



# Aichemist Train Session

CHAP 03 평가

# 실습 목차

## 01. 시작하기 전에

01-1. Guide-Line

01-2. 자료 다운

## 02. 피마 인디언 당뇨병 예측




## Guide-Line

- Github에서 Notebook file 다운 & Kaggle 사이트 참고하기
- 조원과 PPT 내 질문에 답하기
- 평가 챕터의 각 모듈과 기능 숙지하기
- 총 4팀 발표 🪜

# 자료 다운

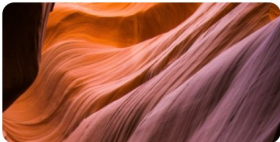
깃허브 주피터 노트북 다운:

 UCI MACHINE LEARNING AND 1 COLLABORATOR · UPDATED 7 YEARS AGO

▲ 4043 New Notebook Download (9 kB) ● ⋮

## Pima Indians Diabetes Database

Predict the onset of diabetes based on diagnostic measures



[Data Card](#) [Code \(2844\)](#) [Discussion \(51\)](#) [Suggestions \(0\)](#)

---

### About Dataset

#### Context

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

#### Content

The datasets consists of several medical predictor variables and one target variable, `Outcome`. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

#### Acknowledgements

Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. *In Proceedings of the Symposium on Computer Applications and Medical Care* (pp. 261--265). IEEE Computer Society Press.

#### Inspiration

Can you build a machine learning model to accurately predict whether or not the patients in the dataset have diabetes or not?

#### Usability

8.82

#### License

[CC0: Public Domain](#)

#### Expected update frequency

Not specified

#### Tags

Earth and NatureHealth

DiabetesIndia

Healthcare

캐글 데이터셋 다운:

<https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

# Data 살펴보기

```
In [2]: diabetes_data = pd.read_csv('diabetes.csv')
print(diabetes_data['Outcome'].value_counts())
diabetes_data.head(3)
```

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

Q1. 레이블 값 분포를 보고, 주의해야 할 점?

```
Out[2]:
```

	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

Q2. 의심스러운 데이터가 있나?

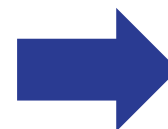
Q2-1. 그 데이터에 고려해볼 수 있는 전처리 방식?

- Pregnancies: 임신 횟수
- Glucose: 포도당 부하 검사 수치
- BloodPressure: 혈압(mm Hg)
- SkinThickness: 팔 삼두근 뒤쪽의 피하지방 측정값(mm)
- Insuline: 혈청 인슐린(mu U/Hg)
- BMI: 체질량지수(체중(kg)/(키(m))^2)
- DiabetesPedigreeFunction: 당뇨 내력 가중치 값
- Age: 나이
- Outcome: 클래스 결정 값(0 또는 1)

# Data 살펴보기

In [3]: `diabetes_data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Pregnancies                          768 non-null    int64
1   Glucose                              768 non-null    int64
2   BloodPressure                        768 non-null    int64
3   SkinThickness                       768 non-null    int64
4   Insulin                             768 non-null    int64
5   BMI                                  768 non-null    float64
6   DiabetesPedigreeFunction             768 non-null    float64
7   Age                                  768 non-null    int64
8   Outcome                             768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```



Q3. 고려해볼 수 있는 전처리 방식?

Q3-1. 고려하지 않아도 되는 전처리 방식?

# Logistic Regression 학습/예측/평가

In [4]: `X = diabetes_data.iloc[:, :-1]` Q4. DataFrame의 구조에서 `iloc[:, :-1]`은 무엇을 의미?  
`y = diabetes_data.iloc[:, -1]`

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 156, stratify=y)
```

```
lr_clf = LogisticRegression(solver='liblinear')  
lr_clf.fit(X_train, y_train)  
pred = lr_clf.predict(X_test)  
pred_proba = lr_clf.predict_proba(X_test)[:, 1]
```

```
get_clf_eval(y_test, pred, pred_proba)
```

오차 행렬

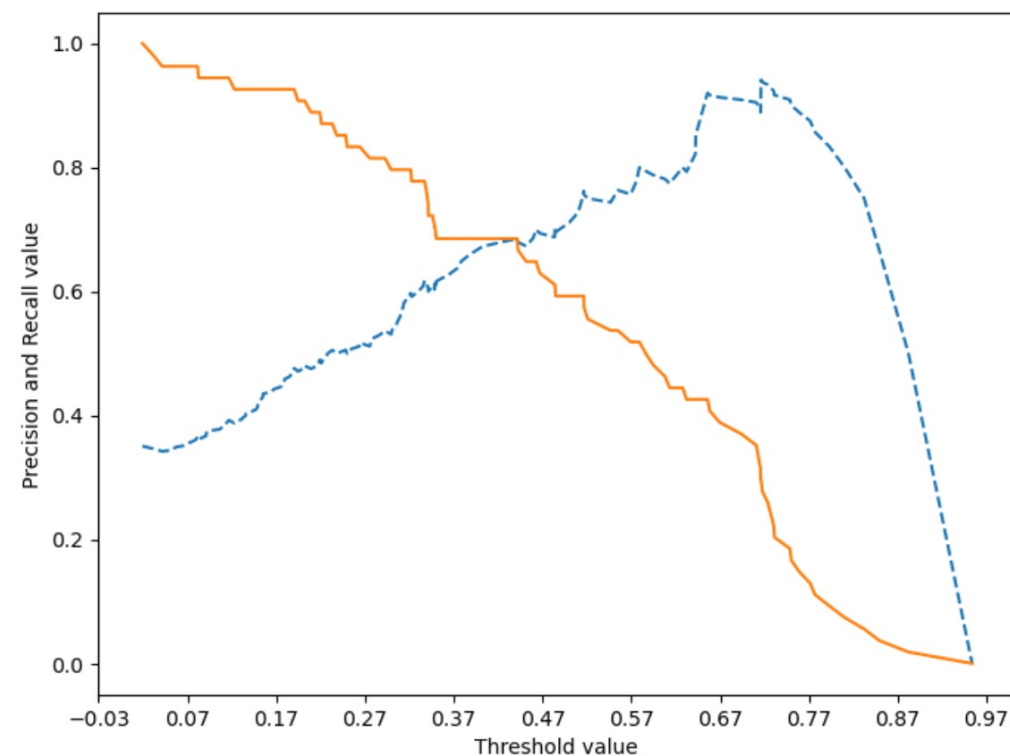
[[87 13]

[22 32]]

정확도: 0.7727, 정밀도: 0.7111, 재현율: 0.5926, F1: 0.6465, AUC: 0.8083

Q5. stratify 매개변수는 교차검증의 'stratified K 폴드' 처럼  
원본 데이터셋에서 클래스의 분포가 어떤 비율로 구성되어 있  
는지를 고려하여, 그 비율을 유지하도록 분할합니다. 그 이유는?

```
pred_proba_c1 = lr_clf.predict_proba(X_test)[:, 1]  
precision_recall_curve_plot(y_test, pred_proba_c1)
```



Q6. 현재 정밀도, 재현율의 문제점?

## 피쳐 값 분포도 확인

In [6]: `diabetes_data.describe()`

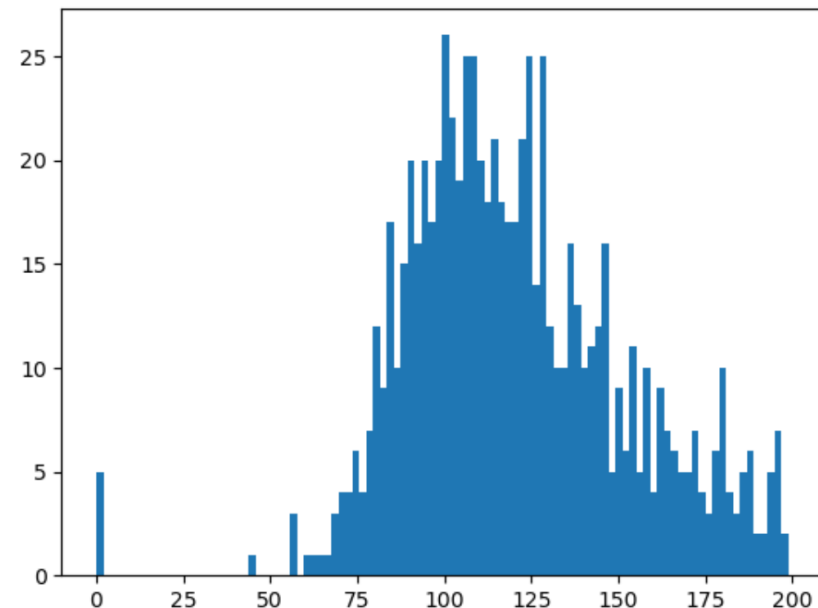
### Q7. describe()의 역할?

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

### Q8. 현재 데이터 세트의 문제점?

In [7]: `plt.hist(diabetes_data['Glucose'], bins=100)`  
`plt.show()`



### Q9. 위 데이터 분포로 생각해볼 수 있는 전처리 방식?



## 피처 값 분포도 확인

```
In [8]: zero_features = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']

total_count = diabetes_data['Glucose'].count()

for feature in zero_features:
    zero_count = diabetes_data[diabetes_data[feature] == 0][feature].count()
    print('{0} 0 건수는 {1}, 퍼센트는 {2:.2f}%'.format(feature, zero_count, 100*zero_count/total_count))
```

Glucose 0 건수는 5, 퍼센트는 0.65%  
BloodPressure 0 건수는 35, 퍼센트는 4.56%  
SkinThickness 0 건수는 227, 퍼센트는 29.56%  
Insulin 0 건수는 374, 퍼센트는 48.70%  
BMI 0 건수는 11, 퍼센트는 1.43%

Q10. 0의 비율을 보고 고려할 수 있는 전처리 방식?

## 전처리 후 학습/예측/평가

```
In [9]: mean_zero_features = diabetes_data[zero_features].mean()  
diabetes_data[zero_features]=diabetes_data[zero_features].replace(0, mean_zero_features)
```

Q11. 0의 비율이 많은 피쳐도 평균값으로 대체한 이유는?

```
In [10]: X = diabetes_data.iloc[:, :-1]  
y = diabetes_data.iloc[:, -1]
```

```
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(X)
```

Q12. 표준화 적용한 이유?

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.2, random_state = 156, stratify = y)
```

```
lr_clf = LogisticRegression()  
lr_clf.fit(X_train, y_train)  
pred = lr_clf.predict(X_test)  
pred_proba = lr_clf.predict_proba(X_test)[:, 1]
```

```
get_clf_eval(y_test, pred, pred_proba)
```

오차 행렬

[[90 10]

[21 33]]

정확도: 0.7987, 정밀도: 0.7674, 재현율: 0.6111, F1: 0.6804, AUC: 0.8433

Q13. 전처리 후 성능 변화?

# 임계값에 따른 평가 수치

```
In [11]: thresholds = [0.3, 0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.50]
pred_proba = lr_clf.predict_proba(X_test)
get_eval_by_threshold(y_test, pred_proba[:, 1].reshape(-1, 1), thresholds)
```

```
임계값: 0.3
오차 행렬
[[67 33]
 [11 43]]
정확도: 0.7143, 정밀도: 0.5658, 재현율: 0.7963, F1: 0.6615, AUC: 0.8433
임계값: 0.33
오차 행렬
[[72 28]
 [12 42]]
정확도: 0.7403, 정밀도: 0.6000, 재현율: 0.7778, F1: 0.6774, AUC: 0.8433
임계값: 0.36
오차 행렬
[[76 24]
 [15 39]]
정확도: 0.7468, 정밀도: 0.6190, 재현율: 0.7222, F1: 0.6667, AUC: 0.8433
임계값: 0.39
오차 행렬
[[78 22]
 [16 38]]
정확도: 0.7532, 정밀도: 0.6333, 재현율: 0.7037, F1: 0.6667, AUC: 0.8433
```

```
임계값: 0.42
오차 행렬
[[84 16]
 [18 36]]
정확도: 0.7792, 정밀도: 0.6923, 재현율: 0.6667, F1: 0.6792, AUC: 0.8433
임계값: 0.45
오차 행렬
[[85 15]
 [18 36]]
정확도: 0.7857, 정밀도: 0.7059, 재현율: 0.6667, F1: 0.6857, AUC: 0.8433
임계값: 0.48
오차 행렬
[[88 12]
 [19 35]]
정확도: 0.7987, 정밀도: 0.7447, 재현율: 0.6481, F1: 0.6931, AUC: 0.8433
임계값: 0.5
오차 행렬
[[90 10]
 [21 33]]
정확도: 0.7987, 정밀도: 0.7674, 재현율: 0.6111, F1: 0.6804, AUC: 0.8433
```

Q14. 어떤 임계값으로 설정하면 좋을까? 그 이유는?

## 임계값 설정 후 예측/평가

```
In [12]: binarizer = Binarizer(threshold = 0.48)

pred_th_048 = binarizer.fit_transform(pred_proba[:, 1].reshape(-1, 1))

get_clf_eval(y_test, pred_th_048, pred_proba[:, 1])
```

오차 행렬

[[88 12]

[19 35]]

정확도: 0.7987, 정밀도: 0.7447, 재현율: 0.6481, F1: 0.6931, AUC: 0.8433

Q15. Binarizer 클래스 사용한 이유?



수고하셨습니다