DEEP LEARNING WITH KERAS

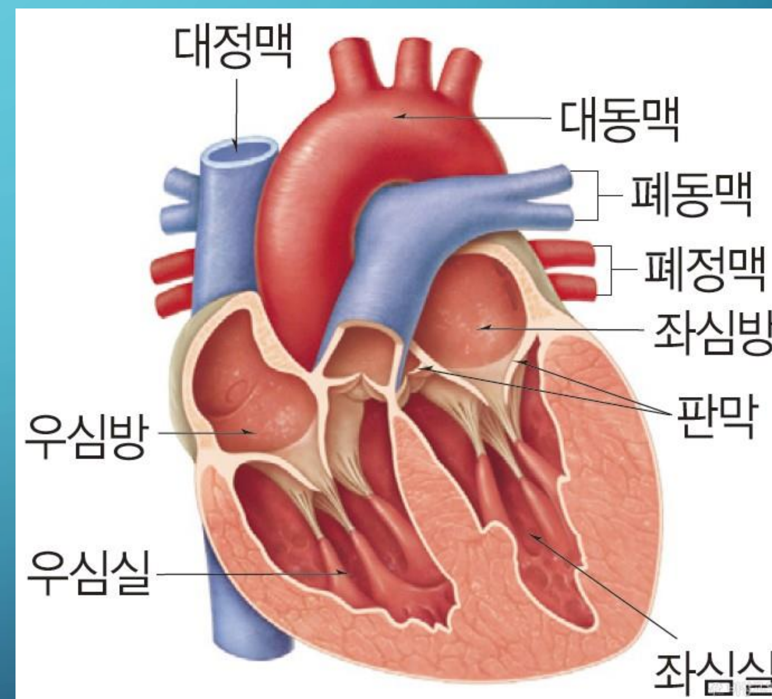
DNN – HEART DISEASE

한국인 사망 원인 순위 (2017)

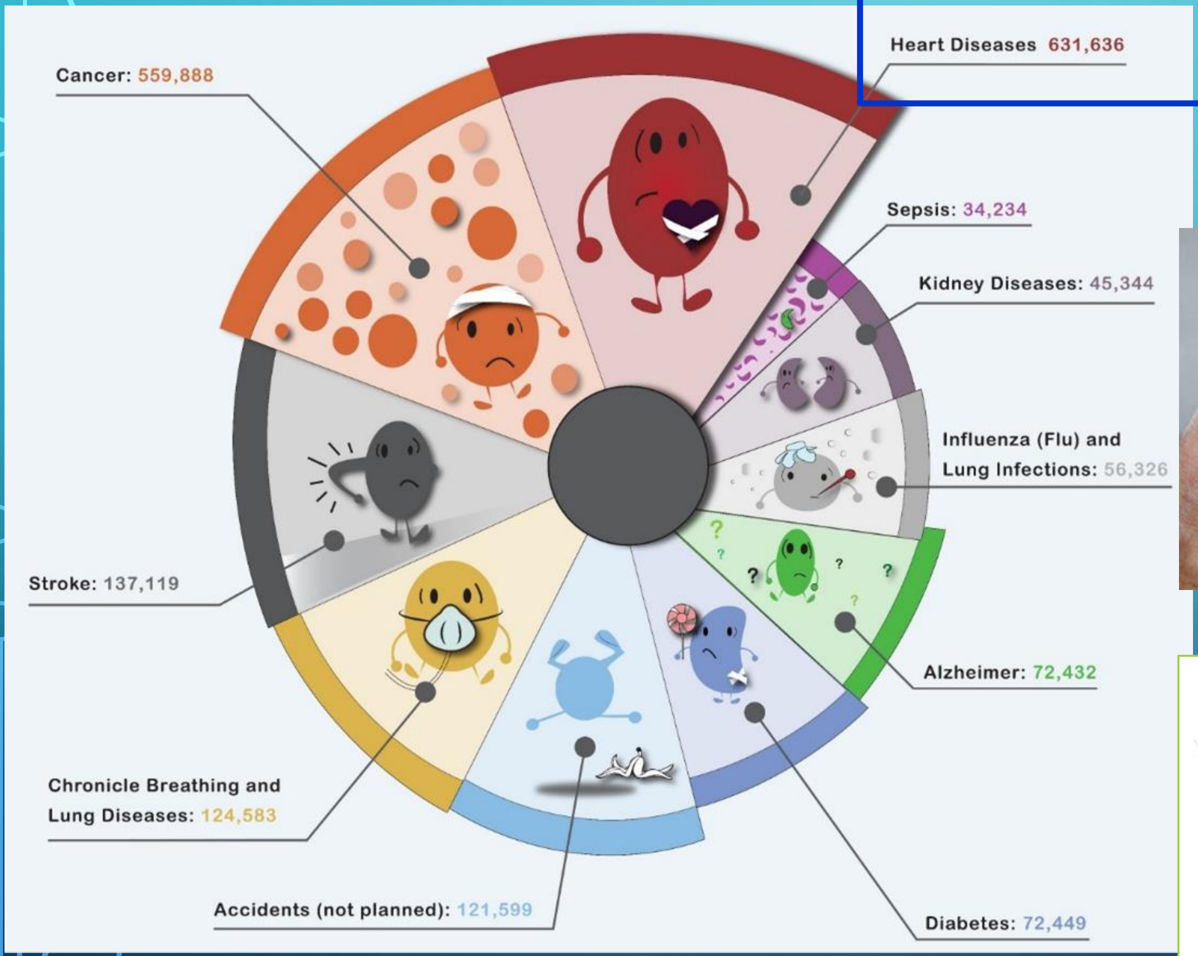
사망 원인 순위

	사망원인	사망자수(명)	사망률 (인구10만명당)
1위	악성신생물(암)	78,863	153.9
2위	심장 질환	30,852	60.2
3위	뇌혈관 질환	22,745	44.4
4위	폐렴	19,378	37.8
5위	자살	12,463	24.3
6위	당뇨병	9,184	17.9
7위	간질환	6,797	13.3
8위	만성하기도 질환	6,750	13.2
9위	고혈압성 질환	5,775	11.3
10위	운수 사고	5,028	9.8

출처 : 통계청, 2017년

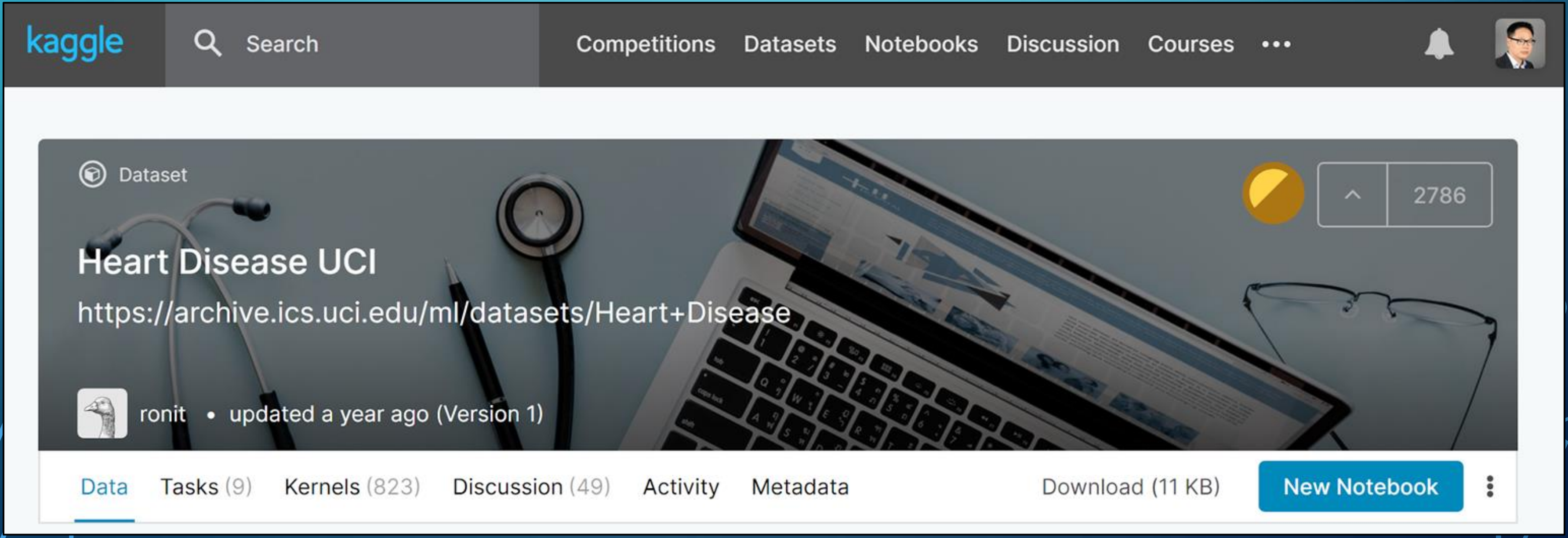


미국인 사망 원인 순위 (2019)



UCI HEART DISEASE 데이터셋

- 303명 환자의 데이터 (www.kaggle.com)
  - 14가지 항목의 정보 제공
- 기계학습의 목적
  - 입력: 13가지 특징, 출력: 환자의 심장병 여부



14가지 항목

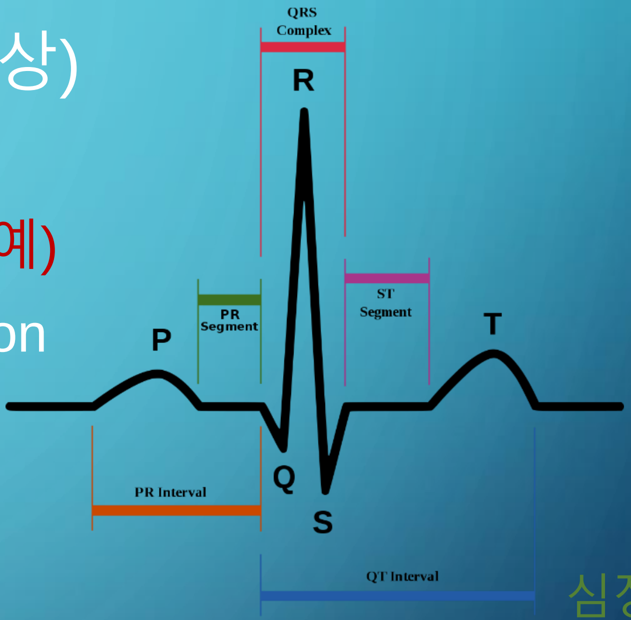
- 처음 7가지 (빨간색: 원핫 인코딩 대상)
  - age: 나이
  - sex: 성별, 0: 여성, 1: 남성
  - cp: 가슴 통증 유형 (4가지)
  - trestbps: 안정 혈압
  - chol: 좋은 콜레스테롤 (HDL)
  - fbs: 공복 혈당 (0: 120 보다 낮음, 1: 반대)
  - restecg: 안정 심전도 (3가지, 0: 정상)

심장병 여부

63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	0	108	1	1.5	2	3	3	1

14가지 항목

- 다음 7가지 (빨간색: 원핫 인코딩 대상)
  - thalach: 최대 심박동 수
  - exang: 운동시 협심증상 (0: 아니오, 1: 예)
  - oldpeak: 운동시 유발되는 ST depression
  - slope: 운동시 ST segment 기울기
  - ca: 형광투시되는 주요 혈관 수 (0-3)
  - thal: 결함 종류 (3: 정상, 6/7: 비정상)
  - HeartDisease: 심장병? 0 = no, 1 = yes



심장병 여부

63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
67	1	4	160	286	0	0	108	1	1.5	2	3	3	1

6. HEART-DISEASE.PY (1/11)

```
# import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense


# global constants and hyper-parameters
INPUT_DIM = 13
MY_EPOCH = 200
MY_BATCH = 32
MY_SPLIT = 0.3
```

6. HEART-DISEASE.PY (2/11)

```
    ####################
    # DATABASE SETTING #
    ####################



# read DB file
heading = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs’,
        'restecg', 'thalach', 'exang', 'oldpeak', 'slope’,
        'ca', 'hal', 'HeartDisease']
file_name = "heart.xlsx"
raw_DB= pd.read_excel(file_name, header = None, names = hea
ding)
```

6. HEART-DISEASE.PY (3/11)

```
# print raw data using pandas data frame
# describe() collects DB statistics
print('\n== FIRST 20 RAW DATA ==')
print(raw_DB.head(20))
summary = raw_DB.describe()
print('\n== SUMMARY OF RAW DATA ==')
print(summary)
print('\n== RAW DATA BEFORE CLEAN UP ==')
print(raw_DB.info())
```

6. HEART-DISEASE.PY (4/11)

```
# handling missing entries in the BD
# our DB contains "?". try searching with "~?" in excel.
# first, we replace "?" with "nan" (not-a-number)
# second, we drop the rows that contain "nan"
clean_DB = raw_DB.replace('?', np.nan)
clean_DB = clean_DB.dropna()

print('\n== RAW DATA AFTER DROPING NAN ROWS ==')
print(clean_DB.info())
```

6. HEART-DISEASE.PY (5/11)

```
# split DB (14 columns) into inputs (13) vs. output (1) fir
st
# so that we scale only the inputs
# output scaling is not useful as it is binary decision
print('\n== DB SHAPE INFO ==')
print('DB shape = ', clean_DB.shape)
keep = heading.pop()
Input = pd.DataFrame(clean_DB.iloc[:, 0:INPUT_DIM],
        columns = heading)
Target = pd.DataFrame(clean_DB.iloc[:, INPUT_DIM],
        columns = [keep])
```

6. HEART-DISEASE.PY (6/11)

```
# scaling with z-score: z = (x - u) / s
# so that mean becomes 0, and standard deviation 1
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_DB = scaler.fit_transform(Input)

# collect scaled DB stats using described()
# framing is needed after scaling
scaled_DB = pd.DataFrame(scaled_DB, columns = heading)
summary = scaled_DB.describe()
summary = summary.transpose()
```

FIT_TRANSFORM()

```
# conducts data centering
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# we must use 2D array
my_array = [[1], [2], [3], [6]]
x = scaler.fit_transform(my_array)
print(x)

# converts scaled data back to original
y = scaler.inverse_transform(x)
print(y)
```

6. HEART-DISEASE.PY (7/11)

```
# display box plot of scaled DB
boxplot = scaled_DB.boxplot(column = heading, showmeans = T
rue)
print('\n== BOX PLOT OF SCALED DATA ==')
plt.show()

# split the DB into training and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(scaled_
DB, Target, test_size = MY_SPLIT, random_state = 5)
```

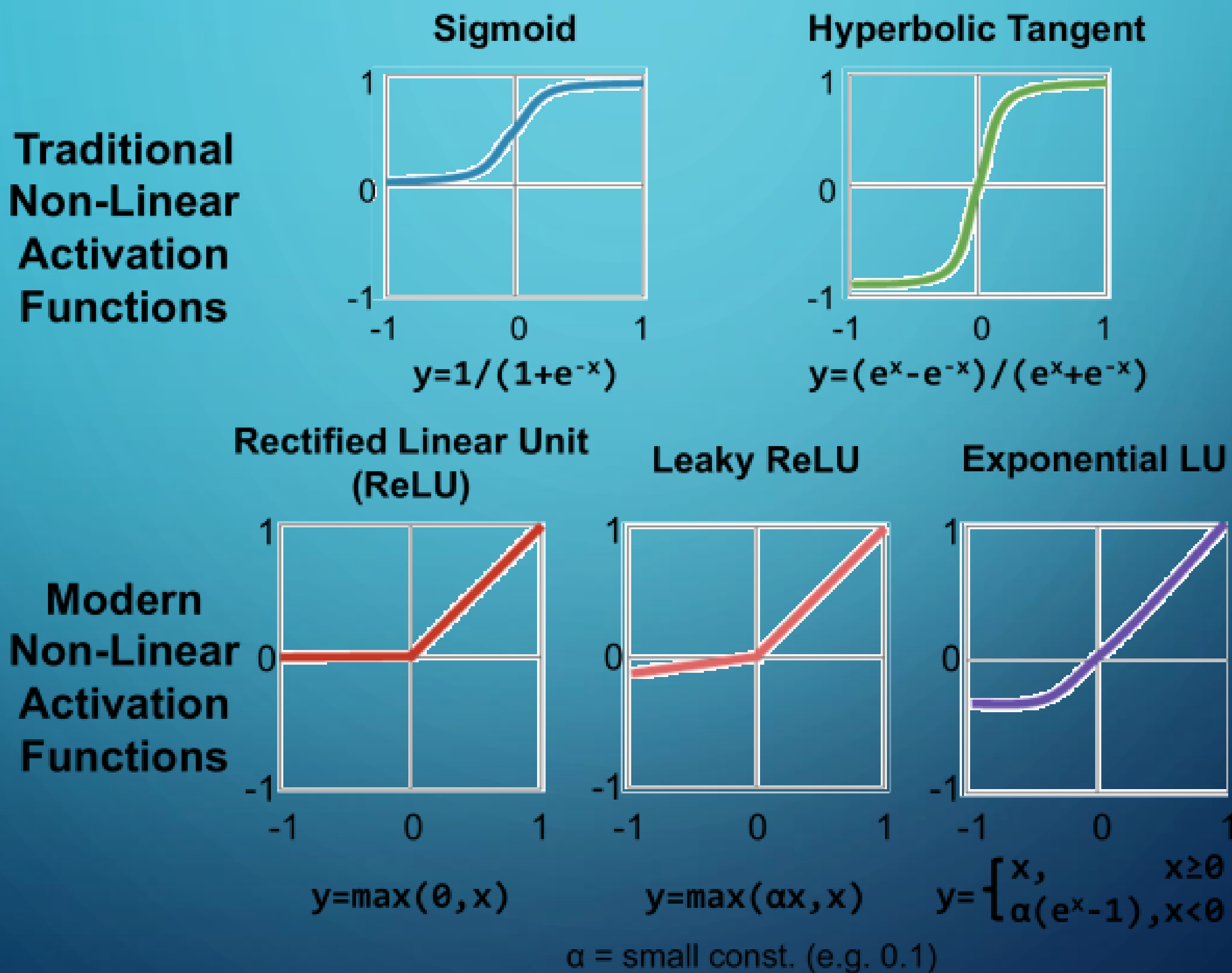
6. HEART-DISEASE.PY (8/11)

```
    ###############################
    # MODEL BUILDING AND TRAINING #
    ###############################



# build a keras sequential model of our DNN
model = Sequential()
model.add(Dense(1000, input_dim = INPUT_DIM, activation = '
tanh'))
model.add(Dense(1000, activation = 'tanh'))
model.add(Dense(1, activation = 'sigmoid'))
model.summary()
```

POPULAR ACTIVATION FUNCTIONS



HOW DO THEY COMPARE?

- ReLU is a fast learner and used in intermediate layers
- ReLU is popular in CNN
- For DNN, it is advisable to start experiments with ReLU
- Tanh is slow, but less prone to gradient vanishing
- Leaky ReLU can be the first solution to gradient vanishing
- Softmax is usually used in output layers for classification

6. HEART-DISEASE.PY (9/11)

```
# model training and saving
model.compile(optimizer = 'adam', loss = 'binary_crossentro
py', metrics = ['accuracy’])

model.fit(X_train, Y_train, epochs = MY_EPOCH, batch_size =
        MY_BATCH, verbose = 1)

model.save('chap6.h5')
```

6. HEART-DISEASE.PY (10/11)

```
    #####################
    # MODEL EVALUATION #
    #####################


# model evaluation
score = model.evaluate(X_test, Y_test, verbose = 1)
print('\nKeras DNN model loss = ', score[0])
print('Keras DNN model accuracy = ', score[1])
```

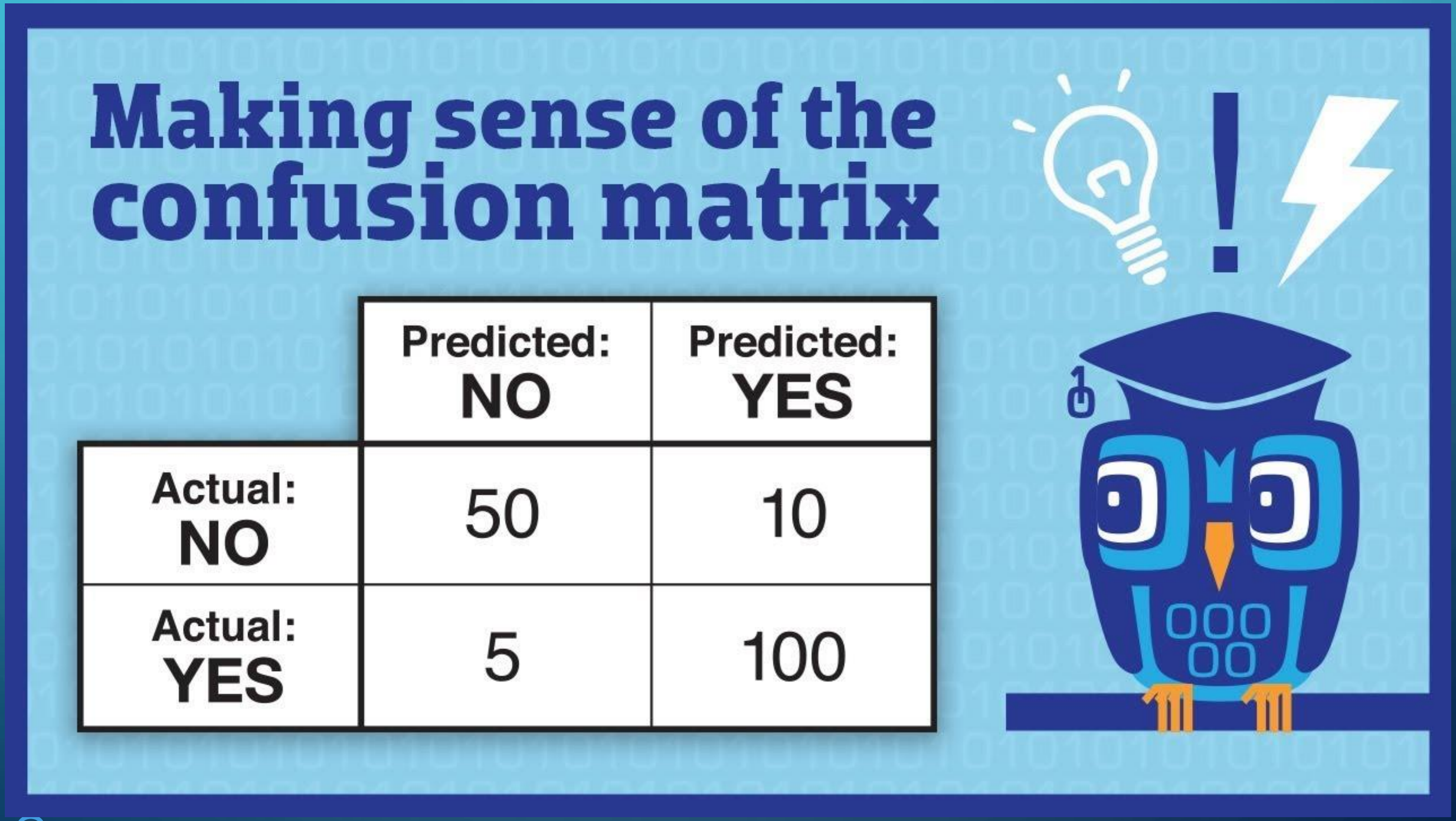
6. HEART-DISEASE.PY (11/11)

```
# display confusion matrix
# the third line converts [0, 1] into true/false
from sklearn.metrics import confusion_matrix
pred = model.predict(X_test)
pred = (pred > 0.5)
print('\n== CONFUSION MATRIX ==')
print(confusion_matrix(Y_test, pred))

# calculate F1 score using confusion matrix
from sklearn.metrics import f1_score
print("\nF1 score:", f1_score(Y_test, pred, average = 'micr
o'))
```

CONFUSION MATRIX

- Useful in multi-class classification



	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100