```
#ML Class 2
import numpy as np
import pandas as pd
```

```
import numpy as np
dt = np.array(["red","blue","white","black"])
st = pd.Series(dt)
print("Array:",dt)
print(st)
```

```
Array: ['red' 'blue' 'white' 'black']
0      red
1     blue
2    white
3    black
dtype: object
```

```
dt = {'a' : 0 , 'a1' : 3, 'a2' : 7, 'a3' : 2}
s = pd.Series(dt)
s
```

```
a     0
a1    3
a2    7
a3    2
dtype: int64
```

```
s.values
```

```
array([0, 3, 7, 2])
```

```
s.index.values
```

```
array(['a', 'a1', 'a2', 'a3'], dtype=object)
```

```
print(s['a2'])
print(st[3])
```

```
7
black
```

```
s['a3']=5
s
```

```
a     0
a1    3
a2    7
a3    5
dtype: int64
```

```
#Creating a Data Frame
data = {'colors' : ["red", "blue","white","black"], 'Availability': [0,7,2,1]}
```

```
stock=pd.DataFrame(data)
stock
```

	colors	Availability
0	red	0
1	blue	7
2	white	2
3	black	1

```
stock=pd.DataFrame(data, index=['a','a1','a2','a3'])
stock
```

	colors	Availability
a	red	0
a1	blue	7
a2	white	2
a3	black	1

```
stock.loc[['a1']]
```

	colors	Availability
a1	blue	7

```
print(stock.loc['a1':'a3',['colors']])
stock.loc[:,['colors']]
```

```
    colors
a1   blue
a2  white
a3  black
```

	colors
a	red
a1	blue
a2	white
a3	black

```
print(stock.iloc[1,1])
```

```
7
```

```
print(stock.iloc[:,1])
```

```
a     0
a1    7
a2    2
a3    1
Name: Availability, dtype: int64
```

```
print(stock.iloc[:-1])
```

```
    colors  Availability
a     red             0
a1   blue             7
a2  white             2
```

```
print(stock.iloc[:,-1])
```

```
a     0
a1    7
a2    2
a3    1
Name: Availability, dtype: int64
```

```
print(stock.iloc[[-1]])
```

```
    colors  Availability
a3  black             1
```

```
'''
Merging Dataframes...
'''
```

```
'\nMerging Dataframes...\n'
```

```
item={'Item':["Ball","Book","Pen","Chair"],'Price':[10,50,5,200],'colors':["red","blue","white","black"]}
item_data=pd.DataFrame(item)
item_data
```

	Item	Price	colors
0	Ball	10	red
1	Book	50	blue
2	Pen	5	white
3	Chair	200	black

```
merge1=pd.merge(item_data,stock,on='colors')
merge1
```

	Item	Price	colors	Availability
0	Ball	10	red	0
1	Book	50	blue	7
2	Pen	5	white	2
3	Chair	200	black	1

```
con=pd.concat([item_data,stock],axis=1)
con
```

	Item	Price	colors	colors	Availability
0	Ball	10.0	red	NaN	NaN
1	Book	50.0	blue	NaN	NaN
2	Pen	5.0	white	NaN	NaN
3	Chair	200.0	black	NaN	NaN
a	NaN	NaN	NaN	red	0.0
a1	NaN	NaN	NaN	blue	7.0
a2	NaN	NaN	NaN	white	2.0
a3	NaN	NaN	NaN	black	1.0

```
'''
Import data frame from a file
'''
```

```
'\nImport data frame from a file\n'
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
import pandas as pd
path="/content/drive/MyDrive/Salary_Data.csv"

data=pd.read_csv("path")
data.describe
```

```
Mounted at /content/drive
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-61-1609a196f00d> in <cell line: 6>()
      4 path="/content/drive/MyDrive/Salary_Data.csv"
      5
----> 6 data=pd.read_csv("path")
      7 data.describe

                                 6 frames
/usr/local/lib/python3.10/dist-packages/pandas/io/common.py in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors,
storage_options)
    854         if ioargs.encoding and "b" not in ioargs.mode:
    855             # Encoding
--> 856             handle = open(
    857                 handle,
    858                 ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory: 'path'
```

```
data.head()
```

```
import pandas as pd
low_memory=False
```

```
ecom=pd.read_csv("/content/Ecommerce Purchases")
```

```
ecom.head()
```

	Address	Lot	AM or PM	Browser Info	Company	Credit Card	CC Exp Date	CC Security Code	CC Provider	Email	
0	16629 Pace Camp Apt. 448\nAlexisborough, NE 77...	46 in	PM	Opera/9.56. (X11; Linux x86_64; sl-SI) Presto/2...	Martinez-Herman	6011929061123406	02/20	900	JCB 16 digit	pdunlap@yahoo.com	Scie product/pr develop
1	9374 Jasmine Spurs Suite 508\nSouth John, TN 8...	28 rn	PM	Opera/8.93. (Windows 98; Win 9x 4.90; en-US) Pr...	Fletcher, Richards and Whitaker	3337758169645356	11/18	561	Mastercard	anthony41@reed.com	Dı eng
2	Unit 0065 Box 5052\nDPO AP 27450	94 vE	PM	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...	Simpson, Williams and Pham	675957666125	08/19	699	JCB 16 digit	amymiller@morales-harrison.com	Cust service ma
3	7780 Julia Fords\nNew Stacy, WA 45798	36 vm	PM	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_0 ...	Williams, Marshall and Buchanan	6011578504430710	02/24	384	Discover	brent16@olson-robinson.info	Dı eng
				Opera/9.58.							

```
ecom.head(10)
```

	Address	Lot	AM or PM	Browser Info	Company	Credit Card	CC Exp Date	CC Security Code	CC Provider	Ema
0	16629 Pace Camp Apt. 448\nAlexisborough, NE 77...	46 in	PM	Opera/9.56.(X11; Linux x86_64; sl-SI) Presto/2...	Martinez-Herman	6011929061123406	02/20	900	JCB 16 digit	pdunlap@yahoo.co
1	9374 Jasmine Spurs Suite 508\nSouth John, TN 8...	28 rn	PM	Opera/8.93. (Windows 98; Win 9x 4.90; en-US) Pr...	Fletcher, Richards and Whitaker	3337758169645356	11/18	561	Mastercard	anthony41@reed.co
2	Unit 0065 Box 5052\nDPO AP 27450	94 vE	PM	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...	Simpson, Williams and Pham	675957666125	08/19	699	JCB 16 digit	amymiller@morale harrison.co
3	7780 Julia Fords\nNew Stacy, WA 45798	36 vm	PM	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_0 ...	Williams, Marshall and Buchanan	6011578504430710	02/24	384	Discover	brent16@olson-robinson.in
4	23012 Munoz Drive Suite 337\nNew Cynthia, TX 5...	20 IE	AM	Opera/9.58.(X11; Linux x86_64; it-IT) Presto/2...	Brown, Watson and Andrews	6011456623207998	10/25	678	Diners Club / Carte Blanche	christopherwright@gmail.co
5	7502 Powell Mission Apt. 768\nTravisland, VA 3...	21 XT	PM	Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10_8_5...	Silva-Anderson	30246185196287	07/25	7169	Discover	ynguyen@gmail.co
6	93971 Conway Causeway\nAndersonburgh, AZ 75107	96 Xt	AM	Mozilla/5.0 (compatible; MSIE 7.0; Windows NT ...	Gibson and Sons	6011398782655569	07/24	714	VISA 16 digit	olivia04@yahoo.co
7	260 Rachel Plains Suite 366\nCastroberg, WV 24...	96 pG	PM	Mozilla/5.0 (X11; Linux i686) AppleWebKit/5350...	Marshall-Collins	561252141909	06/25	256	VISA 13 digit	phillip48@parks.in
8	2129 Dylan Burg\nNew Michelle, ME 28650	45 JN	PM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7...	Galloway and Sons	180041795790001	04/24	899	JCB 16 digit	kdavis@rasmussen.co
9	3795 Dawson Extensions\nLake Tinafort, ID 88739	15 Ug	AM	Mozilla/5.0 (X11; Linux i686; rv:1.9.7.20) Gec...	Rivera, Buchanan and Ramirez	4396283918371	01/17	931	American Express	qcoleman@hunt-huerta.co

```
ecom.tail(10)
```

	Address	Lot	AM or PM	Browser Info	Company	Credit Card	CC Exp Date	CC Security Code	CC Provider	Email
9990	75731 Molly Springs\nWest Danielle, VT 96934-5102	93 ty	PM	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4;...	Pace, Vazquez and Richards	869968197049750	04/24	877	JCB 15 digit	andersonmichael@sherman.biz
9991	PSC 8165, Box 8498\nAPO AP 60327-0346	50 dA	AM	Mozilla/5.0 (compatible; MSIE 8.0; Windows NT ...	Snyder Inc	4221582137197481	02/24	969	Voyager	kking@wise-liu.com
9992	885 Allen Mountains Apt. 230\nWallhaven, LA 16995	40 vH	PM	Mozilla/5.0 (Macintosh; PPC Mac OS X 10_6_5) A...	Wells Ltd	4664825258997302	10/20	431	Discover	bberry@wright.net
9993	7555 Larson Locks Suite 229\nEllisburgh, MA 34...	72 jg	PM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_8...	Colon and Sons	30025560104631	10/25	629	Maestro	chelseawilliams@lopez.biz
9994	6276 Rojas Hollow\nLake Louis, WY 56410-7837	93 Ex	PM	Opera/9.68.(X11; Linux x86_64; sl-SI) Presto/2...	Ritter-Smith	3112186784121077	01/25	1823	Maestro	iroberts@gmail.com
9995	966 Castaneda Locks\nWest Juliafurt, CO 96415	92 XI	PM	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/5352 ...	Randall-Sloan	342945015358701	03/22	838	JCB 15 digit	iscott@wade-garner.com
9996	832 Curtis Dam Suite 785\nNorth Edwardburgh, T...	41 JY	AM	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...	Hale, Collins and Wilson	210033169205009	07/25	207	JCB 16 digit	mary85@hotmail.com
9997	Unit 4434 Box 6343\nDPO AE 28026-0283	74 Zh	AM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7...	Anderson Ltd	6011539787356311	05/21	1	VISA 16 digit	tyler16@gmail.com
9998	0096 English Rest\nRoystad, IA 12457	74 cL	PM	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_8;...	Cook Inc	180003348082930	11/17	987	American Express	elizabethmoore@reid.net
9999	40674 Barrett Stravenue\nGrimesville, WI 79682	64 Hr	AM	Mozilla/5.0 (X11; Linux i686; rv:1.9.5.20) Gec...	Greene Inc	4139972901927273	02/19	302	JCB 15 digit	rachelford@vaughn.com

```
ecom.dtypes
```

```
Address             object
Lot                 object
AM or PM            object
Browser Info        object
Company             object
Credit Card          int64
CC Exp Date         object
CC Security Code     int64
CC Provider         object
Email               object
Job                 object
IP Address          object
Language            object
Purchase Price     float64
dtype: object
```

```
#To check the null values in the dataset
ecom.isnull().sum()
```

```
Address             0
Lot                 0
AM or PM            0
Browser Info        0
Company             0
Credit Card         0
CC Exp Date         0
CC Security Code    0
CC Provider         0
Email               0
Job                 0
IP Address          0
Language            0
```

```
Purchase Price      0
dtype: int64
```

```
#To find the number of rows & columns
ecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Address           10000 non-null  object 
 1   Lot               10000 non-null  object 
 2   AM or PM          10000 non-null  object 
 3   Browser Info      10000 non-null  object 
 4   Company           10000 non-null  object 
 5   Credit Card       10000 non-null  int64  
 6   CC Exp Date       10000 non-null  object 
 7   CC Security Code  10000 non-null  int64  
 8   CC Provider       10000 non-null  object 
 9   Email             10000 non-null  object 
 10  Job               10000 non-null  object 
 11  IP Address        10000 non-null  object 
 12  Language          10000 non-null  object 
 13  Purchase Price    10000 non-null  float64
dtypes: float64(1), int64(2), object(11)
memory usage: 1.1+ MB
```

```
len(ecom.columns)
```

```
14
```

```
len(ecom)
```

```
10000
```

```
#Find the highest & lowest purchase prices
```

```
ecom.columns
```

```
Index(['Address', 'Lot', 'AM or PM', 'Browser Info', 'Company', 'Credit Card',
       'CC Exp Date', 'CC Security Code', 'CC Provider', 'Email', 'Job',
       'IP Address', 'Language', 'Purchase Price'],
      dtype='object')
```

```
import pandas as pd
ecom["Purchase Price "].max()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

                                 4 frames                                 

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Purchase Price '

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:
-> 3804                 raise KeyError(key) from err
   3805             except TypeError:
   3806                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Purchase Price '
```

```
ecom["Purchase Price"].min()
```

```
0.0
```

```
#Find the average of purchase price
```

```
ecom["Purchase Price"].mean()
```

```
50.347302
```

```
ecom.describe()
```

	Credit Card	CC Security Code	Purchase Price
count	1.000000e+04	10000.000000	10000.000000
mean	2.341374e+15	907.217800	50.347302
std	2.256103e+15	1589.693035	29.015836
min	6.040186e+10	0.000000	0.000000
25%	3.056322e+13	280.000000	25.150000
50%	8.699942e+14	548.000000	50.505000
75%	4.492298e+15	816.000000	75.770000
max	6.012000e+15	9993.000000	99.990000

```
#Find the average of Purchase Price
```

```
ecom["Purchase Price "].mean()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

                                 4 frames
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Purchase Price '

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:
-> 3804                 raise KeyError(key) from err
   3805             except TypeError:
   3806                 # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Purchase Price '
```

```
#ML Class 3
#Language contains fr
```

```
ecom[ecom["Language"].str.contains('fr',case=False)]["Language"]
```

```
1       fr
19      fr
53      fr
76      fr
82      fr
       ..
9941    fr
9947    fr
9951    fr
9977    fr
9980    fr
Name: Language, Length: 1097, dtype: object
```

Double-click (or enter) to edit

```
len(ecom[ecom["Language"].str.contains('fr',case=False)]["Language"])
```

```
1097
```

```
#Job Title contains Engineer
```

```
ecom.columns
```

```
Index(['Address', 'Lot', 'AM or PM', 'Browser Info', 'Company', 'Credit Card',
       'CC Exp Date', 'CC Security Code', 'CC Provider', 'Email', 'Job',
       'IP Address', 'Language', 'Purchase Price'],
      dtype='object')
```

```
ecom[ecom["Job"].str.contains('Engineer',case=False)]
```

	Address	Lot	AM or PM	Browser Info	Company	Credit Card	CC Exp Date	CC Security Code	CC Provider	Email	
1	9374 Jasmine Spurs Suite 508\nSouth John, TN 8...	28 rn	PM	Opera/8.93. (Windows 98; Win 9x 4.90; en-US) Pr...	Fletcher, Richards and Whitaker	3337758169645356	11/18	561	Mastercard	anthony41@reed.com	Dı eng
3	7780 Julia Fords\nNew Stacy, WA 45798	36 vm	PM	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_0 ...	Williams, Marshall and Buchanan	6011578504430710	02/24	384	Discover	brent16@olson-robinson.info	Dı eng
50	41159 Michael Centers\nAdamsfort, RI 37108-6674	46 Ce	PM	Mozilla/5.0 (Windows 98; Win 9x 4.90; sl-SI; r...	Wright, Williams and Mendez	4008586485908075	05/19	945	JCB 16 digit	susanvalentine@obrien.org	Mecha eng
55	27635 Maureen Bypass Apt. 883\nSandraview, SD ...	59 LJ	AM	Mozilla/5.0 (iPod; U; CPU iPhone OS 3_3 like M...	Sims-Lyons	3158113629128344	09/19	857	VISA 16 digit	adkinsarthur@yahoo.com	Engı broadca (opera
60	7126 Katherine Squares\nPerkinsview, CO 97299-...	63 qu	AM	Opera/8.68. (X11; Linux x86_64; en-US) Presto/2...	Marshall-Fernandez	349767747049645	05/20	672	JCB 15 digit	sweeneyhannah@jones.biz	Engı agricu
...	...	...	...	...	...	...	...	...	...	...	
9948	95544 Johnson Isle Suite 939\nMichaelberg, RI ...	91 bW	AM	Opera/8.36. (X11; Linux x86_64; sl-SI) Presto/2...	Fox-Peterson	4762924304307	03/17	567	Mastercard	haleybenjamin@gmail.com	Struc eng
9952	9991 Vaughn Hills\nRacheltown, PA 55409	36 KC	PM	Mozilla/5.0 (X11; Linux i686; rv:1.9.5.20) Gec...	Ward, Smith and Castillo	6011679271321726	09/19	964	Voyager	jonesjennifer@olson.com	Engı e
9970	0060 Keith Stream\nWestport, CO 47097	11 nt	PM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_8...	Carpenter, Good and Hart	6011485664704662	07/19	543	Discover	rangelbrian@hotmail.com	Elec eng
9977	02182 Keith Expressway\nEast Shannon, CT 20578...	34 RL	AM	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...	Deleon, Jacobson and Benton	4186094003664688	06/21	397	JCB 16 digit	daltoncarter@yahoo.com	Biome eng

```
#Email address of person with IP=132.207.160.22
```

```
ecom.columns
```

```
Index(['Address', 'Lot', 'AM or PM', 'Browser Info', 'Company', 'Credit Card',
       'CC Exp Date', 'CC Security Code', 'CC Provider', 'Email', 'Job',
       'IP Address', 'Language', 'Purchase Price'],
      dtype='object')
```

```
ecom[ecom["IP Address"]=="132.207.160.22"]
```

	Address	Lot	AM or PM	Browser Info	Company	Credit Card	CC Exp Date	CC Security Code	CC Provider	Email	Job	IP Address	Langua
2	Unit 0065 Box [illegible]	94 [illegible]	PM	Mozilla/5.0 (compatible; MSIE 9.0;	Simpson, Williams [illegible]	675957666125	08/19	699	JCB 16 [illegible]	amymiller@morales-[illegible]	Customer service	132.207.160.22	

```
ecom[ecom["IP Address"]=="132.207.160.22"]["Email"]
```

```
2    amymiller@morales-harrison.com
Name: Email, dtype: object
```

```
#Creating a data frame from pre-exisiting columns
```

```
import pandas as pd
edata=pd.read_csv("Ecommerce Purchases")
print(edata.columns)
print("\nThe original DataFrame:")
print(edata.head())
print("\nThe new DataFrame with selected columns is :\n")
edata_new=pd.DataFrame(edata, columns=["Company","Job","Purchase Price"])
print(edata_new.head())
```

```
Index(['Address', 'Lot', 'AM or PM', 'Browser Info', 'Company', 'Credit Card',
       'CC Exp Date', 'CC Security Code', 'CC Provider', 'Email', 'Job',
       'IP Address', 'Language', 'Purchase Price'],
      dtype='object')

The original DataFrame:
                                              Address  Lot AM or PM  \
0  16629 Pace Camp Apt. 448\nAlexisborough, NE 77...  46 in       PM
1  9374 Jasmine Spurs Suite 508\nSouth John, TN 8...  28 rn       PM
2                Unit 0065 Box 5052\nDPO AP 27450  94 vE       PM
3           7780 Julia Fords\nNew Stacy, WA 45798  36 vm       PM
4  23012 Munoz Drive Suite 337\nNew Cynthia, TX 5...  20 IE       AM

                                        Browser Info  \
0  Opera/9.56.(X11; Linux x86_64; sl-SI) Presto/2...
1  Opera/8.93.(Windows 98; Win 9x 4.90; en-US) Pr...
2  Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...
3  Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_0 ...
4  Opera/9.58.(X11; Linux x86_64; it-IT) Presto/2...

                    Company       Credit Card CC Exp Date  \
0             Martinez-Herman  6011929061123406       02/20
1  Fletcher, Richards and Whitaker  3337758169645356       11/18
2     Simpson, Williams and Pham     675957666125       08/19
3  Williams, Marshall and Buchanan  6011578504430710       02/24
4      Brown, Watson and Andrews  6011456623207998       10/25

  CC Security Code            CC Provider  \
0              900           JCB 16 digit
1              561             Mastercard
2              699           JCB 16 digit
3              384               Discover
4              678  Diners Club / Carte Blanche

                      Email                          Job  \
0           pdunlap@yahoo.com  Scientist, product/process development
1          anthony41@reed.com                Drilling engineer
2  amymiller@morales-harrison.com          Customer service manager
3    brent16@olson-robinson.info                Drilling engineer
4    christopherwright@gmail.com                      Fine artist

       IP Address Language  Purchase Price
0  149.146.147.205       el          98.14
1    15.160.41.51       fr          70.73
2   132.207.160.22       de           0.95
3    30.250.74.19       es          78.04
4    24.140.33.94       es          77.82

The new DataFrame with selected columns is :

                    Company                          Job  \
0             Martinez-Herman  Scientist, product/process development
1  Fletcher, Richards and Whitaker                Drilling engineer
2     Simpson, Williams and Pham          Customer service manager
3  Williams, Marshall and Buchanan                Drilling engineer
4      Brown, Watson and Andrews                      Fine artist

  Purchase Price
```

```
#Grouping
```

```
ecom.columns
```

```
Index(['Address', 'Lot', 'AM or PM', 'Browser Info', 'Company', 'Credit Card',
       'CC Exp Date', 'CC Security Code', 'CC Provider', 'Email', 'Job',
       'IP Address', 'Language', 'Purchase Price'],
      dtype='object')
```

```
ecom.groupby( 'AM' or 'PM ').groups
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-51-f91fd48c1e86> in <cell line: 1>()
----> 1 ecom.groupby( 'AM' or 'PM ').groups

                                  2 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/groupby/grouper.py in get_grouper(obj, key, axis, level, sort, observed, mutated, validate, dropna)
    886                 in_axis, level, gpr = False, gpr, None
    887             else:
--> 888                 raise KeyError(gpr)
    889         elif isinstance(gpr, Grouper) and gpr.key is not None:
    890             # Add key to exclusions

KeyError: 'AM'
```

```
ecom.groupby('AM or PM').get_group("AM")
```

	Address	Lot	AM or PM	Browser Info	Company	Credit Card	CC Exp Date	CC Security Code	CC Provider	Email
4	23012 Munoz Drive Suite 337\nNew Cynthia, TX 5...	20 IE	AM	Opera/9.58.(X11; Linux x86_64; it-IT) Presto/2...	Brown, Watson and Andrews	6011456623207998	10/25	678	Diners Club / Carte Blanche	christopherwright@gmail.com
6	93971 Conway Causeway\nAndersonburgh, AZ 75107	96 Xt	AM	Mozilla/5.0 (compatible; MSIE 7.0; Windows NT ...	Gibson and Sons	6011398782655569	07/24	714	VISA 16 digit	olivia04@yahoo.com
9	3795 Dawson Extensions\nLake Tinafort, ID 88739	15 Ug	AM	Mozilla/5.0 (X11; Linux i686; rv:1.9.7.20) Gec...	Rivera, Buchanan and Ramirez	4396283918371	01/17	931	American Express	qcoleman@hunt-huerta.com
12	733 Heather Rest Apt. 670\nBoltonport, UT 78662	69 DO	AM	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_5_3 ...	Moore-Martin	5115990487067905	05/26	119	VISA 16 digit	tholt@hotmail.com
14	8774 Jason Keys Suite 427\nEast Scottborough, ...	70 zH	AM	Mozilla/5.0 (Windows 98; it-IT; rv:1.9.2.20) G...	Leach, Howe and Ferguson	869967499275071	09/22	427	VISA 16 digit	aburns@yahoo.com
...	...	...	...	...	...	...	...	...	...	...
9989	5674 Cruz Trace\nNew Shelby, KY 51047-5469	12 Kf	AM	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...	Vargas, Nichols and Martinez	5278049688438382	12/23	996	VISA 16 digit	walterswilliam@hotmail.com
9991	PSC 8165, Box 8498\nAPO AP 60327-0346	50 dA	AM	Mozilla/5.0 (compatible; MSIE 8.0; Windows NT ...	Snyder Inc	4221582137197481	02/24	969	Voyager	kking@wise-liu.com
9996	832 Curtis Dam Suite 785\nNorth Edwardburgh, T...	41 JY	AM	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...	Hale, Collins and Wilson	210033169205009	07/25	207	JCB 16 digit	mary85@hotmail.com
9997	Unit 4434 Box 6343\nDPO AE 28026-0283	74 Zh	AM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7...	Anderson Ltd	6011539787356311	05/21	1	VISA 16 digit	tyler16@gmail.com
	40674 Barrett			Mozilla/5.0 (X11; Linux						

```
ecom.groupby(['AM or PM', 'Company']).groups
```

```
{('AM', 'Abbott Ltd'): [7369], ('AM', 'Abbott and Sons'): [8037], ('AM', 'Abbott-Johnson'): [3171], ('AM', 'Acevedo-Watkins'): [4604], ('AM', 'Acosta, Carroll and Jennings'): [7312], ('AM', 'Acosta-Park'): [3022], ('AM', 'Adams Group'): [311, 2773], ('AM', 'Adams Inc'): [6251, 7763], ('AM', 'Adams LLC'): [7044, 8933], ('AM', 'Adams PLC'): [9294], ('AM', 'Adams and Sons'): [521, 7350], ('AM', 'Adams, Barker and Anderson'): [4456], ('AM', 'Adams, Carpenter and Perry'): [6070], ('AM', 'Adams, Garcia and Torres'): [544], ('AM', 'Adams, Gray and Fry'): [5748], ('AM', 'Adams, Marshall and Sullivan'): [6716], ('AM', 'Adams, Ward and Neal'): [3884], ('AM', 'Adams-Banks'): [4309], ('AM', 'Adams-Conrad'): [4265], ('AM', 'Adams-Morrison'): [4120], ('AM', 'Adams-Powell'): [5926], ('AM', 'Adkins PLC'): [6340], ('AM', 'Adkins and Sons'): [9303], ('AM', 'Adkins, Bruce and Pierce'): [6046], ('AM', 'Adkins, Reed and Torres'): [4539], ('AM', 'Adkins-Smith'): [1504], ('AM', 'Aguilar LLC'): [9483], ('AM', 'Aguilar PLC'): [4910], ('AM', 'Aguilar, Garcia and Burke'): [3757], ('AM', 'Aguilar, Torres and Smith'): [1242], ('AM', 'Aguilar-Jones'): [8429], ('AM', 'Aguilar-Perkins'): [381], ('AM', 'Aguirre LLC'): [1865], ('AM', 'Aguirre and Sons'): [3375], ('AM', 'Alexander Ltd'): [4301], ('AM', 'Alexander, Hicks and Ochoa'): [2564], ('AM', 'Alexander, Morgan and Atkinson'): [463], ('AM', 'Alexander, Perez and Thompson'): [4999], ('AM', 'Alexander, Strong and Strong'): [7593], ('AM', 'Alexander-Franklin'): [7136], ('AM', 'Alexander-Kaiser'): [3388], ('AM', 'Alexander-Kelly'): [8548], ('AM', 'Alexander-Ramos'): [7687], ('AM', 'Alexander-Reynolds'): [6842], ('AM', 'Ali Inc'): [2779], ('AM', 'Ali, Orr and Fischer'): [3457], ('AM', 'Ali-Christensen'): [2414], ('AM', 'Ali-Smith'): [7359], ('AM', 'Allen Group'): [5112, 9713], ('AM', 'Allen Inc'): [1081], ('AM', 'Allen Ltd'): [3028, 8615], ('AM', 'Allen and Sons'): [4687], ('AM', 'Allen, Ashley and Suarez'): [7138], ('AM', 'Allen, Johnson and Baker'): [9167], ('AM', 'Allen, Martin and Richards'): [3904], ('AM', 'Allen, Pitts and Nielsen'): [5267], ('AM', 'Allen, Riley and Cox'): [8382], ('AM', 'Allen, West and Miller'): [1280], ('AM', 'Allen-Delgado'): [5952], ('AM', 'Allen-Harris'): [2774], ('AM', 'Allen-Jacobson'): [2648], ('AM', 'Allen-Johnson'): [1905, 3636], ('AM', 'Allen-Peterson'): [1979], ('AM', 'Allen-Taylor'): [1746], ('AM', 'Allen-Watson'): [1923], ('AM', 'Allison LLC'): [1872], ('AM', 'Allison PLC'): [2883], ('AM', 'Allison, Oneal and Gould'): [9343], ('AM', 'Alvarado, Jacobs and Mills'): [3603], ('AM', 'Alvarez Inc'): [1680], ('AM', 'Alvarez PLC'): [8963], ('AM', 'Alvarez, Green and Porter'): [3533], ('AM', 'Alvarez, Rodriguez and Richardson'): [5407], ('AM', 'Alvarez, Vargas and Walker'): [7912], ('AM', 'Alvarez-Gomez'): [460], ('AM', 'Alvarez-Lopez'): [8247], ('AM', 'Alvarez-Perry'): [6168], ('AM', 'Andersen Inc'): [4777], ('AM', 'Andersen, Jarvis and Brown'): [8266], ('AM', 'Andersen, Rivera and Burke'): [7710], ('AM', 'Andersen-Bernard'): [9926], ('AM', 'Andersen-Richards'): [1272], ('AM', 'Anderson Group'): [6050, 6910, 9813], ('AM', 'Anderson Inc'): [154], ('AM', 'Anderson LLC'): [2197, 8781], ('AM', 'Anderson Ltd'): [1513, 5530, 8945, 9997], ('AM', 'Anderson PLC'): [8302], ('AM', 'Anderson, Burns and House'): [5528], ('AM', 'Anderson, Charles and Alexander'): [9922], ('AM', 'Anderson, Davis and Robbins'): [1741], ('AM', 'Anderson, Gray and Knapp'): [7915], ('AM', 'Anderson, Hill and Andrews'): [5327], ('AM', 'Anderson, Houston and Jackson'): [5119], ('AM', 'Anderson, Mitchell and Smith'): [9544], ('AM', 'Anderson, Rojas and Munoz'): [2048], ('AM', 'Anderson, Stevens and Ward'): [9491], ('AM', 'Anderson, Sutton and Dixon'): [716], ('AM', 'Anderson, Vasquez and Gilbert'): [8830], ('AM', 'Anderson, Williams and Diaz'): [443], ('AM', 'Anderson-Adams'): [3452], ...}
```

```
ecom=pd.read_csv("/content/Ecommerce Purchases")
```

```
ap=ecom.groupby(['AM or PM'])
ap['Purchase Price'].mean()
```

```
AM or PM
AM    50.186511
PM    50.503779
Name: Purchase Price, dtype: float64
```

```
ecom.groupby(['AM or PM'])[['Purchase Price']].mean()
```

AM or PM	Purchase Price
AM	50.186511
PM	50.503779

```
#Sorting
```

```
ecom.sort_values(by='Job').head(50)
```

	Address	Lot	AM or PM	Browser Info	Company	Credit Card	CC Exp Date	CC Security Code	CC Provider	
9931	71958 Diaz Roads Apt. 343\nPort Richardville, ...	62 qP	AM	Opera/8.68.(X11; Linux i686; it-IT) Presto/2.9...	Thompson, Arnold and Ho	4432224867137016	12/19	624	JCB 16 digit	david35@b
4214	1265 Duncan Court\nAlvarezview, DE 03874	67 EV	AM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7...	Martinez-Jordan	4982407347457484	05/21	921	JCB 16 digit	michealhaley@rodri
8499	795 Randall Skyway Suite 928\nKiddstad, MS 709...	24 pK	AM	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/5360 ...	Wilson, Patel and Mccoy	5344177216341636	05/22	386	American Express	amylozano@allen-
7769	PSC 4313, Box 0557\nAPO AA 12033-6203	25 WM	PM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6...	Turner-Harris	4105681690623817	04/26	523	Voyager	hatfieldcorey@p
3914	548 Sanchez Dale Suite 325\nNorth Jay, HI 35901	59 iy	AM	Mozilla/5.0 (Windows; U; Windows NT 6.0) Apple...	Haynes and Sons	6011998459310685	10/18	65	American Express	wheeleralexand
8234	87850 Amy Corner Suite 135\nLake Jonathanside,...	81 Hg	PM	Mozilla/5.0 (Windows NT 5.0) AppleWebKit/5331 ...	Tate Inc	30435953333392	06/21	968	American Express	ggonzalez@r
6589	76753 Duncan Pines\nRamirezmouth, RI 18246	74 YW	AM	Mozilla/5.0 (X11; Linux i686) AppleWebKit/5352...	Moran, Ramsey and Navarro	5456352641046696	04/22	120	VISA 13 digit	jason94@warren-cam
2339	06823 Kathryn Wall Suite 698\nNorth John, WV 5...	26 II	AM	Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10_7_2...	Giles Group	5237469400100757	05/24	4113	VISA 13 digit	rgonzales@y
7103	37030 Terri Orchard\nLake Ginaview, WA 06615-1392	15 Ao	PM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6...	Johnson, Kramer and Smith	3088397087539796	12/20	27	VISA 16 digit	kathy35@
1646	58961 Bryan Run\nNew Barbaraborough, OR 04467-...	11 Mc	PM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7...	Odonnell, Long and Jordan	4197434016663	05/17	630	JCB 16 digit	timothy94@g
1461	617 Glover Roads Suite 548\nHodgefort, TN 29876	27 aM	AM	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/53...	Green PLC	4036104378992887	02/20	932	VISA 13 digit	diane09@orozco-n
6564	85368 Charles Meadows Suite 963\nCastillohaven...	43 Aa	PM	Opera/9.96.(X11; Linux i686; en-US) Presto/2.9...	Wright-Gallegos	5279540864997885	05/21	887	VISA 16 digit	yleonard@l
9628	43141 Hayes Village\nNew Zacharyville, VT 0465...	50 EA	PM	Opera/8.49.(Windows NT 5.2; it-IT) Presto/2.9....	Washington and Sons	180012456109823	07/23	1379	VISA 16 digit	pwhite@ho

```
ecom_sorted=ecom.sort_values(by=['Job','CC Security Code'],ascending=[True,False])
ecom_sorted.head(10)
```

	Address	Lot	AM or PM	Browser Info	Company	Credit Card	CC Exp Date	CC Security Code	CC Provider	Emai
2339	06823 Kathryn Wall Suite 698\nNorth John, WV 5...	26 II	AM	Mozilla/5.0 (Macintosh; U; PPC Mac OS X 10_7_2...	Giles Group	5237469400100757	05/24	4113	VISA 13 digit	rgonzales@yahoo.com
8234	87850 Amy Corner Suite 135\nLake Jonathanside,...	81 Hg	PM	Mozilla/5.0 (Windows NT 5.0) AppleWebKit/5331 ...	Tate Inc	30435953333392	06/21	968	American Express	ggonzalez@miles.com
1461	617 Glover Roads Suite 548\nHodgefort, TN 29876	27 aM	AM	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/53...	Green PLC	4036104378992887	02/20	932	VISA 13 digit	diane09@orozco-nunez.com
4214	1265 Duncan Court\nAlvarezview, DE 03874	67 EV	AM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7...	Martinez-Jordan	4982407347457484	05/21	921	JCB 16 digit	michealhaley@rodriguez.com
1646	58961 Bryan Run\nNew Barbaraborough, OR 04467-...	11 Mc	PM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_7...	Odonnell, Long and Jordan	4197434016663	05/17	630	JCB 16 digit	timothy94@gmail.com
9931	71958 Diaz Roads Apt. 343\nPort Richardville, ...	62 qP	AM	Opera/8.68.(X11; Linux i686; it-IT) Presto/2.9...	Thompson, Arnold and Ho	4432224867137016	12/19	624	JCB 16 digit	david35@banks.com
7769	PSC 4313, Box 0557\nAPO AA 12033-6203	25 WM	PM	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6...	Turner-Harris	4105681690623817	04/26	523	Voyager	hatfieldcorey@parker.net
8499	795 Randall Skyway Suite 928\nKiddstad, MS 709...	24 pK	AM	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/5360 ...	Wilson, Patel and Mccoy	5344177216341636	05/22	386	American Express	amylozano@allen-miller.net
6589	76753 Duncan Pines\nRamirezmouth, RI 18246	74 YW	AM	Mozilla/5.0 (X11; Linux i686) AppleWebKit/5352...	Moran, Ramsey and Navarro	5456352641046696	04/22	120	VISA 13 digit	jason94@warren-campbell.com

7777	Neck\nNew Jack, NM	32	AM	Linux x86_64; it-IT)	[illegible]	4590972077595872	09/19	330	Discover	ellislindsey@c

```
ecom['Cost']=ecom["Purchase Price"].apply(lambda x: 'costly' if x>=25 else 'Cheap')
ecom.head()
```

	Address	Lot	AM or PM	Browser Info	Company	Credit Card	CC Exp Date	CC Security Code	CC Provider	Email	
0	16629 Pace Camp Apt. 448\nAlexisborough, NE 77...	46 in	PM	Opera/9.56. (X11; Linux x86_64; sl-SI) Presto/2...	Martinez-Herman	6011929061123406	02/20	900	JCB 16 digit	pdunlap@yahoo.com	Scie product/pr develop
1	9374 Jasmine Spurs Suite 508\nSouth John, TN 8...	28 rn	PM	Opera/8.93. (Windows 98; Win 9x 4.90; en-US) Pr...	Fletcher, Richards and Whitaker	3337758169645356	11/18	561	Mastercard	anthony41@reed.com	Dr eng
2	Unit 0065 Box 5052\nDPO AP 27450	94 vE	PM	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT ...	Simpson, Williams and Pham	675957666125	08/19	699	JCB 16 digit	amymiller@morales-harrison.com	Cust service ma
3	7780 Julia Fords\nNew Stacy, WA 45798	36 vm	PM	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_0 ...	Williams, Marshall and Buchanan	6011578504430710	02/24	384	Discover	brent16@olson-robinson.info	Dr eng
	23012 Munoz Drive			Opera/9.58. (X11; Linux	Brown,				Diners		

```
print(ecom.columns)
```

```
Index(['Address', 'Lot', 'AM or PM', 'Browser Info', 'Company', 'Credit Card',
       'CC Exp Date', 'CC Security Code', 'CC Provider', 'Email', 'Job',
       'IP Address', 'Language', 'Purchase Price'],
      dtype='object')
```

45257 Kristen Mozilla/5.0 (iPod; U; House,

#Data Visualization

```
import seaborn as sns
```

Nicole, CT 21552 Vp Hutchinson Carte

```
print(sns.get_dataset_names())
```

```
['anagrams', 'anscombe', 'attention', 'brain_networks', 'car_crashes', 'diamonds', 'dots', 'dowjones', 'exercise', 'flights', 'fmri', 'geyser', 'glue', 'healthexp', 'iris', 'mpg',
```

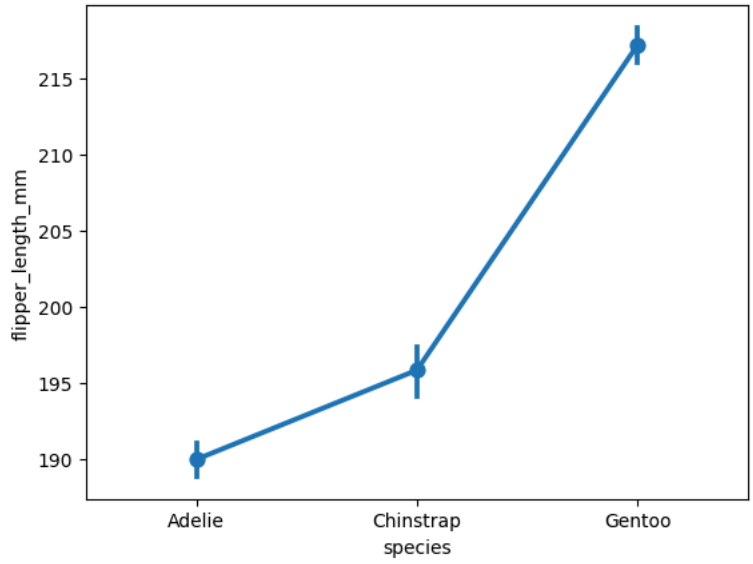
```
tab=sns.load_dataset('penguins')
tab.head(10)
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	Male
6	Adelie	Torgersen	38.9	17.8	181.0	3625.0	Female
7	Adelie	Torgersen	39.2	19.6	195.0	4675.0	Male
8	Adelie	Torgersen	34.1	18.1	193.0	3475.0	NaN
9	Adelie	Torgersen	42.0	20.2	190.0	4250.0	NaN

Opera/8.78

```
sns.pointplot(x='species',y="flipper_length_mm",data=tab)
```

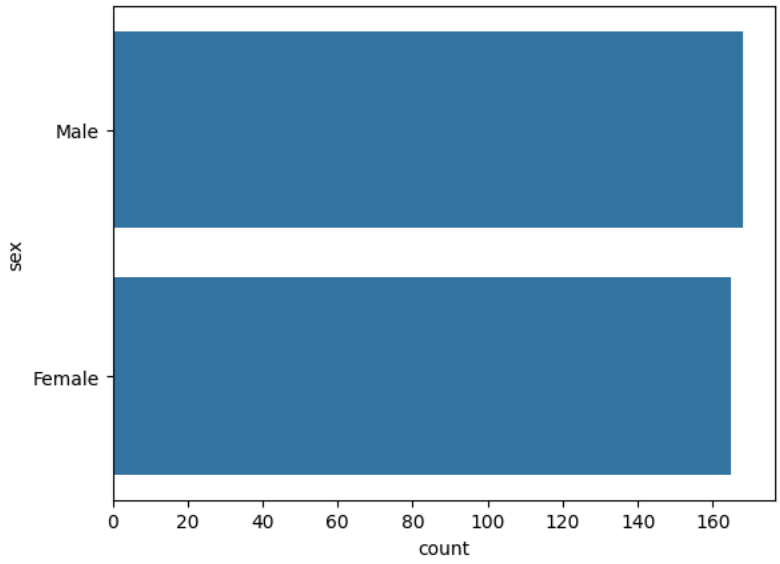
```
<Axes: xlabel='species', ylabel='flipper_length_mm'>
```



#Count Plot

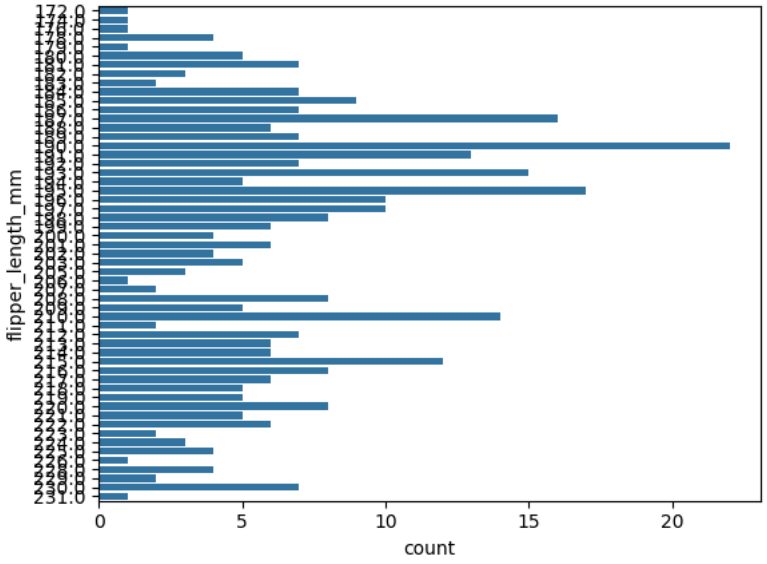
```
sns.countplot(y='sex',data=tab)
```

<Axes: xlabel='count', ylabel='sex'>



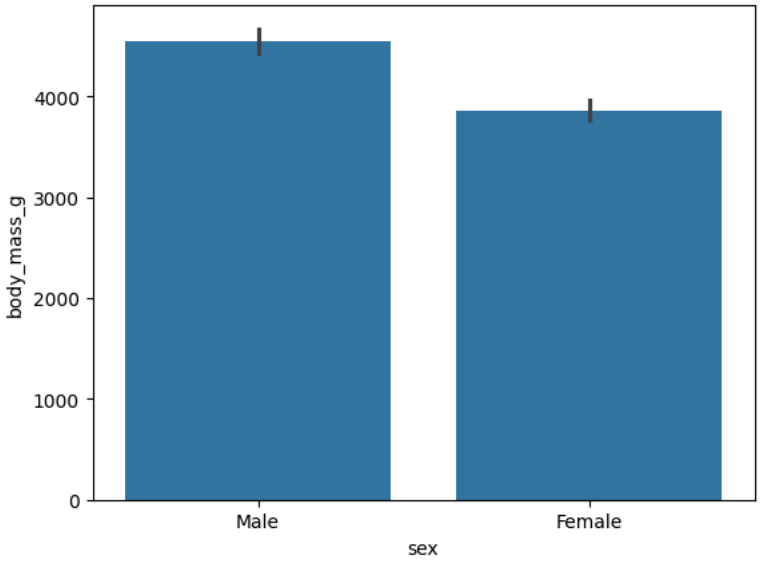

```
sns.countplot(y="flipper_length_mm",data=tab)
```

<Axes: xlabel='count', ylabel='flipper_length_mm'>




```
sns.barplot(x='sex',y='body_mass_g',data=tab)
```

<Axes: xlabel='sex', ylabel='body_mass_g'>

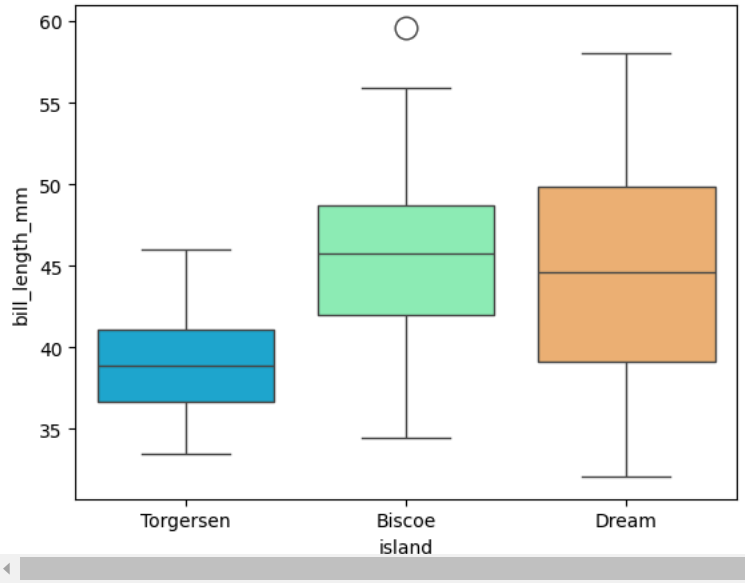



```
sns.boxplot(x='island',y='bill_length_mm',data=tab,fliersize=12,palette='rainbow')
```

```
<ipython-input-82-65522670dd2c>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same e
```

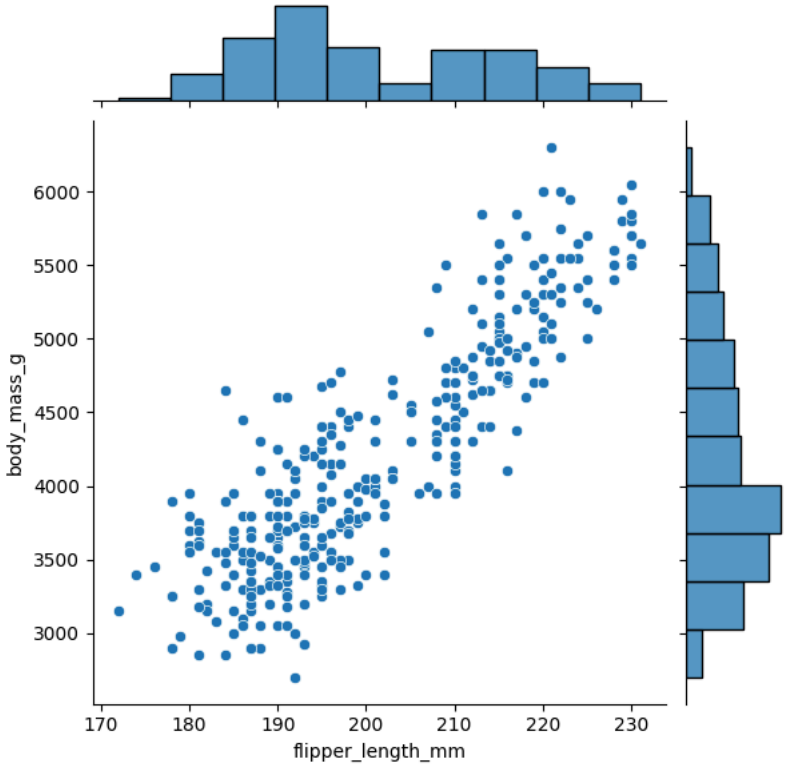
```
sns.boxplot(x='island',y='bill_length_mm',data=tab,fliersize=12,palette='rainbow')
```

```
<Axes: xlabel='island', ylabel='bill_length_mm'>
```



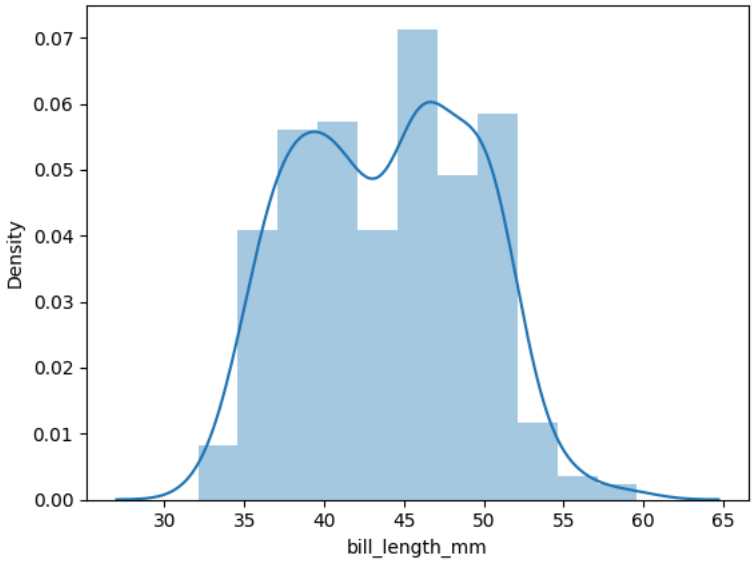

```
sns.jointplot(x='flipper_length_mm',y='body_mass_g',data=tab,dropna=True)
```

```
<seaborn.axisgrid.JointGrid at 0x79fbcca93f10>
```




```
sns.distplot(tab['bill_length_mm'],hist=True)
```

```
<ipython-input-85-64b5f88f4921>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(tab['bill_length_mm'],hist=True)
<Axes: xlabel='bill_length_mm', ylabel='Density'>
```



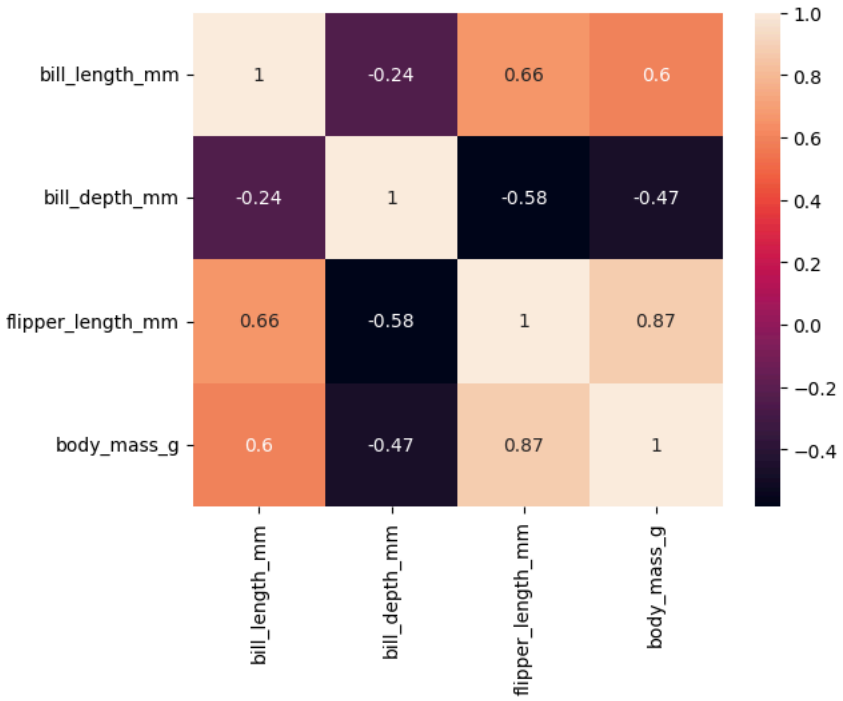

```
tab.corr()
```

```
<ipython-input-86-1ce724e5c9c9>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fa
  tab.corr()
```

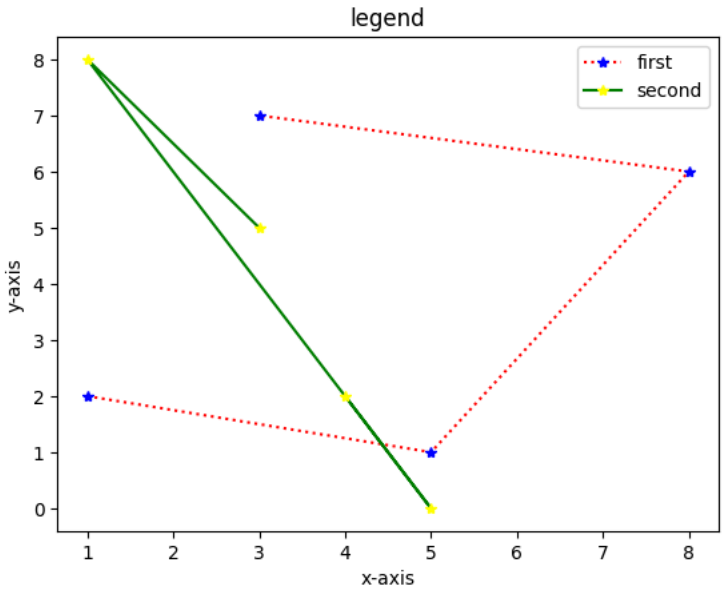
	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
bill_length_mm	1.000000	-0.235053	0.656181	0.595110
bill_depth_mm	-0.235053	1.000000	-0.583851	-0.471916
flipper_length_mm	0.656181	-0.583851	1.000000	0.871202
body_mass_g	0.595110	-0.471916	0.871202	1.000000

```
sns.heatmap(tab.corr(),annot=True)
```

```
<ipython-input-87-3fbdb2413c03>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fa
  sns.heatmap(tab.corr(),annot=True)
<Axes: >
```

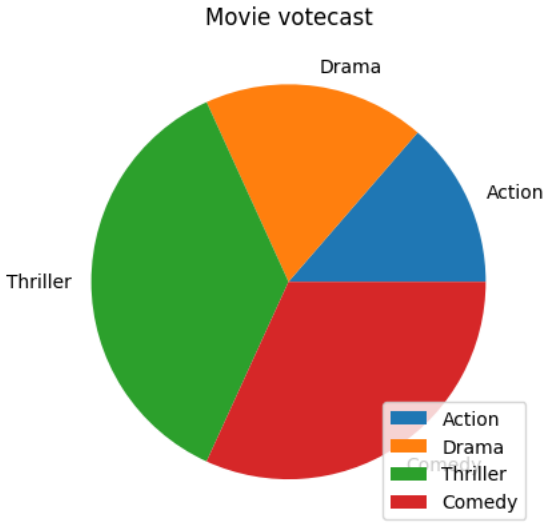



```
from matplotlib import pyplot as p
import numpy as np
x1=[1,5,8,3]
x2=[2,1,6,7]
y1=[4,5,1,3]
y2=[2,0,8,5]
p.plot(x1,x2,'r:*',markerfacecolor='blue',markeredgecolor='blue',label='first')
p.plot(y1,y2,'g-*',markerfacecolor='yellow',markeredgecolor='yellow',label='second')
p.title('legend')
p.xlabel('x-axis')
p.ylabel('y-axis')
p.legend()
p.show()
```

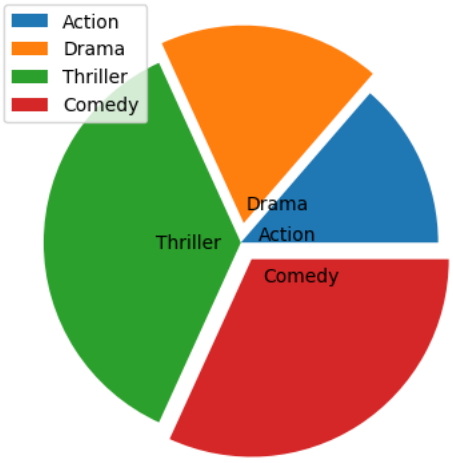



```
cat=['Action','Drama','Thriller','Comedy']
vote=[3,4,8,7]
p.pie(vote)
p.show()
```

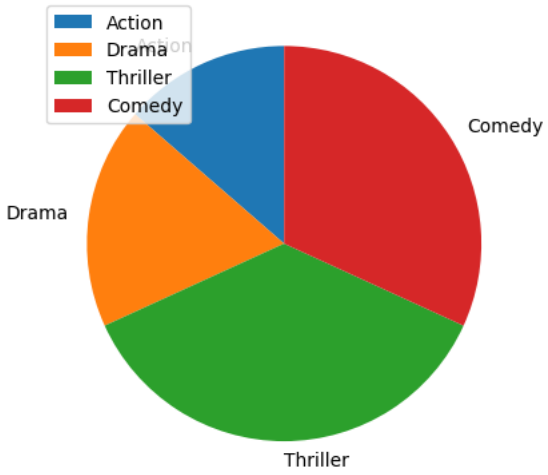
```
p.title("Movie votecast")
p.pie(vote,labels=cat,labeldistance=1.1)
p.legend(loc=4)
p.show()
```



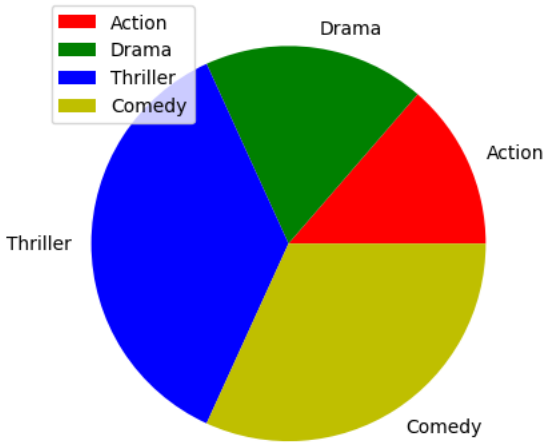

```
#Pull out the two edges
from matplotlib import pyplot as p
vote=[3,4,8,7]
p.pie(vote,labels=cat,labeldistance=0.1,explode=[0,0.1,0,0.1])
p.legend(loc=2)
p.show()
```



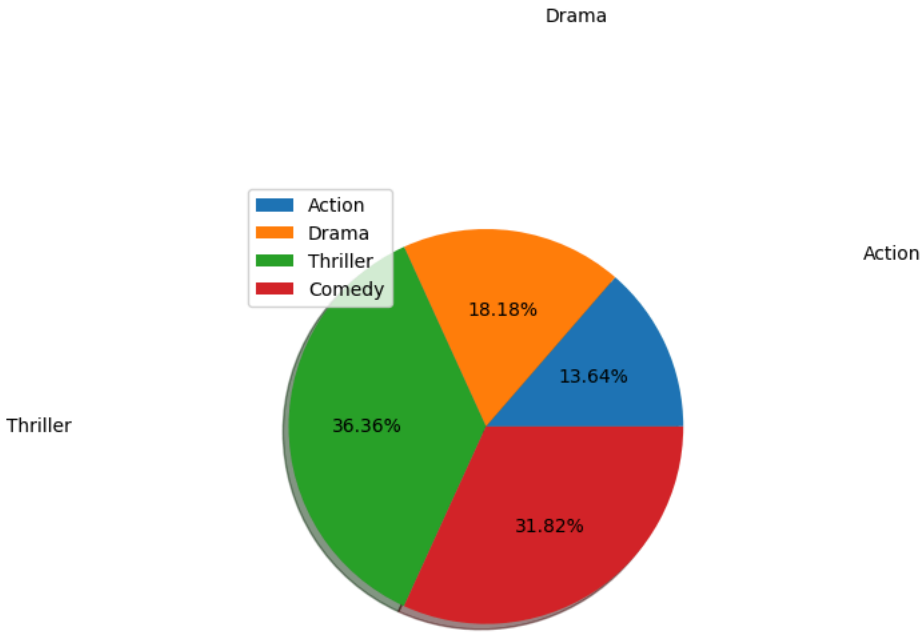
```
#Change start angle to 90 deg
p.pie(vote,labels=cat,labeldistance=1.1,startangle=90)
p.legend(loc=2)
p.show()
```



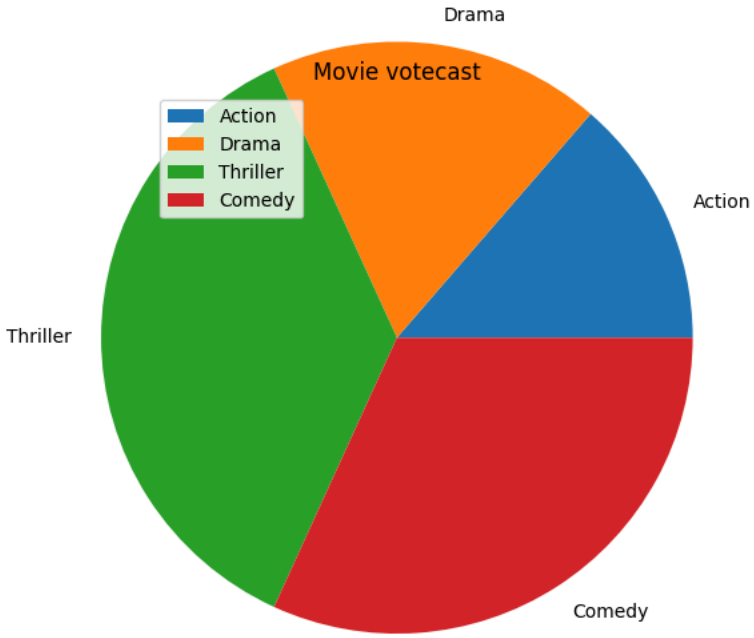

```
#Setting colors for each wedge
p.pie(vote,labels=cat,labeldistance=1.1,colors=['red','g','b','y'])
p.legend(loc=2)
p.show()
```



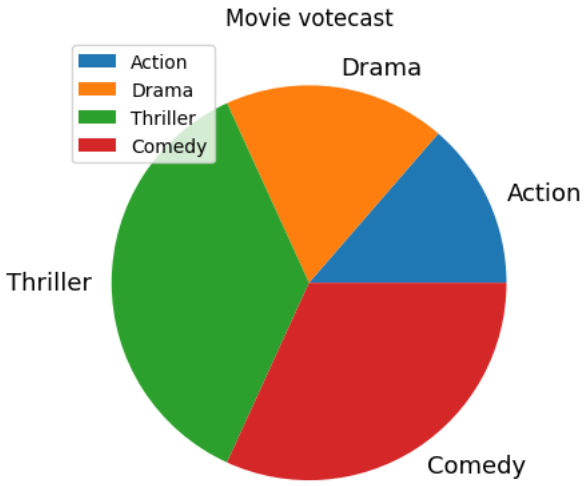

```
#Display percentage & change distance inside each wedge
p.pie(vote,labels=cat,labeldistance=2.1,shadow=True,autopct="%0.2f%%",pctdistance=0.6)
p.legend(loc=2)
p.show()
```




```
#Change radius of the part
p.title("Movie votecast")
p.pie(vote,labels=cat,labeldistance=1.1,radius=1.5)
p.legend(loc=2)
p.show()
```

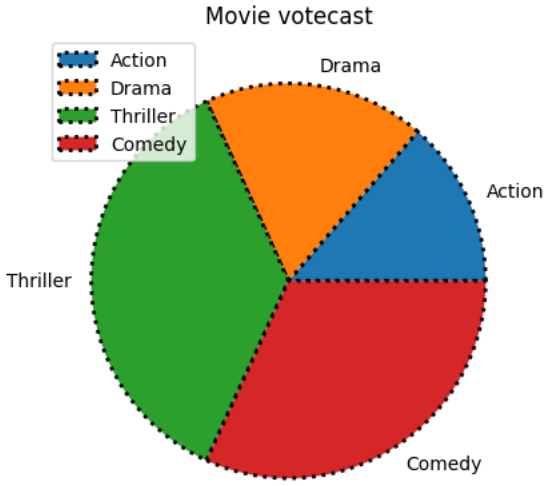



```
#Set label size of Pie Chart
p.title("Movie votecast")
p.pie(vote,labels=cat,labeldistance=1.1,textprops={'fontsize':13})
p.legend(loc=2)
p.show()
```




```
#Set border for Wedges
p.title("Movie votecast")
p.pie(vote,labels=cat,labeldistance=1.1,wedgeprops={'edgecolor':'black','linewidth':2,'linestyle':':'})
p.legend(loc=2)
p.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
#Class 4
#Linear Regression
```

```
#Simple Variable Linear Regression
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
pay_data=pd.read_csv('Salary_Data.csv')
pay_data
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

```
pay_data.info()
```

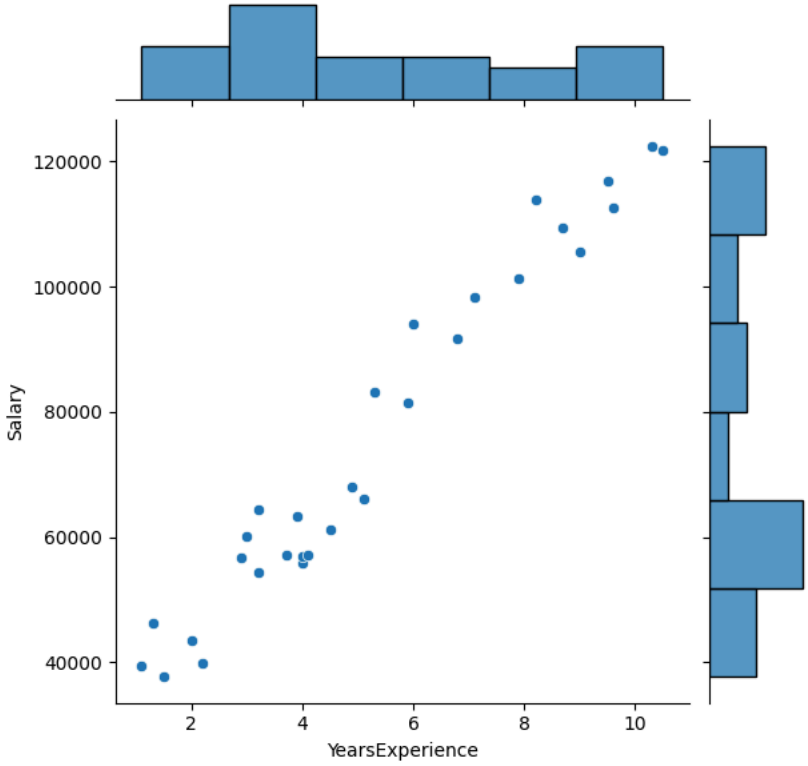
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

```
pay_data.describe()
```

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

```
sns.jointplot(x='YearsExperience',y='Salary',data=pay_data)
```

```
<seaborn.axisgrid.JointGrid at 0x7ac6aa1b5e70>
```




```
#Divide data into inputs & outputs
```

```
X=pay_data.iloc[:,:-1]
X
```

	YearsExperience
0	1.1
1	1.3
2	1.5
3	2.0
4	2.2
5	2.9
6	3.0
7	3.2
8	3.2
9	3.7
10	3.9
11	4.0
12	4.0
13	4.1
14	4.5
15	4.9
16	5.1
17	5.3
18	5.9
19	6.0
20	6.8
21	7.1
22	7.9
23	8.2
24	8.7
25	9.0
26	9.5
27	9.6
28	10.3
29	10.5

```
y=pay_data.iloc[:,1]
y
```

```
0     39343.0
1     46205.0
2     37731.0
3     43525.0
4     39891.0
5     56642.0
6     60150.0
7     54445.0
8     64445.0
9     57189.0
10    63218.0
11    55794.0
12    56957.0
13    57081.0
14    61111.0
15    67938.0
16    66029.0
17    83088.0
18    81363.0
19    93940.0
20    91738.0
21    98273.0
22   101302.0
23   113812.0
24   109431.0
25   105582.0
26   116969.0
27   112635.0
```

```
28    122391.0
29    121872.0
Name: Salary, dtype: float64
```

```
#Splitting data into the Training & Test set
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
#when random_state set to an integer
#train_test_split will return same results for each execution
```

```
len(X_test)
```

```
9
```

```
#Fitting Simple Linear Regression to the Training set
```

```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(X_train,y_train)
```

```
LinearRegression
LinearRegression()
```

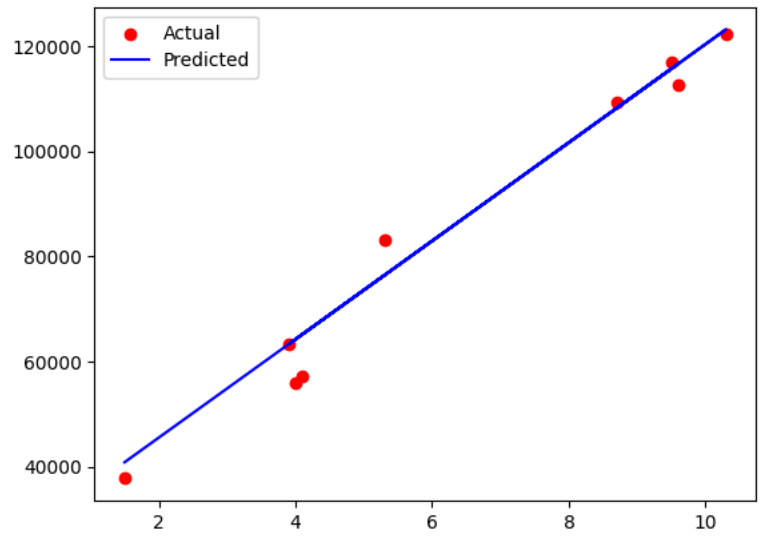
```
len(X_test)
```

```
9
```

```
#Prediction
```

```
y_pred=model.predict(X_test)
```

```
plt.scatter(X_test,y_test,color='red',label='Actual')
plt.plot(X_test,y_pred,'b-',label='Predicted')
plt.legend()
plt.show()
```




```
model.intercept_
```

```
26777.391341197625
```

```
model.coef_
```

```
array([9360.26128619])
```

```
model.predict([[9.8]])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature
  warnings.warn(
array([118507.9519459])
```

```
(9360.26128619*9.8)+26777.39134119764
```

```
118507.95194585965
```

```
model.score(X_test,y_test)
```

```
0.9740993407213511
```

```
from sklearn import metrics
print('MAE',metrics.mean_absolute_error(y_test,y_pred))
print('MSE',metrics.mean_squared_error(y_test,y_pred))
print('RMSE',np.sqrt(metrics.mean_absolute_error(y_test,y_pred)))
```

```
MAE 3737.417861878896
MSE 23370078.800832972
RMSE 61.134424523985636
```

```
#Mulitple Linear Regression
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
hire=pd.read_csv("hiring.csv")
hire.head()
```

	experience	test_score	interview_score	salary
0	NaN	8.0	9	50000
1	five	8.0	6	45000
2	two	6.0	7	60000
3	seven	10.0	10	65000
4	three	9.0	6	70000

```
hire.isna().sum()
```

```
experience         2
test_score         1
interview_score    0
salary             0
dtype: int64
```

```
#word2number conversion & usage
```

```
pip install Word2Number
```

```
Collecting Word2Number
  Downloading word2number-1.1.zip (9.7 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: Word2Number
  Building wheel for Word2Number (setup.py) ... done
  Created wheel for Word2Number: filename=word2number-1.1-py3-none-any.whl size=5568 sha256=9a671edb8084637831c80fbb7427916277e46c2ee3f87b89ac4c
  Stored in directory: /root/.cache/pip/wheels/84/ff/26/d3cfbd971e96c5aa3737ecfced81628830d7359b55fbb8ca3b
Successfully built Word2Number
Installing collected packages: Word2Number
Successfully installed Word2Number-1.1
```

```
from word2number import w2n
```

```
hire['experience']=hire.experience.fillna("Zero")
hire.experience
```

```
0      Zero
1      five
2       two
3     seven
4     three
5       ten
6    eleven
7      Zero
Name: experience, dtype: object
```

```
hire.experience=hire.experience.apply(w2n.word_to_num)
hire.head()
```

	experience	test_score	interview_score	salary
0	0	8.0	9	50000
1	5	8.0	6	45000
2	2	6.0	7	60000
3	7	10.0	10	65000
4	3	9.0	6	70000

```
hire.isnull().sum()
```

```
experience      0
test_score      1
interview_score 0
salary          0
dtype: int64
```

```
hire[hire['test_score'].isnull()]
```

	experience	test_score	interview_score	salary
6	11	NaN	7	72000

```
hire.describe()
```

	experience	test_score	interview_score	salary
count	8.00000	7.000000	8.000000	8.00000
mean	4.75000	7.857143	7.875000	63000.00000
std	4.26782	1.345185	1.642081	11501.55269
min	0.00000	6.000000	6.000000	45000.00000
25%	1.50000	7.000000	6.750000	57500.00000
50%	4.00000	8.000000	7.500000	63500.00000
75%	7.75000	8.500000	9.250000	70500.00000
max	11.00000	10.000000	10.000000	80000.00000

```
hire[hire['salary']>6000]
```

	experience	test_score	interview_score	salary
0	0	8.0	9	50000
1	5	8.0	6	45000
2	2	6.0	7	60000
3	7	10.0	10	65000
4	3	9.0	6	70000
5	10	7.0	10	62000
6	11	NaN	7	72000
7	0	7.0	8	80000

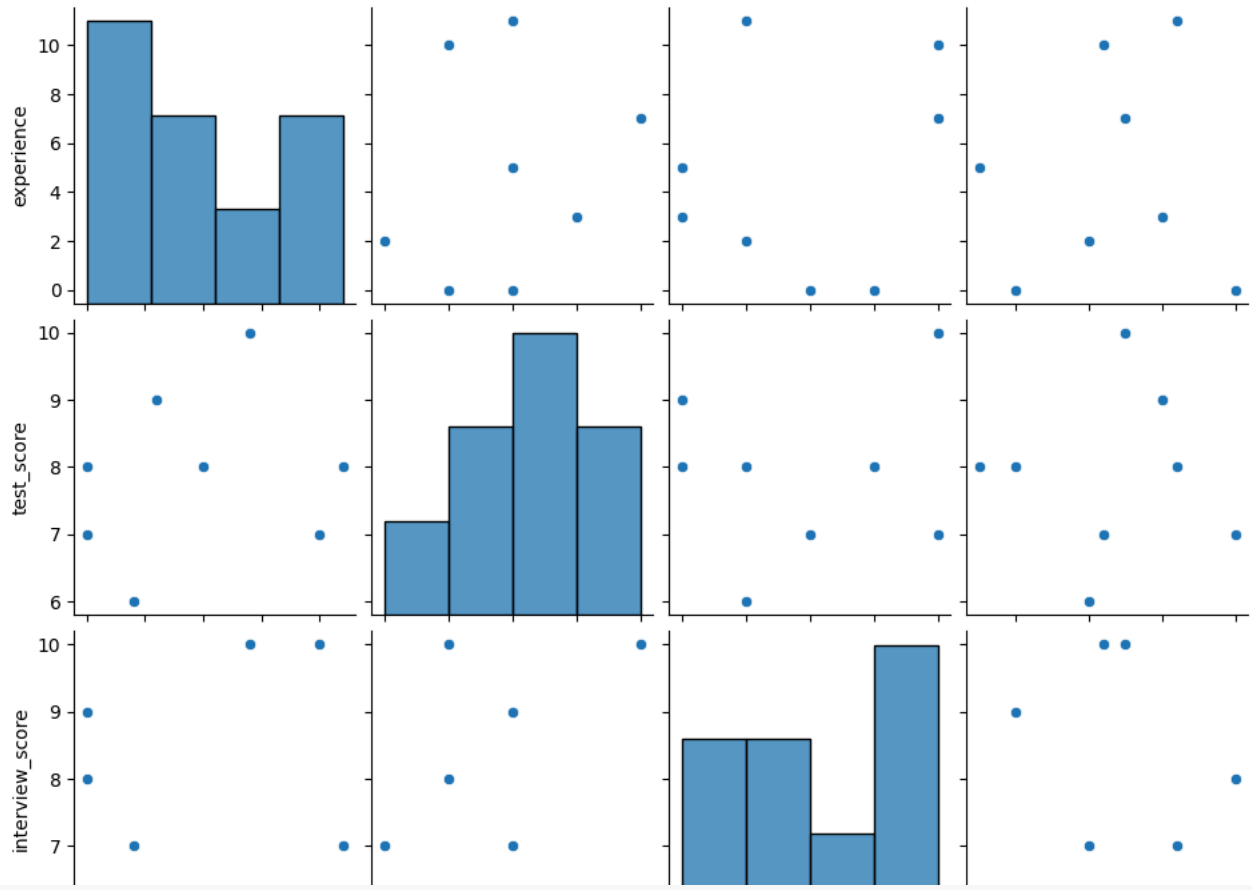
```
med=hire.test_score.median()
hire['test_score']=hire['test_score'].fillna(med)
```

```
med
```

```
8.0
```

```
sns.pairplot(hire)
```

```
<seaborn.axisgrid.PairGrid at 0x7ac6a7aaaf80>
```




```
X=hire.iloc[:,:-1].values
X
```

```
array([[ 0.,  8.,  9.],
       [ 5.,  8.,  6.],
       [ 2.,  6.,  7.],
       [ 7., 10., 10.],
       [ 3.,  9.,  6.],
       [10.,  7., 10.],
       [11.,  8.,  7.],
       [ 0.,  7.,  8.]])
```

```
y=hire.iloc[:,[3]].values
y
```

```
array([[50000],
       [45000],
       [60000],
       [65000],
       [70000],
       [62000],
       [72000],
       [80000]])
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=1/4,random_state=10)
```

```
len(X_train)
```

```
6
```

```
from sklearn.linear_model import LinearRegression
reg=LinearRegression()
reg.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
y_pred=reg.predict(X_test)
```

```
reg.score(X_test,y_test)
```

```
-56.28157146012308
```

```
reg.predict([[15,9,8]])
```

```
array([[55428.61574254]])
```

```
y_pred
```

```
array([[76456.14579294],
       [43899.88367584]])
```

```
reg.intercept_
```

```
array([131271.42303218])
```

```
reg.coef_
```

```
array([[   61.41915471, -6522.91585886, -2257.23148507]])
```

```
reg.predict([[2,9,6]])
```

```
array([[59144.62970143]])
```

```
from sklearn import metrics
print('MAE:',metrics.mean_absolute_error(y_test,y_pred))
print('MSE:',metrics.mean_squared_error(y_test,y_pred))
print('RMSE:',np.sqrt(metrics.mean_absolute_error(y_test,y_pred)))
```

```
MAE: 18778.131058549836
MSE: 358009821.62576926
RMSE: 137.03332097905908
```

```
# ML Class 5
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
social=pd.read_csv('Social_Network_Ads.csv')
```

```
social
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
social.isna().sum()
```

```
User ID           0
Gender            0
Age               0
EstimatedSalary   0
Purchased         0
dtype: int64
```

```
#consider statistical columns only
X=social.iloc[:,[2,3]].values
y=social.iloc[:,2:4]
X
```

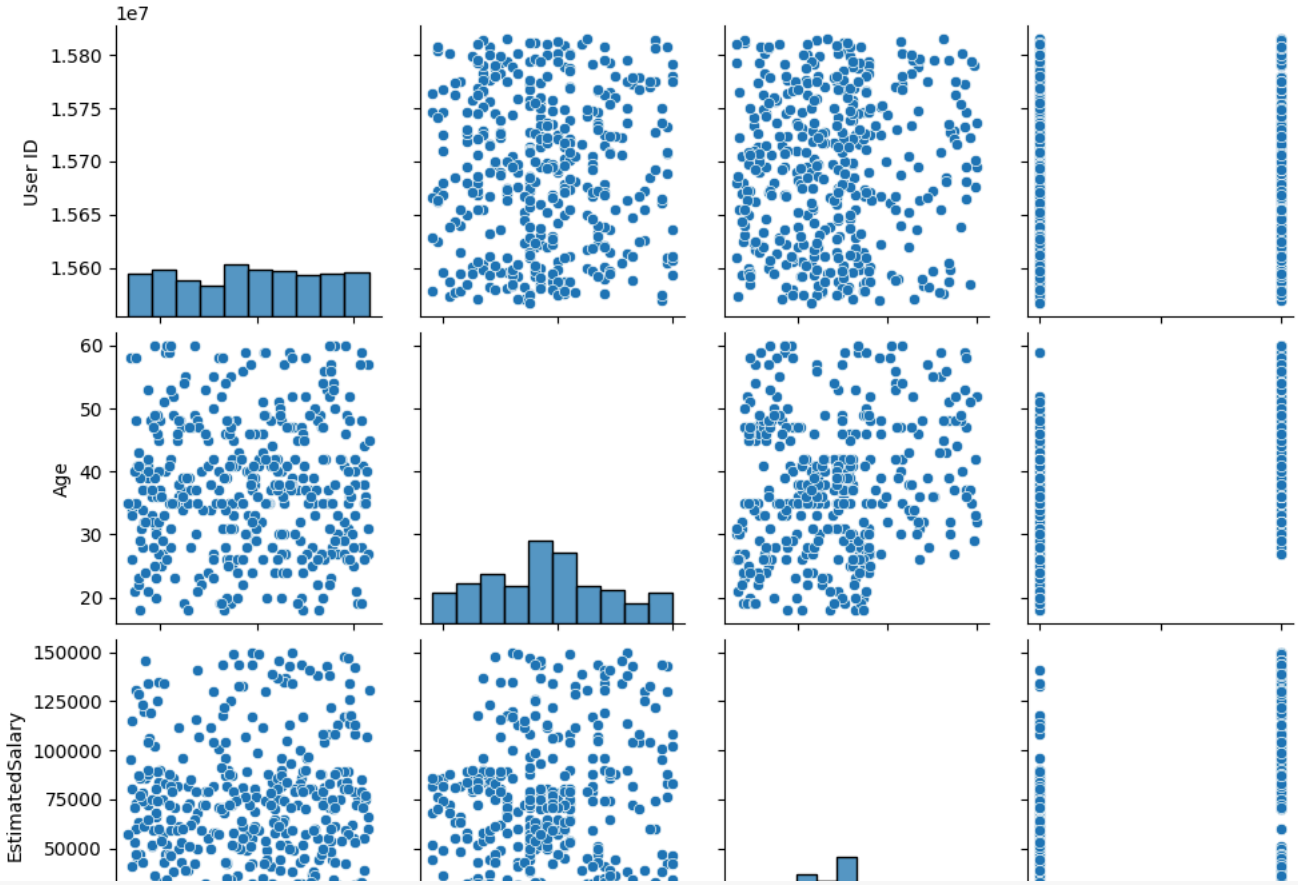
```
array([[   19, 19000],
       [   35, 20000],
       [   26, 43000],
       [   27, 57000],
       [   19, 76000],
       [   27, 58000],
       [   27, 84000],
       [   32, 150000],
       [   25, 33000],
       [   35, 65000],
       [   26, 80000],
       [   26, 52000],
       [   20, 86000],
       [   32, 18000],
       [   18, 82000],
       [   29, 80000],
       [   47, 25000],
       [   45, 26000],
       [   46, 28000],
       [   48, 29000],
       [   45, 22000],
       [   47, 49000],
       [   48, 41000],
       [   45, 22000],
       [   46, 23000],
       [   47, 20000],
       [   49, 28000],
       [   47, 30000],
       [   29, 43000],
       [   31, 18000],
       [   31, 74000],
       [   27, 137000],
       [   21, 16000],
       [   28, 44000],
       [   27, 90000],
       [   35, 27000],
       [   33, 28000],
       [   30, 49000],
       [   26, 72000],
       [   27, 31000],
       [   27, 17000],
       [   33, 51000],
       [   35, 108000],
       [   30, 15000],
       [   28, 84000],
       [   23, 20000],
       [   25, 79000],
       [   27, 54000],
       [   30, 135000],
       [   31, 89000],
       [   24, 32000],
       [   18, 44000],
       [   29, 83000],
       [   35, 23000],
       [   27, 58000],
       [   24, 55000],
       [   23, 48000],
       [   28, 79000],
```

```
len(y)
```

```
400
```

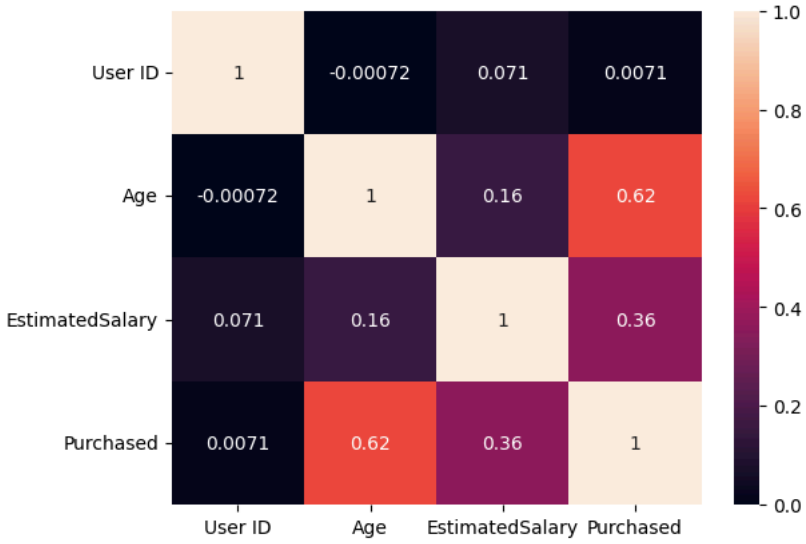
```
sns.pairplot(social)
```

```
<seaborn.axisgrid.PairGrid at 0x7fdea2346bf0>
```




```
sns.heatmap(social.corr(),annot=True)
```

```
<ipython-input-12-c1956e9c5d06>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fa
  sns.heatmap(social.corr(),annot=True)
<Axes: >
```




```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=2)
```

```
#feature scaling of independent variables for accurate predictions
from sklearn.preprocessing import StandardScaler
from sklearn.utils.validation import column_or_1d
from sklearn.linear_model import LogisticRegression
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
y_train=column_or_1d(y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-28-ce39f14ed562> in <cell line: 8>()
      6 X_train=sc.fit_transform(X_train)
      7 X_test=sc.transform(X_test)
----> 8 y_train=column_or_1d(y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in column_or_1d(y, dtype, warn)
   1200         return _asarray_with_order(xp.reshape(y, -1), order="C", xp=xp)
   1201
-> 1202     raise ValueError(
   1203         "y should be a 1d array, got an array of shape {} instead.".format(shape)
   1204     )

ValueError: y should be a 1d array, got an array of shape (300, 2) instead.
```

X_train

```
array([[ 0.03172569,  0.06201266],
       [-0.06539376, -0.50084108],
       [-0.64811043, -1.50805304],
       [ 0.03172569,  0.32862759],
       [ 0.32308402,  0.09163654],
       [-0.45387154, -1.12294258],
       [-0.74522987, -1.53767692],
       [-0.25963265, -0.64896049],
       [-1.13370765,  0.50637087],
       [-0.06539376,  2.22455597],
       [ 0.03172569,  0.06201266],
       [-1.13370765, -1.5673008 ],
       [ 1.10003958,  0.56561863],
       [-0.25963265, -1.24143811],
       [ 1.39139791, -0.91557542],
       [-1.42506599, -1.21181423],
       [-0.93946876, -0.9451993 ],
       [ 1.97411458, -0.64896049],
       [ 0.90580069, -0.56008884],
       [-1.13370765,  0.32862759],
       [ 0.03172569, -0.23422615],
       [ 0.80868124, -1.38955751],
       [-0.25963265, -0.35272168],
       [ 0.90580069,  1.30621566],
       [ 0.32308402, -0.17497839],
       [-0.25963265, -0.56008884],
       [-0.25963265, -1.38955751],
       [ 1.48851736, -1.03407094],
       [-0.06539376,  0.1508843 ],
       [-0.84234932, -0.64896049],
       [-0.06539376,  0.03238878],
       [-0.25963265,  0.12126042],
       [ 0.22596457, -0.29347391],
       [-0.25963265,  0.29900371],
       [ 0.12884513,  0.06201266],
       [ 1.97411458,  2.22455597],
       [-1.03658821, -1.44880527],
       [ 0.32308402,  0.32862759],
       [ 2.07123403, -1.18219034],
       [-1.13370765, -0.50084108],
       [ 0.22596457,  0.18050818],
       [-0.25963265, -0.91557542],
       [-0.64811043,  0.06201266],
       [ 0.22596457,  0.09163654],
       [ 0.42020346, -0.11573063],
       [-1.13370765, -1.0933187 ],
       [-0.06539376,  2.28380373],
       [ 1.10003958, -0.11573063],
       [ 0.90580069,  1.06922461],
       [-0.06539376,  0.29900371],
       [-0.55099098, -1.50805304],
       [-1.13370765, -1.00444706],
       [-0.74522987,  1.3950873 ],
       [ 1.10003958,  0.59524252],
       [ 1.5856368 , -1.27106199],
       [ 0.7115618 , -1.38955751],
       [-0.64811043, -1.03407094],
       [ 1.19715902,  0.56561863],
```

#fitting logisitc regression to training dataset

```
from sklearn.linear_model import LogisticRegression
clf=LogisticRegression(random_state=10)

clf.fit(X_train,y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-20-4050a230a1a3> in <cell line: 3>()
      1 from sklearn.linear_model import LogisticRegression
      2 clf=LogisticRegression(random_state=10)
----> 3 clf.fit(X_train,y_train)

 4 frames
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in column_or_1d(y, dtype, warn)
   1200         return _asarray_with_order(xp.reshape(y, -1), order="C", xp=xp)
   1201
-> 1202     raise ValueError(
   1203         "y should be a 1d array, got an array of shape {} instead.".format(shape)
   1204     )

ValueError: y should be a 1d array, got an array of shape (300, 2) instead.
```

```
#predicting the Test set result
y_predict=clf.predict(X_test)
y_pred
```

```
---------------------------------------------------------------------------
NotFittedError                            Traceback (most recent call last)
<ipython-input-21-43c66b64e670> in <cell line: 1>()
----> 1 y_predict=clf.predict(X_test)
      2 y_pred

 2 frames
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in check_is_fitted(estimator, attributes, msg, all_or_any)
   1388
   1389     if not fitted:
-> 1390         raise NotFittedError(msg % {"name": type(estimator).__name__})
   1391
   1392

NotFittedError: This LogisticRegression instance is not fitted yet. Call 'fit' with appropriate arguments before using this estimator.
```

```
#to get accuracy
from sklearn import metrics
print(metrics.accuracy_score(y_test,y_pred))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-29-75f85745b50e> in <cell line: 3>()
      1 #to get accuracy
      2 from sklearn import metrics
----> 3 print(metrics.accuracy_score(y_test,y_pred))

NameError: name 'y_pred' is not defined
```

```
#confusion matrix evaluation - to check the accuracy of classification
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-30-1e33d9b4afb3> in <cell line: 3>()
      1 #confusion matrix evaluation - to check the accuracy of classification
      2 from sklearn.metrics import confusion_matrix
----> 3 cm=confusion_matrix(y_test,y_pred)

NameError: name 'y_pred' is not defined
```

```
#Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-31-e0139656e6af> in <cell line: 3>()
      1 #Classification Report
      2 from sklearn.metrics import classification_report
----> 3 print(classification_report(y_test,y_pred))

NameError: name 'y_pred' is not defined
```

```
len(X_test)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-18eead9239cb> in <cell line: 1>()
----> 1 len(X_test)

NameError: name 'X_test' is not defined
```

```
#Multiclass Logistic Regression
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
from sklearn.datasets import load_iris
iris=load_iris()
iris
```

```
{'data': 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],
       [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],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
```

```
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
```

```
iris.feature_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
iris.target
```

```
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, 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])
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
iris_data=pd.DataFrame(iris.data,columns=iris.feature_names)
iris_data
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
iris_data['species']=iris.target
iris_data['species_name']=iris_data.species.apply(lambda x:iris.target_names[x])
iris_data
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	species_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

150 rows × 6 columns

```
X=iris_data.drop(['species','species_name'],axis=1)
X
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

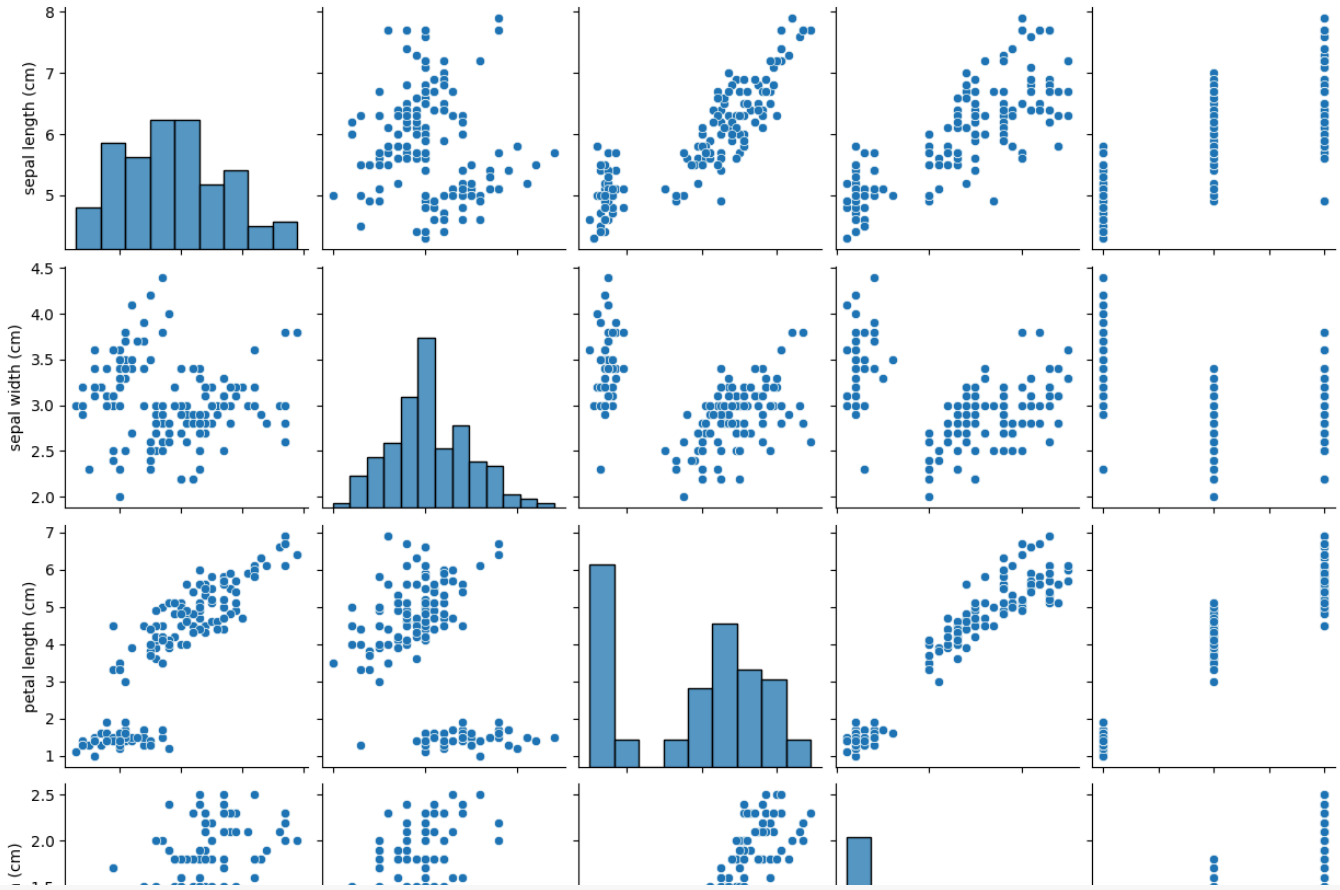
150 rows × 4 columns

```
y=iris_data['species']
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: species, Length: 150, dtype: int64
```

```
sns.pairplot(iris_data)
```

```
<seaborn.axisgrid.PairGrid at 0x7c70185d4460>
```



```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=19)
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(X_train,y_train)
```

```
LogisticRegression()
```

```
y_pred=model.predict(X_test)
y_pred
```

```
array([0, 2, 1, 1, 0, 0, 0, 1, 2, 1, 0, 1, 0, 2, 0, 2, 0, 1, 0, 1, 1,
       1, 1, 2, 1, 2, 2, 1, 2])
```

```
model.score(X_test,y_test)
```

```
1.0
```

```
model.predict(iris.data[0:65])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature
  warnings.warn(
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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

```
iris.target[0:65]
```

```
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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

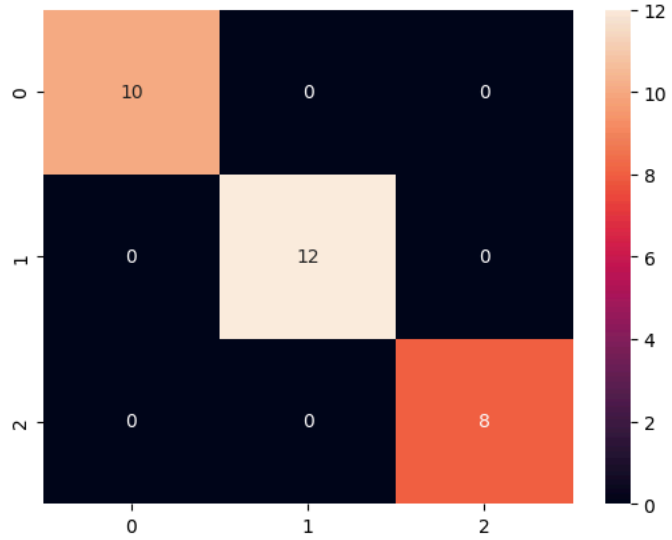
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
```

```
cm
```

```
array([[10,  0,  0],
       [ 0, 12,  0],
       [ 0,  0,  8]])
```

```
sns.heatmap(cm,annot=True)
```

```
<Axes: >
```




```
#to get accuracy
from sklearn import metrics
print(metrics.accuracy_score(y_test,y_pred))
```

```
1.0
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00         8

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

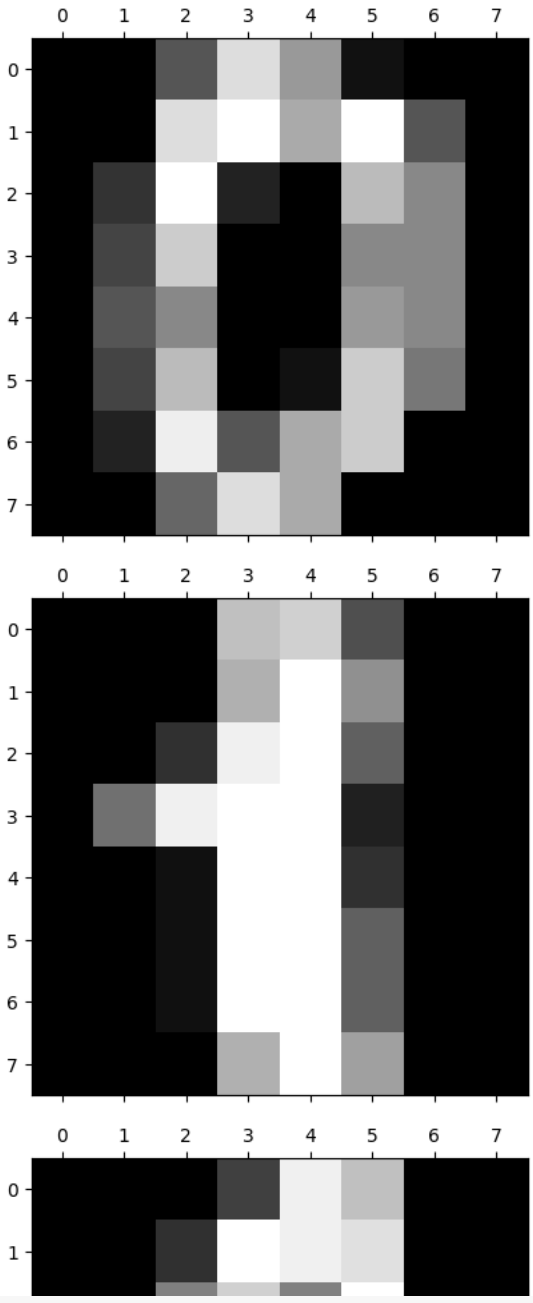
```
#Digital Dataset
```

```
from sklearn.datasets import load_digits
digi=load_digits()
digi
```

```
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  2., ..., 12.,  0.,  0.],
        [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'frame': None,
 'feature_names': ['pixel_0_0',
  'pixel_0_1',
  'pixel_0_2',
  'pixel_0_3',
  'pixel_0_4',
  'pixel_0_5',
  'pixel_0_6',
  'pixel_0_7',
  'pixel_1_0',
  'pixel_1_1',
  'pixel_1_2',
  'pixel_1_3',
  'pixel_1_4',
  'pixel_1_5',
  'pixel_1_6',
  'pixel_1_7',
  'pixel_2_0',
  'pixel_2_1',
  'pixel_2_2',
  'pixel_2_3',
  'pixel_2_4',
  'pixel_2_5',
  'pixel_2_6',
  'pixel_2_7',
  'pixel_3_0',
  'pixel_3_1',
  'pixel_3_2',
  'pixel_3_3',
  'pixel_3_4',
  'pixel_3_5',
  'pixel_3_6',
  'pixel_3_7',
  'pixel_4_0',
  'pixel_4_1',
  'pixel_4_2',
  'pixel_4_3',
  'pixel_4_4',
  'pixel_4_5',
  'pixel_4_6',
  'pixel_4_7',
  'pixel_5_0',
  'pixel_5_1',
  'pixel_5_2',
  'pixel_5_3',
  'pixel_5_4',
  'pixel_5_5',
  'pixel_5_6',
  'pixel_5_7',
  'pixel_6_0',
```

```
plt.gray()
for i in range(5):
  plt.matshow(digi.images[i])
```

```
<Figure size 640x480 with 0 Axes>
```



```
digidata=pd.DataFrame(digi.data,columns=digi.feature_names)
digidata
```

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pixel_6_
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	..
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	4.0	0.
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	1.0	0.
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	8.0	0.

1797 rows × 64 columns

```
digidata['out']=digi.target
digidata
```

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_7	pixel_7_
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	..
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	0.0	0.
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	0.0	0.
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	0.0	0.

1797 rows × 65 columns

```
X=digidata.drop('out',axis='columns')
X
```

	pixel_0_0	pixel_0_1	pixel_0_2	pixel_0_3	pixel_0_4	pixel_0_5	pixel_0_6	pixel_0_7	pixel_1_0	pixel_1_1	...	pixel_6_6	pixel_6_
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	5.0	0.
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	9.0	0.
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	..
1792	0.0	0.0	4.0	10.0	13.0	6.0	0.0	0.0	0.0	1.0	...	4.0	0.
1793	0.0	0.0	6.0	16.0	13.0	11.0	1.0	0.0	0.0	0.0	...	1.0	0.
1794	0.0	0.0	1.0	11.0	15.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.
1795	0.0	0.0	2.0	10.0	7.0	0.0	0.0	0.0	0.0	0.0	...	2.0	0.
1796	0.0	0.0	10.0	14.0	8.0	1.0	0.0	0.0	0.0	2.0	...	8.0	0.

1797 rows × 64 columns

```
y=digidata.out
y
```

```
0       0
1       1
2       2
3       3
4       4
       ..
1792    9
1793    0
1794    8
1795    9
1796    8
Name: out, Length: 1797, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=9)
```

```
from sklearn.linear_model import LogisticRegression
dmodel=LogisticRegression()
dmodel.fit(X_train,y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

▾ LogisticRegression

LogisticRegression()

```
y_pred=dmodel.predict(X_test)
```

```
dmodel.score(X_test,y_test)
```

```
0.9611111111111111
```

```
dmodel.predict(digi.data[0:5])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature
  warnings.warn(
array([0, 1, 2, 3, 4])
```

```
digi.target[0:5]
```

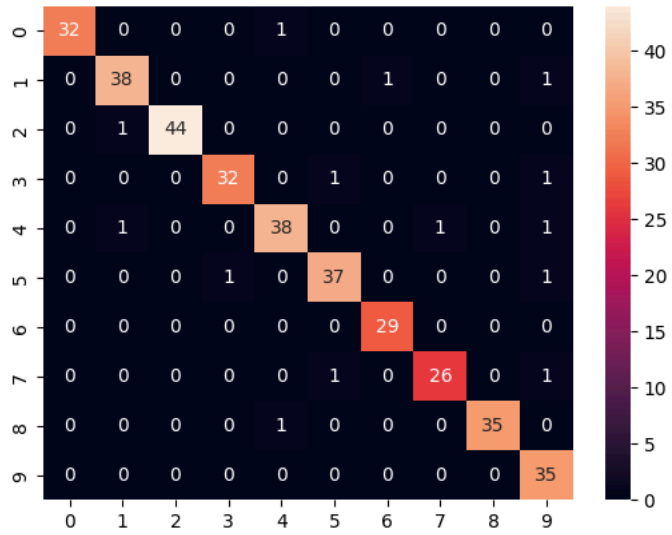
```
array([0, 1, 2, 3, 4])
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[32,  0,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0, 38,  0,  0,  0,  0,  1,  0,  0,  1],
       [ 0,  1, 44,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 32,  0,  1,  0,  0,  0,  1],
       [ 0,  1,  0,  0, 38,  0,  0,  1,  0,  1],
       [ 0,  0,  0,  1,  0, 37,  0,  0,  0,  1],
       [ 0,  0,  0,  0,  0,  0, 29,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  1,  0, 26,  0,  1],
       [ 0,  0,  0,  0,  1,  0,  0,  0, 35,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0, 35]])
```

```
sns.heatmap(cm,annot=True)
```

```
<Axes: >
```




```
#ML Class 6
#Decision Tree
#Classification Dataset
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
social=pd.read_csv('Social_Network_Ads.csv')
social
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
#consider statistical columns only
X=social.iloc[:,[2,3]].values
y=social.iloc[:,4].values
X
```

```
array([[   19,  19000],
       [   35,  20000],
       [   26,  43000],
       [   27,  57000],
       [   19,  76000],
       [   27,  58000],
       [   27,  84000],
       [   32, 150000],
       [   25,  33000],
       [   35,  65000],
       [   26,  80000],
       [   26,  52000],
       [   20,  86000],
       [   32,  18000],
       [   18,  82000],
       [   29,  80000],
       [   47,  25000],
       [   45,  26000],
       [   46,  28000],
       [   48,  29000],
       [   45,  22000],
       [   47,  49000],
       [   48,  41000],
       [   45,  22000],
       [   46,  23000],
       [   47,  20000],
       [   49,  28000],
       [   47,  30000],
       [   29,  43000],
       [   31,  18000],
       [   31,  74000],
       [   27, 137000],
       [   21,  16000],
       [   28,  44000],
       [   27,  90000],
       [   35,  27000],
       [   33,  28000],
       [   30,  49000],
       [   26,  72000],
       [   27,  31000],
       [   27,  17000],
       [   33,  51000],
       [   35, 108000],
       [   30,  15000],
       [   28,  84000],
       [   23,  20000],
       [   25,  79000],
       [   27,  54000],
       [   30, 135000],
```

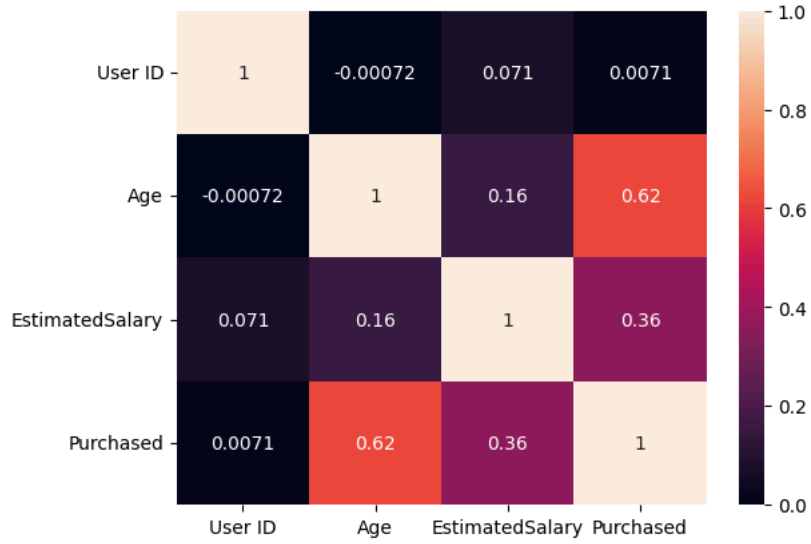
```
[   31,  89000],
[   24,  32000],
[   18,  44000],
[   29,  83000],
[   35,  23000],
[   27,  58000],
[   24,  55000],
[   23,  48000],
[   28,  79000],
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1])
```

```
sns.heatmap(social.corr(),annot=True)
```

```
<ipython-input-6-c1956e9c5d06>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fal
  sns.heatmap(social.corr(),annot=True)
<Axes: >
```




```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=2)

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)

X_train
```

```
array([[ 0.03172569,  0.06201266],
       [-0.06539376, -0.50084108],
       [-0.64811043, -1.50805304],
       [ 0.03172569,  0.32862759],
       [ 0.32308402,  0.09163654],
       [-0.45387154, -1.12294258],
       [-0.74522987, -1.53767692],
       [-0.25963265, -0.64896049],
       [-1.13370765,  0.50637087],
       [-0.06539376,  2.22455597],
       [ 0.03172569,  0.06201266],
       [-1.13370765, -1.5673008 ],
       [ 1.10003958,  0.56561863],
       [-0.25963265, -1.24143811],
```

```
       [ 1.39139791, -0.91557542],
       [-1.42506599, -1.21181423],
       [-0.93946876, -0.9451993 ],
       [ 1.97411458, -0.64896049],
       [ 0.90580069, -0.56008884],
       [-1.13370765,  0.32862759],
       [ 0.03172569, -0.23422615],
       [ 0.80868124, -1.38955751],
       [-0.25963265, -0.35272168],
       [ 0.90580069,  1.30621566],
       [ 0.32308402, -0.17497839],
       [-0.25963265, -0.56008884],
       [-0.25963265, -1.38955751],
       [ 1.48851736, -1.03407094],
       [-0.06539376,  0.1508843 ],
       [-0.84234932, -0.64896049],
       [-0.06539376,  0.03238878],
       [-0.25963265,  0.12126042],
       [ 0.22596457, -0.29347391],
       [-0.25963265,  0.29900371],
       [ 0.12884513,  0.06201266],
       [ 1.97411458,  2.22455597],
       [-1.03658821, -1.44880527],
       [ 0.32308402,  0.32862759],
       [ 2.07123403, -1.18219034],
       [-1.13370765, -0.50084108],
       [ 0.22596457,  0.18050818],
       [-0.25963265, -0.91557542],
       [-0.64811043,  0.06201266],
       [ 0.22596457,  0.09163654],
       [ 0.42020346, -0.11573063],
       [-1.13370765, -1.0933187 ],
       [-0.06539376,  2.28380373],
       [ 1.10003958, -0.11573063],
       [ 0.90580069,  1.06922461],
       [-0.06539376,  0.29900371],
       [-0.55099098, -1.50805304],
       [-1.13370765, -1.00444706],
       [-0.74522987,  1.3950873 ],
       [ 1.10003958,  0.59524252],
       [ 1.5856368 , -1.27106199],
       [ 0.7115618 , -1.38955751],
       [-0.64811043, -1.03407094],
       [ 1.19715902,  0.56561863],
```

```
from sklearn.tree import DecisionTreeClassifier
cf=DecisionTreeClassifier(random_state=1)
cf.fit(X_train,y_train)
y_pred
```

```
array([0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0])
```

```
y_pred=cf.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),y_test.reshape(len(y_test),1) ),1))
```

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

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

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

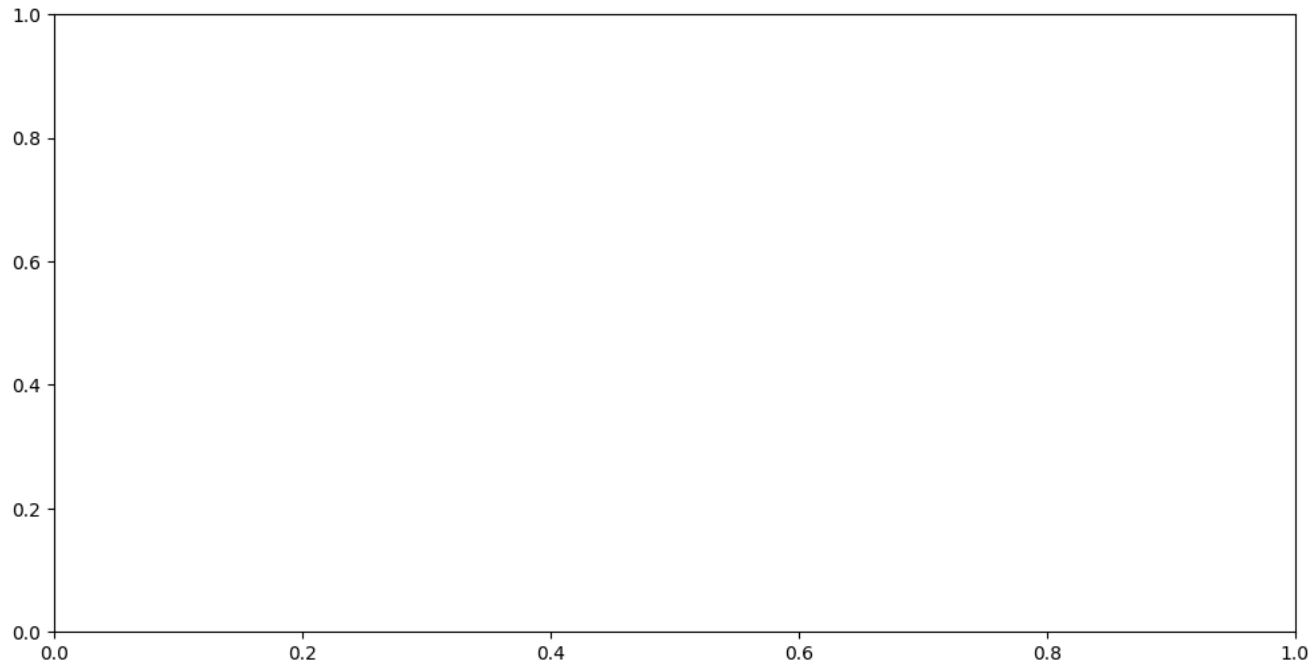
```
array([[54,  8],
       [ 7, 31]])
```

```
#from sklearn import metrics
#print(metrics.accuracy_score(y_test,pred))
from sklearn.metrics import accuracy_score
print('Accuracy of the Model:{}%'.format(accuracy_score(y_test,y_pred)*100))
```

```
Accuracy of the Model:85.0%
```

```
accuracy=[]
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

for i in range (1,10):
  model=DecisionTreeClassifier(max_depth=i,random_state=0)
  model.fit(X_train,y_train)
  pred=model.predict(X_test)
  score=accuracy_score(y_test,pred)
  accuracy.append(score)

  plt.figure(figsize=(12,6))
  plt.plot(range(1,10),accuracy,color='red',linestyle='dashed',marker='o',markerfacecolor='blue',markersize=10)
  plt.title('Finding best Max_depth')
  plt.xlabel('pred')
  plt.ylabel('score')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-6ff3b47e60ef> in <cell line: 6>()
     12
     13   plt.figure(figsize=(12,6))
---> 14   plt.plot(range(1,10),accuracy,color='red',linestyle='dashed',marker='o',markerfacecolor='blue',markersize=10)
     15   plt.title('Finding best Max_depth')
     16   plt.xlabel('pred')

                                 3 frames
/usr/local/lib/python3.10/dist-packages/matplotlib/axes/_base.py in _plot_args(self, tup, kwargs, return_kwargs, ambiguous_fmt_datakey)
    502
    503         if x.shape[0] != y.shape[0]:
--> 504             raise ValueError(f"x and y must have same first dimension, but "
    505                              f"have shapes {x.shape} and {y.shape}")
    506         if x.ndim > 2 or y.ndim > 2:

ValueError: x and y must have same first dimension, but have shapes (9,) and (1,)
```

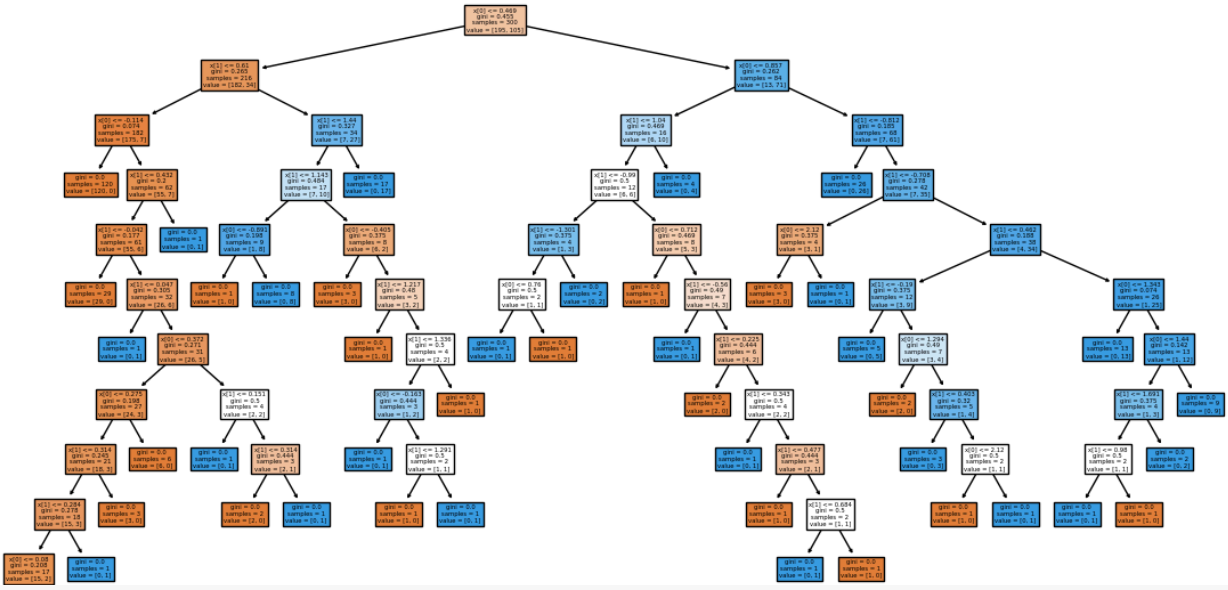



```
from sklearn.tree import DecisionTreeClassifier
cff=DecisionTreeClassifier(criterion='entropy',max_depth=2,random_state=6)
cff.fit(X_train,y_train)
pred=cff.predict(X_test)
```

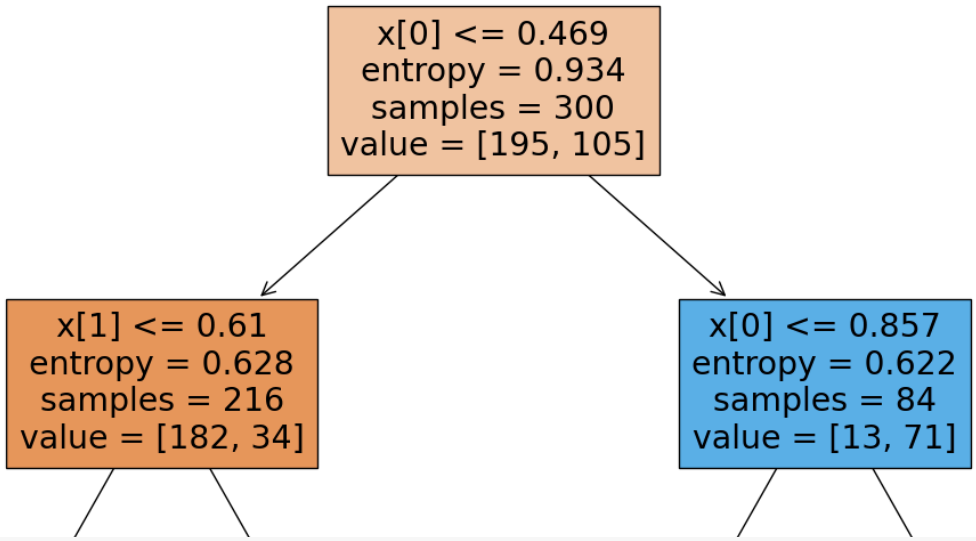
```
from sklearn import metrics
print(metrics.accuracy_score(y_test,pred))
```

```
0.92
```

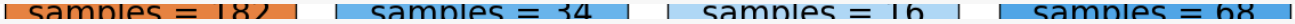
```
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(cf,filled=True)
plt.show()
```



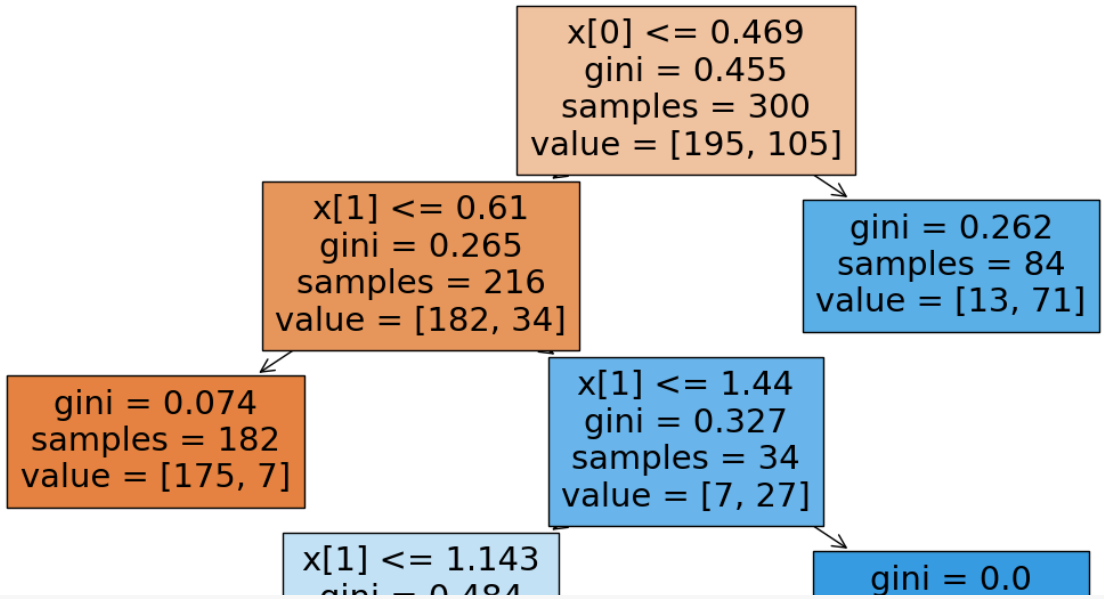
```
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(cff,filled=True)
plt.show()
```




```
from sklearn.tree import DecisionTreeClassifier
cff1=DecisionTreeClassifier(max_leaf_nodes=5,random_state=6)
cff1.fit(X_train,y_train)
cff1_pred=cff1.predict(X_test)
```



```
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(cff1,filled=True)
plt.show()
```

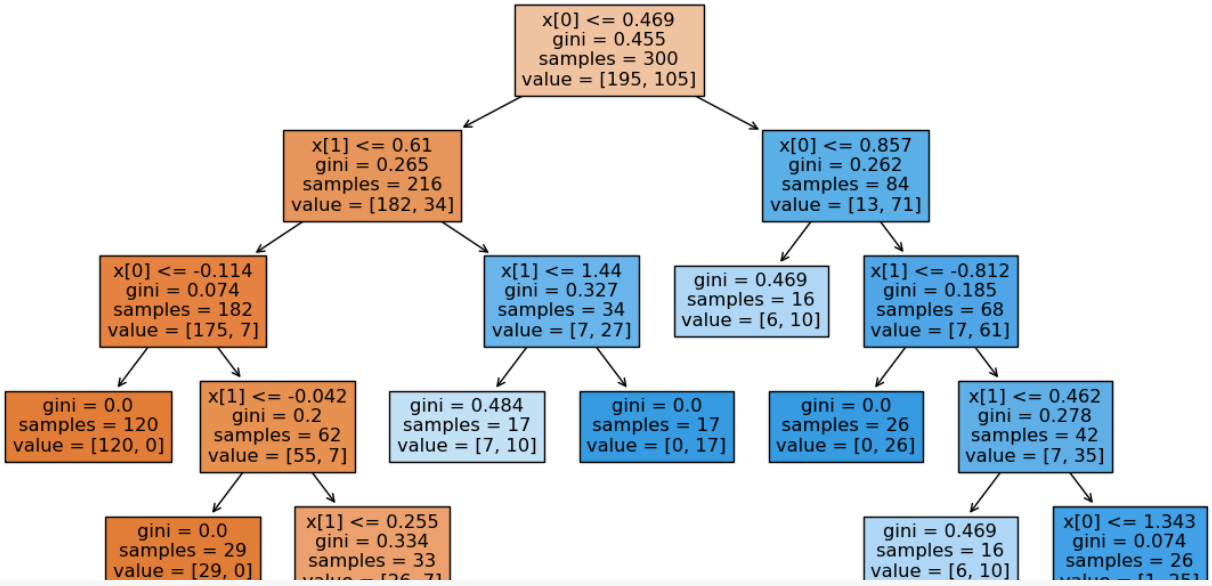



```
from sklearn import metrics
print(metrics.accuracy_score(y_test,cff1_pred))
```

```
0.9
```

```
from sklearn.tree import DecisionTreeClassifier
cff2=DecisionTreeClassifier(min_samples_leaf=10,random_state=6)
cff2.fit(X_train,y_train)
cff2.pred=cff2.predict(X_test)
```

```
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(cff2,filled=True)
plt.show()
```




```
from sklearn import metrics
print(metrics.accuracy_score(y_test,cff2.pred))
```

```
0.92
```

```
#Regression Dataset
```

```
sd=pd.read_csv('Salary_Data.csv')
sd
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

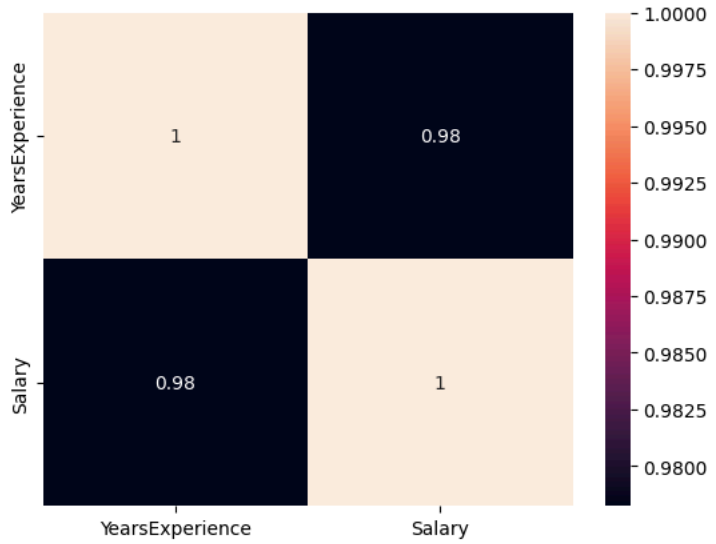
```
X=sd.iloc[:,:1].values
y=sd.iloc[:,1].values
X
```

```
array([[ 1.1],
       [ 1.3],
       [ 1.5],
       [ 2. ],
       [ 2.2],
       [ 2.9],
       [ 3. ],
       [ 3.2],
       [ 3.2],
       [ 3.7],
       [ 3.9],
       [ 4. ],
       [ 4. ],
       [ 4.1],
       [ 4.5],
       [ 4.9],
       [ 5.1],
       [ 5.3],
       [ 5.9],
       [ 6. ],
       [ 6.8],
       [ 7.1],
       [ 7.9],
       [ 8.2],
       [ 8.7],
       [ 9. ],
       [ 9.5],
```

```
 [ 9.6],
 [10.3],
 [10.5]])
```

```
sns.heatmap(sd.corr(),annot=True)
```

```
<Axes: >
```




```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=4)
len(X_train)
```

```
21
```

```
from sklearn.tree import DecisionTreeClassifier
reg=DecisionTreeClassifier(random_state=1)
reg.fit(X_train,y_train)
pred=reg.predict(X_test)
```

```
pred
```

```
array([ 56957., 101302., 112635.,  66029.,  81363., 109431.,  66029.,
       112635.,  81363.])
```

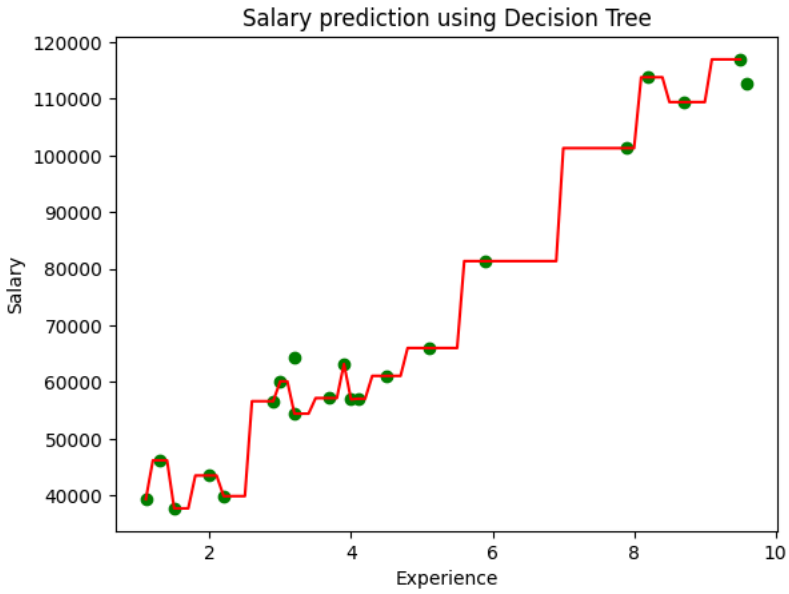
```
len(X_test)
```

```
9
```

```
y_test
```

```
array([ 55794.,  98273., 122391.,  67938.,  91738., 105582.,  83088.,
       121872.,  93940.])
```

```
X_val=np.arange(min(X_train),max(X_train),0.1)
X_val=X_val.reshape(len(X_val),1)
plt.scatter(X_train,y_train,color='green')
plt.plot(X_val,reg.predict(X_val),color='red')
plt.title('Salary prediction using Decision Tree')
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.figure()
plt.show()
```



<Figure size 640x480 with 0 Axes>

```
from sklearn import metrics
print('MAE',metrics.mean_absolute_error(y_test,pred))
print('MSE',metrics.mean_squared_error(y_test,pred))
print('RMSE',np.sqrt(metrics.mean_absolute_error(y_test,pred)))
print('R squared:',metrics.r2_score(y_test,pred)*100)
```

```
MAE 7661.555555555556
MSE 85146581.33333333
RMSE 87.53031220985994
R squared: 80.81898259456418
```

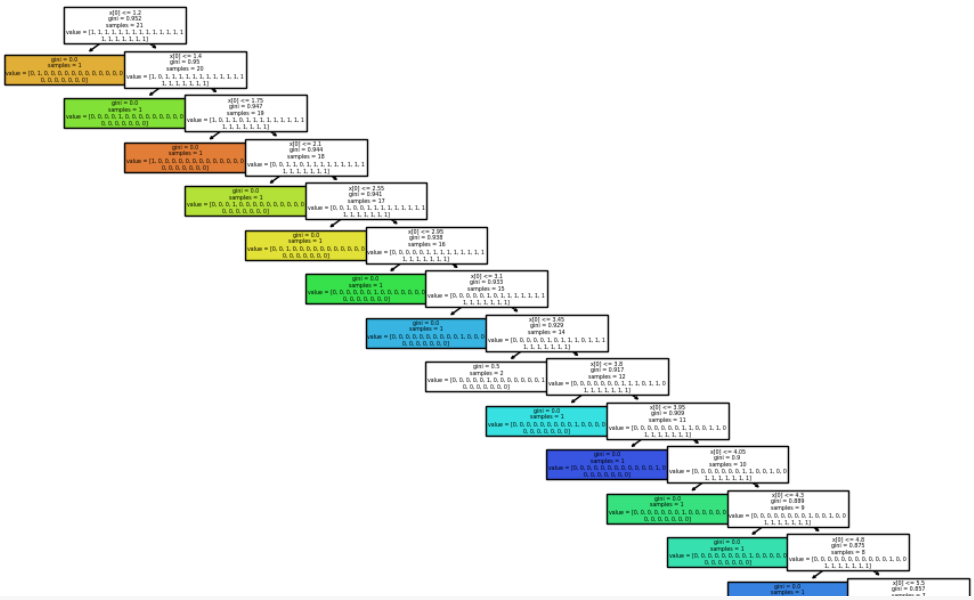
```
reg.score(X_test,y_test)
```

```
0.0
```

```
reg.score(X_train,y_train)
```

```
0.9523809523809523
```

```
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(reg,filled=True)
plt.show()
```



```
score=[]
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt

for i in range(2,11):
  model=DecisionTreeRegressor(min_samples_split=i,random_state=0)
  model.fit(X_train,y_train)
  y_pred=model.predict(X_test)
  score=r2_score(y_test,y_pred)
  score.append(r2_score(y_test,y_pred))

plt.figure(figsize=(12,6))
plt.plot(range(1,10),score,color='red',linestyle='dashed',marker='o',markerfacecolor='blue',markersize=10)
plt.title('Finding best min sample split')
plt.xlabel('pred')
plt.ylabel('score')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-49-cdb009048506> in <cell line: 6>()
      9   y_pred=model.predict(X_test)
     10   score=r2_score(y_test,y_pred)
---> 11   score.append(r2_score(y_test,y_pred))
     12
     13 plt.figure(figsize=(12,6))

AttributeError: 'numpy.float64' object has no attribute 'append'
```

```
from sklearn.tree import DecisionTreeRegressor
reg1=DecisionTreeRegressor(min_samples_split=3,criterion='mae',random_state=1)
reg1.fit(X_train,y_train)
pred=reg1.predict(X_test)
reg1.score(X_test,y_test)
```

```
---------------------------------------------------------------------------
InvalidParameterError                     Traceback (most recent call last)
<ipython-input-50-b12265b8e620> in <cell line: 3>()
      1 from sklearn.tree import DecisionTreeRegressor
      2 reg1=DecisionTreeRegressor(min_samples_split=3,criterion='mae',random_state=1)
----> 3 reg1.fit(X_train,y_train)
      4 pred=reg1.predict(X_test)
      5 reg1.score(X_test,y_test)

                                    3 frames
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py in validate_parameter_constraints(parameter_constraints, params, caller_name)
     95                 )
     96
---> 97             raise InvalidParameterError(
     98                 f"The {param_name!r} parameter of {caller_name} must be"
     99                 f" {constraints_str}. Got {param_val!r} instead."

InvalidParameterError: The 'criterion' parameter of DecisionTreeRegressor must be a str among {'poisson', 'absolute_error', 'friedman_mse', 'squared_error'}. Got 'mae' instead.
```

```
#Height Prediction
```

```
hd=pd.read_csv('heightdataset.csv')
hd.head()

hd.shape

hd.isna().sum()

X=hd.iloc[:,:-1].values
X

y=hd.iloc[:,-1].values
y

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=0)

len(X_train)

from sklearn.tree import DecisionTreeRegressor
htmodel=DecisionTreeRegressor()
htmodel.fit(X_train,y_train)

X_val=np,arange(min(X),max(X_train),0.01)
X_val=X_val.reshape((len(X_val),1))
plt.scatter(X_train,y_train,color='blue')
plt.plot(X_val,htmodel.predict(X_val),color='purple')
plt.title('Height prediction using Decision Tree')
plt.xlabel('Age')
plt.ylabel('Height')
plt.figure()
plt.show()

y.pred=htmodel.predict(X_test)
from sklearn.metrics import r2_score,mean_squared_error
mae=mean_squared_error(y_test,ypred)
rmse=np.sqrt(mse)
print('Root Mean Square Error:',rmse)
r2score=r2_score(y_test,ypred)
print('R2Score',r2score*100)

from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(htmodel,filled=True)
plt.show()

from sklearn.tree import DecisionTreeRegressor
htmodel1=DecisionTreeRegressor(max_depth=4,random_state=8)
htmodel1.fit(X_train,y_train)
y_pred=htmodel1.predict(X_test)

mse1=mean_squared_erro(y_test,y_pred)
rmse1=np.sqrt(mse1)
print('Root Mean Squared Error:',rmse1)
r2score1=r2_score(y_test,y_pred)
print('R2Score:',r2score*100)

plt.figure(figsize=(15,10))
tree.plot_tree(htmodel1,filled=True)
plt.show()
#Dataset not shared...
```

```
  File "<ipython-input-52-bb1ba047d75f>", line 52
    print('Root Mean Squared Error:'rmse1)
                                    ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

```
#ML Class 7
#Random Forest
```

```
soc=pd.read_csv('Social_Network_Ads.csv')
soc
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
#consider statistical columns only
X=soc.iloc[:,[2,3]].values
y=soc.iloc[:,4].values
X
```

```
array([[   19, 19000],
       [   35, 20000],
       [   26, 43000],
       [   27, 57000],
       [   19, 76000],
       [   27, 58000],
       [   27, 84000],
       [   32, 150000],
       [   25, 33000],
       [   35, 65000],
       [   26, 80000],
       [   26, 52000],
       [   20, 86000],
       [   32, 18000],
       [   18, 82000],
       [   29, 80000],
       [   47, 25000],
       [   45, 26000],
       [   46, 28000],
       [   48, 29000],
       [   45, 22000],
       [   47, 49000],
       [   48, 41000],
       [   45, 22000],
       [   46, 23000],
       [   47, 20000],
       [   49, 28000],
       [   47, 30000],
       [   29, 43000],
       [   31, 18000],
       [   31, 74000],
       [   27, 137000],
       [   21, 16000],
       [   28, 44000],
       [   27, 90000],
       [   35, 27000],
       [   33, 28000],
       [   30, 49000],
       [   26, 72000],
       [   27, 31000],
       [   27, 17000],
       [   33, 51000],
       [   35, 108000],
       [   30, 15000],
       [   28, 84000],
       [   23, 20000],
       [   25, 79000],
       [   27, 54000],
       [   30, 135000],
       [   31, 89000],
       [   24, 32000],
       [   18, 44000],
       [   29, 83000],
       [   35, 23000],
       [   27, 58000],
       [   24, 55000],
       [   23, 48000],
       [   28, 79000],
```

```
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1])
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=2)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

```
X_test
```

```
array([[-0.84234932,  0.41749923],
       [-1.61930488, -1.5673008 ],
       [-0.06539376, -0.4712172 ],
       [-0.84234932,  0.18050818],
       [ 1.5856368 ,  0.03238878],
       [ 0.90580069, -1.44880527],
       [-1.42506599,  0.38787535],
       [-0.93946876, -1.0933187 ],
       [ 0.22596457, -0.26385003],
       [-0.1625132 ,  1.69132611],
       [-1.32794654, -1.35993363],
       [-1.52218543, -1.24143811],
       [-0.55099098,  1.42471118],
       [ 0.7115618 ,  0.29900371],
       [-0.25963265, -0.29347391],
       [-0.45387154, -0.76745601],
       [-1.81354377,  0.03238878],
       [ 2.16835347, -0.79707989],
       [ 1.48851736,  0.09163654],
       [-1.13370765, -0.76745601],
       [-0.06539376,  0.26937982],
       [ 0.7115618 , -0.70820825],
       [-0.74522987, -0.20460227],
       [-0.1625132 , -1.06369482],
       [-1.2308271 ,  0.32862759],
       [ 0.7115618 , -1.0933187 ],
       [ 0.7115618 , -1.38955751],
       [ 0.22596457, -0.35272168],
       [ 0.61444235,  2.07643657],
       [-0.55099098,  0.92110521],
       [-1.32794654, -0.41196944],
       [ 0.32308402, -0.50084108],
       [ 0.80868124, -1.35993363],
       [ 0.03172569, -0.29347391],
       [-0.55099098,  2.40229926],
       [-0.25963265,  0.65449028],
       [ 1.00292013,  0.1508843 ],
       [-0.25963265, -1.35993363],
       [-0.84234932, -0.76745601],
       [ 0.90580069,  1.12847237],
       [-0.25963265, -0.88595153],
       [ 0.42020346, -0.11573063],
       [-1.03658821,  0.56561863],
       [ 0.42020346,  0.32862759],
       [-1.91066321, -0.02685899],
       [ 0.90580069, -0.53046496],
       [-1.32794654,  0.44712311],
       [ 0.90580069, -1.03407094],
       [ 1.00292013,  1.48395895],
       [-1.13370765, -1.59692468],
       [-1.03658821,  0.80260968],
       [ 0.42020346,  2.37267538],
       [-0.35675209,  0.09163654],
       [ 0.12884513,  0.29900371],
       [-1.13370765,  1.45433506],
```

```
       [-1.52218543, -0.17497839],
       [ 0.32308402, -1.15256646],
       [ 0.42020346,  1.15809625],
```

```
from sklearn.ensemble import RandomForestClassifier
cfr=RandomForestClassifier(n_estimators=3,random_state=3)
cfr.fit(X_train,y_train)
pred=cfr.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,pred)
cm
```

```
array([[55,  7],
       [ 4, 34]])
```

```
len(y_pred)
```

```
9
```

```
from sklearn import metrics
print(metrics.accuracy_score(y_test,pred))
```

```
0.89
```

```
#from sklearn import metrics
#print(metrics.accuracy_score(y_test,pred))
np.shape(y_test)
np.shape(y_pred)
from sklearn.metrics import accuracy_score
print('Accuracy of the Model:{}%'.format(accuracy_score(y_test,y_pred)*100))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-83-3ebff7d76113> in <cell line: 6>()
      4 np.shape(y_pred)
      5 from sklearn.metrics import accuracy_score
----> 6 print('Accuracy of the Model:{}%'.format(accuracy_score(y_test,y_pred)*100))

                                  3 frames
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in check_consistent_length(*arrays)
    395     uniques = np.unique(lengths)
    396     if len(uniques) > 1:
--> 397         raise ValueError(
    398             "Found input variables with inconsistent numbers of samples: %r"
    399             % [int(l) for l in lengths]

ValueError: Found input variables with inconsistent numbers of samples: [100, 9]
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
#Random Forest Classifier

sd=pd.read_csv('Salary_Data.csv')
sd.shape
sd
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

```
X=sd.iloc[:,:1].values
y=sd.iloc[:,1].values
X
```

```
array([[ 1.1],
       [ 1.3],
       [ 1.5],
       [ 2. ],
       [ 2.2],
       [ 2.9],
       [ 3. ],
       [ 3.2],
       [ 3.2],
       [ 3.7],
       [ 3.9],
       [ 4. ],
       [ 4. ],
       [ 4.1],
       [ 4.5],
       [ 4.9],
       [ 5.1],
       [ 5.3],
       [ 5.9],
       [ 6. ],
       [ 6.8],
       [ 7.1],
       [ 7.9],
       [ 8.2],
       [ 8.7],
       [ 9. ],
       [ 9.5],
```

```
       [ 9.6],
       [10.3],
       [10.5]])
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=4)
```

```
X_train
```

```
array([[1.1],
       [2. ],
       [5.1],
       [3.9],
       [3. ],
       [4. ],
       [1.5],
       [2.2],
       [7.9],
       [4.1],
       [3.2],
       [3.7],
       [5.9],
       [8.7],
       [9.6],
       [3.2],
       [1.3],
       [2.9],
       [8.2],
       [4.5],
       [9.5]])
```

```
len(X_train)
```

```
21
```

```
from sklearn.ensemble import RandomForestRegressor
reg=RandomForestRegressor(n_estimators=10,random_state=8)
reg.fit(X_train,y_train)
pred=reg.predict(X_test)
```

```
pred
```

```
array([ 58117.5, 100559.1, 114048.2,  66175.8,  90589.6, 111259. ,
        68604. , 114048.2,  74737.6])
```

```
y_test
```

```
array([ 55794.,  98273., 122391.,  67938.,  91738., 105582.,  83088.,
       121872.,  93940.])
```

```
from sklearn import metrics
print('MAE',metrics.mean_absolute_error(y_test,pred))
print('MSE',metrics.mean_squared_error(y_test,pred))
print('RMSE',np.sqrt(metrics.mean_absolute_error(y_test,pred)))
```

```
MAE 7005.577777777778
MSE 84067776.2111111
RMSE 83.69932961367002
```

```
#Encoding of Categorical Features
```

```
pdata=pd.read_csv('StudentsPerformance.csv')
pdata
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75
...	...	...	...	...	...	...	...	...
995	female	group E	master's degree	standard	completed	88	99	95
996	male	group C	high school	free/reduced	none	62	55	55
997	female	group C	high school	free/reduced	completed	59	71	65
998	female	group D	some college	standard	completed	68	78	77
999	female	group D	some college	free/reduced	none	77	86	86

1000 rows × 8 columns

```
pdata.isna().sum()
```

```
gender                       0
race/ethnicity               0
parental level of education    0
lunch                        0
test preparation course        0
math score                   0
reading score                0
writing score                0
dtype: int64
```

```
pdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       1000 non-null   object
 1   race/ethnicity               1000 non-null   object
 2   parental level of education  1000 non-null   object
 3   lunch                        1000 non-null   object
 4   test preparation course      1000 non-null   object
 5   math score                   1000 non-null   int64
 6   reading score                1000 non-null   int64
 7   writing score                1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

```
pdata.dtypes.value_counts()
```

```
object    5
int64     3
dtype: int64
```

```
score=pdata[['writing score','reading score','math score']]
pdata['Total Score']=score.sum(axis=1)
pdata
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	Total Score
0	female	group B	bachelor's degree	standard	none	72	72	74	218
1	female	group C	some college	standard	completed	69	90	88	247
2	female	group B	master's degree	standard	none	90	95	93	278
3	male	group A	associate's degree	free/reduced	none	47	57	44	148
4	male	group C	some college	standard	none	76	78	75	229
...	...	...	...	...	...	...	...	...	...
995	female	group E	master's degree	standard	completed	88	99	95	282
996	male	group C	high school	free/reduced	none	62	55	55	172
997	female	group C	high school	free/reduced	completed	59	71	65	195
998	female	group D	some college	standard	completed	68	78	77	223
999	female	group D	some college	free/reduced	none	77	86	86	249

1000 rows × 9 columns

```
pdata.head()
```

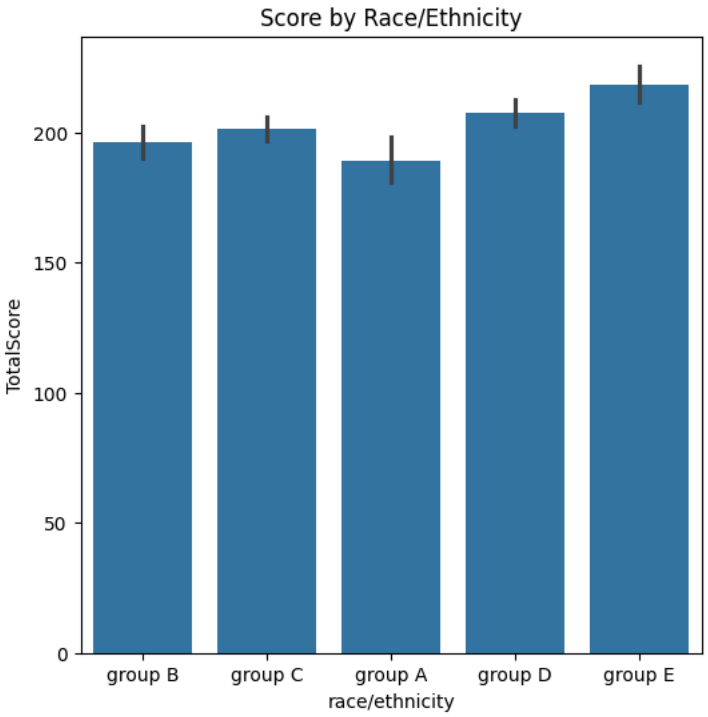
	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	Total Score
0	female	group B	bachelor's degree	standard	none	72	72	74	218
1	female	group C	some college	standard	completed	69	90	88	247
2	female	group B	master's degree	standard	none	90	95	93	278
3	male	group A	associate's degree	free/reduced	none	47	57	44	148
4	male	group C	some college	standard	none	76	78	75	229

```
pdata.columns
```

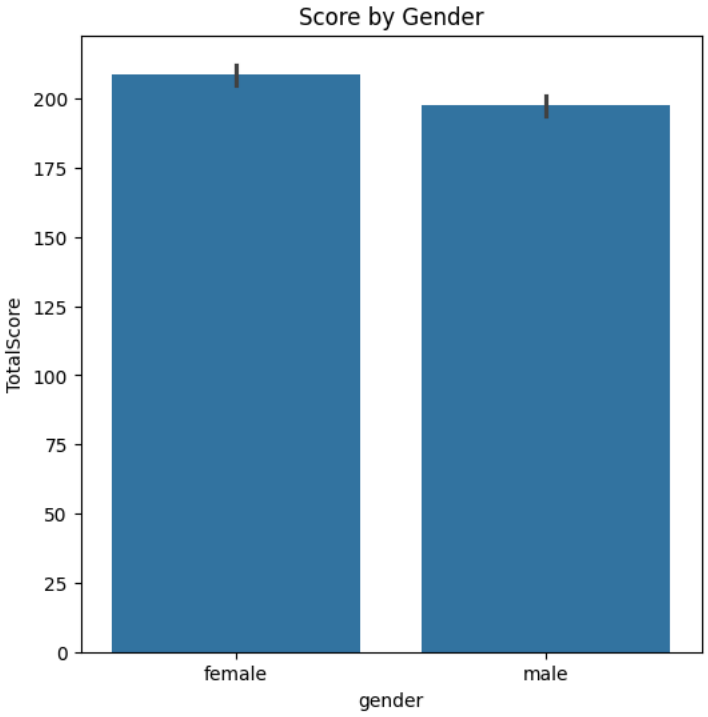
```
Index(['gender', 'race/ethnicity', 'parental level of education', 'lunch',
       'test preparation course', 'math score', 'reading score',
       'writing score', 'Total Score'],
      dtype='object')
```

```
pdata.rename(columns={"Total Score": "TotalScore"}, inplace=True)
```

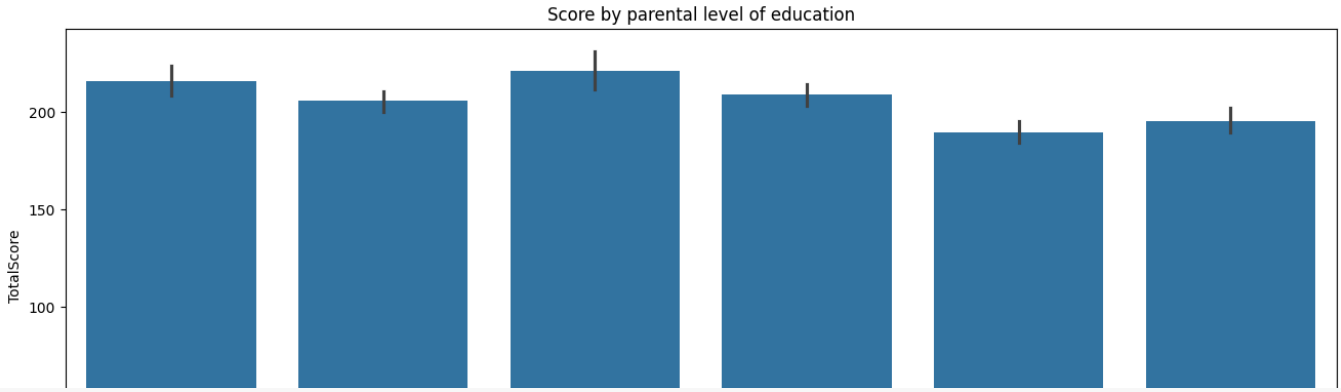
```
plt.figure(figsize=(6,6))
sns.barplot(x='race/ethnicity',y='TotalScore',data=pdata)
plt.title('Score by Race/Ethnicity')
plt.show()
```




```
plt.figure(figsize=(6,6))
sns.barplot(x='gender',y='TotalScore',data=pdata)
plt.title('Score by Gender')
plt.show()
```




```
plt.figure(figsize=(16,6))
sns.barplot(x='parental level of education',y='TotalScore',data=pdata)
plt.title('Score by parental level of education')
plt.show()
```




```
pdata=pd.read_csv('StudentsPerformance.csv')
pd['TotalScore'].describe()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-41-18dd9ff6d6b0> in <cell line: 2>()
      1 pdata=pd.read_csv('StudentsPerformance.csv')
----> 2 pd['TotalScore'].describe()

TypeError: 'module' object is not subscriptable
```

```
#Encoding nominal variable
```

```
#pandas get_dummies approach
```

```
gen=pd.get_dummies(pdata['gender'])
gen
```

	female	male
0	1	0
1	1	0
2	1	0
3	0	1
4	0	1
...	...	...
995	1	0
996	0	1
997	1	0
998	1	0
999	1	0

1000 rows × 2 columns

```
pdata1=pd.concat([pdata,gen],axis=1)
pdata1.drop(['gender','female'],inplace=True,axis=1)
pdata1.head()
```

	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	male
0	group B	bachelor's degree	standard	none	72	72	74	0
1	group C	some college	standard	completed	69	90	88	0
2	group B	master's degree	standard	none	90	95	93	0
3	group A	associate's degree	free/reduced	none	47	57	44	1
4	group C	some college	standard	none	76	78	75	1

```
#using OneHot Encoder
from sklearn.preprocessing import OneHotEncoder
one=OneHotEncoder(drop='first',sparse=False)
```

```
gen2=one.fit_transform(pdata[['gender']])
one.categories_
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and
  warnings.warn(
[array(['female', 'male'], dtype=object)]
```

```
gen2
```

```
array([[0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
```

```
#Encoding ordinal variable
```

```
print(pdata['parental level of education'].unique().tolist())
c=['some high school','high school','some college',"associate's degree","bachelor's degree","master's degree"]
```

```
["bachelor's degree", 'some college', "master's degree", "associate's degree", 'high school', 'some high school']
```

```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder(categories=[c])
```

```
edu=oe.fit_transform(pdata[['parental level of education']])
oe.categories_
```

```
[array(['some high school', 'high school', 'some college',
       "associate's degree", "bachelor's degree", "master's degree"],
      dtype=object)]
```

```
edu[:5]
```

```
array([[4.],
       [2.],
       [5.],
       [3.],
       [2.]])
```

```
pdata['parental level of education'].head()
```

```
0    bachelor's degree
1         some college
2      master's degree
3    associate's degree
4         some college
Name: parental level of education, dtype: object
```

```
from sklearn.compose import ColumnTransformer
ct=ColumnTransformer([('one',OneHotEncoder(drop='first'),['gender','race/ethnicity','lunch','test preparation course']),
                      ('oe',OrdinalEncoder(categories=[c]),['parental level of education'])],remainder='passthrough')
```

```
pdata3=ct.fit_transform(pdata)
pdata3[:5]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

                                 8 frames
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'test preparation'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
KeyError: 'test preparation'

The above exception was the direct cause of the following exception:

ValueError                                Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/__init__.py in _get_column_indices(X, key)
    454
    455         except KeyError as e:
--> 456             raise ValueError("A given column is not a column of the dataframe") from e
    457
    458         return column_indices

ValueError: A given column is not a column of the dataframe
```

```
pdata.head()
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female	group B	bachelor's degree	standard	none	72	72	74
1	female	group C	some college	standard	completed	69	90	88
2	female	group B	master's degree	standard	none	90	95	93
3	male	group A	associate's degree	free/reduced	none	47	57	44
4	male	group C	some college	standard	none	76	78	75

```
#Imputation
```

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

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

```
data.columns.isna().sum()
```

```
0
```

```
from sklearn.impute import SimpleImputer
```

```
df=data.iloc[:,:].values
df
```

```
array([[1.10000e+00, 3.93430e+04],
       [1.30000e+00, 4.62050e+04],
       [1.50000e+00, 3.77310e+04],
       [2.00000e+00, 4.35250e+04],
       [2.20000e+00, 3.98910e+04],
       [2.90000e+00, 5.66420e+04],
       [3.00000e+00, 6.01500e+04],
       [3.20000e+00, 5.44450e+04],
       [3.20000e+00, 6.44450e+04],
       [3.70000e+00, 5.71890e+04],
```

```
       [3.90000e+00, 6.32180e+04],
       [4.00000e+00, 5.57940e+04],
       [4.00000e+00, 5.69570e+04],
       [4.10000e+00, 5.70810e+04],
       [4.50000e+00, 6.11110e+04],
       [4.90000e+00, 6.79380e+04],
       [5.10000e+00, 6.60290e+04],
       [5.30000e+00, 8.30880e+04],
       [5.90000e+00, 8.13630e+04],
       [6.00000e+00, 9.39400e+04],
       [6.80000e+00, 9.17380e+04],
       [7.10000e+00, 9.82730e+04],
       [7.90000e+00, 1.01302e+05],
       [8.20000e+00, 1.13812e+05],
       [8.70000e+00, 1.09431e+05],
       [9.00000e+00, 1.05582e+05],
       [9.50000e+00, 1.16969e+05],
       [9.60000e+00, 1.12635e+05],
       [1.03000e+01, 1.22391e+05],
       [1.05000e+01, 1.21872e+05]])
```

```
df[:,1:]
```

```
array([[ 39343.],
       [ 46205.],
       [ 37731.],
       [ 43525.],
       [ 39891.],
       [ 56642.],
       [ 60150.],
       [ 54445.],
       [ 64445.],
       [ 57189.],
       [ 63218.],
       [ 55794.],
       [ 56957.],
       [ 57081.],
       [ 61111.],
       [ 67938.],
       [ 66029.],
       [ 83088.],
       [ 81363.],
       [ 93940.],
       [ 91738.],
       [ 98273.],
       [101302.],
       [113812.],
       [109431.],
       [105582.],
       [116969.],
       [112635.],
       [122391.],
       [121872.]])
```

```
s_imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
```

```
s_imputer=s_imputer.fit(df[:,1:])
df[:,1:]=s_imputer.transform(df[:,1:])
```

```
df
```

```
array([[1.10000e+00, 3.93430e+04],
       [1.30000e+00, 4.62050e+04],
       [1.50000e+00, 3.77310e+04],
       [2.00000e+00, 4.35250e+04],
       [2.20000e+00, 3.98910e+04],
       [2.90000e+00, 5.66420e+04],
       [3.00000e+00, 6.01500e+04],
       [3.20000e+00, 5.44450e+04],
       [3.20000e+00, 6.44450e+04],
       [3.70000e+00, 5.71890e+04],
       [3.90000e+00, 6.32180e+04],
       [4.00000e+00, 5.57940e+04],
       [4.00000e+00, 5.69570e+04],
       [4.10000e+00, 5.70810e+04],
       [4.50000e+00, 6.11110e+04],
       [4.90000e+00, 6.79380e+04],
       [5.10000e+00, 6.60290e+04],
       [5.30000e+00, 8.30880e+04],
       [5.90000e+00, 8.13630e+04],
       [6.00000e+00, 9.39400e+04],
       [6.80000e+00, 9.17380e+04],
       [7.10000e+00, 9.82730e+04],
       [7.90000e+00, 1.01302e+05],
       [8.20000e+00, 1.13812e+05],
       [8.70000e+00, 1.09431e+05],
```

```
       [9.00000e+00, 1.05582e+05],
       [9.50000e+00, 1.16969e+05],
       [9.60000e+00, 1.12635e+05],
       [1.03000e+01, 1.22391e+05],
       [1.05000e+01, 1.21872e+05]])
```

```
y=pd.DataFrame(df)
y
```

	0	1
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

```
#ML Class 8
#Support Vector Machine Algorithm
#SVM Classification
```

```
fish=pd.read_csv('Fish.csv')
fish
```

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340
...	...	...	...	...	...	...	...
154	Smelt	12.2	11.5	12.2	13.4	2.0904	1.3936
155	Smelt	13.4	11.7	12.4	13.5	2.4300	1.2690
156	Smelt	12.2	12.1	13.0	13.8	2.2770	1.2558
157	Smelt	19.7	13.2	14.3	15.2	2.8728	2.0672
158	Smelt	19.9	13.8	15.0	16.2	2.9322	1.8792

159 rows × 7 columns

```
fish.isna().sum()
```

```
Species    0
Weight     0
Length1    0
Length2    0
Length3    0
Height     0
Width      0
dtype: int64
```

```
len(fish.describe())
```

```
8
```

```
fish['Species'].value_counts()
```

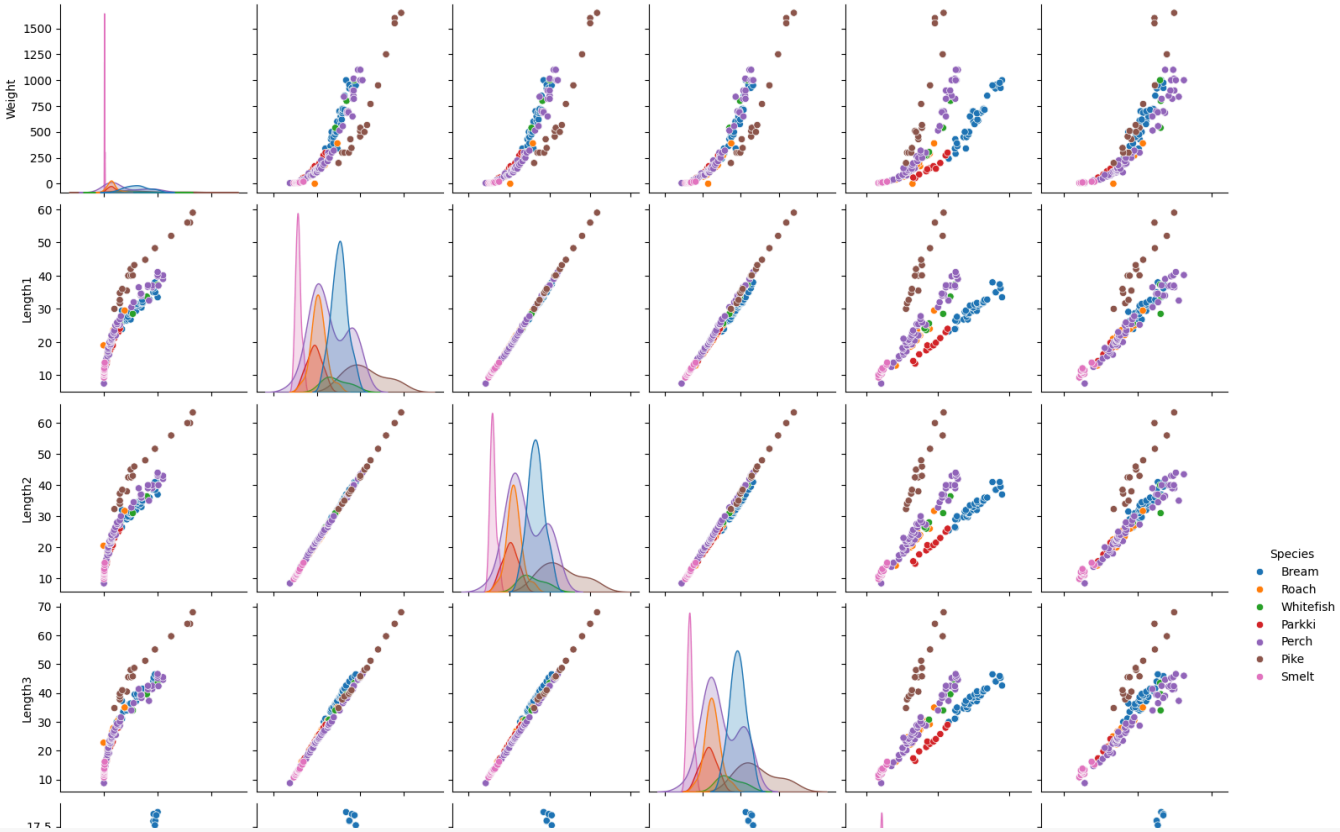
```
Perch        56
Bream        35
Roach        20
Pike         17
Smelt        14
Parkki       11
Whitefish     6
Name: Species, dtype: int64
```

```
fish.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159 entries, 0 to 158
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Species  159 non-null    object 
 1   Weight   159 non-null    float64
 2   Length1  159 non-null    float64
 3   Length2  159 non-null    float64
 4   Length3  159 non-null    float64
 5   Height   159 non-null    float64
 6   Width    159 non-null    float64
dtypes: float64(6), object(1)
memory usage: 8.8+ KB
```

```
sns.pairplot(fish,diag_kind='kde',hue='Species')
```

<seaborn.axisgrid.PairGrid at 0x7eba53f45db0>

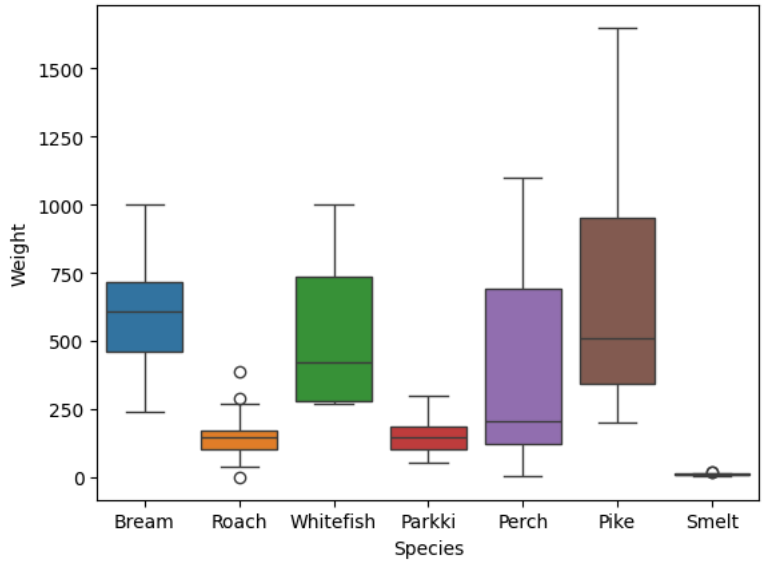



```
sns.countplot('Species',data=fish)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-105-98f49f7b9f4a> in <cell line: 1>()
----> 1 sns.countplot('Species',data=fish)

TypeError: countplot() got multiple values for argument 'data'
```

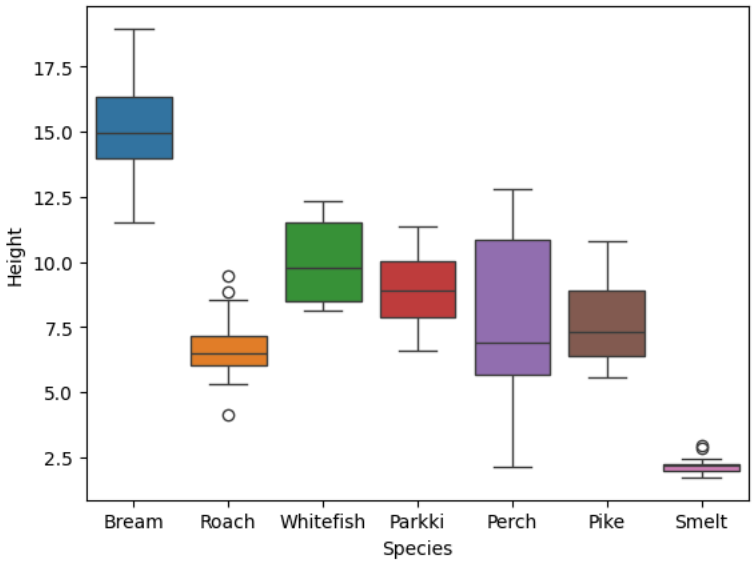
```
sns.boxplot(x='Species',y='Weight',data=fish,hue='Species')
```

<Axes: xlabel='Species', ylabel='Weight'>




```
sns.boxplot(x='Species',y='Height',data=fish,hue='Species')
```

```
<Axes: xlabel='Species', ylabel='Height'>
```




```
fish.columns
```

```
Index(['Species', 'Weight', 'Length1', 'Length2', 'Length3', 'Height',
       'Width'],
      dtype='object')
```

```
X=fish.drop(['Species'],axis=1)
y=fish['Species']
y
```

```
0      Bream
1      Bream
2      Bream
3      Bream
4      Bream
      ...
154    Smelt
155    Smelt
156    Smelt
157    Smelt
158    Smelt
Name: Species, Length: 159, dtype: object
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=7)
```

```
len(X_train)
```

```
127
```

```
len(y_test)
```

```
32
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

```
X_train
```

```
array([[-0.73816369, -0.52147279, -0.54804884, -0.53613593, -0.57579753,
        -0.63388048],
       [-0.37439272, -0.32836288, -0.32294469, -0.28632819,  0.4703526 ,
        -0.15304335],
       [-0.8202173 , -0.7145827 , -0.68311133, -0.74430904, -0.77347795,
        -0.49884686],
       [ 2.29781964,  2.47173084,  2.46834676,  2.35330693,  0.37774826,
         1.51485678],
       [-0.17746407,  0.92685155,  0.89261771,  0.79617202, -0.61129932,
        -0.2510895 ],
       [ 1.61403961,  1.05237299,  1.0276802 ,  1.00434513,  0.76220498,
         1.24484824],
       [ 0.12613426,  1.31307137,  1.25278435,  1.17088363, -0.40750983,
        -0.04824551],
```

```
[ 0.57469396,  0.49235425,  0.53245107,  0.687922  ,  1.49268887,
  0.51396406],
[ 3.11835568,  2.85795066,  2.8285134 ,  2.71136469,  0.12732233,
  1.0211621 ],
[ 0.84820598,  0.54063172,  0.5774719 ,  0.76286432,  1.68612752,
  0.98945855],
[-1.09728497, -1.50633334, -1.55651543, -1.56867459, -1.62973961,
 -1.83145262],
[-0.62875889, -0.2704299 , -0.30493636, -0.29465511, -0.45209454,
 -0.2927738 ],
[-0.8202173 , -0.7049272 , -0.70111966, -0.6943475 , -0.66392957,
 -0.59237232],
[-0.93235722, -0.95597009, -0.93522798, -0.92750139, -0.86435331,
 -0.92936928],
[ 0.10972354,  0.03854595,  0.10025111,  0.27157576,  1.05      ,
  0.30689293],
[-1.09454985, -1.54495532, -1.58352793, -1.61863613, -1.57952994,
 -1.77579528],
[ 0.79350357,  0.3861438 ,  0.39738858,  0.5879989 ,  1.35789734,
  0.46546937],
[ 1.17642039,  0.5889092 ,  0.5774719 ,  0.4880758 ,  0.56265726,
  1.99087967],
[-0.62875889, -0.32836288, -0.32294469, -0.41123206, -0.60127121,
 -0.42340415],
[ 1.3405276 ,  1.0234065 ,  1.0276802 ,  0.92107589,  0.61835357,
  1.65582015],
[-0.7928661 , -0.61802774, -0.59306967, -0.6610398 , -0.67723121,
 -0.58544451],
[ 1.47728361,  2.1144775 ,  2.08116762,  1.97026839, -0.02800953,
  1.0371313 ],
[-1.01167571, -1.21666847, -1.22336129, -1.28555915, -1.20422531,
 -1.15816322],
[ 0.05502114,  0.87857407,  0.84759688,  0.75453739, -0.40520451,
  0.10087858],
[-0.8475685 , -0.98493658, -0.95323631, -1.01909756, -0.88185062,
 -0.63587663],
[ 0.10972354,  0.11578992,  0.12726361,  0.30488346,  1.14278877,
  0.25781114],
[-0.9569733 , -1.16839099, -1.17834046, -1.1689822 , -0.56952708,
 -1.22732392],
[ 1.65506641,  1.0234065 ,  1.0276802 ,  0.91274896,  0.76838321,
  1.79519833],
[ 3.39186769,  3.14761553,  3.13465504,  3.04444167,  0.40672602,
```

```
from sklearn.svm import SVC
svmc=SVC(C=10,gamma=5)
svmc.fit(X_train,y_train)
y_pred=svmc.predict(X_test)
```

```
y_pred
```

```
array(['Perch', 'Pike', 'Roach', 'Perch', 'Perch', 'Perch', 'Perch',
       'Perch', 'Roach', 'Parkki', 'Bream', 'Perch', 'Bream', 'Smelt',
       'Perch', 'Parkki', 'Roach', 'Bream', 'Smelt', 'Bream', 'Pike',
       'Whitefish', 'Parkki', 'Bream', 'Roach', 'Perch', 'Perch', 'Roach',
       'Roach', 'Perch', 'Perch', 'Pike'], dtype=object)
```

```
svmc.score(X_test,y_test)*100
```

```
90.625
```

```
|from sklearn.svm import SVC
svmc1=SVC(kernel='linear')
svmc1.fit(X_train,y_train)
y_pred1=svmc1.predict(X_test)
```

```
y_pred1
```

```
array(['Perch', 'Pike', 'Perch', 'Perch', 'Perch', 'Perch', 'Perch',
       'Perch', 'Perch', 'Parkki', 'Bream', 'Perch', 'Bream', 'Smelt',
       'Perch', 'Parkki', 'Perch', 'Bream', 'Smelt', 'Bream', 'Pike',
       'Perch', 'Parkki', 'Bream', 'Perch', 'Perch', 'Perch', 'Perch',
       'Perch', 'Perch', 'Perch', 'Pike'], dtype=object)
```

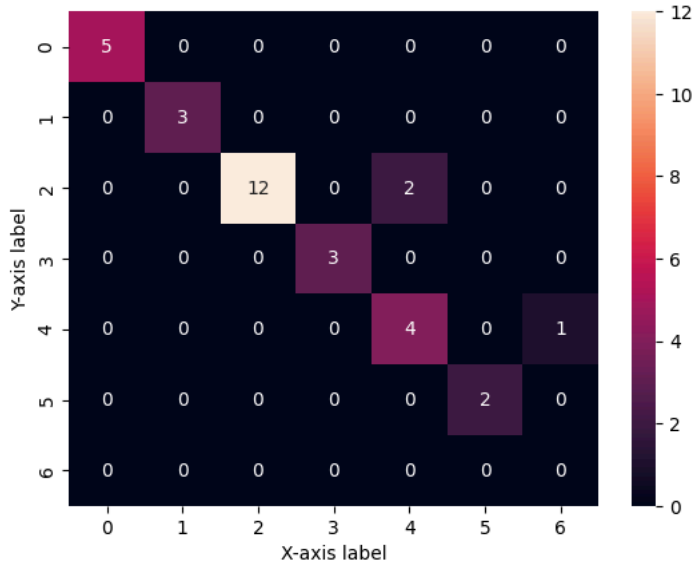
```
svmc1.score(X_test,y_test)*100
```

```
84.375
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
```

```
array([[ 5,  0,  0,  0,  0,  0,  0],
       [ 0,  3,  0,  0,  0,  0,  0],
       [ 0,  0, 12,  0,  2,  0,  0],
       [ 0,  0,  0,  3,  0,  0,  0],
       [ 0,  0,  0,  0,  4,  0,  1],
       [ 0,  0,  0,  0,  0,  2,  0],
       [ 0,  0,  0,  0,  0,  0,  0]])
```

```
import matplotlib.pyplot as plt

ax = sns.heatmap(cm, annot=True)
ax.set_xlabel("X-axis label")
ax.set_ylabel("Y-axis label")
plt.show()
```




```
#With another dataset
```

```
soc=pd.read_csv('Social_Network_Ads.csv')
soc
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
soc.isna().sum()
```

```
User ID          0
Gender           0
Age              0
EstimatedSalary  0
Purchased        0
dtype: int64
```

```
soc['Gender'].value_counts()
```

```
Female    204
Male      196
Name: Gender, dtype: int64
```

```
soc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

```
soc.describe()
```

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

```
X=soc.drop(['Gender'],axis=1)
y=soc['Gender']
y
```

```
0       Male
1       Male
2     Female
3     Female
4       Male
       ...
395   Female
396     Male
397   Female
398     Male
399   Female
Name: Gender, Length: 400, dtype: object
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=7)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

```
from sklearn.svm import SVC
svmc2=SVC(kernel='sigmoid',C=10,gamma=5)
svmc2.fit(X_train,y_train)
pred=svmc2.predict(X_test)
```

```
y_pred
```

```
array(['Female', 'Male', 'Female', 'Female', 'Female', 'Male', 'Female',
       'Female', 'Female', 'Male', 'Female', 'Female', 'Female', 'Female',
       'Female', 'Female', 'Male', 'Female', 'Female', 'Male', 'Male',
       'Male', 'Female', 'Female', 'Female', 'Female', 'Male', 'Male',
       'Male', 'Female', 'Female', 'Female', 'Male', 'Male', 'Male',
       'Male', 'Male', 'Female', 'Male', 'Male', 'Female', 'Male', 'Male',
       'Male', 'Male', 'Male', 'Male', 'Male', 'Female', 'Female', 'Male',
       'Female', 'Male', 'Male', 'Male', 'Male', 'Female', 'Female',
       'Male', 'Female', 'Female', 'Female', 'Male', 'Female', 'Female',
       'Male', 'Female', 'Female', 'Female', 'Female', 'Male', 'Male',
       'Female', 'Female', 'Male', 'Male', 'Female', 'Female', 'Female',
       'Male'], dtype=object)
```

```
svmc2.score(X_test,y_test)*100
```

56.25

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,pred)
cm
```

```
array([[23, 16],
       [22, 19]])
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

      Female       0.51      0.59      0.55        39
        Male       0.54      0.46      0.50        41

    accuracy                           0.53        80
   macro avg       0.53      0.53      0.52        80
weighted avg       0.53      0.53      0.52        80
```

```
#SVM Regression
sns.get_dataset_names()
```

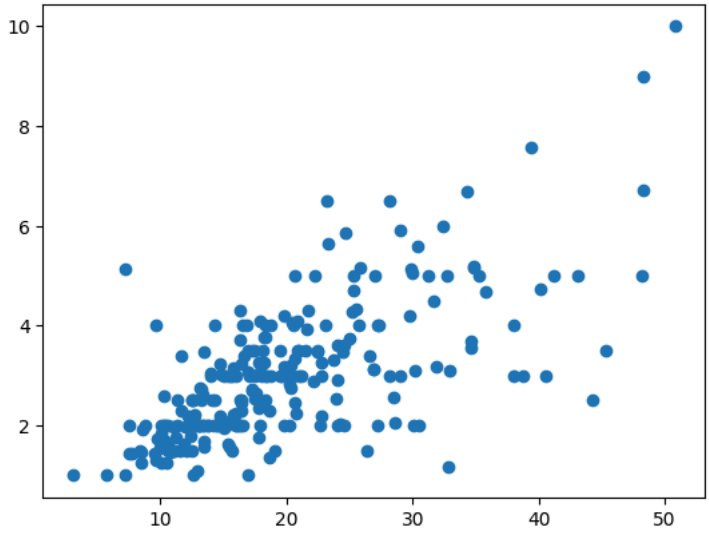
```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

```
g=sns.load_dataset('tips')
tip=g.drop(['sex','smoker','day','time'],axis=1)
tip
```

	total_bill	tip	size
0	16.99	1.01	2
1	10.34	1.66	3
2	21.01	3.50	3
3	23.68	3.31	2
4	24.59	3.61	4
...	...	...	...
239	29.03	5.92	3
240	27.18	2.00	2
241	22.67	2.00	2
242	17.82	1.75	2
243	18.78	3.00	2

244 rows × 3 columns

```
plt.scatter(tip['total_bill'],tip['tip'])
plt.show()
```

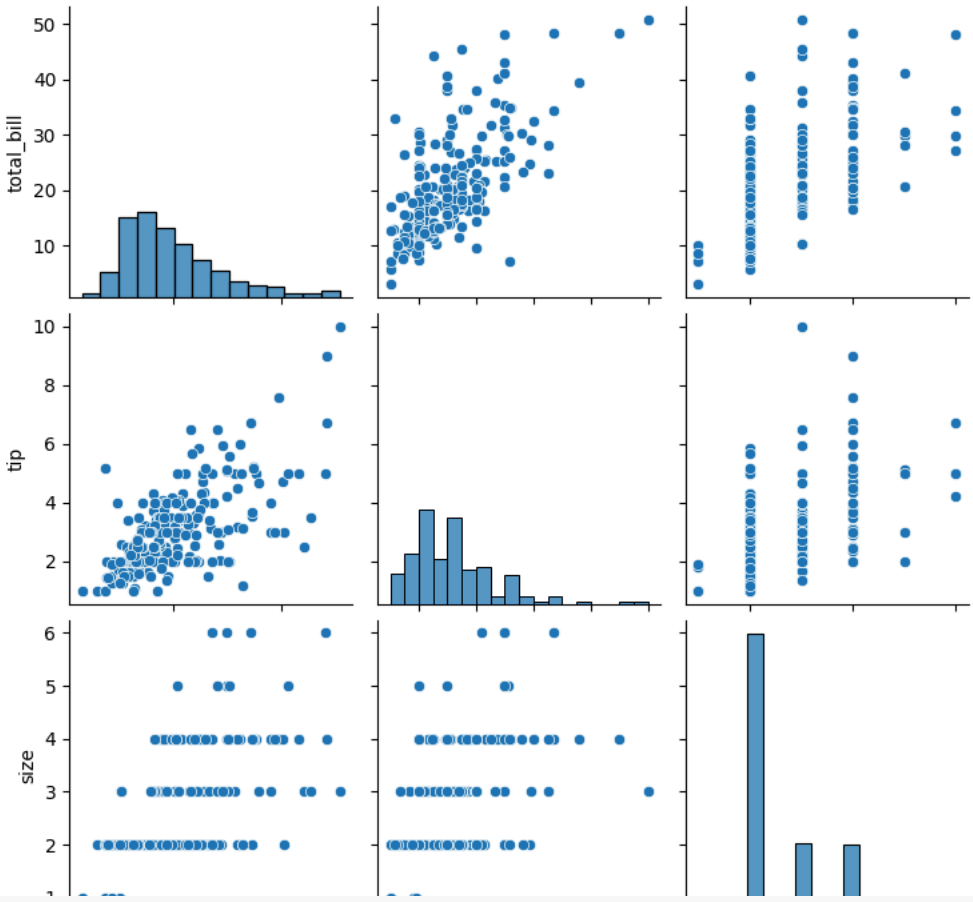


```
X=tip.drop(['tip'],axis=1)
y=tip['tip']
y
```

```
0      1.01
1      1.66
2      3.50
3      3.31
4      3.61
      ...
239    5.92
240    2.00
241    2.00
242    1.75
243    3.00
Name: tip, Length: 244, dtype: float64
```

```
sns.pairplot(tip)
```

```
<seaborn.axisgrid.PairGrid at 0x7eba429df850>
```




```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.5,random_state=6)
```

```
len(X_test)
```

```
122
```

```
from sklearn.svm import SVR
svr = SVR()
sup=svr.fit(X_train, y_train)
pred=sup.predict(X_test)
```

```
sup.score(X_test,y_test)
```

```
0.35536975650629776
```

```
from sklearn import metrics
print('MAE:',metrics.mean_absolute_error(y_test,pred))
print('MSE:',metrics.mean_squared_error(y_test,pred))
print('RMSE:',np.sqrt(metrics.mean_squared_error(y_test,pred)))
```

```
MAE: 0.7636064371510688
MSE: 1.3384313262557594
RMSE: 1.156905928006145
```

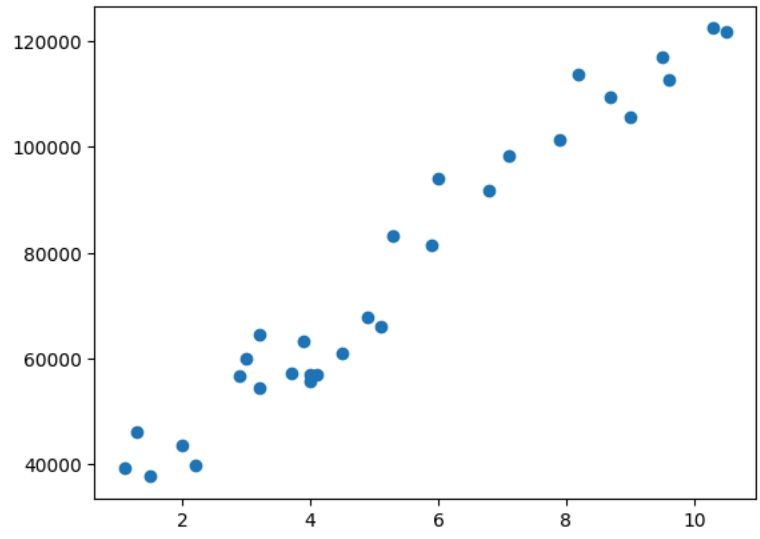
```
sd=pd.read_csv('Salary_Data.csv')
sd
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

```
X=sd[['YearsExperience']].values
#sd.iloc[:,:1].values
y=sd[['Salary']].values
#sd.iloc[:,1].values
X
```

```
array([[ 1.1],
       [ 1.3],
       [ 1.5],
       [ 2. ],
       [ 2.2],
       [ 2.9],
       [ 3. ],
       [ 3.2],
       [ 3.2],
       [ 3.7],
       [ 3.9],
       [ 4. ],
       [ 4. ],
       [ 4.1],
       [ 4.5],
       [ 4.9],
       [ 5.1],
       [ 5.3],
       [ 5.9],
       [ 6. ],
       [ 6.8],
       [ 7.1],
       [ 7.9],
       [ 8.2],
       [ 8.7],
       [ 9. ],
       [ 9.5],
       [ 9.6],
       [10.3],
       [10.5]])
```

```
plt.scatter(X,y)
plt.show()
```




```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.5,random_state=4)
```

```
from sklearn.svm import SVR
sup=SVR(kernel='linear',epsilon=1)
sup.fit(X_train,y_train)
pred=sup.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected
  y = column_or_1d(y, warn=True)
```

```
from sklearn import metrics
print('MAE:',metrics.mean_absolute_error(y_test,pred))
print('MSE:',metrics.mean_squared_error(y_test,pred))
print('RMSE',np.sqrt(metrics.mean_squared_error(y_test,pred)))
```

```
MAE: 23442.904667397826
MSE: 941469280.695135
RMSE 30683.371403663175
```

```
pred
```

```
array([61093.94999781, 61199.65999781, 61308.77999781, 61124.63999781,
       61189.42999781, 61264.44999781, 61138.27999781, 61315.59999781,
       61162.14999781, 60995.05999781, 61025.74999781, 61131.45999781,
       61090.53999781, 61059.84999781, 61093.94999781])
```

```
sup.score(X_test,y_test)
```

```
-0.4212631912823732
```

```
#ML Class 9
#Naive Bayes Algorithm
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline

soc=pd.read_csv('Social_Network_Ads.csv')
soc
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
gen=pd.get_dummies(soc.Gender,drop_first=True)
soc1=pd.concat([soc['Age'],soc['EstimatedSalary'],gen,soc['Purchased']],axis=1)
soc1
```

	Age	EstimatedSalary	Male	Purchased
0	19	19000	1	0
1	35	20000	1	0
2	26	43000	0	0
3	27	57000	0	0
4	19	76000	1	0
...	...	...	...	...
395	46	41000	0	1
396	51	23000	1	1
397	50	20000	0	1
398	36	33000	1	0
399	49	36000	0	1

400 rows × 4 columns

```
X=soc1.iloc[:,:-1].values
y=soc1.iloc[:,3].values
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
1, 1, 0, 1]])
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=2)
```

```
X_train
```

```
array([[   38,  71000,     1],
       [   37,  52000,     1],
       [   31,  18000,     1],
       [   38,  80000,     0],
       [   41,  72000,     0],
       [   33,  31000,     1],
       [   30,  17000,     1],
       [   35,  47000,     0],
       [   26,  86000,     1],
       [   37, 144000,     1],
       [   38,  71000,     1],
       [   26,  16000,     1],
       [   49,  88000,     1],
       [   35,  27000,     1],
       [   52,  38000,     0],
       [   23,  28000,     0],
       [   28,  37000,     0],
       [   58,  47000,     0],
       [   47,  50000,     0],
       [   26,  80000,     0],
       [   38,  61000,     1],
       [   46,  22000,     0],
       [   35,  57000,     0],
       [   47, 113000,     0],
       [   41,  63000,     0],
       [   35,  50000,     1],
       [   35,  22000,     1],
       [   53,  34000,     0],
       [   37,  74000,     1],
       [   29,  47000,     0],
       [   37,  70000,     1],
       [   35,  73000,     1],
       [   40,  59000,     1],
       [   35,  79000,     1],
       [   39,  71000,     1],
       [   58, 144000,     1],
       [   27,  20000,     1],
       [   41,  80000,     0],
       [   59,  29000,     0],
       [   26,  52000,     0],
       [   40,  75000,     1],
       [   35,  38000,     1],
       [   31,  71000,     0],
       [   40,  72000,     0],
       [   42,  65000,     1],
       [   26,  32000,     1],
       [   37, 146000,     0],
       [   49,  65000,     1],
       [   47, 105000,     1],
       [   37,  79000,     1],
       [   32,  18000,     1],
       [   26,  35000,     0],
       [   30, 116000,     0],
       [   49,  89000,     1],
       [   54,  26000,     0],
       [   45,  22000,     0],
       [   31,  34000,     0],
       [   50,  88000,     0],
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

```
X_test
```

```
array([[-0.84234932,  0.41749923, -0.96076892],
       [-1.61930488, -1.5673008 , -0.96076892],
       [-0.06539376, -0.4712172 ,  1.040833  ],
       [-0.84234932,  0.18050818,  1.040833  ],
       [ 1.5856368 ,  0.03238878,  1.040833  ],
       [ 0.90580069, -1.44880527,  1.040833  ],
       [-1.42506599,  0.38787535, -0.96076892],
       [-0.93946876, -1.0933187 ,  1.040833  ],
       [ 0.22596457, -0.26385003, -0.96076892],
       [-0.1625132 ,  1.69132611, -0.96076892],
       [-1.32794654, -1.35993363,  1.040833  ],
       [-1.52218543, -1.24143811, -0.96076892],
       [-0.55099098,  1.42471118, -0.96076892],
       [ 0.7115618 ,  0.29900371,  1.040833  ],
       [-0.25963265, -0.29347391,  1.040833  ],
       [-0.45387154, -0.76745601,  1.040833  ],
       [-1.81354377,  0.03238878,  1.040833  ],
       [ 2.16835347, -0.79707989,  1.040833  ],
       [ 1.48851736,  0.09163654,  1.040833  ],
       [-1.13370765, -0.76745601, -0.96076892],
       [-0.06539376,  0.26937982, -0.96076892],
       [ 0.7115618 , -0.70820825, -0.96076892],
       [-0.74522987, -0.20460227, -0.96076892],
       [-0.1625132 , -1.06369482,  1.040833  ],
       [-1.2308271 ,  0.32862759,  1.040833  ],
       [ 0.7115618 , -1.0933187 ,  1.040833  ],
       [ 0.7115618 , -1.38955751,  1.040833  ],
       [ 0.22596457, -0.35272168,  1.040833  ],
       [ 0.61444235,  2.07643657, -0.96076892],
       [-0.55099098,  0.92110521,  1.040833  ],
       [-1.32794654, -0.41196944, -0.96076892],
       [ 0.32308402, -0.50084108,  1.040833  ],
       [ 0.80868124, -1.35993363,  1.040833  ],
       [ 0.03172569, -0.29347391,  1.040833  ],
       [-0.55099098,  2.40229926, -0.96076892],
       [-0.25963265,  0.65449028,  1.040833  ],
       [ 1.00292013,  0.1508843 ,  1.040833  ],
       [-0.25963265, -1.35993363, -0.96076892],
       [-0.84234932, -0.76745601,  1.040833  ],
       [ 0.90580069,  1.12847237, -0.96076892],
       [-0.25963265, -0.88595153,  1.040833  ],
       [ 0.42020346, -0.11573063,  1.040833  ],
       [-1.03658821,  0.56561863,  1.040833  ],
       [ 0.42020346,  0.32862759,  1.040833  ],
       [-1.91066321, -0.02685899, -0.96076892],
       [ 0.90580069, -0.53046496, -0.96076892],
       [-1.32794654,  0.44712311,  1.040833  ],
       [ 0.90580069, -1.03407094,  1.040833  ],
       [ 1.00292013,  1.48395895, -0.96076892],
       [-1.13370765, -1.59692468, -0.96076892],
       [-1.03658821,  0.80260968, -0.96076892],
       [ 0.42020346,  2.37267538,  1.040833  ],
       [-0.35675209,  0.09163654, -0.96076892],
       [ 0.12884513,  0.29900371, -0.96076892],
       [-1.13370765,  1.45433506, -0.96076892],
       [-1.52218543, -0.17497839, -0.96076892],
       [ 0.32308402, -1.15256646, -0.96076892],
       [ 0.42020346,  1.15809625, -0.96076892],
```

```
len(X_test)
```

```
100
```

```
from sklearn.naive_bayes import GaussianNB
cf=GaussianNB()
cf.fit(X_train,y_train)
pred=cf.predict(X_test)
```

```
from sklearn import metrics
print(metrics.accuracy_score(y_test,pred))
```

```
0.86
```

```
y_test
```

```
array([0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0])
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,pred)
cm
```

```
array([[57,  5],
       [ 9, 29]])
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.92      0.89        62
           1       0.85      0.76      0.81        38

    accuracy                           0.86       100
   macro avg       0.86      0.84      0.85       100
weighted avg       0.86      0.86      0.86       100
```

```
cf.predict_proba(X_test)
```

```
array([[0.9372846 , 0.0627154 ],
       [0.98552985, 0.01447015],
       [0.89713695, 0.10286305],
       [0.96366878, 0.03633122],
       [0.15484576, 0.84515424],
       [0.39774544, 0.60225456],
       [0.97909431, 0.02090569],
       [0.97650321, 0.02349679],
       [0.77411121, 0.22588879],
       [0.15017162, 0.84982838],
       [0.98540917, 0.01459083],
       [0.98824089, 0.01175911],
       [0.47228175, 0.52771825],
       [0.48814957, 0.51185043],
       [0.92445587, 0.07554413],
       [0.94976377, 0.05023623],
       [0.9946957 , 0.0053043 ],
       [0.05739065, 0.94260935],
       [0.17766136, 0.82233864],
       [0.9819141 , 0.0180859 ],
       [0.79649932, 0.20350068],
       [0.56736486, 0.43263514],
       [0.95926616, 0.04073384],
       [0.9015874 , 0.0984126 ],
       [0.97877096, 0.02122904],
       [0.58919161, 0.41080839],
       [0.519942  , 0.480058  ],
       [0.82216098, 0.17783902],
       [0.00848171, 0.99151829],
       [0.81161804, 0.18838196],
       [0.98731045, 0.01268955],
       [0.79645885, 0.20354115],
       [0.47574112, 0.52425888],
       [0.87106429, 0.12893571],
       [0.02371237, 0.97628763],
       [0.80561427, 0.19438573],
       [0.37630902, 0.62369098],
       [0.86783514, 0.13216486],
       [0.9758813 , 0.0241187 ],
       [0.08869378, 0.91130622],
       [0.92486973, 0.07513027],
       [0.73027352, 0.26972648],
       [0.9569394 , 0.0430606 ],
       [0.63108039, 0.36891961],
       [0.99449648, 0.00550352],
       [0.46357185, 0.53642815],
       [0.97896809, 0.02103191],
       [0.49456818, 0.50543182],
       [0.02826199, 0.97173801],
       [0.96374773, 0.03625227],
       [0.91631409, 0.08368591],
       [0.00490061, 0.99509939],
       [0.89635779, 0.10364221],
       [0.71624061, 0.28375939],
       [0.71598491, 0.28401509],
       [0.99011061, 0.00988939],
       [0.70725391, 0.29274609],
       [0.20448326, 0.79551674],
```

```
pred
```

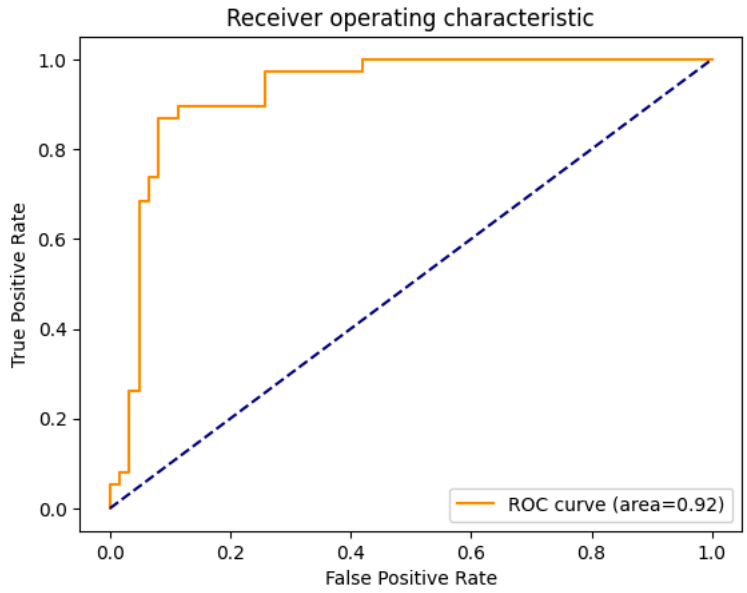
```
array([0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
```

```
0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0,
1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0])
```

```
y_predictprob=cf.predict_proba(X_test)
from sklearn.metrics import auc,roc_curve
fpr,tpr,thresholds=roc_curve(y_test,y_predictprob[::,1])
roc_auc=auc(fpr,tpr)
roc_auc
```

```
0.9248726655348047
```

```
plt.plot(fpr,tpr,color='darkorange',label='ROC curve (area=%0.2f)'%roc_auc)
plt.plot([0,1],[0,1],color='navy',linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc='lower right')
plt.show()
```




```
from sklearn.datasets import load_iris
iris_data=load_iris()
dir(iris_data)
```

```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

```
iris=pd.DataFrame(iris_data.data,columns=iris_data.feature_names)
iris['Species']=pd.Series(iris_data.target)
iris['Names']=iris['Species'].apply(lambda x:iris_data.target_names[x])
iris
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species	Names
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

150 rows × 6 columns

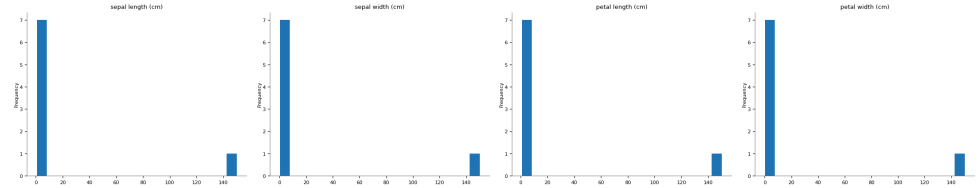
```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   Species            150 non-null    int64  
 5   Names              150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

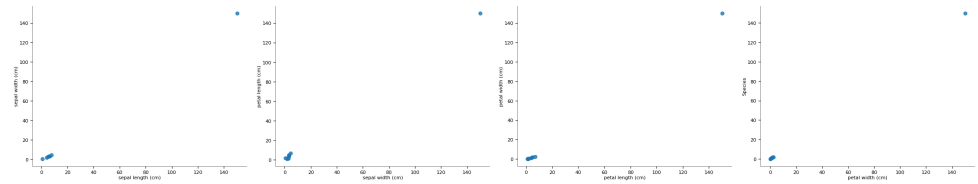
```
iris.describe()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

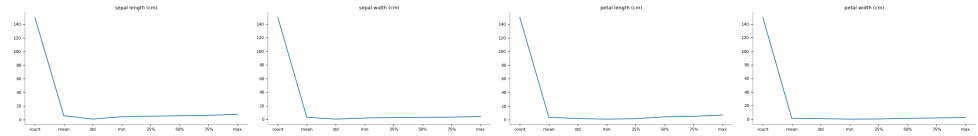
Distributions



2-d distributions



Values



```
X=iris.iloc[:,0:4]
y=iris.iloc[:, -2]
y
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Species, Length: 150, dtype: int64
```

```
y.shape
```

```
(150,)
```

```
X
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
X.shape
```

```
(150, 4)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=10)
```

```
X_train
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
118	7.7	2.6	6.9	2.3
141	6.9	3.1	5.1	2.3
26	5.0	3.4	1.6	0.4
43	5.0	3.5	1.6	0.6
59	5.2	2.7	3.9	1.4
...	...	...	...	...
113	5.7	2.5	5.0	2.0
64	5.6	2.9	3.6	1.3
15	5.7	4.4	1.5	0.4
125	7.2	3.2	6.0	1.8
9	4.9	3.1	1.5	0.1

112 rows × 4 columns

```
X_train.shape
```

```
(112, 4)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

```
X_test
```

```
array([[ 0.55026173, -1.76341327,  0.38316684,  0.14495514],
       [ 0.6699303 , -0.85847725,  0.88672899,  0.92423394],
       [-0.5267554 ,  1.4038628 , -1.23942233, -1.28372268],
       [ 0.31092459, -0.17977524,  0.49506954,  0.27483494],
       [-1.00542968,  0.49892678, -1.29537368, -1.28372268],
       [-1.00542968, -2.44211529, -0.12039532, -0.24468427],
       [ 0.55026173, -1.31094526,  0.66292359,  0.40471474],
       [-0.04808112, -0.85847725,  0.21531279, -0.24468427],
       [-0.88576111,  0.72516079, -1.23942233, -1.28372268],
       [-0.16774969, -0.63224325,  0.43911819,  0.14495514],
       [-0.28741826, -0.17977524,  0.43911819,  0.40471474],
       [-0.04808112, -0.85847725,  0.77482629,  0.92423394],
       [-0.40708683, -1.76341327,  0.15936144,  0.14495514],
       [-1.12509825, -0.17977524, -1.29537368, -1.28372268],
       [-0.88576111,  1.63009681, -1.23942233, -1.15384288],
       [ 1.14860457, -0.17977524,  0.99863169,  1.18399355],
       [ 0.19125602,  0.72516079,  0.43911819,  0.53459454],
       [-1.72344109, -0.17977524, -1.35132503, -1.28372268],
       [-0.88576111,  1.4038628 , -1.23942233, -1.02396308],
       [-1.00542968,  0.27269277, -1.40727638, -1.28372268],
       [ 1.50761028, -0.17977524,  1.22243709,  1.18399355],
       [ 0.6699303 , -0.63224325,  1.05458304,  1.31387335],
       [ 0.43059316, -0.63224325,  0.60697224,  0.79435414],
       [-1.24476682,  0.72516079, -1.01561692, -1.28372268],
       [ 0.07158745, -0.17977524,  0.27126414,  0.40471474],
       [-1.36443539,  0.27269277, -1.35132503, -1.28372268],
       [-0.16774969, -0.17977524,  0.27126414,  0.01507533],
       [-0.40708683, -1.08471126,  0.38316684,  0.01507533],
       [ 1.14860457, -0.63224325,  0.60697224,  0.27483494],
       [ 2.2256217 ,  1.63009681,  1.6700479 ,  1.31387335],
       [ 0.90926743, -0.40600924,  0.49506954,  0.14495514],
       [ 0.43059316, -0.40600924,  0.32721549,  0.14495514],
       [ 1.62727885, -0.17977524,  1.16648574,  0.53459454],
       [-0.04808112, -0.63224325,  0.77482629,  1.57363295],
       [ 0.55026173, -1.31094526,  0.71887494,  0.92423394],
       [-1.48410396,  0.27269277, -1.29537368, -1.28372268],
       [ 1.028936  ,  0.49892678,  1.11053439,  1.18399355],
       [ 1.26827314,  0.27269277,  1.11053439,  1.44375315]])
```

```
from sklearn.naive_bayes import GaussianNB
cf2=GaussianNB()
cf2.fit(X_train,y_train)
y_pred=cf2.predict(X_test)
```

```
cf2.predict_proba(X_test)
```

```
array([[6.66973162e-086, 9.98741015e-001, 1.25898457e-003],
       [5.35514864e-158, 4.91836844e-003, 9.95081632e-001],
```

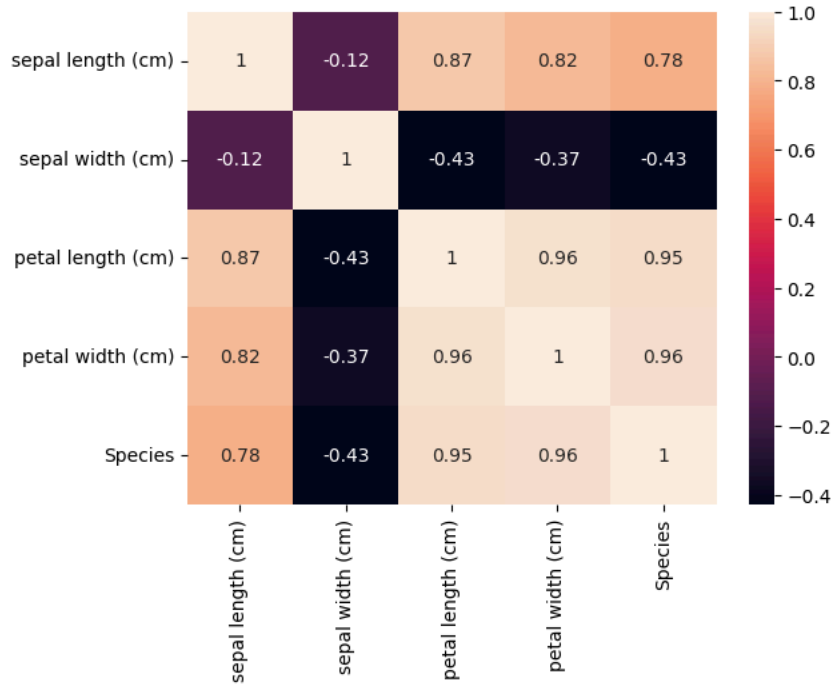
```
       [1.00000000e+000, 2.78902851e-016, 2.18981100e-023],
       [9.72228022e-097, 9.82607184e-001, 1.73928165e-002],
       [1.00000000e+000, 5.20668149e-016, 7.49956880e-024],
       [1.09323417e-039, 9.99999238e-001, 7.61792702e-007],
       [1.43685098e-116, 9.04888753e-001, 9.51112465e-002],
       [8.24263853e-062, 9.99958292e-001, 4.17079666e-005],
       [1.00000000e+000, 9.80776908e-016, 2.43738770e-023],
       [8.81946202e-088, 9.98288208e-001, 1.71179191e-003],
       [8.47250807e-095, 9.85366936e-001, 1.46330636e-002],
       [1.68187119e-144, 3.59530463e-002, 9.64046954e-001],
       [1.96283666e-067, 9.99928695e-001, 7.13052107e-005],
       [1.00000000e+000, 2.47451668e-015, 1.61205837e-023],
       [1.00000000e+000, 4.07049660e-016, 2.49517839e-023],
       [2.66290137e-183, 1.54409185e-005, 9.99984559e-001],
       [4.80306854e-099, 8.23161644e-001, 1.76838356e-001],
       [1.00000000e+000, 5.24271620e-016, 2.75294180e-024],
       [1.00000000e+000, 2.22036793e-014, 7.42864245e-022],
       [1.00000000e+000, 3.20362635e-016, 2.35019743e-024],
       [5.32326302e-210, 7.97890955e-007, 9.99999202e-001],
       [6.27341704e-194, 5.92104356e-006, 9.99994079e-001],
       [8.84727585e-125, 2.25291714e-001, 7.74708286e-001],
       [1.00000000e+000, 5.15549675e-013, 2.93134084e-020],
       [7.10808198e-083, 9.94128918e-001, 5.87108202e-003],
       [1.00000000e+000, 2.35206794e-016, 1.95608144e-024],
       [3.58434167e-071, 9.99665313e-001, 3.34687321e-004],
       [8.65398210e-080, 9.99714135e-001, 2.85865144e-004],
       [2.80938333e-109, 8.87520467e-001, 1.12479533e-001],
       [3.00464565e-275, 1.80925294e-011, 1.00000000e+000],
       [4.49557129e-095, 9.84170694e-001, 1.58293060e-002],
       [1.81993601e-080, 9.98130448e-001, 1.86955208e-003],
       [1.57968099e-176, 1.32794814e-003, 9.98672052e-001],
       [2.23605224e-176, 4.39393779e-006, 9.99995606e-001],
       [3.34077678e-141, 3.63113176e-002, 9.63688682e-001],
       [1.00000000e+000, 4.12471501e-016, 4.21442092e-024],
       [1.73440282e-194, 3.19381365e-006, 9.99996806e-001],
       [7.52781357e-209, 6.53698252e-008, 9.99999935e-001]])
```

```
from sklearn import metrics
print(metrics.accuracy_score(y_test,y_pred)*100)
```

```
100.0
```

```
sns.heatmap(iris.corr(),annot=True)
```

```
<ipython-input-63-fb70caa0f350>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fa
  sns.heatmap(iris.corr(),annot=True)
<Axes: >
```

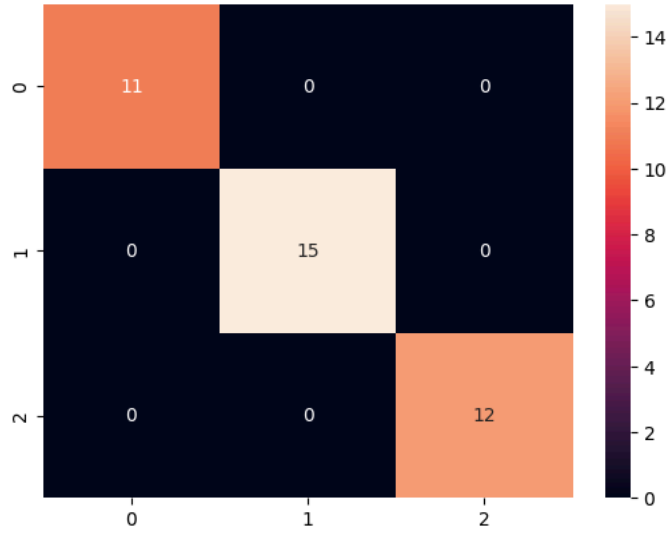



```
from sklearn.metrics import confusion_matrix
cf2=confusion_matrix(y_test,pred)
cf2
```

```
array([[11,  0,  0],
       [ 0, 15,  0],
       [ 0,  0, 12]])
```

```
sns.heatmap(cf2,annot=True)
```

```
<Axes: >
```




```
#ML Class 10
#K-Nearest Neighbors
#Classification
```

```
 import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
soc=pd.read_csv('Social_Network_Ads.csv')
soc
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```
X=soc.iloc[:,[2,3]]
y=soc.iloc[:,4]
X
```

	Age	EstimatedSalary
0	19	19000
1	35	20000
2	26	43000
3	27	57000
4	19	76000
...	...	...
395	46	41000
396	51	23000
397	50	20000
398	36	33000
399	49	36000

400 rows × 2 columns

```
y
```

```
0    0
1    0
2    0
3    0
4    0
    ..
395  1
396  1
397  1
398  0
399  1
Name: Purchased, Length: 400, dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=2)
```

```
X_train
```

	Age	EstimatedSalary
276	38	71000
251	37	52000
29	31	18000
109	38	80000
244	41	72000
...	...	...
299	46	117000
22	48	41000
72	20	23000
15	29	80000
168	29	148000

300 rows × 2 columns

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

```
X_train
```

```
array([[ 0.03172569,  0.06201266],
       [-0.06539376, -0.50084108],
       [-0.64811043, -1.50805304],
       [ 0.03172569,  0.32862759],
       [ 0.32308402,  0.09163654],
       [-0.45387154, -1.12294258],
       [-0.74522987, -1.53767692],
       [-0.25963265, -0.64896049],
       [-1.13370765,  0.50637087],
```

```
       [-0.06539376,  2.22455597],
       [ 0.03172569,  0.06201266],
       [-1.13370765, -1.5673008 ],
       [ 1.10003958,  0.56561863],
       [-0.25963265, -1.24143811],
       [ 1.39139791, -0.91557542],
       [-1.42506599, -1.21181423],
       [-0.93946876, -0.9451993 ],
       [ 1.97411458, -0.64896049],
       [ 0.90580069, -0.56008884],
       [-1.13370765,  0.32862759],
       [ 0.03172569, -0.23422615],
       [ 0.80868124, -1.38955751],
       [-0.25963265, -0.35272168],
       [ 0.90580069,  1.30621566],
       [ 0.32308402, -0.17497839],
       [-0.25963265, -0.56008884],
       [-0.25963265, -1.38955751],
       [ 1.48851736, -1.03407094],
       [-0.06539376,  0.1508843 ],
       [-0.84234932, -0.64896049],
       [-0.06539376,  0.03238878],
       [-0.25963265,  0.12126042],
       [ 0.22596457, -0.29347391],
       [-0.25963265,  0.29900371],
       [ 0.12884513,  0.06201266],
       [ 1.97411458,  2.22455597],
       [-1.03658821, -1.44880527],
       [ 0.32308402,  0.32862759],
       [ 2.07123403, -1.18219034],
       [-1.13370765, -0.50084108],
       [ 0.22596457,  0.18050818],
       [-0.25963265, -0.91557542],
       [-0.64811043,  0.06201266],
       [ 0.22596457,  0.09163654],
       [ 0.42020346, -0.11573063],
       [-1.13370765, -1.0933187 ],
       [-0.06539376,  2.28380373],
       [ 1.10003958, -0.11573063],
       [ 0.90580069,  1.06922461],
       [-0.06539376,  0.29900371],
       [-0.55099098, -1.50805304],
       [-1.13370765, -1.00444706],
       [-0.74522987,  1.3950873 ],
       [ 1.10003958,  0.59524252],
       [ 1.5856368 , -1.27106199],
       [ 0.7115618 , -1.38955751],
       [-0.64811043, -1.03407094],
       [ 1.19715902,  0.56561863],
```

```
len(X_test)
```

```
100
```

```
from sklearn.neighbors import KNeighborsClassifier
cf=KNeighborsClassifier(n_neighbors=5)
cf.fit(X_train,y_train)
pred=cf.predict(X_test)
```

```
pred
```

```
array([0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0])
```

```
from sklearn import metrics
print(metrics.accuracy_score(y_test,pred)*100)
```

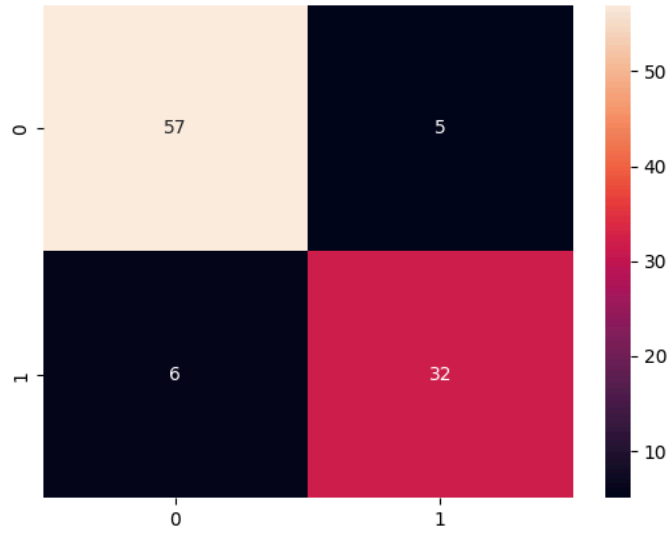
```
89.0
```

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,pred)
cm
```

```
array([[57,  5],
       [ 6, 32]])
```

```
sns.heatmap(cm,annot=True)
```

<Axes: >

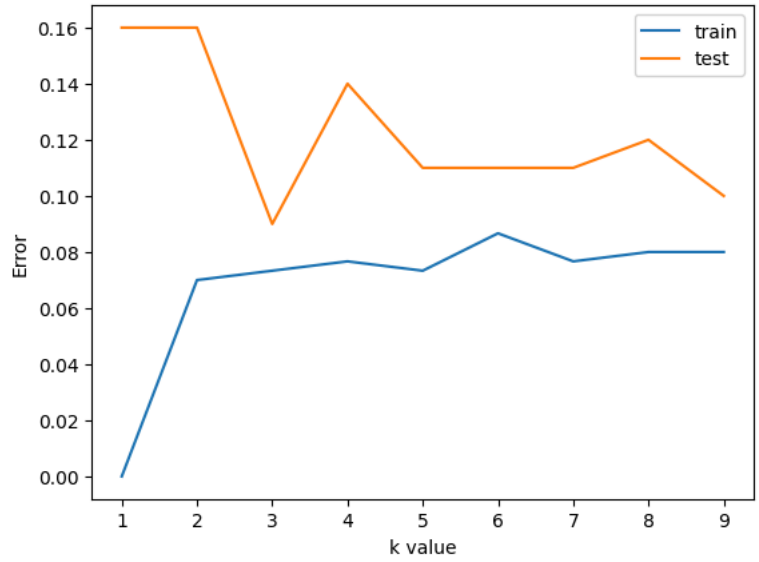



```
from sklearn.metrics import classification_report
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.90      0.92      0.91        62
           1       0.86      0.84      0.85        38

    accuracy                           0.89       100
   macro avg       0.88      0.88      0.88       100
weighted avg       0.89      0.89      0.89       100
```

```
error1=[]
error2=[]

for k in range(1,10):
  knn=KNeighborsClassifier(n_neighbors=k)
  knn.fit(X_train,y_train)
  y_pred1=knn.predict(X_train)
  error1.append(np.mean(y_train!=y_pred1))
  y_pred2=knn.predict(X_test)
  error2.append(np.mean(y_test!=y_pred2))

plt.plot(range(1,10),error1,label='train')
plt.plot(range(1,10),error2,label='test')
plt.xlabel('k value')
plt.ylabel('Error')
plt.legend()
```

<matplotlib.legend.Legend at 0x7ff7069e6110>




```
#Regression
```

```
ht=pd.read_csv('heightdataset.csv')
ht
```

	Index	Height(Inches)	Weight(Pounds)
0	1	65.78331	112.9925
1	2	71.51521	136.4873
2	3	69.39874	153.0269
3	4	68.21660	142.3354
4	5	67.78781	144.2971
...	...	...	...
24995	24996	69.50215	118.0312
24996	24997	64.54826	120.1932
24997	24998	64.69855	118.2655
24998	24999	67.52918	132.2682
24999	25000	68.87761	124.8742

25000 rows × 3 columns

```
X=ht.iloc[0:,[0,2]]
y=ht.iloc[:,1]
X
```

	Index	Weight(Pounds)
0	1	112.9925
1	2	136.4873
2	3	153.0269
3	4	142.3354
4	5	144.2971
...	...	...
24995	24996	118.0312
24996	24997	120.1932
24997	24998	118.2655
24998	24999	132.2682
24999	25000	124.8742

25000 rows × 2 columns

```
y
```

```
0       65.78331
1       71.51521
2       69.39874
3       68.21660
4       67.78781
          ...
24995   69.50215
24996   64.54826
24997   64.69855
24998   67.52918
24999   68.87761
Name: Height(Inches), Length: 25000, dtype: float64
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=0)
```

```
X_train
```

	Index	Weight(Pounds)
10263	10264	121.8602
18409	18410	132.7450
13047	13048	131.8885
21371	21372	120.7458
16392	16393	146.2667
...	...	...
13123	13124	129.3329
19648	19649	118.0422
9845	9846	108.3559
10799	10800	133.7971
2732	2733	143.6484

20000 rows × 2 columns

```
X_train.shape
```

```
(20000, 2)
```

```
X_test.shape
```

```
(5000, 2)
```

```
y_train.shape
```

```
(20000,)
```

```
y_test.shape
```

```
(5000,)
```

```
from sklearn.neighbors import KNeighborsRegressor
rf=KNeighborsRegressor(n_neighbors=5)
rf.fit(X_train,y_train)
pred=rf.predict(X_test)
```

```
pred
```

```
array([68.75087 , 68.46584 , 67.866094, ..., 66.838078, 68.50487 ,
       68.122892])
```

```
y_test
```

```
14149    66.75233
8946     70.23626
22378    67.19022
12162    67.09047
4879     63.65356
         ...
4022     68.49369
17601    67.59295
4718     67.51801
9376     69.06944
8019     66.65813
Name: Height(Inches), Length: 5000, dtype: float64
```

```
rf.score(X_test,y_test)*100
```

```
6.8490140358847595
```

```
print(X_train.dtypes)
```

```
Index            float64
Weight(Pounds)   float64
dtype: object
```

```
X_train = X_train.astype(float)
X_train = pd.to_numeric(X_train, errors='coerce')
X_val=np.arange(min(X_train),max(X_train),0.01)
X_val=X_val.reshape((len(X_val),1))
plt.scatter(X_train,y_train,color='blue')
plt.plot(X_val,rf.predict(X_val),color='purple')
plt.title('Height prediction using KNN')
plt.xlabel('Weight')
plt.ylabel('Height')
plt.figure()
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-55-b69d27ac42eb> in <cell line: 2>()
      1 X_train = X_train.astype(float)
----> 2 X_train = pd.to_numeric(X_train, errors='coerce')
      3 X_val=np.arange(min(X_train),max(X_train),0.01)
      4 X_val=X_val.reshape((len(X_val),1))
      5 plt.scatter(X_train,y_train,color='blue')

/usr/local/lib/python3.10/dist-packages/pandas/core/tools/numeric.py in to_numeric(arg, errors, downcast)
    163         values = np.array([arg], dtype="O")
    164     elif getattr(arg, "ndim", 1) > 1:
--> 165         raise TypeError("arg must be a list, tuple, 1-d array, or Series")
    166     else:
    167         values = arg

TypeError: arg must be a list, tuple, 1-d array, or Series
```

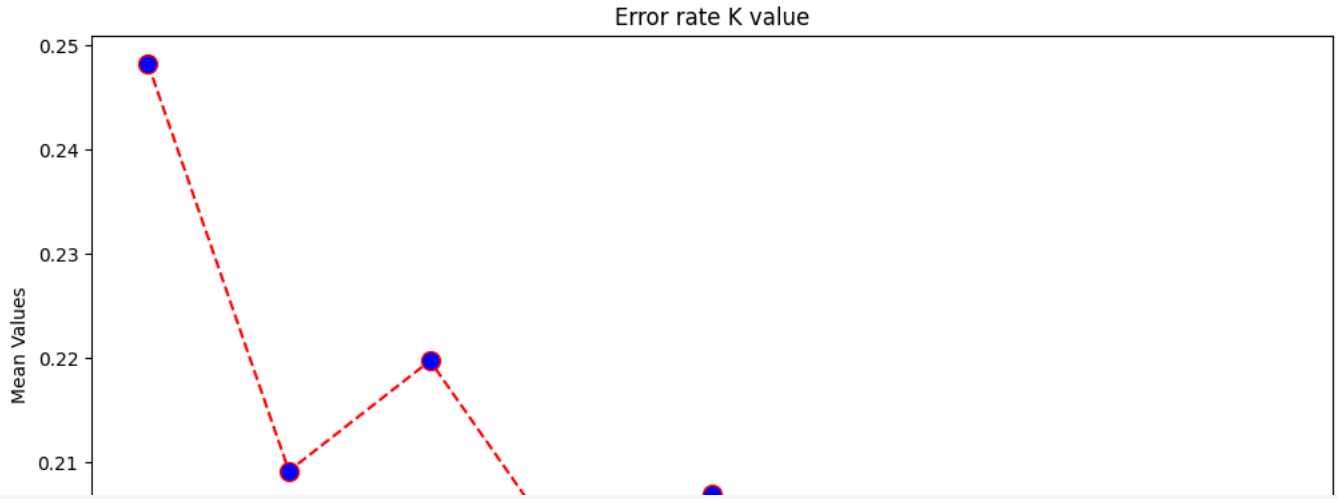
```
ypred=rf.predict(X_test)

from sklearn.metrics import r2_score,mean_squared_error
mae=mean_squared_error(y_test,ypred)
rmse=np.sqrt(mae)
print('Root Mean Squared Error:',rmse)
r2score=r2_score(y_test,ypred)
print('R2Score',r2score*100)
```

```
Root Mean Squared Error: 1.8493973011230738
R2Score 6.8490140358847595
```

```
#KNN Salary Estimator
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline

dataset=pd.read_csv("salary.csv")
print(dataset)

income_set=pd.get_dummies(dataset['income'],drop_first=True)
data=pd.concat([dataset,income_set],axis=1)
data.drop(['income'],inplace=True,axis=1)
data.head()

X= data.iloc[:,:-1].values
print(X)
y=data.iloc[:,-1].values
print(y)

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=2)

from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
X_train = sc.fit_transform(X_train)
X_test=sc.transform(X_test)

error=[]
from sklearn.neighbors import KNeighborsClassifier
for i in range(1,10):
    model=KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train,y_train)
    pred_i=model.predict(X_test)
    error.append(np.mean(pred_i!=y_test))

plt.figure(figsize=[12,6])
plt.plot(range(1,10),error,color='red',linestyle='dashed',marker='o',markerfacecolor='blue',markersize=10)
plt.title('Error rate K value')
plt.xlabel('X-Values')
plt.ylabel('Mean Values')
plt.show()

from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier(n_neighbors = 2,metric='minkowski',p=2)
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
```

```
       age  education.num  capital.gain  hours.per.week income
0       90              9             0              40  <=50K
1       82              9             0              18  <=50K
2       66             10             0              40  <=50K
3       54              4             0              40  <=50K
4       41             10             0              40  <=50K
...    ...            ...           ...             ...    ...
32556   22             10             0              40  <=50K
32557   27             12             0              38  <=50K
32558   40              9             0              40   >50K
32559   58              9             0              40  <=50K
32560   22              9             0              20  <=50K

[32561 rows x 5 columns]
[[90  9  0 40]
 [82  9  0 18]
 [66 10  0 40]
 ...
 [40  9  0 40]
 [58  9  0 40]
 [22  9  0 20]]
[0 0 0 ... 1 0 0]
```




```
age=int(input("Enter New Employee's Age:"))
edu=int(input("Enter New Employee's Education:"))
cg=int(input("Enter New Employee's Capital Gain:"))
wh=int(input("Enter New Employee's Hours per Week:"))
newemp=[[age,edu,cg,wh]]
result=model.predict(sc.transform(newemp))
print(result)


if result ==1:
   print("Employee Got Salary above 50K")
else:
   print("Employee Didn't gert salary above 50k")

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm

# printing The Accuracy of the Model
from sklearn import metrics
print("Accuracy of the model is :{}".format(metrics.accuracy_score(y_test,y_pred)*100))
```

```
Enter New Employee's Age:23
Enter New Employee's Education:10
Enter New Employee's Capital Gain:1000000
Enter New Employee's Hours per Week:25
[1]
Employee Got Salary above 50K
Accuracy of the model is :79.08119395651639
```

```
#ML Class 11
#L1 & L2 Regluralizations
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer

cancer=load_breast_cancer()
cancer_df=pd.DataFrame(cancer.data,columns=cancer.feature_names)
cancer_df
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16

569 rows × 30 columns

```
X=cancer_df
y=cancer.target
y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 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, 0, 0, 0, 0, 0, 0, 1])
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25)
X_train
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter
427	10.80	21.98	68.79	359.9	0.08801	0.05743	0.03614	0.014040	0.2016	0.05977	...	12.76	32.04	83.69
561	11.20	29.37	70.67	386.0	0.07449	0.03558	0.00000	0.000000	0.1060	0.05502	...	11.92	38.30	75.19
195	12.91	16.33	82.53	516.4	0.07941	0.05366	0.03873	0.023770	0.1829	0.05667	...	13.88	22.00	90.81
313	11.54	10.72	73.73	409.1	0.08597	0.05969	0.01367	0.008907	0.1833	0.06100	...	12.34	12.87	81.23
162	19.59	18.15	130.70	1214.0	0.11200	0.16660	0.25080	0.128600	0.2027	0.06082	...	26.73	26.39	174.90
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
474	10.88	15.62	70.41	358.9	0.10070	0.10690	0.05115	0.015710	0.1861	0.06837	...	11.94	19.35	80.78
279	13.85	15.18	88.99	587.4	0.09516	0.07688	0.04479	0.037110	0.2110	0.05853	...	14.98	21.74	98.37
343	19.68	21.68	129.90	1194.0	0.09797	0.13390	0.18630	0.110300	0.2082	0.05715	...	22.75	34.66	157.60
130	12.19	13.29	79.08	455.8	0.10660	0.09509	0.02855	0.028820	0.1880	0.06471	...	13.34	17.81	91.38
289	11.37	18.89	72.17	396.0	0.08713	0.05008	0.02399	0.021730	0.2013	0.05955	...	12.36	26.14	79.29

426 rows × 30 columns

```
#Apply muliple Linear Regression Model
lreg=LinearRegression()
lreg.fit(X_train,y_train)
#Generate Prediction on the test set
lreg_y_pred=lreg.predict(X_test)
```

```
lreg.score(X_train,y_train)
```

```
0.7707490506980715
```

```
lreg.score(X_test,y_test)
```

```
0.7665050501559569
```

```
from sklearn import metrics
```

```
print('MSE:',metrics.mean_squared_error(y_test,lreg_y_pred))
```

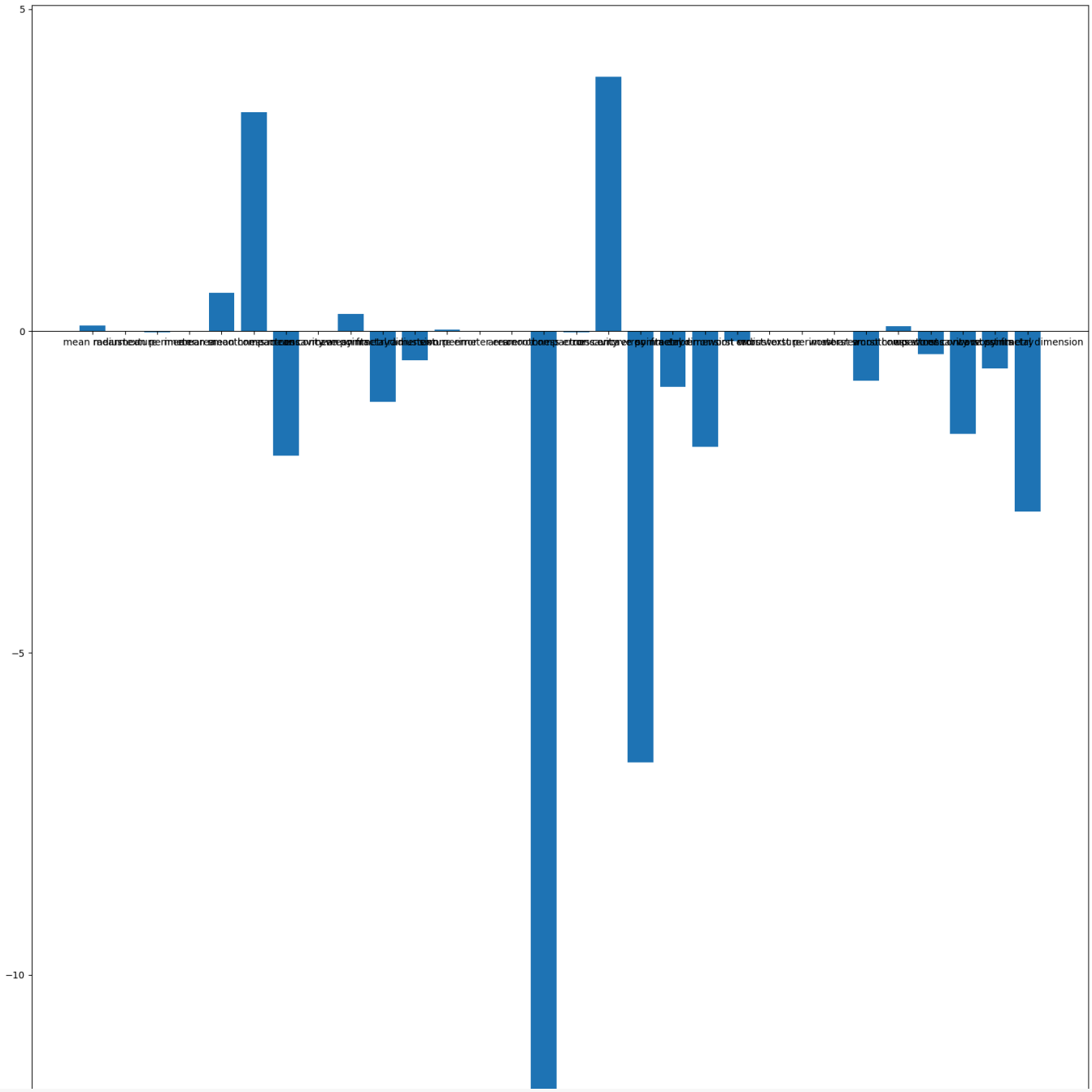
```
MSE: 0.055973017953714066
```

```
#calculating Mean Squared Error(mse)
mean_squared_error=np.mean((lreg_y_pred-y_test)**2)
print('Mean Squared Error on the test set:',mean_squared_error)

#Putting together the coefficient & their corresponding variable names
lreg_coefficient=pd.DataFrame()
lreg_coefficient['Columns']=X_train.columns
lreg_coefficient['Coefficient Estimate']=pd.Series(lreg.coef_)
print(lreg_coefficient)
```

```
Mean Squared Error on the test set: 0.055973017953714066
                 Columns  Coefficient Estimate
0            mean radius              0.082257
1           mean texture             -0.009650
2         mean perimeter             -0.013741
3              mean area              0.000136
4        mean smoothness              0.600076
5       mean compactness              3.401651
6         mean concavity             -1.933753
7     mean concave points             -0.008059
8        mean symmetry               0.262227
9   mean fractal dimension            -1.095223
10         radius error             -0.454819
11        texture error              0.027639
12      perimeter error             -0.010525
13           area error              0.002121
14      smoothness error            -18.282749
15      compactness error             -0.017672
16        concavity error             3.951589
17   concave points error             -6.703832
18       symmetry error              -0.870501
19  fractal dimension error            -1.799137
20         worst radius             -0.153887
21        worst texture             -0.005247
22       worst perimeter              0.005322
```

23	worst area	0.000652
24	worst smoothness	-0.768729
25	worst compactness	0.071886
26	worst concavity	-0.361779
27	worst concave points	-1.598277
28	worst symmetry	-0.580919
29	worst fractal dimension	-2.798785

```
#plotting the coefficient score
fig,ax=plt.subplots(figsize=(20,30))
ax.bar(lreg_coefficient['Columns'],lreg_coefficient['Coefficient Estimate'])
ax.spines['bottom'].set_position('zero')
#plt.style.available
plt.style.use('ggplot')
plt.show()
```




```
lreg_coeff_used=np.sum(lreg.coef_!=0)
print('number of features used:',lreg_coeff_used)
```

```
number of features used: 30
```

```
#import ridge regression from sklearn library
from sklearn.linear_model import Ridge
#Train the model
ridgeR=Ridge(alpha=0.01)
ridgeR.fit(X_train,y_train)
y_pred=ridgeR.predict(X_test)
```

```
ridgeR.score(X_train,y_train)
```

```
0.7656775390177253
```

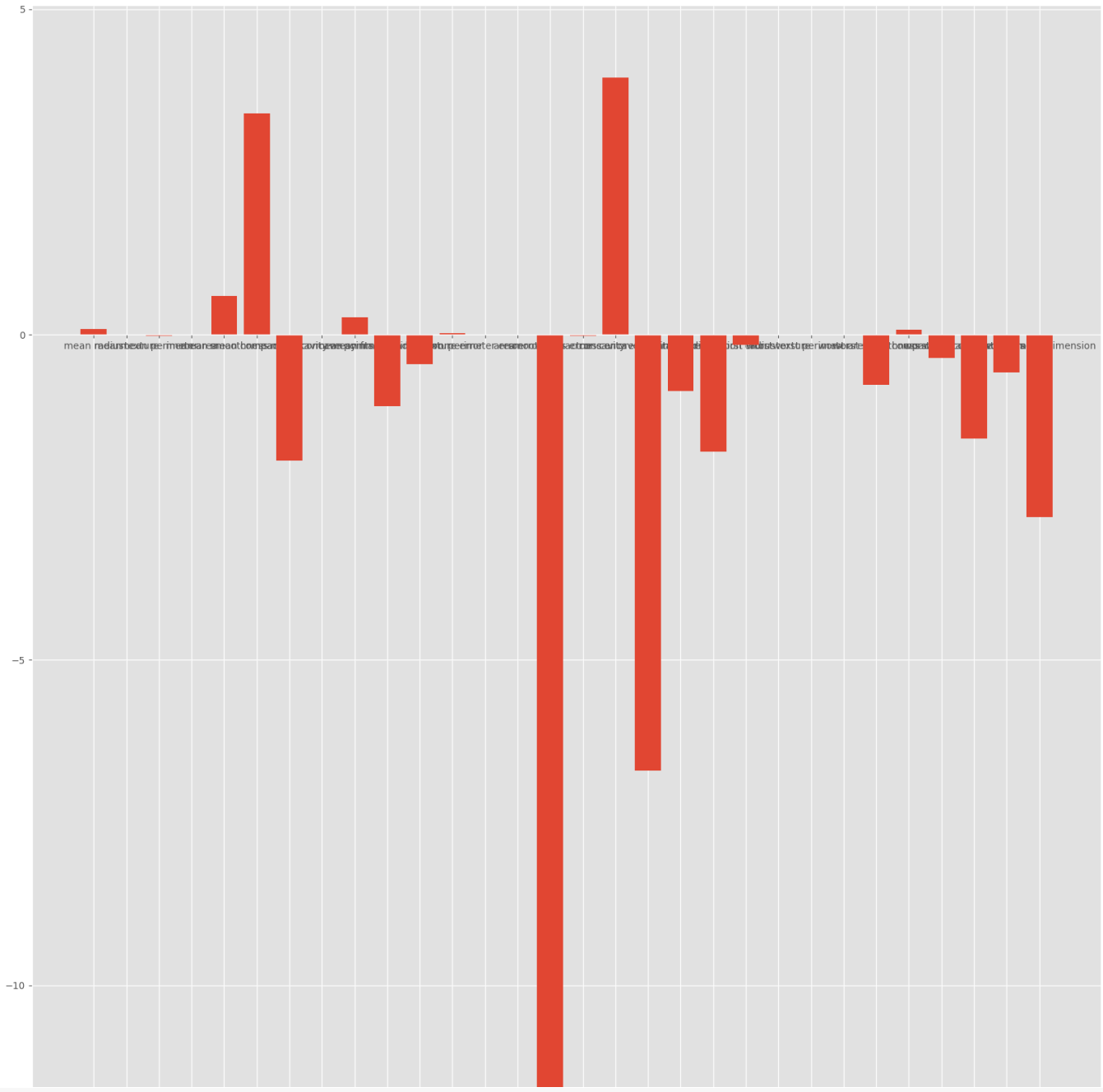
```
ridgeR.score(X_test,y_test)
```

```
0.7645255811530554
```

```
#calculating mean square error
mean_squared_error_ridge=np.mean((y_pred-y_test)**2)
print(mean_squared_error_ridge)

#get ridge coefficient & print them
ridge_coefficient=pd.DataFrame()
ridge_coefficient['Columns']=X_train.columns
ridge_coefficient['Coefficient Estimate']=pd.Series(ridgeR.coef_)
print(ridge_coefficient)
```

```
0.0564475329447759
                  Columns  Coefficient Estimate
0             mean radius              0.056841
1            mean texture             -0.014497
2          mean perimeter             -0.009054
3               mean area              0.000026
4         mean smoothness              0.932227
5        mean compactness              2.517632
6          mean concavity             -1.236265
7     mean concave points             -0.676623
8           mean symmetry              0.208574
9   mean fractal dimension             -0.036823
10           radius error             -0.568098
11          texture error              0.007648
12        perimeter error             -0.010063
13             area error              0.002671
14       smoothness error             -2.561247
15      compactness error             -1.106531
16        concavity error              2.521632
17   concave points error             -1.295744
18         symmetry error             -1.057088
19  fractal dimension error             -0.420804
20            worst radius             -0.138545
21           worst texture             -0.001398
22         worst perimeter              0.005151
23              worst area              0.000594
24        worst smoothness             -2.370538
25       worst compactness              0.160096
26         worst concavity             -0.311178
27    worst concave points             -1.778203
28          worst symmetry             -0.413962
29   worst fractal dimension             -1.927103
```

```
#plotting the coefficient score
fig,ax=plt.subplots(figsize=(20,30))
ax.bar(lreg_coefficient['Columns'],lreg_coefficient['Coefficient Estimate'])
ax.spines['bottom'].set_position('zero')
#plt.style.available
plt.style.use('ggplot')
plt.show()
```




```
ridge_coeff_used=np.sum(ridgeR.coef_!=0)
print('no of features used:',ridge_coeff_used)
```

```
no of features used: 30
```

```
from sklearn.linear_model import Lasso
lasso=Lasso(alpha=0.01)
lasso.fit(X_train,y_train)
y_pred1=lasso.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning: Objective did not converge. You might want
  model = cd_fast.enet_coordinate_descent(
```

```
lasso.score(X_train,y_train)
```

```
0.6900329499484454
```

```
lasso.score(X_test,y_test)
```

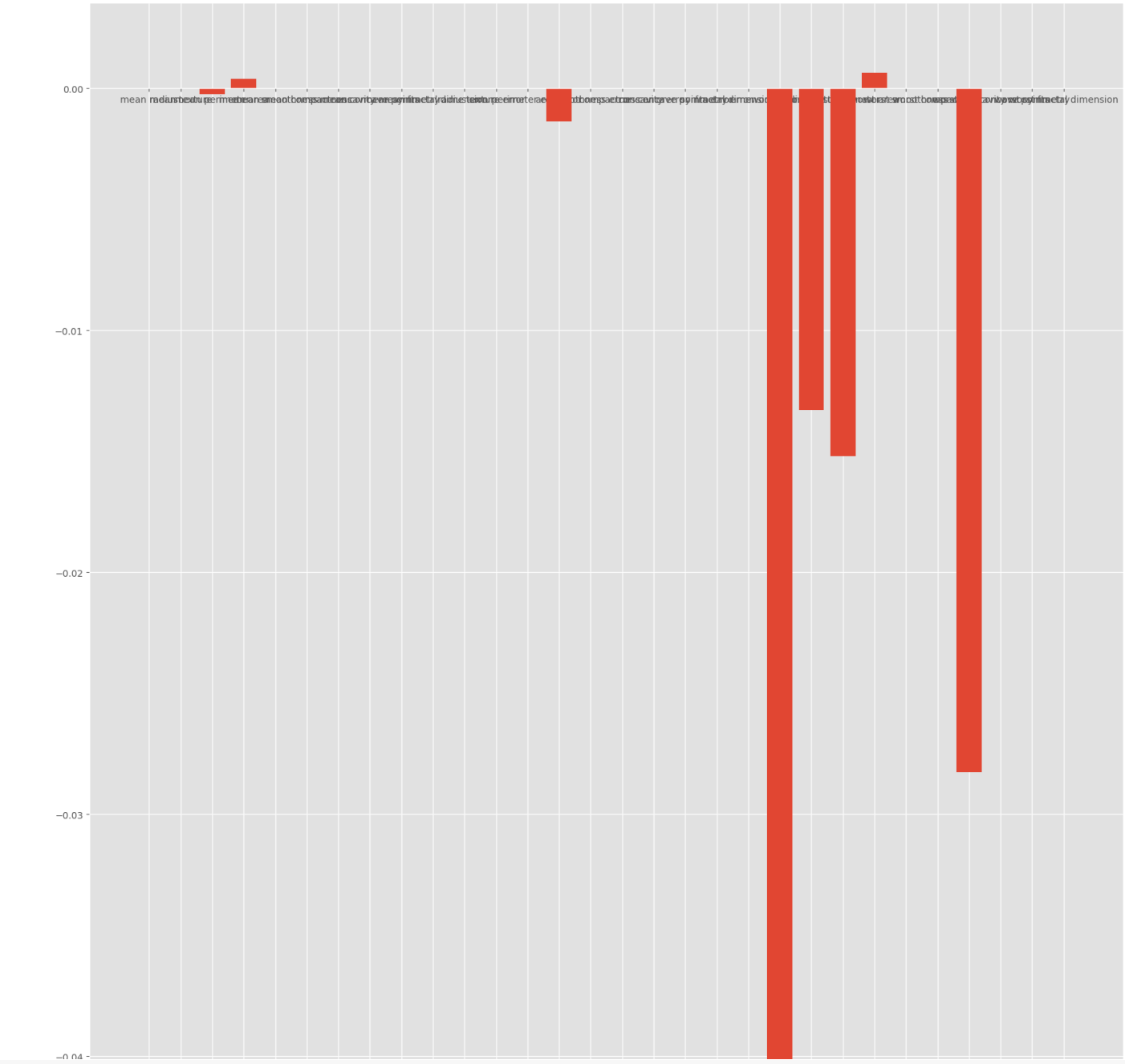
```
0.6530146298105274
```

```
#Calculate Mean Squared Error
mean_squared_error=np.mean((y_pred1-y_test)**2)
print('Mean Squared Error on test set:',mean_squared_error)
lasso_coeff=pd.DataFrame()
lasso_coeff['Columns']=X_train.columns
lasso_coeff['Coefficient Estimate']=pd.Series(lasso.coef_)
lasso_coeff
```

```
Mean Squared Error on test set: 0.08317875126748471
```

	Columns	Coefficient Estimate
0	mean radius	0.000000
1	mean texture	-0.000000
2	mean perimeter	-0.000244
3	mean area	0.000386
4	mean smoothness	-0.000000
5	mean compactness	-0.000000
6	mean concavity	-0.000000
7	mean concave points	-0.000000
8	mean symmetry	-0.000000
9	mean fractal dimension	-0.000000
10	radius error	-0.000000
11	texture error	0.000000
12	perimeter error	-0.000000
13	area error	-0.001374
14	smoothness error	-0.000000
15	compactness error	-0.000000
16	concavity error	-0.000000
17	concave points error	-0.000000
18	symmetry error	-0.000000
19	fractal dimension error	-0.000000
20	worst radius	-0.057206
21	worst texture	-0.013307
22	worst perimeter	-0.015202
23	worst area	0.000638
24	worst smoothness	-0.000000
25	worst compactness	-0.000000
26	worst concavity	-0.028247
27	worst concave points	-0.000000
28	worst symmetry	-0.000000
29	worst fractal dimension	-0.000000

```
#plotting the coefficient score
fig,ax=plt.subplots(figsize=(20,30))

ax.bar(lasso_coeff['Columns'],lasso_coeff['Coefficient Estimate'])
ax.spines['bottom'].set_position('zero')

plt.style.use('ggplot')
plt.show()
```




```
lasso_coeff_used=np.sum(lasso.coef_!=0)
print('no of features used:',lasso_coeff_used)
```

```
no of features used: 8
```

```
#Elastic Net Regression
#import model
from sklearn.linear_model import ElasticNet

#Train the model
e_net=ElasticNet(alpha=0.01,l1_ratio=0.09)
e_net.fit(X_train,y_train)
y_pred2=e_net.predict(X_test)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning: Objective did not converge. You might want
  model = cd_fast.enet_coordinate_descent(
```

```
e_net.score(X_train,y_train)
```

```
0.7257522381333474
```

```
e_net.score(X_test,y_test)
```

```
0.7115746973119066
```

```
from sklearn import metrics
print('MSE',metrics.mean_squared_error(y_test,y_pred2))
```

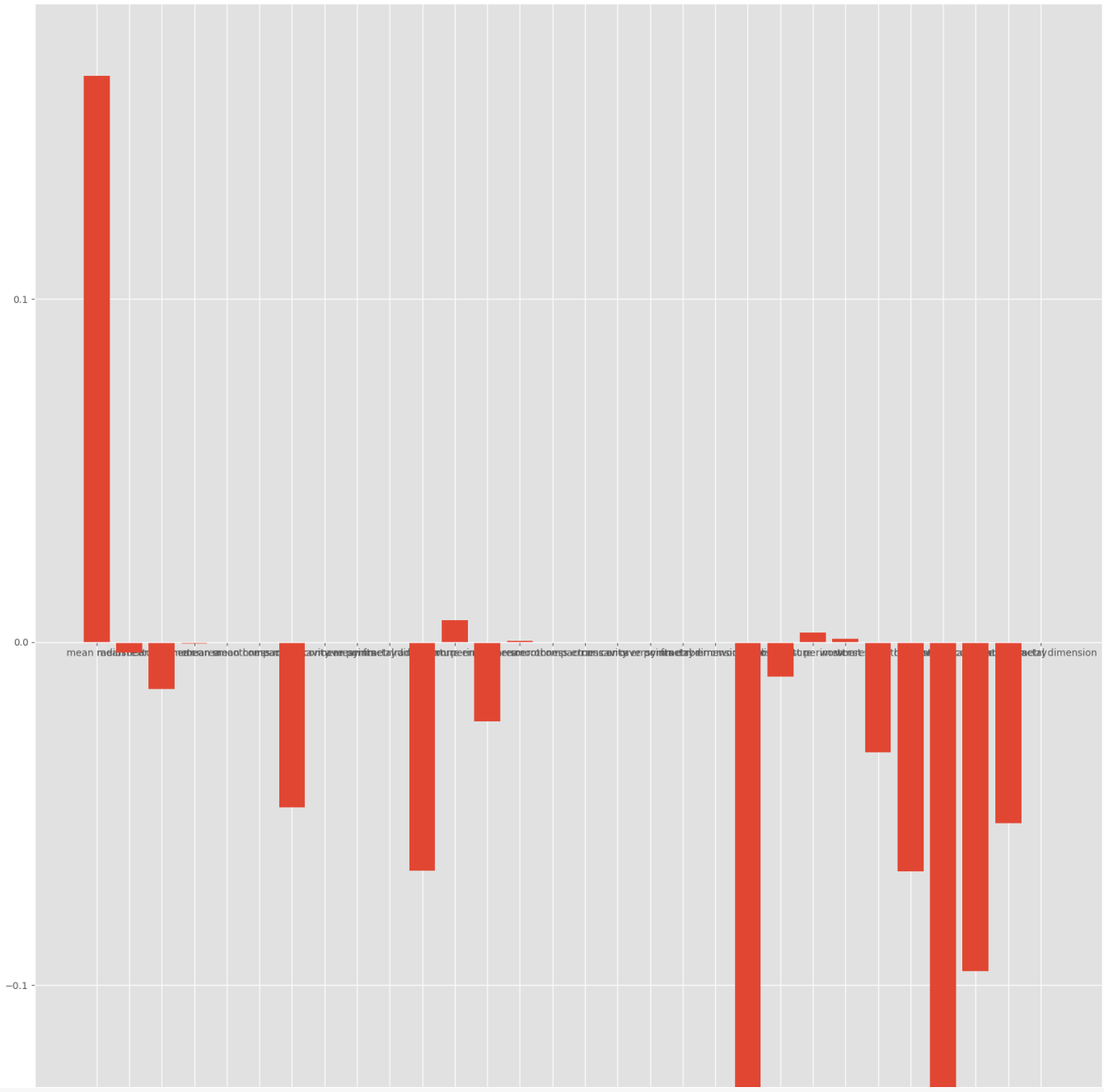
```
MSE 0.06914083005413632
```

```
#calculate the prediction & mean square error
y_pred_elastic=e_net.predict(X_test)
mean_squared_error=np.mean((y_pred_elastic-y_test)**2)
print('Mean Squared Error on test set',mean_squared_error)

e_net_coeff=pd.DataFrame()
e_net_coeff['Columns']=X_train.columns
e_net_coeff['Coefficient Estimate']=pd.Series(e_net.coef_)
e_net_coeff
```

```
Mean Squared Error on test set 0.06914083005413632
```

	Columns	Coefficient Estimate
0	mean radius	0.164992
1	mean texture	-0.003082
2	mean perimeter	-0.013793
3	mean area	-0.000386
4	mean smoothness	-0.000000
5	mean compactness	-0.000000
6	mean concavity	-0.048179
7	mean concave points	-0.000000
8	mean symmetry	-0.000000
9	mean fractal dimension	-0.000000
10	radius error	-0.066651
11	texture error	0.006363
12	perimeter error	-0.023128
13	area error	0.000413
14	smoothness error	-0.000000
15	compactness error	0.000000
16	concavity error	0.000000
17	concave points error	-0.000000
18	symmetry error	-0.000000
19	fractal dimension error	-0.000000
20	worst radius	-0.206359
21	worst texture	-0.010121
22	worst perimeter	0.002640
23	worst area	0.000940
24	worst smoothness	-0.032122
25	worst compactness	-0.066909
26	worst concavity	-0.256225
27	worst concave points	-0.096043
28	worst symmetry	-0.052914
29	worst fractal dimension	-0.000000

```
fig,ax=plt.subplots(figsize=(20,30))

ax.bar(e_net_coeff['Columns'],e_net_coeff['Coefficient Estimate'])
ax.spines['bottom'].set_position('zero')

plt.style.use('ggplot')
plt.show()
```




```
e_net_coef_used=np.sum(e_net.coef_!=0)
print("The No of Feature Used :",e_net_coef_used)
```

```
The No of Feature Used : 18
```

```
#ML Class 12
#Cross Validation Techniques
```

```
X=df.drop(['diagnosis','id','Unnamed: 32'],axis=1)
X
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetr
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	

569 rows × 30 columns

import numpy as np import matplotlib.pyplot as plt import pandas as pd import seaborn as sns %matplotlib inline

df=pd.read_csv('cancer_dataset.csv') df.head()

```
y=df['diagnosis']
print(y)
```

```
0     M
1     M
2     M
3     M
4     M
     ..
564   M
565   M
566   M
567   M
568   B
Name: diagnosis, Length: 569, dtype: object
```

```
y.value_counts()
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

```
#Hold Out Cross Validation
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state=5)
print('Train',X_train,y_train,'Test:',X_test,y_test)
```

```
Train      radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
47       13.170         18.66          85.98      534.6         0.11580
527      12.340         12.27          78.94      468.5         0.09003
435      13.980         19.62          91.12      599.5         0.10600
21        9.504         12.44          60.34      273.9         0.10240
554      12.880         28.92          82.50      514.3         0.08123
..          ...           ...            ...        ...             ...
8        13.000         21.82          87.50      519.8         0.12730
73       13.800         15.79          90.43      584.1         0.10070
400      17.910         21.02         124.40      994.0         0.12300
118      15.780         22.91         105.70      782.6         0.11550
206       9.876         17.27          62.92      295.4         0.10890

     compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
47            0.12310         0.12260              0.07340         0.2128
527           0.06307         0.02958              0.02647         0.1689
435           0.11330         0.11260              0.06463         0.1669
21            0.06492         0.02956              0.02076         0.1815
554           0.05824         0.06195              0.02343         0.1566
..                ...             ...                  ...            ...
8             0.19320         0.18590              0.09353         0.2350
73            0.12800         0.07789              0.05069         0.1662
400           0.25760         0.31890              0.11980         0.2113
118           0.17520         0.21330              0.09479         0.2096
206           0.07232         0.01756              0.01952         0.1934
```

```
     fractal_dimension_mean  ...  radius_worst  texture_worst  \
47                  0.06777  ...         15.67          27.95
527                 0.05808  ...         13.61          19.27
435                 0.06544  ...         17.04          30.80
21                  0.06905  ...         10.23          15.66
554                 0.05708  ...         13.89          35.74
..                      ...  ...           ...            ...
8                   0.07389  ...         15.49          30.73
73                  0.06566  ...         16.57          20.86
400                 0.07115  ...         20.80          27.78
118                 0.07331  ...         20.19          30.50
206                 0.06285  ...         10.42          23.22

     perimeter_worst  area_worst  smoothness_worst  compactness_worst  \
47            102.80       759.4            0.1786             0.4166
527            87.22       564.9            0.1292             0.2074
435           113.90       869.3            0.1613             0.3568
21             65.13       314.9            0.1324             0.1148
554            88.84       595.7            0.1227             0.1620
..               ...         ...               ...                ...
8             106.20       739.3            0.1703             0.5401
73            110.30       812.4            0.1411             0.3542
400           149.60      1304.0            0.1873             0.5917
118           130.30      1272.0            0.1855             0.4925
206            67.08       331.6            0.1415             0.1247

     concavity_worst  concave points_worst  symmetry_worst  \
47           0.50060               0.20880          0.3900
527          0.17910               0.10700          0.3110
435          0.40690               0.18270          0.3179
21           0.08867               0.06227          0.2450
554          0.24390               0.06493          0.2372
```

```
#Leave One Out Cross Validation(LOOCV)
```

```
from sklearn.model_selection import LeaveOneOut
c=[10,20,30,40,50,60,70,80,90,100]
l=LeaveOneOut()
for X_train,X_test in l.split(c): #generate indices
 print('%s %s,'%(X_train,X_test))
```

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

```
#K-Fold Cross Validation
```

```
from sklearn.model_selection import KFold
x=['a','b','c','d','e','f']
kf=KFold(n_splits=3,shuffle=False,random_state=None)
kf
```

```
KFold(n_splits=3, random_state=None, shuffle=False)
```

```
for train,test in kf.split(X):
 print('Train:',train,'Test:',test)
```

```
Train: [190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225
 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243
 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261
 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279
 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297
 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315
 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333
 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351
 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369
 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387
 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405
 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423
 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441
 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459
 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477
 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495
 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513
 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531
 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549
```

```
 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567
 568] Test: [  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  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189]
Train: [  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  48  49  50  51  52  53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179
 180 181 182 183 184 185 186 187 188 189 380 381 382 383 384 385 386 387
 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405
 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423
 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441
 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459
 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477
 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495
 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513
 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531
 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549
 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567
 568] Test: [190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207
 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225
 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243
 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261
 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279
```

#Stratified K-Fold

```
from sklearn.model_selection import StratifiedKFold
X=np.array([[10,20],[30,40],[50,60],[70,80],[90,100],[110,120]])
y=np.array([0,0,1,0,1,1])
print(X)
print(y)
```

```
[[ 10  20]
 [ 30  40]
 [ 50  60]
 [ 70  80]
 [ 90 100]
 [110 120]]
[0 0 1 0 1 1]
```

```
skf=StratifiedKFold(n_splits=3,random_state=None,shuffle=False)
for train_index,test_index in skf.split(X,y):
  print('Train:',train_index,'Test:',test_index)
  X_train,X_test=X[train_index],X[test_index]
  y_train,y_test=y[train_index],y[test_index]
```

```
Train: [1 3 4 5] Test: [0 2]
Train: [0 2 3 5] Test: [1 4]
Train: [0 1 2 4] Test: [3 5]
```

```
print(X_train)
print(X_test)
```

```
[[ 10  20]
 [ 30  40]
 [ 50  60]
 [ 90 100]]
[[ 70  80]
 [110 120]]
```

```
print(X[train_index])
print(y[train_index])
```

```
[[ 10  20]
 [ 30  40]
 [ 50  60]
 [ 90 100]]
[0 0 1 1]
```

```
#Repeated Random Test Train Splits
```

```
#test-trina+kfold
X=['a','b','c','d','e','f']
from sklearn.model_selection import ShuffleSplit
ssplit=ShuffleSplit(n_splits=10,test_size=0.30)
ssplit
```

```
ShuffleSplit(n_splits=10, random_state=None, test_size=0.3, train_size=None)
```

```
for train,test in ssplit.split(X):
  print('Train:',train,'Test:',test)
```

```
Train: [3 2 4 5] Test: [1 0]
Train: [0 1 3 2] Test: [5 4]
Train: [2 0 1 4] Test: [3 5]
Train: [0 3 1 4] Test: [2 5]
Train: [3 1 4 0] Test: [2 5]
Train: [0 1 4 2] Test: [5 3]
Train: [1 3 2 0] Test: [5 4]
Train: [0 1 2 5] Test: [3 4]
Train: [4 5 3 2] Test: [0 1]
Train: [5 1 0 2] Test: [4 3]
```

```
#Cancer dataset
df.drop(['Unnamed: 32'],axis=1,inplace=True)
X=df.iloc[:,2:]
y=df.iloc[:,1]
y.value_counts()
```

```
B    357
M    212
Name: diagnosis, dtype: int64
```

```
X=X.dropna(axis=1)
X
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetr
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	

569 rows × 30 columns

```
#cross_val_score()
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
model=DecisionTreeClassifier()
#EXAMPLE
#Use cross_val_score function
#We are passing the entirety of X & y, not X_train or y_train, it takes care of splitting the
#cv=10 for 10 folds
#scoring='accuracy' for evaluation metric-although there are many
scores=cross_val_score(model,X,y,cv=10)
print(scores)
```

```
[0.92982456 0.84210526 0.92982456 0.87719298 0.89473684 0.89473684
 0.85964912 0.94736842 0.9122807  0.94642857]
```

```
#using hold out method
```

```
from sklearn.metrics import accuracy_score
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=100)
dt=DecisionTreeClassifier()
dtmodel=dt.fit(X_train,y_train)
```

```
dt.score(X_train,y_train)
```

```
1.0
```

```
dt.score(X_test,y_test)
```

```
0.9473684210526315
```

```
hoo_result=dtmodel.score(X_test,y_test)
print('The accuracy score is for Hold one out method:',hoo_result)
```

```
The accuracy score is for Hold one out method: 0.9473684210526315
```

```
#Using k-fold method
```

```
kf=KFold(n_splits=5)
kfold_score=cross_val_score(dt,X,y,cv=kf)
print('The cross validation scores of k-fold method with 5 folds is ',kfold_score)
kfold_score_mean=kfold_score.mean()
```

```
The cross validation scores of k-fold method with 5 folds is  [0.86842105 0.92982456 0.94736842 0.92105263 0.86725664]
```

```
print('The min accuracy from k-fold CV is ',min(kfold_score))
print('The max accuracy from k-fold CV is',max(kfold_score))
print('The mean cross validation scores of k-fold method with 5 folds is',kfold_score_mean)
```

```
The min accuracy from k-fold CV is  0.8672566371681416
The max accuracy from k-fold CV is 0.9473684210526315
The mean cross validation scores of k-fold method with 5 folds is 0.9067846607669615
```

```
kf2=KFold(n_splits=5)
acc_score_kfold=[]
for train_ind,test_ind in kf2.split(X,y):
  #print('Train:',train_ind,'Test:',test_ind)
  X_train,X_test=X.iloc[train_ind,:],X.iloc[test_ind,:]
  y_train,y_test=y[train_ind],y[test_ind]

  dt.fit(X_train,y_train)
  pred_values=dt.predict(X_test)

  acc=accuracy_score(pred_values,y_test)
  acc_score_kfold.append(acc)

avg_acc_score = sum(acc_score_kfold) /5

print('Accuracy of each fold : {}'.format(acc_score_kfold))
print('Avg accuracy : {}'.format(avg_acc_score))
```

```
Accuracy of each fold : [0.868421052631579, 0.9210526315789473, 0.9473684210526315, 0.9385964912280702, 0.8584070796460177]
Avg accuracy : 0.9067691352274492
```

```
#using stratified k-fold method
```

```
skfold=StratifiedKFold(n_splits=10)
skfold_score=cross_val_score(dt,X,y,cv=skfold)
print('The accuracy score of Stratified k-fold method with 10 folds is',skfold_score)
skfold_score_mean=skfold_score.mean()
print('The avg accracy of Stratified k-fold method with 10 folds is ',skfold_score_mean)
```

```
The accuracy score of Stratified k-fold method with 10 folds is [0.94736842 0.85964912 0.9122807  0.87719298 0.94736842 0.9122807
 0.89473684 0.94736842 0.9122807  0.98214286]
The avg accracy of Stratified k-fold method with 10 folds is  0.9192669172932331
```

```
#using leave one out method
```

```
loocv=LeaveOneOut()
loocv_score=cross_val_score(dt,X,y,cv=loocv)
print('The accracy of Leave one out method:',loocv_score)
loocv_score_mean=loocv_score.mean()
print('The avg accracy of Leave one out method with is', loocv_score_mean)
```

```
The accracy of Leave one out method: [1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 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. 0. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 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. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 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. 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. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 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. 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. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 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. 0. 1. 1. 1. 1. 1. 1. 1. 1.]
The avg accracy of Leave one out method with is 0.9261862917398945
```

```
#Digits Data
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.datasets import load_digits
digits=load_digits()
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(digits.data,digits.target,test_size=0.3)
```

```
lr=LogisticRegression(solver='liblinear')
lr.fit(X_train,y_train)
lr.score(X_test,y_test)
```

```
0.9666666666666667
```

```
svm=SVC(gamma='auto')
svm.fit(X_train,y_train)
svm.score(X_test,y_test)
```

```
0.3648148148148148
```

```
rf=RandomForestClassifier(n_estimators=40)
rf.fit(X_train,y_train)
rf.score(X_test,y_test)
```

```
0.9740740740740741
```

```
def get_score(model,X_train,y_train,y_test):
  model.fit(X_train,y_train)
  return model.score(X_test,y_test)
```

```
from sklearn.model_selection import StratifiedKFold
folds=StratifiedKFold(n_splits=3)

scores_logistic=[]
scores_svm=[]
scores_rf=[]

for train_index,test_index in folds.split(digits.data,digits.target):
    X_train,X_test,y_train,y_test = digits.data[train_index],digits.data[test_index],digits.target[train_index],digits.target[test_index]
    scores_logistic.append(get_score((LogisticRegression(solver='liblinear'),X_train,X_test,y_train,y_test)))
    scores_svm.append(get_score((SVC(gamma='auto'),X_train,X_test,y_train,y_test)))
    scores_rf.append(get_score((RandomForestClassifier(n_estimators=40),X_train,X_test,y_train,y_test)))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-90-070237a9bbe6> in <cell line: 8>()
      8 for train_index,test_index in folds.split(digits.data,digits.target):
      9     X_train,X_test,y_train,y_test = digits.data[train_index],digits.data[test_index],digits.target[train_index],digits.target[test_index]
---> 10     scores_logistic.append(get_score((LogisticRegression(solver='liblinear'),X_train,X_test,y_train,y_test)))
     11     scores_svm.append(get_score((SVC(gamma='auto'),X_train,X_test,y_train,y_test)))
     12     scores_rf.append(get_score((RandomForestClassifier(n_estimators=40),X_train,X_test,y_train,y_test)))

TypeError: get_score() missing 3 required positional arguments: 'X_train', 'y_train', and 'y_test'
```

```
scores_logistic
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-76-a04a7c04ce24> in <cell line: 1>()
----> 1 scores_logistic

NameError: name 'scores_logistic' is not defined
```

```
scores_svm
```

```
[]
```

```
scores_rf
```

```
[]
```

```
cross_val_score(LogsiticRegression(solver='liblinear'),digits.data,digits.target,cv=3)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-86-c80d951a44c0> in <cell line: 1>()
----> 1 cross_val_score(LogsiticRegression(solver='liblinear'),digits.data,digits.target,cv=3)

NameError: name 'LogsiticRegression' is not defined
```

```
cross_val_score(SVC(gamma='auto',digits.data,digits.target,cv=3))
```

```
  File "<ipython-input-87-86dd3ea79c08>", line 1
    cross_val_score(SVC(gamma='auto',digits.data,digits.target,cv=3))
                                                 ^
SyntaxError: positional argument follows keyword argument
```

```
# Parameter Tuning using K fold cross Validation
```

```
score1=cross_val_score(RandomForestClassifier(n_estimators=5),digits.data,digits.target,cv=10)
np.average(score1)

score2=cross_val_score(RandomForestClassifier(n_estimators=10),digits.data,digits.target,cv=10)
np.average(score2)

score3=cross_val_score(RandomForestClassifier(n_estimators=15),digits.data,digits.target,cv=10)
np.average(score3)

score4=cross_val_score(RandomForestClassifier(n_estimators=50),digits.data,digits.target,cv=10)
np.average(score4)

score5=cross_val_score(RandomForestClassifier(n_estimators=500),digits.data,digits.target,cv=10)
np.average(score5)

score6=cross_val_score(RandomForestClassifier(n_estimators=500),digits.data,digits.target,cv=100)
np.average(score6)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-53-81966f63a8e7> in <cell line: 1>()
----> 1 score1=cross_val_score(RandomForestClassifier(n_estimators=5),digits.data,digits.target,cv=10)
      2 np.average(score1)
      3
      4 score2=cross_val_score(RandomForestClassifier(n_estimators=10),digits.data,digits.target,cv=10)
      5 np.average(score2)

NameError: name 'cross_val_score' is not defined
```

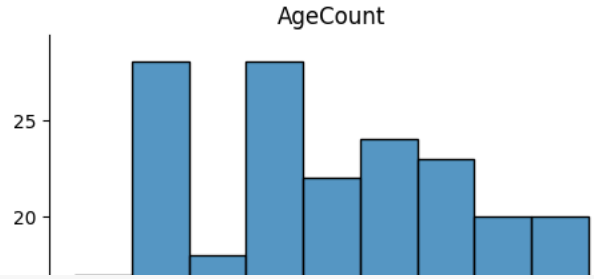
```
#ML Class 13
drug_data=pd.read_csv("drug200.csv")
print(drug_data)

print(drug_data.info())

print(drug_data.isna().sum())

print(drug_data.describe())

plt.figure(figsize=(9,5))
sns.displot(drug_data.Age)
plt.title("AgeCount")
print(plt.show())

print(drug_data.Sex.value_counts())

plt.figure(figsize=(9,5))
sns.countplot(x=drug_data.Sex)
plt.title("SexComparison")
print(plt.show())

print(drug_data.BP.value_counts())

plt.figure(figsize=(9,5))
sns.countplot(x=drug_data.BP)
plt.title("BPLevels")
print(plt.show())

print(drug_data.Cholesterol.value_counts())

plt.figure(figsize=(9,5))
sns.countplot(x=drug_data.Cholesterol)
plt.title("CholesterolLevels")
print(plt.show())

plt.figure(figsize=(9,5))
sns.displot(x=drug_data.Na_to_K,kind='kde')
# plt.title("CholesterolLevels")
print(plt.show())
sns.displot(x=drug_data.Na_to_K)
# plt.title("CholesterolLevels")
print(plt.show())

print(drug_data.Drug.value_counts())

plt.figure(figsize=(9,5))
sns.countplot(x=drug_data.Drug)
plt.title("DifferentDrugs")
print(plt.show())

# age vs Drug
plt.figure(figsize=(9,5))
sns.swarmplot(x="Drug",y="Age",data=drug_data)
plt.legend(drug_data.Drug.value_counts().index)
plt.title("Age vs Drug")
print(plt.show())

print("Minimum Age of Consuming DrugA:",drug_data.Age[drug_data.Drug=='drugA'].min())
print("Minimum Age of Consuming DrugB:",drug_data.Age[drug_data.Drug=='drugB'].min())
print("Minimum Age of Consuming DrugC:",drug_data.Age[drug_data.Drug=='drugC'].min())
print("Minimum Age of Consuming DrugX:",drug_data.Age[drug_data.Drug=='drugX'].min())
print("Minimum Age of Consuming DrugY:",drug_data.Age[drug_data.Drug=='DrugY'].min())

print("Maximum Age of Consuming DrugA:",drug_data.Age[drug_data.Drug=='drugA'].max())
print("Maximum Age of Consuming DrugB:",drug_data.Age[drug_data.Drug=='drugB'].max())
print("Maximum Age of Consuming DrugC:",drug_data.Age[drug_data.Drug=='drugC'].max())
print("Maximum Age of Consuming DrugX:",drug_data.Age[drug_data.Drug=='drugX'].max())
print("Maximum Age of Consuming DrugY:",drug_data.Age[drug_data.Drug=='DrugY'].max())

drug_data_Sex_Drug=drug_data.groupby(['Drug',"Sex"]).size()
print(drug_data_Sex_Drug)

plt.figure(figsize=(9,5))
sns.countplot(x="Drug",hue="Sex",data=drug_data)
plt.title("Sex vs Drug")
print(plt.show())

drug_data_BP_Drug=drug_data.groupby(['Drug',"BP"]).size().reset_index(name='Count')
print(drug_data_BP_Drug)

plt.figure(figsize=(9,5))
```

```
sns.countplot(x="Drug",hue="BP",data=drug_data)
plt.title("BP vs Drug")
print(plt.show())

plt.figure(figsize=(9,5))
sns.boxplot(x="Drug",y="Na_to_K",data=drug_data)
plt.title("Na_to_K vs Drug")
print(plt.show())

print(drug_data.groupby(['Drug'])[['Na_to_K']].min())

print(drug_data.groupby(['Drug'])[['Na_to_K']].max())

print(drug_data.groupby(['Drug'])[['Na_to_K']].mean())

drug_data_CH_Drug=drug_data.groupby(['Drug','Cholesterol']).size().reset_index(name="count")
print(drug_data_CH_Drug)

# Drug C might be given in High Choleaterol
plt.figure(figsize=(9,5))
sns.barplot(x="Drug",y="count",hue="Cholesterol",data=drug_data_CH_Drug)
plt.title("Cholesterol vs Drug")
print(plt.show())

# Drug C for High Cholesteerol
plt.figure(figsize=(9,5))
sns.stripplot(x="Drug",y="Na_to_K",hue="BP",data=drug_data)
plt.legend()
plt.title("Na_to_K and BP vs Drug")
print(plt.show())

# Drug C must be Taken if Low BP

#Feature Engineering
# if Na_to_K is higher than 15 then one must take DrugY

HighNatoK=drug_data.Na_to_K.apply(lambda x:  1 if x>=15 else 0)
High_NatoK_ratio=pd.Series(HighNatoK,name='HighNatoK')
drug_data_1=pd.concat([drug_data,High_NatoK_ratio],axis=1)
print(drug_data)

plt.figure(figsize=(9,5))
sns.violinplot(x="Drug",y="Age",hue="HighNatoK",data=drug_data_1)
plt.title("NatoK higher than 15 and Drug")
print(plt.show())

print(sns.pairplot(drug_data_1))

# Encoding
print(drug_data_1.dtypes)

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import ColumnTransformer
ct=ColumnTransformer([('ohe',OneHotEncoder(drop='first'),['Sex']),
                      ('oe',OrdinalEncoder(),['BP','Cholesterol'])],remainder='passthrough')

drug_data_2=ct.fit_transform(drug_data_1)
print(drug_data_1.head(15))
print(drug_data_2[0:15])

# choosing Imputs and Output
X=drug_data_1.drop(['Drug'],axis=1)
print(X)

y=drug_data_1['Drug']
print(y)

Xnew=ct.fit_transform(X)
print(Xnew[:15])

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(Xnew,y,test_size=0.3,random_state=5)

print(len(X_train))

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
skfold=StratifiedKFold(n_splits=5)

from sklearn.linear_model import LogisticRegression
```

```
lr_model=LogisticRegression(solver='liblinear')
lr_model.fit(X_train,y_train)

lr_trainscore=cross_val_score(lr_model,X_train,y_train,cv=skfold)
lr_pred=lr_model.predict(X_test)
print(lr_trainscore)
print(lr_trainscore.mean())

lr_testscore=lr_model.score(X_test,y_test)
print(lr_testscore)

result_dict_train={}
result_dict_test={}
result_dict_train["LR train score"]=np.mean(lr_trainscore)
result_dict_test["LR Test Score"]=lr_testscore

# Random Forest
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=5)
rfc.fit(X_train,y_train)
rfc_trainscore=cross_val_score(rfc,X_train,y_train,cv=skfold)
print(rfc_trainscore)

print(rfc_trainscore.mean())

rfc_pred=rfc.predict(X_test)
print(rfc_pred)

rfc_testscore=rfc.score(X_test,y_test)
print(rfc_testscore)

result_dict_train["RFC train score"]=np.mean(rfc_trainscore)
result_dict_test["RFC Test Score"]=rfc_testscore

# KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=10,p=1)
knn.fit(X_train,y_train)
knn_trainscore=cross_val_score(knn,X_train,y_train,cv=skfold)
print(knn_trainscore)

knn_pred=knn.predict(X_test)
print(knn_trainscore.mean())
knn_testscore=knn.score(X_train,y_train)
print(knn_testscore)

result_dict_train["KNN train score"]=np.mean(knn_trainscore)
result_dict_test["KNN Test Score"]=knn_testscore

# SVM Classifier
from sklearn.svm import SVC
svc=SVC(kernel='linear',C=0.05,gamma=0.5)
svc.fit(X_train,y_train)
svc_trainscore=cross_val_score(svc,X_train,y_train,cv=skfold)
print(rsvc_trainscore)

svc_pred=svc.predict(X_test)
print(svc_trainscore.mean())
svc_testscore=svc.score(X_train,y_train)
print(svc_testscore)

result_dict_train["svc train score"]=np.mean(svc_trainscore)
result_dict_test["svc Test Score"]=svc_testscore

# Naive Bayes
from sklearn.naive_bayes import GaussianNB
nbc=GaussianNB()
nbc.fit(X_train,y_train)
nbc_trainscore=cross_val_score(nbc,X_train,y_train,cv=skfold)
print(nbc_trainscore)

nbc_pred=nbc.predict(X_test)
print(nbc_trainscore.mean())
nbc_testscore=nbc.score(X_train,y_train)
print(nbc_testscore)

result_dict_train["NBC train score"]=np.mean(nbc_trainscore)
result_dict_test["NBC Test Score"]=nbc_testscore
print(result_dict_train)

test_result=pd.DataFrame.from_dict(result_dict_test,orient="index",columns=["testScores"])
print(test_result)
```

```
train_result=pd.DataFrame.from_dict(result_dict_train,orient="index",columns=["testScores"])
print(train_result)

grid={
    'C':[0.01,0.1,1,10],
    'kernel':["linear",'poly','rbf','sigmoid'],
    'degree':[1,3,5,7],
    'gamma':[0.01,1]
}

svmg=SVC()
svm_cv=GridSearchCV(svmg,grid,cv=5)
svm_cv.fit(X_train,y_train)
print("Best Parameters:",svm_cv.best_params_)
print("train Score:",svm_cv.best_score_)
print("test Score:",svm_cv.score(X_test,y_test))

fig,ax=plt.subplots(1,2,figsize=(20,5))
sns.barplot(x=test_result.index,y=test_result.testScores,ax=ax[0])
sns.barplot(x=train_result.index,y=train_result.testScores,ax=ax[1])
ax[0].set_xticklabels(test_result.index,rotation =45)
ax[1].set_xticklabels(train_result.index,rotation=45)
```

```
     Age Sex      BP Cholesterol  Na_to_K   Drug
0     23   F    HIGH        HIGH   25.355  DrugY
1     47   M     LOW        HIGH   13.093  drugC
2     47   M     LOW        HIGH   10.114  drugC
3     28   F  NORMAL        HIGH    7.798  drugX
4     61   F     LOW        HIGH   18.043  DrugY
..   ...  ..     ...         ...      ...    ...
195   56   F     LOW        HIGH   11.567  drugC
196   16   M     LOW        HIGH   12.006  drugC
197   52   M  NORMAL        HIGH    9.894  drugX
198   23   M  NORMAL      NORMAL   14.020  drugX
199   40   F     LOW      NORMAL   11.349  drugX

[200 rows x 6 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Age          200 non-null    int64
 1   Sex          200 non-null    object
 2   BP           200 non-null    object
 3   Cholesterol  200 non-null    object
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
None
Age            0
Sex            0
BP             0
Cholesterol    0
Na_to_K        0
Drug           0
dtype: int64
              Age     Na_to_K
count  200.000000  200.000000
mean    44.315000   16.084485
std     16.544315    7.223956
min     15.000000    6.269000
25%     31.000000   10.445500
50%     45.000000   13.936500
75%     58.000000   19.380000
max     74.000000   38.247000
<Figure size 900x500 with 0 Axes>
```



```
#ML Class 14
#Unsupervised Learning
#Data Science Regression Project : Predicting Home Prices in Bangalore
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
hdata=pd.read_csv('Bengaluru_House_Data.csv')
hdata
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00
...	...	...	...	...	...	...	...	...	...
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0	231.00
13316	Super built-up Area	Ready To Move	Richards Town	4 BHK	NaN	3600	5.0	NaN	400.00
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	Mahla T	1141	2.0	1.0	60.00
13318	Super built-up Area	18-Jun	Padmanabhanagar	4 BHK	SollyCl	4689	4.0	1.0	488.00
13319	Super built-up Area	Ready To Move	Doddathoguru	1 BHK	NaN	550	1.0	1.0	17.00

13320 rows × 9 columns

```
hdata.columns
```

```
Index(['area_type', 'availability', 'location', 'size', 'society',
       'total_sqft', 'bath', 'balcony', 'price'],
      dtype='object')
```

```
hdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   area_type     13320 non-null  object 
 1   availability  13320 non-null  object 
 2   location      13319 non-null  object 
 3   size          13304 non-null  object 
 4   society       7818 non-null   object 
 5   total_sqft    13320 non-null  object 
 6   bath          13247 non-null  float64
 7   balcony       12711 non-null  float64
 8   price         13320 non-null  float64
dtypes: float64(3), object(6)
memory usage: 936.7+ KB
```

```
hdata['area_type'].unique()
```

```
array(['Super built-up  Area', 'Plot  Area', 'Built-up  Area',
       'Carpet  Area'], dtype=object)
```

```
hdata['area_type'].value_counts()
```

```
Super built-up  Area    8790
Built-up  Area          2418
Plot  Area              2025
Carpet  Area              87
Name: area_type, dtype: int64
```

```
hdata['society'].value_counts()
#drop later
```

```
GrrvaGr    80
PrarePa    76
Sryalan    59
Prtates    59
GMown E    56
          ..
Amionce     1
JaghtDe     1
Jauraht     1
Brity U     1
RSntsAp     1
Name: society, Length: 2688, dtype: int64
```

```
hdata['balcony'].value_counts()
```

```
2.0    5113
1.0    4897
3.0    1672
0.0    1029
Name: balcony, dtype: int64
```

```
#Data Cleaning : Handle NA values
```

```
hdata.isna().sum()
```

```
area_type          0
availability       0
location           1
size              16
society         5502
total_sqft         0
bath              73
balcony          609
price              0
dtype: int64
```

```
hdata.shape
```

```
(13320, 9)
```

```
hdata['balcony'].median()
```

```
2.0
```

```
hdata['balcony']=hdata['balcony'].fillna(hdata.balcony.median())
hdata.isna().sum()
```

```
area_type          0
availability       0
location           1
size              16
society         5502
total_sqft         0
bath              73
balcony            0
price              0
dtype: int64
```

```
None
```

```

```

```
(13320, 9)
```

```
drugB    16
```

```
hdata1=hdata
hdata1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13320 entries, 0 to 13319
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
```

```
 0   area_type   13320 non-null  object
 1   location    13319 non-null  object
 2   size        13304 non-null  object
 3   total_sqft  13320 non-null  object
 4   bath        13247 non-null  float64
 5   balcony     13320 non-null  float64
 6   price       13320 non-null  float64
dtypes: float64(3), object(4)
memory usage: 728.6+ KB
```

```
hdata1=hdata1.dropna()
hdata1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13246 entries, 0 to 13319
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   area_type   13246 non-null  object
 1   location    13246 non-null  object
 2   size        13246 non-null  object
 3   total_sqft  13246 non-null  object
 4   bath        13246 non-null  float64
 5   balcony     13246 non-null  float64
 6   price       13246 non-null  float64
dtypes: float64(3), object(4)
memory usage: 827.9+ KB
```

```
hdata1.head()
```

	area_type	location	size	total_sqft	bath	balcony	price
0	Super built-up Area	Electronic City Phase II	2 BHK	1056	2.0	1.0	39.07
1	Plot Area	Chikka Tirupathi	4 Bedroom	2600	5.0	3.0	120.00
2	Built-up Area	Uttarahalli	3 BHK	1440	2.0	3.0	62.00
3	Super built-up Area	Lingadheeranahalli	3 BHK	1521	3.0	1.0	95.00
4	Super built-up Area	Kothanur	2 BHK	1200	2.0	1.0	51.00

Minimum Age of Consuming DrugA: 19

```
hdata1['size'].value_counts()
```

```
2 BHK         5198
3 BHK         4286
4 Bedroom      818
4 BHK          577
3 Bedroom      546
1 BHK          531
2 Bedroom      329
5 Bedroom      296
6 Bedroom      191
1 Bedroom      105
8 Bedroom       84
7 Bedroom       83
5 BHK           57
9 Bedroom       46
6 BHK           30
7 BHK           17
1 RK            13
10 Bedroom      12
9 BHK            8
8 BHK            5
11 BHK           2
11 Bedroom       2
10 BHK           2
14 BHK           1
13 BHK           1
12 Bedroom       1
27 BHK           1
43 Bedroom       1
16 BHK           1
19 BHK           1
18 Bedroom       1
Name: size, dtype: int64
```

```
hdata1['bhk']=hdata1['size'].apply(lambda x:int(x.split(' ')[0]))
print(hdata1.bhk.unique())
```

```
[ 2  4  3  6  1  8  7  5 11  9 27 10 19 16 43 14 12 13 18]
<ipython-input-25-cfff22022ef9>:1: 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
  hdata1['bhk']=hdata1['size'].apply(lambda x:int(x.split(' ')[0]))
```

```
hdata1.total_sqft.unique()
```

```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

```
def is_float(x):
  try:
    float(x)
  except ValueError:
    return False
  return True
```

```
float(2+3)
```

```
5.0
```

```
hdata1[-hdata1['total_sqft'].apply(is_float)].head(10)
```

	area_type	location	size	total_sqft	bath	balcony	price	bhk
30	Super built-up Area	Yelahanka	4 BHK	2100 - 2850	4.0	0.0	186.000	4
122	Super built-up Area	Hebbal	4 BHK	3067 - 8156	4.0	0.0	477.000	4
137	Super built-up Area	8th Phase JP Nagar	2 BHK	1042 - 1105	2.0	0.0	54.005	2
165	Super built-up Area	Sarjapur	2 BHK	1145 - 1340	2.0	0.0	43.490	2
188	Super built-up Area	KR Puram	2 BHK	1015 - 1540	2.0	0.0	56.800	2
410	Super built-up Area	Kengeri	1 BHK	34.46Sq. Meter	1.0	0.0	18.500	1
549	Super built-up Area	Hennur Road	2 BHK	1195 - 1440	2.0	0.0	63.770	2
648	Built-up Area	Arekere	9 Bedroom	4125Perch	9.0	2.0	265.000	9
661	Super built-up Area	Yelahanka	2 BHK	1120 - 1145	2.0	0.0	48.130	2
672	Built-up Area	Bettahalsoor	4 Bedroom	3090 - 5002	4.0	0.0	445.000	4

```
def convert_sqft_to_num(x):
  tokens=x.split('-')
  if len(tokens)==2:
    return (float(tokens[0])+float(tokens[1]))/2
  try:
    return float(x)
  except:
    return None
```

```
hdata2=hdata1.copy()
#apply function
hdata2.total_sqft=hdata2.total_sqft.apply(convert_sqft_to_num)
#consider only null values
hdata2=hdata2[hdata2.total_sqft.notnull()]
hdata2.head()
```

	area_type	location	size	total_sqft	bath	balcony	price	bhk
0	Super built-up Area	Electronic City Phase II	2 BHK	1056.0	2.0	1.0	39.07	2
1	Plot Area	Chikka Tirupathi	4 Bedroom	2600.0	5.0	3.0	120.00	4
2	Built-up Area	Uttarahalli	3 BHK	1440.0	2.0	3.0	62.00	3
3	Super built-up Area	Lingadheeranahalli	3 BHK	1521.0	3.0	1.0	95.00	3
4	Super built-up Area	Kothanur	2 BHK	1200.0	2.0	1.0	51.00	2

```
print(hdata2.iloc[30])
print('-----------------')
hdata1.iloc[30]
```

```
area_type     Super built-up  Area
location              Yelahanka
size                   4 BHK
total_sqft            2475.0
bath                     4.0
balcony                  0.0
price                  186.0
bhk                        4
Name: 30, dtype: object
-----------------
area_type     Super built-up  Area
location              Yelahanka
size                   4 BHK
total_sqft         2100 - 2850
bath                     4.0
balcony                  0.0
price                  186.0
bhk                        4
Name: 30, dtype: object
```

```
hdata3=hdata2.copy()
hdata3['price_per_sqft']=hdata3['price']*100000/hdata3['total_sqft']
hdata3.head()
```

	area_type	location	size	total_sqft	bath	balcony	price	bhk	price_per_sqft
0	Super built-up Area	Electronic City Phase II	2 BHK	1056.0	2.0	1.0	39.07	2	3699.810606
1	Plot Area	Chikka Tirupathi	4 Bedroom	2600.0	5.0	3.0	120.00	4	4615.384615
2	Built-up Area	Uttarahalli	3 BHK	1440.0	2.0	3.0	62.00	3	4305.555556
3	Super built-up Area	Lingadheeranahalli	3 BHK	1521.0	3.0	1.0	95.00	3	6245.890861
4	Super built-up Area	Kothanur	2 BHK	1200.0	2.0	1.0	51.00	2	4250.000000

```
hdata3.describe()
```

	total_sqft	bath	balcony	price	bhk	price_per_sqft
count	13200.000000	13200.000000	13200.000000	13200.000000	13200.000000	1.320000e+04
mean	1555.302783	2.691136	1.602348	112.276178	2.800833	7.920759e+03
std	1237.323445	1.338915	0.804268	149.175995	1.292843	1.067272e+05
min	1.000000	1.000000	0.000000	8.000000	1.000000	2.678298e+02
25%	1100.000000	2.000000	1.000000	50.000000	2.000000	4.267701e+03
50%	1275.000000	2.000000	2.000000	71.850000	3.000000	5.438331e+03
75%	1672.000000	3.000000	2.000000	120.000000	3.000000	7.317073e+03
max	52272.000000	40.000000	3.000000	3600.000000	43.000000	1.200000e+07

```
hdata3['location'].value_counts()
```

```
Whitefield                 532
Sarjapur  Road             392
Electronic City            302
Kanakpura Road             264
Thanisandra                232
                          ...
Indiranagar HAL 2nd Stage    1
Maruthi HBCS Layout          1
K R C kothanur               1
1Channasandra                1
Abshot Layout                1
Name: location, Length: 1298, dtype: int64
```

```
hdata3.location=hdata3.location.apply(lambda x: x.strip())
location_stats=hdata3['location'].value_counts(ascending=False)
location_stats
```

```
Whitefield                 533
Sarjapur  Road             392
Electronic City            304
Kanakpura Road             264
Thanisandra                235
                          ...
Rajanna Layout               1
Subramanyanagar              1
Lakshmipura Vidyaanyapura    1
Malur Hosur Road             1
Abshot Layout                1
Name: location, Length: 1287, dtype: int64
```

```
print(len(location_stats[location_stats>10]))
print(len(location_stats[location_stats<=10]))
```

```
240
1047
```

```
#Dimensionality Reduction
```

```
location_stats_less_than_10=location_stats[location_stats<=10]
location_stats_less_than_10
```

```
BTM 1st Stage               10
Gunjur Palya                10
Nagappa Reddy Layout        10
Sector 1 HSR Layout         10
Thyagaraja Nagar            10
                            ..
Rajanna Layout               1
Subramanyanagar              1
Lakshmipura Vidyaanyapura    1
Malur Hosur Road             1
Abshot Layout                1
Name: location, Length: 1047, dtype: int64
```

```
print(len(hdata3.location.unique()))
hdata3.location=hdata3.location.apply(lambda x:'other' if x in location_stats_less_than_10 else x)
len(hdata3.location.unique())
```

```
1287
241
```

```
hdata3.info()
```

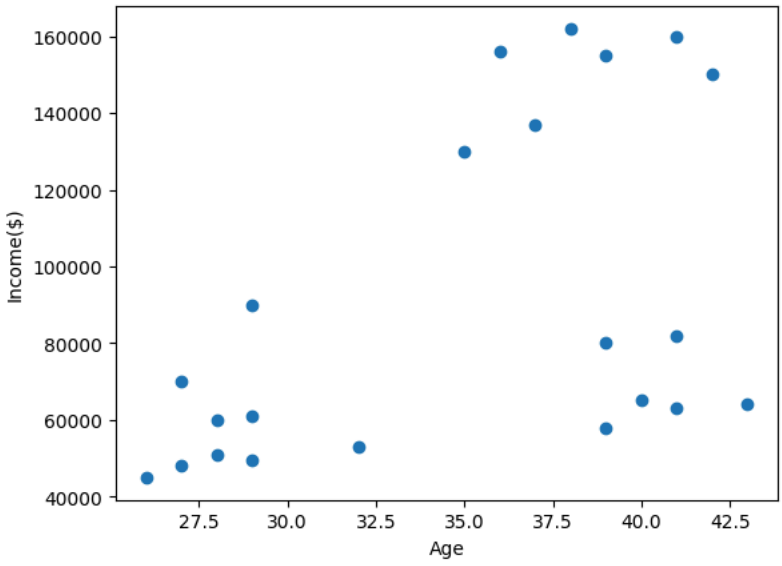
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13200 entries, 0 to 13319
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   area_type       13200 non-null  object
 1   location        13200 non-null  object
 2   size            13200 non-null  object
 3   total_sqft      13200 non-null  float64
 4   bath            13200 non-null  float64
 5   balcony         13200 non-null  float64
 6   price           13200 non-null  float64
 7   bhk             13200 non-null  int64
 8   price_per_sqft  13200 non-null  float64
dtypes: float64(5), int64(1), object(3)
memory usage: 1.0+ MB
```

```
#ML Class 15
#K Means Clustering
```

```
inc=pd.read_csv('income.csv')
inc
```

	Name	Age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000
5	Gautam	39	155000
6	David	41	160000
7	Andrea	38	162000
8	Brad	36	156000
9	Angelina	35	130000
10	Donald	37	137000
11	Tom	26	45000
12	Arnold	27	48000
13	Jared	28	51000
14	Stark	29	49500
15	Ranbir	32	53000
16	Dipika	40	65000
17	Priyanka	41	63000
18	Nick	43	64000
19	Alia	39	80000
20	Sid	41	82000
21	Abdul	39	58000

```
plt.scatter(inc['Age'],inc['Income($)'])
plt.xlabel('Age')
plt.ylabel('Income($)')
plt.show()
```



Age Na_to_K HighNatoK

```
from sklearn.cluster import KMeans
km=KMeans(n_clusters=3,init='random')
y_pred=km.fit_predict(inc[['Age','Income($)']])
y_pred
```

```
/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
  warnings.warn(
```

```
array([2, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1],
      dtype=int32)
```

```
inc['Cluster']=y_pred
inc
```

	Name	Age	Income($)	Cluster
0	Rob	27	70000	2
1	Michael	29	90000	2
2	Mohan	29	61000	1
3	Ismail	28	60000	1
4	Kory	42	150000	0
5	Gautam	39	155000	0
6	David	41	160000	0
7	Andrea	38	162000	0
8	Brad	36	156000	0
9	Angelina	35	130000	0
10	Donald	37	137000	0
11	Tom	26	45000	1
12	Arnold	27	48000	1
13	Jared	28	51000	1
14	Stark	29	49500	1
15	Ranbir	32	53000	1
16	Dipika	40	65000	1
17	Priyanka	41	63000	1
18	Nick	43	64000	1
19	Alia	39	80000	2
20	Sid	41	82000	2
21	Abdul	39	58000	1

```
km.cluster_centers_
```

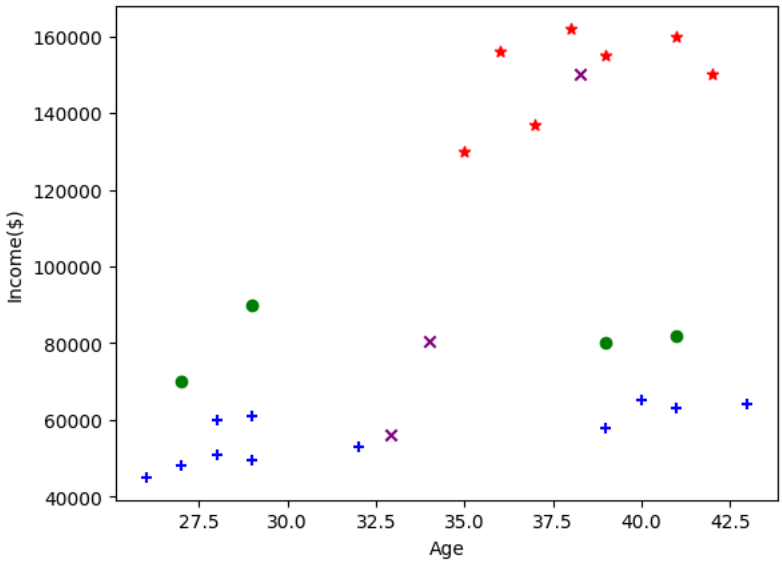
```
array([[3.82857143e+01, 1.50000000e+05],
       [3.29090909e+01, 5.61363636e+04],
       [3.40000000e+01, 8.05000000e+04]])
```

```
inc1=inc[inc.Cluster==0]
inc2=inc[inc.Cluster==1]
inc3=inc[inc.Cluster==2]
inc1
```

	Name	Age	Income($)	Cluster
4	Kory	42	150000	0
5	Gautam	39	155000	0
6	David	41	160000	0
7	Andrea	38	162000	0
8	Brad	36	156000	0
9	Angelina	35	130000	0
10	Donald	37	137000	0

```
plt.scatter(inc1['Age'],inc1['Income($)'],marker='*',c='red')
plt.scatter(inc2['Age'],inc2['Income($)'],marker='+',c='blue')
plt.scatter(inc3['Age'],inc3['Income($)'],marker='o',c='green')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],marker='x',c='purple')
plt.xlabel('Age')
plt.ylabel('Income($)')
```

```
Text(0, 0.5, 'Income($)')
```




```
#Preprocessing using MinMaxScaler
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
scale.fit(inc[['Age']])
inc['Age']=scale.transform(inc[['Age']])
scale.fit(inc[['Income($)']])
inc['Income($)']=scale.transform(inc[['Income($)']])
```

```
inc
```

	Name	Age	Income($)	Cluster
0	Rob	0.058824	0.213675	2
1	Michael	0.176471	0.384615	2
2	Mohan	0.176471	0.136752	1
3	Ismail	0.117647	0.128205	1
4	Kory	0.941176	0.897436	0
5	Gautam	0.764706	0.940171	0
6	David	0.882353	0.982906	0
7	Andrea	0.705882	1.000000	0
8	Brad	0.588235	0.948718	0
9	Angelina	0.529412	0.726496	0
10	Donald	0.647059	0.786325	0
11	Tom	0.000000	0.000000	1
12	Arnold	0.058824	0.025641	1
13	Jared	0.117647	0.051282	1
14	Stark	0.176471	0.038462	1
15	Ranbir	0.352941	0.068376	1
16	Dipika	0.823529	0.170940	1
17	Priyanka	0.882353	0.153846	1
18	Nick	1.000000	0.162393	1
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	1

```
km.cluster_centers_
```

```
array([[3.82857143e+01, 1.50000000e+05],
       [3.29090909e+01, 5.61363636e+04],
       [3.40000000e+01, 8.05000000e+04]])
```

```
km.labels_
```

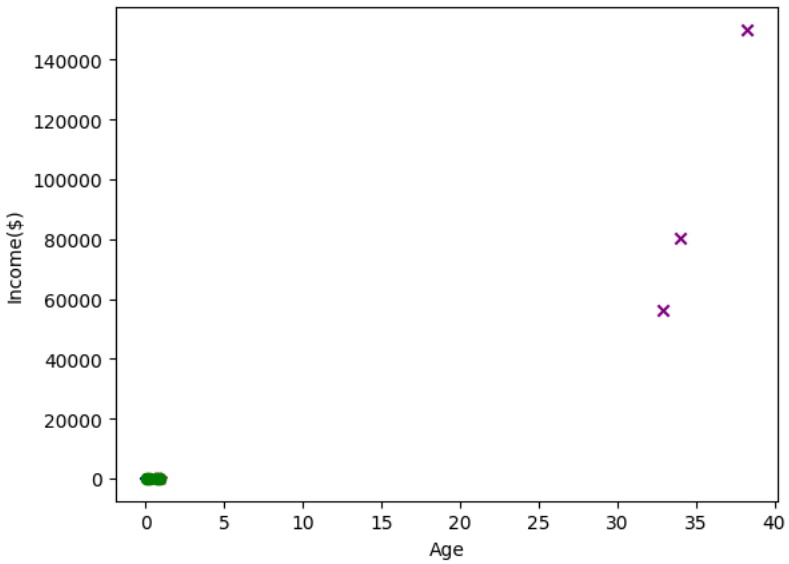
```
array([2, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 1],
      dtype=int32)
```

```
inc1=inc[inc.Cluster==0]
inc2=inc[inc.Cluster==1]
inc3=inc[inc.Cluster==2]
inc1
```

	Name	Age	Income($)	Cluster
4	Kory	0.941176	0.897436	0
5	Gautam	0.764706	0.940171	0
6	David	0.882353	0.982906	0
7	Andrea	0.705882	1.000000	0
8	Brad	0.588235	0.948718	0
9	Angelina	0.529412	0.726496	0
10	Donald	0.647059	0.786325	0

```
plt.scatter(inc1['Age'],inc1['Income($)'],marker='*',c='red')
plt.scatter(inc2['Age'],inc2['Income($)'],marker='+',c='blue')
plt.scatter(inc3['Age'],inc3['Income($)'],marker='o',c='green')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],marker='x',c='purple')
plt.xlabel('Age')
plt.ylabel('Income($)')
```

```
Text(0, 0.5, 'Income($)')
```




```
km.inertia_
```

```
1577046058.883117
```

```
inc1
```

	Name	Age	Income($)	Cluster
4	Kory	0.941176	0.897436	0
5	Gautam	0.764706	0.940171	0
6	David	0.882353	0.982906	0
7	Andrea	0.705882	1.000000	0
8	Brad	0.588235	0.948718	0
9	Angelina	0.529412	0.726496	0
10	Donald	0.647059	0.786325	0

```
inc2
```

	Name	Age	Income($)	Cluster
2	Mohan	0.176471	0.136752	1
3	Ismail	0.117647	0.128205	1
11	Tom	0.000000	0.000000	1
12	Arnold	0.058824	0.025641	1
13	Jared	0.117647	0.051282	1
14	Stark	0.176471	0.038462	1
15	Ranbir	0.352941	0.068376	1
16	Dipika	0.823529	0.170940	1
17	Priyanka	0.882353	0.153846	1
18	Nick	1.000000	0.162393	1
21	Abdul	0.764706	0.111111	1

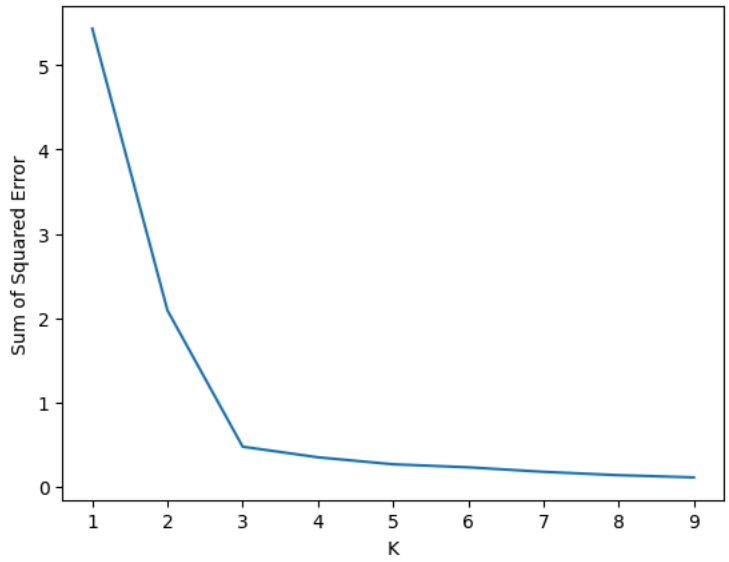
```
inc3
```

	Name	Age	Income($)	Cluster
0	Rob	0.058824	0.213675	2
1	Michael	0.176471	0.384615	2
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2

```
sse=[]
k_rng=range(1,10)
for k in k_rng:
  km=KMeans(n_clusters=k)
  km.fit(inc[['Age','Income($)']])
  #parameter inertia gives sum of error
  sse.append(km.inertia_)
```

```
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
```

```
plt.xlabel('K')
plt.ylabel('Sum of Squared Error')
plt.plot(k_rng,sse)
plt.show()
```




```
#Using Iris Dataset
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
from sklearn.datasets import load_iris
iris=load_iris()
```

```
irisdt=pd.DataFrame(iris['data'],columns=iris['feature_names'])
irisdt
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

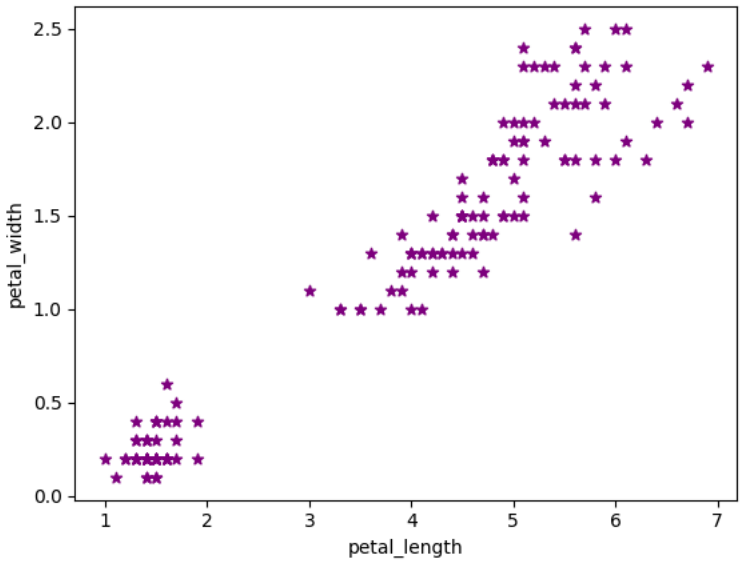
```
irisdata=irisdt.drop(['sepal length (cm)','sepal width (cm)'],axis=1)
irisdata
```

	petal length (cm)	petal width (cm)
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2
...	...	...
145	5.2	2.3
146	5.0	1.9
147	5.2	2.0
148	5.4	2.3
149	5.1	1.8

150 rows × 2 columns

```
plt.scatter(irisdata['petal length (cm)'],irisdata['petal width (cm)'],marker='*',color='purple')
plt.xlabel('petal_length')
plt.ylabel('petal_width')
```

```
Text(0, 0.5, 'petal_width')
```




```
from sklearn.preprocessing import MinMaxScaler

scale=MinMaxScaler()

scale.fit(irisdata[['petal length (cm)']])
irisdata['petal length (cm)']=scale.transform(irisdata[['petal length (cm)']])

scale.fit(irisdata[['petal width (cm)']])
irisdata['petal width (cm)']=scale.transform(irisdata[['petal width (cm)']])
```

```
irisdata
```

	petal length (cm)	petal width (cm)
0	0.067797	0.041667
1	0.067797	0.041667
2	0.050847	0.041667
3	0.084746	0.041667
4	0.067797	0.041667
...	...	...
145	0.711864	0.916667
146	0.677966	0.750000
147	0.711864	0.791667
148	0.745763	0.916667
149	0.694915	0.708333

150 rows × 2 columns

```
from sklearn.cluster import KMeans

km_iris=KMeans(n_clusters=3,init='random')
km_iris.fit(irisdata[['petal length (cm)','petal width (cm)']])
y_pred = km_iris.predict(irisdata[['petal length (cm)','petal width (cm)']])
print(y_pred)
cluster_centers=km_iris.cluster_centers_
print(cluster_centers)
```

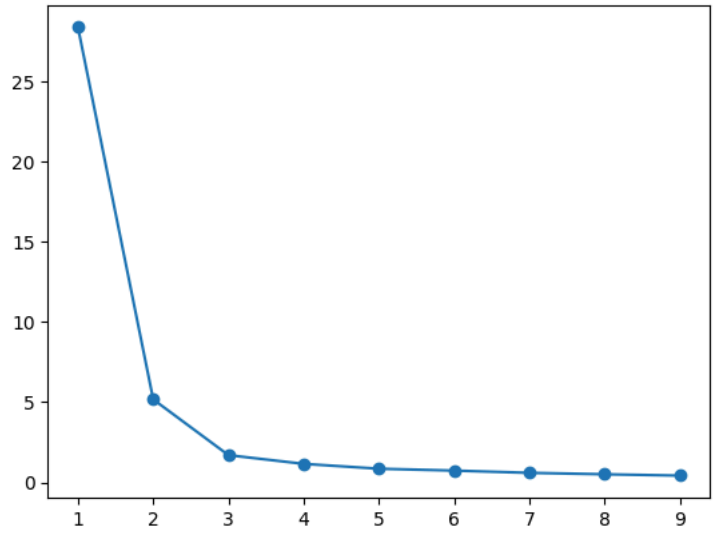
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2
 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 2 1 1 1 1
 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
[[0.07830508 0.06083333]
 [0.7740113  0.81510417]
 [0.55867014 0.51041667]]
/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
  warnings.warn(
```

```
ssei=[]
k_rngi=range(1,10)
for k in k_rngi:
  kmi=KMeans(n_clusters=k,init='random')
  kmi.fit(irisdata[['petal length (cm)','petal width (cm)']])
  ssei.append(kmi.inertia_)
  print(ssei)
```

```
/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
  warnings.warn(
[28.368353219727197]
[28.368353219727197, 5.176463590044368]
[28.368353219727197, 5.176463590044368, 1.701874688192097]
[28.368353219727197, 5.176463590044368, 1.701874688192097, 1.1588792731667124]
[28.368353219727197, 5.176463590044368, 1.701874688192097, 1.1588792731667124, 0.8558757147788012]
[28.368353219727197, 5.176463590044368, 1.701874688192097, 1.1588792731667124, 0.8558757147788012, 0.7388855104730023]
[28.368353219727197, 5.176463590044368, 1.701874688192097, 1.1588792731667124, 0.8558757147788012, 0.7388855104730023, 0.598835172589676]
[28.368353219727197, 5.176463590044368, 1.701874688192097, 1.1588792731667124, 0.8558757147788012, 0.7388855104730023, 0.598835172589676, 0.50938
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
/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
  warnings.warn(
[28.368353219727197, 5.176463590044368, 1.701874688192097, 1.1588792731667124, 0.8558757147788012, 0.7388855104730023, 0.598835172589676, 0.50938
```

```
plt.plot(k_rngi,ssei,marker='o')
```

```
[<matplotlib.lines.Line2D at 0x7a7ca330ae30>]
```




```
km_iris.cluster_centers_
```

```
array([[0.07830508, 0.06083333],
       [0.7740113 , 0.81510417],
       [0.55867014, 0.51041667]])
```

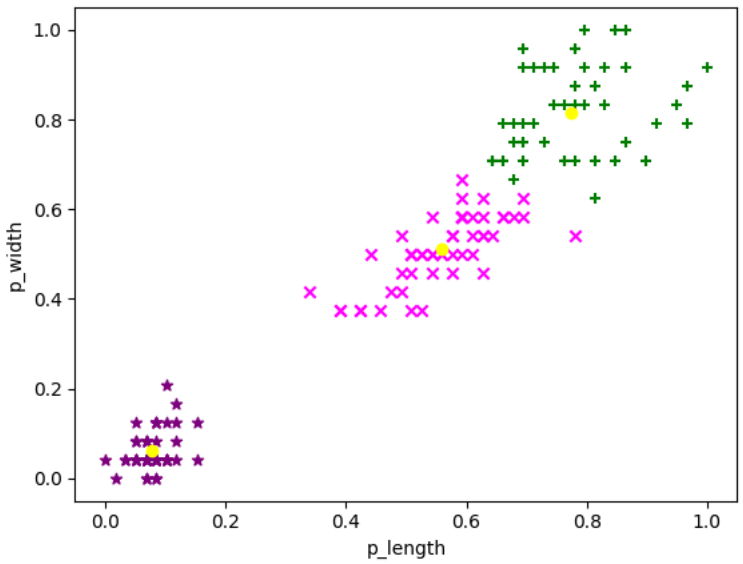
```
irisdata['cluster']=y_pred
irisdata
```

	petal length (cm)	petal width (cm)	cluster
0	0.067797	0.041667	0
1	0.067797	0.041667	0
2	0.050847	0.041667	0
3	0.084746	0.041667	0
4	0.067797	0.041667	0
...	...	...	...
145	0.711864	0.916667	1
146	0.677966	0.750000	1
147	0.711864	0.791667	1
148	0.745763	0.916667	1
149	0.694915	0.708333	1

150 rows × 3 columns

```
iris1=irisdata[irisdata.cluster==0]
iris2=irisdata[irisdata.cluster==1]
iris3=irisdata[irisdata.cluster==2]

plt.scatter(iris1['petal length (cm)'],iris1['petal width (cm)'],marker='*',c='purple')
plt.scatter(iris2['petal length (cm)'],iris2['petal width (cm)'],marker='+',c='green')
plt.scatter(iris3['petal length (cm)'],iris3['petal width (cm)'],marker='x',c='magenta')
plt.scatter(km_iris.cluster_centers_[:,0],km_iris.cluster_centers_[:,1],marker='o',c='yellow')
plt.xlabel('p_length')
plt.ylabel('p_width')
```

Text(0, 0.5, 'p_width')




```
#K-Means++
```

```
mc=pd.read_csv('Mall_Customers.csv')
mc
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

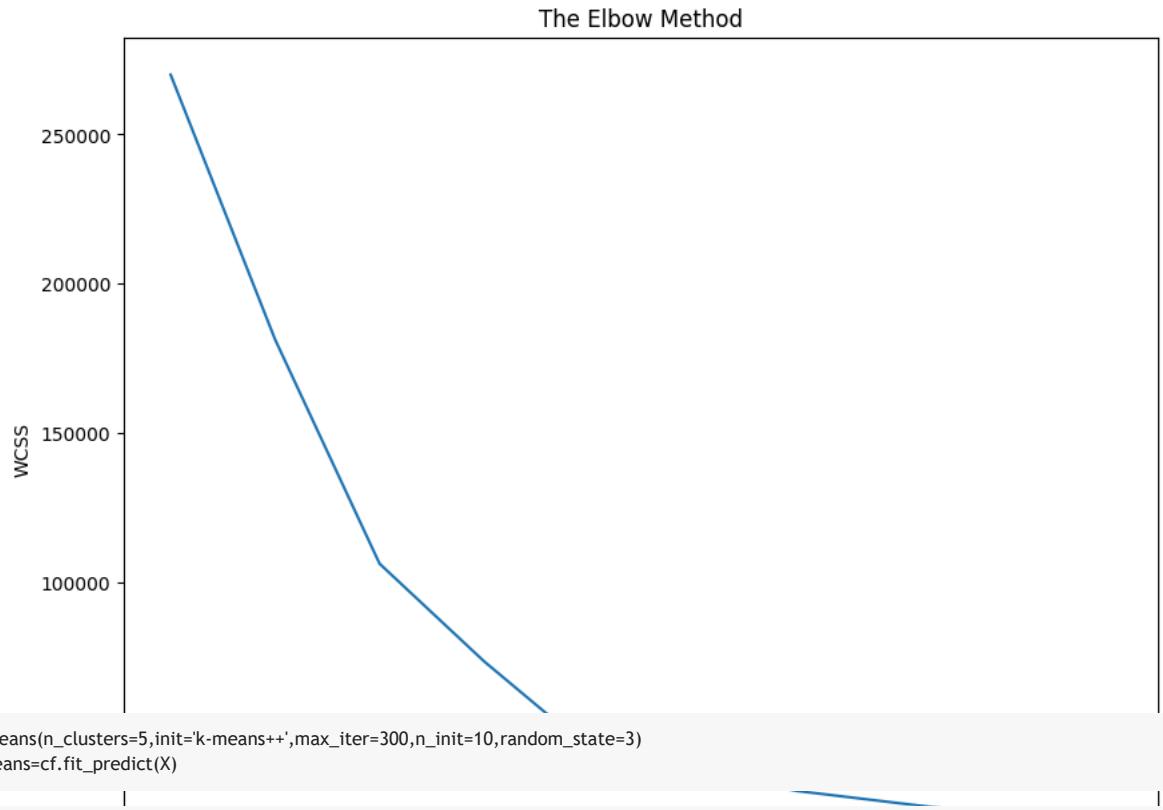
200 rows × 5 columns

```
X=mc.iloc[:,[3,4]].values
X
```

```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
       [ 23,  29],
       [ 23,  98],
       [ 24,  35],
       [ 24,  73],
       [ 25,   5],
       [ 25,  73],
       [ 28,  14],
       [ 28,  82],
       [ 28,  32],
```

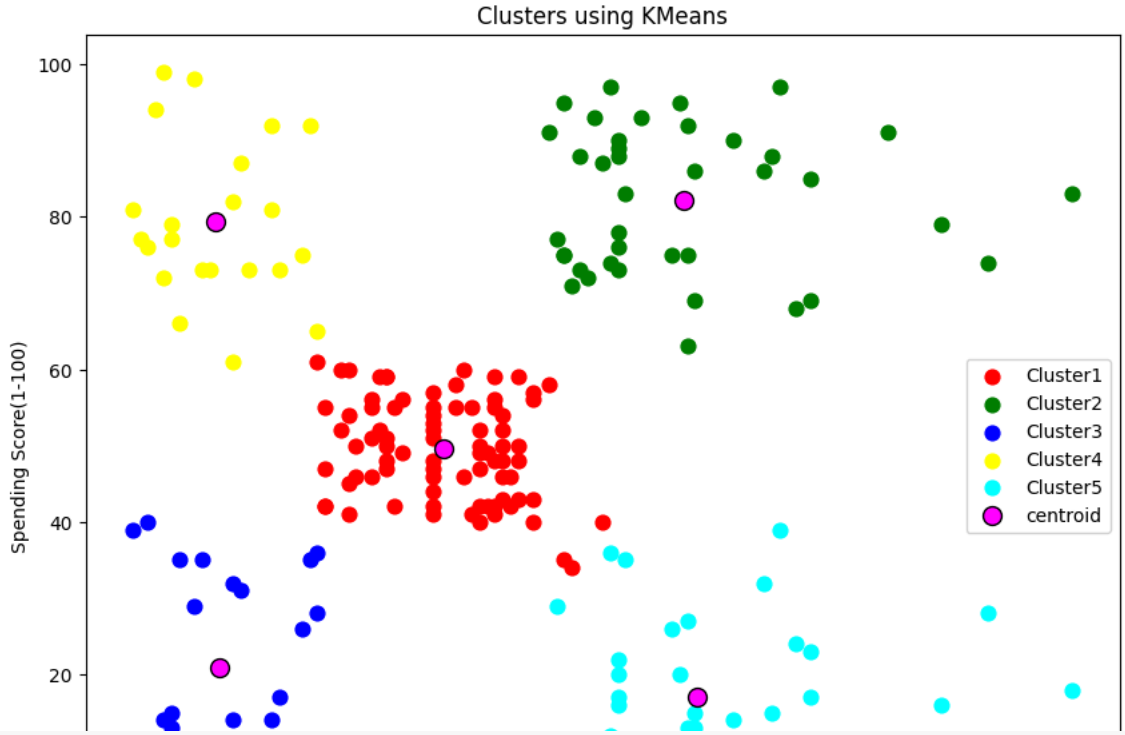
```
[ 28,  61],
[ 29,  31],
[ 29,  87],
[ 30,   4],
[ 30,  73],
[ 33,   4],
[ 33,  92],
[ 33,  14],
[ 33,  81],
[ 34,  17],
[ 34,  73],
[ 37,  26],
[ 37,  75],
[ 38,  35],
[ 38,  92],
[ 39,  36],
[ 39,  61],
[ 39,  28],
[ 39,  65],
[ 40,  55],
[ 40,  47],
[ 40,  42],
[ 40,  42],
[ 42,  52],
[ 42,  60],
[ 43,  54],
[ 43,  60],
[ 43,  45],
[ 43,  41],
[ 44,  50],
[ 44,  46]
```

```
from sklearn.cluster import KMeans
fig=plt.figure(figsize=(10,8))
#within cluster sum of square
WCSS=[]
for i in range(1,11):
  cf=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=3)
  cf.fit(X)
  WCSS.append(cf.inertia_)
plt.plot(range(1,11),WCSS)
plt.title('The Elbow Method')
plt.xlabel('No: of Clusters')
plt.ylabel('WCSS')
plt.show()
```




```
cf=KMeans(n_clusters=5,init='k-means++',max_iter=300,n_init=10,random_state=3)
y_kmeans=cf.fit_predict(X)
```

```
fig=plt.figure(figsize=(10,8))
plt.scatter(X[y_kmeans==0,0],X[y_kmeans==0,1],color='red',s=60,label='Cluster1')
plt.scatter(X[y_kmeans==1,0],X[y_kmeans==1,1],color='green',s=60,label='Cluster2')
plt.scatter(X[y_kmeans==2,0],X[y_kmeans==2,1],color='blue',s=60,label='Cluster3')
plt.scatter(X[y_kmeans==3,0],X[y_kmeans==3,1],color='yellow',s=60,label='Cluster4')
plt.scatter(X[y_kmeans==4,0],X[y_kmeans==4,1],color='cyan',s=60,label='Cluster5')
#cluster centres
plt.scatter(cf.cluster_centers_[:,0],cf.cluster_centers_[:,1],color='magenta',s=100,label='centroid',edgecolors='black')
plt.legend()
plt.title('Clusters using KMeans')
plt.xlabel('Annual Income(k$)')
plt.ylabel('Spending Score(1-100)')
plt.show()
```




```
#Mini Batch K-Means
from sklearn.cluster import MiniBatchKMeans
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

```
from sklearn.datasets import load_iris
iris=load_iris()
```

```
df=pd.DataFrame(iris['data'],columns=iris.feature_names)
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
df['flower']=iris.target
df.tail()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

```
df.drop(['sepal length (cm)','sepal width (cm)','flower'],axis=1,inplace=True)
df.head(3)
```

	petal length (cm)	petal width (cm)
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2

```
km=MiniBatchKMeans(n_clusters=3,random_state=0,batch_size=100)
yp=km.fit_predict(df)
yp
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 3 to 'auto' in 1.4. Set t
  warnings.warn(
array([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, 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, 2, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0, 0, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int32)
```

```
df['cluster']=yp
df.head(2)
```

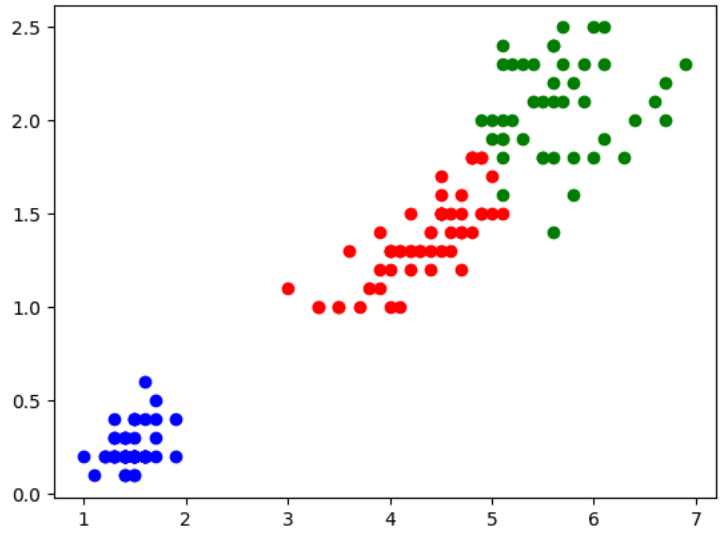
	petal length (cm)	petal width (cm)	cluster
0	1.4	0.2	1
1	1.4	0.2	1

```
df.cluster.unique()
```

```
array([1, 0, 2], dtype=int32)
```

```
df1=df[df.cluster==0]
df2=df[df.cluster==1]
df3=df[df.cluster==2]

plt.scatter(df1['petal length (cm)'],df1['petal width (cm)'],color='red')
plt.scatter(df2['petal length (cm)'],df2['petal width (cm)'],color='blue')
plt.scatter(df3['petal length (cm)'],df3['petal width (cm)'],color='green')
```

```
<matplotlib.collections.PathCollection at 0x7a50d2fd2dd0>
```




```
#ML Class 16
#Hierarchical Clustering
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
insure=pd.read_csv('insurance.csv')
insure
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

```
insure.shape
```

```
(1338, 7)
```

```
insure.columns
```

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype='object')
```

```
insure.describe
```

```
pandas.core.generic.NDFrame.describe
def describe(percentiles=None, include=None, exclude=None, datetime_is_numeric: bool_t=False) -> NDFrameT

Generate descriptive statistics.

Descriptive statistics include those that summarize the central
tendency, dispersion and shape of a
dataset's distribution, excluding ``NaN`` values.
```

```
insure.info
```

```
pandas.core.frame.DataFrame.info
def info(verbose: bool | None=None, buf: WriteBuffer[str] | None=None, max_cols: int | None=None, memory_usage: bool | str | None=None, show_counts: bool | None=None, null_counts: bool | None=None) -> None

Print a concise summary of a DataFrame.

This method prints information about a DataFrame including
the index dtype and columns, non-null values and memory usage.

Parameters
```

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
scale.fit(insure[['age']])
insure['age']=scale.transform(insure[['age']])
scale.fit(insure[['charges']])
insure['charges']=scale.transform(insure[['charges']])
insure
```

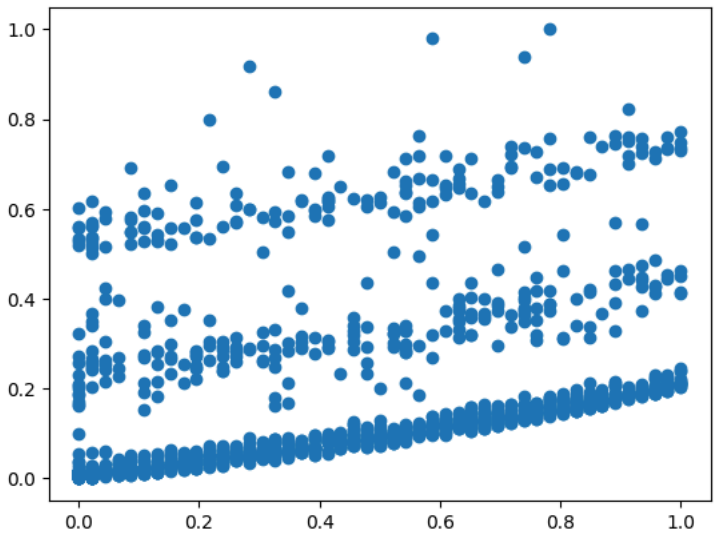
	age	sex	bmi	children	smoker	region	charges
0	0.021739	female	27.900	0	yes	southwest	0.251611
1	0.000000	male	33.770	1	no	southeast	0.009636
2	0.217391	male	33.000	3	no	southeast	0.053115
3	0.326087	male	22.705	0	no	northwest	0.333010
4	0.304348	male	28.880	0	no	northwest	0.043816
...	...	...	...	...	...	...	...
1333	0.695652	male	30.970	3	no	northwest	0.151299
1334	0.000000	female	31.920	0	no	northeast	0.017305
1335	0.000000	female	36.850	0	no	southeast	0.008108
1336	0.065217	female	25.800	0	no	southwest	0.014144
1337	0.934783	female	29.070	0	yes	northwest	0.447249

1338 rows × 7 columns

```
X=insure.iloc[:,[0,6]].values
X
```

```
array([[0.02173913, 0.25161076],
       [0.        , 0.00963595],
       [0.2173913 , 0.05311516],
       ...,
       [0.        , 0.00810808],
       [0.06521739, 0.01414352],
       [0.93478261, 0.44724873]])
```

```
plt.scatter(X[:,0],X[:,1])
plt.show()
```



```
!pip install scikit-learn

# Import necessary module
from sklearn.cluster import hierarchy as sch

# Plot dendrogram
plt.figure(figsize=(20, 7))
plt.title('Dendrogram')
sch.dendrogram(sch.linkage(X, method='centroid'))
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

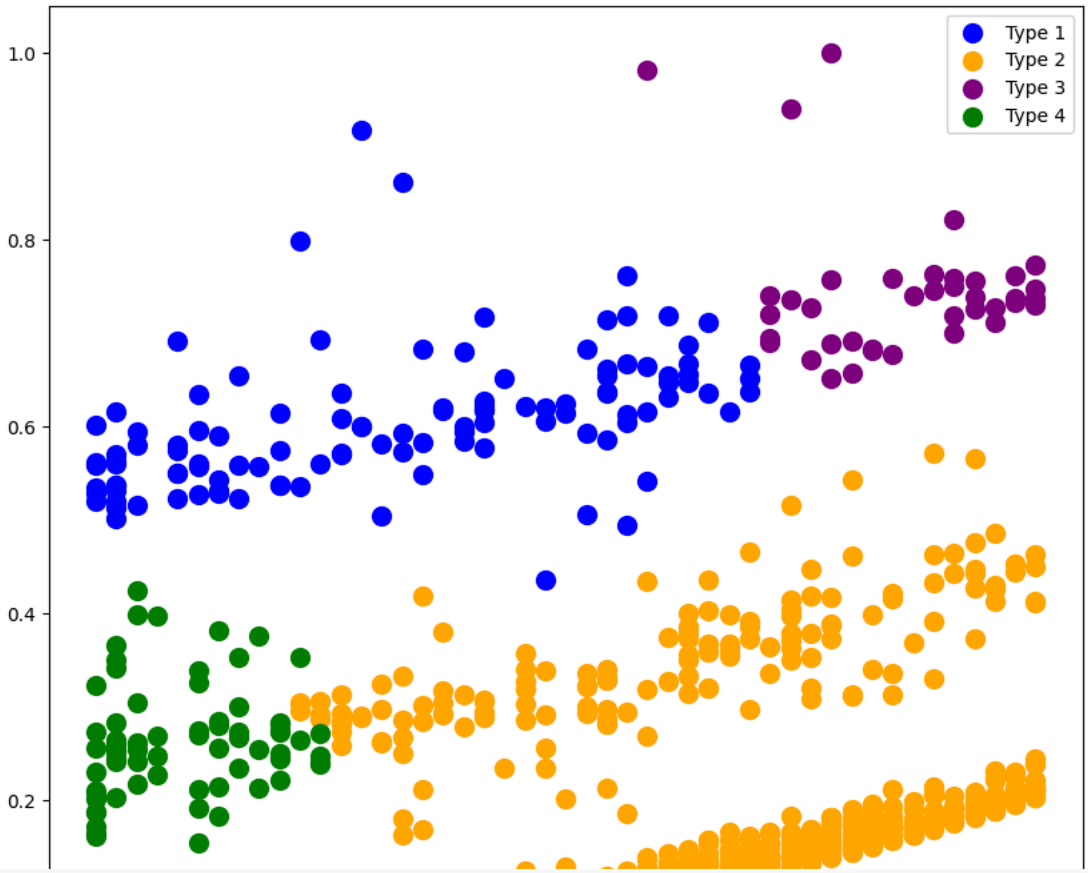
```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.0)
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-31-e07d592a8132> in <cell line: 4>()
      2
      3 # Import necessary module
----> 4 from sklearn.cluster import hierarchy as sch
      5
      6 # Plot dendrogram

ImportError: cannot import name 'hierarchy' from 'sklearn.cluster' (/usr/local/lib/python3.10/dist-packages/sklearn/cluster/__init__.py)

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

```
from sklearn.cluster import AgglomerativeClustering
cluster=AgglomerativeClustering(n_clusters=4,affinity='euclidean',linkage='average')
cluster.fit(X)
lab=cluster.labels_
lab
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983: FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will be
  warnings.warn(
array([3, 3, 3, ..., 3, 3, 1])
```

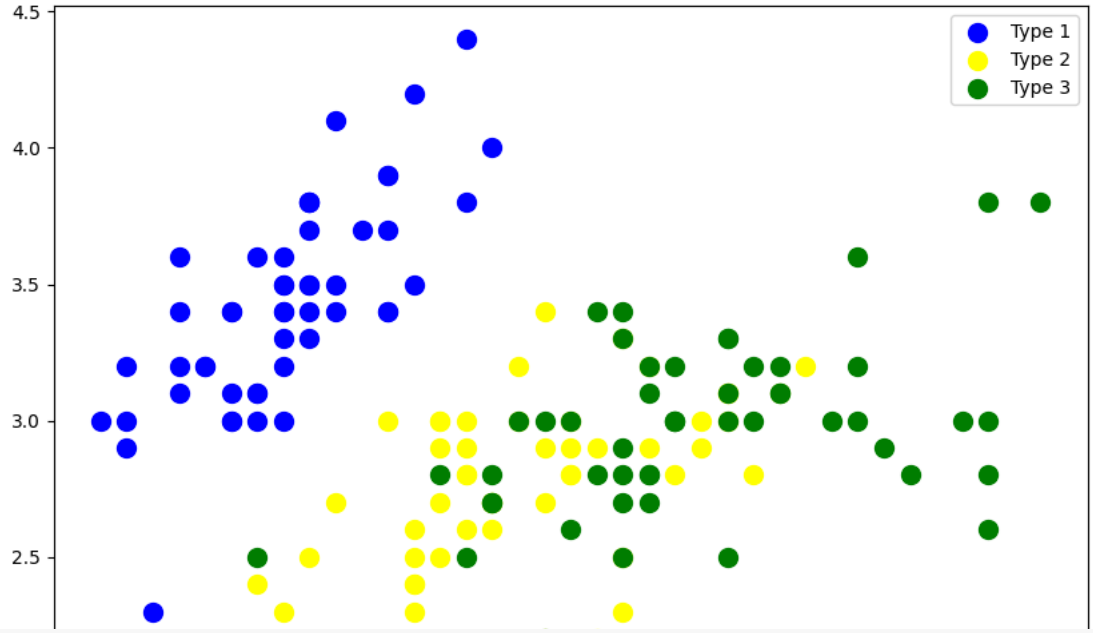
```
plt.figure(figsize=(10,10))
plt.scatter(X[lab==0,0],X[lab==0,1],s=100,c='blue',label='Type 1')
plt.scatter(X[lab==1,0],X[lab==1,1],s=100,c='orange',label='Type 2')
plt.scatter(X[lab==2,0],X[lab==2,1],s=100,c='purple',label='Type 3')
plt.scatter(X[lab==3,0],X[lab==3,1],s=100,c='green',label='Type 4')
plt.legend()
plt.show()
```




```
from sklearn import datasets
iris=datasets.load_iris()
iris_data=pd.DataFrame(iris.data)
iris_data.columns=iris.feature_names
iris_data['flower_type']=iris.target
iris_data.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	flower_type
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
iris_X=iris_data.iloc[:,[0,1,2,3]].values
iris_Y=iris_data.iloc[:,4].values

plt.figure(figsize=(10,7))
plt.scatter(iris_X[iris_Y==0,0],iris_X[iris_Y==0,1],s=100,c='blue',label='Type 1')
plt.scatter(iris_X[iris_Y==1,0],iris_X[iris_Y==1,1],s=100,c='yellow',label='Type 2')
plt.scatter(iris_X[iris_Y==2,0],iris_X[iris_Y==2,1],s=100,c='green',label='Type 3')
plt.legend()
plt.show()
```




```
import scipy.cluster.hierarchy as sc
plt.figure(figsize=(20,7))
plt.title('Dendogram')

sc.dendogram(sc.linkage(iris_x,method))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-4b0e31a864fb> in <cell line: 2>()
      1 import scipy.cluster.hierarchy as sc
----> 2 plt.figure(figsize=(20,7))
      3 plt.title('Dendogram')
      4
      5 sc.dendogram(sc.linkage(iris_x,method))

NameError: name 'plt' is not defined
```

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
```

```
mc=pd.read_csv('Mall_Customers.csv')
mc.head()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler()
scale.fit(mc[['Annual Income (k$)']])
mc['Annual Income (k$)']=scale.transform(mc[['Annual Income (k$)']])
scale.fit(mc[['Spending Score (1-100)']])
mc['Spending Score (1-100)']=scale.transform(mc[['Spending Score (1-100)']])
mc
```

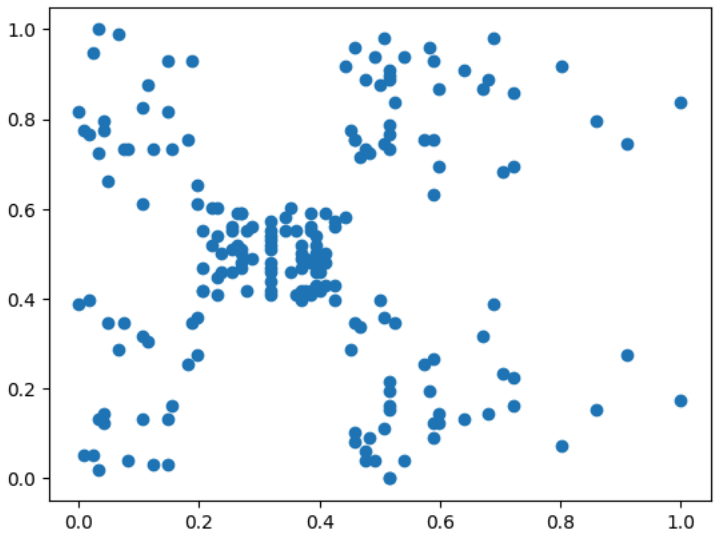
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	0.000000	0.387755
1	2	Male	21	0.000000	0.816327
2	3	Female	20	0.008197	0.051020
3	4	Female	23	0.008197	0.775510
4	5	Female	31	0.016393	0.397959
...	...	...	...	...	...
195	196	Female	35	0.860656	0.795918
196	197	Female	45	0.909836	0.275510
197	198	Male	32	0.909836	0.744898
198	199	Male	32	1.000000	0.173469
199	200	Male	30	1.000000	0.836735

200 rows × 5 columns

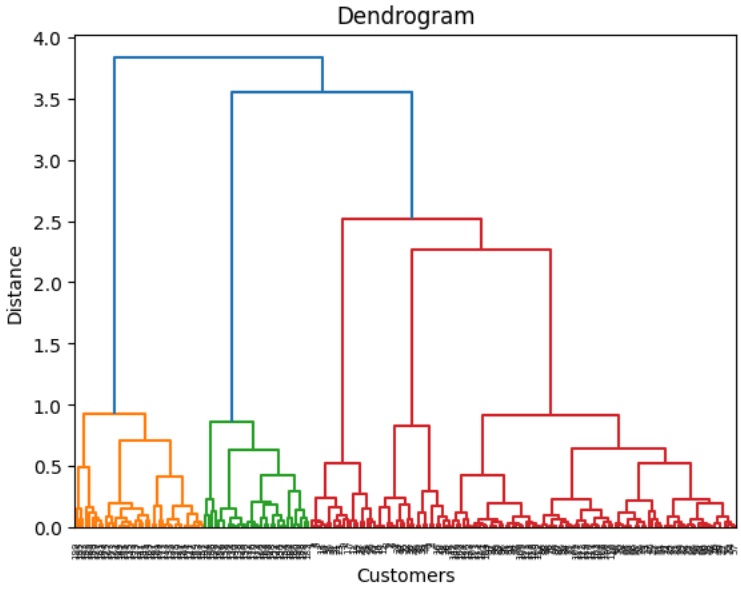
```
X=mc.iloc[:,[3,4]].values
X
```

```
array([[0.        , 0.3877551 ],
       [0.        , 0.81632653],
       [0.00819672, 0.05102041],
       [0.00819672, 0.7755102 ],
       [0.01639344, 0.39795918],
       [0.01639344, 0.76530612],
       [0.02459016, 0.05102041],
       [0.02459016, 0.94897959],
       [0.03278689, 0.02040816],
       [0.03278689, 0.7244898 ],
       [0.03278689, 0.13265306],
       [0.03278689, 1.        ],
       [0.04098361, 0.14285714],
       [0.04098361, 0.7755102 ],
       [0.04098361, 0.12244898],
       [0.04098361, 0.79591837],
       [0.04918033, 0.34693878],
       [0.04918033, 0.66326531],
       [0.06557377, 0.28571429],
       [0.06557377, 0.98979592],
       [0.07377049, 0.34693878],
       [0.07377049, 0.73469388],
       [0.08196721, 0.04081633],
       [0.08196721, 0.73469388],
       [0.10655738, 0.13265306],
       [0.10655738, 0.82653061],
       [0.10655738, 0.31632653],
       [0.10655738, 0.6122449 ],
       [0.1147541 , 0.30612245],
       [0.1147541 , 0.87755102],
       [0.12295082, 0.03061224],
       [0.12295082, 0.73469388],
       [0.14754098, 0.03061224],
       [0.14754098, 0.92857143],
       [0.14754098, 0.13265306],
       [0.14754098, 0.81632653],
       [0.1557377 , 0.16326531],
       [0.1557377 , 0.73469388],
       [0.18032787, 0.25510204],
       [0.18032787, 0.75510204],
       [0.18852459, 0.34693878],
       [0.18852459, 0.92857143],
       [0.19672131, 0.35714286],
       [0.19672131, 0.6122449 ],
       [0.19672131, 0.2755102 ],
       [0.19672131, 0.65306122],
       [0.20491803, 0.55102041],
       [0.20491803, 0.46938776],
       [0.20491803, 0.41836735],
       [0.20491803, 0.41836735],
       [0.22131148, 0.52040816],
       [0.22131148, 0.60204082],
       [0.2295082 , 0.54081633],
       [0.2295082 , 0.60204082],
       [0.2295082 , 0.44897959],
       [0.2295082 , 0.40816327],
       [0.23770492, 0.5       ],
       [0.23770492, 0.45918367],
```

```
plt.scatter(X[:,0],X[:,1])
plt.show()
```



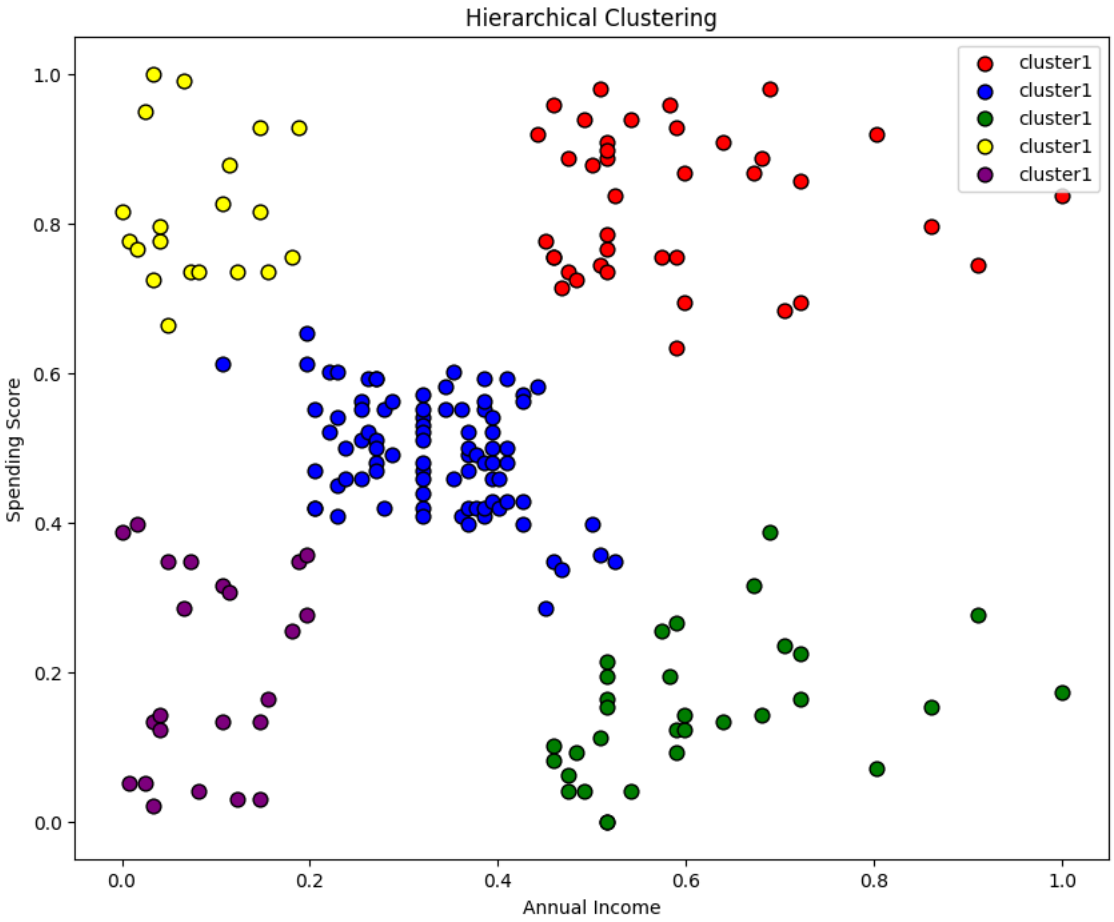
```
import scipy.cluster.hierarchy as sch

# Correct function name
dendogram=sch.dendrogram(sch.linkage(X,method='ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Distance')
plt.show()
```




```
from sklearn.cluster import AgglomerativeClustering
hc=AgglomerativeClustering(n_clusters=5,affinity='euclidean',linkage='ward')
y_hc=hc.fit_predict(X)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983: FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will be
  warnings.warn(
```

```
fit=plt.figure(figsize=(10,8))
plt.scatter(X[y_hc==0,0],X[y_hc==0,1],color='red',s=60,label='cluster1',edgecolor='black')
plt.scatter(X[y_hc==1,0],X[y_hc==1,1],color='blue',s=60,label='cluster1',edgecolor='black')
plt.scatter(X[y_hc==2,0],X[y_hc==2,1],color='green',s=60,label='cluster1',edgecolor='black')
plt.scatter(X[y_hc==3,0],X[y_hc==3,1],color='yellow',s=60,label='cluster1',edgecolor='black')
plt.scatter(X[y_hc==4,0],X[y_hc==4,1],color='purple',s=60,label='cluster1',edgecolor='black')
plt.legend()
plt.title('Hierarchical Clustering')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.show()
```

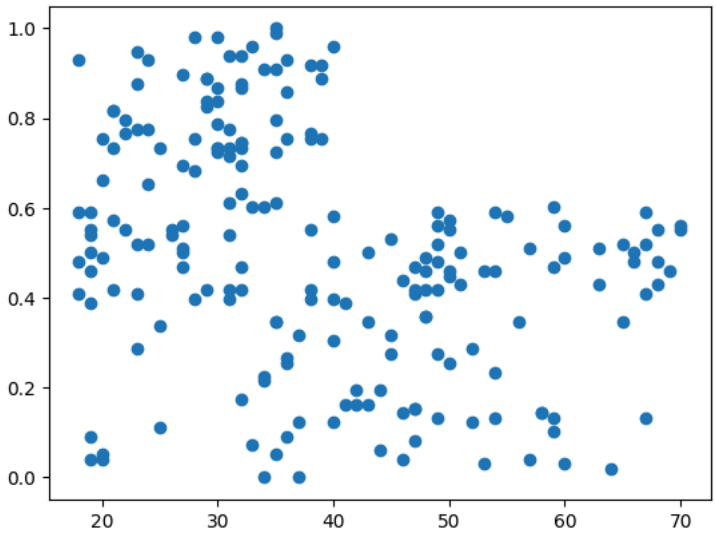



```
#Age vs Spending Score
Y=mc.iloc[:,[2,4]].values
Y
```

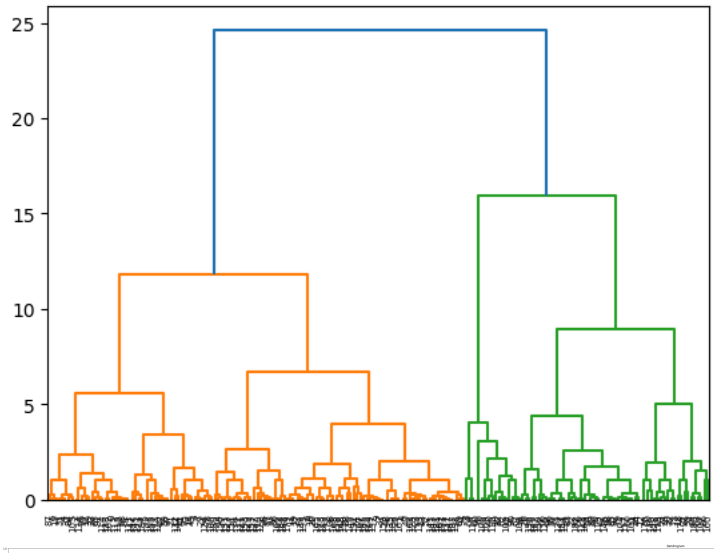
```
array([[1.90000000e+01, 3.87755102e-01],
       [2.10000000e+01, 8.16326531e-01],
       [2.00000000e+01, 5.10204082e-02],
       [2.30000000e+01, 7.75510204e-01],
       [3.10000000e+01, 3.97959184e-01],
       [2.20000000e+01, 7.65306122e-01],
       [3.50000000e+01, 5.10204082e-02],
       [2.30000000e+01, 9.48979592e-01],
       [6.40000000e+01, 2.04081633e-02],
       [3.00000000e+01, 7.24489796e-01],
       [6.70000000e+01, 1.32653061e-01],
       [3.50000000e+01, 1.00000000e+00],
       [5.80000000e+01, 1.42857143e-01],
       [2.40000000e+01, 7.75510204e-01],
       [3.70000000e+01, 1.22448980e-01],
       [2.20000000e+01, 7.95918367e-01],
       [3.50000000e+01, 3.46938776e-01],
       [2.00000000e+01, 6.63265306e-01],
       [5.20000000e+01, 2.85714286e-01],
       [3.50000000e+01, 9.89795918e-01],
       [3.50000000e+01, 3.46938776e-01],
       [2.50000000e+01, 7.34693878e-01],
       [4.60000000e+01, 4.08163265e-02],
       [3.10000000e+01, 7.34693878e-01],
       [5.40000000e+01, 1.32653061e-01],
       [2.90000000e+01, 8.26530612e-01],
       [4.50000000e+01, 3.16326531e-01],
       [3.50000000e+01, 6.12244898e-01],
       [4.00000000e+01, 3.06122449e-01],
       [2.30000000e+01, 8.77551020e-01],
       [6.00000000e+01, 3.06122449e-02],
       [2.10000000e+01, 7.34693878e-01],
       [5.30000000e+01, 3.06122449e-02],
       [1.80000000e+01, 9.28571429e-01],
       [4.90000000e+01, 1.32653061e-01],
       [2.10000000e+01, 8.16326531e-01],
       [4.20000000e+01, 1.63265306e-01],
       [3.00000000e+01, 7.34693878e-01],
       [3.60000000e+01, 2.55102041e-01],
       [2.00000000e+01, 7.55102041e-01],
       [6.50000000e+01, 3.46938776e-01],
       [2.40000000e+01, 9.28571429e-01],
       [4.80000000e+01, 3.57142857e-01],
```

```
[3.10000000e+01, 6.12244898e-01],
[4.90000000e+01, 2.75510204e-01],
[2.40000000e+01, 6.53061224e-01],
[5.00000000e+01, 5.51020408e-01],
[2.70000000e+01, 4.69387755e-01],
[2.90000000e+01, 4.18367347e-01],
[3.10000000e+01, 4.18367347e-01],
[4.90000000e+01, 5.20408163e-01],
[3.30000000e+01, 6.02040816e-01],
[3.10000000e+01, 5.40816327e-01],
[5.90000000e+01, 6.02040816e-01],
[5.00000000e+01, 4.48979592e-01],
[4.70000000e+01, 4.08163265e-01],
[5.10000000e+01, 5.00000000e-01],
[6.90000000e+01, 4.59183673e-01]
```

```
plt.scatter(Y[:,0],Y[:,1])
plt.show()
```




```
import scipy.cluster.hierarchy as sch
dendrogram=sch.dendrogram(sch.linkage(Y,method='average'))
plt.figure(figsize=(100,100))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Distance')
plt.show()
```



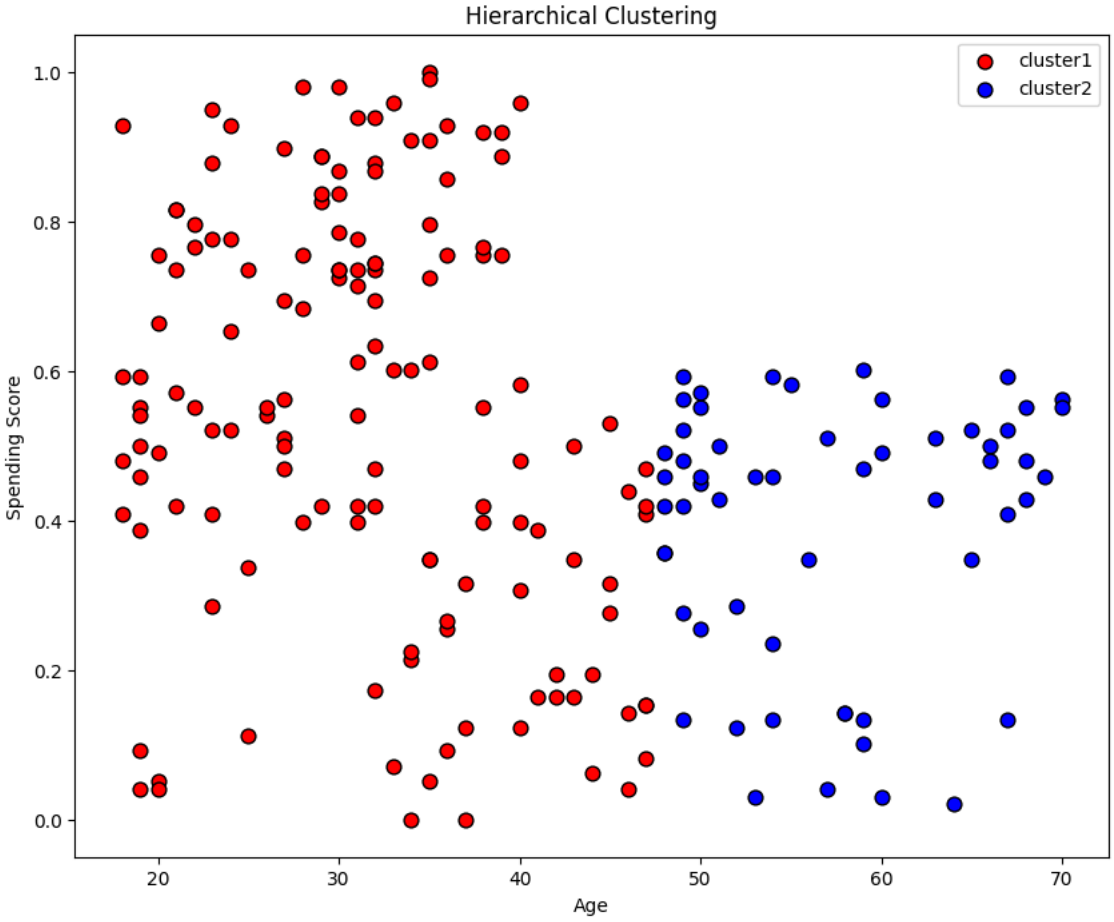

```
from sklearn.cluster import AgglomerativeClustering
hc1=AgglomerativeClustering(n_clusters=2,affinity='euclidean',linkage='complete')
y_hc1=hc1.fit_predict(Y)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983: FutureWarning: Attribute `affinity` was deprecated in version 1.2 and will be
  warnings.warn(
```

```
hc1.labels_
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0])
```

```
fig=plt.figure(figsize=(10,8))
plt.scatter(Y[y_hc1==0,0],Y[y_hc1==0,1],color='red',s=60,label='cluster1',edgecolor='black')
plt.scatter(Y[y_hc1==1,0],Y[y_hc1==1,1],color='blue',s=60,label='cluster2',edgecolor='black')

plt.legend()
plt.title('Hierarchical Clustering')
plt.ylabel('Spending Score')
plt.xlabel('Age')
plt.show()
```




```
#ML Class 17
#Prinicipal Component Analysis
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

```
from sklearn.datasets import load_breast_cancer
cancer=load_breast_cancer()
cancer.keys()
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
print(cancer['data'])
```

```
[[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]]
```

```
can_data=pd.DataFrame(cancer.data,columns=cancer.feature_names)
can_data
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16

569 rows × 30 columns

```
from sklearn.preprocessing import StandardScaler
scale=StandardScaler()
scale.fit(can_data)
```

▾ StandardScaler
StandardScaler()

```
scaled_dt=scale.transform(can_data)
```

```
scaled_dt
```

```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

```
np.mean(scaled_dt)
```

```
-6.118909323768877e-16
```

```
np.std(scaled_dt)
```

```
1.0
```

```
feat_cols=['feature'+str(i) for i in range(can_data.shape[1])]
```

```
normalised_df=pd.DataFrame(scaled_dt,columns=feat_cols)
normalised_df.tail()
```

	feature0	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	...	feature20	feature21	feat
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.219060	1.947285	2.320965	-0.312589	-0.931027	...	1.901185	0.117700	1.7
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.017833	0.693043	1.263669	-0.217664	-1.058611	...	1.536720	2.047399	1.4
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.038680	0.046588	0.105777	-0.809117	-0.895587	...	0.561361	1.374854	0.5
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.272144	3.296944	2.658866	2.137194	1.043695	...	1.961239	2.237926	2.3
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.150752	-1.114873	-1.261820	-0.820070	-0.561032	...	-1.410893	0.764190	-1.4

5 rows × 30 columns

```
normalised_df
```

	feature0	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	feature9	...	feature20	feature21	fea
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.217515	2.255747	...	1.886690	-1.359293	2.
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.001392	-0.868652	...	1.805927	-0.369203	1.
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	0.939685	-0.398008	...	1.511870	-0.023974	1.
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.867383	4.910919	...	-0.281464	0.133984	-0.
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.009560	-0.562450	...	1.298575	-1.466770	1.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.219060	1.947285	2.320965	-0.312589	-0.931027	...	1.901185	0.117700	1.
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.017833	0.693043	1.263669	-0.217664	-1.058611	...	1.536720	2.047399	1.
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.038680	0.046588	0.105777	-0.809117	-0.895587	...	0.561361	1.374854	0.
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.272144	3.296944	2.658866	2.137194	1.043695	...	1.961239	2.237926	2.
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.150752	-1.114873	-1.261820	-0.820070	-0.561032	...	-1.410893	0.764190	-1.

569 rows × 30 columns

```
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca.fit(scaled_dt)
```

```
PCA
PCA(n_components=2)
```

```
X_pca=pca.transform(scaled_dt)
```

```
scaled_dt.shape
```

```
(569, 30)
```

```
X_pca.shape
```

```
(569, 2)
```

```
pca_dt=pd.DataFrame(data=X_pca,columns=['prinicipal component 1','principal component 2'])
pca_dt
```

	prinicipal component 1	principal component 2
0	9.192837	1.948583
1	2.387802	-3.768172
2	5.733896	-1.075174
3	7.122953	10.275589
4	3.935302	-1.948072
...	...	...
564	6.439315	-3.576817
565	3.793382	-3.584048
566	1.256179	-1.902297
567	10.374794	1.672010
568	-5.475243	-0.670637

569 rows × 2 columns

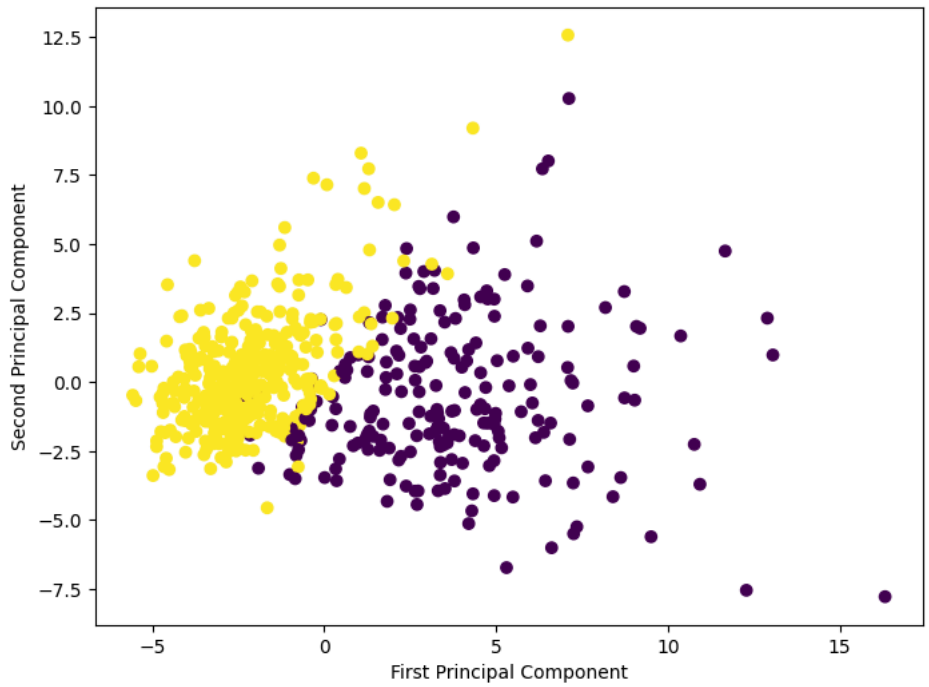
```
X_pca
```

```
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]])
```

```
'''
It will provided us with the amount of information or variance of each
principal component holds after projecting the data to a lower dimensional subspace.
'''
pca.explained_variance_ratio_
```

```
array([0.44272026, 0.18971182])
```

```
plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0],X_pca[:,1],c=cancer['target'])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()
```




```
X_pca
```

```
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]])
```

```
#Make_Classification
#To  evaluate pca with Logistic Regression algorithm for Classification
```

```
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

```
#define dataset
X,y=make_classification(n_samples=1000,n_features=20,n_informative=10,n_redundant=10,random_state=7)
```

```
len(X)
```

```
1000
```

```
X[1]
```

```
array([-2.3302999 , -4.86608574, -3.88291317, -2.23248277,  1.44515322,
        2.59739106,  3.68926886, -1.65118858, -2.47865974, -1.71944863,
        1.73993388, -3.88589606, -1.44039921,  3.12869825, -5.37048832,
        3.88186485,  0.75984387, -0.14561581, -0.55489384,  0.61420772])
```

```
#define the pipeline
steps=[('pca',PCA(n_components=10)),('lrmodel',LogisticRegression())]
model=Pipeline(steps=steps)
```

```
#evaluate model
cv=StratifiedKFold(n_splits=10)
n_scores=cross_val_score(model,X,y,scoring='accuracy',cv=cv)
```

```
n_scores
```

```
array([0.84, 0.75, 0.86, 0.83, 0.85, 0.83, 0.86, 0.83, 0.8 , 0.83])
```

```
#report performance
print('Accuracy: %.3f' % (np.mean(n_scores)))
```

```
Accuracy: 0.828
```

```
#ML Class 18
#Apriori Algorithm
```

```
pip install apyori
```

```
Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl size=5954 sha256=1859c99a3ad0fd3f49be2e29bfabbb169a5015478cfb99e7e7d9477b1d71943
  Stored in directory: /root/.cache/pip/wheels/c4/1a/79/20f55c470a50bb3702a8cb7c94d8ada15573538c7f4baebe2d
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from apyori import apriori
```

```
store_data=pd.read_csv('store_data.csv',header=None)
store_data
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad	mineral water	salmon
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

7501 rows × 20 columns

```
records=[]
for i in range(0,7501):
  records.append([str(store_data.values[i,j]) for j in range(0,20)])
```

```
records
```

```
[['shrimp',
  'almonds',
```

```
 'avocado',
 'vegetables mix',
 'green grapes',
 'whole weat flour',
 'yams',
 'cottage cheese',
 'energy drink',
 'tomato juice',
 'low fat yogurt',
 'green tea',
 'honey',
 'salad',
 'mineral water',
 'salmon',
 'antioxydant juice',
 'frozen smoothie',
 'spinach',
 'olive oil'],
['burgers',
 'meatballs',
 'eggs',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan'],
['chutney',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
 'nan',
```

```
association_rules=apriori(records,min_support=0.0045,min_confidence=0.2,min_lift=3,min_lenght=2)
association_results=list(association_rules)
```

```
len(association_results)
```

```
48
```

```
print(association_results[0])
```

```
RelationRecord(items=frozenset({'light cream', 'chicken'}), support=0.004532728969470737, ordered_statistics=[OrderedStatistic(items_base=frozenset({'light cre
```

```
for item in association_results:
  #First index of inner list
  #Contains base item & add item
  pair=item[0]
  items=[x for x in pair]
  print('Rule: '+items[0]+'->'+items[1])

  #Second index of the inner list
  print('Support: '+str(item[1]))

  #Third index of the list located at 0th
  #Of the third index of the inner list
  print('Confidence: '+str(item[2][0][2]))
  print('Lift: '+str(item[2][0][3]))
  print('==============================')
```

```
Rule: light cream->chicken
Support: 0.004532728969470737
Confidence: 0.29059829059829057
Lift: 4.84395061728395
============================
Rule: escalope->mushroom cream sauce
Support: 0.005732568990801226
Confidence: 0.3006993006993007
Lift: 3.790832696715049
============================
Rule: escalope->pasta
Support: 0.005865884548726837
Confidence: 0.3728813559322034
Lift: 4.700811850163794
============================
Rule: ground beef->herb & pepper
Support: 0.015997866951073192
Confidence: 0.3234501347708895
Lift: 3.2919938411349285
============================
Rule: tomato sauce->ground beef
Support: 0.005332622317024397
Confidence: 0.3773584905660377
Lift: 3.840659481324083
============================
Rule: olive oil->whole wheat pasta
Support: 0.007998933475536596
Confidence: 0.2714932126696833
Lift: 4.122410097642296
============================
Rule: shrimp->pasta
Support: 0.005065991201173177
Confidence: 0.3220338983050847
Lift: 4.506672147735896
============================
Rule: nan->light cream
Support: 0.004532728969470737
Confidence: 0.29059829059829057
Lift: 4.84395061728395
============================
Rule: frozen vegetables->shrimp
Support: 0.005332622317024397
Confidence: 0.23255813953488375
Lift: 3.2545123221103784
============================
Rule: spaghetti->ground beef
Support: 0.004799360085321957
Confidence: 0.5714285714285714
Lift: 3.2819951870487856
============================
Rule: escalope->mushroom cream sauce
Support: 0.005732568990801226
Confidence: 0.3006993006993007
Lift: 3.790832696715049
============================
Rule: pasta->escalope
Support: 0.005865884548726837
Confidence: 0.3728813559322034
```

#FrequentPatternGrowth Algorithm

```
pip install pyfpgrowth
```

```
Collecting pyfpgrowth
  Downloading pyfpgrowth-1.0.tar.gz (1.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 16.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: pyfpgrowth
  Building wheel for pyfpgrowth (setup.py) ... done
  Created wheel for pyfpgrowth: filename=pyfpgrowth-1.0-py2.py3-none-any.whl size=5489 sha256=a6da0980fb08c1a97ea927db36ab9e1587ed895ff4393a73c3303
  Stored in directory: /root/.cache/pip/wheels/09/fc/dc/afff211038bfc745722d8d7e846e854e5791968b22c570a530
Successfully built pyfpgrowth
Installing collected packages: pyfpgrowth
Successfully installed pyfpgrowth-1.0
```

```
import pyfpgrowth
transactions=[
['milk','bread','jam','juice','cream'],
['bread','jam'],
['bread','juice','butter'],
['bread','cream','chicken']
]
```

#To find frequent patterns

```
frequent_patterns=pyfpgrowth.find_frequent_patterns(transactions=transactions,support_threshold=2)
frequent_patterns
```

```
{('jam',): 2,
 ('bread', 'jam'): 2,
 ('juice',): 2,
 ('bread', 'juice'): 2,
 ('cream',): 2,
 ('bread', 'cream'): 2,
 ('bread',): 4}
```

```
rules=pyfpgrowth.generate_association_rules(patterns=frequent_patterns,confidence_threshold=0.5)
rules
```

```
{('bread',): (('cream',), 0.5),
 ('jam',): (('bread',), 1.0),
 ('juice',): (('bread',), 1.0),
 ('cream',): (('bread',), 1.0)}
```

```
#ECLAT Algorithm
```

```
!pip install pyECLAT

import pyECLAT

transactions = [
    ['bear','wine','cheese'],
    ['beer','potato chips'],
    ['eggs','flour','butter','cheese'],
    ['eggs','flour','butter','beer','cheese'],
    ['wine','cheese'],
    ['potato chips'],
    ['eggs','flour','butter','wine','cheese'],
    ['eggs','flour','butter','beer','potato chips'],
    ['wine','beer'],
    ['wine','potato chips'],
    ['flour','eggs'],
    ['beer','potato chips'],
    ['eggs','flour','butter','butter','wine','cheese'],
    ['beer','wine','potato chips','cheese'],
    ['wine','cheese'],
    ['wine','potato chips']
]

transactions
[['bear', 'wine', 'cheese'],
 ['beer', 'potato chips'],
 ['eggs', 'flour', 'butter', 'cheese'],
 ['eggs', 'flour', 'butter', 'beer', 'cheese'],
 ['wine', 'cheese'],
 ['potato chips'],
 ['eggs', 'flour', 'butter', 'wine', 'cheese'],
 ['eggs', 'flour', 'butter', 'beer', 'potato chips'],
 ['wine', 'beer'],
 ['wine', 'potato chips'],
 ['flour', 'eggs'],
 ['beer', 'potato chips'],
 ['eggs', 'flour', 'butter', 'butter', 'wine', 'cheese'],
 ['beer', 'wine', 'potato chips', 'cheese'],
 ['wine', 'cheese'],
 ['wine', 'potato chips']]

import pandas as pd
dt= pd.DataFrame(transactions)
print(dt)

#We are looking for itemSETS
#We don't want to have any individual products returned
min_n_products = 2

#We want to set win support to 7
#but we have to express it as a percentage
min_support = 7/len(transactions)

#We have no limit on the size of association rules
#so we set it to the longest transaction
max_length = max([len(x) for x in transactions])

print(min_n_products)
print(min_support)
print(max_length)

from pyECLAT import ECLAT
#Create an instance of eclat
my_eclat=ECLAT(data=dt,verbose=True)

#Fit the algorithm
rule_indices,rule_supports = my_eclat.fit(min_support=min_support,
                                  min_combination=min_n_products,
                                  max_combination=max_length)


print(rule_supports)
{}
print(rule_indices)
{}

#Creating Rules using Eclat
Rules=eclat.eclat(data=transactions,min_support=0.5)
Rules
```

```
Requirement already satisfied: pyECLAT in /usr/local/lib/python3.10/dist-packages (1.0.2)
Requirement already satisfied: pandas>=0.25.3 in /usr/local/lib/python3.10/dist-packages (from pyECLAT) (1.5.3)
Requirement already satisfied: numpy>=1.17.4 in /usr/local/lib/python3.10/dist-packages (from pyECLAT) (1.25.2)
Requirement already satisfied: tqdm>=4.41.1 in /usr/local/lib/python3.10/dist-packages (from pyECLAT) (4.66.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.25.3->pyECLAT) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.25.3->pyECLAT) (2023.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas>=0.25.3->pyECLAT) (1.16.0)
           0          1          2      3          4      5
0         bear       wine     cheese   None       None   None
1         beer  potato chips       None   None       None   None
2         eggs       flour     butter  cheese       None   None
3         eggs       flour     butter    beer     cheese   None
4         wine     cheese       None   None       None   None
5   potato chips       None       None   None       None   None
6         eggs       flour     butter    wine     cheese   None
7         eggs       flour     butter    beer  potato chips   None
8         wine        beer       None   None       None   None
9         wine  potato chips       None   None       None   None
10       flour        eggs       None   None       None   None
11        beer  potato chips       None   None       None   None
12        eggs       flour     butter  butter       wine  cheese
13        beer        wine  potato chips  cheese       None   None
14        wine     cheese       None   None       None   None
15        wine  potato chips       None   None       None   None
2
0.4375
6
100%|██████████████████| 9/9 [00:00<00:00, 271.15it/s]
100%|██████████████████| 9/9 [00:00<00:00, 59167.30it/s]
100%|██████████████████| 9/9 [00:00<00:00, 746.05it/s]
Combination 2 by 2
3it [00:00, 130.65it/s]
Combination 3 by 3
1it [00:00, 141.14it/s]
Combination 4 by 4
0it [00:00, ?it/s]
Combination 5 by 5
0it [00:00, ?it/s]
Combination 6 by 6
0it [00:00, ?it/s]{}
{}

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-25-3e0525bf6787> in <cell line: 78>()
     76
     77 #Creating Rules using Eclat
---> 78 Rules=eclat.eclat(data=transactions,min_support=0.5)
     79 Rules

NameError: name 'eclat' is not defined
```

```
#ML Class 19
#Reinforcment Learning
```

```
import numpy as np
```

```
#Reward Matrix
R=np.matrix([[-1,-1,-1,-1,0,-1],
          [-1,-1,-1,0,-1,100],
          [-1,-1,-1,0,-1,-1],
          [-1,0,0,-1,0,-1],
          [-1,0,0,-1,-1,100],
          [-1,0,-1,-1,0,100]])

#Q Matrix
Q=np.matrix(np.zeros([6,6]))
R
```

```
matrix([[ -1,  -1,  -1,  -1,   0,  -1],
        [ -1,  -1,  -1,   0,  -1, 100],
        [ -1,  -1,  -1,   0,  -1,  -1],
        [ -1,   0,   0,  -1,   0,  -1],
        [ -1,   0,   0,  -1,  -1, 100],
        [ -1,   0,  -1,  -1,   0, 100]])
```

```
Q
```

```
matrix([[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.]])
```

```
gamma=0.8
initial_state=1
```

```
def available_actions(state):
  current_state_row=R[state,]
  av_act=np.where(current_state_row>=0)[1]
  return av_act
```

```
current_state_row=R[1,]
current_state_row
```

```
matrix([[ -1,  -1,  -1,   0,  -1, 100]])
```

```
av_act=np.where(R[1,]>=0)[1]
av_act
```

```
array([3, 5])
```

```
available_act=available_actions(initial_state)
available_act
```

```
array([3, 5])
```

```
#available_act=available_actions(inital_state)
def sample_next_action(available_actions_range):
  next_action=int(np.random.choice(available_act,1))
  return next_action
```

```
next_action=int(np.random.choice(available_act,1))
next_action
```

```
<ipython-input-12-d89ac7327d39>:1: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you ex
  next_action=int(np.random.choice(available_act,1))
3
```

```
action=sample_next_action(available_act)
```

```
<ipython-input-11-68a9cfb4bbcf>:3: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you ext
  next_action=int(np.random.choice(available_act,1))
```

```
def update(current_state,action,gamma):
  max_index=np.where(Q[action,]==np.max(Q[action,]))[1]
  if max_index.shape[0]>1:
    max_index=int(np.random.choice(max_index,size=1))
  else:
    max_index=int(max_index)
    max_value=Q[action,max_index]
    Q[current_state,action]=R[currnet_state,action]+gamma*max_value

update(initial_state,action,gamma)
```

```
<ipython-input-14-ce6fe7fdf851>:4: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you ext
  max_index=int(np.random.choice(max_index,size=1))
```

```
max_index=np.where(Q[5,]==np.max(Q[5,]))[1]
max_index
```

```
array([0, 1, 2, 3, 4, 5])
```

```
max_index.shape[0]
```

```
6
```

```
Q/np.max(Q)*100
```

```
<ipython-input-18-e7d5bf010b61>:1: RuntimeWarning: invalid value encountered in divide
  Q/np.max(Q)*100
matrix([[nan, nan, nan, nan, nan, nan],
        [nan, nan, nan, nan, nan, nan],
        [nan, nan, nan, nan, nan, nan],
        [nan, nan, nan, nan, nan, nan],
```

```
       [nan, nan, nan, nan, nan, nan],
       [nan, nan, nan, nan, nan, nan]])
```

```
for i in range(10000):
  current_state=np.random.randint(0,int(Q.shape[0]))
  available_act=available_actions(current_state)
  action=sample_next_action(available_act)
  update(current_state,action,gamma)

print('Trained Q matrix:')
print(Q/np.max(Q)*100)
```

```
<ipython-input-11-68a9cfb4bbcf>:3: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you ext
  next_action=int(np.random.choice(available_act,1))
<ipython-input-14-ce6fe7fdf851>:4: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you ext
  max_index=int(np.random.choice(max_index,size=1))
Trained Q matrix:
[[nan nan nan nan nan nan]
 [nan nan nan nan nan nan]
 [nan nan nan nan nan nan]
 [nan nan nan nan nan nan]
 [nan nan nan nan nan nan]
 [nan nan nan nan nan nan]]
<ipython-input-19-ef2397627fe5>:8: RuntimeWarning: invalid value encountered in divide
  print(Q/np.max(Q)*100)
```

```
current_state=1
steps=[current_state]

while current_state!=5:
  next_step_index=np.where(Q[current_state,]==np.max(Q[current_state,]))[1]

  if next_step_index.shape[0]>1:
    next_step_index=int(np.random.choice(next_step_index,size=1))
  else:
    next_step_index=int(next_step_index)

  steps.append(next_step_index)
  current_state=next_step_index

print('Selected Path: ')
print(steps)
```

```
Selected Path:
[1, 3, 5]
<ipython-input-27-23b33a341e12>:8: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you ex
  next_step_index=int(np.random.choice(next_step_index,size=1))
```

```
Q
```

```
matrix([[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.]])
```

```
#ML Class 20
#Wine Quality Prediction
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

```
#Loading the Datasets
```

```
red_wine=pd.read_csv('winequality-red.csv',delimiter=';')
red_wine
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

```
white_wine=pd.read_csv('winequality-white.csv',delimiter=';')
white_wine
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6

4898 rows × 12 columns

```
#Exploratory Data Analysis
```

```
red_wine['quality'].unique()
```

```
array([5, 6, 7, 4, 8, 3])
```

```
white_wine['quality'].unique()
```

```
array([6, 5, 7, 8, 4, 3, 9])
```

```
red_wine.isna().sum()
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
```