Machine Learning

Rechu Vivek Reddy
21STUCHH010365

Department of Data Science and Artificial Intelligence
IcfaiTech (Deemed to be University)
HYDERABAD
APRIL,2023

Machine Learning

Report submitted for Machine Learning Course,

Dr. Kuncham Sreenivasa Rao

Rechu Vivek Reddy
21STUCHH010365

IcfaiTech
Faculty of Science & Technology (FST)

Department of Data Science and Artificial Intelligence
icfaiTech (Deemed to be University)
HYDERABAD

APRIL,2023

numpyclass-365

November 12, 2023

[47]:
```python
import numpy as np
```

[48]:
```python
a=np.array(['d',5,-3,9.5])
a.ndim
```

[48]: 1

U32 is unicode string < lowercase “>”upper case

[49]:
```python
a2=np.array([[2.5,3],[4,7.8],[0,1]])
a2.shape
a2.ndim
```

[49]: 2

[50]:
```python
a3=np.array(range(1,30,3))
a3.size
```

[50]: 10

[51]:
```python
a4=np.arange(1,11,2)
a4
```

[51]: array([1, 3, 5, 7, 9])

[52]:
```python
a5=np.array([range(i,i+3) for i in [2,4,6]])
a5.dtype
```

[52]: dtype('int64')

[53]:
```python
a6=np.zeros(20,dtype=np.double)
a6.itemsize
```

[53]: 8

[54]:
```python
a7=np.zeros((3,4),dtype=int)#default np.zeros create float values
a7
```

[54]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

[55]:
```
print(np.ones((4,5)))
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

[56]:
```
print(np.ones((3,5),dtype=float)) #default np.ones create float values
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

[57]:
```
#an array with linear sequence
print(np.arange(0,21,2)) #(starting poin, ending point, step)
```

```
[ 0  2  4  6  8 10 12 14 16 18 20]
```

[58]:
```
print(np.arange(0,1,0.1))
np.linspace(0,1,10)
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

[58]:
```
array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])
```

[59]:
```
#to create random values np.random is used
np.random.random((3,3))
```

[59]:
```
array([[0.32253379, 0.97604494, 0.10686509],
       [0.65679998, 0.74195017, 0.43019372],
       [0.5467563 , 0.34823922, 0.39091396]])
```

[60]:
```
#to create a bell curve
print(np.random.normal(0,1,(3,3)))
```

```
[[-0.16940372  0.55312268  0.98325844]
 [ 0.59065962 -0.94297668  0.43075304]
 [ 1.05690336  0.47607704 -1.79588036]]
```

[61]:
```
np.identity(3)
```

[61]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
```

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

[62]:
```
a6=np.array([[1,2,3],[5,6,7]])
a6
```

[62]:
```
array([[1, 2, 3],
       [5, 6, 7]])
```

[63]:
```
a6[:,2]
#print(a6[2,1])
```

[63]:
```
array([3, 7])
```

[64]:
```
a7=np.array([[[1,2,3],[1,2,3]],[[5,6,7],[5,6,7]]])
a7
```

[64]:
```
array([[[1, 2, 3],
        [1, 2, 3]],

       [[5, 6, 7],
        [5, 6, 7]]])
```

[65]:
```
a7.ndim
```

[65]:
```
3
```

[66]:
```
a8=np.arange(-2,24,4)
print(a8.ndim)
print(a8.size)
a8.shape
```

```
1
7
```

[66]:
```
(7,)
```

slicing is: x[atart:stop:step] >x[::2] is used to get the numbers a after the other * List item * List item > x[::-1] to get the reverse of the array

[67]:
```
a9=np.array([[-7,0,10,20],[-5,1,40,200],[-1,1,4,30]])
print(a9)
print(a9[1:3,0:2])
```

```
[[ -7   0  10  20]
 [ -5   1  40 200]
 [ -1   1   4  30]]
[[-5  1]
 [-1  1]]
```

[68]:
```
a10=np.array([[1,2],[2,3],[5,6]])
print(a10.ndim)
print(a10.itemsize)
print(a10.dtype)
print(a10.size)
print(a10.shape)
```

```
2
8
int64
6
(3, 2)
```

[69]:
```
b=np.array([[1,2],[2,3],[5,6]],dtype=np.complex)
b.itemsize
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1:
DeprecationWarning: `np.complex` is a deprecated alias for the builtin
`complex`. To silence this warning, use `complex` by itself. Doing this will not
modify any behavior and is safe. If you specifically wanted the numpy scalar
type, use `np.complex128` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  """Entry point for launching an IPython kernel.
```

[69]:
```
16
```

[70]:
```
c=np.zeros((3,2))
c
```

[70]:
```
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
```

[71]:
```
print(np.arange(1,10,3))
e=np.linspace(1,5,20)
e
```

```
[1 4 7]
```

[71]:
```
array([1.        , 1.21052632, 1.42105263, 1.63157895, 1.84210526,
       2.05263158, 2.26315789, 2.47368421, 2.68421053, 2.89473684,
       3.10526316, 3.31578947, 3.52631579, 3.73684211, 3.94736842,
       4.15789474, 4.36842105, 4.57894737, 4.78947368, 5.        ])
```

[113]:
```
print(a10.reshape((2,3)))
print(a10.sort)
```

```
[[1 2 2]
 [3 5 6]]
<built-in method sort of numpy.ndarray object at 0x7f3ed14786f0>
```

[73]:
```
a10.ravel()
```

[73]:
```
array([1, 2, 2, 3, 5, 6])
```

0.0.1 operations on arrays

[74]:
```
a10.min()
```

[74]:
```
1
```

[75]:
```
a10.max()
```

[75]:
```
6
```

[76]:
```
a10.sum()
```

[76]:
```
19
```

[77]:
```
a10.sum(axis=1)
```

[77]:
```
array([ 3,  5, 11])
```

[78]:
```
a10.mean()
```

[78]:
```
3.1666666666666665
```

[79]:
```
a10.std()
```

[79]:
```
1.7716909687891083
```

[80]:
```
a10.std(axis=1)
```

[80]:
```
array([0.5, 0.5, 0.5])
```

[81]:
```
a11=np.array([[0,1],[2,3],[4,5]])
a11
```

[81]:
```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

[107]:
```
a12=np.arange(0,6)
print(a12)
```

```
a12=np.reshape(a12,(3,2))
print(a12.sort(axis=0))
```

```
[0 1 2 3 4 5]
None
```

[93]:
```
print(a12[:2].min())
```

```
0
```

[109]:
```
print(a12.sort())
```

```
None
```

[99]:
```
a12[:2]
print(a12[:,1])
```

```
[1 3 5]
```

[96]:
```
print(a12[0:1].mean())
```

```
0.5
```

[101]:
```
a13=np.array([[3,6],[4,2]])
a14=np.array([[10,20],[30,40]])
print(a13+a14)
print(a14-a13)
```

```
[[13 26]
 [34 42]]
[[ 7 14]
 [26 38]]
```

[105]:
```
print(a13)
print(a14)
print(a13@a14)
print(a13&a14)
```

```
[[3 6]
 [4 2]]
[[10 20]
 [30 40]]
[[210 300]
 [100 160]]
[[2 4]
 [4 0]]
```

[114]:
```
#split(<array name>,start index, end index) is used to split the array
```

ml-pandas-365

November 12, 2023

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

[ ]:
```python
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
```

[ ]:
```python
ser_1.values
```

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

[ ]:
```python
ser_1.index
```

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

[ ]:
```python
ser_2 = Series([1,1,2,-3,-5],index=['a','b','c','d','e'])
ser_2
```

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

[ ]:
```python
ser_2['a']
```

[ ]:
```
1
```

[ ]:
```
ser_2[4]==ser_2['e']
```

[ ]:
```
True
```

[ ]:
```
ser_2[['c','a','b']]
```

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

[ ]:
```
ser_2
```

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

[ ]:
```
ser_2>0
```

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

[ ]:
```
ser_2[ser_2>0]
```

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

[ ]:
```
ser_2*2
```

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

[ ]:
```
np.exp(ser_2)
```

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

[ ]:
```
dict_1={'foo':100,'bar':200,'baz':300}
ser_3=Series(dict_1)
ser_3
```

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

[ ]:
```
index=['foo','bar','baz','qux']
ser_4=Series(dict_1,index=index)
ser_4
```

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

[ ]:
```
pd.isnull(ser_4)
```

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

[ ]:
```
pd.isnull(ser_4).sum()
```

[ ]:
```
1
```

[ ]:
```
print(ser_3)
print(ser_4)
```

```
foo    100
bar    200
baz    300
dtype: int64
foo    100.0
bar    200.0
baz    300.0
```

```
qux        NaN
dtype: float64
```

[ ]:
```
ser_3+ser_4
```

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

[ ]:
```
ser_4.name='qwerty'
ser_4.index.name = 'label'
ser_4
```

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

[ ]:
```
ser_4.index=['fo','br','bz','qx']
ser_4
```

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

1 *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 = pd.DataFrame(data_1)
print(df_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
```

[ ]:
```
print(data_1)
df_1
```

```
{'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
```

[ ]:
```
df_1.describe
```

[ ]:
```
<bound method NDFrame.describe of   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>
```

[ ]:
```
df_2=pd.DataFrame(data_1,columns=['year','State','pop'])
df_2
```

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

[ ]:
```
df_3=pd.DataFrame(data_1,columns=['year','State','pop','unemp1'])
df_3
```

[ ]:
```
   year State  pop unemp1
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
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x789144e3d780>
```

```
import numpy as np
from google.colab import autoviz
```

```
def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_3, *['year'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(df_3, *['pop'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x789144f43280>
```

```
import numpy as np
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,␣
 ↪figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_3, *['year'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,␣
 ↪figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
```

```
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(df_3, *['pop'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x78914ca49ed0>
```

```
import numpy as np
from google.colab import autoviz

def categorical_histogram(df, colname, figscale=1, mpl_palette_name='Dark2'):
  from matplotlib import pyplot as plt
  import seaborn as sns
  df.groupby(colname).size().plot(kind='barh', color=sns.palettes.mpl_palette(mpl_palette_name), figsize=(8*figscale, 4.8*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  return autoviz.MplChart.from_current_mpl_state()

chart = categorical_histogram(df_3, *['State'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7891424265c0>
```

```
import numpy as np
from google.colab import autoviz

def scatter_plots(df, colname_pairs, figscale=1, alpha=.8):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(len(colname_pairs) * 10 * figscale, 10 * figscale))
  for plot_i, (x_colname, y_colname) in enumerate(colname_pairs, start=1):
    ax = plt.subplot(1, len(colname_pairs), plot_i)
    df.plot(kind='scatter', x=x_colname, y=y_colname, s=(32 * figscale), alpha=alpha, ax=ax)
    ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = scatter_plots(df_3, *[[['year', 'pop']]], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7891425304c0>
```

```
import numpy as np
from google.colab import autoviz

def violin_plot(df, value_colname, facet_colname, figscale=1, mpl_palette_name='Dark2', **kwargs):
  from matplotlib import pyplot as plt
  import seaborn as sns
  figsize = (12 * figscale, 1.2 * figscale * len(df[facet_colname].unique()))
```

```
  plt.figure(figsize=figsize)
  sns.violinplot(df, x=value_colname, y=facet_colname, palette=mpl_palette_name,␣
↪**kwargs)
  sns.despine(top=True, right=True, bottom=True, left=True)
  return autoviz.MplChart.from_current_mpl_state()

chart = violin_plot(df_3, *['year', 'State'], **{'inner': 'stick'})
chart
```

```
import numpy as np
from google.colab import autoviz

def violin_plot(df, value_colname, facet_colname, figscale=1,␣
↪mpl_palette_name='Dark2', **kwargs):
  from matplotlib import pyplot as plt
  import seaborn as sns
  figsize = (12 * figscale, 1.2 * figscale * len(df[facet_colname].unique()))
  plt.figure(figsize=figsize)
  sns.violinplot(df, x=value_colname, y=facet_colname, palette=mpl_palette_name,␣
↪**kwargs)
  sns.despine(top=True, right=True, bottom=True, left=True)
  return autoviz.MplChart.from_current_mpl_state()

chart = violin_plot(df_3, *['pop', 'State'], **{'inner': 'stick'})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x78914cfbaf80>
```

```
import numpy as np
from google.colab import autoviz

def time_series_multiline(df, timelike_colname, value_colname, series_colname,␣
↪figscale=1, mpl_palette_name='Dark2'):
  from matplotlib import pyplot as plt
  import seaborn as sns
  figsize = (10 * figscale, 5.2 * figscale)
  palette = list(sns.palettes.mpl_palette(mpl_palette_name))
  def _plot_series(series, series_name, series_index=0):
    if value_colname == 'count()':
      counted = (series[timelike_colname]
                 .value_counts()
                 .reset_index(name='counts')
                 .rename({'index': timelike_colname}, axis=1)
                 .sort_values(timelike_colname, ascending=True))
      xs = counted[timelike_colname]
      ys = counted['counts']
    else:
      xs = series[timelike_colname]
      ys = series[value_colname]
```

```
    plt.plot(xs, ys, label=series_name, color=palette[series_index %␣
↪len(palette)])

  fig, ax = plt.subplots(figsize=figsize, layout='constrained')
  df = df.sort_values(timelike_colname, ascending=True)
  if series_colname:
    for i, (series_name, series) in enumerate(df.groupby(series_colname)):
      _plot_series(series, series_name, i)
    fig.legend(title=series_colname, bbox_to_anchor=(1, 1), loc='upper left')
  else:
    _plot_series(df, '')
  sns.despine(fig=fig, ax=ax)
  plt.xlabel(timelike_colname)
  plt.ylabel(value_colname)
  return autoviz.MplChart.from_current_mpl_state()

chart = time_series_multiline(df_3, *['year', 'pop', 'State'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def time_series_multiline(df, timelike_colname, value_colname, series_colname,␣
↪figscale=1, mpl_palette_name='Dark2'):
  from matplotlib import pyplot as plt
  import seaborn as sns
  figsize = (10 * figscale, 5.2 * figscale)
  palette = list(sns.palettes.mpl_palette(mpl_palette_name))
  def _plot_series(series, series_name, series_index=0):
    if value_colname == 'count()':
      counted = (series[timelike_colname]
                 .value_counts()
                 .reset_index(name='counts')
                 .rename({'index': timelike_colname}, axis=1)
                 .sort_values(timelike_colname, ascending=True))
      xs = counted[timelike_colname]
      ys = counted['counts']
    else:
      xs = series[timelike_colname]
      ys = series[value_colname]
    plt.plot(xs, ys, label=series_name, color=palette[series_index %␣
↪len(palette)])

  fig, ax = plt.subplots(figsize=figsize, layout='constrained')
  df = df.sort_values(timelike_colname, ascending=True)
  if series_colname:
    for i, (series_name, series) in enumerate(df.groupby(series_colname)):
      _plot_series(series, series_name, i)
```

```
      fig.legend(title=series_colname, bbox_to_anchor=(1, 1), loc='upper left')
    else:
      _plot_series(df, '')
    sns.despine(fig=fig, ax=ax)
    plt.xlabel(timelike_colname)
    plt.ylabel(value_colname)
    return autoviz.MplChart.from_current_mpl_state()

  chart = time_series_multiline(df_3, *['year', 'count()', 'State'], **{})
  chart
```

[ ]:
```
# df_3['State']
df_3.State
```

[ ]:
```
0    VA
1    VA
2    VA
3    MD
4    MD
Name: State, dtype: object
```

[ ]:
```
df_3.year
```

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

[ ]:
```
df_3.iloc[0]
```

[ ]:
```
dtype('O')
```

[ ]:
```
df_3.dtypes
```

[ ]:
```
year         int64
State       object
pop        float64
unemp1      object
dtype: object
```

[ ]:
```
df_3['unemp1']=np.arange(5)
df_3
```

[ ]:
```
   year State  pop  unemp1
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
```

[ ]:
```
unemp1=Series([6.0,6.0,6.1],index=[2,3,4])
df_3['unemp1']=unemp1
df_3
```

[ ]:
```
   year State  pop  unemp1
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
```

[ ]:
```
df_3['state_dup']=df_3['State']
df_3
```

[ ]:
```
   year State  pop  unemp1 state_dup
0  2012    VA  5.0     NaN        VA
1  2013    VA  5.1     NaN        VA
2  2014    VA  5.2     6.0        VA
3  2014    MD  4.0     6.0        MD
4  2015    MD  4.1     6.1        MD
```

[ ]:
```
del df_3['state_dup']
df_3
```

[ ]:
```
   year State  pop  unemp1
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}}
df_4=DataFrame(pop)
df_4
```

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

[ ]:
```
```

movie-data-analysis-365

November 12, 2023

[ ]:
```python
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).
```

[ ]:
```python
import numpy as np
import pandas as pd
movies = pd.read_csv("/content/drive/MyDrive/Machine learning/movie_analysis/
 ↪MOVIE DATA ANALYSIS/archive/movies.dat", delimiter='::')
print(movies.head())
users = pd.read_csv("/content/drive/MyDrive/Machine learning/movie_analysis/
 ↪MOVIE DATA ANALYSIS/archive/users.dat", delimiter='::')
print(users.head())
```

```
<ipython-input-25-1ac679ab3093>:3: ParserWarning:

Falling back to the 'python' engine because the 'c' engine does not support
regex separators (separators > 1 char and different from '\s+' are interpreted
as regex); you can avoid this warning by specifying engine='python'.


   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
<ipython-input-25-1ac679ab3093>:5: ParserWarning:

Falling back to the 'python' engine because the 'c' engine does not support
```

```
regex separators (separators > 1 char and different from '\s+' are interpreted
as regex); you can avoid this warning by specifying engine='python'.


   1  139564917
0  2   17528189
1  3  522540374
2  4  475571186
3  5  215022153
4  6  349681331
```

[ ]:
```python
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
```

[ ]:
```python
ratings = pd.read_csv("/content/drive/MyDrive/Machine learning/movie_analysis/
  ↪MOVIE DATA ANALYSIS/archive/ratings.dat", delimiter='::')
print(ratings.head())
```

```
<ipython-input-27-71e6f6c052bc>:1: ParserWarning:

Falling back to the 'python' engine because the 'c' engine does not support
regex separators (separators > 1 char and different from '\s+' are interpreted
as regex); you can avoid this warning by specifying engine='python'.


   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
```

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

```
   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
        User                 ID
0          2           17528189
1          3          522540374
2          4          475571186
3          5          215022153
4          6          349681331
…        …                  …
70777  70779          441446292
70778  70780           36878476
70779  70781          330301436
70780  70782  1244805465323835397
70781  70783          491884729

[70782 rows x 2 columns]
```

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

[ ]:
```python
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()
```

[ ]:
```python
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
```

[ ]:
```
data2 = data.query("Ratings == 10")
data2
```

[ ]:
```
              ID                              Title  \
0             10  La sortie des usines Lumière (1895)
1             12        The Arrival of a Train (1896)
15           417            A Trip to the Moon (1902)
18           417            A Trip to the Moon (1902)
20           417            A Trip to the Moon (1902)
…            …                                  …
908617  14544192            Bo Burnham: Inside (2021)
908618  14544192            Bo Burnham: Inside (2021)
908626  14544192            Bo Burnham: Inside (2021)
908627  14544192            Bo Burnham: Inside (2021)
908628  14544192            Bo Burnham: Inside (2021)

                                             Genre   User  Ratings  \
0                                Documentary|Short  70577       10
1                                Documentary|Short  69535       10
15      Short|Action|Adventure|Comedy|Fantasy|Sci-Fi  27589       10
18      Short|Action|Adventure|Comedy|Fantasy|Sci-Fi  37621       10
20      Short|Action|Adventure|Comedy|Fantasy|Sci-Fi  39522       10
…                                              …     …        …
908617                          Comedy|Drama|Music   3040       10
908618                          Comedy|Drama|Music  11908       10
908626                          Comedy|Drama|Music  54886       10
908627                          Comedy|Drama|Music  55241       10
908628                          Comedy|Drama|Music  57060       10

         Timestamp
0       1412878553
1       1439248579
15      1538187753
18      1529844360
20      1437579236
…            …
908617  1622966424
908618  1623004815
908626  1622766966
```

```
908627  1622416491
908628  1623092790

[107284 rows x 6 columns]
```

ml-dt-iris-365

November 12, 2023

[1]:
```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

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

[ ]:
```python
df=pd.read_csv('/content/drive/MyDrive/Machine learning/DT-IRIS/DT-IRIS/iris.
↪csv')
```

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

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

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

[ ]:
```python
df.info()
```

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

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

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

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

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

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

[ ]:
```python
sns.heatmap(df.corr(),annot=True,cmap='viridis')
```

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

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

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

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

[ ]:
```
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
```

[ ]:
```
X_test.shape
y_test.shape
```

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

[ ]:
```
DTC=DecisionTreeClassifier()
```

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

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

[ ]:
```
prediction
```

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

[ ]:
```
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))
```

[ ]:
```
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)
```

tennisnb-365

November 12, 2023

[ ]:
```
weather=['sunny','sunny','overcast','rainy','rainy','rainy','overcast','sunny','sunny','rainy'
temp=['hot','hot','hot','mild','cool','cool','cool','mild','cool','mild','mild','mild','hot','
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)
```

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

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

```
Predicted Value: [1]
```

[ ]:
```python
from sklearn import datasets

wine = datasets.load_wine()
```

print("Features:",wine.feature_names)

[ ]:
```python
print("\nlabels: ", wine.target_names)
```

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

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

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

[ ]:
```python
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]]
```

[ ]:
```python
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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 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]
```

[ ]:
```python
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)
```

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

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

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

[ ]:
```
y_pred=gnb.predict(X_test)
```

[ ]:
```
print("Y predicted values: ", y_pred)
```

```
Y predicted values:  [0 0 1 2 0 1 0 0 1 0 2 2 2 2 0 1 1 0 0 1 2 1 0 2 0 0 1 2 0
1 2 1 1 0 1 1 0
 2 2 0 2 1 0 0 0 2 2 0 1 1 2 0 0 2]
```

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

```
Accuracy:  0.9074074074074074
```

ml-dt-play-tennis-365

November 12, 2023

[ ]:
```python
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).
```

[ ]:
```python
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from sklearn import metrics #Import scikit-learn metrics module for accuracy␣
↪calculation
```

[ ]:
```python
df=pd.read_csv("/content/drive/MyDrive/Machine learning/Play Tennis/Play Tennis/
↪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

[ ]:
```python
df.describe()      #To see statistical details of the dataset:
```

[ ]:

	Day	Outlook	Temprature	Humidity	Wind	Play_Tennis
count	14.0000	14.000000	14.000000	14.000000	14.000000	14.000000
mean	6.5000	1.071429	1.142857	0.500000	0.571429	0.642857

```
std      4.1833   0.828742   0.864438   0.518875   0.513553     0.497245
min      0.0000   0.000000   0.000000   0.000000   0.000000     0.000000
25%      3.2500   0.250000   0.250000   0.000000   0.000000     0.000000
50%      6.5000   1.000000   1.000000   0.500000   1.000000     1.000000
75%      9.7500   2.000000   2.000000   1.000000   1.000000     1.000000
max     13.0000   2.000000   2.000000   1.000000   1.000000     1.000000
```

[ ]:
```
len(df)            #Dataset Length
```

[ ]: 14

[ ]:
```
print(df.shape)  #To see the number of rows and columns in our dataset:
```

```
(14, 6)
```

[ ]:
```
df.head()          #To inspect the first five records of the dataset:
```

[ ]:

	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

[ ]:
```
df.tail()          #To inspect the last five records of the dataset:
```

[ ]:

	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

[ ]:
```
from sklearn import preprocessing
string_to_int= preprocessing.LabelEncoder()#encode your data
df=df.apply(string_to_int.fit_transform) #fit and transform it
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 ]                                        #contains the attribute
y = df.Play_Tennis                                           #contains the label
```

[ ]:
```
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                                  #␣
 ↪import the classifier
classifier =DecisionTreeClassifier(criterion="entropy", random_state=100)     #␣
 ↪create a classifier object
classifier.fit(X_train, y_train)                                              #␣
 ↪fit the classifier with X and Y data or
```

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

[ ]:
```
#Predict the response for test dataset
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
2	VA	2014	5.2
3	MD	2014	4.0
4	MD	2015	4.1

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

[ ]:
```
  Outlook Temprature Humidity Wind
0       2           1        0    1
```

[ ]:
```
y_pred2= classifier.predict(df_2)
y_pred2
```

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

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

```
Accuracy: 0.6
```

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

[ ]:

	Actual	Predicted
12	1	1
4	1	0
2	1	1
1	0	0
13	0	1

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

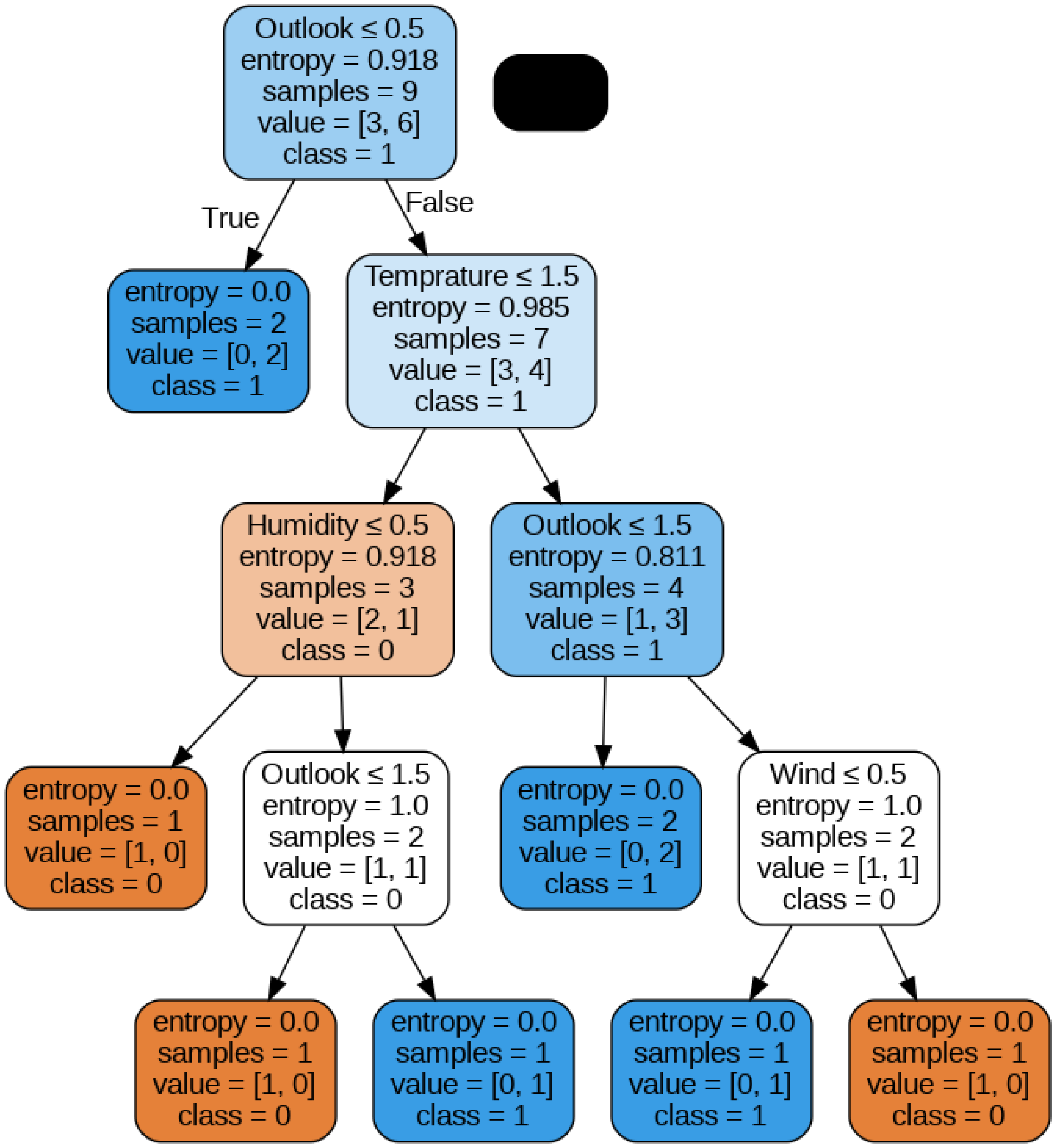
```
[[1 1]
 [1 2]]
              precision    recall  f1-score   support

           0       0.50      0.50      0.50         2
           1       0.67      0.67      0.67         3

    accuracy                           0.60         5
   macro avg       0.58      0.58      0.58         5
weighted avg       0.60      0.60      0.60         5
```

[ ]:
```
# https://pypi.python.org/pypi/pydot
!apt-get -qq install -y graphviz && pip install pydot
import pydot
```

```
Requirement already satisfied: pydot in /usr/local/lib/python3.10/dist-packages
(1.4.2)
Requirement already satisfied: pyparsing>=2.1.4 in
/usr/local/lib/python3.10/dist-packages (from pydot) (3.1.1)
```

[ ]:
```
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
import io

# Assuming you have defined and trained 'classifier' already
# and 'value' contains your list of feature names

dot_data = io.StringIO()  # Using io.StringIO instead of StringIO
export_graphviz(classifier, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,
                feature_names=value,  # Replace 'value' with your list of␣
 ↪feature names
                class_names=['0', '1'])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Play_Tennis.png')  # Changed the filename to remove spaces
Image(graph.create_png())
```

[ ]:


Outlook ≤ 0.5
entropy = 0.918
samples = 9
value = [3, 6]
class = 1
[redacted]
True
False
entropy = 0.0
samples = 2
value = [0, 2]
class = 1
Temprature ≤ 1.5
entropy = 0.985
samples = 7
value = [3, 4]
class = 1
Humidity ≤ 0.5
entropy = 0.918
samples = 3
value = [2, 1]
class = 0
Outlook ≤ 1.5
entropy = 0.811
samples = 4
value = [1, 3]
class = 1
entropy = 0.0
samples = 1
value = [1, 0]
class = 0
Outlook ≤ 1.5
entropy = 1.0
samples = 2
value = [1, 1]
class = 0
entropy = 0.0
samples = 2
value = [0, 2]
class = 1
Wind ≤ 0.5
entropy = 1.0
samples = 2
value = [1, 1]
class = 0
entropy = 0.0
samples = 1
value = [1, 0]
class = 0
entropy = 0.0
samples = 1
value = [0, 1]
class = 1
entropy = 0.0
samples = 1
value = [0, 1]
class = 1
entropy = 0.0
samples = 1
value = [1, 0]
class = 0

ml-movierating-dt-365

November 12, 2023

[ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as ssn
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
```

[ ]:
```python
movies=pd.read_csv('http://bit.ly/imdbratings')
```

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

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

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

[ ]:
```python
movies.columns
```

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

[ ]:
```python
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-27-6cbec47f27d9>:1: DeprecationWarning: `np.object` is a
deprecated alias for the builtin `object`. To silence this warning, use `object`
by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  categorical_features=[i for i  in movies.select_dtypes(include=np.object)]
```

[ ]:
```
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  \
0       0          0          0          0       0      1      0       0
```

```
1        0          0          0          0        0       1       0       0
2        0          0          0          0        0       1       0       0
3        1          0          0          0        0       0       0       0
4        0          0          0          0        0       1       0       0

   Fantasy  Film-Noir  History  Horror  Mystery  Sci-Fi  Thriller  Western  \
0        0          0        0       0        0       0         0        0
1        0          0        0       0        0       0         0        0
2        0          0        0       0        0       0         0        0
3        0          0        0       0        0       0         0        0
4        0          0        0       0        0       0         0        0

   duration
0       142
1       175
2       200
3       152
4       154
```

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

[ ]:
```python
train_df.shape
```

[ ]: (979, 18)

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

[ ]:
```python
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.
↪2,random_state=42)
```

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

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

[ ]: LinearRegression()

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

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

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

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

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

[ ]: SVR()

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

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

```
RMSE using SVR is 0.001221107353436723
```

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

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

[ ]: 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 0.19074159580498865
```

imdb-ratings

November 12, 2023

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

    content_rating      genre  duration  \
0                R      Crime       142
1                R      Crime       175
2                R      Crime       200
3            PG-13     Action       152
4                R      Crime       154
..             …          …        …
974             PG     Comedy       116
975             PG  Adventure       118
976          PG-13     Action       138
```

```
977             PG     Horror       114
978              R      Crime       126

                                            actors_list
0    [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…
1      [u'Marlon Brando', u'Al Pacino', u'James Caan']
2    [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
3    [u'Christian Bale', u'Heath Ledger', u'Aaron E…
4    [u'John Travolta', u'Uma Thurman', u'Samuel L…
..                                                …
974  [u'Dustin Hoffman', u'Jessica Lange', u'Teri G…
975  [u'Michael J. Fox', u'Christopher Lloyd', u'Ma…
976  [u'Russell Crowe', u'Paul Bettany', u'Billy Bo…
977  [u'JoBeth Williams', u"Heather O'Rourke", u'Cr…
978  [u'Charlie Sheen', u'Michael Douglas', u'Tamar…

[979 rows x 6 columns]
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7839bc95e4a0>
```

```
import numpy as np
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(movies, *['star_rating'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def value_plot(df, y, figscale=1):
  from matplotlib import pyplot as plt
  df[y].plot(kind='line', figsize=(8 * figscale, 4 * figscale), title=y)
  plt.gca().spines[['top', 'right']].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = value_plot(movies, *['duration'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7839bc7a7d90>
```

```
import numpy as np
```

```
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,␣
↪figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(movies, *['star_rating'], **{})
chart
```

```
import numpy as np
from google.colab import autoviz

def histogram(df, colname, num_bins=20, figscale=1):
  from matplotlib import pyplot as plt
  df[colname].plot(kind='hist', bins=num_bins, title=colname,␣
↪figsize=(8*figscale, 4*figscale))
  plt.gca().spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = histogram(movies, *['duration'], **{})
chart
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7839ba70d1e0>
```

```
import numpy as np
from google.colab import autoviz

def scatter_plots(df, colname_pairs, figscale=1, alpha=.8):
  from matplotlib import pyplot as plt
  plt.figure(figsize=(len(colname_pairs) * 6 * figscale, 6 * figscale))
  for plot_i, (x_colname, y_colname) in enumerate(colname_pairs, start=1):
    ax = plt.subplot(1, len(colname_pairs), plot_i)
    df.plot(kind='scatter', x=x_colname, y=y_colname, s=(32 * figscale),␣
↪alpha=alpha, ax=ax)
    ax.spines[['top', 'right',]].set_visible(False)
  plt.tight_layout()
  return autoviz.MplChart.from_current_mpl_state()

chart = scatter_plots(movies, *[[['star_rating', 'duration']]], **{})
chart
```

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

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

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

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

[ ]:
```
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  \
0       0          0          0          0       0      1      0       0
1       0          0          0          0       0      1      0       0
2       0          0          0          0       0      1      0       0
3       1          0          0          0       0      0      0       0
4       0          0          0          0       0      1      0       0

   Fantasy  Film-Noir  History  Horror  Mystery  Sci-Fi  Thriller  Western  \
0        0          0        0       0        0       0         0        0
1        0          0        0       0        0       0         0        0
2        0          0        0       0        0       0         0        0
3        0          0        0       0        0       0         0        0
4        0          0        0       0        0       0         0        0

   duration
0       142
1       175
2       200
3       152
4       154
```

[ ]:
```
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()
```

[ ]:
```
y_pred=LR.predict(x_test)
```

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

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

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

[ ]:
```
sv.fit(x_train,y_train)
```

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

[ ]:
```
sv_pred=sv.predict(x_test)
```

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

```
RMSE using SVR is 0.09749560506058148
```

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

[ ]:
```
clf.fit(x_train,y_train)
```

[ ]:
```
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
```

perceptron-scratch-365

November 12, 2023

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

[2]:
```
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))
```

[3]:
```
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)
```

[4]:
```
p_model_and.w
```

[4]:
```
array([-0.08297623, -0.17307808,  0.08361381])
```

[5]:
```
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:       1
Data:   [0 1],  Target: [0],  predicted:       0
Data:   [1 0],  Target: [0],  predicted:       1
Data:   [1 1],  Target: [1],  predicted:       0
```

[6]:
```
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=        100)
```

[7]:
```
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
```

[8]:
```
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:       0
Data:   [1 0],  Target: [1],  predicted:       0
Data:   [1 1],  Target: [1],  predicted:       0
```

[9]:
```
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:       0
Data:   [0 1],  Target: [1],  predicted:       1
```

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

[10]:
```
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
```

[11]:
```
x         = np.arange(36).reshape(-1,         9)
x
```

[11]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8],
       [ 9, 10, 11, 12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23, 24, 25, 26],
       [27, 28, 29, 30, 31, 32, 33, 34, 35]])
```

[12]:
```
x[0]
```

[12]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

[13]:
```
x[0].shape
```

[13]:
```
(9,)
```

[14]:
```
x.shape
```

[14]:
```
(4, 9)
```

[15]:
```
x.shape[0]
```

[15]:
```
4
```

[16]:
```
name=["Manjeet",  "Nikhil",  "Shambhavi","Asthan"]
roll_no=[        4,         1,         3,         2]
mapped=zip(name,roll_no)
print(set(mapped))
```

```
{('Nikhil', 1), ('Shambhavi', 3), ('Manjeet', 4), ('Asthan', 2)}
```

[17]:
```
in_num        =        10
print        ("Input number in_num",in_num)
out_arr =  np.atleast_2d(in_num)
print        ("output 2d array from input number:",out_arr)
```

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

gender-classification-oct5-365

November 12, 2023

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

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

[3]:
```
results = ['male', 'male',
'female', 'female',
'male', 'male',
'female', 'female','female',
'male', 'male']
```

[4]:
```
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,␣
 ↪shuffle=True, verbose=0, warm_start=False)
```

[5]:
```
model.fit(data,results)
```

[5]:
```
Perceptron(n_jobs=1)
```

[6]:
```
predicted_results = model.predict(data)
acc_per = accuracy_score(results, predicted_results) * 100
print('Accuracy for perceptron: {} %'.format(acc_per))
```

```
Accuracy for perceptron: 54.54545454545454 %
```

[7]:
```
prediction = model.predict([[1.62, 0.49, 0.38]])
print(prediction)
```

```
['male']
```

[8]:
```python
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
```

[9]:
```python
methods = ['Decision Trees', 'SVM', 'Perceptron', 'K nearest neighbour']
```

[10]:
```python
X = [[181, 88, 44], [177, 70, 43], [160, 60, 38], [154, 54, 37], [166, 65, 40],␣
↪[190, 90, 47], [175, 64, 39], [177, 78, 40], [159, 55, 37], [171, 75, 42],␣
↪[181, 85, 43]]
Y = ['male', 'male', 'female', 'female', 'male', 'male', 'female', 'female',␣
↪'female', 'male', 'male']
```

[11]:
```python
clf_tree = DecisionTreeClassifier()
clf_svm = SVC()
clf_percept = Perceptron()
clf_KNN = NearestNeighbors()
```

[12]:
```python
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)
```

[13]:
```python
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:
```

[13]:
```
(None, 100.0, '%')
```

[14]:
```python
clf_svm_prediction = clf_svm.predict(X)
acc_svm = accuracy_score (Y, clf_svm_prediction)*100
print ("Labels for training set using SVM:'"),acc_svm, "%"
```

```
Labels for training set using SVM:'
```

[14]:
```
(None, 54.54545454545454, '%')
```

[15]:
```python
clf_percept_prediction = clf_percept.predict(X)
acc_per = accuracy_score (Y, clf_percept_prediction)*100
print ("Labels for training set using Perceptron:"), acc_per, "%"
```

```
Labels for training set using Perceptron:
```

[15]: (None, 45.45454545454545, '%')

[16]:
```
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:
```

[16]: (None, 100.0, '%')

[17]:
```
acc_all = [acc_tree,acc_svm,acc_per,acc_knn]
score_bestmethod = np.max(acc_all)
best_method = np.argmax(acc_all)
```

[18]:
```
print (methods[best_method], "is the best method with accuracy of"),␣
↪score_bestmethod, "%"
```

```
Decision Trees is the best method with accuracy of
```

[18]: (None, 100.0, '%')

xor-nn-365

November 12, 2023

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

[ ]:
```python
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 = []
```

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

[ ]:
```python
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
```

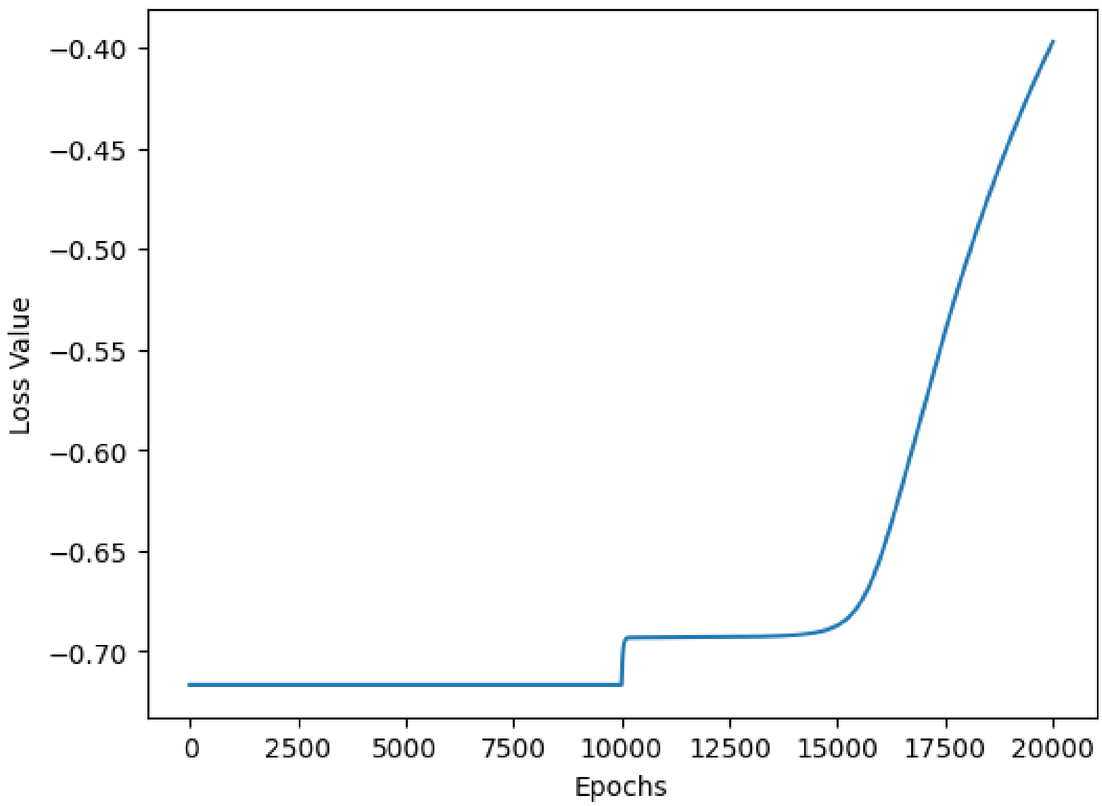
[ ]:
```python
def back_prop (m, w1,w2, z1,a1, z2, a2, y):
  dz2 = a2-y
  dw2 = np.dot (dz2, a1.T)/m
  dz1 = np.dot (w2.T, dz2) * a1*(1-a1)
  dw1 = np.dot (dz1, x . T)/m
  dw1 = np.reshape (dw1,w1.shape)
  dw2 = np.reshape (dw2,w2.shape)
  return dz2, dw2, dz1, dw1
```

[ ]:
```python
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", )
```

multilayer-on-mnist-365

November 12, 2023

[17]:
```
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
```

[18]:
```
(x_train, y_train),(x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

[19]:
```
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,)
```

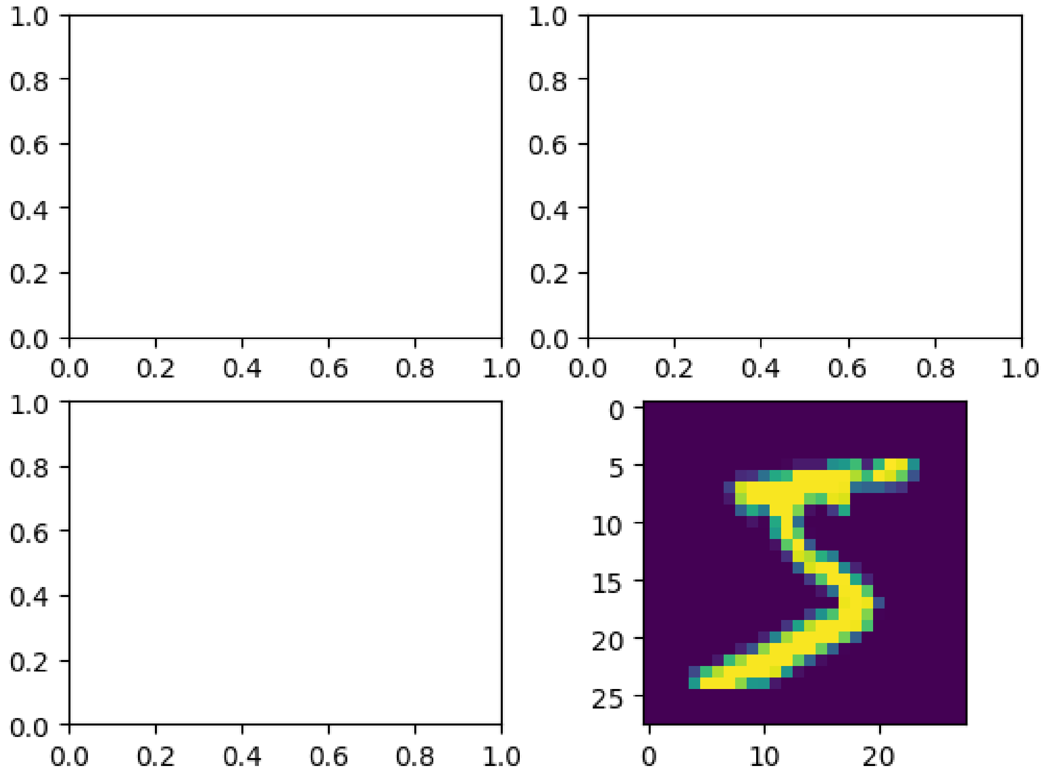
[20]:
```
print(x_train[0])
```

```
[[  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
```

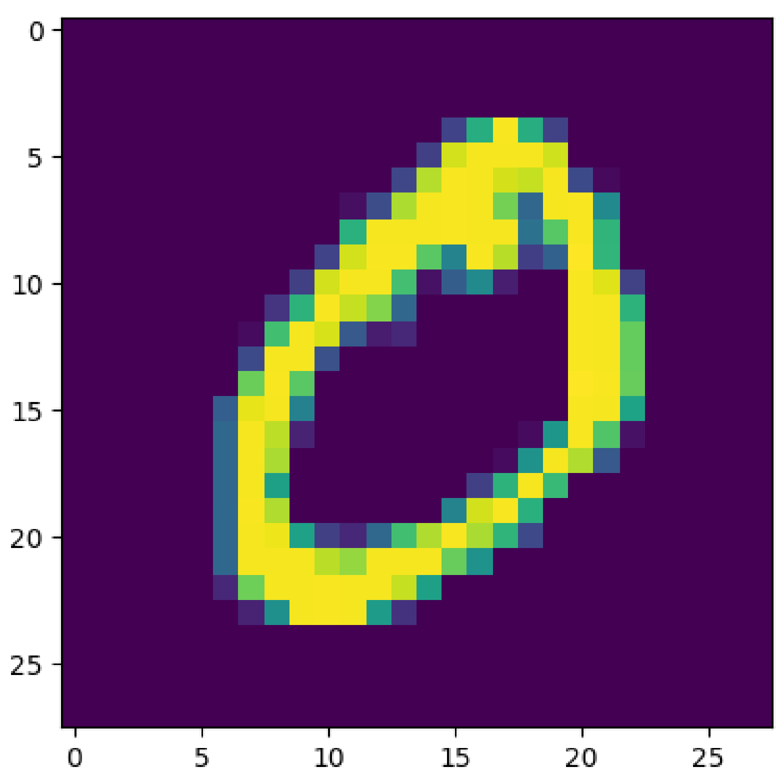
```
   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   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]]
```

[21]:

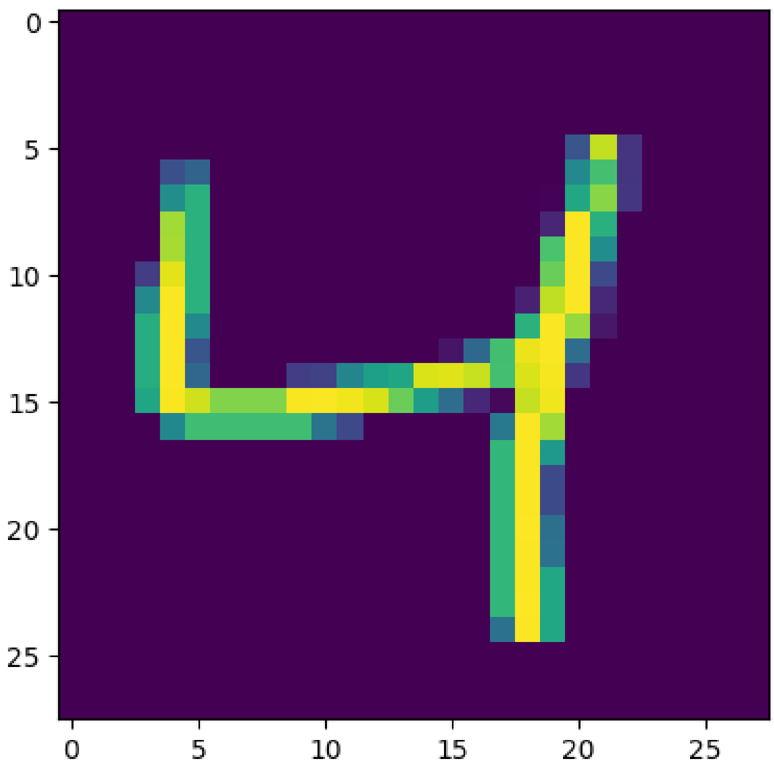
```
ax = plt.subplots(2, 2)
k = 0
for i in range(2):
  for j in range(2):
    plt.imshow(x_train[k])
    k += 1
    plt.show()
```

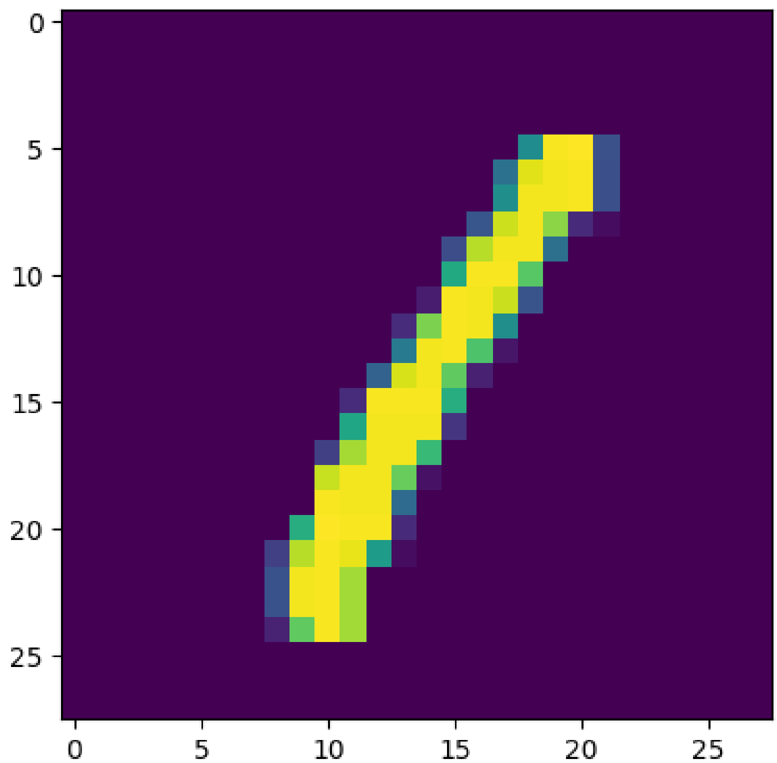


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
25



0
5
10
15
20
25


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




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

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

```
[23]: x_train = x_train / 255
      x_test = x_test / 255
```

```
[24]: x_train[0]
```

```
[24]: array([[0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        , 0.        , 0.        ,
         0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        , 0.        ,
```

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

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

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

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

[25]:
```
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

[26]:
```
y_train.shape
```

[26]:
```
(60000, 10)
```

[27]:
```
model = Sequential()
model.add(Flatten(input_shape = (28, 28)))
model.add(Dense(256, activation = 'relu'))
model.add(Dense(128, activation = 'relu'))
model.add(Dense(64, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))
model.summary()
```

```
Model: "sequential_1"

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

 dense_4 (Dense)             (None, 256)               200960

 dense_5 (Dense)             (None, 128)               32896

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 10)                650

=================================================================
Total params: 242762 (948.29 KB)
Trainable params: 242762 (948.29 KB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[28]:
```
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)
```

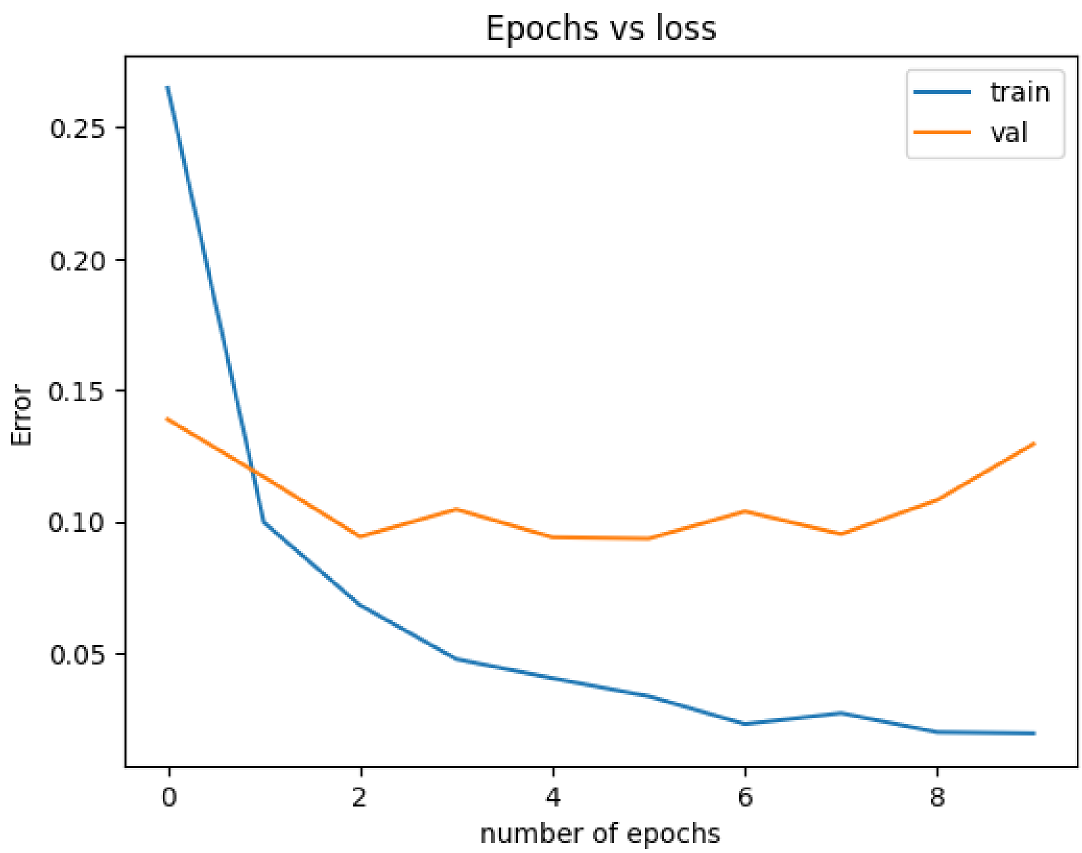
```
Epoch 1/10
750/750 [==============================] - 6s 8ms/step - loss: 0.2650 -
accuracy: 0.9216 - val_loss: 0.1389 - val_accuracy: 0.9582
Epoch 2/10
750/750 [==============================] - 6s 8ms/step - loss: 0.0996 -
accuracy: 0.9700 - val_loss: 0.1169 - val_accuracy: 0.9647
Epoch 3/10
750/750 [==============================] - 5s 6ms/step - loss: 0.0681 -
accuracy: 0.9786 - val_loss: 0.0942 - val_accuracy: 0.9721
Epoch 4/10
750/750 [==============================] - 6s 8ms/step - loss: 0.0476 -
accuracy: 0.9854 - val_loss: 0.1046 - val_accuracy: 0.9696
Epoch 5/10
750/750 [==============================] - 5s 6ms/step - loss: 0.0404 -
accuracy: 0.9874 - val_loss: 0.0940 - val_accuracy: 0.9744
Epoch 6/10
750/750 [==============================] - 5s 6ms/step - loss: 0.0335 -
accuracy: 0.9893 - val_loss: 0.0935 - val_accuracy: 0.9748
Epoch 7/10
750/750 [==============================] - 6s 8ms/step - loss: 0.0229 -
accuracy: 0.9920 - val_loss: 0.1039 - val_accuracy: 0.9737
Epoch 8/10
750/750 [==============================] - 4s 6ms/step - loss: 0.0271 -
accuracy: 0.9909 - val_loss: 0.0952 - val_accuracy: 0.9745
Epoch 9/10
750/750 [==============================] - 6s 7ms/step - loss: 0.0198 -
accuracy: 0.9936 - val_loss: 0.1082 - val_accuracy: 0.9770
Epoch 10/10
750/750 [==============================] - 4s 6ms/step - loss: 0.0194 -
accuracy: 0.9935 - val_loss: 0.1295 - val_accuracy: 0.9692
```

[29]:
```
train_history.history.keys()
```

[29]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[30]:
```
plt.plot(train_history.history['loss'])
plt.plot(train_history.history['val_loss'])
plt.title("Epochs vs loss")
plt.xlabel("number of epochs")
```

```python
plt.ylabel("Error")
plt.legend(['train', 'val'])
plt.show()
```



[31]:
```python
score = model.evaluate(x_test, y_test, batch_size = 64)
```

```
157/157 [==============================] - 0s 2ms/step - loss: 0.1095 -
accuracy: 0.9741
```

[32]:
```python
print("testing accuracy: ", score[1])
```

```
testing accuracy:  0.9740999937057495
```

multilayer-lfw-365

November 12, 2023

[28]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_lfw_people

lfw=fetch_lfw_people(min_faces_per_person=100)

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

```
Number of sample faces and its height and width: 1140 62 47
```

[29]:
```
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)
```

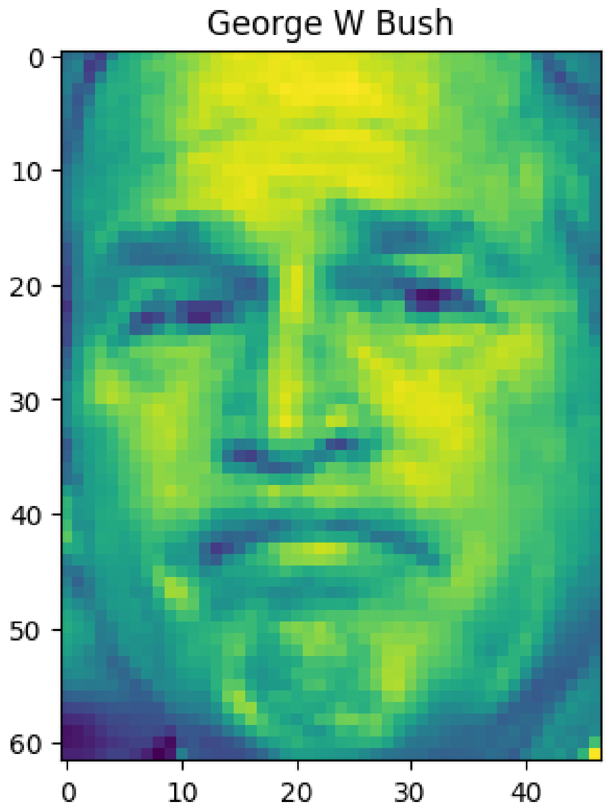
```
input data shape: (1140, 2914)
target length: 1140
target names: ['Colin Powell' 'Donald Rumsfeld' 'George W Bush' 'Gerhard
Schroeder'
 'Tony Blair']
```

[30]:
```
X[0]
```

[30]:
```
array([0.32026145, 0.34771243, 0.26013073, …, 0.4       , 0.5542484 ,
       0.82483655], dtype=float32)
```

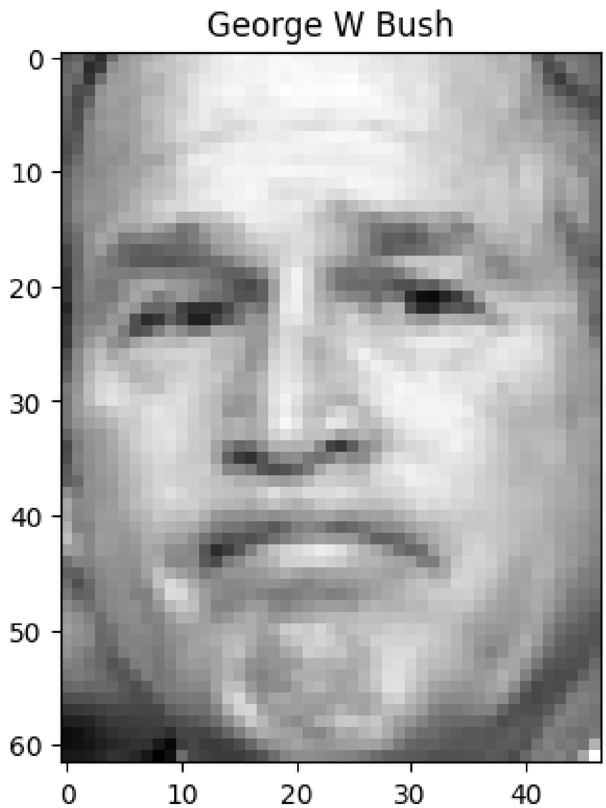
[31]:
```
plt.imshow(lfw.images[0])
plt.title(target_names[Y[0]])
plt.show
```

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



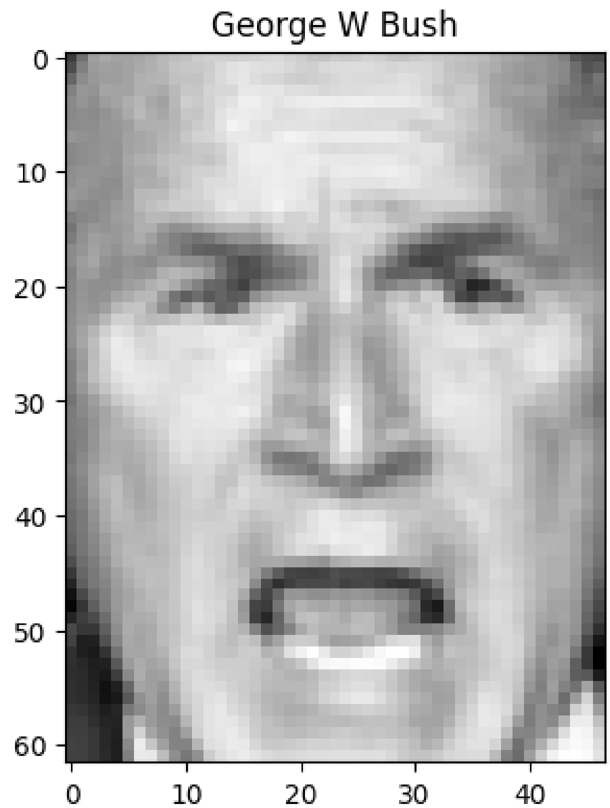
[32]:
```
plt.imshow(lfw.images[0],cmap='gray')
plt.title(target_names[Y[0]])
plt.show
```

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



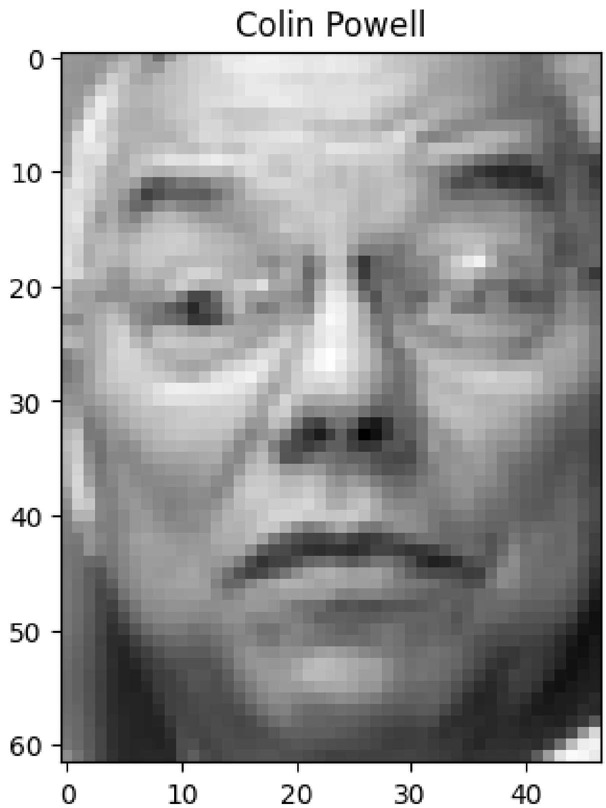
[33]:
```
plt.imshow(lfw.images[100],cmap='gray')
plt.title(target_names[Y[100]])
plt.show
```

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



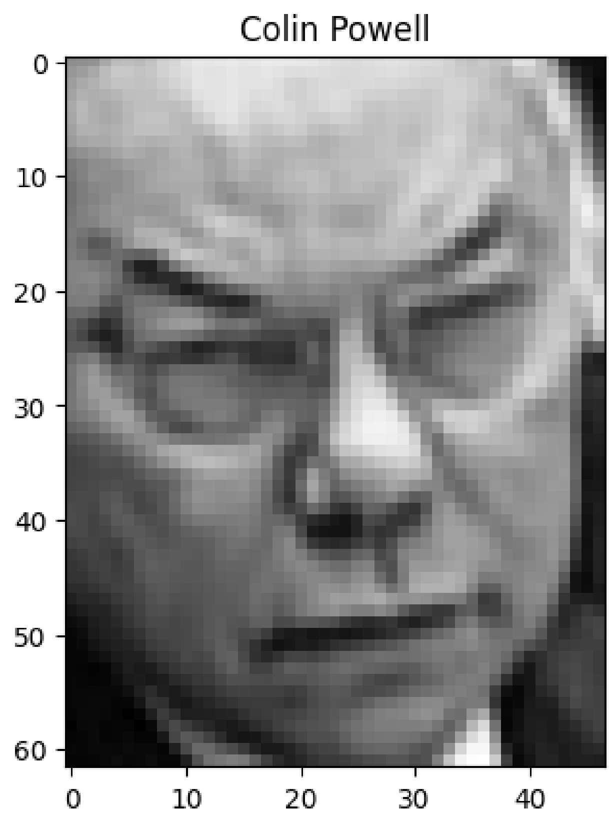
[34]:
```
plt.imshow(lfw.images[101],cmap='gray')
plt.title(target_names[Y[101]])
plt.show
```

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



[35]:
```
plt.imshow(lfw.images[105],cmap='gray')
plt.title(target_names[Y[105]])
plt.show
```

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



[36]:
```
target_names.shape[0]
```

[36]: 5

[37]:
```
X.shape[0]
```

[37]: 1140

[38]:
```
X.shape[1]
```

[38]: 2914

[39]:
```
model=Sequential()
model.add(Dense(256,input_dim=X.shape[1],activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(target_names.shape[0],activation='softmax'))
model.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

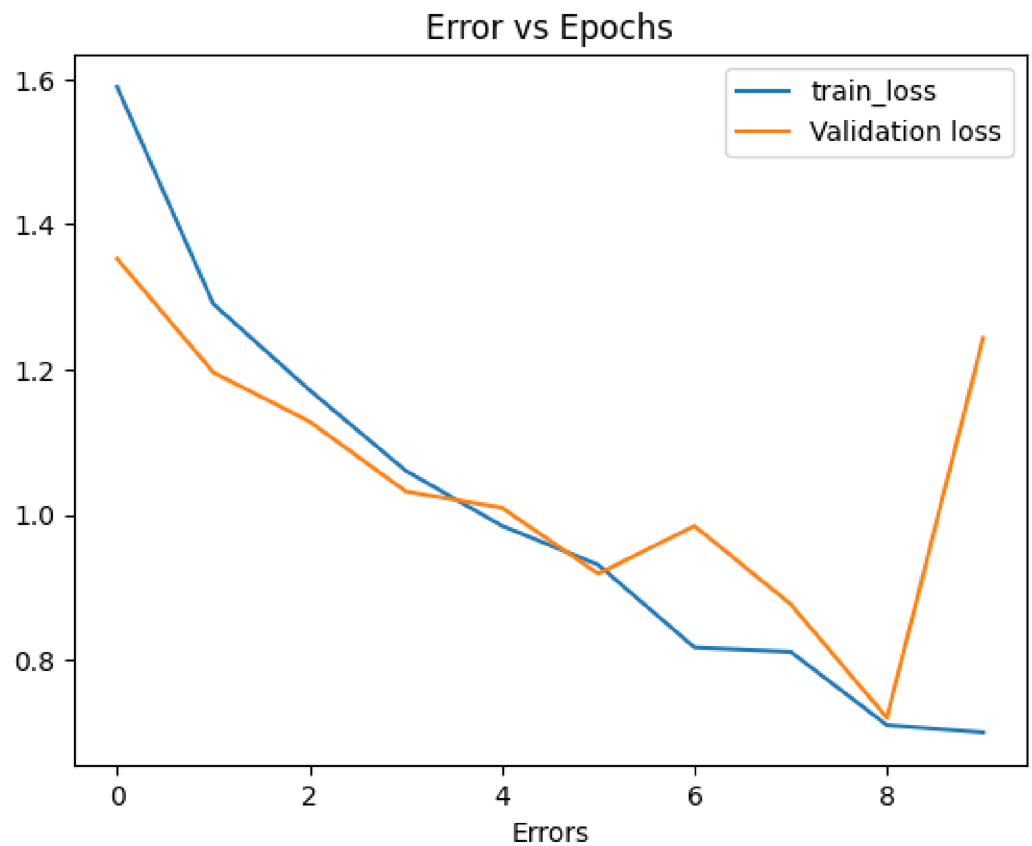
```
=================================================================
 dense_3 (Dense)             (None, 256)               746240

 dense_4 (Dense)             (None, 128)               32896

 dense_5 (Dense)             (None, 5)                 645

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

[40]:

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

```
Epoch 1/10
29/29 [==============================] - 1s 18ms/step - loss: 1.5895 - accuracy:
0.4298 - val_loss: 1.3527 - val_accuracy: 0.5044
Epoch 2/10
29/29 [==============================] - 0s 14ms/step - loss: 1.2903 - accuracy:
0.5439 - val_loss: 1.1960 - val_accuracy: 0.5439
Epoch 3/10
29/29 [==============================] - 0s 12ms/step - loss: 1.1717 - accuracy:
0.5691 - val_loss: 1.1284 - val_accuracy: 0.5833
Epoch 4/10
29/29 [==============================] - 0s 12ms/step - loss: 1.0605 - accuracy:
0.6086 - val_loss: 1.0319 - val_accuracy: 0.6184
Epoch 5/10
29/29 [==============================] - 0s 12ms/step - loss: 0.9846 - accuracy:
0.6327 - val_loss: 1.0095 - val_accuracy: 0.6974
Epoch 6/10
29/29 [==============================] - 0s 12ms/step - loss: 0.9310 - accuracy:
0.6721 - val_loss: 0.9186 - val_accuracy: 0.6974
Epoch 7/10
29/29 [==============================] - 0s 13ms/step - loss: 0.8169 - accuracy:
0.7171 - val_loss: 0.9844 - val_accuracy: 0.6184
Epoch 8/10
29/29 [==============================] - 0s 12ms/step - loss: 0.8110 - accuracy:
0.7138 - val_loss: 0.8766 - val_accuracy: 0.6491
Epoch 9/10
29/29 [==============================] - 0s 13ms/step - loss: 0.7100 - accuracy:
0.7544 - val_loss: 0.7200 - val_accuracy: 0.7368
Epoch 10/10
29/29 [==============================] - 0s 11ms/step - loss: 0.6999 - accuracy:
0.7467 - val_loss: 1.2438 - val_accuracy: 0.4430
```

[41]:
```
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.xlabel("Errors")
plt.legend()
plt.show()
```



[42]:
```
model.compile(tf.keras.optimizers.Adadelta(learning_rate=0.0001,rho=0.
↪9),'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 28ms/step - loss: 1.0265 - accuracy:
0.5735 - val_loss: 1.2399 - val_accuracy: 0.4518
Epoch 2/25
15/15 [==============================] - 0s 20ms/step - loss: 1.0228 - accuracy:
0.5768 - val_loss: 1.2360 - val_accuracy: 0.4518
Epoch 3/25
15/15 [==============================] - 0s 17ms/step - loss: 1.0191 - accuracy:
```

```
0.5789 - val_loss: 1.2319 - val_accuracy: 0.4518
Epoch 4/25
15/15 [==============================] - 0s 19ms/step - loss: 1.0152 - accuracy:
0.5789 - val_loss: 1.2276 - val_accuracy: 0.4561
Epoch 5/25
15/15 [==============================] - 0s 28ms/step - loss: 1.0112 - accuracy:
0.5800 - val_loss: 1.2233 - val_accuracy: 0.4605
Epoch 6/25
15/15 [==============================] - 0s 28ms/step - loss: 1.0072 - accuracy:
0.5811 - val_loss: 1.2190 - val_accuracy: 0.4781
Epoch 7/25
15/15 [==============================] - 0s 28ms/step - loss: 1.0032 - accuracy:
0.5844 - val_loss: 1.2144 - val_accuracy: 0.4781
Epoch 8/25
15/15 [==============================] - 0s 26ms/step - loss: 0.9989 - accuracy:
0.5866 - val_loss: 1.2098 - val_accuracy: 0.4781
Epoch 9/25
15/15 [==============================] - 0s 28ms/step - loss: 0.9946 - accuracy:
0.5866 - val_loss: 1.2052 - val_accuracy: 0.4781
Epoch 10/25
15/15 [==============================] - 0s 30ms/step - loss: 0.9903 - accuracy:
0.5910 - val_loss: 1.2005 - val_accuracy: 0.4825
Epoch 11/25
15/15 [==============================] - 0s 25ms/step - loss: 0.9860 - accuracy:
0.5910 - val_loss: 1.1957 - val_accuracy: 0.4868
Epoch 12/25
15/15 [==============================] - 0s 21ms/step - loss: 0.9815 - accuracy:
0.5910 - val_loss: 1.1910 - val_accuracy: 0.4912
Epoch 13/25
15/15 [==============================] - 0s 18ms/step - loss: 0.9771 - accuracy:
0.5932 - val_loss: 1.1861 - val_accuracy: 0.4956
Epoch 14/25
15/15 [==============================] - 0s 17ms/step - loss: 0.9726 - accuracy:
0.5965 - val_loss: 1.1813 - val_accuracy: 0.4956
Epoch 15/25
15/15 [==============================] - 0s 19ms/step - loss: 0.9681 - accuracy:
0.5965 - val_loss: 1.1764 - val_accuracy: 0.4956
Epoch 16/25
15/15 [==============================] - 0s 18ms/step - loss: 0.9636 - accuracy:
0.6009 - val_loss: 1.1713 - val_accuracy: 0.4956
Epoch 17/25
15/15 [==============================] - 0s 18ms/step - loss: 0.9589 - accuracy:
0.6064 - val_loss: 1.1664 - val_accuracy: 0.4956
Epoch 18/25
15/15 [==============================] - 0s 18ms/step - loss: 0.9544 - accuracy:
0.6064 - val_loss: 1.1614 - val_accuracy: 0.5000
Epoch 19/25
15/15 [==============================] - 0s 17ms/step - loss: 0.9498 - accuracy:
```

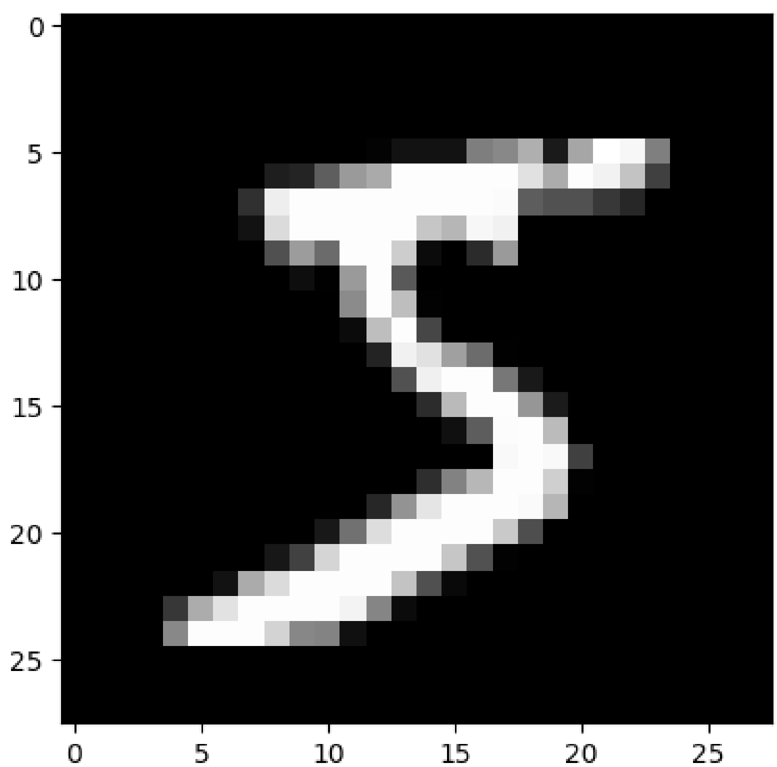
```
0.6107 - val_loss: 1.1562 - val_accuracy: 0.5000
Epoch 20/25
15/15 [==============================] - 0s 19ms/step - loss: 0.9451 - accuracy:
0.6118 - val_loss: 1.1514 - val_accuracy: 0.5044
Epoch 21/25
15/15 [==============================] - 0s 17ms/step - loss: 0.9406 - accuracy:
0.6173 - val_loss: 1.1463 - val_accuracy: 0.5175
Epoch 22/25
15/15 [==============================] - 0s 18ms/step - loss: 0.9360 - accuracy:
0.6239 - val_loss: 1.1412 - val_accuracy: 0.5175
Epoch 23/25
15/15 [==============================] - 0s 18ms/step - loss: 0.9313 - accuracy:
0.6272 - val_loss: 1.1361 - val_accuracy: 0.5175
Epoch 24/25
15/15 [==============================] - 0s 18ms/step - loss: 0.9266 - accuracy:
0.6283 - val_loss: 1.1312 - val_accuracy: 0.5263
Epoch 25/25
15/15 [==============================] - 0s 18ms/step - loss: 0.9221 - accuracy:
0.6316 - val_loss: 1.1262 - val_accuracy: 0.5351
```

[48]:

```
(X_train,Y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data()
```

[49]:

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



knnpima-365

November 12, 2023

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
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
import seaborn as sns
```

[ ]:
```python
path="/content/drive/MyDrive/Machine learning/09 nov/diabetes.csv"

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

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

[ ]:
```python
diabetes.head()
```

[ ]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
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

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

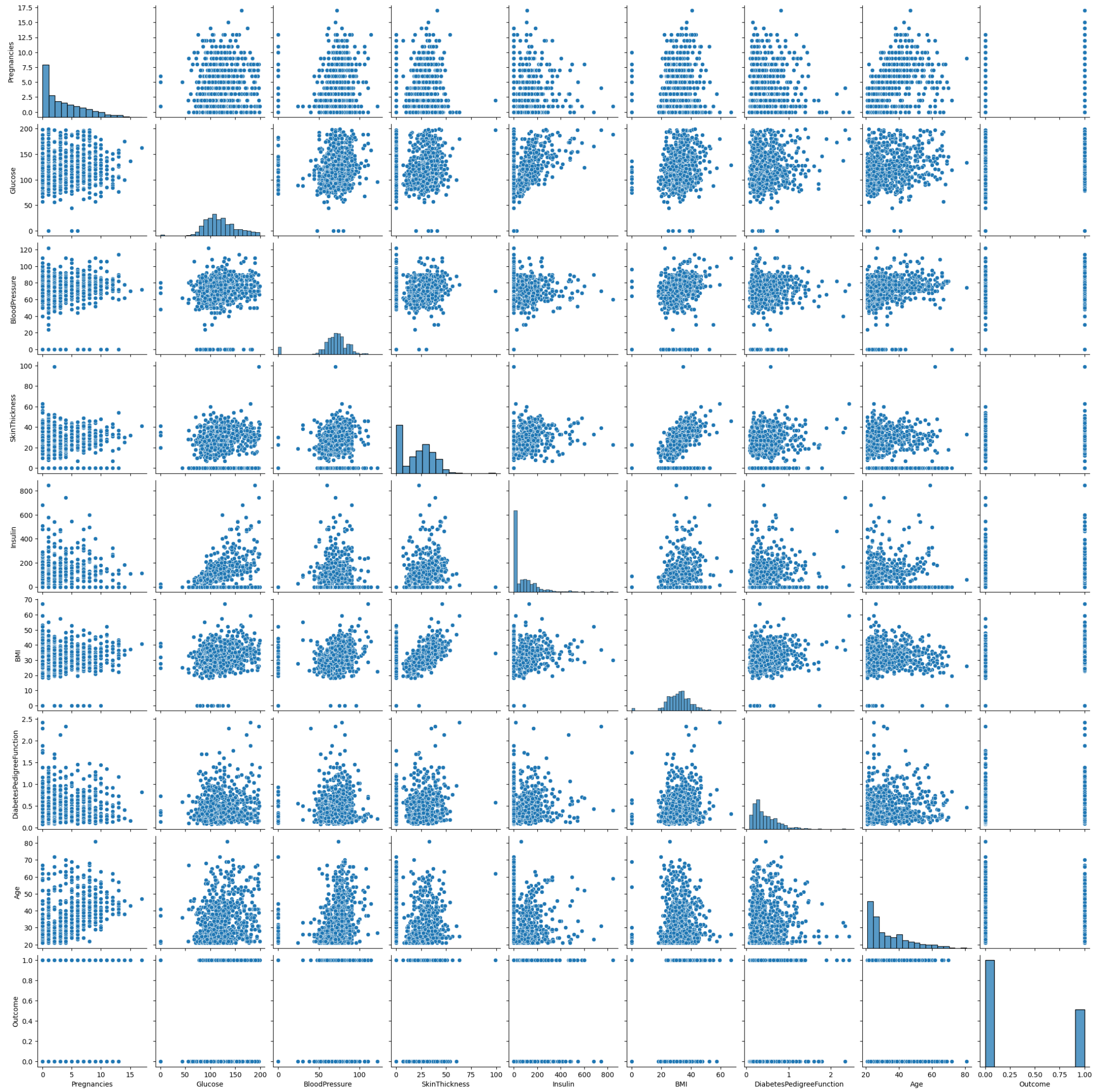
[ ]:
```python
diabetes.describe()
```

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

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[ ]:
```
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)
plt.show()
```



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

[ ]:

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

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

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

[ ]:
```
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  \
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

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
```

[ ]:
```
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)
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 0.7857142857142857
```

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

```python
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
```

breastcancer-365

November 12, 2023

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

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

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

print(feature_names)
print("Feature label: ",features[0])
```

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

[4]:
```
train, test, train_labels, test_labels = train_test_split(features, labels,␣
↪test_size=0.2,random_state=42)
```

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

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

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

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

[7]:
```
print(accuracy_score(test_labels,preds))
```

```
0.9736842105263158
```

loan-prediction-nb-365

November 12, 2023

[4]:
```python
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
```

[5]:
```python
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).
```

[6]:
```python
df=pd.read_csv("/content/drive/MyDrive/Machine learning/Naive Bayes/
↪Bank_Personal_Loan_Modelling.csv")
```

[7]:
```python
df
```

[7]:
```
        ID  Age  Experience  Income  ZIP Code  Family  CCAvg  Education  \
0        1   25           1      49     91107       4    1.6          1
1        2   45          19      34     90089       3    1.5          1
2        3   39          15      11     94720       1    1.0          1
3        4   35           9     100     94112       1    2.7          2
4        5   35           8      45     91330       4    1.0          2
…      …    …           …      …       …       …      …          …
4995  4996   29           3      40     92697       1    1.9          3
4996  4997   30           4      15     92037       4    0.4          1
4997  4998   63          39      24     93023       2    0.3          3
4998  4999   65          40      49     90034       3    0.5          2
4999  5000   28           4      83     92612       3    0.8          1

      Mortgage  Personal Loan  Securities Account  CD Account  Online  \
0            0              0                   1           0       0
1            0              0                   1           0       0
2            0              0                   0           0       0
3            0              0                   0           0       0
```

```
4           0            0                 0           0      0
…       …          …                …        …     …
4995        0            0                 0           0      1
4996       85            0                 0           0      1
4997        0            0                 0           0      0
4998        0            0                 0           0      1
4999        0            0                 0           0      1

      CreditCard
0              0
1              0
2              0
3              0
4              1
…            …
4995           0
4996           0
4997           0
4998           0
4999           1

[5000 rows x 14 columns]
```

[8]:
```
df.shape
```

[8]:
```
(5000, 14)
```

[9]:
```
df.columns
```

[9]:
```
Index(['ID', 'Age', 'Experience', 'Income', 'ZIP Code', 'Family', 'CCAvg',
       'Education', 'Mortgage', 'Personal Loan', 'Securities Account',
       'CD Account', 'Online', 'CreditCard'],
      dtype='object')
```

[10]:
```
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
```

[11]:
```python
df.describe()
```

[11]:
```
                ID          Age   Experience       Income      ZIP Code  \
count  5000.000000  5000.000000  5000.000000  5000.000000   5000.000000
mean   2500.500000    45.338400    20.104600    73.774200  93152.503000
std    1443.520003    11.463166    11.467954    46.033729   2121.852197
min       1.000000    23.000000    -3.000000     8.000000   9307.000000
25%    1250.750000    35.000000    10.000000    39.000000  91911.000000
50%    2500.500000    45.000000    20.000000    64.000000  93437.000000
75%    3750.250000    55.000000    30.000000    98.000000  94608.000000
max    5000.000000    67.000000    43.000000   224.000000  96651.000000

            Family        CCAvg    Education     Mortgage  Personal Loan  \
count  5000.000000  5000.000000  5000.000000  5000.000000    5000.000000
mean      2.396400     1.937938     1.881000    56.498800       0.096000
std       1.147663     1.747659     0.839869   101.713802       0.294621
min       1.000000     0.000000     1.000000     0.000000       0.000000
25%       1.000000     0.700000     1.000000     0.000000       0.000000
50%       2.000000     1.500000     2.000000     0.000000       0.000000
75%       3.000000     2.500000     3.000000   101.000000       0.000000
max       4.000000    10.000000     3.000000   635.000000       1.000000

       Securities Account  CD Account       Online   CreditCard
count         5000.000000  5000.00000  5000.000000  5000.000000
mean             0.104400     0.06040     0.596800     0.294000
std              0.305809     0.23825     0.490589     0.455637
min              0.000000     0.00000     0.000000     0.000000
25%              0.000000     0.00000     0.000000     0.000000
50%              0.000000     0.00000     1.000000     0.000000
75%              0.000000     0.00000     1.000000     1.000000
max              1.000000     1.00000     1.000000     1.000000
```

[12]:
```python
df.isnull().sum()
```

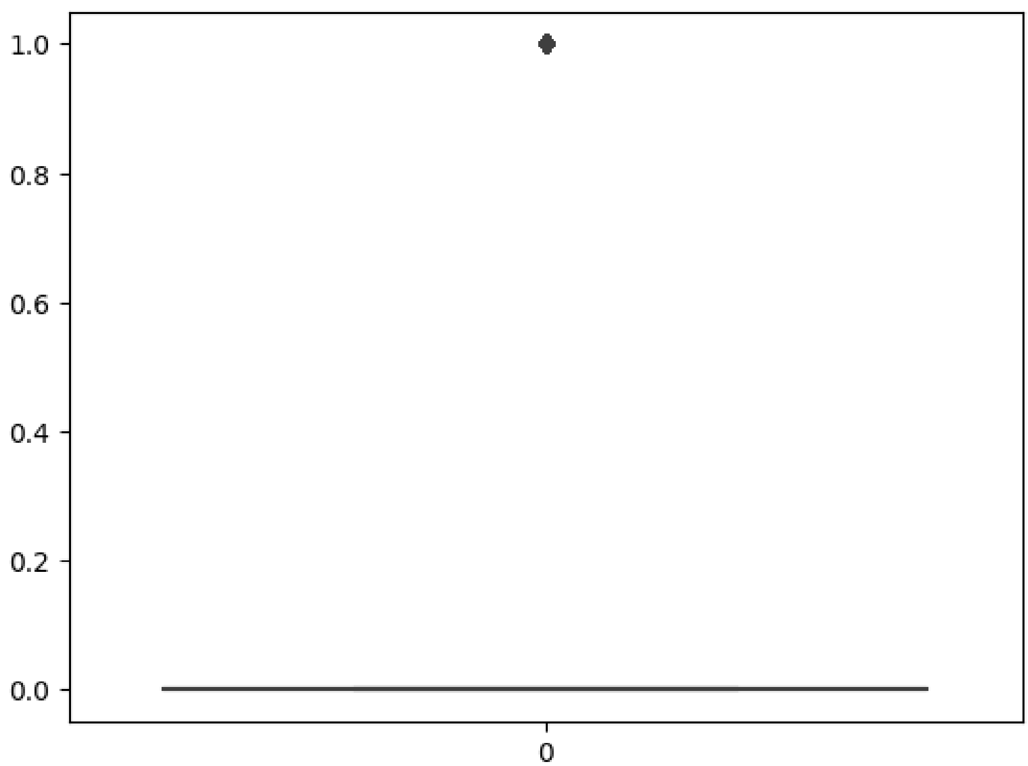
[12]:
```
ID                    0
Age                   0
```

```
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
```

[13]:
```
df.drop('ID',axis=1,inplace=True)
```

[14]:
```
sns.boxplot(df['Personal Loan']);
plt.show
```

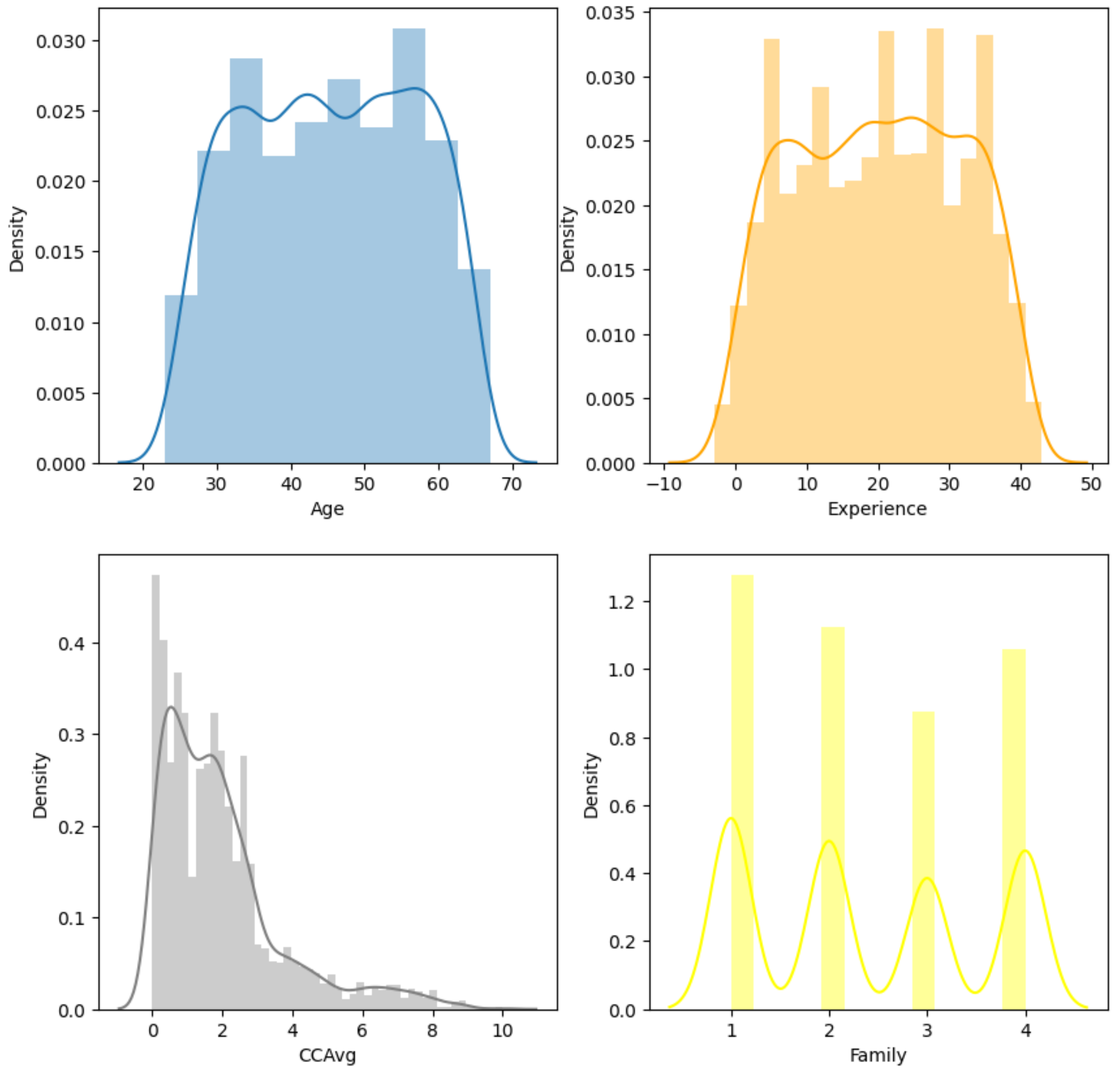
[14]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[15]:
```
fig, axis = plt.subplots (2, 2, figsize=(10, 10), sharex=False)
sns.distplot(df['Age'], bins=10,ax=axis[0,0]);
sns.distplot(df['Experience'], ax=axis [0,1],color='orange');
sns.distplot(df['CCAvg'], ax=axis[1,0], color='gray');
sns.distplot(df['Family'], ax=axis[1,1], color='yellow');
plt.show()
```

```
<ipython-input-15-908094a8f162>:2: UserWarning:

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

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

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

  sns.distplot(df['Age'], bins=10,ax=axis[0,0]);
<ipython-input-15-908094a8f162>:3: UserWarning:

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

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

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

  sns.distplot(df['Experience'], ax=axis [0,1],color='orange');
<ipython-input-15-908094a8f162>:4: UserWarning:

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

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

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

  sns.distplot(df['CCAvg'], ax=axis[1,0], color='gray');
<ipython-input-15-908094a8f162>:5: UserWarning:

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

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

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

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

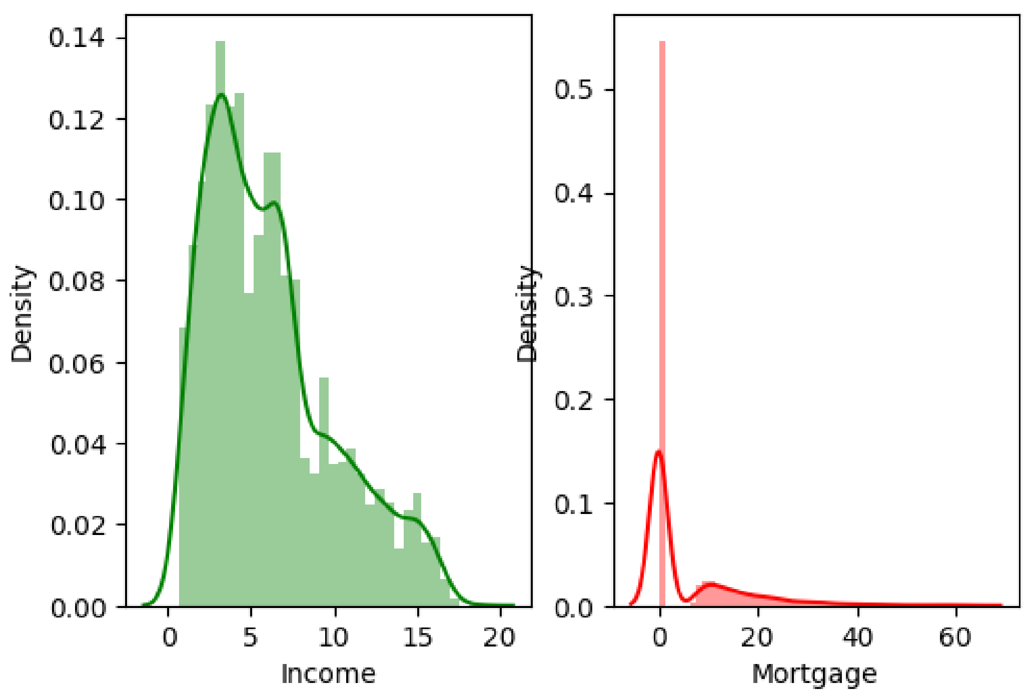


[16]:
```
df['Income']=df['Income']/12
df['Mortgage']=df['Mortgage']/10
```

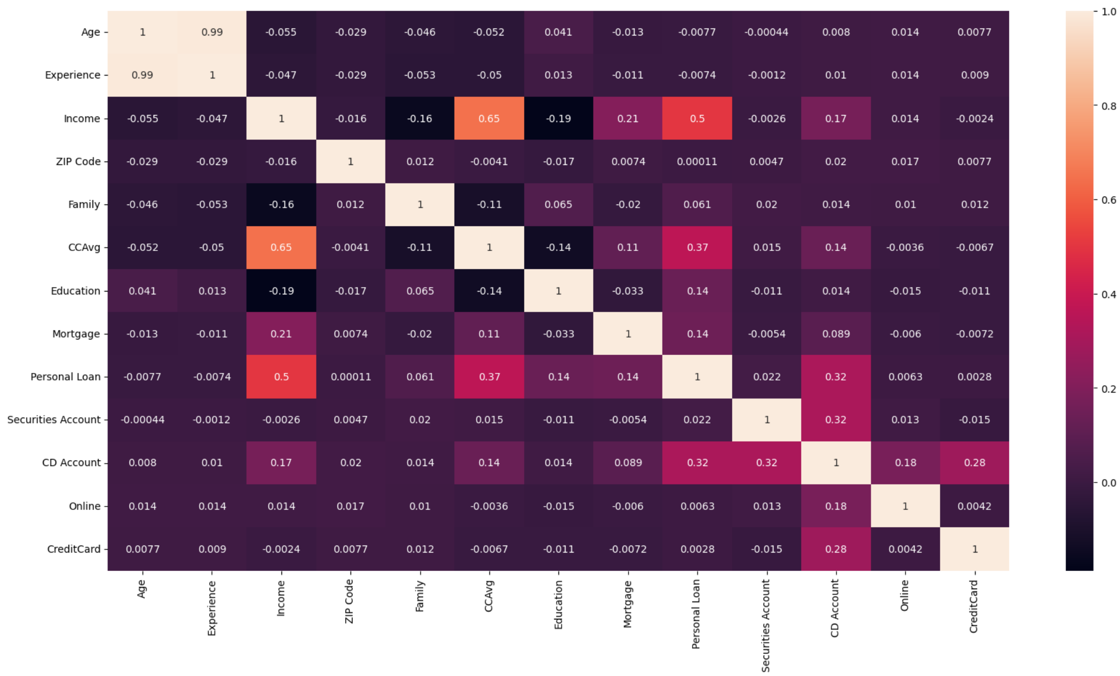
[17]:
```
fig, axis = plt.subplots(1,2, figsize=(6,4), sharex=False)
sns.distplot(df['Income'], ax=axis[0], color='green');
sns.distplot(df['Mortgage'], ax=axis[1], color='red');
plt.show()
```

```
<ipython-input-17-4e2a47603b0c>:2: UserWarning:
```

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

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

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

  sns.distplot(df['Income'], ax=axis[0], color='green');
<ipython-input-17-4e2a47603b0c>:3: UserWarning:

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

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

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

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



[18]:
```python
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True);
plt.show()
```



[19]:
```
x = df.drop(['Personal Loan'], axis=1)
y = df['Personal Loan']
```

[20]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.
↪3,random_state=100)
```

[21]:
```
from sklearn.linear_model import LogisticRegression
```

[22]:
```
logiR = LogisticRegression()
logiR.fit(x_train,y_train)
```

[22]:
```
LogisticRegression()
```

[23]:
```
logiR_test = logiR.predict(x_test)
```

[24]:
```
print("Classification Report")
print(classification_report(y_test, logiR_test))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.97      0.95      1342
           1       0.57      0.32      0.41       158

    accuracy                           0.90      1500
```

```
   macro avg       0.75      0.64      0.68      1500
weighted avg       0.89      0.90      0.89      1500
```

[25]:
```
logiR_predict_train=logiR.predict_proba(x_train)[:,1] > 0.8
logiR_predict_test=logiR.predict_proba(x_test) [:,1]> 0.8
```

[26]:
```
print("Classification Report")
cm =classification_report(y_test,logiR_predict_test, labels=[1,0])
print(cm)
```

```
Classification Report
              precision    recall  f1-score   support

           1       0.33      0.01      0.02       158
           0       0.90      1.00      0.94      1342

    accuracy                           0.89      1500
   macro avg       0.61      0.50      0.48      1500
weighted avg       0.84      0.89      0.85      1500
```

[27]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train,y_train)
```

[27]: GaussianNB()

[28]:
```
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      0.55      0.53       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
```

[29]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

data = load_breast_cancer()

label_names = data['target_names']
labels = data['target']
```

```python
feature_names = data['feature_names']
features = data['data']

print(label_names)
print(labels[0])
print(feature_names[0])
print(features[0])
```

```
['malignant' 'benign']
0
mean radius
[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]
```

[30]:
```python
X_train, X_test, y_train, y_test =␣
 ↪train_test_split(features,labels,random_state=42)
```