MACHINE LEARNING ASSIGNMENT

1. Keerthan Devineni
2. Anand Sabbineni
3. Shivani Erigineni
4. Srinivas Gundluri

```
#NUMPY A

import numpy as np
a=[1,1],[1,1]
twodimensional_array= np.array(a)
twodimensionalones_array=np.ones((2,2),dtype=np.int64)
print(twodimensional_array)
print(twodimensionalones_array)


[[1 1]
 [1 1]]
[[1 1]
 [1 1]]

#NUMPY B

b=[1,0,0],[0,1,0],[0,0,1]
threedimensionalidentity_array= np.matrix(b)
print(threedimensionalidentity_array)


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

#NUMPY C

a = np.array([[1,3,5,9], [6,6,8,8], [12,11,11,12]])
print(a[0, 2:])
a.shape
np.shape(a)

#NUMPY D
print(a>10)

#NUMPY E
print("Memory size of a NumPy array before:",
      a.nbytes)
```

```
a = a.astype('int64')
print(a.dtype)
print("Memory size of a NumPy array after:",
      a.nbytes)
```

```
[5 9]
[[False False False False]
 [False False False False]
 [ True  True  True  True]]
Memory size of a NumPy array before: 96
int64
Memory size of a NumPy array after: 96
```

```
#PANDAS A

import pandas as pd
np_array = np.array([10,20,30,30,30,10,20,100,100, np.nan])
np_array=np_array.astype('int64')
print("NumPy array:")
print(np_array)
new_series = pd.Series(np_array)
print("Converted Pandas series:")
print(new_series)
```

```
NumPy array:
[                  10                   20                   30
                   30                   30                   10
                   20                  100                  100
 -9223372036854775808]
Converted Pandas series:
0                     10
1                     20
2                     30
3                     30
4                     30
5                     10
6                     20
7                    100
8                    100
9   -9223372036854775808
dtype: int64
```

```
#PANDAS B

unilist = {"University":
['UIC','NEU','NIU','ASU','UTD','UMASS','UTA','IOWA','UOM','UOD'] ,
"Rankings":[1,3,5,2,7,6,9,4,10,8] }
df=pd.DataFrame(unilist,

index=pd.Index(['Row1','Row2','Row3','Row4','Row5','Row6','Row7','Row8
','Row9','Row10'],name='series'),columns=
```

```
pd.Index(['University','Rankings'],name='checklist'))
print(df)
```

```
checklist University  Rankings
series
Row1              UIC         1
Row2              NEU         3
Row3              NIU         5
Row4              ASU         2
Row5              UTD         7
Row6            UMASS         6
Row7              UTA         9
Row8             IOWA         4
Row9              UOM        10
Row10             UOD         8
```

```
type(df.loc['Row1'])
```

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

```
type(df.iloc[0:1,0:2])
```

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

```
#PANDAS C

df.dtypes
```

```
checklist
University    object
Rankings       int64
dtype: object
```

```
df.head()
```

```
checklist University  Rankings
series
Row1              UIC         1
Row2              NEU         3
Row3              NIU         5
Row4              ASU         2
Row5              UTD         7
```

```
df.tail()
```

```
checklist University  Rankings
series
Row6            UMASS         6
Row7              UTA         9
Row8             IOWA         4
```

```
Row9              UOM        10
Row10             UOD         8
```

```python
#PANDAS D
#describe is used to generate various summary statstics , excluding
NaN values.

df.describe()
```

```
checklist  Rankings
count      10.00000
mean        5.50000
std         3.02765
min         1.00000
25%         3.25000
50%         5.50000
75%         7.75000
max        10.00000
```

```python
#from google.colab import files
#uploaded_files = files.upload()

#PANDAS E

readcsvfile = pd.read_csv("Iris.csv")
df=pd.DataFrame(readcsvfile)
print(df)
```

```
      Id  SepalLengthCm  ...  PetalWidthCm         Species
0      1            5.1  ...           0.2     Iris-setosa
1      2            4.9  ...           0.2     Iris-setosa
2      3            4.7  ...           0.2     Iris-setosa
3      4            4.6  ...           0.2     Iris-setosa
4      5            5.0  ...           0.2     Iris-setosa
..   ...            ...  ...           ...             ...
145  146            6.7  ...           2.3  Iris-virginica
146  147            6.3  ...           1.9  Iris-virginica
147  148            6.5  ...           2.0  Iris-virginica
148  149            6.2  ...           2.3  Iris-virginica
149  150            5.9  ...           1.8  Iris-virginica

[150 rows x 6 columns]
```

```python
df.dtypes #datatypes of the file
```

```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
```

```
Species           object
dtype: object
```

```
print(df.head()) #sample
df.describe() #describing the data
```

```
   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
Species
0   1            5.1           3.5            1.4           0.2  Iris-
setosa
1   2            4.9           3.0            1.4           0.2  Iris-
setosa
2   3            4.7           3.2            1.3           0.2  Iris-
setosa
3   4            4.6           3.1            1.5           0.2  Iris-
setosa
4   5            5.0           3.6            1.4           0.2  Iris-
setosa
```

```
               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
max    150.000000       7.900000      4.400000       6.900000
2.500000
```

```
df = df.drop(labels="Species", axis=1)
print(df)
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
```

```
148  149              6.2           3.4            5.4           2.3
149  150              5.9           3.0            5.1           1.8

[150 rows x 5 columns]
```

```
#MATLAB A

data = {"YEAR":
['1984','1995','1997','2000','2001','2002','2003','2004','2005','2006'
] , "USA":
[1.16,1.14,1.13,1.11,1.11,1.15,1.23,1.23,1.89,1.17],"CANADA":
[1.87,1.92,1.73,1.57,1.53,1.61,1.62,1.35,1.52,2.15],"TYPE":
['DIESEL','GAS','DIESEL','GAS','GAS','DIESEL','DIESEL','GAS','GAS','DI
ESEL'] }
df1=pd.DataFrame(data,

index=pd.Index(['Row1','Row2','Row3','Row4','Row5','Row6','Row7','Row8
','Row9','Row10']),columns= pd.Index(['YEAR','USA','CANADA','TYPE']))
print(df1)
```

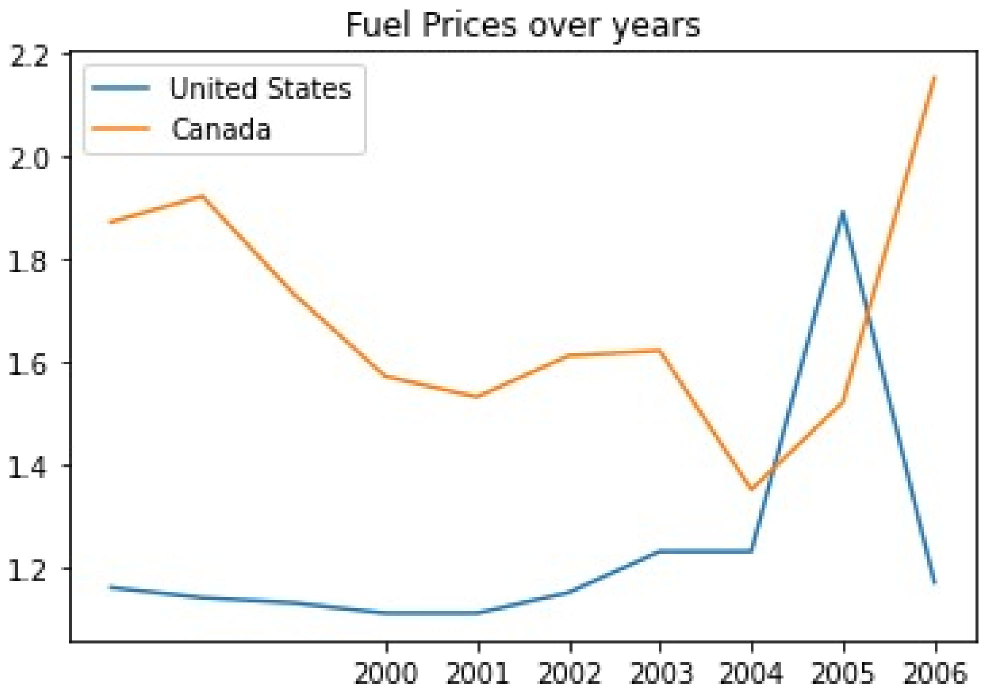
```
       YEAR   USA  CANADA    TYPE
Row1   1984  1.16    1.87  DIESEL
Row2   1995  1.14    1.92     GAS
Row3   1997  1.13    1.73  DIESEL
Row4   2000  1.11    1.57     GAS
Row5   2001  1.11    1.53     GAS
Row6   2002  1.15    1.61  DIESEL
Row7   2003  1.23    1.62  DIESEL
Row8   2004  1.23    1.35     GAS
Row9   2005  1.89    1.52     GAS
Row10  2006  1.17    2.15  DIESEL
```

```
df1 = df1.drop(labels="TYPE", axis=1)
print(df1)
```

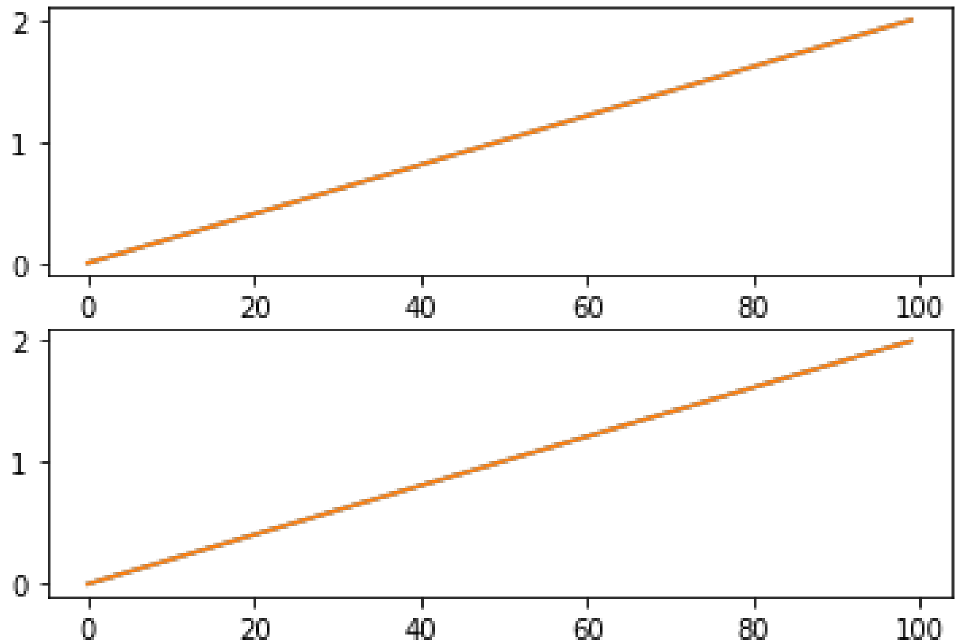
```
       YEAR   USA  CANADA
Row1   1984  1.16    1.87
Row2   1995  1.14    1.92
Row3   1997  1.13    1.73
Row4   2000  1.11    1.57
Row5   2001  1.11    1.53
Row6   2002  1.15    1.61
Row7   2003  1.23    1.62
Row8   2004  1.23    1.35
Row9   2005  1.89    1.52
Row10  2006  1.17    2.15
```

```
import matplotlib.pyplot as plt
plt.plot(df1.YEAR,df1.USA, label ='United States')
plt.plot(df1.YEAR,df1.CANADA, label='Canada')
```

```
#MATLAB B
plt.xticks(df1.YEAR[ 3::])

#MATLAB C
plt.title('Fuel Prices over years')

plt.legend()

plt.show()
```




```
#MATLAB D

x = np.linspace(0, 2, 100)
y = np.linspace(0, 2, 100)

fig, axs = plt.subplots(2)
axs[0].plot(x)
axs[1].plot(x)

axs[0].plot(x)
axs[1].plot(x)

[<matplotlib.lines.Line2D at 0x7f1852ac9cd0>]
```




```
#MATLAB E
val1=['Chicago','NYC','Austin','Seattle']
arr=np.array(val1)
ypos=np.arange(len(arr))
ypos
```

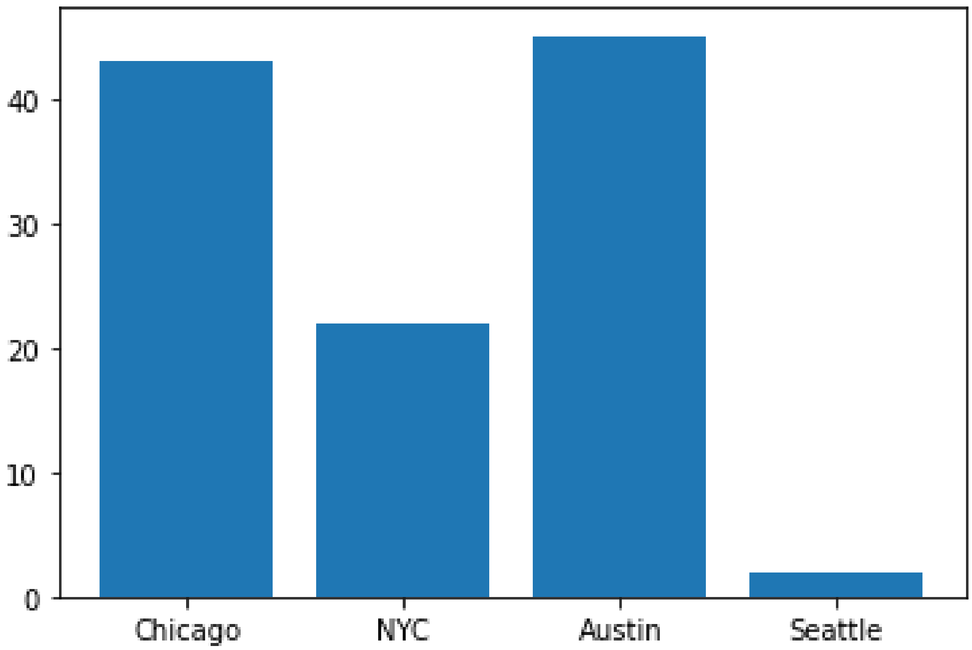
```
array([0, 1, 2, 3])
```

```
val =np.random.randint(50, size=(4))
#arr1=np.array(val)

print(val)
```

```
[43 22 45  2]
```

```
plt.xticks(ypos,val1)
plt.bar(ypos,val)
```

```
<BarContainer object of 4 artists>
```



Question 2:

```
# linear regression feature importance
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_classification
from matplotlib import pyplot
import seaborn as sb
import seaborn as sns
from pandas import DataFrame
```

make_classification

```
dfc = make_classification(n_samples=1000, n_features=2, n_classes=2,
random_state = 7,n_redundant=0,
                                    class_sep=2, n_informative=2)

#This dataset has 1000 samples , 2 columns , 2 classes , 0 redundant
features,
#larger values spread out the clusters/classes, 2 informative
features,
#and is reproducible

#dfc
#converting to pandas df
```

```
dclass = pd.DataFrame(dfc[0], columns=["X","Y"])
dclass.head()
```

```
          X         Y
0  2.476945 -2.689452
1  3.318997  2.257764
2  2.542647 -1.609172
3 -3.116124  3.931041
4  2.225907 -2.121408
```

Statistics of make_classification dataset

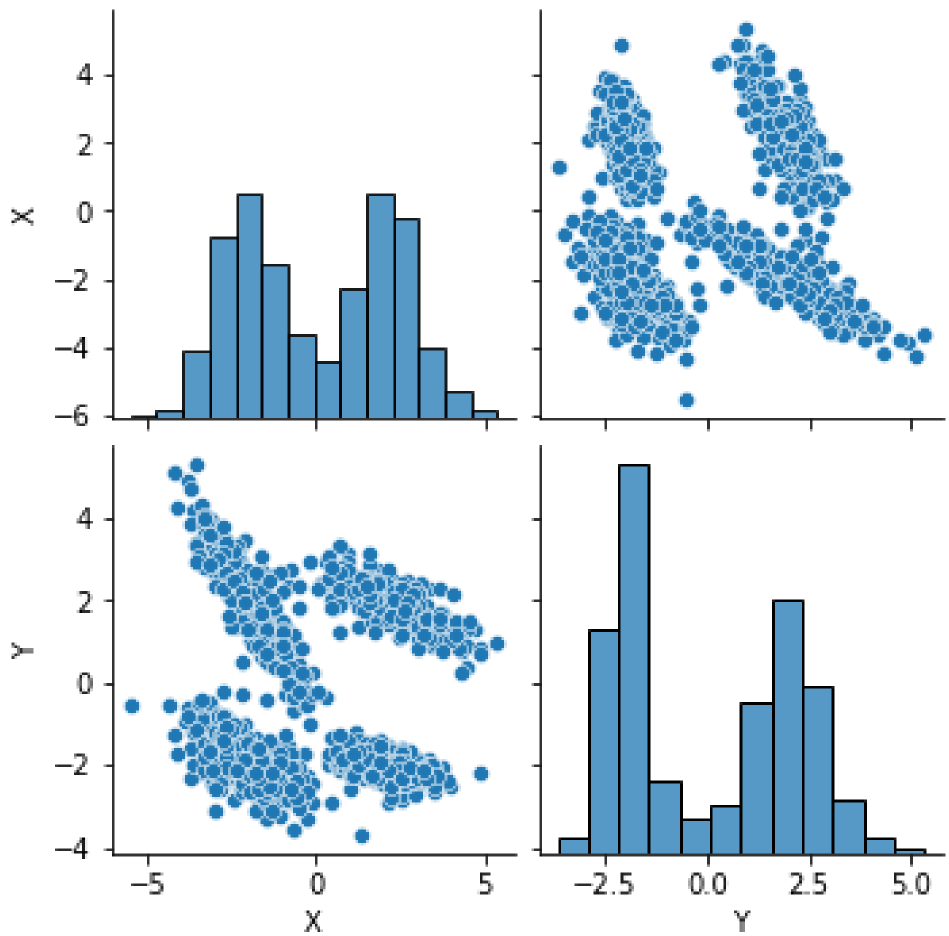
```
dclass[["X", "Y"]].describe()
```

```
                 X            Y
count  1000.000000  1000.000000
mean      0.041483    -0.000783
std       2.271408     2.117396
min      -5.516128    -3.685144
25%      -2.043944    -2.013886
50%      -0.113535    -0.430406
75%       2.136730     2.044094
max       5.373494     5.317234
```

exploratory data analysis

```
#distribution of x and y
sb.pairplot(dclass)
```

```
<seaborn.axisgrid.PairGrid at 0x7f18529f90d0>
```




```
dclass.head()
```

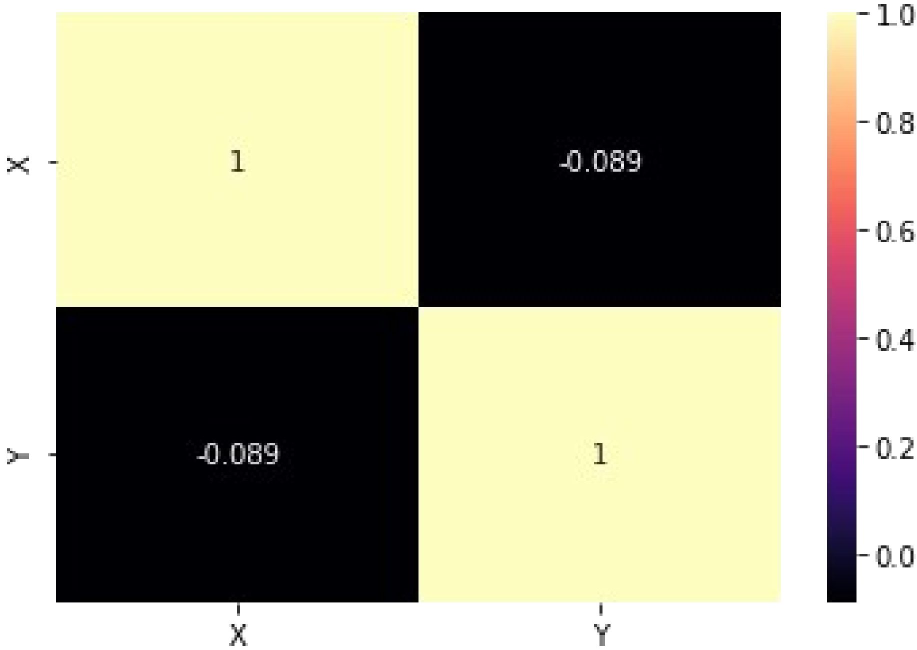
```
          X         Y
0  2.476945 -2.689452
1  3.318997  2.257764
2  2.542647 -1.609172
3 -3.116124  3.931041
4  2.225907 -2.121408
```

```
#dimensions
dclass.shape
```

```
(1000, 2)
```

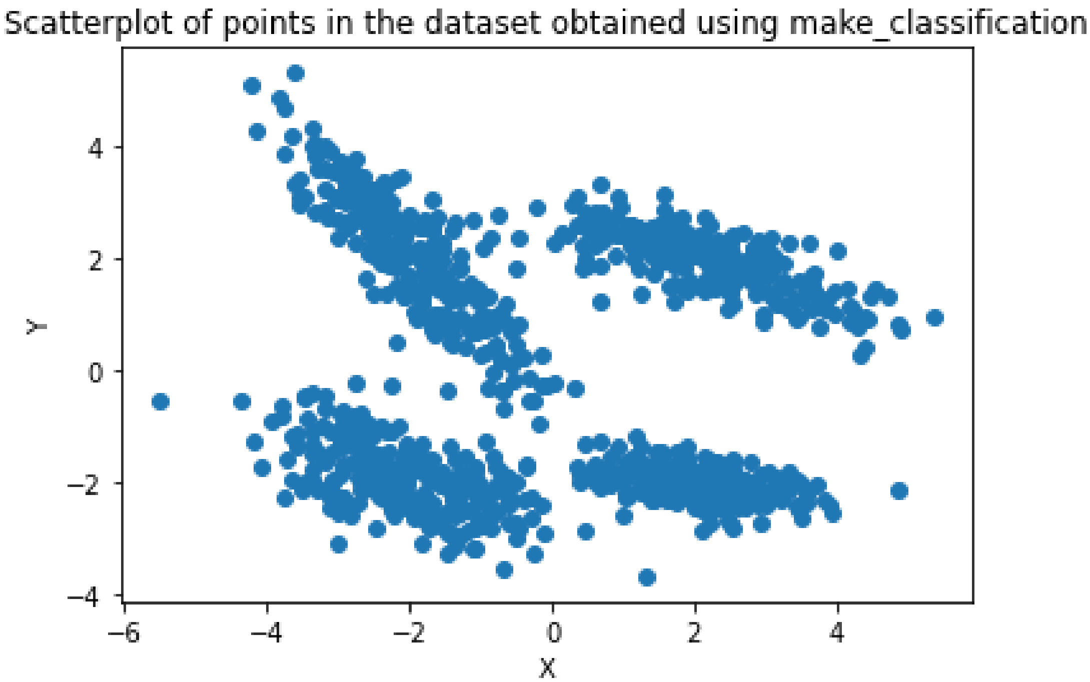
```
#correlation of x and y
sb.heatmap(dclass.corr(),annot=True,cmap='magma')

#Correlation Matrix
corr_matrix1 = dclass.corr()
print(corr_matrix1)
```

```
          X         Y
X  1.00000 -0.08879
Y -0.08879  1.00000
```



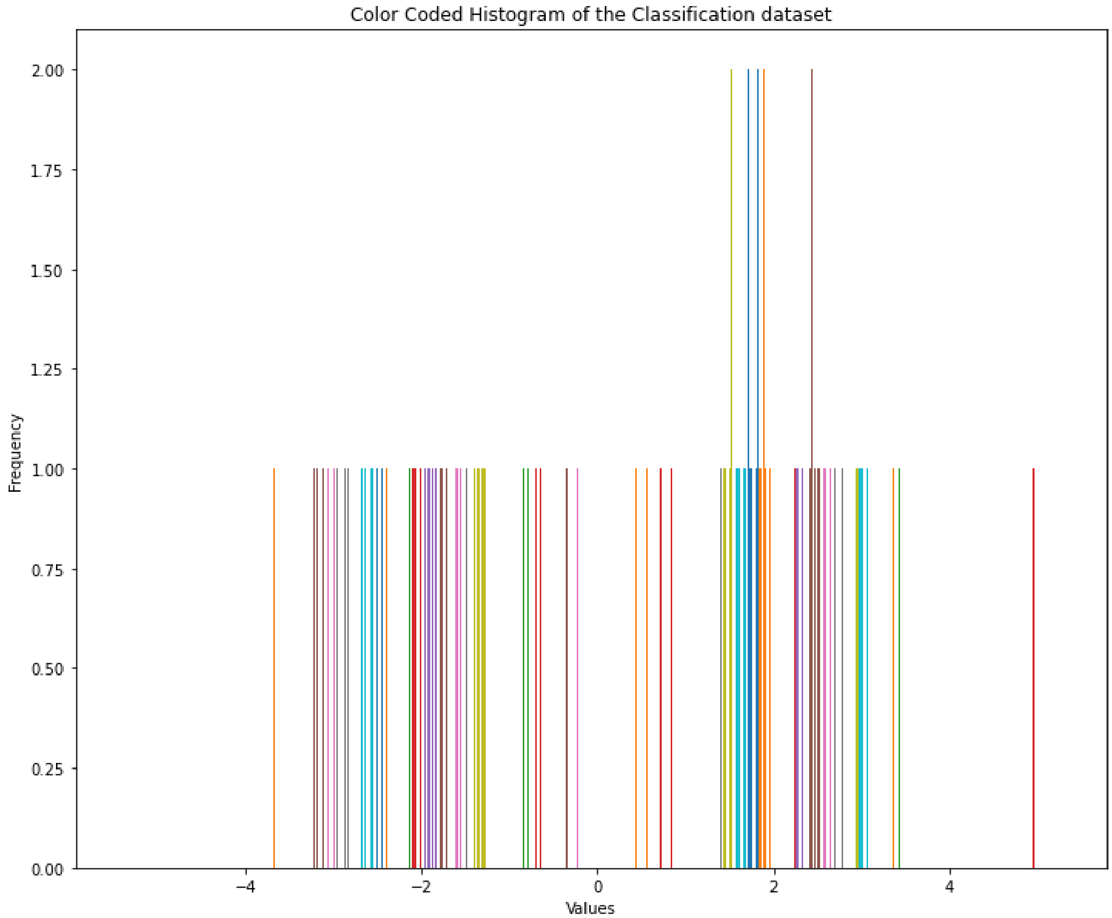

```
plt.scatter(dfc[0][:,0],dfc[0][:,1])
plt.xlabel('X')
plt.ylabel('Y')
plt.title("Scatterplot of points in the dataset obtained using
make_classification")
```

```
Text(0.5, 1.0, 'Scatterplot of points in the dataset obtained using
make_classification')
```



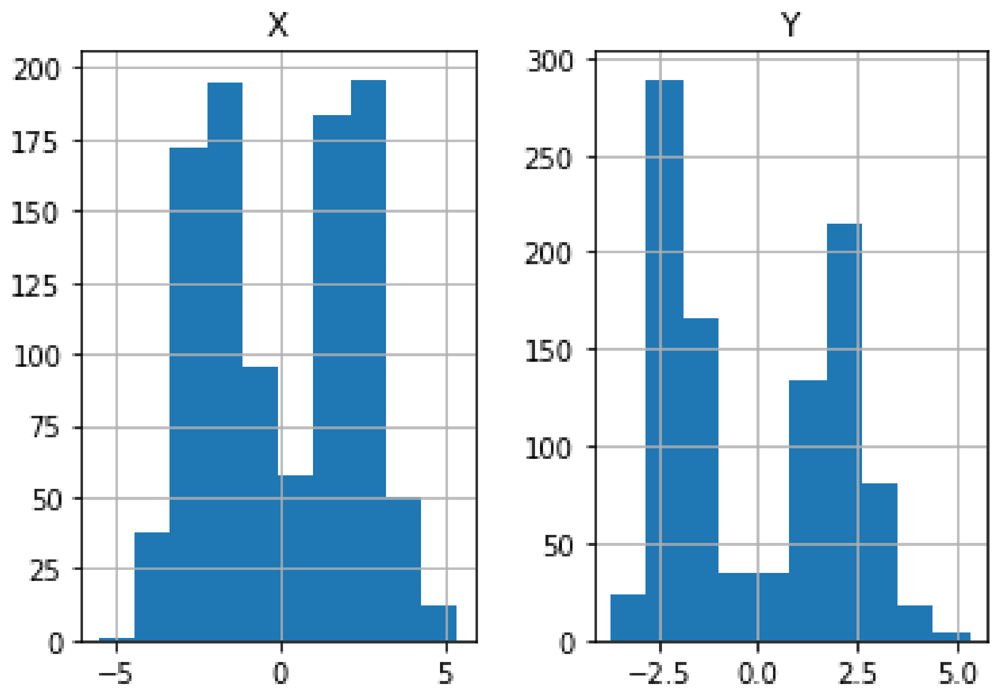

```
plt.figure(figsize=(12,10))
plt.hist(dclass)
plt.xlabel("Values")
plt.ylabel("Frequency")
plt.title("Color Coded Histogram of the Classification dataset")
```

```
Text(0.5, 1.0, 'Color Coded Histogram of the Classification dataset')
```



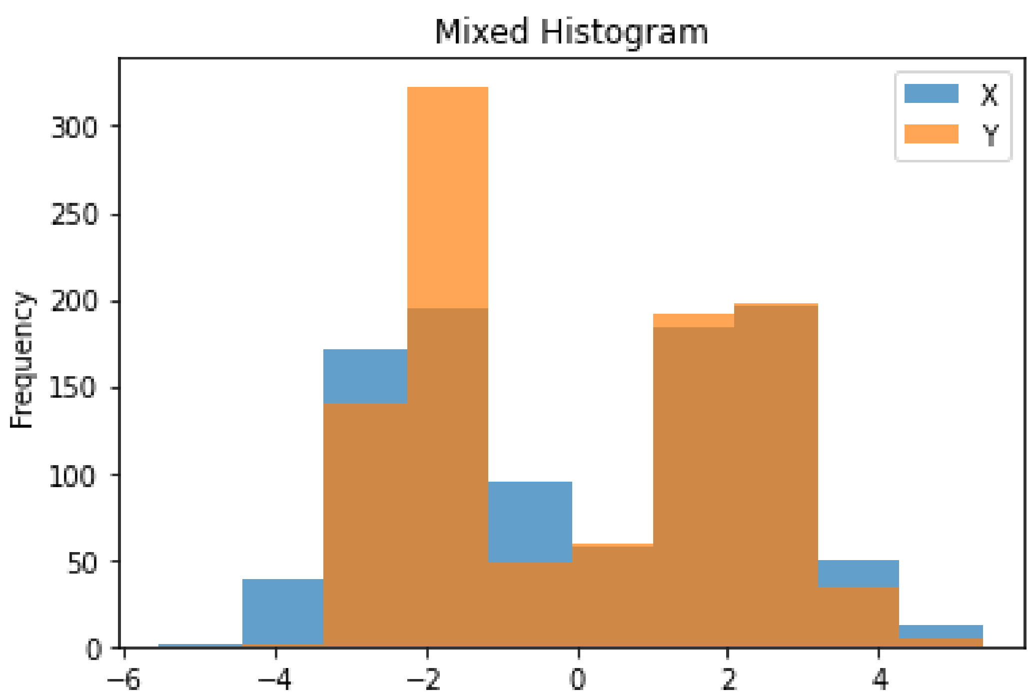

```
dclass.hist()
#displaying a histogram for each class
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at
0x7f1852724210>,
        <matplotlib.axes._subplots.AxesSubplot object at
0x7f184ca0a150>]],
      dtype=object)
```



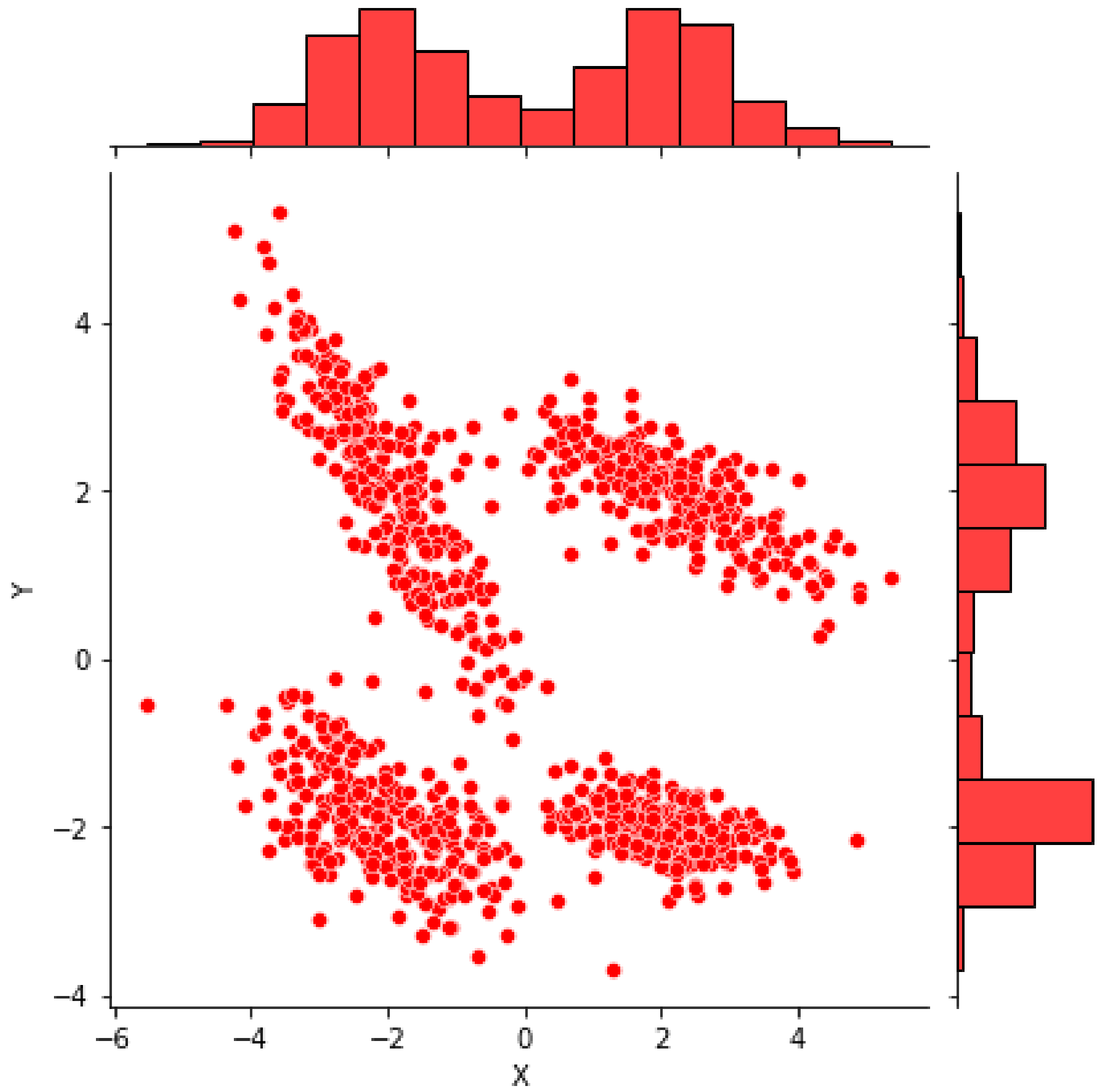

```
dclass[["X", "Y"]].plot.hist(alpha=0.7)
plt.title("Mixed Histogram")
```

```
Text(0.5, 1.0, 'Mixed Histogram')
```



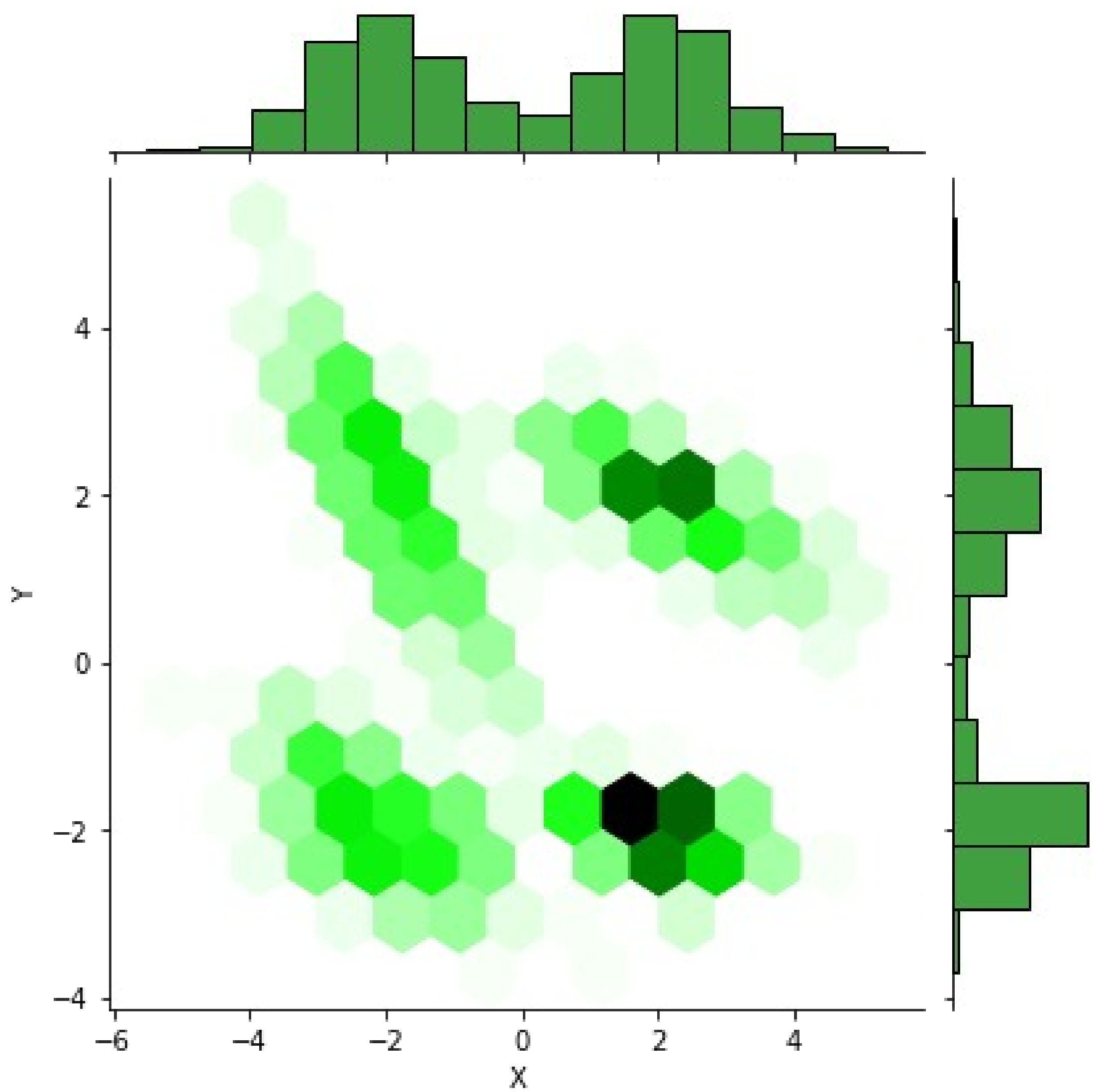
```
sns.jointplot(x=dclass["X"], y=dclass["Y"], kind='scatter', color
="red")
#Marginal distribution through a scatter plot
```

```
<seaborn.axisgrid.JointGrid at 0x7f184c1b7710>
```



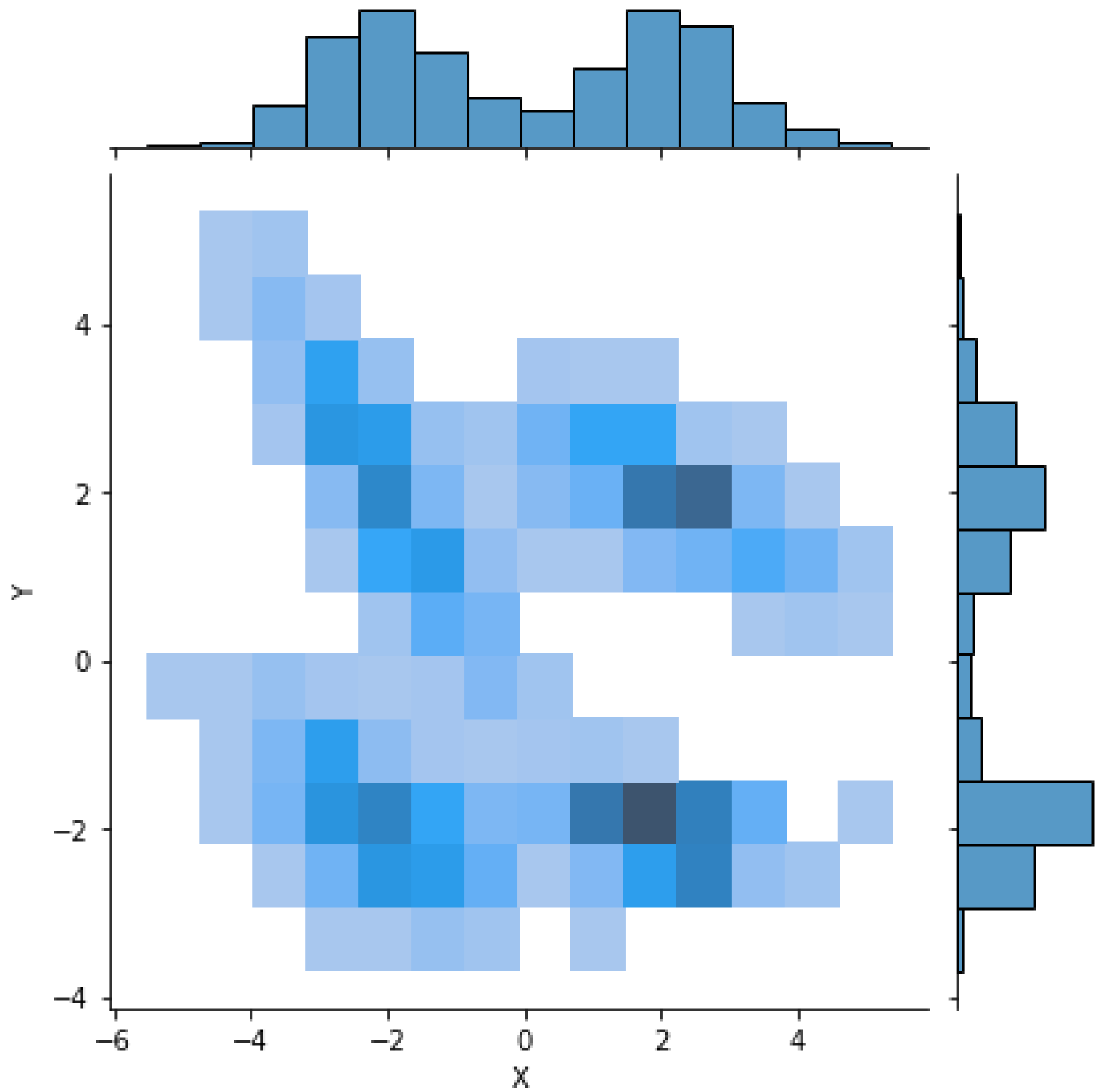

```
sns.jointplot(x=dclass["X"], y=dclass["Y"], kind='hex',color =
"green")
#Marginal distribution through a HEX
```

```
<seaborn.axisgrid.JointGrid at 0x7f184c2b9710>
```

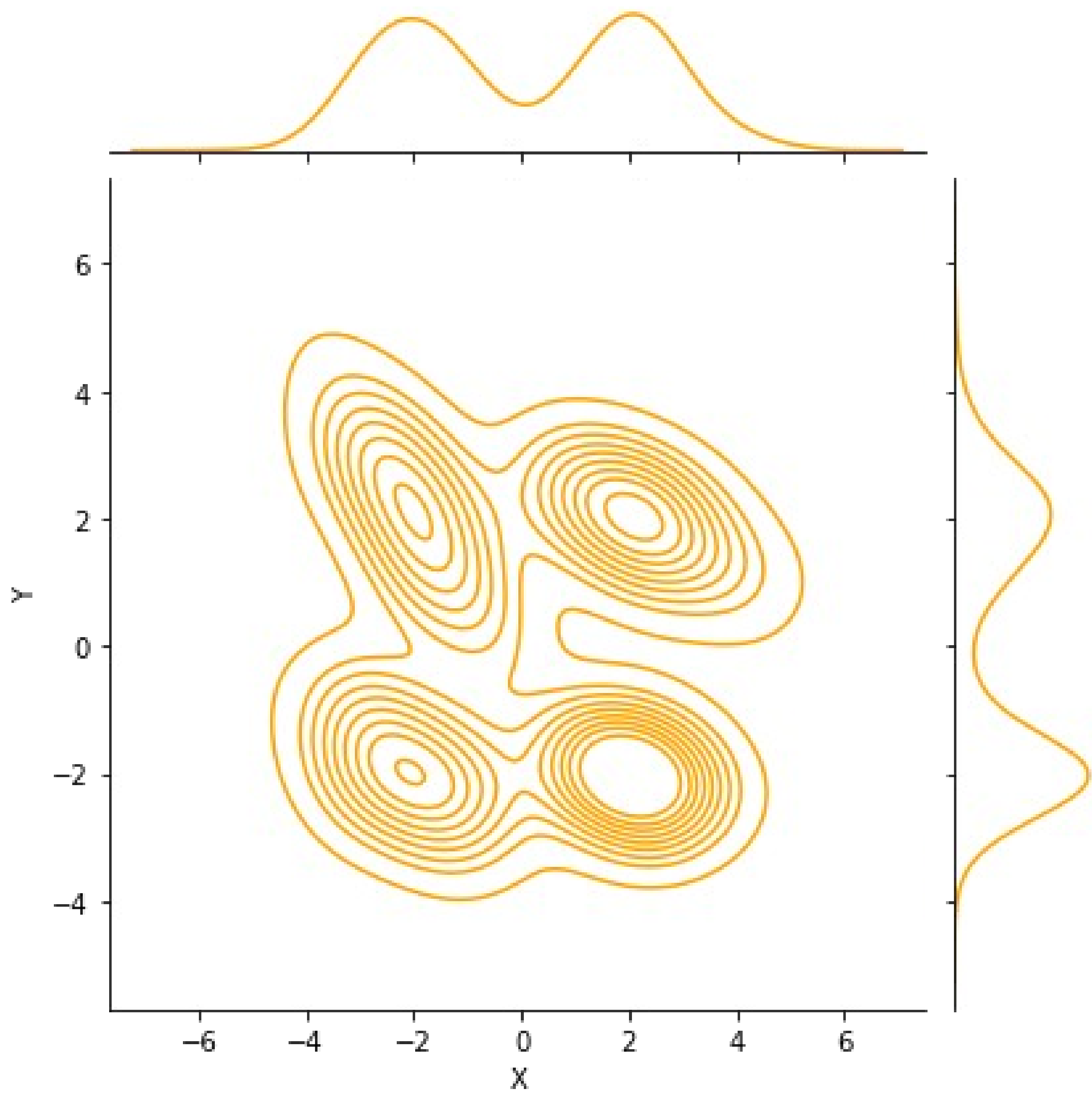



```
sns.jointplot(x=dclass["X"], y=dclass["Y"], kind='hist',)
#Marginal distribution through a hist
```

```
<seaborn.axisgrid.JointGrid at 0x7f184c407250>
```




```
sns.jointplot(x=dclass["X"], y=dclass["Y"], kind='kde', color =
"orange")
#Marginal distribution through a kde plot

<seaborn.axisgrid.JointGrid at 0x7f184baa5310>
```




```
print(dclass.size)
print(dclass.ndim)
#these functions gives the input space and the dimensions
```

```
2000
2
```

OUTPUT SPACE: The ouput space for this problem is the number of classes

---

Make_Regression

```
X, y = make_regression(n_samples=1000, n_features=2, n_informative=2,
random_state=1)
#dfr = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
```

```
dfr = pd.DataFrame(X) #Converting the samples from the dataset into a
pandas dataframe

dfr['output']=pd.DataFrame(y) #Adding the output values to the pandas
dataframe
#This dataset has 1000 samples , 2 columns , 2 informative features
```

statistics of make_regression dataset

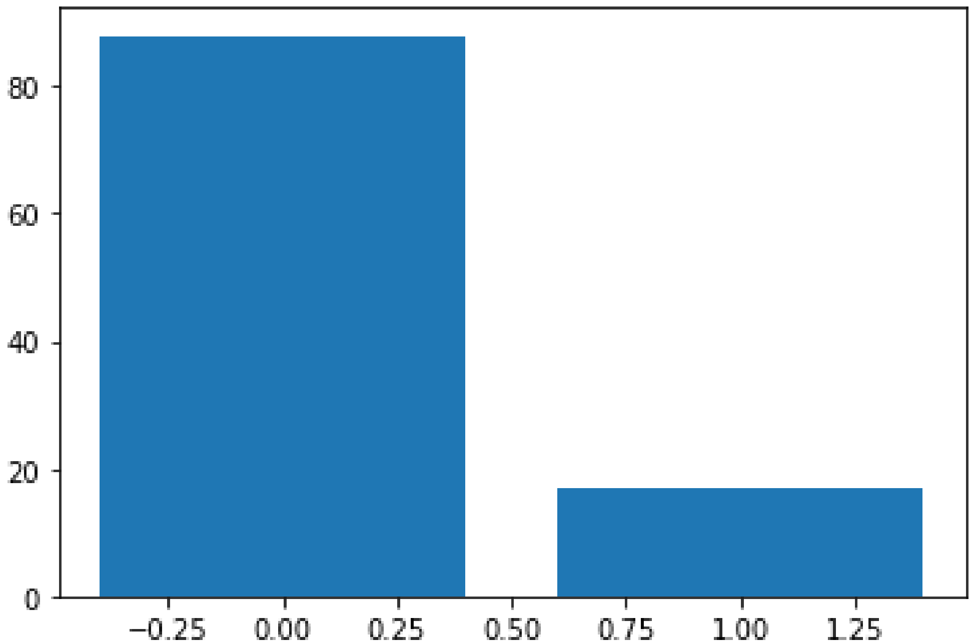
```
dfr.describe()
```

```
                 0            1       output
count  1000.000000  1000.000000  1000.000000
mean      0.039502     0.026636     3.922613
std       1.008864     1.003767    89.674542
min      -3.053764    -3.153357  -260.744396
25%      -0.625621    -0.619410   -56.534934
50%       0.052295     0.008454     7.846791
75%       0.698858     0.725486    64.070405
max       3.321079     3.958603   281.856951
```

Finding feature importance of the data model and plotting it.

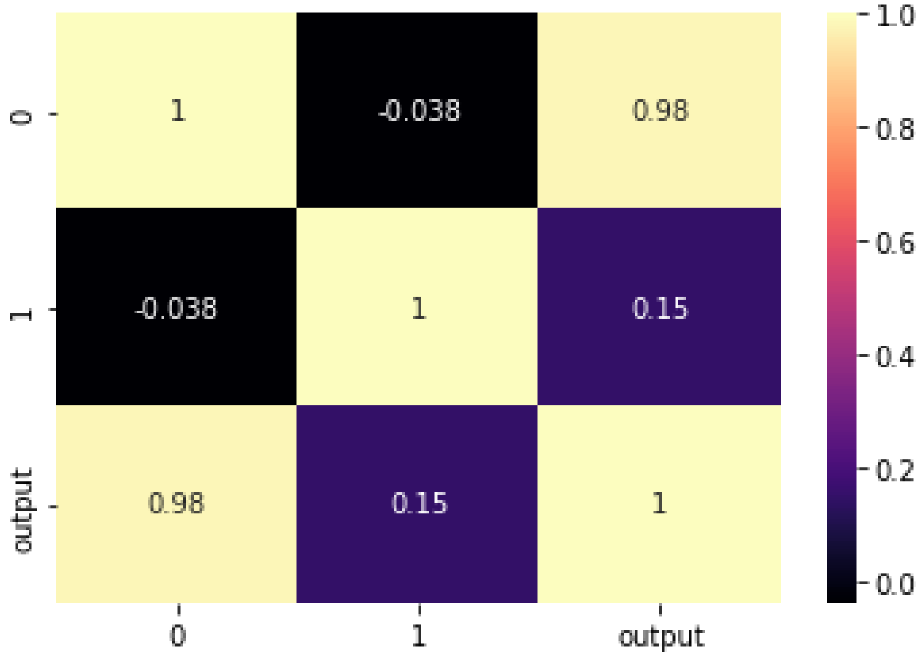
```
# define the model
model = LinearRegression()
# fit the model
model.fit(X, y)
# get importance
importance = model.coef_
# summarize feature importance
for i,v in enumerate(importance):
     print('Feature: %0d, Score: %.5f' % (i,v))

# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
Feature: 0, Score: 87.91986
Feature: 1, Score: 16.88002
```



Correlation Matrix and plots

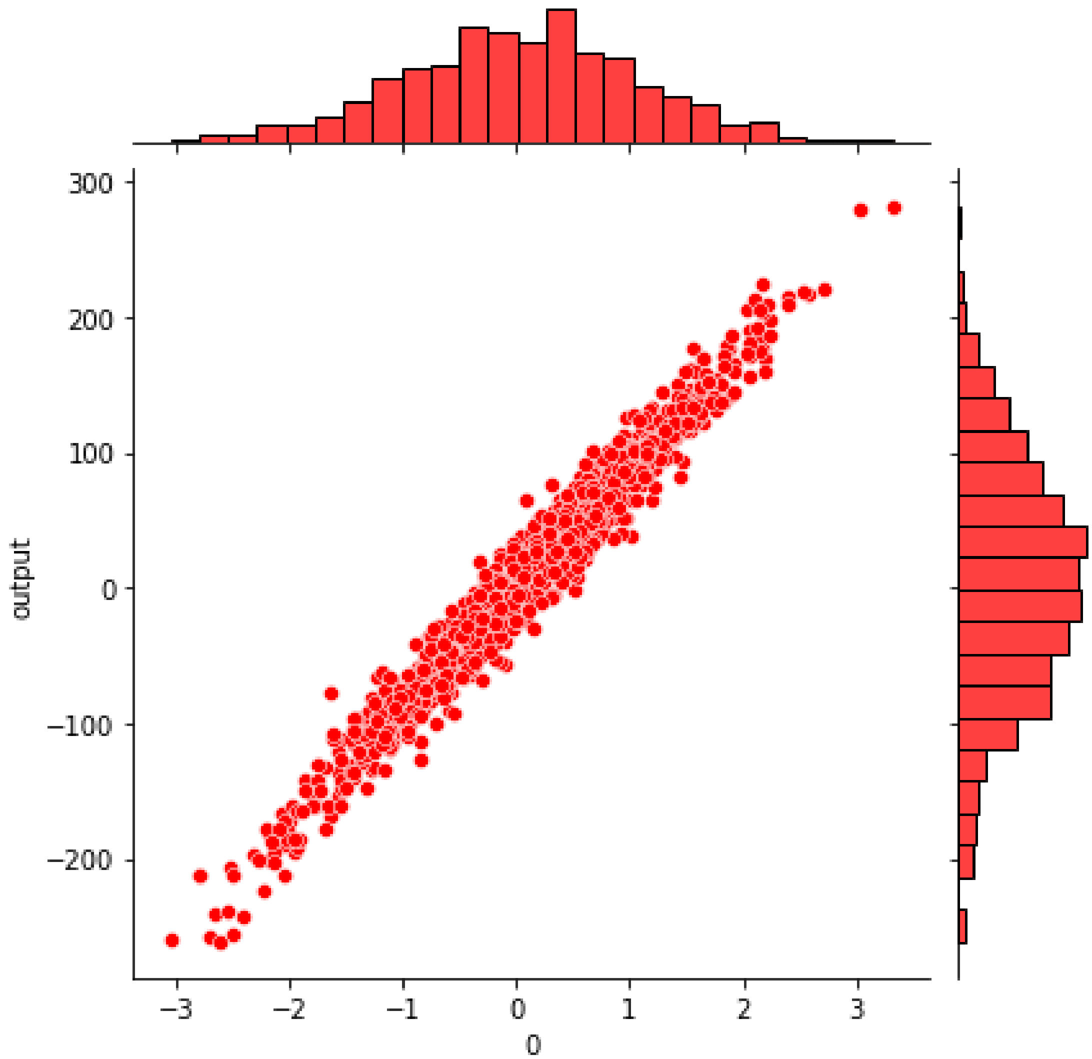
```
#correlation of x and y
sb.heatmap(dfr.corr(),annot=True,cmap='magma')

#Correlation Matrix
corr_matrix1 = dfr.corr()
print(corr_matrix1)

               0         1    output
0       1.000000 -0.037629  0.982013
1      -0.037629  1.000000  0.151726
output  0.982013  0.151726  1.000000
```



Marginal Distribution

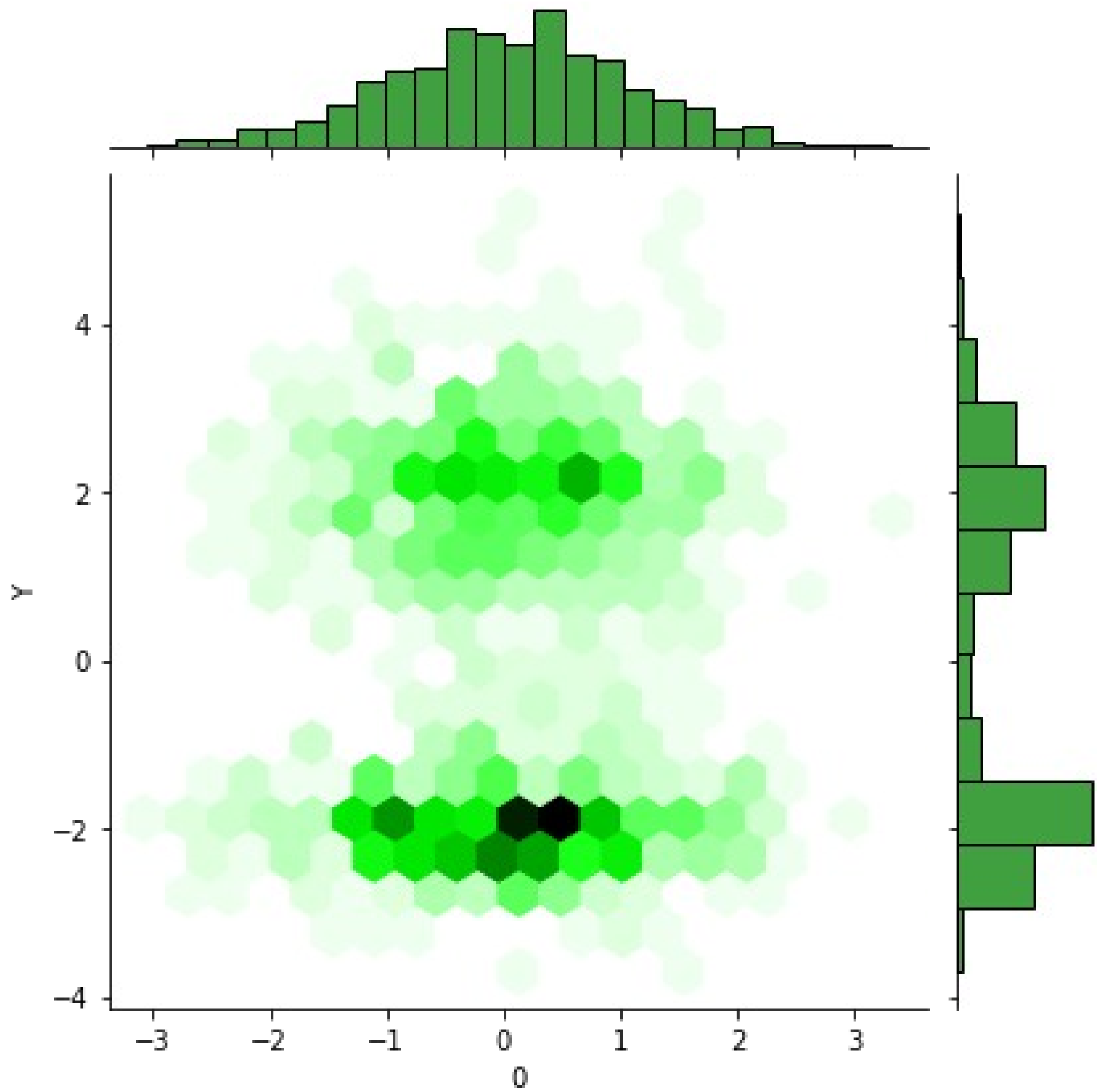
```
sns.jointplot(x=dfr[0], y=dfr["output"], kind='scatter', color =
"red")
#Marginal distribution through a scatter plot
```

```
<seaborn.axisgrid.JointGrid at 0x7f184b9af3d0>
```



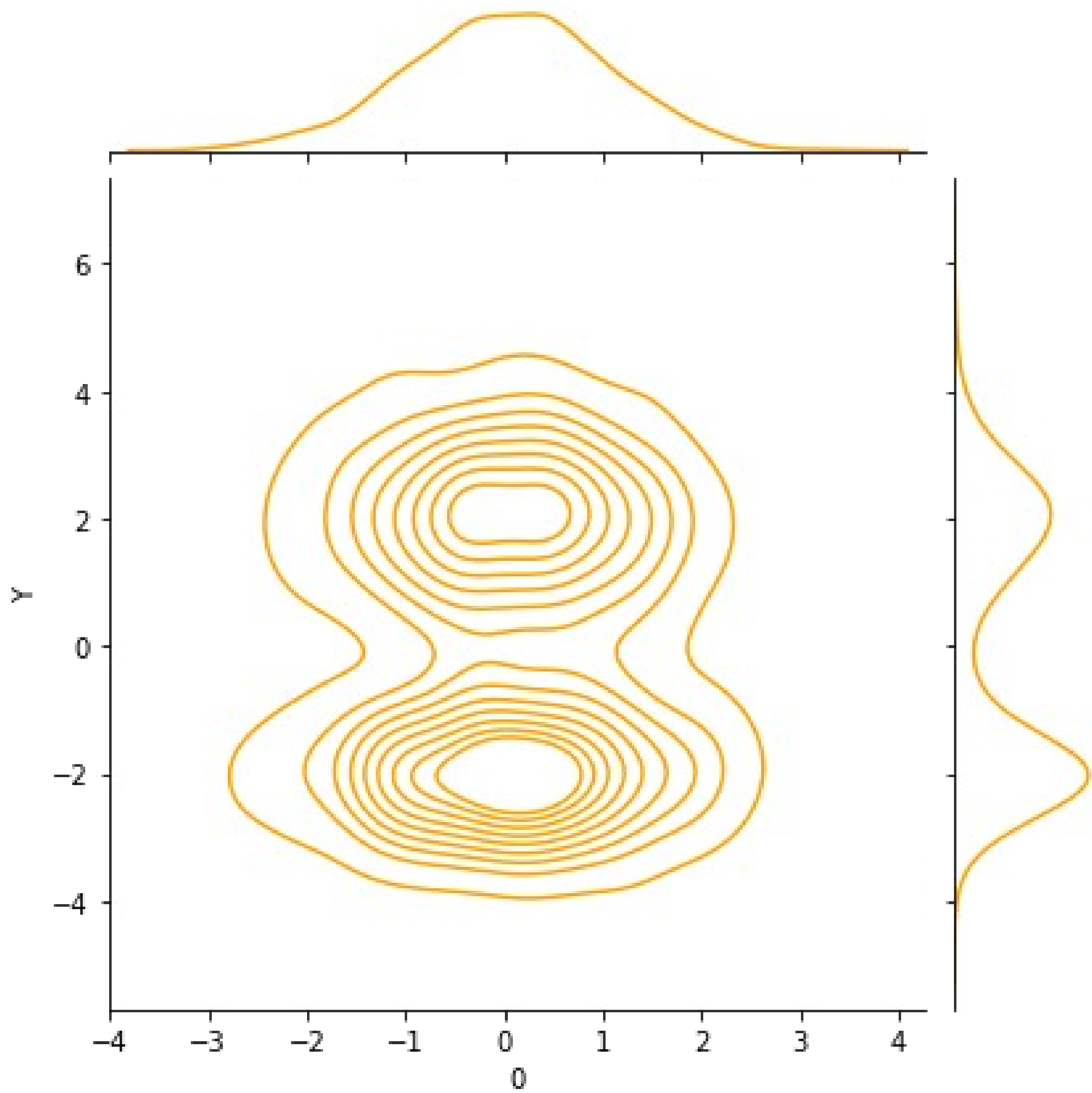

```
sns.jointplot(x=dfr[0], y=dclass["Y"], kind='hex',color = "green")
#Marginal distribution through a HEX
```

<seaborn.axisgrid.JointGrid at 0x7f184b84a7d0>




```
sns.jointplot(x=dfr[0], y=dclass["Y"], kind='kde',color = "orange")
#Marginal distribution through a KDE
```

```
<seaborn.axisgrid.JointGrid at 0x7f184b769ed0>
```




```
print("The Size of the input space is:", X.size)
print("The shape of the input sample is:", X.shape)
print("number of dimensions in the input space are:", X.ndim)
print("The size of the output space is :", y.size)
```

```
The Size of the input space is: 2000
The shape of the input sample is: (1000, 2)
number of dimensions in the input space are: 2
The size of the output space is : 1000
```

Normally, as X belongs to R asin real numbers, the range of X must be infinite.

2.E) We can obtain the same datasets if the functions are rerun using the parameter (Random_state). This parameter is responsible for making the output reproducable across multiple function calls when an integer value is passed through it.

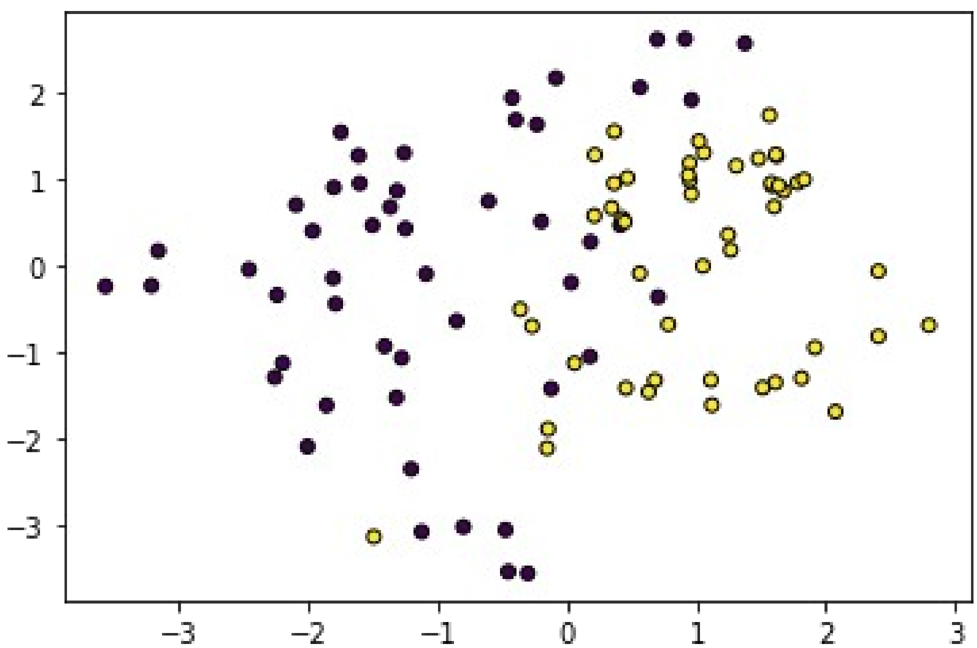
3. Implement K Nearest Neighbor

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.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
import pandas as pd
from sklearn.datasets import make_classification,make_regression
```

```
import numpy as np
X_class, y_class = make_classification(n_samples=100, n_features=2,
n_informative=2, n_redundant=0, n_repeated=0,n_classes=2)

plt.scatter(X_class[:, 0],X_class[:, 1], marker="o", c= y_class, s=25,
edgecolor="k")
```

```
<matplotlib.collections.PathCollection at 0x7f184b5a1d50>
```



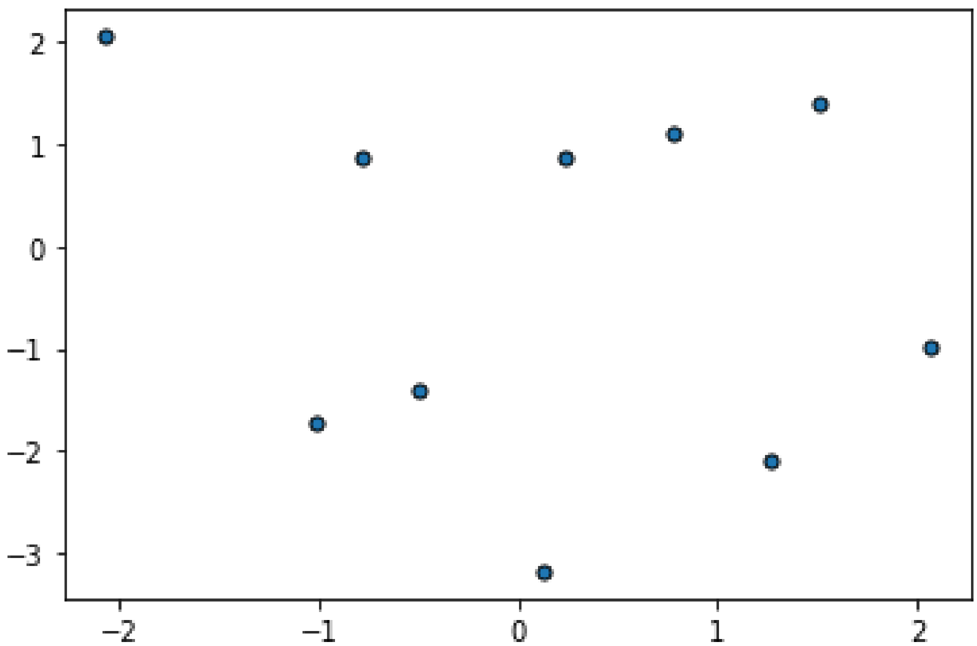
2.Generate 10 new points that belong to the same input space. Plot them using a different color in the above plot.

```
new_Train_x, new_Train_y = make_classification(n_samples=10,
n_features=2, n_informative=2, n_redundant=0,
n_repeated=0,n_classes=2)
```

```
# summarize the dataset
print(new_Train_x.shape, new_Train_y.shape)
plt.scatter(new_Train_x[:, 0],new_Train_x[:, 1], marker="o", s=26,
edgecolor="k")
```

```
(10, 2) (10,)
```

```
<matplotlib.collections.PathCollection at 0x7f184b560390>
```



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.

```
class Knn_classifier(object):
     def euclidian_distance(a, b):
          return np.sqrt(np.sum((a-b)**2, axis=1))


     def knn_distances(xTrain,xTest,k):

          #the following formula calculates the Euclidean distances.
          import numpy as np
          distances = -2 * xTrain@xTest.T + np.sum(xTest**2,axis=1) +
np.sum(xTrain**2,axis=1)[:, np.newaxis]
          #because of float precision, some small numbers can become
negatives. Need to be replace with 0.
          distances[distances < 0] = 0
          distances = distances**.5
          indices = np.argsort(distances, 0) #get indices of sorted
items
```

```
        distances = np.sort(distances,0) #distances sorted in axis 
0
        #returning the top-k closest distances.
        return indices[0:k,:], distances[0:k,:]


    def knn_predictions(xTrain,yTrain,xTest,k):

        import numpy as np
        indices, distances = 
Knn_classifier.knn_distances(xTrain,xTest,k)
        yTrain = yTrain.flatten()
        rows, columns = indices.shape
        predictions = list()
        for j in range(columns):
            temp = list()
            for i in range(rows):
                cell = indices[i][j]
                temp.append(yTrain[cell])
            predictions.append(max(temp,key=temp.count)) #this is 
the key function, brings the mode value
        predictions=np.array(predictions)
        return predictions

z=Knn_classifier.knn_predictions(X_class,y_class,X_class,3)

print("Actual Class")
print(y_class)
print("predicted_class")
print(z)
```

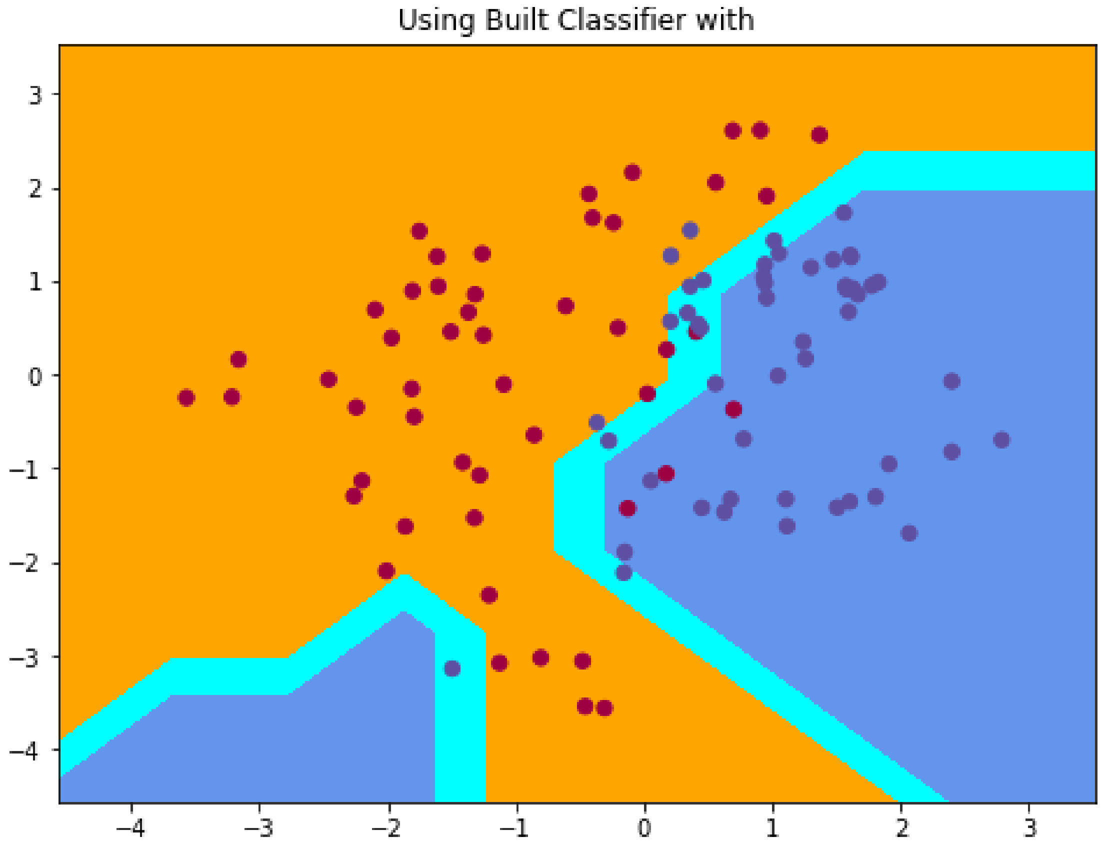
```
Actual Class
[0 1 0 1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1 1 1 0 0 0 0 1 0 1 
1 0
 1 0 0 1 0 0 0 1 1 0 0 1 1 1 1 1 0 1 0 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 0 
1 1
 0 1 0 0 0 1 0 0 1 1 0 1 0 1 1 0 1 0 1 0 0 0 1 0 0 1]
predicted_class
[0 1 0 1 1 0 1 0 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 1 1 1 1 0 0 0 0 1 0 1 
1 0
 1 1 0 1 0 0 0 1 0 0 0 1 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 0 0 1 1 1 0 0 0 
1 1
 0 1 0 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 1]
```

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

```
cmap_light = ListedColormap(["orange", "cyan", "cornflowerblue"])
cmap_bold = ["darkorange", "c", "darkblue"]

h=0.9
k_neighbors=[1,5,10]
```

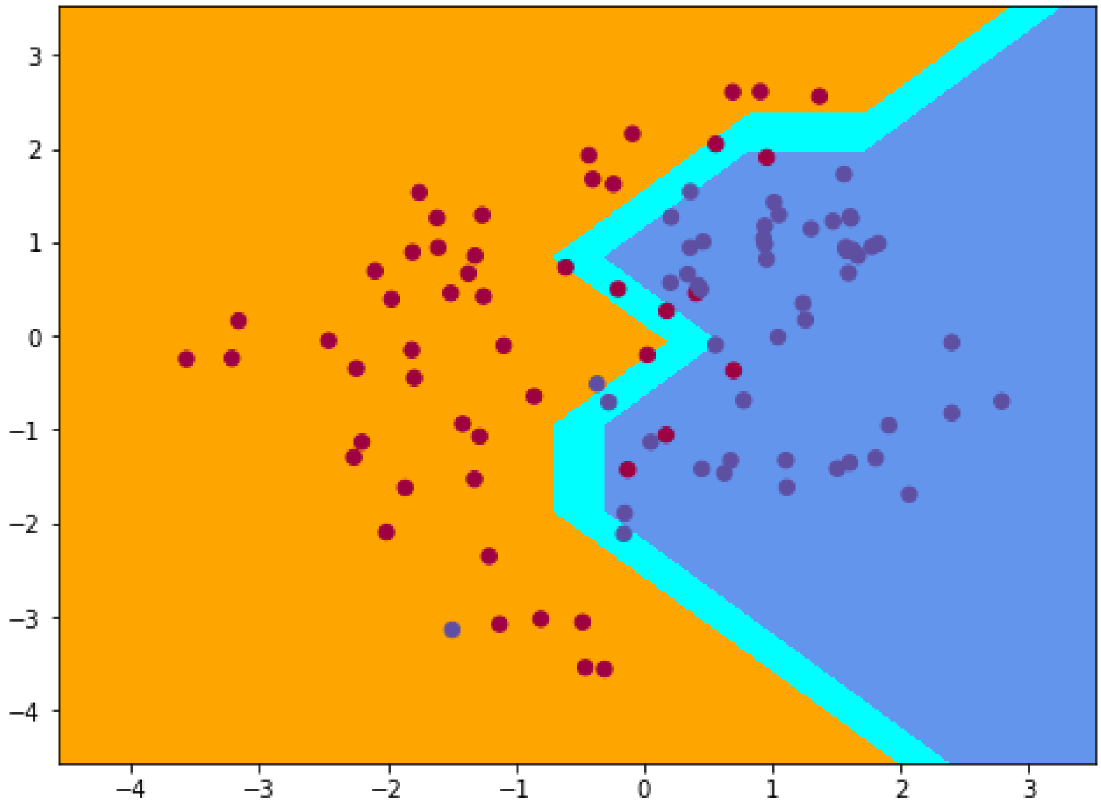
```
# we create an instance of Neighbours Classifier and fit the data.
for i in range(3):

  # 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 = X_class[:, 0].min() - 1, X_class[:, 0].max() + 1
  y_min, y_max = X_class[:, 1].min() - 1, X_class[:, 1].max() + 1
  xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
  #print(xx.shape)
  print("nearest neighbors are ", k_neighbors[i])
  #print(yy)
  #Z = knn_predictions(X,y,X)
  Z = Knn_classifier.knn_predictions(X_class,y_class,np.c_[xx.ravel(),
yy.ravel()],k_neighbors[i])
  # Put the result into a color plot
  Z = Z.reshape(xx.shape)
  plt.figure(figsize=(8, 6))
  plt.contourf(xx, yy, Z, cmap=cmap_light)
  #plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.7)
  plt.scatter(X_class[:, 0], X_class[:, 1], c=y_class, s=40,
cmap=plt.cm.Spectral)
  plt.title('Using Built Classifier with')
```

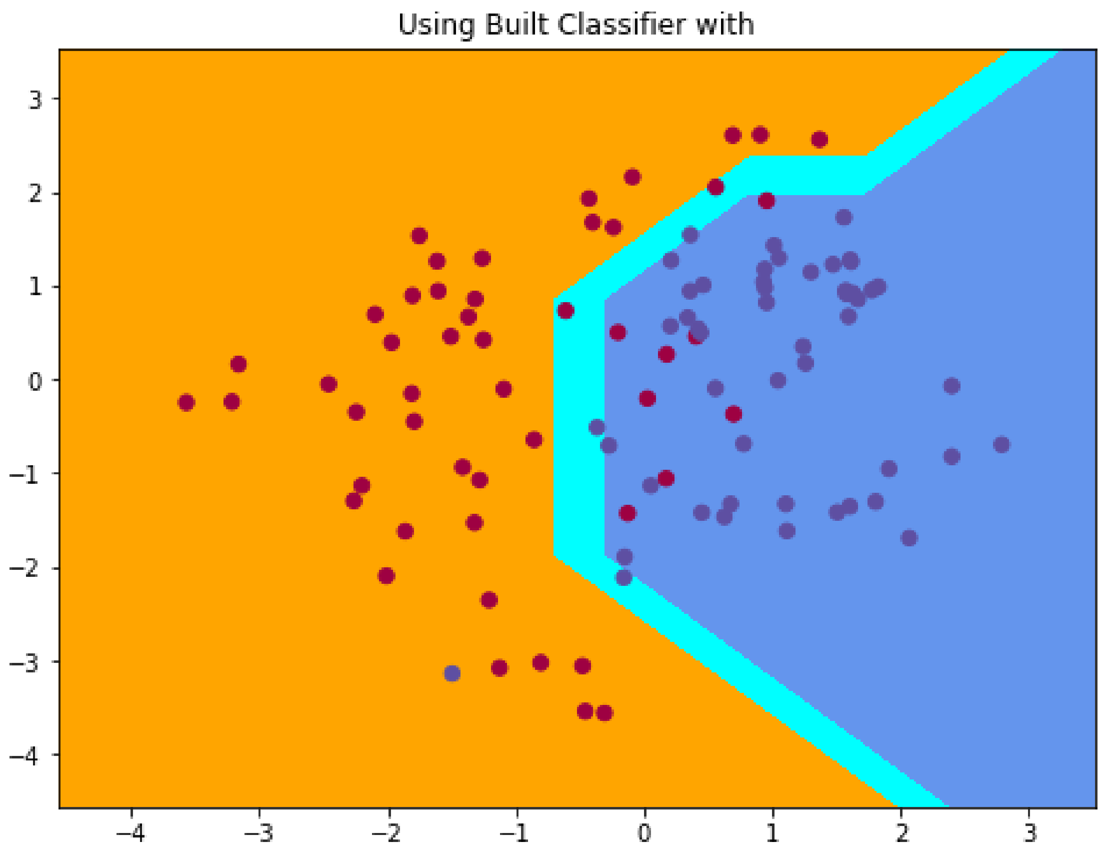
```
nearest neighbors are  1
nearest neighbors are  5
nearest neighbors are  10
```



Using Built Classifier with
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

Using Built Classifier with





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.

```
df=np.zeros((10,3))
k_neighbors=[1,10,100]
for i in range(3):
    df[:,i] = 
Knn_classifier.knn_predictions(new_Train_x,new_Train_y,new_Train_x,k_n
eighbors[i])
df_new=pd.DataFrame(df)
df_new.columns=['K=1','k=10','k=100']
df_new
```

	K=1	k=10	k=100
0	1.0	1.0	1.0
1	0.0	0.0	0.0
2	1.0	1.0	1.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0
5	1.0	1.0	1.0
6	1.0	1.0	1.0
7	0.0	0.0	0.0

```
8  0.0   0.0    0.0
9  1.0   1.0    1.0
```

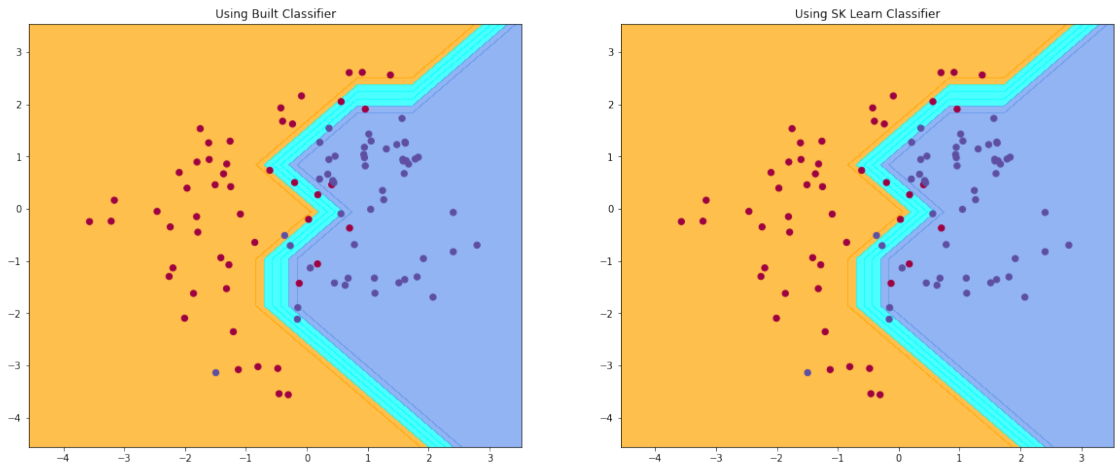
6.Instantiate the k nearest neighbor from scikit-learn and compare the decision boundaries and labels obtained from your method.

```
# KNN using Scikitlearn
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_class, y_class)
train_preds = knn_model.predict(X_class)
print(train_preds)
print(train_preds.shape)

# Comparison of Labels
df_class=np.zeros((100,3))
df_class[:,0]=(Knn_classifier.knn_predictions(X_class,y_class,X_class,
5))
df_class[:,1]=(train_preds)
df_class[:,2]=y_class
df_class=pd.DataFrame(df_class)
df_class.columns=['Using Built Class','Using SK Learn','Actual Class']

print(df_class.head())

cmap_light = ListedColormap(["orange", "cyan", "cornflowerblue"])
cmap_bold = ["darkorange", "c", "darkblue"]

# Companrison of Decision Boundaries
x_min, x_max = X_class[:, 0].min() - 1, X_class[:, 0].max() + 1
y_min, y_max = X_class[:, 1].min() - 1, X_class[:, 1].max() + 1
h=0.9

# Using Built Classifier
fig,axes = plt.subplots(1,2,figsize=(20,8))
xx_class, yy_class = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
Z_class=Knn_classifier.knn_predictions(X_class,y_class,np.c_[xx_class.
ravel(), yy_class.ravel()],5)
Z_class = Z_class.reshape(xx_class.shape)
plt.figure(figsize=(8, 6))
axes[0].contourf(xx_class, yy_class, Z_class, cmap=cmap_light,
alpha=0.7)
axes[0].scatter(X_class[:, 0], X_class[:, 1], c=y_class, s=40,
cmap=plt.cm.Spectral)
axes[0].set_title('Using Built Classifier')

# Using Scikit Learn Classifier
xx_sk, yy_sk = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
```

```
print(xx.shape)
Z_class_sk=knn_model.predict(np.c_[xx_sk.ravel(), yy_sk.ravel()])
Z_class_sk = Z_class_sk.reshape(xx.shape)
axes[1].contourf(xx_sk, yy_sk, Z_class_sk, cmap=cmap_light, alpha=0.7)
axes[1].scatter(X_class[:, 0], X_class[:, 1], c=y_class, s=40,
cmap=plt.cm.Spectral)
axes[1].set_title('Using SK Learn Classifier')
```

```
[0 1 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1
1 0
 1 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0
1 1
 0 1 0 0 0 1 0 0 1 1 0 1 0 1 1 0 1 1 1 0 0 0 1 0 0 1]
(100,)
   Using Built Class  Using SK Learn  Actual Class
0                0.0             0.0           0.0
1                1.0             1.0           1.0
2                0.0             0.0           0.0
3                1.0             1.0           1.0
4                1.0             1.0           1.0
(10, 10)
```

```
Text(0.5, 1.0, 'Using SK Learn Classifier')
```




```
<Figure size 576x432 with 0 Axes>
```

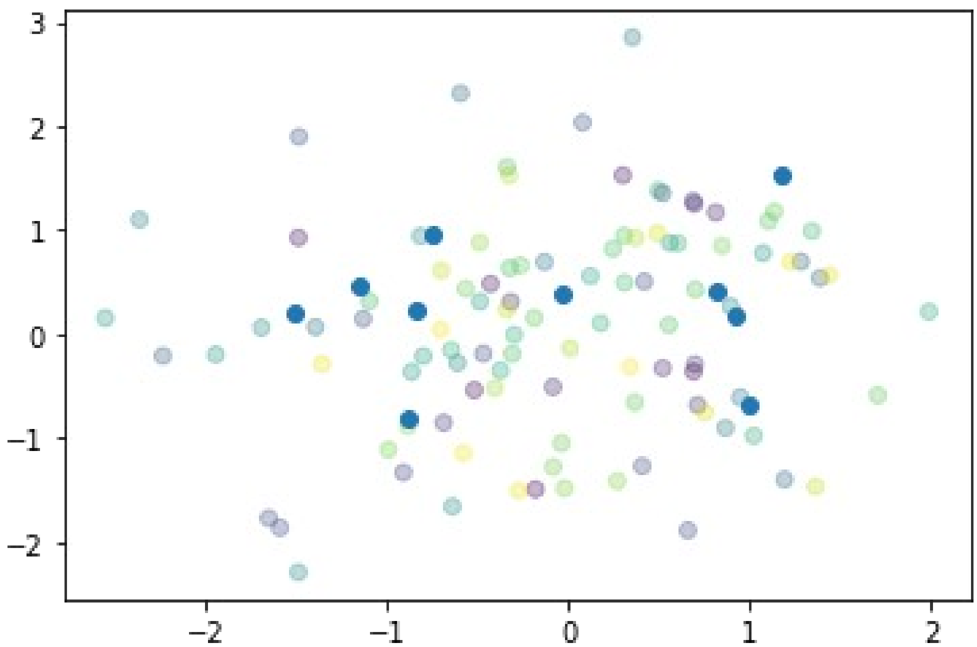
7.Repeat steps 1,2,3 for a random two dimensional regression dataset based on Q2. Repeat step 5 where you compute and display the estimated numeric output for each of the three settings (k equals 1, 10 and 100). Repeat step 6 with the equivalent implementation from scikit-learn and compare the predictions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

```
X_reg,y_reg = make_regression(n_samples=100, n_features=2,
n_informative=2)
```

```
plt.scatter(X_reg[:,0],X_reg[:,1],c=np.random.rand(100),alpha=0.3)
```

```
new_X_reg,new_y_reg = make_regression(n_samples=10, n_features=2,
n_informative=2)
plt.scatter(new_X_reg[:,0],new_X_reg[:,1],alpha=1)
```

```
<matplotlib.collections.PathCollection at 0x7f184b213950>
```




```
df=np.zeros((10,3))
k_neighbors=[1,10,100]
for i in range(3):
    df[:,i] =
Knn_classifier.knn_predictions(new_X_reg,new_y_reg,new_X_reg,k_neighbo
rs[i])
df_new=pd.DataFrame(df)
df_new.columns=['K=1','k=10','k=100']
df_new
```

	K=1	k=10	k=100
0	-31.069706	-31.069706	-31.069706
1	16.305586	16.305586	16.305586
2	14.247252	14.247252	14.247252
3	7.772111	7.772111	7.772111
4	-23.634193	-23.634193	-23.634193
5	7.419413	7.419413	7.419413
6	34.508093	34.508093	34.508093
7	16.217405	16.217405	16.217405
8	6.117984	6.117984	6.117984
9	57.648749	57.648749	57.648749

```
from sklearn.neighbors import KNeighborsRegressor
knn_model = KNeighborsRegressor(n_neighbors=5)
knn_model.fit(X_reg,y_reg)
```

```
train_preds = knn_model.predict(X_reg)
print(train_preds)
print(train_preds.shape)

# Comparison of Labels
df_reg=np.zeros((100,3))
df_reg[:,0]=(Knn_classifier.knn_predictions(X_reg,y_reg,X_reg,5))
df_reg[:,1]=(train_preds)
df_reg[:,2]=y_reg
df_reg=pd.DataFrame(df_reg)
df_reg.columns=['Using Built Class','Using SciKit Learn','Actual
Class']
print(df_reg.head())
```

```
[-7.31293871e+01  4.13569995e+01  4.53025297e+01  4.53025297e+01
  1.04856964e+02 -2.83558408e-03 -8.39339983e+00  4.20486848e+01
  2.96520903e+01 -2.64650571e+01 -5.34086212e+01  9.72108389e+00
  8.83387224e+01 -1.02405620e+00 -2.67872626e+01 -7.31293871e+01
  9.31772154e+01  6.29962817e+01  4.57371114e+01  5.65579413e+01
 -2.34336384e+01 -4.49817271e+01  6.25014128e+01  8.41248646e+00
 -8.41722471e+01  1.03583258e+02 -1.60317504e+01  1.16588715e+01
 -7.26453809e+00  2.73972542e+01 -9.78016958e+01  5.59663222e+01
  3.98980497e+01  3.65850445e+00 -3.86746734e+01 -1.63589049e+01
  8.19853988e+01 -8.88358598e+01 -9.82099808e+01  2.01610482e+01
 -3.72398986e+01 -9.19375576e+01  6.29962817e+01 -6.22462136e+01
  2.46045530e+01  2.03909550e+01  2.41991091e+01 -1.01614051e+02
  5.08397874e+01 -9.82099808e+01  1.41414458e+02  7.16374354e+01
 -1.34014152e+01  4.77523728e+01  1.28145931e+02  4.57371114e+01
 -1.14959854e+02  6.22154534e+01  1.37131651e+01  8.83387224e+01
  1.37131651e+01  6.22154534e+01 -1.01835674e+02 -2.45905077e+01
 -1.09754772e+01  3.85100267e+01 -2.13637459e+01  5.78196816e+01
 -4.49817271e+01 -5.63712870e+00 -2.67872626e+01  4.56799171e+01
  3.85100267e+01 -9.04167774e+01 -3.44812935e+00 -1.39721285e+01
 -5.34086212e+01 -9.49267229e+01  2.93023543e+01 -1.09754772e+01
 -2.57857208e+01  9.31772154e+01  1.23200547e+02 -5.34086212e+01
  9.31772154e+01 -9.32232469e+01  4.53025297e+01  6.40507785e+01
  7.16374354e+01 -1.22427290e+02 -3.01627591e+01  4.58486444e+00
  1.03583258e+02  6.16950408e+01 -1.14959854e+02 -2.64650571e+01
  4.53025297e+01  1.58464639e+01 -7.31293871e+01 -9.04167774e+01]
(100,)
   Using Built Class  Using SciKit Learn  Actual Class
0         -78.456827          -73.129387    -78.456827
1          47.312616           41.356999     47.312616
2          37.346869           45.302530     37.346869
3          39.411680           45.302530     39.411680
4         129.296406          104.856964    129.296406
```

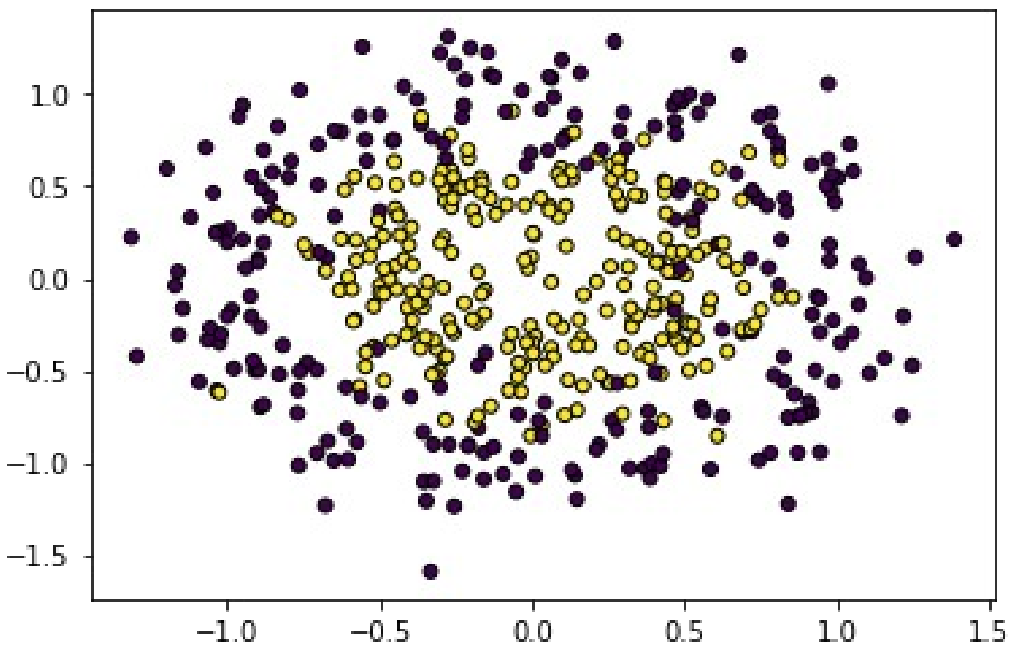
Question 4:

```
from sklearn.datasets import make_moons, make_circles, make_blobs
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import random
from pprint import pprint
```

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

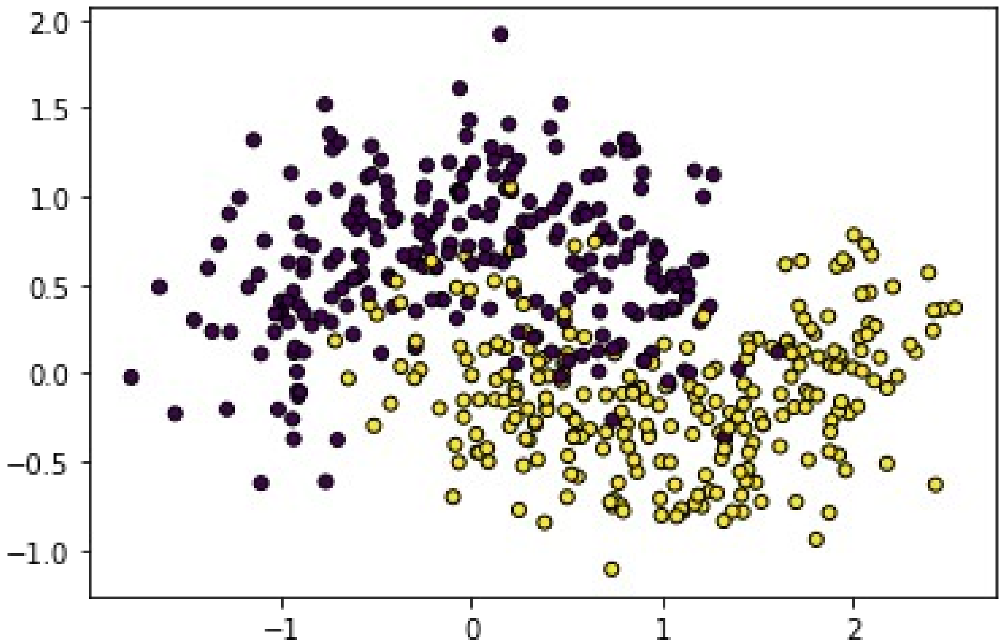
```
X1, Y1 = make_circles(noise=0.2, factor=0.5, random_state=1,
n_samples=500)
plt.scatter(X1[:, 0], X1[:, 1], marker="o", c=Y1, s=25, edgecolor="k")
```

```
<matplotlib.collections.PathCollection at 0x7f184b287b90>
```



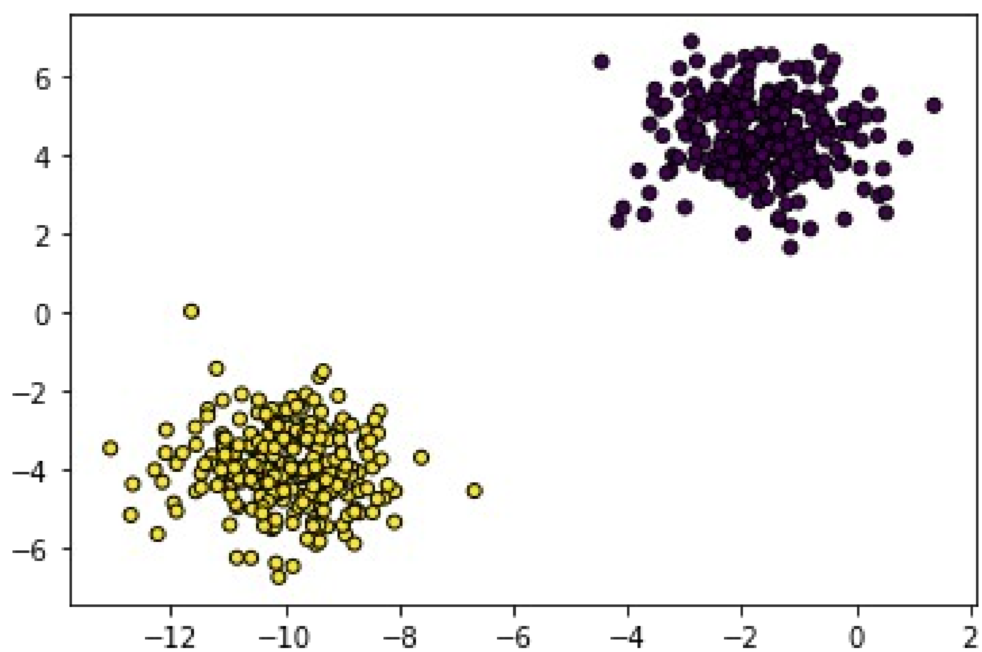

```
X1, Y1 = make_moons(noise=0.3, random_state=1, n_samples=500)
plt.scatter(X1[:, 0], X1[:, 1], marker="o", c=Y1, s=25, edgecolor="k")
```

```
<matplotlib.collections.PathCollection at 0x7f184b1416d0>
```




```
X1, Y1 = make_blobs(n_samples=500, n_features=2, centers=2,
random_state=1)
plt.scatter(X1[:, 0], X1[:, 1], marker="o", c=Y1, s=25, edgecolor="k")
```

```
<matplotlib.collections.PathCollection at 0x7f184b0b1250>
```




```
datasets = {
    'moons': dict(zip(['x', 'y'], make_moons(noise=0.3,
random_state=1, n_samples=500))),
    'circles': dict(zip(['x', 'y'], make_circles(noise=0.2,
```

```
factor=0.5, random_state=1, n_samples=500))),
    'blobs': dict(zip(['x', 'y'], make_blobs(n_samples =
500,n_features=3, centers=2, random_state=1))),
}
```

Decision Tree algorithm class from scratch

```
class Node:
    def __init__(self, predicted_class):
        self.predicted_class = predicted_class
        self.feature_index = 0
        self.threshold = 0
        self.left = None
        self.right = None


class DecisionTreeClassifierFromScratch:
    def __init__(self, max_depth=None,criteria=None):
        self.max_depth = max_depth
        self.criteria = criteria

    def fit(self, X, y):
        self.n_classes_ = len(set(y))
        self.n_features_ = X.shape[1]
        self.tree_ = self._grow_tree(X, y)

    def predict(self, X):
        return [self._predict(inputs) for inputs in X]

    def entropy(self,y):
        m = y.size
        best_entropy = sum(-(n / m)*np.log2((n / m)+1e-9) for n in
self.num_parent)
        return(best_entropy)

    def gini(self,y):
        m = y.size
        best_gini = 1.0 - sum((n / m) ** 2 for n in self.num_parent)
        return(best_gini)


    def _best_split(self, X, y):
        m = y.size
        if m <= 1:
            return None, None
        self.num_parent = [sum(y == c) for c in
range(self.n_classes_)]
        if self.criteria == 'gini':
            best_gini = self.gini(y)
        else:
            best_gini = self.entropy(y)
```

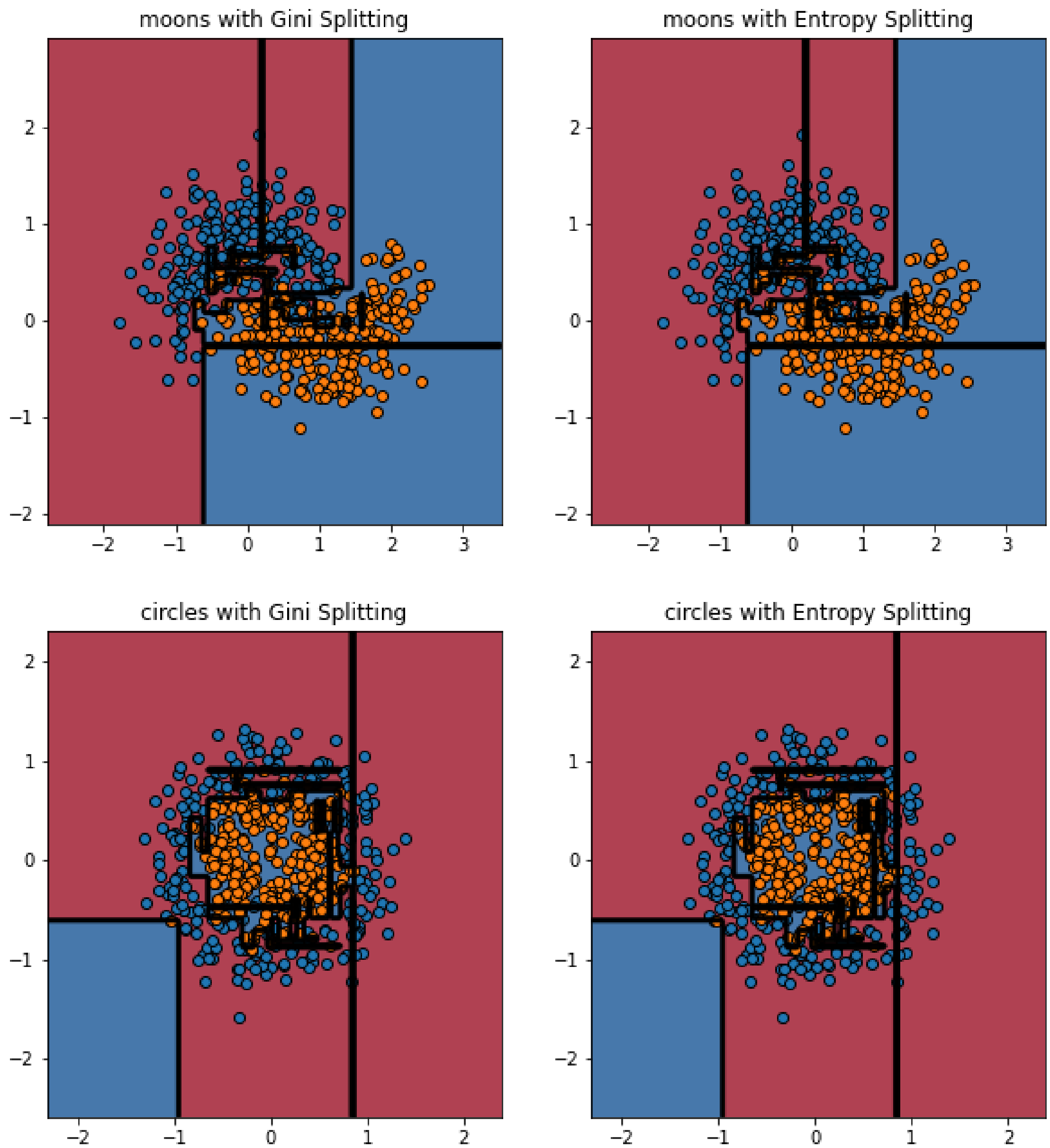
```
        best_idx, best_thr = None, None
        for idx in range(self.n_features_):
            thresholds, classes = zip(*sorted(zip(X[:, idx], y)))
            num_left = [0] * self.n_classes_
            num_right = self.num_parent.copy()
            for i in range(1, m):
                c = classes[i - 1]
                num_left[c] += 1
                num_right[c] -= 1
                gini_left = 1.0 - sum(
                    (num_left[x] / i) ** 2 for x in
range(self.n_classes_)
                )
                gini_right = 1.0 - sum(
                    (num_right[x] / (m - i)) ** 2 for x in
range(self.n_classes_)
                )
                gini = (i * gini_left + (m - i) * gini_right) / m
                if thresholds[i] == thresholds[i - 1]:
                    continue
                if gini < best_gini:
                    best_gini = gini
                    best_idx = idx
                    best_thr = (thresholds[i] + thresholds[i - 1]) / 2
        return best_idx, best_thr

    def _grow_tree(self, X, y, depth=0):
        num_samples_per_class = [np.sum(y == i) for i in
range(self.n_classes_)]
        predicted_class = np.argmax(num_samples_per_class)
        node = Node(predicted_class=predicted_class)
        if depth < self.max_depth:
            idx, thr = self._best_split(X, y)
            if idx is not None:
                indices_left = X[:, idx] < thr
                X_left, y_left = X[indices_left], y[indices_left]
                X_right, y_right = X[~indices_left], y[~indices_left]
                node.feature_index = idx
                node.threshold = thr
                node.left = self._grow_tree(X_left, y_left, depth + 1)
                node.right = self._grow_tree(X_right, y_right, depth +
1)
        return node


    def _predict(self, inputs):
        node = self.tree_
        while node.left:
            if inputs[node.feature_index] < node.threshold:
                node = node.left
```

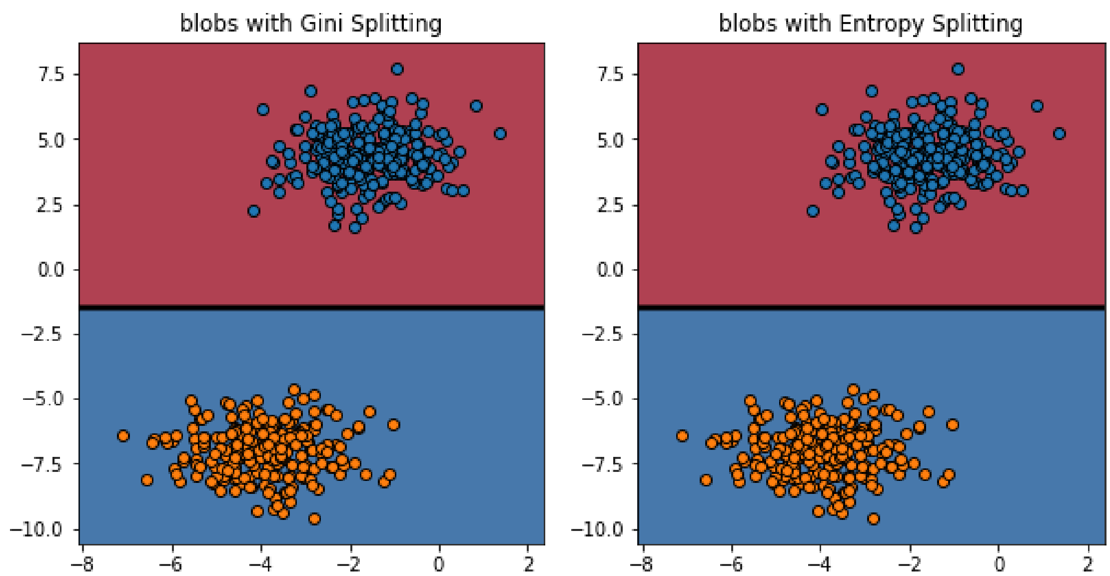
```
            else:
                node = node.right
        return node.predicted_class

    def accuracy(self,actual, predicted):
        correct = 0
        for i in range(len(actual)):
            if actual[i] == predicted[i]:
                correct += 1
        return correct / float(len(actual)) * 100.0
```

Plotting Decison Boundaries

```
def decision_boundary(model, df, ax=None):

    max_x = np.max(df['x']) + 1
    max_y = np.max(df['y']) + 1

    min_x = np.min(df['x']) - 1
    min_y = np.min(df['y']) - 1

    xs = np.linspace(min_x, max_x, 200)
    ys = np.linspace(min_y, max_y, 200)
    zs = np.zeros((200, 200))

    for i,x in enumerate(xs):
        for j,y in enumerate(ys):
            zs[j, i] = model.predict(np.array([[x, y]]))[0]
    cm = plt.cm.RdBu
    ax.contour(xs, ys, zs, linewidths=2, colors='black', alpha=0.5)
    ax.contourf(xs, ys, zs, cmap=cm, alpha=0.8)

    cm_bright = ListedColormap(["#FF0000", "#0000FF"])
    ax.scatter(x='x', y='y', data=df[df['label'] == 0],
cmap=cm_bright, edgecolors="k")
    ax.scatter(x='x', y='y', data=df[df['label'] == 1],
cmap=cm_bright, edgecolors="k")

    ax.set_xlim((min_x, max_x))
    ax.set_ylim((min_y, max_y))

from matplotlib.colors import ListedColormap
for name in datasets.keys():
    df = pd.DataFrame(datasets[name]['x'])
    df = df.rename(columns={0: 'x', 1: 'y'})
    df['label'] = datasets[name]['y'].tolist()
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
    clf_gini = DecisionTreeClassifierFromScratch(max_depth=20,
criteria='gini')
    clf_gini.fit(datasets[name]['x'],datasets[name]['y'])
    ax1.set_title('{} with Gini Splitting'.format(name))
```

```
    decision_boundary(clf_gini, df, ax1)
    clf_entropy = DecisionTreeClassifierFromScratch(max_depth=20,
criteria='entropy')
    clf_entropy.fit(datasets[name]['x'],datasets[name]['y'])
    ax2.set_title('{} with Entropy Splitting'.format(name))
    decision_boundary(clf_entropy, df, ax2)
```





Scikit-learn Classifier

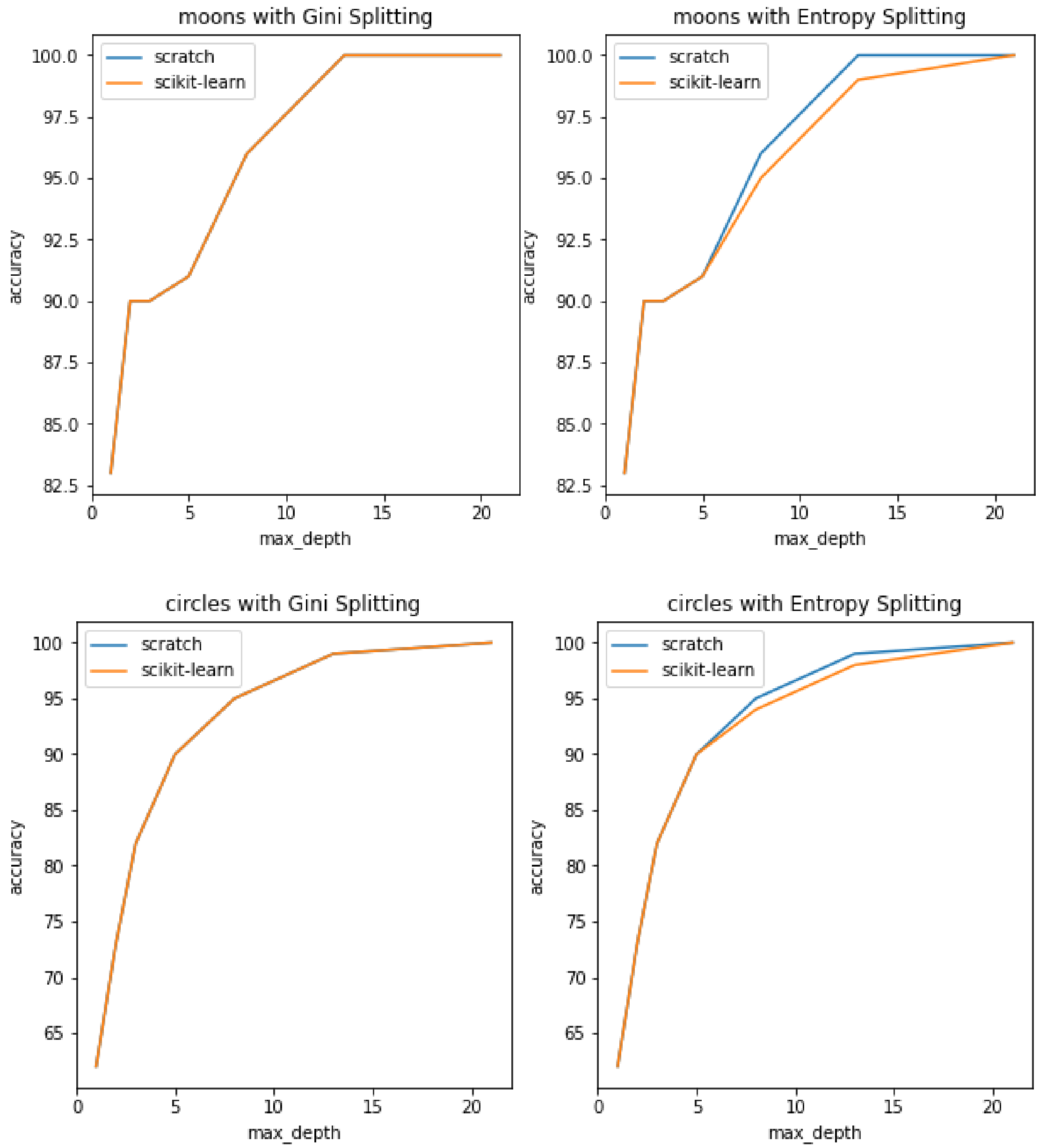
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
max_depths = [1, 2, 3, 5, 8, 13, 21]
criteria = ['gini', 'entropy']
for criterion in criteria:
    for max_depth in max_depths:
        clf = DecisionTreeClassifier(max_depth=max_depth, 
criterion=criterion)
        clf.fit(X1,Y1)
        y_pred = clf.predict(X1)
        print('accuracy =', int(accuracy_score(Y1,y_pred)*100),'when 
max_depth =',clf.max_depth, 'and criteria =',criterion)
```

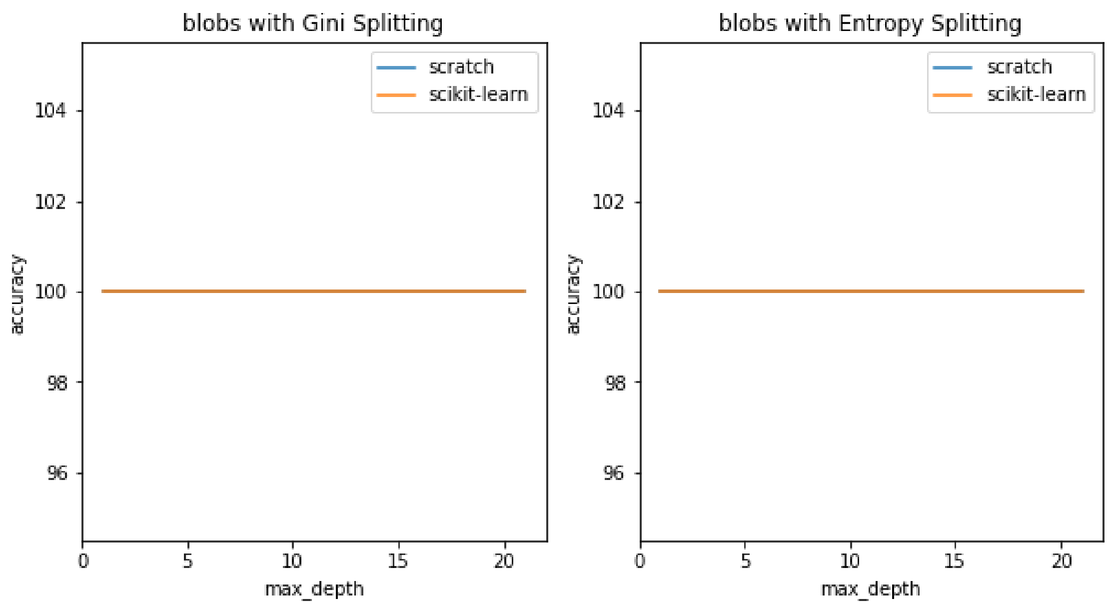
```
accuracy = 100 when max_depth = 1 and criteria = gini
accuracy = 100 when max_depth = 2 and criteria = gini
accuracy = 100 when max_depth = 3 and criteria = gini
accuracy = 100 when max_depth = 5 and criteria = gini
accuracy = 100 when max_depth = 8 and criteria = gini
accuracy = 100 when max_depth = 13 and criteria = gini
accuracy = 100 when max_depth = 21 and criteria = gini
accuracy = 100 when max_depth = 1 and criteria = entropy
accuracy = 100 when max_depth = 2 and criteria = entropy
accuracy = 100 when max_depth = 3 and criteria = entropy
accuracy = 100 when max_depth = 5 and criteria = entropy
accuracy = 100 when max_depth = 8 and criteria = entropy
accuracy = 100 when max_depth = 13 and criteria = entropy
accuracy = 100 when max_depth = 21 and criteria = entropy
```

Comparing both the classifier's performance

```
max_depths = [1, 2, 3, 5, 8, 13, 21]
criteria = ['gini', 'entropy']
```

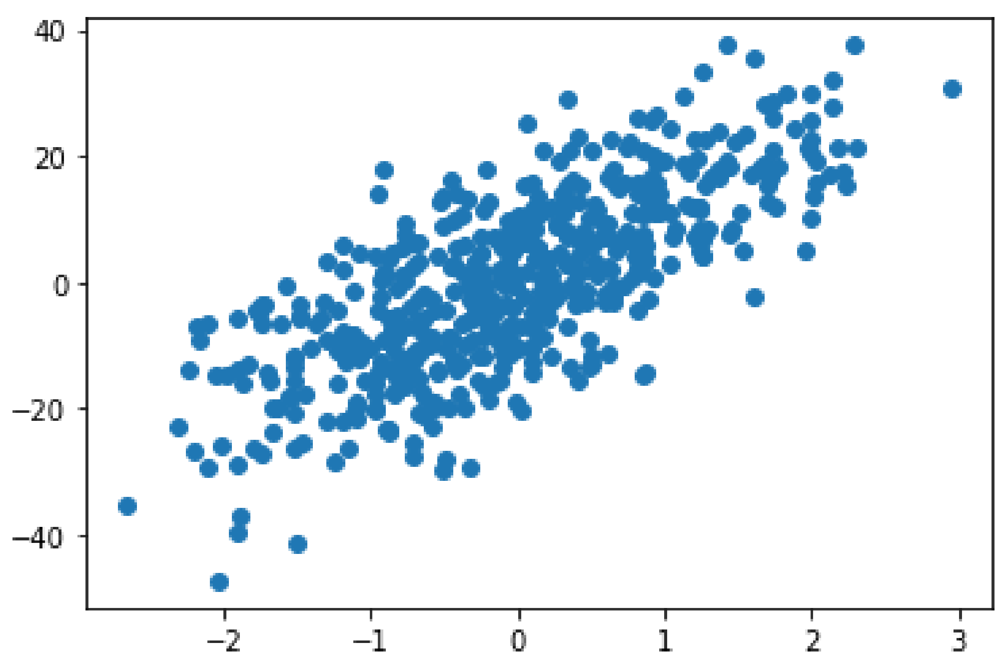
```
for name in datasets.keys():
    accs1 = {}
    accs2 = {}
    for criterion in criteria:
        accs1[criterion] = []
        accs2[criterion] = []
        for max_depth in max_depths:
            clf1 = 
DecisionTreeClassifierFromScratch(max_depth=max_depth, 
criteria=criterion)
            clf2 = DecisionTreeClassifier(max_depth=max_depth, 
criterion=criterion)
            clf1.fit(datasets[name]['x'],datasets[name]['y'])
            clf2.fit(datasets[name]['x'],datasets[name]['y'])
            y_pred1 = clf1.predict(datasets[name]['x'])
            y_pred2 = clf2.predict(datasets[name]['x'])
            acc1 = int(clf1.accuracy(datasets[name]['y'],y_pred1))
            acc2 = int(accuracy_score(datasets[name]
['y'],y_pred2)*100)
            accs1[criterion].append(acc1)
            accs2[criterion].append(acc2)
            #print('accuracy =', acc1,'when max_depth 
=',clf1.max_depth, 'and criteria =',criterion)

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
    ax1.set_xlabel('max_depth')
    ax1.set_ylabel('accuracy')
    ax1.set_title('{} with Gini Splitting'.format(name))
    ax1.plot(max_depths,accs1['gini'],label = 'scratch')
    ax1.plot(max_depths,accs2['gini'],label = 'scikit-learn')
    ax1.legend()
    ax2.set_xlabel('max_depth')
    ax2.set_ylabel('accuracy')
    ax2.set_title('{} with Entropy Splitting'.format(name))
    ax2.plot(max_depths,accs1['entropy'], label = 'scratch')
    ax2.plot(max_depths,accs2['entropy'], label = 'scikit-learn')
    ax2.legend()
    plt.show()
```


moons with Gini Splitting
moons with Entropy Splitting
circles with Gini Splitting
circles with Entropy Splitting
scratch
scikit-learn
accuracy
max_depth



Random two dimensional regression dataset

```
from sklearn.datasets import make_regression
X,y = make_regression(n_samples = 500, n_features = 1, noise =10)
plt.scatter(X,y)
plt.show()
```



Regression Tree from Scratch

```
class DecisionTreeRegressorFromScratch:
    def __init__(self,min_leaf=5):
        self.min_leaf = min_leaf
```

```
    def fit(self, X, y):
        self.dtree = Node(X, y, np.array(np.arange(len(y))), min_leaf
= self.min_leaf)
        return self
    def predict(self, X):
        return self.dtree.predict(X)

class Node:
    def __init__(self, x, y, idxs, min_leaf=5):
        self.x = x
        self.y = y
        self.idxs = idxs
        self.min_leaf = min_leaf
        self.row_count = len(idxs)
        self.col_count = x.shape[1]
        self.val = np.mean(y[idxs])
        self.score = float('inf')
        self.find_varsplit()
    def find_varsplit(self):
        for c in range(self.col_count): self.find_better_split(c)
        if self.is_leaf: return
        x = self.split_col
        lhs = np.nonzero(x <= self.split)[0]
        rhs = np.nonzero(x > self.split)[0]
        self.lhs = Node(self.x, self.y, self.idxs[lhs], self.min_leaf)
        self.rhs = Node(self.x, self.y, self.idxs[rhs], self.min_leaf)
    def find_better_split(self, var_idx):
        x = self.x[self.idxs, var_idx]
        for r in range(self.row_count):
            lhs = x <= x[r]
            rhs = x > x[r]
            if rhs.sum() < self.min_leaf or lhs.sum() < self.min_leaf:
continue
            curr_score = self.find_score(lhs, rhs)
            if curr_score < self.score:
                self.var_idx = var_idx
                self.score = curr_score
                self.split = x[r]
    def find_score(self, lhs, rhs):
        y = self.y[self.idxs]
        lhs_std = y[lhs].std()
        rhs_std = y[rhs].std()
        return lhs_std * lhs.sum() + rhs_std * rhs.sum()
    @property
    def split_col(self): return self.x[self.idxs,self.var_idx]
    @property
    def is_leaf(self): return self.score == float('inf')
    def predict(self, x):
        return np.array([self.predict_row(xi) for xi in x])
    def predict_row(self, xi):
```

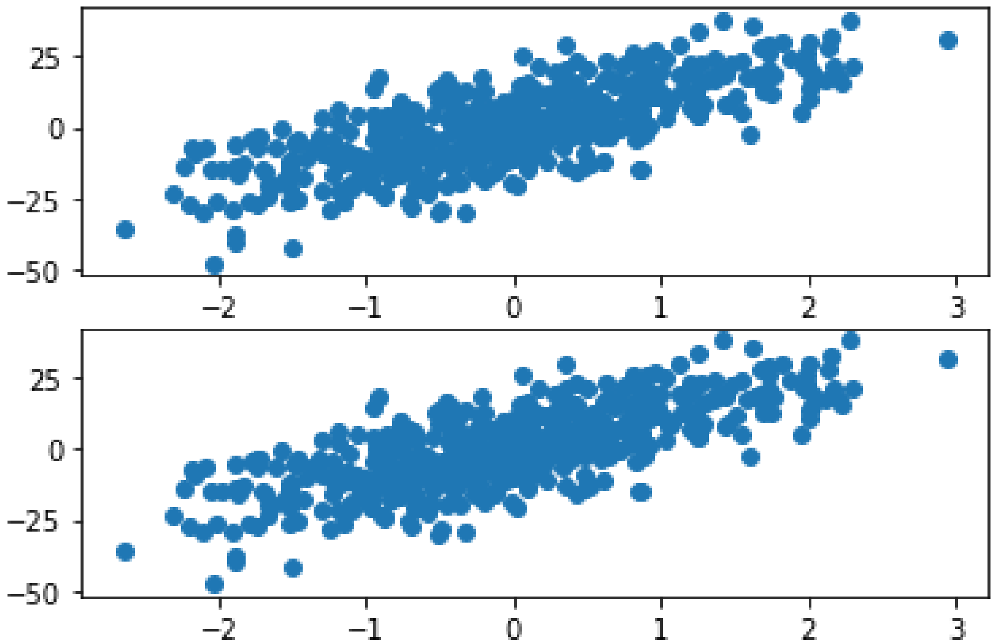
```
        if self.is_leaf: return self.val
        node = self.lhs if xi[self.var_idx] <= self.split else
self.rhs
        return node.predict_row(xi)

def rmse(y1,y2):
    MSE = np.square(np.subtract(y1,y2)).mean()
    RMSE = math.sqrt(MSE)
    return RMSE

import math
from sklearn.metrics import r2_score
min_leaves = [20,15,10,5,1]
for leaf in min_leaves:
    regressor = DecisionTreeRegressorFromScratch(min_leaf =
leaf).fit(X, y)
    y_pred = regressor.predict(X)
    rmse_score = rmse(y_pred,y)
    r2 = r2_score(y_pred,y)
    print('RMSE =',rmse_score,'R2_score=',r2,'when min_leaf =',leaf)

RMSE = 9.133804908792984 R2_score= 0.3000000056004042 when min_leaf =
20
RMSE = 9.075345461637353 R2_score= 0.3150496441717555 when min_leaf =
15
RMSE = 8.794210077732416 R2_score= 0.38262267138496375 when min_leaf =
10
RMSE = 8.081568919037222 R2_score= 0.5242968610338308 when min_leaf =
5
RMSE = 0.0 R2_score= 1.0 when min_leaf = 1
```

Splitting Criteria Used - Reduction in Variance

- **Variance is used for calculating the homogeneity of a node. If a node is entirely homogeneous, then the variance is zero.**
- **So, we calculate the variance of each split by taking a weighted average variance of child nodes and select the split with the least overall vaiance**

Plot the predictions on the training data used to build the tree

```
rmse_score = rmse(y_pred,y)
r2 = r2_score(y_pred,y)
print('RMSE =',rmse_score,'R2_score=',r2)
plt.subplot(2,1,1)
plt.scatter(X,y)
plt.subplot(2,1,2)
plt.scatter(X,y_pred)
plt.show()

RMSE = 0.0 R2_score= 1.0
```




```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(min_samples_leaf = 10).fit(X, y)
y_pred = regressor.predict(X)
rmse_score = rmse(y_pred,y)
r2 = r2_score(y_pred,y)
print('RMSE =',rmse_score,'R2_score=',r2)
```

```
RMSE = 8.733242389329464 R2_score= 0.39630304121349447
```

```
min_leaves = [20,15,10,5,1]
reg1_rmse = []
reg2_rmse = []
reg1_r2 = []
reg2_r2 = []
for leaf in min_leaves:
    regressor1 = DecisionTreeRegressorFromScratch(min_leaf =
leaf).fit(X, y)
    regressor2 = DecisionTreeRegressor(min_samples_leaf = leaf).fit(X,
y)
    y_pred1 = regressor1.predict(X)
    y_pred2 = regressor2.predict(X)
    rmse_score1 = rmse(y_pred1,y)
    reg1_rmse.append(rmse_score1)
    r2_1 = r2_score(y_pred1,y)
    reg1_r2.append(r2_1)
    rmse_score2 = rmse(y_pred2,y)
    reg2_rmse.append(rmse_score2)
    r2_2 = r2_score(y_pred2,y)
    reg2_r2.append(r2_2)
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
ax1.set_xlabel('min_leaves')
ax1.set_ylabel('RMSE')
ax1.plot(min_leaves,reg1_rmse,label = 'scratch')
ax1.plot(min_leaves,reg2_rmse,label = 'scikit-learn')
ax1.legend()
ax2.set_xlabel('min_leaves')
ax2.set_ylabel('R2 Score')
ax2.plot(min_leaves,reg1_r2,label = 'scratch')
ax2.plot(min_leaves,reg2_r2,label = 'scikit-learn')
ax2.legend()
plt.show()
```