데이터 다루기

판다스를 활용한 데이터 조사

```
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')

# 처음 5줄을 봅니다.
df.head(5)
```

	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

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

```
0    500
1    268
Name: diabetes, dtype: int64
```

```
# 각 정보별 특징을 좀 더 자세히 출력합니다.
df.describe()
```

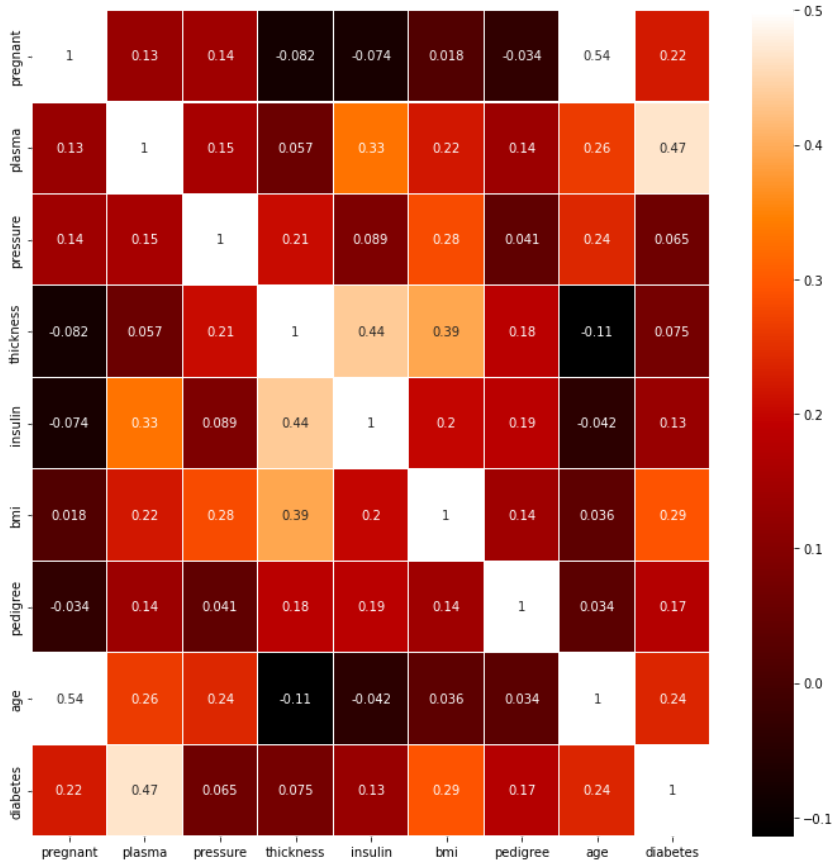
	pregnant	plasma	pressure	thickness	insulin	bmi	pedigr
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000(
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078(
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420(

```
# 각 항목이 어느 정도의 상관 관계를 가지고 있는지 알아봅니다.
df.corr()
```

```
              pregnant   plasma  pressure  thickness   insulin       bmi  pedigree
```

```
# 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()
```

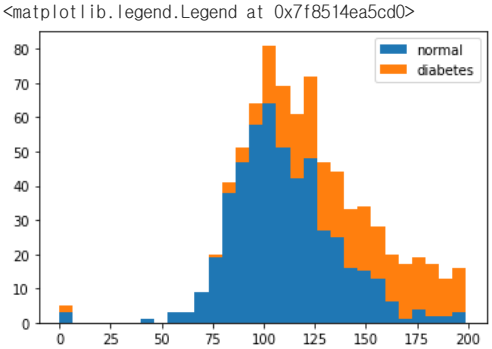


4. 중요한 데이터 추출하기

```
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
plt.legend()
```

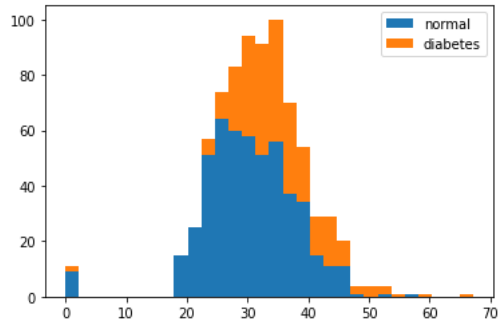
```
<matplotlib.legend.Legend at 0x7f8514ea5cd0>
```




```
# 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=['nor
```

```
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f851475ce50>
```



5. 피마 인디언 당뇨병 예측 실행

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

# pandas 라이브러리를 불러옵니다.
import pandas as pd

# 깃허브에 준비된 데이터를 가져옵니다. 이미 앞에서 가져왔으므로 주석 처리합니다. 5번 예제만 별도 실행 시
# !git clone https://github.com/taehojo/data.git

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

# 세부 정보를 X로 지정합니다.
X = df.iloc[:,0:8]
# 당뇨병 여부를 y로 지정합니다.
y = df.iloc[:,8]

# 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
Trainable params: 221
Non-trainable params: 0
_________________________________________________________________
```

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

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

```
154/154 [==============================] - 0s 2ms/step - loss: 0.5299 - accuracy: 0.7409
Epoch 76/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5368 - accuracy: 0.7266
Epoch 77/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5229 - accuracy: 0.7474
Epoch 78/100
154/154 [==============================] - 0s 2ms/step - loss: 0.5421 - accuracy: 0.7448
Epoch 79/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5443 - accuracy: 0.7305
Epoch 80/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5259 - accuracy: 0.7214
Epoch 81/100
154/154 [==============================] - 0s 2ms/step - loss: 0.5672 - accuracy: 0.7214
Epoch 82/100
154/154 [==============================] - 0s 2ms/step - loss: 0.5195 - accuracy: 0.7435
Epoch 83/100
154/154 [==============================] - 0s 2ms/step - loss: 0.5311 - accuracy: 0.7409
Epoch 84/100
154/154 [==============================] - 0s 2ms/step - loss: 0.5358 - accuracy: 0.7396
Epoch 85/100
154/154 [==============================] - 0s 2ms/step - loss: 0.5322 - accuracy: 0.7266
Epoch 86/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5427 - accuracy: 0.7344
Epoch 87/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5313 - accuracy: 0.7565
Epoch 88/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5290 - accuracy: 0.7383
Epoch 89/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5494 - accuracy: 0.7161
Epoch 90/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5298 - accuracy: 0.7344
Epoch 91/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5150 - accuracy: 0.7513
Epoch 92/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5174 - accuracy: 0.7539
Epoch 93/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5226 - accuracy: 0.7539
Epoch 94/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5155 - accuracy: 0.7721
Epoch 95/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5312 - accuracy: 0.7383
Epoch 96/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5131 - accuracy: 0.7513
Epoch 97/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5102 - accuracy: 0.7500
Epoch 98/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5208 - accuracy: 0.7513
Epoch 99/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5154 - accuracy: 0.7513
Epoch 100/100
154/154 [==============================] - 0s 1ms/step - loss: 0.5244 - accuracy: 0.7448
```

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

```
24/24 [==============================] - 0s 1ms/step - loss: 0.5326 - accuracy: 0.7435

 Accuracy: 0.7435
```