GMM

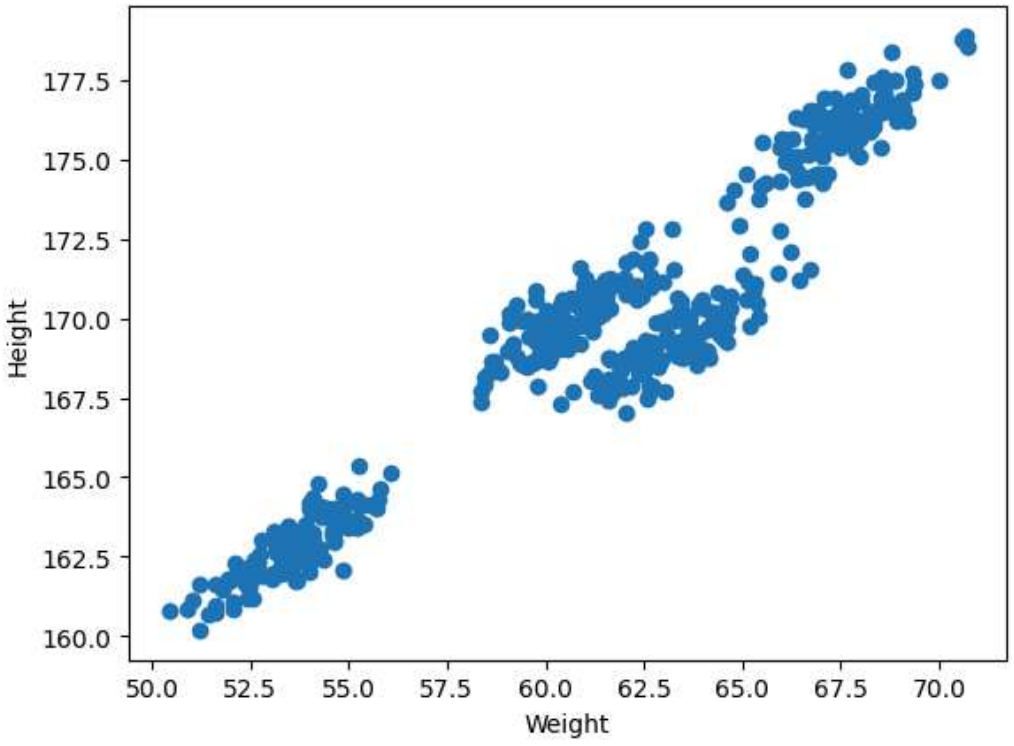
In [103]:
```
import pandas as pd
dataset=pd.read_csv("Clustering_gmm.csv")
dataset
```

Out[103]:

	Weight	Height
0	67.062924	176.086355
1	68.804094	178.388669
2	60.930863	170.284496
3	59.733843	168.691992
4	65.431230	173.763679
...	...	...
495	59.976983	169.679741
496	66.423814	174.625574
497	53.604698	161.919208
498	50.433644	160.794875
499	60.224392	169.689709

500 rows × 2 columns

In [104]:
```
import matplotlib.pyplot as plt
plt.scatter("Weight","Height",data=dataset)
plt.xlabel("Weight")
plt.ylabel("Height")
plt.figure(figsize=(8,8))
plt.show()
```



<Figure size 800x800 with 0 Axes>

In [105]:
```
from sklearn.cluster import KMeans
model=KMeans(n_clusters=4)
model.fit(dataset)
```

```
C:\Users\Neel\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks than
available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=
2.
  warnings.warn(
```

Out[105]:

KMeans (https://scikit-learn.org/1.5/modules/generated/sklearn.cluster.KMeans.html)

```
KMeans(n_clusters=4)
```

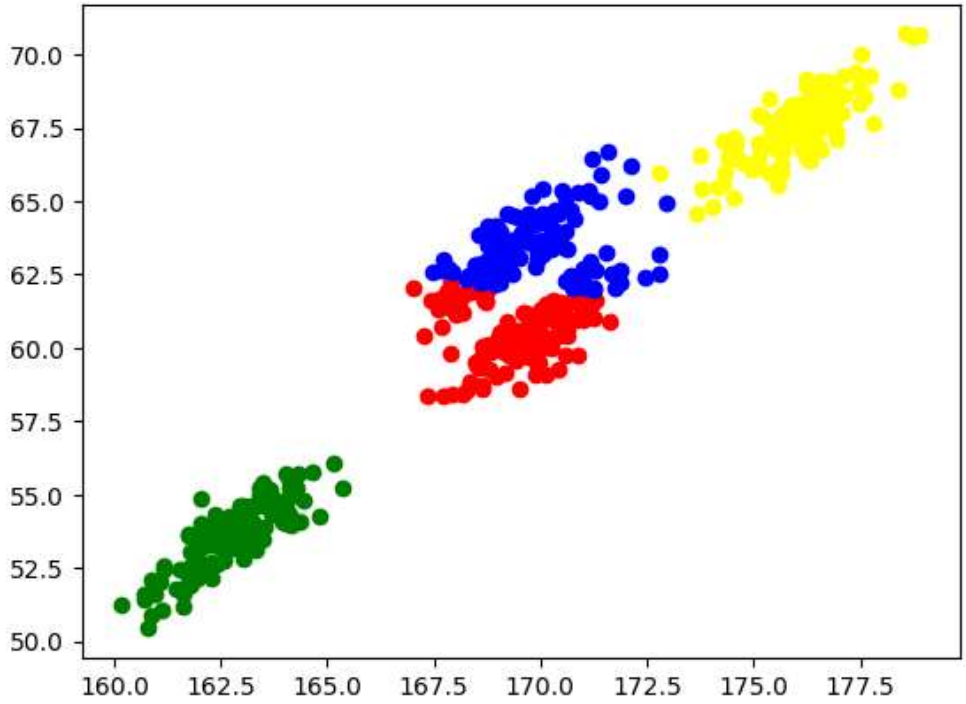
In [106]:
```
pred=model.predict(dataset)
dataset1=pd.DataFrame(dataset)
dataset1["prediction"]=pred
dataset1
```

Out[106]:

	Weight	Height	prediction
0	67.062924	176.086355	0
1	68.804094	178.388669	0
2	60.930863	170.284496	1
3	59.733843	168.691992	1
4	65.431230	173.763679	0
...	...	...	...
495	59.976983	169.679741	1
496	66.423814	174.625574	0
497	53.604698	161.919208	2
498	50.433644	160.794875	2
499	60.224392	169.689709	1

500 rows × 3 columns

In [107]:
```
color=["yellow","red","green","blue"]
for i in range(0,4):
    data=dataset1[dataset1["prediction"]==i]
    plt.scatter(data["Height"],data["Weight"],c=color[i])
```



In [108]:
```
import pandas as pd
df=pd.read_csv("Clustering_gmm.csv")
df
```

Out[108]:

	Weight	Height
0	67.062924	176.086355
1	68.804094	178.388669
2	60.930863	170.284496
3	59.733843	168.691992
4	65.431230	173.763679
...	...	...
495	59.976983	169.679741
496	66.423814	174.625574
497	53.604698	161.919208
498	50.433644	160.794875
499	60.224392	169.689709

500 rows × 2 columns

In [109]:
```
from sklearn.mixture import GaussianMixture
model1=GaussianMixture(n_components=4)
model1.fit(df)
```

```
C:\Users\Neel\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1429: UserWarning: K
Means is known to have a memory leak on Windows with MKL, when there are less chunks than
available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=
2.
  warnings.warn(
```

Out[109]:

GaussianMixture (https://scikit-learn.org/1.5/modules/generated/sklearn.mixture.GaussianMixture.html)

GaussianMixture(n_components=4)

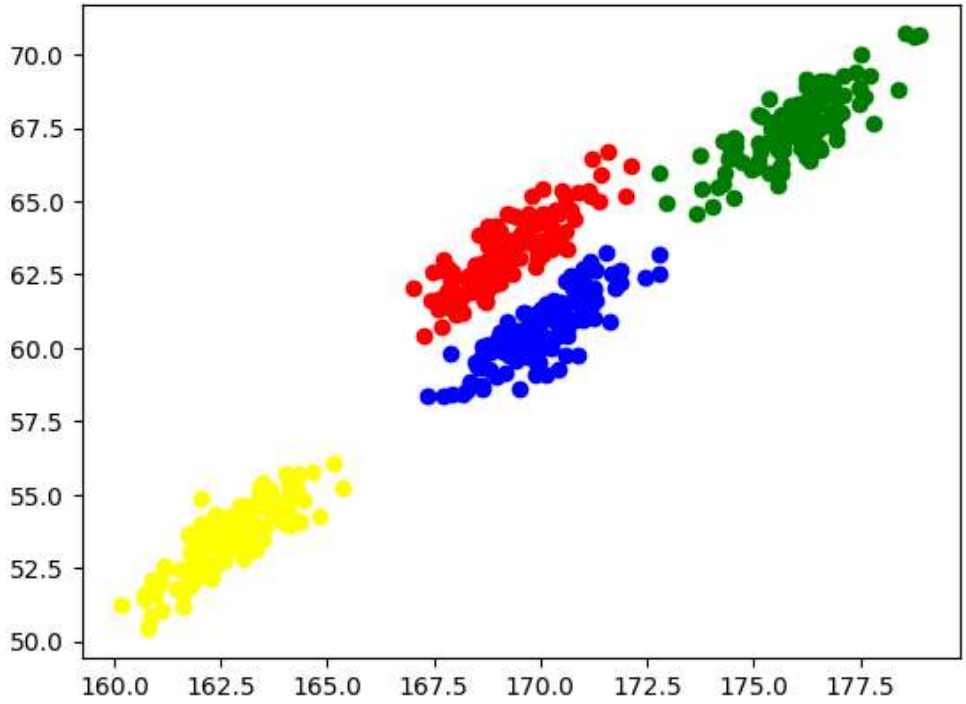
In [110]:
```
pred1=model1.predict(df)
dataset2=pd.DataFrame(df)
dataset2["Predictions"]=pred1
dataset2
```

Out[110]:

	Weight	Height	Predictions
0	67.062924	176.086355	2
1	68.804094	178.388669	2
2	60.930863	170.284496	3
3	59.733843	168.691992	3
4	65.431230	173.763679	2
...	...	...	...
495	59.976983	169.679741	3
496	66.423814	174.625574	2
497	53.604698	161.919208	0
498	50.433644	160.794875	0
499	60.224392	169.689709	3

500 rows × 3 columns

In [111]:
```
color=["yellow","red","green","blue"]
for i in range(0,4):
    data=dataset2[dataset2["Predictions"]==i]
    plt.scatter(data["Height"],data["Weight"],c=color[i])
```



ANN on breast cancer

In [112]:
```
from sklearn.datasets import load_breast_cancer
from keras.layers import Dense
from keras.models import Sequential
df=load_breast_cancer()
x=df.data
y=df.target
x.shape
```

Out[112]:
```
(569, 30)
```

In [113]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=4,test_size=0.2)
```

In [114]:
```
nnmodel=Sequential()
nnmodel.add(Dense(20,activation="relu",input_dim=30))
nnmodel.add(Dense(18,activation="relu"))
nnmodel.add(Dense(1,activation="sigmoid"))
```

```
C:\Users\Neel\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential model
s, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [115]:
```
nnmodel.compile(loss="binary_crossentropy",optimizer="adam",metrics=["accuracy"])
```

In [116]:
```
nnmodel.fit(xtrain,ytrain,batch_size=10,epochs=300)
```

```
46/46 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.5566 - loss: 17.5255
Epoch 2/300
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6062 - loss: 1.1514
Epoch 3/300
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8121 - loss: 0.4683
Epoch 4/300
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8765 - loss: 0.3463
Epoch 5/300
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8929 - loss: 0.3037
Epoch 6/300
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8982 - loss: 0.3141
Epoch 7/300
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8147 - loss: 0.4617
Epoch 8/300
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9095 - loss: 0.2333
Epoch 9/300
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8340 - loss: 0.4450
Epoch 10/300
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9315 - loss: 0.1861
Epoch 11/300
```

In [117]:
```
predictions=nnmodel.predict(xtest)
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```

In [118]:
```
pred=[]
for i in range(predictions.size):
    if(predictions[i]>0.5):
        pred.append(1)
    else:
        pred.append(0)
```

In [119]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
print(confusion_matrix(ytest,pred))
print(accuracy_score(ytest,pred))
```

```
[[34  0]
 [17 63]]
0.8508771929824561
```

Same on entire data

In [120]:
```
nnmodel1=Sequential()
nnmodel1.add(Dense(18,activation="relu",input_dim=30))
nnmodel1.add(Dense(12,activation="relu"))
nnmodel1.add(Dense(1,activation="sigmoid"))
```

```
C:\Users\Neel\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [121]:
```
nnmodel1.compile(loss="binary_crossentropy",
                 optimizer="adam",metrics=['accuracy'])
```

In [122]:
```
nnmodel1.fit(x,y,batch_size=10,epochs=300)
```

```
Epoch 2/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8131 - loss: 1.8480
Epoch 3/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8711 - loss: 0.7653
Epoch 4/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8610 - loss: 0.7645
Epoch 5/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9054 - loss: 0.5196
Epoch 6/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9143 - loss: 0.3172
Epoch 7/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8836 - loss: 0.3607
Epoch 8/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9065 - loss: 0.3057
Epoch 9/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9169 - loss: 0.2154
Epoch 10/300
57/57 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8797 - loss: 0.3785
Epoch 11/300
```

In [123]:
```
prediction=nnmodel1.predict(x)
```

```
18/18 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

In [124]:
```
pred1=[]
for i in range(prediction.size):
    if(prediction[i]>0.5):
        pred1.append(1)
    else:
        pred1.append(0)
```

In [125]:
```
from sklearn.metrics import accuracy_score,confusion_matrix
print(confusion_matrix(y,pred1))
print(accuracy_score(y,pred1))
```

```
[[208   4]
 [ 16 341]]
0.9648506151142355
```

ANN using Regression

In [126]:
```
import numpy as np
x=np.random.rand(1000,10)
y=np.random.rand(1000)
x.shape
```

Out[126]:
```
(1000, 10)
```

In [127]:
```
from keras.layers import Dense
from keras.models import Sequential
nnmodel2=Sequential()
nnmodel2.add(Dense(8,activation="relu",input_dim=10))
nnmodel2.add(Dense(6,activation="relu"))
nnmodel2.add(Dense(1,activation="linear"))
```

```
C:\Users\Neel\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [128]:
```
nnmodel2.compile(loss="mse",optimizer="adam",metrics=['mse'])
```

In [129]:
```
nnmodel2.fit(x,y,batch_size=10,epochs=300)
```

```
Epoch 2/300
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.1065 - mse: 0.1065
Epoch 3/300
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0929 - mse: 0.0929
Epoch 4/300
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0837 - mse: 0.0837
Epoch 5/300
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0881 - mse: 0.0881
Epoch 6/300
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0918 - mse: 0.0918
Epoch 7/300
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0914 - mse: 0.0914
Epoch 8/300
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0881 - mse: 0.0881
Epoch 9/300
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0784 - mse: 0.0784
Epoch 10/300
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.0888 - mse: 0.0888
Epoch 11/300
```

In [130]:
```
predict=nnmodel2.predict(x)
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

In [131]:
```
from sklearn.metrics import r2_score,mean_absolute_error
print(r2_score(y,predict))
print(mean_absolute_error(y,predict))
```

```
0.0953542764407127
0.23531676042354532
```

ANN using Multiclass

In [132]:
```
import pandas as pd
import keras as kr
import tensorflow as tf
df=pd.read_csv("flowers.csv")
df
```

Out[132]:

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

150 rows × 5 columns

In [133]:
```
from sklearn.preprocessing import LabelEncoder
lb=LabelEncoder()
df["species"]=lb.fit_transform(df["species"])
```

In [134]:
```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
y.shape
```

Out[134]: (150,)

In [135]:
```
encoded_y=kr.utils.to_categorical(y)
```

In [136]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,encoded_y,random_state=23,test_size=0.2)
xtrain.shape
```

Out[136]: (120, 4)

In [137]:
```
from keras.layers import Dense
from keras.models import Sequential
nnmodel3=Sequential()
nnmodel3.add(Dense(4,activation="relu",input_dim=4))
nnmodel3.add(Dense(2,activation="relu"))
nnmodel3.add(Dense(3,activation="softmax"))
```

```
C:\Users\Neel\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential model
s, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [138]:
```
nnmodel3.compile(loss="categorical_crossentropy",optimizer="adam",metrics=['accuracy'])
```

In [139]:
```
nnmodel3.fit(xtrain,ytrain,batch_size=10,epochs=300)
```

```
Epoch 2/300
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3543 - loss: 1.0987
Epoch 3/300
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3887 - loss: 1.0983
Epoch 4/300
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3477 - loss: 1.0983
Epoch 5/300
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3664 - loss: 1.0985
Epoch 6/300
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.2920 - loss: 1.0994
Epoch 7/300
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3968 - loss: 1.0980
Epoch 8/300
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3409 - loss: 1.0984
Epoch 9/300
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3259 - loss: 1.0989
Epoch 10/300
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.3513 - loss: 1.0989
Epoch 11/300
```

In [140]:
```
prediction1=nnmodel3.predict(xtest)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
```

In [141]:
```
for i in range(1,30,3):
    print(prediction1[i],ytest[i])
```

```
[0.31665173 0.34997615 0.33337203] [0. 0. 1.]
[0.31665173 0.34997615 0.33337203] [0. 0. 1.]
[0.31665173 0.34997615 0.33337203] [0. 0. 1.]
[0.31665173 0.34997615 0.33337203] [0. 1. 0.]
[0.31665173 0.34997615 0.33337203] [1. 0. 0.]
[0.31665173 0.34997615 0.33337203] [0. 1. 0.]
[0.31665173 0.34997615 0.33337203] [1. 0. 0.]
[0.31665173 0.34997615 0.33337203] [1. 0. 0.]
[0.31665173 0.34997615 0.33337203] [1. 0. 0.]
[0.31665173 0.34997615 0.33337203] [0. 1. 0.]
```

In [142]:
```
from sklearn.metrics import r2_score,mean_absolute_error
print(r2_score(ytest,prediction1))
print(mean_absolute_error(ytest,prediction1))
```

```
-0.021478868678458163
0.44592551986376444
```

GaussianNB on breast_cancer and iris

In [143]:
```
from sklearn.datasets import load_breast_cancer,load_iris
df=load_breast_cancer()
df1=load_iris()
x=df.data
y=df.target
x1=df1.data
y1=df1.target
```

In [144]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=32,test_size=0.2)
xtrain1,xtest1,ytrain1,ytest1=train_test_split(x1,y1,random_state=32,test_size=0.2)
```

In [145]:
```
from sklearn.naive_bayes import GaussianNB
GNB1=GaussianNB()
GNB2=GaussianNB()
GNB1.fit(xtrain,ytrain)
GNB2.fit(xtrain1,ytrain1)
```

Out[145]:
```
GaussianNB
GaussianNB()
```

In [146]:
```
pred_brest_cancer=GNB1.predict(xtest)
pred_iris=GNB2.predict(xtest1)
```

In [147]:
```
from sklearn.metrics import accuracy_score
print("Load_Breast_cancer",accuracy_score(ytest,pred_brest_cancer))
print("Load_iris",accuracy_score(ytest1,pred_iris))
```

```
Load_Breast_cancer 0.9122807017543859
Load_iris 0.9666666666666667
```

Bagging classifier

In [148]:
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
import pandas as pd
df=load_iris()
df1=pd.DataFrame(data=df.data,columns=df.feature_names)
df1['species']=df.target
df1
```

Out[148]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
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
...	...	...	...	...	...
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

150 rows × 5 columns

In [149]:
```
x=df1.iloc[:,:-1]
y=df1.iloc[:,-1]
```

In [150]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=21,test_size=0.2)
```

In [151]:
```
DT=DecisionTreeClassifier()
DT.fit(xtrain,ytrain)
pred=DT.predict(xtest)
```

In [152]:
```
RF=RandomForestClassifier()
RF.fit(xtrain,ytrain)
pred1=RF.predict(xtest)
```

In [153]:
```
BC=BaggingClassifier()
BC.fit(xtrain,ytrain)
pred2=BC.predict(xtest)
```

In [154]:
```
from sklearn.metrics import accuracy_score
print("Accuracy using Decision Tree =",accuracy_score(ytest,pred))
print("Accuracy using Random Forest =",accuracy_score(ytest,pred1))
print("Accuracy using Bagging Classifier =",accuracy_score(ytest,pred2))
```

```
Accuracy using Decision Tree = 0.9333333333333333
Accuracy using Random Forest = 0.9333333333333333
Accuracy using Bagging Classifier = 0.9333333333333333
```

Bernoulli and multinomial

In [155]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB,BernoulliNB
df=fetch_20newsgroups(subset="all")
```

In [156]:
```
x=df.data
x1=CountVectorizer(binary=True)
x2=CountVectorizer(binary=False)
vector_x1=x1.fit_transform(x)
vector_x2=x2.fit_transform(x)
```

In [157]:
```
y=df.target
```

In [158]:
```
from sklearn.model_selection import train_test_split
xtrain1,xtest1,ytrain1,ytest1=train_test_split(vector_x1,y,random_state=21,test_size=0.2)
xtrain2,xtest2,ytrain2,ytest2=train_test_split(vector_x2,y,random_state=21,test_size=0.2)
```

In [159]:
```
Bernoulli=BernoulliNB()
Bernoulli.fit(xtrain1,ytrain1)
pred=Bernoulli.predict(xtest1)
```

In [160]:
```
Multinomial=MultinomialNB()
Multinomial.fit(xtrain2,ytrain2)
pred2=Multinomial.predict(xtest2)
```

In [161]:
```
from sklearn.metrics import accuracy_score
print("Accuracy using BernoulliNB =",accuracy_score(ytest1,pred))
print("Accuracy using MultinomialNB =",accuracy_score(ytest2,pred2))
```

```
Accuracy using BernoulliNB = 0.7082228116710876
Accuracy using MultinomialNB = 0.8511936339522547
```

Ada-Boosting

In [162]:
```
from sklearn.datasets import load_iris
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
```

In [163]:
```
data=load_iris()
x=data.data
y=data.target
```

In [164]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=3,test_size=0.2)
```

In [165]:
```
adb=AdaBoostClassifier(n_estimators=50)
adb.fit(xtrain,ytrain)
```

```
C:\Users\Neel\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

Out[165]:

AdaBoostClassifier (https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.AdaBoostClassifier.html)

AdaBoostClassifier()

In [166]:
```
pred=adb.predict(xtest)
from sklearn.metrics import accuracy_score
print(accuracy_score(ytest,pred))
```

```
0.9666666666666667
```

Gradient Boosting

In [167]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import LabelEncoder
import pandas as pd
```

In [168]:
```
df=pd.read_csv("titanic.csv")
df.head(5)
```

Out[168]:

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

In [169]:
```
df.describe()
```

Out[169]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [170]:
```
df.drop(columns={"Name","SibSp","Parch","Ticket","Cabin","PassengerId"},inplace=True)
```

In [171]:
```
df
```

Out[171]:

	Survived	Pclass	Sex	Age	Fare	Embarked
0	0	3	male	22.0	7.2500	S
1	1	1	female	38.0	71.2833	C
2	1	3	female	26.0	7.9250	S
3	1	1	female	35.0	53.1000	S
4	0	3	male	35.0	8.0500	S
...	...	...	...	...	...	...
886	0	2	male	27.0	13.0000	S
887	1	1	female	19.0	30.0000	S
888	0	3	female	NaN	23.4500	S
889	1	1	male	26.0	30.0000	C
890	0	3	male	32.0	7.7500	Q

891 rows × 6 columns

In [172]:
```
df.drop(columns={"Age"},inplace=True)
lb=LabelEncoder()
df["Sex"]=lb.fit_transform(df["Sex"])
df["Embarked"]=lb.fit_transform(df["Embarked"])
```

In [173]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
mms.fit_transform(df)
```

Out[173]:
```
array([[0.        , 1.        , 1.        , 0.01415106, 0.66666667],
       [1.        , 0.        , 0.        , 0.13913574, 0.        ],
       [1.        , 1.        , 0.        , 0.01546857, 0.66666667],
       ...,
       [0.        , 1.        , 0.        , 0.04577135, 0.66666667],
       [1.        , 0.        , 1.        , 0.0585561 , 0.        ],
       [0.        , 1.        , 1.        , 0.01512699, 0.33333333]])
```

In [174]:
```
y=df.iloc[:,0]
x=df.iloc[:,1:5]
```

In [175]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,random_state=6,test_size=0.2)
```

In [176]:
```
learning_rate=[0.05,0.075,0.1,0.25,0.5,0.75,1]
for lr in learning_rate:
    gb=GradientBoostingClassifier(max_depth=2,learning_rate=lr,max_features=2,
                                  n_estimators=20,random_state=5)
    gb.fit(xtrain,ytrain)
    print(f"Learning rate:{lr:.3f}\t Accuracy Score(Training):{gb.score(xtrain,ytrain):.3f
```

```
Learning rate:0.050      Accuracy Score(Training):0.785  Accuracy Score(Training):0.810
Learning rate:0.075      Accuracy Score(Training):0.806  Accuracy Score(Training):0.832
Learning rate:0.100      Accuracy Score(Training):0.791  Accuracy Score(Training):0.810
Learning rate:0.250      Accuracy Score(Training):0.812  Accuracy Score(Training):0.838
Learning rate:0.500      Accuracy Score(Training):0.815  Accuracy Score(Training):0.855
Learning rate:0.750      Accuracy Score(Training):0.829  Accuracy Score(Training):0.832
Learning rate:1.000      Accuracy Score(Training):0.819  Accuracy Score(Training):0.866
```

In [177]:
```
gb=GradientBoostingClassifier(max_depth=2,learning_rate=0.75,max_features=2,n_estimators=2
gb.fit(xtrain,ytrain)
```

Out[177]:
```
GradientBoostingClassifier

GradientBoostingClassifier(learning_rate=0.75, max_depth=2, max_features=2,
                           n_estimators=20, random_state=5)
```

In [178]:
```
pred=gb.predict(xtest)
```

In [179]:
```
from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(ytest,pred))
confusion_matrix(ytest,pred)
```

```
              precision    recall  f1-score   support

           0       0.83      0.92      0.87       113
           1       0.83      0.68      0.75        66

    accuracy                           0.83       179
   macro avg       0.83      0.80      0.81       179
weighted avg       0.83      0.83      0.83       179
```

Out[179]:
```
array([[104,   9],
       [ 21,  45]], dtype=int64)
```

Stacking

In [180]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import StackingRegressor
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score
```

In [181]:
```
x,y=make_regression(n_samples=1000,n_features=20,random_state=23)
```

In [182]:
```
def get_stacking():
    lev0=[] #base model
    lev0.append(("knn",KNeighborsRegressor()))
    lev0.append(("svr",SVR()))
    lev1=LinearRegression() #Meta model
    model=StackingRegressor(estimators=lev0,final_estimator=lev1)
    return model
```

In [183]:
```
def get_model():
    models=dict()
    models["knn"]=KNeighborsRegressor()
    models["svr"]=SVR()
    models["dt"]=DecisionTreeRegressor()
    models["stacking"]=get_stacking()
    return models
```

In [184]:
```
def evaluate_model(model,x,y):
    cv=RepeatedKFold(n_repeats=3,n_splits=10,random_state=9)
    score=cross_val_score(model,x,y,scoring="neg_mean_absolute_error",cv=cv)
    return score
```

In [185]:
```
import numpy as np
models=get_model()
results,names=list(),list()
for name,model in models.items():
    score=evaluate_model(model,x,y)
    names.append(model)
    results.append(score)
    print(name,np.mean(score))
```

```
knn -98.62691545729396
svr -152.18876813622038
dt -117.67273472642043
stacking -61.406801665541295
```

Genetic Algorithm

In [186]:
```
pip install pygad
```

```
Requirement already satisfied: pygad in c:\users\neel\anaconda3\lib\site-packages (3.
3.1)
Requirement already satisfied: cloudpickle in c:\users\neel\anaconda3\lib\site-package
s (from pygad) (2.2.1)
Requirement already satisfied: matplotlib in c:\users\neel\anaconda3\lib\site-packages
(from pygad) (3.7.1)
Requirement already satisfied: numpy in c:\users\neel\anaconda3\lib\site-packages (fro
m pygad) (1.24.3)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\neel\anaconda3\lib\site-pa
ckages (from matplotlib->pygad) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\neel\anaconda3\lib\site-packag
es (from matplotlib->pygad) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\neel\anaconda3\lib\site-p
ackages (from matplotlib->pygad) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\neel\anaconda3\lib\site-p
ackages (from matplotlib->pygad) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\neel\anaconda3\lib\site-pac
kages (from matplotlib->pygad) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\neel\anaconda3\lib\site-packa
```

In [187]:
```
import pygad
import numpy as np
x=[4,-2,3.5,5,-11,-4,7]
desired_output=44
```

In [188]:
```
def fitness_function(ga_instance,solution,solution_idx):
    output=np.sum(solution*x)
    fitness=1.0/np.abs(output-desired_output)
    return fitness
```

In [189]:
```
ga_instance=pygad.GA(num_generations=50,
                     crossover_type="single_point",
                     init_range_high=5,
                     init_range_low=-2,
                     num_genes=len(x),
                     fitness_func=fitness_function,
                     mutation_type="random",
                     mutation_percent_genes=10,
                     keep_parents=1,sol_per_pop=8,
                     parent_selection_type="sss",
                     num_parents_mating=8,)
```

```
C:\Users\Neel\anaconda3\Lib\site-packages\pygad\pygad.py:748: UserWarning: The percentage
of genes to mutate (mutation_percent_genes=10) resulted in selecting (0) genes. The numbe
r of genes to mutate is set to 1 (mutation_num_genes=1).
If you do not want to mutate any gene, please set mutation_type=None.
  warnings.warn(f"The percentage of genes to mutate (mutation_percent_genes={mutation_per
cent_genes}) resulted in selecting ({mutation_num_genes}) genes. The number of genes to m
utate is set to 1 (mutation_num_genes=1).\nIf you do not want to mutate any gene, please
set mutation_type=None.")
C:\Users\Neel\anaconda3\Lib\site-packages\pygad\pygad.py:1139: UserWarning: The 'delay_af
ter_gen' parameter is deprecated starting from PyGAD 3.3.0. To delay or pause the evoluti
on after each generation, assign a callback function/method to the 'on_generation' parame
ter to adds some time delay.
  warnings.warn("The 'delay_after_gen' parameter is deprecated starting from PyGAD 3.3.0.
To delay or pause the evolution after each generation, assign a callback function/method
to the 'on_generation' parameter to adds some time delay.")
```

In [190]:
```
ga_instance.run()
```

In [191]:
```
solution,solution_fitness,solution_idx=ga_instance.best_solution()
```

In [192]:
```
print("solution",solution)
print("fitness",solution_fitness)
print("Solution IDX",solution_idx)
```

```
solution [ 2.94717861 -0.9442382   0.75406583  2.67380395 -1.4319947   5.39504285
  2.87223405]
fitness 26.91759750863977
Solution IDX 0
```

Content Based Filtering

In [193]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

In [194]:
```
df=pd.read_csv("movies1.csv")
df
```

Out[194]:

	index	budget	genres	homepage	id	keywor
0	0	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	19995	culture clash futu space war spa colony sc
1	1	300000000	Adventure Fantasy Action	http://disney.go.com/disneypictures/pirates/	285	ocean drug abu exotic island ea india trac
2	2	245000000	Action Adventure Crime	http://www.sonypictures.com/movies/spectre/	206647	spy based on no secret agent sequ

In [195]:
```
features=["genres","keywords","original_language","title","cast","director"]
for feature in features:
    df[feature]=df[feature].fillna(" ")
```

In [196]:
```
def combined_features(row):
    return row["title"]+","+row["genres"]+","+row["keywords"]+","+row["original_language"]
df["combined_features"]=df.apply(combined_features,axis=1)
```

In [197]:
```
df["combined_features"]
```

Out[197]:
```
0       Avatar,Action Adventure Fantasy Science Fictio...
1       Pirates of the Caribbean: At World's End,Adven...
2       Spectre,Action Adventure Crime,spy based on no...
3       The Dark Knight Rises,Action Crime Drama Thril...
4       John Carter,Action Adventure Science Fiction,b...
                              ...
4798    El Mariachi,Action Crime Thriller,united state...
4799    Newlyweds,Comedy Romance, ,en,Edward Burns Ker...
4800    Signed, Sealed, Delivered,Comedy Drama Romance...
4801    Shanghai Calling, , ,en,Daniel Henney Eliza Co...
4802    My Date with Drew,Documentary,obsession camcor...
Name: combined_features, Length: 4803, dtype: object
```

In [198]:
```
tfidf=TfidfVectorizer()
tfidf_vector=tfidf.fit_transform(df["combined_features"])
tfidf_vector.toarray()
```

Out[198]:
```
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.]])
```

In [199]:
```
tfidf_vector.shape
```

Out[199]:
```
(4803, 17502)
```

In [200]:
```
cosine_sim=cosine_similarity(tfidf_vector)
cosine_sim.shape
```

Out[200]:
```
(4803, 4803)
```

In [201]:
```
movies=input("Enter movie name....")
def get_index(mm):
    return df[df.title==mm].index[0]
mi=get_index(movies)
mi
```

```
Enter movie name....Avatar
```

Out[201]:
```
0
```

In [202]:
```
sm=list(enumerate(cosine_sim[mi]))
```

In [203]:
```
sorted_sm=sorted(sm,key=lambda x:x[1],reverse=True)
```

In [204]:
```
def get_info(index):
    return df[df.index==index]["title"].values[0]+","+df[df.index==index]["cast"].values[0
i=0
for movie in sorted_sm:
    print(get_info(movie[0]))
    i+=1
    if i>10:
        break;
```

```
Avatar,Sam Worthington Zoe Saldana Sigourney Weaver Stephen Lang Michelle Rodriguez
Guardians of the Galaxy,Chris Pratt Zoe Saldana Dave Bautista Vin Diesel Bradley Cooper
Aliens,Sigourney Weaver Michael Biehn James Remar Paul Reiser Lance Henriksen
Alien,Tom Skerritt Sigourney Weaver Veronica Cartwright Harry Dean Stanton John Hurt
Galaxy Quest,Tim Allen Sigourney Weaver Alan Rickman Tony Shalhoub Sam Rockwell
Star Trek Into Darkness,Chris Pine Zachary Quinto Zoe Saldana Karl Urban Simon Pegg
Star Trek Beyond,Chris Pine Zachary Quinto Karl Urban Simon Pegg Zoe Saldana
Jason X,Kane Hodder Lexa Doig Chuck Campbell Lisa Ryder David Cronenberg
Space Dogs,Anna Bolshova Evgeny Mironov Sergey Garmash Aleksandr Bashirov Elena Yakovleva
Alien³,Sigourney Weaver Charles S. Dutton Charles Dance Pete Postlethwaite Ralph Brown
Gravity,Sandra Bullock George Clooney Ed Harris Orto Ignatiussen Phaldut Sharma
```

In [ ]: