데이터 다루기

판다스를 활용한 데이터 조사

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 피마 인디언 당뇨병 데이터셋을 불러옵니다.
df = pd.read_csv('./data/pima-indians-diabetes3.csv')
```

In [4]:
```
# 처음 5줄을 봅니다.
df.head(5)
```

Out[4]:

	pregnant	plasma	pressure	thickness	insulin	bmi	pedigree	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [5]:
```
# 정상과 당뇨 환자가 각각 몇 명씩인지 조사해 봅니다.
df["diabetes"].value_counts()
```

Out[5]:
```
diabetes
0    500
1    268
Name: count, dtype: int64
```

In [6]:
```
# 각 정보별 특징을 좀 더 자세히 출력합니다.
df.describe()
```

Out[6]:

	pregnant	plasma	pressure	thickness	insulin	bmi	pedigree	age	diabetes
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [7]:
```
# 각 항목이 어느 정도의 상관 관계를 가지고 있는지 알아봅니다.
df.corr()
```

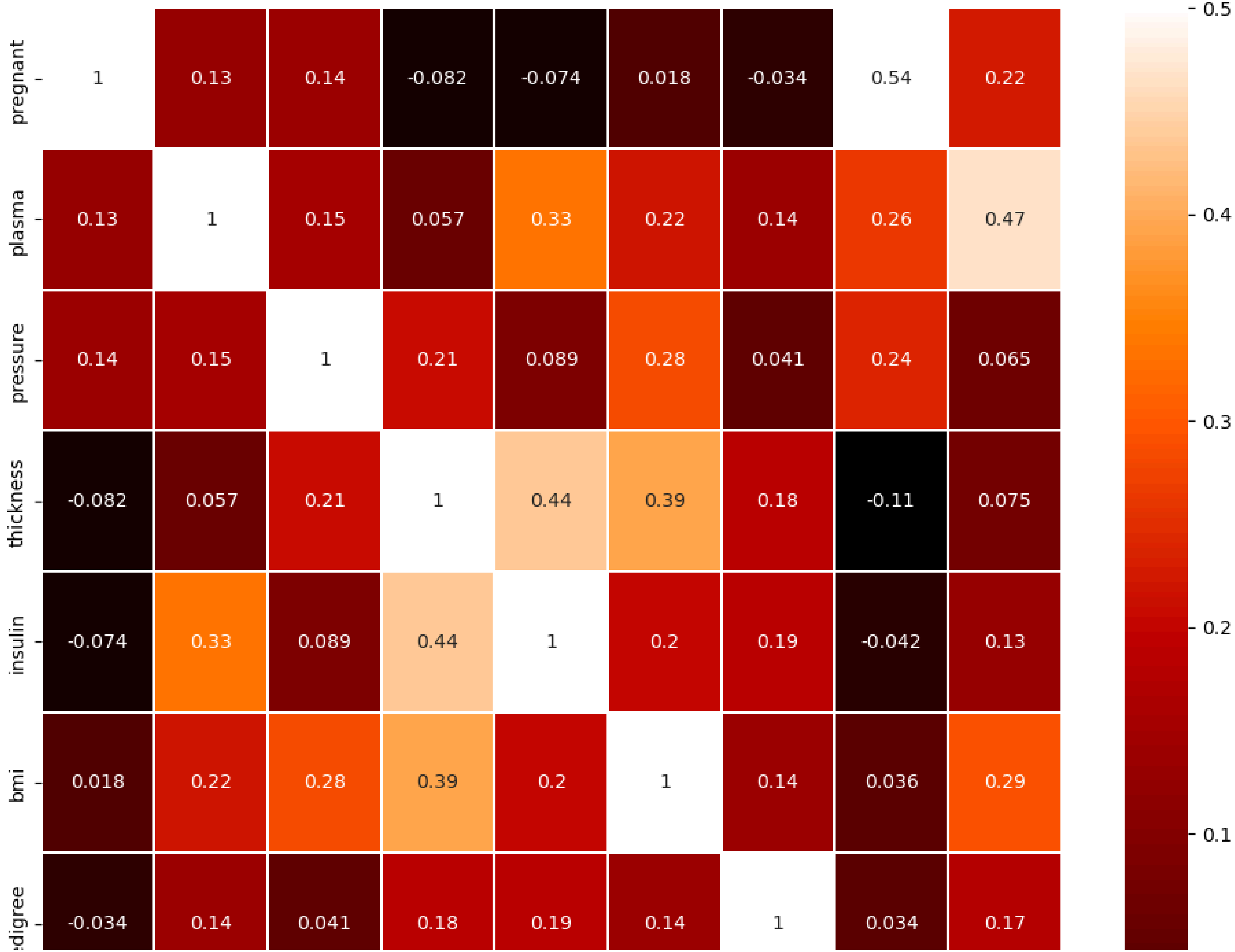
Out[7]:

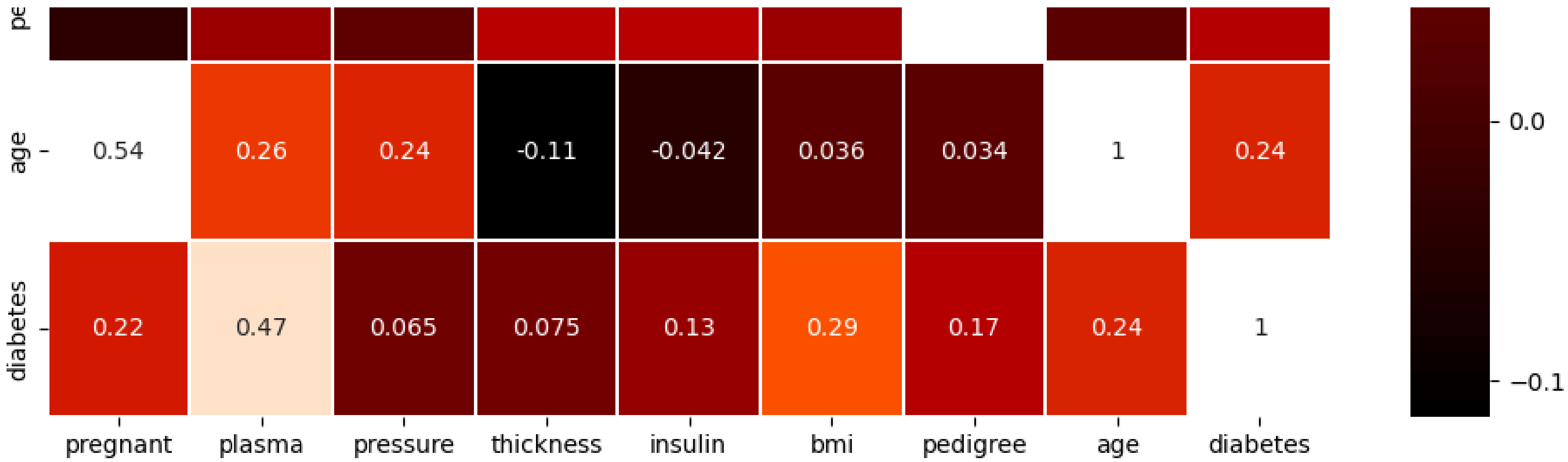
	pregnant	plasma	pressure	thickness	insulin	bmi	pedigree	age	diabetes
pregnant	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
plasma	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
pressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
thickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
bmi	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
pedigree	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
diabetes	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

In [8]:

```
# 데이터 간의 상관 관계를 그래프로 표현해 봅니다.
colormap = plt.cm.gist_heat   # 그래프의 색상 구성을 정합니다.
plt.figure(figsize=(12,12))   # 그래프의 크기를 정합니다.

# 그래프의 속성을 결정합니다. vmax의 값을 0.5로 지정해 0.5에 가까울수록 밝은색으로 표시되게 합니다.
sns.heatmap(df.corr(),linewidths=0.1,vmax=0.5, cmap=colormap,
            linecolor='white', annot=True)
plt.show()
```


pregnant
plasma
pressure
thickness
insulin
bmi
edigree
1 0.13 0.14 -0.082 -0.074 0.018 -0.034 0.54 0.22
0.13 1 0.15 0.057 0.33 0.22 0.14 0.26 0.47
0.14 0.15 1 0.21 0.089 0.28 0.041 0.24 0.065
-0.082 0.057 0.21 1 0.44 0.39 0.18 -0.11 0.075
-0.074 0.33 0.089 0.44 1 0.2 0.19 -0.042 0.13
0.018 0.22 0.28 0.39 0.2 1 0.14 0.036 0.29
-0.034 0.14 0.041 0.18 0.19 0.14 1 0.034 0.17
0.5
0.4
0.3
0.2
0.1



4. 중요한 데이터 추출하기

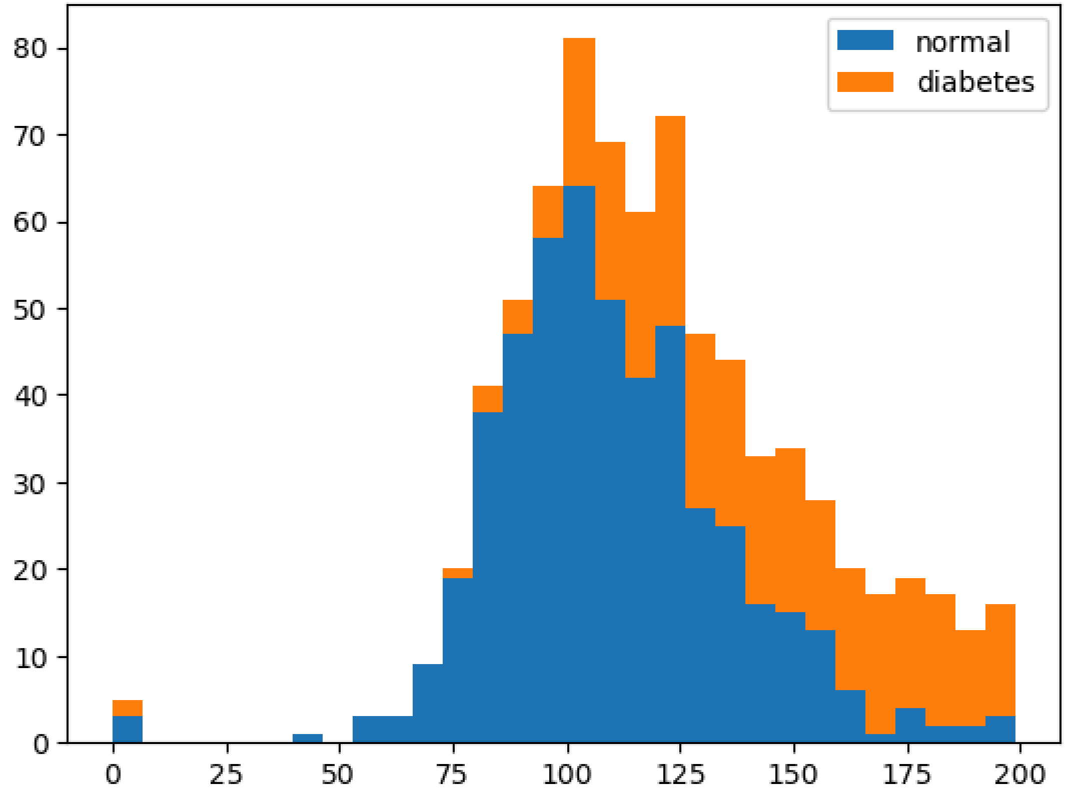
In [10]:

```
import warnings
warnings.filterwarnings("ignore")

# plasma를 기준으로 각각 정상과 당뇨가 어느 정도 비율로 분포하는지 살펴봅니다.
plt.hist(x=[df.plasma[df.diabetes==0], df.plasma[df.diabetes==1]], bins=30,
         histtype='barstacked', label=['normal','diabetes'])
plt.legend()
```

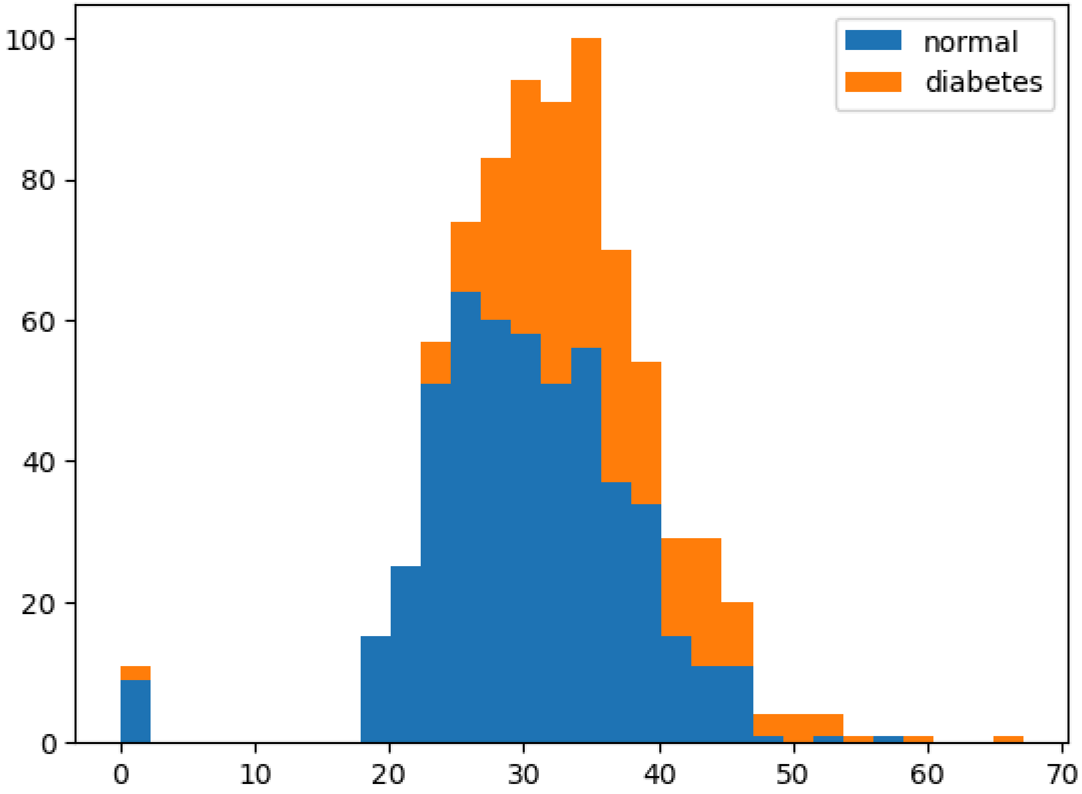
Out[10]:

```
<matplotlib.legend.Legend at 0x1a32dfe6ab0>
```



In [11]:
```
# BMI를 기준으로 각각 정상과 당뇨가 어느 정도 비율로 분포하는지 살펴봅니다.
plt.hist(x=[df.bmi[df.diabetes==0], df.bmi[df.diabetes==1]], bins=30,
         histtype='barstacked', label=['normal','diabetes'])
plt.legend()
```

Out[11]:
```
<matplotlib.legend.Legend at 0x1a32d277a70>
```



5. 피마 인디언 당뇨병 예측 실행

In [13]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# pandas 라이브러리를 불러옵니다.
import pandas as pd

# 피마 인디언 당뇨병 데이터셋을 불러옵니다.
df = pd.read_csv('./data/pima-indians-diabetes3.csv')
```

In [14]:
```
# 세부 정보를 X로 지정합니다.
X = df.iloc[:,0:8]
```

```
# 당뇨병 여부를 y로 지정합니다.
y = df.iloc[:,8]
```

In [15]:

```
# 모델을 설정합니다.
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu', name='Dense_1'))
model.add(Dense(8, activation='relu', name='Dense_2'))
model.add(Dense(1, activation='sigmoid',name='Dense_3'))
model.summary()
```

Model: "sequential"

Layer (type)	**Output Shape**	**Param #**
Dense_1 (Dense)	(None, 12)	108
Dense_2 (Dense)	(None, 8)	104
Dense_3 (Dense)	(None, 1)	9

Total params: 221 (884.00 B)

Trainable params: 221 (884.00 B)

Non-trainable params: 0 (0.00 B)

In [16]:

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

# 모델을 실행합니다.
history=model.fit(X, y, epochs=100, batch_size=5)
```

```
Epoch 1/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 1s 927us/step - accuracy: 0.5365 - loss: 2.8902
Epoch 2/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 910us/step - accuracy: 0.6390 - loss: 1.1825
Epoch 3/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - accuracy: 0.6601 - loss: 0.9206
Epoch 4/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 791us/step - accuracy: 0.5786 - loss: 1.0341
Epoch 5/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - accuracy: 0.6447 - loss: 0.8250
Epoch 6/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 758us/step - accuracy: 0.6410 - loss: 0.8132
Epoch 7/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 803us/step - accuracy: 0.6430 - loss: 0.8093
Epoch 8/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 776us/step - accuracy: 0.6580 - loss: 0.7157
Epoch 9/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 777us/step - accuracy: 0.6203 - loss: 0.8318
Epoch 10/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 783us/step - accuracy: 0.6725 - loss: 0.6819
Epoch 11/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 861us/step - accuracy: 0.6297 - loss: 0.7660
Epoch 12/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - accuracy: 0.6571 - loss: 0.6649
Epoch 13/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 776us/step - accuracy: 0.6630 - loss: 0.6808
Epoch 14/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - accuracy: 0.6263 - loss: 0.8030
Epoch 15/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - accuracy: 0.6719 - loss: 0.6374
Epoch 16/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 924us/step - accuracy: 0.6564 - loss: 0.7278
Epoch 17/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 781us/step - accuracy: 0.6711 - loss: 0.6465
Epoch 18/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 812us/step - accuracy: 0.7208 - loss: 0.6101
Epoch 19/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - accuracy: 0.7027 - loss: 0.6048
Epoch 20/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 754us/step - accuracy: 0.7143 - loss: 0.6019
Epoch 21/100
```

```
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6563 - loss: 0.6568
Epoch 22/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.6714 - loss: 0.6535
Epoch 23/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 781us/step - accuracy: 0.6967 - loss: 0.6017
Epoch 24/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - accuracy: 0.6971 - loss: 0.6470
Epoch 25/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - accuracy: 0.7067 - loss: 0.5890
Epoch 26/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 844us/step - accuracy: 0.6882 - loss: 0.5977
Epoch 27/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - accuracy: 0.6844 - loss: 0.6217
Epoch 28/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - accuracy: 0.6460 - loss: 0.6364
Epoch 29/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - accuracy: 0.6626 - loss: 0.5985
Epoch 30/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - accuracy: 0.6425 - loss: 0.6307
Epoch 31/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 852us/step - accuracy: 0.7025 - loss: 0.5741
Epoch 32/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 806us/step - accuracy: 0.7101 - loss: 0.5915
Epoch 33/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 923us/step - accuracy: 0.6734 - loss: 0.6301
Epoch 34/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 799us/step - accuracy: 0.6752 - loss: 0.6221
Epoch 35/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 782us/step - accuracy: 0.7178 - loss: 0.6084
Epoch 36/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 835us/step - accuracy: 0.7075 - loss: 0.5971
Epoch 37/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 804us/step - accuracy: 0.7123 - loss: 0.5593
Epoch 38/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - accuracy: 0.7354 - loss: 0.5713
Epoch 39/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 795us/step - accuracy: 0.7292 - loss: 0.5929
Epoch 40/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 780us/step - accuracy: 0.6956 - loss: 0.5879
Epoch 41/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 824us/step - accuracy: 0.6871 - loss: 0.5982
```

```
Epoch 42/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 857us/step - accuracy: 0.6635 - loss: 0.6448
Epoch 43/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - accuracy: 0.6953 - loss: 0.5737
Epoch 44/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 764us/step - accuracy: 0.7110 - loss: 0.5824
Epoch 45/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7511 - loss: 0.5404
Epoch 46/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - accuracy: 0.7206 - loss: 0.5412
Epoch 47/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 911us/step - accuracy: 0.7112 - loss: 0.6021
Epoch 48/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 774us/step - accuracy: 0.7165 - loss: 0.6058
Epoch 49/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7298 - loss: 0.5570
Epoch 50/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 917us/step - accuracy: 0.7156 - loss: 0.5744
Epoch 51/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - accuracy: 0.7437 - loss: 0.5336
Epoch 52/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 770us/step - accuracy: 0.7479 - loss: 0.5348
Epoch 53/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - accuracy: 0.7158 - loss: 0.5588
Epoch 54/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 808us/step - accuracy: 0.7224 - loss: 0.5798
Epoch 55/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 782us/step - accuracy: 0.6888 - loss: 0.6272
Epoch 56/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 841us/step - accuracy: 0.7124 - loss: 0.5739
Epoch 57/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - accuracy: 0.7117 - loss: 0.5585
Epoch 58/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 999us/step - accuracy: 0.7218 - loss: 0.5447
Epoch 59/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 820us/step - accuracy: 0.7059 - loss: 0.5492
Epoch 60/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 796us/step - accuracy: 0.7444 - loss: 0.5303
Epoch 61/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 817us/step - accuracy: 0.7544 - loss: 0.5302
Epoch 62/100
```

```
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - accuracy: 0.7428 - loss: 0.5333
Epoch 63/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 801us/step - accuracy: 0.7481 - loss: 0.5458
Epoch 64/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 812us/step - accuracy: 0.7243 - loss: 0.5591
Epoch 65/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - accuracy: 0.7775 - loss: 0.5067
Epoch 66/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - accuracy: 0.7456 - loss: 0.5448
Epoch 67/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - accuracy: 0.7501 - loss: 0.5695
Epoch 68/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 786us/step - accuracy: 0.7225 - loss: 0.5709
Epoch 69/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 780us/step - accuracy: 0.7596 - loss: 0.5278
Epoch 70/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 792us/step - accuracy: 0.7349 - loss: 0.5372
Epoch 71/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 767us/step - accuracy: 0.7429 - loss: 0.5434
Epoch 72/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - accuracy: 0.7543 - loss: 0.5252
Epoch 73/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 765us/step - accuracy: 0.7177 - loss: 0.5613
Epoch 74/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 981us/step - accuracy: 0.7385 - loss: 0.5531
Epoch 75/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 854us/step - accuracy: 0.7436 - loss: 0.5294
Epoch 76/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 855us/step - accuracy: 0.7261 - loss: 0.5206
Epoch 77/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - accuracy: 0.7275 - loss: 0.5898
Epoch 78/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 807us/step - accuracy: 0.7078 - loss: 0.5381
Epoch 79/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 821us/step - accuracy: 0.7220 - loss: 0.5519
Epoch 80/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7430 - loss: 0.5291
Epoch 81/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 842us/step - accuracy: 0.7625 - loss: 0.5137
Epoch 82/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 753us/step - accuracy: 0.7314 - loss: 0.5326
```

```
Epoch 83/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 779us/step - accuracy: 0.7763 - loss: 0.5197
Epoch 84/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 775us/step - accuracy: 0.7469 - loss: 0.5179
Epoch 85/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 767us/step - accuracy: 0.7444 - loss: 0.5158
Epoch 86/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 947us/step - accuracy: 0.7056 - loss: 0.5737
Epoch 87/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 763us/step - accuracy: 0.7559 - loss: 0.5398
Epoch 88/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 827us/step - accuracy: 0.7032 - loss: 0.5613
Epoch 89/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 764us/step - accuracy: 0.7651 - loss: 0.4987
Epoch 90/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 797us/step - accuracy: 0.7281 - loss: 0.5502
Epoch 91/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 757us/step - accuracy: 0.7357 - loss: 0.5226
Epoch 92/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 747us/step - accuracy: 0.7396 - loss: 0.5375
Epoch 93/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 775us/step - accuracy: 0.7392 - loss: 0.5304
Epoch 94/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 778us/step - accuracy: 0.7403 - loss: 0.5225
Epoch 95/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 767us/step - accuracy: 0.7700 - loss: 0.4906
Epoch 96/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 760us/step - accuracy: 0.7543 - loss: 0.5300
Epoch 97/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 768us/step - accuracy: 0.7413 - loss: 0.5390
Epoch 98/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 767us/step - accuracy: 0.6862 - loss: 0.5992
Epoch 99/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 787us/step - accuracy: 0.7442 - loss: 0.5505
Epoch 100/100
154/154 ━━━━━━━━━━━━━━━━━━━━ 0s 767us/step - accuracy: 0.7602 - loss: 0.4981
```

In [17]:

```
print("\n loss: %.4f" % (model.evaluate(X, y)[0]))
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7416 - loss: 0.5021

loss: 0.4776
```

In [18]:
```python
print("\n Accuracy: %.4f" % (model.evaluate(X, y)[1]))
```

```
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7416 - loss: 0.5021

 Accuracy: 0.7708
```