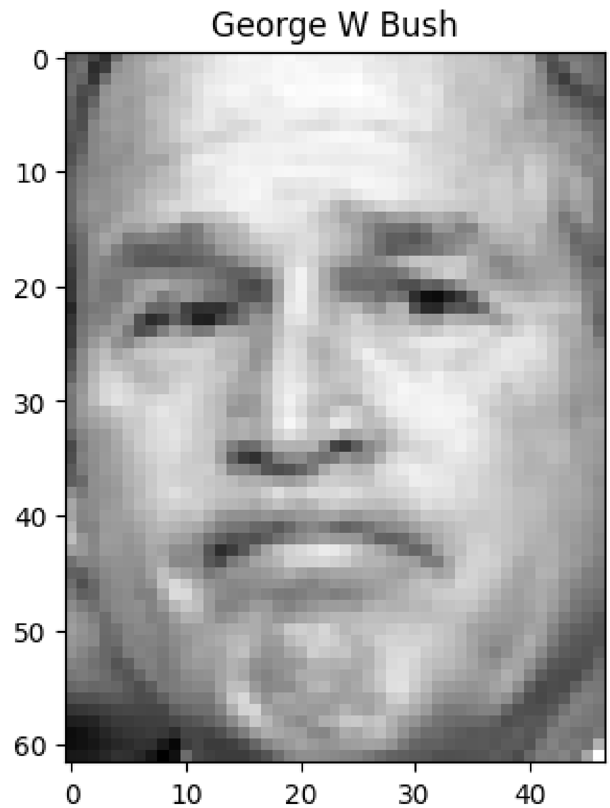
[35]:
```
X[0]
```

[35]:
```
array([0.32026145, 0.34771243, 0.26013073, …, 0.4       , 0.5542484 ,
       0.82483655], dtype=float32)
```

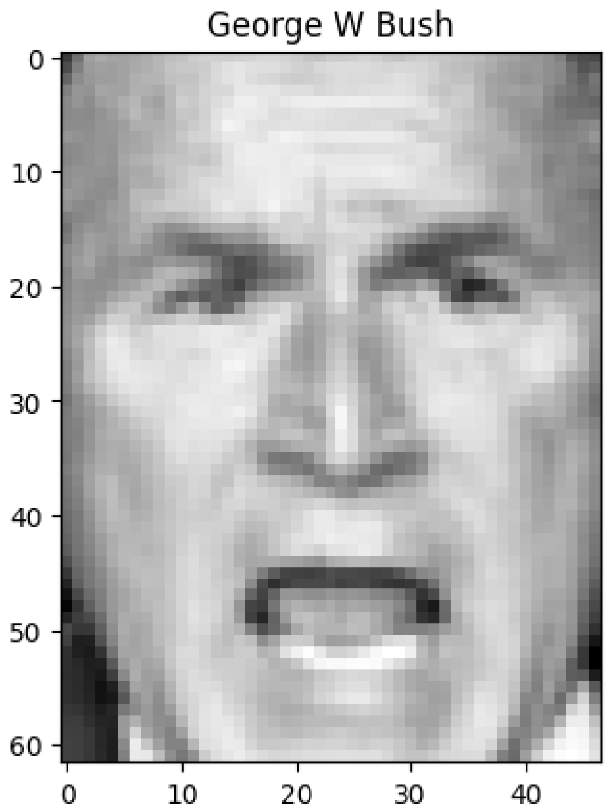
[36]:
```
plt.imshow(lfw.images[0])
plt.title(target_names[y[0]])
plt.show()
```



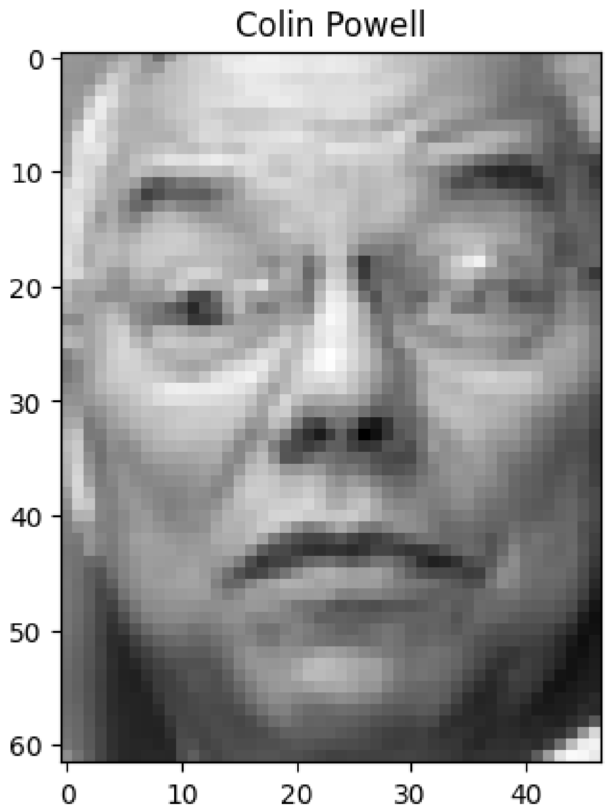
[37]:
```
plt.imshow(lfw.images[0], cmap= 'gray')
plt.title(target_names[y[0]])
plt.show()
```



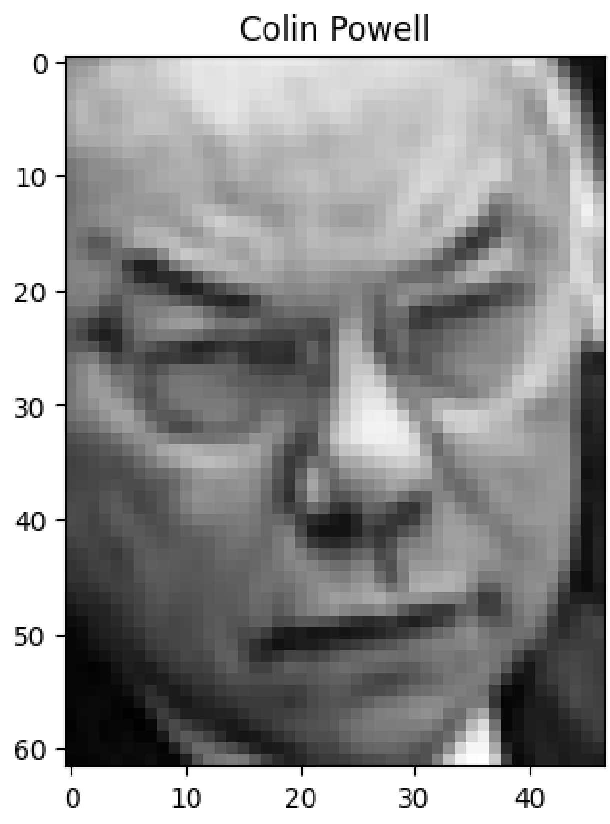
[38]:
```
plt.imshow(lfw.images[100], cmap= 'gray')
plt.title(target_names[y[100]])
plt.show()
```



[39]:
```
i=101
plt.imshow(lfw.images[i], cmap= 'gray')
plt.title(target_names[y[i]])
plt.show()
```



[40]:
```
i=105
plt.imshow(lfw.images[i], cmap= 'gray')
plt.title(target_names[y[i]])
plt.show()
```



[41]:
```
target_names.shape[0]
```

[41]: 5

[42]:
```
X.shape[0]
```

[42]: 1140

[43]:
```
X.shape[1]
```

[43]: 2914

[44]:
```
target_names[0]
```

[44]: 'Colin Powell'

[45]:
```
model=Sequential()
model.add(Dense(256,input_dim=X.shape[1],activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(target_names.shape[0],activation='softmax'))
model.summary()
```

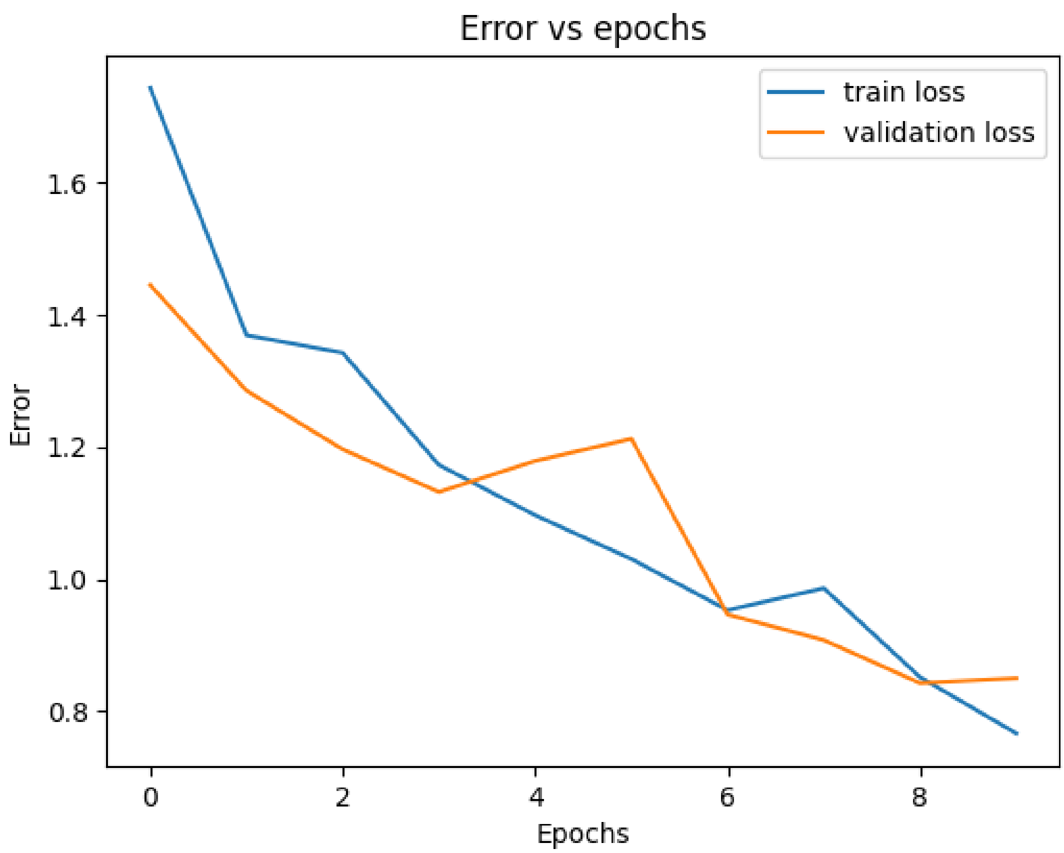
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 256)               746240

 dense_7 (Dense)             (None, 128)               32896

 dense_8 (Dense)             (None, 5)                 645

=================================================================
Total params: 779781 (2.97 MB)
Trainable params: 779781 (2.97 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[46]:
```
model.
 ↪compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history=model.fit(X,y,batch_size=32,epochs=10,validation_split=0.2)
```

```
Epoch 1/10
29/29 [==============================] - 1s 9ms/step - loss: 1.7431 - accuracy:
0.3914 - val_loss: 1.4448 - val_accuracy: 0.5000
Epoch 2/10
29/29 [==============================] - 0s 4ms/step - loss: 1.3690 - accuracy:
0.4956 - val_loss: 1.2851 - val_accuracy: 0.5307
Epoch 3/10
29/29 [==============================] - 0s 5ms/step - loss: 1.3423 - accuracy:
0.4868 - val_loss: 1.1962 - val_accuracy: 0.5570
Epoch 4/10
29/29 [==============================] - 0s 7ms/step - loss: 1.1725 - accuracy:
0.5713 - val_loss: 1.1316 - val_accuracy: 0.5526
Epoch 5/10
29/29 [==============================] - 0s 9ms/step - loss: 1.0964 - accuracy:
0.5877 - val_loss: 1.1786 - val_accuracy: 0.5482
Epoch 6/10
29/29 [==============================] - 0s 8ms/step - loss: 1.0302 - accuracy:
0.6151 - val_loss: 1.2120 - val_accuracy: 0.6272
Epoch 7/10
29/29 [==============================] - 0s 4ms/step - loss: 0.9528 - accuracy:
0.6678 - val_loss: 0.9462 - val_accuracy: 0.6404
Epoch 8/10
29/29 [==============================] - 0s 4ms/step - loss: 0.9859 - accuracy:
0.6261 - val_loss: 0.9075 - val_accuracy: 0.6842
Epoch 9/10
29/29 [==============================] - 0s 4ms/step - loss: 0.8514 - accuracy:
0.6886 - val_loss: 0.8424 - val_accuracy: 0.7456
Epoch 10/10
```

```
29/29 [==============================] - 0s 4ms/step - loss: 0.7662 - accuracy:
0.7237 - val_loss: 0.8498 - val_accuracy: 0.7237
```

[47]:
```python
plt.plot(history.history['loss'],label='train loss')
plt.plot(history.history['val_loss'],label='validation loss')
plt.title("Error vs epochs")
plt.xlabel("Epochs")
plt.ylabel("Error")
plt.legend()
plt.show()
```



[48]:
```python
model.compile(tf.keras.optimizers.Adadelta(learning_rate=0.0001, rho = 0.
↪9),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history=model.fit(X,y,batch_size=64 ,epochs= 25,validation_split=0.2)
```

```
Epoch 1/25
15/15 [==============================] - 1s 15ms/step - loss: 0.6908 - accuracy:
0.8136 - val_loss: 0.8484 - val_accuracy: 0.7193
Epoch 2/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6897 - accuracy:
```

```
0.8147 - val_loss: 0.8469 - val_accuracy: 0.7149
Epoch 3/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6886 - accuracy:
0.8158 - val_loss: 0.8455 - val_accuracy: 0.7193
Epoch 4/25
15/15 [==============================] - 0s 6ms/step - loss: 0.6875 - accuracy:
0.8136 - val_loss: 0.8441 - val_accuracy: 0.7193
Epoch 5/25
15/15 [==============================] - 0s 6ms/step - loss: 0.6863 - accuracy:
0.8136 - val_loss: 0.8427 - val_accuracy: 0.7237
Epoch 6/25
15/15 [==============================] - 0s 7ms/step - loss: 0.6852 - accuracy:
0.8147 - val_loss: 0.8413 - val_accuracy: 0.7237
Epoch 7/25
15/15 [==============================] - 0s 6ms/step - loss: 0.6841 - accuracy:
0.8136 - val_loss: 0.8399 - val_accuracy: 0.7281
Epoch 8/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6830 - accuracy:
0.8147 - val_loss: 0.8385 - val_accuracy: 0.7281
Epoch 9/25
15/15 [==============================] - 0s 6ms/step - loss: 0.6819 - accuracy:
0.8136 - val_loss: 0.8371 - val_accuracy: 0.7325
Epoch 10/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6808 - accuracy:
0.8125 - val_loss: 0.8357 - val_accuracy: 0.7325
Epoch 11/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6798 - accuracy:
0.8114 - val_loss: 0.8345 - val_accuracy: 0.7325
Epoch 12/25
15/15 [==============================] - 0s 6ms/step - loss: 0.6788 - accuracy:
0.8103 - val_loss: 0.8331 - val_accuracy: 0.7325
Epoch 13/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6778 - accuracy:
0.8103 - val_loss: 0.8321 - val_accuracy: 0.7325
Epoch 14/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6770 - accuracy:
0.8114 - val_loss: 0.8310 - val_accuracy: 0.7281
Epoch 15/25
15/15 [==============================] - 0s 6ms/step - loss: 0.6762 - accuracy:
0.8114 - val_loss: 0.8299 - val_accuracy: 0.7281
Epoch 16/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6753 - accuracy:
0.8103 - val_loss: 0.8289 - val_accuracy: 0.7281
Epoch 17/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6745 - accuracy:
0.8103 - val_loss: 0.8278 - val_accuracy: 0.7237
Epoch 18/25
15/15 [==============================] - 0s 7ms/step - loss: 0.6736 - accuracy:
```

```
0.8092 - val_loss: 0.8266 - val_accuracy: 0.7237
Epoch 19/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6728 - accuracy:
0.8092 - val_loss: 0.8255 - val_accuracy: 0.7281
Epoch 20/25
15/15 [==============================] - 0s 6ms/step - loss: 0.6719 - accuracy:
0.8092 - val_loss: 0.8245 - val_accuracy: 0.7281
Epoch 21/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6711 - accuracy:
0.8092 - val_loss: 0.8235 - val_accuracy: 0.7281
Epoch 22/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6704 - accuracy:
0.8103 - val_loss: 0.8225 - val_accuracy: 0.7281
Epoch 23/25
15/15 [==============================] - 0s 6ms/step - loss: 0.6696 - accuracy:
0.8092 - val_loss: 0.8215 - val_accuracy: 0.7281
Epoch 24/25
15/15 [==============================] - 0s 5ms/step - loss: 0.6689 - accuracy:
0.8081 - val_loss: 0.8205 - val_accuracy: 0.7325
Epoch 25/25
15/15 [==============================] - 0s 6ms/step - loss: 0.6681 - accuracy:
0.8092 - val_loss: 0.8194 - val_accuracy: 0.7325
```

13-breast-cancer-predi-nb

November 11, 2023

[1]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
```

[2]:
```
data = load_breast_cancer()
label_names =data['target_names']
labels=data['target']
feature_names=data["feature_names"]
features=data['data']
```

[3]:
```
print(label_names)
print("Class label :",labels[0])
print(feature_names)
print(features[0])
```

```
['malignant' 'benign']
Class label : 0
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]
```

[4]:
```
train,test,train_labels,test_labels=train_test_split(features,labels,test_size=0.
↪2,random_state=42)
```

[5]:
```
gnb=GaussianNB()
gnb.fit(train,train_labels)
```

[5]:
```
GaussianNB()
```

[6]:
```
preds= gnb.predict(test)
print(preds,"\n")
print(accuracy_score(test_labels,preds))
```

```
[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 1 0]

0.9736842105263158
```

[ ]:

14-nb-play-tennis-wine-dataset

November 11, 2023

[1]:
```python
from sklearn import preprocessing
le=preprocessing.LabelEncoder()
```

[2]:
```python
weather=['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast',␣
↪'Sunny', 'Sunny',
'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy']
temp=['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool',␣
↪'Mild', 'Mild', 'Mild', 'Hot', 'Mild']
play=['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes',␣
↪'Yes', 'Yes', 'No']
```

[3]:
```python
weather_encoded =le.fit_transform(weather)
weather_encoded
```

[3]:
```
array([2, 2, 0, 1, 1, 1, 0, 2, 2, 1, 2, 0, 0, 1])
```

[4]:
```python
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
```

[5]:
```python
features=[tup for tup in zip(weather_encoded,temp_encoded)]
```

[6]:
```python
from sklearn.naive_bayes import GaussianNB

model=GaussianNB()
```

[7]:
```python
model.fit(features,label)
```

[7]:
```
GaussianNB()
```

[8]:
```python
predicted = model.predict([[0,2]])
print("Predicted Value ",predicted)
```

```
Predicted Value  [1]
```

[9]:
```python
from sklearn.datasets import load_wine
```

[10]:
```python
wine= load_wine()
```

[11]:
```
print("Feature: ",wine.feature_names)
```

```
Feature:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',
'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
```

[12]:
```
print("Labels: ",wine.target_names)
```

```
Labels:  ['class_0' 'class_1' 'class_2']
```

[13]:
```
wine.data.shape
```

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

[14]:
```
print(wine.data[0:5])
```

```
[[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
  2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
 [1.320e+01 1.780e+00 2.140e+00 1.120e+01 1.000e+02 2.650e+00 2.760e+00
  2.600e-01 1.280e+00 4.380e+00 1.050e+00 3.400e+00 1.050e+03]
 [1.316e+01 2.360e+00 2.670e+00 1.860e+01 1.010e+02 2.800e+00 3.240e+00
  3.000e-01 2.810e+00 5.680e+00 1.030e+00 3.170e+00 1.185e+03]
 [1.437e+01 1.950e+00 2.500e+00 1.680e+01 1.130e+02 3.850e+00 3.490e+00
  2.400e-01 2.180e+00 7.800e+00 8.600e-01 3.450e+00 1.480e+03]
 [1.324e+01 2.590e+00 2.870e+00 2.100e+01 1.180e+02 2.800e+00 2.690e+00
  3.900e-01 1.820e+00 4.320e+00 1.040e+00 2.930e+00 7.350e+02]]
```

[15]:
```
print(wine.target)
```

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

[16]:
```
from sklearn.model_selection import train_test_split
```

[17]:
```
x_train,x_test,y_train,y_test=train_test_split(wine.data,wine.
 ↪target,test_size=0.3)
```

[18]:
```
gnb=GaussianNB()
```

[19]:
```
gnb.fit(x_train,y_train)
```

[19]:
```
GaussianNB()
```

[20]:
```
y_pred=gnb.predict(x_test)
```

[21]:
```
from sklearn.metrics import accuracy_score
```

[22]:
```
print("ACCURACY SCORE IS ",accuracy_score(y_test,y_pred))
```

```
ACCURACY SCORE IS  0.9629629629629629
```

[ ]:
```

```

15-loan-prediction-nb

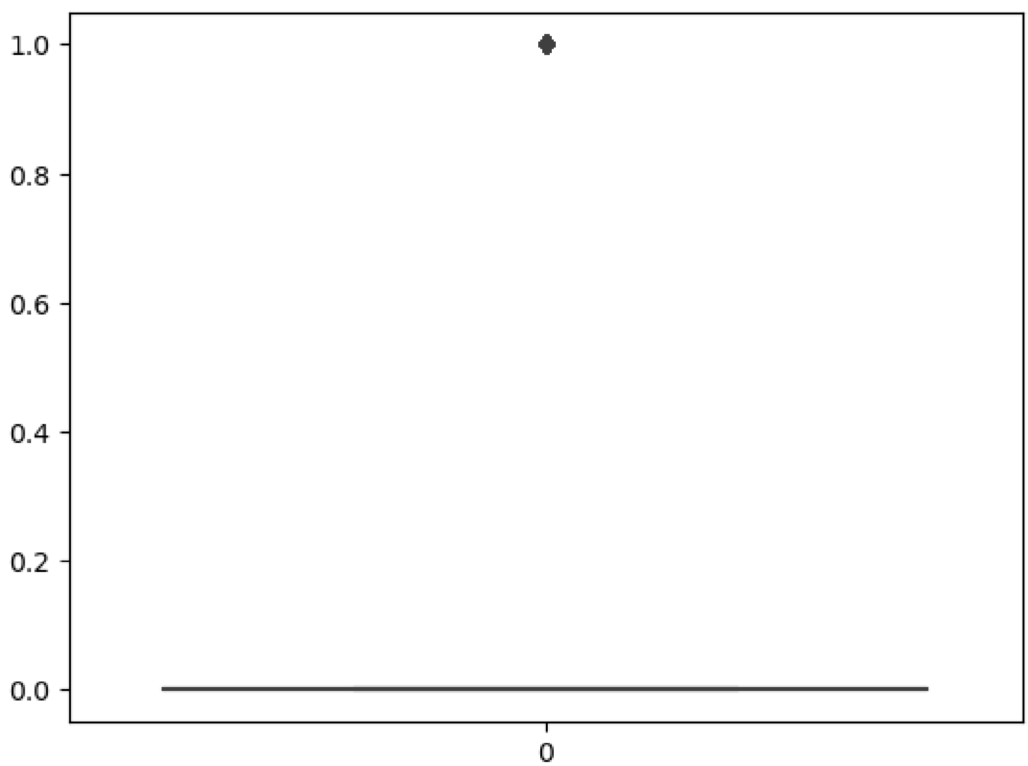
November 11, 2023

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

[2]:
```
df= pd.read_csv('Bank_Personal_Loan_Modelling.csv')
```

[3]:
```
import matplotlib.pyplot as plt
```

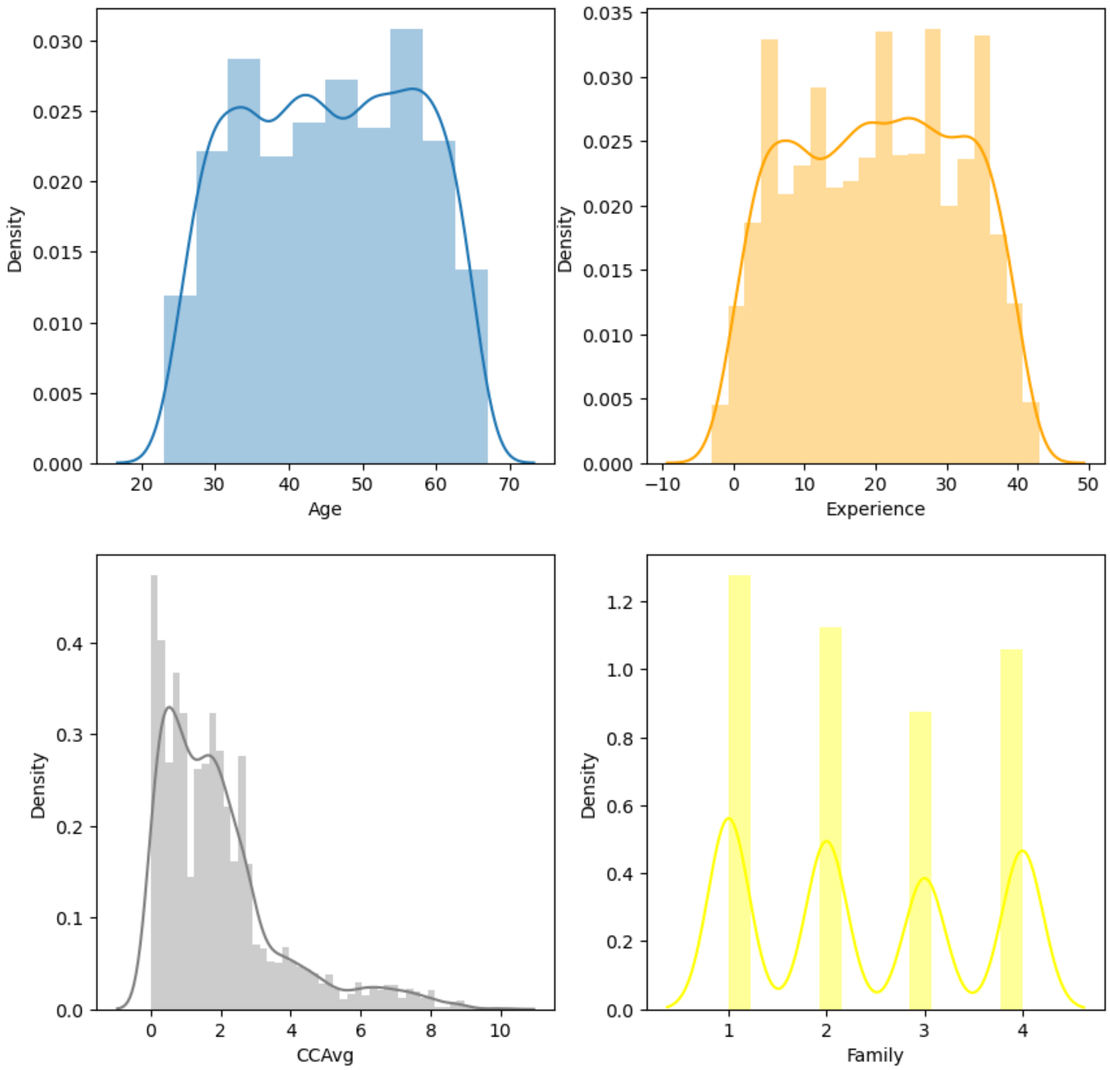
[4]:
```
sns.boxplot(df['Personal Loan'])
plt.show()
```



[5]:
```python
fig, axis =plt.subplots(2,2,figsize=(10,10),sharex=False)
sns.distplot(df['Age'],bins=10,ax=axis[0,0])
sns.distplot(df['Experience'],ax=axis[0,1],color='orange')
sns.distplot(df['CCAvg'],ax=axis[1,0],color='gray')
sns.distplot(df['Family'],ax=axis[1,1],color='yellow')
plt.show()
```

```
/tmp/ipykernel_53860/727864870.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Age'],bins=10,ax=axis[0,0])
/tmp/ipykernel_53860/727864870.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Experience'],ax=axis[0,1],color='orange')
/tmp/ipykernel_53860/727864870.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['CCAvg'],ax=axis[1,0],color='gray')
/tmp/ipykernel_53860/727864870.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
```

```
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Family'],ax=axis[1,1],color='yellow')
```

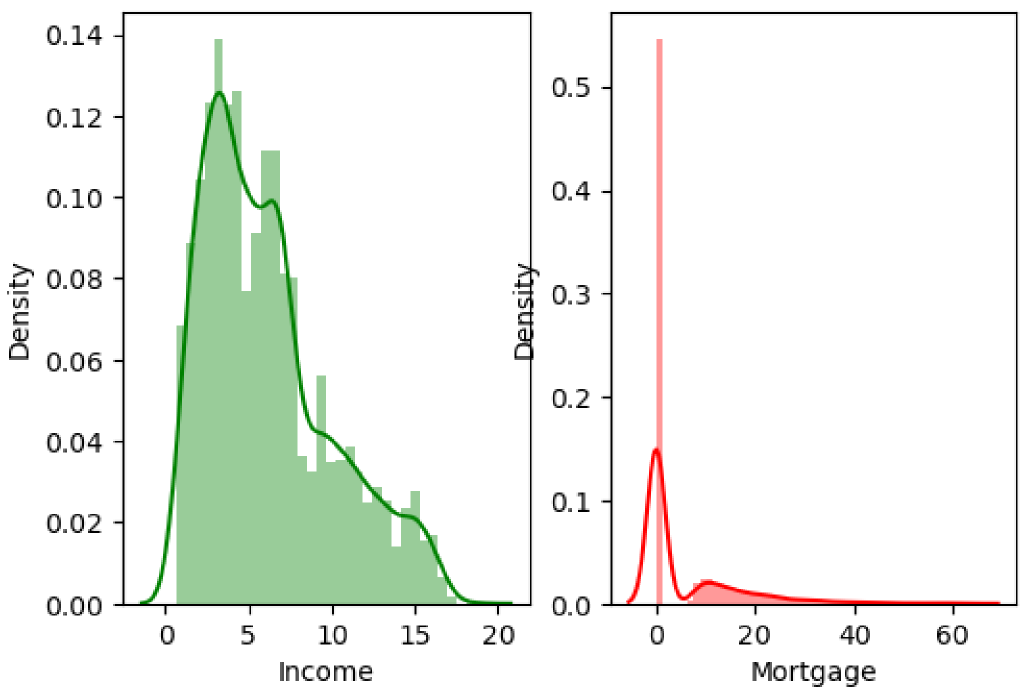


[6]:
```
df['Income']=df['Income']/12
df['Mortgage']=df['Mortgage']/10
```

[7]:
```
fig, axis =plt.subplots(1,2,figsize=(6,4),sharex=False)
sns.distplot(df['Income'],ax=axis[0],color='green')
sns.distplot(df['Mortgage'],ax=axis[1],color='red')
```

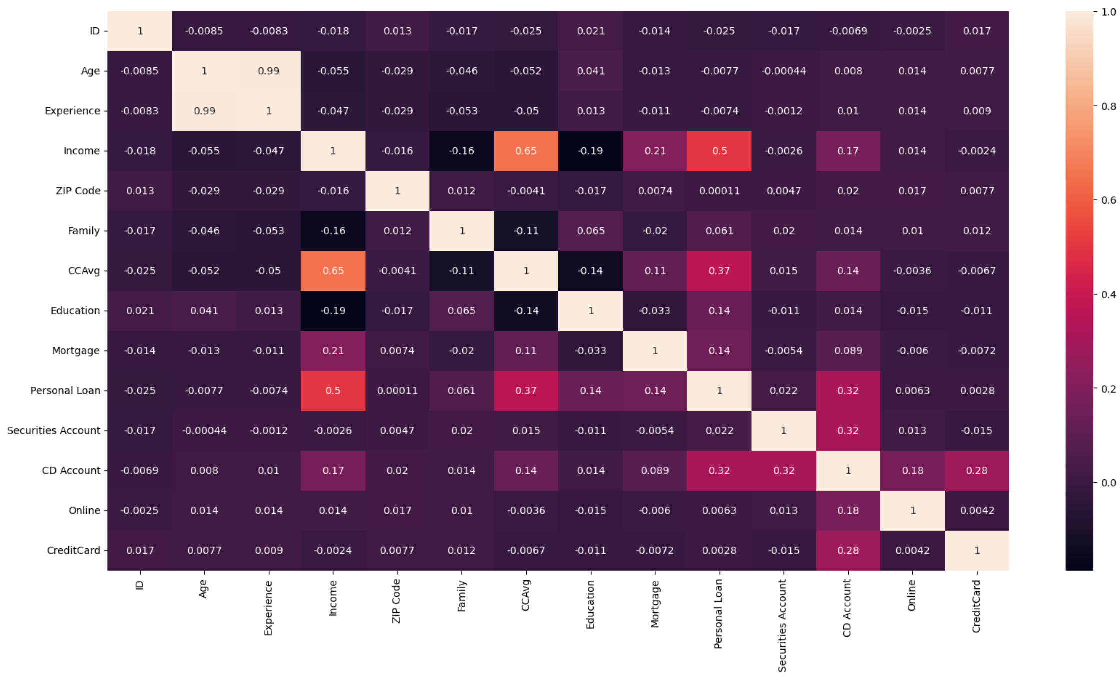
```
plt.show()
```

```
/tmp/ipykernel_53860/797622508.py:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Income'],ax=axis[0],color='green')
/tmp/ipykernel_53860/797622508.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['Mortgage'],ax=axis[1],color='red')
```



[8]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True)
plt.show
```

[8]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[9]:
```
x=df.drop(['Personal Loan'],axis=1)
y=df['Personal Loan']
```

[10]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

[11]:
```
from sklearn.linear_model import LogisticRegression
```

[12]:
```
logiR=LogisticRegression()
logiR.fit(x_train,y_train)
```

[12]:
```
LogisticRegression()
```

[13]:
```
logiR_test=logiR.predict(x_test)
```

[14]:
```
print("classification report")
print(classification_report(y_test,logiR_test))
```

```
classification report
              precision    recall  f1-score   support
```

```
           0       0.93      0.97      0.95      1358
           1       0.49      0.32      0.38       142

    accuracy                           0.90      1500
   macro avg       0.71      0.64      0.67      1500
weighted avg       0.89      0.90      0.89      1500
```

[15]:
```python
logiR_predict_train=logiR.predict_proba(x_train)[:,1]>0.8
logiR_predict_test=logiR.predict_proba(x_test)[:,1]>0.8
```

[16]:
```python
print("classification report")
cm=classification_report(y_test,logiR_predict_test,labels=[1,0])
print(cm)
```

```
classification report
              precision    recall  f1-score   support

           1       0.30      0.02      0.04       142
           0       0.91      0.99      0.95      1358

    accuracy                           0.90      1500
   macro avg       0.60      0.51      0.49      1500
weighted avg       0.85      0.90      0.86      1500
```

[17]:
```python
from sklearn.naive_bayes import GaussianNB

gnb=GaussianNB()
gnb.fit(x_train,y_train)
```

[17]: GaussianNB()

[18]:
```python
gnb_predict_test=logiR.predict_proba(x_test)[:,1]>0.8
cm=classification_report(y_test,gnb_predict_test,labels=[1,0])
print(cm)
```

```
              precision    recall  f1-score   support

           1       0.30      0.02      0.04       142
           0       0.91      0.99      0.95      1358

    accuracy                           0.90      1500
   macro avg       0.60      0.51      0.49      1500
weighted avg       0.85      0.90      0.86      1500
```

[ ]:
```python

```

16-knn-on-pima-dataset

November 11, 2023

[70]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics  import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
import pandas as pd
```

[71]:
```
diabetes=pd.read_csv('/content/drive/MyDrive/ML 385/diabetes.csv')
```

[72]:
```
diabetes.shape
```

[72]:
```
(768, 9)
```

[73]:
```
diabetes.describe()
```

[73]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
```

```
75%      36.600000                    0.626250   41.000000    1.000000
max      67.100000                    2.420000   81.000000    1.000000
```

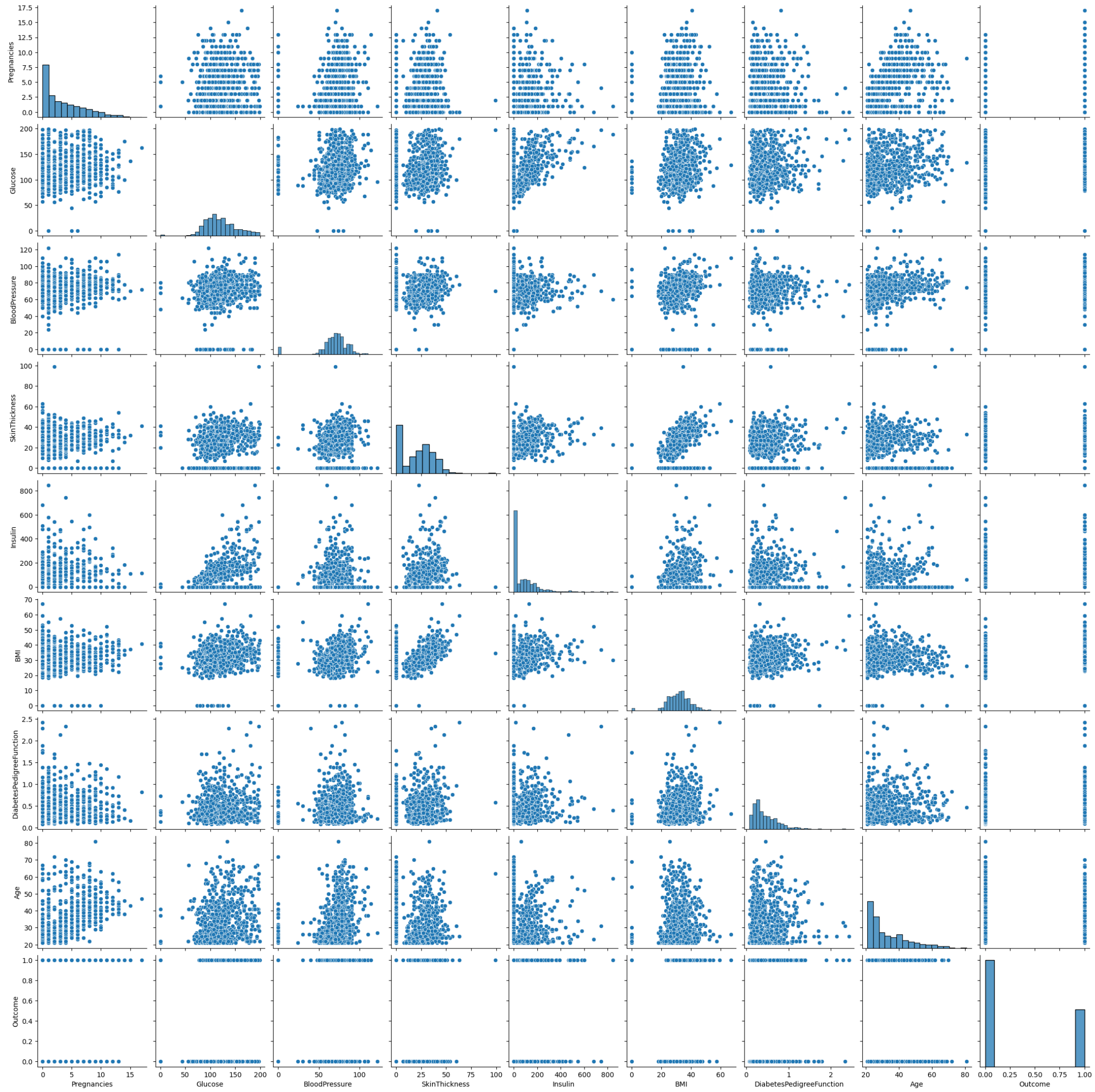
[74]:
```
diabetes.Outcome.value_counts()
```

[74]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

[75]:
```
diabetes.isna().sum()
```

[75]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[76]:
```
sns.pairplot(diabetes)
plt.show()
```



[77]:
```
diabetes.corr()
```

[77]:

	Pregnancies	Glucose	BloodPressure	SkinThickness \
Pregnancies	1.000000	0.129459	0.141282	-0.081672
Glucose	0.129459	1.000000	0.152590	0.057328
BloodPressure	0.141282	0.152590	1.000000	0.207371
SkinThickness	-0.081672	0.057328	0.207371	1.000000
Insulin	-0.073535	0.331357	0.088933	0.436783
BMI	0.017683	0.221071	0.281805	0.392573
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928
Age	0.544341	0.263514	0.239528	-0.113970
Outcome	0.221898	0.466581	0.065068	0.074752

```
                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.073535  0.017683                 -0.033523
Glucose                   0.331357  0.221071                  0.137337
BloodPressure             0.088933  0.281805                  0.041265
SkinThickness             0.436783  0.392573                  0.183928
Insulin                   1.000000  0.197859                  0.185071
BMI                       0.197859  1.000000                  0.140647
DiabetesPedigreeFunction  0.185071  0.140647                  1.000000
Age                      -0.042163  0.036242                  0.033561
Outcome                   0.130548  0.292695                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.263514  0.466581
BloodPressure             0.239528  0.065068
SkinThickness            -0.113970  0.074752
Insulin                  -0.042163  0.130548
BMI                       0.036242  0.292695
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

[78]:
```
feat=diabetes.columns[::-1]
feat
```

[78]:
```
Index(['Outcome', 'Age', 'DiabetesPedigreeFunction', 'BMI', 'Insulin',
       'SkinThickness', 'BloodPressure', 'Glucose', 'Pregnancies'],
      dtype='object')
```

[79]:
```
y=diabetes['Outcome']
```

[80]:
```
x=diabetes[feat]
x.head()
```

[80]:
```
   Outcome  Age  DiabetesPedigreeFunction   BMI  Insulin  SkinThickness  \
0        1   50                     0.627  33.6        0             35
1        0   31                     0.351  26.6        0             29
2        1   32                     0.672  23.3        0              0
3        0   21                     0.167  28.1       94             23
4        1   33                     2.288  43.1      168             35

   BloodPressure  Glucose  Pregnancies
0             72      148            6
1             66       85            1
2             64      183            8
3             66       89            1
4             40      137            0
```

[81]:
```python
ss=StandardScaler()
```

[82]:
```python
x_scaled=-ss.fit_transform(x)
```

[83]:
```python
x_train,x_test,y_train,y_test = train_test_split(x_scaled,y,test_size=0.
  ↪2,random_state=41)
```

[84]:
```python
x_train.shape
```

[84]:
```
(614, 9)
```

[85]:
```python
knn=KNeighborsClassifier(n_neighbors=3,algorithm = 'ball_tree',p=3)
knn.fit(x_train,y_train)
y_train_pred_knn=knn.predict(x_train)
y_test_pred_knn=knn.predict(x_test)
```

[86]:
```python
acc=accuracy_score(y_train,y_train_pred_knn)
print("Train accuracy ",acc)
acc=accuracy_score(y_test,y_test_pred_knn)
print("Test accuracy ",acc)
```

```
Train accuracy  0.996742671009772
Test accuracy  0.987012987012987
```

[87]:
```python
nb=GaussianNB()
nb.fit(x_train,y_train)
y_train_pred_nb=nb.predict(x_train)
y_test_pred_nb=nb.predict(x_test)
acc=accuracy_score(y_train,y_train_pred_nb)
print("Train accuracy ",acc)
acc=accuracy_score(y_test,y_test_pred_nb)
print("Test accuracy ",acc)
```

```
Train accuracy  1.0
Test accuracy  1.0
```

[88]:
```python
svm=SVC(kernel='rbf',C=5)
svm.fit(x_train,y_train)
```

[88]:
```
SVC(C=5)
```

[89]:
```python
y_train_pred_nb=svm.predict(x_train)
y_test_pred_nb=svm.predict(x_test)
```

[90]:
```python
acc=accuracy_score(y_train,y_train_pred_nb)
print("Train accuracy ",acc)
acc=accuracy_score(y_test,y_test_pred_nb)
```

```python
print("Test accuracy ",acc)
```

```
Train accuracy  1.0
Test accuracy  0.9935064935064936
```