Darshan UNIVERSITY

योग: कर्मसु कौशलम्

Data Mining

Lab - 1

Prit Kanani 23010101126 142

Introduction to Pandas Library Function:

Step-1 Import the pandas Libraries

In [18]:
```
import pandas as pd
```

Step-2 Import the dataset from this:....

In [ ]:
```
df = pd.read_csv("titanic.csv")
```

Step-3 Read csv or excel File

In [ ]:
```
df
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STC 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3703

891 rows × 12 columns

Step-4 Print Data from csv or excel File

In [8]:

```
df
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticl
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STC (31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3703

891 rows × 12 columns

Step-5 See the First 10 Rows

In [16]:

```
df.head(10)
```

Out[16]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

Step-6 See the Last 10 Rows

In [10]:

```
df.tail(10)
```

Out[10]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	T
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	34
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./S(3
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTO 39
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	38
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	21
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	11
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C.
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	11
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	37

Step-7 Data type of each columns

In [26]:

```
df.dtypes
```

Out[26]:
```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object
```

Step-8 Display Summary Information

In [44]:
```
df.describe()
```

Out[44]:

	PassengerId	Survived	Pclass	Age	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	6.000000	512.329200

Step-9 Access a specific column

In [31]:
```
df["PassengerId"]
```

Out[31]:
```
0        1
1        2
2        3
3        4
4        5
      ...
886    887
887    888
888    889
889    890
890    891
Name: PassengerId, Length: 891, dtype: int64
```

Step-10 Access rows by their integer location

In [47]:
```
df.iloc[1:3]
```

Out[47]:

	PassengerId	Survived	Pclass	Name	Sex	Age	Parch	Ticket	Fa
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	0	PC 17599	71.28
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	STON/O2. 3101282	7.92

Step-11 Delete a specific Column

In [40]:
```
df.drop("SibSp",axis='columns',inplace=True)
```

Step-12 Create a new Column

In [46]:
```
df["isCbin"] = ~df["Cabin"].isnull()
```

Step-13 Perform Condition Selection on DataFrame

In [55]:
```
df[df["Age"] == 3]
```

Out[55]:

	PassengerId	Survived	Pclass	Name	Sex	Age	Parch	Ticket	Fa
43	44	1	2	Laroche, Miss. Simonne Marie Anne Andree	female	3.0	2	SC/ Paris 2123	41.57
193	194	1	2	Navratil, Master. Michel M	male	3.0	1	230080	26.00
261	262	1	3	Asplund, Master. Edvin Rojj Felix	male	3.0	2	347077	31.38
348	349	1	3	Coutts, Master. William Loch "William"	male	3.0	1	C.A. 37671	15.90
374	375	0	3	Palsson, Miss. Stina Viola	female	3.0	1	349909	21.07
407	408	1	2	Richards, Master. William Rowe	male	3.0	1	29106	18.75

Step-14 Compute the sum of value

In [56]:

```
sum(df["Fare"])#df["Fare"].sum()
```

Out[56]: 28693.9493

Step-15 Compute the mean of value

In [57]:

```
df["Fare"].mean()
```

Out[57]: np.float64(32.204207968574636)

Step-16 Count non-null value (column)

In [61]:

```
(~df.isnull()).sum()
```

Out[61]:
```
PassengerId    891
Survived       891
Pclass         891
Name           891
Sex            891
Age            714
Parch          891
Ticket         891
Fare           891
Cabin          204
Embarked       889
isCbin         891
dtype: int64
```

Step-17 Find Minimun or Maximum values

In [63]:
```
df["Fare"].min()
```

Out[63]:
```
np.float64(0.0)
```

In [64]:
```
df["Fare"].max()
```

Out[64]:
```
np.float64(512.3292)
```

Data Mining - Lab - 2

Numpy & Perform Data Exploration with Pandas

Prit Kanani 23010101126 142

Numpy

1. NumPy (Numerical Python) is a powerful open-source library in Python used for numerical and scientific computing.
2. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on them efficiently.
3. NumPy is highly optimized and written in C, making it much faster than using regular Python lists for numerical operations.
4. It serves as the foundation for many other Python libraries in data science and machine learning, like pandas, TensorFlow, and scikit-learn.
5. With features like broadcasting, vectorization, and integration with C/C++ code, NumPy allows for cleaner and faster code in numerical computations.

Step 1. Import the Numpy library

In [5]:
```
import numpy as np
```

Step 2. Create a 1D array of numbers

In [8]:
```
a = np.arange(11)
print(a)
type(a)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10]
```

Out[8]:
```
numpy.ndarray
```

In [9]:
```
a = np.arange(2,9)
a
```

Out[9]:
```
array([2, 3, 4, 5, 6, 7, 8])
```

Step 3. Reshape 1D to 2D Array

In [16]:
```
a = np.arange(12).reshape(6,2)
a
```

Out[16]:
```
array([[ 0,  1],
       [ 2,  3],
       [ 4,  5],
       [ 6,  7],
       [ 8,  9],
       [10, 11]])
```

Step 4. Create a Linspace array

In [30]:
```
np.linspace(0,5,6)
```

Out[30]:
```
array([0., 1., 2., 3., 4., 5.])
```

Step 5. Create a Random Numbered Array

In [25]:
```
np.random.rand(2)
```

Out[25]:
```
array([0.03861186, 0.7275866 ])
```

In [26]:
```
np.random.rand(2,4)
```

Out[26]:
```
array([[0.21246282, 0.83472818, 0.63740315, 0.75614208],
       [0.4694655 , 0.1977654 , 0.7495659 , 0.67554901]])
```

Step 6. Create a Random Integer Array

In [35]:
```
np.random.randint(1,100,size=10)
```

Out[35]:
```
array([ 3, 97, 23, 35, 67, 79, 33,  3, 39, 84], dtype=int32)
```

In [32]:
```
np.random.randint(1,100,size=(2,4))
```

Out[32]:
```
array([[64, 70, 23, 25],
       [79,  3,  8, 47]], dtype=int32)
```

Step 7. Create a 1D Array and get Max,Min,ArgMax,ArgMin

In [50]:
```
a = np.random.randint(1,100,size=10)
print(a)
print(a.max())
print(a.min())
print(a.argmax())
```

```
print(a.argmin())
```

```
[96 17 42 18 61 90 75 79  7 31]
96
7
0
8
```

Step 8. Indexing in 1D Array

In [55]:
```
print(a[0])
```

```
96
```

In [54]:
```
print(a[1:4])
```

```
[17 42 18]
```

Step 9. Indexing in 2D Array

In [59]:
```
a = np.random.randint(1,100,size=(4,5))
print(a[:])
```

```
[[95 54 19 52 56]
 [18 26 56 26 96]
 [78 98 15 45  9]
 [56 16 52 28  6]]
```

In [61]:
```
print(a[1::2])
```

```
[[18 26 56 26 96]
 [56 16 52 28  6]]
```

In [63]:
```
print(a[::2,::2])
```

```
[[95 19 56]
 [78 15  9]]
```

In [66]:
```
print(a[1:3:,1:4:])
```

```
[[26 56 26]
 [98 15 45]]
```

Step 10. Conditional Selection

In [72]:
```
print(a[a>25])
```

```
[95 54 52 56 26 56 26 96 78 98 45 56 52 28]
```

�You did it! 10 exercises down — you're on fire! �

Pandas

Step 1. Import the necessary libraries

In [67]:
```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called users and use the 'user_id' as index

In [89]:
```
users = pd.read_csv("https://raw.githubusercontent.com/justmarkham/DAT8/master
print(users)
```

```
          age gender     occupation zip_code
user_id
1          24      M     technician    85711
2          53      F          other    94043
3          23      M         writer    32067
4          24      M     technician    43537
5          33      F          other    15213
...       ...    ...            ...      ...
939        26      F        student    33319
940        32      M  administrator    02215
941        20      M        student    97229
942        48      F      librarian    78209
943        22      M        student    77841

[943 rows x 4 columns]
```

Step 4. See the first 25 entries

In [90]:
```
users.head(25)
```

Out[90]:

	age	gender	occupation	zip_code
user_id				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213
6	42	M	executive	98101
7	57	M	administrator	91344
8	36	M	administrator	05201
9	29	M	student	01002
10	53	M	lawyer	90703
11	39	F	other	30329
12	28	F	other	06405
13	47	M	educator	29206
14	45	M	scientist	55106
15	49	F	educator	97301
16	21	M	entertainment	10309
17	30	M	programmer	06355
18	35	F	other	37212
19	40	M	librarian	02138
20	42	F	homemaker	95660
21	26	M	writer	30068
22	25	M	writer	40206
23	30	F	artist	48197
24	21	F	artist	94533
25	39	M	engineer	55107

Step 5. See the last 10 entries

In [91]:

```
users.tail(10)
```

Out[91]:

user_id	age	gender	occupation	zip_code
934	61	M	engineer	22902
935	42	M	doctor	66221
936	24	M	other	32789
937	48	M	educator	98072
938	38	F	technician	55038
939	26	F	student	33319
940	32	M	administrator	02215
941	20	M	student	97229
942	48	F	librarian	78209
943	22	M	student	77841

Step 6. What is the number of observations in the dataset?

In [92]:
```
users.size
```

Out[92]:
```
3772
```

Step 7. What is the number of columns in the dataset?

In [94]:
```
users.shape[1]
```

Out[94]:
```
4
```

Step 8. Print the name of all the columns.

In [95]:
```
users.columns
```

Out[95]:
```
Index(['age', 'gender', 'occupation', 'zip_code'], dtype='object')
```

Step 9. How is the dataset indexed?

In [99]:
```
users.index
```

Out[99]:
```
Index([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
       ...
       934, 935, 936, 937, 938, 939, 940, 941, 942, 943],
      dtype='int64', name='user_id', length=943)
```

Step 10. What is the data type of each column?

In [97]:
```
users.dtypes
```

Out[97]:
```
age            int64
gender        object
occupation    object
zip_code      object
dtype: object
```

Step 11. Print only the occupation column

In [98]:
```
users["occupation"]
```

Out[98]:
```
user_id
1         technician
2              other
3             writer
4         technician
5              other
           ...
939          student
940    administrator
941          student
942        librarian
943          student
Name: occupation, Length: 943, dtype: object
```

Step 12. How many different occupations are in this dataset?

In [113…
```
users.occupation.nunique()
```

Out[113…
```
21
```

In [114…
```
users.occupation.unique()
```

Out[114…
```
array(['technician', 'other', 'writer', 'executive', 'administrator',
       'student', 'lawyer', 'educator', 'scientist', 'entertainment',
       'programmer', 'librarian', 'homemaker', 'artist', 'engineer',
       'marketing', 'none', 'healthcare', 'retired', 'salesman', 'doctor'],
      dtype=object)
```

Step 13. What is the most frequent occupation?

In [112…
```
users.occupation.value_counts()
```

Out[112…

```
occupation
student          196
other            105
educator          95
administrator     79
engineer          67
programmer        66
librarian         51
writer            45
executive         32
scientist         31
artist            28
technician        27
marketing         26
entertainment     18
healthcare        16
retired           14
lawyer            12
salesman          12
none               9
homemaker          7
doctor             7
Name: count, dtype: int64
```

Step 14. Summarize the DataFrame.

In [125…

```
users.describe()
```

Out[125…

	age
count	943.000000
mean	34.051962
std	12.192740
min	7.000000
25%	25.000000
50%	31.000000
75%	43.000000
max	73.000000

Step 15. Summarize all the columns

In [126…

```
users.describe(include='all')
```

Out[126…

	age	gender	occupation	zip_code
count	943.000000	943	943	943
unique	NaN	2	21	795
top	NaN	M	student	55414
freq	NaN	670	196	9
mean	34.051962	NaN	NaN	NaN
std	12.192740	NaN	NaN	NaN
min	7.000000	NaN	NaN	NaN
25%	25.000000	NaN	NaN	NaN
50%	31.000000	NaN	NaN	NaN
75%	43.000000	NaN	NaN	NaN
max	73.000000	NaN	NaN	NaN

Step 16. Summarize only the occupation column

In [128…

```
users.occupation.describe()
```

Out[128…

```
count         943
unique         21
top       student
freq          196
Name: occupation, dtype: object
```

Step 17. What is the mean age of users?

In [130…

```
users['age'].mean()
```

Out[130…

```
np.float64(34.05196182396607)
```

Step 18. What is the age with least occurrence?

In [137…

```
users['age'].value_counts().tail()
```

Out[137…

```
age
7     1
11    1
66    1
10    1
73    1
Name: count, dtype: int64
```

You're not just learning, you're mastering it. Keep aiming higher! �

Data Mining

Lab - 3

Prit Kanani 142 23010101126

1) First, you need to read the titanic dataset from local disk and display first five records

In [7]:
```
import pandas as pd
import numpy as np
```

In [5]:
```
df = pd.read_csv("titanic.csv")
df
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticl
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STO 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3703

891 rows × 12 columns

In [6]:

```
df.head(10)
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736

2) Identify Nominal, Ordinal, Binary and Numeric attributes from data sets and display all values.

In [16]:

```
Nominal = ['Name', 'sex','cabin' , 'Embarked','ticket']
Ordinal = ['Pclass']
```

```
Binary  = ['sex','survived']
Numeric = ['age','fare', 'sibsp',' parch']
print("Nominal" , Nominal )
print("Ordinal" , Ordinal )
print("Binary" , Binary )
print("Numeric" , Numeric )
```

```
Nominal ['Name', 'sex', 'cabin', 'Embarked', 'ticket']
Ordinal ['Pclass']
Binary ['sex', 'survived']
Numeric ['age', 'fare', 'sibsp', ' parch']
```

3) Identify symmetric and asymmetric binary attributes from data sets and display all values.

In [15]:
```
print(df['Survived'].value_counts())
print('-------------------------')
print(df['Sex'].value_counts())
```

```
Survived
0    549
1    342
Name: count, dtype: int64
-------------------------
Sex
male      577
female    314
Name: count, dtype: int64
```

4) For each quantitative attribute, calculate its average, standard deviation, minimum, mode, range and maximum values.

In [28]:
```
li = ['PassengerId','Survived','Pclass','Age','SibSp','Parch','Fare']
for i in li:
    print(li.index(i) + 1,'',i)
    print('\tMean : ',df[i].mean())
    print('\tStandard deviation : ',df[i].std())
    print('\tMinimum : ',df[i].min())
    print('\tMaximum : ',df[i].max())
    print('\tMod : ',df[i].mode()[0])
    print('----------------------------------------------------')
```

```
1  PassengerId
        Mean :  446.0
        Standard deviation :  257.3538420152301
        Minimum :  1
        Maximum :  891
        Mod :  1
---------------------------------------------------
2  Survived
        Mean :  0.3838383838383838
        Standard deviation :  0.4865924542648575
        Minimum :  0
        Maximum :  1
        Mod :  0
---------------------------------------------------
3  Pclass
        Mean :  2.308641975308642
        Standard deviation :  0.836071240977049
        Minimum :  1
        Maximum :  3
        Mod :  3
---------------------------------------------------
4  Age
        Mean :  29.69911764705882
        Standard deviation :  14.526497332334042
        Minimum :  0.42
        Maximum :  80.0
        Mod :  24.0
---------------------------------------------------
5  SibSp
        Mean :  0.5230078563411896
        Standard deviation :  1.1027434322934317
        Minimum :  0
        Maximum :  8
        Mod :  0
---------------------------------------------------
6  Parch
        Mean :  0.38159371492704824
        Standard deviation :  0.8060572211299483
        Minimum :  0
        Maximum :  6
        Mod :  0
---------------------------------------------------
7  Fare
        Mean :  32.204207968574636
        Standard deviation :  49.6934285971809
        Minimum :  0.0
        Maximum :  512.3292
        Mod :  8.05
---------------------------------------------------
```

6) For the qualitative attribute (class), count the frequency for each of its distinct values.

In [24]:
```
print('Passenger class frequency :')
print(df['Pclass'].value_counts())
```

```
Passenger class frequency :
Pclass
3    491
1    216
2    184
Name: count, dtype: int64
```

7) It is also possible to display the summary for all the attributes simultaneously in a table using the describe() function. If an attribute is quantitative, it will display its mean, standard deviation and various quantiles (including minimum, median, and maximum) values. If an attribute is qualitative, it will display its number of unique values and the top (most frequent) values.

In [34]:
```
print('this is only describe')
print(df.describe())

print('this is all describe')
print(df.describe(include='all'))

print('this is object describe')
print(df.describe(include=['object']))
```

```
this is only describe
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
this is all describe
        PassengerId    Survived      Pclass                 Name   Sex  \
count    891.000000  891.000000  891.000000                  891   891
unique          NaN         NaN         NaN                  891     2
top             NaN         NaN         NaN  Dooley, Mr. Patrick  male
freq            NaN         NaN         NaN                    1   577
mean     446.000000    0.383838    2.308642                  NaN   NaN
std      257.353842    0.486592    0.836071                  NaN   NaN
min        1.000000    0.000000    1.000000                  NaN   NaN
25%      223.500000    0.000000    2.000000                  NaN   NaN
50%      446.000000    0.000000    3.000000                  NaN   NaN
75%      668.500000    1.000000    3.000000                  NaN   NaN
max      891.000000    1.000000    3.000000                  NaN   NaN

               Age       SibSp       Parch  Ticket        Fare Cabin Embarked
count   714.000000  891.000000  891.000000     891  891.000000   204      889
unique         NaN         NaN         NaN     681         NaN   147        3
top            NaN         NaN         NaN  347082         NaN    G6        S
freq           NaN         NaN         NaN       7         NaN     4      644
mean     29.699118    0.523008    0.381594     NaN   32.204208   NaN      NaN
std      14.526497    1.102743    0.806057     NaN   49.693429   NaN      NaN
min       0.420000    0.000000    0.000000     NaN    0.000000   NaN      NaN
25%      20.125000    0.000000    0.000000     NaN    7.910400   NaN      NaN
50%      28.000000    0.000000    0.000000     NaN   14.454200   NaN      NaN
75%      38.000000    1.000000    0.000000     NaN   31.000000   NaN      NaN
max      80.000000    8.000000    6.000000     NaN  512.329200   NaN      NaN
this is object describe
                       Name   Sex  Ticket Cabin Embarked
count                   891   891     891   204      889
unique                  891     2     681   147        3
top     Dooley, Mr. Patrick  male  347082    G6        S
freq                      1   577       7     4      644
```

8) For multivariate statistics, you can compute the covariance and correlation between pairs of attributes.

In [ ]:

In [38]:

```
print("correlation matrix : ")
print(df.corr(numeric_only=True))
```

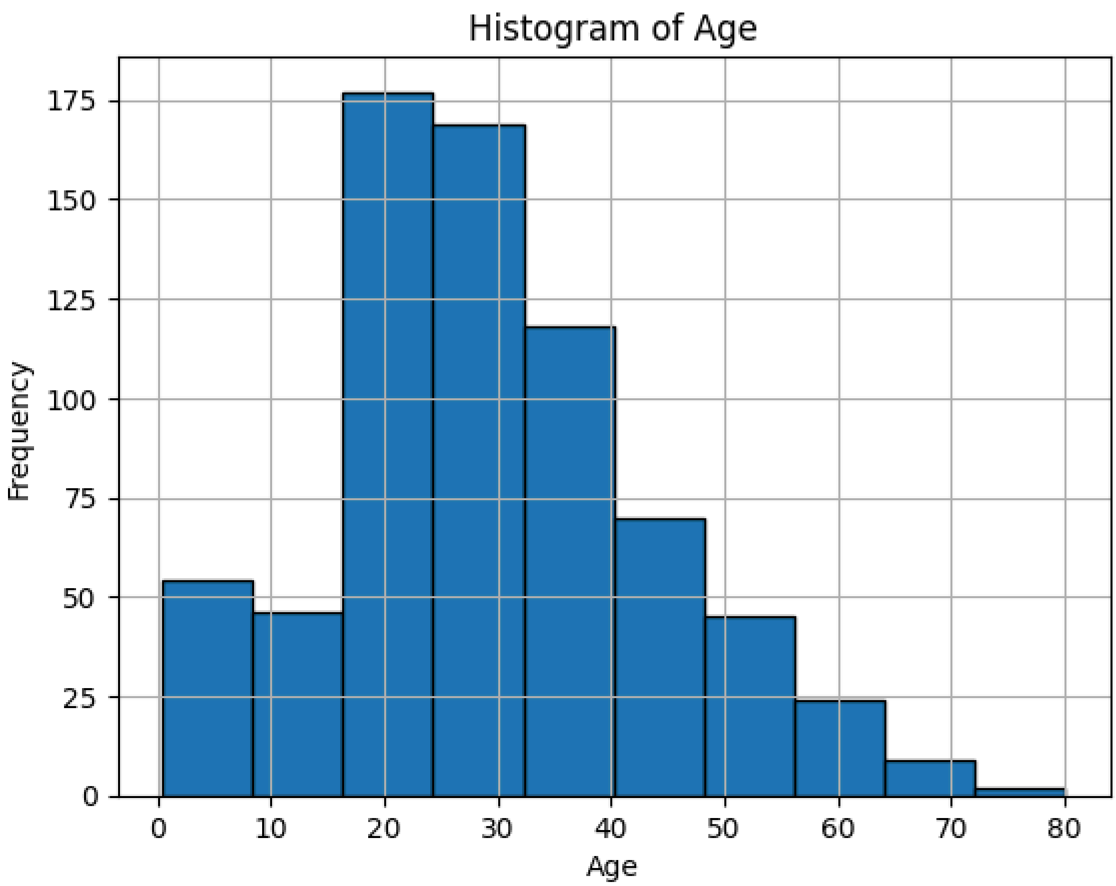
```
correlation matrix :
             PassengerId  Survived    Pclass       Age     SibSp     Parch  \
PassengerId     1.000000 -0.005007 -0.035144  0.036847 -0.057527 -0.001652
Survived       -0.005007  1.000000 -0.338481 -0.077221 -0.035322  0.081629
Pclass         -0.035144 -0.338481  1.000000 -0.369226  0.083081  0.018443
Age             0.036847 -0.077221 -0.369226  1.000000 -0.308247 -0.189119
SibSp          -0.057527 -0.035322  0.083081 -0.308247  1.000000  0.414838
Parch          -0.001652  0.081629  0.018443 -0.189119  0.414838  1.000000
Fare            0.012658  0.257307 -0.549500  0.096067  0.159651  0.216225

                 Fare
PassengerId  0.012658
Survived     0.257307
Pclass      -0.549500
Age          0.096067
SibSp        0.159651
Parch        0.216225
Fare         1.000000
```

9) Display the histogram for Age attribute by discretizing it into 8 separate bins and counting the frequency for each bin.

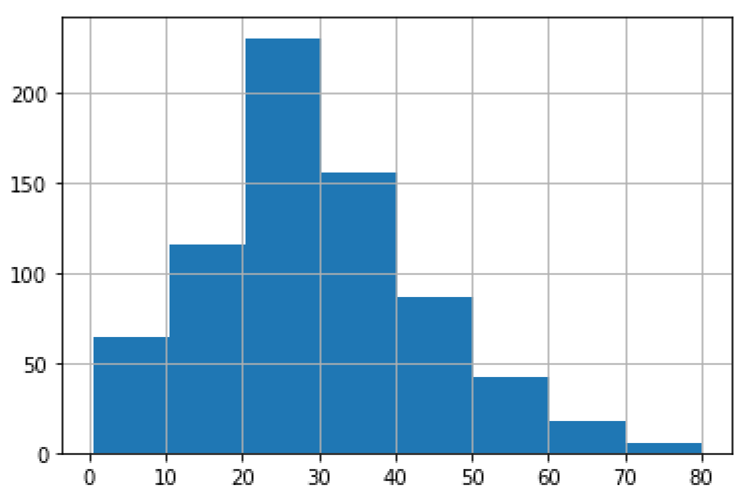
In [55]:

```
import matplotlib.pyplot as plt
df["Age"].dropna().hist(bins=10,edgecolor='black')
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



In [13]:

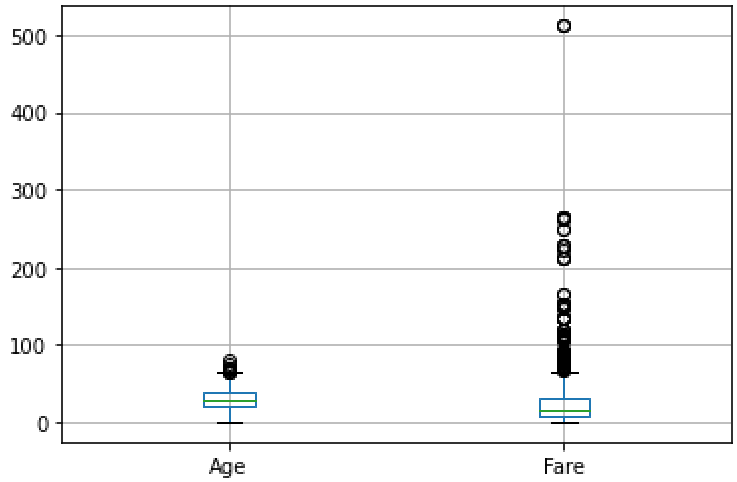
Out[13]: <AxesSubplot:>



10) A boxplot can also be used to show the distribution of values for each attribute.

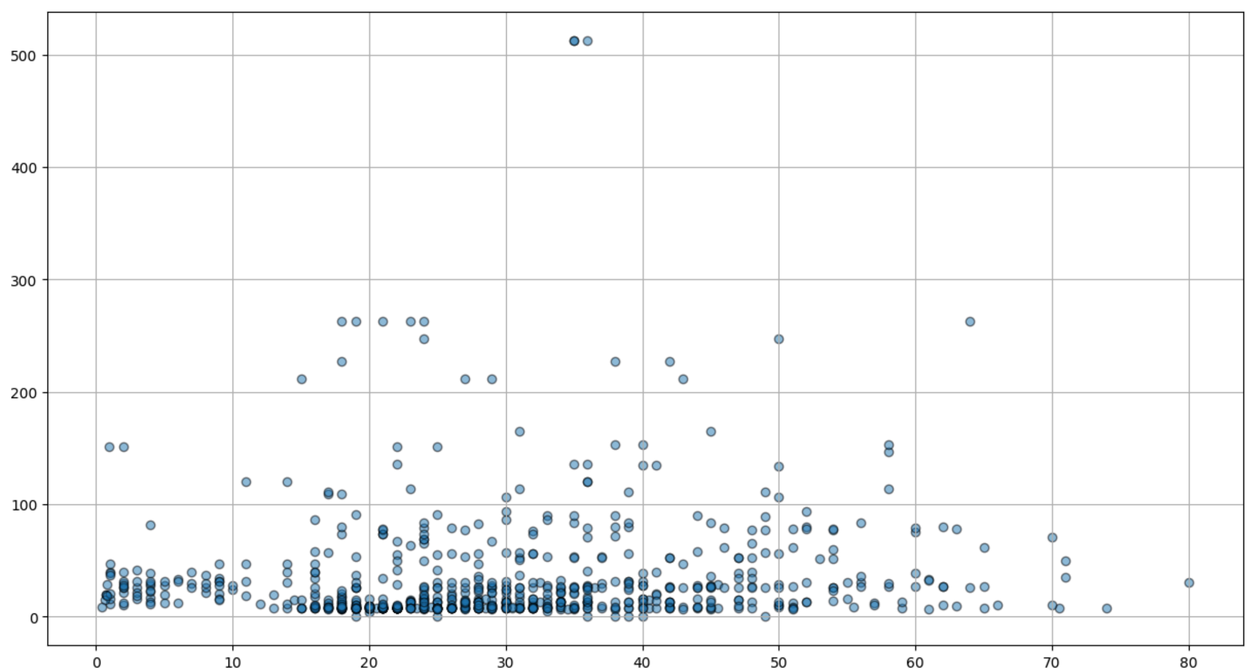
In [17]:

Out[17]: <AxesSubplot:>



11) Display scatter plot for any 5 pair of attributes , we can use a scatter plot to visualize their joint distribution.

In [54]:
```
plt.figure(figsize=(15,8))
plt.scatter(df['Age'],df['Fare'],alpha = 0.5, edgecolor = 'k')
plt.xlabel('')
plt.ylabel('')
plt.grid(True)
plt.show()
```

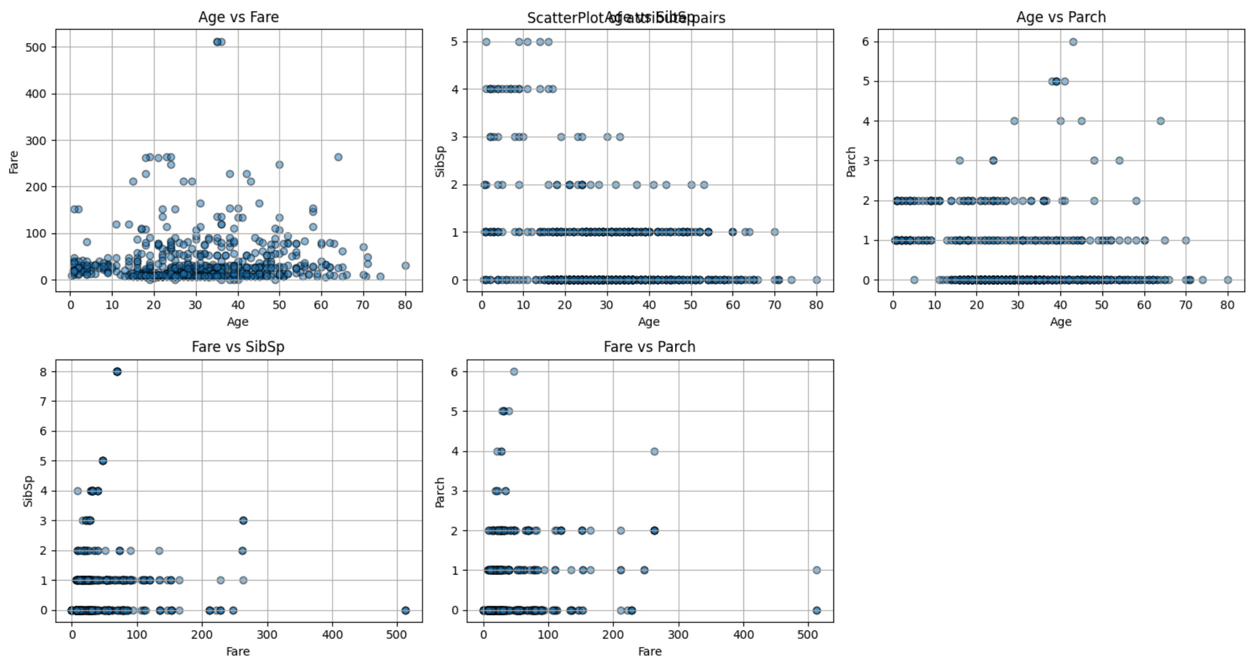


In [61]:
```
pairs = [
    ('Age','Fare'),('Age','SibSp'),('Age','Parch'),('Fare','SibSp'),('Fare','P
]
plt.figure(figsize=(15,8))
```

```
for i, (x,y) in enumerate(pairs):
    plt.subplot(2,3,i+1)
    plt.scatter(df[x],df[y],alpha=0.5,edgecolor='k')
    plt.xlabel(x)
    plt.ylabel(y)
    plt.title(f'{x} vs {y}')
    plt.grid(True)

plt.tight_layout()
plt.suptitle("ScatterPlot of attribute pairs")
```

Out[61]: Text(0.5, 0.98, 'ScatterPlot of attribute pairs')



In [ ]:

In [ ]:

Data Mining

Lab - 4

Prit Kanani 23010101126 142

Step 1. Import the necessary libraries

In [1]:
```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called chipo.

In [4]:
```
chipo = pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master
chipo
```

Out[4]:

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
...	...	...	...	...	...
4617	1833	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Sour ...	$11.75
4618	1833	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Sour Cream, Cheese...	$11.75
4619	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Pinto...	$11.25
4620	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Lettu...	$8.75
4621	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Pinto...	$8.75

4622 rows × 5 columns

Step 4. See the first 10 entries

In [5]:
```
chipo.head(10)
```

Out[5]:

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	$10.98
6	3	1	Side of Chips	NaN	$1.69
7	4	1	Steak Burrito	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	$11.75
8	4	1	Steak Soft Tacos	[Tomatillo Green Chili Salsa, [Pinto Beans, Ch...	$9.25
9	5	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Pinto...	$9.25

Step 5. What is the number of observations in the dataset?

In [12]:
```
# Solution 1

print(chipo.order_id.count())

chipo.shape[0]
```

4622

Out[12]: 4622

In [19]:
```
# Solution 2

chipo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4622 entries, 0 to 4621
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   order_id            4622 non-null   int64 
 1   quantity            4622 non-null   int64 
 2   item_name           4622 non-null   object
 3   choice_description  3376 non-null   object
 4   item_price          4622 non-null   object
dtypes: int64(2), object(3)
memory usage: 180.7+ KB
```

Step 6. What is the number of columns in the dataset?

In [35]:
```
chipo.shape[1]
```

Out[35]:
```
5
```

Step 7. Print the name of all the columns.

In [32]:
```
chipo.columns
```

Out[32]:
```
Index(['order_id', 'quantity', 'item_name', 'choice_description',
       'item_price'],
      dtype='object')
```

Step 8. How is the dataset indexed?

In [39]:
```
chipo.index
```

Out[39]:
```
RangeIndex(start=0, stop=4622, step=1)
```

Step 9. Number of Unique Items ?

In [41]:
```
print(chipo['item_name'].nunique())
```

```
50
```

Step 10. Which was the most-ordered item?

In [53]:
```
c = chipo.groupby('item_name')
c = c.sum()
c = c.sort_values(['quantity'],ascending = False)
c.head(1)
```

Out[53]:

item_name	order_id	quantity	choice_description	item_price
Chicken Bowl	713926	761	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98 $10.98 $11.25 $8.75 $8.49 $11.25 $8.75 ...

Step 11. How many items were orderd in total?

In [51]:
```
chipo['order_id'].nunique()
```

Out[51]: 1834

Step 12. Turn the item price into a float

Step 12.a. Check the item price type

In [48]:
```
chipo['item_price'].dtype
```

Out[48]: dtype('O')

Step 12.b. Create a lambda function and change the type of item price

In [59]:
```
a = lambda a : float(a[1:-1])
chipo.item_price = chipo.item_price.apply(a)
```

Step 12.c. Check the item price type

In [60]:
```
chipo['item_price'].dtype
```

Out[60]: dtype('float64')

Step 14. How much was the revenue for the period in the dataset?

In [74]:
```
revenue = (chipo.quantity * chipo.item_price).sum()
print(revenue)
```

```
39237.02
```

Step 15. How many orders were made ?

In [77]:
```
chipo.order_id.nunique()
```

Out[77]: 1834

Step 17. How many different choice descriptions are there?

In [78]:
```
chipo.choice_description.nunique()
```

Out[78]:
```
1043
```

Step 18. What items have been ordered more than 100 times?

In [80]:
```
item = chipo.groupby('item_name')['quantity'].sum()
item[item>100]
```

Out[80]:
```
item_name
Bottled Water                   211
Canned Soda                     126
Canned Soft Drink               351
Chicken Bowl                    761
Chicken Burrito                 591
Chicken Salad Bowl              123
Chicken Soft Tacos              120
Chips                           230
Chips and Fresh Tomato Salsa    130
Chips and Guacamole             506
Side of Chips                   110
Steak Bowl                      221
Steak Burrito                   386
Name: quantity, dtype: int64
```

Step 19. What is the average revenue amount per order?

In [85]:
```
# Solution 1
chipo['revenue'] = chipo.quantity * chipo.item_price
g = chipo.groupby(by=['order_id']).sum()
print(g['revenue'].mean())
```

```
21.39423118865867
```

In [88]:
```
# Solution 2
```

Out[88]:
```
21.39423118865867
```

Lab - 5 - Data Preprocessing

1) First, you need to read the titanic dataset from local disk and display Last five records

In [73]:
```
import pandas as pd
df = pd.read_csv('./../Lab_1/titanic.csv')
```

In [71]:
```
df.tail(5)
```

Out[71]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticke
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	21153
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	11205
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C 660
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	11136
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	37037

2) Handle Missing Values in data set [use dropna(), fillna(), and interpolate]

In [20]:
```
df.dropna()
```

Out[20]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	175
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	174
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	95
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	1137
...	...	...	...	...	...	...	...	...	
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.0	1	1	117
872	873	0	1	Carlsson, Mr. Frans Olof	male	33.0	0	0	6
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	117
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113

183 rows × 12 columns

In [78]:

```
df.fillna(value=0)
```

Out[78]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticl
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STO 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	0.0	1	2	W 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3703

891 rows × 14 columns

In [68]:

```
df.fillna({'Age' : df['Age'].mean() , 'Cabin' : 'Not Available'})
```

Out[68]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0
...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.000000	0	0
887	888	1	1	Graham, Miss. Margaret Edith	female	19.000000	0	0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.699118	1	2
889	890	1	1	Behr, Mr. Karl Howell	male	26.000000	0	0
890	891	0	3	Dooley, Mr. Patrick	male	32.000000	0	0

891 rows × 13 columns

In [63]:

```
df.interpolate()
```

```
C:\Users\LENOVO\AppData\Local\Temp\ipykernel_3520\4002874584.py:1: FutureWarnin
g: DataFrame.interpolate with object dtype is deprecated and will raise in a fu
ture version. Call obj.infer_objects(copy=False) before interpolating instead.
  df.interpolate()
```

Out[63]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STO 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	22.5	1	2	W 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3703

891 rows × 13 columns

3) Apply Scaling to AGE attribute with min max, decimal scaling and z score.

In [41]:

```
df.fillna(df.Age.mean(),inplace=True)
mina = d.Age.min()
maxa = d.Age.max()
df['MinMaxAge'] = (df['Age'] - mina)/(maxa-mina)
df
```

Out[41]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0
...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.000000	0	0
887	888	1	1	Graham, Miss. Margaret Edith	female	19.000000	0	0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.699118	1	2
889	890	1	1	Behr, Mr. Karl Howell	male	26.000000	0	0
890	891	0	3	Dooley, Mr. Patrick	male	32.000000	0	0

891 rows × 13 columns

In [74]:

```
lenofage =  len(str(int(df.Age.max())))
df['nAge'] = (df['Age'])/10**lenofage
df
```

Out[74]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticl
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STC (31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3703

891 rows × 13 columns

In [79]:

```
meana = df.Age.mean()
stda = df.Age.std()
df['newAge'] = (df['Age'] - meana)/stda
df
```

Out[79]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	211
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	175
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STO 31012
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	1138
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	3734
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	2115
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	1120
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W 66
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	3703

891 rows × 14 columns

In [ ]:

Data Mining

Lab - 6

Prit kanani 142

```
# Dimensionality Reduction using NumPy
```

⯑ What is Data Reduction?

Data reduction refers to the process of reducing the amount of data that needs to be processed and stored, while preserving the essential patterns in the data.

Why do we reduce data?

- To reduce computational cost.
- To remove noise and redundant features.
- To improve model performance and training time.
- To visualize high-dimensional data in 2D or 3D.

Common data reduction techniques include:

- Principal Component Analysis (PCA)
- Feature selection
- Sampling

⯑ What is Principal Component Analysis (PCA)?

PCA is a **dimensionality reduction technique** that transforms a dataset into a new coordinate system. It identifies the **directions (principal components)** where the variance of the data is maximized.

Key Concepts:

- **Principal Components**: New features (linear combinations of original features) capturing most variance.
- **Eigenvectors & Eigenvalues**: Used to compute these principal directions.
- **Covariance Matrix**: Measures how features vary with each other.

PCA helps in **visualizing high-dimensional data**, **noise reduction**, and **speeding up algorithms**.

⍰ NumPy Functions Summary for PCA

Function	Purpose
`np.mean(X, axis=0)`	Compute mean of each column (feature-wise mean).
`X - np.mean(X, axis=0)`	Centering the data (zero mean).
`np.cov(X, rowvar=False)`	Compute covariance matrix for features.
`np.linalg.eigh(cov_mat)`	Get eigenvalues and eigenvectors (for symmetric matrices).
`np.argsort(values)[::-1]`	Sort values in descending order.
`np.dot(X, eigenvectors)`	Project original data onto new axes.

Step 1: Load the Iris Dataset

In [13]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
iris = pd.read_csv('iris.csv')
print(iris.shape)
iris
```

```
(150, 5)
```

Out[13]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [14]:
```
x = iris.drop(columns='species')
y = iris['species'].map({'setosa':0,'versicolor':1,'virginica':2})
print(x.shape)
```

```
(150, 4)
```

Step 2: Standardize the data (zero mean)

In [16]:
```
x_meaned = x - np.mean(x,axis=0)
print(x_meaned.head(5))
```

```
   sepal_length  sepal_width  petal_length  petal_width
0     -0.743333     0.442667        -2.358    -0.999333
1     -0.943333    -0.057333        -2.358    -0.999333
2     -1.143333     0.142667        -2.458    -0.999333
3     -1.243333     0.042667        -2.258    -0.999333
4     -0.843333     0.542667        -2.358    -0.999333
```

Step 3: Compute the Covariance Matrix

In [17]:
```
cov_mat = np.cov(x_meaned,rowvar=False)
print(cov_mat.shape)
```

```
(4, 4)
```

Step 4: Compute eigenvalues and eigenvectors

In [26]:
```
eigen_value,eigen_vectors = np.linalg.eigh(cov_mat)
print(eigen_value)
print(eigen_vectors[:,:2])
```

```
[0.02383509 0.0782095  0.24267075 4.22824171]
[[ 0.31548719  0.58202985]
 [-0.3197231  -0.59791083]
 [-0.47983899 -0.07623608]
 [ 0.75365743 -0.54583143]]
```

Step 5: Compute eigenvalues and eigenvectors

In [27]:
```
sorted_index = np.argsort(eigen_value[::-1])
sorted_eigenvalues = eigen_value[sorted_index]
sorted_eigenvecotores = eigen_vectors[:,sorted_index]
print(sorted_index)
print(sorted_eigenvalues)
print(sorted_eigenvecotores)
```

```
[3 2 1 0]
[4.22824171 0.24267075 0.0782095  0.02383509]
[[-0.36138659  0.65658877  0.58202985  0.31548719]
 [ 0.08452251  0.73016143 -0.59791083 -0.3197231 ]
 [-0.85667061 -0.17337266 -0.07623608 -0.47983899]
 [-0.3582892  -0.07548102 -0.54583143  0.75365743]]
```

Step 6: Select the top k eigenvectors (top 2)

In [31]:
```
k = 2
eigenvector_subset = sorted_eigenvecotores[:,0:k]
print(eigenvector_subset)
```

```
[[-0.36138659  0.65658877]
 [ 0.08452251  0.73016143]
 [-0.85667061 -0.17337266]
 [-0.3582892  -0.07548102]]
```

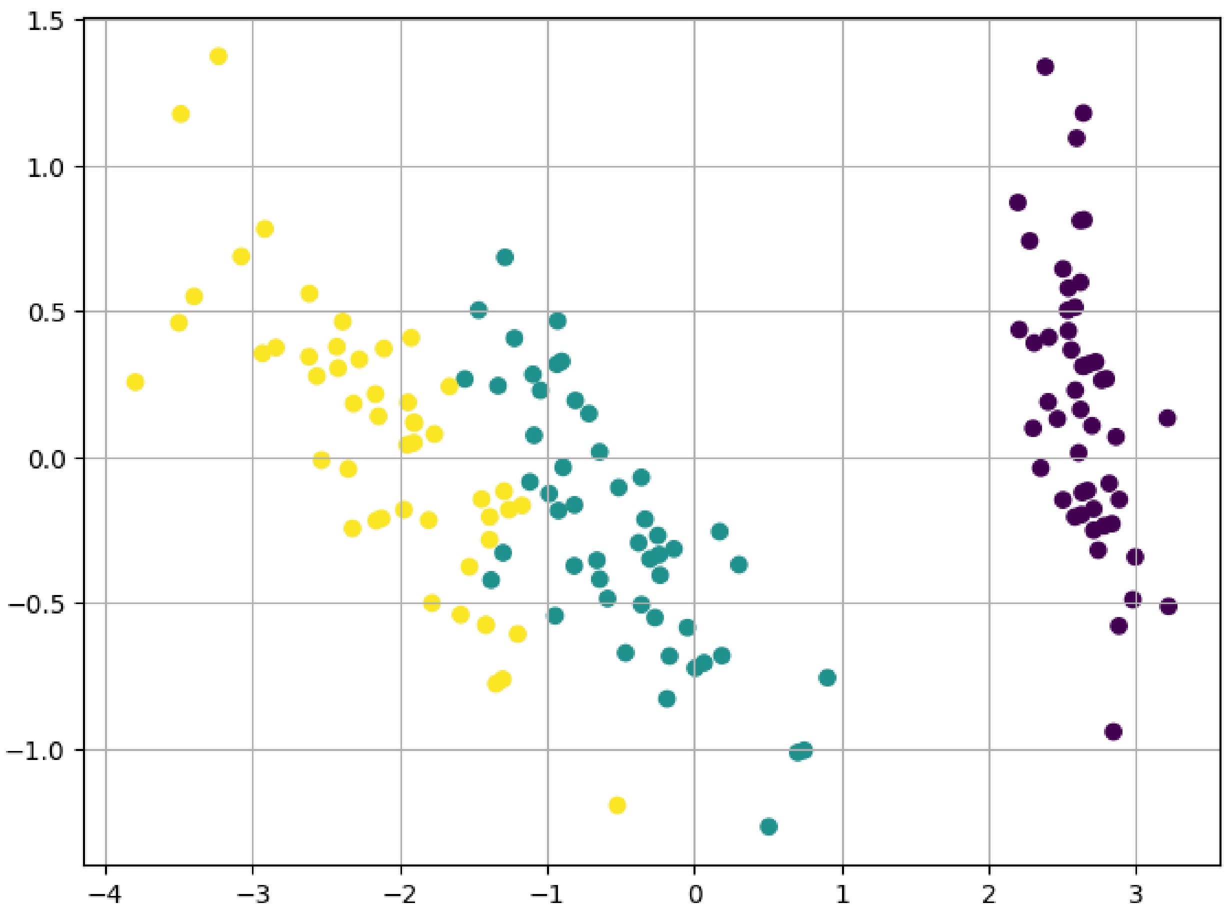
Step 7: Project the data onto the top k eigenvectors

In [34]:
```
x_reduced = np.dot(x_meaned,eigenvector_subset)
print(x_reduced.shape)
```

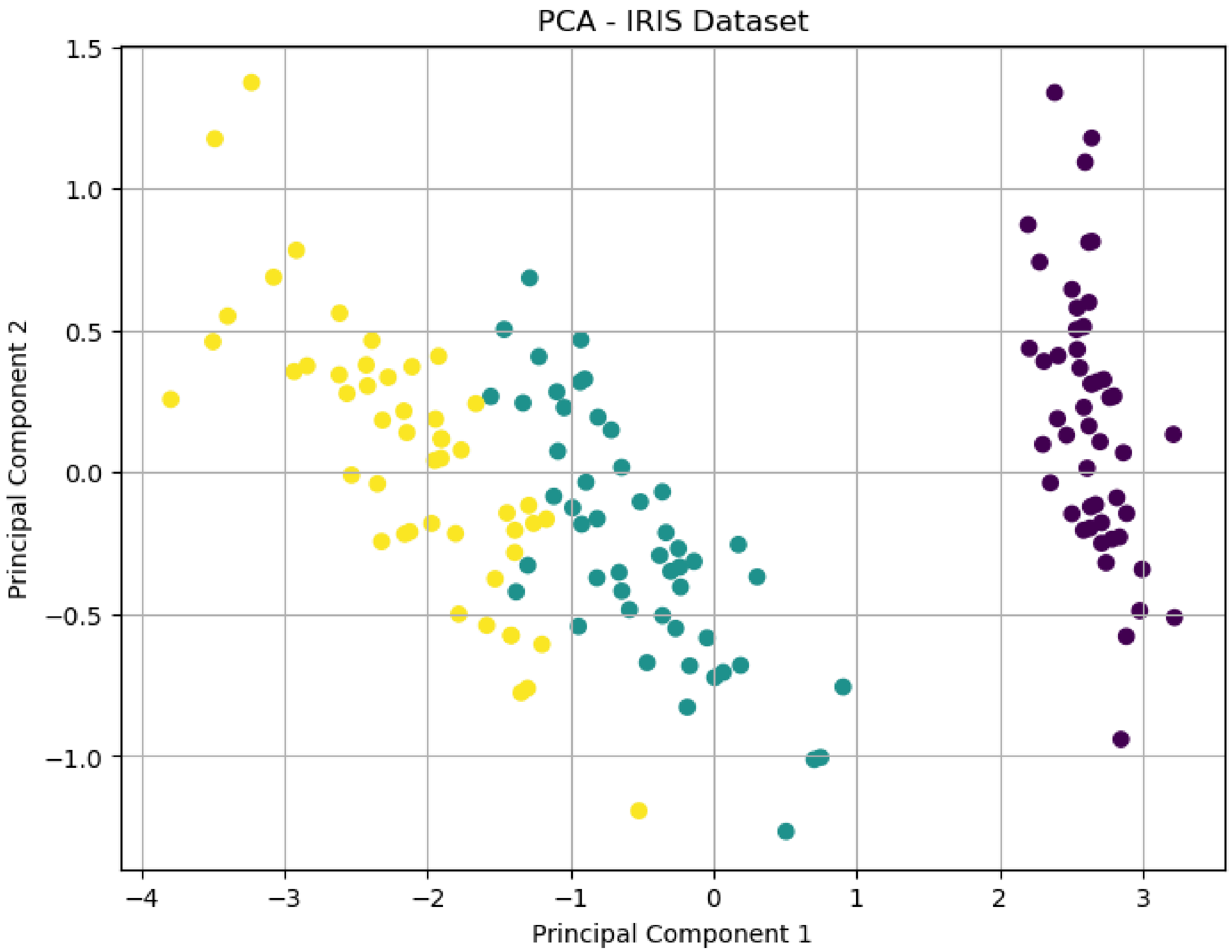
```
(150, 2)
```

Step 8: Plot the PCA-Reduced Data

In [35]:
```
plt.figure(figsize=(8,6))
plt.scatter(x_reduced[:,0],x_reduced[:,1],c=y)
plt.grid(True)
plt.show()
```



In [21]:



Extra - Bining Method

5,10,11,13,15,35,50,55,72,92,204,215.

Partition them into three bins by each of the following methods: (a) equal-frequency (equal-depth) partitioning (b) equal-width partitioning

In [37]:
```
values = np.array([5,10,11,13,15,35,50,55,72,92,204,215])
equal_freq_bins = pd.qcut(values,q=3,labels=['bin1','bin2','bin3'])
equal_width_bins = pd.cut(values,bins=3,labels=['bin1','bin2','bin3'])
binning_results = pd.DataFrame({'values':values,'Equal-Frequency-Bin':equal_fr
binning_results
```

Out[37]:

	values	Equal-Frequency-Bin	equal_width_bins
0	5	bin1	bin1
1	10	bin1	bin1
2	11	bin1	bin1
3	13	bin1	bin1
4	15	bin2	bin1
5	35	bin2	bin1
6	50	bin2	bin1
7	55	bin2	bin1
8	72	bin3	bin1
9	92	bin3	bin2
10	204	bin3	bin3
11	215	bin3	bin3

In [ ]:

```

```

Data Mining

Lab - 7 (Part 2)

Step 1: Load the Dataset

Load the `Tdata.csv` file and display the first few rows.

In [9]:
```
import pandas as pd
df = pd.read_csv("Tdata.csv")
df
```

Out[9]:

	Transaction	bread	butter	coffee	eggs	jam	milk
0	T1	1	1	0	0	0	1
1	T2	1	1	0	0	1	0
2	T3	1	0	0	1	0	1
3	T4	1	1	0	0	0	1
4	T5	1	0	1	0	0	0
5	T6	0	0	1	1	1	0

Step 2: Drop the 'Transaction' Column

We're only interested in the items (not the transaction IDs).

In [17]:
```
df
```

Out[17]:

	bread	butter	coffee	eggs	jam	milk
0	1	1	0	0	0	1
1	1	1	0	0	1	0
2	1	0	0	1	0	1
3	1	1	0	0	0	1
4	1	0	1	0	0	0
5	0	0	1	1	1	0

Step 3: Count Single Items

See how many transactions include each item.

In [19]:

```
df.sum()
```

Out[19]:

```
bread     5
butter    3
coffee    2
eggs      2
jam       2
milk      3
dtype: int64
```

Step 4: Define Apriori Function

This function finds frequent itemsets of size 1, 2, and 3 with minimum support.

In [40]:

```
from itertools import combinations
def find_frequent_itemsets(df,min_support):
    n = len(df)
    result = []
    for k in [1,2,3]:
        for items in combinations(df.columns,k):
            mask = df[list(items)].all(axis=1)
            support = mask.sum()/n
            if support >= min_support:
                result.append((frozenset(items),round(support,2)))
    return result
```

Step 5: Run Apriori

Set `min_support = 0.6` and display the frequent itemsets.

In [41]:

```
frequent_itemsets = find_frequent_itemsets(df,min_support=0.6)
```

```
for itemset, support in frequent_itemsets:
    print(f"{set(itemset)} --->support : {support}")
```

```
{'bread'} --->support : 0.83
```

Step 6 Display as a DataFrame

In [46]:
```
data = pd.DataFrame(frequent_itemsets,columns=["itemset","support"])
data
```

Out[46]:

	itemset	support
0	(bread)	0.83

In [ ]:
```
```

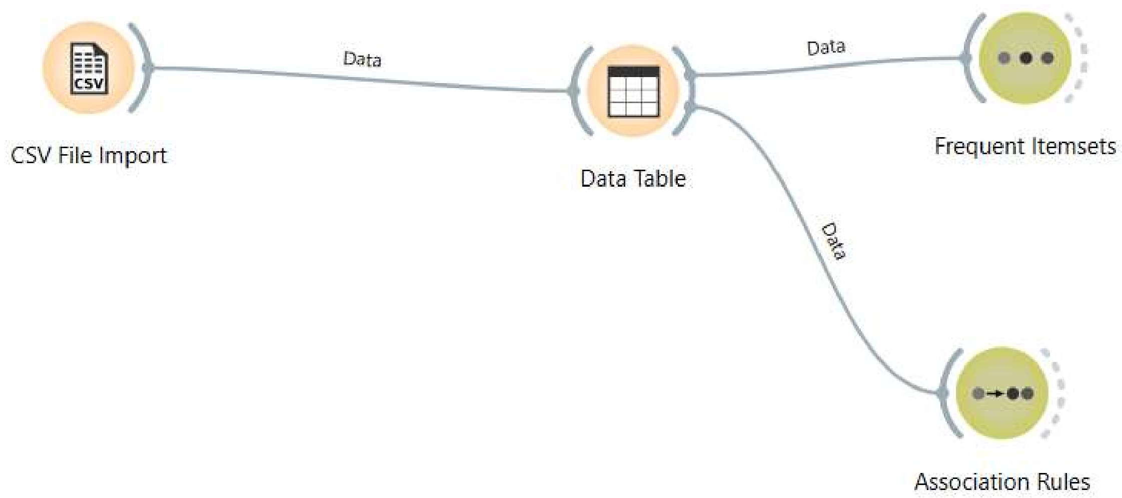
Orange Tool : - >Generate Same Frequent Patterns in Orange tools

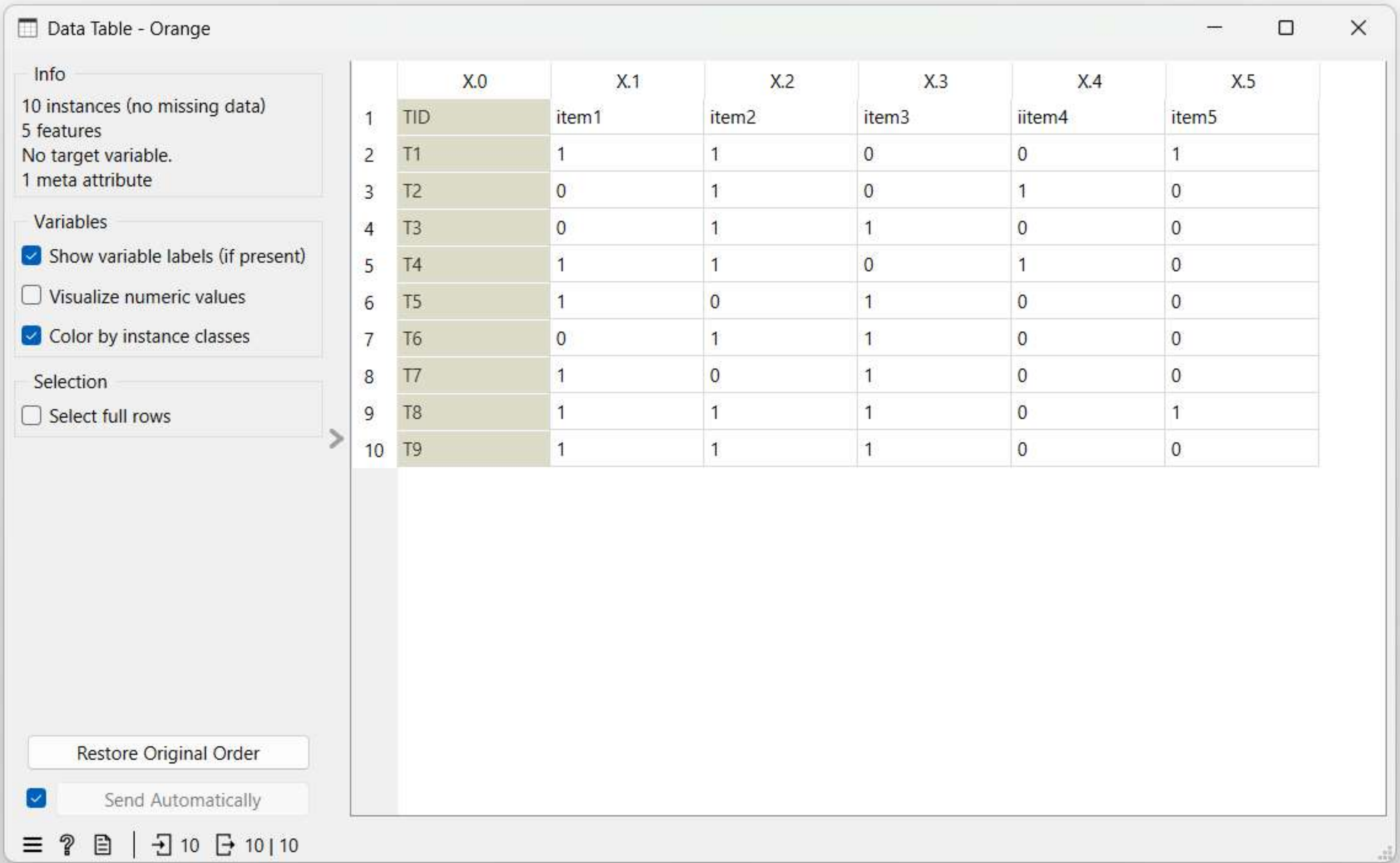
In [ ]:
```
```

Extra : - > Define Apriori Function without itertools

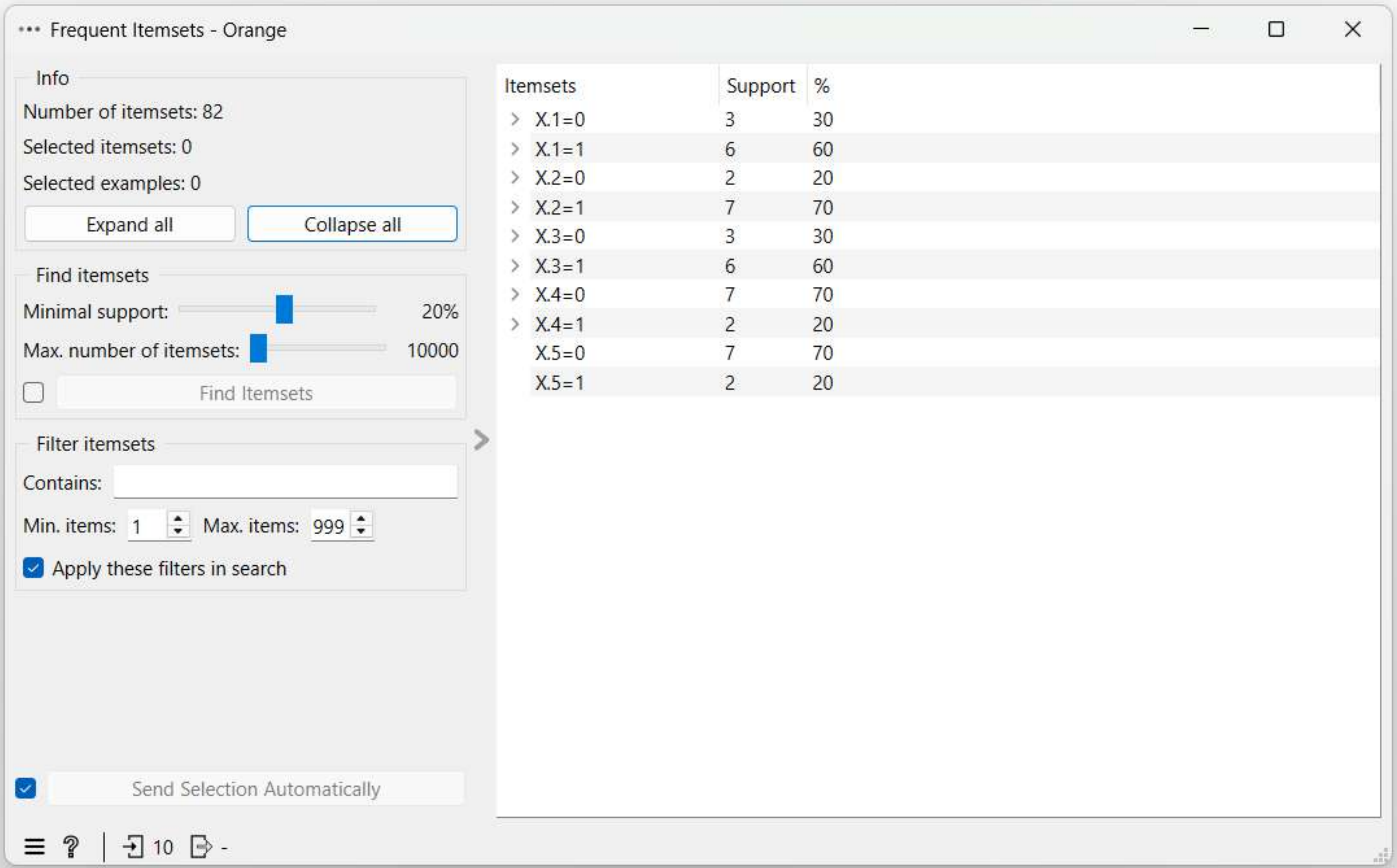
In [ ]:
```
```

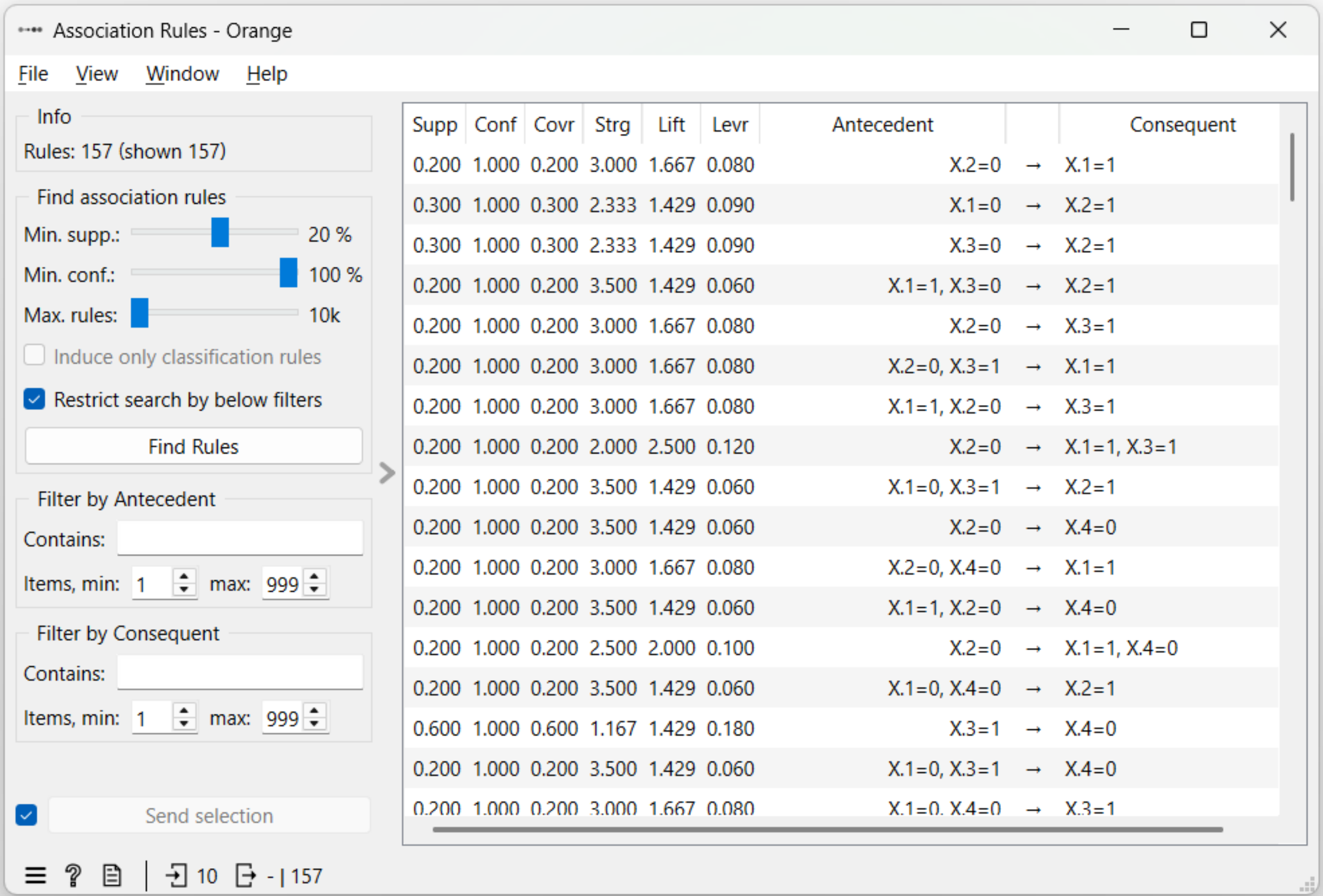
In [ ]:
```
```


Data
Data
Data
CSV File Import
Data Table
Frequent Itemsets
Association Rules

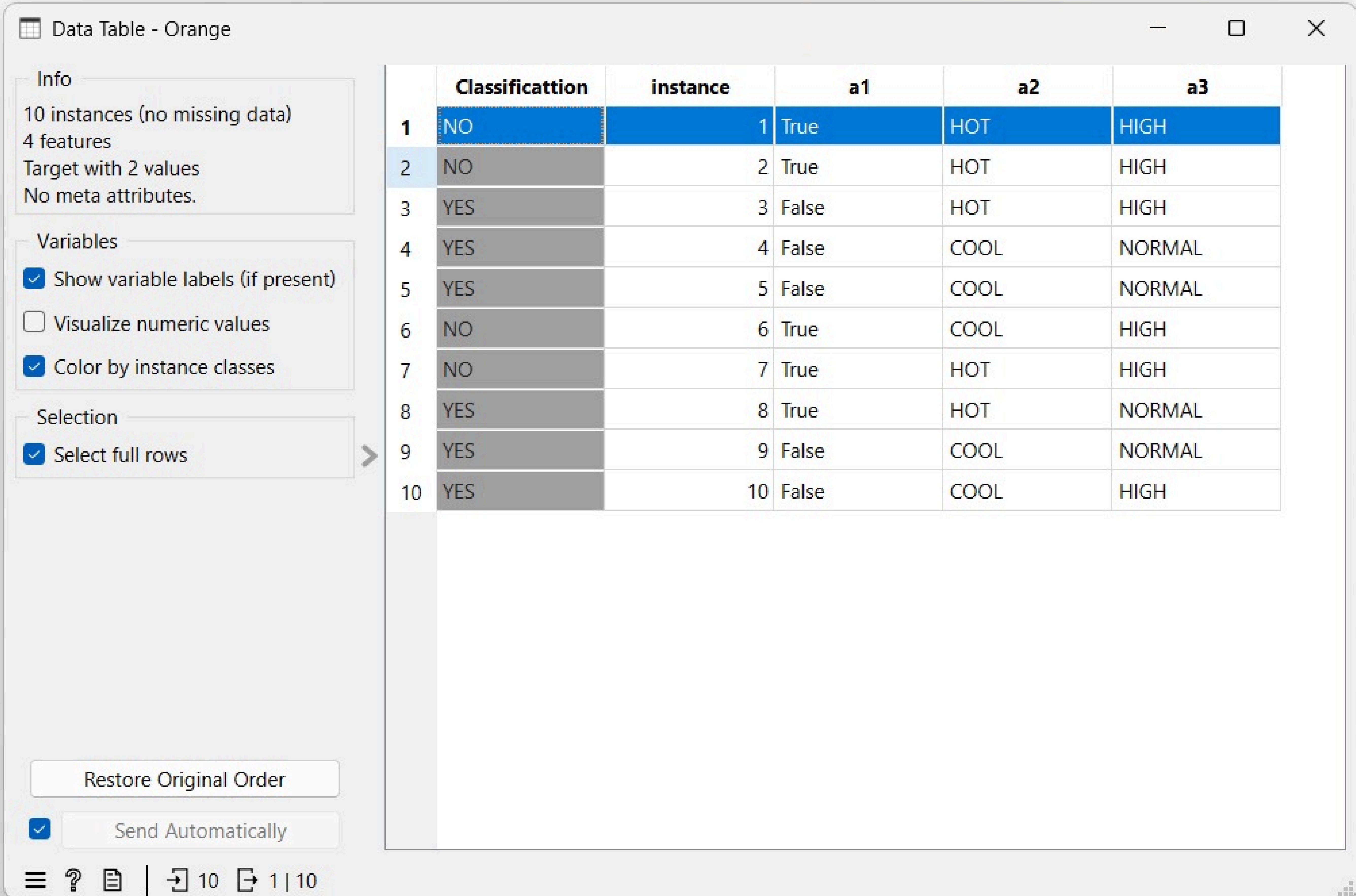


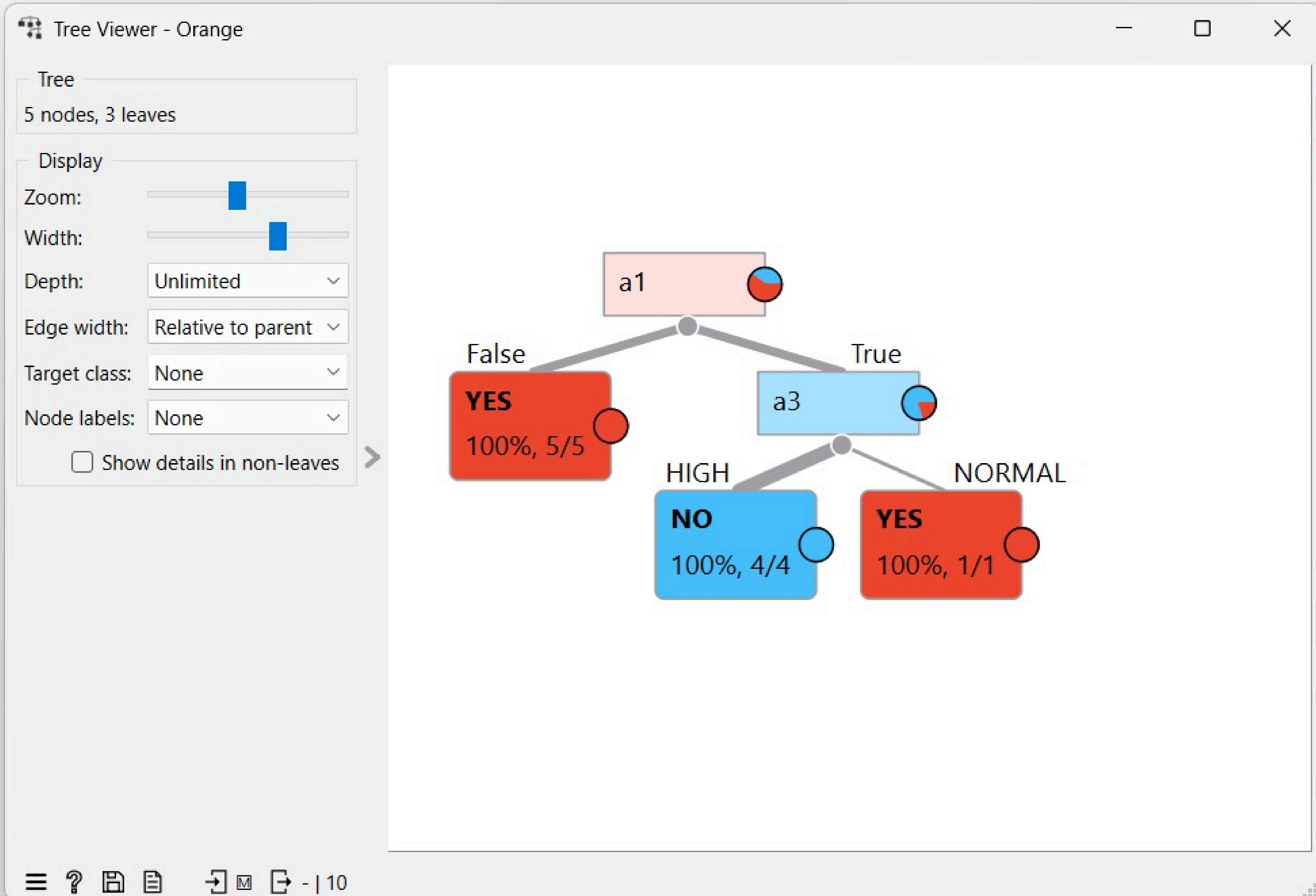
	X.0	X.1	X.2	X.3	X.4	X.5
1	TID	item1	item2	item3	iitem4	item5
2	T1	1	1	0	0	1
3	T2	0	1	0	1	0
4	T3	0	1	1	0	0
5	T4	1	1	0	1	0
6	T5	1	0	1	0	0
7	T6	0	1	1	0	0
8	T7	1	0	1	0	0
9	T8	1	1	1	0	1
10	T9	1	1	1	0	0


Frequent Itemsets - Orange
Info
Number of itemsets: 82
Selected itemsets: 0
Selected examples: 0
Expand all
Collapse all
Find itemsets
Minimal support:
20%
Max. number of itemsets:
10000
Find Itemsets
Filter itemsets
Contains:
Min. items: 1
Max. items: 999
Apply these filters in search
Send Selection Automatically
10
Itemsets
Support
%
X.1=0 3 30
X.1=1 6 60
X.2=0 2 20
X.2=1 7 70
X.3=0 3 30
X.3=1 6 60
X.4=0 7 70
X.4=1 2 20
X.5=0 7 70
X.5=1 2 20


Association Rules - Orange
File
View
Window
Help
Info
Rules: 157 (shown 157)
Find association rules
Min. supp.:
20 %
Min. conf.:
100 %
Max. rules:
10k
Induce only classification rules
Restrict search by below filters
Find Rules
Filter by Antecedent
Contains:
Items, min: 1 max: 999
Filter by Consequent
Contains:
Items, min: 1 max: 999
Send selection
10
- | 157
Supp Conf Covr Strg Lift Levr Antecedent Consequent
0.200 1.000 0.200 3.000 1.667 0.080 X.2=0 → X.1=1
0.300 1.000 0.300 2.333 1.429 0.090 X.1=0 → X.2=1
0.300 1.000 0.300 2.333 1.429 0.090 X.3=0 → X.2=1
0.200 1.000 0.200 3.500 1.429 0.060 X.1=1, X.3=0 → X.2=1
0.200 1.000 0.200 3.000 1.667 0.080 X.2=0 → X.3=1
0.200 1.000 0.200 3.000 1.667 0.080 X.2=0, X.3=1 → X.1=1
0.200 1.000 0.200 3.000 1.667 0.080 X.1=1, X.2=0 → X.3=1
0.200 1.000 0.200 2.000 2.500 0.120 X.2=0 → X.1=1, X.3=1
0.200 1.000 0.200 3.500 1.429 0.060 X.1=0, X.3=1 → X.2=1
0.200 1.000 0.200 3.500 1.429 0.060 X.2=0 → X.4=0
0.200 1.000 0.200 3.000 1.667 0.080 X.2=0, X.4=0 → X.1=1
0.200 1.000 0.200 3.500 1.429 0.060 X.1=1, X.2=0 → X.4=0
0.200 1.000 0.200 2.500 2.000 0.100 X.2=0 → X.1=1, X.4=0
0.200 1.000 0.200 3.500 1.429 0.060 X.1=0, X.4=0 → X.2=1
0.600 1.000 0.600 1.167 1.429 0.180 X.3=1 → X.4=0
0.200 1.000 0.200 3.500 1.429 0.060 X.1=0, X.3=1 → X.4=0
0.200 1.000 0.200 3.000 1.667 0.080 X.1=0, X.4=0 → X.3=1


Data
Data
Data
Model → Tree
File
Data Table
Select Columns
Tree
Tree Viewer


Data Table - Orange
Info
10 instances (no missing data)
4 features
Target with 2 values
No meta attributes.
Variables
Show variable labels (if present)
Visualize numeric values
Color by instance classes
Selection
Select full rows
Restore Original Order
Send Automatically
10
1 | 10
Classificattion instance a1 a2 a3
1 NO 1 True HOT HIGH
2 NO 2 True HOT HIGH
3 YES 3 False HOT HIGH
4 YES 4 False COOL NORMAL
5 YES 5 False COOL NORMAL
6 NO 6 True COOL HIGH
7 NO 7 True HOT HIGH
8 YES 8 True HOT NORMAL
9 YES 9 False COOL NORMAL
10 YES 10 False COOL HIGH


Tree Viewer - Orange
Tree
5 nodes, 3 leaves
Display
Zoom:
Width:
Depth: Unlimited
Edge width: Relative to parent
Target class: None
Node labels: None
Show details in non-leaves
a1
False
True
YES
100%, 5/5
a3
HIGH
NORMAL
NO
100%, 4/4
YES
100%, 1/1
- | 10

Data Mining Lab - 10
Prit kanani 142

Implement Decision Tree(ID3) in python

Uses Information Gain to choose the best feature to split.

Recursively builds the tree until stopping conditions are met.

1) Calculate Entropy for the dataset. 2) Calculate Information Gain for each feature. 3) Choose the feature with maximum Information Gain. 4) Split dataset into subsets for that feature. 5) Repeat recursively until:

All samples in a node have the same label. No features are left. No data is left.

Step 2. Import the dataset from this address.

import Pandas, Numpy

```
import pandas as pd
import numpy as np
```

Create Following Data

```
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain',
'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast',
'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool',
'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal',
'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal',
'High'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong',
'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
'Strong'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
})

data
```

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes

```
5       Rain        Cool   Normal  Strong          No
6   Overcast        Cool   Normal  Strong         Yes
7      Sunny        Mild     High    Weak          No
8      Sunny        Cool   Normal    Weak         Yes
9       Rain        Mild   Normal    Weak         Yes
10     Sunny        Mild   Normal  Strong         Yes
11  Overcast        Mild     High  Strong         Yes
12  Overcast         Hot   Normal    Weak         Yes
13      Rain        Mild     High  Strong          No
```

Now Define Function to Calculate Entropy

```
def entropy(y):
    values, counts = np.unique(y,return_counts=True)
    print("values = ",values,"\ncounts = ",counts)
    probabilities = counts/ counts.sum()
    print('probabilities = ',probabilities)
    return -np.sum(probabilities * np.log2(probabilities))
```

Testing of Above Function -

y = np.array(['Yes', 'No', 'Yes', 'Yes'])
Function Call - > entropy(y))

output - 0.8112781244591328

```
y = np.array(['Yes', 'No', 'Yes', 'Yes'])
entropy(y)

values =  ['No' 'Yes']
counts =  [1 3]
probabilities =  [0.25 0.75]

np.float64(0.8112781244591328)
```

Define function to Calculate Information Gain

```
def information_gain(data, split_attribute, target):
    total_entropy = entropy(data[target])
    print(total_entropy)
    values, counts =
np.unique(data[split_attribute],return_counts=True)
    print("values = ",values,"\ncounts = ",counts)

    weighted_entropy = 0
    for i in range(len(values)):
        subset = data[data[split_attribute] == values[i]]
        print(subset)
        weighted_entropy += (counts[i] / counts.sum())*
```

```
entropy(subset[target])
        print(weighted_entropy)
    return total_entropy - weighted_entropy
```

Testing of Above Function-

data = pd.DataFrame({ 'Weather': ['Sunny', 'Sunny', 'Rain', 'Rain'], 'Play': ['Yes', 'No', 'Yes', 'Yes'] })

Function Call - > information_gain(data, 'Weather', 'Play')

Output - 0.31127812445913283

```
data = pd.DataFrame({ 'Weather': ['Sunny', 'Sunny', 'Rain', 'Rain'],
'Play': ['Yes', 'No', 'Yes', 'Yes'] })
information_gain(data, 'Weather', 'Play')

values =  ['No' 'Yes']
counts =  [1 3]
probabilities =  [0.25 0.75]
0.8112781244591328
values =  ['Rain' 'Sunny']
counts =  [2 2]
  Weather Play
2    Rain  Yes
3    Rain  Yes
values =  ['Yes']
counts =  [2]
probabilities =  [1.]
0.0
  Weather Play
0   Sunny  Yes
1   Sunny   No
values =  ['No' 'Yes']
counts =  [1 1]
probabilities =  [0.5 0.5]
0.5

np.float64(0.31127812445913283)
```

Implement ID3 Algo

```
def id3(data, features, target):
    # If all labels are same → return the label
    if len(np.unique(data[target])) == 1:
        return np.unique(data[target])[0]

    # If no features left → return majority label
    if len(features) == 0:
```

```
        return data[target].mode()[0]

    # Choose best feature
    gains = [information_gain(data,feature,target) for feature in
features]
    best_feature = features[np.argmax(gains)]

    tree = {best_feature: {}}
    # For each value of best feature → branch
    for value in np.unique(data[best_feature]):
        sub_data = data[data[best_feature] == value].drop(columns =
[best_feature])
        subtree = id3(sub_data, [f for f in features if f !=
best_feature], target)
        tree[best_feature][value] = subtree

    return tree
```

Use ID3

```
features = list(data.columns[:-1])
target = 'PlayTennis'
tree = id3(data, features, target)
```

```
values =  ['No' 'Yes']
counts =  [5 9]
probabilities =  [0.35714286 0.64285714]
0.9402859586706311
values =  ['Overcast' 'Rain' 'Sunny']
counts =  [4 5 5]
     Outlook Temperature Humidity    Wind PlayTennis
2   Overcast         Hot     High    Weak        Yes
6   Overcast        Cool   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
12  Overcast         Hot   Normal    Weak        Yes
values =  ['Yes']
counts =  [4]
probabilities =  [1.]
0.0
   Outlook Temperature Humidity    Wind PlayTennis
3     Rain        Mild     High    Weak        Yes
4     Rain        Cool   Normal    Weak        Yes
5     Rain        Cool   Normal  Strong         No
9     Rain        Mild   Normal    Weak        Yes
13    Rain        Mild     High  Strong         No
values =  ['No' 'Yes']
counts =  [2 3]
probabilities =  [0.4 0.6]
0.3467680694480959
```

```
   Outlook Temperature Humidity    Wind PlayTennis
0    Sunny         Hot     High    Weak         No
1    Sunny         Hot     High  Strong         No
7    Sunny        Mild     High    Weak         No
8    Sunny        Cool   Normal    Weak        Yes
10   Sunny        Mild   Normal  Strong        Yes
values =  ['No' 'Yes']
counts =  [3 2]
probabilities =  [0.6 0.4]
0.6935361388961918
values =  ['No' 'Yes']
counts =  [5 9]
probabilities =  [0.35714286 0.64285714]
0.9402859586706311
values =  ['Cool' 'Hot' 'Mild']
counts =  [4 4 6]
    Outlook Temperature Humidity    Wind PlayTennis
4      Rain        Cool   Normal    Weak        Yes
5      Rain        Cool   Normal  Strong         No
6  Overcast        Cool   Normal  Strong        Yes
8     Sunny        Cool   Normal    Weak        Yes
values =  ['No' 'Yes']
counts =  [1 3]
probabilities =  [0.25 0.75]
0.23179374984546652
     Outlook Temperature Humidity    Wind PlayTennis
0      Sunny         Hot     High    Weak         No
1      Sunny         Hot     High  Strong         No
2   Overcast         Hot     High    Weak        Yes
12  Overcast         Hot   Normal    Weak        Yes
values =  ['No' 'Yes']
counts =  [2 2]
probabilities =  [0.5 0.5]
0.5175080355597522
     Outlook Temperature Humidity    Wind PlayTennis
3       Rain        Mild     High    Weak        Yes
7      Sunny        Mild     High    Weak         No
9       Rain        Mild   Normal    Weak        Yes
10     Sunny        Mild   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
13      Rain        Mild     High  Strong         No
values =  ['No' 'Yes']
counts =  [2 4]
probabilities =  [0.33333333 0.66666667]
0.9110633930116763
values =  ['No' 'Yes']
counts =  [5 9]
probabilities =  [0.35714286 0.64285714]
0.9402859586706311
```

```
values =  ['High' 'Normal']
counts =  [7 7]
     Outlook Temperature Humidity    Wind PlayTennis
0      Sunny         Hot     High    Weak         No
1      Sunny         Hot     High  Strong         No
2   Overcast         Hot     High    Weak        Yes
3       Rain        Mild     High    Weak        Yes
7      Sunny        Mild     High    Weak         No
11  Overcast        Mild     High  Strong        Yes
13      Rain        Mild     High  Strong         No
values =  ['No' 'Yes']
counts =  [4 3]
probabilities =  [0.57142857 0.42857143]
0.49261406801712576
     Outlook Temperature Humidity    Wind PlayTennis
4       Rain        Cool   Normal    Weak        Yes
5       Rain        Cool   Normal  Strong         No
6   Overcast        Cool   Normal  Strong        Yes
8      Sunny        Cool   Normal    Weak        Yes
9       Rain        Mild   Normal    Weak        Yes
10     Sunny        Mild   Normal  Strong        Yes
12  Overcast         Hot   Normal    Weak        Yes
values =  ['No' 'Yes']
counts =  [1 6]
probabilities =  [0.14285714 0.85714286]
0.7884504573082896
values =  ['No' 'Yes']
counts =  [5 9]
probabilities =  [0.35714286 0.64285714]
0.9402859586706311
values =  ['Strong' 'Weak']
counts =  [6 8]
     Outlook Temperature Humidity    Wind PlayTennis
1      Sunny         Hot     High  Strong         No
5       Rain        Cool   Normal  Strong         No
6   Overcast        Cool   Normal  Strong        Yes
10     Sunny        Mild   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
13      Rain        Mild     High  Strong         No
values =  ['No' 'Yes']
counts =  [3 3]
probabilities =  [0.5 0.5]
0.42857142857142855
     Outlook Temperature Humidity  Wind PlayTennis
0      Sunny         Hot     High  Weak         No
2   Overcast         Hot     High  Weak        Yes
3       Rain        Mild     High  Weak        Yes
4       Rain        Cool   Normal  Weak        Yes
7      Sunny        Mild     High  Weak         No
```

```
8      Sunny        Cool   Normal  Weak        Yes
9       Rain        Mild   Normal  Weak        Yes
12  Overcast         Hot   Normal  Weak        Yes
values =  ['No' 'Yes']
counts =  [2 6]
probabilities =  [0.25 0.75]
0.8921589282623617
values =  ['No' 'Yes']
counts =  [2 3]
probabilities =  [0.4 0.6]
0.9709505944546686
values =  ['Cool' 'Mild']
counts =  [2 3]
  Temperature Humidity    Wind PlayTennis
4        Cool   Normal    Weak        Yes
5        Cool   Normal  Strong         No
values =  ['No' 'Yes']
counts =  [1 1]
probabilities =  [0.5 0.5]
0.4
   Temperature Humidity    Wind PlayTennis
3         Mild     High    Weak        Yes
9         Mild   Normal    Weak        Yes
13        Mild     High  Strong         No
values =  ['No' 'Yes']
counts =  [1 2]
probabilities =  [0.33333333 0.66666667]
0.9509775004326937
values =  ['No' 'Yes']
counts =  [2 3]
probabilities =  [0.4 0.6]
0.9709505944546686
values =  ['High' 'Normal']
counts =  [2 3]
   Temperature Humidity    Wind PlayTennis
3         Mild     High    Weak        Yes
13        Mild     High  Strong         No
values =  ['No' 'Yes']
counts =  [1 1]
probabilities =  [0.5 0.5]
0.4
  Temperature Humidity    Wind PlayTennis
4        Cool   Normal    Weak        Yes
5        Cool   Normal  Strong         No
9        Mild   Normal    Weak        Yes
values =  ['No' 'Yes']
counts =  [1 2]
probabilities =  [0.33333333 0.66666667]
0.9509775004326937
```

```
values =  ['No' 'Yes']
counts =  [2 3]
probabilities =  [0.4 0.6]
0.9709505944546686
values =  ['Strong' 'Weak']
counts =  [2 3]
   Temperature Humidity    Wind PlayTennis
5         Cool   Normal  Strong         No
13        Mild     High  Strong         No
values =  ['No']
counts =  [2]
probabilities =  [1.]
0.0
  Temperature Humidity  Wind PlayTennis
3        Mild     High  Weak        Yes
4        Cool   Normal  Weak        Yes
9        Mild   Normal  Weak        Yes
values =  ['Yes']
counts =  [3]
probabilities =  [1.]
0.0
values =  ['No' 'Yes']
counts =  [3 2]
probabilities =  [0.6 0.4]
0.9709505944546686
values =  ['Cool' 'Hot' 'Mild']
counts =  [1 2 2]
  Temperature Humidity  Wind PlayTennis
8        Cool   Normal  Weak        Yes
values =  ['Yes']
counts =  [1]
probabilities =  [1.]
0.0
  Temperature Humidity    Wind PlayTennis
0         Hot     High    Weak         No
1         Hot     High  Strong         No
values =  ['No']
counts =  [2]
probabilities =  [1.]
0.0
   Temperature Humidity    Wind PlayTennis
7         Mild     High    Weak         No
10        Mild   Normal  Strong        Yes
values =  ['No' 'Yes']
counts =  [1 1]
probabilities =  [0.5 0.5]
0.4
values =  ['No' 'Yes']
counts =  [3 2]
```

```
probabilities =  [0.6 0.4]
0.9709505944546686
values =  ['High' 'Normal']
counts =  [3 2]
  Temperature Humidity    Wind PlayTennis
0         Hot     High    Weak         No
1         Hot     High  Strong         No
7        Mild     High    Weak         No
values =  ['No']
counts =  [3]
probabilities =  [1.]
0.0
   Temperature Humidity    Wind PlayTennis
8         Cool   Normal    Weak        Yes
10        Mild   Normal  Strong        Yes
values =  ['Yes']
counts =  [2]
probabilities =  [1.]
0.0
values =  ['No' 'Yes']
counts =  [3 2]
probabilities =  [0.6 0.4]
0.9709505944546686
values =  ['Strong' 'Weak']
counts =  [2 3]
   Temperature Humidity    Wind PlayTennis
1          Hot     High  Strong         No
10        Mild   Normal  Strong        Yes
values =  ['No' 'Yes']
counts =  [1 1]
probabilities =  [0.5 0.5]
0.4
  Temperature Humidity  Wind PlayTennis
0         Hot     High  Weak         No
7        Mild     High  Weak         No
8        Cool   Normal  Weak        Yes
values =  ['No' 'Yes']
counts =  [2 1]
probabilities =  [0.66666667 0.33333333]
0.9509775004326937
```

Print Tree

```
tree
```

```
{'Outlook': {'Overcast': 'Yes',
  'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
  'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Implement K- Means without Library

Sample data points

data = [[1, 2], [2, 3], [3, 4], [10, 11], [11, 12], [12, 13], [50, 51], [51, 52], [52, 53]]

```
import math

data = [
    [1, 2], [2, 3], [3, 4],
    [10, 11], [11, 12], [12, 13],
    [50, 51], [51, 52], [52, 53]
]

def distance(x1,x2):
    return math.sqrt(((x1[0] - x2[0])**2) + ((x1[1] - x2[1])**2))

distance([1,1],[1,1])
```

```
0.0
```

```
def update_cluster_center(cluster_data):
    sum = [0,0]
    for i in cluster_data:
        sum[0] =  sum[0] + i[0]
        sum[1] =  sum[1] + i[1]
    return [sum[0]/len(cluster_data),sum[1]/len(cluster_data)]

update_cluster_center([[1,1],[2,2],[1,1]])
```

```
[1.3333333333333333, 1.3333333333333333]
```

Now Implement code

```
import numpy as np

def kmeans_du(k,data):
    # select random center
    center_data = [data[np.random.randint(0,len(data))] for i in
range(0,k)]
    print(center_data)
```

```
    #cluster data
    cluster_data = [[] for i in range(0,k)]
    for i in range(0,k):
        cluster_data[i].append(center_data[i])
    print(cluster_data)

    for j in range(0,5):
        cluster_data = [[] for i in range(0,k)]
        for d in data:
            mindistance = []
            for i in range(0,k):
                mindistance.append(distance(center_data[i],d))
            print(d ,"-->",mindistance)

cluster_data[mindistance.index(min(mindistance))].append(d)

        # print Cluster data

        for i in range(0,k):
            print(i,"-->",cluster_data[i])

        # update Cluster center
        for i in range(0,k):
            center_data[i] = update_cluster_center(cluster_data[i])
        print("NEW Cluster Center",center_data)

kmeans_du(3,data)
```

```
[[1, 2], [11, 12], [10, 11]]
[[[1, 2]], [[11, 12]], [[10, 11]]]
[1, 2] --> [0.0, 14.142135623730951, 12.727922061357855]
[2, 3] --> [1.4142135623730951, 12.727922061357855,
11.313708498984761]
[3, 4] --> [2.8284271247461903, 11.313708498984761, 9.899494936611665]
[10, 11] --> [12.727922061357855, 1.4142135623730951, 0.0]
[11, 12] --> [14.142135623730951, 0.0, 1.4142135623730951]
[12, 13] --> [15.556349186104045, 1.4142135623730951,
2.8284271247461903]
[50, 51] --> [69.29646455628166, 55.154328932550705,
56.568542494923804]
[51, 52] --> [70.71067811865476, 56.568542494923804,
57.982756057296896]
[52, 53] --> [72.12489168102785, 57.982756057296896,
59.39696961966999]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[11, 12], [12, 13], [50, 51], [51, 52], [52, 53]]
2 --> [[10, 11]]
NEW Cluster Center [[2.0, 3.0], [35.2, 36.2], [10.0, 11.0]]
[1, 2] --> [1.4142135623730951, 48.366103833159855,
12.727922061357855]
```

```
[2, 3] --> [0.0, 46.95189027078676, 11.313708498984761]
[3, 4] --> [1.4142135623730951, 45.53767670841366, 9.899494936611665]
[10, 11] --> [11.313708498984761, 35.638181771802, 0.0]
[11, 12] --> [12.727922061357855, 34.223968209428904,
1.4142135623730951]
[12, 13] --> [14.142135623730951, 32.80975464705581,
2.8284271247461903]
[50, 51] --> [67.88225099390856, 20.9303607231218, 56.568542494923804]
[51, 52] --> [69.29646455628166, 22.344574285494897,
57.982756057296896]
[52, 53] --> [70.71067811865476, 23.758787847867993,
59.39696961966999]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[50, 51], [51, 52], [52, 53]]
2 --> [[10, 11], [11, 12], [12, 13]]
NEW Cluster Center [[2.0, 3.0], [51.0, 52.0], [11.0, 12.0]]
[1, 2] --> [1.4142135623730951, 70.71067811865476, 14.142135623730951]
[2, 3] --> [0.0, 69.29646455628166, 12.727922061357855]
[3, 4] --> [1.4142135623730951, 67.88225099390856, 11.313708498984761]
[10, 11] --> [11.313708498984761, 57.982756057296896,
1.4142135623730951]
[11, 12] --> [12.727922061357855, 56.568542494923804, 0.0]
[12, 13] --> [14.142135623730951, 55.154328932550705,
1.4142135623730951]
[50, 51] --> [67.88225099390856, 1.4142135623730951,
55.154328932550705]
[51, 52] --> [69.29646455628166, 0.0, 56.568542494923804]
[52, 53] --> [70.71067811865476, 1.4142135623730951,
57.982756057296896]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[50, 51], [51, 52], [52, 53]]
2 --> [[10, 11], [11, 12], [12, 13]]
NEW Cluster Center [[2.0, 3.0], [51.0, 52.0], [11.0, 12.0]]
[1, 2] --> [1.4142135623730951, 70.71067811865476, 14.142135623730951]
[2, 3] --> [0.0, 69.29646455628166, 12.727922061357855]
[3, 4] --> [1.4142135623730951, 67.88225099390856, 11.313708498984761]
[10, 11] --> [11.313708498984761, 57.982756057296896,
1.4142135623730951]
[11, 12] --> [12.727922061357855, 56.568542494923804, 0.0]
[12, 13] --> [14.142135623730951, 55.154328932550705,
1.4142135623730951]
[50, 51] --> [67.88225099390856, 1.4142135623730951,
55.154328932550705]
[51, 52] --> [69.29646455628166, 0.0, 56.568542494923804]
[52, 53] --> [70.71067811865476, 1.4142135623730951,
57.982756057296896]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[50, 51], [51, 52], [52, 53]]
2 --> [[10, 11], [11, 12], [12, 13]]
```

```
NEW Cluster Center [[2.0, 3.0], [51.0, 52.0], [11.0, 12.0]]
[1, 2] --> [1.4142135623730951, 70.71067811865476, 14.142135623730951]
[2, 3] --> [0.0, 69.29646455628166, 12.727922061357855]
[3, 4] --> [1.4142135623730951, 67.88225099390856, 11.313708498984761]
[10, 11] --> [11.313708498984761, 57.982756057296896,
1.4142135623730951]
[11, 12] --> [12.727922061357855, 56.568542494923804, 0.0]
[12, 13] --> [14.142135623730951, 55.154328932550705,
1.4142135623730951]
[50, 51] --> [67.88225099390856, 1.4142135623730951,
55.154328932550705]
[51, 52] --> [69.29646455628166, 0.0, 56.568542494923804]
[52, 53] --> [70.71067811865476, 1.4142135623730951,
57.982756057296896]
0 --> [[1, 2], [2, 3], [3, 4]]
1 --> [[50, 51], [51, 52], [52, 53]]
2 --> [[10, 11], [11, 12], [12, 13]]
NEW Cluster Center [[2.0, 3.0], [51.0, 52.0], [11.0, 12.0]]
```

Implement K-Medoids without Library

Sample data points

data = [[1, 2], [2, 3], [3, 4], [10, 11], [11, 12], [12, 13], [50, 51], [51, 52], [52, 53]]

```
import random
import math

def euclidean_distance(p1, p2):
    return math.sqrt(sum((x - y) ** 2 for x, y in zip(p1, p2)))

def assign_points(data, medoids):
    clusters = {i: [] for i in range(len(medoids))}
    for point in data:
        distances = [euclidean_distance(point, medoid) for medoid in
medoids]
        nearest = distances.index(min(distances))
        clusters[nearest].append(point)
    return clusters

def calculate_cost(clusters, medoids):
    cost = 0
    for i, points in clusters.items():
        for p in points:
            cost += euclidean_distance(p, medoids[i])
    return cost
```

```python
def k_medoids(data, k, max_iter=100):
    # Step 1: Randomly select initial medoids
    medoids = random.sample(data, k)

    for _ in range(max_iter):
        clusters = assign_points(data, medoids)
        current_cost = calculate_cost(clusters, medoids)

        best_medoids = medoids[:]
        improved = False

        # Step 2: Try swapping medoids with non-medoids
        for i in range(len(medoids)):
            for candidate in data:
                if candidate not in medoids:
                    new_medoids = medoids[:]
                    new_medoids[i] = candidate
                    new_clusters = assign_points(data, new_medoids)
                    new_cost = calculate_cost(new_clusters,
new_medoids)

                    if new_cost < current_cost:
                        best_medoids = new_medoids
                        current_cost = new_cost
                        improved = True

        medoids = best_medoids
        if not improved:
            break  # convergence

    final_clusters = assign_points(data, medoids)
    return medoids, final_clusters

k = 3
medoids, clusters = k_medoids(data, k)

print("Final Medoids:", medoids)
print("Clusters:")
for i, points in clusters.items():
    print(f"Cluster {i+1}: {points}")
```

```
Final Medoids: [[51, 52], [11, 12], [2, 3]]
Clusters:
Cluster 1: [[50, 51], [51, 52], [52, 53]]
Cluster 2: [[10, 11], [11, 12], [12, 13]]
Cluster 3: [[1, 2], [2, 3], [3, 4]]
```