INDEX

S NO	DATE	EXPERIMENT NAME	PAGE NO	SIGN	MARKS

NUMPY BASICS- 1

Create an array of 10 elements.

1. Find the unique elements
2. Add 5 elements in array
3. Delete 2 elements
4. Update an element at position 3

Create another array of 5 elements

1. Add two arrays
2. Append the arrays
3. Find the common elements between two arrays

Create an array of order 5X5

1. Find the average of the 3rd column
2. Count number of non zero elements in 4th column
3. Find max and min values in the entire array
4. Print elements greater than 2 in second column
5. Find the row having the max element

```
import numpy as np
arr1=np.array([1,2,3,3,4,6,7,8,9,10])
arr1
```
OUTPUT: array([1, 2, 3, 3, 4, 6, 7, 8, 9, 10])

```
np.unique(arr1)
```
OUTPUT: array([1, 2, 3, 4, 6, 7, 8, 9, 10])

```
x=np.array([11,20,50,30,2])
arr1=np.concatenate((arr1,x))
arr1
```
OUTPUT: array([1, 2, 3, 3, 4, 6, 7, 8, 9, 10, 11, 20, 50, 30, 2])

```
# give indices to delete as second argument
arr1=np.delete(arr1,[2,5])
arr1[3]=100
```

```
arr1
```
OUTPUT: array([1, 2, 3, 100, 7, 8, 9, 10, 11, 20, 50, 30, 2])

```
arr2=np.array([101,102,103,104])
arr2
```
OUTPUT: array([101, 102, 103, 104])

```
arr2=np.append(arr2,[10,20,30])
arr2
```
OUTPUT: array([101, 102, 103, 104, 10, 20, 30])

```
# gives common elements in both arrays
np.intersect1d(arr1,arr2)
```
OUTPUT: array([10, 20, 30])

```
# takes random integers from given range (1,10)--first argument
# shape 5X5 ---second argument
arr3=np.random.randint(1,10,(5,5))
arr3
```
OUTPUT: array([[6, 9, 8, 3, 4],
[6, 9, 6, 3, 9],
[1, 4, 3, 6, 1],
[3, 6, 7, 9, 3],
[7, 1, 9, 3, 4]])

```
column3=arr3[:,2]
column3
```
OUTPUT: array([8, 6, 3, 7, 9])

```
column3.mean()
```
OUTPUT: 6.6

```
non=np.nonzero(arr3[:,3])
len(non[0])
```
OUTPUT: 5

```
np.count_nonzero(arr3[:,3])
```
OUTPUT: 5

```
arr3.min()
```
OUTPUT: 1

```
arr3.max()
```
OUTPUT: 9

```
# where funtion returns the indices of places where the condition is satisfied
i=np.where(arr3[:,1]>2)
arr3[i,1]
```
OUTPUT: array([[9, 9, 4, 6]])

```
ind=np.where(arr3==np.max(arr3))
print("Rows having max element are : ",ind[0])
```
OUTPUT: Rows having max element are : [0 1 1 3 4]

NUMPY BASICS - 2

DATA SET:

age	job	marital	balance	housing	loan	contact	month
30	unemployed	married	1,787.00	no	no	cellular	oct
33	services	married	4,789.00	yes	yes	cellular	may
35	management	single	1,350.00	yes	no	cellular	apr
30	management	married	1,476.00	yes	yes	unknown	jun
59	blue-collar	married	0	yes	no	unknown	may
35	management	single	747	no	no	cellular	feb
36	self-employed	married	307	yes	no	cellular	may
39	technician	married	147	yes	no	cellular	may
41	entrepreneur	married	221	yes	no	unknown	may
43	services	married	-88	yes	yes	cellular	apr
39	services	married	9,374.00	yes	no	unknown	may
43	admin.	married	264	yes	no	cellular	apr
36	technician	married	1,109.00	no	no	cellular	aug
20	student	single	502	no	no	cellular	apr
31	blue-collar	married	360	yes	yes	cellular	jan

```
import numpy as np
data = np.array([[30.0, 'unemployed', 'married', 1787.0, 'no', 'no',
'cellular','oct'],

       [33.0, 'services', 'married', 4789.0, 'yes', 'yes',
'cellular','may'],
```

```
      [35.0, 'management', 'single', 1350.0, 'yes', 'no',
'cellular','apr'],

      [30.0, 'management', 'married', 1476.0, 'yes', 'yes',
'unknown','jun'],

      [59.0, 'blue-collar', 'married', 0.0, 'yes', 'no',
'unknown','may'],

      [35.0, 'management', 'single', 747.0, 'no', 'no',
'cellular','feb'],

      [36.0, 'self-employed', 'married', 307.0, 'yes', 'no',
'cellular','may'],

      [39.0, 'technician', 'married', 147.0, 'yes', 'no',
'cellular','may'],

      [41.0, 'entrepreneur', 'married', 221.0, 'yes', 'no',
'unknown','may'],

      [43.0, 'services', 'married', -88.0, 'yes', 'yes',
'cellular','apr'],

      [39.0, 'services', 'married', 9374.0, 'yes', 'no',
'unknown','may'],

      [43.0, 'admin.', 'married', 264.0, 'yes', 'no', 'cellular', 'apr'],

      [36.0, 'technician', 'married', 1109.0, 'no', 'no',
'cellular','aug'],

      [20.0, 'student', 'single', 502.0, 'no', 'no', 'cellular', 'apr'],

      [31.0, 'blue-collar', 'married', 360.0, 'yes', 'yes',
'cellular','jan'],

      [40.0, 'management', 'married', 194.0, 'no', 'yes',
'cellular','aug'],

      [56.0, 'technician', 'married', 4073.0, 'no', 'no',
'cellular','aug'],

      [37.0, 'admin.', 'single', 2317.0, 'yes', 'no', 'cellular', 'apr'],
```

```
       [25.0, 'blue-collar', 'single', -221.0, 'yes', 'no',
'unknown','may'],

       [31.0, 'services', 'married', 132.0, 'no', 'no',
'cellular','jul']], dtype=object)
data
```

OUTPUT:

```
array([[30.0, 'unemployed', 'married', 1787.0, 'no', 'no', 'cellular',
'oct'], [33.0, 'services', 'married', 4789.0, 'yes', 'yes', 'cellular',
'may'], [35.0, 'management', 'single', 1350.0, 'yes', 'no', 'cellular',
'apr'], [30.0, 'management', 'married', 1476.0, 'yes', 'yes', 'unknown',
'jun'], [59.0, 'blue-collar', 'married', 0.0, 'yes', 'no', 'unknown',
'may'], [35.0, 'management', 'single', 747.0, 'no', 'no', 'cellular',
'feb'], [36.0, 'self-employed', 'married', 307.0, 'yes', 'no', 'cellular',
'may'], [39.0, 'technician', 'married', 147.0, 'yes', 'no', 'cellular',
'may'], [41.0, 'entrepreneur', 'married', 221.0, 'yes', 'no', 'unknown',
'may'], [43.0, 'services', 'married', -88.0, 'yes', 'yes', 'cellular',
'apr'], [39.0, 'services', 'married', 9374.0, 'yes', 'no', 'unknown',
'may'], [43.0, 'admin.', 'married', 264.0, 'yes', 'no', 'cellular',
'apr'], [36.0, 'technician', 'married', 1109.0, 'no', 'no', 'cellular',
'aug'], [20.0, 'student', 'single', 502.0, 'no', 'no', 'cellular', 'apr'],
[31.0, 'blue-collar', 'married', 360.0, 'yes', 'yes', 'cellular', 'jan'],
[40.0, 'management', 'married', 194.0, 'no', 'yes', 'cellular', 'aug'],
[56.0, 'technician', 'married', 4073.0, 'no', 'no', 'cellular', 'aug'],
[37.0, 'admin.', 'single', 2317.0, 'yes', 'no', 'cellular', 'apr'], [25.0,
'blue-collar', 'single', -221.0, 'yes', 'no', 'unknown', 'may'], [31.0,
'services', 'married', 132.0, 'no', 'no', 'cellular', 'jul']],
dtype=object)
```

1.What is the total bank balance of the first 5 records from the data?

```
np.sum(data[:5,3])
```

OUTPUT: 9402.0

2. What is the average bank balance?

```
np.mean(data[:,3])
```

OUTPUT: 1442.0

3. Count the number of people who has a housing?

```
print("Indices of people with housing : ",np.where(data[:,4]=="yes"))
```

OUTPUT: Indices of people with housing : (array([1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 14, 17, 18]),)

```
print("Number of people with housing:",len(np.where(data[:,4]=="yes")[0]))
```

OUTPUT: Number of people with housing: 13

4. No of people who have a "cellular" contact and also "married" and hold a "management" job?

```
c1=data[:,1]=="management"
c2=data[:,6]=='cellular'
c3=data[:,2]=='married'
print("Indices of people Satisfying all 3 conditions : ",np.where(c1 & c2 & c3)[0])
print("Number of people Satisfying all 3 conditions : ",len(np.where(c1 & c2 & c3)[0]))
```

OUTPUT: Indices of people Satisfying all 3 conditions : [15]
Number of people Satisfying all 3 conditions : 1

PANDAS BASICS

```
#upload the dataset in the same folder of jupyter notebook or else copy the entire path
import pandas as pd
df=pd.read_csv("IMDBMovieData.csv")
df
```

OUTPUT:

	ID	Title	Genre	Director	Year	Runtime_minutes	Rating	Votes	Revenue_millions
0	1	Guardians of the Galaxy	Action	James Gunn	2014	121	8.1	757074	333.13
1	2	Prometheus	Adventure	Ridley Scott	2012	124	7.0	485820	126.46
2	3	Split	Horror	M. Night Shyamalan	2016	117	7.3	157606	138.12
3	4	Sing	Animation	Christophe Lourdelet	2016	108	7.2	60545	270.32
4	5	Suicide Squad	Action	David Ayer	2016	123	6.2	393727	325.02
...	...	...	...	...	...	...	...	...	...
995	996	Secret in Their Eyes	Crime	Billy Ray	2015	111	6.2	27585	NaN
996	997	Hostel: Part II	Horror	Eli Roth	2007	94	5.5	73152	17.54
997	998	Step Up 2: The Streets	Drama	Jon M. Chu	2008	98	6.2	70699	58.01
998	999	Search Party	Adventure	Scot Armstrong	2014	93	5.6	4881	NaN
999	1000	Nine Lives	Comedy	Barry Sonnenfeld	2016	87	5.3	12435	19.64

1000 rows × 9 columns

Checking The Data

1. df.head(),df.tail(), df.info(),df.columns

```
# Head return the first 5 rows of the dataset
df.head()
```

OUTPUT:

	ID	Title	Genre	Director	Year	Runtime_minutes	Rating	Votes	Revenue_millions
0	1	Guardians of the Galaxy	Action	James Gunn	2014	121	8.1	757074	333.13
1	2	Prometheus	Adventure	Ridley Scott	2012	124	7.0	485820	126.46
2	3	Split	Horror	M. Night Shyamalan	2016	117	7.3	157606	138.12
3	4	Sing	Animation	Christophe Lourdelet	2016	108	7.2	60545	270.32
4	5	Suicide Squad	Action	David Ayer	2016	123	6.2	393727	325.02

```
#tail return the last 5 rows of the dataset
df.tail()
```

OUTPUT:

	ID	Title	Genre	Director	Year	Runtime_minutes	Rating	Votes	Revenue_millions
995	996	Secret in Their Eyes	Crime	Billy Ray	2015	111	6.2	27585	NaN
996	997	Hostel: Part II	Horror	Eli Roth	2007	94	5.5	73152	17.54
997	998	Step Up 2: The Streets	Drama	Jon M. Chu	2008	98	6.2	70699	58.01
998	999	Search Party	Adventure	Scot Armstrong	2014	93	5.6	4881	NaN
999	1000	Nine Lives	Comedy	Barry Sonnenfeld	2016	87	5.3	12435	19.64

```
#info return the metadata of dataset
df.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ID                1000 non-null   int64  
 1   Title             1000 non-null   object 
 2   Genre             1000 non-null   object 
 3   Director          1000 non-null   object 
 4   Year              1000 non-null   int64  
 5   Runtime_minutes   1000 non-null   int64  
 6   Rating            1000 non-null   float64
 7   Votes             1000 non-null   int64  
 8   Revenue_millions  872 non-null    float64
dtypes: float64(2), int64(4), object(3)
memory usage: 70.4+ KB
```

```
# returns a list of all the column names in the dataset
df.columns
```

OUTPUT:

```
Index(['ID', 'Title', 'Genre', 'Director', 'Year', 'Runtime_minutes',
'Rating', 'Votes', 'Revenue_millions'], dtype='object')
```

2. Identify the null values.

```
#Entire dataset will True if Null or False if not null
df.isnull()
```

OUTPUT:

	ID	Title	Genre	Director	Year	Runtime_minutes	Rating	Votes	Revenue_millions
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
995	False	False	False	False	False	False	False	False	True
996	False	False	False	False	False	False	False	False	False
997	False	False	False	False	False	False	False	False	False
998	False	False	False	False	False	False	False	False	True
999	False	False	False	False	False	False	False	False	False

1000 rows × 9 columns

```
#Count of null rows for each column
df.isnull().sum()
```

OUTPUT:

```
ID                  0
Title               0
Genre               0
Director            0
Year                0
Runtime_minutes     0
Rating              0
Votes               0
Revenue_millions    128
dtype: int64
```

```
#Count of rows containing null values
df.isnull().sum().sum()
```

OUTPUT: 128

```
#Rows With null values
df[df.isnull().any(axis=1)]
```

OUTPUT:

	ID	Title	Genre	Director	Year	Runtime_minutes	Rating	Votes	Revenue_millions
7	8	Mindhorn	Comedy	Sean Foley	2016	89	6.4	2490	NaN
22	23	Hounds of Love	Crime	Ben Young	2016	108	6.7	1115	NaN
25	26	Paris pieds nus	Comedy	Dominique Abel	2016	83	6.8	222	NaN
39	40	5- 25- 77	Comedy	Patrick Read Johnson	2007	113	7.1	241	NaN
42	43	Don't Fuck in the Woods	Horror	Shawn Burkett	2016	73	2.7	496	NaN
...	...	...	...	...	...	...	...	...	...
977	978	Amateur Night	Comedy	Lisa Addario	2016	92	5.0	2229	NaN
978	979	It's Only the End of the World	Drama	Xavier Dolan	2016	97	7.0	10658	NaN
988	989	Martyrs	Horror	Pascal Laugier	2008	99	7.1	63785	NaN
995	996	Secret in Their Eyes	Crime	Billy Ray	2015	111	6.2	27585	NaN
998	999	Search Party	Adventure	Scot Armstrong	2014	93	5.6	4881	NaN

128 rows × 9 columns

Aggregate Functions

3. What is the average, max,min,runtime?

```
#Using Pandas

print("Average Runtime : ",df['Runtime_minutes'].mean())
print("Maximum Runtime : ",df['Runtime_minutes'].max())
print("Minimum Runtime : ",df['Runtime_minutes'].min())
```

OUTPUT:

```
Average Runtime :  113.172
Maximum Runtime :  191
Minimum Runtime :  66
```

```
#Using Numpy

import numpy as np
data=np.array(df)
print("Average Runtime : ",np.mean(data[:,5]))
print("Maximum Runtime : ",np.max(data[:,5]))
print("Minimum Runtime : ",np.min(data[:,5]))
```

OUTPUT:

```
Average Runtime :  113.172
Maximum Runtime :  191
Minimum Runtime :  66
```

Filtering

4. How many movies got revenue greater than 50

```
#indices where Revenue is >50
df['Revenue_millions']>50
```

OUTPUT:

```
0       True
1       True
2       True
3       True
4       True
       ...
995    False
996    False
997     True
998    False
999    False
Name: Revenue_millions, Length: 1000, dtype: bool
```

```
#Rows where Revenue is >50
df[df['Revenue_millions']>50]
```

OUTPUT:

	ID	Title	Genre	Director	Year	Runtime_minutes	Rating	Votes	Revenue_millions
0	1	Guardians of the Galaxy	Action	James Gunn	2014	121	8.1	757074	333.13
1	2	Prometheus	Adventure	Ridley Scott	2012	124	7.0	485820	126.46
2	3	Split	Horror	M. Night Shyamalan	2016	117	7.3	157606	138.12
3	4	Sing	Animation	Christophe Lourdelet	2016	108	7.2	60545	270.32
4	5	Suicide Squad	Action	David Ayer	2016	123	6.2	393727	325.02
...	...	...	...	...	...	...	...	...	...
983	984	Let's Be Cops	Comedy	Luke Greenfield	2014	104	6.5	112729	82.39
989	990	Selma	Biography	Ava DuVernay	2014	128	7.5	67637	52.07
993	994	Resident Evil: Afterlife	Action	Paul W.S. Anderson	2010	97	5.9	140900	60.13
994	995	Project X	Comedy	Nima Nourizadeh	2012	88	6.7	164088	54.72
997	998	Step Up 2: The Streets	Drama	Jon M. Chu	2008	98	6.2	70699	58.01

429 rows × 9 columns

```
print("Count of movies got revenue greater than 50 :
",len(df[df['Revenue_millions']>50]))
```

OUTPUT: Count of movies got revenue greater than 50 : 429

5. How many movies got revenue greater than 50 and rating less than 7

```
#Rows where movies got revenue greater than 50 and rating less than 7
df[(df['Rating']<7) & (df['Revenue_millions']>50)]
```

OUTPUT:

	ID	Title	Genre	Director	Year	Runtime_minutes	Rating	Votes	Revenue_millions
4	5	Suicide Squad	Action	David Ayer	2016	123	6.2	393727	325.02
15	16	The Secret Life of Pets	Animation	Chris Renaud	2016	87	6.6	120259	368.31
17	18	Jason Bourne	Action	Paul Greengrass	2016	123	6.7	150823	162.16
23	24	Trolls	Animation	Walt Dohrn	2016	92	6.5	38552	153.69
24	25	Independence Day: Resurgence	Action	Roland Emmerich	2016	120	5.3	127553	103.14
...	...	...	...	...	...	...	...	...	...
981	982	Annie	Comedy	Will Gluck	2014	118	5.3	27312	85.91
983	984	Let's Be Cops	Comedy	Luke Greenfield	2014	104	6.5	112729	82.39
993	994	Resident Evil: Afterlife	Action	Paul W.S. Anderson	2010	97	5.9	140900	60.13
994	995	Project X	Comedy	Nima Nourizadeh	2012	88	6.7	164088	54.72
997	998	Step Up 2: The Streets	Drama	Jon M. Chu	2008	98	6.2	70699	58.01

211 rows × 9 columns

```
print("Number of movies got revenue greater than 50 and rating less than 7
: ",len(df[(df['Rating']<7) & (df['Revenue_millions']>50)]))
```

OUTPUT:

Number of movies got revenue greater than 50 and rating less than 7 : 211

6. Show year wise revenue.

```
x=df.groupby('Year')
y=x['Revenue_millions'].sum()
y
```

OUTPUT:

```
Year
2006     3624.46
2007     4306.23
2008     5053.22
2009     5292.26
2010     5989.65
2011     5431.96
2012     6910.29
2013     7666.72
2014     7997.40
2015     8854.12
2016    11211.65
Name: Revenue_millions, dtype: float64
```

7. Show mean rating of each genre

```
a=df.groupby('Genre')
b=a['Rating'].mean()
b
```

OUTPUT:

```
Genre Action        6.592491
Adventure           6.908000
Animation           7.324490
Biography           7.318750
Comedy              6.493143
Crime               6.807042
Drama               6.954872
Fantasy             5.850000
Horror              5.867391
Mystery             6.876923
Romance             6.600000
Sci-Fi              4.966667
Thriller            5.960000
Name: Rating, dtype: float64
```

Q1. BAYE'S THEOREM

1. The probability that it is Friday and that a student is absent is 3 %. Since there are 5 school days in a week, the probability that it is Friday is 20 %. What is theprobability that a student is absent given that today is Friday? Apply Baye's rule in python to get the result. (Ans: 15%)

```
def bayestheorem(Pf , Pfa):
    return Pfa/Pf

Pfriday = 0.2
Pfriday_about = 0.03
bayestheorem(Pfriday,Pfriday_about)
```

OUTPUT: 0.15

2. Extract the data from database using python

```
pip install mysql-connector-python
import mysql.connector as m
conn=m.connect( host="localhost",
  user="root",
  password="password",)
print(conn)
```

OUTPUT: <mysql.connector.connection.MySQLConnection object at 0x000001A9897EBDD0>

1. Creating DataBase using a cursor

```
cur=conn.cursor()
cur.execute("CREATE DATABASE CSM")
```

2. Connecting to the DataBase

```
db=m.connect(
  host="localhost",
  user="root",
  password="password",
  database="CSM")
```

```
print(db)
dbcur=db.cursor()
```

OUTPUT: <mysql.connector.connection.MySQLConnection object at 0x000001A9897F80D0>

3. Creating a Table

```
dbcur.execute("CREATE TABLE Products(Productname varchar(100),Cost int)");
dbcur.execute("SHOW TABLES")

for q in dbcur:
    print(q)
```

OUTPUT: ('products',)

4. Inserting Data into the Table

```
sql='INSERT INTO products (Productname,Cost) Values (%s,%s)'
val=[('Pen',10),('Book',110),('Marker',30)]
dbcur.executemany(sql,val)
db.commit()

print(dbcur.rowcount,"rows are inserted.")
```

OUTPUT: 3 rows are inserted.

5. Executing Some Basic Queries

```
dbcur.execute("SELECT * FROM products")
for row in dbcur:
    print(row)
```

OUTPUT: ('Pen', 10)
('Book', 110)
('Marker', 30)

```
dbcur.execute("SELECT Productname FROM products where cost<50")
for row in dbcur:
    print(row)
```

OUTPUT:
('Pen',)
('Marker',)

3. Implement k-nearest neighbours classification using python

1. Import necessary modules and Loading data

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report, confusion_matrix

irisData = load_iris()
```

2. Creating feature and target arrays

```
X = irisData.data
y = irisData.target
print("DataSet of Lengths and Breadth of sepals and petals : ")
X
```

OUTPUT :

DataSet of Lengths and Breadth of sepals and petals :
array([[5.1, 3.5, 1.4, 0.2],
[4.9, 3. , 1.4, 0.2],
[4.7, 3.2, 1.3, 0.2],
[4.6, 3.1, 1.5, 0.2],
[5. , 3.6, 1.4, 0.2],
[5.4, 3.9, 1.7, 0.4],
....
[6.7, 3. , 5.2, 2.3],
[6.3, 2.5, 5. , 1.9],
[6.5, 3. , 5.2, 2.],
[6.2, 3.4, 5.4, 2.3],
[5.9, 3. , 5.1, 1.8]])

```
print("Targets for above data : ")
y
```

OUTPUT :

Targets for above data :
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

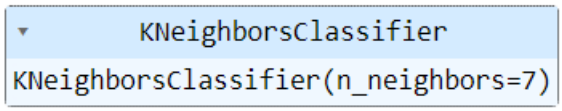
3. Spliting the Data into training and test set

```
# X_train --> 80% of original data used to train the model
# y_train --> 80% of original Targets used to train the model

# X_test --> 20% of original data used to test the model
# y_test --> 20% of original Targets used to test the model

X_train, X_test, y_train, y_test = train_test_split( X, y, test_size =
0.2, random_state=42)
```

4. Training the Model

```
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
```

OUTPUT:



5. Predict on dataset which model has not seen before

```
y_pred=knn.predict(X_test)
print("Predicted values by the model : ")
print(y_pred)
print("Actual Target vslues in Test Dataset :")
print(y_test)
```

OUTPUT:

```
Predicted values by the model :
[1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
Actual Target vslues in Test Dataset :
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0]
```

```
print("Confusion Matrix of above y_pred and y_test :
\n",confusion_matrix(y_test,y_pred))
```

OUTPUT:

```
Confusion Matrix of above y_pred and y_test :
 [[10  0  0]
 [ 0  8  1]
 [ 0  0 11]]
```

```
print("Classification report : \n",classification_report(y_test,y_pred))
```

OUTPUT:

```
Classification report :
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30
```

4. Given the following data, which specify classifications for nine combinations of VAR1 and VAR2 predict a classification for a case where VAR1=0.906 and VAR2=0.606, using the result of k-means clustering with 3 means (i.e., 3 centroids)

VAR1	VAR2	CLASS
1.713	1.586	0
0.180	1.786	1
0.353	1.240	1
0.940	1.566	0
1.486	0.759	1
1.266	1.106	0
1.540	0.419	1
0.459	1.799	1
0.773	0.186	1

1. Importing necessary modules and Loading data

```
from sklearn.cluster import KMeans
import numpy as np

X = np.array([[1.713,1.586], [0.180,1.786],
[0.353,1.240],[0.940,1.566],[1.486,0.759],[1.266,1.106],[1.540,0.419],[0.4
59,1.799],[0.773,0.186]])
```

2. Training the kmeans model by using 3 centroids

```
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
```

OUTPUT:

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto'
in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
# Number of iterations taken by the model
kmeans.n_iter_
```

OUTPUT: 3

```
# The Final Centroid values
print(kmeans.cluster_centers_)
```

OUTPUT:

```
[[1.26633333 0.45466667]
 [0.33066667 1.60833333]
 [1.30633333 1.41933333]]
```

3. Prediction

```
# Cluster of each Data point
print(kmeans.labels_)
```

OUTPUT:

```
[2 1 1 2 0 2 0 1 0]
```

```
print('Cluster number of Data Point (0.906,0.606) is :
',kmeans.predict([[0.906, 0.606]]))
```

OUTPUT:

```
Cluster number of Data Point (0.906,0.606) is :  [0]
```