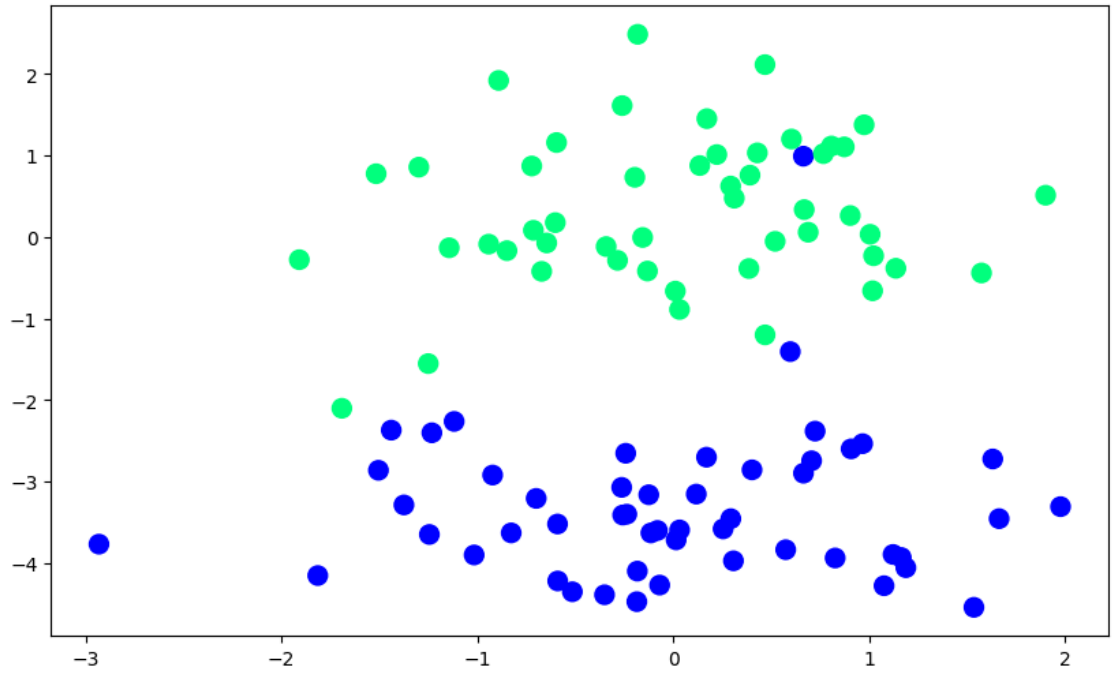
Start coding or generate with AI.

```
from sklearn.datasets import make_classification
import numpy as np
X , y = make_classification(n_samples = 100,n_features=2,n_informative = 1,n_redundant=0,n_classes = 2,n_clusters_per_class = 1,random
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.scatter(X[:,0],X[:,1],c=y,cmap ='winter',s=100)
```

```
<matplotlib.collections.PathCollection at 0x7cbadf1e95d0>
```




```
def perceptron(X,y):
  w1=w2=b=1
  lr = 0.1
  for j in range(1000):
    for i in range(X.shape[0]):
      #check condition
      z = w1*X[i][0] + w2*X[i][1] + b

      if z*y[i] < 0:
        w1 = w1 + lr *y[i] * x[i][0]
        w2 = w2 + lr *y[i] * x[i][1]
        b = b + lr * y[i]

  return w1,w2,b
```

```
w1 , w2, b = perceptron(X,y)
```

```
b
```

```
1.5000000000000004
```

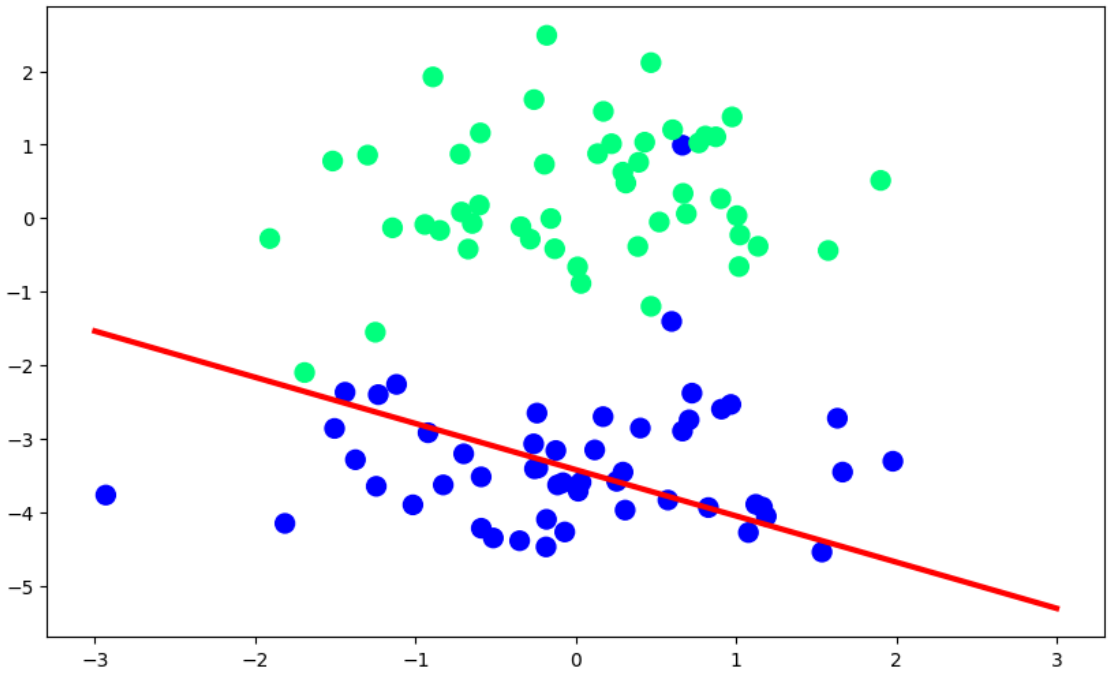
```
m = -(w1/w2)
c = -(b/w2)
```

```
print(m, c)
```

```
-0.6288317273984662 -3.424439973383999
```

```
x_input = np.linspace(-3,3,100)
y_input = m * x_input + c
plt.figure(figsize=(10,6))
plt.plot(x_input,y_input,color='red',linewidth=3)
plt.scatter(X[:,0],X[:,1],c=y,cmap = 'winter',s = 100)
#plt.ylim(-3,2)
```

```
<matplotlib.collections.PathCollection at 0x7cbadedabc50>
```



```
#deep lerning back propagation
#using keras libary
import pandas as pd
#customer churn predection
df = pd.read_csv('/content/Churn_Modelling.csv')
```

```
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMemb
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.duplicated().sum()
```

```
0
```

```
df['Exited'].value_counts()
```

Exited	count
0	7963
1	2037

dtype: int64

```
df['Geography'].value_counts()
```

Geography	count
France	5014
Germany	2509
Spain	2477

dtype: int64

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

Gender	count
Male	5457
Female	4543

dtype: int64

```
df.drop(columns = ['RowNumber','CustomerId','Surname'],inplace = True)
```

```
df.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df =pd.get_dummies(df,columns =['Geography','Gender'],drop_first = 1)
```

```
df.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography_Germany	Geog
0	619	42	2	0.00	1	1	1	101348.88	1	False	
1	608	41	1	83807.86	1	0	1	112542.58	0	False	
2	502	42	8	159660.80	3	1	0	113931.57	1	False	
3	699	39	1	0.00	2	0	0	93826.63	0	False	
4	850	43	2	125510.82	1	1	1	79084.10	0	False	

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
X = df.drop(columns=['Exited'])
y = df['Exited']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_state = 1)
```

```
X_train.shape
#y_train.shape
```

```
(8000, 11)
```

```
from sklearn.preprocessing import StandardScaler
Scaler = StandardScaler()
X_train_scaled = Scaler.fit_transform(X_train)
X_test_scaled = Scaler.transform(X_test)
```

```
X_train_scaled
```

```
array([[-0.23082038, -0.94449979, -0.70174202, ...,  1.71490137,
        -0.57273139,  0.91509065],
       [-0.25150912, -0.94449979, -0.35520275, ..., -0.58312392,
        -0.57273139, -1.09278791],
       [-0.3963303 ,  0.77498705,  0.33787579, ...,  1.71490137,
        -0.57273139, -1.09278791],
       ...,
       [ 0.22433188,  0.58393295,  1.3774936 , ..., -0.58312392,
        -0.57273139, -1.09278791],
       [ 0.13123255,  0.01077067,  1.03095433, ..., -0.58312392,
        -0.57273139, -1.09278791],
       [ 1.1656695 ,  0.29735181,  0.33787579, ...,  1.71490137,
        -0.57273139,  0.91509065]])
```

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

```
#sequwntial model
model = Sequential()
model.add(Dense(3,activation = 'sigmoid',input_dim = 11))
model.add(Dense(1,activation = 'sigmoid'))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 3)	36
dense_1 (Dense)	(None, 1)	4

Total params: 40 (160.00 B)
Trainable params: 40 (160.00 B)
Non-trainable params: 0 (0.00 B)

```
#model compile stage
model.compile(loss='binary_crossentropy',optimizer = 'Adam')
# how many time data circulate in this model
model.fit(X_train_scaled,y_train,epochs = 10)
# in this model circulate time set 10
```

```
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.5862
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 0.5032
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.4588
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4583
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4515
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4426
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4395
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4274
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.4340
Epoch 10/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 0.4316
<keras.src.callbacks.history.History at 0x7c329abde350>
```

```
# if first layer 11 * 3 weight  is indicate 0 as a first layer and byas
model.layers[0].get_weights()
```

```
[array([[ 0.18344118, -0.17309414, -0.13008623],
        [-1.1072409 , -1.3260381 ,  0.8599433 ],
        [ 0.01090914,  0.06667478, -0.03120087],
        [-0.17348062, -0.12819386,  0.11614373],
        [-0.17748234,  1.1105272 , -0.04130099],
        [-0.01975251,  0.1264648 , -0.05268726],
        [ 0.825447  ,  0.06386437, -0.8465255 ],
```

```
               [ 0.07326419, -0.1307233 ,  0.18334378],
               [-0.5747617 , -0.17225416,  0.6857121 ],
               [ 0.1701609 , -0.03414181,  0.46545658],
               [ 0.403089  ,  0.20984532, -0.42680144]], dtype=float32),
        array([ 0.4416321 ,  0.49437752, -0.36710724], dtype=float32)]
```

```
model.predict(X_test_scaled)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
array([[0.10058147],
       [0.13377489],
       [0.11883842],
       ...,
       [0.05824847],
       [0.12421601],
       [0.37958062]], dtype=float32)
```

```
y_log  = model.predict(X_test_scaled)
# this value convert to o and 1 as we dicided if value > 0.5 then 1 and if value <0.5 vale is 1
y_pred = np.where(y_log > 0.5,1,0)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

```
#find the accuracy in this model
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

```
0.814
```

```
Start coding or generate with AI.
```

to get more accuracy improve activation function ralue and use more epochs value

increase number of node increase hidden layer