Data Mining

Lab - 1

Keval Dhandhukiya

23010101064

Introduction to Pandas Library Function:

Step-1 Import the pandas Libraries

In [64]:
```
import pandas as pd
```

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

In [65]:
```
df = pd.read_csv('titanic.csv')
```

Step-3 Read csv or excel File

In [66]:
```
df
```

Out[66]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

Step-4 Print Data from csv or excel File

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

Out[67]: (891, 12)

Step-5 See the First 10 Rows

In [68]:
```
df.head(10)
```

Out[68]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

Step-6 See the Last 10 Rows

In [69]:
```
df.tail(10)
```

Out[69]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958	NaN	S
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167	NaN	S
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000	NaN	S
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500	NaN	S
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250	NaN	Q
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

Step-7 Data type of each columns

In [70]:
```
df.dtypes
```

Out[70]:
```
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 [71]:
```
df.describe()
```

Out[71]:

	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 [72]:
```
df['Name']
```

Out[72]:
```
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 [73]:
```
df.Name
```

Out[73]:
```
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 [74]:
```
df[['Name','Age']]
```

Out[74]:

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

891 rows × 2 columns

Step-10 Access rows by their integer location

In [75]:
```
df.iloc[50]
```

Out[75]:
```
PassengerId                      51
Survived                          0
Pclass                            3
Name           Panula, Master. Juha Niilo
Sex                            male
Age                             7.0
SibSp                             4
Parch                             1
Ticket                      3101295
Fare                        39.6875
Cabin                           NaN
Embarked                          S
Name: 50, dtype: object
```

In [76]:
```
df.iloc[400:500]
```

Out[76]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
400	401	1	3	Niskanen, Mr. Juha	male	39.0	0	0	STON/O 2. 3101289	7.9250	NaN	S
401	402	0	3	Adams, Mr. John	male	26.0	0	0	341826	8.0500	NaN	S
402	403	0	3	Jussila, Miss. Mari Aina	female	21.0	1	0	4137	9.8250	NaN	S
403	404	0	3	Hakkarainen, Mr. Pekka Pietari	male	28.0	1	0	STON/O2. 3101279	15.8500	NaN	S
404	405	0	3	Oreskovic, Miss. Marija	female	20.0	0	0	315096	8.6625	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
495	496	0	3	Yousseff, Mr. Gerious	male	NaN	0	0	2627	14.4583	NaN	C
496	497	1	1	Eustis, Miss. Elizabeth Mussey	female	54.0	1	0	36947	78.2667	D20	C
497	498	0	3	Shellard, Mr. Frederick William	male	NaN	0	0	C.A. 6212	15.1000	NaN	S
498	499	0	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0	1	2	113781	151.5500	C22 C26	S
499	500	0	3	Svensson, Mr. Olof	male	24.0	0	0	350035	7.7958	NaN	S

100 rows × 12 columns

Step-11 Delete a specific Column

In [77]:
```
df.drop('Cabin', axis=1)
```

Out[77]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S
...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	Q

891 rows × 11 columns

In [78]:
```
df
```

Out[78]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [80]:

```
df.drop('Cabin',axis=1,inplace=True)
df
```

Out[80]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	isCabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S	False
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	True
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S	False
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	True
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S	False
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	S	False
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	S	True
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	S	False
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C	True
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	Q	False

891 rows × 12 columns

Step-12 Create a new Column

In [79]:

```
df['isCabin'] = ~df['Cabin'].isnull()
df
```

Out[79]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	isCabin
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	False
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	True
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	False
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	True
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S	False
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S	True
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S	False
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C	True
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q	False

891 rows × 13 columns

Step-13 Perform Condition Selection on DataFrame

In [81]:
```
df[df['Fare']>10]
```

Out[81]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked	isCabin
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	True
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S	True
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	S	True
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	S	False
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	S	False
...	...	...	...	...	...	...	...	...	...	...	...	...
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250	Q	False
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	S	False
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	S	True
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	S	False
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C	True

555 rows × 12 columns

Step-14 Compute the sum of value

In [82]:
```
df['Fare'].sum()
```

Out[82]: 28693.9493

Step-15 Compute the mean of value

In [83]:
```
df['Fare'].mean()
```

Out[83]: 32.204207968574636

Step-16 Count non-null value (column)

In [84]:
```
(~df.isnull()).sum()
```

Out[84]:
```
PassengerId    891
Survived       891
Pclass         891
Name           891
Sex            891
Age            714
SibSp          891
Parch          891
Ticket         891
Fare           891
Embarked       889
isCabin        891
dtype: int64
```

In [85]:
```
df.count()
```

Out[85]:
```
PassengerId    891
Survived       891
Pclass         891
Name           891
Sex            891
Age            714
SibSp          891
Parch          891
Ticket         891
Fare           891
Embarked       889
isCabin        891
dtype: int64
```

Step-17 Find Minimun or Maximum values

In [86]:
```
print(f'{df['Fare'].min()} \n{df['Fare'].max()}')
```

```
0.0
512.3292
```

Data Mining

Lab - 2

Keval Dhandhukiya

23010101064

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

Step 2. Create a 1D array of numbers

In [2]:
```
arr = np.arange(10)
arr
```

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

In [3]:
```
arr = np.arange(15,40)
arr
```

Out[3]:
```
array([15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
       32, 33, 34, 35, 36, 37, 38, 39])
```

In [12]:
```
arr = np.array([12,14,16,190])
arr
```

Out[12]:
```
array([ 12,  14,  16, 190])
```

Step 3. Reshape 1D to 2D Array

In [13]:
```
arr31 = np.arange(20).reshape(4,5)
arr31
```

Out[13]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

In [14]:
```
arr32 = np.arange(20).reshape(5,4)
arr32
```

Out[14]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])
```

Step 4. Create a Linspace array

In [15]:
```
np.linspace(14,15)
```

Out[15]:
```
array([14.        , 14.02040816, 14.04081633, 14.06122449, 14.08163265,
       14.10204082, 14.12244898, 14.14285714, 14.16326531, 14.18367347,
       14.20408163, 14.2244898 , 14.24489796, 14.26530612, 14.28571429,
       14.30612245, 14.32653061, 14.34693878, 14.36734694, 14.3877551 ,
       14.40816327, 14.42857143, 14.44897959, 14.46938776, 14.48979592,
       14.51020408, 14.53061224, 14.55102041, 14.57142857, 14.59183673,
       14.6122449 , 14.63265306, 14.65306122, 14.67346939, 14.69387755,
       14.71428571, 14.73469388, 14.75510204, 14.7755102 , 14.79591837,
       14.81632653, 14.83673469, 14.85714286, 14.87755102, 14.89795918,
       14.91836735, 14.93877551, 14.95918367, 14.97959184, 15.        ])
```

In [16]:
```
np.linspace(14,15,10)
```

Out[16]:
```
array([14.        , 14.11111111, 14.22222222, 14.33333333, 14.44444444,
       14.55555556, 14.66666667, 14.77777778, 14.88888889, 15.        ])
```

Step 5. Create a Random Numbered Array

In [17]:
```
arr51 = np.random.rand(3)
print(arr51)
```

```
[0.9121003  0.60765183 0.40175187]
```

In [18]:
```
arr52 = np.random.rand(40)
arr52
```

Out[18]:
```
array([0.05369228, 0.49586253, 0.5844925 , 0.44328472, 0.84132423,
       0.42677907, 0.02401034, 0.00909088, 0.23066913, 0.88698235,
       0.6591622 , 0.02837572, 0.57704904, 0.88537378, 0.42894068,
       0.14289023, 0.84756936, 0.29335911, 0.51605974, 0.92647626,
       0.74469277, 0.52254297, 0.64769799, 0.52844649, 0.73257727,
       0.49913568, 0.7418767 , 0.8507475 , 0.84131819, 0.27376769,
       0.12139945, 0.76758101, 0.1694517 , 0.24880172, 0.8986889 ,
       0.97162934, 0.29123327, 0.25627122, 0.92482952, 0.81553773])
```

Step 6. Create a Random Integer Array

In [19]:
```
np.random.randint(10)
```

Out[19]: 7

In [20]:
```
np.random.randint(10,20)
```

Out[20]: 11

In [21]:
```
np.random.randint(10,20,size=11)
```

Out[21]:
```
array([14, 17, 16, 14, 14, 14, 17, 19, 14, 15, 16])
```

In [22]:
```
np.random.randint(10,20,(5,5))
```

Out[22]:
```
array([[13, 15, 10, 12, 19],
       [14, 14, 15, 14, 13],
       [10, 16, 18, 15, 12],
       [12, 13, 19, 16, 18],
       [13, 15, 19, 16, 11]])
```

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

In [23]:
```
arr7 = np.random.randint(11,20,10)
arr7
```

Out[23]:
```
array([18, 15, 14, 16, 14, 11, 15, 19, 11, 18])
```

In [24]:
```
np.max(np.random.randint(11,20,10))
```

Out[24]: 19

In [25]:
```
arr7.min()
```

Out[25]: 11

In [26]:
```
arr7.argmax()
```

Out[26]: 7

In [27]:
```
arr7.argmin()
```

Out[27]: 5

Step 8. Indexing in 1D Array

In [28]:
```
arr8 = np.random.randint(10,35,9)
arr8
```

Out[28]:
```
array([11, 25, 27, 22, 14, 17, 34, 18, 23])
```

In [29]:
```
arr8[3]
```

Out[29]:
```
22
```

In [30]:
```
arr8[3:4]
```

Out[30]:
```
array([22])
```

In [31]:
```
arr8[3:]
```

Out[31]:
```
array([22, 14, 17, 34, 18, 23])
```

Step 9. Indexing in 2D Array

In [32]:
```
arr9 = np.random.randint(10,20,(5,5))
arr9
```

Out[32]:
```
array([[14, 17, 12, 14, 14],
       [16, 14, 10, 19, 10],
       [12, 19, 11, 16, 10],
       [14, 17, 17, 18, 18],
       [10, 11, 17, 10, 12]])
```

In [33]:
```
arr9[2,2]
```

Out[33]:
```
11
```

In [34]:
```
arr9[2]
```

Out[34]:
```
array([12, 19, 11, 16, 10])
```

Step 10. Conditional Selection

In [35]:
```
arr10 = np.random.randint(10,19,5)
arr10
```

Out[35]:
```
array([10, 10, 17, 10, 13])
```

In [36]:
```
arr10>=15
```

Out[36]:
```
array([False, False,  True, False, False])
```

In [37]:
```
arr10[(arr10>1)&(arr10<15)]
```

Out[37]:
```
array([10, 10, 10, 13])
```

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

Pandas

Step 1. Import the necessary libraries

In [38]:
```
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 [39]:
```
users = pd.read_csv("https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.user",sep="|",index_col="user_id")
users
```

Out[39]:

	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

In [40]:
```
users.head(25)
```

Out[40]:

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

Step 5. See the last 10 entries

In [41]:
```
users.tail(10)
```

Out[41]:

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

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

In [42]:
```
users.shape[0]
```

Out[42]:
```
943
```

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

In [43]:
```
users.shape[1]
```

Out[43]:
```
4
```

Step 8. Print the name of all the columns.

In [44]:
```
users.columns
```

Out[44]:
```
Index(['age', 'gender', 'occupation', 'zip_code'], dtype='object')
```

Step 9. How is the dataset indexed?

In [45]:
```
users.index
```

Out[45]:
```
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 [46]:
```
users.dtypes
```

Out[46]:
```
age            int64
gender        object
occupation    object
zip_code      object
dtype: object
```

Step 11. Print only the occupation column

In [47]:
```
users["occupation"]
```

Out[47]:
```
user_id
1         technician
2              other
3             writer
4         technician
5              other
           ...
939          student
940    administrator
941          student
942        librarian
943          student
Name: occupation, Length: 943, dtype: object
```

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

In [48]:
```
users["occupation"].nunique()
```

Out[48]: 21

Step 13. What is the most frequent occupation?

In [49]:
```
users["occupation"].value_counts().head(1)
```

Out[49]:
```
occupation
student    196
Name: count, dtype: int64
```

Step 14. Summarize the DataFrame.

In [55]:
```
users.describe()
```

Out[55]:

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

Step 15. Summarize all the columns

In [56]:
```
users.describe(include="all")
```

Out[56]:

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

Step 16. Summarize only the occupation column

In [57]:
```
users['occupation'].describe()
```

Out[57]:
```
count         943
unique         21
top       student
freq          196
Name: occupation, dtype: object
```

Step 17. What is the mean age of users?

In [58]:
```
users["age"].mean()
```

Out[58]: 34.05196182396607

Step 18. What is the age with least occurrence?

In [60]:
```
users["age"].value_counts().idxmin()
```

Out[60]: 7

Data Mining

Lab - 3

Keval Dhandhukiya

23010101064

---

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

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

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

In [4]:
```
qw.head(5)
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

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

In [11]:
```
nominal = ['Name', 'Sex', 'cabin', 'Ticket', 'Embarked']
ordinal = ['pclass']
binary = ['Survived', 'SibSp']
numeric = ['Age', 'Fare', 'SibSp', 'Parch']

print("Nominal", nominal)
print("Ordinal", ordinal)
print("Binary", binary)
print("Numeric", numeric)
```

```
Nominal ['Name', 'Sex', 'cabin', 'Ticket', 'Embarked']
Ordinal ['pclass']
Binary ['Survived', 'SibSp']
Numeric ['Age', 'Fare', 'SibSp', 'Parch']
```

In [12]:
```
qw["PassengerId"].nunique()
```

Out[12]: 891

In [13]:
```
qw["Sex"].unique()
```

Out[13]:
```
array(['male', 'female'], dtype=object)
```

In [14]:
```
qw["Fare"].unique()
```

Out[14]:
```
array([  7.25  ,  71.2833,   7.925 ,  53.1   ,   8.05  ,   8.4583,
        51.8625,  21.075 ,  11.1333,  30.0708,  16.7   ,  26.55  ,
        31.275 ,   7.8542,  16.    ,  29.125 ,  13.    ,  18.    ,
         7.225 ,  26.    ,   8.0292,  35.5   ,  31.3875, 263.    ,
         7.8792,   7.8958,  27.7208, 146.5208,   7.75  ,  10.5   ,
        82.1708,  52.    ,   7.2292,  11.2417,   9.475 ,  21.    ,
        41.5792,  15.5   ,  21.6792,  17.8   ,  39.6875,   7.8   ,
        76.7292,  61.9792,  27.75  ,  46.9   ,  80.    ,  83.475 ,
        27.9   ,  15.2458,   8.1583,   8.6625,  73.5   ,  14.4542,
        56.4958,   7.65  ,  29.    ,  12.475 ,   9.    ,   9.5   ,
         7.7875,  47.1   ,  15.85  ,  34.375 ,  61.175 ,  20.575 ,
        34.6542,  63.3583,  23.    ,  77.2875,   8.6542,   7.775 ,
        24.15  ,   9.825 ,  14.4583, 247.5208,   7.1417,  22.3583,
         6.975 ,   7.05  ,  14.5   ,  15.0458,  26.2833,   9.2167,
        79.2   ,   6.75  ,  11.5   ,  36.75  ,   7.7958,  12.525 ,
        66.6   ,   7.3125,  61.3792,   7.7333,  69.55  ,  16.1   ,
        15.75  ,  20.525 ,  55.    ,  25.925 ,  33.5   ,  30.6958,
        25.4667,  28.7125,   0.    ,  15.05  ,  39.    ,  22.025 ,
        50.    ,   8.4042,   6.4958,  10.4625,  18.7875,  31.    ,
       113.275 ,  27.    ,  76.2917,  90.    ,   9.35  ,  13.5   ,
         7.55  ,  26.25  ,  12.275 ,   7.125 ,  52.5542,  20.2125,
        86.5   , 512.3292,  79.65  , 153.4625, 135.6333,  19.5   ,
        29.7   ,  77.9583,  20.25  ,  78.85  ,  91.0792,  12.875 ,
         8.85  , 151.55  ,  30.5   ,  23.25  ,  12.35  , 110.8833,
       108.9   ,  24.    ,  56.9292,  83.1583, 262.375 ,  14.    ,
       164.8667, 134.5   ,   6.2375,  57.9792,  28.5   , 133.65  ,
        15.9   ,   9.225 ,  35.    ,  75.25  ,  69.3   ,  55.4417,
       211.5   ,   4.0125, 227.525 ,  15.7417,   7.7292,  12.    ,
       120.    ,  12.65  ,  18.75  ,   6.8583,  32.5   ,   7.875 ,
        14.4   ,  55.9   ,   8.1125,  81.8583,  19.2583,  19.9667,
        89.1042,  38.5   ,   7.725 ,  13.7917,   9.8375,   7.0458,
         7.5208,  12.2875,   9.5875,  49.5042,  78.2667,  15.1   ,
         7.6292,  22.525 ,  26.2875,  59.4   ,   7.4958,  34.0208,
        93.5   , 221.7792, 106.425 ,  49.5   ,  71.    ,  13.8625,
         7.8292,  39.6   ,  17.4   ,  51.4792,  26.3875,  30.    ,
        40.125 ,   8.7125,  15.    ,  33.    ,  42.4   ,  15.55  ,
        65.    ,  32.3208,   7.0542,   8.4333,  25.5875,   9.8417,
         8.1375,  10.1708, 211.3375,  57.    ,  13.4167,   7.7417,
         9.4833,   7.7375,   8.3625,  23.45  ,  25.9292,   8.6833,
         8.5167,   7.8875,  37.0042,   6.45  ,   6.95  ,   8.3   ,
         6.4375,  39.4   ,  14.1083,  13.8583,  50.4958,   5.    ,
         9.8458,  10.5167])
```

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

In [17]:
```
print("Survived values(asymmetric binary) : ")
print(qw['Survived'].value_counts())

print("Survived values(symmetric binary) : ")
print(qw['Sex'].value_counts())
```

```
Survived values(asymmetric binary) :
Survived
0    549
1    342
Name: count, dtype: int64
Survived values(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 [23]:
```
x = ['PassengerId','Survived','Pclass','Age','Parch']

for col in x:
    print(f"{col} : ")
    print(f"\tAverage = {qw[col].mean()}")
    print(f"\tStandard deviation = {qw[col].std()}")
    print(f"\tMinimum = {qw[col].min()}")
    print(f"\tMaximum = {qw[col].max()}")
    print(f"\tRange  = {qw[col].max() - qw[col].min()}")
    print(f"\tMode = {qw[col].mode()[0]}")
```

```
PassengerId :
        Average = 446.0
        Standard deviation = 257.3538420152301
        Minimum = 1
        Maximum = 891
        Range  = 890
        Mode = 1
Survived :
        Average = 0.3838383838383838
        Standard deviation = 0.4865924542648585
        Minimum = 0
        Maximum = 1
        Range  = 1
        Mode = 0
Pclass :
        Average = 2.308641975308642
        Standard deviation = 0.8360712409770513
        Minimum = 1
        Maximum = 3
        Range  = 2
        Mode = 3
Age :
        Average = 29.69911764705882
        Standard deviation = 14.526497332334044
        Minimum = 0.42
        Maximum = 80.0
        Range  = 79.58
        Mode = 24.0
Parch :
        Average = 0.38159371492704824
        Standard deviation = 0.8060572211299559
        Minimum = 0
        Maximum = 6
        Range  = 6
        Mode = 0
```

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

In [24]:
```
qw['Pclass'].value_counts()
```

Out[24]:
```
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 [27]:
```
# qw.describe()
# qw.describe(include='object')
# qw['Fare'].describe()

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

Out[27]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.000000	891	891.000000	204	889
unique	NaN	NaN	NaN	891	2	NaN	NaN	NaN	681	NaN	147	3
top	NaN	NaN	NaN	Braund, Mr. Owen Harris	male	NaN	NaN	NaN	347082	NaN	B96 B98	S
freq	NaN	NaN	NaN	1	577	NaN	NaN	NaN	7	NaN	4	644
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.381594	NaN	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.806057	NaN	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.000000	NaN	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.000000	NaN	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.000000	NaN	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.000000	NaN	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.000000	NaN	512.329200	NaN	NaN

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

In [29]:
```
qw.cov(numeric_only=True)
```

Out[29]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	66231.000000	-0.626966	-7.561798	138.696504	-16.325843	-0.342697	161.883369
Survived	-0.626966	0.236772	-0.137703	-0.551296	-0.018954	0.032017	6.221787
Pclass	-7.561798	-0.137703	0.699015	-4.496004	0.076599	0.012429	-22.830196
Age	138.696504	-0.551296	-4.496004	211.019125	-4.163334	-2.344191	73.849030
SibSp	-16.325843	-0.018954	0.076599	-4.163334	1.216043	0.368739	8.748734
Parch	-0.342697	0.032017	0.012429	-2.344191	0.368739	0.649728	8.661052
Fare	161.883369	6.221787	-22.830196	73.849030	8.748734	8.661052	2469.436846

In [31]:
```
qw.corr(numeric_only=True)
```

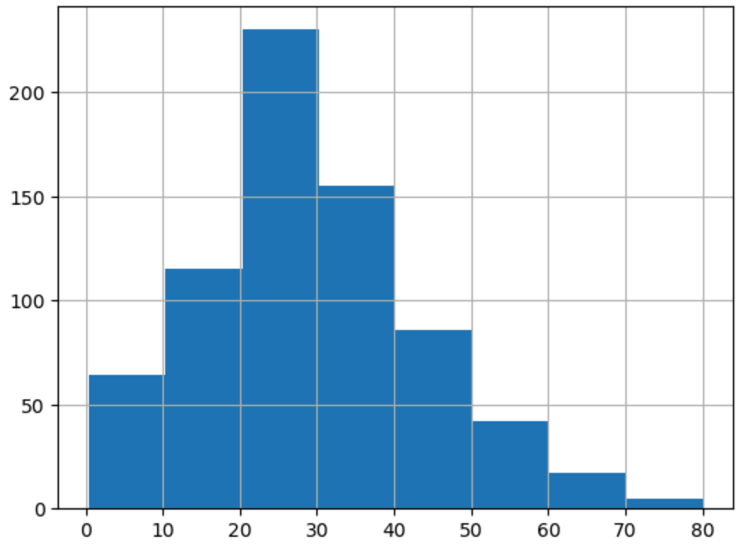
Out[31]:

	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 [33]:
```
qw['Age'].hist(bins=8)
```

Out[33]: <Axes: >

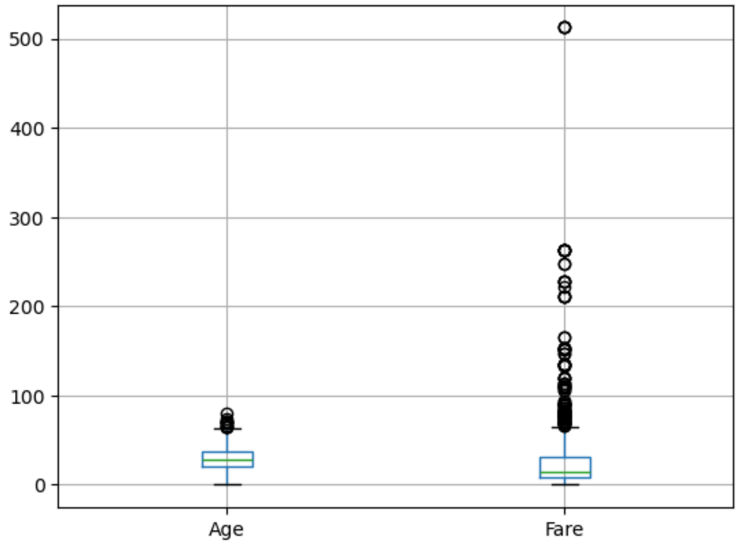


10) A boxplot can also be used to show the distribution of values for each attribute.

In [18]:
```
import matplotlib.pyplot as plt
```

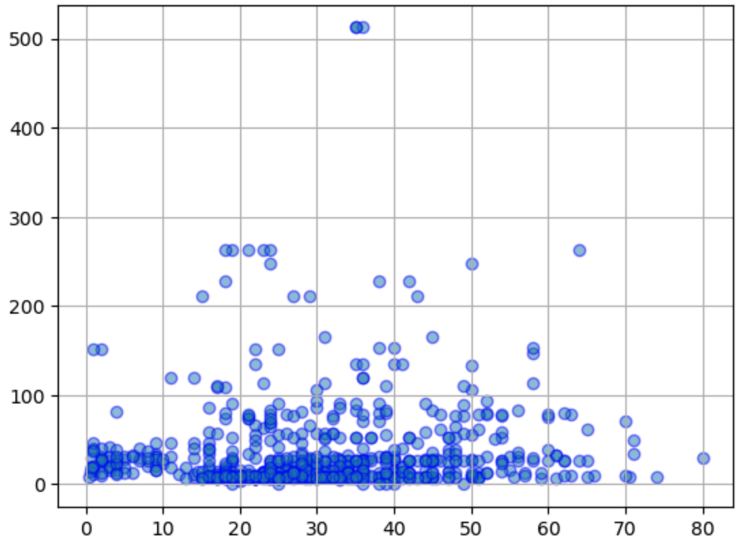
In [19]:
```
qw[['Age','Fare']].boxplot()
```

Out[19]: <Axes: >



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

In [29]:
```
plt.scatter(x=qw['Age'],y=qw['Fare'],alpha=0.5, edgecolor='b')
plt.grid(True)
plt.show()
```



Data Mining

Lab - 4

Keval Dhandhukiya

23010101064

Step 1. Import the necessary libraries

In [1]:
```
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]:
```
url = "https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv"
```

In [3]:
```
chipo = pd.read_csv(url, sep='\t')
```

Step 4. See the first 10 entries

In [4]:
```
chipo.head(10)
```

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
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 [5]:
```
# Solution 1
chipo.shape
```

Out[5]: (4622, 5)

In [6]:
```
# Solution 2
chipo.info
```

Out[6]:
```
<bound method DataFrame.info of       order_id  quantity                              item_name  \
0            1         1           Chips and Fresh Tomato Salsa
1            1         1                                   Izze
2            1         1                       Nantucket Nectar
3            1         1  Chips and Tomatillo-Green Chili Salsa
4            2         2                           Chicken Bowl
...        ...       ...                                    ...
4617      1833         1                          Steak Burrito
4618      1833         1                          Steak Burrito
4619      1834         1                     Chicken Salad Bowl
4620      1834         1                     Chicken Salad Bowl
4621      1834         1                     Chicken Salad Bowl

                                     choice_description item_price
0                                                   NaN     $2.39
1                                          [Clementine]     $3.39
2                                               [Apple]     $3.39
3                                                   NaN     $2.39
4     [Tomatillo-Red Chili Salsa (Hot), [Black Beans...    $16.98
...                                                 ...        ...
4617  [Fresh Tomato Salsa, [Rice, Black Beans, Sour ...    $11.75
4618  [Fresh Tomato Salsa, [Rice, Sour Cream, Cheese...    $11.75
4619  [Fresh Tomato Salsa, [Fajita Vegetables, Pinto...    $11.25
4620  [Fresh Tomato Salsa, [Fajita Vegetables, Lettu...     $8.75
4621  [Fresh Tomato Salsa, [Fajita Vegetables, Pinto...     $8.75

[4622 rows x 5 columns]>
```

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

In [7]:
```python
len(chipo.columns)
```

Out[7]:
```
5
```

Step 7. Print the name of all the columns.

In [8]:
```python
chipo.columns
```

Out[8]:
```
Index(['order_id', 'quantity', 'item_name', 'choice_description',
       'item_price'],
      dtype='object')
```

Step 8. How is the dataset indexed?

In [9]:
```python
chipo.index
```

Out[9]:
```
RangeIndex(start=0, stop=4622, step=1)
```

Step 9. Number of Unique Items ?

In [10]:
```python
chipo["item_name"].nunique()
```

Out[10]:
```
50
```

Step 10. Which was the most-ordered item?

In [11]:
```python
chipo.groupby("item_name")['quantity'].sum().sort_values(ascending=False).head(1)
```

Out[11]:
```
item_name
Chicken Bowl    761
Name: quantity, dtype: int64
```

Step 11. How many items were orderd in total?

In [12]:
```python
chipo['quantity'].sum()
```

Out[12]:
```
4972
```

Step 12. Turn the item price into a float

Step 12.a. Check the item price type

In [13]:
```python
 chipo['item_name'].dtypes
```

Out[13]:
```
dtype('O')
```

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

In [14]:
```
chipo["item_price"]=chipo["item_price"].apply(lambda x : float(x[1::]))
```

Step 12.c. Check the item price type

In [15]:
```
chipo["item_price"].dtypes
```

Out[15]:
```
dtype('float64')
```

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

In [20]:
```
chipo['revenue'] = chipo['quantity']* chipo['item_price']
chipo['revenue'].sum()
```

Out[20]:
```
39237.02
```

Step 15. How many orders were made ?

In [23]:
```
chipo['order_id'].nunique()
```

Out[23]:
```
1834
```

Step 17. How many different choice descriptions are there?

In [24]:
```
chipo['choice_description'].nunique()
```

Out[24]:
```
1043
```

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

In [16]:
```
arr = []
arr = chipo.groupby("item_name")['quantity'].sum()
for i in arr:
    if(i>100):
        print(chipo["item_name"][i])
```

```
Chicken Bowl
Canned Soda
Chicken Burrito
Veggie Burrito
Chicken Salad Bowl
Chicken Bowl
Chicken Burrito
Chips and Guacamole
Barbacoa Burrito
Carnitas Bowl
Chicken Bowl
Chips
Side of Chips
```

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

In [19]:
```
# Solution 1
chipo['Total_Amount'] = chipo['quantity']* chipo['item_price']
chipo.groupby('order_id')['Total_Amount'].sum().mean()
```

Out[19]:
```
21.39423118865867
```

Data Mining

Lab - 5

Keval Dhandhukiya

23010101064

Data Preprocessing

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

In [8]:
```
import pandas as pd
```

In [10]:
```
data = pd.read_csv("titanic.csv")
```

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

In [16]:
```
# with dropna
data_withdropna = data.copy()
#data_withdropna = data_withdropna.dropna()
data_withdropna = data_withdropna.dropna(how="any",axis=1)
print(data_withdropna)
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex  SibSp  Parch  \
0                              Braund, Mr. Owen Harris    male      1      0
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female      1      0
2                               Heikkinen, Miss. Laina  female      0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female      1      0
4                             Allen, Mr. William Henry    male      0      0
..                                                 ...     ...    ...    ...
886                              Montvila, Rev. Juozas    male      0      0
887                       Graham, Miss. Margaret Edith  female      0      0
888           Johnston, Miss. Catherine Helen "Carrie"  female      1      2
889                              Behr, Mr. Karl Howell    male      0      0
890                                Dooley, Mr. Patrick    male      0      0

               Ticket     Fare
0           A/5 21171   7.2500
1            PC 17599  71.2833
2    STON/O2. 3101282   7.9250
3              113803  53.1000
4              373450   8.0500
..                ...      ...
886            211536  13.0000
887            112053  30.0000
888        W./C. 6607  23.4500
889            111369  30.0000
890            370376   7.7500

[891 rows x 9 columns]
```

In [14]:
```
# with fillna
data_withfillna = data.copy()
#data_withfillna = data_withfillna.fillna(1000)
meanAge = data_withfillna.Age.mean()
print(meanAge)
data_withfillna = data_withfillna.fillna({'Age':meanAge, 'Cabin':'Not Available'})
data_withfillna
```

```
29.69911764705882
```

Out[14]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.000000	1	0	A/5 21171	7.2500	Not Available	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.000000	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.000000	0	0	STON/O2. 3101282	7.9250	Not Available	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.000000	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.000000	0	0	373450	8.0500	Not Available	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.000000	0	0	211536	13.0000	Not Available	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.000000	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	29.699118	1	2	W./C. 6607	23.4500	Not Available	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.000000	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.000000	0	0	370376	7.7500	Not Available	Q

891 rows × 12 columns

In [18]:

```
# with interpolate
data_interpolate = data.copy()
data_interpolate = data_interpolate.interpolate()
data_interpolate
```

```
C:\Users\Keval\AppData\Local\Temp\ipykernel_23004\2911420591.py:3: FutureWarning: DataFrame.interpolate with object dtype is de
precated and will raise in a future version. Call obj.infer_objects(copy=False) before interpolating instead.
  data_interpolate = data_interpolate.interpolate()
```

Out[18]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	22.5	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

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

In [6]:

```
# with min-max
data_minmax = data.copy()
min_age = data_minmax.Age.min()
max_age = data_minmax.Age.max()

data_minmax['Age'] = (data_minmax['Age'] - min_age) / (max_age - min_age)
data_minmax
```

Out[6]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	0.271174	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	0.472229	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	0.321438	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	0.434531	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	0.434531	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	0.334004	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	0.233476	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	0.321438	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	0.396833	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

In [7]:

```
# with Decimal Scaling
data_decimal = data.copy()
maxAge = data_decimal.Age.max()

Ages = data_decimal['Age']
d = len(str(int(maxAge)))

print(maxAge,'\n',d)

data_decimal['DecimalScalingAge'] = Ages / (10 ** d)
data_decimal
```

```
80.0
 2
```

Out[7]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	DecimalScalingAge
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0.22
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	0.38
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	0.26
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	0.35
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	0.35
...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S	0.27
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S	0.19
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S	NaN
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C	0.26
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q	0.32

891 rows × 13 columns

In [8]:

```
# with Z-score Normalization
data_zscore = data.copy()
mean_age = data_zscore.Age.mean()
std_age = data_zscore.Age.std()

data_zscore['Age'] = (data_zscore['Age'] - mean_age) / std_age
data_zscore
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	-0.530005	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	0.571430	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	-0.254646	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	0.364911	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	0.364911	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	-0.185807	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	-0.736524	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	-0.254646	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	0.158392	0	0	370376	7.7500	NaN	Q

891 rows × 12 columns

Data Mining

Lab - 6

Keval Dhandhukiya

23010101064

🔍 What is Data Reduction?

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

Why do we reduce data?

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

Common data reduction techniques include:

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

📉 What is Principal Component Analysis (PCA)?

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

Key Concepts:

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

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

🧠 NumPy Functions Summary for PCA

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

Step 1: Load the Iris Dataset

In [13]:
```
# Dimensionality Reduction using NumPy
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [14]:
```
iris = pd.read_csv("iris.csv")
iris
```

Out[14]:

	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 [15]:
```
x = iris.drop(columns="species")
y = iris['species'].map({
    'setosa' : 0,
    'versicolor' : 1,
    'virginica' : 2
})
```

In [16]:
```
x.shape
```

Out[16]: (150, 4)

Step 2: Standardize the data (zero mean)

In [30]:
```
x_mean = x - np.mean(x, axis=0)
print("data after centering ()first 5 rows:",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 [18]:
```
cov_mat = np.cov(x_meaned, rowvar=False)
print("Covariance Matrix Shape", cov_mat)
```

```
Covariance Matrix Shape [[ 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 [26]:
```
eigen_value, eigen_vector = np.linalg.eigh(cov_mat)

print("eigen value:\n", eigen_value)
print("eigen vector:\n", eigen_vector[:, :2])
```

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

Step 5: Compute eigenvalues and eigenvectors

In [27]:
```
# sort eigen value
sorted_index = np.argsort(eigen_value)[::-1]
sorted_eigenvalue = eigen_value[sorted_index]
sorted_eigenvector = eigen_vector[:, sorted_index]

print(sorted_index)
```

```
print(sorted_eigenvalue)
print(sorted_eigenvector)
```

```
[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 [28]:
```
k=2
eigenvector_subset = sorted_eigenvector[:, 0:k]
print(eigenvector_subset)
```

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

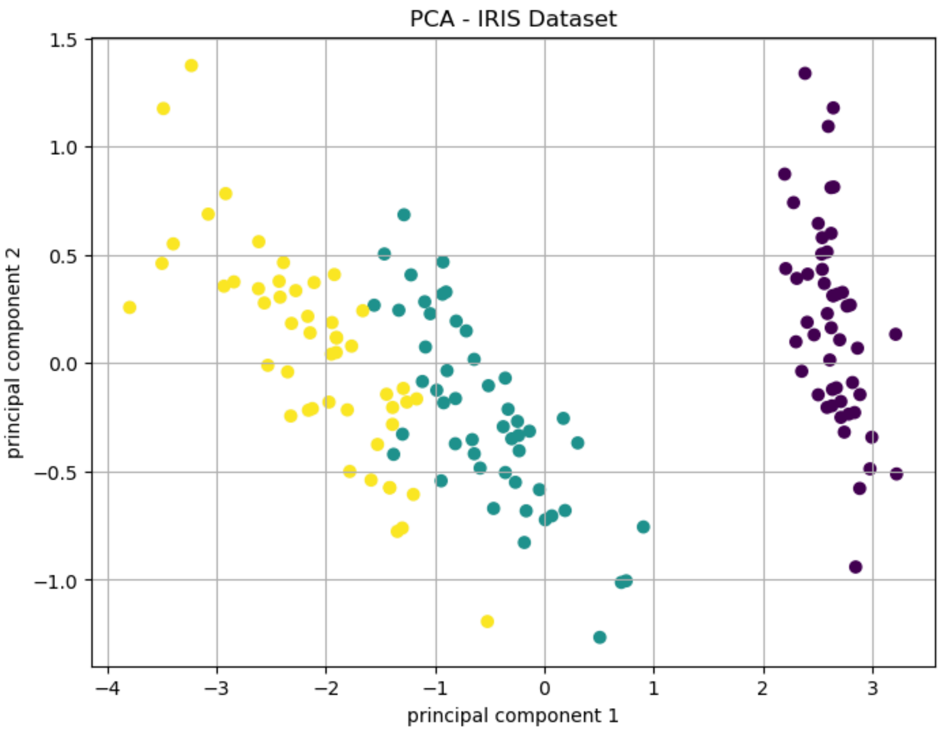
Step 7: Project the data onto the top k eigenvectors

In [31]:
```
x_reduced = np.dot(x_mean, eigenvector_subset)
print("reduced data shape:" , x_reduced.shape)
```

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

Step 8: Plot the PCA-Reduced Data

In [32]:
```
plt.figure(figsize = (8,6))
plt.scatter(x_reduced[:,0], x_reduced[:,1], c=y) # c=y species vali column ne color
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 [34]:
```
data = [5,10,11,13,15,35,50,55,72,92,204,215]
df = pd.DataFrame({'data' : data})
```

```
df['equal_frequency_bins'] = pd.qcut(df['data'], q=3)
# print(df)
df['equal_width_bins'] = pd.cut(df['data'], bins=3)

a = df['equal_frequency_bins'].value_counts().count()
b = df['equal_width_bins'].value_counts().count()

print(a)
print(b)
```

```
3
3
```

Data Mining

Lab - 7

Keval Dhandhukiya

23010101064

Step 1: Load the Dataset

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

In [2]:
```
import pandas as pd
df = pd.read_csv("Tdata.csv")
df.head(6)
```

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.drop('Transaction',axis=1,inplace=True)
df
```

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
5	0	0	1	1	1	0

Step 3: Count Single Items

See how many transactions include each item.

In [4]:
```
df.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):
```

```python
    n = len(df)
    result = []

    for k in [1, 2, 3]:
        for items in combinations(df.columns, k):
            mask = df[list(items)].all(axis=1)
            print("items", items, "\t mask", mask.sum())
            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]:
```python
find_frequent_itemsets(df,0.6)
```

```
items ('bread',)         mask 5
items ('butter',)        mask 3
items ('coffee',)        mask 2
items ('eggs',)          mask 2
items ('jam',)   mask 2
items ('milk',)          mask 3
items ('bread', 'butter')        mask 3
items ('bread', 'coffee')        mask 1
items ('bread', 'eggs')          mask 1
items ('bread', 'jam')   mask 1
items ('bread', 'milk')          mask 3
items ('butter', 'coffee')       mask 0
items ('butter', 'eggs')         mask 0
items ('butter', 'jam')          mask 1
items ('butter', 'milk')         mask 2
items ('coffee', 'eggs')         mask 1
items ('coffee', 'jam')          mask 1
items ('coffee', 'milk')         mask 0
items ('eggs', 'jam')    mask 1
items ('eggs', 'milk')   mask 1
items ('jam', 'milk')    mask 0
items ('bread', 'butter', 'coffee')      mask 0
items ('bread', 'butter', 'eggs')        mask 0
items ('bread', 'butter', 'jam')         mask 1
items ('bread', 'butter', 'milk')        mask 2
items ('bread', 'coffee', 'eggs')        mask 0
items ('bread', 'coffee', 'jam')         mask 0
items ('bread', 'coffee', 'milk')        mask 0
items ('bread', 'eggs', 'jam')   mask 0
items ('bread', 'eggs', 'milk')          mask 1
items ('bread', 'jam', 'milk')   mask 0
items ('butter', 'coffee', 'eggs')       mask 0
items ('butter', 'coffee', 'jam')        mask 0
items ('butter', 'coffee', 'milk')       mask 0
items ('butter', 'eggs', 'jam')          mask 0
items ('butter', 'eggs', 'milk')         mask 0
items ('butter', 'jam', 'milk')          mask 0
items ('coffee', 'eggs', 'jam')          mask 1
items ('coffee', 'eggs', 'milk')         mask 0
items ('coffee', 'jam', 'milk')          mask 0
items ('eggs', 'jam', 'milk')    mask 0
```

Out[6]:
```
[(frozenset({'bread'}), 0.83)]
```

Step 6 Display as a DataFrame

In [7]:
```python
frequent_itemsets = find_frequent_itemsets(df, min_support=0.6)

pd.DataFrame(frequent_itemsets, columns=['Itemset', 'Support'])
```

```
items ('bread',)        mask 5
items ('butter',)       mask 3
items ('coffee',)       mask 2
items ('eggs',)         mask 2
items ('jam',)   mask 2
items ('milk',)         mask 3
items ('bread', 'butter')       mask 3
items ('bread', 'coffee')       mask 1
items ('bread', 'eggs')         mask 1
items ('bread', 'jam')   mask 1
items ('bread', 'milk')         mask 3
items ('butter', 'coffee')      mask 0
items ('butter', 'eggs')        mask 0
items ('butter', 'jam')         mask 1
items ('butter', 'milk')        mask 2
items ('coffee', 'eggs')        mask 1
items ('coffee', 'jam')         mask 1
items ('coffee', 'milk')        mask 0
items ('eggs', 'jam')    mask 1
items ('eggs', 'milk')   mask 1
items ('jam', 'milk')    mask 0
items ('bread', 'butter', 'coffee')     mask 0
items ('bread', 'butter', 'eggs')       mask 0
items ('bread', 'butter', 'jam')        mask 1
items ('bread', 'butter', 'milk')       mask 2
items ('bread', 'coffee', 'eggs')       mask 0
items ('bread', 'coffee', 'jam')        mask 0
items ('bread', 'coffee', 'milk')       mask 0
items ('bread', 'eggs', 'jam')   mask 0
items ('bread', 'eggs', 'milk')         mask 1
items ('bread', 'jam', 'milk')   mask 0
items ('butter', 'coffee', 'eggs')      mask 0
items ('butter', 'coffee', 'jam')       mask 0
items ('butter', 'coffee', 'milk')      mask 0
items ('butter', 'eggs', 'jam')         mask 0
items ('butter', 'eggs', 'milk')        mask 0
items ('butter', 'jam', 'milk')         mask 0
items ('coffee', 'eggs', 'jam')         mask 1
items ('coffee', 'eggs', 'milk')        mask 0
items ('coffee', 'jam', 'milk')         mask 0
items ('eggs', 'jam', 'milk')    mask 0
```

Out[7]:

	Itemset	Support
0	(bread)	0.83

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

Extra : - > Define Apriori Function without itertools

Darshan UNIVERSITY

योग: कर्मसु कौशलम्

Data Mining

Lab - 10

Keval Dhandhukiya

23010101064

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

Create Following Data

In [4]:
```
data = pd.DataFrame({
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', '
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak',
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
})
```

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 [6]:
```
def entropy(y):
    values, counts = np.unique(y, return_counts=True)
    print(values,"\n",counts)
    pro = counts / counts.sum()
    print(pro)
    return -np.sum(pro * np.log2(pro))
```

Testing of Above Function -

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

Function Call - > entropy(y))

output - 0.8112781244591328

In [7]:
```
entropy(data['PlayTennis'])
```

```
['No' 'Yes']
 [5 9]
[0.35714286 0.64285714]
```

Out[7]: `0.9402859586706311`

Define function to Calculate Information Gain

In [8]:
```
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,"\n",counts)

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

    return total_entropy - weight_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 [9]:
```
information_gain(data, 'Outlook', 'PlayTennis')
```

```
['No' 'Yes']
 [5 9]
[0.35714286 0.64285714]
0.9402859586706311
['Overcast' 'Rain' 'Sunny']
 [4 5 5]
     Outlook Temperature Humidity    Wind PlayTennis
2   Overcast         Hot     High    Weak        Yes
6   Overcast        Cool   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
12  Overcast         Hot   Normal    Weak        Yes
['Yes']
 [4]
[1.]
0.0
   Outlook Temperature Humidity    Wind PlayTennis
3     Rain        Mild     High    Weak        Yes
4     Rain        Cool   Normal    Weak        Yes
5     Rain        Cool   Normal  Strong         No
9     Rain        Mild   Normal    Weak        Yes
13    Rain        Mild     High  Strong         No
['No' 'Yes']
 [2 3]
[0.4 0.6]
0.3467680694480959
   Outlook Temperature Humidity    Wind PlayTennis
0    Sunny         Hot     High    Weak         No
1    Sunny         Hot     High  Strong         No
7    Sunny        Mild     High    Weak         No
8    Sunny        Cool   Normal    Weak        Yes
10   Sunny        Mild   Normal  Strong        Yes
['No' 'Yes']
 [3 2]
[0.6 0.4]
0.6935361388961918
```

Out[9]: 0.246749819774439 33

Implement ID3 Algo

In [10]:
```python
import numpy as np

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

    # If no features left → return majority label
    if len(features) == 0:
        return data[target].mode()[0]

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

    tree = {best_feature: {}}

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

    return tree
```

Use ID3

In [11]:
```python
features = list(data.columns[:-1])
target = 'PlayTennis'

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

```
['No' 'Yes']
 [5 9]
[0.35714286 0.64285714]
0.9402859586706311
['Overcast' 'Rain' 'Sunny']
 [4 5 5]
     Outlook Temperature Humidity    Wind PlayTennis
2   Overcast         Hot     High    Weak        Yes
6   Overcast        Cool   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
12  Overcast         Hot   Normal    Weak        Yes
['Yes']
 [4]
[1.]
0.0
   Outlook Temperature Humidity    Wind PlayTennis
3     Rain        Mild     High    Weak        Yes
4     Rain        Cool   Normal    Weak        Yes
5     Rain        Cool   Normal  Strong         No
9     Rain        Mild   Normal    Weak        Yes
13    Rain        Mild     High  Strong         No
['No' 'Yes']
 [2 3]
[0.4 0.6]
0.3467680694480959
   Outlook Temperature Humidity    Wind PlayTennis
0    Sunny         Hot     High    Weak         No
1    Sunny         Hot     High  Strong         No
7    Sunny        Mild     High    Weak         No
8    Sunny        Cool   Normal    Weak        Yes
10   Sunny        Mild   Normal  Strong        Yes
['No' 'Yes']
 [3 2]
[0.6 0.4]
0.6935361388961918
['No' 'Yes']
 [5 9]
[0.35714286 0.64285714]
0.9402859586706311
['Cool' 'Hot' 'Mild']
 [4 4 6]
    Outlook Temperature Humidity    Wind PlayTennis
4      Rain        Cool   Normal    Weak        Yes
5      Rain        Cool   Normal  Strong         No
6  Overcast        Cool   Normal  Strong        Yes
8     Sunny        Cool   Normal    Weak        Yes
['No' 'Yes']
 [1 3]
[0.25 0.75]
0.23179374984546652
     Outlook Temperature Humidity    Wind PlayTennis
0      Sunny         Hot     High    Weak         No
1      Sunny         Hot     High  Strong         No
2   Overcast         Hot     High    Weak        Yes
12  Overcast         Hot   Normal    Weak        Yes
['No' 'Yes']
 [2 2]
[0.5 0.5]
0.5175080355597522
     Outlook Temperature Humidity    Wind PlayTennis
3       Rain        Mild     High    Weak        Yes
7      Sunny        Mild     High    Weak         No
9       Rain        Mild   Normal    Weak        Yes
10     Sunny        Mild   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
13      Rain        Mild     High  Strong         No
['No' 'Yes']
 [2 4]
[0.33333333 0.66666667]
0.9110633930116763
['No' 'Yes']
 [5 9]
[0.35714286 0.64285714]
0.9402859586706311
['High' 'Normal']
 [7 7]
     Outlook Temperature Humidity    Wind PlayTennis
0      Sunny         Hot     High    Weak         No
1      Sunny         Hot     High  Strong         No
2   Overcast         Hot     High    Weak        Yes
3       Rain        Mild     High    Weak        Yes
7      Sunny        Mild     High    Weak         No
11  Overcast        Mild     High  Strong        Yes
13      Rain        Mild     High  Strong         No
['No' 'Yes']
 [4 3]
[0.57142857 0.42857143]
0.49261406801712576
```

```
     Outlook Temperature Humidity    Wind PlayTennis
4       Rain        Cool   Normal    Weak        Yes
5       Rain        Cool   Normal  Strong         No
6   Overcast        Cool   Normal  Strong        Yes
8      Sunny        Cool   Normal    Weak        Yes
9       Rain        Mild   Normal    Weak        Yes
10     Sunny        Mild   Normal  Strong        Yes
12  Overcast         Hot   Normal    Weak        Yes
['No' 'Yes']
 [1 6]
[0.14285714 0.85714286]
0.7884504573082896
['No' 'Yes']
 [5 9]
[0.35714286 0.64285714]
0.9402859586706311
['Strong' 'Weak']
 [6 8]
     Outlook Temperature Humidity    Wind PlayTennis
1      Sunny         Hot     High  Strong         No
5       Rain        Cool   Normal  Strong         No
6   Overcast        Cool   Normal  Strong        Yes
10     Sunny        Mild   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
13      Rain        Mild     High  Strong         No
['No' 'Yes']
 [3 3]
[0.5 0.5]
0.42857142857142855
     Outlook Temperature Humidity  Wind PlayTennis
0      Sunny         Hot     High  Weak         No
2   Overcast         Hot     High  Weak        Yes
3       Rain        Mild     High  Weak        Yes
4       Rain        Cool   Normal  Weak        Yes
7      Sunny        Mild     High  Weak         No
8      Sunny        Cool   Normal  Weak        Yes
9       Rain        Mild   Normal  Weak        Yes
12  Overcast         Hot   Normal  Weak        Yes
['No' 'Yes']
 [2 6]
[0.25 0.75]
0.8921589282623617
['No' 'Yes']
 [2 3]
[0.4 0.6]
0.9709505944546686
['Cool' 'Mild']
 [2 3]
  Temperature Humidity    Wind PlayTennis
4        Cool   Normal    Weak        Yes
5        Cool   Normal  Strong         No
['No' 'Yes']
 [1 1]
[0.5 0.5]
0.4
   Temperature Humidity    Wind PlayTennis
3         Mild     High    Weak        Yes
9         Mild   Normal    Weak        Yes
13        Mild     High  Strong         No
['No' 'Yes']
 [1 2]
[0.33333333 0.66666667]
0.9509775004326937
['No' 'Yes']
 [2 3]
[0.4 0.6]
0.9709505944546686
['High' 'Normal']
 [2 3]
   Temperature Humidity    Wind PlayTennis
3         Mild     High    Weak        Yes
13        Mild     High  Strong         No
['No' 'Yes']
 [1 1]
[0.5 0.5]
0.4
  Temperature Humidity    Wind PlayTennis
4        Cool   Normal    Weak        Yes
5        Cool   Normal  Strong         No
9        Mild   Normal    Weak        Yes
['No' 'Yes']
 [1 2]
[0.33333333 0.66666667]
0.9509775004326937
['No' 'Yes']
 [2 3]
[0.4 0.6]
0.9709505944546686
```

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

Print Tree

In [12]:
```
tree
```

Out[12]:
```
{'Outlook': {'Overcast': 'Yes',
  'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}},
  'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}
```

Extra: Create Predict Function

In [ ]:
```
def predict(tree, sample):
```