데이터베이스응용 2차 과제

과목	데이터베이스응용(01)
담당교수	김현희 교수님
이름	변서영
학과	컴퓨터학과
학번	20220769
제출일	2023.12.21

1. 분류 예측 모델

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
# Import modules
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import numpy as np
from sklearn import tree
from sklearn.model_selection import train_test_split

# Figures inline and set visualization style
%matplotlib inline
sns.set()
```

: 분류 예측 모델에 필요한 부분 import

```
df = pd.read_csv("/content/gdrive/MyDrive/데이터베이스응용/diabetes.csv")

print(df.shape)
df.head()
```

```
(768, 9)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

: df로 diabetes의 데이터 저장 후 데이터 형태 관찰(768행, 9열)

```
df.isnull().sum()
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

: df 데이터 결측지 탐색->diabetes 데이터에는 결측치가 없으므로 별도의 처리가 필요 없다.

```
Y = df['Outcome']
X = df.drop(['Outcome'], axis=1, inplace=False)

X.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

1. Y에 df 데이터의 Outcome 열 데이터 저장

2. X에 df 데이터의 Outcome 열 제거 후 저장

3. X.head()로 X 데이터 형태 관찰(8개 변수->Outcome이 제거된 데이터)

```
Y.head()
```

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

: Y 데이터 관찰(Outcome 데이터 관찰)

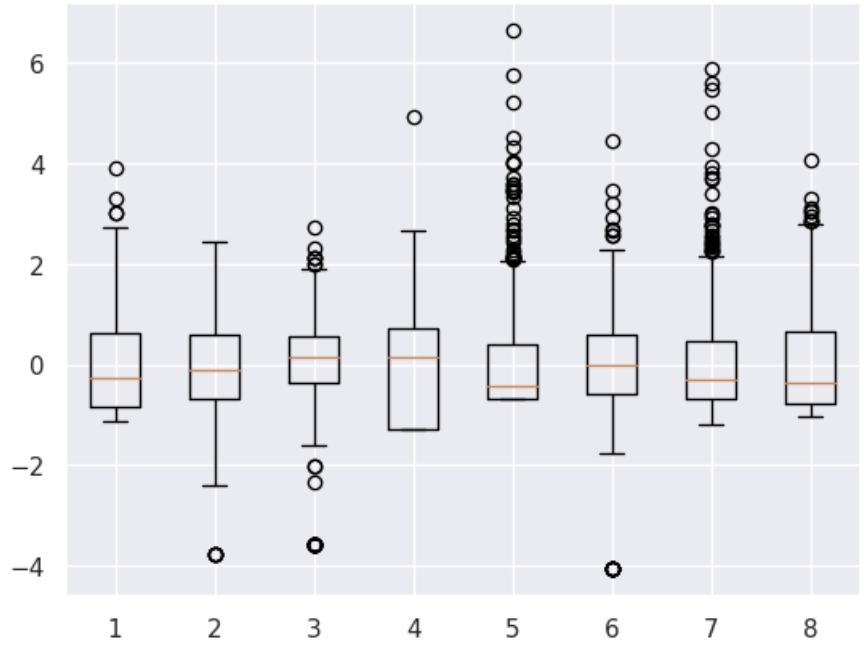
```
mean = X.mean()
std = X.std()

nX = (X - mean)/std
nX.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.639530	0.847771	0.149543	0.906679	-0.692439	0.203880	0.468187	1.425067
1	-0.844335	-1.122665	-0.160441	0.530556	-0.692439	-0.683976	-0.364823	-0.190548
2	1.233077	1.942458	-0.263769	-1.287373	-0.692439	-1.102537	0.604004	-0.105515
3	-0.844335	-0.997558	-0.160441	0.154433	0.123221	-0.493721	-0.920163	-1.040871
4	-1.141108	0.503727	-1.503707	0.906679	0.765337	1.408828	5.481337	-0.020483

: Z-score 정규화 진행(label별 값의 규모차이가 존재하기 때문에 정규화 필요)

1. X.mean()으로 X 데이터의 평균 구하기
2. X.std()로 X 데이터의 표준편차 구하기
3. nX에 X 정규화한 값 저장
4. nX의 데이터 형태 관찰->정규화된 값들 관찰 가능

```
plt.boxplot(nX)
plt.show()
```



: 박스플롯을 이용한 이상치 탐색

```
df_ = pd.concat([nX, Y], axis=1)
df_.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.639530	0.847771	0.149543	0.906679	-0.692439	0.203880	0.468187	1.425067	1
1	-0.844335	-1.122665	-0.160441	0.530556	-0.692439	-0.683976	-0.364823	-0.190548	0
2	1.233077	1.942458	-0.263769	-1.287373	-0.692439	-1.102537	0.604004	-0.105515	1
3	-0.844335	-0.997558	-0.160441	0.154433	0.123221	-0.493721	-0.920163	-1.040871	0
4	-1.141108	0.503727	-1.503707	0.906679	0.765337	1.408828	5.481337	-0.020483	1

```
df_ = df_.drop(df_[df_.Glucose < -3].index)
df_ = df_.drop(df_[df_.BloodPressure <= -2].index)
df_ = df_.drop(df_[df_.SkinThickness > 4].index)
df_ = df_.drop(df_[df_.BMI < -3].index)
df_ = df_.drop(df_[df_.DiabetesPedigreeFunction  >= 4].index)
```

```
Y = df_['Outcome']
X = df_.drop(['Outcome'], axis=1, inplace=False)
```

1. df에 정규화 된 데이터 값 nX와 Outcome값 통합하기(pd.concat)

2. 박스플롯으로 관찰한 이상치들을 df_.drop을 통해 제거하고 완료되면 X, Y에 각각 Outcome 값을 제거한 나머지 feature 값, Outcome 데이터 값을 저장

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=156)
```

```
train = X_train.values
y_train = Y_train.values

test = X_test.values
y_test = Y_test.values
```

1. X, Y를 훈련집합, 테스트집합으로 나눈다(훈련집합 0.7, 테스트집합 0.3 비율)

2. X의 훈련집합 값(values)을 train에 저장

3. Y의 훈련집합 값(values)을 Y_train에 저장

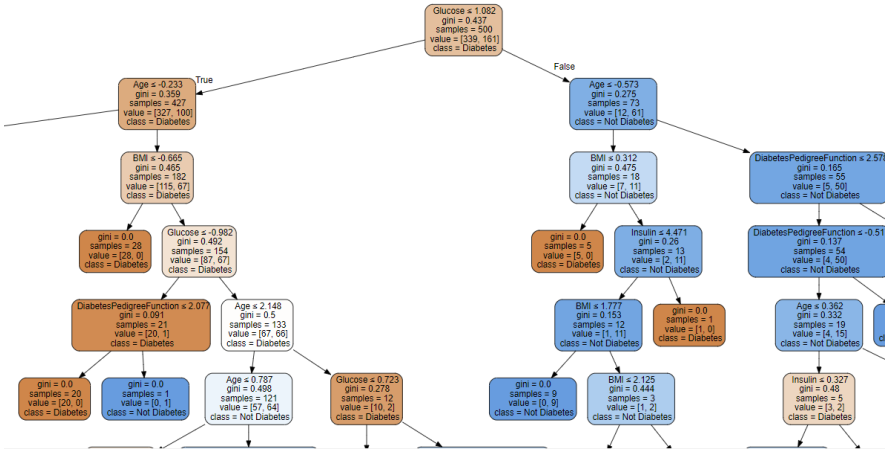
4. X의 테스트집합 값(values)을 test에 저장

5. Y의 테스트집합 값(values)을 Y_test에 저장

```
clf = tree.DecisionTreeClassifier()
clf.fit(train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
import graphviz
from PIL import Image
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("diabetes")

dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=X_train.columns.values,
                                class_names=['Diabetes','Not Diabetes'],
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph
```



1. clf에 의사결정트리 모델 생성

2. 생성된 clf에 train, y_train 데이터 학습

3. 만들어진 모델 그리기

->결과적으로 당뇨병 예측에 가장 중요한 feature는 'Glucose'이다. 두 번째로 중요한 feature는 Age이고 clf가 그려지는 것에 따라 알 수 있다.

```
Y_pred = clf.predict(test)
clf.score(test, y_test)
```

```
0.7023255813953488
```

```
print(Y_pred)
```

```
[0 0 1 0 1 0 0 0 0 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 1 1 0 0
 0 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 0 0 1 1
 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 1
 1 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
 1 0 1 1 1 0 0 0 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0
 0 1 1 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0]
```

```
src = graphviz.Source(dot_data)
src.view()
```

```
'Source.gv.pdf'
```

1. 학습된 clf 모델로 test를 예측해 예측값을 Y_pred에 저장

2. clf.score()을 통해 모델의 정확도 구하기->clf 모델의 정확도 0.7

3. test 데이터의 예측값 Y_pred를 출력하고 이를 통해 당뇨병을 예측한다.

4. 의사결정 트리 모델 pdf로 저장

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

model = RandomForestClassifier(n_estimators=100)
model.fit(train, y_train)

y_pred = model.predict(test)

print('RandomForestClassifier: %.2f' % (metrics.accuracy_score(y_pred, y_test) * 100))
```

```
RandomForestClassifier: 77.21
```

: RandomForest model을 만들고 model에 tarin, y_train 데이터를 학습시켜 test의 예측값을 y_pred에 저장한다

이때 accuracy_score을 통해 모델의 정확도를 구하면 당뇨병을 예측하는 RandomForestClassfier모델의 정확도는 77.21

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='lbfgs', max_iter=2000)
model.fit(train, y_train)

y_pred = model.predict(test)

print('LogisticRegression: %.2f' % (metrics.accuracy_score(y_pred, y_test) * 100))
```

```
LogisticRegression: 75.81
```

: LogisticRegression model을 만들고 model에 tarin, y_train 데이터를 학습시켜 test의 예측값을 y_pred에 저장한다

이때 accuracy_score을 통해 모델의 정확도를 구하면 당뇨병을 예측하는 LogisticRegression 모델의 정확도는 75.81

2. 딥러닝 적용

2. 딥러닝 적용

```
from keras.models import Sequential
from keras import layers
from keras import optimizers
```

```
model = Sequential()
model.add(layers.Dense(units=8, activation='relu'))
model.add(layers.Dense(units=16, activation='relu'))
model.add(layers.Dense(units=32, activation='relu'))
model.add(layers.Dense(units=2, activation='softmax'))
```

1. 딥러닝 모델에 필요한 부분 import

2. Sequential()로 딥러닝 모델 생성

3. model.add()를 통해 층을 추가한다. 이때 은닉 노드는 각각 8, 16, 32로 설정하고(검증 손실 방지를 위한 층 개수 조절-은닉층 3개)

4. 출력 노드 2개 설정(units=2, activation="softmax")

```
model.compile(loss='categorical_crossentropy', optimizer=optimizers.RMSprop(learning_rate=0.001), metrics=['accuracy'])
```

: 손실 함수를 categorical_crossentropy로 설정하고 매개변수 이동값을 learning_rate를 통해 0.001로 설정한다.

```
from tensorflow.keras.utils import to_categorical
df = pd.read_csv("/content/gdrive/MyDrive/데이터베이스응용/diabetes.csv")

Y = df['Outcome']
X = df.drop(['Outcome'], axis=1, inplace=False)

mean = X.mean()
std = X.std()
nX = (X - mean)/std

df_ = pd.concat([nX, Y], axis=1)

df_ = df_.drop(df_[df_.Glucose < -3].index)
df_ = df_.drop(df_[df_.BloodPressure <= -2].index)
df_ = df_.drop(df_[df_.SkinThickness > 4].index)
df_ = df_.drop(df_[df_.BMI < -3].index)
df_ = df_.drop(df_[df_.DiabetesPedigreeFunction  >= 4].index)

train, test = train_test_split(df_, test_size=0.3, random_state=156)
```

: 분류 예측 모델에서 진행한 데이터 전처리 과정을 똑같이 진행한다

train, test에 df_값을 0.7, 0.3비율로 나눠 저장해 훈련집합, 테스트 집합을 생성

```
train_y = to_categorical(train['Outcome'])
```

```
train_np = train.drop(['Outcome'], axis=1, inplace=False)
test = test.drop(['Outcome'], axis=1, inplace=False)
```

1. train_y에는 to_categorical()을 이용해 train 데이터의 Outcome 항목 값을 binary 형식으로 변경해 저장

2. train_np에 train 데이터 Outcome 열을 삭제한 후 저장(훈련집합에서 예측하고자 하는 label 데이터 값 삭제)

3. test에 test 데이터 Outcome 열을 삭제한 후 저장(테스트집합에서 예측하고자 하는 label 데이터 값 삭제)

```
print(train.values)
```

```
[[-0.25078869  0.22223606  0.1495433  ...  0.23277103 -0.53067709
   1.        ]
 [-0.54756176 -1.43543237 -0.26376935 ... -0.30747815 -0.02048305
   0.        ]
 [ 0.04598437  0.81649456  0.25287146 ... -0.26220587 -0.27558007
   0.        ]
 ...
 [-0.25078869 -1.06011122 -0.47042568 ... -0.08413491 -1.04087112
   0.        ]
 [-1.14110788 -0.93500417 -0.05711303 ... -0.27427848 -0.70074177
   0.        ]
 [-0.84433482  0.09712901  0.25287146 ... -1.12237914 -0.27558007
   0.        ]]
```

```
print(train['Outcome'])
```

```
541    1
174    0
364    0
655    1
676    1
      ..
13     1
623    0
525    0
452    0
602    0
Name: Outcome, Length: 500, dtype: int64
```

1. 테스트 집합 값 출력(train 데이터의 값)

2. 테스트 집합의 Outcome label 값 출력

```
hist = model.fit(train_np,train_y,epochs= 100, batch_size=100, validation_split=0.3)
```

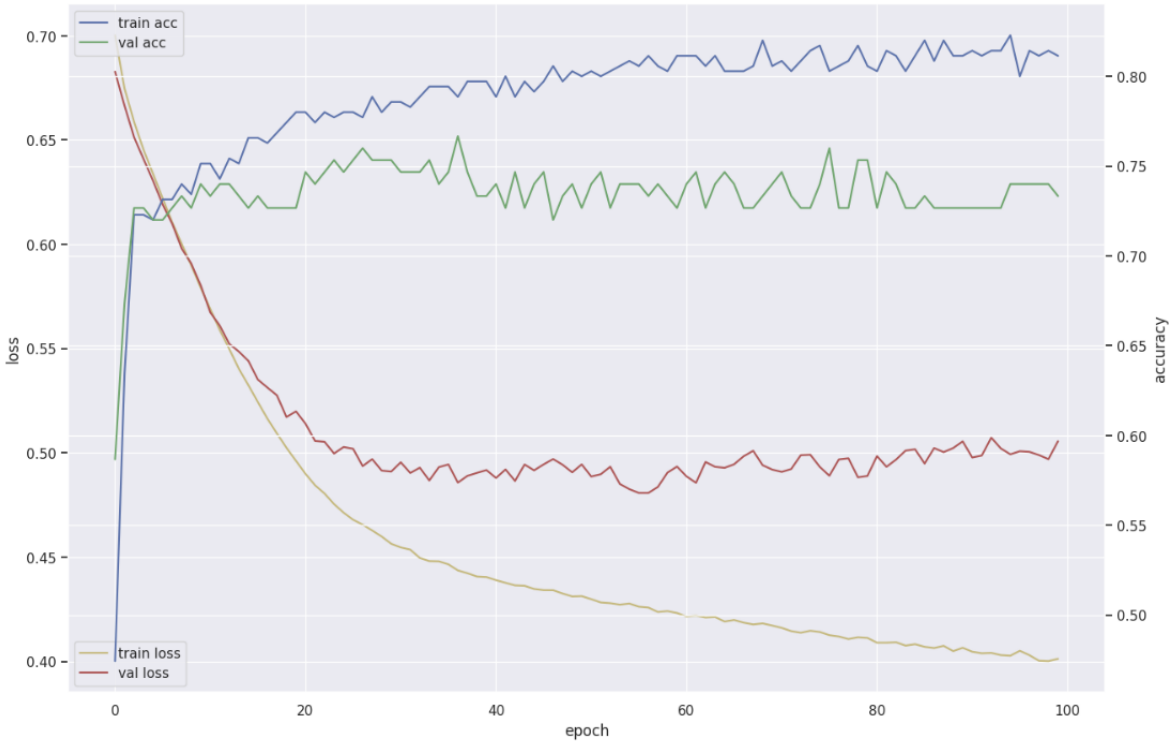
```
Epoch 1/100
4/4 [==============================] - 2s 125ms/step - loss: 0.7003 - accuracy: 0.4743 - val_loss: 0.6828 - val_accuracy: 0.5867
Epoch 2/100
4/4 [==============================] - 0s 24ms/step - loss: 0.6747 - accuracy: 0.6343 - val_loss: 0.6665 - val_accuracy: 0.6733
Epoch 3/100
4/4 [==============================] - 0s 25ms/step - loss: 0.6588 - accuracy: 0.7229 - val_loss: 0.6514 - val_accuracy: 0.7267
Epoch 4/100
4/4 [==============================] - 0s 26ms/step - loss: 0.6453 - accuracy: 0.7229 - val_loss: 0.6409 - val_accuracy: 0.7267
Epoch 5/100
4/4 [==============================] - 0s 22ms/step - loss: 0.6338 - accuracy: 0.7200 - val_loss: 0.6304 - val_accuracy: 0.7200
Epoch 6/100
4/4 [==============================] - 0s 26ms/step - loss: 0.6221 - accuracy: 0.7314 - val_loss: 0.6191 - val_accuracy: 0.7200
Epoch 7/100
4/4 [==============================] - 0s 20ms/step - loss: 0.6106 - accuracy: 0.7314 - val_loss: 0.6100 - val_accuracy: 0.7267
Epoch 8/100
4/4 [==============================] - 0s 24ms/step - loss: 0.6003 - accuracy: 0.7400 - val_loss: 0.5980 - val_accuracy: 0.7333
Epoch 9/100
4/4 [==============================] - 0s 25ms/step - loss: 0.5899 - accuracy: 0.7343 - val_loss: 0.5908 - val_accuracy: 0.7267
Epoch 10/100
4/4 [==============================] - 0s 27ms/step - loss: 0.5791 - accuracy: 0.7514 - val_loss: 0.5803 - val_accuracy: 0.7400
Epoch 11/100
4/4 [==============================] - 0s 21ms/step - loss: 0.5691 - accuracy: 0.7514 - val_loss: 0.5674 - val_accuracy: 0.7333
Epoch 12/100
4/4 [==============================] - 0s 25ms/step - loss: 0.5589 - accuracy: 0.7429 - val_loss: 0.5610 - val_accuracy: 0.7400
Epoch 13/100
```

1. train_np, train_y(테스트 집합의 Outcome 항목 제외 나머지 feature 값, 테스트 집합의 Outcome 항목 값)을 통해 model 학습(model.fit)

이때 val loss 상승을 막기 위해 epoch을 100회로 설정

2. hist에 모델의 학습 동안의 accuracy, loss의 변화하는 값을 저장한다.

```
import matplotlib.pyplot as plt

def drawHistory(hist):
  fig = plt.figure(figsize=(15,10))
  loss_ax = plt.gca()
  acc_ax = loss_ax.twinx()

  loss_ax.plot(hist.history['loss'], 'y', label='train loss')
  loss_ax.plot(hist.history['val_loss'], 'r', label='val loss')
  loss_ax.set_xlabel('epoch')
  loss_ax.set_ylabel('loss')
  loss_ax.legend(loc='lower left')

  acc_ax.plot(hist.history['accuracy'], 'b', label='train acc')
  acc_ax.plot(hist.history['val_accuracy'], 'g', label='val acc')
  acc_ax.set_ylabel('accuracy')
  acc_ax.legend(loc='upper left')

  plt.show()

drawHistory(hist)
```



: 모델 학습동안 저장된 accuracy, loss 값을 그래프로 나타낸다.

학습이 진행됨에 따라 accuracy값은 상승하고 loss 값은 하락한다.(epoch을 110회 이상으로 하면 accuracy값이 하락하므로 100회로 제한)

```
print(test.shape)
```

```
(215, 8)
```

: test데이터의 데이터 형태 관찰(215행, 8열) //이때 8열->Outcome 제외 데이터 값

```
o = model.predict(test.values)
```

```
7/7 [==============================] - 0s 3ms/step
```

```
print(o)
```

```
[[0.9740073  0.02599269]
 [0.97321564 0.02678439]
 [0.9781999  0.02180014]
 [0.41456527 0.5854347 ]
 [0.32635868 0.6736413 ]
 [0.94969344 0.05030655]
 [0.27971315 0.7202869 ]
 [0.76116425 0.23883581]
 [0.8146624  0.18533759]
 [0.15214905 0.847851  ]
 [0.92336893 0.07663096]
 [0.9275937  0.07240622]
 [0.5748561  0.425144  ]
 [0.15498693 0.8450131 ]
 [0.9690429  0.03095713]
```

: 학습시킨 model을 통해 test데이터의 예측값(당뇨병 여부)을 o에 저장한 후 o를 출력한다. 이때 o의 값은 당뇨병에 걸릴 확률/걸리지 않을 확률로 나타난다.

```
o = np.argmax(o,-1)
```

```
print(o)
```

```
[0 0 0 1 1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0
 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1
 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1
 1 0 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 1 0 0 1 0 0 1 1 1 1 0 0 1 0 1 0 1
 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0
 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0]
```

1. o에 저장된 확률값을 더 큰 값을 1, 작은 값을 0으로 변환한다.(np.argmax 함수)

2. o 출력 시 당뇨병 여부가 0과 1로 나타난다.

```
answer = np.zeros([215,1])
answer[:,0] = o
```

```
df_answer=pd.DataFrame(answer)
df_answer.columns=['Outcome']
df_answer = df_answer.astype('int32')
df_answer.to_csv('test_pridict.csv', index = False)
```

```
print(df_answer)
```

```
     Outcome
0          0
1          0
2          0
3          1
4          1
..       ...
210        1
211        0
212        0
213        0
214        0

[215 rows x 1 columns]
```

1. answer에 215열, 1행으로 구성된 0으로 초기화된 배열을 저장

2. answer의 첫번째 행의 값은 o의 값으로 초기화

3. df_answer에 answer배열로 만든 dataframe 저장

4. df_answer의 행의 label 이름을 Outcome으로 초기화

5. df_answer의 값을 정수형으로 변환한다

6. df_answer를 test_pridict.csv(csv 파일)에 저장

7. df_answer 출력->test 데이터를 이용한 model의 예측값을 출력