Palestine Technical University (Kadoorie)

Faculty of Engineering and Technology

Department of Computer Systems Engineering

Data Mining Assignment

Handling Poker Hand Data Set

Prepared by:

Eng. Aseel Nabeel Rajab 201810273

Eng. Mays Khalil Najjar 201811598

Supervised by:

Dr. Nashat Jallad.

Saturday, 30th April 2022.

Description of the Poker Hand Data Set

Each record is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is describe using two attributes (suit and rank), for a total of 10 predictive attributes.

There is one Class attribute that describes the "Poker Hand". The order of cards is important, which is why there are 480 possible Royal Flush hands as compared to 4 (one for each suit – explained in more detail below) Attribute Information:

(1 S1 "Suit of card #1"

Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}

(2 C1 "Rank of card #1"

Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King). And so on .. S2, C2 , S3, C3 ..

The last one is the CLASS "Poker Hand"

Ordinal (0-9).

Our data is just containing numerical values.

Write Python program that achieve the following goals:

- Load your dataset

jupyter Untitled2 Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns

         #loading my data set

        data = pd.read_csv('C:\\Users\\Pascal\\Desktop\\DataMining\\pokerhand.csv', encoding= 'unicode_escape')
        print(data)
```

```
         S1    C1   S2    C2   S3    C3   S4    C4  S5    C5  CLASS
0       1.0   1.0  1.0  13.0  2.0   4.0  2.0   3.0   1  12.0      0
1       NaN  12.0  3.0   2.0  3.0  11.0  4.0   5.0   2   5.0      1
2       1.0   9.0  4.0   6.0  1.0   4.0  NaN   2.0   3   9.0      1
3       1.0   4.0  3.0  13.0  2.0  13.0  NaN   1.0   3   6.0      1
4       3.0  10.0  2.0   7.0  1.0   2.0  NaN  11.0   4   9.0      0
...     ...   ...  ...   ...  ...   ...  ...   ...  ..   ...    ...
999995  3.0   1.0  1.0  12.0  2.0   9.0  NaN   9.0   2   6.0      1
999996  3.0   3.0  4.0   5.0  2.0   7.0  NaN   4.0   4   3.0      1
999997  1.0  11.0  NaN   7.0  3.0   9.0  NaN  13.0   2   7.0      1
999998  3.0   NaN  1.0   8.0  1.0   1.0  NaN  13.0   2   NaN      1
999999  2.0   5.0  2.0   9.0  4.0   9.0  NaN   3.0   3   3.0      2

[1000000 rows x 11 columns]
```

```
In [ ]:
```

The describe of data

jupyter Untitled2 Last Checkpoint: 4 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

```
In [2]: print(data.describe())
```

```
                  S1             C1             S2             C2  \
count  999997.000000  999993.000000  999997.000000  999996.000000
mean        2.500494       6.997934       2.499894       7.006103
std         1.117768       3.743372       1.118568       3.743476
min         1.000000       1.000000       1.000000       1.000000
25%         2.000000       4.000000       1.000000       4.000000
50%         3.000000       7.000000       3.000000       7.000000
75%         3.000000      10.000000       4.000000      10.000000
max         4.000000      13.000000       4.000000      13.000000

                  S3             C3             S4             C4  \
count  999999.000000  999999.000000  999923.000000  999999.000000
mean        2.500873       6.998877       2.500409       7.002300
std         1.118224       3.741890       1.117248       3.741271
min         1.000000       1.000000       1.000000       1.000000
25%         1.000000       4.000000       2.000000       4.000000
50%         3.000000       7.000000       3.000000       7.000000
75%         4.000000      10.000000       3.000000      10.000000
max         4.000000      13.000000       4.000000      13.000000

                   S5             C5           CLASS
count  1000000.000000  999995.000000  1000000.000000
mean         2.499451       6.989486        0.616902
std          1.118948       3.739900        0.773377
min          1.000000       1.000000        0.000000
25%          1.000000       4.000000        0.000000
50%          2.000000       7.000000        0.000000
75%          4.000000      10.000000        1.000000
max          4.000000      13.000000        9.000000
```

Apply data cleaning based on your data needs

➢ Missing handling.

jupyter Untitled2 Last Checkpoint: 10 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [3]:
```python
print('Data before handle the missing value')

print (data)

#Handling the missing value Using two ways
#1. Drop all the rows with null value in the certain column [S4]

data= data[pd.notnull(data['S4'])]
print('Data after handle the missing value by drop all the rows with null value in the certain column [S4]')
print(data)
```

```
Data before handle the missing value
         S1    C1   S2    C2   S3    C3   S4    C4  S5    C5  CLASS
0       1.0   1.0  1.0  13.0  2.0   4.0  2.0   3.0   1  12.0      0
1       NaN  12.0  3.0   2.0  3.0  11.0  4.0   5.0   2   5.0      1
2       1.0   9.0  4.0   6.0  1.0   4.0  NaN   2.0   3   9.0      1
3       1.0   4.0  3.0  13.0  2.0  13.0  NaN   1.0   3   6.0      1
4       3.0  10.0  2.0   7.0  1.0   2.0  NaN  11.0   4   9.0      0
...     ...   ...  ...   ...  ...   ...  ...   ...  ..   ...    ...
999995  3.0   1.0  1.0  12.0  2.0   9.0  NaN   9.0   2   6.0      1
999996  3.0   3.0  4.0   5.0  2.0   7.0  NaN   4.0   4   3.0      1
999997  1.0  11.0  NaN   7.0  3.0   9.0  NaN  13.0   2   7.0      1
999998  3.0   NaN  1.0   8.0  1.0   1.0  NaN  13.0   2   NaN      1
999999  2.0   5.0  2.0   9.0  4.0   9.0  NaN   3.0   3   3.0      2

[1000000 rows x 11 columns]
```

a. Drop all the rows with null value in the certain column [S4]

In [4]:
```python
data= data[pd.notnull(data['S4'])]
print('Data after handle the missing value by drop all the rows with null value in the certain column [S4]')
print(data)
```

```
Data after handle the missing value by drop all the rows with null value in the certain column [S4]
         S1    C1   S2    C2   S3    C3   S4    C4  S5    C5  CLASS
0       1.0   1.0  1.0  13.0  2.0   4.0  2.0   3.0   1  12.0      0
1       NaN  12.0  3.0   2.0  3.0  11.0  4.0   5.0   2   5.0      1
5       1.0   3.0  4.0   5.0  3.0   4.0  1.0  12.0   4   6.0      0
6       2.0   6.0  4.0  11.0  2.0   3.0  4.0   9.0   1   7.0      0
7       3.0   2.0  4.0   9.0  3.0   7.0  4.0   3.0   4   NaN      0
...     ...   ...  ...   ...  ...   ...  ...   ...  ..   ...    ...
999990  2.0  12.0  4.0   NaN  1.0   3.0  3.0   5.0   3   2.0      1
999991  1.0   4.0  4.0   8.0  4.0   5.0  3.0   9.0   2   1.0      0
999992  1.0   NaN  3.0   6.0  2.0   8.0  3.0   5.0   2   9.0      1
999993  1.0  12.0  3.0   9.0  3.0   6.0  1.0   3.0   1   9.0      1
999994  3.0   7.0  1.0   6.0  4.0  12.0  2.0   1.0   1   4.0      0

[999923 rows x 11 columns]
```

b. filling a missing value with previous ones.

```
In [5]: #2. filling a missing value with previous ones
        data = data.fillna(method ='pad')
        print('Data after handle the missing value by filling it automatically with previous ones ')
        print(data)

        Data after handle the missing value by filling it automatically with previous ones
                 S1    C1   S2    C2   S3    C3   S4    C4  S5    C5  CLASS
        0       1.0   1.0  1.0  13.0  2.0   4.0  2.0   3.0   1  12.0      0
        1       1.0  12.0  3.0   2.0  3.0  11.0  4.0   5.0   2   5.0      1
        5       1.0   3.0  4.0   5.0  3.0   4.0  1.0  12.0   4   6.0      0
        6       2.0   6.0  4.0  11.0  2.0   3.0  4.0   9.0   1   7.0      0
        7       3.0   2.0  4.0   9.0  3.0   7.0  4.0   3.0   4   7.0      0
        ...     ...   ...  ...   ...  ...   ...  ...   ...  ..   ...    ...
        999990  2.0  12.0  4.0  13.0  1.0   3.0  3.0   5.0   3   2.0      1
        999991  1.0   4.0  4.0   8.0  4.0   5.0  3.0   9.0   2   1.0      0
        999992  1.0   4.0  3.0   6.0  2.0   8.0  3.0   5.0   2   9.0      1
        999993  1.0  12.0  3.0   9.0  3.0   6.0  1.0   3.0   1   9.0      1
        999994  3.0   7.0  1.0   6.0  4.0  12.0  2.0   1.0   1   4.0      0

        [999923 rows x 11 columns]
```

➢ Remove noise value.

As we see in description of dataset, the max and the min in each column is satisfied with the data rule.
For example: S1, S2, S3, S4, S5 the values in these column are in the range [1 – 4], and there is no noise value as shown in descript of data.

```
jupyter Untitled2 Last Checkpoint: 4 minutes ago (autosaved)                Logout
File  Edit  View  Insert  Cell  Kernel  Widgets  Help        Trusted  Python 3 (ipykernel)
Run  Code

In [2]: print(data.describe())

                     S1             C1             S2             C2  \
        count  999997.000000  999993.000000  999997.000000  999996.000000
        mean        2.500494       6.997934       2.499894       7.006103
        std         1.117768       3.743372       1.118568       3.743476
        min         1.000000       1.000000       1.000000       1.000000
        25%         2.000000       4.000000       1.000000       4.000000
        50%         3.000000       7.000000       3.000000       7.000000
        75%         3.000000      10.000000       4.000000      10.000000
        max         4.000000      13.000000       4.000000      13.000000

                     S3             C3             S4             C4  \
        count  999999.000000  999999.000000  999923.000000  999999.000000
        mean        2.500873       6.998877       2.500409       7.002300
        std         1.118224       3.741890       1.117248       3.741271
        min         1.000000       1.000000       1.000000       1.000000
        25%         1.000000       4.000000       2.000000       4.000000
        50%         3.000000       7.000000       3.000000       7.000000
        75%         4.000000      10.000000       3.000000      10.000000
        max         4.000000      13.000000       4.000000      13.000000

                      S5             C5          CLASS
        count  1000000.000000  999995.000000  1000000.000000
        mean         2.499451       6.989486        0.616902
        std          1.118948       3.739900        0.773377
        min          1.000000       1.000000        0.000000
        25%          1.000000       4.000000        0.000000
        50%          2.000000       7.000000        0.000000
        75%          4.000000      10.000000        1.000000
        max          4.000000      13.000000        9.000000
```

For example: C1, C2, C3, C4, C5 the values in these column are in the range [1 – 13], and there is no noise value as shown in descript of data.

- Remove duplicate records.

```
In [6]: #Remove duplicate records
        data = data.drop_duplicates()
        print('Data after remove the duplicate row')
        print(data)

        Data after remove the duplicate row
                 S1    C1   S2    C2   S3    C3   S4    C4  S5    C5  CLASS
        0        1.0   1.0  1.0  13.0  2.0   4.0  2.0   3.0   1  12.0      0
        1        1.0  12.0  3.0   2.0  3.0  11.0  4.0   5.0   2   5.0      1
        5        1.0   3.0  4.0   5.0  3.0   4.0  1.0  12.0   4   6.0      0
        6        2.0   6.0  4.0  11.0  2.0   3.0  4.0   9.0   1   7.0      0
        7        3.0   2.0  4.0   9.0  3.0   7.0  4.0   3.0   4   7.0      0
        ...      ...   ...  ...   ...  ...   ...  ...   ...  ..   ...    ...
        999990   2.0  12.0  4.0  13.0  1.0   3.0  3.0   5.0   3   2.0      1
        999991   1.0   4.0  4.0   8.0  4.0   5.0  3.0   9.0   2   1.0      0
        999992   1.0   4.0  3.0   6.0  2.0   8.0  3.0   5.0   2   9.0      1
        999993   1.0  12.0  3.0   9.0  3.0   6.0  1.0   3.0   1   9.0      1
        999994   3.0   7.0  1.0   6.0  4.0  12.0  2.0   1.0   1   4.0      0

        [997797 rows x 11 columns]
```

- Remove correlated attributes.

First we want to use method corr() to see the correlation between the columns.

```
File  Edit  View  Insert  Cell  Kernel  Widgets  Help          Trusted   Python 3 (ipykernel)
Run  Code

In [7]: # Corrolation
        print('Correlation')
        corr_matri1 = data.corr()    #to get the relationship between columns
        print(corr_matri1)

        Correlation
                     S1        C1        S2        C2        S3        C3        S4  \
         S1      1.000000 -0.001435 -0.021579  0.001221 -0.019324  0.000388 -0.019301
        C1      -0.001435  1.000000 -0.000736 -0.021568 -0.000153 -0.019410 -0.000208
        S2      -0.021579 -0.000736  1.000000  0.000082 -0.019212  0.000931 -0.020501
        C2       0.001221 -0.021568  0.000082  1.000000 -0.000404 -0.020863  0.001639
        S3      -0.019324 -0.000153 -0.019212 -0.000404  1.000000  0.001007 -0.019806
        C3       0.000388 -0.019410  0.000931 -0.020863  0.001007  1.000000 -0.000212
        S4      -0.019301 -0.000208 -0.020501  0.001639 -0.019806 -0.000212  1.000000
        C4       0.000005 -0.018753 -0.000112 -0.020660  0.000651 -0.018970 -0.000331
        S5      -0.018942  0.000198 -0.020273 -0.000674 -0.020556  0.001312 -0.019479
        C5       0.001569 -0.021042  0.001241 -0.017221 -0.000040 -0.020350 -0.000319
         CLASS   0.000062  0.003926 -0.000277  0.001712  0.001179  0.002423 -0.001129

                     C4        S5        C5     CLASS
         S1      0.000005 -0.018942  0.001569  0.000062
        C1      -0.018753  0.000198 -0.021042  0.003926
        S2      -0.000112 -0.020273  0.001241 -0.000277
        C2      -0.020660 -0.000674 -0.017221  0.001712
        S3       0.000651 -0.020556 -0.000040  0.001179
        C3      -0.018970  0.001312 -0.020350  0.002423
        S4      -0.000331 -0.019479 -0.000319 -0.001129
        C4       1.000000  0.000928 -0.020813  0.003043
        S5       0.000928  1.000000 -0.000032 -0.001533
        C5      -0.020813 -0.000032  1.000000  0.001584
         CLASS   0.003043 -0.001533  0.001584  1.000000
```

<u>As we see,</u> the correlated between columns is very weak. But we want to check this by coding.

➔ Correlation rate greater than or equal 0.8 for positive correlation

```
In [8]: correlated1 = set()     #initial the set, which we want to put in it the name of columns that correlated

        for i in range (len(corr_matri1.columns)):   #i is number of columns
            for j in range (i) :
                if abs(corr_matri1.iloc[i ,j]) >= 0.8:      #if correlation >= .8 add the column name to the set
                    nomeOfColumn = corr_matri1.columns[i]
                    correlated1.add(nomeOfColumn)

        print("Correlation rate greater than or equal 0.8 for positive, its size = ") #print the size of correlated
        print(len(correlated1))
        print(correlated1) #print the correlated

        Correlation rate greater than or equal 0.8 for positive, its size =
        0
        set()
```

The set is empty.

➔ Correlation rate less than or equal - 0.8 for negative correlation.

```
set()

In [9]: correlated2 = set()       #initial the set, which we want to put in it the name of columns that correlated

        for i in range (len(corr_matri1.columns)):   #i is number of columns
            for j in range (i) :
                if abs(corr_matri1.iloc[i ,j]) <=  -0.8:    #if correlation <= -.8 add the column name to the set
                    nomeOfColumn = corr_matri1.columns[i]
                    correlated2.add(nomeOfColumn)
        len(correlated2)
        print("Correlation rate less than or equal -0.8 for negative")  #print the size of correlated
        print(len(correlated2))
        print(correlated2)

        Correlation rate less than or equal -0.8 for negative
        0
        set()
```

The set is empty.

So, there is no relationship between the columns.

- Apply discretization on numeric attributes as possible.

```
In [25]: data['Class_'] = pd.cut(x=data['\xa0CLASS'], bins=[0,3,6,9],
                        labels=["0 to 3","3 to 6","6 to 9"])
         print(data)

         S1    C1   S2    C2   S3    C3   S4   C4  S5    C5  CLASS   Class  \
0       1.0   1.0  1.0  13.0  2.0   4.0  2.0  3.0  1  12.0      0     NaN
1       1.0  12.0  3.0   2.0  3.0  11.0  4.0  5.0  2   5.0      1  0 to 3
```

- Remove irrelevant attributes.

To remove them, we need to know the importance feature in the dataset. Then we can decide which columns that we need.

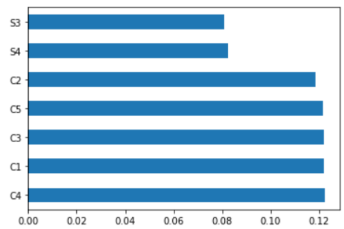
```
In [7]: #Because the data is big so want to take a sample of it to see the importance feature and to remove the irrelevant attributes
        new_data = data.head(1000)

        X = new_data.iloc[:,0:10]  #independent columns
        y = new_data.iloc[:,-1]    #target column which is class

        from sklearn.ensemble import ExtraTreesClassifier
        import matplotlib.pyplot as plt

        model = ExtraTreesClassifier()
        model.fit(X,y)
        print(model.feature_importances_)     #use inbuilt class feature_importances of tree based classifiers
        #plot graph of feature importances for better visualization
        feat_importances = pd.Series(model.feature_importances_, index=X.columns)
        feat_importances.nlargest(7).plot(kind='barh')
        plt.show()

        [0.07542771 0.12201803 0.07908549 0.11865413 0.08085439 0.12187337
         0.08257098 0.12256016 0.07512916 0.12182657]
```



As we can see, the most seven attributes are:

[C4, C1, C3, C5, C2, S4, S3]

So we want to drop [S1, S2, S5]

```
In [13]: data = data.drop(['\xa0S1'], axis = 1) #Delete first culomn S1
         data = data.drop(['S2'], axis = 1) #Delete culomn S2
         data = data.drop(['S5'], axis = 1) #Delete culomn S5

In [14]: print (data)

           C1    C2   S3    C3   S4    C4    C5  CLASS
0         1.0  13.0  2.0   4.0  2.0   3.0  12.0      0
1        12.0   2.0  3.0  11.0  4.0   5.0   5.0      1
5         3.0   5.0  3.0   4.0  1.0  12.0   6.0      0
6         6.0  11.0  2.0   3.0  4.0   9.0   7.0      0
7         2.0   9.0  3.0   7.0  4.0   3.0   7.0      0
...       ...   ...  ...   ...  ...   ...   ...    ...
999990   12.0  13.0  1.0   3.0  3.0   5.0   2.0      1
999991    4.0   8.0  4.0   5.0  3.0   9.0   1.0      0
999992    4.0   6.0  2.0   8.0  3.0   5.0   9.0      1
999993   12.0   9.0  3.0   6.0  1.0   3.0   9.0      1
999994    7.0   6.0  4.0  12.0  2.0   1.0   4.0      0

[997797 rows x 8 columns]
```

➢ Split your dataset into training and testing sets

80% and 20% for training and testing sets respectively.

In [15]:
```
# Now we want to split our dataset to two parts
# First part for training and its 80% from our data
# Second part for testing and its 20% from our data
x = data.iloc[:,0:7]        # X is all the predictive attributes
y =data.iloc[:, -1]          # Y is the goal attribute
#import the suitable library
from sklearn.model_selection import train_test_split

#call method to split it with size .2 of data to test, so .8 of data to train
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.2)

#Now print x_train & x_test with its length
print('x_train')
print(x_train)
print('The length of x_train' )
print(len(x_train))
```

```
x_train
          C1    C2   S3    C3   S4    C4    C5
759727   9.0   6.0  2.0  11.0  1.0   6.0  12.0
255513  10.0  13.0  2.0  10.0  1.0   5.0   6.0
666481   1.0   6.0  2.0   6.0  2.0   7.0   1.0
170425   7.0  10.0  2.0   8.0  2.0   4.0  10.0
586173   6.0   3.0  4.0   1.0  1.0  13.0   3.0
...      ...   ...  ...   ...  ...   ...   ...
990795   2.0   9.0  2.0   1.0  1.0   7.0   8.0
837622   8.0   6.0  2.0   5.0  2.0   9.0   6.0
648235  11.0   1.0  1.0  13.0  1.0   7.0  11.0
354504   4.0   5.0  3.0  12.0  1.0  10.0  12.0
4850     7.0  12.0  1.0   3.0  3.0   7.0   6.0

[798237 rows x 7 columns]
The length of x_train
798237
```

In [17]:
```
print('x_test')
print(x_test)
print('The length of x_test' )
print(len(x_test))
```

```
x_test
          C1    C2   S3    C3   S4    C4    C5
58455    8.0   1.0  1.0   5.0  3.0  12.0   8.0
580220   4.0  12.0  2.0   9.0  4.0   5.0  12.0
113767   1.0   1.0  3.0   4.0  2.0   6.0   3.0
698355   6.0   7.0  2.0   4.0  1.0   9.0   2.0
387197   8.0  10.0  2.0   1.0  2.0   9.0   2.0
...      ...   ...  ...   ...  ...   ...   ...
373002   2.0  11.0  2.0   5.0  1.0  12.0   8.0
856800   6.0  12.0  2.0   6.0  3.0  10.0  13.0
694388   4.0   2.0  1.0  11.0  3.0   6.0  10.0
492650  11.0   7.0  3.0   8.0  2.0  13.0   7.0
177155   2.0  12.0  4.0   6.0  4.0   2.0   8.0

[199560 rows x 7 columns]
The length of x_test
199560
```

```
In [18]: #Now print y_train & y_test with its length
         print('y_train')
         print(y_train)
         print('The length of y_train' )
         print(len(y_train))

y_train
759727    1
255513    1
666481    2
170425    1
586173    1
         ..
990795    0
837622    1
648235    1
354504    1
4850      1
Name:  CLASS, Length: 798237, dtype: int64
The length of y_train
798237
```

```
In [19]: print('y_test')
         print(y_test)
         print('The length of y_test' )
         print(len(y_test))

y_test
58455     1
580220    1
113767    1
698355    0
387197    0
         ..
373002    0
856800    1
694388    0
492650    1
177155    1
Name:  CLASS, Length: 199560, dtype: int64
The length of y_test
199560
```

Save them in separated file.

```
In [33]: import os
         a_path = "C:Users/Pascal/Desktop"
         a_folder = "DataMining"
         a_file = "test.tsv"

         joined_path = os.path.join(a_path, a_folder, a_file)

         test_path =  joined_path

         b_path = "C:Users/Pascal/Desktop"
         b_folder = "DataMining"
         b_file = "train.tsv"

         joined_path1 = os.path.join(b_path, b_folder, b_file)

         train_path =  joined_path
         # save the train and test file
         # again using the '\t' separator to create tab-separated-values files

         train=['x_train', 'y_train']
         train.to_csv(train_path, sep='\t', index=8)
         test=['x_test', 'y_test']
         test.to_csv(test_path, sep='\t', index=8)
```

➢ Classification

We used KNN (k nearest neighbor) to build a classifier model in our data.
Fist of all we drop the class attribute in our data and save it to other data

```
#split data attribute and label attribute
#Because we're using unsupervised
attributes = data.drop(['\xa0CLASS'], axis=1)
labels = data['\xa0CLASS']
```

Then build the model

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(x_train, y_train)

# Predict on dataset which model has not seen before
print("Classification")
print(knn.predict(x_test))
```

The output is

```
Classification
[0 1 1 ... 1 1 3]
```

➢ Clustering

By K- mean clustering algorithm S

First of all, we should split data (Dataset shouldn’t be labeled, because we’re dealing with Unsupervised type).

```python
#split data attribute and label attribute
#Because we're using unsupervised
attributes = data.drop(['\xa0CLASS'], axis=1)
labels = data['\xa0CLASS']
```

Second:

```python
#import and create KMeans object
from sklearn.cluster import KMeans
model = KMeans(n_clusters=10)
model.fit(attributes)
```

Then,

```python
#predict the clusters lists for the dataset
y_pred = model.predict(attributes)
print(y_pred)
```

And here is the output:

```
[5 2 0 ... 9 1 2]
```

Last stage:

```
In [3]: #evaluation stage
        from sklearn import metrics
        contingecyMatrix = metrics.cluster.contingency_matrix(labels, y_pred)
        print(contingecyMatrix)

[[43437 42921 54774 55786 51396 54509 44631 45284 51249 56114]
 [44576 44095 41744 39694 41739 41557 43352 43396 41294 40131]
 [ 6043  5848  4367  3871  4469  4414  4992  5088  4434  3983]
 [ 2911  2965  1630  1293  2054  1712  2584  2486  2095  1344]
 [  608   680   161   241   272   253   713   571   202   175]
 [  171   176   233   231   204   226   173   178   210   190]
 [  253   282    85   104   105    99   171   153    93    77]
 [   61    48     9     5    26    12    19    19    31     0]
 [    0     4     2     1     0     0     2     1     1     1]
 [    0     1     0     0     0     1     1     0     0     0]]
```

➢ Association

By FP-Growth Algorithm

First of all:

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

In [27]: #get association rules by using FP-growth algorithm
         import pyfpgrowth
```

Second: We have to get patterns

```
Patterns
{('1',): 1, ('1', 'C'): 1, ('2',): 1, ('2', 'C'): 1, ('5',): 1, ('5', 'C'): 1, ('\xa0',): 1, ('S', '\xa0'): 2, ('C', '\xa0'): 1, ('C', 'S', '\xa0'): 1, ('L',): 1, ('L', '\xa0'): 1, ('L', 'S'): 2, ('C', 'L'): 1, ('L', 'S', '\xa0'): 1, ('C', 'L', '\xa0'): 1, ('C', 'L', 'S'): 1, ('C', 'L', 'S', '\xa0'): 1, ('A',): 1, ('A', 'L'): 1, ('A', '\xa0'): 1, ('A', 'S'): 2, ('A', 'C'): 1, ('A', 'L', '\xa0'): 1, ('A', 'L', 'S'): 1, ('A', 'C', 'L'): 1, ('A', 'S', '\xa0'): 1, ('A', 'C', '\xa0'): 1, ('A', 'C', 'S'): 1, ('A', 'L', 'S', '\xa0'): 1, ('A', 'C', 'L', '\xa0'): 1, ('A', 'C', 'L', 'S'): 1, ('A', 'C', 'S', '\xa0'): 1, ('A', 'C', 'L', 'S', '\xa0'): 1, ('3', 'S'): 1, ('3', 'C'): 1, ('4', 'S'): 1, ('4', 'C'): 1, ('S',): 4, ('C', 'S'): 2, ('S', 'S'): 1, ('C', 'S', 'S'): 1, ('C',): 6}
```

Finally:

```
In [38]: rules= pyfpgrowth.generate_association_rules(patterns, 0.3)
         print('Rules\n', rules)
```

The output:

```
Rules
 {('1',): (('C',), 1.0), ('2',): (('C',), 1.0), ('5',): (('C',), 1.0), ('S',): (('C',), 0.5), ('\xa0',): (('A', 'C', 'L', 'S'),
1.0), ('C', 'S'): ((), 0.5), ('C', '\xa0'): (('A', 'L', 'S'), 1.0), ('S', '\xa0'): (('A', 'C', 'L'), 0.5), ('L',): (('A', 'C',
'S', '\xa0'), 1.0), ('L', 'S'): (('A', 'C', '\xa0'), 0.5), ('L', '\xa0'): (('A', 'C', 'S'), 1.0), ('C', 'L'): (('A', 'S', '\xa
0'), 1.0), ('C', 'L', 'S'): (('A', '\xa0'), 1.0), ('C', 'L', '\xa0'): (('A', 'S'), 1.0), ('C', 'S', '\xa0'): (('A', 'L'), 1.0),
('L', 'S', '\xa0'): (('A', 'C'), 1.0), ('A',): (('C', 'L', 'S', '\xa0'), 1.0), ('A', 'L'): (('C', 'S', '\xa0'), 1.0), ('A', '\x
a0'): (('C', 'L', 'S'), 1.0), ('A', 'S'): (('C', 'L', '\xa0'), 0.5), ('A', 'C'): (('L', 'S', '\xa0'), 1.0), ('A', 'L', 'S'):
(('C', '\xa0'), 1.0), ('A', 'L', '\xa0'): (('C', 'S'), 1.0), ('A', 'S', '\xa0'): (('C', 'L'), 1.0), ('A', 'C', 'L'): (('S', '\x
a0'), 1.0), ('A', 'C', '\xa0'): (('L', 'S'), 1.0), ('A', 'C', 'S'): (('L', '\xa0'), 1.0), ('A', 'C', 'L', 'S'): (('\xa0',), 1.
0), ('A', 'C', 'L', '\xa0'): (('S',), 1.0), ('A', 'C', 'S', '\xa0'): (('L',), 1.0), ('A', 'L', 'S', '\xa0'): (('C',), 1.0),
('C', 'L', 'S', '\xa0'): (('A',), 1.0), ('C',): (('S',), 0.3333333333333333), ('S', 'S'): (('C',), 1.0)}
```