```
# Find the correlation matrix.

import numpy as np
# x= sales
x=[215,325,185,332,406,522]

# y= temperature
y=[14.2 , 16.4 , 11.9 , 15.2 , 18.5 , 22.1 ]

matrix=np.corrcoef(x,y)

print(matrix)
```

```
[[1.         0.97665315]
 [0.97665315 1.        ]]
```

```
# x=age
x=[43,21,25,42,57,59]
# y= glucose level
y=[99,65,79,75,87,81]

matrix=np.corrcoef(x,y)

print(matrix)
```

```
[[1.        0.5298089]
 [0.5298089 1.       ]]
```

```
import pandas as pd
data={
      'x':[45,37,42,35,39],
      'y':[38,31,26,28,33],
      'z':[10,15,17,21,12]
}

dataframe=pd.DataFrame(data,columns=['x','y','z'])
print("Data Frame is: ")
print(dataframe)

matrix=dataframe.corr()
print(matrix)
```

```
Data Frame is:
    x   y   z
0  45  38  10
1  37  31  15
2  42  26  17
```

```
3  35  28  21
4  39  33  12
          x         y         z
x  1.000000  0.518457 -0.701886
y  0.518457  1.000000 -0.860941
z -0.701886 -0.860941  1.000000
```

```
import pandas as pd

dataframe=pd.read_csv("/content/drive/MyDrive/KRAI/corr - Sheet1.csv")
print("Data Frame is: ")
print(dataframe)

matrix=dataframe.corr()
print(matrix)
```

```
Data Frame is: 
         date  AVG Temp C   Ice Creamproduction
0    1/1/2011         1.2                 55942
1    2/1/2011         1.8                 61802
2    3/1/2011         6.1                 74616
3    4/1/2011        11.1                 74088
4    5/1/2011        16.0                 74980
5    6/1/2011        20.4                 75131
6    7/1/2011        22.9                 71229
7    8/1/2011        22.2                 77396
8    9/1/2011        18.4                 69286
9   10/1/2011        12.6                 59559
10  11/1/2011         6.3                 52314
11  12/1/2011         1.4                 50894
                    AVG Temp C   Ice Creamproduction
AVG Temp C             1.000000             0.718032
Ice Creamproduction    0.718032             1.000000
<ipython-input-3-93db14ef3f25>:7: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only
  matrix=dataframe.corr()
```

```
# Plot the correlation plot on dataset and visualize giving an
# overview of relationships among data on iris data.

import pandas as pd
iris=pd.read_csv("/content/drive/MyDrive/KRAI/iris.csv")
print(iris)
print(iris.head())
print(iris.tail())
print(iris.dtypes)
```

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

[150 rows x 5 columns]
   sepal.length  sepal.width  petal.length  petal.width variety
0           5.1          3.5           1.4          0.2  Setosa
1           4.9          3.0           1.4          0.2  Setosa
2           4.7          3.2           1.3          0.2  Setosa
3           4.6          3.1           1.5          0.2  Setosa
4           5.0          3.6           1.4          0.2  Setosa
     sepal.length  sepal.width  petal.length  petal.width    variety
145           6.7          3.0           5.2          2.3  Virginica
146           6.3          2.5           5.0          1.9  Virginica
147           6.5          3.0           5.2          2.0  Virginica
148           6.2          3.4           5.4          2.3  Virginica
149           5.9          3.0           5.1          1.8  Virginica
sepal.length    float64
sepal.width     float64
petal.length    float64
petal.width     float64
variety          object
dtype: object
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import metrics
sns.set()
```

```
iris_data=pd.read_csv("/content/drive/MyDrive/KRAI/iris.csv")
print(iris_data)
```

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

[150 rows x 5 columns]
```

```
iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal.length  150 non-null    float64
 1   sepal.width   150 non-null    float64
 2   petal.length  150 non-null    float64
 3   petal.width   150 non-null    float64
 4   variety       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
iris_data.describe()
```

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

```
iris_data[iris_data.duplicated()]
```

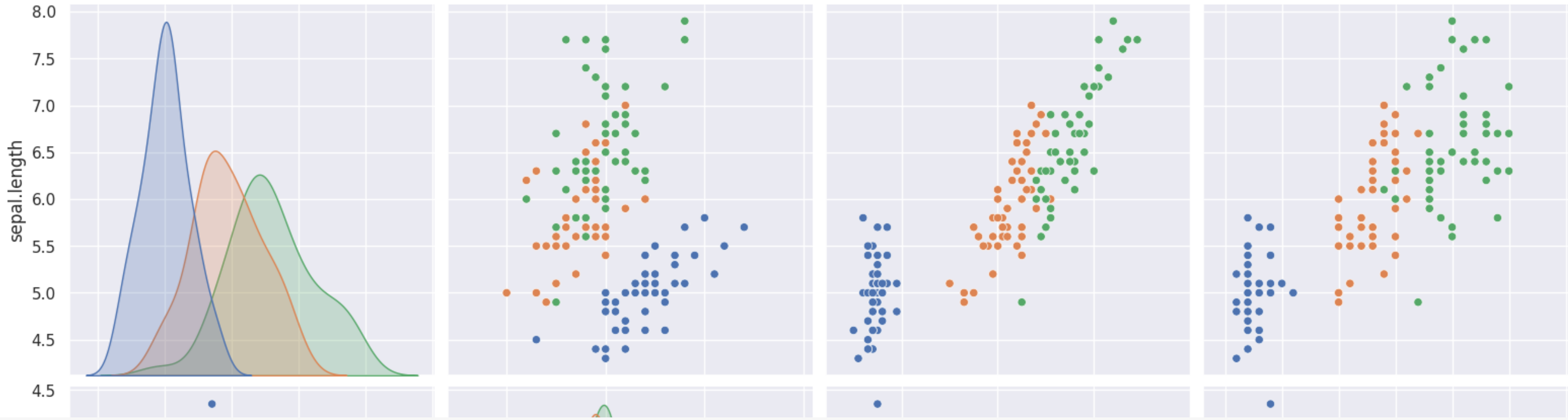
	sepal.length	sepal.width	petal.length	petal.width	variety
142	5.8	2.7	5.1	1.9	Virginica

```
iris_data['variety'].value_counts()
```

```
Setosa        50
Versicolor    50
Virginica     50
Name: variety, dtype: int64
```

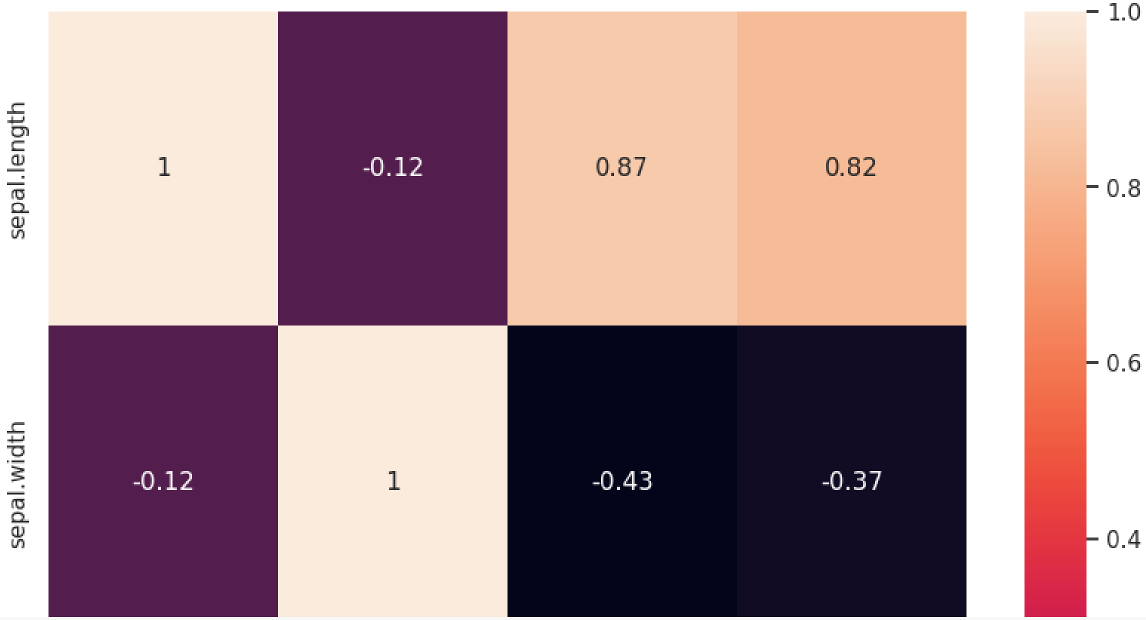
```
sns.pairplot(iris_data,hue='variety',height=4)
```

<seaborn.axisgrid.PairGrid at 0x7940b6d33370>



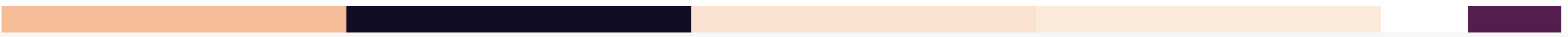

```
plt.figure(figsize=(10,11))
sns.heatmap(iris_data.corr(),annot=True)
plt.plot()
```

```
<ipython-input-14-0a05fdd33f33>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only
  sns.heatmap(iris_data.corr(),annot=True)
[]
```

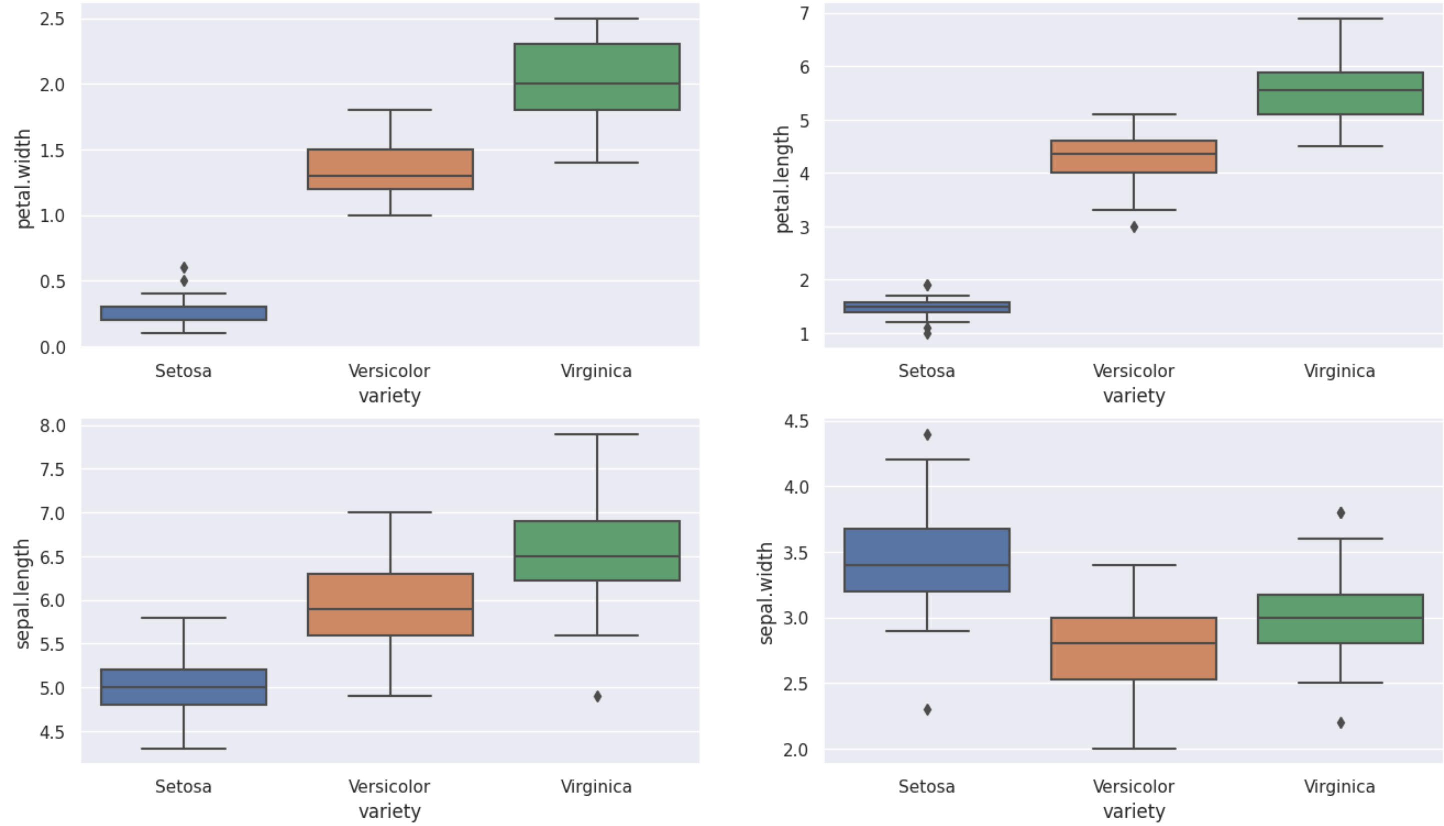



```
iris_data.groupby('variety').agg(['mean','median'])
```

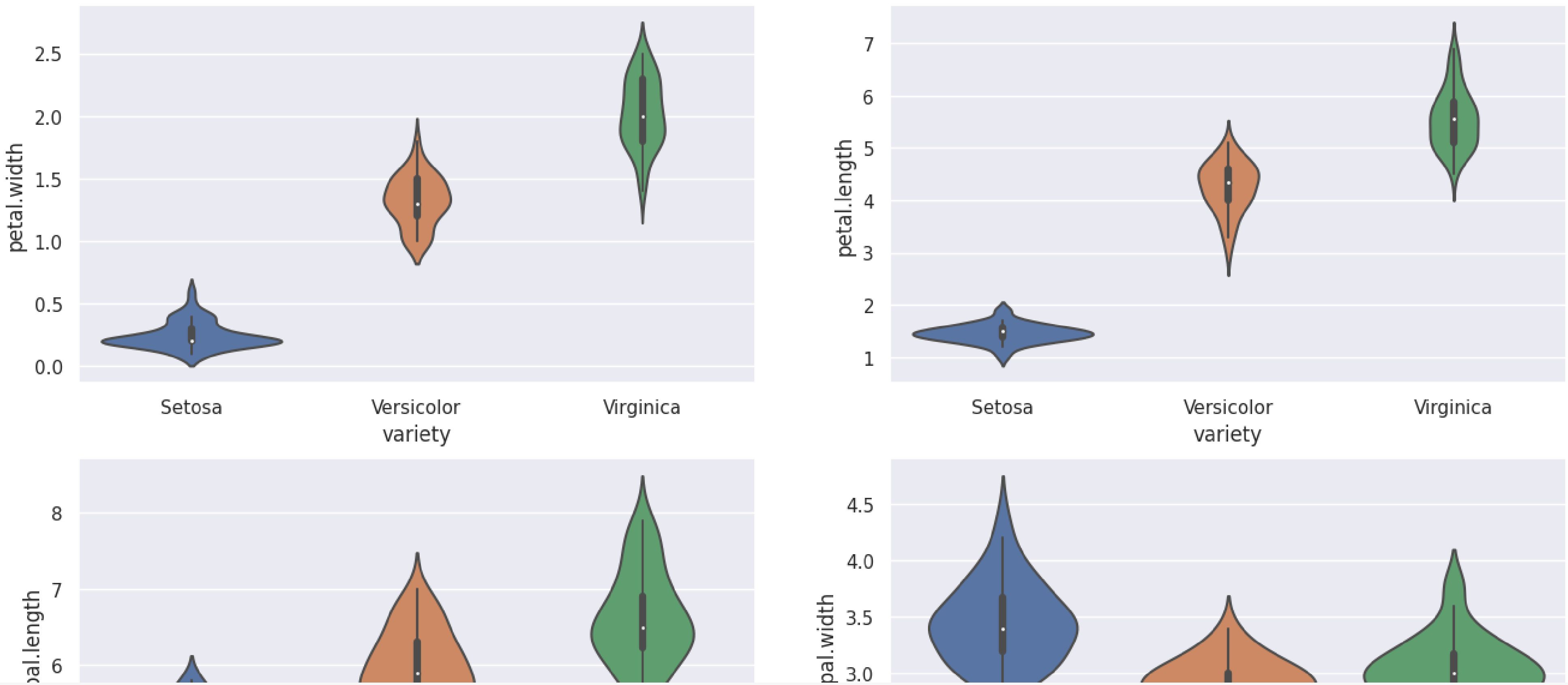
	sepal.length		sepal.width		petal.length		petal.width	
	mean	median	mean	median	mean	median	mean	median
variety								
Setosa	5.006	5.0	3.428	3.4	1.462	1.50	0.246	0.2
Versicolor	5.936	5.9	2.770	2.8	4.260	4.35	1.326	1.3
Virginica	6.588	6.5	2.974	3.0	5.552	5.55	2.026	2.0



```
fig,axes=plt.subplots(2,2,figsize=(16,9))
sns.boxplot(y= 'petal.width' , x='variety' , data=iris_data , orient='v' , ax=axes[0,0])
sns.boxplot(y='petal.length',x='variety',data=iris_data,orient='v',ax=axes[0,1])
sns.boxplot(y='sepal.length',x='variety',data=iris_data,orient='v',ax=axes[1,0])
sns.boxplot(y='sepal.width',x='variety',data=iris_data,orient='v',ax=axes[1,1])
plt.show()
```




```
fig,axes=plt.subplots(2,2,figsize=(16,9))
sns.violinplot(y= 'petal.width', x='variety' , data=iris_data , orient='v' , ax=axes[0,0])
sns.violinplot(y='petal.length', x='variety' , data=iris_data , orient='v' , ax=axes[0,1])
sns.violinplot(y='sepal.length', x='variety' , data=iris_data , orient='v' , ax=axes[1,0])
sns.violinplot(y='sepal.width' , x='variety' , data=iris_data , orient='v' , ax=axes[1,1])
plt.show()
```




```
jupyter nbconvert --to pdf 2_prac_iris.ipynb
```

```
  File "<ipython-input-15-c903ee054443>", line 1
    jupyter nbconvert --to pdf 2_prac_iris.ipynb
                               ^
SyntaxError: invalid decimal literal
```

SEARCH STACK OVERFLOW

```
# 3 Analysis of covariance: variance (ANOVA), if data have
categorical variables on iris data.
import pandas as pd
from sklearn.datasets import load_iris

iris=load_iris()
df=pd.DataFrame(data=iris.data)
df
```

	0	1	2	3
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

```
import scipy.stats as stats
stats.f_oneway(df.iloc[:,0],df.iloc[:,1],df.iloc[:,2],df.iloc[:,3],)
```

```
F_onewayResult(statistic=482.91531656927964, pvalue=4.660592480454751e-159)
```

```
iris2=pd.read_excel('/content/drive/MyDrive/KRAI/iris_csv.xlsx')
iris2
```

	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
df=pd.read_excel('/content/drive/MyDrive/KRAI/iris_csv.xlsx')
df
```

	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

```
ano=ols('sepallength~sepalwidth',data=df).fit()
df
```

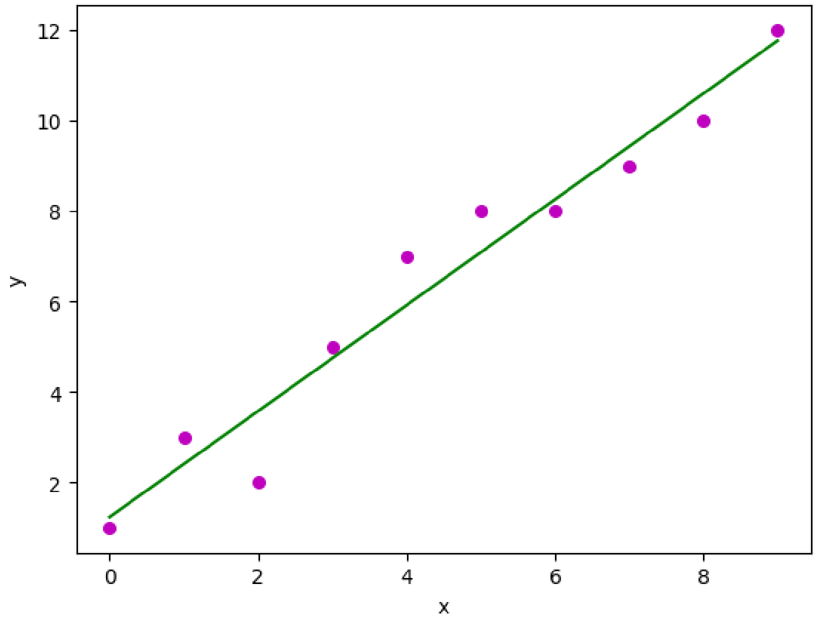
	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica

```
one=sm.stats.anova_lm(ano,type=2)
one
```

	df	sum_sq	mean_sq	F	PR(>F)
sepalwidth	1.0	1.222100	1.222100	1.791754	0.182765
Residual	148.0	100.946233	0.682069	NaN	NaN

```
#4  Apply linear regression Model techniques to predict the data
# on any dataset.

import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x,y):
  n = np.size(x)

  m_x = np.mean(x)
  m_y = np.mean(y)

  SS_xy = np.sum(y*x) - n*m_y*m_x
  SS_xx = np.sum(x*x) - n*m_x*m_x

  b_1 = SS_xy / SS_xx
  b_0 = m_y - b_1*m_x

  return(b_0,b_1)

def plot_regression_line(x,y,b):
    plt.scatter(x,y, color = "m", marker="o" , s=30)

    y_pred = b[0] + b[1]*x

    plt.plot(x,y_pred,color ="g")

    plt.xlabel('x')
    plt.ylabel('y')

    plt.show()

def main():
      x= np.array([0,1,2,3,4,5,6,7,8,9])
      y = np.array([1,3,2,5,7,8,8,9,10,12])

      b = estimate_coef(x,y)
      print("Estimated coefficient:\nb_0 = {} \nb_1 = {}".format(b[0],b[1]))

      plot_regression_line(x,y,b)

main()
```

```
Estimated coefficient:
b_0 = 1.2363636363636363
b_1 = 1.1696969696969697
```



```
# 5 Apply logical regression Model techniques to predict the
# data on any dataset.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dataset=pd.read_csv('/content/drive/MyDrive/KRAI/User_data.csv')
dataset
```

	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=dataset.iloc[:,[2,3]].values

y=dataset.iloc[:,4].values
print(x)
print(y)
```

```
 [    59  29000]
 [    58  47000]
 [    46  88000]
 [    38  71000]
 [    54  26000]
 [    60  46000]
 [    60  83000]
 [    39  73000]
 [    59 130000]
 [    37  80000]
 [    46  32000]
 [    46  74000]
 [    42  53000]
 [    41  87000]
 [    58  23000]
 [    42  64000]
 [    48  33000]
 [    44 139000]
 [    49  28000]
 [    57  33000]
 [    56  60000]
 [    49  39000]
 [    39  71000]
 [    47  34000]
 [    48  35000]
 [    48  33000]
 [    47  23000]
 [    45  45000]
 [    60  42000]
 [    39  59000]
 [    46  41000]
 [    51  23000]
 [    50  20000]
 [    36  33000]
 [    49  36000]]
[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 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 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 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 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=0)
```

```
from sklearn.preprocessing import StandardScaler
sc_x=StandardScaler()
X_train=sc_x.fit_transform(X_train)
X_test=sc_x.transform(X_test)
print(X_train[0:10,:])
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]]
```

```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression(random_state=0)
classifier.fit(X_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression(random_state=0)
```

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

```
# import the metrics class
from sklearn.metrics import confusion_matrix

cnf=confusion_matrix(y_test, y_pred)
cnf
```

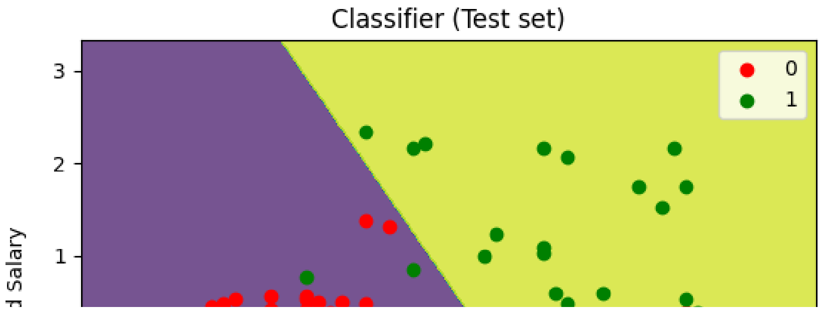
```
array([[65,  3],
       [ 8, 24]])
```

```
from sklearn.metrics import accuracy_score
print("Accuracy:",accuracy_score(y_test,y_pred))
```

```
Accuracy: 0.89
```

```
from matplotlib.colors import ListedColormap

X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                            stop = X_set[:, 0].max() + 1, step = 0.01),
                    np.arange(start = X_set[:, 1].min() - 1,
                            stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(
            np.array([X1.ravel(), X2.ravel()]).T).reshape(
            X1.shape), alpha = 0.75, cmsp = ListedColormap(('green','red')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == i, 1],
                c = ListedColormap(('red','green'))(i), label = j)

plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
<ipython-input-19-1f549b236efd>:9: UserWarning: The following kwargs were not used by contour: 'cmsp'
  plt.contourf(X1, X2, classifier.predict(
<ipython-input-19-1f549b236efd>:17: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence i
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == i, 1],
```



```
# 6A Clustering algorithms for unsupervised classification.
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
df=pd.read_csv("/content/drive/MyDrive/KRAI/Mall_Customers_dataset.csv")
df.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

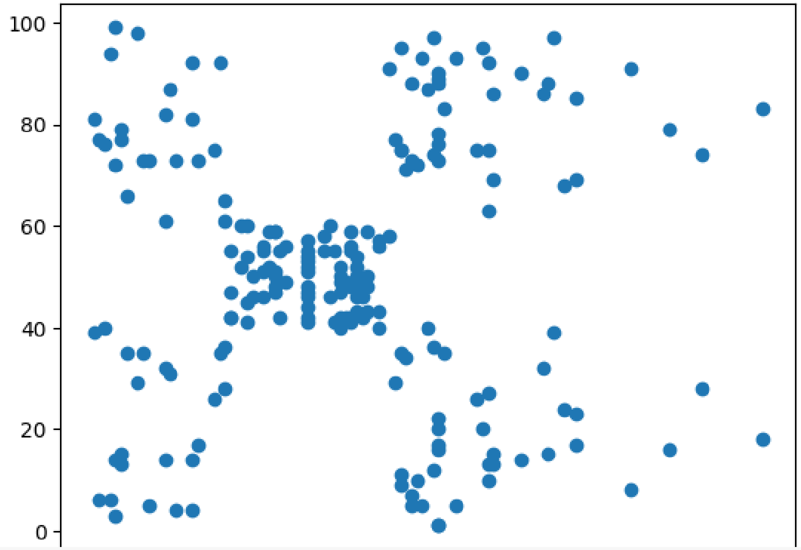
```
X=df[['Annual Income (k$)','Spending Score (1-100)']]
X
```

	Annual Income (k$)	Spending Score (1-100)
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
...	...	...
195	120	79
196	126	28
197	126	74
198	137	18
199	137	83

200 rows × 2 columns

```
plt.scatter(X['Annual Income (k$)'],X['Spending Score (1-100)'])
```

<matplotlib.collections.PathCollection at 0x7d84d5a3d930>




```
from sklearn.cluster import KMeans
model= KMeans(n_clusters=5)
model.fit(X)
```

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

▾ KMeans

KMeans(n_clusters=5)

```
model.cluster_centers_
```

```
array([[88.2       , 17.11428571],
       [55.2962963 , 49.51851852],
       [25.72727273, 79.36363636],
       [86.53846154, 82.12820513],
       [26.30434783, 20.91304348]])
```

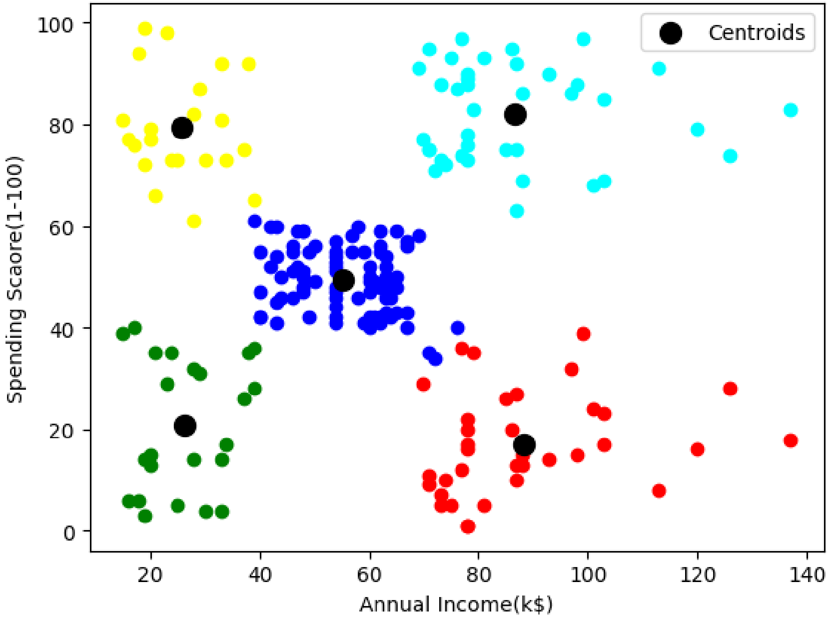
```
cluster_number=model.predict(X)
```

```
len(cluster_number)
```

200

```
c0=X[cluster_number==0]
c1=X[cluster_number==1]
c2=X[cluster_number==2]
c3=X[cluster_number==3]
c4=X[cluster_number==4]
```

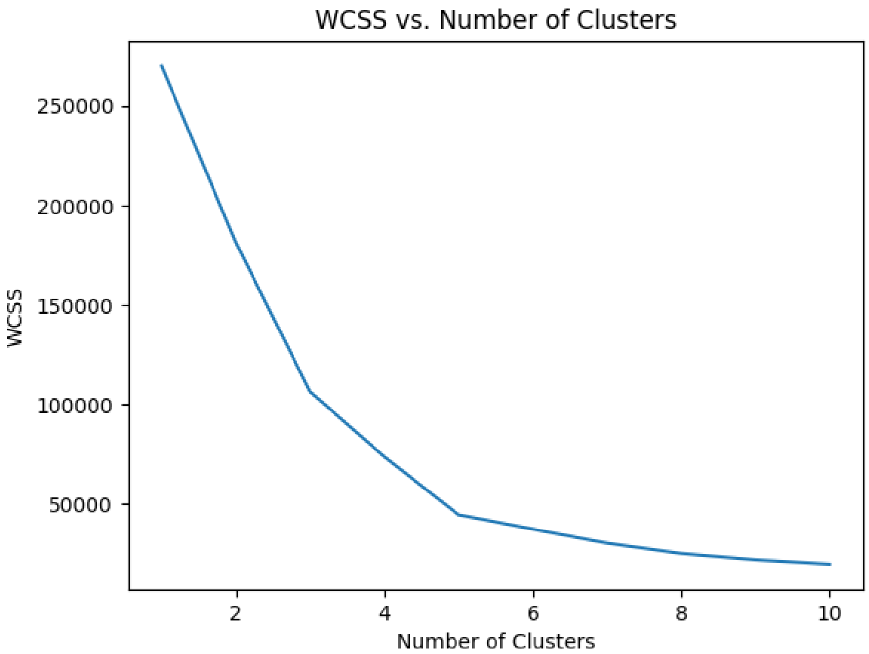
```
plt.scatter(c0['Annual Income (k$)'],c0['Spending Score (1-100)'],c='red')
plt.scatter(c1['Annual Income (k$)'],c1['Spending Score (1-100)'],c='blue')
plt.scatter(c2['Annual Income (k$)'],c2['Spending Score (1-100)'],c='yellow')
plt.scatter(c3['Annual Income (k$)'],c3['Spending Score (1-100)'],c='cyan')
plt.scatter(c4['Annual Income (k$)'],c4['Spending Score (1-100)'],c='green')
plt.scatter(model.cluster_centers_[:,0],model.cluster_centers_[:,1],s=100,c='black',label='Centroids')
plt.xlabel('Annual Income(k$)')
plt.ylabel('Spending Scaore(1-100)')
plt.legend()
plt.show()
```




```
model.inertia_
```

44448.4554479337

```
WCSS = []
for i in range(1, 11):
  model = KMeans(n_clusters=i, n_init=10)  # Explicitly set n_init to suppress warning
  model.fit(X)
  WCSS.append(model.inertia_)

plt.plot(range(1, 11), WCSS)
plt.title('WCSS vs. Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```

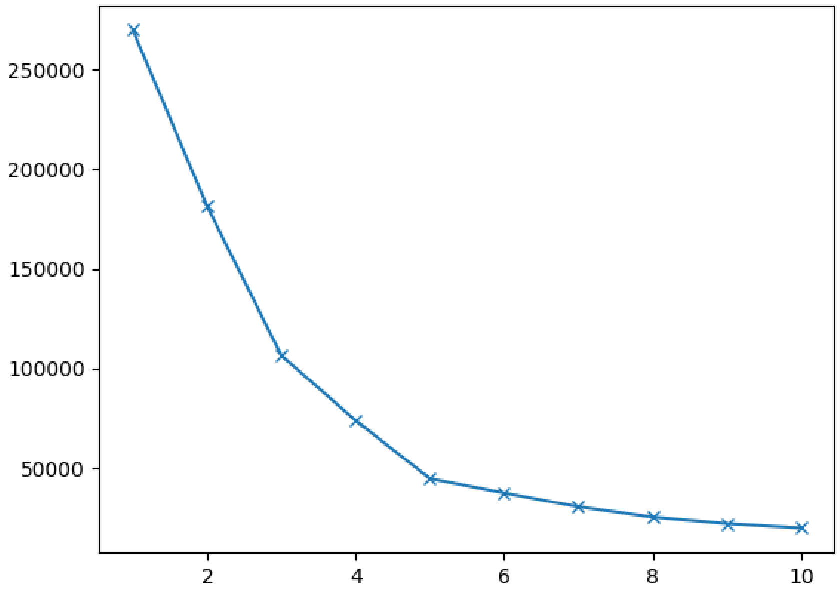



```
WCSS
```

```
[269981.28,
 181363.59595959593,
 106348.37306211122,
 73679.78903948836,
 44448.4554479337,
 37233.814510710006,
 30273.394312070042,
 25043.89004329005,
 21838.86369282892,
 19636.753964898147]
```

```
plt.plot(range(1,11),WCSS,marker='x')
```

```
[<matplotlib.lines.Line2D at 0x7d84c85ac370>]
```



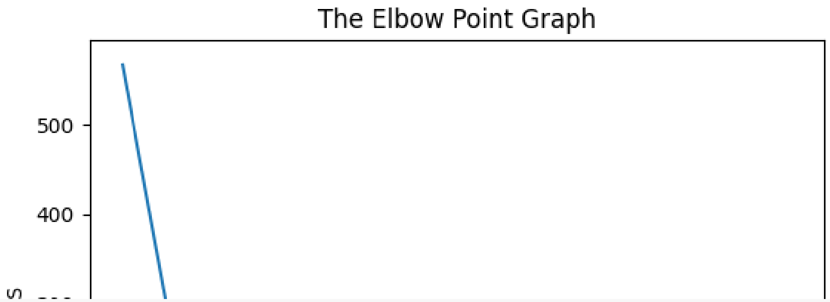
```
#6B Clustering algorithms for unsupervised classification.
import pandas as pd
import matplotlib.pyplot as plt
#Jupyter Notebooks to show the plots
%matplotlib inline
```

```
#Importing the dataset
iris = pd.read_csv("/content/drive/MyDrive/KRAI/iris_csv.csv")
```

```
iris_clustering = iris.drop(columns = ['class'])
X = iris_clustering.iloc[:, [0,2]].values
X
```

```
       [7.4, 6.1],
       [7.9, 6.4],
       [6.4, 5.6],
       [6.3, 5.1],
       [6.1, 5.6],
       [7.7, 6.1],
       [6.3, 5.6],
       [6.4, 5.5],
       [6. , 4.8],
       [6.9, 5.4],
       [6.7, 5.6],
       [6.9, 5.1],
       [5.8, 5.1],
       [6.8, 5.9],
       [6.7, 5.7],
       [6.7, 5.2],
       [6.3, 5. ],
       [6.5, 5.2],
       [6.2, 5.4],
       [5.9, 5.1]])
```

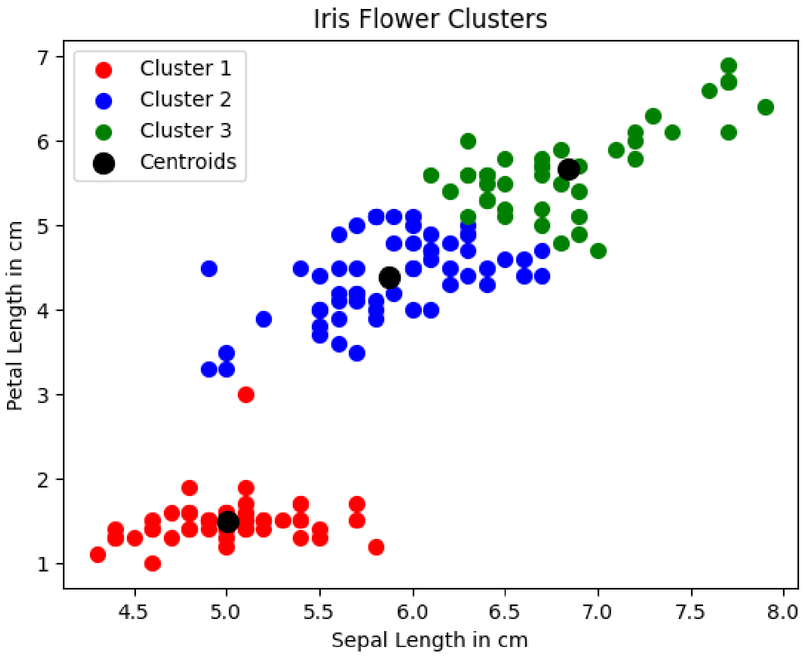
```
from sklearn.cluster import KMeans
import warnings

warnings.filterwarnings('ignore', category=FutureWarning)
wcss = []
for i in range(1, 11):
   kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
   kmeans.fit(X)
   wcss.append(kmeans.inertia_)
#Plotting The Elbow graph
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Point Graph')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```




```
#Initialising K-Means With Optimum Number Of Clusters
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 0)
y = kmeans.fit_predict(X)
print(y)
```

```
[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 1 2 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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 2 1 2 2 2 2 1 2 2 2 2
 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2
 2 1]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_
  warnings.warn(
```

```
plt.scatter(X[y == 0, 0], X[y == 0, 1], s = 50, c = 'red', label = 'Cluster 1')
plt.scatter(X[y == 1, 0], X[y == 1, 1], s = 50, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y == 2, 0], X[y == 2, 1], s = 50, c = 'green', label = 'Cluster 3')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 100, c = 'black', label = 'Centroids')

plt.title('Iris Flower Clusters')
plt.xlabel('Sepal Length in cm')
plt.ylabel('Petal Length in cm')
plt.legend()
plt.show()
```


Iris Flower Clusters
Cluster 1
Cluster 2
Cluster 3
Centroids
Petal Length in cm
Sepal Length in cm
7
6
5
4
3
2
1
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

```
# 7A Developing and implementing Decision Tree model on the
# dataset.

import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
data_set= pd.read_csv('/content/drive/MyDrive/KRAI/User_data.csv')
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
print(x)
print(y)
```

```
[[    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]
```

```
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=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

```
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
y_pred= classifier.predict(x_test)
y_pred
```

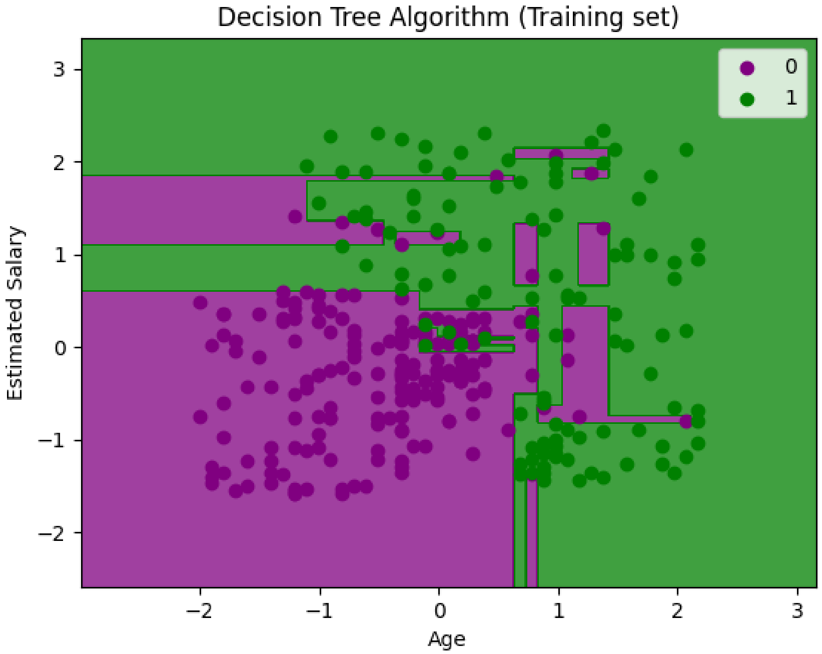
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1,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

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

```
array([[62,  6],
       [ 3, 29]])
```

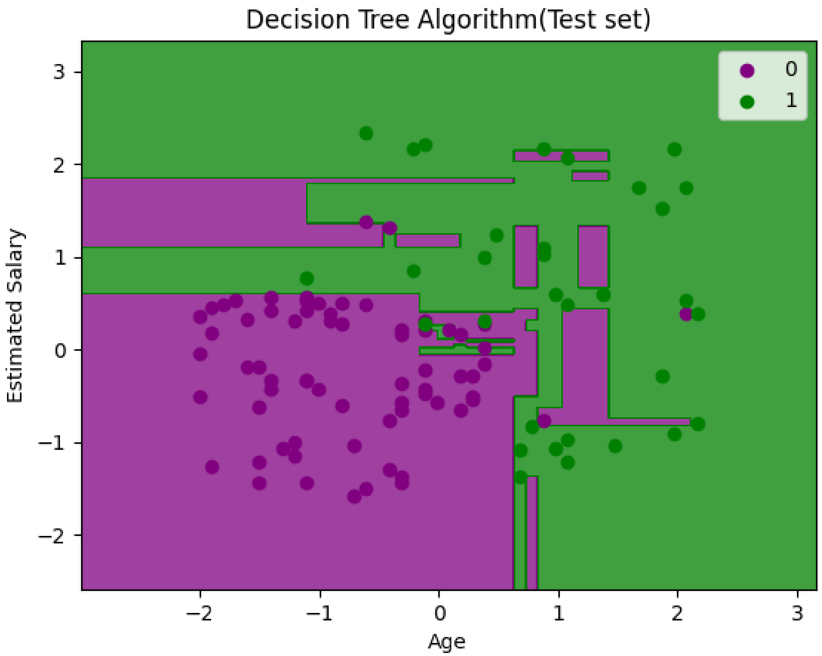
```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
<ipython-input-7-7887977755a3>:10: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],c = ListedColormap(('purple', 'green'))(i), label = j)
```



```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
<ipython-input-8-a6028a9cd450>:11: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c = ListedColormap(('purple', 'green'))(i), label = j)
```



```
# 7B Developing and implementing Decision Tree model on the dataset.

import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

data_set= pd.read_csv('/content/drive/MyDrive/KRAI/User_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
print(x)
print(y)

# Splitting the dataset into training and 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= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

```
[[    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]
```

```
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

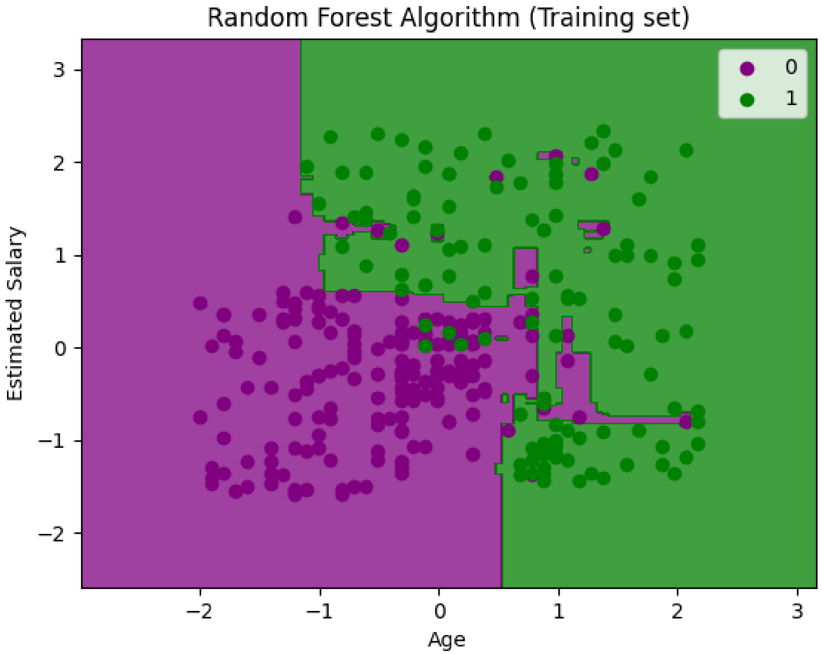
```
y_pred= classifier.predict(x_test)
```

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

```
array([[66,  2],
       [ 3, 29]])
```

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Random Forest Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
<ipython-input-7-ed9d87ca4c08>:10: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
  mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],c = ListedColormap(('purple', 'green'))(i), label = j)
```



```
# 8 Bayesian classification on any dataset.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

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

```
x=iris.data[:,]
#four features value of iris flower
```

```
y=iris.target
y
#corrsponding three flower classes 0, 1,2
```

```
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])
```

```
print("Features=",iris['feature_names'])
```

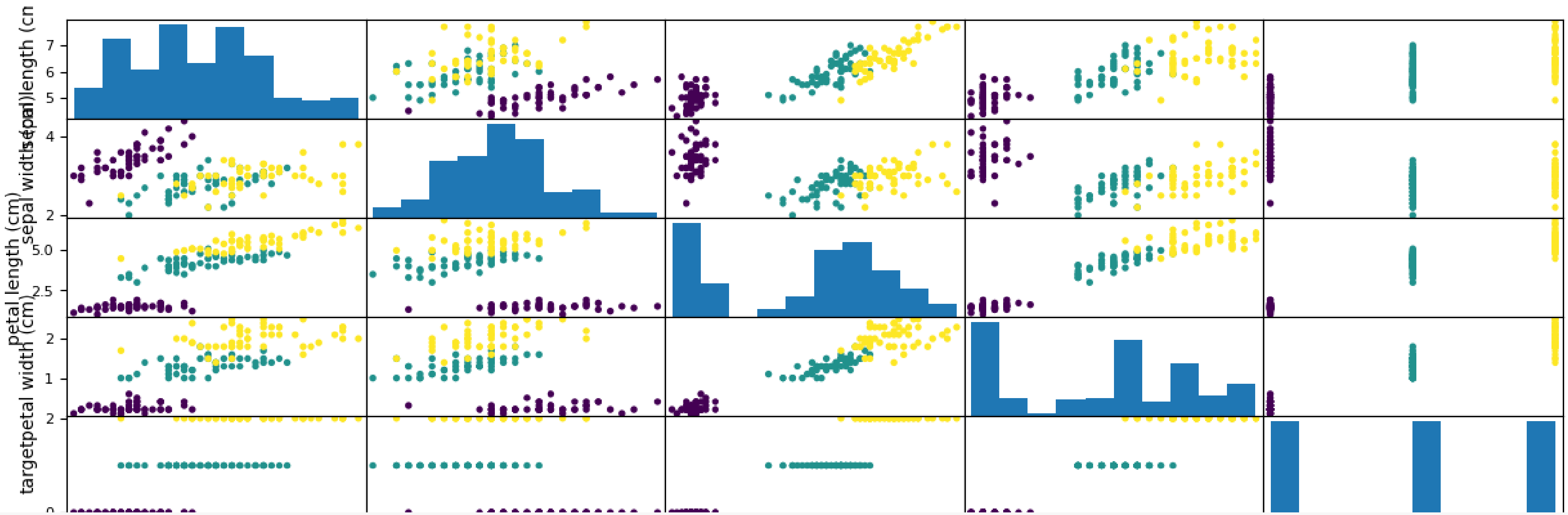
```
Features= ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```
iris_dataframe = pd.DataFrame(data=np.c_[iris['data'],iris['target']], columns=iris['feature_names']+['target'])
```

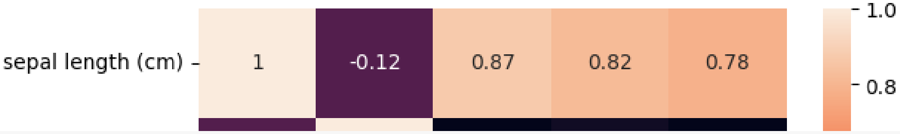
```
plt.figure()
```

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

```
import pandas as pd
import matplotlib.pyplot as plt
plt = pd.plotting.scatter_matrix(iris_dataframe, c=iris['target'],
                                 figsize=(15,5),
                                 s=60,alpha=1)
plt.show()
```



```
#step-3 visualising corelation &checking assumptions of Naive bayes
import matplotlib.pyplot as plt
import seaborn as sns
dataplot = sns.heatmap(iris_dataframe.corr(), annot=True)
plt.show()
```

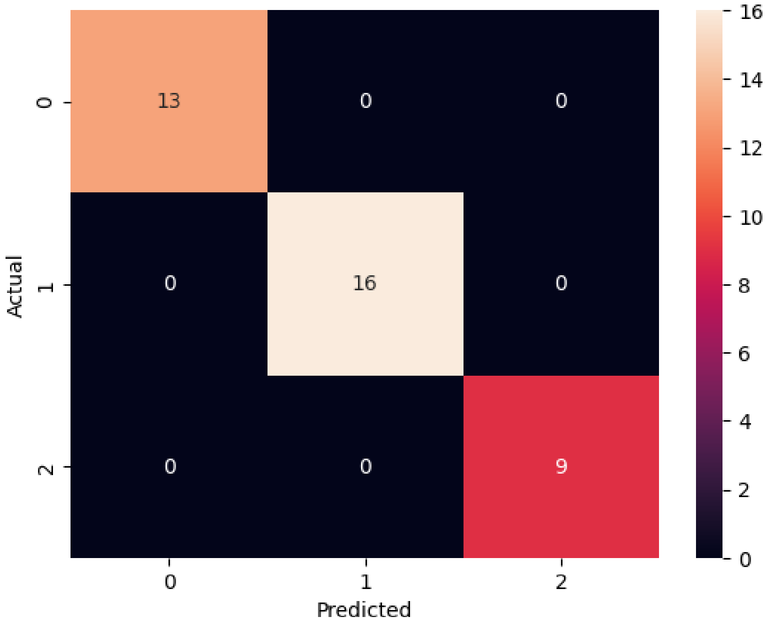



```
#step-4  split dataset

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=0 )
```

```
#step-5  fit the model
from sklearn.naive_bayes import GaussianNB
NB = GaussianNB()
NB.fit(X_train, y_train)
```

```
▾ GaussianNB
GaussianNB()
```

```
#step-6  Evaluate the model
import matplotlib.pyplot as plt
Y_pred = NB.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, Y_pred)
df_cm = pd.DataFrame(cm, columns=np.unique(y_test), index = np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
sns.heatmap(df_cm, annot=True) #font size
plt.show()
```


13
0
0
0
16
0
0
0
9
Actual
Predicted
0
1
2
0
1
2
0
2
4
6
8
10
12
14
16

```
# 9 SVM classification on any dataset.
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

```
df=pd.read_csv('/content/drive/MyDrive/KRAI/Social_Network_Ads.csv')
```

```
df.head()
```

	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

+ Code + Text

```
X=df[['Age','EstimatedSalary']]
y=df['Purchased']
print(X)
print(y)
```

```
     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 x 2 columns]
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=0.23, random_state=91)
```

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
```

```
from sklearn.svm import SVC
```

```
model_lin = SVC(kernel='linear')
model_lin.fit(X_train_scaled,y_train)
model_lin.score(X_test_scaled,y_test)
```

```
0.8043478260869565
```

```
model_poly = SVC(kernel='poly')
model_poly.fit(X_train_scaled,y_train)
model_poly.score(X_test_scaled,y_test)
```

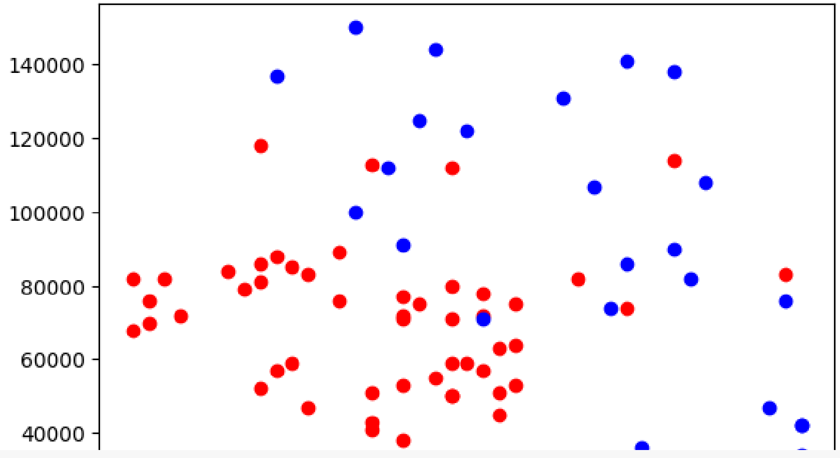
```
0.8913043478260869
```

```
model_rbf = SVC(kernel='rbf')
model_rbf.fit(X_train_scaled,y_train)
model_rbf.score(X_test_scaled,y_test)
```

```
0.8913043478260869
```

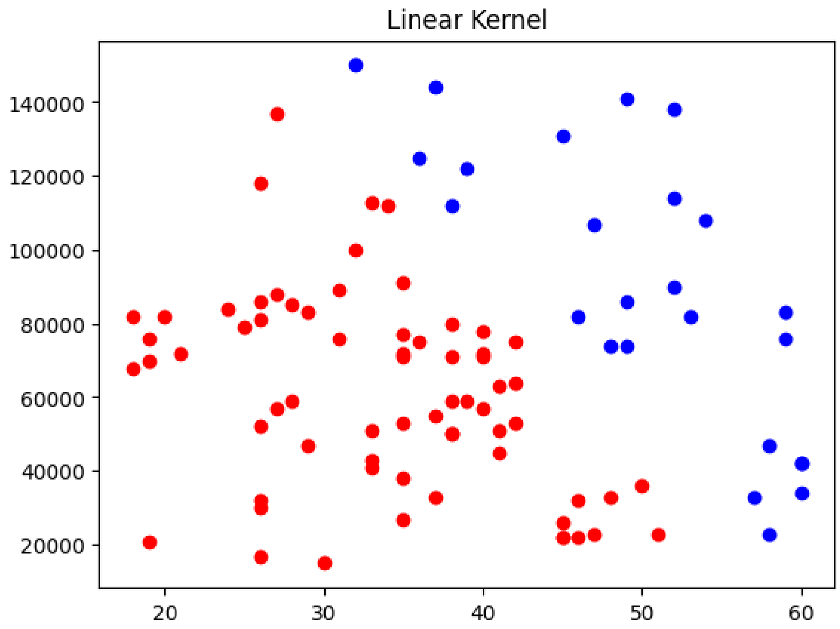
```
#Actual data
class_0_act = X_test[y_test==0]
class_1_act = X_test[y_test==1]
plt.scatter(class_0_act['Age'],class_0_act['EstimatedSalary'],c='red')
plt.scatter(class_1_act['Age'],class_1_act['EstimatedSalary'],c='blue')
```

```
<matplotlib.collections.PathCollection at 0x78b89920b5b0>
```



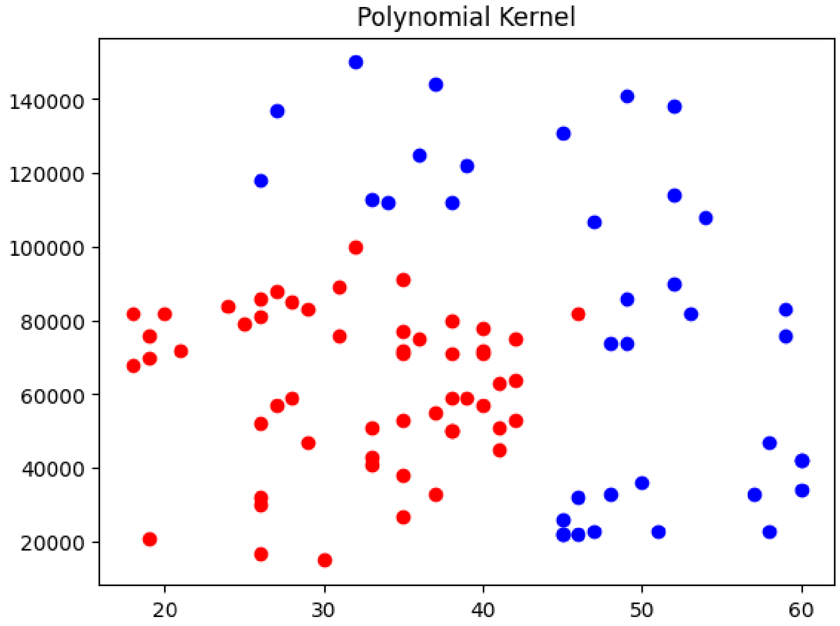

```
#Plot points according to predicted values of linear kernel
y_pre = model_lin.predict(X_test_scaled)
class_0_pre = X_test[y_pre==0]
class_1_pre = X_test[y_pre==1]
plt.scatter(class_0_pre['Age'],class_0_pre['EstimatedSalary'],c='red')
plt.scatter(class_1_pre['Age'],class_1_pre['EstimatedSalary'],c='blue')
plt.title('Linear Kernel')
```

```
Text(0.5, 1.0, 'Linear Kernel')
```



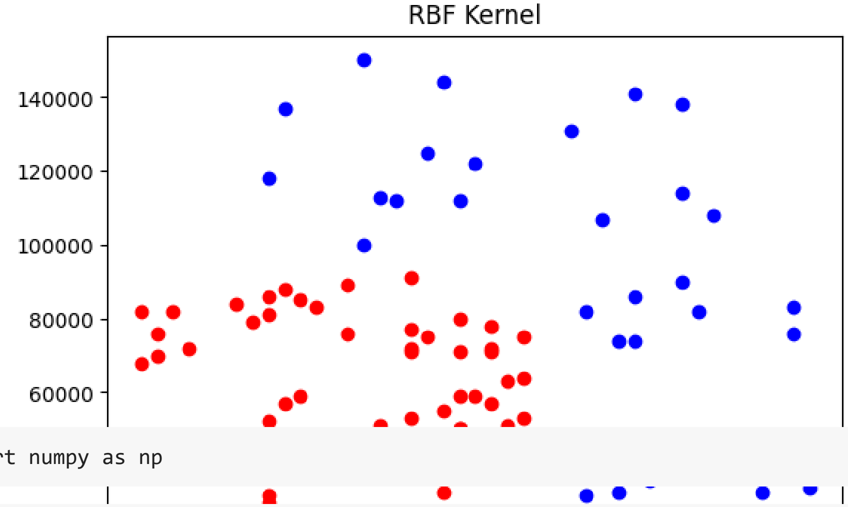
```
#Plot points according to predicted values of polynomial kernel
y_pre = model_poly.predict(X_test_scaled)
class_0_pre = X_test[y_pre==0]
class_1_pre = X_test[y_pre==1]
plt.scatter(class_0_pre['Age'],class_0_pre['EstimatedSalary'],c='red')
plt.scatter(class_1_pre['Age'],class_1_pre['EstimatedSalary'],c='blue')
plt.title('Polynomial Kernel')
```

```
Text(0.5, 1.0, 'Polynomial Kernel')
```




```
#Plot points according to predicted values of rbf kernel
y_pre = model_rbf.predict(X_test_scaled)
class_0_pre = X_test[y_pre==0]
class_1_pre = X_test[y_pre==1]
plt.scatter(class_0_pre['Age'],class_0_pre['EstimatedSalary'],c='red')
plt.scatter(class_1_pre['Age'],class_1_pre['EstimatedSalary'],c='blue')
plt.title('RBF Kernel')
```

```
Text(0.5, 1.0, 'RBF Kernel')
```




```
import numpy as np
```

```
#usingarray
plot_data = []
for x in range(0,100,1):
    for y in range(0,100,1):
        plot_data.append([x,y])
plot_data=np.array(plot_data)/100
```

```
plot_data
```

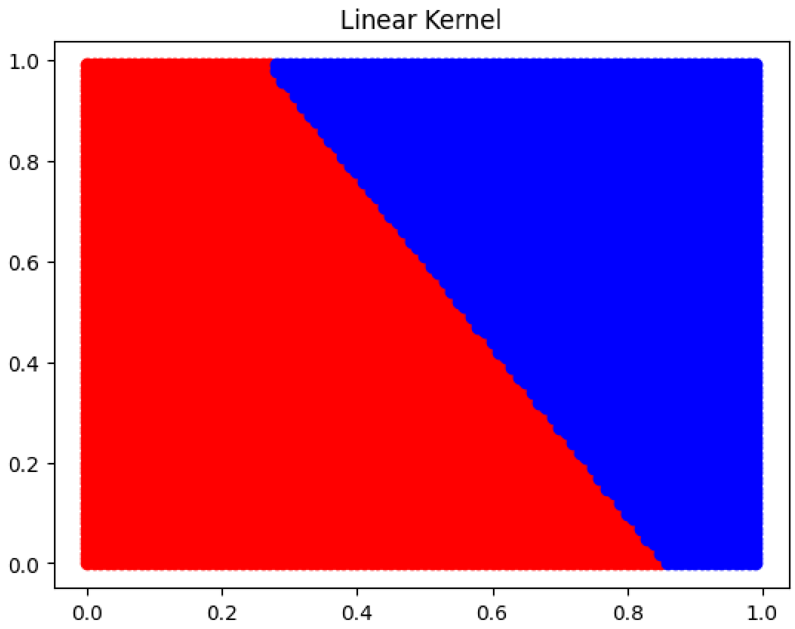
```
array([[0.  , 0.  ],
       [0.  , 0.01],
       [0.  , 0.02],
       ...,
       [0.99, 0.97],
       [0.99, 0.98],
       [0.99, 0.99]])
```

```
plot_data.shape
```

```
(10000, 2)
```

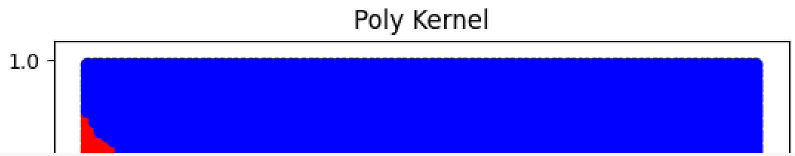
```
y_plot = model_lin.predict(plot_data)
class_0 = plot_data[y_plot==0]
class_1 = plot_data[y_plot==1]
plt.scatter(class_0[:,0],class_0[:,1],c='red')
plt.scatter(class_1[:,0],class_1[:,1],c='blue')
plt.title('Linear Kernel')
```

```
Text(0.5, 1.0, 'Linear Kernel')
```



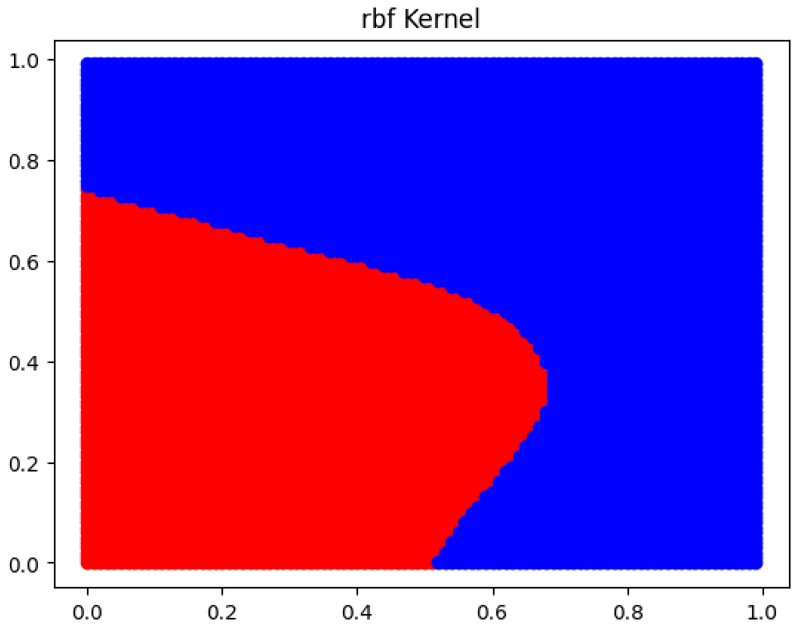

```
y_plot = model_poly.predict(plot_data)
class_0 = plot_data[y_plot==0]
class_1 = plot_data[y_plot==1]
plt.scatter(class_0[:,0],class_0[:,1],c='red')
plt.scatter(class_1[:,0],class_1[:,1],c='blue')
plt.title('Poly Kernel')
```

```
Text(0.5, 1.0, 'Poly Kernel')
```




```
y_plot = model_rbf.predict(plot_data)
class_0 = plot_data[y_plot==0]
class_1 = plot_data[y_plot==1]
plt.scatter(class_0[:,0],class_0[:,1],c='red')
plt.scatter(class_1[:,0],class_1[:,1],c='blue')
plt.title('rbf Kernel')
```

```
Text(0.5, 1.0, 'rbf Kernel')
```




```
pts = np.array([[25,60000],[50,120000]])
pts_scaled = scaler.transform(pts)
```

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

```
pts_scaled
```

```
array([[0.16666667, 0.33333333],
       [0.76190476, 0.77777778]])
```

```
y = model_rbf.predict(pts_scaled)
y
```

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

```
# 10  Plot the cluster data using python visualizations.
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import numpy as np
data = load_digits().data
print(data)
pca = PCA(2)
```

```
[[ 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.]]
```

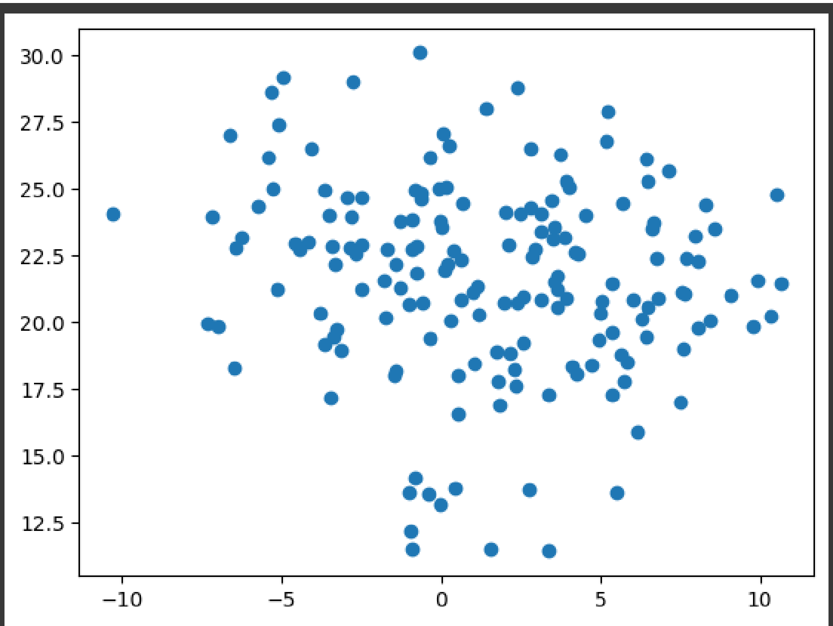
```
df = pca.fit_transform(data)
df
```

```
array([[ -1.2594668 ,  21.2748827 ],
       [  7.95761055, -20.76869699],
       [  6.99192373,  -9.95598477],
       ...,
       [ 10.80128419,  -6.96025882],
       [ -4.87210056,  12.42396585],
       [ -0.34438856,   6.36554123]])
```

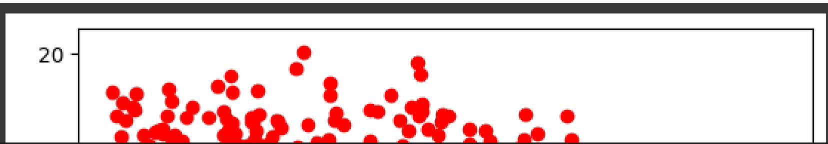
```
df.shape
(1797, 2)
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters= 10)
label = kmeans.fit_predict(df)
print(label)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_
  warnings.warn(
[0 9 1 ... 1 2 7]
```

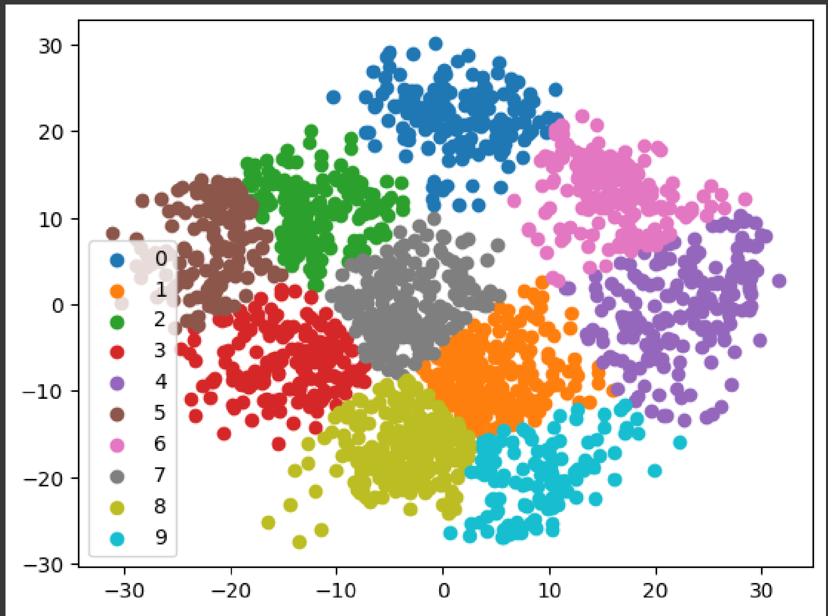
```
import matplotlib.pyplot as plt
filtered_label0 = df[label == 0]
plt.scatter(filtered_label0[:,0] , filtered_label0[:,1])
plt.show()
```



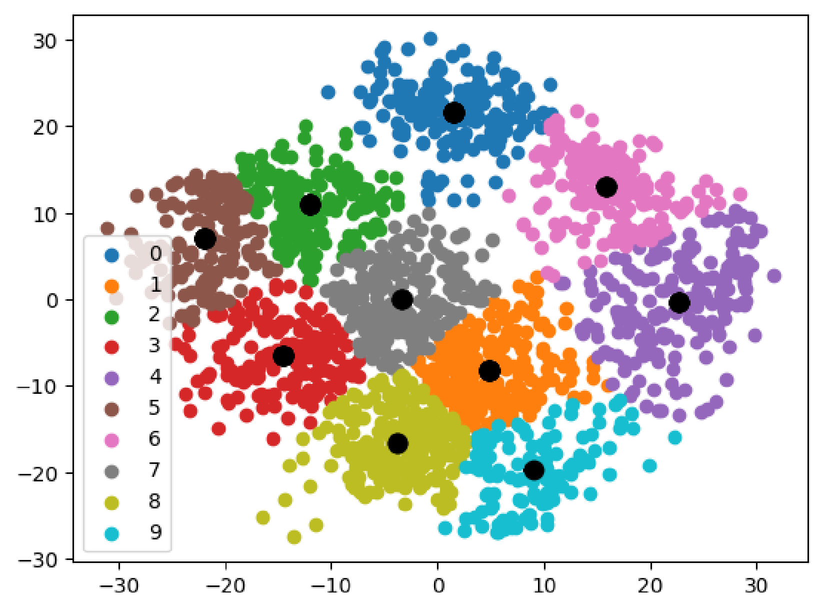

```
filtered_label2 = df[label == 2]
filtered_label8 = df[label == 8]
plt.scatter(filtered_label2[:,0] , filtered_label2[:,1] , color =
'red')
plt.scatter(filtered_label8[:,0] , filtered_label8[:,1] , color =
'black')
plt.show()
```




```
u_labels = np.unique(label)
for i in u_labels:
 plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
plt.legend()
plt.show()
```




```
centroids = kmeans.cluster_centers_
u_labels = np.unique(label)
for i in u_labels:
 plt.scatter(df[label == i , 0] , df[label == i , 1] , label = i)
 plt.scatter(centroids[:,0] , centroids[:,1] , s = 80, color = 'k')
plt.legend()
plt.show()
```



30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30
0
1
2
3
4
5
6
7
8
9

```
#Method-1
# Creating & Visualizing Neural Network for the given data.  (Use python)
from keras.models import Sequential
from keras.layers import Dense, Activation
import numpy as np
```

```
# Use numpy arrays to store inputs (x) and outputs (y):
x = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])

model = Sequential()
model.add(Dense(2, input_shape=(2,)))
model.add(Activation('sigmoid'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

# Compile the model and calculate its accuracy:
model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])

# Print a summary of the Keras model:
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 2)                 6         
                                                                 
 activation (Activation)     (None, 2)                 0         
                                                                 
 dense_1 (Dense)             (None, 1)                 3         
                                                                 
 activation_1 (Activation)   (None, 1)                 0         
                                                                 
=================================================================
Total params: 9 (36.00 Byte)
Trainable params: 9 (36.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#Method-2
#Create a Neural Network from Scratch
# Import python libraries required in this example:
import numpy as np
from scipy.special import expit as activation_function
from scipy.stats import truncnorm

# DEFINE THE NETWORK

# Generate random numbers within a truncated (bounded)
# normal distribution:
def truncated_normal(mean=0, sd=1, low=0, upp=10):
    return truncnorm(
        (low - mean) / sd, (upp - mean) / sd, loc=mean, scale=sd)

# Create the ‘Nnetwork’ class and define its arguments:
# Set the number of neurons/nodes for each layer
# and initialize the weight matrices:
class Nnetwork:

    def __init__(self,
                 no_of_in_nodes,
                 no_of_out_nodes,
                 no_of_hidden_nodes,
                 learning_rate):
        self.no_of_in_nodes = no_of_in_nodes
        self.no_of_out_nodes = no_of_out_nodes
        self.no_of_hidden_nodes = no_of_hidden_nodes
        self.learning_rate = learning_rate
        self.create_weight_matrices()

    def create_weight_matrices(self):
        """ A method to initialize the weight matrices of the neural network"""
        rad = 1 / np.sqrt(self.no_of_in_nodes)
        X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
        self.weights_in_hidden = X.rvs((self.no_of_hidden_nodes,
                                        self.no_of_in_nodes))
        rad = 1 / np.sqrt(self.no_of_hidden_nodes)
```

```
        X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
        self.weights_hidden_out = X.rvs((self.no_of_out_nodes,
                                         self.no_of_hidden_nodes))

    def train(self, input_vector, target_vector):
        pass # More work is needed to train the network

    def run(self, input_vector):
        """
        running the network with an input vector 'input_vector'.
        'input_vector' can be tuple, list or ndarray
        """
        # Turn the input vector into a column vector:
        input_vector = np.array(input_vector, ndmin=2).T
        # activation_function() implements the expit function,
        # which is an implementation of the sigmoid function:
        input_hidden = activation_function(self.weights_in_hidden @   input_vector)
        output_vector = activation_function(self.weights_hidden_out @ input_hidden)
        return output_vector

# RUN THE NETWORK AND GET A RESULT

# Initialize an instance of the class:
simple_network = Nnetwork(no_of_in_nodes=2,
                               no_of_out_nodes=2,
                               no_of_hidden_nodes=4,
                               learning_rate=0.6)

# Run simple_network for arrays, lists and tuples with shape (2):
# and get a result:
simple_network.run([(3, 4)])
```

```
array([[0.39420718],
       [0.50790498]])
```

```
#12 Recognize optical character using ANN.
from tensorflow.keras.datasets import mnist
```

```
(x_train,y_train),(x_test,y_test)=mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
x_train.shape
```

```
(60000, 28, 28)
```

```
X_train=x_train.reshape(60000,784)
print("x-train")
print(X_train)
X_test=x_test.reshape(10000,784)
print("x-test")
print(X_test)
```

```
x-train
[[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]]
x-test
[[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]]
```

```
from tensorflow.keras.utils import to_categorical
```

```
y_train=to_categorical(y_train,num_classes=10)
y_test=to_categorical(y_test,num_classes=10)
```

```
X_train=X_train/255
X_test=X_test/255
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
model=Sequential()
model.add(Dense(50,activation='relu',input_shape=(784,)))
model.add(Dense(50,activation='relu'))
model.add(Dense(10,activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                39250

 dense_1 (Dense)             (None, 50)                2550

 dense_2 (Dense)             (None, 10)                510

=================================================================
Total params: 42310 (165.27 KB)
Trainable params: 42310 (165.27 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy',metrics=['accuracy'])
```

```
model.fit(X_train,y_train,batch_size=64,epochs=10,validation_data=(X_test,y_test))
```

```
Epoch 1/10
938/938 [==============================] - 2s 2ms/step - loss: 0.3326 - accuracy: 0.9051 - val_loss: 0.1704 - val_accuracy: 0.9496
Epoch 2/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1570 - accuracy: 0.9536 - val_loss: 0.1392 - val_accuracy: 0.9557
Epoch 3/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1189 - accuracy: 0.9646 - val_loss: 0.1155 - val_accuracy: 0.9656
Epoch 4/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0956 - accuracy: 0.9711 - val_loss: 0.1070 - val_accuracy: 0.9689
Epoch 5/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0803 - accuracy: 0.9764 - val_loss: 0.1065 - val_accuracy: 0.9694
Epoch 6/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0703 - accuracy: 0.9790 - val_loss: 0.0939 - val_accuracy: 0.9728
Epoch 7/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0616 - accuracy: 0.9815 - val_loss: 0.0972 - val_accuracy: 0.9709
Epoch 8/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0555 - accuracy: 0.9827 - val_loss: 0.0981 - val_accuracy: 0.9730
Epoch 9/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0508 - accuracy: 0.9847 - val_loss: 0.0907 - val_accuracy: 0.9737
Epoch 10/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0456 - accuracy: 0.9862 - val_loss: 0.1045 - val_accuracy: 0.9712
<keras.src.callbacks.History at 0x786b8d9212d0>
```

```
import numpy as np
```

```
X_train
```

```
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.]])
```

```
y_train[:5,:]
```

```
array([[0., 0., 0., 0., 0., 1., 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., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

```
img0 = np.array(X_train[0]).reshape(1,784)
```

```
model.predict(img0).argmax()
```

```
1/1 [==============================] - 0s 61ms/step
5
```

```
y_train[0].argmax()
```

```
5
```

```
def recognise(img):
  img=np.array(img).reshape(1,784)
  return model.predict(img).argmax()
```

```
y_pre=model.predict(X_test).argmax(axis=1)
```

```
313/313 [==============================] - 0s 1ms/step
```

```
y_pre
```

```
array([7, 2, 1, ..., 4, 5, 6])
```

```
len(y_pre)
```

```
10000
```

```
y_test.argmax(axis=1)
```

```
array([7, 2, 1, ..., 4, 5, 6])
```

```
sum(y_pre==y_test.argmax(axis=1))
```

```
9712
```

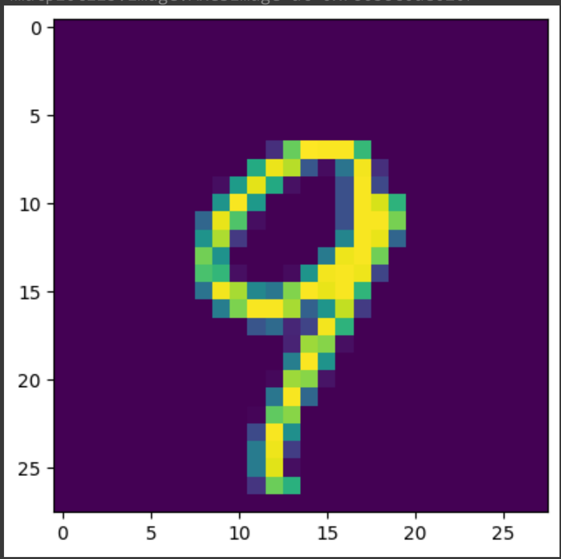
```
9737/10000
```

```
0.9737
```

```
import matplotlib.pyplot as plt
```

```
plt.imshow(np.array(X_test[560]).reshape(28,28))
```

```
<matplotlib.image.AxesImage at 0x786b8c0ae620>
```



```
recognise(X_test[560])
```

```
1/1 [==============================] - 0s 13ms/step
9
```

```
# 13 Write a program to implement CNN

#implement  CNN

# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version u
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
!unzip drive/My\ Drive/data/dog-vs-cat.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: train/train/dog.5499.jpg
  inflating: train/train/dog.55.jpg
  inflating: train/train/dog.550.jpg
  inflating: train/train/dog.5500.jpg
  inflating: train/train/dog.5501.jpg
  inflating: train/train/dog.5502.jpg
  inflating: train/train/dog.5503.jpg
  inflating: train/train/dog.5504.jpg
  inflating: train/train/dog.5505.jpg
  inflating: train/train/dog.5506.jpg
  inflating: train/train/dog.5507.jpg
  inflating: train/train/dog.5508.jpg
  inflating: train/train/dog.5509.jpg
  inflating: train/train/dog.551.jpg
  inflating: train/train/dog.5510.jpg
  inflating: train/train/dog.5511.jpg
  inflating: train/train/dog.5512.jpg
  inflating: train/train/dog.5513.jpg
  inflating: train/train/dog.5514.jpg
  inflating: train/train/dog.5515.jpg
  inflating: train/train/dog.5516.jpg
  inflating: train/train/dog.5517.jpg
  inflating: train/train/dog.5518.jpg
  inflating: train/train/dog.5519.jpg
  inflating: train/train/dog.552.jpg
  inflating: train/train/dog.5520.jpg
  inflating: train/train/dog.5521.jpg
  inflating: train/train/dog.5522.jpg
  inflating: train/train/dog.5523.jpg
  inflating: train/train/dog.5524.jpg
  inflating: train/train/dog.5525.jpg
  inflating: train/train/dog.5526.jpg
  inflating: train/train/dog.5527.jpg
  inflating: train/train/dog.5528.jpg
  inflating: train/train/dog.5529.jpg
  inflating: train/train/dog.553.jpg
  inflating: train/train/dog.5530.jpg
  inflating: train/train/dog.5531.jpg
  inflating: train/train/dog.5532.jpg
  inflating: train/train/dog.5533.jpg
  inflating: train/train/dog.5534.jpg
  inflating: train/train/dog.5535.jpg
  inflating: train/train/dog.5536.jpg
  inflating: train/train/dog.5537.jpg
  inflating: train/train/dog.5538.jpg
  inflating: train/train/dog.5539.jpg
  inflating: train/train/dog.554.jpg
  inflating: train/train/dog.5540.jpg
  inflating: train/train/dog.5541.jpg
  inflating: train/train/dog.5542.jpg
  inflating: train/train/dog.5543.jpg
  inflating: train/train/dog.5544.jpg
  inflating: train/train/dog.5545.jpg
```

```
inflating: train/train/dog.5546.jpg
inflating: train/train/dog.5547.jpg
inflating: train/train/dog.5548.jpg
inflating: train/train/dog.5549.jpg
```

```
os.listdir('/content/train/train')
#os.listdir('/kaggle/input/dogs-vs-cats/')
```

```
['cat.5159.jpg',
 'cat.4869.jpg',
 'cat.6287.jpg',
 'cat.4140.jpg',
 'cat.7473.jpg',
 'cat.7708.jpg',
 'cat.9746.jpg',
 'dog.10648.jpg',
 'cat.9488.jpg',
 'dog.6925.jpg',
 'cat.2023.jpg',
 'cat.2067.jpg',
 'cat.4734.jpg',
 'dog.3496.jpg',
 'dog.7765.jpg',
 'dog.1560.jpg',
 'dog.174.jpg',
 'cat.10510.jpg',
 'cat.6041.jpg',
 'cat.10383.jpg',
 'dog.429.jpg',
 'dog.1104.jpg',
 'dog.6480.jpg',
 'dog.7422.jpg',
 'cat.8339.jpg',
 'dog.8490.jpg',
 'dog.1128.jpg',
 'dog.11297.jpg',
 'cat.3558.jpg',
 'cat.4831.jpg',
 'cat.4311.jpg',
 'cat.3760.jpg',
 'dog.6678.jpg',
 'cat.2635.jpg',
 'dog.10873.jpg',
 'cat.552.jpg',
 'dog.6220.jpg',
 'cat.4452.jpg',
 'dog.5030.jpg',
 'dog.3626.jpg',
 'cat.8091.jpg',
 'dog.9545.jpg',
 'cat.2848.jpg',
 'cat.791.jpg',
 'cat.4522.jpg',
 'cat.2638.jpg',
 'dog.9447.jpg',
 'cat.6560.jpg',
 'cat.7394.jpg',
 'cat.5153.jpg',
 'cat.7299.jpg',
 'cat.8475.jpg',
 'dog.6026.jpg',
 'cat.8148.jpg',
 'cat.117.jpg',
 'cat.1445.jpg',
 'cat.11986.jpg',
 'cat.7351.jpg',
```

```
filenames=os.listdir('/content/train/train')
```

```
len(filenames)
```

```
25000
```

```
filenames[:10]
```

```
['cat.5159.jpg',
 'cat.4869.jpg',
 'cat.6287.jpg',
 'cat.4140.jpg',
 'cat.7473.jpg',
 'cat.7708.jpg',
 'cat.9746.jpg',
 'dog.10648.jpg',
 'cat.9488.jpg',
 'dog.6925.jpg']
```

```
df=pd.DataFrame({'filename':filenames})
df.head()
```

	filename
0	cat.5159.jpg
1	cat.4869.jpg
2	cat.6287.jpg
3	cat.4140.jpg
4	cat.7473.jpg

```
df['class']=df['filename'].apply(lambda X:X[:3])
```

```
df.head()
```

	filename	class
0	cat.5159.jpg	cat
1	cat.4869.jpg	cat
2	cat.6287.jpg	cat
3	cat.4140.jpg	cat
4	cat.7473.jpg	cat

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
data_gen=ImageDataGenerator(zoom_range=0.2,shear_range=0.2,horizontal_flip=True,rescale=1/255)
```

```
train_data=data_gen.flow_from_dataframe(df,'/content/train/train',X='filename',y='class',target_size=(224,224))
```

```
Found 25000 validated image filenames belonging to 2 classes.
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPool2D,Flatten,Dense
```

```
model=Sequential()
model.add(Conv2D(16,(3,3),activation='relu',input_shape=(224,224,3)))
model.add(MaxPool2D())
model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(64,(5,5),activation='relu'))
model.add(MaxPool2D())
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPool2D())
model.add(Flatten())
model.add(Dense(2,activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 16)      448

 max_pooling2d (MaxPooling2D  (None, 111, 111, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 22, 22, 64)        102464

 max_pooling2d_3 (MaxPooling  (None, 11, 11, 64)       0
```

```
 2D)

 conv2d_4 (Conv2D)           (None, 9, 9, 128)         73856

 max_pooling2d_4 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 2)                 4098

=================================================================
Total params: 204,002
Trainable params: 204,002
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
model.fit_generator(train_data,epochs=5)
```

```
<ipython-input-49-fd4c89a97472>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please
  model.fit_generator(train_data,epochs=5)
Epoch 1/5
782/782 [==============================] - 1472s 2s/step - loss: 0.2344 - accuracy: 0.9004
Epoch 2/5
782/782 [==============================] - 1460s 2s/step - loss: 0.2147 - accuracy: 0.9092
Epoch 3/5
331/782 [===========>..................] - ETA: 14:03 - loss: 0.1974 - accuracy: 0.9192
```

```
import cv2
def get_class(img_path):
    img=cv2.imread(img_path)
    img=cv2.resize(img,(224,224))
    img=img/255
    op=model.predict(img.reshape(1,224,224,3)).argmax()
    return 'cat' if op==0 else 'dog'
```

```
train_data.class_mode
```

```
'categorical'
```

```
get_class('model.fit_generator(train_data,epochs=5)')
```

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-48-d9060c6ec3ad> in <module>
----> 1 get_class('model.fit_generator(train_data,epochs=5)')

<ipython-input-46-132abfc3b56d> in get_class(img_path)
      2 def get_class(img_path):
      3     img=cv2.imread(img_path)
----> 4     img=cv2.resize(img,(224,224))
      5     img=img/255
      6     op=model.predict(img.reshape(1,224,224,3)).argmax()

error: OpenCV(4.6.0) /io/opencv/modules/imgproc/src/resize.cpp:4052: error:
(-215:Assertion failed) !ssize.empty() in function 'resize'
```

SEARCH STACK OVERFLOW

```
#14 Write a program to implement RNN
from tensorflow.keras.datasets import imdb
```

```
(X_train,y_train),(X_test,y_test)=imdb.load_data(num_words=20000)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
```

```
X_train.shape,X_test.shape
```

```
((25000,), (25000,))
```

```
len(X_train[0]),len(X_train[1]),len(X_train[2]),len(X_train[3]),len(X_train[4])
```

```
(218, 189, 141, 550, 147)
```

```
y_train[:5]
```

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

```
X_train[0]
```

```
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
 173,
 36,
 256,
 5,
 25,
 100,
 43,
 838,
 112,
 50,
 670,
 2,
 9,
 35,
 480,
 284,
 5,
 150,
 4,
 172,
 112,
 167,
 2,
 336,
 385,
 39,
 4,
 172,
 4536,
 1111,
 17,
 546,
 38,
 13,
 447,
 4,
 192,
 50,
 16,
 6,
 147,
 2025,
 19,
```

```
import numpy as np
```

```
np.array(X_train[0])
```

```
array([    1,    14,    22,    16,    43,   530,   973,  1622,  1385,
          65,   458,  4468,    66,  3941,     4,   173,    36,   256,
           5,    25,   100,    43,   838,   112,    50,   670,     2,
           9,    35,   480,   284,     5,   150,     4,   172,   112,
         167,     2,   336,   385,    39,     4,   172,  4536,  1111,
          17,   546,    38,    13,   447,     4,   192,    50,    16,
           6,   147,  2025,    19,    14,    22,     4,  1920,  4613,
         469,     4,    22,    71,    87,    12,    16,    43,   530,
          38,    76,    15,    13,  1247,     4,    22,    17,   515,
          17,    12,    16,   626,    18, 19193,     5,    62,   386,
          12,     8,   316,     8,   106,     5,     4,  2223,  5244,
          16,   480,    66,  3785,    33,     4,   130,    12,    16,
          38,   619,     5,    25,   124,    51,    36,   135,    48,
          25,  1415,    33,     6,    22,    12,   215,    28,    77,
          52,     5,    14,   407,    16,    82, 10311,     8,     4,
         107,   117,  5952,    15,   256,     4,     2,     7,  3766,
           5,   723,    36,    71,    43,   530,   476,    26,   400,
         317,    46,     7,     4, 12118,  1029,    13,   104,    88,
           4,   381,    15,   297,    98,    32,  2071,    56,    26,
         141,     6,   194,  7486,    18,     4,   226,    22,    21,
         134,   476,    26,   480,     5,   144,    30,  5535,    18,
          51,    36,    28,   224,    92,    25,   104,     4,   226,
          65,    16,    38,  1334,    88,    12,    16,   283,     5,
          16,  4472,   113,   103,    32,    15,    16,  5345,    19,
         178,    32])
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
X=pad_sequences(X_train,maxlen=200)
X_val=pad_sequences(X_test,maxlen=200)
```

```
len(X[0])
```

```
200
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM,Dense,Embedding
```

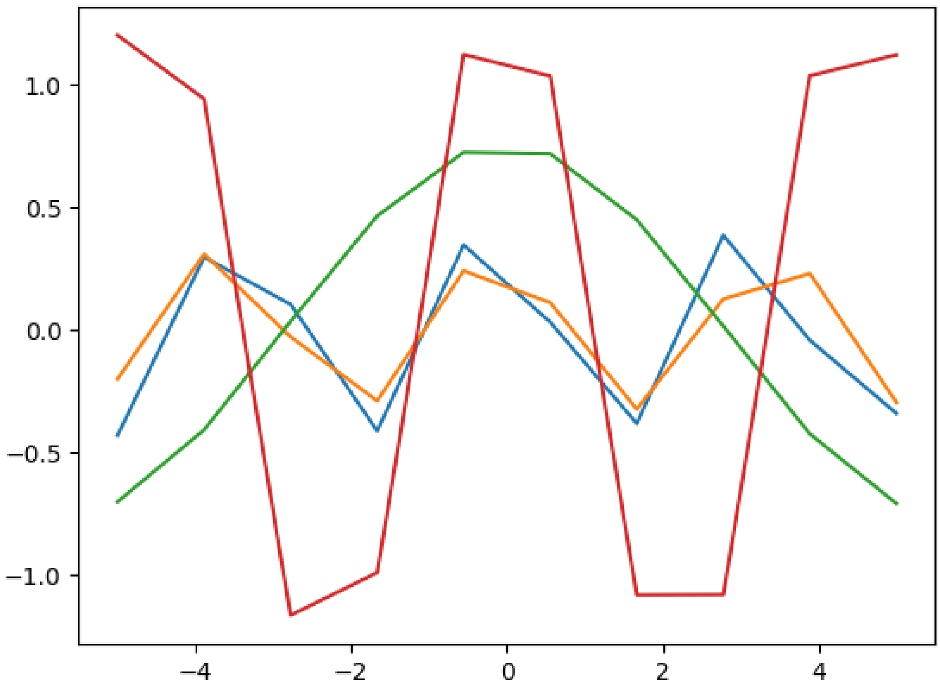
```
model=Sequential()
model.add(Embedding(20000,128,input_shape=(200,)))
model.add(LSTM(100,return_sequences=True))
model.add(LSTM(100))
model.add(Dense(1,activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
model.fit(X,y_train,validation_data=(X_val,y_test),epochs=5,batch_size=64)
```

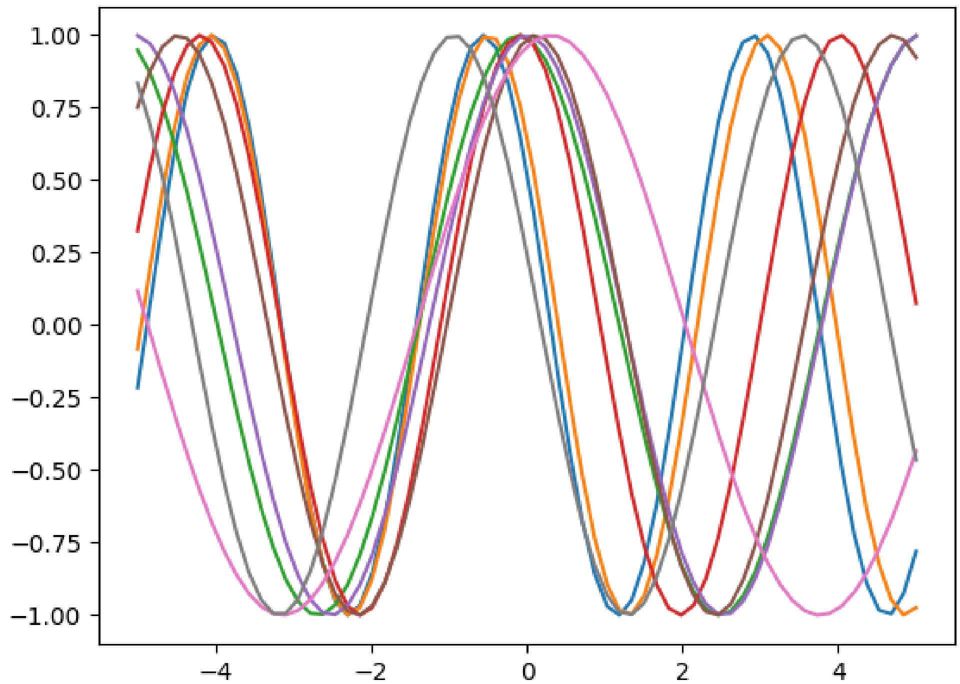
```
Epoch 1/5
391/391 [==============================] - 351s 888ms/step - loss: 0.4098 - accuracy: 0.8132 - val_loss: 0.3552 - val_accuracy: 0.8
Epoch 2/5
391/391 [==============================] - 337s 862ms/step - loss: 0.2205 - accuracy: 0.9155 - val_loss: 0.3394 - val_accuracy: 0.8
Epoch 3/5
391/391 [==============================] - 339s 869ms/step - loss: 0.1483 - accuracy: 0.9462 - val_loss: 0.3804 - val_accuracy: 0.8
Epoch 4/5
391/391 [==============================] - 340s 869ms/step - loss: 0.1003 - accuracy: 0.9636 - val_loss: 0.4294 - val_accuracy: 0.8
Epoch 5/5
391/391 [==============================] - 340s 869ms/step - loss: 0.0738 - accuracy: 0.9738 - val_loss: 0.5268 - val_accuracy: 0.8
<keras.src.callbacks.History at 0x7be8988a2ce0>
```

```
#     GAN

#https://www.codemotion.com/magazine/ai-ml/deep-learning/how-to-build-a-gan-in-python/

import matplotlib.pyplot as plt
import numpy as np
from numpy.random import randint, uniform

X_MIN = -5.0
X_MAX = 5.0
SAMPLE_LEN=10
X_COORDS = np.linspace(X_MIN , X_MAX, SAMPLE_LEN)
fig, axis = plt.subplots(1, 1)
for i in range(4):
    axis.plot(X_COORDS, uniform(0.1,2.0)*np.sin(uniform(0.2,2.0)*X_COORDS + uniform(2)))
```




```
import numpy as np
from numpy.random import uniform

import matplotlib.pyplot as plt

SAMPLE_LEN = 64       # number N of points where a curve is sampled
SAMPLE_SIZE = 32768   # number of curves in the training set
X_MIN = -5.0          # least ordinate where to sample
X_MAX = 5.0           # last ordinate where to sample

# The set of coordinates over which curves are sampled
X_COORDS = np.linspace(X_MIN , X_MAX, SAMPLE_LEN)

# The training set
SAMPLE = np.zeros((SAMPLE_SIZE, SAMPLE_LEN))
for i in range(0, SAMPLE_SIZE):
    b = uniform(0.5, 2.0)
    c = uniform(np.math.pi)
    SAMPLE[i] = np.array([np.sin(b*x + c) for x in X_COORDS])
```

```
# We plot the first 8 curves
fig, axis = plt.subplots(1, 1)
for i in range(8):
    axis.plot(X_COORDS, SAMPLE[i])
```




```
from keras.models import Sequential
from keras.layers import Dense, Dropout, LeakyReLU

DROPOUT = Dropout(0.4)        # Empirical hyperparameter
discriminator = Sequential()
discriminator.add(Dense(SAMPLE_LEN, activation="relu"))
discriminator.add(DROPOUT)
discriminator.add(Dense(SAMPLE_LEN, activation="relu"))
discriminator.add(DROPOUT)
discriminator.add(Dense(1, activation = "sigmoid"))
discriminator.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])


LEAKY_RELU = LeakyReLU(0.2)   # Empirical hyperparameter
generator = Sequential()
generator.add(Dense(SAMPLE_LEN))
generator.add(LEAKY_RELU)
generator.add(Dense(512))
generator.add(LEAKY_RELU)
generator.add(Dense(SAMPLE_LEN, activation = "tanh"))
generator.compile(optimizer = "adam", loss = "mse", metrics = ["accuracy"])


gan = Sequential()
gan.add(generator)
gan.add(discriminator)
gan.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])
```

```
EPOCHS = 3
BATCH=10

NOISE = uniform(X_MIN, X_MAX, size = (SAMPLE_SIZE, SAMPLE_LEN))
ONES = np.ones((SAMPLE_SIZE))
ZEROS = np.zeros((SAMPLE_SIZE))
print("epoch | dis. loss | dis. acc | gen. loss | gen. acc")
print("------+-----------+----------+-----------+----------")

fig = plt.figure(figsize = (8, 12))
ax_index = 1
for e in range(EPOCHS):
    for k in range(SAMPLE_SIZE//BATCH):
        # Addestra il discriminatore a riconoscere le sinusoidi vere da quelle prodotte dal genera
        n = randint(0, SAMPLE_SIZE, size = BATCH)
        # Ora prepara un batch di training record per il discriminatore
        p = generator.predict(NOISE[n])
        x = np.concatenate((SAMPLE[n], p))
        y = np.concatenate((ONES[n], ZEROS[n]))
        d_result = discriminator.train_on_batch(x, y)
        discriminator.trainable = False
        g_result = gan.train_on_batch(NOISE[n], ONES[n])
        discriminator.trainable = True
    print(f" {e:04n} |  {d_result[0]:.5f}  |  {d_result[1]:.5f} |  {g_result[0]:.5f}  |  {d_result
    # At 3, 13, 23, ... plots the last generator prediction
    if e % 10 == 3:
        ax = fig.add_subplot(8, 1, ax_index)
        plt.plot(X_COORDS, p[-1])
        ax.xaxis.set_visible(False)
        plt.ylabel(f"Epoch: {e}")
        ax_index += 1

# Plots a curve generated by the GAN
y = generator.predict(uniform(X_MIN, X_MAX, size = (1, SAMPLE_LEN)))[0]
ax = fig.add_subplot(8, 1, ax_index)
plt.plot(X_COORDS, y)
```

```
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 56ms/step
 0002 |  0.95137   |  0.50000 |  0.25160   |  0.50000
1/1 [==============================] - 0s 32ms/step
[<matplotlib.lines.Line2D at 0x7aa57904e5f0>]
```