Artificial Intelligence & Machin Learning

Experiment No. 5

Introduction to Python Programming: Learn the different libraries – NumPy, Pandas, SciPy, Matplotlib .

Name: Sumit Bhatkar Roll No. 05

PYTHON PRACTICALS

PRACTICAL NO. 5

Aim: Introduction to Python Programming: Learn the different libraries – NumPy, Pandas, SciPy, Matplotlib .

Objective: To learn different libraries in python.

Software Requirement:

- **Anaconda Navigator:** Anaconda Navigator is a desktop graphical user interface included in Anaconda that allows you to launch applications and easily manage conda packages, environments and channels without the need to use command line commands.

Theory:

- NumPy: NumPy can be used **to perform a wide variety of mathematical operations on arrays.**
- Pandas: **Pandas** is a **Python** library. **Pandas** is used to analyze data.
- SciPy: SciPy is **a scientific computation library that uses NumPy underneath**. SciPy stands for Scientific Python. It provides more utility functions for optimization, stats and signal processing.
- Matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

Code & Output:

1. **NumPy:**

Import numpy

```
In [74]: import numpy as np
```

```
In [82]: # create an array
         digits = np.array([
               [1, 2, 3],
               [4, 5, 6],
             [6, 7, 9],
             ])
```

```
In [3]: digits
```

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

```
In [83]: # addition of two integers
         a=2
         b=4
```

```
In [5]: c=a+b
        c
```

```
Out[5]: 6
```

In [85]:
```
# Study shape and axes of an array.
temperatures = np.array([
      29.3, 42.1, 18.8, 16.1, 38.0, 12.5,
       12.6, 49.9, 38.6, 31.3, 9.2, 22.2
    ]).reshape(2, 2, 3)
```

In [7]:
```
In [3]: temperatures.shape
```

Out[7]:
```
(2, 2, 3)
```

In [9]:
```
In [4]: temperatures
```

Out[9]:
```
array([[[29.3, 42.1, 18.8],
        [16.1, 38. , 12.5]],

       [[12.6, 49.9, 38.6],
        [31.3,  9.2, 22.2]]])
```

In [10]:
```
In [5]: np.swapaxes(temperatures, 1, 2)
```

Out[10]:
```
array([[[29.3, 16.1],
        [42.1, 38. ],
        [18.8, 12.5]],

       [[12.6, 31.3],
        [49.9,  9.2],
        [38.6, 22.2]]])
```

In [11]:
```
table = np.array([
   ...:     [5, 3, 7, 1],
   ...:     [2, 6, 7 ,9],
   ...:     [1, 1, 1, 1],
   ...:     [4, 3, 2, 0],
   ...: ])
```

In [12]:
```
table.max()
```

Out[12]:
```
9
```

In [13]:
```
 table.max(axis=0)
```

Out[13]:
```
array([5, 6, 7, 9])
```

In [14]:
```
 table.max(axis=1)
```

Out[14]:
```
array([7, 9, 1, 4])
```

```
In [89]: #Study of Broadcasting with an array.
         A= np.arange(32).reshape(4, 1, 8)

In [16]: A

Out[16]: array([[[ 0,  1,  2,  3,  4,  5,  6,  7]],

                [[ 8,  9, 10, 11, 12, 13, 14, 15]],

                [[16, 17, 18, 19, 20, 21, 22, 23]],

                [[24, 25, 26, 27, 28, 29, 30, 31]]])

In [17]: B = np.arange(48).reshape(1, 6, 8)

In [18]: B

Out[18]: array([[[ 0,  1,  2,  3,  4,  5,  6,  7],
                 [ 8,  9, 10, 11, 12, 13, 14, 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],
                 [40, 41, 42, 43, 44, 45, 46, 47]]])
```


```
In [89]: #Study of Broadcasting with an array.
         A= np.arange(32).reshape(4, 1, 8)

In [16]: A

Out[16]: array([[[ 0,  1,  2,  3,  4,  5,  6,  7]],

                [[ 8,  9, 10, 11, 12, 13, 14, 15]],

                [[16, 17, 18, 19, 20, 21, 22, 23]],

                [[24, 25, 26, 27, 28, 29, 30, 31]]])

In [17]: B = np.arange(48).reshape(1, 6, 8)

In [18]: B

Out[18]: array([[[ 0,  1,  2,  3,  4,  5,  6,  7],
                 [ 8,  9, 10, 11, 12, 13, 14, 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],
                 [40, 41, 42, 43, 44, 45, 46, 47]]])
```

```
In [90]: # Addition of two Arrays.
         A+B
```

```
Out[90]: array([[[ 0,  2,  4,  6,  8, 10, 12, 14],
                 [ 8, 10, 12, 14, 16, 18, 20, 22],
                 [16, 18, 20, 22, 24, 26, 28, 30],
                 [24, 26, 28, 30, 32, 34, 36, 38],
                 [32, 34, 36, 38, 40, 42, 44, 46],
                 [40, 42, 44, 46, 48, 50, 52, 54]],

                [[ 8, 10, 12, 14, 16, 18, 20, 22],
                 [16, 18, 20, 22, 24, 26, 28, 30],
                 [24, 26, 28, 30, 32, 34, 36, 38],
                 [32, 34, 36, 38, 40, 42, 44, 46],
                 [40, 42, 44, 46, 48, 50, 52, 54],
                 [48, 50, 52, 54, 56, 58, 60, 62]],

                [[16, 18, 20, 22, 24, 26, 28, 30],
                 [24, 26, 28, 30, 32, 34, 36, 38],
                 [32, 34, 36, 38, 40, 42, 44, 46],
                 [40, 42, 44, 46, 48, 50, 52, 54],
                 [48, 50, 52, 54, 56, 58, 60, 62],
                 [56, 58, 60, 62, 64, 66, 68, 70]],

                [[24, 26, 28, 30, 32, 34, 36, 38],
                 [32, 34, 36, 38, 40, 42, 44, 46],
                 [40, 42, 44, 46, 48, 50, 52, 54],
                 [48, 50, 52, 54, 56, 58, 60, 62],
                 [56, 58, 60, 62, 64, 66, 68, 70],
                 [64, 66, 68, 70, 72, 74, 76, 78]]])
```

```
In [91]: #Find the Square of an array.
         square = np.array([
                [16, 3, 2, 13],
                [5, 10, 11, 8],
                [9, 6, 7, 12],
                [4, 15, 14, 1]
            ])
```

```
In [21]: for i in range(4):
            ...:     assert square[:, i].sum() == 34
            ...:     assert square[i, :].sum() == 34
            ...:
```

```
In [22]: assert square[:2, :2].sum() == 34
```

```
In [23]: assert square[2:, :2].sum() == 34
```

```
In [24]:  assert square[:2, 2:].sum() == 34
```

```
In [25]: assert square[2:, 2:].sum() == 34
```

```
In [92]: #Study of masking and filtering.
         numbers = np.linspace(5, 50, 24, dtype=int).reshape(4, -1)
```

```
In [27]: numbers
```

```
Out[27]: array([[ 5,  6,  8, 10, 12, 14],
                [16, 18, 20, 22, 24, 26],
                [28, 30, 32, 34, 36, 38],
                [40, 42, 44, 46, 48, 50]])
```

```
In [28]: mask = numbers % 4 == 0

In [29]: mask

Out[29]: array([[False, False,  True, False,  True, False],
                [ True, False,  True, False,  True, False],
                [ True, False,  True, False,  True, False],
                [ True, False,  True, False,  True, False]])

In [30]: numbers[mask]

Out[30]: array([ 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48])

In [31]: by_four = numbers[numbers % 4 == 0]

In [32]: by_four

Out[32]: array([ 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48])

In [33]: from numpy.random import default_rng

In [34]: rng = default_rng()

In [35]: values = rng.standard_normal(10000)

In [36]:  values[:5]

Out[36]: array([-1.78348667e-01, -1.41216557e+00, -2.25700591e+00,  1.80981733e-03,
                3.64960139e-01])
```

```
In [37]: std = values.std()

In [38]: std

Out[38]: 1.0010075480221816

In [39]: filtered = values[(values > -2 * std) & (values < 2 * std)]

In [40]: filtered.size

Out[40]: 9532

In [41]:  values.size

Out[41]: 10000

In [42]: filtered.size / values.size

Out[42]: 0.9532
```

```
In [94]: a = np.array([
             [1, 2],
           [3, 4],
           [5, 6],
         ])

In [95]: #Transposing, Sorting, and Concatenating of arrays.
         a.T

Out[95]: array([[1, 3, 5],
                [2, 4, 6]])
```

```
In [45]: a.transpose()
Out[45]: array([[1, 3, 5],
                [2, 4, 6]])

In [46]: data = np.array([
   ...:     [7, 1, 4],
   ...:     [8, 6, 5],
   ...:     [1, 2, 3]
   ...: ])

In [47]: np.sort(data)
Out[47]: array([[1, 4, 7],
                [5, 6, 8],
                [1, 2, 3]])

In [48]: np.sort(data, axis=None)
Out[48]: array([1, 1, 2, 3, 4, 5, 6, 7, 8])

In [49]: np.sort(data, axis=0)
Out[49]: array([[1, 1, 3],
                [7, 2, 4],
                [8, 6, 5]])

In [50]: a = np.array([
   ...:     [4, 8],
   ...:     [6, 1]
   ...: ])

In [51]: b = np.array([
   ...:     [3, 5],
   ...:     [7, 2],
   ...: ])

In [52]: np.hstack((a, b))
Out[52]: array([[4, 8, 3, 5],
                [6, 1, 7, 2]])

In [53]: np.vstack((b, a))
Out[53]: array([[3, 5],
                [7, 2],
                [4, 8],
                [6, 1]])

In [54]: np.concatenate((a, b))
Out[54]: array([[4, 8],
                [6, 1],
                [3, 5],
                [7, 2]])

In [55]: np.concatenate((a, b), axis=None)
Out[55]: array([4, 8, 6, 1, 3, 5, 7, 2])
```

In [96]:
```
#Implementation of Maclaurin Series.
from math import e, factorial
fac = np.vectorize(factorial)

def e_x(x, terms=10):
    """Approximates e^x using a given number of terms of
    the Maclaurin series
    """
    n = np.arange(terms)
    return np.sum((x ** n) / fac(n))

if __name__ == "__main__":
    print("Actual:", e ** 3)  # Using e from the standard library

    print("N (terms)\tMaclaurin\tError")

    for n in range(1, 14):
        maclaurin = e_x(3, terms=n)
        print(f"{n}\t\t{maclaurin:.03f}\t\t{e**3 - maclaurin:.03f}")
```

```
Actual: 20.085536923187664
N (terms)       Maclaurin       Error
1               1.000           19.086
2               4.000           16.086
3               8.500           11.586
4               13.000          7.086
5               16.375          3.711
6               18.400          1.686
7               19.412          0.673
8               19.846          0.239
9               20.009          0.076
10              20.063          0.022
11              20.080          0.006
12              20.084          0.001
13              20.085          0.000
```

In [97]:
```
#Study of different Datatypes(numerical,String)
a = np.array([1, 3, 5.5, 7.7, 9.2], dtype=np.single)
a
```

Out[97]:
```
array([1. , 3. , 5.5, 7.7, 9.2], dtype=float32)
```

In [60]:
```
b = np.array([1, 3, 5.5, 7.7, 9.2], dtype=np.uint8)
b
```

Out[60]:
```
array([1, 3, 5, 7, 9], dtype=uint8)
```

In [61]:
```
names = np.array(["bob", "amy", "han"], dtype=str)
```

In [62]:
```
names
```

Out[62]:
```
array(['bob', 'amy', 'han'], dtype='<U3')
```

In [63]:
```
names.itemsize
```

Out[63]:
```
12
```

In [64]:
```
names = np.array(["bob", "amy", "han"])
```

In [65]:
```
names
```

Out[65]:
```
array(['bob', 'amy', 'han'], dtype='<U3')
```

In [66]:
```
more_names = np.array(["bobo", "jehosephat"])
```

In [67]:
```
np.concatenate((names, more_names))
```

Out[67]:
```
array(['bob', 'amy', 'han', 'bobo', 'jehosephat'], dtype='<U10')
```

In [68]:
```
names[2] = "jasica"
```

In [69]:
```
names
```

Out[69]:
```
array(['bob', 'amy', 'jas'], dtype='<U3')
```

In [98]:
```
#Study of structured array
data = np.array([
     ("joe", 32, 6),
     ("mary", 15, 20),
    ("felipe", 80, 100),
    ("beyonce", 38, 9001),
   ], dtype=[("name", str, 10), ("age", int), ("power", int)])
```

In [71]:
```
data[0]
```

Out[71]:
```
('joe', 32, 6)
```

In [72]:
```
data["name"]
```

Out[72]:
```
array(['joe', 'mary', 'felipe', 'beyonce'], dtype='<U10')
```

In [73]:
```
data[data["power"] > 9000]["name"]
```

Out[73]:
```
array(['beyonce'], dtype='<U10')
```

In [108]:
```
#numpy version
print("Numpy Version:",np.version.version)
```

```
Numpy Version: 1.19.5
```

2. Pandas:

In [4]:
```
# import Pandas Print version
import pandas as pd
print(pd.__version__)
```

```
1.2.4
```

In [5]:
```
#create a dataframe
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
```

In [6]:
```
purchases = pd.DataFrame(data)

purchases
```

Out[6]:

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

In [7]:
```
purchases = pd.DataFrame(data, index=['June', 'Robert', 'Lily', 'David'])

purchases
```

Out[7]:

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

In [8]:
```
purchases.loc['June']
```

Out[8]:
```
apples     3
oranges    0
Name: June, dtype: int64
```

In [10]:
```
#read csv file
df = pd.read_csv('f.csv')

df
```

Out[10]:

	1	ram	7
0	2	sonali	8
1	3	teena	9
2	4	rahul	0

In [11]:
```
#read csv with index
df = pd.read_csv('f.csv', index_col=0)

df
```

Out[11]:

	ram	7
1		
2	sonali	8
3	teena	9
4	rahul	0

In [1]:
```
#Create Dataframe:
import pandas as pd
df = pd.DataFrame({'X':[78,85,96,80,86], 'Y':[84,94,89,83,86],'Z':[86,97,96,72,83]});
print(df)
```

```
    X   Y   Z
0  78  84  86
1  85  94  97
2  96  89  96
3  80  83  72
4  86  86  83
```

In [2]:
```
#create series
s = pd.Series([2, 4, 6, 8, 10])
print(s)
```

```
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

In [5]:
```
#load the data and make sure to change the path for your localdirectory
data = pd.read_csv('project_data.csv')
```

In [6]:
```
#first 5 rows
data.head()
```

Out[6]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
201701	1982	Graduation	Single	58138.0	9/4/2012	58	8	4	0	3	11
201702	1950	Graduation	Married	46344.0	3/8/2014	38	1	2	0	3	11
201703	1965	Graduation	Divorced	71613.0	8/21/2013	26	8	10	0	3	11
201704	1984	Graduation	Relationship	26646.0	2/10/2014	26	2	4	0	3	11
201705	1981	PhD	Widowed	58293.0	1/19/2014	94	5	6	0	3	11

In [7]:
```
#last 5 rows
data.tail()
```

Out[7]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
202195	1944	PhD	Divorced	55614.0	11/27/2013	85	9	6	0	3	11
202196	1962	Master	Divorced	59432.0	4/13/2013	88	5	11	0	3	11
202197	1978	Graduation	Divorced	55563.0	4/5/2014	22	2	3	0	3	11
202198	1971	PhD	Relationship	43624.0	4/21/2013	83	4	4	0	6	11
202199	1949	PhD	Relationship	41461.0	5/22/2014	63	6	11	0	6	11

In [8]:
```
#check the basic information of the data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499 entries, 0 to 498
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   customer_id        499 non-null    int64
 1   year_of_birth      499 non-null    int64
 2   educational_level  499 non-null    object
 3   marital_status     499 non-null    object
 4   annual_income      486 non-null    float64
 5   purhcase_date      499 non-null    object
 6   recency            499 non-null    int64
 7   online_purchases   499 non-null    int64
 8   store_purchases    499 non-null    int64
 9   complaints         499 non-null    int64
 10  calls              499 non-null    int64
 11  intercoms          499 non-null    int64
dtypes: float64(1), int64(8), object(3)
memory usage: 46.9+ KB
```

In [9]:
```
#extract the shape of the data
data.shape
```

Out[9]:
```
(499, 12)
```

In [10]:
```
data['marital_status'].unique()
```

Out[10]:
```
array(['Single', 'Married', 'Divorced', 'Relationship', 'Widowed',
       'Widow'], dtype=object)
```

In [12]:
```
#count educational level
round(data['educational_level'].value_counts(normalize=True),2)
```

Out[12]:
```
Graduation     0.52
PhD            0.23
Master         0.16
High School    0.08
Basic          0.01
Name: educational_level, dtype: float64
```

In [13]:
```
#missing values or duplicate values
data.isnull()
data.duplicated().sum()
data['educational_level'].isnull().sum()
#specifying Education as a variable where we should look for the sum of missing values
```

Out[13]:
```
0
```

In [14]:
```
#Select and filter data: loc and iloc
subset_data = data[['year_of_birth ', 'educational_level', 'annual_income']]
subset_data
```

Out[14]:

	year_of_birth	educational_level	annual_income
0	1982	Graduation	58138.0
1	1950	Graduation	46344.0
2	1965	Graduation	71613.0
3	1984	Graduation	26646.0
4	1981	PhD	58293.0
...	...	...	...
494	1944	PhD	55614.0
495	1962	Master	59432.0
496	1978	Graduation	55563.0
497	1971	PhD	43624.0
498	1949	PhD	41461.0

499 rows × 3 columns

In [15]:
```
#select a unique category of the education by specifying that only "Master" should be returned from the data frame.
data[data["educational_level"] == "Master"]
```

Out[15]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
201706	1967	Master	Relationship	62000.0	9/9/2013	16	6	10	5	3	11
201714	1952	Master	Single	59354.0	11/15/2013	53	6	5	0	3	11
201719	1980	Master	Single	76995.0	3/28/2013	91	11	9	4	3	11
201731	1989	Master	Divorced	10979.0	5/22/2014	34	3	3	8	3	11
201732	1963	Master	Single	38820.0	5/11/2013	56	2	3	0	3	11
...	...	...	...	...	...	...	...	...	...	...	...
202182	1972	Master	Divorced	37760.0	8/11/2013	54	2	3	0	3	11
202185	1960	Master	Relationship	29027.0	10/10/2012	93	5	4	0	0	11
202187	1976	Master	Relationship	56290.0	11/14/2013	4	3	7	0	11	11
202194	1964	Master	Single	58308.0	1/12/2013	77	2	3	0	3	11
202196	1962	Master	Divorced	59432.0	4/13/2013	88	5	11	0	3	11

2 columns

In [16]:
```
#specify the rows and columns as labels
data.loc[:6, ['educational_level', 'recency']]
```

Out[16]:

	educational_level	recency
0	Graduation	58
1	Graduation	38
2	Graduation	26
3	Graduation	26
4	PhD	94
5	Master	16
6	Graduation	34

In [17]:
```
#specify rows and columns as integer based values
data.iloc[:6, [2,6]]
```

Out[17]:

	educational_level	recency
0	Graduation	58
1	Graduation	38
2	Graduation	26
3	Graduation	26
4	PhD	94
5	Master	16

In [22]:
```
#choosing the customers with an income higher than 75,000 and with a master's degree.
data.iloc[list((data.annual_income > 75000) & (data.educational_level == 'Master')),  :,]
```

Out[22]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
201719	1980	Master	Single	76995.0	3/28/2013	91	11	9	4	3	11
201752	1964	Master	Single	79143.0	8/11/2012	2	6	13	0	3	11
201756	1955	Master	Married	82384.0	11/19/2012	55	3	13	0	3	11
201761	1982	Master	Single	75777.0	7/4/2013	12	3	11	0	3	11
201777	1993	Master	Married	75251.0	8/27/2012	34	7	5	0	3	11
201810	1993	Master	Single	89058.0	12/7/2012	18	5	4	0	3	7
201821	1957	Master	Relationship	88193.0	6/20/2013	65	6	10	0	5	11
201841	1987	Master	Single	92859.0	10/19/2012	46	5	12	0	3	2
201918	1985	Master	Widowed	83790.0	11/15/2013	81	8	6	0	3	11
201978	1981	Master	Single	77882.0	4/30/2014	29	3	5	0	3	11
202006	1983	Master	Widowed	80950.0	3/28/2013	44	6	9	0	3	11
202124	1973	Master	Relationship	82584.0	6/4/2013	26	3	8	0	3	11
202136	1983	Master	Relationship	82634.0	6/21/2013	49	1	3	0	0	11

#Apply data operations: index, new variables, data types

In [26]:
```
#set the index as customer_id
data.set_index("customer_id")
```

Out[26]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
201701	1982	Graduation	Single	58138.0	9/4/2012	58	8	4	0	3	11
201702	1950	Graduation	Married	46344.0	3/8/2014	38	1	2	0	3	11
201703	1965	Graduation	Divorced	71613.0	8/21/2013	26	8	10	0	3	11
201704	1984	Graduation	Relationship	26646.0	2/10/2014	26	2	4	0	3	11
201705	1981	PhD	Widowed	58293.0	1/19/2014	94	5	6	0	3	11
...	...	...	...	...	...	...	...	...	...	...	...
202195	1944	PhD	Divorced	55614.0	11/27/2013	85	9	6	0	3	11
202196	1962	Master	Divorced	59432.0	4/13/2013	88	5	11	0	3	11
202197	1978	Graduation	Divorced	55563.0	4/5/2014	22	2	3	0	3	11
202198	1971	PhD	Relationship	43624.0	4/21/2013	83	4	4	0	6	11
202199	1949	PhD	Relationship	41461.0	5/22/2014	63	6	11	0	6	11

ws × 11 columns

In [27]:
```
#sort the data by year_of_birth, ascending is default;
data.sort_values(by = ['year_of_birth '], ascending = True)
# if we want it in descending we should set ascending = False
```

Out[27]:

ner_id	year_of_birth	educational_level	marital_status	annual_income	purhcase_date	recency	online_purchases	store_purchases	complaints	calls	intercoms
202040	1899	PhD	Single	83532.0	9/26/2013	36	4	4	0	3	11
201733	1940	Graduation	Married	40548.0	10/10/2012	31	2	4	0	3	11
202059	1943	Master	Married	65073.0	8/20/2013	65	5	5	1	3	11
201740	1943	PhD	Divorced	48948.0	2/1/2013	53	7	5	0	3	11
202195	1944	PhD	Divorced	55614.0	11/27/2013	85	9	6	0	3	11
...	...	...	...	...	...	...	...	...	...	...	...
201717	2000	Graduation	Married	41850.0	12/24/2012	51	3	3	7	3	11
202119	2000	Graduation	Single	91065.0	2/22/2013	33	7	9	0	3	11
201817	2000	Graduation	Relationship	90765.0	1/24/2014	25	4	5	0	3	11
201886	2000	Graduation	Single	25271.0	12/5/2012	45	1	2	0	3	11
202153	2000	Master	Single	36230.0	10/17/2013	17	2	4	0	3	11

12 columns

In [28]:
```
#create a new variable which is the sum of all purchases performed by customers
data['sum_purchases'] = data.online_purchases + data.store_purchases
data['sum_purchases']
```

Out[28]:
```
0      12
1       3
2      18
3       6
4      11
       ..
494    15
495    16
496     5
497     8
498    17
Name: sum_purchases, Length: 499, dtype: int64
```

In [29]:
```
#create an income category (low, meduim, high) based on the income variable
income_categories = ['Low', 'Meduim', 'High'] #set the categories
bins = [0,75000,120000,600000] #set the income boundaries
cats= pd.cut(data['annual_income'],bins, labels=income_categories) #apply the pd.cut method
data['Income_Category'] = cats #assign the categories based on income
data[['annual_income', 'Income_Category']]
```

Out[29]:

	annual_income	Income_Category
0	58138.0	Low
1	46344.0	Low
2	71613.0	Low
3	26646.0	Low
4	58293.0	Low
...	...	...
494	55614.0	Low
495	59432.0	Low
496	55563.0	Low
497	43624.0	Low
498	41461.0	Low

499 rows × 2 columns

In [46]:
```
#apply groupby to find the mean of income, recency, number of web and store purchases by educational group
aggregate_view = pd.DataFrame(data.groupby(by='educational_level')[['annual_income', 'recency', 'store_purchases',  'online_purch
aggregate_view
```

Out[46]:

	educational_level	annual_income	recency	store_purchases	online_purchases
0	Basic	19514.571429	53.571429	2.857143	1.571429
1	Graduation	51607.827309	47.171206	5.840467	3.887160
2	High School	44154.717949	58.400000	4.600000	3.450000
3	Master	51191.700000	45.000000	5.691358	4.049383
4	PhD	55878.990991	49.008772	6.298246	4.429825

In [49]:
```
#apply pivot table to find the aggregated sum of purchases and mean of recency per education and marital status group
import numpy as np
pivot_table = pd.DataFrame(pd.pivot_table(data, values=['sum_purchases', 'recency'], index=['marital_status'],
columns=['educational_level'], aggfunc={'recency': np.mean, 'sum_purchases': np.sum}, fill_value=0)).reset_index()
pivot_table
```

Out[49]:

	marital_status	recency					sum_purchases				
educational_level		Basic	Graduation	High School	Master	PhD	Basic	Graduation	High School	Master	PhD
0	Divorced	68.333333	54.897959	64.666667	60.083333	41.350000	13	481	31	134	232
1	Married	0.000000	42.701493	66.866667	50.315789	60.000000	0	652	120	193	236
2	Relationship	39.333333	48.196078	49.615385	36.800000	43.161290	15	464	99	159	364
3	Single	52.000000	44.278689	49.000000	42.761905	49.315789	3	623	54	218	173
4	Widow	0.000000	61.000000	96.000000	14.000000	25.000000	0	34	8	14	6
5	Widowed	0.000000	46.760000	52.000000	40.000000	53.684211	0	246	10	71	212

3. SciPy:

```
In [4]: #Data Analysis with SciPy
        # import numpy library
        import numpy as np
        A = np.array([[1,2,3],[4,5,6],[7,8,8]])
```

```
In [5]: #Linear Algebra
        #Determinant of a Matrix
        # importing linalg function from scipy
        from scipy import linalg

        # Compute the determinant of a matrix
        linalg.det(A)
```

```
Out[5]: 2.999999999999997
```

```
In [6]: #pivoted LU decomposition of a matrix
        P, L, U = linalg.lu(A)
        print(P)
        print(L)
        print(U)
        # print LU decomposition
        print(np.dot(L,U))
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]]
[[1.         0.         0.        ]
 [0.14285714 1.         0.        ]
 [0.57142857 0.5        1.        ]]
[[7.         8.         8.        ]
 [0.         0.85714286 1.85714286]
 [0.         0.         0.5       ]]
[[7. 8. 8.]
 [1. 2. 3.]
 [4. 5. 6.]]
```

```
In [7]: #Eigen values and eigen vectors of above matrix
        eigen_values, eigen_vectors = linalg.eig(A)
        print(eigen_values)
        print(eigen_vectors)
```

```
[15.55528261+0.j -1.41940876+0.j -0.13587385+0.j]
[[-0.24043423 -0.67468642  0.51853459]
 [-0.54694322 -0.23391616 -0.78895962]
 [-0.80190056  0.70005819  0.32964312]]
```

```
In [8]: #linear equations
        v = np.array([[2],[3],[5]])
        print(v)
        s = linalg.solve(A,v)
        print(s)
```

```
[[2]
 [3]
 [5]]
[[-2.33333333]
 [ 3.66666667]
 [-1.        ]]
```

In [9]:
```
#Sparse Linear Algebra
from scipy import sparse
# Row-based linked list sparse matrix
A = sparse.lil_matrix((1000, 1000))
print(A)

A[0,:100] = np.random.rand(100)
A[1,100:200] = A[0,:100]
A.setdiag(np.random.rand(1000))
print(A)
```

```
  (0, 0)        0.11616892671917378
  (0, 1)        0.13503257433879967
  (0, 2)        0.9618747187565171
  (0, 3)        0.02899256849300469
  (0, 4)        0.850262131087913
  (0, 5)        0.9346351616745983
  (0, 6)        0.21428777850603808
  (0, 7)        0.7398978235086023
  (0, 8)        0.09159219936893082
  (0, 9)        0.21523310318480082
  (0, 10)       0.9050708143647447
  (0, 11)       0.8348462936615604
  (0, 12)       0.9042726075329924
  (0, 13)       0.5666525054114153
  (0, 14)       0.27382290310454094
  (0, 15)       0.8697402189342641
  (0, 16)       0.3328942783310157
  (0, 17)       0.382150244305717
```

In [10]:
```
#Integration
import scipy.integrate
f= lambda x:np.exp(-x**2)
# print results
i = scipy.integrate.quad(f, 0, 1)
print(i)
```

```
(0.7468241328124271, 8.291413475940725e-15)
```

In [11]:
```
#Double Integrals
from scipy import integrate
f = lambda y, x: x*y**2
i = integrate.dblquad(f, 0, 2, lambda x: 0, lambda x: 1)
# print the results
print(i)
```

```
(0.6666666666666667, 7.401486830834377e-15)
```

4. Matplotlib:

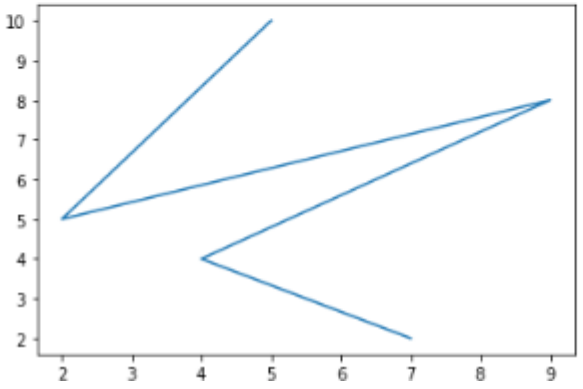
```
In [1]: # importing matplotlib module
        from matplotlib import pyplot as plt
```

```
In [2]: # x-axis values
        x = [5, 2, 9, 4, 7]
```

```
In [3]: # Y-axis values
        y = [10, 5, 8, 4, 2]
```

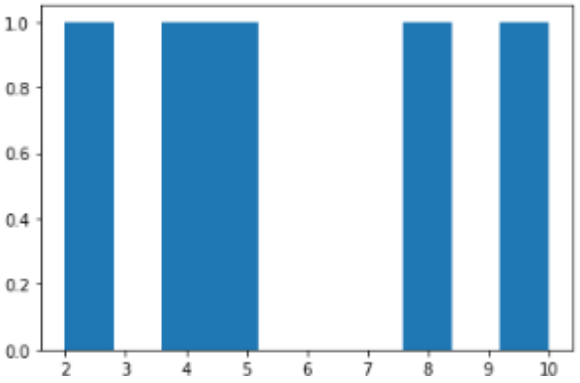
```
In [4]: # Function to plot
        plt.plot(x, y)
```

```
Out[4]: [<matplotlib.lines.Line2D at 0x1f57f64f460>]
```

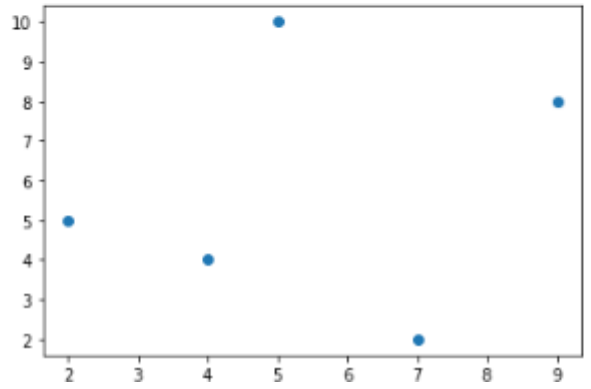



```
In [5]: # function to show the plot
        plt.show()
```

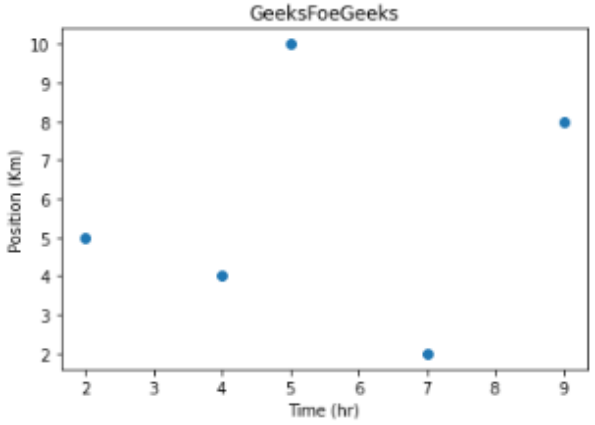
```
In [1]: #Histogram
        from matplotlib import pyplot as plt

        # Y-axis values
        y = [10, 5, 8, 4, 2]

        # Function to plot histogram
        plt.hist(y)

        # Function to show the plot
        plt.show()
```



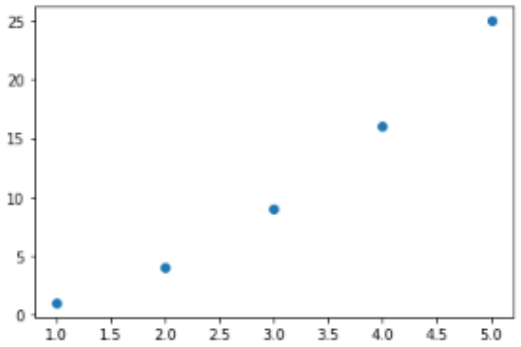
In [2]:
```
#Scatter Plot
# x-axis values
x = [5, 2, 9, 4, 7]

# Y-axis values
y = [10, 5, 8, 4, 2]

# Function to plot scatter
plt.scatter(x, y)

# function to show the plot
plt.show()
```

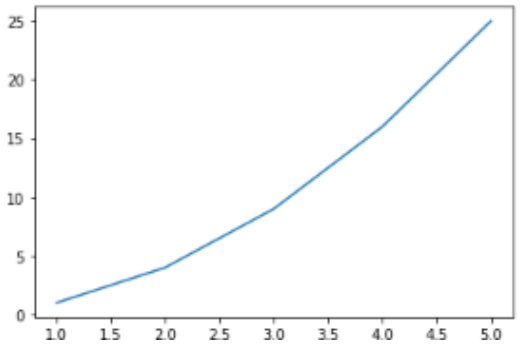


In [3]:
```
#Adding title and Labeling the Axes in the graph
# x-axis values
x = [5, 2, 9, 4, 7]

# Y-axis values
y = [10, 5, 8, 4, 2]

# Function to plot
plt.scatter(x, y)

# Adding Title
plt.title("GeeksFoeGeeks")

# Labeling the axes
plt.xlabel("Time (hr)")
plt.ylabel("Position (Km)")

# function to show the plot
plt.show()
```



In [4]:
```
#Multiple Graphs
x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]
plt.scatter(x, y)

# function to show the plot
plt.show()

plt.plot(x, y)

# function to show the plot
plt.show()
```





Artificial Intelligence & Machin Learning

Experiment No. 6

Introduction to Linear Regression, Logistic regression, KNN- classification.

Name: Sumit Bhatkar Roll No. 05

PRACTICAL NO. 6

Aim: Implementation of linear regression, logistic regression, KNN,- classification.

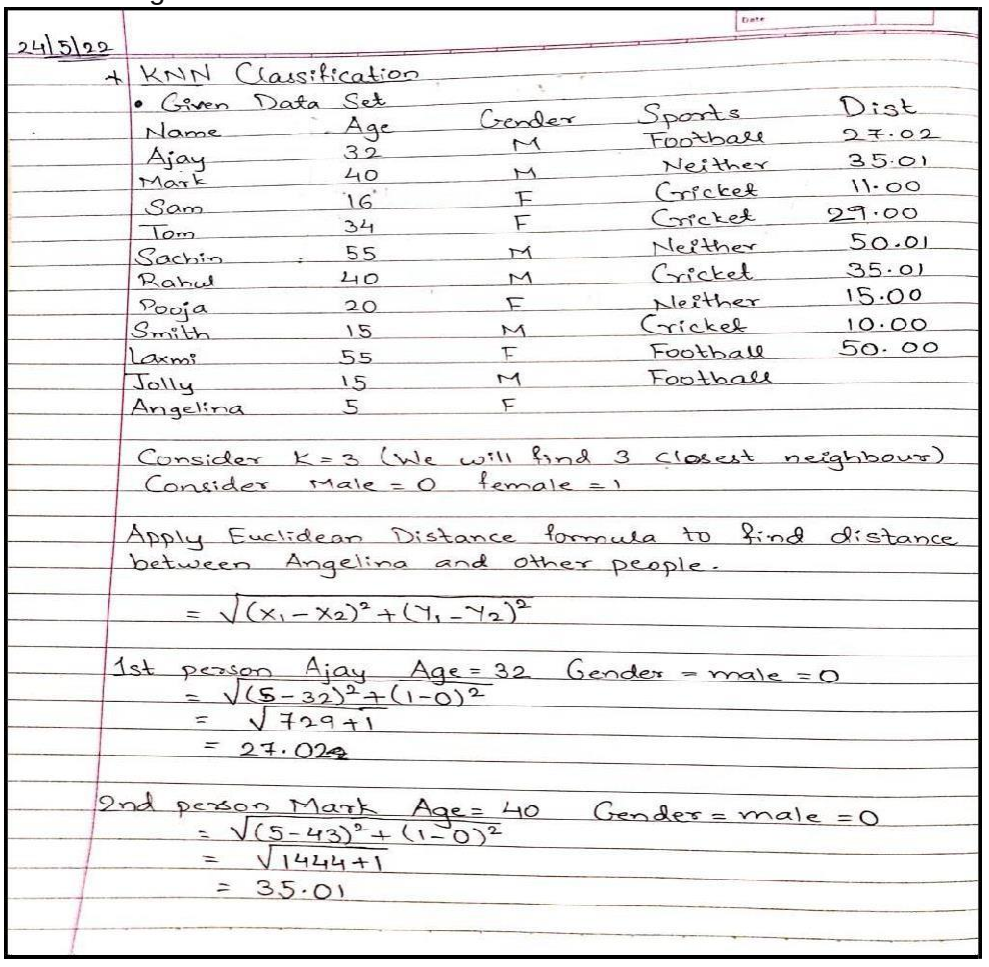
Objective: Understand linear regression, logistic regression, KNN,- classification.

Software Requirement:

- **Anaconda Navigator:** Anaconda Navigator is a desktop graphical user interface included in Anaconda that allows you to launch applications and easily manage conda packages, environments and channels without the need to use command line commands.

Theory:

- **Linear Regression:** Linear regression strives to show the relationship between two variables by applying a linear equation to observed data. One variable is supposed to be an independent variable, and the other is to be a dependent variable.
- **Logistic Regression:** Logistic regression is a process of **modeling the probability of a discrete outcome given an input variable**. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on.
- **KNN Classification:** k-nearest neighbours (knn) is a **non-parametric classification method**, i.e. we do not have to assume a parametric model for the data of the classes Calculate the distance between the query-instance (new observation) and all the training samples Sort the distances and determine the nearest neighbours based on the k-th minimum distance.



24/5/22

* KNN Classification

• Given Data Set

Name	Age	Gender	Sports	Dist
Ajay	32	M	Football	27.02
Mark	40	M	Neither	35.01
Sam	16	F	Cricket	11.00
Tom	34	F	Cricket	29.00
Sachin	55	M	Neither	50.01
Rahul	40	M	Cricket	35.01
Pooja	20	F	Neither	15.00
Smith	15	M	Cricket	10.00
Laxmi	55	F	Football	50.00
Jolly	15	M	Football	
Angelina	5	F		

Consider K = 3 (We will find 3 closest neighbours)
Consider male = 0 female = 1

Apply Euclidean Distance formula to find distance between Angelina and other people.

$= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

1st person Ajay Age = 32 Gender = male = 0

$= \sqrt{(5-32)^2 + (1-0)^2}$

$= \sqrt{729 + 1}$

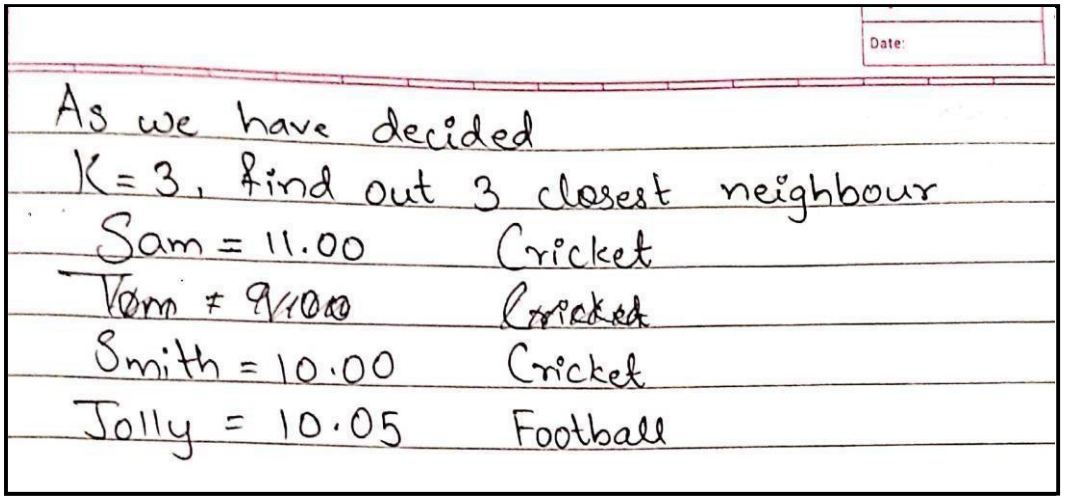
$= 27.02$

2nd person Mark Age = 40 Gender = male = 0

$= \sqrt{(5-43)^2 + (1-0)^2}$

$= \sqrt{1444 + 1}$

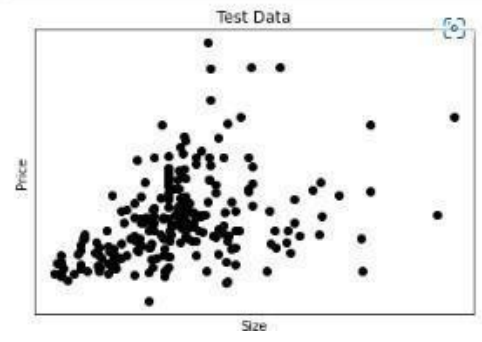
$= 35.01$



Code & Output:

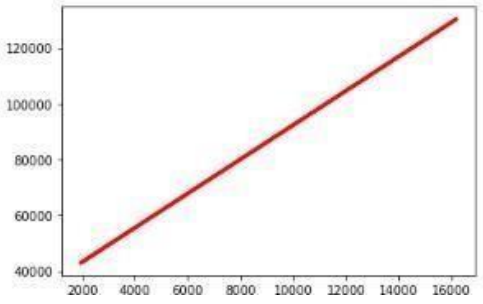
Linear Regression:

In [10]:
```
import matplotlib

import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
import pandas as pd

# Load CSV and columns
df = pd.read_csv("Housing.csv")

Y = df['price']
X = df['lotsize']

X=X.values.reshape(len(X),1)
Y=Y.values.reshape(len(Y),1)

# Split the data into training/testing sets
X_train = X[:-250]
X_test = X[-250:]

# Split the targets into training/testing sets
Y_train = Y[:-250]
Y_test = Y[-250:]

# Plot outputs
plt.scatter(X_test, Y_test,  color='black')
plt.title('Test Data')
plt.xlabel('Size')
plt.ylabel('Price')
plt.xticks(())
plt.yticks(())

plt.show()
```



In [11]:
```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X_train, Y_train)

# Plot outputs
plt.plot(X_test, regr.predict(X_test), color='red',linewidth=3)
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x2cc4ed76430>]
```



Logistic Regression:

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

from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
import seaborn as sns
sns.set(style="white") #white background style for seaborn plots
sns.set(style="whitegrid", color_codes=True)

import warnings
warnings.simplefilter(action='ignore')
```

In [4]:
```
# Read CSV train data file into DataFrame
train_df = pd.read_csv("titanic_train.csv")

# Read CSV test data file into DataFrame
test_df = pd.read_csv("titanic_test.csv")

# preview train data
train_df.head()
```

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

In [5]:
```
print('The number of samples into the train data is {}.'.format(train_df.shape[0]))
```

```
The number of samples into the train data is 891.
```

In [6]:
```
test_df.head()
```

Out[6]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

In [7]:
```
print('The number of samples into the test data is {}.'.format(test_df.shape[0]))
```

```
The number of samples into the test data is 418.
```

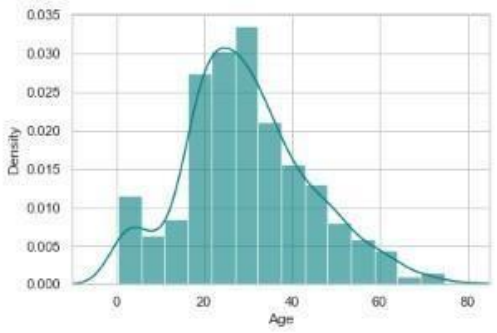
In [8]:
```
# check missing values in train data
train_df.isnull().sum()
```

Out[8]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [9]:
```
# percent of missing "Age"
print('Percent of missing "Age" records is %.2f%%' %((train_df['Age'].isnull().sum()/train_df.shape[0])*100))
```

```
Percent of missing "Age" records is 19.87%
```

In [10]:
```
ax = train_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', alpha=0.6)
train_df["Age"].plot(kind='density', color='teal')
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```



In [11]:
```
# mean age
print('The mean of "Age" is %.2f' %(train_df["Age"].mean(skipna=True)))
# median age
print('The median of "Age" is %.2f' %(train_df["Age"].median(skipna=True)))
```

```
The mean of "Age" is 29.70
The median of "Age" is 28.00
```

In [12]:
```
# percent of missing "Cabin"
print('Percent of missing "Cabin" records is %.2f%%' %((train_df['Cabin'].isnull().sum()/train_df.shape[0])*100))
```

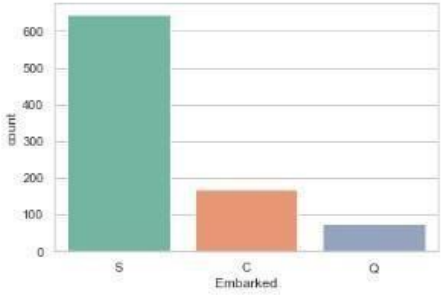
```
Percent of missing "Cabin" records is 77.10%
```

In [13]:
```
#percent of missing "Embarked"
print('Percent of missing "Embarked" records is %.2f%%' %((train_df['Embarked'].isnull().sum()/train_df.shape[0])*100))
```

```
Percent of missing "Embarked" records is 0.22%
```

In [14]:
```
print('Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton):')
print(train_df['Embarked'].value_counts())
sns.countplot(x='Embarked', data=train_df, palette='Set2')
plt.show()
```

```
Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton):
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```



In [15]:
```
print('The most common boarding port of embarkation is %s.' %train_df['Embarked'].value_counts().idxmax())
```

```
The most common boarding port of embarkation is S.
```

In [16]:
```
train_data = train_df.copy()
train_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
train_data["Embarked"].fillna(train_df['Embarked'].value_counts().idxmax(), inplace=True)
train_data.drop('Cabin', axis=1, inplace=True)
```

In [17]:
```
# check missing values in adjusted train data
train_data.isnull().sum()
```

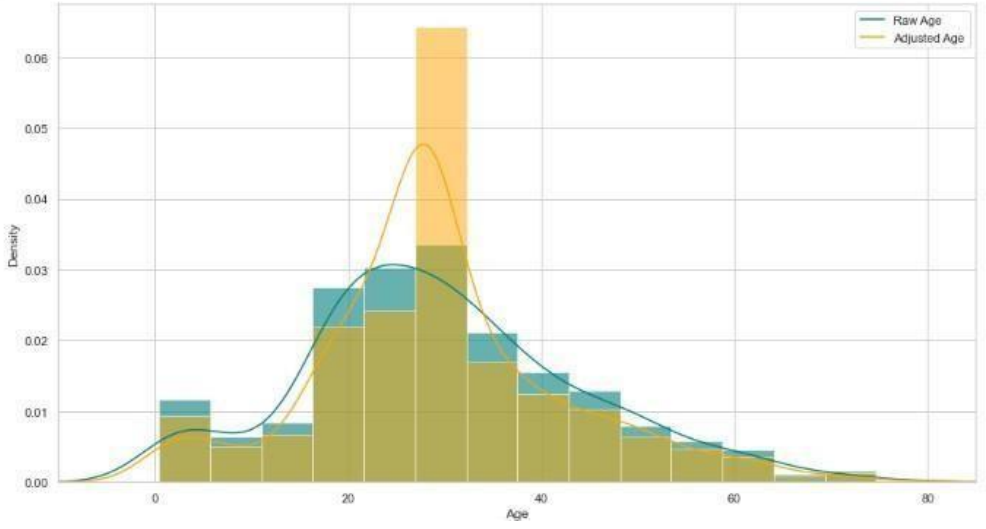
Out[17]:
```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
dtype: int64
```

In [18]:
```
# preview adjusted train data
train_data.head()
```

Out[18]:

	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

In [19]:
```
plt.figure(figsize=(15,8))
ax = train_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', alpha=0.6)
train_df["Age"].plot(kind='density', color='teal')
ax = train_data["Age"].hist(bins=15, density=True, stacked=True, color='orange', alpha=0.5)
train_data["Age"].plot(kind='density', color='orange')
ax.legend(['Raw Age', 'Adjusted Age'])
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```



In [20]:
```
# Create categorical variable for traveling alone
train_data['TravelAlone']=np.where((train_data["SibSp"]+train_data["Parch"])>0, 0, 1)
train_data.drop('SibSp', axis=1, inplace=True)
train_data.drop('Parch', axis=1, inplace=True)
```

In [21]:
```
#create categorical variables and drop some variables
training=pd.get_dummies(train_data, columns=["Pclass","Embarked","Sex"])
training.drop('Sex_female', axis=1, inplace=True)
training.drop('PassengerId', axis=1, inplace=True)
training.drop('Name', axis=1, inplace=True)
training.drop('Ticket', axis=1, inplace=True)

final_train = training
final_train.head()
```

Out[21]:

	Survived	Age	Fare	TravelAlone	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Embarked_Q	Embarked_S	Sex_male
0	0	22.0	7.2500	0	0	0	1	0	0	1	1
1	1	38.0	71.2833	0	1	0	0	1	0	0	0
2	1	26.0	7.9250	1	0	0	1	0	0	1	0
3	1	35.0	53.1000	0	1	0	0	0	0	1	0
4	0	35.0	8.0500	1	0	0	1	0	0	1	1

Now, apply the same changes to the test data. I will apply to same imputation for "Age" in the Test data as I did for my Training data (if missing, Age = 28). I'll also remove the "Cabin" variable from the test data, as I've decided not to include it in my analysis. There were no missing values in the "Embarked" port variable. I'll add the dummy variables to finalize the test set. Finally, I'll impute the 1 missing value for "Fare" with the median, 14.45.

In [22]:
```
test_df.isnull().sum()
```

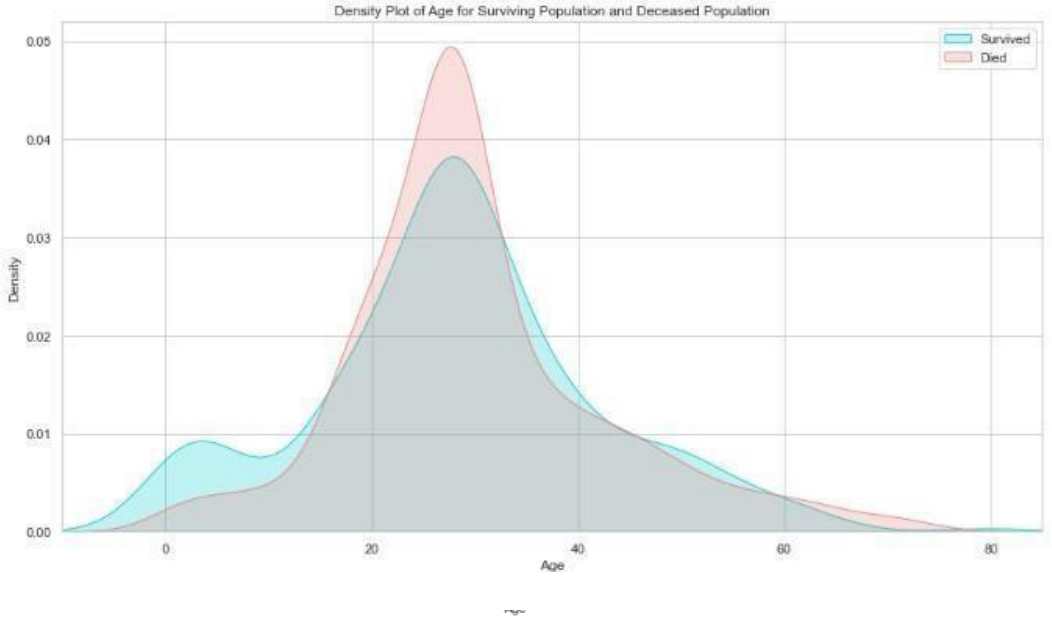
Out[22]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [23]:
```
test_data = test_df.copy()
test_data["Age"].fillna(train_df["Age"].median(skipna=True), inplace=True)
test_data["Fare"].fillna(train_df["Fare"].median(skipna=True), inplace=True)
test_data.drop('Cabin', axis=1, inplace=True)

test_data['TravelAlone']=np.where((test_data["SibSp"]+test_data["Parch"])>0, 0, 1)

test_data.drop('SibSp', axis=1, inplace=True)
test_data.drop('Parch', axis=1, inplace=True)

testing = pd.get_dummies(test_data, columns=["Pclass","Embarked","Sex"])
testing.drop('Sex_female', axis=1, inplace=True)
testing.drop('PassengerId', axis=1, inplace=True)
testing.drop('Name', axis=1, inplace=True)
testing.drop('Ticket', axis=1, inplace=True)

final_test = testing
final_test.head()
```

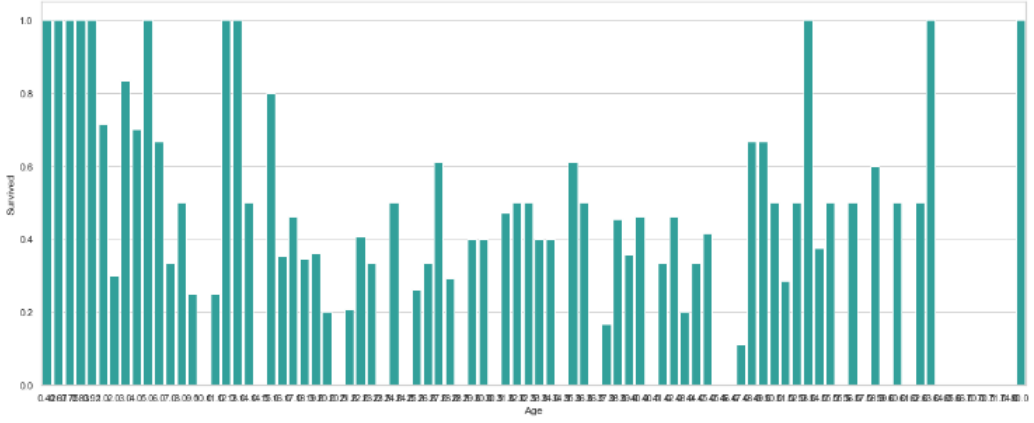
Out[23]:

	Age	Fare	TravelAlone	Pclass_1	Pclass_2	Pclass_3	Embarked_C	Embarked_Q	Embarked_S	Sex_male
0	34.5	7.8292	1	0	0	1	0	1	0	1
1	47.0	7.0000	0	0	0	1	0	0	1	0
2	62.0	9.6875	1	0	1	0	0	1	0	1
3	27.0	8.6625	1	0	0	1	0	0	1	1
4	22.0	12.2875	0	0	0	1	0	0	1	0

In [24]:
```
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Age"][final_train.Survived == 1], color="darkturquoise", shade=True)
sns.kdeplot(final_train["Age"][final_train.Survived == 0], color="lightcoral", shade=True)
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Age for Surviving Population and Deceased Population')
ax.set(xlabel='Age')
plt.xlim(-10,85)
plt.show()
```

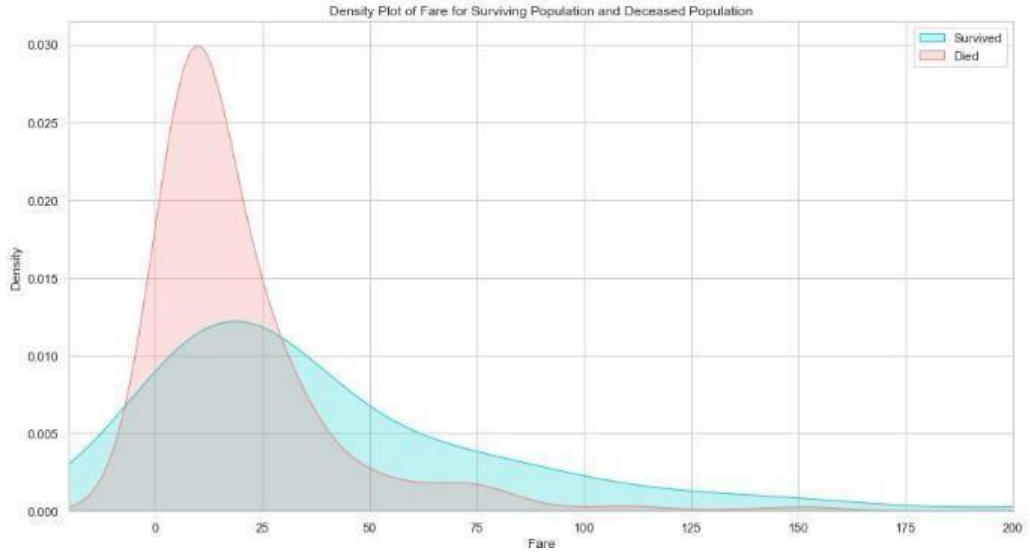


In [25]:
```
plt.figure(figsize=(20,8))
avg_survival_byage = final_train[["Age", "Survived"]].groupby(['Age'], as_index=False).mean()
g = sns.barplot(x='Age', y='Survived', data=avg_survival_byage, color="LightSeaGreen")
plt.show()
```

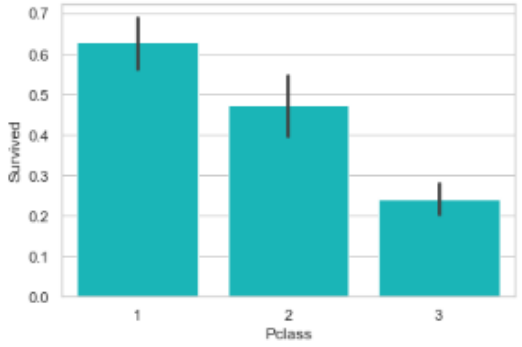


In [26]:
```
final_train['IsMinor']=np.where(final_train['Age']<=16, 1, 0)

final_test['IsMinor']=np.where(final_test['Age']<=16, 1, 0)
```

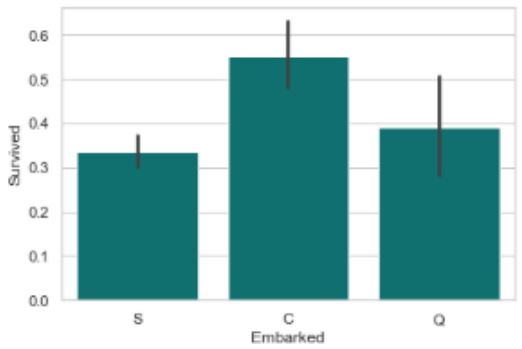
In [27]:
```
#Exploration of Fare
plt.figure(figsize=(15,8))
ax = sns.kdeplot(final_train["Fare"][final_train.Survived == 1], color="darkturquoise", shade=True)
sns.kdeplot(final_train["Fare"][final_train.Survived == 0], color="lightcoral", shade=True)
plt.legend(['Survived', 'Died'])
plt.title('Density Plot of Fare for Surviving Population and Deceased Population')
ax.set(xlabel='Fare')
plt.xlim(-20,200)
plt.show()
```



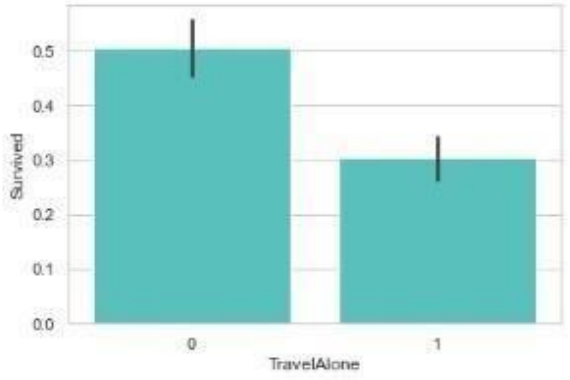
In [28]:
```
#Exploration of Passenger Class
sns.barplot('Pclass', 'Survived', data=train_df, color="darkturquoise")
plt.show()
```



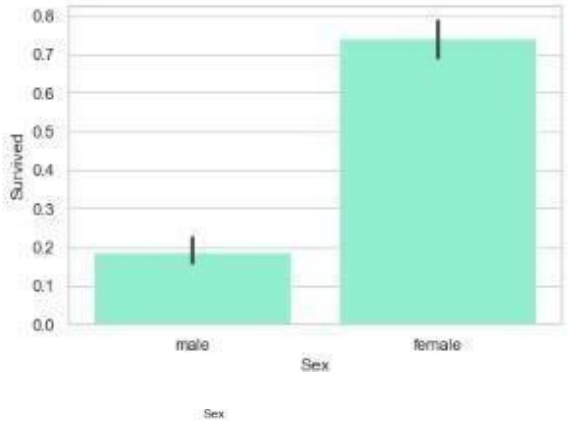
In [30]:
```
#Exploration of Embarked Port
sns.barplot('Embarked', 'Survived', data=train_df, color="teal")
plt.show()
```



In [31]:
```
#Exploration of Traveling Alone vs. With Family
sns.barplot('TravelAlone', 'Survived', data=final_train, color="mediumturquoise")
plt.show()
```



In [32]:
```
#Exploration of Gender Variable
sns.barplot('Sex', 'Survived', data=train_df, color="aquamarine")
plt.show()
```



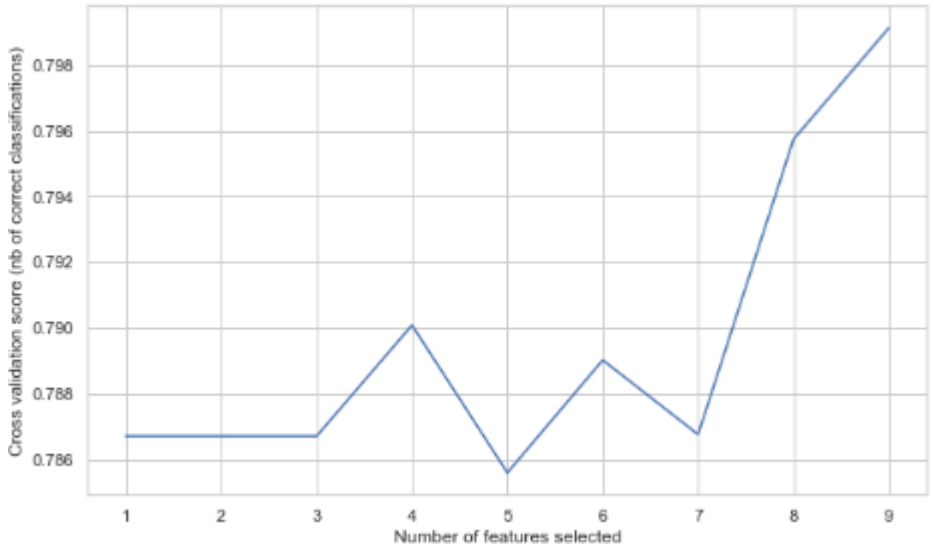
Sex

In [33]:
```
#Logistic Regression and Results
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

cols = ["Age","Fare","TravelAlone","Pclass_1","Pclass_2","Embarked_C","Embarked_S","Sex_male","IsMinor"]
X = final_train[cols]
y = final_train['Survived']
# Build a logreg and compute the feature importances
model = LogisticRegression()
# create the RFE model and select 8 attributes
rfe = RFE(model, 8)
rfe = rfe.fit(X, y)
# summarize the selection of the attributes
print('Selected features: %s' % list(X.columns[rfe.support_]))
```

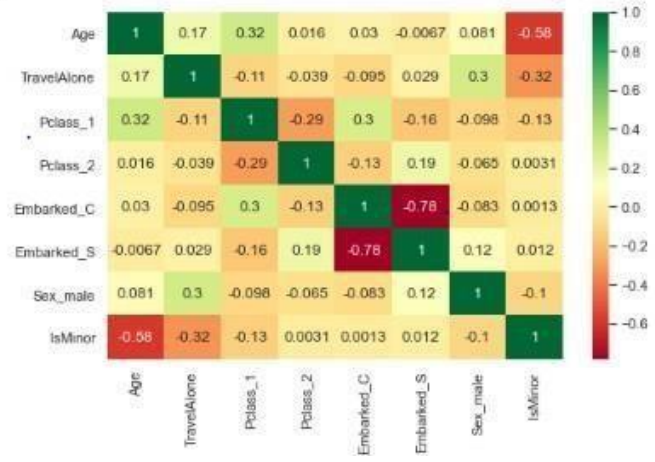
```
Selected features: ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C', 'Embarked_S', 'Sex_male', 'IsMinor']
```

In [34]:
```
from sklearn.feature_selection import RFECV
# Create the RFE object and compute a cross-validated score.
# The "accuracy" scoring is proportional to the number of correct classifications
rfecv = RFECV(estimator=LogisticRegression(), step=1, cv=10, scoring='accuracy')
rfecv.fit(X, y)

print("Optimal number of features: %d" % rfecv.n_features_)
print('Selected features: %s' % list(X.columns[rfecv.support_]))

# Plot number of features VS. cross-validation scores
plt.figure(figsize=(10,6))
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

```
Optimal number of features: 9
Selected features: ['Age', 'Fare', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C', 'Embarked_S', 'Sex_male', 'IsMinor']
```

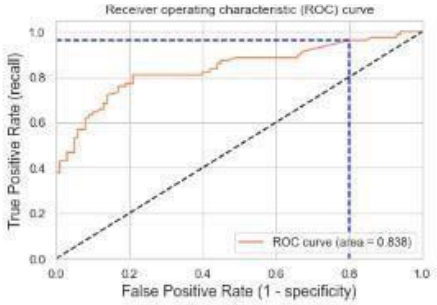



```
In [35]: Selected_features = ['Age', 'TravelAlone', 'Pclass_1', 'Pclass_2', 'Embarked_C',
                            'Embarked_S', 'Sex_male', 'IsMinor']
         X = final_train[Selected_features]

         plt.subplots(figsize=(8, 5))
         sns.heatmap(X.corr(), annot=True, cmap="RdYlGn")
         plt.show()
```



In [37]:
```
#Review of model evaluation procedures
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_curve, auc, log_loss

# create X (features) and y (response)
X = final_train[Selected_features]
y = final_train['Survived']

# use train/test split with different random_state values
# we can change the random_state values that changes the accuracy scores
# the scores change a lot, this is why testing scores is a high-variance estimate
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)

# check classification scores of logistic regression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
y_pred_proba = logreg.predict_proba(X_test)[:, 1]
[fpr, tpr, thr] = roc_curve(y_test, y_pred_proba)
print('Train/Test split results:')
print(logreg.__class__.__name__+" accuracy is %2.3f" % accuracy_score(y_test, y_pred))
print(logreg.__class__.__name__+" log_loss is %2.3f" % log_loss(y_test, y_pred_proba))
print(logreg.__class__.__name__+" auc is %2.3f" % auc(fpr, tpr))

idx = np.min(np.where(tpr > 0.95)) # index of the first threshold for which the sensibility > 0.95

plt.figure()
plt.plot(fpr, tpr, color='coral', label='ROC curve (area = %0.3f)' % auc(fpr, tpr))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot([0,fpr[idx]], [tpr[idx],tpr[idx]], 'k--', color='blue')
plt.plot([fpr[idx],fpr[idx]], [0,tpr[idx]], 'k--', color='blue')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (1 - specificity)', fontsize=14)
plt.ylabel('True Positive Rate (recall)', fontsize=14)
plt.title('Receiver operating characteristic (ROC) curve')
plt.legend(loc="lower right")
plt.show()
print("Using a threshold of %.3f " % thr[idx] + "guarantees a sensitivity of %.3f " % tpr[idx] +
      "and a specificity of %.3f" % (1-fpr[idx]) +
      ", i.e. a false positive rate of %.2f%%." % (np.array(fpr[idx])*100))
```

```
Train/Test split results:
LogisticRegression accuracy is 0.782
LogisticRegression log_loss is 0.504
LogisticRegression auc is 0.838
```




```
Using a threshold of 0.070 guarantees a sensitivity of 0.962 and a specificity of 0.200, i.e. a false positive rate of 80.00%.
```

In [38]:
```
# 10-fold cross-validation logistic regression
logreg = LogisticRegression()
# Use cross_val_score function
# We are passing the entirety of X and y, not X_train or y_train, it takes care of splitting the data
# cv=10 for 10 folds
# scoring = {'accuracy', 'neg_log_loss', 'roc_auc'} for evaluation metric - althought they are many
scores_accuracy = cross_val_score(logreg, X, y, cv=10, scoring='accuracy')
scores_log_loss = cross_val_score(logreg, X, y, cv=10, scoring='neg_log_loss')
scores_auc = cross_val_score(logreg, X, y, cv=10, scoring='roc_auc')
print('K-fold cross-validation results:')
print(logreg.__class__.__name__+" average accuracy is %2.3f" % scores_accuracy.mean())
print(logreg.__class__.__name__+" average log_loss is %2.3f" % -scores_log_loss.mean())
print(logreg.__class__.__name__+" average auc is %2.3f" % scores_auc.mean())
```

```
K-fold cross-validation results:
LogisticRegression average accuracy is 0.796
LogisticRegression average log_loss is 0.454
LogisticRegression average auc is 0.850
```

In [39]:
```
from sklearn.model_selection import cross_validate

scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}

modelCV = LogisticRegression()

results = cross_validate(modelCV, X, y, cv=10, scoring=list(scoring.values()),
                         return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['test_%s' % list(scorin
                                if list(scoring.values())[sc]=='neg_log_loss'
                                else results['test_%s' % list(scoring.values())[sc]].mean(),
                                results['test_%s' % list(scoring.values())[sc]].std()))
```

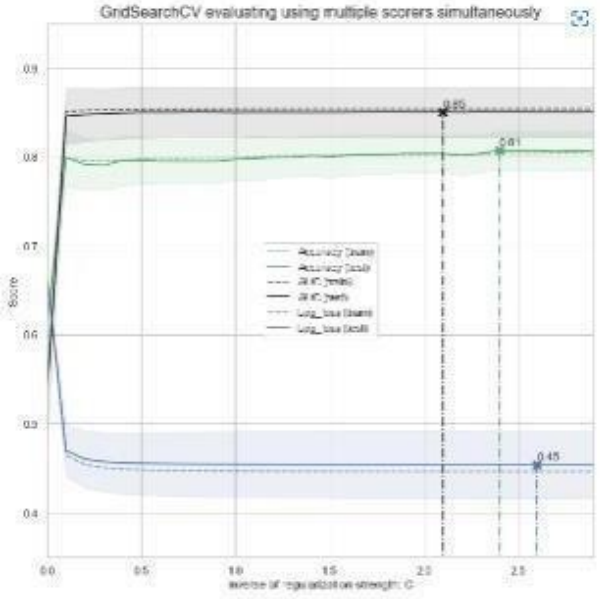
```
K-fold cross-validation results:
LogisticRegression average accuracy: 0.796 (+/-0.024)
LogisticRegression average log_loss: 0.454 (+/-0.037)
LogisticRegression average auc: 0.850 (+/-0.028)
```

In [41]:
```
#What happens when we add the feature "Fare"?

cols = ["Age","Fare","TravelAlone","Pclass_1","Pclass_2","Embarked_C","Embarked_S","Sex_male","IsMinor"]
X = final_train[cols]

scoring = {'accuracy': 'accuracy', 'log_loss': 'neg_log_loss', 'auc': 'roc_auc'}

modelCV = LogisticRegression()

results = cross_validate(modelCV, final_train[cols], y, cv=10, scoring=list(scoring.values()),
                         return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['test_%s' % list(scorin
                                if list(scoring.values())[sc]=='neg_log_loss'
                                else results['test_%s' % list(scoring.values())[sc]].mean(),
                                results['test_%s' % list(scoring.values())[sc]].std()))
```

```
K-fold cross-validation results:
LogisticRegression average accuracy: 0.799 (+/-0.028)
LogisticRegression average log_loss: 0.455 (+/-0.037)
LogisticRegression average auc: 0.849 (+/-0.028)

K-fold cross-validation results:
LogisticRegression average accuracy: 0.799 (+/-0.028)
LogisticRegression average log_loss: 0.455 (+/-0.037)
LogisticRegression average auc: 0.849 (+/-0.028)
```

In [44]:
```
from sklearn.model_selection import GridSearchCV
X = final_train[Selected_features]

param_grid = {'C': np.arange(1e-05, 3, 0.1)}
scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}

gs = GridSearchCV(LogisticRegression(), return_train_score=True,
                  param_grid=param_grid, scoring=scoring, cv=10, refit='Accuracy')

gs.fit(X, y)
results = gs.cv_results_

print('='*20)
print("best params: " + str(gs.best_estimator_))
print("best params: " + str(gs.best_params_))
print('best score:', gs.best_score_)
print('='*20)

plt.figure(figsize=(10, 10))
plt.title("GridSearchCV evaluating using multiple scorers simultaneously",fontsize=16)

plt.xlabel("Inverse of regularization strength: C")
plt.ylabel("Score")
plt.grid()

ax = plt.axes()
ax.set_xlim(0, param_grid['C'].max())
ax.set_ylim(0.35, 0.95)

# Get the regular numpy array from the MaskedArray
X_axis = np.array(results['param_C'].data, dtype=float)

for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
    for sample, style in (('train', '--'), ('test', '-')):
        sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if scoring[scorer]=='neg_log_loss' else results['mean_%s_%
        sample_score_std = results['std_%s_%s' % (sample, scorer)]
        ax.fill_between(X_axis, sample_score_mean - sample_score_std,
                        sample_score_mean + sample_score_std,
                        alpha=0.1 if sample == 'test' else 0, color=color)
        ax.plot(X_axis, sample_score_mean, style, color=color,
                alpha=1 if sample == 'test' else 0.7,
                label="%s (%s)" % (scorer, sample))

    best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
    best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss' else results['mean_test_%s' %

    # Plot a dotted vertical line at the best score for that scorer marked by x
    ax.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)

    # Annotate the best score for that scorer
    ax.annotate("%0.2f" % best_score,
                (X_axis[best_index], best_score + 0.005))

plt.legend(loc="best")
plt.grid('off')
plt.show()
```

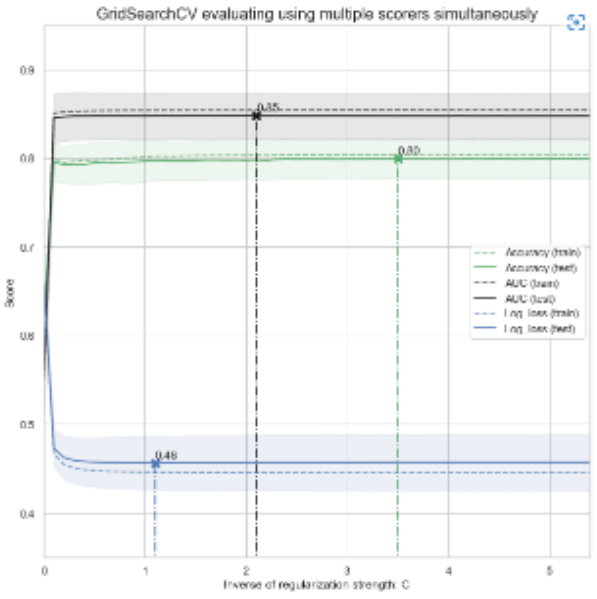
```
====================
best params: LogisticRegression(C=2.4000100000000004)
best params: {'C': 2.4000100000000004}
best score: 0.8069662921348316
====================
```



In [45]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.pipeline import Pipeline

#Define simple model
###############################################################################
C = np.arange(1e-05, 5.5, 0.1)
scoring = {'Accuracy': 'accuracy', 'AUC': 'roc_auc', 'Log_loss': 'neg_log_loss'}
log_reg = LogisticRegression()

#Simple pre-processing estimators
###############################################################################
std_scale = StandardScaler(with_mean=False, with_std=False)
#std_scale = StandardScaler()

#Defining the CV method: Using the Repeated Stratified K Fold
###############################################################################

n_folds=5
n_repeats=5

rskfold = RepeatedStratifiedKFold(n_splits=n_folds, n_repeats=n_repeats, random_state=2)

#Creating simple pipeline and defining the gridsearch
###############################################################################

log_clf_pipe = Pipeline(steps=[('scale',std_scale), ('clf',log_reg)])

log_clf = GridSearchCV(estimator=log_clf_pipe, cv=rskfold,
              scoring=scoring, return_train_score=True,
              param_grid=dict(clf__C=C), refit='Accuracy')

log_clf.fit(X, y)
results = log_clf.cv_results_

print('='*20)
print("best params: " + str(log_clf.best_estimator_))
print("best params: " + str(log_clf.best_params_))
print('best score:', log_clf.best_score_)
print('='*20)

plt.figure(figsize=(10, 10))
plt.title("GridSearchCV evaluating using multiple scorers simultaneously",fontsize=16)
```

```
plt.xlabel("Inverse of regularization strength: C")
plt.ylabel("Score")
plt.grid()

ax = plt.axes()
ax.set_xlim(0, C.max())
ax.set_ylim(0.35, 0.95)

# Get the regular numpy array from the MaskedArray
X_axis = np.array(results['param_clf__C'].data, dtype=float)

for scorer, color in zip(list(scoring.keys()), ['g', 'k', 'b']):
    for sample, style in (('train', '--'), ('test', '-')):
        sample_score_mean = -results['mean_%s_%s' % (sample, scorer)] if scoring[scorer]=='neg_log_loss' else results['mean_%s_%
        sample_score_std = results['std_%s_%s' % (sample, scorer)]
        ax.fill_between(X_axis, sample_score_mean - sample_score_std,
                        sample_score_mean + sample_score_std,
                        alpha=0.1 if sample == 'test' else 0, color=color)
        ax.plot(X_axis, sample_score_mean, style, color=color,
                alpha=1 if sample == 'test' else 0.7,
                label="%s (%s)" % (scorer, sample))

    best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
    best_score = -results['mean_test_%s' % scorer][best_index] if scoring[scorer]=='neg_log_loss' else results['mean_test_%s' %

    # Plot a dotted vertical line at the best score for that scorer marked by x
    ax.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)

    # Annotate the best score for that scorer
    ax.annotate("%0.2f" % best_score,
                (X_axis[best_index], best_score + 0.005))

plt.legend(loc="best")
plt.grid('off')
plt.show()
```

```
====================
best params: Pipeline(steps=[('scale', StandardScaler(with_mean=False, with_std=False)),
                ('clf', LogisticRegression(C=3.50001))])
best params: {'clf__C': 3.50001}
best score: 0.7995518172117255
====================
```



In [46]:
```
final_test['Survived'] = log_clf.predict(final_test[Selected_features])
final_test['PassengerId'] = test_df['PassengerId']

submission = final_test[['PassengerId','Survived']]

submission.to_csv("submission.csv", index=False)

submission.tail()
```

Out[46]:

	PassengerId	Survived
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

3.KNN Classification:

In [1]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
```

In [2]:
```
iris=datasets.load_iris()
```

In [3]:
```
x = iris.data
y = iris.target
```

In [4]:
```
print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)
```

```
sepal-length sepal-width petal-length petal-width
[[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]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
```

In [5]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
```

In [6]:
```
#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
```

Out[6]:
```
KNeighborsClassifier()
```

In [7]:
```
#To make predictions on our test data
y_pred=classifier.predict(x_test)
```

In [8]:
```
print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
Confusion Matrix
[[12  0  0]
 [ 0 12  1]
 [ 0  1 19]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.92      0.92      0.92        13
           2       0.95      0.95      0.95        20

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

Artificial Intelligence & Machine Learning

Experiment No. 7

Implementation of Dimensionality reduction techniques

Name: Sumit Bhatkar Roll No. 05

Aim : Write a programme to demonstrate dimensionality reduction techniques using PCA.

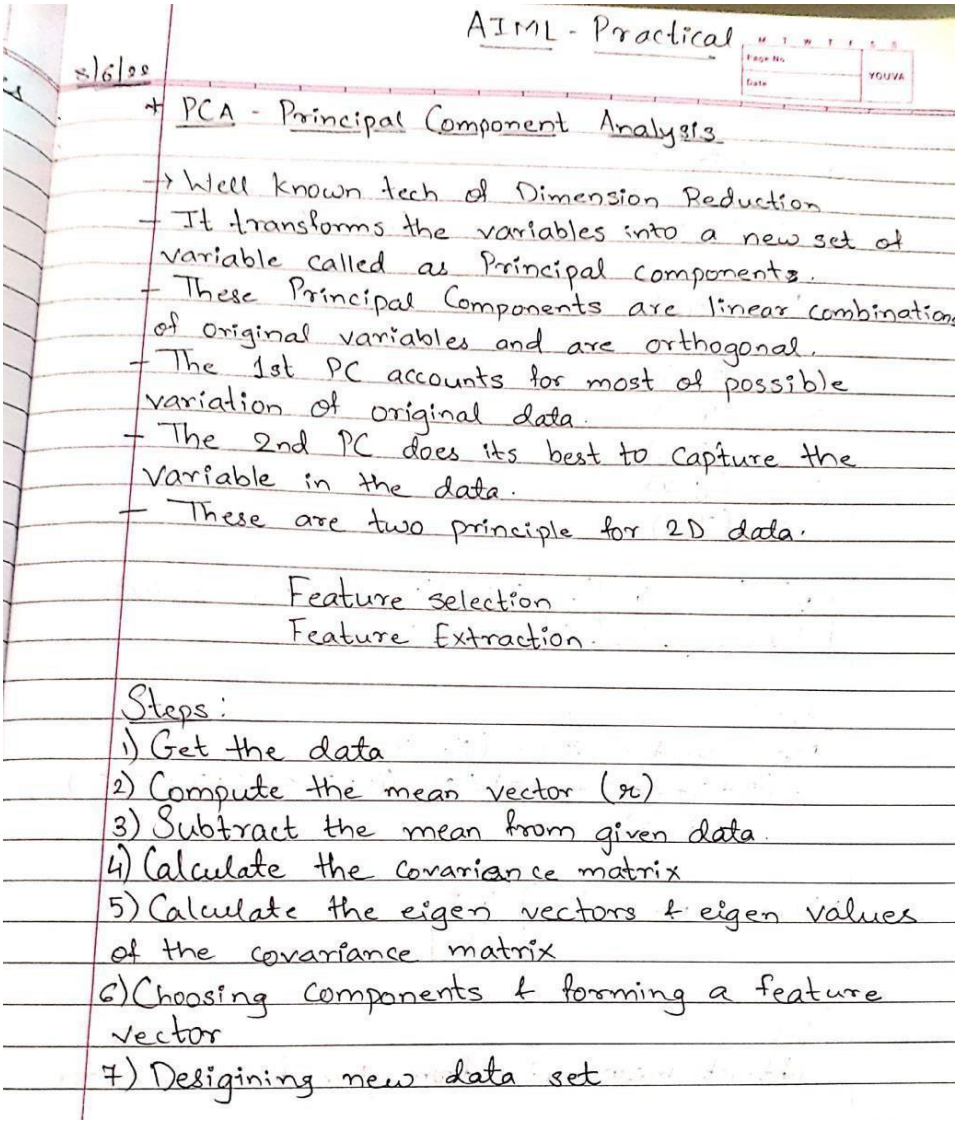
Objective: To learn dimensionality reduction techniques using PCA.

Software Requirement:

- **Anaconda Navigator:** Anaconda Navigator is a desktop graphical user interface included in Anaconda that allows you to launch applications and easily manage conda packages, environments and channels without the need to use command line commands.

Theory:

- **PCA: Principal component analysis** (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.



AIML - Practical

5/6/22

* PCA - Principal Component Analysis

→ Well known tech of Dimension Reduction
- It transforms the variables into a new set of variable called as Principal components.
- These Principal Components are linear combination of original variables and are orthogonal.
- The 1st PC accounts for most of possible variation of original data.
- The 2nd PC does its best to capture the variable in the data.
- These are two principle for 2D data.

Feature selection
Feature Extraction.

Steps:
1) Get the data
2) Compute the mean vector (x)
3) Subtract the mean from given data.
4) Calculate the covariance matrix
5) Calculate the eigen vectors & eigen values of the covariance matrix
6) Choosing components & forming a feature vector
7) Designing new data set

Given data:

2D data

(2,1), (3,5), (4,3), (5,6), (6,7), (7,8)

Soln: Step 1:

$X_1 = (2,1)$

$X_2 = (3,5)$

$X_3 = (4,3)$

$X_4 = (5,6)$

$X_5 = (6,7)$

$X_6 = (7,8)$

$\begin{bmatrix}2\\1\end{bmatrix}\begin{bmatrix}3\\5\end{bmatrix}\begin{bmatrix}4\\3\end{bmatrix}\begin{bmatrix}5\\6\end{bmatrix}\begin{bmatrix}6\\7\end{bmatrix}\begin{bmatrix}7\\8\end{bmatrix}$ → feature vector

Step 2:

Mean vector $(r) = (X_1, X_2)$

$= ((2+3+4+5+6+7)/6), ((1+5+3+6+7+8)/6))$

$= (4.5, 5)$

Mean vector $(r) = \begin{bmatrix}4.5\\5\end{bmatrix}$

Step 3:

Subtract mean vector from given feature vector

$X_1 - r = (2-4.5, 1-5) = (-2.5, -4)$

$X_2 - r = (3-4.5, 5-5) = (-1.5, 0)$

$X_3 - r = (4-4.5, 3-5) = (-0.5, -2)$

$X_4 - r = (5-4.5, 6-5) = (0.5, 1)$

$X_5 - r = (6-4.5, 7-5) = (1.5, 2)$

$X_6 - r = (7-4.5, 8-5) = (2.5, 3)$

Feature vector (X_i) after subtracting mean vector (r) are

$$\begin{bmatrix}-2.5\\-4\end{bmatrix}\begin{bmatrix}-1.5\\0\end{bmatrix}\begin{bmatrix}-0.5\\-2\end{bmatrix}\begin{bmatrix}0.5\\1\end{bmatrix}\begin{bmatrix}1.5\\2\end{bmatrix}\begin{bmatrix}2.5\\3\end{bmatrix}$$

Step 4:

Calculate Covariance matrix

Covariance matrix = $\frac{\Sigma (X_i - r)(X_i - r)}{n}$

X	$X-\bar{X}$	$(x-\bar{x})(x-\bar{x})$	Y	$Y-\bar{Y}$	$(Y-\bar{Y})(Y-\bar{Y})$
2	-2.5	6.25	1	-4	16
3	-1.5	2.25	5	0	0
4	-0.5	0.25	3	-2	4
5	0.5	0.25	6	1	1
6	1.5	2.25	7	2	4
7	2.5	6.25	8	3	9
		$\Sigma(X-\bar{X})^2 = 17.5$			$\Sigma(Y-\bar{Y})^2 = 34$

X	Y	$X-\bar{X}$	$Y-\bar{Y}$	$(x-\bar{x}).(Y-\bar{Y})$
2	1	-2.5	-4	10
3	5	-1.5	0	0
4	3	-0.5	-2	1
5	6	-0.5	1	0.5
6	7	-1.5	2	3
7	8	2.5	3	7.5
				$\Sigma(X-\bar{X}).(Y-\bar{Y}) = 22$

Covariance matrix $= \frac{M_1 + M_2 + M_3 + M_4 + M_5 + M_6}{6}$

on addition of above matrices

$$\text{Covariance matrix} = \frac{1}{6}\begin{bmatrix} (x-\bar{x})^2 & (x-\bar{x})(y-\bar{y}) \\ (x-\bar{x})(y-\bar{y}) & (y-\bar{y})^2 \end{bmatrix}$$

$$= \frac{1}{6}\begin{bmatrix} 17.5 & 22 \\ 22 & 34 \end{bmatrix}$$

$$= \begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix}$$

Step 5:

Calculate Eigen value & Eigen vectors of the covariance matrix

λ is the Eigen value of matrix M if it is a solution of characteristic equation

$$|M - \lambda| = 0$$

from here $= (2.92 - \lambda)(5.67 - \lambda) - (3.67 - \lambda)(3.67 - \lambda) = 0$

$$(16.56 - 2.92\lambda - 5.67\lambda + \lambda^2 - 13.47) = 0$$

$$\lambda^2 - 8.59\lambda + 3.09 = 0$$

Solving this quadratic equation

$$\lambda = 8.22, 0.38$$

$$\lambda_1 = 8.22 \quad , \quad \lambda_2 = 0.38$$

Eigen vector corresponding to the greatest eigen value is the principle component for given data set, so, we find the eigen vector corresponding to eigen value λ_1.

We use eqn $MX = \lambda X$

M = Covariance matrix

X = Eigen vector

λ = Eigen value

$$2.92X_1 + 3.67X_2 = 8.22X_1$$
$$3.67X_1 + 5.67X_2 = 8.22X_2$$

By simplifying eqn

$$5.3X_1 = 3.67X_2 \quad — (1)$$
$$3.67X_1 = 2.55X_2 \quad — (2)$$

from (1) & (2)

$$X_1 = 0.69X_2$$

$$\text{Eigen vector} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$$

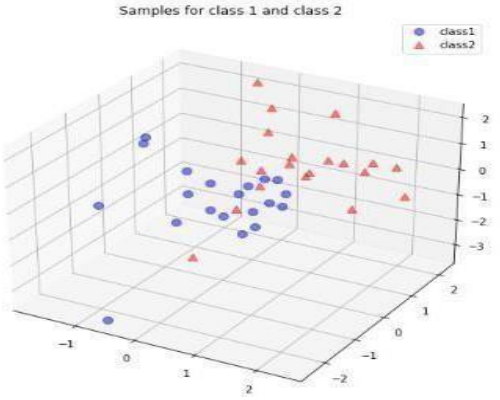
Code & Output:

feature_selection_PCA

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

#np.random.seed(23423478238423978) # random seed for consistency

# A reader pointed out that Python 2.7 would raise a
# "ValueError: object of too small depth for desired array".
# This can be avoided by choosing a smaller random seed, e.g. 1
# or by completely omitting this line, since I just used the random seed for
# consistency.

mu_vec1 = np.array([0,0,0])
cov_mat1 = np.array([[1,0,0],[0,1,0],[0,0,1]])
class1_sample = np.random.multivariate_normal(mu_vec1, cov_mat1, 20).T
assert class1_sample.shape == (3,20), "The matrix has not the dimensions 3x20"

mu_vec2 = np.array([1,1,1])
cov_mat2 = np.array([[1,0,0],[0,1,0],[0,0,1]])
class2_sample = np.random.multivariate_normal(mu_vec2, cov_mat2, 20).T
assert class2_sample.shape == (3,20), "The matrix has not the dimensions 3x20"
```

In [4]:
```
%pylab inline
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d

fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
plt.rcParams['legend.fontsize'] = 10
ax.plot(class1_sample[0,:], class1_sample[1,:], class1_sample[2,:], 'o', markersize=8, color='blue', alpha=0.5, label='class1')
ax.plot(class2_sample[0,:], class2_sample[1,:], class2_sample[2,:], '^', markersize=8, alpha=0.5, color='red', label='class2')

plt.title('Samples for class 1 and class 2')
ax.legend(loc='upper right')

plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



In [5]:
```
all_samples = np.concatenate((class1_sample, class2_sample), axis=1)
assert all_samples.shape == (3,40), "The matrix has not the dimensions 3x40"
```

In [6]:
```
mean_x = np.mean(all_samples[0,:])
mean_y = np.mean(all_samples[1,:])
mean_z = np.mean(all_samples[2,:])

mean_vector = np.array([[mean_x],[mean_y],[mean_z]])
```

In [7]:
```
scatter_matrix = np.zeros((3,3))
for i in range(all_samples.shape[1]):
    scatter_matrix += (all_samples[:,i].reshape(3,1) - mean_vector).dot((all_samples[:,i].reshape(3,1) - mean_vector).T)
print('Scatter Matrix:\n', scatter_matrix)
```

```
Scatter Matrix:
 [[33.71266861 10.08651371  8.12777968]
 [10.08651371 48.14712763 18.31418846]
 [ 8.12777968 18.31418846 53.04123406]]
```

In [8]:
```
cov_mat = np.cov([all_samples[0,:],all_samples[1,:],all_samples[2,:]])
print('Covariance Matrix:\n', cov_mat)
```

```
Covariance Matrix:
 [[0.8644274  0.25862856 0.20840461]
 [0.25862856 1.23454173 0.46959458]
 [0.20840461 0.46959458 1.36003164]]
```

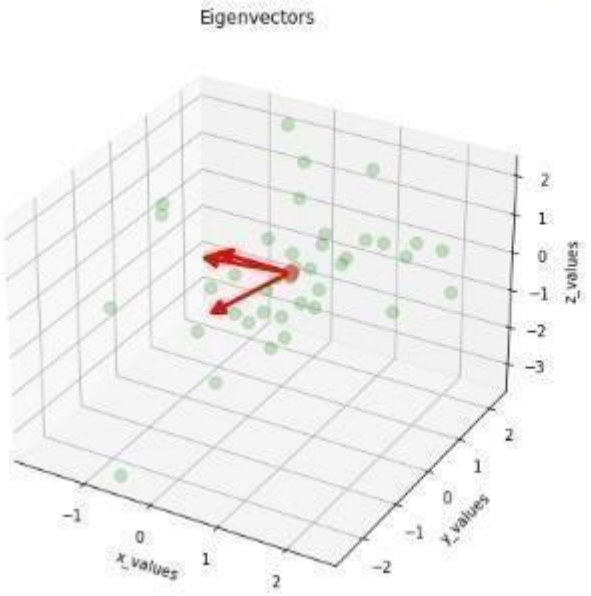
In [9]:
```
# eigenvectors and eigenvalues for the from the scatter matrix
eig_val_sc, eig_vec_sc = np.linalg.eig(scatter_matrix)

# eigenvectors and eigenvalues for the from the covariance matrix
eig_val_cov, eig_vec_cov = np.linalg.eig(cov_mat)

for i in range(len(eig_val_sc)):
    eigvec_sc = eig_vec_sc[:,i].reshape(1,3).T
    eigvec_cov = eig_vec_cov[:,i].reshape(1,3).T
    assert eigvec_sc.all() == eigvec_cov.all(), 'Eigenvectors are not identical'

    print('Eigenvector {}: \n{}'.format(i+1, eigvec_sc))
    print('Eigenvalue {} from scatter matrix: {}'.format(i+1, eig_val_sc[i]))
    print('Eigenvalue {} from covariance matrix: {}'.format(i+1, eig_val_cov[i]))
    print('Scaling factor: ', eig_val_sc[i]/eig_val_cov[i])
    print(40 * '-')
```

```
Eigenvector 1:
[[-0.30824235]
 [-0.63907963]
 [-0.70467289]]
Eigenvalue 1 from scatter matrix: 73.20598118090528
Eigenvalue 1 from covariance matrix: 1.8770764405360343
Scaling factor:  38.99999999999997
----------------------------------------
Eigenvector 2:
[[-0.82885341]
 [ 0.54396773]
 [-0.13077128]]
Eigenvalue 2 from scatter matrix: 28.37534621609244
Eigenvalue 2 from covariance matrix: 0.7275729798998062
Scaling factor:  39.0
----------------------------------------
Eigenvector 3:
[[-0.46689258]
 [-0.54376128]
 [ 0.69737722]]
Eigenvalue 3 from scatter matrix: 33.31970289768161
Eigenvalue 3 from covariance matrix: 0.8543513563508106
Scaling factor:  38.99999999999999
----------------------------------------
```

In [10]:
```
for i in range(len(eig_val_sc)):
    eigv = eig_vec_sc[:,i].reshape(1,3).T
    np.testing.assert_array_almost_equal(scatter_matrix.dot(eigv), eig_val_sc[i] * eigv,
                                         decimal=6, err_msg='', verbose=True)
```

In [11]:
```
%pylab inline

from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d
from matplotlib.patches import FancyArrowPatch


class Arrow3D(FancyArrowPatch):
    def __init__(self, xs, ys, zs, *args, **kwargs):
        FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
        self._verts3d = xs, ys, zs

    def draw(self, renderer):
        xs3d, ys3d, zs3d = self._verts3d
        xs, ys, zs = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
        self.set_positions((xs[0],ys[0]),(xs[1],ys[1]))
        FancyArrowPatch.draw(self, renderer)

fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(111, projection='3d')

ax.plot(all_samples[0,:], all_samples[1,:], all_samples[2,:], 'o', markersize=8, color='green', alpha=0.2)
ax.plot([mean_x], [mean_y], [mean_z], 'o', markersize=10, color='red', alpha=0.5)
for v in eig_vec_sc.T:
    a = Arrow3D([mean_x, v[0]], [mean_y, v[1]], [mean_z, v[2]], mutation_scale=20, lw=3, arrowstyle="-|>", color="r")
    ax.add_artist(a)
ax.set_xlabel('x_values')
ax.set_ylabel('y_values')
ax.set_zlabel('z_values')

plt.title('Eigenvectors')

plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



In [12]:
```
for ev in eig_vec_sc:
    numpy.testing.assert_array_almost_equal(1.0, np.linalg.norm(ev))
    # instead of 'assert' because of rounding errors
```

In [13]:
```
# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_val_sc[i]), eig_vec_sc[:,i]) for i in range(len(eig_val_sc))]

# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# Visually confirm that the list is correctly sorted by decreasing eigenvalues
for i in eig_pairs:
    print(i[0])
```

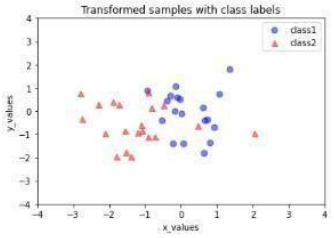
```
73.20598118090528
33.31970289768161
28.37534621609244
```

In [14]:
```
matrix_w = np.hstack((eig_pairs[0][1].reshape(3,1), eig_pairs[1][1].reshape(3,1)))
print('Matrix W:\n', matrix_w)
```

```
Matrix W:
 [[-0.30824235 -0.46689258]
 [-0.63907963 -0.54376128]
 [-0.70467289  0.69737722]]
```

In [15]:
```
transformed = matrix_w.T.dot(all_samples)
assert transformed.shape == (2,40), "The matrix is not 2x40 dimensional."
```

In [16]:
```
plt.plot(transformed[0,0:20], transformed[1,0:20], 'o', markersize=7, color='blue', alpha=0.5, label='class1')
plt.plot(transformed[0,20:40], transformed[1,20:40], '^', markersize=7, color='red', alpha=0.5, label='class2')
plt.xlim([-4,4])
plt.ylim([-4,4])
plt.xlabel('x_values')
plt.ylabel('y_values')
plt.legend()
plt.title('Transformed samples with class labels')

plt.show()
```



feature_extraction_PCA

In [18]:
```
# Import packages
import numpy as np
from sklearn import decomposition, datasets
from sklearn.preprocessing import StandardScaler
```

In [19]:
```
# Load the breast cancer dataset
dataset = datasets.load_breast_cancer()

# Load the features
X = dataset.data
```

In [20]:
```
# View the shape of the dataset
X.shape
```

Out[20]:
```
(569, 30)
```

In [21]:
```
# View the data
X
```

Out[21]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

In [22]:
```
# Create a scaler object
sc = StandardScaler()

# Fit the scaler to the features and transform
X_std = sc.fit_transform(X)
```

In [23]:
```
# Create a pca object with the 2 components as a parameter
pca = decomposition.PCA(n_components=2)

# Fit the PCA and transform the data
X_std_pca = pca.fit_transform(X_std)
```

In [24]:
```
# View the new feature data's shape
X_std_pca.shape
```

Out[24]:
```
(569, 2)
```

In [25]:
```
# View the new feature data
X_std_pca
```

Out[25]:
```
array([[ 9.19283683,  1.94858307],
       [ 2.3878018 , -3.76817174],
       [ 5.73389628, -1.0751738 ],
       ...,
       [ 1.25617928, -1.90229671],
       [10.37479406,  1.67201011],
       [-5.4752433 , -0.67063679]])
```

Artificial Intelligence & Machine Learning

Experiment No. 8

Implementation of clustering algorithm.

Name: Sumit Bhatkar Roll No. 05

1. Aim: Write a program to implement classification of iris dataset using K-Means clustering algorithm with K=3.

Objective: Understanding K-Means clustering algorithm.

Software Requirement:

- **Anaconda Navigator:** Anaconda Navigator is a desktop graphical user interface included in Anaconda that allows you to launch applications and easily manage conda packages, environments and channels without the need to use command line commands.

Theory:

- **K-Means Clustering : *k*-means clustering** is a method of vector quantization, originally from signal processing, that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster.

Output & Code:

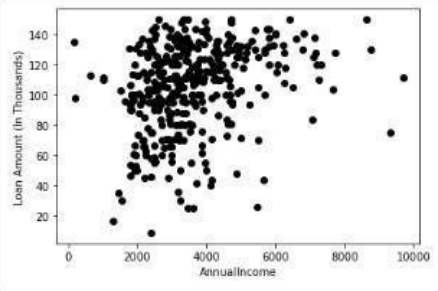
In [1]:
```
#import libraries
import pandas as pd
import numpy as np
import random as rd
import matplotlib.pyplot as plt
```

In [2]:
```
data = pd.read_csv('clustering.csv')
data.head()
```

Out[2]:

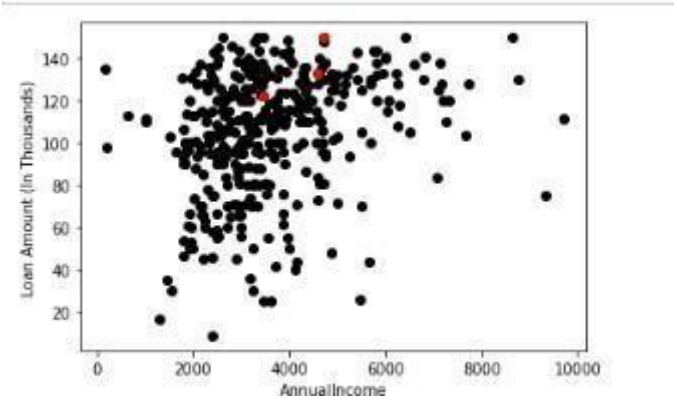
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
1	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
2	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
3	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
4	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0	1.0

In [3]:
```
X = data[["LoanAmount","ApplicantIncome"]]
#Visualise data points
plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')
plt.xlabel('AnnualIncome')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```



Steps 1 and 2 of K-Means were about choosing the number of clusters (k) and selecting random centroids for each cluster. We will pick 3 clusters and then select random observations from the data as the centroids:

In [4]:
```
K=3

# Select random observation as centroids
Centroids = (X.sample(n=K))
plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')
plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')
plt.xlabel('AnnualIncome')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```



In [5]:
```
Centroids
```

Out[5]:

	LoanAmount	ApplicantIncome
131	100.0	3850
344	102.0	3017
332	110.0	3775

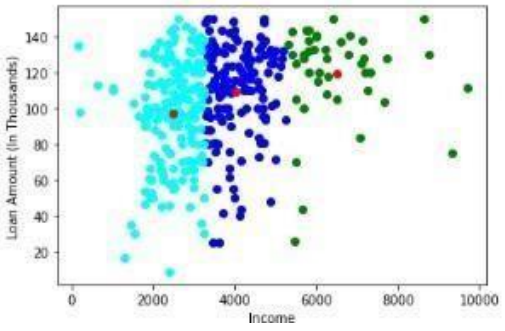
In [5]:
```
# Step 3 - Assign all the points to the closest cluster centroid
# Step 4 - Recompute centroids of newly formed clusters
# Step 5 - Repeat step 3 and 4

diff = 1
j=0

while(diff!=0):
    XD=X
    i=1
    for index1,row_c in Centroids.iterrows():
        ED=[]
        for index2,row_d in XD.iterrows():
            d1=(row_c["ApplicantIncome"]-row_d["ApplicantIncome"])**2
            d2=(row_c["LoanAmount"]-row_d["LoanAmount"])**2
            d=np.sqrt(d1+d2)
            ED.append(d)
        X[i]=ED
        i=i+1

    C=[]
    for index,row in X.iterrows():
        min_dist=row[1]
        pos=1
        for i in range(K):
            if row[i+1] < min_dist:
                min_dist = row[i+1]
                pos=i+1
        C.append(pos)
    X["Cluster"]=C
    Centroids_new = X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]
    if j == 0:
        diff=1
        j=j+1
    else:
        diff = (Centroids_new['LoanAmount'] - Centroids['LoanAmount']).sum() + (Centroids_new['ApplicantIncome'] - Centroids['Ap
        print(diff.sum())
    Centroids = X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]
```

```
<ipython-input-5-912b3f05973f>:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[i]=ED
<ipython-input-5-912b3f05973f>:30: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X["Cluster"]=C
```

```
251.4174066524006
-2.482417068691774
-104.64723507412567
-46.7501216812597
-46.4934676023523
-55.03495831127759
-9.190752402517077
-9.19844100901777
-9.237706177129652
0.0
```

When this difference is 0, we are stopping the training. Let's now visualize the clusters we have got:

In [6]:
```
color=['blue','green','cyan']
for k in range(K):
    data=X[X["Cluster"]==k+1]
    plt.scatter(data["ApplicantIncome"],data["LoanAmount"],c=color[k])
plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')
plt.xlabel('Income')
plt.ylabel('Loan Amount (In Thousands)')
plt.show()
```



2. Aim: Write a program to demonstrate K-Medoid Algorithm with K=3.

Objective: Understanding K-Means and K-medoid clustering algorithm.

Software Requirement:

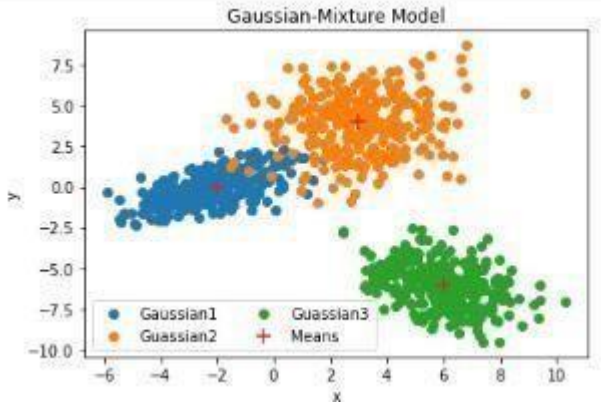
- **Anaconda Navigator:** Anaconda Navigator is a desktop graphical user interface included in Anaconda that allows you to launch applications and easily manage conda packages, environments and channels without the need to use command line commands.

Theory:

- **K-Medoid:** k -medoids is **a classical partitioning technique of clustering that splits the data set of n objects into k clusters, where the number k of clusters assumed known a priori** (which implies that the programmer must specify k before the execution of a k -medoids algorithm).

Output & Code:

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import sys
import random


mean1 = [-2, 0]
cov1 = [[2, 0.9], [0.9, 1]]
data1 = np.random.multivariate_normal(mean1, cov1, 300)

mean2 = [3, 4]
cov2 = [[3, 0.1], [0.1, 3]]
data2 = np.random.multivariate_normal(mean2, cov2, 300)

mean3 = [6, -6]
cov3 = [[2, -1], [-1, 2]]
data3 = np.random.multivariate_normal(mean3, cov3, 300)

data = np.concatenate((data1, data2, data3), axis=0)
centers = np.asarray([mean1, mean2, mean3])

# Plot
fig1 = plt.scatter(data1[:, 0], data1[:, 1])
fig2 = plt.scatter(data2[:, 0], data2[:, 1])
fig3 = plt.scatter(data3[:, 0], data3[:, 1])
fig4 = plt.scatter(centers[:, 0], centers[:, 1], marker="+", s=100)
# plt.scatter(data[:, 0], data[:, 1])
plt.title('Gaussian-Mixture Model')
plt.xlabel('x')
plt.ylabel('y')
plt.legend((fig1, fig2, fig3, fig4),
           ('Gaussian1', 'Guassian2', 'Guassian3', 'Means'),
           scatterpoints=1,
           loc='lower left',
           ncol=2,
           fontsize=10)
plt.savefig('before_k_medoids.png')
plt.show()
```



In [5]:
```
# construct the similarity matrix
num = len(data)
similarity_matrix = np.zeros((num, num))

for i in range(0, num):
    for j in range(i+1, num):
        diff = data[i] - data[j]
        dist_tmp = np.linalg.norm(diff)
        similarity_matrix[i][j] = dist_tmp
        similarity_matrix[j][i] = dist_tmp

similarity_matrix = torch.from_numpy(similarity_matrix)
```

In [6]:
```
def k_medoids(similarity_matrix, k):

    # Step 1: Select initial medoids
    num = len(similarity_matrix)
    row_sums = torch.sum(similarity_matrix, dim=1)
    normalized_sim = similarity_matrix.T / row_sums
    normalized_sim = normalized_sim.T
    priority_scores = -torch.sum(normalized_sim, dim=0)
    values, indices = priority_scores.topk(k)

    tmp = -similarity_matrix[:, indices]
    tmp_values, tmp_indices = tmp.topk(1, dim=1)
    min_distance = -torch.sum(tmp_values)
    cluster_assignment = tmp_indices.resize_(num)
    print(min_distance)

    # Step 2: Update medoids
    for i in range(k):
        sub_indices = (cluster_assignment == i).nonzero()
        sub_num = len(sub_indices)
        sub_indices = sub_indices.resize_(sub_num)
        sub_similarity_matrix = torch.index_select(similarity_matrix, 0, sub_indices)
        sub_similarity_matrix = torch.index_select(sub_similarity_matrix, 1, sub_indices)
        sub_row_sums = torch.sum(sub_similarity_matrix, dim=1)
        sub_medoid_index = torch.argmin(sub_row_sums)
        # update the cluster medoid index
        indices[i] = sub_indices[sub_medoid_index]

    # Step 3: Assign objects to medoids
    tmp = -similarity_matrix[:, indices]
    tmp_values, tmp_indices = tmp.topk(1, dim=1)
    total_distance = -torch.sum(tmp_values)
    cluster_assignment = tmp_indices.resize_(num)
    print(total_distance)
```

```
    while (total_distance < min_distance):
        min_distance = total_distance
        # Step 2: Update medoids
        for i in range(k):
            sub_indices = (cluster_assignment == i).nonzero()
            sub_num = len(sub_indices)
            sub_indices = sub_indices.resize_(sub_num)
            sub_similarity_matrix = torch.index_select(similarity_matrix, 0, sub_indices)
            sub_similarity_matrix = torch.index_select(sub_similarity_matrix, 1, sub_indices)
            sub_row_sums = torch.sum(sub_similarity_matrix, dim=1)
            sub_medoid_index = torch.argmin(sub_row_sums)
            # update the cluster medoid index
            indices[i] = sub_indices[sub_medoid_index]

        # Step 3: Assign objects to medoids
        tmp = -similarity_matrix[:, indices]
        tmp_values, tmp_indices = tmp.topk(1, dim=1)
        total_distance = -torch.sum(tmp_values)
        cluster_assignment = tmp_indices.resize_(num)
        print(total_distance)

    return indices
```

In [7]:
```
indices = k_medoids(similarity_matrix, k=3)
medoids = []
for i in range(3):
    medoids.append(data[indices[i]])

medoids = np.asarray(medoids)
fig1 = plt.scatter(data1[:, 0], data1[:, 1])
fig2 = plt.scatter(data2[:, 0], data2[:, 1])
fig3 = plt.scatter(data3[:, 0], data3[:, 1])
fig4 = plt.scatter(centers[:, 0], centers[:, 1], marker="+", s=200)
fig5 = plt.scatter(medoids[:, 0], medoids[:, 1], marker="*", s=200)
plt.title('Gaussian-Mixture Model')
plt.xlabel('x')
plt.ylabel('y')
plt.legend((fig1, fig2, fig3, fig4, fig5),
           ('Gaussian1', 'Guassian2', 'Guassian3', 'Means', 'Medoids'),
           scatterpoints=1,
           loc='lower left',
           ncol=2,
           fontsize=10)
plt.savefig('after_k_medoids.png')
plt.show()
```

```
tensor(4639.0377, dtype=torch.float64)
tensor(1657.7306, dtype=torch.float64)
tensor(1599.6259, dtype=torch.float64)
tensor(1598.3199, dtype=torch.float64)
tensor(1598.3199, dtype=torch.float64)
```