Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Звіт № 2 з
дисципліни
«Програмування інтелектуальних
інформаційних систем»

Виконав студент ІП-13 Романюк Діана Олексіївна
(шифр, прізвище, ім'я, по батькові)

Перевірив Баришич Лука Маріянович
(прізвище, ім'я, по батькові)

Київ 2023

Лабораторна робота 2

Постановка задачі:

1. Dataset1: /kaggle/input/adult-dataset/adult.csv'

Bayesian Classification + Support Vector Machine

Зробити предікшн двома вищезгаданими алгоритмами. Порівняти наступні метрики: Recall, f1-score, Confusion matrix, accuracy score. Порівняти з нуль-гіпотезою і перевірити на оверфітинг. Пояснити результати.

2. Dataset2: https://www.kaggle.com/code/stieranka/k-nearest-neighbors

K nearest neighbours.

Те саме що і в 1 завданні, але порівнюємо між собою метрики. Euclidean, Manhattan, Minkowski. Кластери потрібно візуалізувати. Метрики аналогічно п.1

3. Dataset3: https://www.kaggle.com/code/nuhashafnan/cluster-analysis-kmeans-kmediod-agnes-birch-dbscan

Agnes,Birch,DBSCAN

Інші методи можна ігнорувати. Зняти метрики (Silhouette Coefficient, ARI, NMI. Можна з п.1-2), пояснити.

4. Dataset4: https://www.kaggle.com/code/datark1/customers-clustering-k-means-dbscan-and-ap

Affinity propagation.

Порівняти з k-means. Метрики - Silhouette Coefficient, ARI, NMI

Task 1: Bayesian Classification + Support Vector Machine

In [6]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization purposes
import seaborn as sns # for statistical data visualization
```

In [7]:

```
from google.colab import files

uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving (task1)_adult.csv to (task1)_adult (1).csv
```

In [8]:

```
for filename in uploaded.keys():
    print(f'Uploaded file: {filename}')
```

```
Uploaded file: (task1)_adult (1).csv
```

In [86]:

```
df = pd.read_csv('(task1)_adult.csv', header=None)
```

In [10]:

```
df.shape
```

Out[10]:

```
(32561, 15)
```

In [11]:

```
df.head()
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

In [12]:

```
col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status', 'occupation', 'relationship',
             'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income']

df.columns = col_names

df.columns
```

Out[12]:

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income'],
      dtype='object')
```

In [13]:

```
df.head()
```

Out[13]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

In [14]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education_num   32561 non-null  int64 
 5   marital_status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital_gain    32561 non-null  int64 
 11  capital_loss    32561 non-null  int64 
 12  hours_per_week  32561 non-null  int64 
 13  native_country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [15]:

```
categorical = [var for var in df.columns if df[var].dtype=='O']
```

```
print('There are {} categorical variables\n'.format(len(categorical)))
print('The categorical variables are :\n\n', categorical)
There are 9 categorical variables

The categorical variables are :

 ['workclass', 'education', 'marital_status', 'occupation', 'relationship', 'race'
, 'sex', 'native_country', 'income']
```

In [16]:

```
df[categorical].head()
```

Out[16]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	<=50K
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	<=50K
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=50K
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=50K
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=50K

In [53]:

```
# encode remaining variables with one-hot encoding

encoder = ce.OneHotEncoder(cols=['workclass', 'education', 'marital_status', 'occu
pation', 'relationship',
                                 'race', 'sex', 'native_country'])

X_train = encoder.fit_transform(X_train)

X_test = encoder.transform(X_test)
```

In [54]:

```
cols = X_train.columns
```

In [55]:

```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

In [56]:

```
X_train = pd.DataFrame(X_train, columns=[cols])
```

In [57]:

```
X_test = pd.DataFrame(X_test, columns=[cols])
```

Naive Bayes Classifier

In [58]:

```
# train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB


# instantiate the model
gnb = GaussianNB()


# fit the model
gnb.fit(X_train, y_train)
```

Out[58]:

```
GaussianNB
GaussianNB()
```

In [59]:

```
y_pred = gnb.predict(X_test)

y_pred
```

Out[59]:

```
array(['<=50K', '<=50K', '>50K', ..., '>50K', '<=50K', '<=50K'],
      dtype='<U5')
```

Accuracy score

In [60]:

```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
Model accuracy score: 0.8083
```

Check for overfitting

In [62]:

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(gnb.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(gnb.score(X_test, y_test)))
Training set score: 0.8067
Test set score: 0.8083
```

The training-set accuracy score is 0.8067 while the test-set accuracy to be 0.8083. These two values are quite comparable. So, there is **no sign of overfitting**.

Compare with null-accuracy

In [63]:

```
# check class distribution in test set

y_test.value_counts()
```

Out[63]:

```
<=50K    7407
>50K     2362
Name: income, dtype: int64
```

In [64]:

```
# check null accuracy score
```

```
null_accuracy = (7407/(7407+2362))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
Null accuracy score: 0.7582
```

We can see that our model accuracy score is 0.8083 but null accuracy score is 0.7582. So, we can conclude that our Gaussian Naive **Bayes Classification model is doing a very good job** in predicting the class labels

Confusion matrix

In [65]:

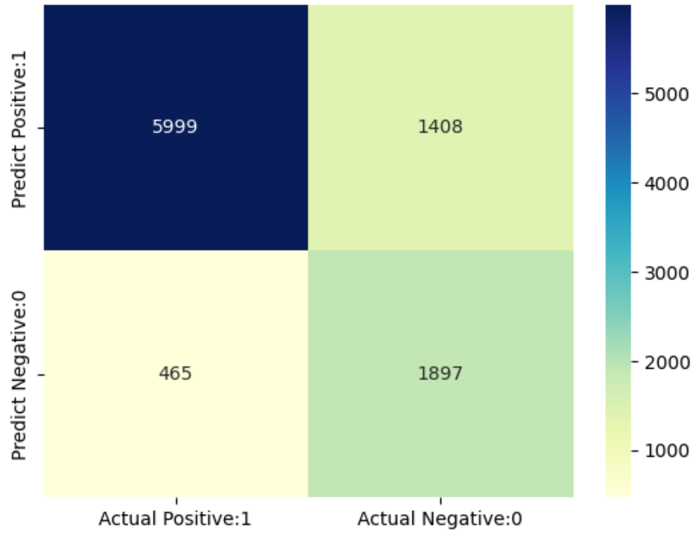
```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
Confusion matrix

 [[5999 1408]
 [ 465 1897]]

True Positives(TP) =  5999

True Negatives(TN) =  1897

False Positives(FP) =  1408

False Negatives(FN) =  465
```

In [66]:

```
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0
'],
                                 index=['Predict Positive:1', 'Predict Negative:0'
])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[66]:

```
<Axes: >
```



Recall and f1-score

In [67]:

```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

In [68]:

```
# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))
```

Precision : 0.8099

Recall

In [69]:

```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

Recall or Sensitivity : 0.9281

f1-score

In [70]:

```
def calculate_f1_score(prec, rec):
    if [prec] + rec == 0:
        return 0
    f1_score = 2 * (prec * rec) / (prec + rec)
    return round(f1_score, 4)
f1_score_result = calculate_f1_score(precision, recall)
print(f'f1-score: {f1_score_result}')
```

f1-score: 0.865

Support Vector Machine

In [71]:

```
from sklearn import svm
```

In [72]:

```
svm = svm.SVC()
svm.fit(X_train, y_train)
```

Out[72]:

```
SVC
SVC()
```

In [73]:

```
y_pred1=svm.predict(X_test)
y_pred1
```

Out[73]:

```
array(['<=50K', '<=50K', '<=50K', ..., '>50K', '<=50K', '<=50K'],
      dtype=object)
```

Accuracy score

In [74]:

```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred1)))
Model accuracy score: 0.8029
```

Check for overfitting

In [75]:

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(svm.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(svm.score(X_test, y_test)))
Training set score: 0.8021
Test set score: 0.8029
```

The training-set accuracy score is 0.8021 while the test-set accuracy to be 0.8029. These two values are quite comparable. So, there is **no sign of overfitting**.

Compare with null-accuracy

In [76]:

```
# check class distribution in test set

y_test.value_counts()
```

Out[76]:

```
<=50K    7407
>50K     2362
Name: income, dtype: int64
```

In [77]:

```
# check null accuracy score

null_accuracy = (7407/(7407+2362))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
Null accuracy score: 0.7582
```

We can see that our model accuracy score is 0.8029 but null accuracy score is 0.7582. So, we can conclude that our Gaussian Naive **Bayes Classification model is doing a very good job** in predicting the class labels

Confusion matrix

In [78]:

```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred1)

print('Confusion matrix\n\n', cm)
```

```
print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

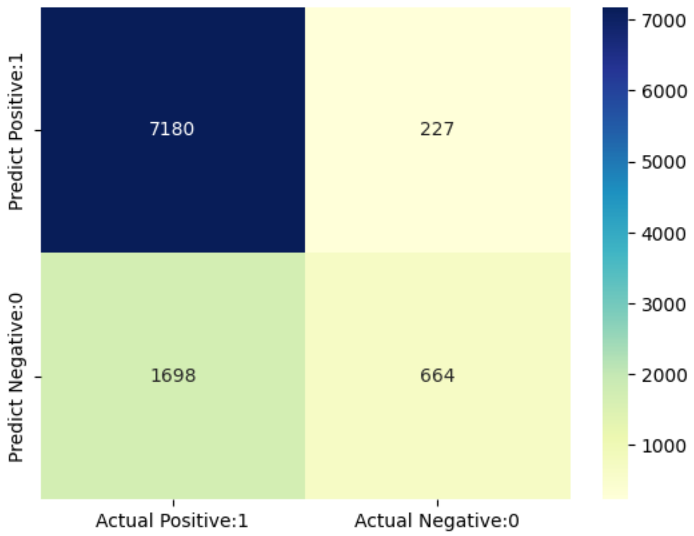
```
Confusion matrix

 [[7180  227]
 [1698  664]]

True Positives(TP) =  7180

True Negatives(TN) =  664

False Positives(FP) =  227

False Negatives(FN) =  1698
```

In [79]:

```
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[79]:

```
<Axes: >
```



Recall and f1-score

In [80]:

```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

In [81]:

```
# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))
Precision : 0.9694
```

Recall

In [82]:

```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
Recall or Sensitivity : 0.8087
```

f1-score

In [83]:

```
def calculate_f1_score(prec, rec):
    if [prec] + rec == 0:
        return 0
    f1_score = 2 * (prec * rec) / (prec + rec)
    return f1_score
f1_score_result = calculate_f1_score(precision, recall)
print(f'f1-score: {f1_score_result}')
f1-score: 0.8817930610991711
```

	NB	SVM
recall	0.9281	0.8087
f1-score	0.8650	0.8818
conf. matrix	5999 1408 465 1897	7180 227 1698 664
accuracy score	0.8083	0.8029
With null accuracy	+	+
overfitting	-	-

In [ ]:

Task 2: K nearest neighbours

In [ ]:

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import matplotlib.ticker as ticker
from sklearn import preprocessing
```

In [ ]:

```
from google.colab import files
```

```
uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving teleCust1000t.csv to teleCust1000t (1).csv
```

In [ ]:

```
df = pd.read_csv('teleCust1000t.csv', sep=',')
```

In [ ]:

```
df.head()
```

Out[ ]:

	region	tenure	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	2	13	44	1	9	64.0	4	5	0.0	0	2	1
1	3	11	33	1	7	136.0	5	5	0.0	0	6	4
2	3	68	52	1	24	116.0	1	29	0.0	1	2	3
3	2	33	33	0	12	33.0	2	0	0.0	1	1	1
4	2	23	30	1	9	30.0	1	2	0.0	0	4	3

In [ ]:

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

Out[ ]:

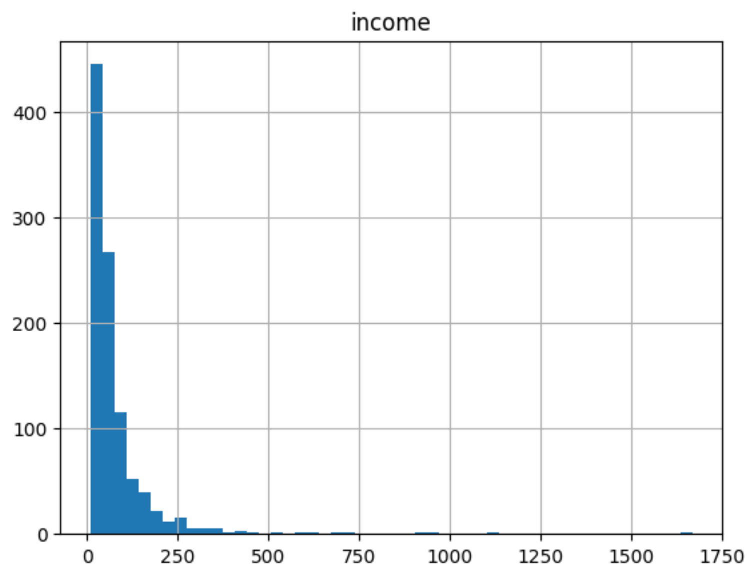
```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

In [ ]:

```
df.hist(column='income', bins = 50)
```

Out[ ]:

```
array([[<Axes: title={'center': 'income'}>]], dtype=object)
```



In [ ]:

```
df.columns
```

Out[ ]:

```
Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
       'employ', 'retire', 'gender', 'reside', 'custcat'],
      dtype='object')
```

In [ ]:

```
X = df[['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
       'employ', 'retire', 'gender', 'reside']].values
X[0:5]
```

Out[ ]:

```
array([[  2.,  13.,  44.,   1.,   9.,  64.,   4.,   5.,   0.,   0.,   2.],
       [  3.,  11.,  33.,   1.,   7., 136.,   5.,   5.,   0.,   0.,   6.],
       [  3.,  68.,  52.,   1.,  24., 116.,   1.,  29.,   0.,   1.,   2.],
       [  2.,  33.,  33.,   0.,  12.,  33.,   2.,   0.,   0.,   1.,   1.],
       [  2.,  23.,  30.,   1.,   9.,  30.,   1.,   2.,   0.,   0.,   4.]])
```

In [ ]:

```
y = df['custcat'].values
y[0:5]
```

Out[ ]:

```
array([1, 4, 3, 1, 3])
```

In [ ]:

```
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
X[0:5]
```

Out[ ]:

```
array([[-0.02696767, -1.055125  ,  0.18450456,  1.0100505 , -0.25303431,
        -0.12650641,  1.0877526 , -0.5941226 , -0.22207644, -1.03459817,
        -0.23065004],
       [ 1.19883553, -1.14880563, -0.69181243,  1.0100505 , -0.4514148 ,
         0.54644972,  1.9062271 , -0.5941226 , -0.22207644, -1.03459817,
         2.55666158],
       [ 1.19883553,  1.52109247,  0.82182601,  1.0100505 ,  1.23481934,
         0.35951747, -1.36767088,  1.78752803, -0.22207644,  0.96655883,
        -0.23065004],
       [-0.02696767, -0.11831864, -0.69181243, -0.9900495 ,  0.04453642,
        -0.41625141, -0.54919639, -1.09029981, -0.22207644,  0.96655883,
        -0.92747794],
       [-0.02696767, -0.58672182, -0.93080797,  1.0100505 , -0.25303431,
        -0.44429125, -1.36767088, -0.89182893, -0.22207644, -1.03459817,
         1.16300577]])
```

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_s
tate=4)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
Train set: (800, 11) (800,)
Test set: (200, 11) (200,)
```

K nearest neighbor (K-NN)

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:

```
k = 4
```

```
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

Out[ ]:

```
KNeighborsClassifier
 KNeighborsClassifier(n_neighbors=4)
```

In [ ]:

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

Out[ ]:

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

Accuracy

In [ ]:

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_trai
n)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
Train set Accuracy:  0.5475
Test set Accuracy:  0.32
```

In [ ]:

Vizualization

In [ ]:

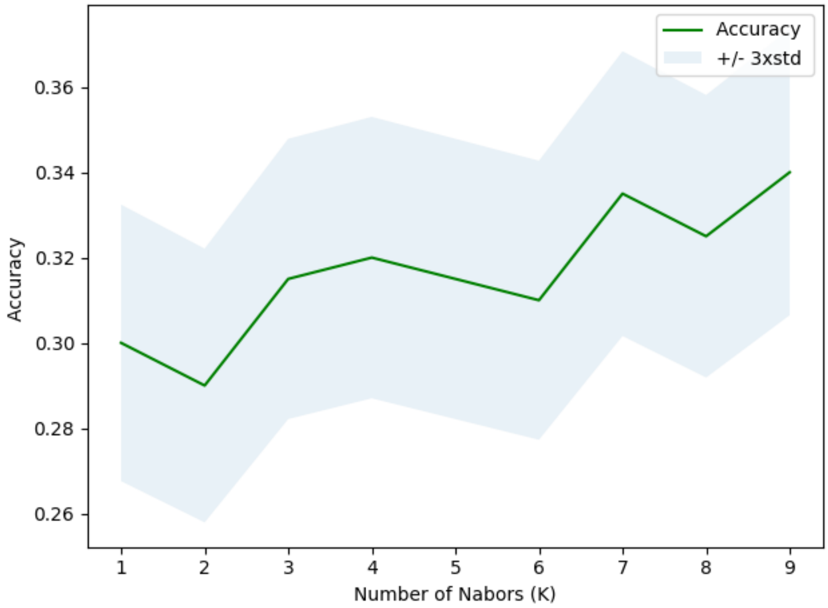
```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):

    #Train Model and Predict
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_train)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)


    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc
```

Out[ ]:

```
array([0.3  , 0.29 , 0.315, 0.32 , 0.315, 0.31 , 0.335, 0.325, 0.34 ])
```

In [ ]:

```
plt.plot(range(1,Ks),mean_acc,'g')
plt.fill_between(range(1,Ks),mean_acc - 1 * std_acc,mean_acc + 1 * std_acc, alpha=
0.10)
plt.legend(('Accuracy ', '+/- 3xstd'))
plt.ylabel('Accuracy ')
plt.xlabel('Number of Nabors (K)')
plt.tight_layout()
plt.show()
```



In [ ]:

```
print( "The best accuracy was with", mean_acc.max(), "with k=", mean_acc.argmax()+
1)
```

The best accuracy was with 0.34 with k= 9

In [ ]:

```
from sklearn import metrics
```

In [ ]:

```
import seaborn as sns

def plot_confusion_matrix(conf_matrix, classes, metric):
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=classe
s, yticklabels=classes)
    plt.title(f'{metric} Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
```

In [ ]:

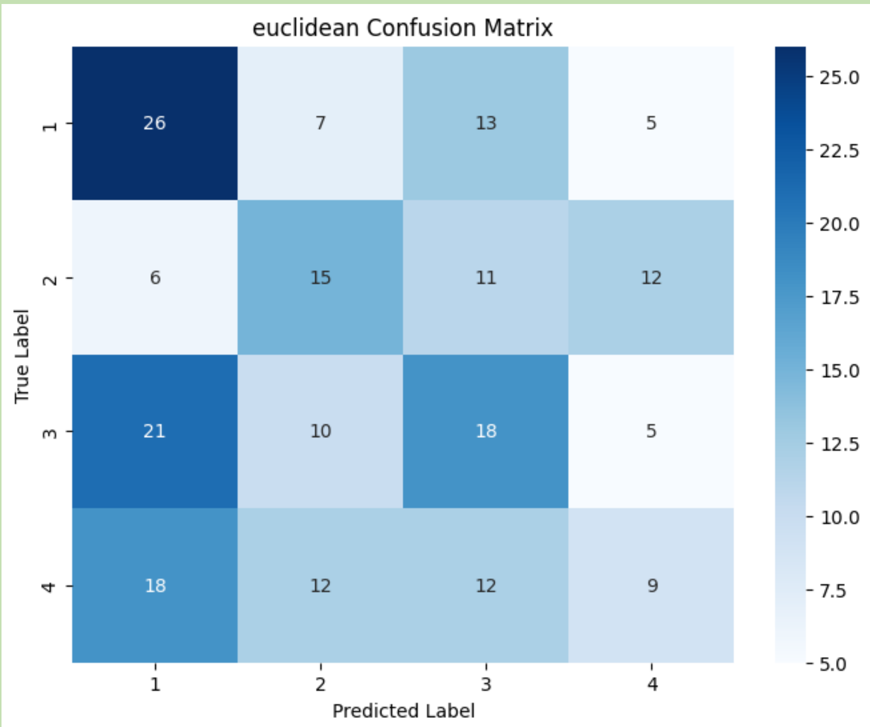
```
def evaluate_knn(metric):
    knn = KNeighborsClassifier(n_neighbors=9, metric=metric)
    knn.fit(X_train, y_train)
    yhat = knn.predict(X_test)

    # Accuracy
    accuracy = metrics.accuracy_score(y_test, yhat)
    print(f'{metric} Accuracy:', metrics.accuracy_score(y_test, yhat))

    #Check for overfittig
    print(f'{metric} Train accuracy:', metrics.accuracy_score(y_train, knn.predict
(X_train)))
    print(f'{metric} Test accuracy:', metrics.accuracy_score(y_test, yhat))

    #null
    unique_elements, counts = np.unique(y_test, return_counts=True)
    majority_class_count = counts.max()
    null_accuracy = majority_class_count / len(y_test)
```

```
    print(f'null Accuracy: {null_accuracy}')

    # Confusion Matrix
    conf_matrix = metrics.confusion_matrix(y_test, yhat)
    print(f'{metric} Confusion Matrix:')
    print(conf_matrix)

    # Visualize Confusion Matrix
    plot_confusion_matrix(conf_matrix, classes=np.unique(y_test), metric=metric)

    # Recall
    recall = metrics.recall_score(y_test, yhat, average='weighted')
    print(f'{metric} Recall:', recall)

    # F1 Score
    f1 = metrics.f1_score(y_test, yhat, average='weighted')
    print(f'{metric} F1 Score:', f1)
```

In [ ]:

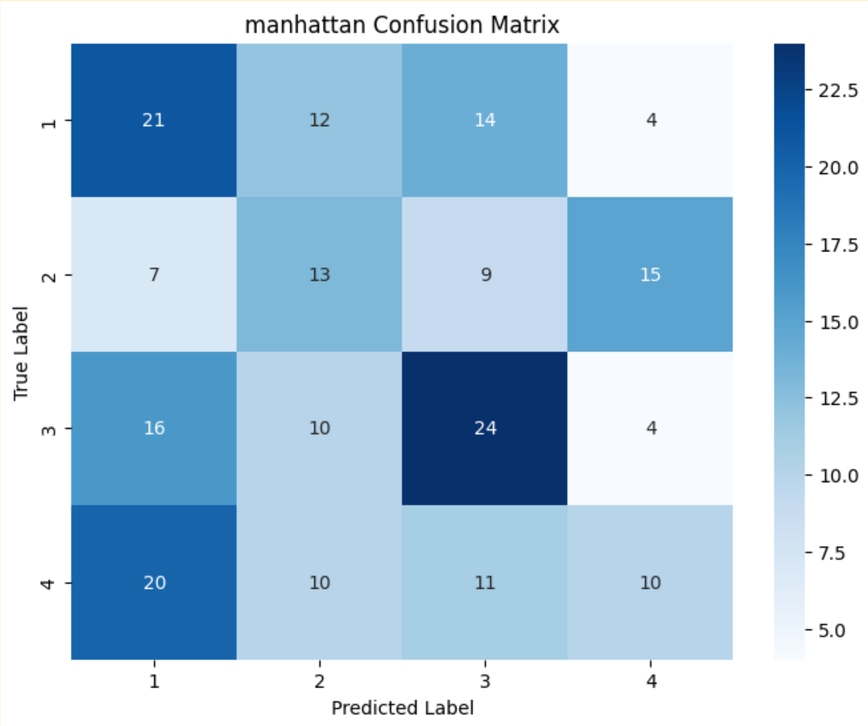
```
distance_metrics = ['euclidean', 'manhattan', 'minkowski']

for metric in distance_metrics:
    evaluate_knn(metric)
```

```
euclidean Accuracy: 0.34
euclidean Train accuracy: 0.5025
euclidean Test accuracy: 0.34
null Accuracy: 0.27
euclidean Confusion Matrix:
[[26  7 13  5]
 [ 6 15 11 12]
 [21 10 18  5]
 [18 12 12  9]]
```

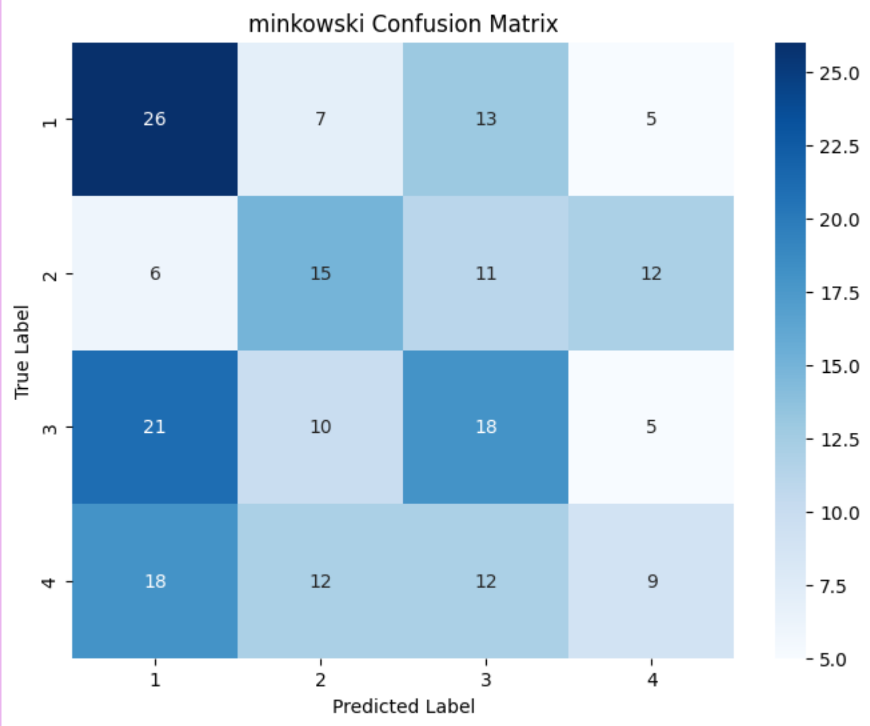



```
euclidean Recall: 0.34
euclidean F1 Score: 0.3296641343462616
```

```
manhattan Accuracy: 0.34
manhattan Train accuracy: 0.4875
manhattan Test accuracy: 0.34
null Accuracy: 0.27
manhattan Confusion Matrix:
[[21 12 14  4]
 [ 7 13  9 15]
 [16 10 24  4]
 [20 10 11 10]]
```



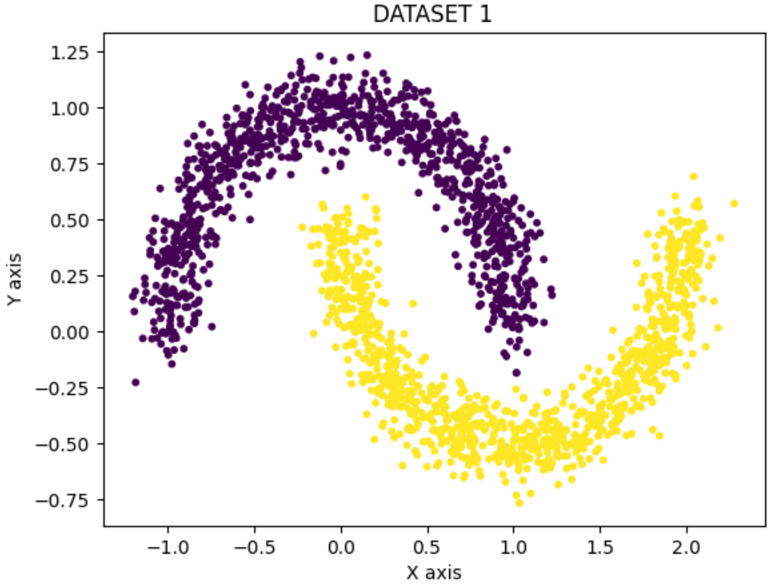

```
manhattan Recall: 0.34
manhattan F1 Score: 0.3338286691325284
minkowski Accuracy: 0.34
minkowski Train accuracy: 0.5025
minkowski Test accuracy: 0.34
null Accuracy: 0.27
minkowski Confusion Matrix:
[[26  7 13  5]
 [ 6 15 11 12]
 [21 10 18  5]
 [18 12 12  9]]
```



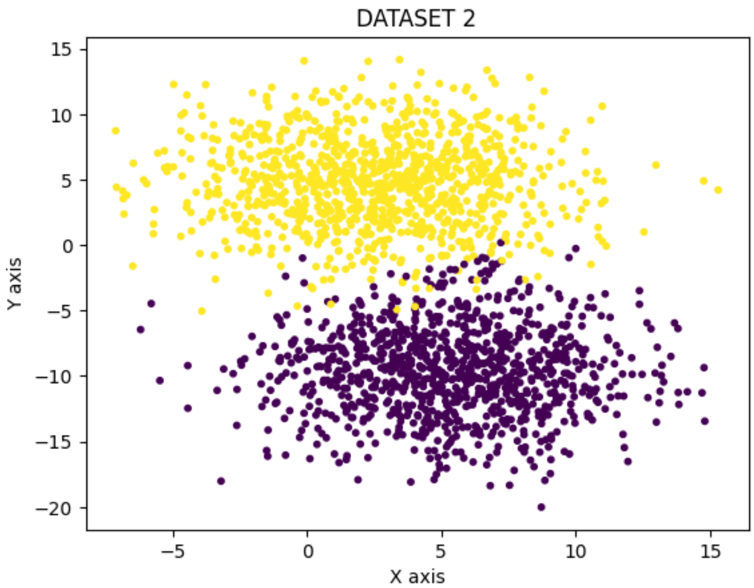

```
minkowski Recall: 0.34
minkowski F1 Score: 0.3296641343462616
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt #Data Visualization
import seaborn as sns  #Python library for Vidualization
import os
np.random.seed(10)

from sklearn import cluster, datasets, mixture
X1,Y1 = datasets.make_moons(n_samples=2000, noise=.09,random_state=10)
#plt.scatter(X1[:, 0], X1[:, 1], marker='o', c=Y1,s=25, edgecolor='r')
print(X1.shape)
print(Y1.shape)
plt.scatter(X1[:, 0], X1[:, 1], s=10, c=Y1)
plt.title('DATASET 1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
#plt.savefig('Dataset1')

plt.show()

from sklearn.datasets import make_blobs
X3,Y3  = make_blobs(n_samples=2000,cluster_std=3.5,centers=2, n_features=2,random_
state=10)
print(X3.shape)
print(Y3.shape)
plt.title('DATASET 2')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.scatter(X3[:, 0], X3[:, 1], s=10, c=Y3)
#plt.savefig('Dataset2')
```

```
plt.show()
(2000, 2)
(2000,)
```




```
(2000, 2)
(2000,)
```



Task 3

In [5]:

```
from sklearn.cluster import KMeans
from sklearn.cluster import Birch
from sklearn.cluster import AgglomerativeClustering
```

Agnes,Birch,DBSCAN Інші методи можна ігнорувати. Зняти метрики (Silhouette Coefficient, ARI, NMI. Можна з п.1-2), пояснити.

In [10]:

```
#Model Build dataset1
kmeansmodel = KMeans(n_clusters= 2, init='k-means++',max_iter=1000,random_state=10
)
y_kmeans= kmeansmodel.fit_predict(X1)

birchmodel=Birch(n_clusters=2,threshold=0.5,branching_factor=100)
```

```
y_birch=birchmodel.fit_predict(X1)
print(y_birch.shape)

agnesmodel = AgglomerativeClustering(n_clusters=2)
y_agnes=birchmodel.fit_predict(X1)
print(y_agnes.shape)
```

```
(2000,)
(2000,)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
ing: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  warnings.warn(
```

In [7]:

```
#Model Build dataset 2
kmeansmodel2 = KMeans(n_clusters= 2, init='k-means++',max_iter=1000,random_state=1
0)
y_kmeans2= kmeansmodel2.fit_predict(X3)

birchmodel2=Birch(n_clusters=2,threshold=0.1,branching_factor=100)
y_birch2=birchmodel2.fit_predict(X3)
print(y_birch2.shape)

agnesmodel2 = AgglomerativeClustering(n_clusters=2)
y_agnes2=agnesmodel2.fit_predict(X3)
print(y_agnes2.shape)
```

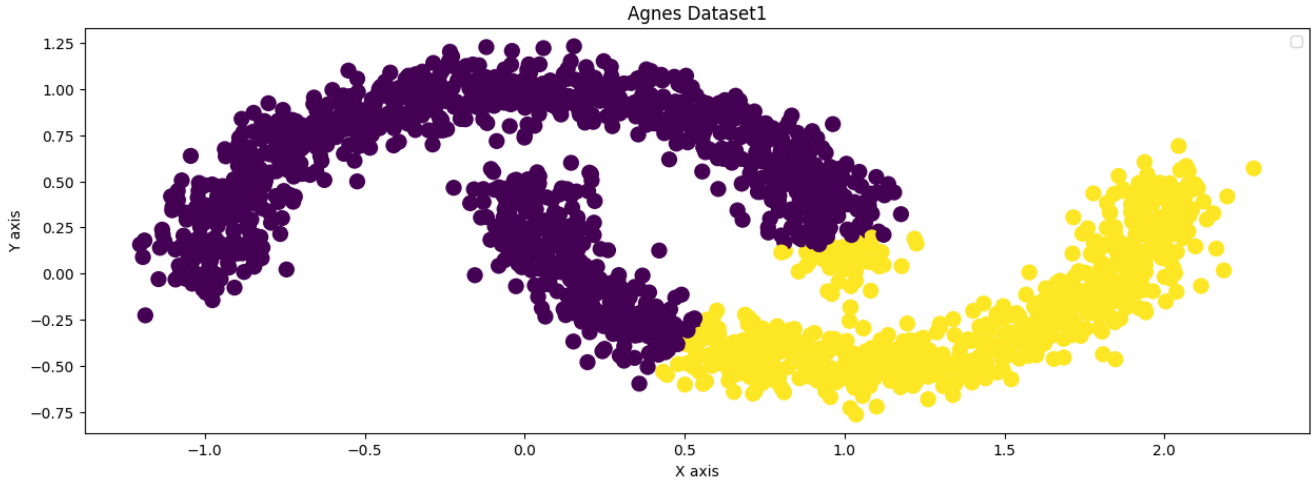
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
ing: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  warnings.warn(
(2000,)
(2000,)
```

Agnes

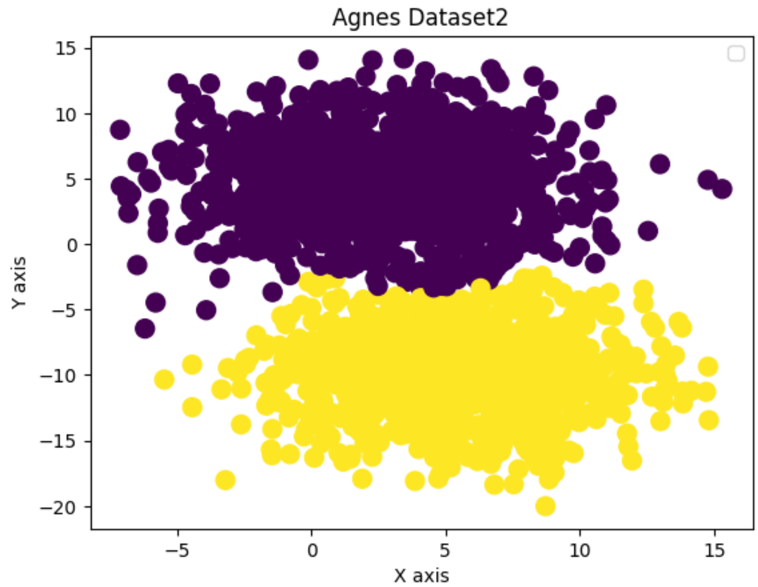
In [11]:

```
plt.figure(figsize=(15,5))
#plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_agnes)
plt.title('Agnes Dataset1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend()
#plt.savefig('Kmeansd1',dpi=300)
plt.show()

#plt.subplot(1,2,2)
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y_agnes2)
plt.title('Agnes Dataset2')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend()
#plt.savefig('Kmeansd1d2',dpi=300)
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
 with no argument.
```




```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
 with no argument.
```



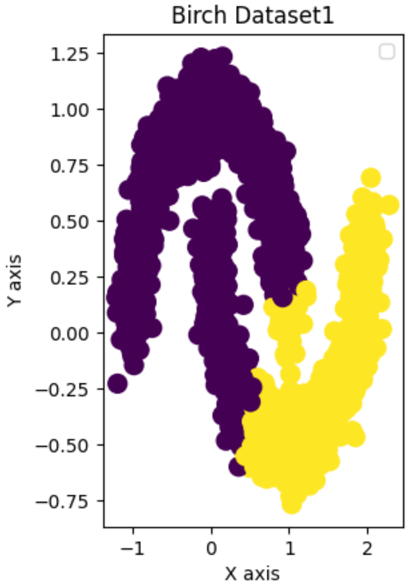
BIRCH

In [12]:

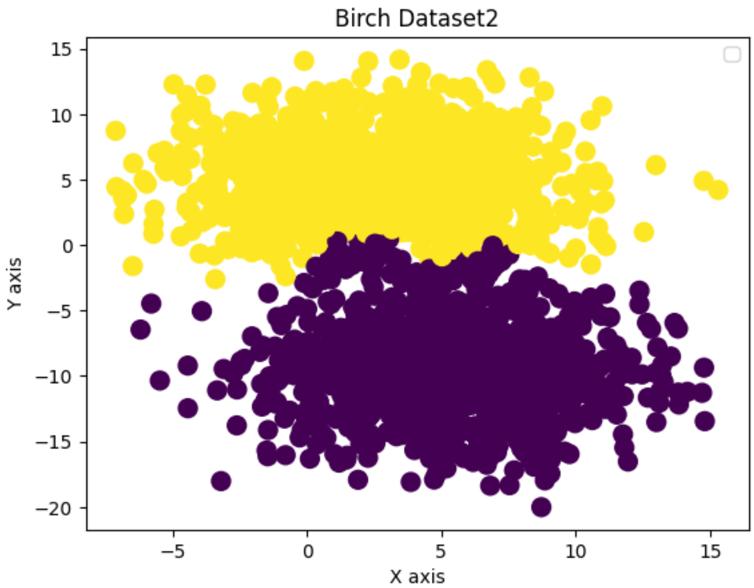
```
#plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_birch)
plt.title('Birch Dataset1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend()
#plt.savefig('Kmeansd1',dpi=300)
plt.show()

#plt.subplot(1,2,2)
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y_birch2)
plt.title('Birch Dataset2')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend()
```

```
#plt.savefig('birchd1d2',dpi=300)
plt.show()
```

WARNING:matplotlib.legend:No artists with labels found to put in legend. Note tha
t artists whose label start with an underscore are ignored when legend() is called
 with no argument.



WARNING:matplotlib.legend:No artists with labels found to put in legend. Note tha
t artists whose label start with an underscore are ignored when legend() is called
 with no argument.



DBSCAN

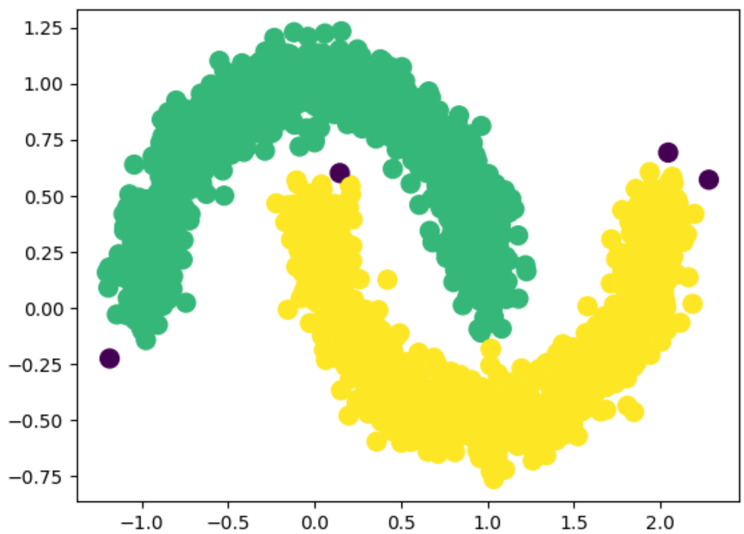
In [14]:

```
def MyDBSCAN(D, eps, MinPts):
    labels = [0]*len(D)
    C = 0
    for P in range(0, len(D)):
        if not (labels[P] == 0):
           continue
        NeighborPts = regionQuery(D, P, eps)
        if len(NeighborPts) < MinPts:
            labels[P] = -1
        else:
           C += 1
```

```
            growCluster(D, labels, P, NeighborPts, C, eps, MinPts)

    return labels

def growCluster(D, labels, P, NeighborPts, C, eps, MinPts):
    labels[P] = C
    i = 0
    while i < len(NeighborPts):
        Pn = NeighborPts[i]
        if labels[Pn] == -1:
           labels[Pn] = C
        elif labels[Pn] == 0:
            labels[Pn] = C
            PnNeighborPts = regionQuery(D, Pn, eps)
            if len(PnNeighborPts) >= MinPts:
                NeighborPts = NeighborPts + PnNeighborPts
        i += 1

def regionQuery(D, P, eps):

    neighbors = []

    for Pn in range(0, len(D)):

        if np.linalg.norm(D[P] - D[Pn]) < eps:
           neighbors.append(Pn)

    return neighbors
```

In [15]:

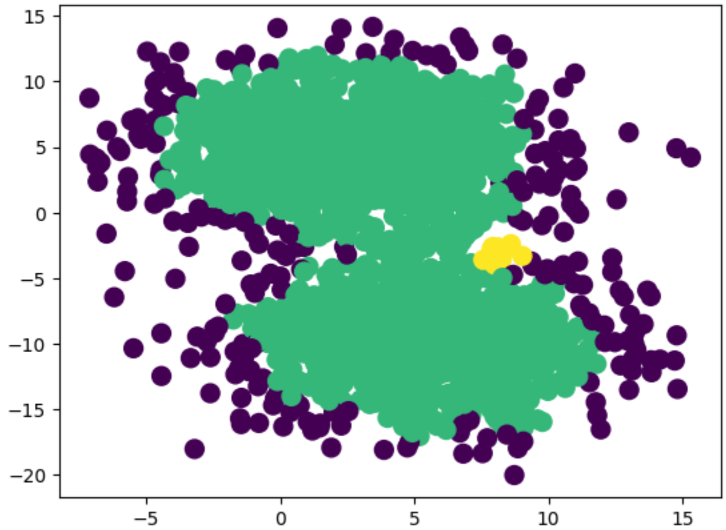
```
dbscan_labels1=MyDBSCAN(X1, .2, 70)
#plt.figure(figsize=(10,8))
#plt.subplot(1, 2, 5)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=dbscan_labels1)
plt.show()
inti_point = np.random.randint(0, len(X1)-1, 2 )
medoids=X1[inti_point]
```



In [16]:

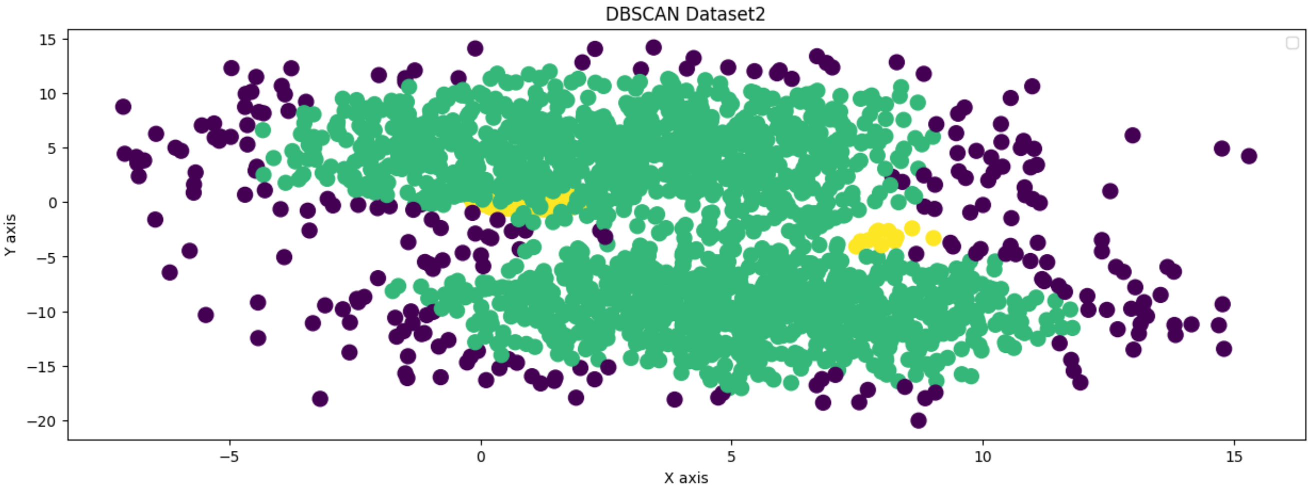
```
dbscan_labels2=MyDBSCAN(X3,1,10)
```

```
#plt.figure(figsize=(10,8))
#plt.subplot(1, 2, 5)
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=dbscan_labels2)
plt.show()
inti_point = np.random.randint(0, len(X3)-1, 2 )
medoids=X3[inti_point]
```



In [17]:

```
plt.figure(figsize=(15,5))
#plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=dbscan_labels1)
plt.title('DBSCAN Dataset1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend()
#plt.savefig('Kmeansd1',dpi=300)
#plt.show()

#plt.subplot(1,2,2)
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=dbscan_labels2)
plt.title('DBSCAN Dataset2')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend()
#plt.savefig('dbscand1d2',dpi=300)
plt.show()
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
 with no argument.
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note tha
t artists whose label start with an underscore are ignored when legend() is called
 with no argument.
```



METRICS

In [18]:

```
#ARI
#NMI
#Silhouette Coefficient
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.metrics.cluster import normalized_mutual_info_score
from sklearn.metrics import silhouette_samples, silhouette_score
```

ARI

In [22]:

```
ari_birch=adjusted_rand_score(Y1,y_birch)
ari_dbscan=adjusted_rand_score(Y1,dbscan_labels1)
ari_agnes=adjusted_rand_score(Y1,y_agnes)

print("DATASET1:")
print("ARI of Birch :"+ str(ari_birch))
print("ARI of Dbscan: "+ str(ari_dbscan))
print("ARI of Agnes: "+ str(ari_agnes))

ari_birch=adjusted_rand_score(Y3,y_birch2)
ari_dbscan=adjusted_rand_score(Y3,dbscan_labels2)
ari_agnes=adjusted_rand_score(Y3,y_agnes2)

print("DATASET2:")
print("ARI of Birch :"+ str(ari_birch))
print("ARI of Dbscan: "+ str(ari_dbscan))
print("ARI of Agnes: "+ str(ari_agnes))
```

```
DATASET1:
ARI of Birch :0.3767076067566142
ARI of Dbscan: 0.9920149895714532
ARI of Agnes: 0.3767076067566142
DATASET2:
ARI of Birch :0.872292314560211
ARI of Dbscan: -0.00016436623002222448
ARI of Agnes: 0.90816307882887
```

NMI

In [24]:

```
nmi_birch=normalized_mutual_info_score(Y1,y_birch)
nmi_dbscan=normalized_mutual_info_score(Y1,dbscan_labels1)
nmi_agnes=normalized_mutual_info_score(Y1,y_agnes)
```

```python
print("DATASET1:")
print("NMI of Birch :"+ str(nmi_birch))
print("NMI of Dbscan: "+ str(nmi_dbscan))
print("NMI of Agnes: "+ str(nmi_agnes))

nmi_birch=normalized_mutual_info_score(Y3,y_birch2)
nmi_dbscan=normalized_mutual_info_score(Y3,dbscan_labels2)
nmi_agnes=normalized_mutual_info_score(Y3,y_agnes2)
print("DATASET2:")
print("NMI of Birch :"+ str(nmi_birch))
print("NMI of Dbscan: "+ str(nmi_dbscan))
print("NMI of Agnes: "+ str(nmi_agnes))
```

```
DATASET1:
NMI of Birch :0.341366173543779
NMI of Dbscan: 0.9787649300611727
NMI of Agnes: 0.341366173543779
DATASET2:
NMI of Birch :0.8102453395167878
NMI of Dbscan: 0.0022364550336834766
NMI of Agnes: 0.8427393441408568
```

Silhouette Coefficient

In [25]:

```python
sil_birch=silhouette_score(X1,y_birch)
sil_dbscan=silhouette_score(X1,dbscan_labels1)
sil_agnes=silhouette_score(X1,y_agnes)

print("Dataset1:")
print("Silhouette Coefficient with Birch :"+ str(sil_birch))
print("Silhouette Coefficient with Dbscan : "+ str(sil_dbscan))
print("Silhouette Coefficient with Agnes : "+ str(sil_agnes))

sil_birch=silhouette_score(X3,y_birch2)
sil_dbscan=silhouette_score(X3,dbscan_labels2)
sil_agnes=silhouette_score(X3,y_agnes2)

print("Dataset2:")
print("Silhouette Coefficient with Birch :"+ str(sil_birch))
print("Silhouette Coefficient with Dbscan : "+ str(sil_dbscan))
print("Silhouette Coefficient with Agnes : "+ str(sil_agnes))
```

```
Dataset1:
Silhouette Coefficient with Birch :0.45835031870569487
Silhouette Coefficient with Dbscan : 0.3010813290557993
Silhouette Coefficient with Agnes : 0.45835031870569487
Dataset2:
Silhouette Coefficient with Birch :0.5760880178842558
Silhouette Coefficient with Dbscan : -0.16260384238870432
Silhouette Coefficient with Agnes : 0.5878339188420266
```

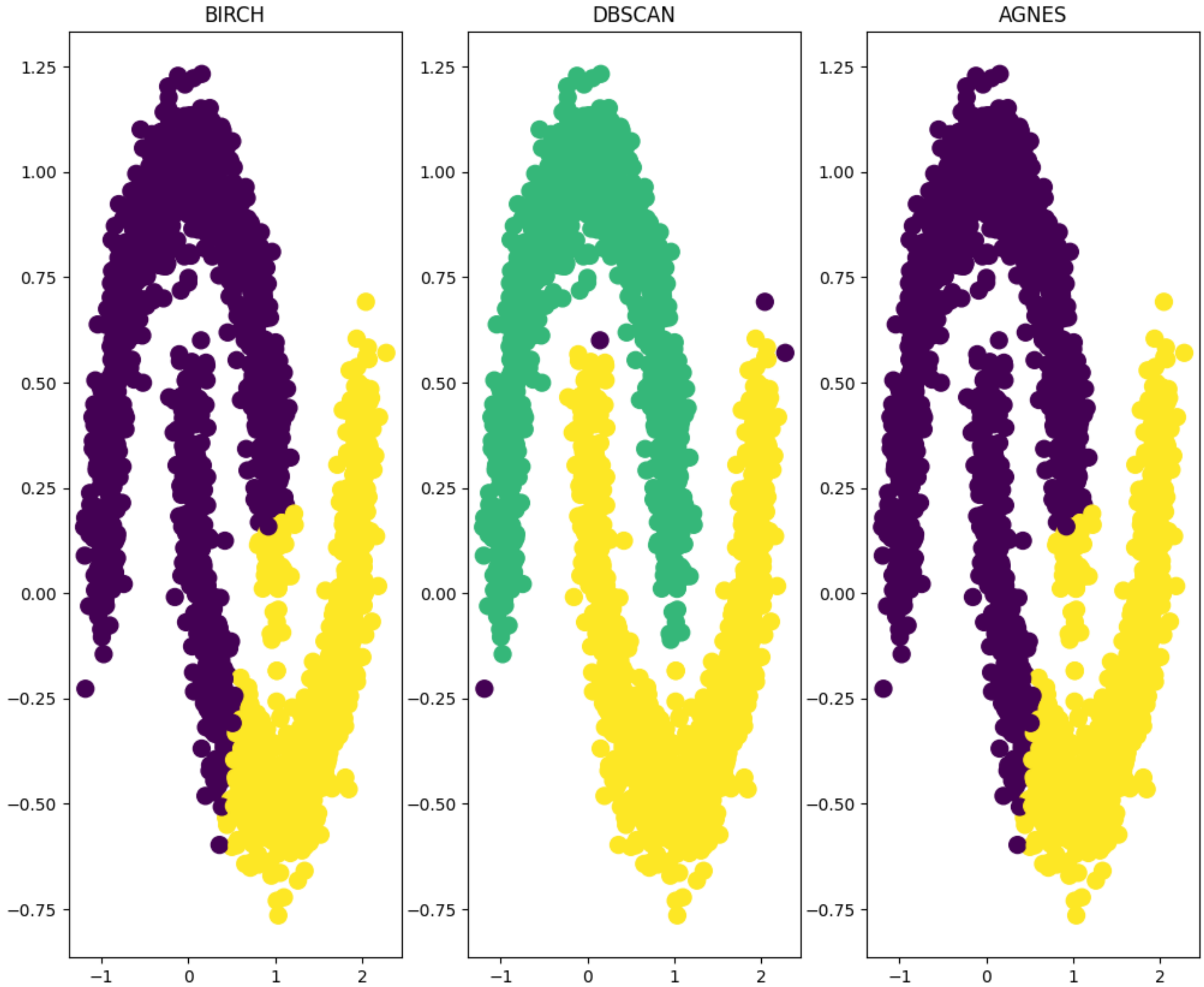
In [27]:

```python
plt.figure(figsize=(25,10))
#Visualizing all the clusters of birch
plt.subplot(1,6,3)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_birch)
```

```
plt.title('BIRCH')


plt.subplot(1,6,4)
plt.title('DBSCAN')
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=dbscan_labels1)

plt.subplot(1,6,5)
plt.title("AGNES")
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_agnes)
```

Out[27]:

```
<matplotlib.collections.PathCollection at 0x7e3139888970>
```



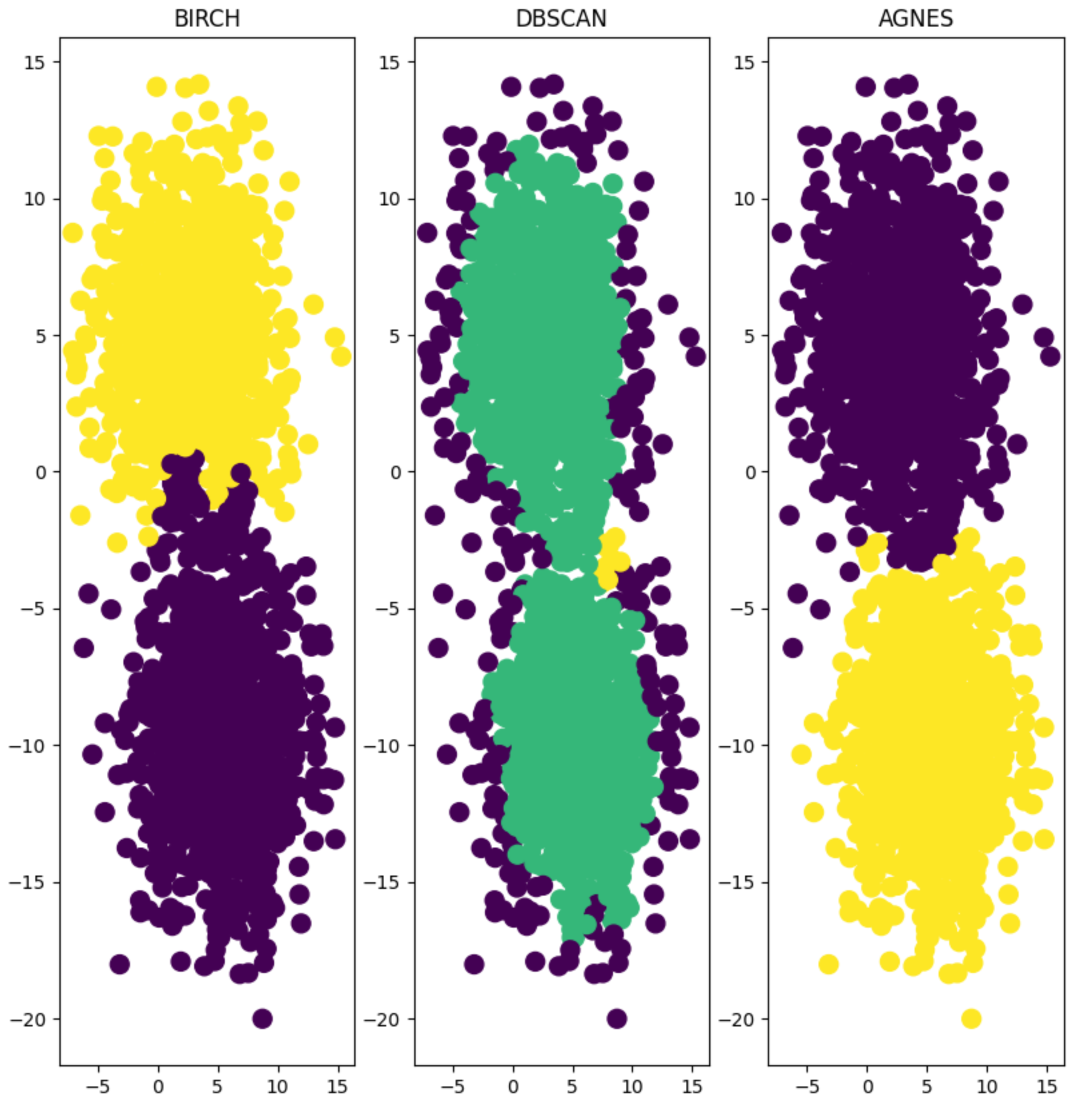
In [28]:

```
plt.figure(figsize=(20,10))
plt.subplot(1,6,3)
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y_birch2)
plt.title("BIRCH")

plt.subplot(1,6,4)
plt.title("DBSCAN")
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=dbscan_labels2)

plt.subplot(1,6,5)
plt.title("AGNES")
```

```
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y_agnes2)
```

Out[28]:

```
<matplotlib.collections.PathCollection at 0x7e313c309900>
```



Task 4

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

print("pandas version: {}".format(pd.__version__))
print("numpy version: {}".format(np.__version__))
print("seaborn version: {}".format(sns.__version__))
```

```
pandas version: 1.5.3
numpy version: 1.23.5
seaborn version: 0.12.2
```

In [2]:

```
from google.colab import files

uploaded = files.upload()
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving (task4)_Mall_Customers.csv to (task4)_Mall_Customers.csv
```

In [3]:

```
for filename in uploaded.keys():
    print(f'Uploaded file: {filename}')
Uploaded file: (task4)_Mall_Customers.csv
```

In [4]:

```
mall_data = pd.read_csv('(task4)_Mall_Customers.csv')
```

In [5]:

```
print('There are {} rows and {} columns in our dataset.'.format(mall_data.shape[0]
,mall_data.shape[1]))
There are 200 rows and 5 columns in our dataset.
```

In [6]:

```
mall_data.sample(10)
```

Out[6]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
34	35	Female	49	33	14
38	39	Female	36	37	26
142	143	Female	28	76	40
146	147	Male	48	77	36
20	21	Male	35	24	35
31	32	Female	21	30	73
96	97	Female	47	60	47
64	65	Male	63	48	51
183	184	Female	29	98	88
132	133	Female	25	72	34

In [7]:

```
mall_data.describe()
```

Out[7]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

In [8]:

```
mall_data.isnull().sum()
```

Out[8]:

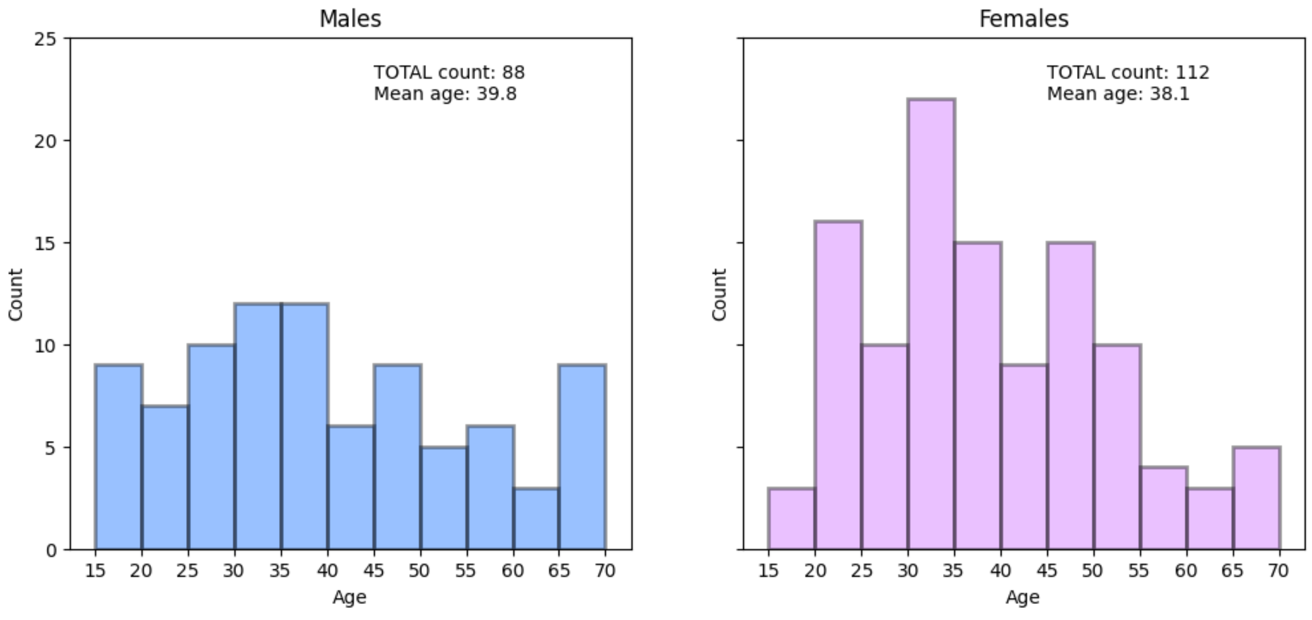
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

Exploratory Data Analysis

In [9]:

```
males_age = mall_data[mall_data['Gender']=='Male']['Age'] # subset with males age
females_age = mall_data[mall_data['Gender']=='Female']['Age'] # subset with female
s age

age_bins = range(15,75,5)

# males histogram
fig2, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,5), sharey=True)
sns.distplot(males_age, bins=age_bins, kde=False, color='#0066ff', ax=ax1, hist_kw
s=dict(edgecolor="k", linewidth=2))
ax1.set_xticks(age_bins)
ax1.set_ylim(top=25)
ax1.set_title('Males')
ax1.set_ylabel('Count')
ax1.text(45,23, "TOTAL count: {}".format(males_age.count()))
ax1.text(45,22, "Mean age: {:.1f}".format(males_age.mean()))

# females histogram
sns.distplot(females_age, bins=age_bins, kde=False, color='#cc66ff', ax=ax2, hist_
kws=dict(edgecolor="k", linewidth=2))
ax2.set_xticks(age_bins)
ax2.set_title('Females')
ax2.set_ylabel('Count')
ax2.text(45,23, "TOTAL count: {}".format(females_age.count()))
ax2.text(45,22, "Mean age: {:.1f}".format(females_age.mean()))

plt.show()
```

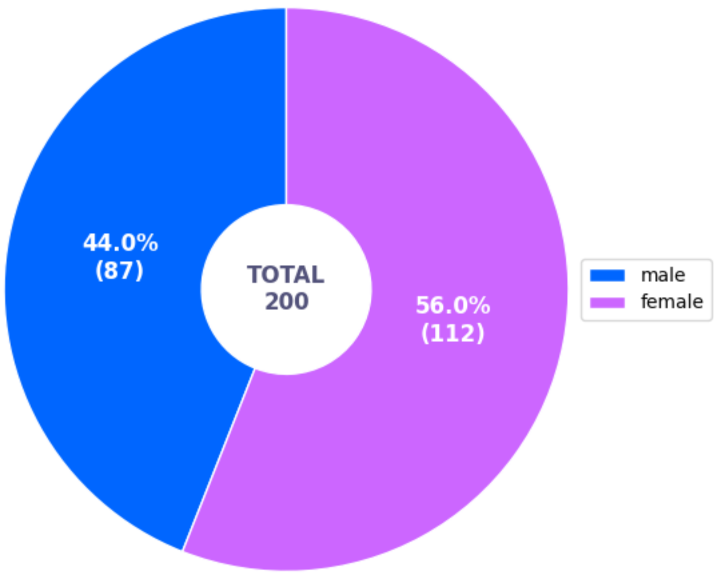
```
<ipython-input-9-90e711b08c1a>:8: UserWarning:

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

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

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

  sns.distplot(males_age, bins=age_bins, kde=False, color='#0066ff', ax=ax1, hist_
kws=dict(edgecolor="k", linewidth=2))
<ipython-input-9-90e711b08c1a>:17: UserWarning:

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

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

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

  sns.distplot(females_age, bins=age_bins, kde=False, color='#cc66ff', ax=ax2, his
t_kws=dict(edgecolor="k", linewidth=2))
```



In [10]:

```
def labeler(pct, allvals):
    absolute = int(pct/100.*np.sum(allvals))
    return "{:.1f}%\n({:d})".format(pct, absolute)

sizes = [males_age.count(),females_age.count()] # wedge sizes

fig0, ax1 = plt.subplots(figsize=(6,6))
wedges, texts, autotexts = ax1.pie(sizes,
                                   autopct=lambda pct: labeler(pct, sizes),
                                   radius=1,
                                   colors=['#0066ff','#cc66ff'],
```

```
                                    startangle=90,
                                    textprops=dict(color="w"),
                                    wedgeprops=dict(width=0.7, edgecolor='w'))

ax1.legend(wedges, ['male','female'],
           loc='center right',
           bbox_to_anchor=(0.7, 0, 0.5, 1))

plt.text(0,0, 'TOTAL\n{}'.format(mall_data['Age'].count()),
         weight='bold', size=12, color='#52527a',
         ha='center', va='center')

plt.setp(autotexts, size=12, weight='bold')
ax1.axis('equal')  # Equal aspect ratio
plt.show()
```



In [11]:

```
from sklearn.cluster import KMeans
```

In [12]:

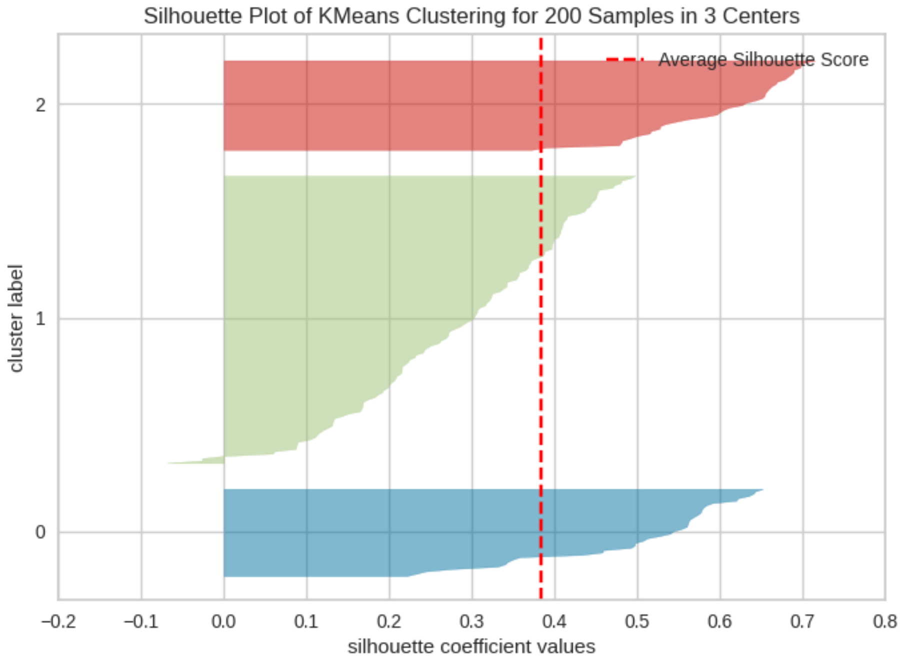
```
X_numerics = mall_data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']] # subset with numeric variables only
```

In [13]:

```
from sklearn.metrics import silhouette_score, adjusted_rand_score, normalized_mutual_info_score
from yellowbrick.cluster import SilhouetteVisualizer

# Fit KMeans clustering model
kmeans_model = KMeans(n_clusters=3, random_state=1)
mall_data['Cluster'] = kmeans_model.fit_predict(X_numerics)

# Visualize Silhouette Scores
silhouette_visualizer = SilhouetteVisualizer(kmeans_model, colors='yellowbrick')
silhouette_visualizer.fit(X_numerics)
silhouette_visualizer.show()
plt.show()

# Calculate metrics
```

```
silhouette_avg = silhouette_score(X_numerics, mall_data['Cluster'])
ari_score = adjusted_rand_score(mall_data['Gender'], mall_data['Cluster'])
nmi_score = normalized_mutual_info_score(mall_data['Gender'], mall_data['Cluster']
)

print(f"Silhouette Coefficient: {silhouette_avg}")
print(f"ARI (Adjusted Rand Index): {ari_score}")
print(f"NMI (Normalized Mutual Information): {nmi_score}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarn
ing: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the v
alue of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does n
ot have valid feature names, but KMeans was fitted with feature names
  warnings.warn(
```




```
Silhouette Coefficient: 0.3839349967742105
ARI (Adjusted Rand Index): 0.008682744107674455
NMI (Normalized Mutual Information): 0.005482866113818055
```

In [19]:

```
from sklearn.cluster import AffinityPropagation
```

In [22]:

```
from sklearn.metrics import silhouette_score, adjusted_rand_score, normalized_mutu
al_info_score
from yellowbrick.cluster import SilhouetteVisualizer
```

In [26]:

```
# Fit Affinity Propagation clustering model
affinity_model = AffinityPropagation(damping=0.9)
mall_data['Affinity_Cluster'] = affinity_model.fit_predict(X_numerics)

# Calculate metrics
silhouette_avg_affinity = silhouette_score(X_numerics, mall_data['Affinity_Cluster
'])
ari_score_affinity = adjusted_rand_score(mall_data['Gender'], mall_data['Affinity_
Cluster'])
```

```python
nmi_score_affinity = normalized_mutual_info_score(mall_data['Gender'], mall_data['
Affinity_Cluster'])

print(f"Affinity Propagation Silhouette Coefficient: {silhouette_avg_affinity}")
print(f"ARI (Adjusted Rand Index) for Affinity Propagation: {ari_score_affinity}")
print(f"NMI (Normalized Mutual Information) for Affinity Propagation: {nmi_score_a
ffinity}")
```

```
Affinity Propagation Silhouette Coefficient: 0.3425030297045019
ARI (Adjusted Rand Index) for Affinity Propagation: -0.0036640622018178546
NMI (Normalized Mutual Information) for Affinity Propagation: 0.013951958422389967
```