### 1) Linear and Logistic regression

### Linear Regression

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_squared\_error  
from sklearn.model\_selection import train\_test\_split

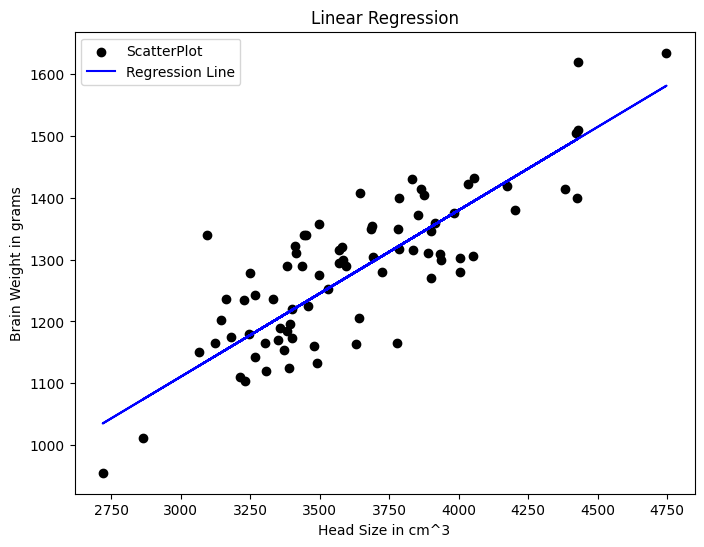
data = pd.read\_csv('headbrain.csv')

X = data['Head Size(cm^3)']  
y = data['Brain Weight(grams)']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.33, random\_state = 42)  
  
model = LinearRegression()  
  
model.fit(np.expand\_dims(X\_train, axis = 1), np.expand\_dims(y\_train, axis = 1))  
  
prediction = model.predict(np.expand\_dims(X\_test, axis = 1))

plt.figure(figsize = (8,6))  
plt.scatter(X\_test, y\_test, color = 'black', label = 'ScatterPlot')  
plt.plot(X\_test, prediction, color = 'blue', label = 'Regression Line')  
plt.title('Linear Regression')  
plt.xlabel('Head Size in cm^3')  
plt.ylabel('Brain Weight in grams')  
plt.legend()

<matplotlib.legend.Legend at 0x1a437878410>



### Logistic Regression

import pandas as pd  
import numpy as np  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score,classification\_report,confusion\_matrix

data = pd.read\_csv('diabetes.csv')

data

Pregnancies Glucose BloodPressure SkinThickness Insulin BMI \  
0 4 183 0 0 0 28.4   
1 5 162 104 0 0 37.7   
2 2 197 70 99 0 34.7   
3 13 158 114 0 0 42.3   
4 0 162 76 56 100 53.2   
.. ... ... ... ... ... ...   
503 7 179 95 31 0 34.2   
504 0 113 76 0 0 33.3   
505 3 128 72 25 190 32.4   
506 1 119 88 41 170 45.3   
507 3 84 72 32 0 37.2   
  
 DiabetesPedigreeFunction Age Outcome   
0 0.212 36 1   
1 0.151 52 1   
2 0.575 62 1   
3 0.257 44 1   
4 0.759 25 1   
.. ... ... ...   
503 0.164 60 0   
504 0.278 23 1   
505 0.549 27 1   
506 0.507 26 0   
507 0.267 28 0   
  
[508 rows x 9 columns]

X = data.drop('Outcome', axis = 1)

y = data['Outcome']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.33)  
  
model = LogisticRegression()  
  
model.fit(X\_train, y\_train)  
  
predictions = model.predict(X\_test)

C:\Users\Gowtham R\AppData\Roaming\Python\Python311\site-packages\sklearn\linear\_model\\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.  
  
Increase the number of iterations (max\_iter) or scale the data as shown in:  
 https://scikit-learn.org/stable/modules/preprocessing.html  
Please also refer to the documentation for alternative solver options:  
 https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression  
 n\_iter\_i = \_check\_optimize\_result(

print("Accuracy Score : ", accuracy\_score(y\_test, predictions))

Accuracy Score : 0.7083333333333334

print("Classification Report : \n", classification\_report(y\_test, predictions))

Classification Report :   
 precision recall f1-score support  
  
 0 0.76 0.83 0.79 112  
 1 0.58 0.46 0.51 56  
  
 accuracy 0.71 168  
 macro avg 0.67 0.65 0.65 168  
weighted avg 0.70 0.71 0.70 168

print("Confusion Matrix : \n", confusion\_matrix(y\_test, predictions))

Confusion Matrix :   
 [[93 19]  
 [30 26]]

#### with titanic dataset

import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.preprocessing import StandardScaler  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score,classification\_report,confusion\_matrix

data = pd.read\_csv('titanic.csv')

data

PassengerId Survived Pclass Lname \  
0 1 0 3 Braund   
1 2 1 1 Cumings   
2 3 1 3 Heikkinen   
3 4 1 1 Futrelle   
4 5 0 3 Allen   
.. ... ... ... ...   
151 152 1 1 Pears   
152 153 0 3 Meo   
153 154 0 3 van Billiard   
154 155 0 3 Olsen   
155 156 0 1 Williams   
  
 Name Sex Age SibSp Parch \  
0 Mr. Owen Harris male 22.0 1 0   
1 Mrs. John Bradley (Florence Briggs Thayer) female 38.0 1 0   
2 Miss. Laina female 26.0 0 0   
3 Mrs. Jacques Heath (Lily May Peel) female 35.0 1 0   
4 Mr. William Henry male 35.0 0 0   
.. ... ... ... ... ...   
151 Mrs. Thomas (Edith Wearne) female 22.0 1 0   
152 Mr. Alfonzo male 55.5 0 0   
153 Mr. Austin Blyler male 40.5 0 2   
154 Mr. Ole Martin male NaN 0 0   
155 Mr. Charles Duane male 51.0 0 1   
  
 Ticket Fare Cabin Embarked   
0 A/5 21171 7.2500 NaN S   
1 PC 17599 71.2833 C85 C   
2 STON/O2. 3101282 7.9250 NaN S   
3 113803 53.1000 C123 S   
4 373450 8.0500 NaN S   
.. ... ... ... ...   
151 113776 66.6000 C2 S   
152 A.5. 11206 8.0500 NaN S   
153 A/5. 851 14.5000 NaN S   
154 Fa 265302 7.3125 NaN S   
155 PC 17597 61.3792 NaN C   
  
[156 rows x 13 columns]

data['Age'].fillna(data['Age'].mean(),inplace = True)  
data['Embarked'].fillna(data['Embarked'].mode()[0],inplace=True)

data = pd.get\_dummies(data, columns=['Sex','Embarked'],drop\_first=True)

X = data[['Pclass', 'Age', 'SibSp', 'Parch', 'Fare', 'Sex\_male', 'Embarked\_Q', 'Embarked\_S']]  
y = data['Survived']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)

model = LogisticRegression()  
model.fit(X\_train, y\_train)

LogisticRegression()

predictions = model.predict(X\_test)

print(f'Accuracy:',accuracy\_score(y\_test, predictions))  
print('Classification Report',classification\_report(y\_test, predictions))  
print('Confusion Matrix \n',confusion\_matrix(y\_test,predictions))

Accuracy: 0.71875  
Classification Report precision recall f1-score support  
  
 0 0.77 0.81 0.79 21  
 1 0.60 0.55 0.57 11  
  
 accuracy 0.72 32  
 macro avg 0.69 0.68 0.68 32  
weighted avg 0.71 0.72 0.72 32  
  
Confusion Matrix   
 [[17 4]  
 [ 5 6]]

### (2) Univariate and Multivariate Normal Distribution

### Univariate

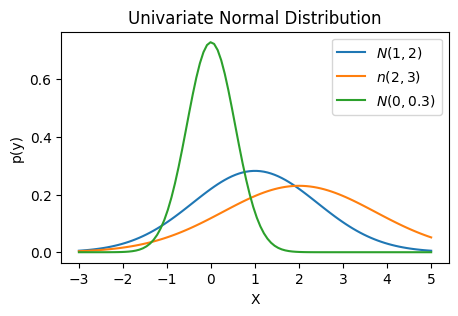
import numpy as np  
import matplotlib.pyplot as plt

def univariate\_normal(x, mean, variance):  
 return((1 / np.sqrt(2 \* np.pi \* variance)) \* np.exp(-(x - mean)\*\*2 / (2 \* variance)))

x = np.linspace(-3, 5, 100)

plt.figure(figsize = (5, 3))  
plt.plot(x, univariate\_normal(x, mean = 1, variance = 2), label = "$N(1, 2)$")  
plt.plot(x, univariate\_normal(x, mean = 2, variance = 3), label = "$n(2, 3)$")  
plt.plot(x, univariate\_normal(x, mean = 0, variance = 0.3), label = "$N(0, 0.3)$")  
plt.title("Univariate Normal Distribution")  
plt.xlabel("X")  
plt.ylabel("p(y)")  
plt.legend()

<matplotlib.legend.Legend at 0x1a437ae4110>



### Multivariate

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.stats import multivariate\_normal  
from mpl\_toolkits.mplot3d import Axes3D

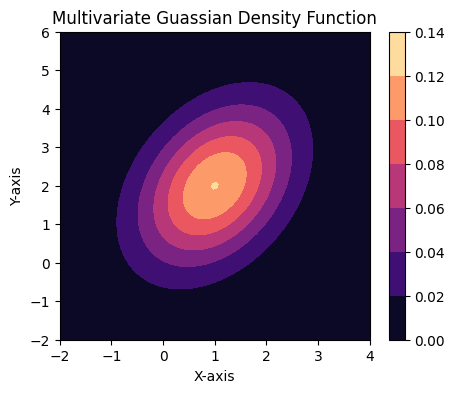
mean = np.array([1,2])  
covariance\_matrix = np.array([[1, 0.5],[0.5, 2]]) #Diagonal cov-matrix

x, y = np.meshgrid(np.linspace(-2, 4, 100), np.linspace(-2, 6, 100))  
j = np.dstack((x,y))

pdf = multivariate\_normal(mean, covariance\_matrix)  
z = pdf.pdf(j)

plt.figure(figsize = (5,4))  
plt.contourf(x, y, z, cmap = 'magma')  
plt.colorbar()  
plt.title('Multivariate Guassian Density Function')  
plt.xlabel('X-axis')  
plt.ylabel('Y-axis')

Text(0, 0.5, 'Y-axis')



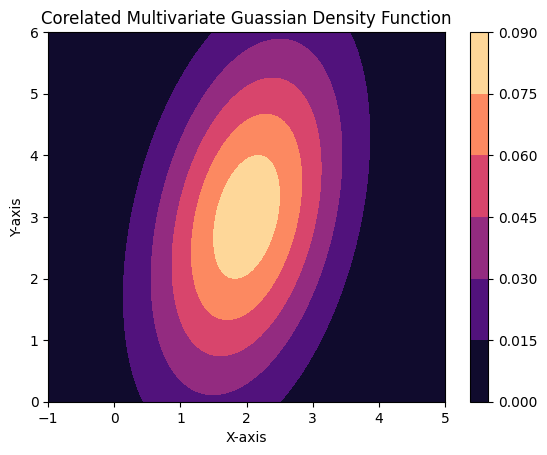
mean = np.array([2, 3])  
covariance\_matrix = np.array([[1, 0.7],[0.7, 4]]) #non-diagonal cov-matrix

x, y = np.meshgrid(np.linspace(-1, 5, 100), np.linspace(0, 6, 100))  
j = np.dstack((x, y))

pdf = multivariate\_normal(mean, covariance\_matrix)  
z = pdf.pdf(j)

plt.contourf(x, y, z, cmap = 'magma')  
plt.colorbar()  
plt.title('Corelated Multivariate Guassian Density Function')  
plt.xlabel('X-axis')  
plt.ylabel('Y-axis')

Text(0, 0.5, 'Y-axis')



### (3) PCA

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.decomposition import PCA  
from sklearn.preprocessing import StandardScaler

data = pd.read\_csv('Iris.csv')

data

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm \  
0 1 5.1 3.5 1.4 0.2   
1 2 4.9 3.0 1.4 0.2   
2 3 4.7 3.2 1.3 0.2   
3 4 4.6 3.1 1.5 0.2   
4 5 5.0 3.6 1.4 0.2   
.. ... ... ... ... ...   
145 146 6.7 3.0 5.2 2.3   
146 147 6.3 2.5 5.0 1.9   
147 148 6.5 3.0 5.2 2.0   
148 149 6.2 3.4 5.4 2.3   
149 150 5.9 3.0 5.1 1.8   
  
 Species   
0 Iris-setosa   
1 Iris-setosa   
2 Iris-setosa   
3 Iris-setosa   
4 Iris-setosa   
.. ...   
145 Iris-virginica   
146 Iris-virginica   
147 Iris-virginica   
148 Iris-virginica   
149 Iris-virginica   
  
[150 rows x 6 columns]

X = data[data.columns[1:-1]]

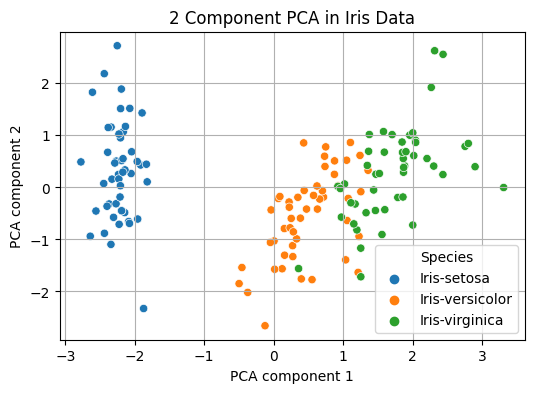
X

SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm  
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 x 4 columns]

scaled\_X = StandardScaler().fit\_transform(X)

reduced\_X = PCA(n\_components = 2).fit\_transform(scaled\_X)

plt.figure(figsize=(6,4))  
sns.scatterplot(x = np.ndarray.ravel(reduced\_X[:,:1]), y = np.ndarray.ravel(reduced\_X[:,1:]), hue = data['Species'])  
plt.title('2 Component PCA in Iris Data')  
plt.xlabel('PCA component 1')  
plt.ylabel('PCA component 2')  
plt.grid()



#### PCA for breast\_cancer dataset

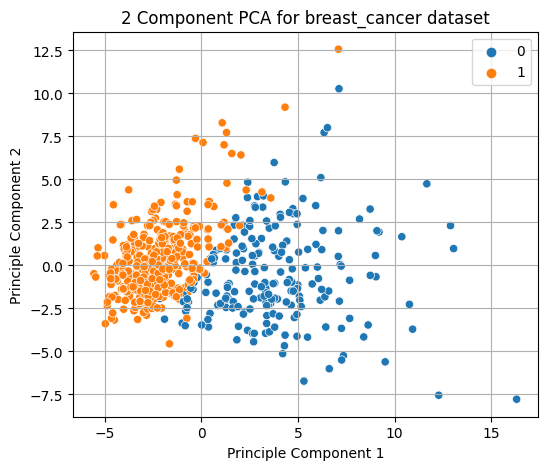
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.decomposition import PCA  
from sklearn.preprocessing import StandardScaler  
from sklearn.datasets import load\_breast\_cancer

data = load\_breast\_cancer()

X = data['data']

scaled\_X = StandardScaler().fit\_transform(X)  
reduced\_X = PCA(n\_components = 2).fit\_transform(scaled\_X)

plt.figure(figsize=(6,5))  
sns.scatterplot(x = np.ndarray.ravel(reduced\_X[:,:1]), y = np.ndarray.ravel(reduced\_X[:,1:]), hue = data['target'])  
plt.title("2 Component PCA for breast\_cancer dataset")  
plt.xlabel("Principle Component 1")  
plt.ylabel("Principle Component 2")  
plt.grid()  
plt.show()



### 4) KMeans

import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.datasets import load\_iris  
from sklearn.cluster import KMeans

iris = load\_iris()

X = iris.data

y = iris.target

num\_clusters = 3

kmeans = KMeans(n\_clusters = num\_clusters,random\_state = 42)

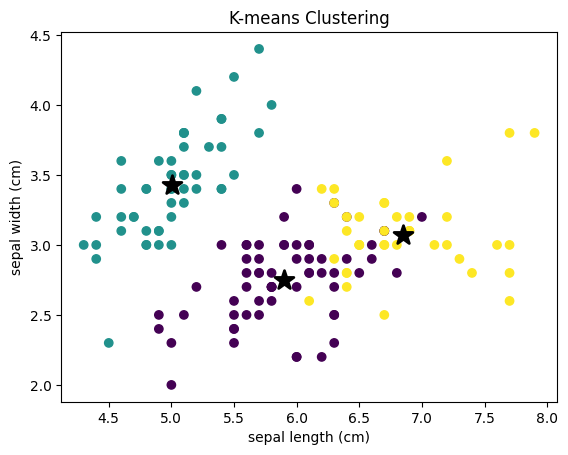
kmeans.fit(X)

C:\Users\Gowtham R\AppData\Roaming\Python\Python311\site-packages\sklearn\cluster\\_kmeans.py:870: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning  
 warnings.warn(

KMeans(n\_clusters=3, random\_state=42)

cluster\_labels= kmeans.labels\_  
centroids = kmeans.cluster\_centers\_

plt.scatter(X[:, 0], X[:, 1], c=cluster\_labels, cmap='viridis')  
plt.scatter(centroids[:, 0], centroids[:, 1], marker='\*', s=200, linewidths=2, color='black')  
plt.xlabel(iris.feature\_names[0])  
plt.ylabel(iris.feature\_names[1])  
plt.title('K-means Clustering')  
plt.show()



### GMM

import pandas as pd  
import matplotlib.pyplot as plt  
from sklearn.datasets import load\_iris  
from sklearn.mixture import GaussianMixture

iris = load\_iris()

X = iris.data

num\_components = 3

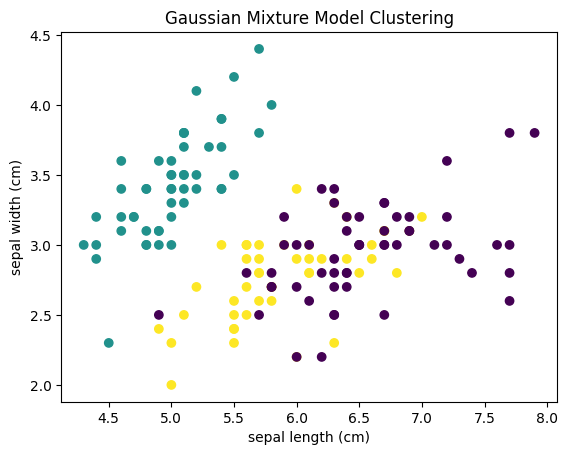
gmm = GaussianMixture(n\_components = num\_components,random\_state=42)

gmm.fit(X)

GaussianMixture(n\_components=3, random\_state=42)

cluster\_labels = gmm.predict(X)

plt.scatter(X[:, 0], X[:, 1], c = cluster\_labels, cmap = 'viridis')  
plt.xlabel(iris.feature\_names[0])  
plt.ylabel(iris.feature\_names[1])  
plt.title('Gaussian Mixture Model Clustering')  
plt.show()



### 5) BPNN

from sklearn.datasets import make\_circles  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns

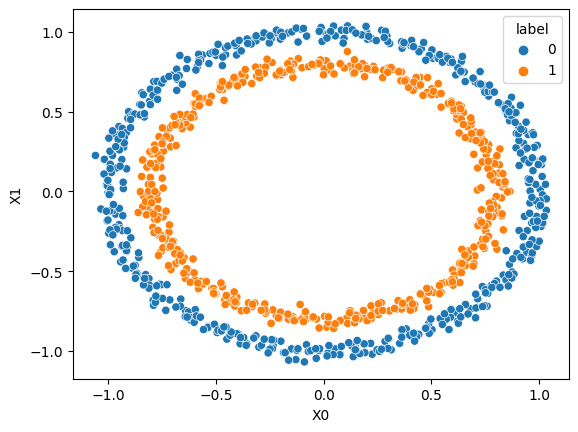
samples = 1000

X, y = make\_circles(samples,random\_state = 42, noise = 0.03)

circles = {"X0":X[:,0],"X1":X[:,1],"label" : y}

sns.scatterplot(data = circles, x="X0", y="X1", hue = 'label')

<Axes: xlabel='X0', ylabel='X1'>



from sklearn.model\_selection import train\_test\_split  
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
from tensorflow.keras.optimizers import Adam

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,random\_state = 42, test\_size = 0.3)

model = tf.keras.Sequential([  
 tf.keras.layers.Dense(16,activation="relu",input\_shape=(2,)),  
 tf.keras.layers.Dense(8, activation="relu"),  
 tf.keras.layers.Dense(1,activation="sigmoid")  
])

model.compile (loss="binary\_crossentropy", optimizer = tf.keras.optimizers.Adam(learning\_rate=0.2),metrics=['accuracy'])

model.summary()

Model: "sequential"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 Layer (type) Output Shape Param #   
=================================================================  
 dense (Dense) (None, 16) 48   
   
 dense\_1 (Dense) (None, 8) 136   
   
 dense\_2 (Dense) (None, 1) 9   
   
=================================================================  
Total params: 193 (772.00 Byte)  
Trainable params: 193 (772.00 Byte)  
Non-trainable params: 0 (0.00 Byte)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

history = model.fit(X\_train,y\_train,epochs=5)

Epoch 1/5  
22/22 [==============================] - 1s 3ms/step - loss: 0.6889 - accuracy: 0.5657  
Epoch 2/5  
22/22 [==============================] - 0s 2ms/step - loss: 0.6446 - accuracy: 0.6414  
Epoch 3/5  
22/22 [==============================] - 0s 2ms/step - loss: 0.6369 - accuracy: 0.6129  
Epoch 4/5  
22/22 [==============================] - 0s 2ms/step - loss: 0.6207 - accuracy: 0.6771  
Epoch 5/5  
22/22 [==============================] - 0s 2ms/step - loss: 0.6731 - accuracy: 0.5400

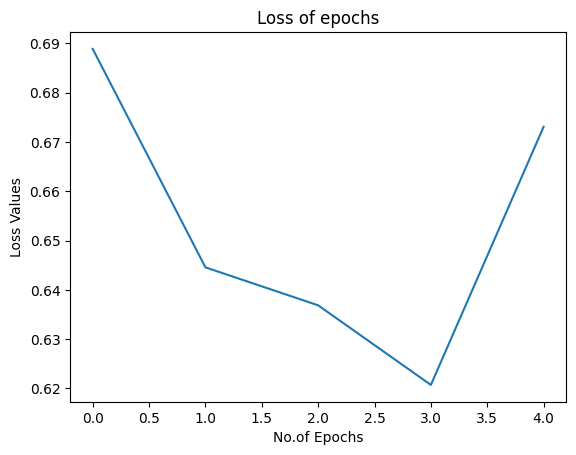
model.evaluate(X\_test,y\_test)

10/10 [==============================] - 0s 3ms/step - loss: 0.6406 - accuracy: 0.5200

[0.6405813694000244, 0.5199999809265137]

plt.plot(history.history['loss'])  
plt.title('Loss of epochs')  
plt.xlabel('No.of Epochs')  
plt.ylabel('Loss Values')

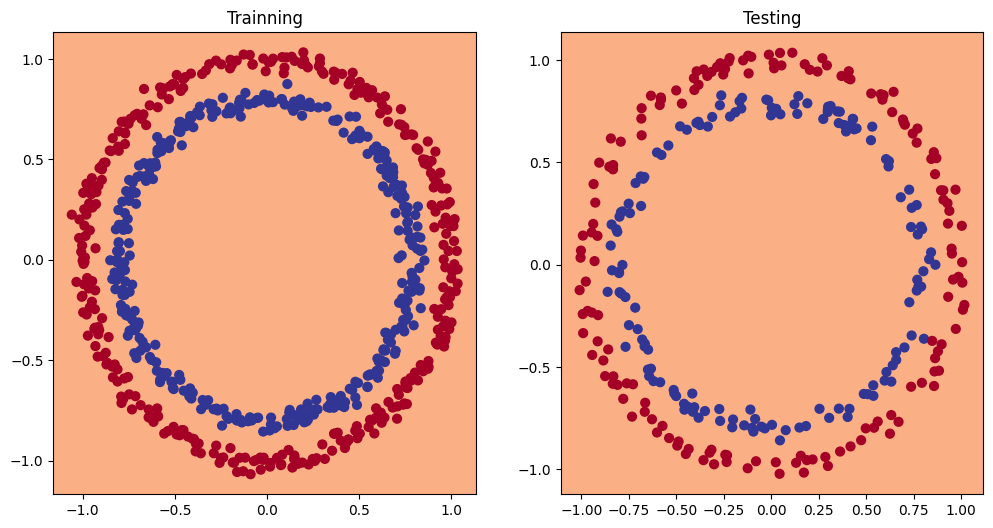
Text(0, 0.5, 'Loss Values')



def plotq(model,X,y):  
 x\_min, x\_max = X[:,0].min()-0.1,X[:,0].max()+0.1  
 y\_min, y\_max = X[:,1].min()-0.1,X[:,1].max()+0.1  
 xx,yy = np.meshgrid(np.linspace(x\_min,x\_max,100),np.linspace(y\_min,y\_max,100))  
 x\_in = np.c\_[xx.ravel(),yy.ravel()]  
 y\_pred = model.predict(x\_in)  
 if len(y\_pred[0])>1:  
 print('Doing Multiple Classification')  
 y\_pred = argmax(y\_pred,axis=1).reshape(xx.shape)  
 else:  
 print('Doing Binary')  
 y\_pred = np.round(y\_pred).reshape(xx.shape)  
   
 plt.contourf(xx,yy,y\_pred,cmap=plt.cm.RdYlBu,alpha=0.7)  
 plt.scatter(X[:,0],X[:,1],c=y,s=40,cmap=plt.cm.RdYlBu)  
 plt.xlim(xx.min(),xx.max())  
 plt.ylim(yy.min(),yy.max())

plt.figure(figsize=(12,6))  
plt.subplot(1,2,1)  
plt.title("Trainning")  
plotq(model,X\_train,y\_train)  
plt.subplot(1,2,2)  
plt.title("Testing")  
plotq(model,X\_test,y\_test)

313/313 [==============================] - 1s 1ms/step  
Doing Binary  
313/313 [==============================] - 0s 1ms/step  
Doing Binary



### SVM

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.datasets import load\_iris  
from sklearn.model\_selection import train\_test\_split  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score,classification\_report

iris = load\_iris()

X = iris.data[:,:2]

y = iris.target

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X, y, random\_state = 42, test\_size = 0.3)

svm\_classifier = SVC(kernel = 'linear',C=1)

svm\_classifier.fit(X\_train, y\_train)  
y\_pred = svm\_classifier.predict(X\_test)

print('Accuracy Score : ',accuracy\_score(y\_test,y\_pred))

Accuracy Score : 0.8

print('Classification Report : ', classification\_report(y\_test,y\_pred))

Classification Report : precision recall f1-score support  
  
 0 1.00 1.00 1.00 19  
 1 0.70 0.54 0.61 13  
 2 0.62 0.77 0.69 13  
  
 accuracy 0.80 45  
 macro avg 0.78 0.77 0.77 45  
weighted avg 0.81 0.80 0.80 45

x\_min,x\_max = X[:, 0].min()-1,X[:, 0].max()+1  
y\_min, y\_max = X[:, 1].min()-1,X[:, 1].max()+1

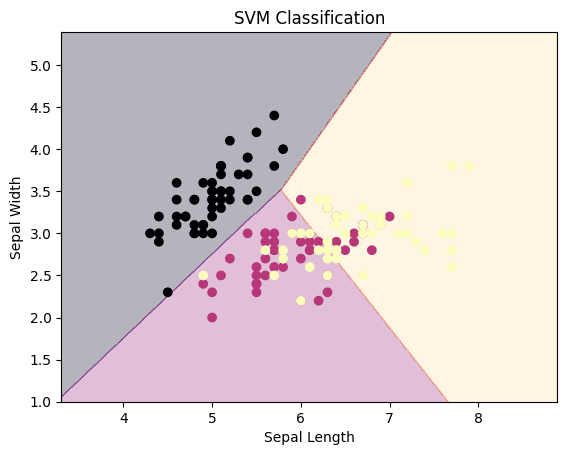
xx, yy = np.meshgrid(np.arange(x\_min,x\_max,0.01),np.arange(y\_min,y\_max,0.01))

z = svm\_classifier.predict(np.c\_[xx.ravel(),yy.ravel()])

z = z.reshape(xx.shape)

plt.contourf(xx,yy,z,cmap='magma', alpha = 0.3)  
plt.scatter(X[:,0],X[:,1],c=y,cmap='magma')  
plt.xlabel('Sepal Length')  
plt.ylabel('Sepal Width')  
plt.title('SVM Classification')

Text(0.5, 1.0, 'SVM Classification')



### Decision Tree and Random Forest

# Decision Tree   
  
import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import accuracy\_score,classification\_report,confusion\_matrix

data = pd.read\_csv('kyphosis.csv')

X = data.drop('Kyphosis',axis=1)

y = data['Kyphosis']

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,random\_state=42,test\_size=0.3)

model = DecisionTreeClassifier()  
model.fit(X\_train,y\_train)  
y\_pred = model.predict(X\_test)

print('Accuracy Score : ',accuracy\_score(y\_pred,y\_test))

Accuracy Score : 0.64

print('Classification Report :',classification\_report(y\_pred,y\_test))

Classification Report : precision recall f1-score support  
  
 absent 0.79 0.75 0.77 20  
 present 0.17 0.20 0.18 5  
  
 accuracy 0.64 25  
 macro avg 0.48 0.47 0.48 25  
weighted avg 0.66 0.64 0.65 25

print('Confusion Matrix : \n',confusion\_matrix(y\_pred,y\_test))

Confusion Matrix :   
 [[15 5]  
 [ 4 1]]

# Random Forest  
  
from sklearn.ensemble import RandomForestClassifier

randf = RandomForestClassifier(max\_depth = 10, min\_samples\_split = 5, min\_samples\_leaf = 2)

randf = randf.fit(X\_train,y\_train)

y\_pred = randf.predict(X\_test)

print('Accuracy Score : ',accuracy\_score(y\_pred,y\_test))

Accuracy Score : 0.8

print('Classification Report : ', classification\_report(y\_pred,y\_test))

Classification Report : precision recall f1-score support  
  
 absent 1.00 0.79 0.88 24  
 present 0.17 1.00 0.29 1  
  
 accuracy 0.80 25  
 macro avg 0.58 0.90 0.58 25  
weighted avg 0.97 0.80 0.86 25

print('Confusion matrix : \n ',confusion\_matrix(y\_pred,y\_test))

Confusion matrix :   
 [[19 5]  
 [ 0 1]]