Data Mining

2301010148

Lab - 1

Introduction to Pandas Library Function:

Step-1 Import the pandas Libraries

In [2]:
```
import pandas as pd
```

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

In [ ]:
```
```

Step-3 Read csv or excel File

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

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0	1	0	3	Mr. OwenBraund,Harris	male	22.0	1	0	
211A									

STO O	**1**	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	17
	2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	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
W. 66	**888**	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	
	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 [4]:
```
data
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick

0	1	0	3	Mr. OwenBraund,Harris	male	22.0	1	0	

211A

1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STO O 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 [6]:

```
data.head(10)
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Mr. OwenBraund,Harris	male	22.0	1	0	21171A/5
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 [7]:

```
data.tail(10)
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ti
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./SO3
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 [8]:

```
data.dtypes
```

Out[8]:

```
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 [10]:

```
data.describe()
```

Out[10]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.00000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000

75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000

Step-9 Access a specific column

In [13]:
```
data['Ticket']
```

Out[13]:
```
0              A/5 21171
1               PC 17599
2        STON/O2. 3101282
3                 113803
4                 373450 ...
886               211536
887               112053
888           W./C. 6607
889               111369
890               370376
Name: Ticket, Length: 891, dtype: object
```

In [14]:
```
data.shape
```

Out[14]:
```
(891, 12)
```

Step-10 Access rows by their integer location

In [15]:
```
data.iloc[2]
```

Out[15]:
```
PassengerId                          3
Survived                             1
Pclass                               3
Name            Heikkinen, Miss. Laina
Sex                             female
Age                               26.0
SibSp                                0
Parch                                0
Ticket                STON/O2. 3101282
Fare                             7.925
Cabin                              NaN
Embarked                             S
Name: 2, dtype: object
```

Step-11 Delete a specific Column

In [17]:
```
data.drop("Embarked" , axis='columns' , inplace = True)
```

In [18]:
```
data
```

Out[18]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0 211A	1	0	3	Mr. OwenBraund,Harris	male	22.0	1	0	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	17
STO O **2**	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	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 × 11 columns

Step-12 Create a new Column

In [19]:

```
data["isCabin"] = ~data["Cabin"].isnull()
```

In [20]:

```
data
```

Out[20]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0	1	0	3	Mr. OwenBraund,Harris	male	22.0	1	0	211A

1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	STO O 17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	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-13 Perform Condition Selection on DataFrame

In [21]:

```
data[data['Pclass'] != 3]

#data[data['Age'] > 30]
```

Out[21]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	T
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 1
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	1
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	
...	...	...	...	...	...	...	...	...	
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SO

11

23

11

23

3

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	
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	

11

11

400 rows × 12 columns

Step-14 Compute the sum of value

In [22]:
```
data['Fare'].sum()
```

Out[22]: 28693.9493

Step-15 Compute the mean of value

In [23]:
```
data['Fare'].mean()
```

Out[23]: 32.204207968574636

Step-16 Count non-null value (column)

In [24]:
```
(~data.isnull()).sum()
```

Out[24]:
```
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 [25]:
```
data['Fare'].max()
```

Out[25]: 512.3292

In [26]:
```
data['Fare'].min()
```

Out[26]: 0.0 In [ ]:

Data Mining

23010101048

Lab - 2

Numpy & Perform Data Exploration with Pandas

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 [3]:
```
import numpy as np
```

Step 2. Create a 1D array of numbers

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

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

In [104…

```
a = np.arange(2,10)
print(a)
```

```
[2 3 4 5 6 7 8 9]
```

Step 3. Reshape 1D to 2D Array

In [24]:

```
b = np.arange(12).reshape(4,3)
print(b)
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

Step 4. Create a Linspace array

In [11]:

```
np.linspace(0,5,20)
```

Out[11]:

```
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 [20]:

```
c = np.random.rand(2,4)
print(c)
```

```
[[0.95394937 0.80705321 0.76256859 0.38664459]
 [0.86244858 0.55639819 0.51719692 0.48473668]]
```

Step 6. Create a Random Integer Array

In [25]:

```
c = np.random.randint(10,20,5) d =
np.random.randint(10,20,size = (2,4))
print(c) print(d)
```

```
[12 18 18 17 16]
[[10 15 14 11]
 [15 19 13 16]]
```

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

In [28]:

```
arr = np.random.randint(1,100,10)
arr
```

Out[28]:

```
array([ 5, 18, 98, 34, 63, 63, 65, 63, 94, 13])
```

In [29]:
```python
print(arr.min())
```
```
5
```

In [30]:
```python
print(arr.max())
```
```
98
```

In [31]:
```python
arr.argmin()
```
Out[31]:
```
0
```

In [32]:
```python
arr.argmax()
```
Out[32]:
```
2
```

Step 8. Indexing in 1D Array

In [33]:
```python
arr[2]
```
Out[33]:
```
98
```

In [34]:
```python
arr[2:5]
```
Out[34]:
```
array([98, 34, 63])
```

Step 9. Indexing in 2D Array

In [36]:
```python
f = arr.reshape(2,5)
f
```
Out[36]:
```
array([[ 5, 18, 98, 34, 63],
       [63, 65, 63, 94, 13]])
```

In [37]:
```python
f[0]
```
Out[37]:
```
array([ 5, 18, 98, 34, 63])
```

In [39]:
```python
f[1][2]
```
Out[39]:
```
63
```

Step 10. Conditional Selection

[42]:
```
f[:1,2:]
```

Out[42]:
```
array([[98, 34, 63]])
```

In [45]:
```
f[:1,:]
```

Out[45]:
```
array([[ 5, 18, 98, 34, 63]])
```

In [48]:
```
arr[arr>4]
```

Out[48]:
```
array([ 5, 18, 98, 34, 63, 63, 65, 63, 94, 13])
```

In [58]:
```
array =
np.arange(15).reshape(3,5) array
array[1:2:,1::]
```

Out[58]:
```
array([[6, 7, 8, 9]])
```

In [61]:
```
array[array>4]
```

Out[61]:
```
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

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

Pandas

Step 1. Import the necessary libraries

In [75]:
```
import pandas as pd
import numpy as np
```

Step 2. Import the dataset from this address.

In [64]:
```
users                                                                              =
pd.read_csv("https://raw.githubusercontent.com/justmarkham/DAT8/master  users
```

Out[64]:

	user_id	age	gender	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043

In

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

In [66]:

```
users.set_index('user_id')
```

Out[66]:

	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 × 4 columns

Step 4. See the first 25 entries

```
[67]: users.head(25)
```

Out[67]:

	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

In

23	24	21	F	artist	94533
24	25	39	M	engineer	55107

Step 5. See the last 10 entries

In [121…

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

Out[121…

	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 [68]:

```
users.count()
```

Out[68]:

```
user_id       943
age           943
gender        943
occupation    943
zip_code      943
dtype: int64
```

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

In [69]:

```
users.shape[1]
```

Out[69]: 5

Step 8. Print the name of all the columns.

In [71]:

```
users.columns
```

Out[71]: Index(['user_id', 'age', 'gender', 'occupation', 'zip_code'],

dtype='object')

Step 9. How is the dataset indexed?

In [78]:
```
users.index
```

Out[78]: RangeIndex(start=0, stop=943, step=1)

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

[80]:
```
users.dtypes
```

Out[80]:
```
user_id        int64
age            int64
gender        object
occupation    object
zip_code      object
dtype: object
```

Step 11. Print only the occupation column

In [81]:
```
users['occupation']
```

Out[81]:
```
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 [84]:
```
users['occupation'].nunique()
```

Out[84]: 21

Step 13. What is the most frequent occupation?

In [89]:
```
users['occupation'].value_counts().head(1)
```

Out[89]:
```
occupation student
196 Name: count, dtype:
int64
```

In

Step 14. Summarize the DataFrame.

In [91]:
```
users.describe()
```

Out[91]:

	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 [95]:
```
users.describe(include = 'all')
```

Out[95]:

	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 [96]:
```
users['occupation'].describe()
```

Out[96]:
```
count          943
unique          21
```

```
top        student
freq           196
Name: occupation, dtype: object
```

Step 17. What is the mean age of users?

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

```
Out[98]: 34.05196182396607
```

Step 18. What is the age with least occurrence?

```
In [103… users['age'].value_counts().tail()
```

```
Out[103… 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 [ ]:
```

In

Darshan UNIVERSITY

योगः कर्मसु कौशलम्

Data Mining

23010101048

Lab - 3

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

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

In [2]:
```
import numpy as np
```

In [11]:
```
data = pd.read_csv('titanic.csv')
data.head(5)
```

Out[11]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Mr. OwenBraund,Harris	male	22.0	1	0	21171A/5
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

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

In [12]:

```
nominal=['Name','Sex','Ticket','Cabin','Embarked']
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', 'Ticket', 'Cabin', 'Embarked']
Ordinal:  ['Pclass']
Binary:  ['Sex', 'Survived']
Numeric:  ['Age', 'Fare', 'SibSp', 'Parch']
```

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

In [14]:
```
print('survived values (asymmetric binry): ')
print(data['Survived'].value_counts())
print('Gender count (symmetric binary): ')
print(data['Sex'].value_counts())
```

```
survived values (asymmetric binry):
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 [16]:
```
data.dtypes quantitative =

['PassengerId','Survived','Pclass','SibSp','Age','Parch','Fare'

for col in quantitative:
    print(col)
    print('Average of',col, 'is : ',data[col].mean())
    print('Standard deviation of',col, 'is :
    ',data[col].std()) print('Minimum of',col, 'is :
    ',data[col].min()) print('Maximum of',col, 'is :
    ',data[col].max()) print('Mode of',col, 'is :
    ',data[col].mode()[0])
    print('Range of',col, 'is : ',data[col].max()-data[col].min())
    print()
```

```
PassengerId
Average of PassengerId is :  446.0
Standard deviation of PassengerId is :  257.3538420152301
Minimum of PassengerId is :  1
Maximum of PassengerId is :  891
Mode of PassengerId is :  1
Range of PassengerId is :  890

Survived
Average of Survived is :  0.3838383838383838
Standard deviation of Survived is :  0.4865924542648585
Minimum of Survived is :  0
Maximum of Survived is :  1
Mode of Survived is :  0
Range of Survived is :  1

Pclass
Average of Pclass is :  2.308641975308642
Standard deviation of Pclass is :  0.8360712409770513
Minimum of Pclass is :  1
Maximum of Pclass is :  3
Mode of Pclass is :  3
Range of Pclass is :  2

SibSp
Average of SibSp is :  0.5230078563411896
Standard deviation of SibSp is :  1.1027434322934275
Minimum of SibSp is :  0
Maximum of SibSp is :  8
Mode of SibSp is :  0
Range of SibSp is :  8

Age
Average of Age is :  29.69911764705882
Standard deviation of Age is :  14.526497332334044
Minimum of Age is :  0.42
Maximum of Age is :  80.0
Mode of Age is :  24.0
Range of Age is :  79.58

Parch
Average of Parch is :  0.38159371492704824
Standard deviation of Parch is :  0.8060572211299559
Minimum of Parch is :  0
Maximum of Parch is :  6
Mode of Parch is :  0
Range of Parch is :  6

Fare
Average of Fare is :  32.204207968574636
Standard deviation of Fare is :  49.693428597180905
Minimum of Fare is :  0.0
```

```
Maximum of Fare is :  512.3292
Mode of Fare is :  8.05
Range of Fare is :  512.3292
```

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

In [17]:
```
data['Pclass'].value_counts()
```

Out[17]:
```
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 [19]:
```
print('Summary for numeric attributes: ')
print(data.describe())
print()
print('summary for all')
print(data.describe(include='all'))
print()
print('Summary for catagorical attributes')
data.describe(include = ['object'])
```

```
Summary for numeric attributes:
       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

summary for all
        PassengerId    Survived      Pclass                     Name   Sex
\ count    891.000000  891.000000  891.000000                      891   891
unique          NaN         NaN         NaN                      891     2
top             NaN         NaN         NaN  Braund, Mr. Owen Harris  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  \
count  714.000000  891.000000  891.000000     891  891.000000      204
unique        NaN         NaN         NaN     681         NaN      147
top           NaN         NaN         NaN  347082         NaN  B96 B98
freq          NaN         NaN         NaN       7         NaN        4
mean    29.699118    0.523008    0.381594     NaN   32.204208      NaN
std     14.526497    1.102743    0.806057     NaN   49.693429      NaN
min      0.420000    0.000000    0.000000     NaN    0.000000      NaN
25%     20.125000    0.000000    0.000000     NaN    7.910400      NaN
50%     28.000000    0.000000    0.000000     NaN   14.454200      NaN
75%     38.000000    1.000000    0.000000     NaN   31.000000      NaN
max     80.000000    8.000000    6.000000     NaN  512.329200      NaN

       Embarke
d         count
889      unique
```

```
3 top           S
freq          644
mean          NaN
std           NaN
min           NaN
25%           NaN
50%           NaN
75%           NaN
max           NaN

Summary for catagorical attributes
```

Out[19]:

	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 thecovariance and correlation between pairs of attributes.

In [20]:

```python
print('Covariance Matrix: ')
data.cov(numeric_only=True)
```

```
Covariance Matrix:
```

Out[20]:

	PassengerId	Survived	Pclass	Age	SibSp	Par
PassengerId	66 31.000000	-0.626966	-7.561798	138.696504	-16.325843	-0.342
Survived	-0.626966	0.236772	-0.137703	-0.551296	-0.018954	0.0320
Pclass	-7.561798	-0.137703	0.699015	-4.496004	0.076599	0.0124
Age	138.696504	-0.551296	-4.496004	211.019125	-4.163334	-2.3441
SibSp	-16.325843	-0.018954	0.076599	-4.163334	1.216043	0.3687
Parch	-0.342697	0.032017	0.012429	-2.344191	0.368739	0.6497
Fare	161.883369	6.221787	-22.830196	73.849030	8.748734	8.6610

In [21]:
```
print('Correlation Matrix: ')
data.corr(numeric_only=True)
```

```
Correlation Matrix:
```

Out[21]:

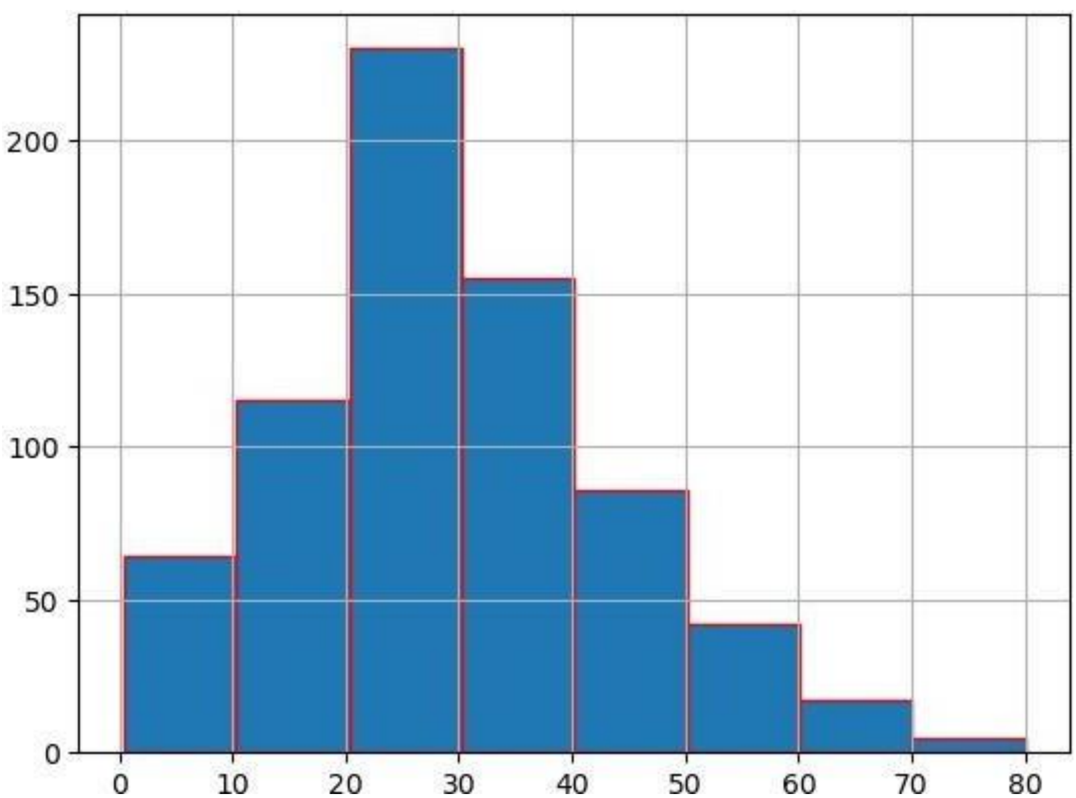
	PassengerId	Survived	Pclass	Age	SibSp	Parch
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119
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

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

In [23]:
```
import matplotlib.pyplot as plt
```

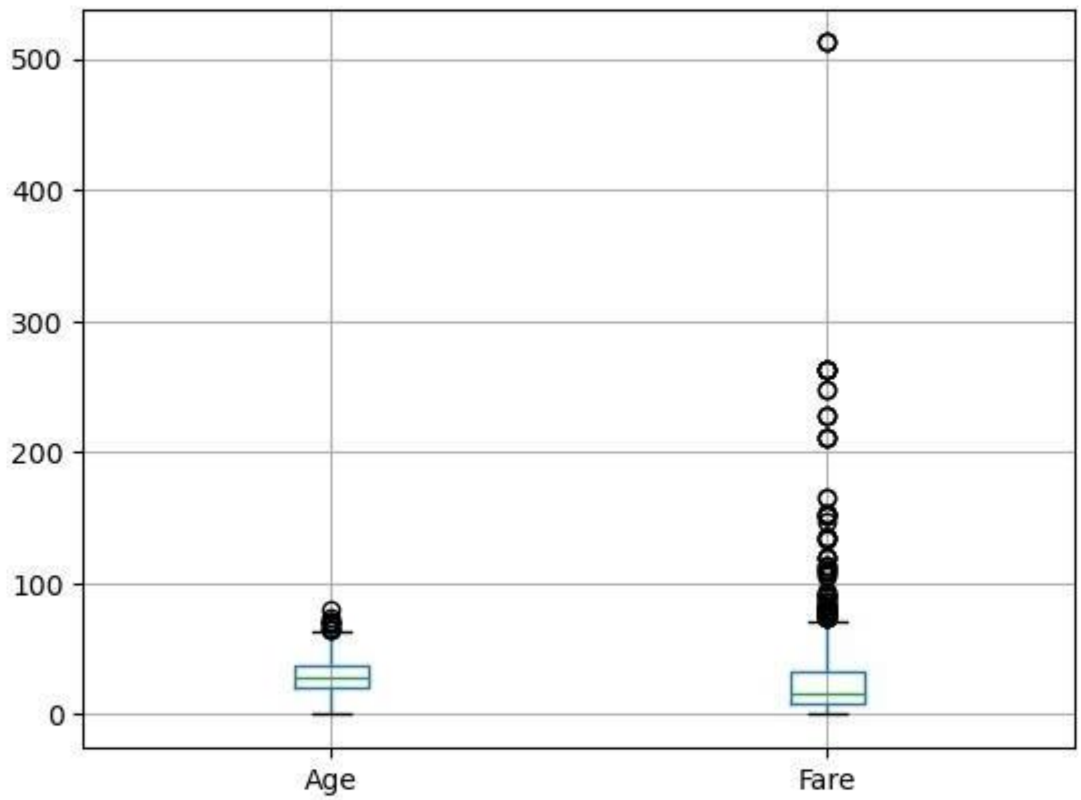
In [24]:
```
data['Age'].dropna().hist(bins=8,edgecolor='red')
```

Out[24]: <Axes: >



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

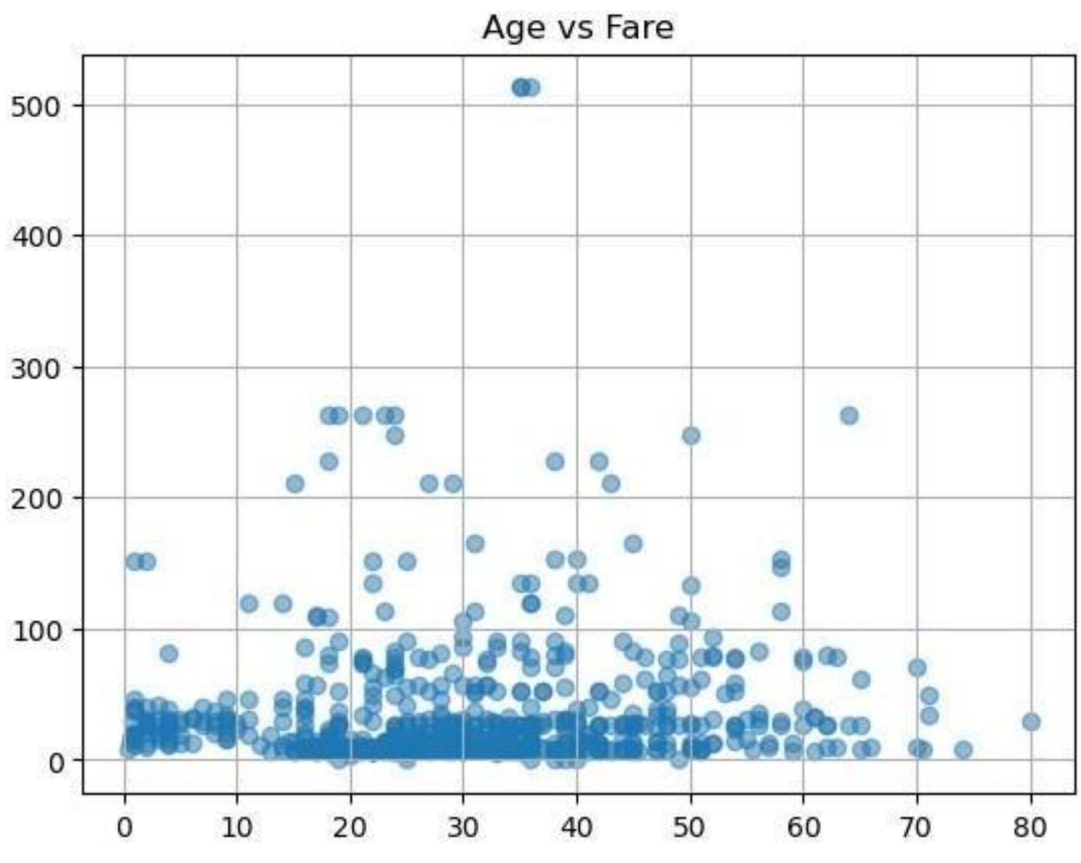
In [26]:
```
numeric = ['Age', 'Fare']
data = data[numeric].dropna()
data.boxplot(column =
numeric)
plt.grid(True)
plt.show()
```



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

In [51]:
```
plt.scatter(data['Age'],data['Fare'], alpha=0.5)
plt.title('Age vs Fare')
plt.grid()
plt.show()


# pair = [('Age', 'Fare'), ('Age', 'SibSp'), ('Age', 'Parch'),('Fare', 'Parch'
# plt.figure(figsize = (15,10)) #
for i, (x,y) in enumerate(pair):
#     plt.subplot(2,3,i+1)
#     plt.scatter(data[x], data[y], alpha=0.6)
#     plt.title(f"{x.capitalize()} vs {y.capitalize()}")
#     plt.xlabel(x.capitalize())
#     plt.ylabel(y.capitalize())
#     plt.grid(True)
```



In [ ]:

Data Mining

2301010148

Lab - 4

Step 1. Import the necessary libraries

In [22]:
```
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 [2]:
```
chipo =
pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master
```

Step 4. See the first 10 entries

In [3]:
```
chipo.head(10)
```

Out[3]:

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and FreshTomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and TomatilloGreen 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 [8]:
```
# Solution 1
print(f' Number of observation in the dataset are: {chipo.shape[0]}')
```

```
Number of observation in the dataset are: 4622
```

In [10]:
```
# Solution 2
print(f' Number of observation in the dataset are: {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   objectdtypes: int64(2),
 object(3) memory usage: 180.7+ KB
 Number of observation in the dataset are: None
```

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

In [11]:
```
print(f' Number of columns in the dataset are: {chipo.shape[1]}')
```

```
Number of columns in the dataset are: 5
```

Step 7. Print the name of all the columns.

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

Out[12]:
```
Index(['order_id', 'quantity', 'item_name', 'choice_description',
       'item_price'],
      dtype='object')
```

Step 8. How is the dataset indexed?

In [13]:
```
chipo.index
```

Out[13]:
```
RangeIndex(start=0, stop=4622, step=1)
```

Step 9. Number of Unique Items ?

In [15]:
```
chipo['item_name'].nunique()
```

Out[15]: 50

Step 10. Which was the most-ordered item?

In [18]:
```
chipo.groupby('item_name').sum().sort_values(['quantity'],
ascending=False).he
```

Out[18]:

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

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

Out[19]:

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

Step 11. How many items were orderd in total?

In [20]:
```
chipo['quantity'].sum()
```

Out[20]: 4972

Step 12. Turn the item price into a float

Step 12.a. Check the item price type

In [30]:
```
chipo['item_price'].replace
```

Out[30]:
```
<bound method NDFrame.replace of 0         2.39
1         3.39
2         3.39
3         2.39
4         16.98 ...
4617      11.75
4618      11.75
4619      11.25
4620      8.75
4621      8.75
Name: item_price, Length: 4622, dtype: float64>
```

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

In [27]:
```
dollarizer = lambda x: float(x[1:-1])
chipo.item_price = chipo.item_price.apply(dollarizer)
```

Step 12.c. Check the item price type

In [29]:
```
chipo['item_price'].dtype
```

Out[29]:
```
dtype('float64')
```

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

In [36]:
```
total_revenue = (chipo['quantity']*chipo['item_price']).sum()
total_revenue
print(f' The revenue for the period in the dataset is: {total_revenue}')
```

```
The revenue for the period in the dataset is: 39237.02
```

Step 15. How many orders were made ?

In [41]:
```
chipo['order_id'].value_counts().count()
```

Out[41]:
```
1834
```

Step 17. How many different choice descriptions are there?

In [42]:
```
chipo['choice_description'].nunique()
```

Out[42]:
```
1043
```

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

In [44]:
```
items = chipo.groupby('item_name')['quantity'].sum()
items[items>100]
```

Out[44]:
```
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 [49]:
```
# Solution
chipo['revenue'] = chipo['quantity']*chipo['item_price']
chipo.groupby('order_id')['revenue'].sum().mean()
```

Out[49]: 21.39423118865867

In [52]:
```
# Solution 2
chipo['revenue'] = chipo['quantity'] * chipo['item_price']
order_grp = chipo.groupby(by=['order_id']).sum()
order_grp['revenue'].mean()
```

Out[52]: 21.39423118865867

In [ ]:

Data Mining

2301010148

Lab - 5

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

In [2]:
```
import pandas as pd
import numpy as np
```

In [3]:
```
data = pd.read_csv('titanic.csv')
data
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0	1	0	3Mr. OwenBraund,Harris		male	22.0	1	0	
211A									

STO O	**1**	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	17
	2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	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
W. 66	**888**	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	
	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 [4]:
```
data.tail(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticke
		0			male				

Out[4]:

886	887		2	Montvila, Rev. Juozas		27.0	0	0	211536
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C6607

```
data.head(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376

In [27]: Out[27]:

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

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

In [6]:

```
# Missing values in dataset using dropna()
#parameter for dropna() : dropna(how='any') => delete values(row) having any
n
                          # dropna(how='any', axis=1) => delete values(column)
                          h
                          # dropna(how='all') => delete values having all null
                          v
                          # dropna() == dropna(how='all')
                          # By default how='all'
                          # By default axis=0



data_drop = data.dropna() data_drop =
data.dropna(how='any') data_drop =
data.dropna(how='any', axis=1) #
data_drop = data.dropna(how='all')
data_drop
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	SibSp	Parch	Ticket	
0	1	0	3	Mr. OwenBraund,Harris	male	1	0	21171A/5	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	1	0	PC 17599	7
2	3	1	3	Heikkinen, Miss. Laina	female	0	0	STON/O2. 3101282	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	1	0	113803	5
4	5	0	3	Allen, Mr. William Henry	male	0	0	373450	
...	...	...	...	...	...	...	...	...	
886	887	0	2	Montvila, Rev. Juozas	male	0	0	211536	1
887	888	1	1	Graham, Miss. Margaret Edith	female	0	0	112053	3
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	1	2	W./C. 6607	2
889	890	1	1	Behr, Mr. Karl Howell	male	0	0	111369	3
890	891	0	3	Dooley, Mr. Patrick	male	0	0	370376	

891 rows × 9 columns

In [22]:

```
# Missing values in dataset using fillna()
# It will fill the null value with the given value..

data_fillna = data.fillna({'Age':35, 'Cabin':'Not Available'})


age_mean = data['Age'].mean()
# age_mean = data.Age.mean()
data_fillna = data.fillna({'Age':age_mean})

age_median = data['Age'].median()
data_fillna = data.fillna({'Age':age_median})
data_fillna
```

Out[22]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0	1	0	3	Mr. OwenBraund,Harris	male	22.0	1	0	211A
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	17
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STO O 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	28.0	1	2	
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

W.
66

891 rows × 12 columns

In [28]:
```
# Missing values in dataset using interpolate()
# It takes values from its surrounding values
# It is only for integer values
data_interpolate = data.interpolate()


data_interpolate
```

```
C:\Users\DELL\AppData\Local\Temp\ipykernel_1796\2310400087.py:4:
FutureWarning: DataFrame.interpolate with object dtype is deprecated and will
raise in a futur e version. Call obj.infer_objects(copy=False) before
interpolating instead. data_interpolate = data.interpolate()
```

Out[28]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Tick
0	1	0	3	Mr. OwenBraund,Harris	male	22.0	1	0	

211A

		PassengerId	Survived	Pclass	Name	Sex				
STO O	**1**	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	17
	2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	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
	...	...	...	...	...	...	...	...	...	
W. 66	**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	
	889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	1113
	890	891	0	3	Mr. Patrick Dooley,	male	32.0	0	0	3703

891 rows × 12 columns

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

In [42]:
```
# MinMax Scaling
# Formula => v' = [(v - Min)/(Max - Min)  * (NewMax - NewMin)] - NewMin

data.fillna(data.Age.mean(),inplace=True)
# For filling null values with mean
values

data2 = data.copy() min_age =
data.Age.min() max_age =
data.Age.max() # Here NewMax =
1, NewMin = 0..
data['ScalAge'] = (data['Age'] - min_age)/(max_age - min_age)
data2
```

Out[42]:

	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 [48]:

```
# Decimal Scaling
# Max no. => No.of digit(length)
# Divide by 10^length
data3 = data.copy()


max_age = data.Age.max()

noOfDigit =

len(str(int(max_age)))

print(max_age ,' => ', noOfDigit)

data3['AgeDS'] = data3['Age']/ (10**noOfDigit)
data3
```

```
80.0  =>  2
```

Out[48]:

	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 × 14 columns

In [61]:
```
# Z -score
# Formula => Z = (value - Mean)/ std
# It shows how much the value is deviated from mean of data
data4 = data.copy()


mean_age = data.Age.mean()
std_age = data.Age.std()

data4['Z-Score[Age]'] = (data['Age'] - mean_age)/std_age
data4
```

Out[61]:

	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 × 14 columns

In [ ]:

Data Mining

2301010148

Lab - 6

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 symmetricmatrices).
`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 [4]:
```python
import pandas as pd
import numpy as np
```

In [8]:
```python
data = pd.read_csv('iris.csv')
data
```

Out[8]:

	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 [13]:
```
X      = data.drop(columns = 'species')
Y      = data['species'].map({'setosa': 0,'versicolor':1,'virginica':2})
print('Original Shape:',X.shape)
```

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

In [9]:
```
data.shape
```

Out[9]: (150, 5)

Step 2: Standardize the data (zero mean)

In [14]:
```
X_meaned = X - np.mean(X,axis=0)
X_meaned
```

Out[14]:

	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
...	...	...	...	...
145	0.856667	-0.057333	1.442	1.100667
146	0.456667	-0.557333	1.242	0.700667
147	0.656667	-0.057333	1.442	0.800667
148	0.356667	0.342667	1.642	1.100667

149	0.056667	-0.057333	1.342	0.600667

150 rows × 4 columns

Step 3: Compute the Covariance Matrix

In [16]:
```
cov_matrix = np.cov(X_meaned,rowvar = False)
print('Covariance Matrix shape: ',cov_matrix.shape)
print(cov_matrix)
```

```
Covariance Matrix shape:  (4, 4)
[[ 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 [19]:
```
eigen_values,eigen_vectors = np.linalg.eigh(cov_matrix)

print('Eigen Values: ',eigen_values)
print('Eigen Vectors: ',eigen_vectors)
```

```
Eigen Values:  [0.02383509 0.0782095  0.24267075 4.22824171]
Eigen Vectors:  [[ 0.31548719  0.58202985  0.65658877 -0.36138659]
 [-0.3197231  -0.59791083  0.73016143  0.08452251]
 [-0.47983899 -0.07623608 -0.17337266 -0.85667061]
 [ 0.75365743 -0.54583143 -0.07548102 -0.3582892 ]]
```

Step 5: Compute eigenvalues and eigenvectors

In [21]:
```
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalues = eigen_values[sorted_index]
sorted_eigenvectors = eigen_vectors[:, sorted_index]

sorted_index
sorted_eigenvalues
sorted_eigenvectors
```

Out[21]:
```
array([[-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 [22]:
```
k = 2
eigenvector_subset = sorted_eigenvectors[:,0:k]
eigenvector_subset
```

Out[22]:
```
array([[-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 [24]:
```
X_reduced = np.dot(X_meaned, eigenvector_subset)
X_reduced
X_reduced.shape
```

Out[24]:
```
(150, 2)
```

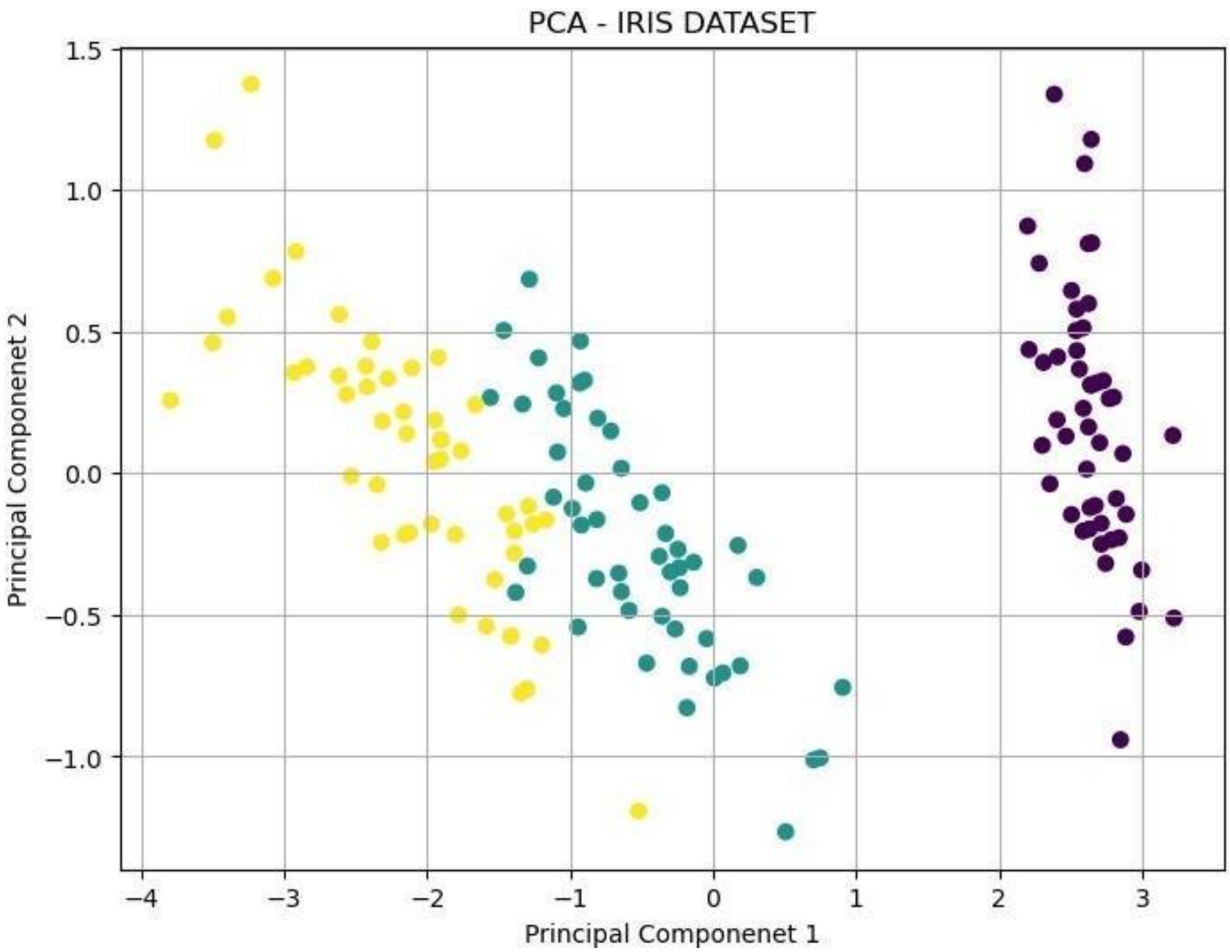
Step 8: Plot the PCA-Reduced Data

In [29]:
```
import matplotlib.pyplot as plt

plt.figure(figsize =(8,6))
plt.scatter(X_reduced[:,0],X_reduced[:,1],c=Y)

plt.xlabel('Principal Componenet 1')
plt.ylabel('Principal Componenet 2')
plt.title('PCA - IRIS DATASET')
plt.grid(True)
plt.show
```

Out[29]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



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) equalfrequency (equal-depth) partitioning (b) equal-width partitioning

In [30]:
```
data2 = [5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204,
215] data2.sort() data2

#Equal frequency
```

```
n = len(data2) k
= 3 #no. of bins
size = n//k

print('Equal frequency bins: ')
for i in range(0,n,size):
    bin_data = data2[i:i+size]
    print(f'Bin (i//size + 1)', bin_data)

#Equal Width min_val =
min(data2) max_val =
max(data2) width = max_val
- min_val print('Equal
Width bins: ')
```

```
Equal frequency bins:
Bin (i//size + 1) [5, 10, 11, 13]
Bin (i//size + 1) [15, 35, 50, 55]
Bin (i//size + 1) [72, 92, 204, 215]
Equal Width bins:
```

In [ ]:

Data Mining

2301010148

Lab - 7(Part-2)

Step 1: Load the Dataset

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

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

data = pd.read_csv('Tdata.csv')
data
```

Out[3]:

	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 [12]:
```
data2 = data.drop(columns = ['Transaction'])
data2
```

Out[12]:

	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 [13]:

```
data2.sum()
```

Out[13]:

```
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 [32]:

```
from itertools import combinations

def find_frequent_itemset(data, min_support):
    n = len(data)
    result = []

    for k in [1,2,3]:
        for items in combinations(data.columns, k):
            mask = data[list(items)].all(axis=1)
            support = mask.sum()/ n
            if support >=
            min_support:
                result.append((frozenset(items),round(support,2)))

    return result data3 =

find_frequent_itemset(data2,0.5)


for itemset,support in data3:
    print(f"{set(itemset)} -> supoort: {support}")
```

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

Step 5: Run Apriori

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

In [33]:
```
data4= find_frequent_itemset(data2,0.6)
data4

for itemset,support in data4:
    print(f"{set(itemset)} -> supoort: {support}")
```

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

Step 6 Display as a DataFrame

In [40]:
```
data5 = pd.DataFrame(data4, columns=['itemset','support'])
data5
```

Out[40]:

	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 Mining

2301010148

Lab - 10

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

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

Create Following Data

In [3]:
```
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcas
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal',
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'We
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes',
})
```

In [5]:
```
data
```

Out[5]:

	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

In [11]:
```
def entropy(y):
    values,counts = np.unique(y,return_counts=True)
    probabilities = counts / counts.sum()
    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

In [13]:
```
y = np.array(['Yes','No','Yes','Yes'])
entropy(y)
```

Out[13]:
```
0.8112781244591328
```

Define function to Calculate Information Gain

In [23]:
```
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)
    print(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

In [25]:
```
data2 = pd.DataFrame({'Weather' : ['Sunny', 'Sunny', 'Rain' , 'Rain'], 'Play':
information_gain(data2,'Weather','Play')
```

```
0.8112781244591328
['Rain' 'Sunny']
[2 2]
  Weather Play
2    Rain  Yes
3    Rain  Yes
0.0
  Weather Play
0   Sunny  Yes
1   Sunny   No
0.5
```

Out[25]: 0.31127812445913283

Implement ID3 Algo

In [33]:

```python
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_feat
        subtree = id3(sub_data, [f for f in features if f != best_feature], ta
        tree[best_feature][value] = subtree

    return tree
```

Use ID3

In [ ]:

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

tree = id3(data, features, target)
```

Print Tree

In [37]:
```
tree
```

Out[37]:
```
{'Outlook': {'Overcast': 'Yes',
  'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
  'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Data Mining

Lab - 14

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]]

In [1]:
```
import math
```

In [9]:
```
data = [
    [1, 2], [2, 3], [3, 4],
    [10, 11], [11, 12], [12, 13],
    [50, 51], [51, 52], [52, 53]
]
```

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

In [3]:
```
distance([1,1],[1,1])
```

Out[3]:
```
0.0
```

In [4]:
```
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)]
```

In [5]:
```
update_cluster_center([[1,1],[2,2],[1,1]])
```

Out[5]: [1.3333333333333333, 1.3333333333333333]

Now Implement code

In [7]:

```
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)
```

In [10]:

```
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]]

In [17]:
```
import random
import math
```

In [12]:
```
def euclidean_distance(p1, p2):
    return math.sqrt(sum((x - y) ** 2 for x, y in zip(p1, p2)))
```

In [13]:
```
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
```

In [14]:
```
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
```

In [15]:
```
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)
```

```python
        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
```

In [18]:

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

In [19]:

```python
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]]
```

In [ ]: