Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Data Mining

Lab - 1

137 | Vishal Baraiya | 23010101014

Introduction to Pandas Library Function:

Step-1 Import the pandas Libraries

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

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

In [7]:
```
```

Step-3 Read csv or excel File

In [3]:
```
df= pd.read_csv("titanic.csv")
df
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7

891 rows × 12 columns

Step-4 Print Data from csv or excel File

In [5]:
```
df.shape
```

Out[5]: (891, 12)

Step-5 See the First 10 Rows

In [6]:
```
df.head(10)
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.458
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.862
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.075
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.133
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.070

Step-6 See the Last 10 Rows

In [7]:
```
df.tail(10)
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.

Step-7 Data type of each columns

In [9]:
```
df.dtypes
```

Out[9]:
```
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 [12]:
```
df.describe()
```

Out[12]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Step-9 Access a specific column

In [21]:
```
# specifically colmun access karava mate
df["Name"]
```

Out[21]:
```
0                                Braund, Mr. Owen Harris
1      Cumings, Mrs. John Bradley (Florence Briggs Th...
2                                 Heikkinen, Miss. Laina
3           Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                               Allen, Mr. William Henry
                              ...
886                               Montvila, Rev. Juozas
887                        Graham, Miss. Margaret Edith
888            Johnston, Miss. Catherine Helen "Carrie"
889                               Behr, Mr. Karl Howell
890                                 Dooley, Mr. Patrick
Name: Name, Length: 891, dtype: object
```

In [8]:
```
df[["Name","Age","Parch","Fare"]]
```

Out[8]:

	Name	Age	Parch	Fare
0	Braund, Mr. Owen Harris	22.0	0	7.2500
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	38.0	0	71.2833
2	Heikkinen, Miss. Laina	26.0	0	7.9250
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	35.0	0	53.1000
4	Allen, Mr. William Henry	35.0	0	8.0500
...	...	...	...	...
886	Montvila, Rev. Juozas	27.0	0	13.0000
887	Graham, Miss. Margaret Edith	19.0	0	30.0000
888	Johnston, Miss. Catherine Helen "Carrie"	NaN	2	23.4500
889	Behr, Mr. Karl Howell	26.0	0	30.0000
890	Dooley, Mr. Patrick	32.0	0	7.7500

891 rows × 4 columns

Step-10 Access rows by their integer location

In [20]:
```
# specifically raw access karava mate
df.iloc[0]
```

Out[20]:
```
PassengerId                     1
Survived                        0
Pclass                          3
Name           Braund, Mr. Owen Harris
Sex                          male
Age                          22.0
SibSp                           1
Parch                           0
Ticket                      A/5 21171
Fare                         7.25
Cabin                         NaN
Embarked                        S
Name: 0, dtype: object
```

In [9]:
```
df.iloc[[0, 2]]
```

Out[9]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925

In [10]:
```
df.iloc[0:3]  # Rows 0 to 2 (not including 3)
```

Out[10]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250

In [11]:
```
df[df["Age"] > 25]
```

Out[11]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.
...	...	...	...	...	...	...	...	...	...	
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.

413 rows × 12 columns

In [18]:
```
df.loc[883]
```

Out[18]:
```
PassengerId                              884
Survived                                   0
Pclass                                     2
Name           Banfield, Mr. Frederick James
Sex                                     male
Age                                     28.0
SibSp                                      0
Parch                                      0
Ticket                       C.A./SOTON 34068
Fare                                    10.5
Cabin                                    NaN
Embarked                                   S
Name: 883, dtype: object
```

Step-11 Delete a specific Column

In [26]:
```
# specific colmun delete karava mate
df.drop("Embarked",axis="columns",inplace=True)
```

Step-12 Create a new Column

In [30]:
```
df
```

Out[30]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7

891 rows × 12 columns

In [31]:
```
df["isCabin"] = ~ df["Cabin"].isnull()
```

In [32]:
```
df
```

Out[32]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7

891 rows × 12 columns

Step-13 Perform Condition Selection on DataFrame

In [33]:
```
df[df["Age"] > 30]
```

Out[33]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500
...	...	...	...	...	...	...	...	...	...	...
873	874	0	3	Vander Cruyssen, Mr. Victor	male	47.0	0	0	345765	9.0000
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500

305 rows × 12 columns

In [34]:
```
df[df["Pclass"] == 3]
```

Out[34]:

	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	7.
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.
...	...	...	...	...	...	...	...	...	...	
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.

491 rows × 12 columns

Step-14 Compute the sum of value

In [35]:
```
df["Fare"].sum()
```

Out[35]:
```
28693.9493
```

Step-15 Compute the mean of value

In [36]:
```
df["Fare"].mean()
```

Out[36]:
```
32.204207968574636
```

Step-16 Count non-null value (column)

In [37]:
```
(~df.isnull()).sum()
```

Out[37]:
```
PassengerId    891
Survived       891
Pclass         891
Name           891
Sex            891
Age            714
SibSp          891
Parch          891
Ticket         891
Fare           891
Cabin          204
isCabin        891
dtype: int64
```

Step-17 Find Minimun or Maximum values

In [38]:
```
df["Fare"].min()
```

Out[38]:
```
0.0
```

In [39]:
```
df["Fare"].max()
```

Out[39]:
```
512.3292
```

In [40]:
```
df["Fare"].median()
```

Out[40]:
```
14.4542
```

In [41]:
```
df["Fare"].mode()
```

Out[41]:
```
0    8.05
Name: Fare, dtype: float64
```

In [42]:
```
df["Fare"].std()
```

Out[42]: 49.693428597180905

In [ ]:
```
# genrate a excel file
df.to_excel("output.xlsx")
```

Data Mining - Lab - 2

Numpy & Perform Data Exploration with Pandas

137 | Vishal Baraiya | 23010101014

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 [1]:
```
import numpy as np
```

Step 2. Create a 1D array of numbers

In [4]:
```
a = np.arange(11)
print(a)
print(type(a))
```

```
[ 0  1  2  3  4  5  6  7  8  9 10]
<class 'numpy.ndarray'>
```

In [6]:
```
a = np.arange(2,9)
print(a)
```

```
[2 3 4 5 6 7 8]
```

Step 3. Reshape 1D to 2D Array

In [15]:
```
# error will occurs if it can not fit in 2-D array dimension
a = np.arange(12).reshape(3,4)
a
```

Out[15]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Step 4. Create a Linspace array

In [18]:
```
# ( 0 + 5 ) / 19 = 0.2631578947368421
np.linspace(0,5,20)
```

Out[18]:
```
array([0.        , 0.26315789, 0.52631579, 0.78947368, 1.05263158,
       1.31578947, 1.57894737, 1.84210526, 2.10526316, 2.36842105,
       2.63157895, 2.89473684, 3.15789474, 3.42105263, 3.68421053,
       3.94736842, 4.21052632, 4.47368421, 4.73684211, 5.        ])
```

Step 5. Create a Random Numbered Array

In [19]:
```
np.random.rand(2)
```

Out[19]:
```
array([0.46224826, 0.3958225 ])
```

In [20]:
```
np.random.rand(2,4)
```

Out[20]:
```
array([[0.9651745 , 0.57112977, 0.83058656, 0.21649437],
       [0.14483748, 0.25548204, 0.43413858, 0.04114541]])
```

Step 6. Create a Random Integer Array

In [21]:
```
np.random.randint(1,100,size=10)
```

Out[21]:
```
array([83, 99, 30, 66,  8, 41, 28,  6, 26, 60])
```

In [22]:
```
np.random.randint(1,100,size=(2,4))
```

Out[22]:
```
array([[67, 43, 75, 10],
       [22, 99, 54, 39]])
```

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

In [33]:
```
arr = np.random.randint(1,100,size=10)
arr
```

Out[33]:
```
array([12, 21, 30, 39, 14, 42, 53, 13, 36, 20])
```

In [34]:
```
arr.max()
```

Out[34]:
```
53
```

In [35]:
```
arr.min()
```

Out[35]:
```
12
```

In [36]:
```
arr.argmax()
```

Out[36]:
```
6
```

In [37]:
```
arr.argmin()
```

Out[37]:
```
0
```

In [38]:
```
arr.mean()
```

Out[38]:
```
28.0
```

Step 8. Indexing in 1D Array

In [40]:
```
arr[8]
```

Out[40]:
```
36
```

In [39]:
```
arr[1:5]
```

Out[39]:
```
array([21, 30, 39, 14])
```

Step 9. Indexing in 2D Array

In [46]:
```
arr2d = np.array([
    [101,102,103,104,105],
    [201,202,203,204,205],
    [301,302,303,304,305],
    [401,402,403,404,405]])
arr2d
```

Out[46]:
```
array([[101, 102, 103, 104, 105],
       [201, 202, 203, 204, 205],
       [301, 302, 303, 304, 305],
       [401, 402, 403, 404, 405]])
```

In [48]:
```
arr2d
```

Out[48]:
```
array([[101, 102, 103, 104, 105],
       [201, 202, 203, 204, 205],
       [301, 302, 303, 304, 305],
       [401, 402, 403, 404, 405]])
```

In [49]:
```
arr2d[1::2]
```

Out[49]:
```
array([[201, 202, 203, 204, 205],
       [401, 402, 403, 404, 405]])
```

In [50]:
```
arr2d[::2,::2]
```

Out[50]:
```
array([[101, 103, 105],
       [301, 303, 305]])
```

In [51]:
```
arr2d[1:3:,1:4:]
```

Out[51]:
```
array([[202, 203, 204],
       [302, 303, 304]])
```

Step 10. Conditional Selection

In [57]:
```
arr[arr>20]
```

Out[57]:
```
array([21, 30, 39, 42, 53, 36])
```

In [59]:
```
arr2d[arr2d>120]
```

Out[59]:
```
array([201, 202, 203, 204, 205, 301, 302, 303, 304, 305, 401, 402, 403,
       404, 405])
```

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

Pandas

Step 1. Import the necessary libraries

In [60]:
```
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 [62]:
```
df= pd.read_csv("https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.u
df
```

Out[62]:

	user_id	age	gender	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213
...	...	...	...	...	...
938	939	26	F	student	33319
939	940	32	M	administrator	02215
940	941	20	M	student	97229
941	942	48	F	librarian	78209
942	943	22	M	student	77841

943 rows × 5 columns

Step 4. See the first 25 entries

In [63]:
```
df.head(25)
```

Out[63]:

	user_id	age	gender	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213
5	6	42	M	executive	98101
6	7	57	M	administrator	91344
7	8	36	M	administrator	05201
8	9	29	M	student	01002
9	10	53	M	lawyer	90703
10	11	39	F	other	30329
11	12	28	F	other	06405
12	13	47	M	educator	29206
13	14	45	M	scientist	55106
14	15	49	F	educator	97301
15	16	21	M	entertainment	10309
16	17	30	M	programmer	06355
17	18	35	F	other	37212
18	19	40	M	librarian	02138
19	20	42	F	homemaker	95660
20	21	26	M	writer	30068
21	22	25	M	writer	40206
22	23	30	F	artist	48197
23	24	21	F	artist	94533
24	25	39	M	engineer	55107

Step 5. See the last 10 entries

In [64]:
```
df.tail(10)
```

Out[64]:

	user_id	age	gender	occupation	zip_code
933	934	61	M	engineer	22902
934	935	42	M	doctor	66221
935	936	24	M	other	32789
936	937	48	M	educator	98072
937	938	38	F	technician	55038
938	939	26	F	student	33319
939	940	32	M	administrator	02215
940	941	20	M	student	97229
941	942	48	F	librarian	78209
942	943	22	M	student	77841

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

In [66]:
```
df.shape[0]
```

Out[66]:
```
943
```

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

In [67]:
```
df.shape[1]
```

Out[67]:
```
5
```

Step 8. Print the name of all the columns.

In [71]:
```
df.columns
```

Out[71]:
```
Index(['user_id', 'age', 'gender', 'occupation', 'zip_code'], dtype='object')
```

Step 9. How is the dataset indexed?

In [77]:
```
df.index
```

Out[77]:
```
RangeIndex(start=0, stop=943, step=1)
```

In [ ]:
```

```

Out[ ]:
```
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 [74]:
```
df.dtypes
```

Out[74]:
```
user_id        int64
age            int64
gender        object
occupation    object
zip_code      object
dtype: object
```

Step 11. Print only the occupation column

In [78]:
```
df["occupation"]
```

Out[78]:
```
0         technician
1              other
2             writer
3         technician
4              other
           ...
938          student
939    administrator
940          student
941        librarian
942          student
Name: occupation, Length: 943, dtype: object
```

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

In [86]:
```
df.occupation.nunique()
```

Out[86]:
```
21
```

Step 13. What is the most frequent occupation?

In [87]:
```
df.occupation.value_counts().head(1)
```

Out[87]:
```
occupation
student    196
Name: count, dtype: int64
```

Step 14. Summarize the DataFrame.

In [94]:
```
df.describe()
```

Out[94]:

	user_id	age
count	943.000000	943.000000
mean	472.000000	34.051962
std	272.364951	12.192740
min	1.000000	7.000000
25%	236.500000	25.000000
50%	472.000000	31.000000
75%	707.500000	43.000000
max	943.000000	73.000000

Step 15. Summarize all the columns

In [91]:
```
df.describe(include="all")
```

Out[91]:

	user_id	age	gender	occupation	zip_code
count	943.000000	943.000000	943	943	943
unique	NaN	NaN	2	21	795
top	NaN	NaN	M	student	55414
freq	NaN	NaN	670	196	9
mean	472.000000	34.051962	NaN	NaN	NaN
std	272.364951	12.192740	NaN	NaN	NaN
min	1.000000	7.000000	NaN	NaN	NaN
25%	236.500000	25.000000	NaN	NaN	NaN
50%	472.000000	31.000000	NaN	NaN	NaN
75%	707.500000	43.000000	NaN	NaN	NaN
max	943.000000	73.000000	NaN	NaN	NaN

Step 16. Summarize only the occupation column

In [93]:
```
df.occupation.describe()
```

Out[93]:
```
count         943
unique         21
top       student
freq          196
Name: occupation, dtype: object
```

Step 17. What is the mean age of users?

In [89]:
```
df["age"].mean()
```

Out[89]:
```
34.05196182396607
```

Step 18. What is the age with least occurrence?

In [92]:
```
df.age.value_counts().tail()
```

Out[92]:
```
age
7     1
66    1
11    1
10    1
73    1
Name: count, dtype: int64
```

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

In [ ]:
```

```

Darshan UNIVERSITY

योग: कर्मसु कौशलम्

Data Mining

Lab - 3

137 | Vishal Baraiya | 23010101014

https://tinyurl.com/dm12025

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

In [4]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

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

In [3]:
```
df.head(5)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

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

In [4]:
```
nominal  = ["Name","Sex","Cabin","Embarked","Ticket"]
ordinal = ["Pclass"]
binary = ["Sex","Survived"]
numaric = ["Age","Fare","SibSp","Parch"]

print("Nominal : ",nominal)
print("Ordinal : ",ordinal)
print("Binary : ",binary)
print("Numaric : ",numaric)
```

```
Nominal :  ['Name', 'Sex', 'Cabin', 'Embarked', 'Ticket']
Ordinal :  ['Pclass']
Binary :  ['Sex', 'Survived']
Numaric :  ['Age', 'Fare', 'SibSp', 'Parch']
```

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

In [6]:
```
print("Survived values (Asymmetric binary):")
print(df['Survived'].value_counts())
print("-"*50)
```

```
print("Gender Count (Symmetric binary):")
print(df['Sex'].value_counts())
```

```
Survived values (Asymmetric binary):
Survived
0    549
1    342
Name: count, dtype: int64
--------------------------------------------------
Gender Count (Symmetric binary):
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 [10]:
```
quantitative = ['PassengerId','Survived','Pclass','Age','SibSp','Parch','Fare']

for col in quantitative:
    print("::::::",col,"::::::")
    print("Mean : ",df[col].mean())
    print("Standard Devition : ",df[col].std())
    print("Minimum : ",df[col].min())
    print("Maximum : ",df[col].max())
    print("Mode : ",df[col].mode()[0])
    print("Range : ",df[col].max()-df[col].min())
    print("-"*50)
```

```
:::::: PassengerId ::::::
Mean :  446.0
Standard Devition :  257.3538420152301
Minimum :  1
Maximum :  891
Mode :  1
Range :  890
--------------------------------------------------
:::::: Survived ::::::
Mean :  0.3838383838383838
Standard Devition :  0.4865924542648585
Minimum :  0
Maximum :  1
Mode :  0
Range :  1
--------------------------------------------------
:::::: Pclass ::::::
Mean :  2.308641975308642
Standard Devition :  0.8360712409770513
Minimum :  1
Maximum :  3
Mode :  3
Range :  2
--------------------------------------------------
:::::: Age ::::::
Mean :  29.69911764705882
Standard Devition :  14.526497332334044
Minimum :  0.42
Maximum :  80.0
Mode :  24.0
Range :  79.58
--------------------------------------------------
:::::: SibSp ::::::
Mean :  0.5230078563411896
Standard Devition :  1.1027434322934275
Minimum :  0
Maximum :  8
Mode :  0
Range :  8
--------------------------------------------------
:::::: Parch ::::::
Mean :  0.38159371492704824
Standard Devition :  0.8060572211299559
Minimum :  0
Maximum :  6
Mode :  0
Range :  6
--------------------------------------------------
:::::: Fare ::::::
Mean :  32.204207968574636
Standard Devition :  49.693428597180905
Minimum :  0.0
Maximum :  512.3292
Mode :  8.05
Range :  512.3292
--------------------------------------------------
```

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

In [11]:
```
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 [22]:
```
print("\nSummary for Numeric Attributes : ")
df.describe()
```

```
Summary for Numeric Attributes :
```

Out[22]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [23]:
```
print("\nSummary for All attributes : ")
df.describe(include='all')
```

```
Summary for All attributes :
```

Out[23]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	F
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.00
unique	NaN	NaN	NaN	891	2	NaN	NaN	
top	NaN	NaN	NaN	Braund, Mr. Owen Harris	male	NaN	NaN	
freq	NaN	NaN	NaN	1	577	NaN	NaN	
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.38
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.80
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.00
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.00
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.00
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.00
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.00

In [24]:
```
print("\nSummary for Categorical Attributes : ")
df.describe(include="object")
#df.describe(include=["object"]) # Also valid
```

```
Summary for Categorical Attributes :
```

Out[24]:

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Braund, Mr. Owen Harris	male	347082	B96 B98	S
freq	1	577	7	4	644

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

In [26]:
```
print("Covariance Matrix : ")
df.cov(numeric_only=True)
```

```
Covariance Matrix :
```

Out[26]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	
PassengerId	66231.000000	-0.626966	-7.561798	138.696504	-16.325843	-0.342697	161.8
Survived	-0.626966	0.236772	-0.137703	-0.551296	-0.018954	0.032017	6.2
Pclass	-7.561798	-0.137703	0.699015	-4.496004	0.076599	0.012429	-22.8
Age	138.696504	-0.551296	-4.496004	211.019125	-4.163334	-2.344191	73.8
SibSp	-16.325843	-0.018954	0.076599	-4.163334	1.216043	0.368739	8.7
Parch	-0.342697	0.032017	0.012429	-2.344191	0.368739	0.649728	8.6
Fare	161.883369	6.221787	-22.830196	73.849030	8.748734	8.661052	2469.4

In [27]:
```
print("Correlation Matrix : ")
df.corr(numeric_only=True)
```

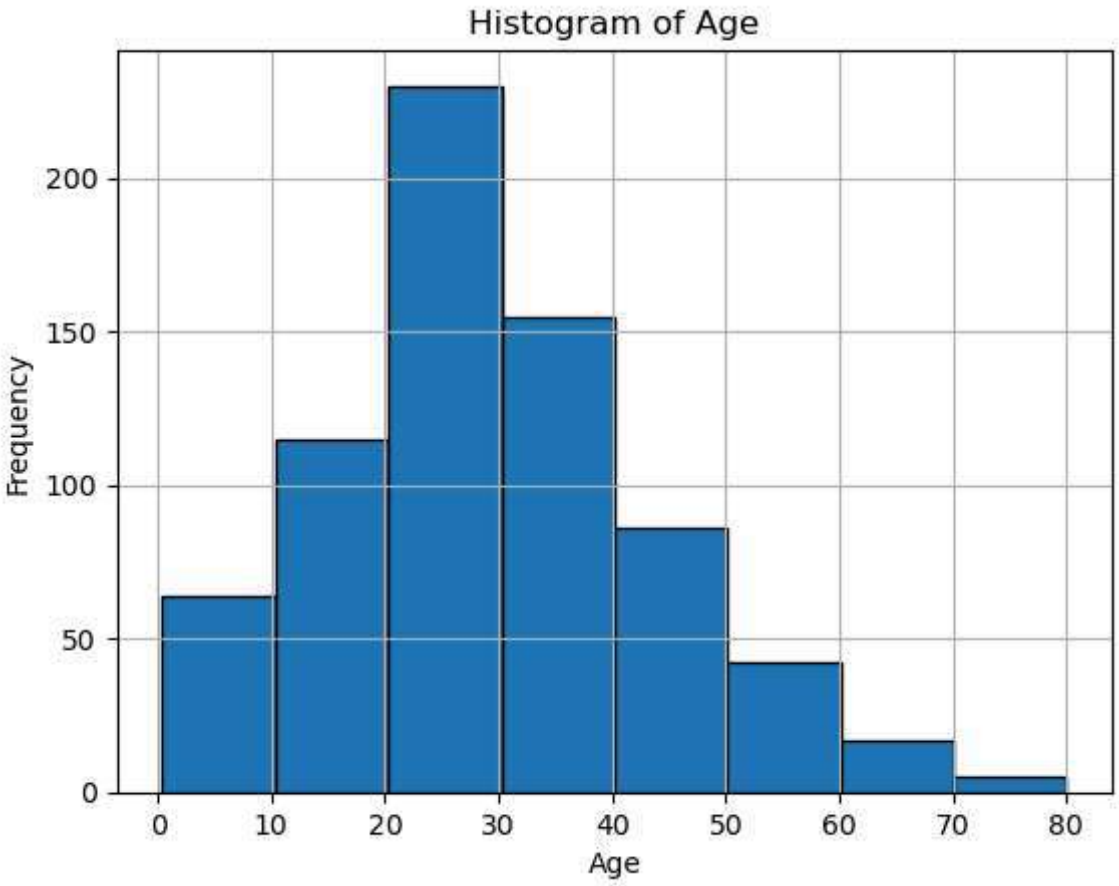
```
Correlation Matrix :
```

Out[27]:

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

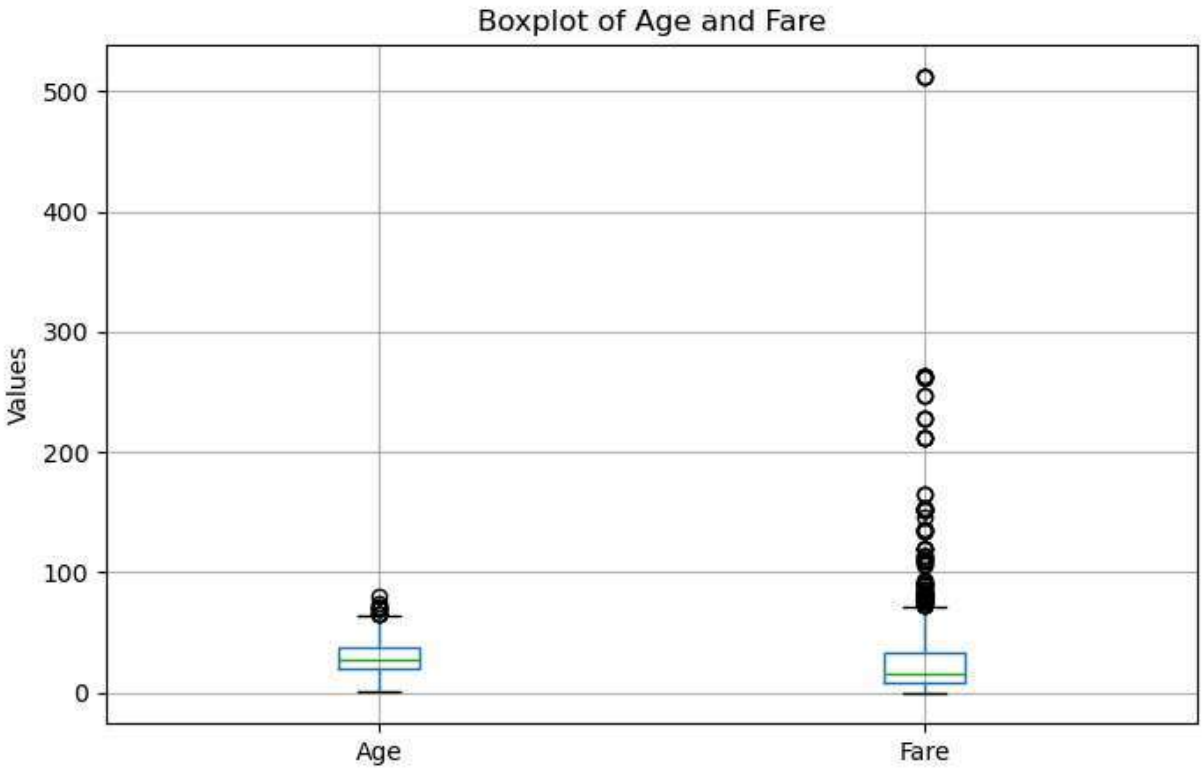
9) Display the histogram for Age attribute by discretizing it into 8 separate bins and counting the frequency for each bin.

In [6]:
```
df["Age"].dropna().hist(bins=8, edgecolor='black')
plt.title("Histogram of Age")
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.show()
```



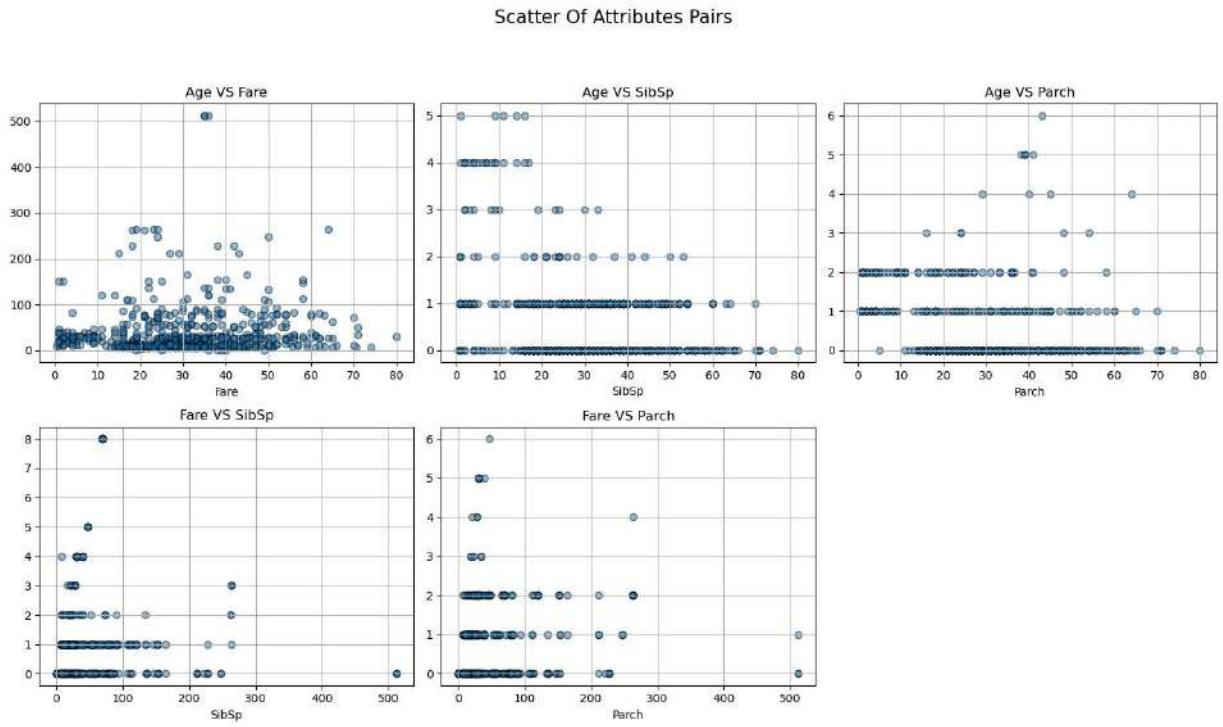
10) A boxplot can also be used to show the distribution of values for each attribute.

In [8]:
```
df_filtered = df[['Age', 'Fare']].dropna()

# Plot boxplots for Age and Fare
plt.figure(figsize=(8, 5))
df_filtered.boxplot(column=['Age', 'Fare'])
plt.title('Boxplot of Age and Fare')
plt.ylabel('Values')
plt.grid(True)
plt.show()
```



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

In [43]:
```
pairs = [
    ('Age','Fare'),
    ('Age','SibSp'),
    ('Age','Parch'),
    ('Fare','SibSp'),
    ('Fare','Parch')
]

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,edgecolors='k')
    plt.xlabel(x)
    plt.xlabel(y)
    plt.title(f"{x} VS {y}")
    plt.grid(True)

plt.tight_layout()
plt.suptitle("Scatter Of Attributes Pairs",fontsize=16, y=1.10)
plt.show()
```


Scatter Of Attributes Pairs
Age VS Fare
Age VS SibSp
Age VS Parch
Fare VS SibSp
Fare VS Parch
Fare
SibSp
Parch

In [ ]:

Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Data Mining

Lab - 4

137 | Vishal Baraiya | 23010101014

Step 1. Import the necessary libraries

In [25]:
```
import pandas as pd
import numpy as np
```

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/data
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 [9]:
```
# Solution 1
chipo.shape[0]
```

Out[9]: 4622

In [8]:
```
# 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 [10]:
```
chipo.shape[1]
```

Out[10]: 5

Step 7. Print the name of all the columns.

In [11]:
```
chipo.columns
```

Out[11]:
```
Index(['order_id', 'quantity', 'item_name', 'choice_description',
       'item_price'],
      dtype='object')
```

Step 8. How is the dataset indexed?

In [12]:
```
chipo.index
```

Out[12]:
```
RangeIndex(start=0, stop=4622, step=1)
```

Step 9. Number of Unique Items ?

In [17]:
```
chipo["item_name"].nunique()
```

Out[17]: 50

In [18]:
```
chipo["item_name"].unique()
```

Out[18]:
```
array(['Chips and Fresh Tomato Salsa', 'Izze', 'Nantucket Nectar',
       'Chips and Tomatillo-Green Chili Salsa', 'Chicken Bowl',
       'Side of Chips', 'Steak Burrito', 'Steak Soft Tacos',
       'Chips and Guacamole', 'Chicken Crispy Tacos',
       'Chicken Soft Tacos', 'Chicken Burrito', 'Canned Soda',
       'Barbacoa Burrito', 'Carnitas Burrito', 'Carnitas Bowl',
       'Bottled Water', 'Chips and Tomatillo Green Chili Salsa',
       'Barbacoa Bowl', 'Chips', 'Chicken Salad Bowl', 'Steak Bowl',
       'Barbacoa Soft Tacos', 'Veggie Burrito', 'Veggie Bowl',
       'Steak Crispy Tacos', 'Chips and Tomatillo Red Chili Salsa',
       'Barbacoa Crispy Tacos', 'Veggie Salad Bowl',
       'Chips and Roasted Chili-Corn Salsa',
       'Chips and Roasted Chili Corn Salsa', 'Carnitas Soft Tacos',
       'Chicken Salad', 'Canned Soft Drink', 'Steak Salad Bowl',
       '6 Pack Soft Drink', 'Chips and Tomatillo-Red Chili Salsa', 'Bowl',
       'Burrito', 'Crispy Tacos', 'Carnitas Crispy Tacos', 'Steak Salad',
       'Chips and Mild Fresh Tomato Salsa', 'Veggie Soft Tacos',
       'Carnitas Salad Bowl', 'Barbacoa Salad Bowl', 'Salad',
       'Veggie Crispy Tacos', 'Veggie Salad', 'Carnitas Salad'],
      dtype=object)
```

Step 10. Which was the most-ordered item?

In [19]:
```
c = chipo.groupby('item_name')
c = c.sum()
c = c.sort_values(['quantity'],ascending=False)
c.head(1)
```

Out[19]:

item_name	order_id	quantity	choice_description	item_price
Chicken Bowl	713926	761	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	16.9810.98 11.258.75 8.4911.25 $8.75 ...

In [20]:
```
chipo.groupby('item_name')['quantity'].sum().sort_values(ascending=False).head(1)
```

Out[20]:
```
item_name
Chicken Bowl    761
Name: quantity, dtype: int64
```

Step 11. How many items were orderd in total?

In [21]:
```
chipo.quantity.sum()
```

Out[21]: 4972

Step 12. Turn the item price into a float

Step 12.a. Check the item price type

In [22]:
```
chipo.item_price.dtype
```

Out[22]: dtype('O')

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

In [23]:
```
dollarizer = lambda x: float(x[1:-1])
chipo.item_price = chipo.item_price.apply(dollarizer)
```

Step 12.c. Check the item price type

In [24]:
```
chipo.item_price.dtype
```

Out[24]: dtype('float64')

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

In [26]:
```
revenue = (chipo['quantity']* chipo['item_price']).sum()
print(f"Revenue Was: ${str(np.round(revenue,2))}")
```

```
Revenue Was: $39237.02
```

Step 15. How many orders were made ?

In [29]:
```
orders = chipo.order_id.value_counts().count()
orders
```

Out[29]: 1834

Step 17. How many different choice descriptions are there?

In [30]:
```
chipo["choice_description"].nunique()
```

Out[30]: 1043

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

In [32]:
```
items = chipo.groupby('item_name')['quantity'].sum()
items[items > 100]
```

Out[32]:
```
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 [33]:
```
# Solution 1
chipo['revenue'] = chipo['quantity'] * chipo['item_price']
order_grouped = chipo.groupby(by=['order_id']).sum()
order_grouped['revenue'].mean()
```

Out[33]: 21.39423118865867

In [34]:
```
# Solution 2
chipo.groupby(by=['order_id']).sum()['revenue'].mean()
```

Out[34]: 21.39423118865867

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Data Mining

Lab-5 - Data Preprocessing

137 | Vishal Baraiya | 23010101014

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

In [3]:
```
import pandas as pd
import numpy as np
```

In [4]:
```
# Try reading the CSV with ISO-8859-1 encodingo
df = pd.read_csv("titanic.csv")
df
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7

891 rows × 12 columns

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

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.45
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75

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

In [6]:
```
data_dropna = df.dropna()
data_dropna
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Far
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.862
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.700
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.550
...	...	...	...	...	...	...	...	...	...	
871	872	1	1	Beckwith, Mrs. Richard Leonard (Sallie Monypeny)	female	47.0	1	1	11751	52.554
872	873	0	1	Carlsson, Mr. Frans Olof	male	33.0	0	0	695	5.000
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.158
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.000
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.000

183 rows × 12 columns

In [7]:
```
# delete row in which every column value is null
# data_dropna = df.dropna(how='all')

# delete row in which any
# data_dropna = df.dropna(how='any',axis = 1)
```

In [11]:
```
# using fillna
data_fillna = df.fillna(30)
data_fillna
```

Out[11]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.28
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.92
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.10
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	30.0	1	2	W./C. 6607	23.45
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75

891 rows × 12 columns

In [12]:
```
data_fillna = df.fillna({'Age' : 35, 'Cabin' : 'Not Available'})
data_fillna
```

Out[12]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.28
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.92
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.10
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	35.0	1	2	W./C. 6607	23.45
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75

891 rows × 12 columns

In [14]:
```
age_mean = df.Age.mean()
data_fillna = df.fillna({'Age':age_mean})
data_fillna
```

Out[14]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450
...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.000000	0	0	211536
887	888	1	1	Graham, Miss. Margaret Edith	female	19.000000	0	0	112053
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.699118	1	2	W./C. 6607
889	890	1	1	Behr, Mr. Karl Howell	male	26.000000	0	0	111369
890	891	0	3	Dooley, Mr. Patrick	male	32.000000	0	0	370376

891 rows × 12 columns

In [16]:
```
data_interpolate = df.interpolate()
data_interpolate
```

```
C:\Users\ASUS\AppData\Local\Temp\ipykernel_1768\2280711911.py:1: FutureWarning: Data
Frame.interpolate with object dtype is deprecated and will raise in a future versio
n. Call obj.infer_objects(copy=False) before interpolating instead.
  data_interpolate = df.interpolate()
```

Out[16]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.25
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.28
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.92
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.10
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.05
...	...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.00
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.00
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	22.5	1	2	W./C. 6607	23.45
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.00
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.75

891 rows × 12 columns

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

In [17]:
```
df.fillna(df.Age.mean(), inplace=True)

data2 = df.copy()
minAge = df.Age.min()
maxAge = df.Age.max()
data2['MinMaxAge'] = (data2['Age']-minAge)/(maxAge-minAge)
data2
```

Out[17]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450
...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.000000	0	0	211536
887	888	1	1	Graham, Miss. Margaret Edith	female	19.000000	0	0	112053
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.699118	1	2	W./C. 6607
889	890	1	1	Behr, Mr. Karl Howell	male	26.000000	0	0	111369
890	891	0	3	Dooley, Mr. Patrick	male	32.000000	0	0	370376

891 rows × 13 columns

In [19]:
```
data3 = df.copy()
maxAge = df.Age.max()
noOfDigits = len(str(int(maxAge)))

data2['AgeDS'] = data2['Age'] / ( 10 ** noOfDigits )
data2
```

Out[19]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450
...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.000000	0	0	211536
887	888	1	1	Graham, Miss. Margaret Edith	female	19.000000	0	0	112053
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.699118	1	2	W./C. 6607
889	890	1	1	Behr, Mr. Karl Howell	male	26.000000	0	0	111369
890	891	0	3	Dooley, Mr. Patrick	male	32.000000	0	0	370376

891 rows × 14 columns

In [21]:
```
meanAge = df.Age.mean()
stdAge = df.Age.std()
data3['AgeZScore'] = (data3['Age']-minAge)/stdAge
data3
```

Out[21]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450
...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.000000	0	0	211536
887	888	1	1	Graham, Miss. Margaret Edith	female	19.000000	0	0	112053
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.699118	1	2	W./C. 6607
889	890	1	1	Behr, Mr. Karl Howell	male	26.000000	0	0	111369
890	891	0	3	Dooley, Mr. Patrick	male	32.000000	0	0	370376

891 rows × 13 columns

In [ ]:

Darshan UNIVERSITY

योगः कर्मसु कौशलम्

Data Mining

Lab - 6

137 | Vishal Baraiya | 23010101014

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 [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
iris = pd.read_csv('iris.csv')
iris.shape
```

Out[2]:
```
(150, 5)
```

In [3]:
```
iris
```

Out[3]:

	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

Step 2: Standardize the data (zero mean)

In [4]:
```
x = iris.drop(columns='species')
y = iris['species'].map({
    'setosa' : 0,
    'versicolor' : 1,
    'virginica' : 2
})
```

In [5]:
```
print("Original shape : ", x.shape)
```

```
Original shape :  (150, 4)
```

In [6]:
```
x_meaned = x - np.mean(x, axis = 0)
print("Data after centering ( first 5 rows ) : \n",x_meaned[:5])
```

```
Data after centering ( first 5 rows ) :
    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 [7]:
```
cov_mat = np.cov(x_meaned, rowvar=False)
print("Covariance matrix  shape :", cov_mat.shape)
cov_mat
```

```
Covariance matrix  shape : (4, 4)
```

Out[7]:
```
array([[ 0.68569351, -0.042434  ,  1.27431544,  0.51627069],
       [-0.042434  ,  0.18997942, -0.32965638, -0.12163937],
       [ 1.27431544, -0.32965638,  3.11627785,  1.2956094 ],
       [ 0.51627069, -0.12163937,  1.2956094 ,  0.58100626]])
```

Step 4: Compute eigenvalues and eigenvectors

In [8]:
```
eigen_values, eigen_vectors = np.linalg.eigh(cov_mat)

print("Eigenvalues:\n", eigen_values)
print("Eigenvectors (first 2):\n",eigen_vectors[:,:2])
```

```
Eigenvalues:
 [0.02383509 0.0782095  0.24267075 4.22824171]
Eigenvectors (first 2):
 [[ 0.31548719  0.58202985]
 [-0.3197231  -0.59791083]
 [-0.47983899 -0.07623608]
 [ 0.75365743 -0.54583143]]
```

Step 5: Sort eigenvalues and eigenvectors in descending order

In [9]:
```
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalues = eigen_values[sorted_index]
sorted_eigenvectors = eigen_vectors[:,sorted_index]
```

In [10]:
```
print(sorted_index)
print(sorted_eigenvalues)
print(sorted_eigenvectors)
```

```
[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 [11]:
```
k = 2
eigenvector_subset = sorted_eigenvectors[:, 0:k]
print(eigenvector_subset);
```

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

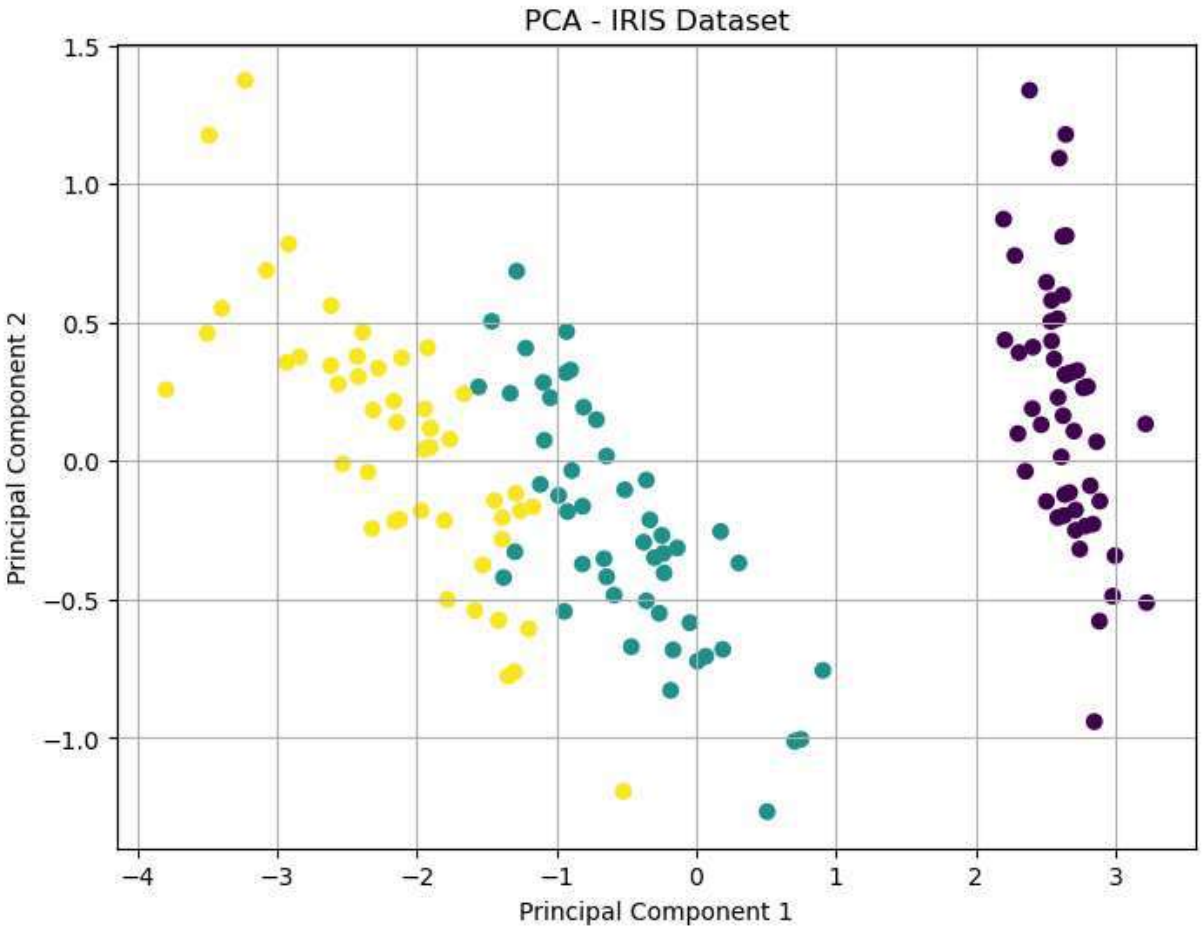
Step 7: Project the data onto the top k eigenvectors

In [12]:
```
x_reduced = np.dot(x_meaned, eigenvector_subset)
print("Reduced data shape:", x_reduced.shape)
```

```
Reduced data shape: (150, 2)
```

Step 8: Plot the PCA-Reduced Data

In [13]:
```
plt.figure(figsize=(8, 6))
plt.scatter(x_reduced[:, 0],x_reduced[:, 1], c=y)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA - IRIS Dataset")
plt.grid(True)
plt.show()
```



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 [14]:
```
values = np.array([5,10,11,13,15,35,50,55,72,92,204,215])

# (a) Equal-Frequency Bins:
equal_freq_bins = pd.qcut(values, q=3, labels=["Bin 1","Bin 2","Bin 3"])

# (b) Equal-Width Bins:
# max - min / noOfBins
equal_width_bins = pd.cut(values, bins=3, labels=["Bin 1","Bin 2","Bin 3"])

# Combine results
binning_results = pd.DataFrame({
    "Value": values,
    "Equal-Frequency Bin": equal_freq_bins,
    "Equal-Width Bin" : equal_width_bins
```

```
})
binning_results
```

Out[14]:

	Value	Equal-Frequency Bin	Equal-Width Bin
0	5	Bin 1	Bin 1
1	10	Bin 1	Bin 1
2	11	Bin 1	Bin 1
3	13	Bin 1	Bin 1
4	15	Bin 2	Bin 1
5	35	Bin 2	Bin 1
6	50	Bin 2	Bin 1
7	55	Bin 2	Bin 1
8	72	Bin 3	Bin 1
9	92	Bin 3	Bin 2
10	204	Bin 3	Bin 3
11	215	Bin 3	Bin 3

In [ ]:

★ Apriori Algorithm :-

minimum support = 2

Ex-1

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

C_1

frequent-1

Itemset	min sup	
{1}	2	
{2}	3	
{3}	3	
{4}	1	✗
{5}	3	

L_1

Items	support
{1}	2
{2}	3
{3}	3
{5}	3

frequent-2

Itemset	support	
{1,2}	1	✗
{1,3}	2	
{1,5}	1	✗
{2,3}	2	
{2,5}	3	
{3,5}	2	

L_2

Items	support
{1,3}	2
{2,3}	2
{2,5}	3
{3,5}	3

frequent-3

Itemset	support	
{1,2,3}	1	✗
{1,3,5}	1	✗
{2,3,5}	2	✓

L_3

Items	support
{2,3,5}	2

* Rule generation

confidence = support count of {2^3^5} / support count of {2^3}

Associan Rule	support	confidence	confidence (%)
1^2 → 5	2	2/2 = 1	100%
3^5 → 2	2	2/2 = 1	100%
2^5 → 3	2	2/3 = 0.66	66%
2 → 3^5	2	2/3 = 0.66	66%
3 → 2^5	2	2/3 = 0.66	66%
5 → 2^3	2	2/3 = 0.66	66%

ex-2

minimum support = 3

TID	Items
1	Bread, Milk
2	Bread, Diapes, Beer, Eggs
3	Milk, Diapes, Beer, Coke
4	Milk, Diapes, Beer, Coke
5	Bread, Milk, Diapes, Coke

Frequent-1

C_1

Items	support
{Bread}	3
{Milk}	4
{Diapes}	4
{Beer}	3
{Eggs}	1 ✗
{Coke}	3

L_1

Items	support
{Bread}	3
{Milk}	4
{Diapes}	4
{Beer}	3
{Coke}	3

Frequent-2

C_2

Items	support	
{Bread, Milk}	2	✗
{Bread, Diapes}	2	✗
{Bread, Beer}	1	✗
{Bread, Coke}	1	✗
{Milk, Diapes}	3	
{Milk, Beer}	2	✗
{Milk, Coke}	3	
{Diapes, Beer}	3	
{Diapes, Coke}	3	
{Beer, Coke}	2	✗

L_2

Items	support
{Milk, Diapes}	3
{Milk, Coke}	3
{Diapes, Beer}	3
{Diapes, Coke}	3

Frequent-3

C_3

Items	support
{milk, Diapes, cola}	3
{milk, Diapes, Bees}	2
{Diapes, Bees, cola}	2

L_3

Items	support
{milk, Diapes, cola}	3

*Rule generation

Association Rule	Support	Confidence	Confidence (%)	
{milk, Diapes} → {cola}	3	3/3 = 1	100%	~
{milk, cola} → {Diapes}	3	3/3 = 1	100%	~
{Diapes, cola} → {milk}	3	3/3 = 1	100%	~
{cola} → {milk, Diapes}	3	3/3 = 1	100%	~
{Diapes} → {milk, cola}	3	3/4 = 0.75	75%	
{milk} → {Diapes, cola}	3	3/4 = 0.75	75%	

Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Data Mining

Lab - 7 (Part 2)

137 | Vishal Baraiya | 23010101014

Step 1: Load the Dataset

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

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

In [2]:
```
Tdata = pd.read_csv("Tdata.csv")
Tdata
```

Out[2]:

	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 [3]:
```
df_item = Tdata.drop(columns=["Transaction"])
df_item.head()
```

Out[3]:

	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

Step 3: Count Single Items

See how many transactions include each item.

In [4]:
```
df_item.sum()
```

Out[4]:
```
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 [5]:
```
from itertools import combinations

def find_frequent_itemsets(df,min_support):
    n = len(df)
    result = []

    for k in [1,2,3]: # for 1-item, 2-item, 3-item
        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 [6]:
```
frequent_itemsets = find_frequent_itemsets(df_item,min_support=0.5)
```

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

```
{'bread'} -> support : 0.83
{'butter'} -> support : 0.5
{'milk'} -> support : 0.5
{'butter', 'bread'} -> support : 0.5
{'bread', 'milk'} -> support : 0.5
```

Step 6 Display as a DataFrame

In [7]:
```
result_df = pd.DataFrame(frequent_itemsets, columns=['Itemset','Support'])
result_df
```

Out[7]:

	Itemset	Support
0	(bread)	0.83
1	(butter)	0.50
2	(milk)	0.50
3	(butter, bread)	0.50
4	(bread, milk)	0.50

In [ ]:
```

```

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

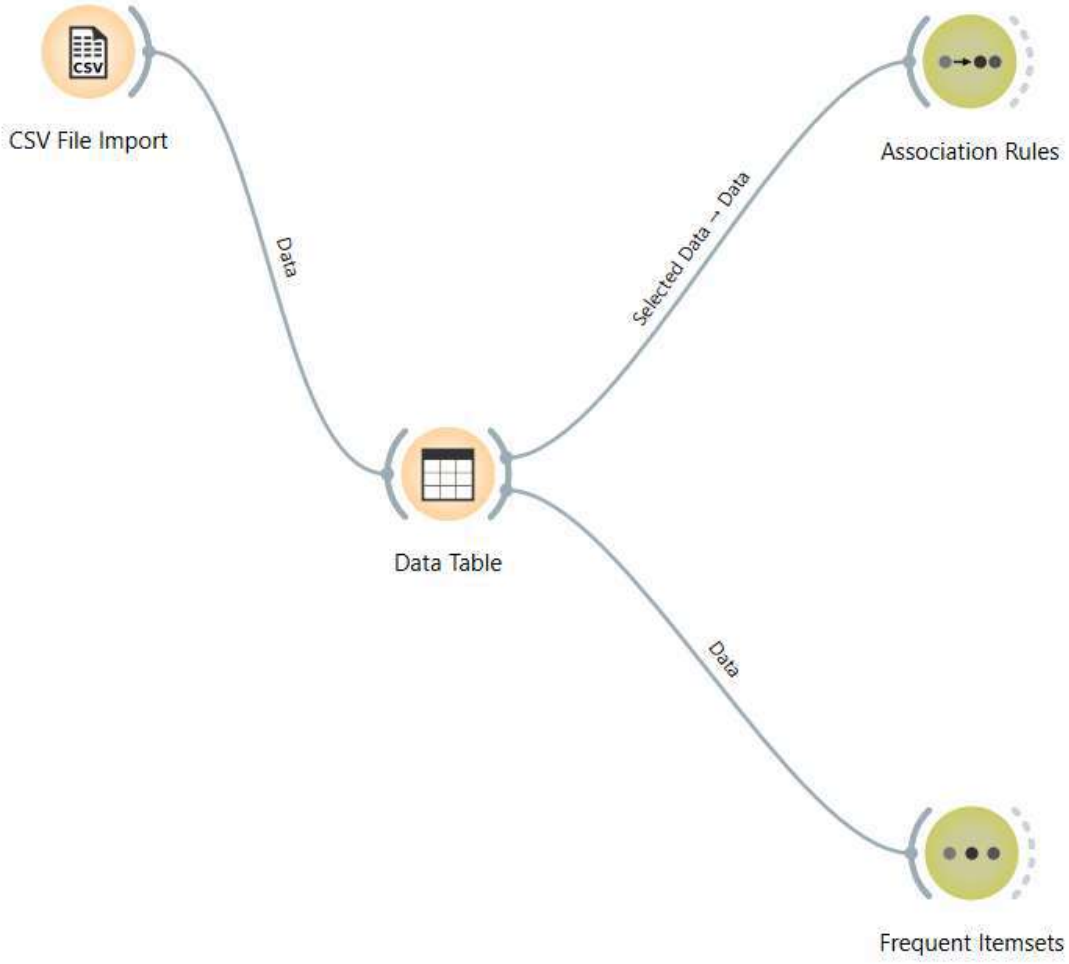
In [ ]:
```

```

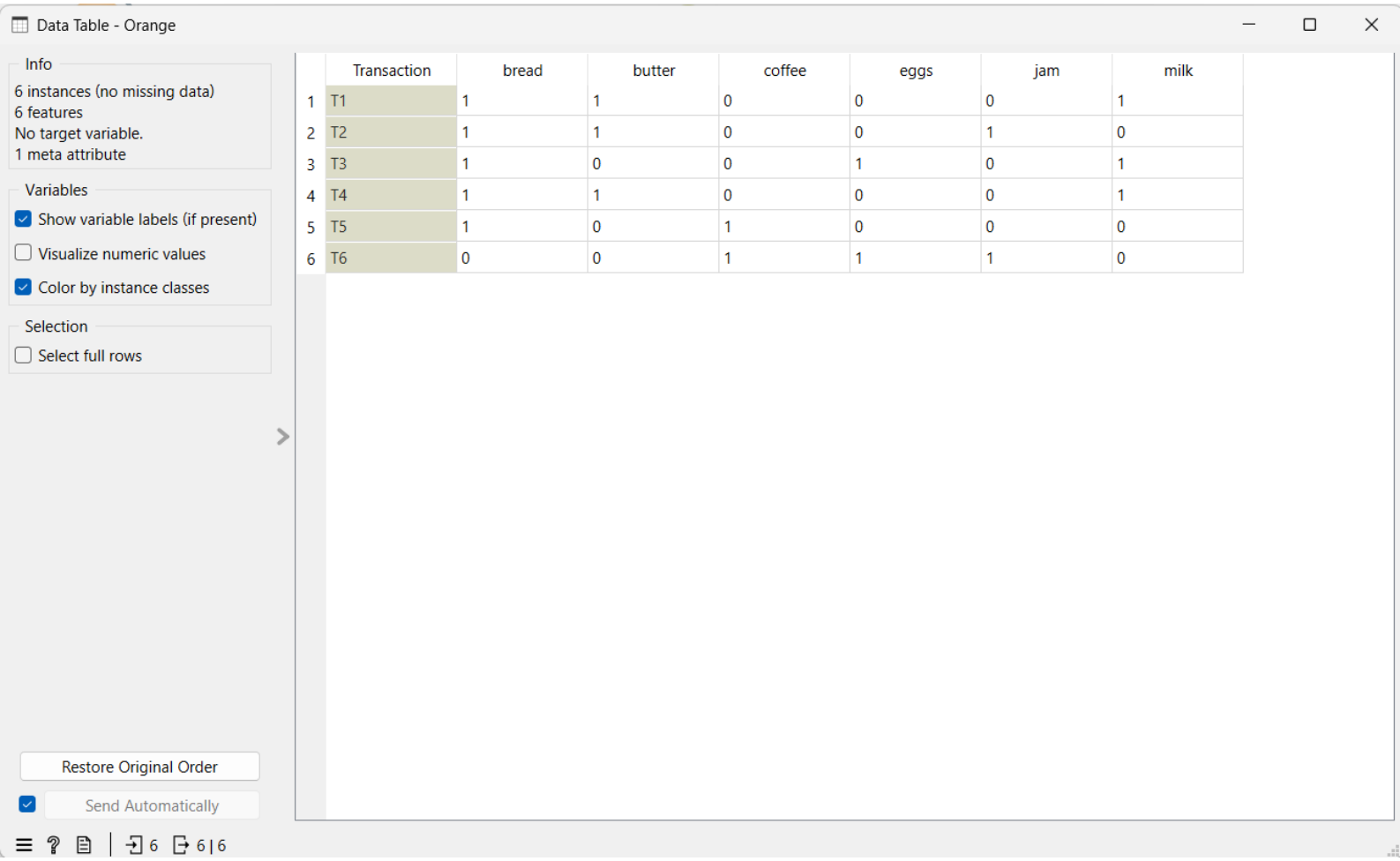
Extra : - > Define Apriori Function without itertools

In [8]:
```
# def find_frequent_itemsets(df, min_support):
#     n = len(df)
#     result = []

#     columns = list(df.columns)

#     # 1-itemsets
#     for i in range(len(columns)):
#         item1 = columns[i]
#         mask = df[item1] == 1
#         support = mask.sum() / n
#         if support >= min_support:
#             result.append((frozenset([item1]), round(support, 2)))
```

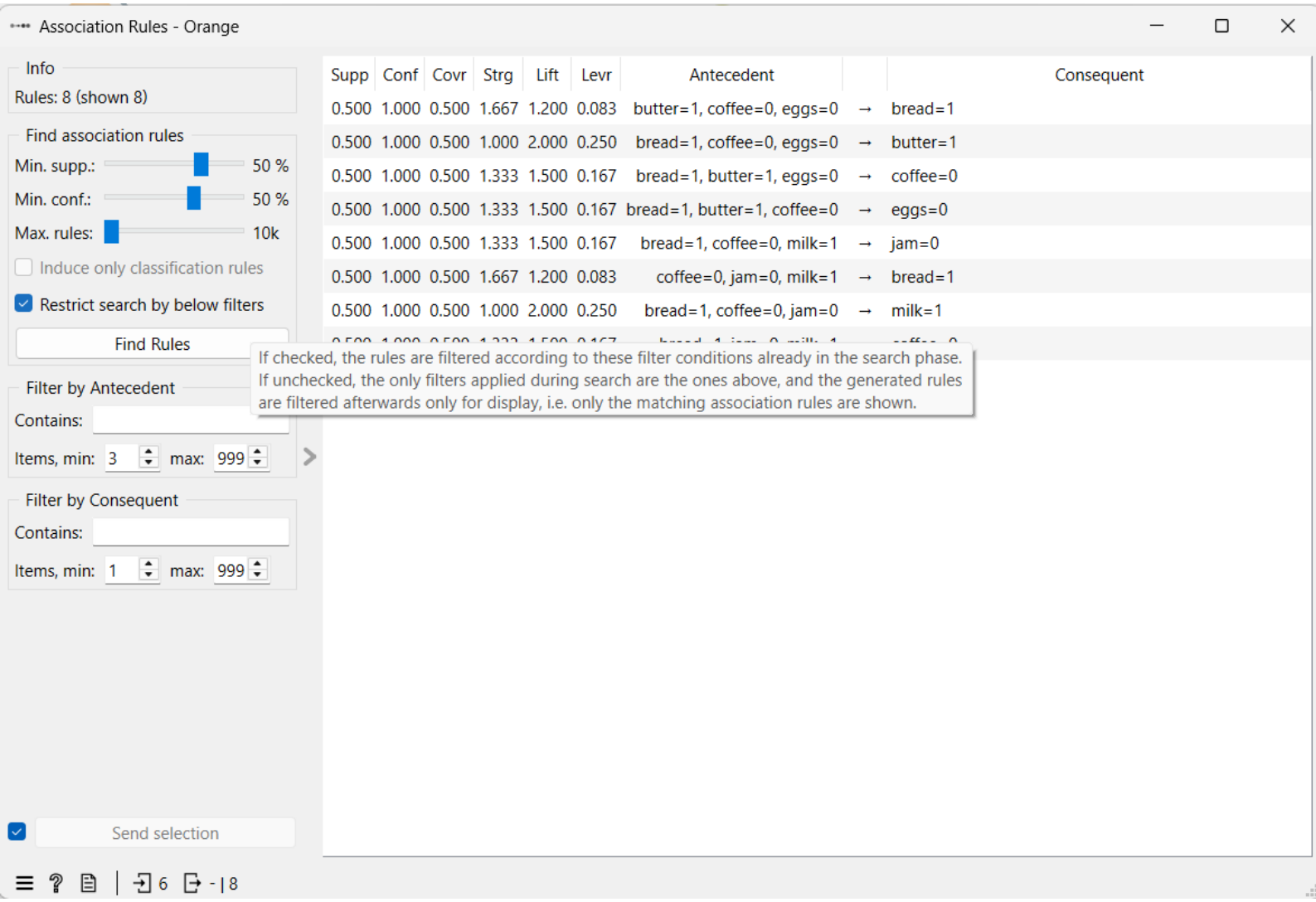
```
#     # 2-itemsets
#     for i in range(len(columns)):
#         for j in range(i + 1, len(columns)):
#             item1 = columns[i]
#             item2 = columns[j]
#             mask = df[item1] & df[item2]
#             support = mask.sum() / n
#             if support >= min_support:
#                 result.append((frozenset([item1, item2]), round(support, 2)))

#     # 3-itemsets
#     for i in range(len(columns)):
#         for j in range(i + 1, len(columns)):
#             for k in range(j + 1, len(columns)):
#                 item1 = columns[i]
#                 item2 = columns[j]
#                 item3 = columns[k]
#                 mask = df[item1] & df[item2] & df[item3]
#                 support = mask.sum() / n
#                 if support >= min_support:
#                     result.append((frozenset([item1, item2, item3]), round(suppor

#     return result
```

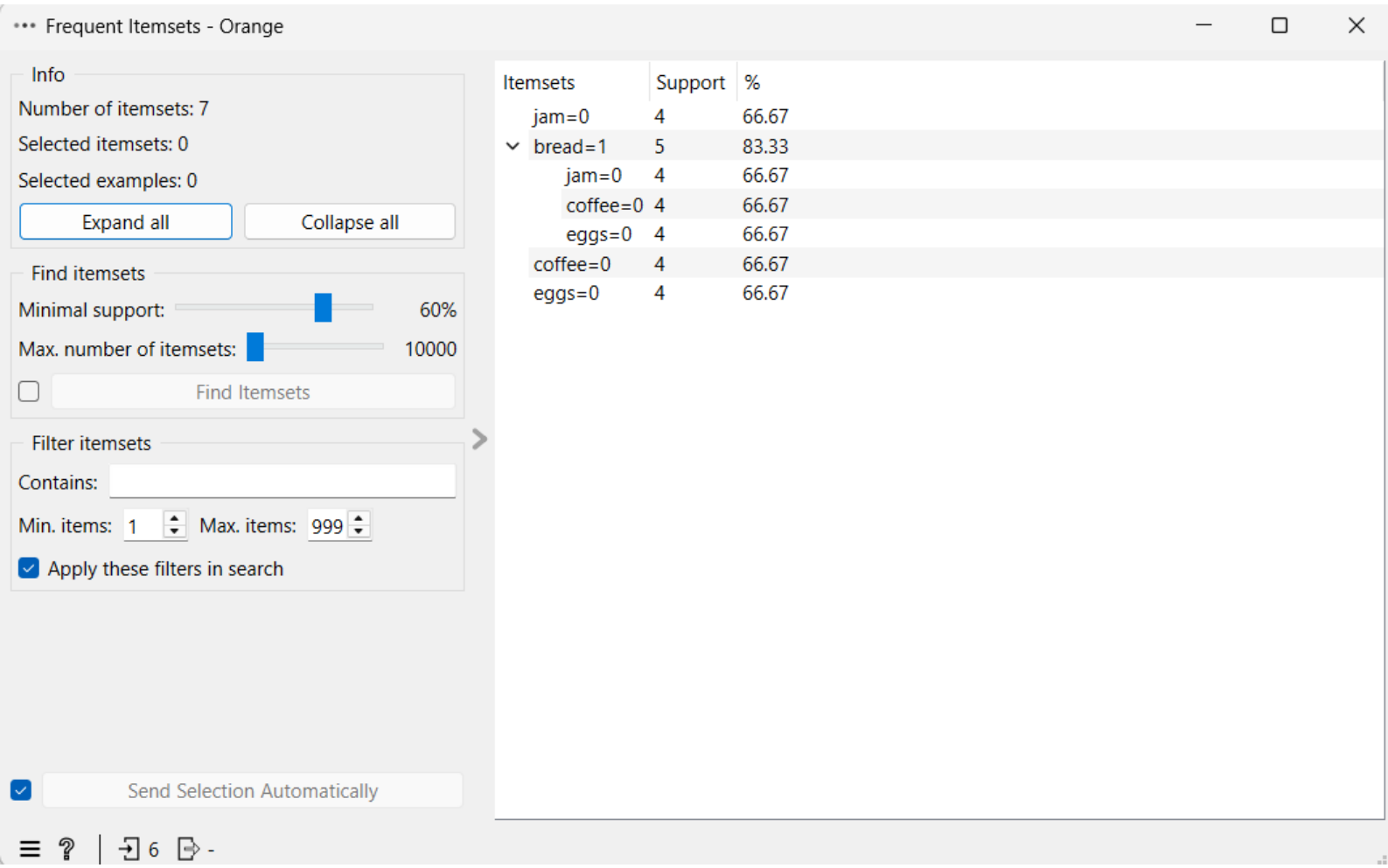
In [ ]:


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



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


Association Rules - Orange
Info
Rules: 8 (shown 8)
Find association rules
Min. supp.: 50 %
Min. conf.: 50 %
Max. rules: 10k
Induce only classification rules
Restrict search by below filters
Find Rules
If checked, the rules are filtered according to these filter conditions already in the search phase. If unchecked, the only filters applied during search are the ones above, and the generated rules are filtered afterwards only for display, i.e. only the matching association rules are shown.
Filter by Antecedent
Contains:
Items, min: 3 max: 999
Filter by Consequent
Contains:
Items, min: 1 max: 999
Send selection
6
8
Supp Conf Covr Strg Lift Levr Antecedent Consequent
0.500 1.000 0.500 1.667 1.200 0.083 butter=1, coffee=0, eggs=0 → bread=1
0.500 1.000 0.500 1.000 2.000 0.250 bread=1, coffee=0, eggs=0 → butter=1
0.500 1.000 0.500 1.333 1.500 0.167 bread=1, butter=1, eggs=0 → coffee=0
0.500 1.000 0.500 1.333 1.500 0.167 bread=1, butter=1, coffee=0 → eggs=0
0.500 1.000 0.500 1.333 1.500 0.167 bread=1, coffee=0, milk=1 → jam=0
0.500 1.000 0.500 1.667 1.200 0.083 coffee=0, jam=0, milk=1 → bread=1
0.500 1.000 0.500 1.000 2.000 0.250 bread=1, coffee=0, jam=0 → milk=1


Frequent Itemsets - Orange
Info
Number of itemsets: 7
Selected itemsets: 0
Selected examples: 0
Expand all
Collapse all
Find itemsets
Minimal support:
60%
Max. number of itemsets:
10000
Find Itemsets
Filter itemsets
Contains:
Min. items:
1
Max. items:
999
Apply these filters in search
Send Selection Automatically
6
-
Itemsets
Support
%
jam=0 4 66.67
bread=1 5 83.33
jam=0 4 66.67
coffee=0 4 66.67
eggs=0 4 66.67
coffee=0 4 66.67
eggs=0 4 66.67

☆ Frequent pattern Tree :-

ex-1

TID	Items
1	E K M N R O Y
2	D E K N O Y
3	A E K M
4	C K M U Y
5	C E I K O

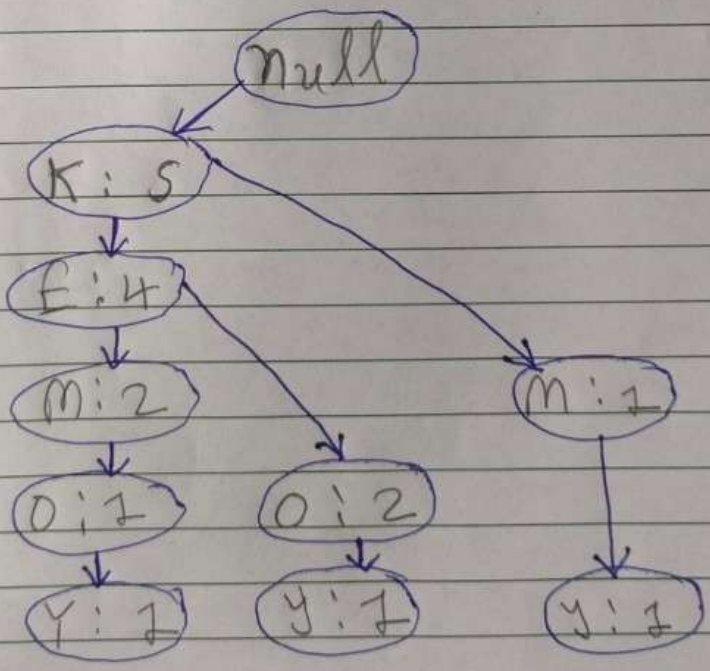
Step:-1 min_sup ≥ 3

Items	frequency	
A	1	X
C	2	X
D	1	X
E	4	✓
I	1	X
K	5	✓
M	3	✓
N	2	X
O	3	✓
U	1	X
Y	3	✓

step:-2

TID	sorted_Items
1	K E M O Y
2	K E O Y
3	K E M
4	K M Y
5	K E O

step:-3

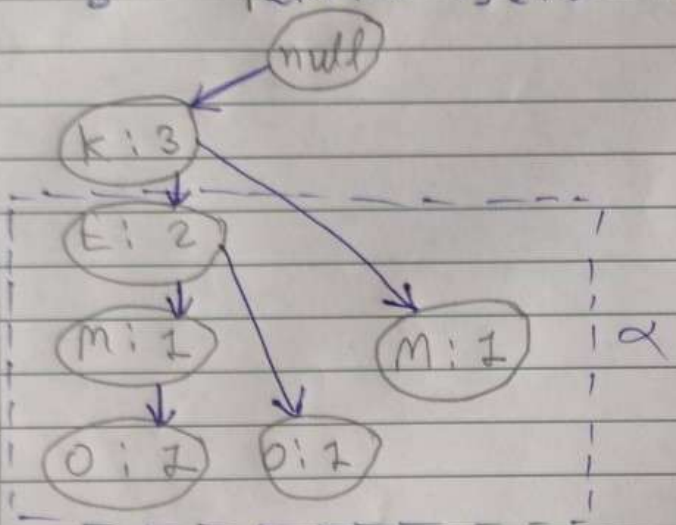


step:-4 conditional pattern

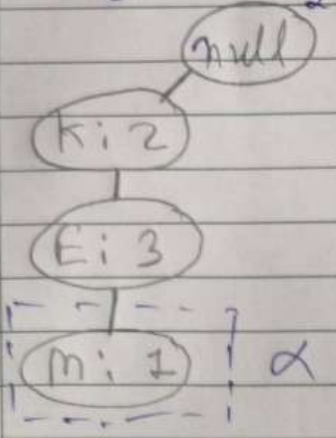
Item	conditional pattern
Y	{KEMO : 1} {KEO : 1} {KM : 1}
O	{KEM : 1} {KE : 2}
M	{KE : 2} {K : 1}
E	{K : 4}
K	-

step:-5

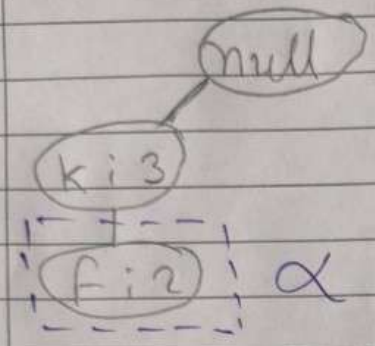
Item	conditional	fp-tree
Y	{KEMO : 1} {KEO : 1} {KM : 1}	{K : 3}



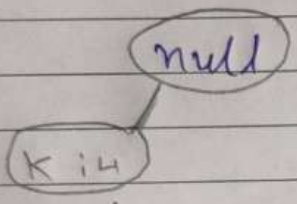
Item	conditional Pattern.	fp-tree
O	{KEM : 1} {KE : 2}	{K : 3, E : 3}



item	conditional Pattern	fp-tree
m	{kf:2} {k:1}	{k:3}



item	conditional Pattern	fp-tree
f	{k:4}	{k:4}



step: 6

Item	conditional Pattern	fp-tree	frequent Pattern generated
y	{kfmo:1} {kfo:1} {km:1}	{k:3}	{k,y:3}
o	{kfm:1} {kf:2}	{k:3, f:3}	{k,o:3}
m	{kf:2} {k:1}	{k:3}	{k,m:3} {f,o:3} {k,f,o:3}
f	{k:4}	{k:4}	{k,f:4}
k	-		

ex-2

TID	Items
1	1 2 5
2	2 4
3	2 3
4	1 2 4
5	1 3
6	2 3
7	1 3
8	1 2 3 5
9	1 3 3

min support = 2

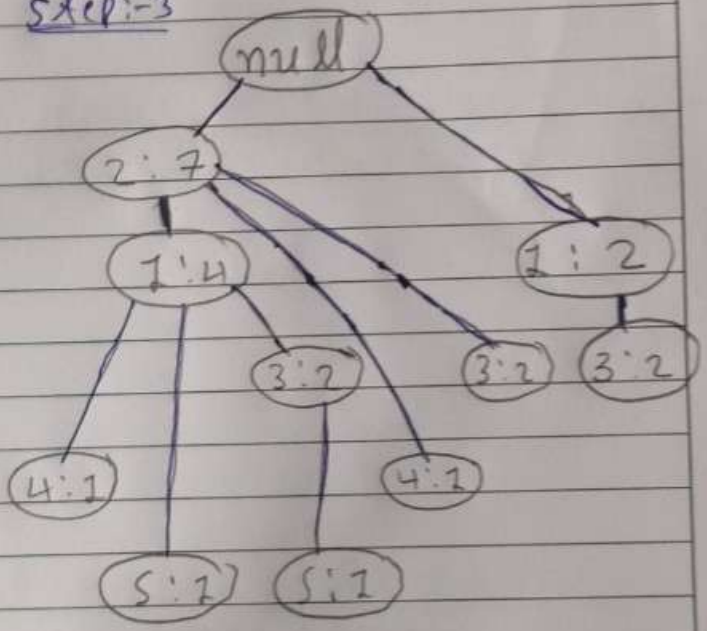
step:-1

Item	frequency
1	6 ✓
2	7 ✓
3	6 ✓
4	2 ✓
5	2 ✓

step:-2

TID	Items
1	2 1 5
2	2 4
3	2 3
4	2 1 4
5	1 3
6	2 3
7	1 3
8	2 1 3 5
9	2 1 3

step:-3

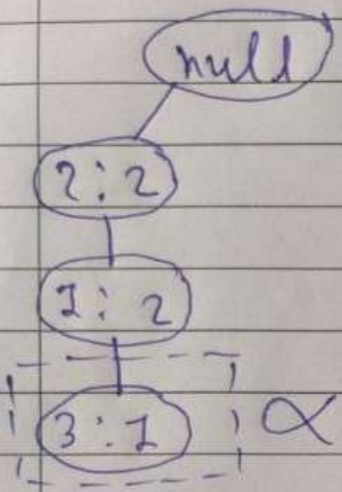


step:-4

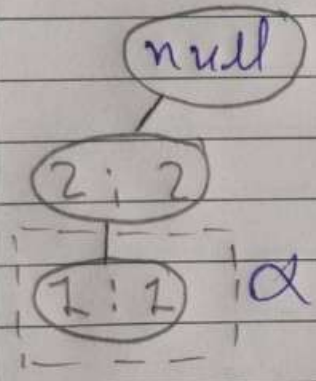
Item	conditioned pattern
5	{2 1 3:1} {2 1:1}
4	{2 1:1} {2:1}
3	{2 1:2} {2:2} {1:2}
1	{2 0:4}
2	

step:-5

Item	conditional pattern	fp-tree
5	{2 1 3:1} {2 1:1}	{2:2, 1:2}

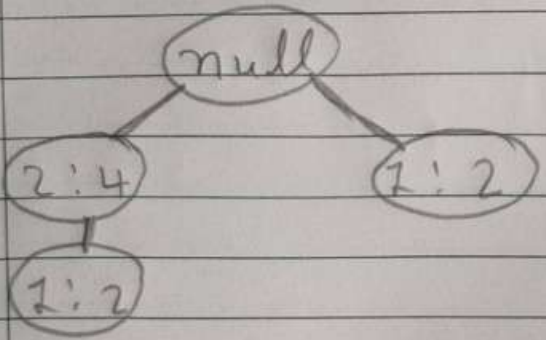


Item	conditional pattern	fp-tree
4	{2 1:1} {2:1}	{2:2}

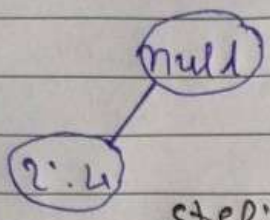


Item	conditional pattern	fp-tree
3	{2 1:2} {2:2} {1:2}	{2:4, 1:2}

duplicate just 1 & 2 ani bcs it is a set



Item	conditional Pattern	FP- Pattern
1	{2:4}	{2:4}



step:-6

Item	conditional Pattern	Frequent FP-tree Patterns gen.
5	{2 1 3:1} {2 1:1}	{2:2, 1:2}
4	{2 1:1} {2:1}	{2:2}
3	{2 1:2} {2:2} {1:2}	{2:4, 1:2}
1	{2:4}	{2:4}
2	-	

{2,5:2} {1,5:2} {2,1,5:2}
{2,4:2}
{2,3:4} {1,3:2} {2,1,3:2}
{2,1:4}