```
import numpy as np
```

```
np.arange(19)
```

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

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

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

```
a = np.array(a)
print(type(a))
print(a.dtype)
```

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

```
np.arange(-3,3,0.5, dtype=int)
```

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

```
np.arange(-3,3,0.5, dtype=float)
```

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

```
l = [i**2 for i in range(100)]
```

```
l
```

```
[0,
 1,
 4,
 9,
 16,
 25,
 36,
 49,
 64,
 81,
 100,
 121,
 144,
 169,
 196,
 225,
 256,
 289,
 324,
 361,
 400,
 441,
 484,
 529,
 576,
 625,
 676,
 729,
 784,
 841,
 900,
 961,
 1024,
 1089,
 1156,
 1225,
 1296,
 1369,
 1444,
 1521,
 1600,
 1681,
 1764,
 1849,
 1936,
 2025,
 2116,
```

```
       2209,
       2304,
       2401,
       2500,
       2601,
       2704,
       2809,
       2916,
       3025,
       3136,
       3240
```

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

```
34.1 ms ± 4.95 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
a = np.arange(10000000)
%timeit a**2
```

```
44.6 ms ± 11.3 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

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

```
a.ndim
```

```
1
```

```
a % 2
```

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

```
a.shape
```

```
(5,)
```

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

```
b
```

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

```
b.ndim
```

```
2
```

```
b.shape
```

```
(2, 3)
```

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

```
(2, 3)
```

```
import pandas as pd
```

```
pd.DataFrame(b)
```

	0	1	2
0	1	2	3
1	4	5	6

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

```
x = np.array(x)
```

```
x.ndim
```

```
3
```

```
d = list(map(int, input().split()))
```

```
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)
```

```
b = np.linspace(1,4,10)
```

```
c = np.ones((4,5))
c
```

```
d = np.zeros((4,4))
```

```
e = np.eye(3,3)
```

```
np.diag(e)
```

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

```
h = np.random.rand(3)
h
h.dtype
```

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

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

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

Unsupported Cell Type. Double-Click to inspect/edit the content.

```
a
```

```
a[0] = 1000
```

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

```
c
```

```
np.shares_memory(a,c)
```

```
b = np.random.randint(20,30,10)
```

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

```
p = pd.read_csv("Salary_Data.csv")
```

```
p
```

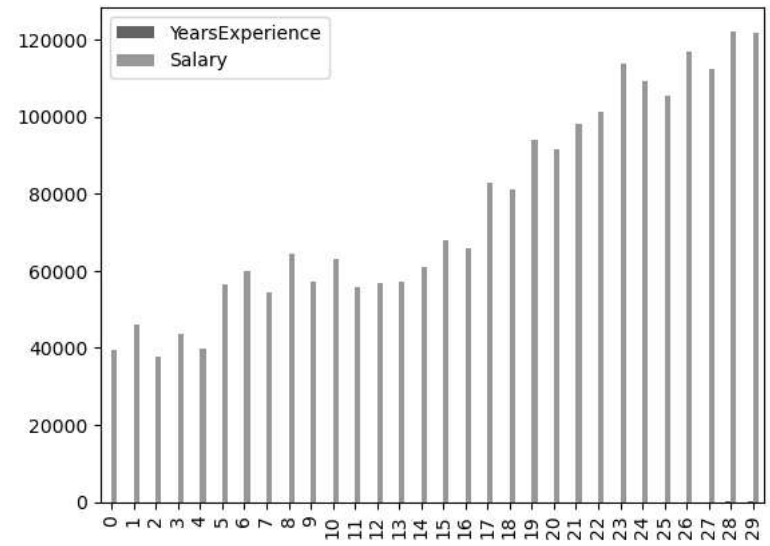
	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	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

```
p.plot()
```

```
<Axes: >
```


```
p.plot.bar()
```

```
<Axes: >
```




```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

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

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

```
ser_1.values
```

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

```
ser_1.index
```

```
RangeIndex(start=0, stop=7, step=1)
```

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

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

```
s2['a']
```

```
1
```

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

```
True
```

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

```
c    2
a    1
```

```
b    1
dtype: int64
```

```
s2
```

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

```
s2>0
```

```
a     True
b     True
c     True
d    False
e    False
dtype: bool
```

```
s2[s2>0]
```

```
a    1
b    1
c    2
dtype: int64
```

```
s2*2
```

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

```
np.exp(s2)
```

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

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

```
foo    100
bar    200
baz    300
dtype: int64
```

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

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

```
pd.isnull(s4).sum()
```

```
1
```

```
s4.isnull()
```

```
foo    False
bar    False
baz    False
qux     True
dtype: bool
```

```
s3 + s4
```

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

```
s4.name = 'foobarbazqux'
```

```
s4.index.name = 'label'
```

```
s4
```

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

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

```
s4
```

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

```
#DataFrame
```

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

```
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]}
```

	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

```
df1.describe()
```

	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

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

	year	state	pop
0	2012	VA	5.0
1	2013	VA	5.1

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

	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

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

```
0    VA
1    VA
2    VA
3    MD
4    MD
Name: state, dtype: object
```

```
df3['year']
```

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

```
df3.iloc[0]
```

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

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

	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

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

	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

```
df3['state_dp'] = df3['state']
df3
```

	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

```
del df3['state_dp']
df3
```

	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

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

```
df4 = DataFrame(pop)
df4
```

	VA	MD
2013	5.1	NaN
2014	5.2	4.0
2015	NaN	4.1

```
df4.T
```

	2013	2014	2015
VA	5.1	5.2	NaN
MD	NaN	4.0	4.1

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

	VA	MD
2014	5.2	NaN
2015	NaN	4.1

```
df5.index.name = 'year'
df5
```

year	VA	MD
2014	5.2	NaN
2015	NaN	4.1

```
df5.columns.name = 'state'
df5
```

state	VA	MD
year		
2014	5.2	NaN
2015	NaN	4.1

```
df5.values
```

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

```
df3.values
```

```
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)
```

```
df3
```

	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

```
df3.reindex(list(reversed(range(0,7))), fill_value = 0)
```

	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

```
df3.reindex(range(6,0), fill_value=0)
```

	year	state	pop	unempl

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

```
0    foo
2    bar
4    baz
dtype: object
```

```
s5.reindex(range(5),method='ffill')
```

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

```
s5.reindex(range(5),method='bfill')
```

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

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

	state	pop	unempl	year
0	VA	5.0	NaN	2012
1	VA	5.1	NaN	2013
2	VA	5.2	6.0	2014

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

	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

```
df6 = df3.loc[range(0,7),['state','pop','unempl','year']]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[160], line 1
----> 1 df6 = df3.loc[range(0,7),['state','pop','unempl','year']]

File ~\anaconda3\lib\site-packages\pandas\core\indexing.py:1067, in _LocationIndexer.__getitem__(self, key)
   1065     if self._is_scalar_access(key):
```

```
df3.shape
```

```
(5, 4)
```

```
   1070     axis = self.axis or 0
```

```
#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
```

	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

```
   1206     axis: self._get_listlike_indexer(key, axis)
```

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

	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

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

	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

SEARCH STACK OVERFLOW

```
df7 = df7.drop('unempl',axis = 0)
df7
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[114], line 1
----> 1 df7 = df7.drop('unempl',axis = 0)
      2 df7

File ~\anaconda3\lib\site-packages\pandas\util\_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.decorate.
<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:5399, in DataFrame.drop(self, labels, axis, index, columns, level,
inplace, errors)
   5251 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self", "labels"])
   5252 def drop(  # type: ignore[override]
   5253     self,
   (...)
   5260     errors: IgnoreRaise = "raise",
   5261 ) -> DataFrame | None:
   5262     """
   5263     Drop specified labels from rows or columns.
   5264
   (...)
   5397             weight  1.0     0.8
   5398     """
-> 5399     return super().drop(
   5400         labels=labels,
   5401         axis=axis,
   5402         index=index,
   5403         columns=columns,
   5404         level=level,
   5405         inplace=inplace,
   5406         errors=errors,
   5407     )

File ~\anaconda3\lib\site-packages\pandas\util\_decorators.py:331, in deprecate_nonkeyword_arguments.<locals>.decorate.
<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:4505, in NDFrame.drop(self, labels, axis, index, columns, level,
inplace, errors)
   4503 for axis, labels in axes.items():
   4504     if labels is not None:
-> 4505         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4507 if inplace:
   4508     self._update_inplace(obj)
```

```
s2
```

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

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

```
True
```

```
   6935     indexer = indexer[~mask]
```

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

```
b    1
c    2
d   -3
dtype: int64
```

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

```
b    1
c    2
d   -3
dtype: int64
```

```
s2[s2>0]
```

```
a    1
b    1
c    2
dtype: int64
```

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

```
a    1
b    1
dtype: int64
```

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

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

```
df6
```

	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

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

	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

```
df6[:3]
```

	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

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

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

```
df6
```

	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

```
df6[df6>5]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[146], line 1
----> 1 df6[df6>5]

File ~\anaconda3\lib\site-packages\pandas\core\ops\common.py:72, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
     68             return NotImplemented
     70 other = item_from_zerodim(other)
---> 72 return method(self, other)

File ~\anaconda3\lib\site-packages\pandas\core\arraylike.py:58, in OpsMixin.__gt__(self, other)
     56 @unpack_zerodim_and_defer("__gt__")
     57 def __gt__(self, other):
---> 58     return self._cmp_method(other, operator.gt)

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:7582, in DataFrame._cmp_method(self, other, op)
   7579 self, other = ops.align_method_FRAME(self, other, axis, flex=False, level=None)
   7581 # See GH#4537 for discussion of scalar op behavior
-> 7582 new_data = self._dispatch_frame_op(other, op, axis=axis)
   7583 return self._construct_result(new_data)

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:7621, in DataFrame._dispatch_frame_op(self, right, func, axis)
   7618 if not is_list_like(right):
   7619     # i.e. scalar, faster than checking np.ndim(right) == 0
   7620     with np.errstate(all="ignore"):
-> 7621         bm = self._mgr.apply(array_op, right=right)
   7622     return self._constructor(bm)
   7624 elif isinstance(right, DataFrame):

File ~\anaconda3\lib\site-packages\pandas\core\internals\managers.py:350, in BaseBlockManager.apply(self, f, align_keys,
ignore_failures, **kwargs)
    348 try:
    349     if callable(f):
--> 350         applied = b.apply(f, **kwargs)
    351     else:
    352         applied = getattr(b, f)(**kwargs)

File ~\anaconda3\lib\site-packages\pandas\core\internals\blocks.py:351, in Block.apply(self, func, **kwargs)
    345 @final
    346 def apply(self, func, **kwargs) -> list[Block]:
    347     """
    348     apply the function to my values; return a block if we are not
    349     one
    350     """
--> 351     result = func(self.values, **kwargs)
    353     return self._split_op_result(result)

File ~\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py:287, in comparison_op(left, right, op)
    284     return invalid_comparison(lvalues, rvalues, op)
    286 elif is_object_dtype(lvalues.dtype) or isinstance(rvalues, str):
--> 287     res_values = comp_method_OBJECT_ARRAY(op, lvalues, rvalues)
    289 else:
    290     res_values = _na_arithmetic_op(lvalues, rvalues, op, is_cmp=True)

File ~\anaconda3\lib\site-packages\pandas\core\ops\array_ops.py:75, in comp_method_OBJECT_ARRAY(op, x, y)
     73     result = libops.vec_compare(x.ravel(), y.ravel(), op)
     74 else:
---> 75     result = libops.scalar_compare(x.ravel(), y, op)
     76 return result.reshape(x.shape)

File ~\anaconda3\lib\site-packages\pandas\_libs\ops.pyx:107, in pandas._libs.ops.scalar_compare()

TypeError: '>' not supported between instances of 'str' and 'int'
```

SEARCH STACK OVERFLOW

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

```
df8
```

	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

```
df8 > 5
```

	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

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

	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

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

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

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

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

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

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

```
ser_6 + ser_7
```

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

```
ser_6.add(ser_7, fill_value=0)
```

```
import numpy as np
import pandas as pd
movies = pd.read_csv("/content/drive/MyDrive/movies.dat", delimiter='::')
print(movies.head())
```

```
   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
C:\Users\DSAI\AppData\Local\Temp\ipykernel_17216\4236904445.py:3: ParserWarning:
  movies = pd.read_csv("C:/Users/DSAI/Desktop/21STUCHH010385_lab/Datasets_385/MC
```

```
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/ratings.dat", delimiter='::')
print(ratings.head())
```

```
C:\Users\DSAI\AppData\Local\Temp\ipykernel_17216\1822516052.py:1: ParserWarning:
  ratings = pd.read_csv("C:/Users/DSAI/Desktop/21STUCHH010385_lab/Datasets_385/M
   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]
```

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

```
df=pd.read_csv('/content/drive/MyDrive/iris.csv')
```

```
df.head()
```

	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

```
df.tail()
```

	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

```
df.shape
```

```
(150, 5)
```

```
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
```

```
df.isnull().sum()
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

```
df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000

```
df['species'].unique()
```

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

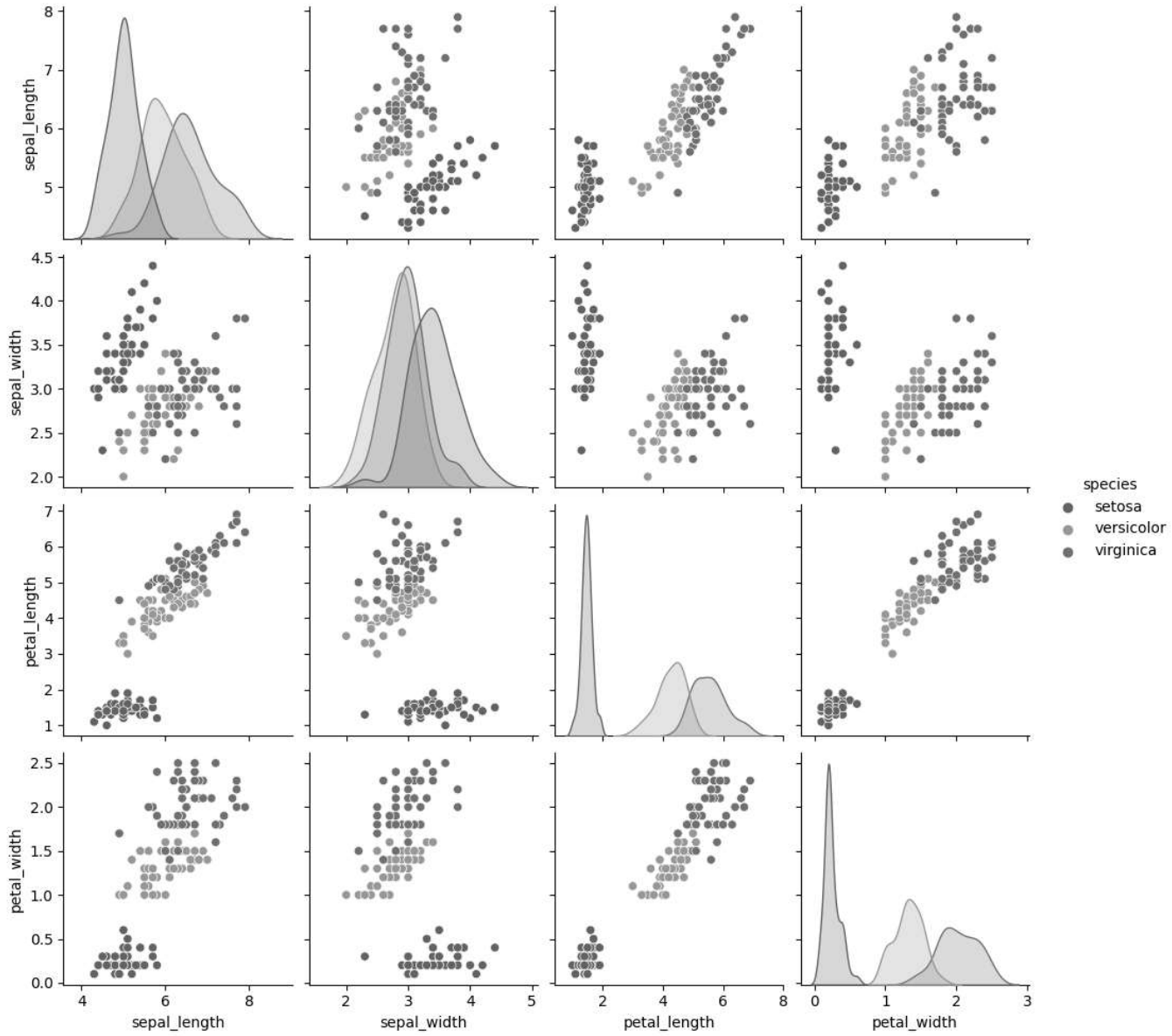
min	4.300000	2.000000	1.000000	0.100000

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

```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

```
sns.pairplot(df,hue='species')
```

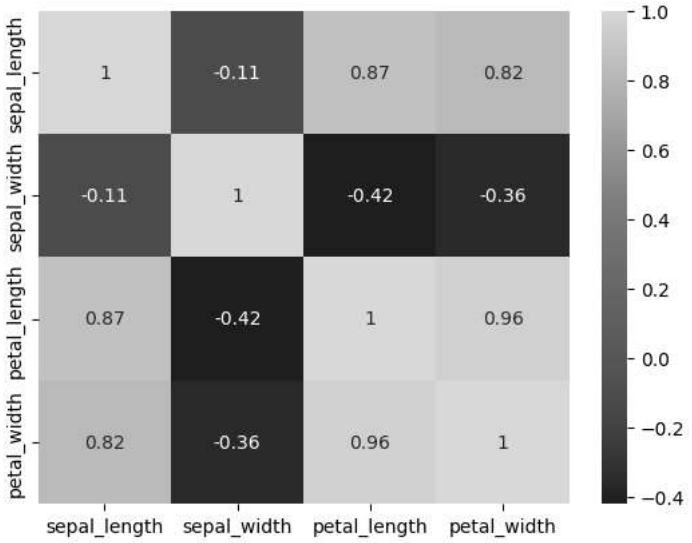
```
<seaborn.axisgrid.PairGrid at 0x7b416d7d7790>
```




```
df.corr()
```

```
sns.heatmap(df.corr(),annot=True,cmap='viridis')
```

```
<Axes: >
```




```
X=df.drop(['species'],axis=1)
```

```
y=df['species']
```

```
X.shape
```

```
(150, 4)
```

```
y.shape
```

```
(150,)
```

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

```
X_train.shape
y_train.shape
```

```
(120,)
```

```
X_test.shape
y_test.shape
```

```
(30,)
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
DTC=DecisionTreeClassifier()
```

```
DTC.fit(X_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
prediction=DTC.predict(X_test)
```

```
prediction
```

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

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

	Actual	Prediction
73	versicolor	versicolor
18	setosa	setosa
118	virginica	virginica
78	versicolor	versicolor
76	versicolor	versicolor
31	setosa	setosa
64	versicolor	versicolor
141	virginica	virginica
68	versicolor	versicolor
82	versicolor	versicolor
110	virginica	virginica
12	setosa	setosa
36	setosa	setosa
9	setosa	setosa
19	setosa	setosa
56	versicolor	versicolor
104	virginica	virginica
69	versicolor	versicolor
55	versicolor	versicolor
132	virginica	virginica
29	setosa	setosa
127	virginica	virginica
26	setosa	setosa
128	virginica	virginica
131	virginica	virginica
145	virginica	virginica
108	virginica	virginica
143	virginica	virginica
45	setosa	setosa
30	setosa	setosa

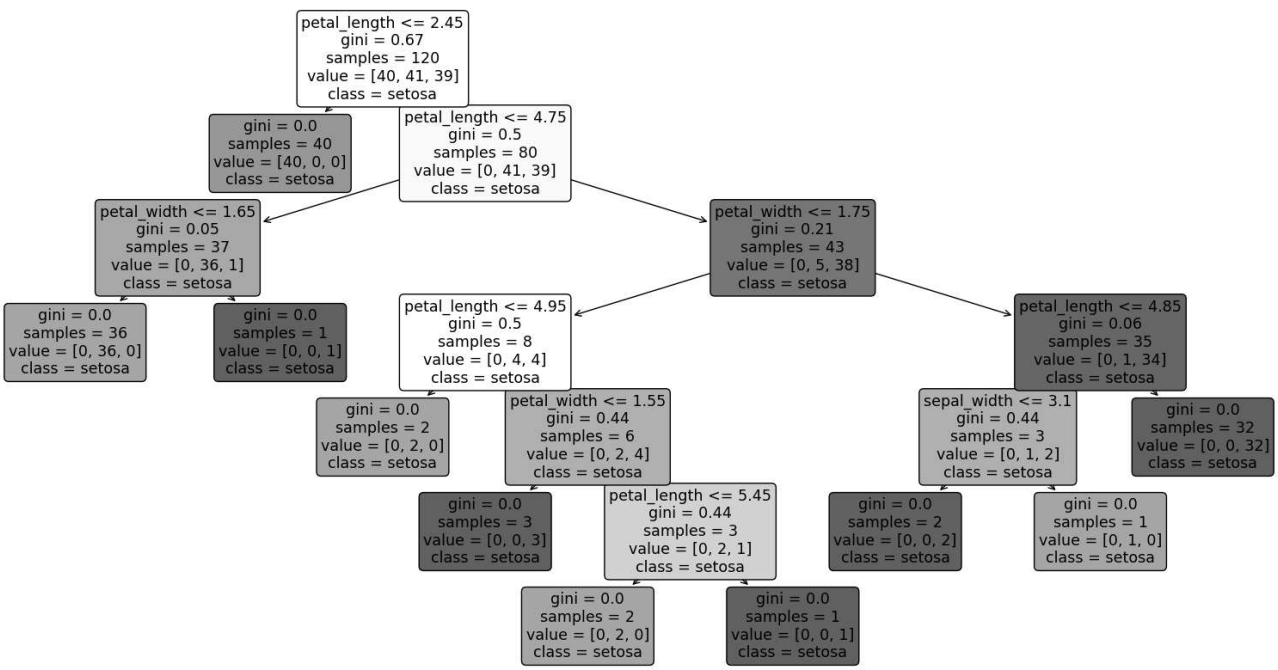
```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

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

print(confusion_matrix(y_test,prediction))
```

```
                  precision    recall  f1-score   support

           setosa       1.00      1.00      1.00        10
       versicolor       1.00      1.00      1.00         9
        virginica       1.00      1.00      1.00        11

         accuracy                           1.00        30
        macro avg       1.00      1.00      1.00        30
     weighted avg       1.00      1.00      1.00        30

     [[10  0  0]
      [ 0  9  0]
      [ 0  0 11]]
```

```
Accuracy = accuracy_score(y_test,prediction)
```

```
Precision = precision_score(y_test, prediction,average='weighted')


Sensitivity_recall = recall_score(y_test, prediction,average='weighted')


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)
```



```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from sklearn import metrics
```

```
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).
```

```
df=pd.read_csv("/content/drive/MyDrive/Play Tennis.csv")
value=['Outlook','Temprature','Humidity','Wind']
df
```

	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
5	D6	Rain	Cool	Normal	Strong	No
6	D7	Overcast	Cool	Normal	Strong	Yes
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes
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

```
len(df)
```

```
14
```

```
df.shape
```

```
(14, 6)
```

```
df.head()
```

	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

```
df.tail()
```

	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

```
df.describe()
```

	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

```
from sklearn import preprocessing
string_to_int= preprocessing.LabelEncoder()
df=df.apply(string_to_int.fit_transform)
df
```

	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

```
feature_cols = ['Outlook','Temprature','Humidity','Wind']
X = df[feature_cols ]
y = df.Play_Tennis
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

```
from sklearn.tree import DecisionTreeClassifier
classifier =DecisionTreeClassifier(criterion="entropy", random_state=100)
classifier.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=100)
```

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

```
type(X_test)
```

```
pandas.core.frame.DataFrame
```

```
data_1 = {'state' : ['VA', 'VA', 'VA', 'MD', 'MD'],
          'year' : [2012, 2013, 2014, 2014, 2015],
          'pop' : [5.0, 5.1, 5.2, 4.0, 4.1]}
df_1 = DataFrame(data_1)
df_1
```

	state	year	pop
0	VA	2012	5.0
1	VA	2013	5.1

```
data_2 = {'Outlook' : ['1'], 'Temprature' : ['2'], 'Humidity' : ['9'], 'Wind' : ['9']}
df_2 = DataFrame(data_2)
df_2
```

	Outlook	Temprature	Humidity	Wind
0	1	2	9	9

```
y_pred2= classifier.predict(df_2)
y_pred2
```

```
array([1])
```

```
from sklearn.metrics import accuracy_score
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.4
```

```
data_p=pd.DataFrame({'Actual':y_test, 'Predicted':y_pred})
data_p
```

	Actual	Predicted
9	1	1
11	1	0
0	0	0
8	1	0
6	1	0

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

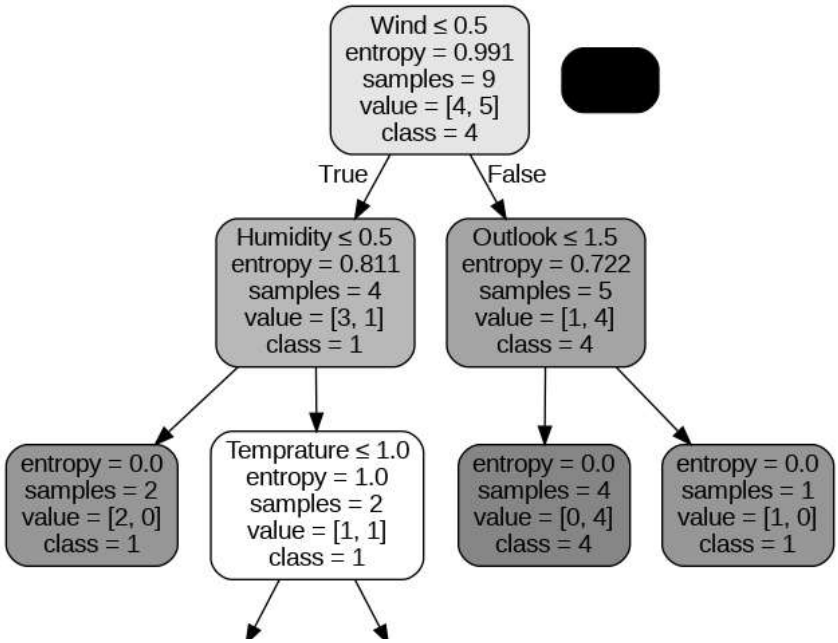
```
[[1 0]
 [3 1]]
              precision    recall  f1-score   support

           0       0.25      1.00      0.40         1
           1       1.00      0.25      0.40         4

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

```
from sklearn.tree import export_graphviz
```

```
from six import StringIO
```

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

```
dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data,
filled=True, rounded=True,
special_characters=True,feature_names =value,class_names=['1','4'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Play Tennis.png')
Image(graph.create_png())
```




```
print(X_test)
```

```
    Outlook  Temprature  Humidity  Wind
9         1           2         1     1
11        0           2         0     0
0         2           1         0     1
8         2           0         1     1
6         0           0         1     0
```

```
print(X_train)
```

```
    Outlook  Temprature  Humidity  Wind
3         1           2         0     1
12        0           1         1     1
10        2           2         1     0
4         1           0         1     1
13        1           2         0     0
5         1           0         1     0
1         2           1         0     0
2         0           1         0     1
7         2           2         0     1
```

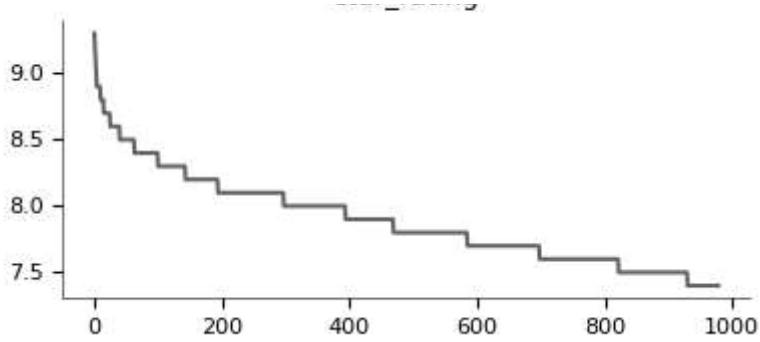
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.svm import SVR
from sklearn import tree


movies=pd.read_csv( '/content/imdbratings (1).csv')



movies
```

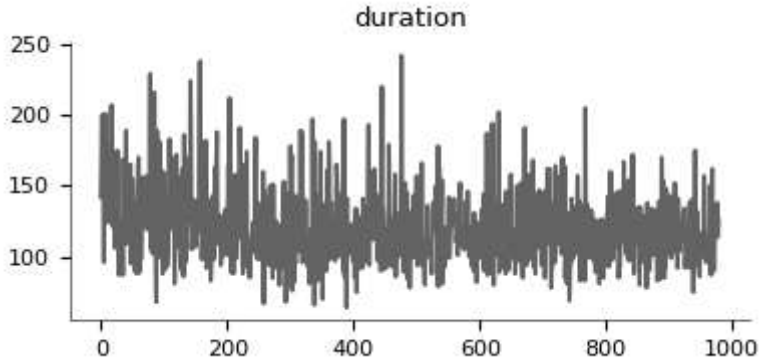
	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
...	...	...	...	...	...	...
974	7.4	Tootsie	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
977	7.4	Poltergeist	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

979 rows × 6 columns

Values

star_rating





Distributions


```
movies. head()
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....

```
movies.columns
```

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

```
movies.isnull().sum()
```

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

```
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
```

```
movies.iloc[187,2]='pg13'
movies.iloc[649,2]='pg'
movies.iloc[936,2]='pg13'
```

```
movies.drop([ 'title'],axis=1,inplace=True)
```

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

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

```
<ipython-input-29-305901486a81>:1: DeprecationWarning: `np.object` is a deprecated al
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re
  categorical_features=[i for i in movies.select_dtypes(include=np.object)]
```

```
dummy_df=pd.DataFrame()
```

```
dummy_df['duration']=movies.duration
```

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

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

```
train_df.head()
```

	Action	Adventure	Animation	Biography	Comedy	Crime	Drama	Family	Fantasy	Fil Nc
0	0	0	0	0	0	1	0	0	0	
1	0	0	0	0	0	1	0	0	0	
2	0	0	0	0	0	1	0	0	0	

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

```
train_df.shape
```

```
(979, 18)
```

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

```
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2, random_state=42)
```

```
LR=LinearRegression()
```

```
LR.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred=LR.predict(x_test)
```

```
print('RMSE using Linear regression is', metrics.mean_squared_error(y_test,y_pred,sample_we
```

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

```
sv=SVR()
```

```
sv.fit(x_train,y_train)
```

```
▾ SVR
SVR()
```

```
sv_pred=sv.predict(x_test)
```

```
print('RMSE using SVR is', metrics.mean_squared_error(y_test,sv_pred,sample_weight=
```

```
RMSE using SVR is 0.09749560506058148
```

```
clf=tree.DecisionTreeRegressor()
```

```
clf.fit(x_train,y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
DT_pred=clf.predict(x_test)
```

```
print('RMSE using DT is', metrics.mean_squared_error(y_test,DT_pred,sample_weight=None))
```

```
RMSE using DT is 0.18168370181405893
```

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

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

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

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
path = '/content/drive/MyDrive/IMDB Dataset (3).csv'
```

```
imdb = pd.read_csv(path, encoding="latin-1")
```

```
imdb.shape
```

```
(50000, 2)
```

```
imdb.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. 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

```
imdb['review'][1]
review = imdb['review']
labels = imdb['sentiment']
```

```
#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)

#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
```

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

```
processedReviews[1]

vectorizer = CountVectorizer(analyzer='word') # Convert a collection of text docume
featurevector = vectorizer.fit_transform(processedReviews)

featurevector.shape
```

```
     (50000, 162851)
```

```
from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]


vectorizer = CountVectorizer()


X = vectorizer.fit_transform(corpus)


vectorizer.get_feature_names_out()


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


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]]


 vectorizer2 = CountVectorizer(analyzer='word', ngram_range=(2, 2))


X2 = vectorizer2.fit_transform(corpus)


 vectorizer2.get_feature_names_out()

     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)


 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]]


X_train, X_test, y_train, y_test = train_test_split(featurevector, labels, test_size=0.30,



X_train.shape, X_test.shape, y_train.shape, y_test.shape


     ((35000, 162851), (15000, 162851), (35000,), (15000,))
```

```
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
```

```
dt = DecisionTreeClassifier(max_depth = 15)
```

```
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=15)
```

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

```
Train Accuracy:  0.8045142857142857
```

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

```
Test Accuracy:  0.7432
```

```
from sklearn.feature_extraction.text import TfidfVectorizer    # tf-idf method
#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)
```

```
tfidf_feature.get_shape()
```

```
(50000, 162851)
```

```
feature_names = tfidf.get_feature_names_out()
len(feature_names)
```

```
162851
```

```
feature_names[:10]
```

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

```
X_train, X_test, y_train, y_test = train_test_split(tfidf_feature, labels, test_size=0.30,
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((35000, 162851), (15000, 162851), (35000,), (15000,))
```

```
dt = DecisionTreeClassifier(max_depth = 15)
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=15)
```

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

```
Train Accuracy:  0.806
```

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

```
Test Accuracy:  0.7336
```

```
from sklearn.linear_model import LogisticRegression

logit = LogisticRegression()

logit.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

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

```
Training Accuracy : 0.9311714285714285
```

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

```
Testing Accuracy : 0.8978
```

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

```
import numpy as np
X = np.array([[0, 0],[0,1],[1,0],[1,1]])
y = np.array([[0], [0], [0], [1]])
p_model_and = Perceptron(X.shape[1], neta = 0.01)
p_model_and.fit(X, y, epochs= 50)
```

```
p_model_and.w
```

```
array([ 0.6675719 ,  0.10752548, -0.75202372])
```

```
for (x, t) in zip(X, y):
  pred = p_model_and.predict(x)
  print(f"Data: {x}, Target: {t}, predicted: {pred}")
```

```
Data: [0 0], Target: [0], predicted: 0
Data: [0 1], Target: [0], predicted: 0
Data: [1 0], Target: [0], predicted: 0
Data: [1 1], Target: [1], predicted: 1
```

```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[1]])
p_model_or = Perceptron(X.shape[1], neta = 0.1)
p_model_or.fit(X, y, epochs= 50)
```

```
for (x, t) in zip(X, y):
  pred = p_model_or.predict(x)
  print(f"Data: {x}, Target: {t}, predicted: {pred}")
```

```
Data: [0 0], Target: [0], predicted: 0
Data: [0 1], Target: [1], predicted: 1
Data: [1 0], Target: [1], predicted: 1
Data: [1 1], Target: [1], predicted: 1
```

```
X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])
p_model_xor = Perceptron(X.shape[1], neta = 0.1)
p_model_xor.fit(X, y, epochs= 50)
for (x, t) in zip(X, y):
  pred = p_model_xor.predict(x)
  print(f"Data: {x}, Target: {t}, predicted: {pred}")
```

```
Data: [0 0], Target: [0], predicted: 1
Data: [0 1], Target: [1], predicted: 1
Data: [1 0], Target: [1], predicted: 0
Data: [1 1], Target: [0], predicted: 0
```

```
from sklearn.linear_model import Perceptron
from sklearn.datasets import load_digits
X, y = load_digits(return_X_y = True)
p = Perceptron()
p.fit(X,y)
print(p.score(X, y))
```

```
0.9393433500278241
```

```
x = np.arange(36).reshape(-1, 9)
X
```

```
array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 10.,  0.,  0.],
       [ 0.,  0.,  0., ..., 16.,  9.,  0.],
       ...,
       [ 0.,  0.,  1., ...,  6.,  0.,  0.],
       [ 0.,  0.,  2., ..., 12.,  0.,  0.],
       [ 0.,  0., 10., ..., 12.,  1.,  0.]])
```

```
x[0]
```

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

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

```
(9,)
```

```
x.shape
```

```
(4, 9)
```

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

```
4
```

```
name = [ "Manjeet", "Nikhil", "shambhavi", "Astha"]
roll_no = [ 4, 1, 3, 2]
mapped = zip(name, roll_no)
print(set(mapped))
```

```
{('shambhavi', 3), ('Manjeet', 4), ('Astha', 2), ('Nikhil', 1)}
```

```
import numpy as np
in_num = 10
print("Input number : ", in_num)
out_arr = np.atleast_2d(in_num)
print("output 2d arrary from input number: ", out_arr)
```

```
Input number :  10
output 2d arrary from input number:  [[10]]
```

```
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
import numpy as np



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']


model = Perceptron(alpha=0.0001, class_weight=None, eta0=1.0, fit_intercept=True, max_iter=1000, n_jobs=1, penalty=None, random_state=0,


model.fit(data, results)
```

```
Perceptron
Perceptron(n_jobs=1)
```

```
predicted_results = model.predict(data)
acc_per = accuracy_score(results, predicted_results) * 100
print('Accuracy for perception: {} %'.format(acc_per))
```

```
Accuracy for perception: 54.54545454545454 %
```

```
prediction = model.predict([[1.62, 0.49, 0.38]])
print(prediction)
```

```
['male']
```

```
prediction = model.predict([[1.60, 0.60, 0.38]])
print(prediction)
```

```
['male']
```

```
prediction = model.predict([[1.59, 0.55, 0.37]])
print(prediction)
```

```
['male']
```

```
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import Perceptron
from sklearn.neighbors import NearestNeighbors


methods = ['Decision Trees', 'SVM', 'Perceptron', 'K nearest neighbour']


x = [[181, 80, 44], [177, 70, 43], [160,60, 38], [154, 54, 37], [166, 65, 40], [190, 90, 47], [175, 64, 39],
     [177,70, 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_percept = Perceptron()
clf_KNN = NearestNeighbors()


clf_tree = clf_tree.fit(x,y)
clf_svm = clf_svm.fit(x,y)
clf_percept = clf_percept.fit(x,y)
clf_KNN = clf_KNN.fit(x,y)


clf_tree_prediction = clf_tree.predict(x)
acc_tree = accuracy_score(y, clf_tree_prediction)*100
```

```
print("Accuracy using Decision Trees:"), acc_tree, "%"
```

```
Accuracy using Decision Trees:
(None, 100.0, '%')
```

```
clf_svm_prediction = clf_svm.predict(x)
acc_svm = accuracy_score(y, clf_svm_prediction)*100
print("Accuracy using Decision Trees:"), acc_svm, "%"
```

```
Accuracy using Decision Trees:
(None, 54.54545454545454, '%')
```

```
clf_percept_prediction = clf_percept.predict(x)
acc_percept = accuracy_score(y, clf_percept_prediction)*100
print("Accuracy using Decision Trees:"), acc_percept, "%"
```

```
Accuracy using Decision Trees:
(None, 45.45454545454545, '%')
```

```
distances, indices = clf_KNN.kneighbors(x)
new_label = indices[:,0]
clf_KNN_prediction = [y[i][:] for i in new_label]
acc_knn = accuracy_score(y, clf_KNN_prediction)*100
print("Labels for training set using k-nearst neighbour:"),acc_knn, "%"
```

```
Labels for training set using k-nearst neighbour:
(None, 100.0, '%')
```

```
acc_all = [acc_tree,acc_svm,acc_per,acc_knn]
```

```
score_bestmethod = np.max(acc_all)
best_method = np.argmax(acc_all)
```

```
print (methods[best_method], "is the best method with accuracy of"), score_bestmethod, "%"
```

```
Decision Trees is the best method with accuracy of
(None, 100.0, '%')
```

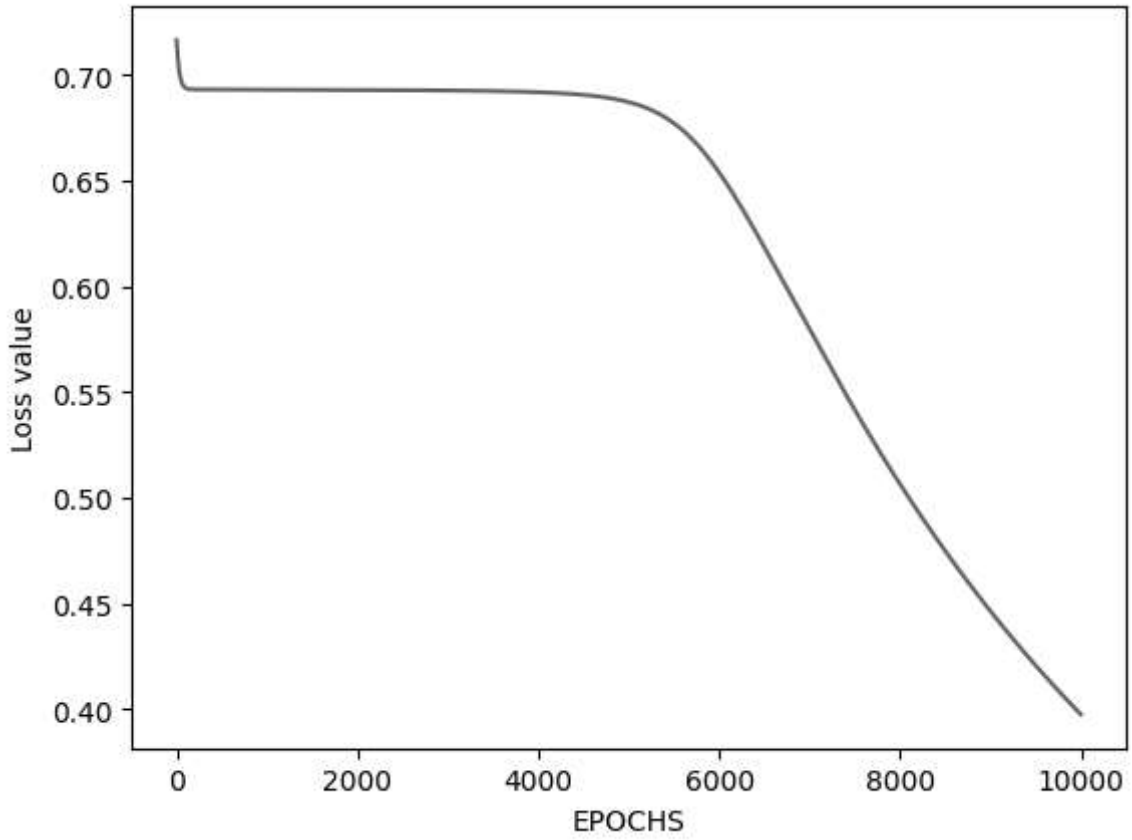
```
import numpy as np
import matplotlib.pyplot as plt


x=np.array([[0,0,1,1],[0,1,0,1]])
y=np.array([[0,1,1,0]])



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)
losses = []
```

+ Code + Text

```
def sigmoid(z):
  z= 1/(1+np.exp(-z))
  return z


def forward_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
```

```
iterations = 10000
for i in range(iterations):
  z1,a1,z2,a2 = forward_prop(w1,w2,x)
  loss = -(1/m)*np.sum(y*np.log(a2)+(1-y)*np.log(1-a2))
  losses.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(losses)
plt.xlabel("EPOCHS")
plt.ylabel("Loss value")
```

```
Text(0, 0.5, 'Loss value')
```




```
def predict(w1,w2,input):
  z1,a1,z2,a2 = forward_prop(w1,w2,test)
  a2 = np.squeeze(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")


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
```

```
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
```

```
(x_train, y_train),(x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

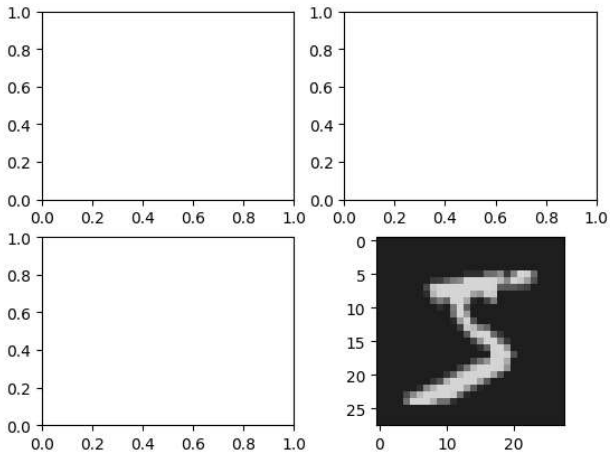
```
print("number of Training example: ", x_train.shape)
print("number of Training example target: ", y_train.shape)
print("number of testing example: ", x_test.shape)
print("number of Testing example target: ", y_test.shape)
```

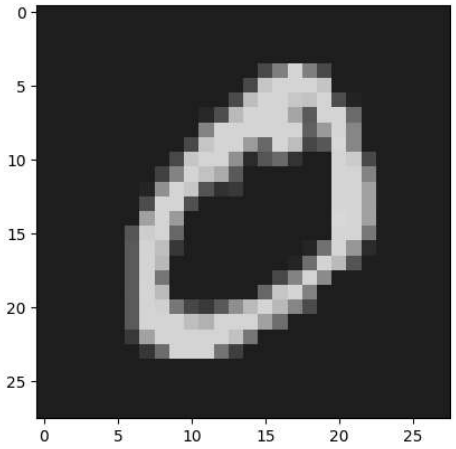
```
number of Training example:  (60000, 28, 28)
number of Training example target:  (60000,)
number of testing example:  (10000, 28, 28)
number of Testing example target:  (10000,)
```

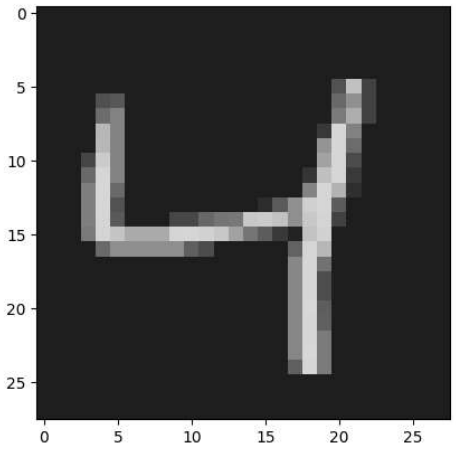
```
print(x_train[0])
```

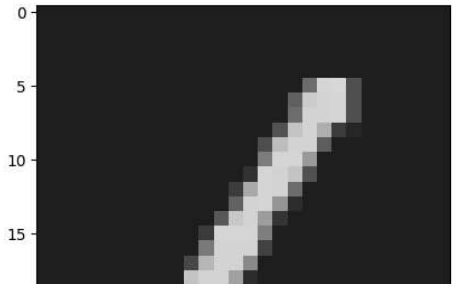
```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 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

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









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

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



```
x_train = x_train / 255
x_test = x_test / 255
```

```
x_train[0]
```

```
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,
```

```
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

```
y_train.shape
```

```
(60000, 10)
```

```
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"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 256)               200960

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 10)                650

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

```
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)
```

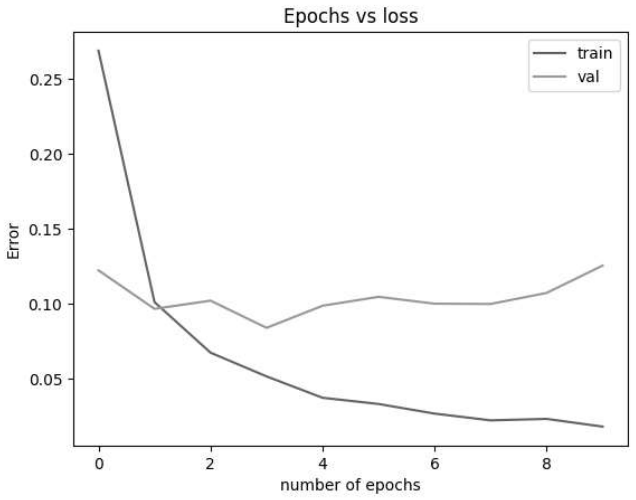
```
Epoch 1/10
750/750 [==============================] - 7s 8ms/step - loss: 0.2691 - accuracy: 0.9198 - val_loss: 0.1223 - val_accuracy: 0.9623
Epoch 2/10
750/750 [==============================] - 5s 6ms/step - loss: 0.1011 - accuracy: 0.9691 - val_loss: 0.0966 - val_accuracy: 0.9708
Epoch 3/10
750/750 [==============================] - 6s 8ms/step - loss: 0.0673 - accuracy: 0.9789 - val_loss: 0.1021 - val_accuracy: 0.9708
Epoch 4/10
750/750 [==============================] - 4s 6ms/step - loss: 0.0515 - accuracy: 0.9834 - val_loss: 0.0839 - val_accuracy: 0.9747
Epoch 5/10
750/750 [==============================] - 5s 6ms/step - loss: 0.0372 - accuracy: 0.9882 - val_loss: 0.0987 - val_accuracy: 0.9740
Epoch 6/10
750/750 [==============================] - 6s 8ms/step - loss: 0.0330 - accuracy: 0.9893 - val_loss: 0.1047 - val_accuracy: 0.9729
Epoch 7/10
750/750 [==============================] - 5s 6ms/step - loss: 0.0266 - accuracy: 0.9912 - val_loss: 0.1001 - val_accuracy: 0.9737
Epoch 8/10
750/750 [==============================] - 6s 8ms/step - loss: 0.0221 - accuracy: 0.9927 - val_loss: 0.0999 - val_accuracy: 0.9761
```

```
Epoch 9/10
750/750 [==============================] - 5s 6ms/step - loss: 0.0231 - accuracy: 0.9924 - val_loss: 0.1072 - val_accuracy: 0.9737
Epoch 10/10
750/750 [==============================] - 5s 6ms/step - loss: 0.0179 - accuracy: 0.9943 - val_loss: 0.1255 - val_accuracy: 0.9720
```

```
train_history.history.keys()
```

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

```
plt.plot(train_history.history['loss'])
plt.plot(train_history.history['val_loss'])
plt.title("Epochs vs loss")
plt.xlabel("number of epochs")
plt.ylabel("Error")
plt.legend(['train', 'val'])
plt.show()
```




```
score = model.evaluate(x_test, y_test, batch_size = 64)
```

```
157/157 [==============================] - 0s 3ms/step - loss: 0.1009 - accuracy: 0.9759
```

```
print("testing accuracy: ", score[1])
```

```
testing accuracy:  0.9758999943733215
```

```
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 tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
```

```
lfw=fetch_lfw_people(min_faces_per_person=100)
```

```
n_samples,h,w=lfw.images.shape
print("number of samples faces and its height and width:",n_samples,h,w)
```

```
number of samples faces and its height and width: 1140 62 47
```

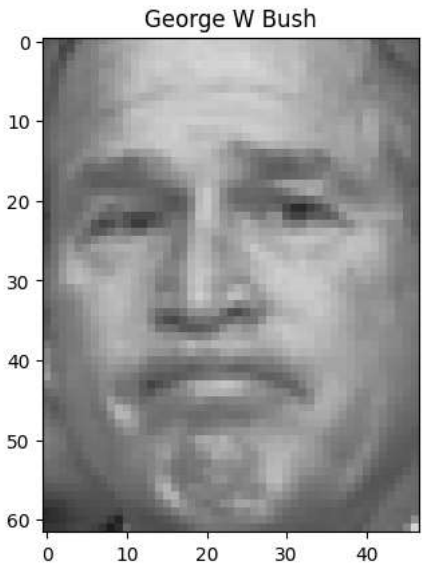
```
X=lfw.data
y=lfw.target
target_names=lfw.target_names
print("input data shape:",X.shape)
print("target length:",len(y))
print("target_names:",target_names)
```

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

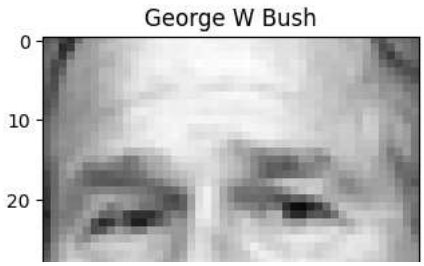
```
X[0]
```

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

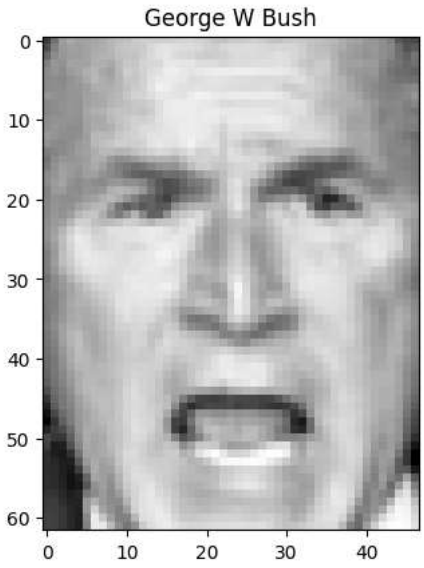
```
plt.imshow(lfw.images[0])
plt.title(target_names[y[0]])
plt.show()
```




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

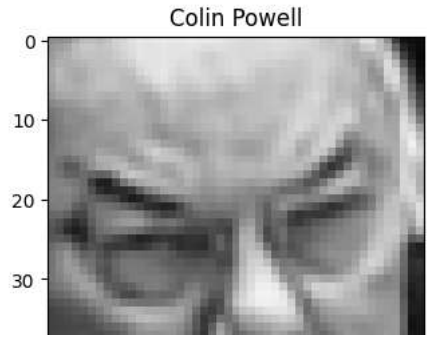



```
plt.imshow(lfw.images[100],cmap='gray')
plt.title(target_names[y[100]])
plt.show()
```




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


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




```
target_names.shape[0]
```

```
5
```

```
X.shape[0]
```

```
1140
```



```
X.shape[1]
```

```
2914
```

```
target_names.shape[0]
```

```
5
```

```
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()
```

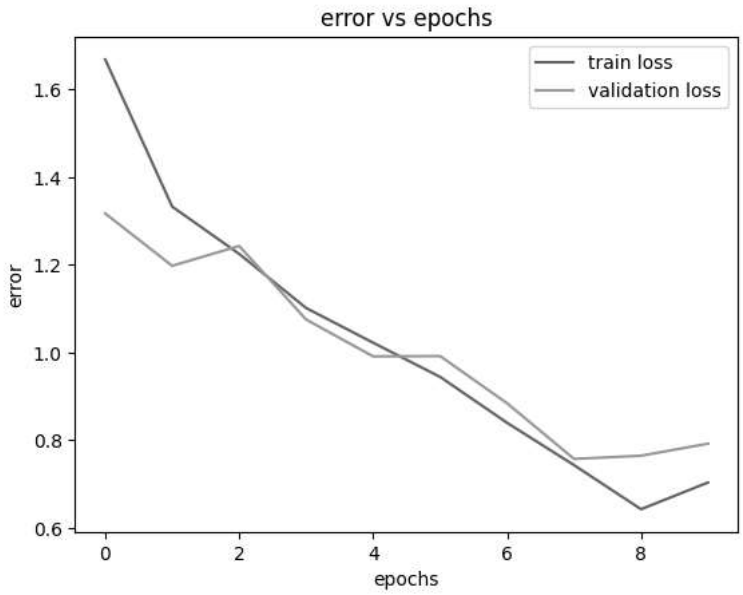
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               746240

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 5)                 645

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

```
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 [==============================] - 2s 21ms/step - loss: 1.6685 - accuracy: 0.3980 - val_loss: 1.3173 - val_accuracy: 0.5000
Epoch 2/10
29/29 [==============================] - 0s 14ms/step - loss: 1.3330 - accuracy: 0.4945 - val_loss: 1.1980 - val_accuracy: 0.5395
Epoch 3/10
29/29 [==============================] - 0s 15ms/step - loss: 1.2251 - accuracy: 0.5691 - val_loss: 1.2432 - val_accuracy: 0.5395
Epoch 4/10
29/29 [==============================] - 0s 13ms/step - loss: 1.1017 - accuracy: 0.5976 - val_loss: 1.0755 - val_accuracy: 0.6053
Epoch 5/10
29/29 [==============================] - 0s 12ms/step - loss: 1.0226 - accuracy: 0.6162 - val_loss: 0.9914 - val_accuracy: 0.6272
Epoch 6/10
29/29 [==============================] - 0s 13ms/step - loss: 0.9446 - accuracy: 0.6371 - val_loss: 0.9921 - val_accuracy: 0.6184
Epoch 7/10
29/29 [==============================] - 0s 14ms/step - loss: 0.8399 - accuracy: 0.6897 - val_loss: 0.8845 - val_accuracy: 0.6842
Epoch 8/10
29/29 [==============================] - 0s 13ms/step - loss: 0.7434 - accuracy: 0.7357 - val_loss: 0.7576 - val_accuracy: 0.7018
Epoch 9/10
29/29 [==============================] - 0s 14ms/step - loss: 0.6429 - accuracy: 0.7708 - val_loss: 0.7649 - val_accuracy: 0.6711
Epoch 10/10
29/29 [==============================] - 0s 13ms/step - loss: 0.7038 - accuracy: 0.7412 - val_loss: 0.7924 - val_accuracy: 0.7061
```

```
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()
```




```
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)
```

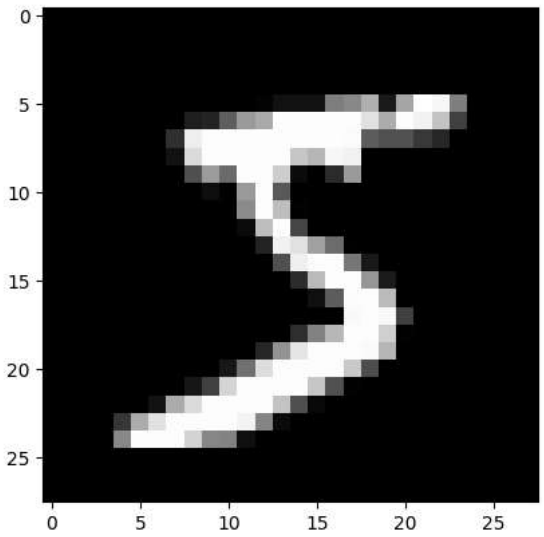
```
Epoch 1/25
15/15 [==============================] - 1s 31ms/step - loss: 0.6168 - accuracy: 0.7588 - val_loss: 0.7905 - val_accuracy: 0.7061
Epoch 2/25
15/15 [==============================] - 0s 18ms/step - loss: 0.6150 - accuracy: 0.7588 - val_loss: 0.7886 - val_accuracy: 0.7061
Epoch 3/25
15/15 [==============================] - 0s 18ms/step - loss: 0.6133 - accuracy: 0.7599 - val_loss: 0.7867 - val_accuracy: 0.7061
Epoch 4/25
15/15 [==============================] - 0s 18ms/step - loss: 0.6115 - accuracy: 0.7610 - val_loss: 0.7848 - val_accuracy: 0.7061
Epoch 5/25
15/15 [==============================] - 0s 18ms/step - loss: 0.6097 - accuracy: 0.7610 - val_loss: 0.7830 - val_accuracy: 0.7061
Epoch 6/25
15/15 [==============================] - 0s 19ms/step - loss: 0.6080 - accuracy: 0.7610 - val_loss: 0.7810 - val_accuracy: 0.7061
Epoch 7/25
15/15 [==============================] - 0s 18ms/step - loss: 0.6062 - accuracy: 0.7632 - val_loss: 0.7791 - val_accuracy: 0.7061
Epoch 8/25
15/15 [==============================] - 0s 24ms/step - loss: 0.6044 - accuracy: 0.7664 - val_loss: 0.7771 - val_accuracy: 0.7061
Epoch 9/25
15/15 [==============================] - 0s 30ms/step - loss: 0.6026 - accuracy: 0.7675 - val_loss: 0.7753 - val_accuracy: 0.7105
Epoch 10/25
15/15 [==============================] - 0s 23ms/step - loss: 0.6009 - accuracy: 0.7697 - val_loss: 0.7735 - val_accuracy: 0.7061
Epoch 11/25
15/15 [==============================] - 0s 18ms/step - loss: 0.5992 - accuracy: 0.7719 - val_loss: 0.7716 - val_accuracy: 0.7061
Epoch 12/25
15/15 [==============================] - 0s 19ms/step - loss: 0.5974 - accuracy: 0.7730 - val_loss: 0.7698 - val_accuracy: 0.7061
Epoch 13/25
15/15 [==============================] - 0s 19ms/step - loss: 0.5958 - accuracy: 0.7741 - val_loss: 0.7680 - val_accuracy: 0.7105
Epoch 14/25
15/15 [==============================] - 0s 18ms/step - loss: 0.5940 - accuracy: 0.7752 - val_loss: 0.7662 - val_accuracy: 0.7105
Epoch 15/25
15/15 [==============================] - 0s 18ms/step - loss: 0.5923 - accuracy: 0.7774 - val_loss: 0.7642 - val_accuracy: 0.7149
Epoch 16/25
15/15 [==============================] - 0s 21ms/step - loss: 0.5906 - accuracy: 0.7785 - val_loss: 0.7625 - val_accuracy: 0.7149
Epoch 17/25
15/15 [==============================] - 0s 18ms/step - loss: 0.5889 - accuracy: 0.7785 - val_loss: 0.7608 - val_accuracy: 0.7149
Epoch 18/25
15/15 [==============================] - 0s 19ms/step - loss: 0.5873 - accuracy: 0.7796 - val_loss: 0.7590 - val_accuracy: 0.7149
Epoch 19/25
15/15 [==============================] - 0s 18ms/step - loss: 0.5857 - accuracy: 0.7796 - val_loss: 0.7572 - val_accuracy: 0.7149
Epoch 20/25
15/15 [==============================] - 0s 19ms/step - loss: 0.5841 - accuracy: 0.7818 - val_loss: 0.7555 - val_accuracy: 0.7149
Epoch 21/25
15/15 [==============================] - 0s 20ms/step - loss: 0.5825 - accuracy: 0.7829 - val_loss: 0.7539 - val_accuracy: 0.7149
Epoch 22/25
15/15 [==============================] - 0s 25ms/step - loss: 0.5811 - accuracy: 0.7840 - val_loss: 0.7522 - val_accuracy: 0.7149
Epoch 23/25
15/15 [==============================] - 0s 30ms/step - loss: 0.5795 - accuracy: 0.7840 - val_loss: 0.7507 - val_accuracy: 0.7149
Epoch 24/25
15/15 [==============================] - 0s 31ms/step - loss: 0.5780 - accuracy: 0.7862 - val_loss: 0.7491 - val_accuracy: 0.7105
Epoch 25/25
15/15 [==============================] - 0s 27ms/step - loss: 0.5766 - accuracy: 0.7862 - val_loss: 0.7475 - val_accuracy: 0.7105
```

```
import tensorflow as tf
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten
(x_train,y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
plt.imshow(x_train[0],cmap='gray')
plt.show()
```



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

```
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).
```

```
path = "/content/drive/MyDrive/diabetes.csv"
```

```
diabetes = pd.read_csv(path, sep = ",")
```

```
diabetes.shape
```

```
(768, 9)
```

```
diabetes.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
diabetes.describe()
```

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

```
diabetes.Outcome.value_counts()
```

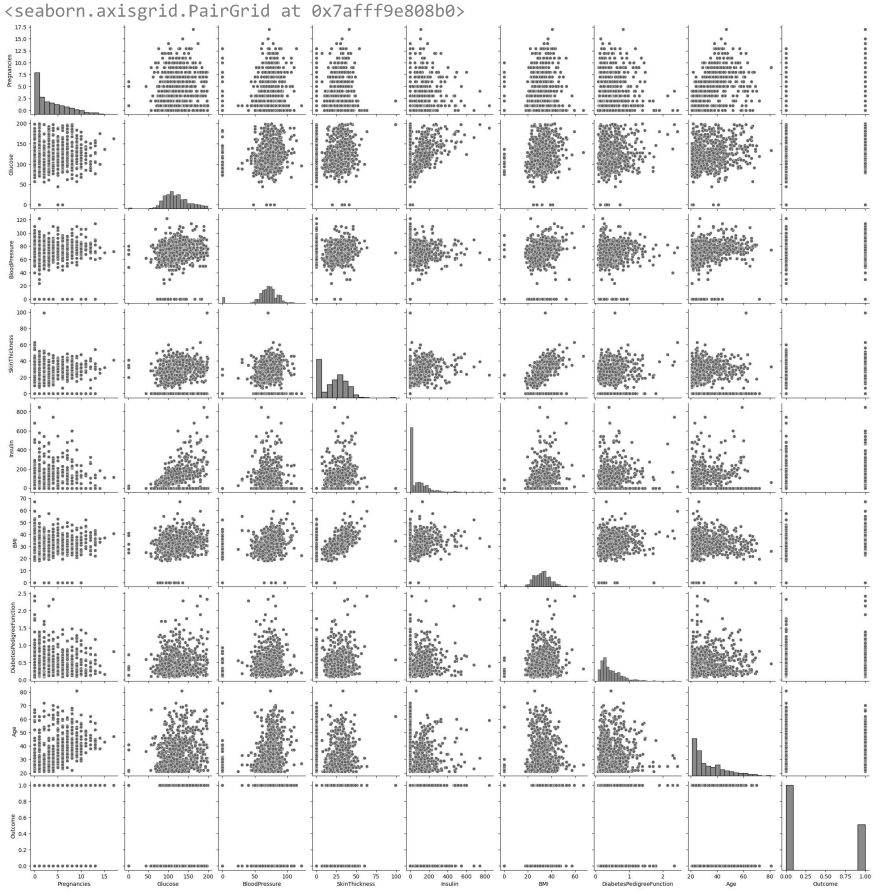
```
0    500
1    268
Name: Outcome, dtype: int64
```

```
diabetes.isna().sum()
```

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

```
sns.pairplot(diabetes)
```

```
<seaborn.axisgrid.PairGrid at 0x7afff9e808b0>
```




```
plt.show()
```

```
diabetes.corr()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insuli
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.07353
Glucose	0.129459	1.000000	0.152590	0.057328	0.33135
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.08893
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.43678
Insulin	-0.073535	0.331357	0.088933	0.436783	1.00000
BMI	0.017683	0.221071	0.281805	0.392573	0.19785
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.18507
Age	0.544341	0.263514	0.239528	-0.113970	-0.04216

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

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

```
y = diabetes['Outcome']
x = diabetes[feat]
x.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

```
ss = StandardScaler()
x_scaled = ss.fit_transform(x)
```

```
x_train, x_test, y_train, y_test = train_test_split(x_scaled, y, test_size = 0.2, random_state = 41)
```

```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((614, 8), (154, 8), (614,), (154,))
```

```
knm = KNeighborsClassifier(n_neighbors = 3, algorithm = 'ball_tree', p = 3)
```

```
knm.fit(x_train, y_train)
```

```
KNeighborsClassifier(algorithm='ball_tree', n_neighbors=3, p=3)
```

```
y_train_pred_knm = knm.predict(x_train)
y_test_pred_knm = knm.predict(x_test)
```

```
print("Train accuracy", accuracy_score(y_train, y_train_pred_knm))
print("Test accuracy", accuracy_score(y_test, y_test_pred_knm))
```

```
Train accuracy 0.8501628664495114
Test accuracy 0.7727272727272727
```

```
confusion_matrix(y_test, y_test_pred_knm)
```

```
array([[86, 13],
       [22, 33]])
```

```
nb = GaussianNB()
nb.fit(x_train, y_train)
y_train_pred_nb = nb.predict(x_train)
y_test_pred_nb = nb.predict(x_test)
print("Train accuracy", accuracy_score(y_train, y_train_pred_nb))
print("Test accuracy", accuracy_score(y_test, y_test_pred_nb))
```

```
Train accuracy 0.755700325732899
Test accuracy 0.7467532467532467
```

```
dt = DecisionTreeClassifier(max_depth = 5, class_weight = {0:0.5, 1:1})
dt.fit(x_train, y_train)
y_train_pred_dt = dt.predict(x_train)
y_test_pred_dt = dt.predict(x_test)
print("Train accuracy", accuracy_score(y_train, y_train_pred_dt))
print("Test accuracy", accuracy_score(y_test, y_test_pred_dt))
```

```
Train accuracy 0.8306188925081434
Test accuracy 0.7792207792207793
```

```
svm = SVC(kernel = 'rbf', C = 5)
svm.fit(x_train, y_train)
y_train_pred_svm = svm.predict(x_train)
y_test_pred_svm = svm.predict(x_test)
print("Train accuracy", accuracy_score(y_train, y_train_pred_svm))
print("Test accuracy", accuracy_score(y_test, y_test_pred_svm))
```

```
Train accuracy 0.8664495114006515
Test accuracy 0.7922077922077922
```

```
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


data = load_breast_cancer()
label_names = data["target_names"]
labels = data["target"]
feature_names = data["feature_names"]
features = data["data"]


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

```
['malignant' 'benign']
Class label : 0
```

```
print(feature_names)
print(features[0], "\n")
```

```
['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]
```

```
train, test, train_labels, test_labels = train_test_split(features, labels, test_size = 0.2, random_state = 42)


gnb = GaussianNB()
gnb.fit(train, train_labels)
```

```
GaussianNB()
```

```
preds = gnb.predict(test)
print(preds, "\n")
```

```
[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 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 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 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]
```

```
print(accuracy_score(test_labels, preds))
```

```
0.9736842105263158
```

```
weather=['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast',
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']
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
```

```
weather_encoded = le.fit_transform(weather)
print(weather_encoded)
```

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

```
temp_encoded = le.fit_transform(temp)
label = le.fit_transform(play)
print("Temp: ", temp_encoded)
print("play: ", label)
```

```
Temp:  [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
play:  [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
features = [tup for tup in zip(weather_encoded, temp_encoded)]
print (features)
```

```
[(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0), (2, 2), (2, 0), (1, 2), (2, 2), (0, 2), (0, 1), (1, 2)]
```

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(features,label)
predicted = model.predict([[0,2]])
print("predicted Value:", predicted)
```

```
predicted Value: [1]
```

```
from sklearn import datasets
wine = datasets.load_wine()
```

```
print("Features: ", wine.feature_names)
```

```
Features:  ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols',
```

```
print("\nLabels: ", wine.target_names)
```

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

```
wine.data.shape
```

```
(178, 13)
```

```
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]]
```

```
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 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 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 2 2 2 2 2 2 2]
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size = 0.3, random_state = 109)
```

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

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

```
from sklearn import metrics
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9074074074074074
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import seaborn as sns
```

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

```
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).
```

```
df = pd.read_csv("/Bank_Personal_Loan_Modelling.csv")
```

+ Code + Text

```
df
```

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	4996	29	3	40	92697	1	1.9	3	0	0	0	0	1	0
4996	4997	30	4	15	92037	4	0.4	1	85	0	0	0	1	0
4997	4998	63	39	24	93023	2	0.3	3	0	0	0	0	0	0
4998	4999	65	40	49	90034	3	0.5	2	0	0	0	0	1	0
4999	5000	28	4	83	92612	3	0.8	1	0	0	0	0	1	1

5000 rows × 14 columns

```
df.shape
```

```
(5000, 14)
```

```
df.columns
```

```
Index(['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg',
       'Education', 'Mortgage', 'Personal Loan', 'Securities Account',
       'CD Account', 'Online', 'CreditCard'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ID                  5000 non-null   int64  
 1   Age                 5000 non-null   int64  
 2   Experience          5000 non-null   int64  
 3   Income              5000 non-null   int64  
 4   ZIP Code            5000 non-null   int64  
 5   Family              5000 non-null   int64  
 6   CCAvg               5000 non-null   float64
 7   Education           5000 non-null   int64  
 8   Mortgage            5000 non-null   int64  
 9   Personal Loan       5000 non-null   int64  
 10  Securities Account  5000 non-null   int64  
 11  CD Account          5000 non-null   int64  
 12  Online              5000 non-null   int64  
 13  CreditCard          5000 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 547.0 KB
```

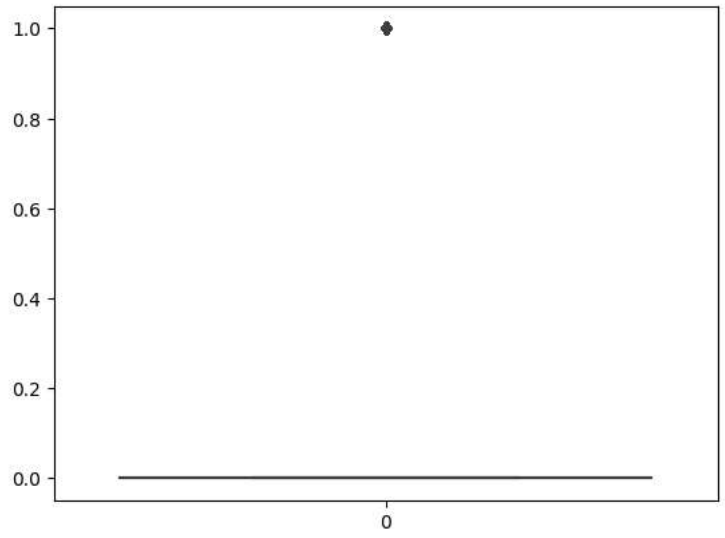
```
df.describe()
```

	ID	Age	Experience	Income	ZIP Code	Family	
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5(
mean	2500.500000	45.338400	20.104600	73.774200	93152.503000	2.396400	
std	1443.520003	11.463166	11.467954	46.033729	2121.852197	1.147663	
min	1.000000	23.000000	-3.000000	8.000000	9307.000000	1.000000	
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	

```
df.isnull().sum()
```

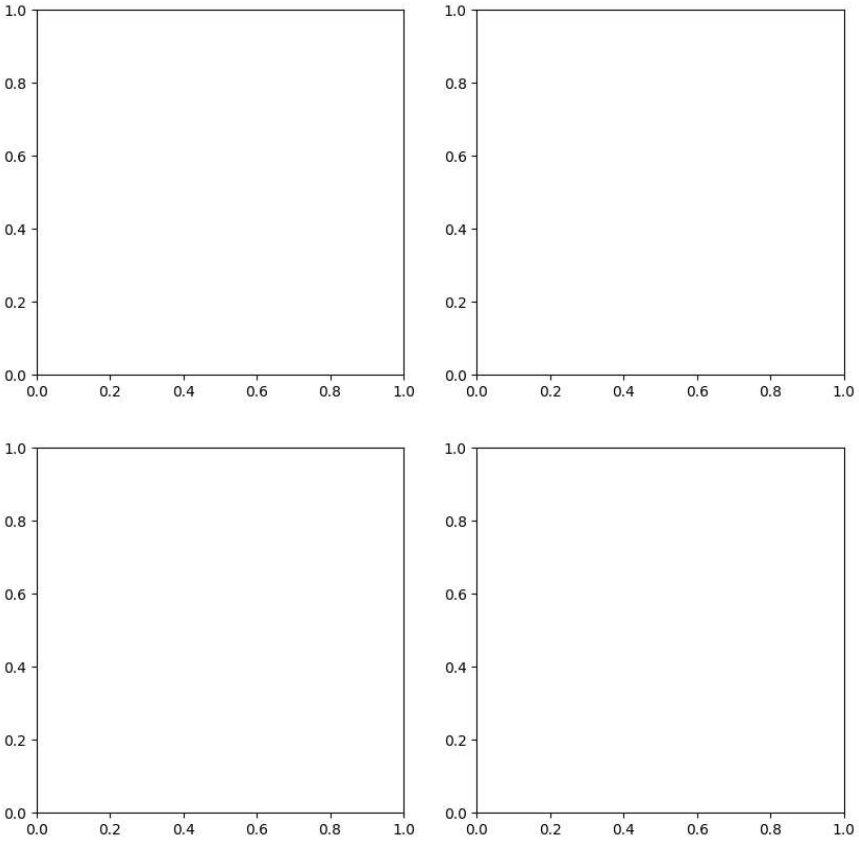
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

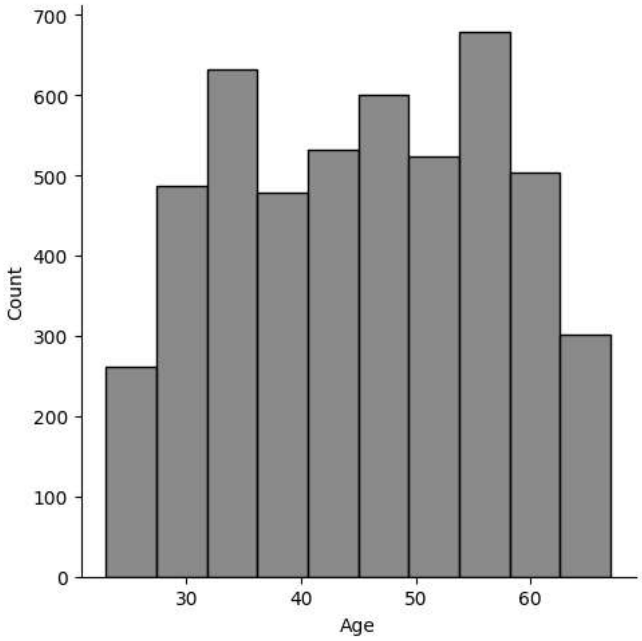
```
sns.boxplot(df['Personal Loan']);
plt.show()
```

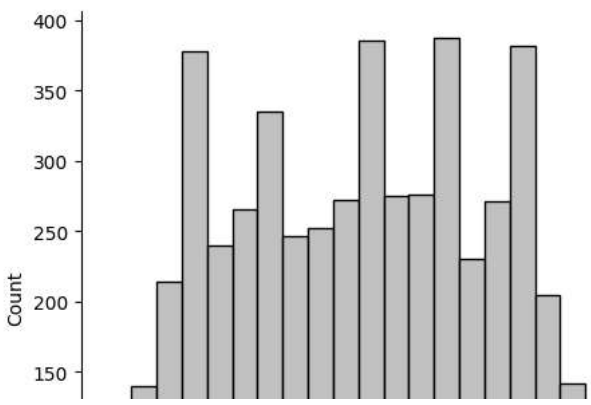


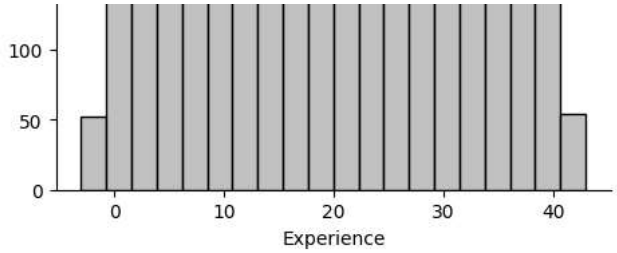

```
fig, axis = plt.subplots(2, 2, figsize=(10,10), sharex=False)
sns.displot(df['Age'],bins=10,ax=axis[0,0]);
sns.displot(df['Experience'],ax=axis[0,1],color='orange');
sns.displot(df['CCAvg'],ax=axis[1,0],color='gray');
sns.displot(df['Family'],ax=axis[1,1],color='Yellow');
plt.show()
```

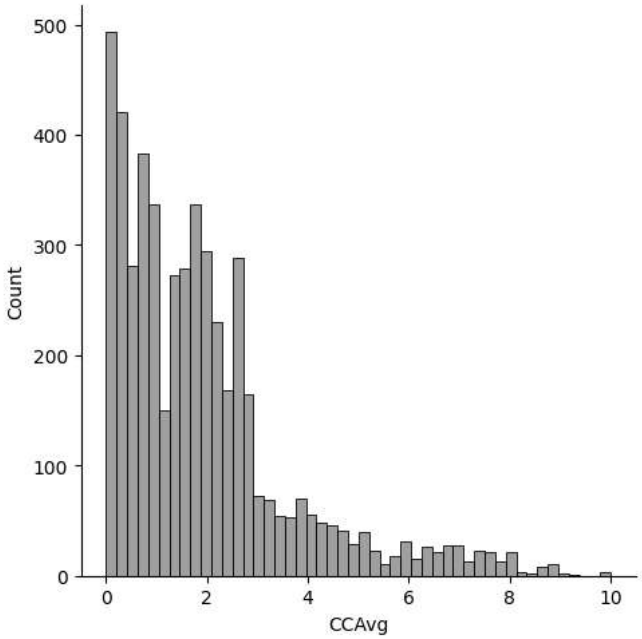
```
/usr/local/lib/python3.10/dist-packages/seaborn/distributions.py:2142: UserWarning: `
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/distributions.py:2142: UserWarning: `
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/distributions.py:2142: UserWarning: `
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/distributions.py:2142: UserWarning: `
  warnings.warn(msg, UserWarning)
```

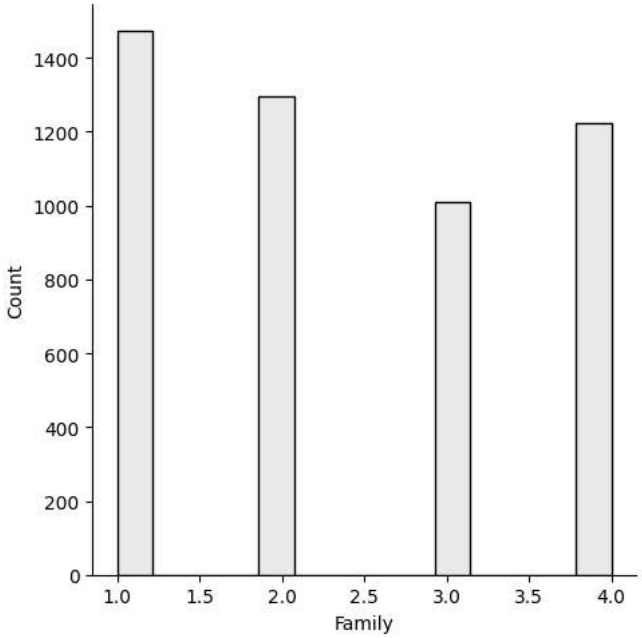






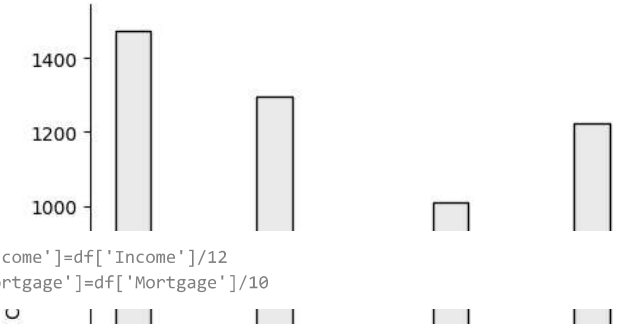







```
sns.displot(df['Family'],ax=axis[1,1],color='yellow');
```

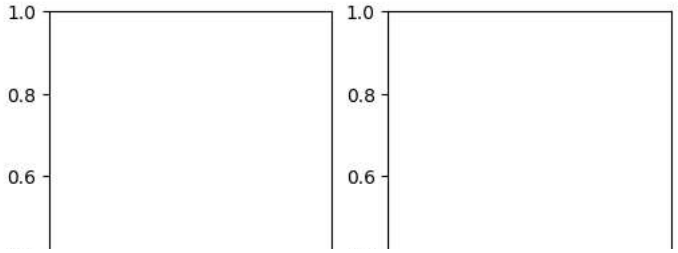
```
/usr/local/lib/python3.10/dist-packages/seaborn/distributions.py:2142: UserWarning: `
  warnings.warn(msg, UserWarning)
```



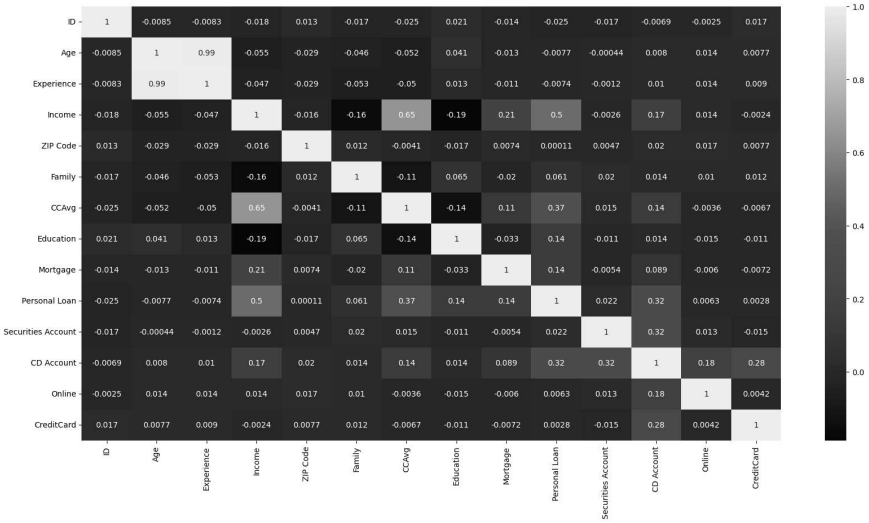

```
df['Income']=df['Income']/12
df['Mortgage']=df['Mortgage']/10
```

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

```
/usr/local/lib/python3.10/dist-packages/seaborn/distributions.py:2142: UserWarning: `
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/seaborn/distributions.py:2142: UserWarning: `
  warnings.warn(msg, UserWarning)
```




```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True);
plt.show()
```




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

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=100)
```

```
from sklearn.linear_model import LogisticRegression
logiR = LogisticRegression()
logiR.fit(x_train,y_train)
logiR_test = logiR.predict(x_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
print("Classification Report")
print(classification_report(y_test,logiR_test))
```

```
     Classification Report
                   precision    recall  f1-score   support

                0       0.93      0.98      0.95      1342
                1       0.63      0.35      0.45       158

         accuracy                           0.91      1500
        macro avg       0.78      0.66      0.70      1500
     weighted avg       0.90      0.91      0.90      1500
```

```
logiR_predict_train = logiR.predict_proba(x_train)[:,1] > 0.8
logiR_predict_test = logiR.predict_proba(x_test)[:,1] >0.8
print("Classification Report")
cm=classification_report(y_test,logiR_predict_test,labels=[1,0])
print(cm)
```

```
     Classification Report
                   precision    recall  f1-score   support

                1       0.74      0.09      0.16       158
                0       0.90      1.00      0.95      1342

         accuracy                           0.90      1500
        macro avg       0.82      0.54      0.55      1500
     weighted avg       0.89      0.90      0.86      1500
```

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train,y_train)
```

```
▾ GaussianNB
GaussianNB()
```

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

```
                   precision    recall  f1-score   support

                1       0.50      0.54      0.52       158
                0       0.95      0.94      0.94      1342

         accuracy                           0.90      1500
        macro avg       0.72      0.74      0.73      1500
     weighted avg       0.90      0.90      0.90      1500
```