Python Basics

Numpy

A) Create a 2x2 numpy array of all ones.

In [75]:

```
#Load numpy package
import numpy as np

s = (2,2)
np.ones(s)
```

Out[75]:

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

B) Create a 3X3 numpy identity matrix.

In [2]:

```
import numpy as np
mat=np.identity(3)
print('3x3: Matrix')
print(mat)
```

```
3x3: Matrix
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

C) Let a = np.array([[1,3,5,9], [6,6,8,8], [12,11,11,12]]). Use slicing to obtain the first row thrid and fourth cells. Display the shape of the array using array attributes.

In [3]:

```
import numpy as np
a = np.array([[1,3,5,9], [6,6,8,8], [12,11,11,12]])
print(a[0,(2,3)])
a.shape #Return the shape of an array
```

```
[5 9]
```

Out[3]:

```
(3, 4)
```

D) Find all elements in "a" that are greater than 10 using boolean indexing.

In [4]:

```
import numpy as np
a = np.array([[1,3,5,9], [6,6,8,8],[12,11,11,12]])
print(a>10)
```

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

E) Change the data type of the elements in "a" to "numpy.int64". Can you find the memory footprint of a before and after this change?

In [5]:

```
type_32 = a.dtype
```

```
type_32 #returning 32-bit integer
```

Out[5]:
```
dtype('int32')
```

In [6]:
```
type_64 = a.astype(np.int64)
type_64 #returning 64-bit integer
```

Out[6]:
```
array([[ 1,  3,  5,  9],
       [ 6,  6,  8,  8],
       [12, 11, 11, 12]], dtype=int64)
```

In [7]:
```
from sys import getsizeof
getsizeof(type_32) #returning the size of the array which is 96 bytes
```

Out[7]:
```
96
```

In [9]:
```
from sys import getsizeof
getsizeof(type_64)  #returning the size of the which is 216 bytes
```

Out[9]:
```
216
```

Pandas

A)Create a pandas series object with entries [10,20,30,30,30,10,20,100,100, numpy.nan].

In [10]:
```
import pandas as pd
import numpy as np
series = np.array([10,20,30,30,30,10,20,100,100, np.nan])
series1 = pd.Series(series)
print(series1)
```

```
0     10.0
1     20.0
2     30.0
3     30.0
4     30.0
5     10.0
6     20.0
7    100.0
8    100.0
9      NaN
dtype: float64
```

B)Create a pandas dataframe from a dictionary (create an example dictionary with different types of values such as strings, integers, pandas series objects etc).

In [11]:
```
Name = ["Joseph", "Pam", "Arnold", "James","Sam","Catherine"]
ID = [210, 211, 114, 178,374,324, ]
likes  = ["apple", "banana", "cherry","blueberry",'blackberry','Mango']
States = ["Utah", "Washington", "Texas", "Hawaii",'Nevada','California']
age = [25, 10, 15, 20, 40, 29]
```

```
df = { 'Name': Name, 'ID': ID, 'likes' : likes, 'States': States,'Age': age}
DF = pd.DataFrame(df)
DF
```

Out[11]:

	Name	ID	likes	States	Age
0	Joseph	210	apple	Utah	25
1	Pam	211	banana	Washington	10
2	Arnold	114	cherry	Texas	15
3	James	178	blueberry	Hawaii	20
4	Sam	374	blackberry	Nevada	40
5	Catherine	324	Mango	California	29

C) Display the data types of the above dataframe. Also show the head and tail samples.

In [12]:

```
print(DF.dtypes)
```

```
Name       object
ID          int64
likes      object
States     object
Age         int64
dtype: object
```

In [13]:

```
print ("The first two rows of the series: ")
print(DF.head(2), "\n \n")
print("The last two rows of the series: ")
print(DF.tail(2))
```

```
The first two rows of the series:
     Name   ID   likes      States  Age
0  Joseph  210   apple        Utah   25
1     Pam  211  banana  Washington   10


The last two rows of the series:
        Name   ID       likes      States  Age
4        Sam  374  blackberry      Nevada   40
5  Catherine  324       Mango  California   29
```

D) What does the describe method do? Show its operation on the dataframe created above.

In [14]:

```
# Pandas describe() is used to view some basic statistical details like percentile, mea
DF.describe()
```

Out[14]:

	ID	Age
count	6.000000	6.000000
mean	235.166667	23.166667
std	96.263008	10.684880
min	114.000000	10.000000

	ID	Age
25%	186.000000	16.250000
50%	210.500000	22.500000
75%	295.750000	28.000000
max	374.000000	40.000000

E) Download the iris dataset and use the read_csv method to read the data into a data frame. Display the data types, show a sample and describe the dataframe.

In [15]:

```
df2 = pd.read_csv(r'C:\Users\arasto6\Downloads\Iris.csv')
```

In [16]:

```
df2.dtypes
```

Out[16]:

```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

In [17]:

```
df2.sample(5) #return random 5 samples of the dataframe
```

Out[17]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
133	134	6.3	2.8	5.1	1.5	Iris-virginica
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
78	79	6.0	2.9	4.5	1.5	Iris-versicolor

In [18]:

```
df2.describe()
```

Out[18]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
max	150.000000	7.900000	4.400000	6.900000	2.500000

MATPLOTLIB

A) Create a x-y plot of the series object create above after dropping the last element. Change the y-label and x-label to 'prices' and 'years' respectively.

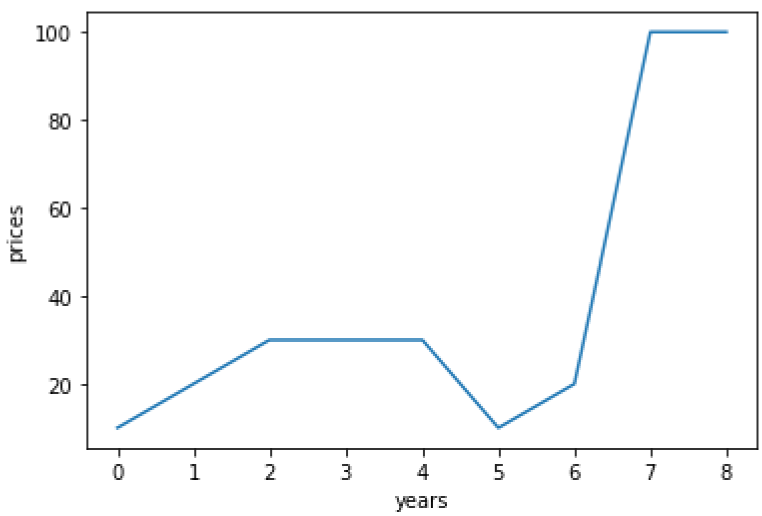
In [19]:

```
new_series= series1.drop([9]);
```

In [20]:

```
import matplotlib.pyplot as plt
%matplotlib inline

p = plt.plot(new_series)
plt.xlabel("years")
plt.ylabel("prices")
```

Out[20]: `Text(0, 0.5, 'prices')`

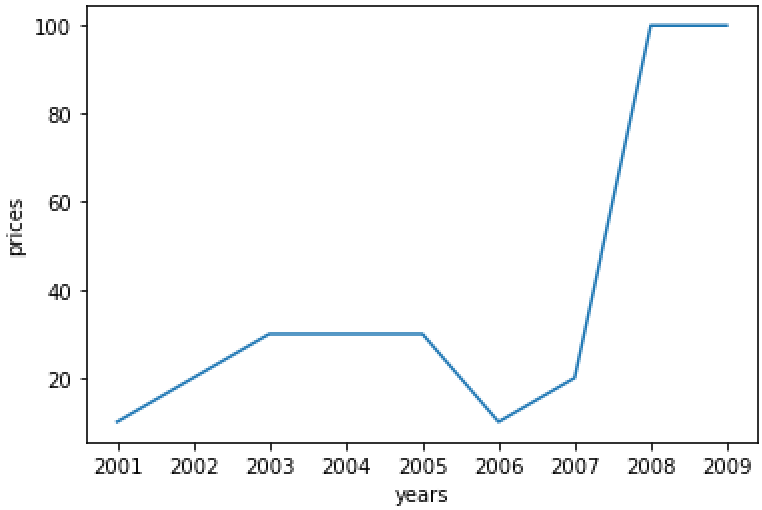


B) Change the tick labels to reflect years starting from 2001.

In [21]:

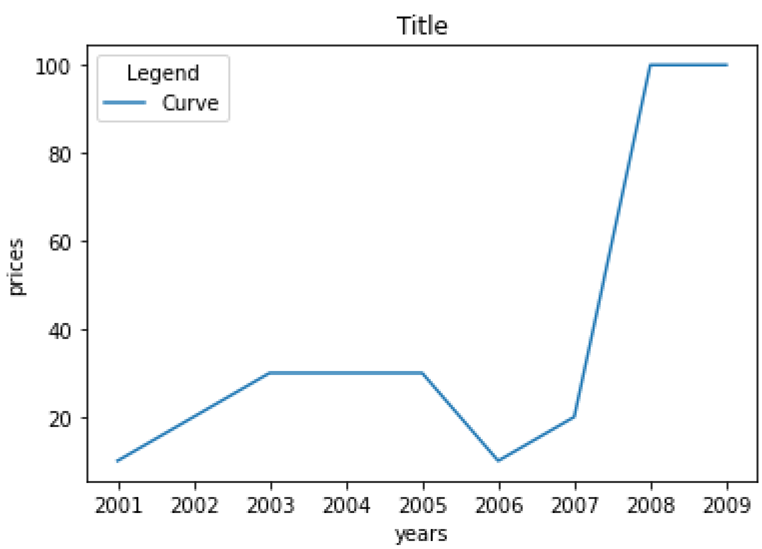
```
x_ticks = np.arange(2001, 2010, 1)
plt.xticks(x_ticks)
plt.plot(x_ticks, new_series)
plt.xlabel("years")
plt.ylabel("prices")
```

Out[21]: `Text(0, 0.5, 'prices')`



C) Add a legend and a plot title.

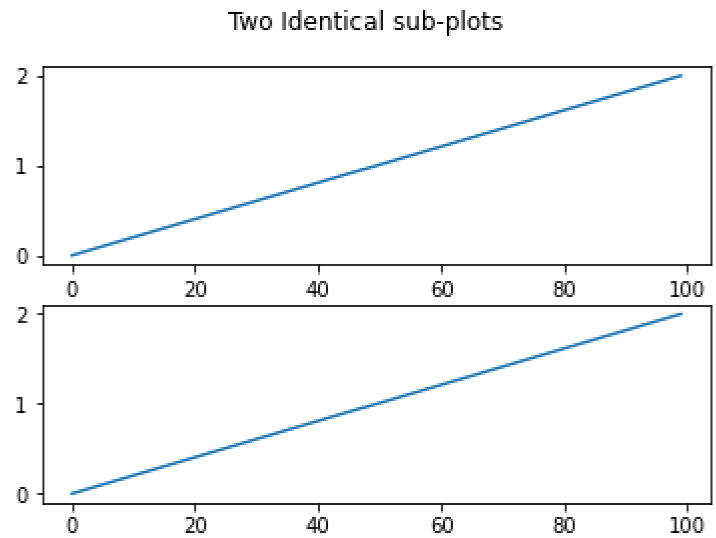
In [22]:
```
x_ticks = np.arange(2001, 2010, 1)
plt.xticks(x_ticks)
plt.plot(x_ticks, new_series)
plt.xlabel("years")
plt.ylabel("prices")

plt.legend(['Curve'], title = "Legend")
plt.title("Title")

plt.show()
```



D) In a new figure, make two identical sub-plots with the data numpy.linspace(0, 2, 100).

In [23]:
```
fig, axs = plt.subplots(2)
fig.suptitle('Two Identical sub-plots')

axs[0].plot(np.linspace(start = 0, stop = 2, num = 100))
axs[1].plot(np.linspace(start = 0, stop = 2, num = 100))
```

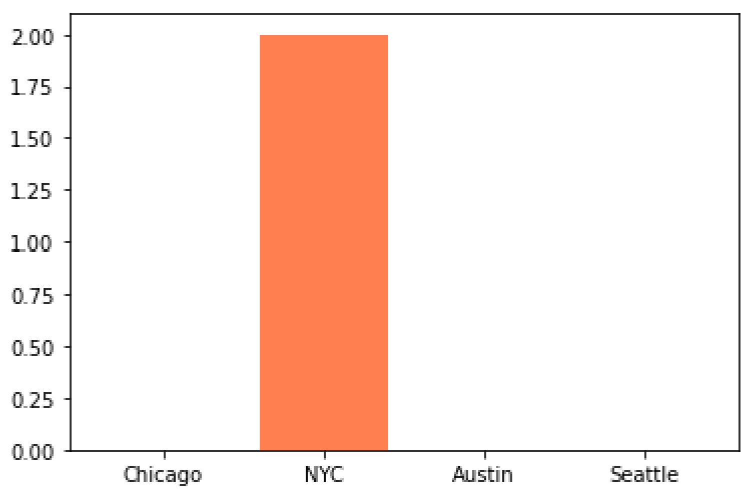
Out[23]:
```
[<matplotlib.lines.Line2D at 0x1eef5f0b400>]
```



E) Create categories ['Chicago','NYC','Austin','Seattle']. Generate a random number corresponding to each category using numpy. Make a bar plot with the appropriate category labels.

In [25]:
```
cat = pd.Series(['Chicago','NYC','Austin','Seattle'], dtype= "category")
ran= np.random.randint(3, size=4) #random numbers are choosed between 0-3

plt.bar(cat, ran, color ='coral')
```

Out[25]: <BarContainer object of 4 artists>



Create Datasets

A) Use the make_regression and make_classification functions in scikit-learn to create two datasets.

In [26]:
```
from sklearn.datasets import make_regression
X,Y = make_regression(n_samples=100, n_features=4)

data1 = pd.DataFrame(X, columns=('1st','2nd','3rd','4th'))

data2 = pd.DataFrame(Y, columns=['label'])
```

```
newdata= pd.concat([data1, data2], axis=1)
newdata
```

Out[26]:

	1st	2nd	3rd	4th	label
0	-0.499138	0.858177	0.159986	-1.171860	-74.259002
1	-0.866263	0.109520	1.174105	-0.158697	74.650888
2	-0.340721	-1.220383	-0.836579	0.513191	-87.340864
3	-1.361040	-0.359777	-2.362010	-0.472662	-309.946293
4	-1.505780	1.837600	-0.009188	0.977922	134.678667
...	...	...	...	...	...
95	-0.198902	0.468632	0.206657	0.914912	122.222515
96	-0.125744	0.158545	3.214112	0.377784	332.680268
97	-0.185615	0.118428	0.460292	-0.145522	28.294041
98	0.305829	0.314943	0.918509	-0.425073	63.679880
99	-0.766469	0.361219	1.699721	-0.475276	105.197958

100 rows × 5 columns

In [27]:

```
import collections
from sklearn.datasets import make_classification

x,y = make_classification(n_samples=100, n_features=4, n_informative=2, n_classes=2, n_

df1 = pd.DataFrame(x, columns=('1st','2nd','3rd','4th'))

df2 = pd.DataFrame(y, columns=['label'])

newdf =pd.concat([df1, df2], axis=1)
newdf
```

Out[27]:

	1st	2nd	3rd	4th	label
0	-0.975324	1.695904	2.238686	2.184191	1
1	-0.078048	-0.113583	-0.273714	-0.019894	1
2	0.174480	0.317546	0.727483	0.094162	1
3	-1.434376	1.573440	1.619898	2.493244	1
4	-1.179521	-0.071453	-1.148141	0.984038	0
...	...	...	...	...	...
95	-1.172189	1.527408	1.762636	2.226159	1
96	-0.956861	-0.257089	-1.293129	0.642778	0
97	0.263846	1.397709	2.766832	0.858902	1
98	0.906950	-0.219745	0.383834	-0.971149	1

	1st	2nd	3rd	4th	label
99	1.254329	-0.799407	-0.369254	-1.730060	0

100 rows × 5 columns

2) Perform a exploratory analysis of these two datasets.

A) Describe the feature wise statistics if any.

In [28]:

```
#For Classification
import statsmodels.api as sm

SM = sm.OLS(df2, df1).fit()
print(SM.summary())
```

```
                                 OLS Regression Results                                
=======================================================================================
Dep. Variable:                  label   R-squared (uncentered):                   0.379
Model:                            OLS   Adj. R-squared (uncentered):              0.366
Method:                 Least Squares   F-statistic:                              29.89
Date:                Sun, 06 Feb 2022   Prob (F-statistic):                    7.34e-11
Time:                        22:27:40   Log-Likelihood:                         -83.425
No. Observations:                 100   AIC:                                      170.8
Df Residuals:                      98   BIC:                                      176.1
Df Model:                           2                                                  
Covariance Type:            nonrobust                                                  
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
1st            0.0984      0.019      5.268      0.000       0.061       0.135
2nd            0.1016      0.017      5.908      0.000       0.067       0.136
3rd            0.2696      0.036      7.519      0.000       0.198       0.341
4th           -0.0074      0.021     -0.356      0.723      -0.049       0.034
==============================================================================
Omnibus:                        7.080   Durbin-Watson:                   0.719
Prob(Omnibus):                  0.029   Jarque-Bera (JB):                6.571
Skew:                           0.551   Prob(JB):                       0.0374
Kurtosis:                       3.603   Cond. No.                     9.75e+15
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a con
stant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[3] The smallest eigenvalue is 4.53e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [29]:

```
#for Regression
import statsmodels.api as sm

SM1 = sm.OLS(data2, data1).fit()
print(SM1.summary())
```

```
                                 OLS Regression Results                                
=======================================================================================
Dep. Variable:                  label   R-squared (uncentered):                   1.000
Model:                            OLS   Adj. R-squared (uncentered):              1.000
Method:                 Least Squares   F-statistic:                          1.921e+32
```

```
Date:                  Sun, 06 Feb 2022   Prob (F-statistic):                   0.00
Time:                          22:27:44   Log-Likelihood:                     2909.5
No. Observations:                   100   AIC:                                -5811.
Df Residuals:                        96   BIC:                                -5801.
Df Model:                             4
Covariance Type:              nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
1st           24.9244   5.81e-15   4.29e+15      0.000      24.924      24.924
2nd           42.8018   5.56e-15   7.69e+15      0.000      42.802      42.802
3rd           91.0246    5.4e-15   1.68e+16      0.000      91.025      91.025
4th           96.5240   5.45e-15   1.77e+16      0.000      96.524      96.524
==============================================================================
Omnibus:                        1.238   Durbin-Watson:                   1.874
Prob(Omnibus):                  0.538   Jarque-Bera (JB):                1.315
Skew:                          -0.219   Prob(JB):                        0.518
Kurtosis:                       2.647   Cond. No.                         1.18
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a con
stant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
```

In [30]:

```
print('regression:','\n', newdata.head(), '\n')
print('classification:','\n', newdf.head())
```

```
regression:
         1st       2nd       3rd       4th       label
0 -0.499138  0.858177  0.159986 -1.171860  -74.259002
1 -0.866263  0.109520  1.174105 -0.158697   74.650888
2 -0.340721 -1.220383 -0.836579  0.513191  -87.340864
3 -1.361040 -0.359777 -2.362010 -0.472662 -309.946293
4 -1.505780  1.837600 -0.009188  0.977922  134.678667

classification:
         1st       2nd       3rd       4th  label
0 -0.975324  1.695904  2.238686  2.184191      1
1 -0.078048 -0.113583 -0.273714 -0.019894      1
2  0.174480  0.317546  0.727483  0.094162      1
3 -1.434376  1.573440  1.619898  2.493244      1
4 -1.179521 -0.071453 -1.148141  0.984038      0
```

In [31]:

```
print('Classification', '\n')
newdf.info()
print('\n\n')
print('Regression', '\n')
newdata.info()
```

```
Classification

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   1st     100 non-null    float64
 1   2nd     100 non-null    float64
 2   3rd     100 non-null    float64
 3   4th     100 non-null    float64
 4   label   100 non-null    int32  
```

```
dtypes: float64(4), int32(1)
memory usage: 3.6 KB



Regression

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   1st     100 non-null    float64
 1   2nd     100 non-null    float64
 2   3rd     100 non-null    float64
 3   4th     100 non-null    float64
 4   label   100 non-null    float64
dtypes: float64(5)
memory usage: 4.0 KB
```

In [32]:

```
mean_1 = newdata['1st'].mean()
mean_2 = newdata['2nd'].mean()
mean_3 = newdata['3rd'].mean()
mean_4 = newdata['4th'].mean()

print('First column mean', mean_1,'\n','Second column mean', mean_2,'\n', 'Third column
```

```
First column mean -0.18824120220381332
 Second column mean 0.12082432305423069
 Third column mean 0.11286808246259367
 Fourth column mean 0.12970186816027854
```

In [33]:

```
mean1 = newdf['1st'].mean()
mean2 = newdf['2nd'].mean()
mean3 = newdf['3rd'].mean()
mean4 = newdf['4th'].mean()

print('First column mean', mean1,'\n','Second column mean', mean2,'\n', 'Third column m
```

```
First column mean 0.06267426012929316
 Second column mean 0.022682800692285506
 Third column mean 0.09531486224398249
 Fourth column mean -0.037538567634796884
```

In [34]:

```
newdf.describe()
```

Out[34]:

	1st	2nd	3rd	4th	label
count	100.000000	100.000000	100.000000	100.000000	100.000000
mean	0.062674	0.022683	0.095315	-0.037539	0.500000
std	1.316736	0.761540	1.252700	1.555522	0.502519
min	-2.688992	-1.489646	-2.989087	-3.201692	0.000000
25%	-0.979421	-0.744798	-0.961837	-1.113194	0.000000
50%	0.203368	-0.095507	0.073274	-0.388874	0.500000
75%	0.916830	0.436108	1.173546	1.098918	1.000000

	1st	2nd	3rd	4th	label
max	3.172562	2.043305	2.919978	3.884182	1.000000

In [35]:
```
newdata.describe()
```

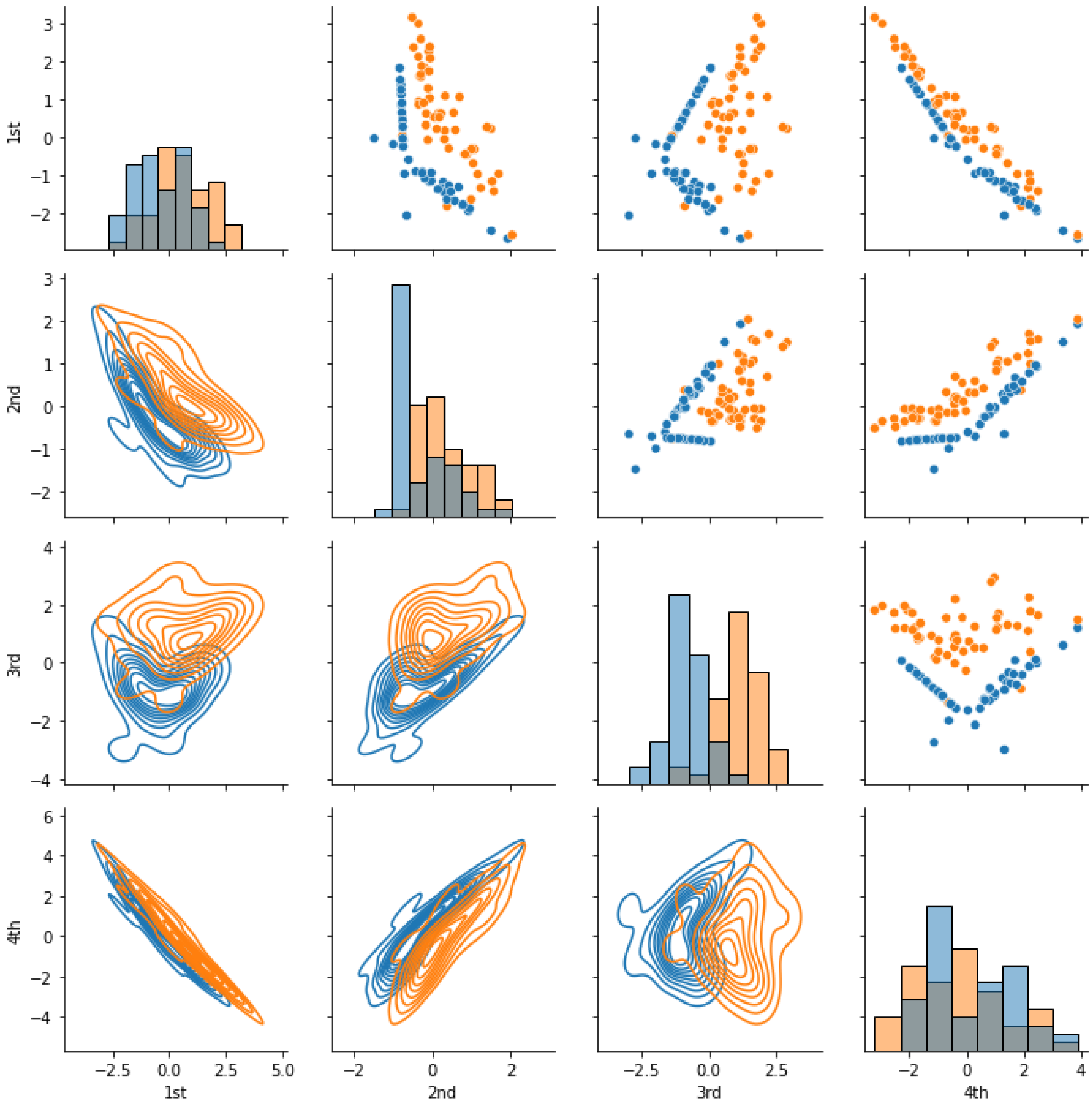
Out[35]:

	1st	2nd	3rd	4th	label
count	100.000000	100.000000	100.000000	100.000000	100.000000
mean	-0.188241	0.120824	0.112868	0.129702	23.272817
std	0.974252	1.036614	1.065376	1.051905	157.456441
min	-2.759686	-3.222560	-2.362010	-1.905228	-309.946293
25%	-0.868872	-0.415260	-0.629849	-0.525902	-68.412983
50%	-0.243841	0.159292	0.075351	0.165066	27.257292
75%	0.561043	0.715583	0.918760	0.946936	129.143127
max	1.918251	3.162471	3.214112	2.765834	380.321231

B) Plot the marginal distributions.

In [36]:
```
import seaborn as sns
import matplotlib.pyplot as plt

g = sns.PairGrid(newdf, hue="label")

g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.histplot)
```

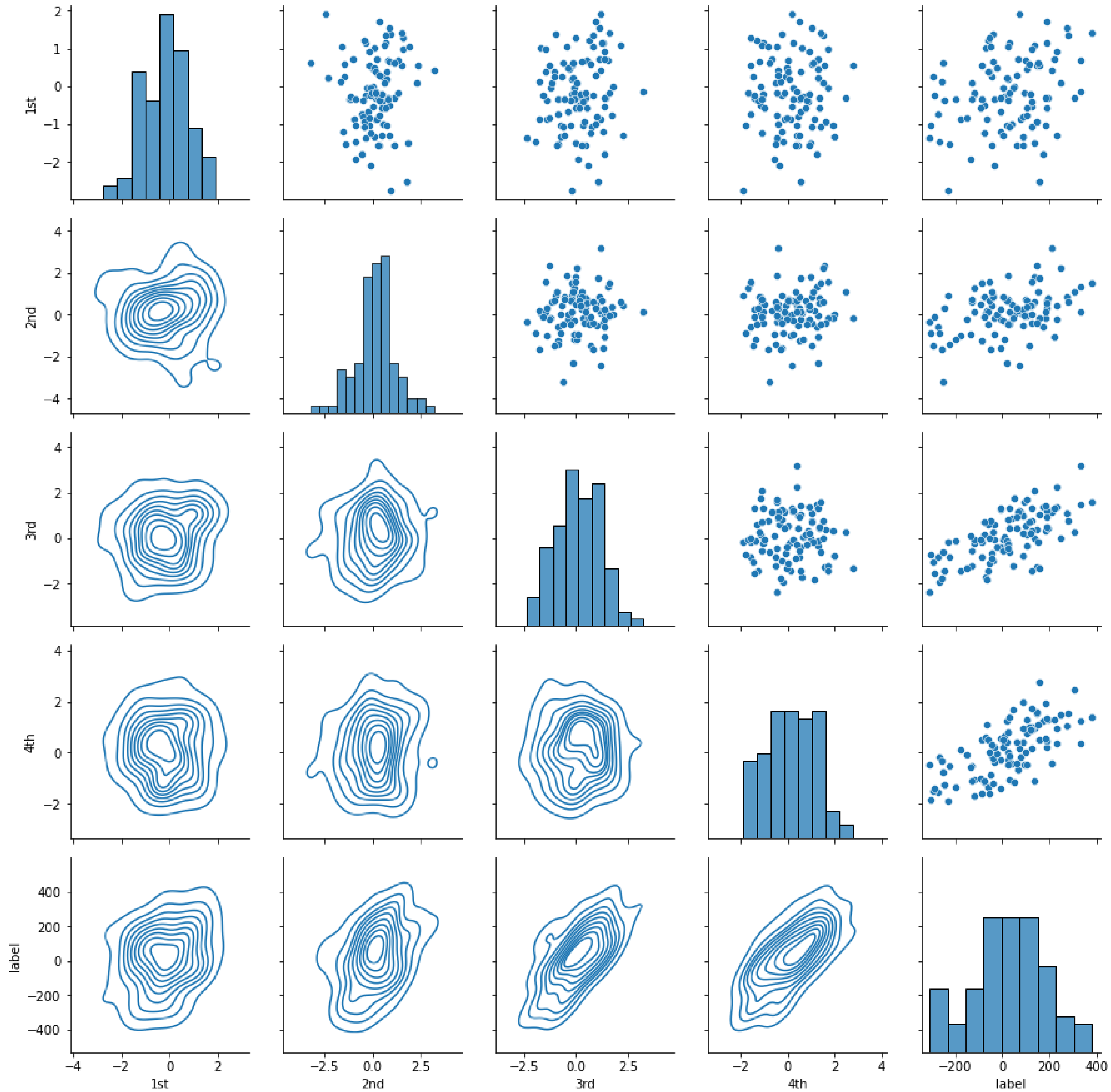
Out[36]:
```
<seaborn.axisgrid.PairGrid at 0x1eef9313d90>
```



In [37]:

```
h = sns.PairGrid(newdata)

h.map_upper(sns.scatterplot)
h.map_lower(sns.kdeplot)
h.map_diag(sns.histplot)
```

Out[37]: <seaborn.axisgrid.PairGrid at 0x1eefafa3970>



C) What is the size of the input space?

In [38]:

```
#regression
import sys

size_reg = data1.shape
print("Size of input space of Regression: ", size_reg)
```

```
Size of input space of Regression:  (100, 4)
```

In [39]:

```
#classification
import sys

size = df1.shape
print("Size of input space of Classification: ", size)
```

```
Size of input space of Classification:  (100, 4)
```

D) What is the size of the output space?

In [40]:

```
#regression
import sys

size_out_reg = data2.shape
print("Size of output space of Regression: ", size_out_reg)
```

```
Size of output space of Regression:  (100, 1)
```

In [41]:

```
import sys

size_output = df2.shape
print("Size of output space of Classification: ", size_output)
```

```
Size of output space of Classification:  (100, 1)
```

E) How can the same datasets be obtained if the functions are rerun? Explain.

In [42]:

```
X,Y = make_regression(n_samples=100, n_features=4, random_state=42)

data1 = pd.DataFrame(X, columns=('1st','2nd','3rd','4th'))

data2 = pd.DataFrame(Y, columns=['label'])

newdata= pd.concat([data1, data2], axis=1)

print('Same Datasets in Regression: ', '\n\n', newdata)
```

```
Same Datasets in Regression:

          1st       2nd       3rd       4th       label
0  -1.448084 -1.407464  0.232050 -0.471038 -143.065368
1  -1.918771 -0.026514 -0.074446  0.257550   -4.461594
2   0.005113 -0.234587  0.261055  0.296120   -0.835365
3  -0.485364  0.081874 -0.236819 -0.772825  -34.189506
4   0.024510  0.497998 -0.773010  0.097676   26.118365
..       ...       ...       ...       ...         ...
95 -0.463418 -0.465730  0.542560 -0.469474  -47.525972
96 -0.908024 -1.412304  0.314247 -1.012831 -160.666558
97  2.189803 -0.808298  0.183342  0.872321  -16.351977
98 -0.981509  0.462103  0.010233  1.163164   82.955898
99 -0.818221  2.092387  0.595157  0.096996  196.616246

[100 rows x 5 columns]
```

In [43]:

```
x,y = make_classification(n_samples=100, n_features=4, n_informative=2, n_classes=2, n_

df1 = pd.DataFrame(x, columns=('1st','2nd','3rd','4th'))

df2 = pd.DataFrame(y, columns=['label'])

newdf =pd.concat([df1, df2], axis=1)

print('Same Datasets in Classification: ', '\n\n', newdf)
```

```
Same Datasets in Classification:

          1st       2nd       3rd       4th  label
0  -1.053839 -1.027544 -0.329294  0.826007      1
1   1.569317  1.306542 -0.239385 -0.331376      0
2  -0.358856 -0.691021 -1.225329  1.652145      1
```

```
3  -0.136856  0.460938  1.896911 -2.281386      0
4  -0.048629  0.502301  1.778730 -2.171053      0
..       ...       ...       ...       ...    ...
95 -2.241820 -1.248690  2.357902 -2.009185      0
96  0.573042  0.362054 -0.462814  0.341294      1
97 -0.375121 -0.149518  0.588465 -0.575002      0
98  1.594888  0.780256 -2.030223  1.863789      1
99 -0.149941 -0.566037 -1.416933  1.804741      1

[100 rows x 5 columns]
```

In [44]:

```
#alternative method for Regression
import random
random.seed(1234)
newdata
```

Out[44]:

	1st	2nd	3rd	4th	label
0	-1.448084	-1.407464	0.232050	-0.471038	-143.065368
1	-1.918771	-0.026514	-0.074446	0.257550	-4.461594
2	0.005113	-0.234587	0.261055	0.296120	-0.835365
3	-0.485364	0.081874	-0.236819	-0.772825	-34.189506
4	0.024510	0.497998	-0.773010	0.097676	26.118365
...	...	...	...	...	...
95	-0.463418	-0.465730	0.542560	-0.469474	-47.525972
96	-0.908024	-1.412304	0.314247	-1.012831	-160.666558
97	2.189803	-0.808298	0.183342	0.872321	-16.351977
98	-0.981509	0.462103	0.010233	1.163164	82.955898
99	-0.818221	2.092387	0.595157	0.096996	196.616246

100 rows × 5 columns

In [45]:

```
#alternative method for Classification
import random
random.seed(1234)
newdf
```

Out[45]:

	1st	2nd	3rd	4th	label
0	-1.053839	-1.027544	-0.329294	0.826007	1
1	1.569317	1.306542	-0.239385	-0.331376	0
2	-0.358856	-0.691021	-1.225329	1.652145	1
3	-0.136856	0.460938	1.896911	-2.281386	0
4	-0.048629	0.502301	1.778730	-2.171053	0
...	...	...	...	...	...
95	-2.241820	-1.248690	2.357902	-2.009185	0

	1st	2nd	3rd	4th	label
96	0.573042	0.362054	-0.462814	0.341294	1
97	-0.375121	-0.149518	0.588465	-0.575002	0
98	1.594888	0.780256	-2.030223	1.863789	1
99	-0.149941	-0.566037	-1.416933	1.804741	1

100 rows × 5 columns

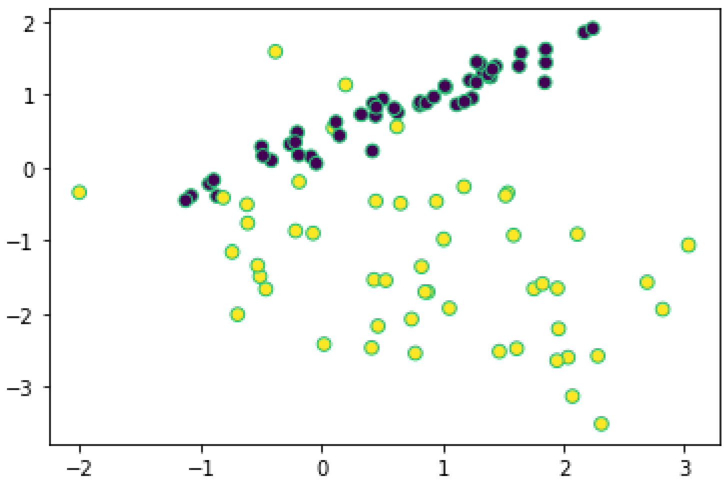
Implement K Nearest Neighbor

1) Create a random two dimensional classification dataset (N=100) based on Q2. Assign a different color to each class uniformly at random and plot using matplotib.

In [77]:

```
from sklearn.datasets import make_classification
import random
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


feat, output = make_classification(n_samples=100, n_features=2, n_classes=2, n_redundan

dataframe = pd.DataFrame(feat, columns=('Column1', 'Column2'))

dataframe1 = pd.DataFrame(output, columns=['OUTPUT'])

dataframe2 = dataframe.join(dataframe1)

print(dataframe2)
r = random.random()
b = random.random()
g = random.random()
color = (r, g, b)

plt.scatter(feat[:, 0], feat[:, 1], marker="o", c=output, s=45, edgecolor=color)

plt.show()
```

```
     Column1   Column2  OUTPUT
0   1.332630  1.301400       0
1   2.113671 -0.913082       1
2   2.073144 -3.127121       1
3   1.844294  1.167010       0
4  -0.928564 -0.223226       0
..       ...       ...     ...
95  1.180458  0.903490       0
96  0.463968 -2.164827       1
97 -0.215246  0.348911       0
98  0.824642 -1.350172       1
99  1.410364  1.346341       0

[100 rows x 3 columns]
```

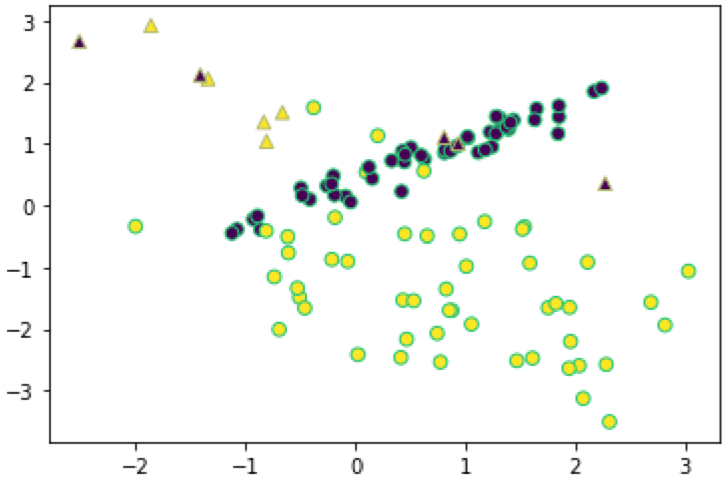


2) Generate 10 new points that belong to the same input space. Plot them using a different color in the above plot.

In [78]:

```
fig=plt.figure()
ax1=fig.add_subplot(111)

a = random.random()
x = random.random()
c = random.random()

colors = (a, x, c)

x, y = make_classification(n_samples=10, n_features=2, n_classes=2, n_redundant=0, n_cl
dataframe3 = pd.DataFrame(x, columns=('Column1', 'Column2'))

dataframe4 = pd.DataFrame(y, columns=['OUTPUT'])

dataframe5 = dataframe3.join(dataframe4)
frames=[dataframe2,dataframe5]
result=pd.concat(frames)
print(result)

plt.scatter(feat[:, 0], feat[:, 1], marker="o", c=output, s=45, edgecolor=color)
plt.scatter(x[:, 0], x[:, 1], marker="^", c=y, s=45, edgecolor=colors)
plt.show()
```

```
     Column1   Column2  OUTPUT
0   1.332630  1.301400       0
1   2.113671 -0.913082       1
2   2.073144 -3.127121       1
3   1.844294  1.167010       0
4  -0.928564 -0.223226       0
..       ...       ...     ...
5   0.806625  1.107072       0
6  -0.832515  1.354747       1
7  -1.862120  2.923807       1
8  -0.808645  1.042214       1
9  -1.412206  2.120806       0

[110 rows x 3 columns]
```



3) Write the unweighted K Nearest Neighbor API from scratch. Define the class appropraitely such that knn object instances with different values of k can be created. Define similarity using Euclidean distance.

In [79]:

```
import operator



def euc_dist(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))


class KNearestNeighbors():

    def __init__(self, K):
        self.K = K

    def fit(self, x_train, y_train):
        self.X_train = x_train
        self.Y_train = y_train

    def predict(self, X_test):

        # list to store all our predictions
        predictions = []

        # loop over all observations
        for i in range(len(X_test)):

            # calculate the distance between the test point and all other points in the
            dist = np.array([euc_dist(X_test[i], x_t) for x_t in self.X_train])

            # sort the distances and return the indices of K neighbors
            dist_sorted = dist.argsort()[:self.K]

            # get the neighbors
            neigh_count = {}

            # for each neighbor find the class
            for idx in dist_sorted:
                if self.Y_train[idx] in neigh_count:
                    neigh_count[self.Y_train[idx]] += 1
```

```
                else:
                    neigh_count[self.Y_train[idx]] = 1

            # get the most common class label
            sorted_neigh_count = sorted(neigh_count.items(), key=operator.itemgetter(1)

            predictions.append(sorted_neigh_count[0][0])
        return predictions


import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification

feat, output = make_classification(n_samples=100, n_features=2, n_classes=2, n_redundan
X_train, X_test, y_train, y_test = train_test_split(feat, output, test_size=0.2)

X_train, X_test, y_train, y_test = train_test_split(feat, output, test_size=0.2, random

clf = KNearestNeighbors(K=3)
clf.fit(X_train, y_train)

predictions = clf.predict(X_test)

print('Accuracy:', accuracy_score(y_test, predictions))
```

```
Accuracy: 1.0
```

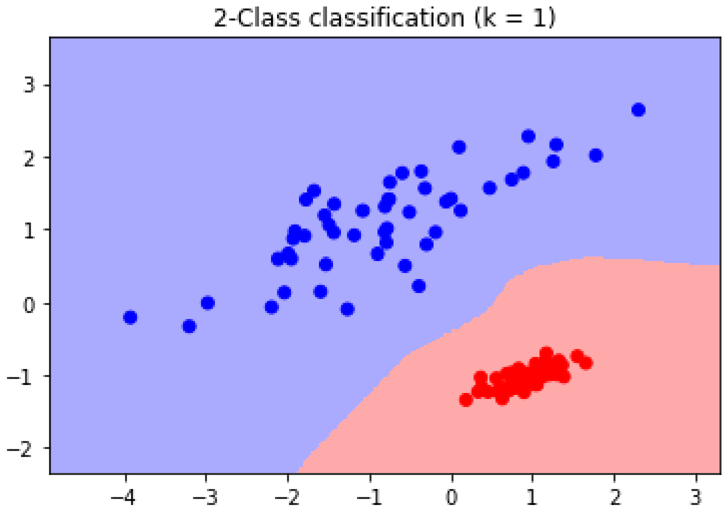
4)Draw the decision boundaries when k equals 1, 10 and 100.

In [80]:

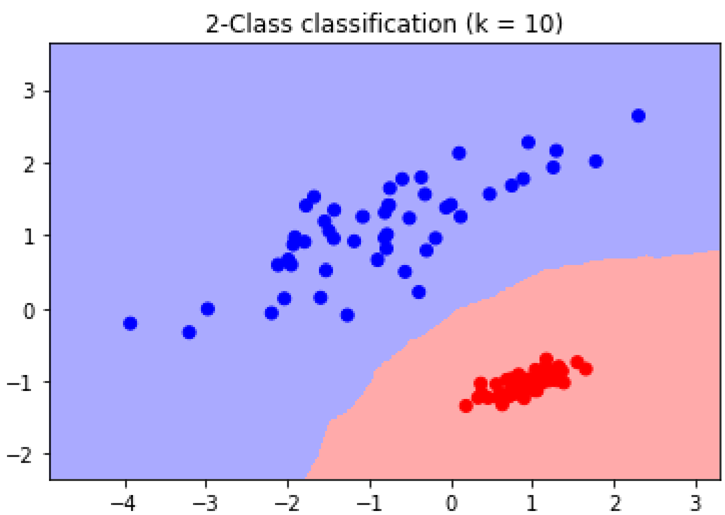
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors
x=[1,10,100]
for n_neighbors in x:


    h = .02  # step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

    for weights in ['distance']:
        # we create an instance of Neighbours Classifier and fit the data.
        clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
        clf.fit(feat, output)

        # Plot the decision boundary. For that, we will assign a color to each
        # point in the mesh [x_min, x_max]x[y_min, y_max].
        x_min, x_max = feat[:, 0].min() - 1, feat[:, 0].max() + 1
        y_min, y_max = feat[:, 1].min() - 1, feat[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                             np.arange(y_min, y_max, h))
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        plt.figure()
        plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

        # Plot also the training points
        plt.scatter(feat[:, 0], feat[:, 1], c=output, cmap=cmap_bold)
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.title("2-Class classification (k = %i)"
                  % (n_neighbors))

    plt.show()
```

```
<ipython-input-80-bddb6c9f0e3d>:31: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

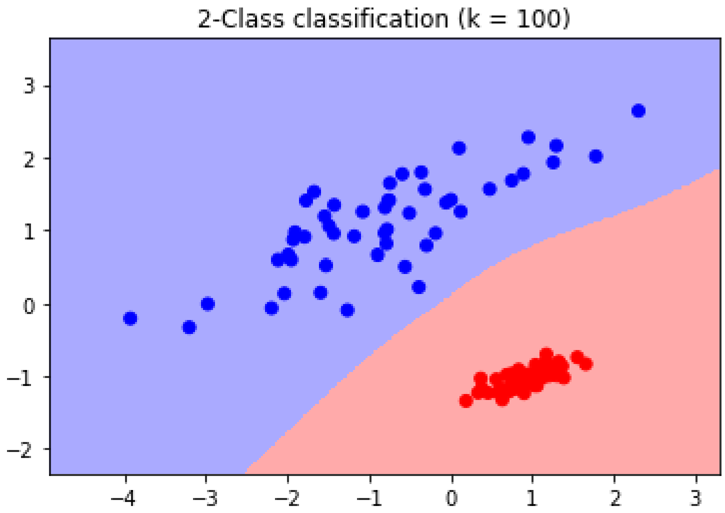



```
<ipython-input-80-bddb6c9f0e3d>:31: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or set rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```




```
<ipython-input-80-bddb6c9f0e3d>:31: MatplotlibDeprecationWarning: shading='flat' when X and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or s
```

```
et rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```



5) Provide the labels for the 10 unlabeled points for each of these settings in a pandas dataframe with 10 rows and three columns (corresponding to each value of k) and display it.

In [50]:

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
classifier1 = KNeighborsClassifier(n_neighbors=1)
classifier2 = KNeighborsClassifier(n_neighbors=5)
classifier3 = KNeighborsClassifier(n_neighbors=10)

import pandas as pd

data,target = make_classification(n_samples=100, n_classes=2)


X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random
x, y = make_classification(n_samples=12,  n_classes=2)


x_train,x_test,Y_train, Y_test= train_test_split(x, y, test_size=.9, random_state=5656)


classifier1.fit(X_train, y_train)
classifier2.fit(X_train, y_train)
classifier3.fit(X_train, y_train)


y_pred1 = classifier1.predict(x_test)
y_pred2 = classifier2.predict(x_test)
y_pred3 = classifier3.predict(x_test)

df1= pd.DataFrame(y_pred1,columns=['K=1'])
df1['k=5']=y_pred2
df1['k=10']=y_pred3
print(df1)
```

```
     K=1  k=5  k=10
0      0    1     0
1      1    1     1
2      0    0     0
```

```
3     1    1     1
4     0    0     0
5     0    1     0
6     0    0     0
7     1    1     1
8     0    0     0
9     1    0     0
10    0    0     0
```

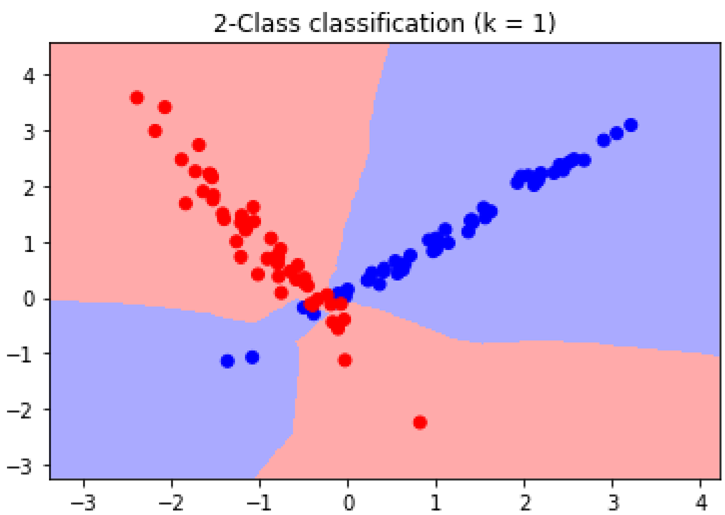
6) Instantiate the k nearest neighbor from scikit-learned compare the decision boundaries and labels obtained from your method.

In [51]:

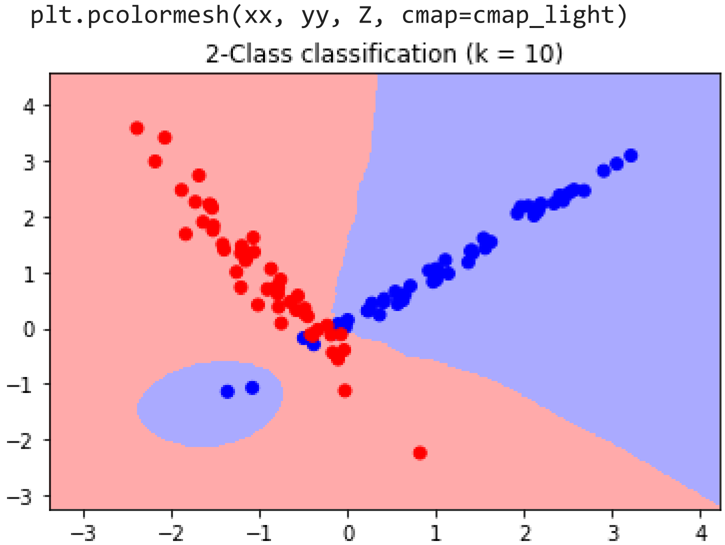
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)

data,target = make_classification(n_samples=100, n_features=2, n_classes=2, n_redundant
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2)

X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random

classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

print('Accuracy:', accuracy_score(y_test, y_pred))


import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors
x=[1,10,100]
for n_neighbors in x:


    h = .02  # step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

    for weights in ['distance']:
        # we create an instance of Neighbours Classifier and fit the data.
        clf = neighbors.KNeighborsClassifier(n_neighbors, weights=weights)
        clf.fit(data, target)

        # Plot the decision boundary. For that, we will assign a color to each
        # point in the mesh [x_min, x_max]x[y_min, y_max].
        x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
        y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                             np.arange(y_min, y_max, h))
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        plt.figure()
        plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

```
        # Plot also the training points
        plt.scatter(data[:, 0], data[:,1], c=target, cmap=cmap_bold)
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.title("2-Class classification (k = %i)"
                  % (n_neighbors))

    plt.show()
```

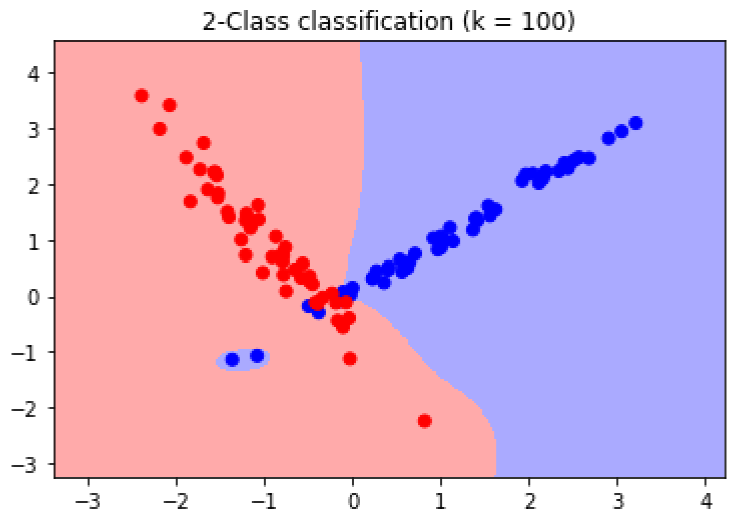
```
Accuracy: 0.95
<ipython-input-51-abea4b075226>:48: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners
of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or s
et rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```




```
<ipython-input-51-abea4b075226>:48: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners
of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or s
et rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```




```
<ipython-input-51-abea4b075226>:48: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners
of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or s
et rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```



7) Repeat steps 1,2,3 for a random two dimensional regression dataset based on Q2. Repeat step 5 where you compute and display the estimated numeric output for each of the three settings (k equals 1, 10 and 100). Repeat step 6 with the equivalent implementation from scikit-learn and compare the predictions with those obtained by your method.

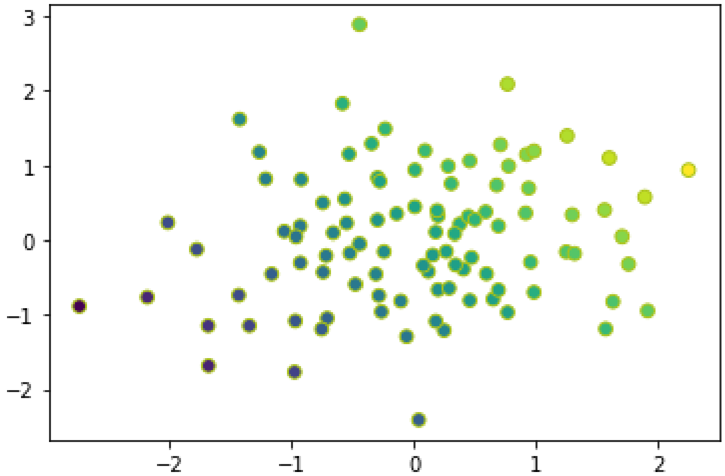
STEP 1

In [52]:

```
from sklearn.datasets import make_regression
import random
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


feat, output = make_regression(n_samples=100, n_features=2)

dataframe = pd.DataFrame(feat, columns=('Column1', 'Column2'))

dataframe1 = pd.DataFrame(output, columns=['OUTPUT'])

dataframe2 = dataframe.join(dataframe1)

print(dataframe2)
r = random.random()
b = random.random()
g = random.random()
color = (r, g, b)

plt.scatter(feat[:, 0], feat[:, 1], marker="o", c=output, s=45, edgecolor=color)

plt.show()
```

```
     Column1   Column2      OUTPUT
0   1.246998 -0.155669   95.591683
1   1.889121  0.574435  187.406258
2  -0.295130  0.832910   19.220979
3   0.914898  0.358718   94.993005
4   1.295258  0.336020  125.453240
..       ...       ...         ...
95  0.180666 -1.086734  -42.088811
96  0.306760  0.753562   65.138991
97  0.289482 -0.643804   -9.750819
```

```
98 -0.957725  0.045363  -77.316803
99  0.281371  0.988315   75.365823

[100 rows x 3 columns]
```



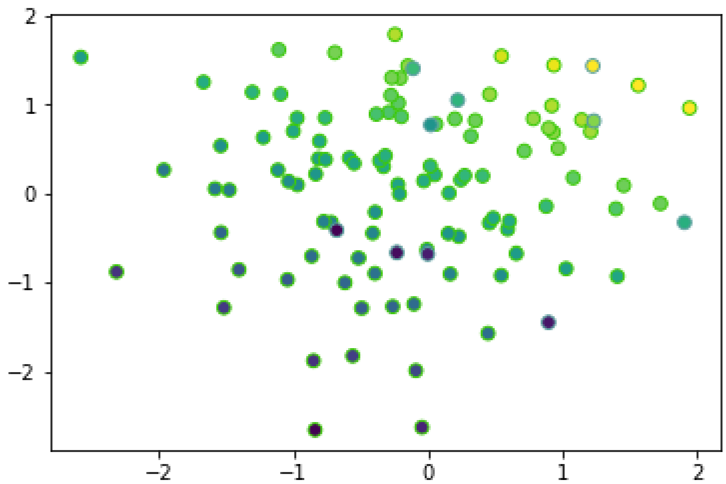
STEP 2

In [53]:

```
from sklearn.datasets import make_regression
import random
import matplotlib.pyplot as plt
import pandas as pd
fig=plt.figure()
ax1=fig.add_subplot(111)


feat, output = make_regression(n_samples=100, n_features=2)

dataframe = pd.DataFrame(feat, columns=('Column1', 'Column2'))

dataframe1 = pd.DataFrame(output, columns=['OUTPUT'])

dataframe2 = dataframe.join(dataframe1)

r = random.random()
b = random.random()
g = random.random()
color = (r, g, b)


a = random.random()
x = random.random()
c = random.random()

colors = (a, x, c)

x, y = make_regression(n_samples=10, n_features=2)
dataframe3 = pd.DataFrame(x, columns=('Column1', 'Column2'))

dataframe4 = pd.DataFrame(y, columns=['OUTPUT'])

dataframe5 = dataframe3.join(dataframe4)
frames=[dataframe2,dataframe5]
result=pd.concat(frames)
print(result)
```

```
plt.scatter(feat[:, 0], feat[:, 1], marker="o", c=output, s=45, edgecolor=color)
plt.scatter(x[:, 0], x[:, 1], marker="o", c=y, s=45, edgecolor=colors)
plt.show()
```

```
     Column1   Column2      OUTPUT
0   1.725636 -0.116611   62.493421
1  -0.584727  0.397368    0.532175
2   0.225209 -0.485760  -20.387671
3  -0.973707  0.846360   12.087232
4  -0.093647 -1.991729 -124.595857
..       ...       ...         ...
5  -0.679259 -0.416529  -76.796028
6   1.222169  1.435271  193.873322
7   0.892832 -1.449191  -61.210995
8   0.016371  0.770014   63.569372
9   1.229808  0.814584  143.949761

[110 rows x 3 columns]
```



STEP 3

In [54]:

```
import operator



def euc_dist(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))


class KNearestNeighbors():

    def __init__(self, K):
        self.K = K

    def fit(self, x_train, y_train):
        self.X_train = x_train
        self.Y_train = y_train

    def predict(self, X_test):

        # list to store all our predictions
        predictions = []

        # loop over all observations
```

```
        for i in range(len(X_test)):

            # calculate the distance between the test point and all other points in the
            dist = np.array([euc_dist(X_test[i], x_t) for x_t in self.X_train])

            # sort the distances and return the indices of K neighbors
            dist_sorted = dist.argsort()[:self.K]

            # get the neighbors
            neigh_count = {}

            # for each neighbor find the class
            for idx in dist_sorted:
                if self.Y_train[idx] in neigh_count:
                    neigh_count[self.Y_train[idx]] += 1
                else:
                    neigh_count[self.Y_train[idx]] = 1

            # get the most common class label
            sorted_neigh_count = sorted(neigh_count.items(), key=operator.itemgetter(1)

            predictions.append(sorted_neigh_count[0][0])
        return predictions


import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_regression

data,target = make_regression(n_samples=100, n_features=2)
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2)

X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random

clf = KNearestNeighbors(K=3)
clf.fit(X_train, y_train)

predictions = clf.predict(X_test)

print('prdicted value for the test data are')
print(predictions)
```

```
prdicted value for the test data are
[65.3245220173632, 322.1931191355148, -51.874107884330634, 111.2657881550978, -51.874107
884330634, 146.05692746920892, 9.101964199697008, -98.28089132831109, -51.87410788433063
4, 6.472872306946655, 115.00475617764577, -81.79795089917862, -137.99804308844801, -98.2
8089132831109, 209.39486141683676, -122.53013485997297, 142.84635025390028, -3.842938724
453589, 111.2657881550978, -22.9704117239922]
```

STEP 5

In [55]:

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression
from sklearn.neighbors import KNeighborsRegressor
reg1 = KNeighborsRegressor(n_neighbors=1)
reg2 = KNeighborsRegressor(n_neighbors=5)
reg3 = KNeighborsRegressor(n_neighbors=10)
```

```
import pandas as pd

data,target = make_regression(n_samples=100, n_features=2)


X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random
x, y = make_regression(n_samples=12,  n_features=2)


x_train,x_test,Y_train, Y_test= train_test_split(x, y, test_size=.9, random_state=5656)


reg1.fit(X_train, y_train)
reg2.fit(X_train, y_train)
reg3.fit(X_train, y_train)


y_pred1 = reg1.predict(x_test)
y_pred2 = reg2.predict(x_test)
y_pred3 = reg3.predict(x_test)

df1= pd.DataFrame(y_pred1,columns=['K=1'])
df1['k=5']=y_pred2
df1['k=10']=y_pred3
print('labels for 10 point with differnt k values')
print(df1)
```

```
labels for 10 point with differnt k values
           K=1        k=5       k=10
0    56.555386  40.657752  24.255420
1    47.260617  53.817123  38.810631
2   -18.500012 -14.355914 -22.458731
3   -57.166981 -36.270509 -37.185144
4   -57.166981 -64.935413 -48.977929
5    43.797490  24.396822  15.445826
6   -67.916800 -53.380454 -57.655584
7    54.111143  64.259750  81.244437
8  -102.128945 -59.578110 -43.400204
9   -32.400165 -22.473659  -7.326858
10  -67.916800 -77.435923 -69.098099
```

STEP 6

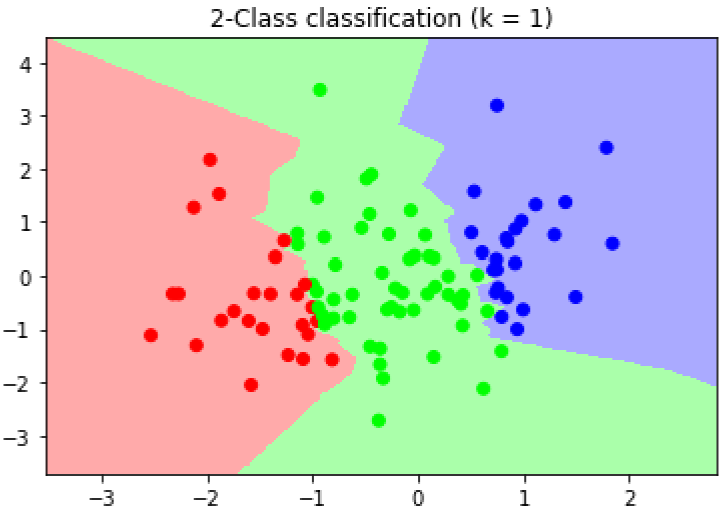
In [56]:

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression
from sklearn.neighbors import KNeighborsRegressor
reg = KNeighborsRegressor(n_neighbors=5)

data,target = make_regression(n_samples=100, n_features=2)
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2)

X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random

reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
print('prediction for the test data')
print(y_pred)

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

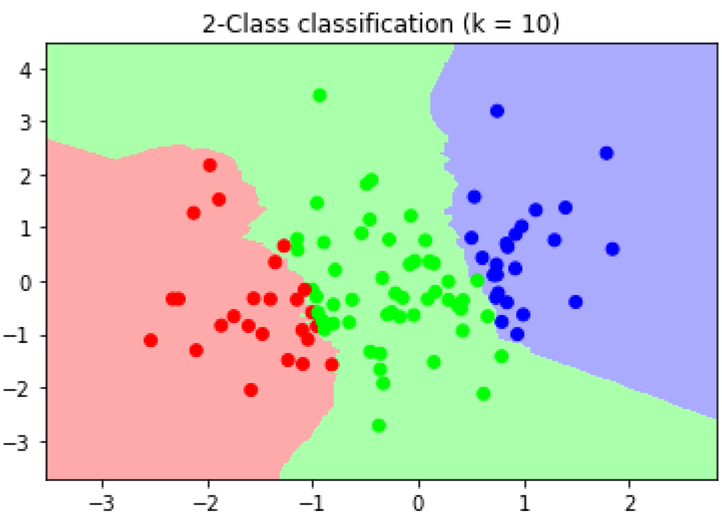
```python
from matplotlib.colors import ListedColormap
from sklearn import neighbors
x=[1,10,100]
for n_neighbors in x:


    h = .02  # step size in the mesh

    # Create color maps
    cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

    for weights in ['distance']:
        # we create an instance of Neighbours Classifier and fit the data.
        clf = neighbors.KNeighborsRegressor(n_neighbors, weights=weights)
        clf.fit(data, target)

        # Plot the decision boundary. For that, we will assign a color to each
        # point in the mesh [x_min, x_max]x[y_min, y_max].
        x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
        y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                             np.arange(y_min, y_max, h))
        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

        # Put the result into a color plot
        Z = Z.reshape(xx.shape)
        plt.figure()
        plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

        # Plot also the training points
        plt.scatter(data[:, 0], data[:, 1], c=target, cmap=cmap_bold)
        plt.xlim(xx.min(), xx.max())
        plt.ylim(yy.min(), yy.max())
        plt.title("2-Class classification (k = %i)"
                  % (n_neighbors))

    plt.show()
```

```
prediction for the test data
[   5.46119906 -115.32709345  -41.03562306  -67.86232368  -64.83504888
  -22.59597752   18.13585307   20.09185403   57.36894003  -42.69983752
  -66.25379893   42.19350082  -51.04699059  -74.79951329   73.19797591
   16.5934079   -69.56295932   -3.00605399   78.38251861  -22.59597752]
```

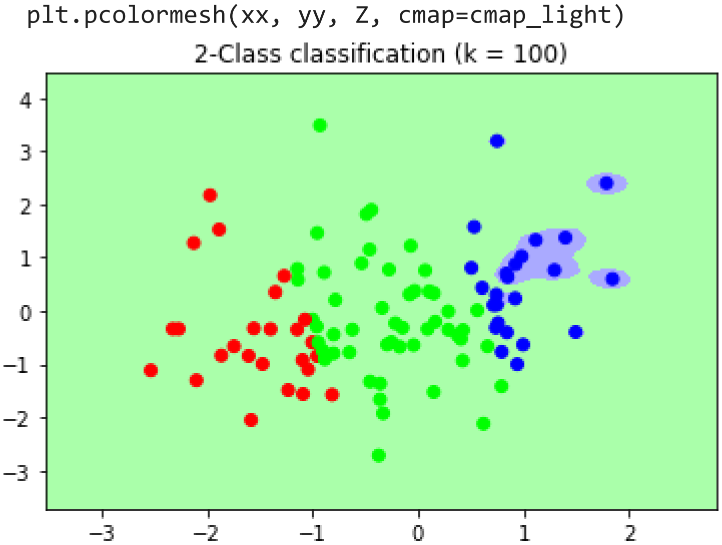
```
<ipython-input-56-90b7adc59075>:46: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners
of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or s
et rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```




```
<ipython-input-56-90b7adc59075>:46: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners
of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or s
et rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```




```
<ipython-input-56-90b7adc59075>:46: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corners
of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud', or s
et rcParams['pcolor.shading'].  This will become an error two minor releases later.
  plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```



IMPLEMENT DECISION TREES

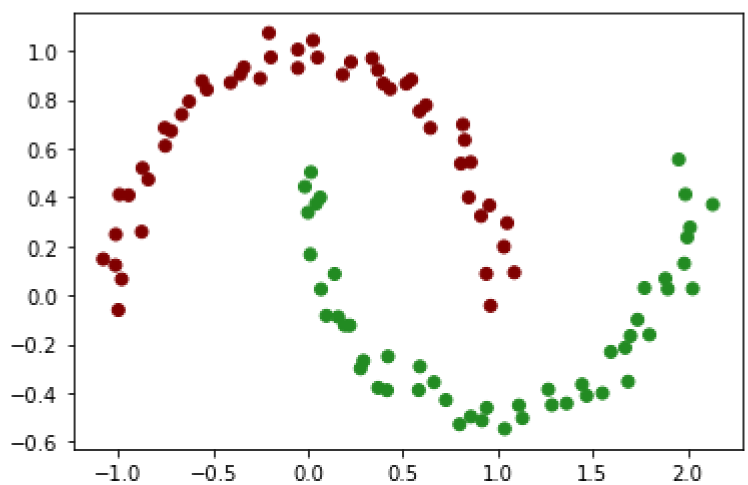
1) Create three two dimensional datasets (with two classes and 100 rows) using 1, 2 and 3. Plot them using matplotlib.

In [57]:
```
import numpy as np
import sklearn
from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from pandas import DataFrame
from mlxtend.plotting import plot_decision_regions
from sklearn import model_selection
import random
from math import log, e
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
```

In [58]:
```
feature1, target1 = datasets.make_moons(n_samples=100, noise=0.05, random_state=42)
feature2, target2= datasets.make_circles(n_samples=100, shuffle=True, noise=None, rando
feature3, target3 = datasets.make_blobs(n_samples=100, centers=None, random_state=None)
```

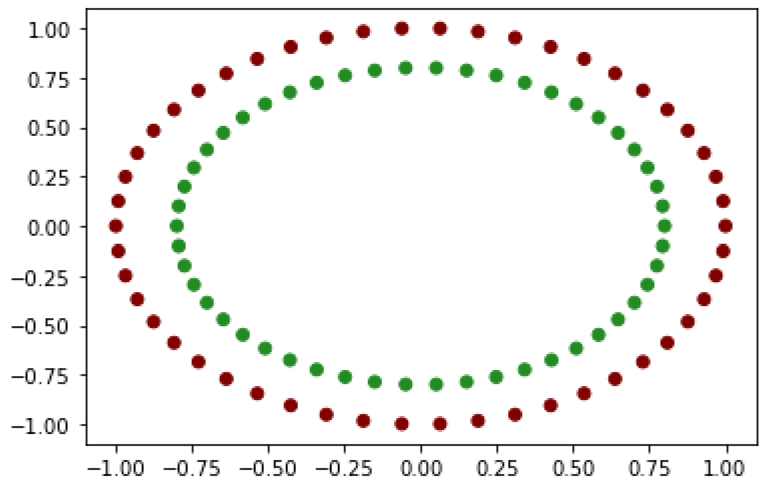
In [59]:
```
#Scatter Plot of a Moon dataset
colors = ['maroon', 'forestgreen']
vectorizer = np.vectorize(lambda x: colors[x % len(colors)])
plt.scatter(feature1[:,0], feature1[:,1], c=vectorizer(target1))
```

Out[59]: <matplotlib.collections.PathCollection at 0x1ee80243220>



In [60]:
```
#Scatter Plot of a Circle dataset
colors = ['maroon', 'forestgreen']
vectorizer = np.vectorize(lambda x: colors[x % len(colors)])
plt.scatter(feature2[:,0], feature2[:,1], c=vectorizer(target2))
```

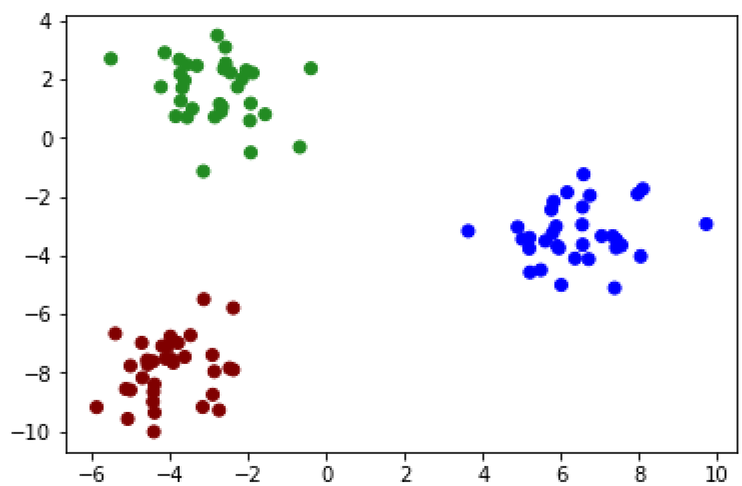
Out[60]: <matplotlib.collections.PathCollection at 0x1eefaeb9dc0>



In [61]:

```
#Scatter Plot of a Blob dataset
colors = ['maroon', 'forestgreen','blue']
vectorizer = np.vectorize(lambda x: colors[x % len(colors)])
plt.scatter(feature3[:,0], feature3[:,1], c=vectorizer(target3))
```

Out[61]: <matplotlib.collections.PathCollection at 0x1ee816c54c0>



Write the decision tree model class that uses training accuracy to make splits from scratch. The class should be able to handle other splitting criteria, namely entropy and Gini index as well.

In [62]:

```
# Converting to dataframe
df_moon = DataFrame(dict(x1=feature1[:,0], x2=feature1[:,1], label=target1))
df_circle = DataFrame(dict(x1=feature2[:,0], x2=feature2[:,1], label=target2))
df_blobs =  DataFrame(dict(x1=feature3[:,0], x2=feature3[:,1], label=target3))
```

In [63]:

```
# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

In [64]:

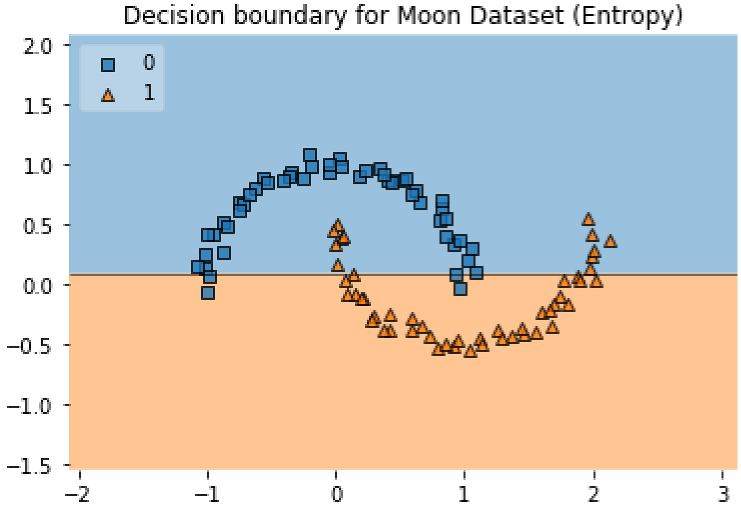
```
#Code for make moon dataset
# Function to split the dataset
def splitdataset(data):

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split( feature1, target1, test_size =
    return feature1, target1, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to perform training with entropy.
def train_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy

# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

#Function to calculate confusion matrix
def Confusion_Matrix(counts):
    classes = int(max(y_test) - min(y_test)) + 1 #find number of classes
    counts = [[sum([(y_test[i] == true_class) and (y_pred[i] == pred_class)
                for i in range(len(y_test))])
           for pred_class in range(1, classes + 1)]
           for true_class in range(1, classes + 1)]
    print("CM is" , (counts))
    return counts


# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
```

```python
    classification_report(y_test, y_pred))


def entropy4(label, base=None):
    value,counts = np.unique(label, return_counts=True)
    norm_counts = counts / counts.sum()
    base = e if base is None else base
    entropyx = -(norm_counts * np.log(norm_counts)/np.log(base)).sum()
    print( "The entropy is : ", (entropyx))
    return entropyx


# Driver code
def main():

    # Building Phase
    print('Results for Moon Dataset')
    data = df_moon
    feature1,target1,X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = train_using_entropy(X_train, X_test, y_train)

    # Operational Phase
    print("Results Using Gini Index of moon dataset:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    print("Results Using Entropy of Moon dataset:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)
    en = clf_entropy.fit(X_train,y_train)
    fig1 = plot_decision_regions(X=feature1, y=target1, clf=en, legend=2)
    plt.title('Decision boundary for Moon Dataset (Entropy)')
    plt.show()

    cl = clf_gini.fit(X_train, y_train)
    fig = plot_decision_regions(X=feature1, y=target1, clf=cl, legend=2)
    plt.title('Decision boundary for Moon Dataset (Gini)')
    plt.show()
    y_pred = cl.predict(X_test)
    y_pred_train = cl.predict(X_train)
    print("Training error of Moon Dataset",100 - (accuracy_score(y_train,y_pred_train)*

    mse = np.mean(y_test != y_pred)
    print("The MSE of Moon dataset is",mse )
    entropy4(target1)

main()
```

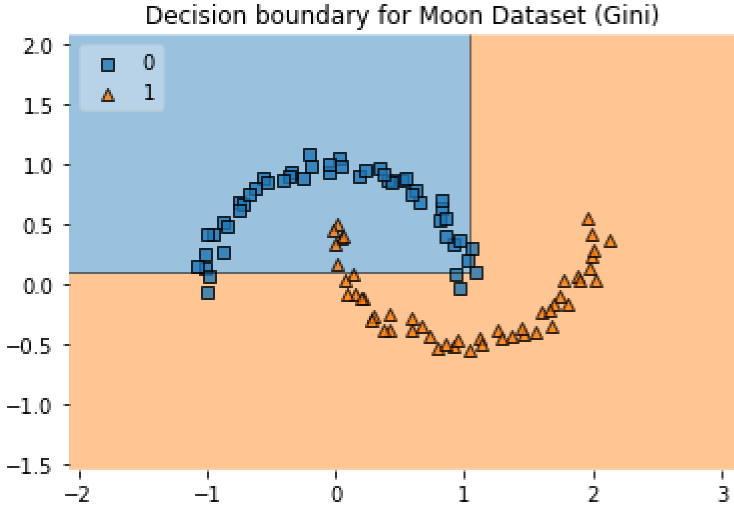
```
Results for Moon Dataset
Results Using Gini Index of moon dataset:
Predicted values:
[1 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 1 0 1 0 1 0 0 1 0 0 0 1 0 0]
Confusion Matrix:  [[15  1]
 [ 3 11]]
Accuracy :  86.66666666666667
Report :               precision    recall  f1-score   support
```

```
           0       0.83      0.94      0.88        16
           1       0.92      0.79      0.85        14

    accuracy                           0.87        30
   macro avg       0.88      0.86      0.86        30
weighted avg       0.87      0.87      0.87        30

Results Using Entropy of Moon dataset:
Predicted values:
[1 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0]
Confusion Matrix:  [[15  1]
 [ 6  8]]
Accuracy :  76.66666666666667
Report :               precision    recall  f1-score   support

           0       0.71      0.94      0.81        16
           1       0.89      0.57      0.70        14

    accuracy                           0.77        30
   macro avg       0.80      0.75      0.75        30
weighted avg       0.80      0.77      0.76        30
```

Decision boundary for Moon Dataset (Entropy)



Decision boundary for Moon Dataset (Gini)




```
Training error of Moon Dataset 10.0
The MSE of Moon dataset is 0.13333333333333333
The entropy is :  0.6931471805599453
```

In [66]:

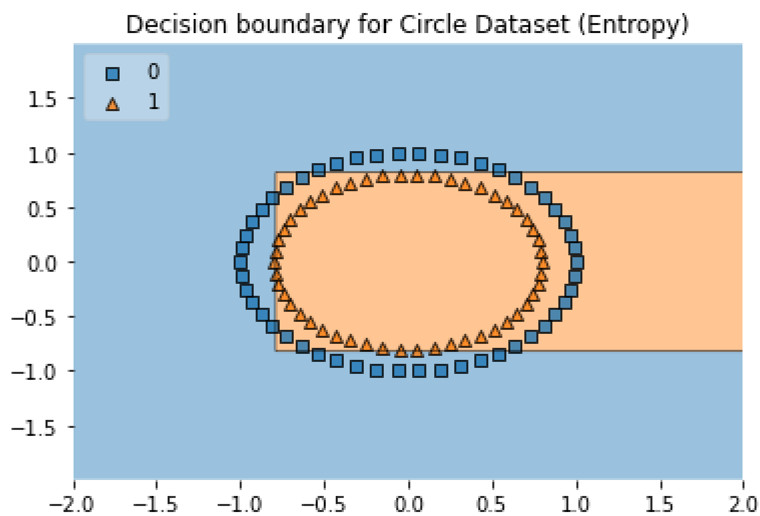
```
# Function to split the dataset
```

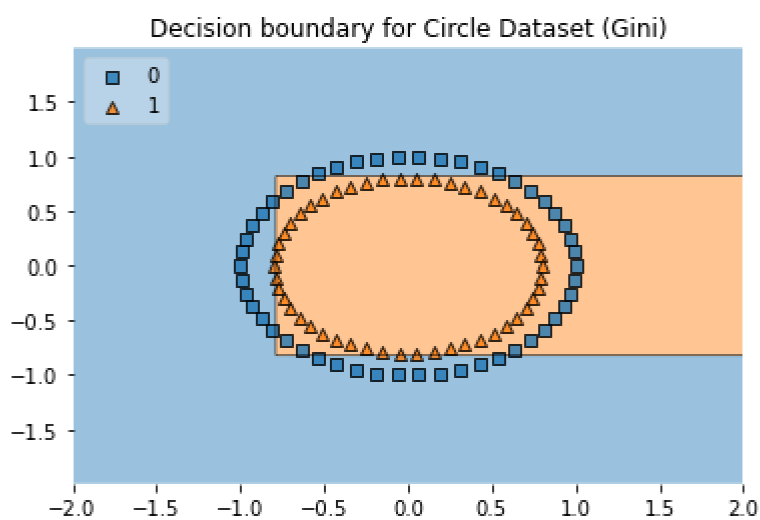
```
def splitdataset(data):

    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split( feature2, target2, test_size =
    return feature2, target2, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to perform training with entropy.
def train_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy

# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

#Function to calculate confusion matrix
def Confusion_Matrix(counts):
    classes = int(max(y_test) - min(y_test)) + 1 #find number of classes
    counts = [[sum([(y_test[i] == true_class) and (y_pred[i] == pred_class)
                for i in range(len(y_test))])
           for pred_class in range(1, classes + 1)]
           for true_class in range(1, classes + 1)]
    print("CM is" , (counts))
    return counts


# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
    classification_report(y_test, y_pred))
```

```python
def entropy4(label, base=None):
    value,counts = np.unique(label, return_counts=True)
    norm_counts = counts / counts.sum()
    base = e if base is None else base
    entropyx = -(norm_counts * np.log(norm_counts)/np.log(base)).sum()
    print( "The entropy is : ", (entropyx))
    return entropyx


# Driver code
def main():

    # Building Phase
    print('Results for Circle Dataset')
    data = df_moon
    feature2,target2,X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = train_using_entropy(X_train, X_test, y_train)

    # Operational Phase
    print("Results Using Gini Index of Circle dataset:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    print("Results Using Entropy of Circle dataset:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)
    en = clf_entropy.fit(X_train,y_train)
    fig2 = plot_decision_regions(X=feature2, y=target2, clf=en, legend=2)
    plt.title('Decision boundary for Circle Dataset (Entropy)')
    plt.show()


    cl = clf_gini.fit(X_train, y_train)
    fig = plot_decision_regions(X=feature2, y=target2, clf=cl, legend=2)
    plt.title('Decision boundary for Circle Dataset (Gini)')
    plt.show()
    y_pred = cl.predict(X_test)
    y_pred_train = cl.predict(X_train)
    print("Training error of Circle Dataset",100 - (accuracy_score(y_train,y_pred_train

    mse = np.mean(y_test != y_pred)
    print("The MSE of Circle dataset is",mse )
    entropy4(target2)

main()
```

```
Results for Circle Dataset
Results Using Gini Index of Circle dataset:
Predicted values:
[1 1 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1]
Confusion Matrix:  [[ 5  5]
 [ 3 17]]
Accuracy :  73.33333333333333
Report :                precision    recall  f1-score   support
```

```
           0       0.62      0.50      0.56        10
           1       0.77      0.85      0.81        20

    accuracy                           0.73        30
   macro avg       0.70      0.68      0.68        30
weighted avg       0.72      0.73      0.72        30

Results Using Entropy of Circle dataset:
Predicted values:
[1 1 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1]
Confusion Matrix:  [[ 5  5]
 [ 3 17]]
Accuracy :  73.33333333333333
Report :                precision    recall  f1-score   support

           0       0.62      0.50      0.56        10
           1       0.77      0.85      0.81        20

    accuracy                           0.73        30
   macro avg       0.70      0.68      0.68        30
weighted avg       0.72      0.73      0.72        30
```






```
Training error of Circle Dataset 20.0
The MSE of Circle dataset is 0.26666666666666666
The entropy is :  0.6931471805599453
```

In [67]:

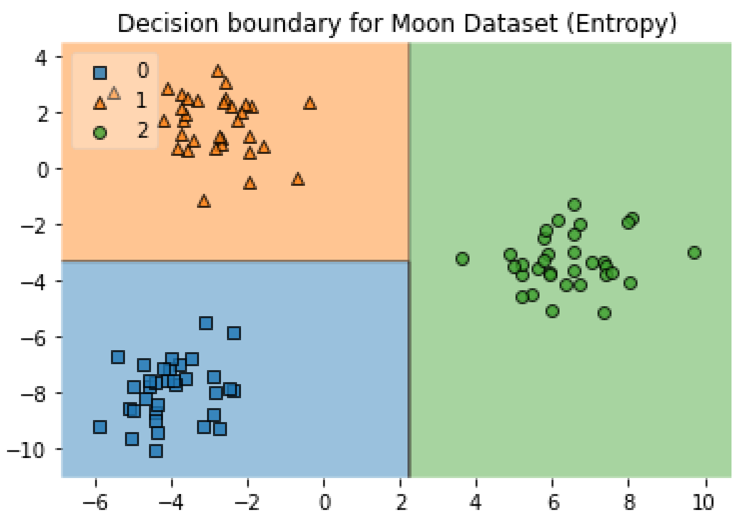
```
# Function to split the dataset
def splitdataset(data):
```

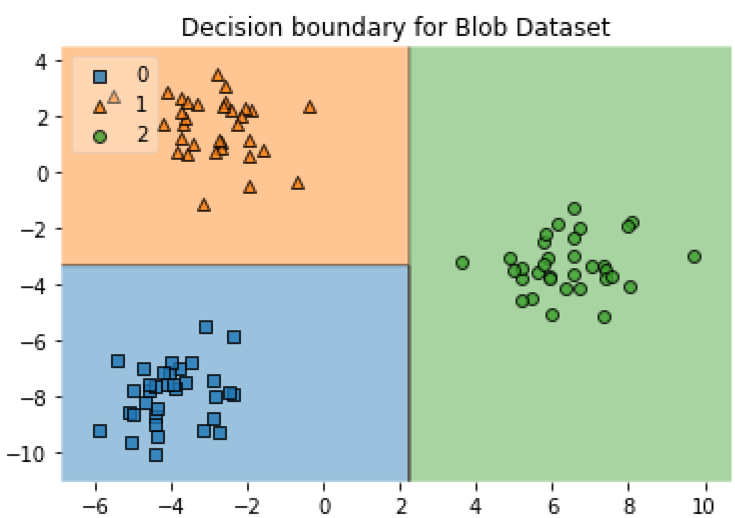
```
    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split( feature3, target3, test_size =
    return feature3, target3, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):
    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)

    # Performing training
    clf_gini.fit(X_train, y_train)
    return clf_gini

# Function to perform training with entropy.
def train_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy

# Function to make predictions
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred

#Function to calculate confusion matrix
def Confusion_Matrix(counts):
    classes = int(max(y_test) - min(y_test)) + 1 #find number of classes
    counts = [[sum([(y_test[i] == true_class) and (y_pred[i] == pred_class)
                for i in range(len(y_test))])
           for pred_class in range(1, classes + 1)]
           for true_class in range(1, classes + 1)]
    print("CM is" , (counts))
    return counts


# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))

    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)

    print("Report : ",
    classification_report(y_test, y_pred))
```

```
def entropy4(label, base=None):
    value,counts = np.unique(label, return_counts=True)
    norm_counts = counts / counts.sum()
    base = e if base is None else base
    entropyx = -(norm_counts * np.log(norm_counts)/np.log(base)).sum()
    print( "The entropy is : ", (entropyx))
    return entropyx


# Driver code
def main():

    # Building Phase
    print('Results for Blob Dataset')
    data = df_moon
    feature3,target3,X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = train_using_entropy(X_train, X_test, y_train)

    # Operational Phase
    print("Results Using Gini Index of Blob dataset:")

    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)

    print("Results Using Entropy of Blob dataset:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)
    en = clf_entropy.fit(X_train,y_train)
    fig = plot_decision_regions(X=feature3, y=target3, clf=en, legend=2)
    plt.title('Decision boundary for Moon Dataset (Entropy)')
    plt.show()


    cl = clf_gini.fit(X_train, y_train)
    fig = plot_decision_regions(X=feature3, y=target3, clf=cl, legend=2)
    plt.title('Decision boundary for Blob Dataset')
    plt.show()
    y_pred = cl.predict(X_test)
    y_pred_train = cl.predict(X_train)
    print("Training error of Blob Dataset",100 - (accuracy_score(y_train,y_pred_train)*

    mse = np.mean(y_test != y_pred)
    print("The MSE of Blob dataset is",mse)
    entropy4(target3)

main()
```

```
Results for Blob Dataset
Results Using Gini Index of Blob dataset:
Predicted values:
[1 0 1 2 0 2 2 0 2 1 2 1 1 0 1 0 1 1 0 2 2 0 2 0 1 0 0 1 0 1]
Confusion Matrix:  [[11  0  0]
 [ 0 11  0]
 [ 0  0  8]]
Accuracy :  100.0
Report :                precision    recall  f1-score   support
```

```
           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00         8

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

Results Using Entropy of Blob dataset:
Predicted values:
[1 0 1 2 0 2 2 0 2 1 2 1 1 0 1 0 1 1 0 2 2 0 2 0 1 0 0 1 0 1]
Confusion Matrix:  [[11  0  0]
 [ 0 11  0]
 [ 0  0  8]]
Accuracy :  100.0
Report :                precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00         8

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






```
Training error of Blob Dataset 0.0
The MSE of Blob dataset is 0.0
The entropy is :  1.0985126170507196
```

4)How does the splitting criteria influence the decision boundaries?

Answer) From the above observation, we concluded that the splitting criteria didn't influence the decision boundary of blob and circle dataset. Whereas gini criteria performed well in moon dataset.

---

5) Comment on whether zero training error is achieved for each of the three datasets.

Answer) As mentioned in above code, training error for three datasets are:

Make_moon dataset - 10% training error

Make_circle dataset - 17% training error

Make_Blob dataset - 0% training error. That means the model is overfitted.

REGRESSOR TREE

In [68]:

```
#Import necessarily Libraries
import pandas as pd
import numpy as np
from collections import Counter
from sklearn import datasets
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
from sklearn import model_selection
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.metrics import r2_score,mean_squared_error
from pandas import DataFrame
from mlxtend.plotting import plot_decision_regions
from sklearn import model_selection
import random
from math import log, e
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
```

In [69]:

```
#Class to grow a regression decision tree
class NodeReg():
    def __init__(
        self,
        Y: list,
        X: pd.DataFrame,
        min_samples_split=None,
        max_depth=None,
        depth=None,
        node_type=None,
        rule=None
    ):
        # Saving the data to the node
        self.Y = Y
        self.X = X

        # Saving the hyper parameters
```

```
        self.min_samples_split = min_samples_split if min_samples_split else 20
        self.max_depth = max_depth if max_depth else 5

        # Default current depth of node
        self.depth = depth if depth else 0

        # Extracting all the features
        self.features = list(self.X.columns)

        # Type of node
        self.node_type = node_type if node_type else 'root'

        # Rule for spliting
        self.rule = rule if rule else ""

        # Getting the mean of Y
        self.ymean = np.mean(Y)

        # Getting the residuals
        self.residuals = self.Y - self.ymean

        # Calculating the mse of the node
        self.mse = self.get_mse(Y, self.ymean)

        # Saving the number of observations in the node
        self.n = len(Y)

        # Initiating the left and right nodes as empty nodes
        self.left = None
        self.right = None

        # Default values for splits
        self.best_feature = None
        self.best_value = None

#Method to calculate the mean squared error
    @staticmethod
    def get_mse(ytrue, yhat) -> float:
        """

        """
        # Getting the total number of samples
        n = len(ytrue)

        # Getting the residuals
        r = ytrue - yhat

        # Squering the residuals
        r = r ** 2

        # Suming
        r = np.sum(r)

        # Getting the average and returning
        return r / n

    #Calculates the moving average of the given list.
    @staticmethod
    def ma(x: np.array, window: int) -> np.array:
        return np.convolve(x, np.ones(window), 'valid') / window
```

```python
    #Given the X features and Y targets calculates the best split for a decision tree
    def best_split(self) -> tuple:
        # Creating a dataset for spliting
        df = self.X.copy()
        df['Y'] = self.Y

        # Getting the GINI impurity for the base input
        mse_base = self.mse

        # Finding which split yields the best GINI gain
        #max_gain = 0

        # Default best feature and split
        best_feature = None
        best_value = None

        for feature in self.features:
            # Droping missing values
            Xdf = df.dropna().sort_values(feature)

            # Sorting the values and getting the rolling average
            xmeans = self.ma(Xdf[feature].unique(), 2)

            for value in xmeans:
                # Getting the left and right ys
                left_y = Xdf[Xdf[feature]<value]['Y'].values
                right_y = Xdf[Xdf[feature]>=value]['Y'].values

                # Getting the means
                left_mean = np.mean(left_y)
                right_mean = np.mean(right_y)

                # Gettinghe left and right residuals
                res_left = left_y - left_mean
                res_right = right_y - right_mean

                # Concatenating the residuals
                r = np.concatenate((res_left, res_right), axis=None)

                # Calculating the mse
                n = len(r)
                r = r ** 2
                r = np.sum(r)
                mse_split = r / n

                # Checking if this is the best split so far
                if mse_split < mse_base:
                    best_feature = feature
                    best_value = value

                    # Setting the best gain to the current one
                    mse_base = mse_split

        return (best_feature, best_value)

    #Recursive method to create the decision tree
    def grow_tree(self):
        # Making a df from the data
        df = self.X.copy()
```

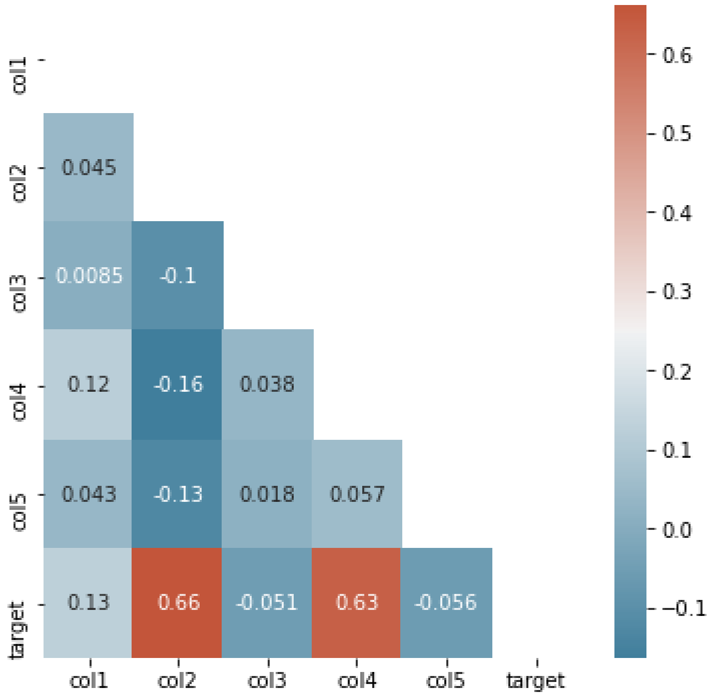
```python
        df['Y'] = self.Y

        # If there is GINI to be gained, we split further
        if (self.depth < self.max_depth) and (self.n >= self.min_samples_split):

            # Getting the best split
            best_feature, best_value = self.best_split()

            if best_feature is not None:
                # Saving the best split to the current node
                self.best_feature = best_feature
                self.best_value = best_value

                # Getting the left and right nodes
                left_df, right_df = df[df[best_feature]<=best_value].copy(), df[df[best

                # Creating the left and right nodes
                left = NodeReg(
                    left_df['Y'].values.tolist(),
                    left_df[self.features],
                    depth=self.depth + 1,
                    max_depth=self.max_depth,
                    min_samples_split=self.min_samples_split,
                    node_type='left_node',
                    rule=f"{best_feature} <= {round(best_value, 3)}"
                    )

                self.left = left
                self.left.grow_tree()

                right = NodeReg(
                    right_df['Y'].values.tolist(),
                    right_df[self.features],
                    depth=self.depth + 1,
                    max_depth=self.max_depth,
                    min_samples_split=self.min_samples_split,
                    node_type='right_node',
                    rule=f"{best_feature} > {round(best_value, 3)}"
                    )

                self.right = right
                self.right.grow_tree()

    #Method to print the infromation about the tree
    def print_info(self, width=4):
        # Defining the number of spaces
        const = int(self.depth * width ** 1.5)
        spaces = "-" * const

        if self.node_type == 'root':
            print("Root")
        else:
            print(f"|{spaces} Split rule: {self.rule}")
        print(f"{' ' * const}   | MSE of the node: {round(self.mse, 2)}")
        print(f"{' ' * const}   | Count of observations in node: {self.n}")
        print(f"{' ' * const}   | Prediction of node: {round(self.ymean, 3)}")

    # Prints the whole tree from the current node to the bottom
    def print_tree(self):
        self.print_info()
```

```
        if self.left is not None:
            self.left.print_tree()

        if self.right is not None:
            self.right.print_tree()
```

In [70]:

```
#Creating a Regression Dataset

from sklearn import datasets
import random
import matplotlib.pyplot as plt
X,Y = datasets.make_regression(n_samples=100, n_features=5, n_informative=2, random_sta
random.seed(100)

# Create Pandas Dataframe and processes correlation

df = pd.DataFrame(X,Y)
df.columns = ['col1', 'col2', 'col3', 'col4', 'col5']
df['target'] = Y
X = df.iloc[:,df.columns!='target']
Y = df.iloc[:,-1]


# Determine correlations

corr = df.corr()

# Draw the correlation heatmap

f, ax = plt.subplots(figsize=(6, 6))
mask = np.triu(np.ones_like(corr, dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr, annot=True, mask = mask, cmap=cmap)
```

Out[70]: <AxesSubplot:>



In [71]:

```
# Initiating the Node
root = NodeReg(Y, X, max_depth=2, min_samples_split=3)
# Growing the tree
root.grow_tree()
# Printing tree
root.print_tree()
```

```
Root
   | MSE of the node: 5414.76
   | Count of observations in node: 100
   | Prediction of node: 1.502
|-------- Split rule: col2 <= -0.262
           | MSE of the node: 3646.84
           | Count of observations in node: 40
           | Prediction of node: -53.207
|---------------- Split rule: col4 <= 0.213
                   | MSE of the node: 1457.65
                   | Count of observations in node: 22
                   | Prediction of node: -95.94
|---------------- Split rule: col4 > 0.213
                   | MSE of the node: 1362.75
                   | Count of observations in node: 18
                   | Prediction of node: -0.978
|-------- Split rule: col2 > -0.262
           | MSE of the node: 3267.72
           | Count of observations in node: 60
           | Prediction of node: 37.975
|---------------- Split rule: col4 <= -0.305
                   | MSE of the node: 1700.32
                   | Count of observations in node: 22
                   | Prediction of node: -13.104
|---------------- Split rule: col4 > -0.305
                   | MSE of the node: 1790.13
                   | Count of observations in node: 38
                   | Prediction of node: 67.547
```

In [72]:

```
#Create Decision Tree Regression Model
```

```python
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(X, Y)
```

Out[72]: DecisionTreeRegressor(random_state=0)

In [73]:

```python
#The accuracy Score for the model
regressor.score(X,Y)
```

Out[73]: 1.0

In [ ]:

References

1. https://stackoverflow.com/
2. https://www.tutorialspoint.com/index.htm
3. https://www.geeksforgeeks.org/decision-tree/