Data Science
#HW2

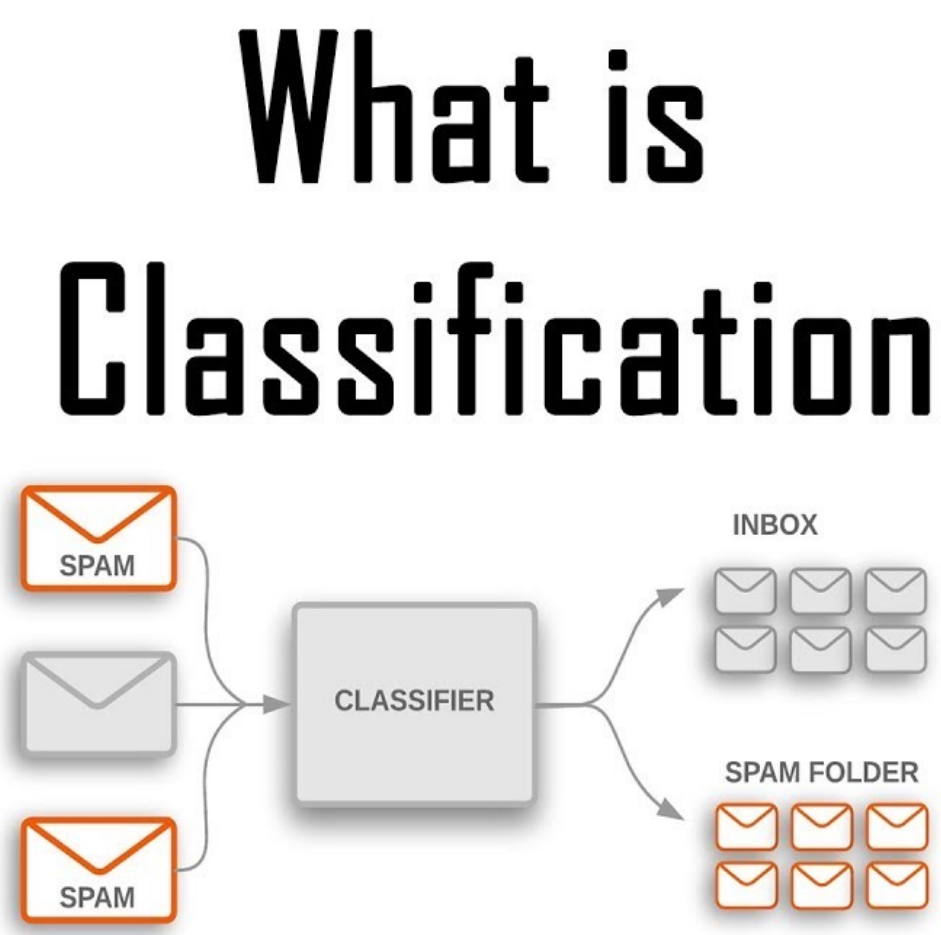
“Classification” Binary & Multi Class

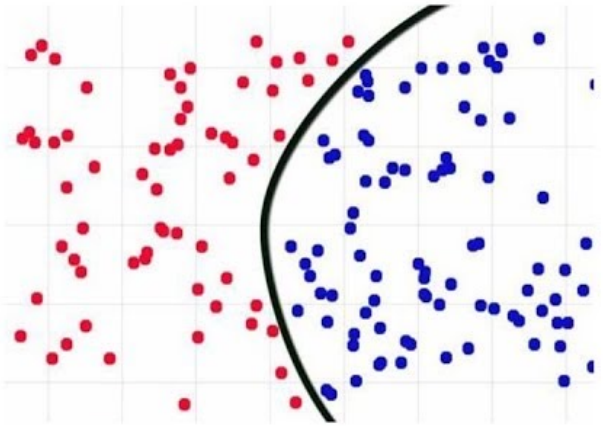
산업공학과 2019911532 송용재 (2023.10)

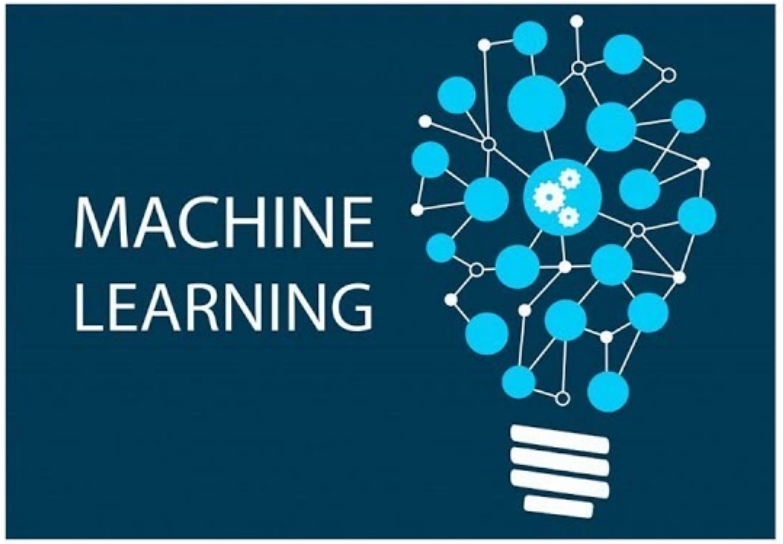
Table of Contents

Preface

분류 예측을 통해, 어느 집단에 속한 것일지 판단하기

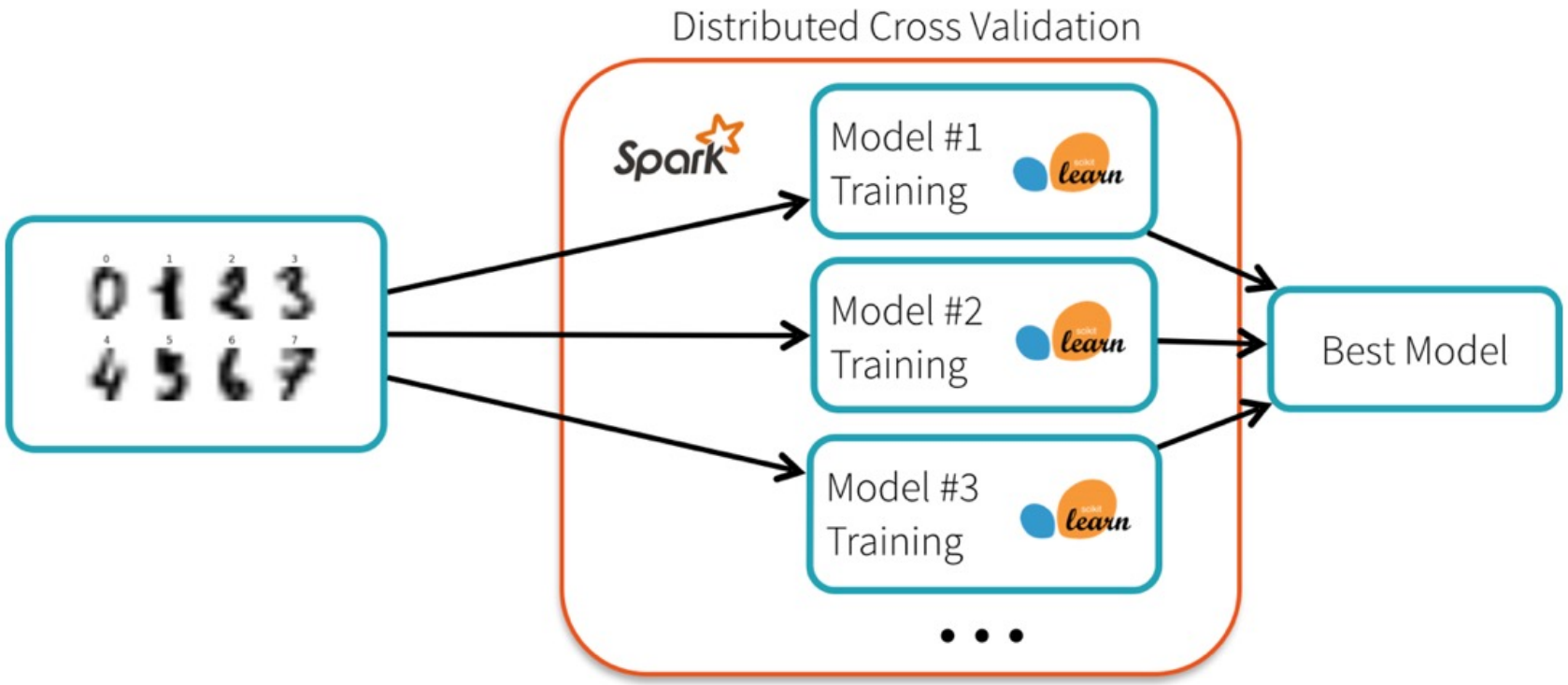






Preface

여러 모델들을 사용하고, 이를 평가하여 '과연 최적의 모델이 무엇일까?'에 대한 고찰



Binary Class

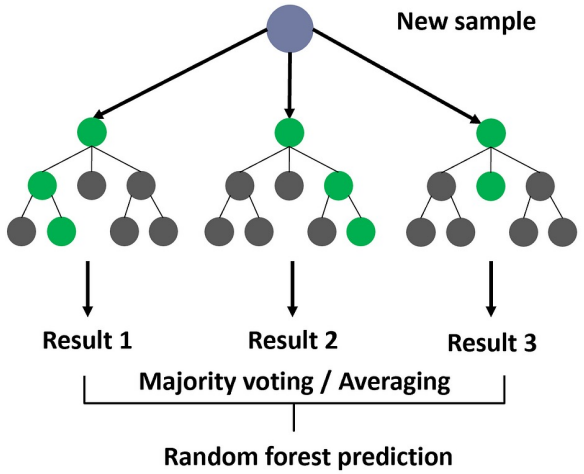
Binary Class(loan.data)

method: LogisticRegression, RandomForest, AdaBoost

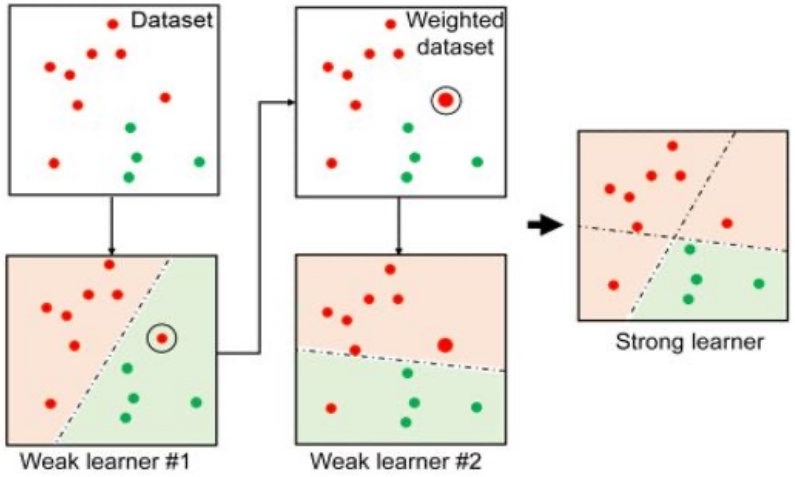
```
[1]
from db_conn import *
from pprint import pprint
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from scipy.special import expit
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
```

Logistic Regression

RandomForest



AdaBoost



loan.data.csv

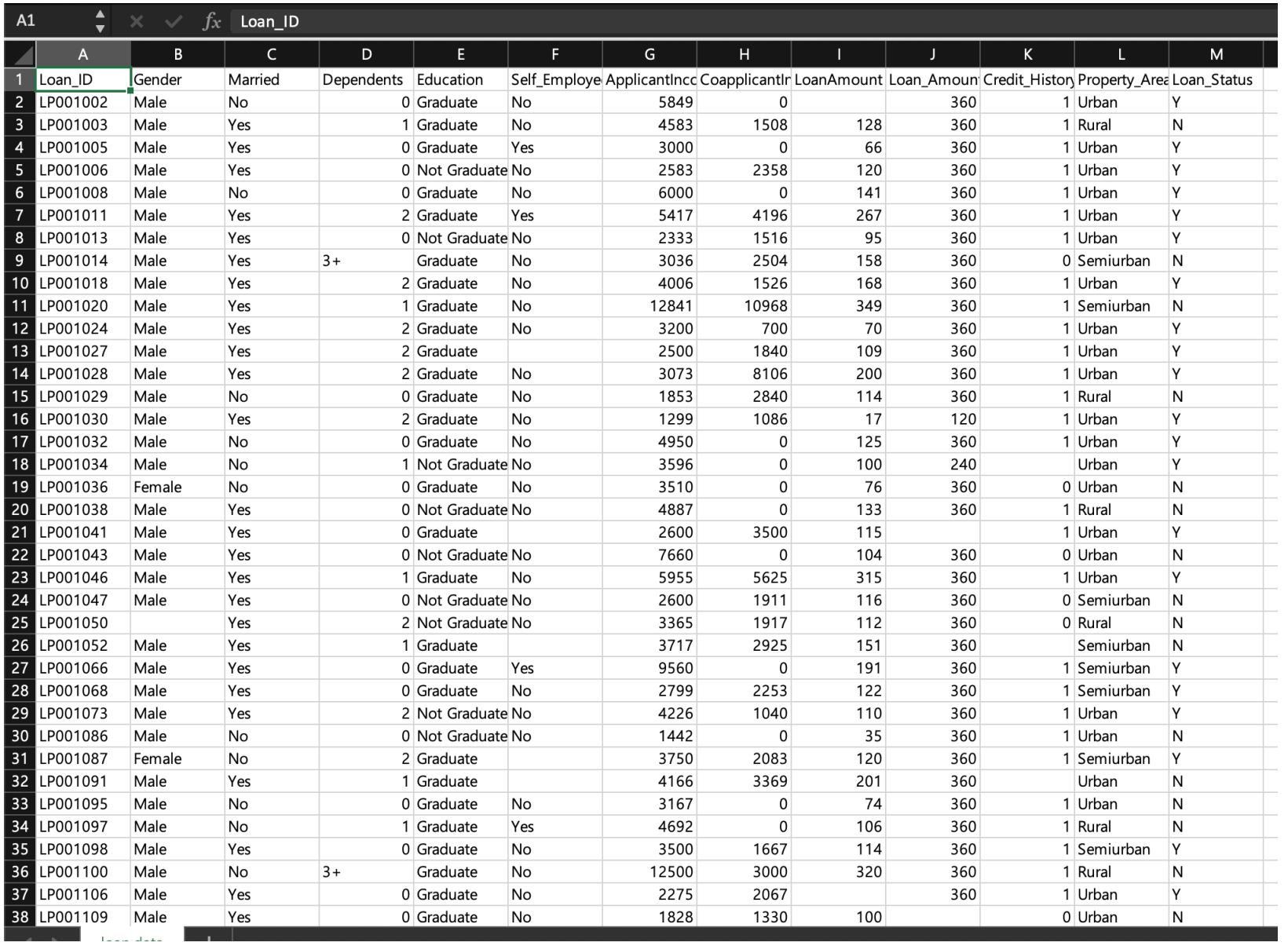
csv 데이터 설명: 대출 여부에 대한 Yes/No를 판단, 본 과제에서는 'ApplicantIncome'과, 'LoanAmount'를 활용.

Dataset1: Loan Prediction Problem

- Loan_ID: 대출 신청 고유 식별자.
- Gender: 신청자 성별. 'Male' 또는 'Female' 값.
- Married: 신청자 결혼 여부. 'Yes' 또는 'No' 값.
- Dependents: 신청자 부양 가족 수.
- Education: 신청자 교육 수준. 'Graduate' 또는 'Not Graduate' 값.
- Self_Employed: 신청자가 자영업자인지 여부. 'Yes' 또는 'No' 값.
- ApplicantIncome: 신청자 월별 소득.
- CoapplicantIncome: 공동 신청자 월별 소득.
- LoanAmount: 신청 대출 금액.
- Loan_Amount_Term: 대출 기간, 월로 표시.
- Credit_History: 신청자 신용 기록. 1은 긍정적, 0은 부정적, NaN은 알 수 없음.
- Property_Area: 부동산 위치 지역. 'Urban', 'Semiurban', 'Rural' 등의 값.

- Loan_Status: Y or N

excel



A1 | Loan_ID

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Loan_ID	Gender	Married	Dependents	Education	Self_Employe	ApplicantIncc	CoapplicantIr	LoanAmount	Loan_Amoun	Credit_Histor	Property_Area	Loan_Status
2	LP001002	Male	No	0	Graduate	No	5849	0		360	1	Urban	Y
3	LP001003	Male	Yes	1	Graduate	No	4583	1508	128	360	1	Rural	N
4	LP001005	Male	Yes	0	Graduate	Yes	3000	0	66	360	1	Urban	Y
5	LP001006	Male	Yes	0	Not Graduate	No	2583	2358	120	360	1	Urban	Y
6	LP001008	Male	No	0	Graduate	No	6000	0	141	360	1	Urban	Y
7	LP001011	Male	Yes	2	Graduate	Yes	5417	4196	267	360	1	Urban	Y
8	LP001013	Male	Yes	0	Not Graduate	No	2333	1516	95	360	1	Urban	Y
9	LP001014	Male	Yes	3+	Graduate	No	3036	2504	158	360	0	Semiurban	N
10	LP001018	Male	Yes	2	Graduate	No	4006	1526	168	360	1	Urban	Y
11	LP001020	Male	Yes	1	Graduate	No	12841	10968	349	360	1	Semiurban	N
12	LP001024	Male	Yes	2	Graduate	No	3200	700	70	360	1	Urban	Y
13	LP001027	Male	Yes	2	Graduate		2500	1840	109	360	1	Urban	Y
14	LP001028	Male	Yes	2	Graduate	No	3073	8106	200	360	1	Urban	Y
15	LP001029	Male	No	0	Graduate	No	1853	2840	114	360	1	Rural	N
16	LP001030	Male	Yes	2	Graduate	No	1299	1086	17	120	1	Urban	Y
17	LP001032	Male	No	0	Graduate	No	4950	0	125	360	1	Urban	Y
18	LP001034	Male	No	1	Not Graduate	No	3596	0	100	240		Urban	Y
19	LP001036	Female	No	0	Graduate	No	3510	0	76	360	0	Urban	N
20	LP001038	Male	Yes	0	Not Graduate	No	4887	0	133	360	1	Rural	N
21	LP001041	Male	Yes	0	Graduate		2600	3500	115		1	Urban	Y
22	LP001043	Male	Yes	0	Not Graduate	No	7660	0	104	360	0	Urban	N
23	LP001046	Male	Yes	1	Graduate	No	5955	5625	315	360	1	Urban	Y
24	LP001047	Male	Yes	0	Not Graduate	No	2600	1911	116	360	0	Semiurban	N
25	LP001050		Yes	2	Not Graduate	No	3365	1917	112	360	0	Rural	N
26	LP001052	Male	Yes	1	Graduate		3717	2925	151	360		Semiurban	N
27	LP001066	Male	Yes	0	Graduate	Yes	9560	0	191	360	1	Semiurban	Y
28	LP001068	Male	Yes	0	Graduate	No	2799	2253	122	360	1	Semiurban	Y
29	LP001073	Male	Yes	2	Not Graduate	No	4226	1040	110	360	1	Urban	Y
30	LP001086	Male	No	0	Not Graduate	No	1442	0	35	360	1	Urban	N
31	LP001087	Female	No	2	Graduate		3750	2083	120	360	1	Semiurban	Y
32	LP001091	Male	Yes	1	Graduate		4166	3369	201	360		Urban	N
33	LP001095	Male	No	0	Graduate	No	3167	0	74	360	1	Urban	N
34	LP001097	Male	No	1	Graduate	Yes	4692	0	106	360	1	Rural	N
35	LP001098	Male	Yes	0	Graduate	No	3500	1667	114	360	1	Semiurban	Y
36	LP001100	Male	No	3+	Graduate	No	12500	3000	320	360	1	Rural	N
37	LP001106	Male	Yes	0	Graduate	No	2275	2067		360	1	Urban	Y
38	LP001109	Male	Yes	0	Graduate	No	1828	1330	100		0	Urban	N

db_conn.py

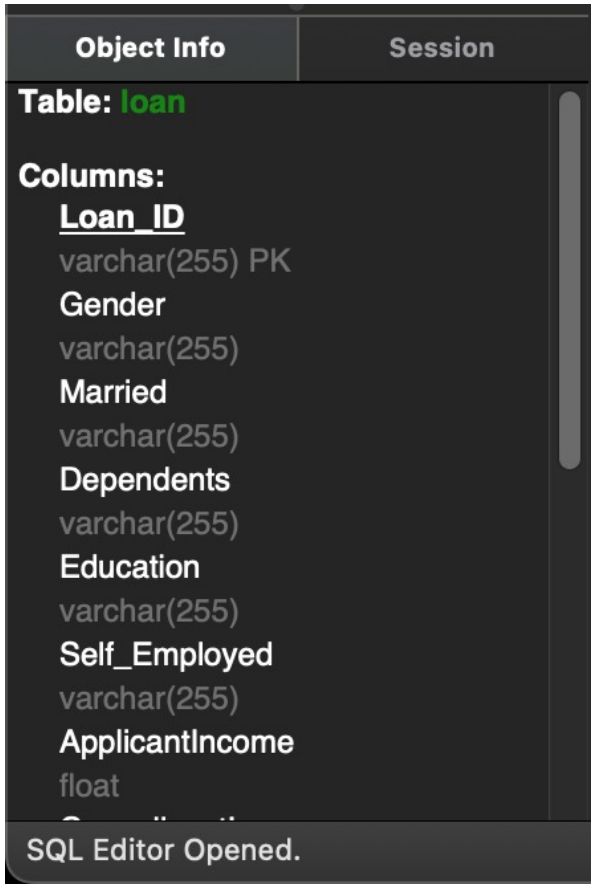
pymysql 라이브러리 활용, MySQL Workbench에 저장된 데이터를 활용하기

```python
class class_loan_classification():
    def __init__(self, import_data_flag=True):
        self.conn, self.cur = open_db()
        if import_data_flag:
            self.import_loan_data()
```

db_conn.py > close_db

```python
import pymysql

from pymysql.constants.CLIENT import MULTI_STATEMENTS

def open_db(dbname='DS2023'):

    conn =pymysql.connect(host='localhost',
                          user='root',
                          passwd='Steve72041!',
                          db=dbname,
                          client_flag=MULTI_STATEMENTS,
                          charset='utf8mb4')

    cursor = conn.cursor(pymysql.cursors.DictCursor)

    return conn, cursor

def close_db(conn, cur):
    cur.close()
    conn.close()

if __name__ == '__main__':
    conn, cur = open_db()
    close_db(conn, cur)

```

```python
def import_loan_data(self):
    drop_sql =""" drop table if exists loan;"""
    self.cur.execute(drop_sql)
    self.conn.commit()

    create_sql = """
        create table loan (
            Loan_ID varchar(255) PRIMARY KEY,
            Gender varchar(255),
            Married varchar(255),
            Dependents varchar(255),
            Education varchar(255),
            Self_Employed varchar(255),
            ApplicantIncome float,
            CoapplicantIncome float,
            LoanAmount float,
            Loan_Amount_Term float,
            Credit_History INT,
            Property_Area varchar(255),
            Loan_Status varchar(255)
            );
    """

    self.cur.execute(create_sql)
    self.conn.commit()

    file_name = 'loan.data.csv'
    loan_data = pd.read_csv(file_name)

    for col in loan_data.columns:
        loan_data[col].fillna(0, inplace=True)
```



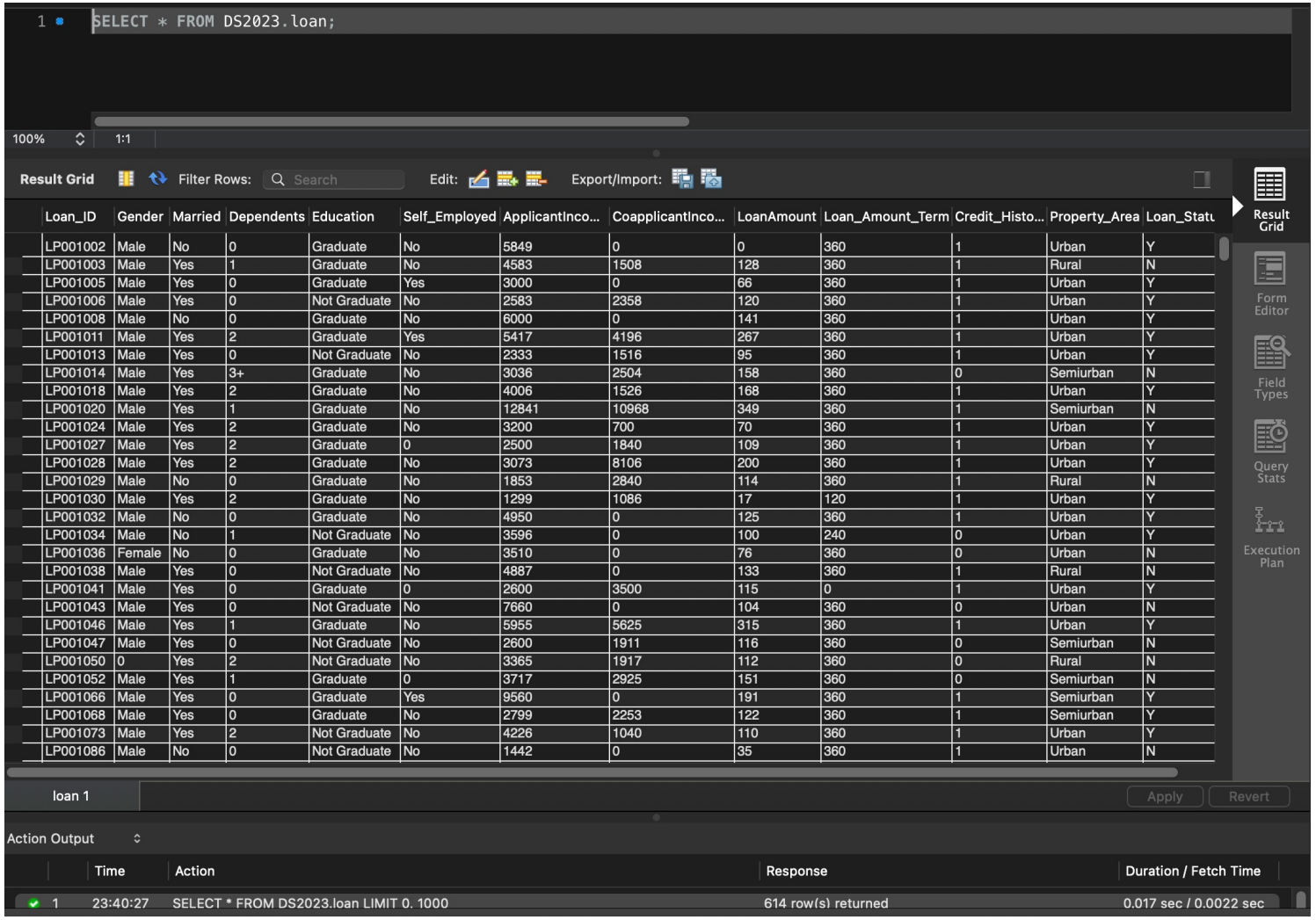
Input

Insert into “ “ 구문 활용, csv데이터의 값들을 Workbench로 기입하고, 이 중 ‘ApplicanIncome’과 ‘LoanAmount’ 활용

```
        rows = []

        insert_sql = """insert into loan
(Loan_ID,Gender,Married,Dependents,Education,Self_Employed,ApplicantIncome,CoapplicantIncome,LoanAmount,Loan_Amount_Term,Credit_History,Property_
Area,Loan_Status)
                    values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s);"""

        for t in loan_data.values:
            rows.append(tuple(t))

        self.cur.executemany(insert_sql, rows)
        self.conn.commit()

    def load_data_for_binary_classification(self, Loan_Status):
        sql = "select * from loan;"
        self.cur.execute(sql)

        data = self.cur.fetchall()

        self.X = [ (t['ApplicantIncome'], t['LoanAmount'] ) for t in data ]
        self.X = np.array(self.X)

        self.y = [ 1 if (t['Loan_Status'] == Loan_Status) else 0 for t in data]
        self.y = np.array(self.y)
```

MySQL Workbench

Workbench 안에 정상적으로 기입이 된 모습



```
1 • SELECT * FROM DS2023.loan;
```

100% 1:1

Result Grid | Filter Rows: Search | Edit: | Export/Import:

Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantInco...	CoapplicantInco...	LoanAmount	Loan_Amount_Term	Credit_Histo...	Property_Area	Loan_Statu
LP001002	Male	No	0	Graduate	No	5849	0	0	360	1	Urban	Y
LP001003	Male	Yes	1	Graduate	No	4583	1508	128	360	1	Rural	N
LP001005	Male	Yes	0	Graduate	Yes	3000	0	66	360	1	Urban	Y
LP001006	Male	Yes	0	Not Graduate	No	2583	2358	120	360	1	Urban	Y
LP001008	Male	No	0	Graduate	No	6000	0	141	360	1	Urban	Y
LP001011	Male	Yes	2	Graduate	Yes	5417	4196	267	360	1	Urban	Y
LP001013	Male	Yes	0	Not Graduate	No	2333	1516	95	360	1	Urban	Y
LP001014	Male	Yes	3+	Graduate	No	3036	2504	158	360	0	Semiurban	N
LP001018	Male	Yes	2	Graduate	No	4006	1526	168	360	1	Urban	Y
LP001020	Male	Yes	1	Graduate	No	12841	10968	349	360	1	Semiurban	N
LP001024	Male	Yes	2	Graduate	No	3200	700	70	360	1	Urban	Y
LP001027	Male	Yes	2	Graduate	0	2500	1840	109	360	1	Urban	Y
LP001028	Male	Yes	2	Graduate	No	3073	8106	200	360	1	Urban	Y
LP001029	Male	No	0	Graduate	No	1853	2840	114	360	1	Rural	N
LP001030	Male	Yes	2	Graduate	No	1299	1086	17	120	1	Urban	Y
LP001032	Male	No	0	Graduate	No	4950	0	125	360	1	Urban	Y
LP001034	Male	No	1	Not Graduate	No	3596	0	100	240	0	Urban	Y
LP001036	Female	No	0	Graduate	No	3510	0	76	360	0	Urban	N
LP001038	Male	Yes	0	Not Graduate	No	4887	0	133	360	1	Rural	N
LP001041	Male	Yes	0	Graduate	0	2600	3500	115	0	1	Urban	Y
LP001043	Male	Yes	0	Not Graduate	No	7660	0	104	360	0	Urban	N
LP001046	Male	Yes	1	Graduate	No	5955	5625	315	360	1	Urban	Y
LP001047	Male	Yes	0	Not Graduate	No	2600	1911	116	360	0	Semiurban	N
LP001050	0	Yes	2	Not Graduate	No	3365	1917	112	360	0	Rural	N
LP001052	Male	Yes	1	Graduate	0	3717	2925	151	360	0	Semiurban	N
LP001066	Male	Yes	0	Graduate	Yes	9560	0	191	360	1	Semiurban	Y
LP001068	Male	Yes	0	Graduate	No	2799	2253	122	360	1	Semiurban	Y
LP001073	Male	Yes	2	Not Graduate	No	4226	1040	110	360	1	Urban	Y
LP001086	Male	No	0	Not Graduate	No	1442	0	35	360	1	Urban	N

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

loan 1 | Apply | Revert

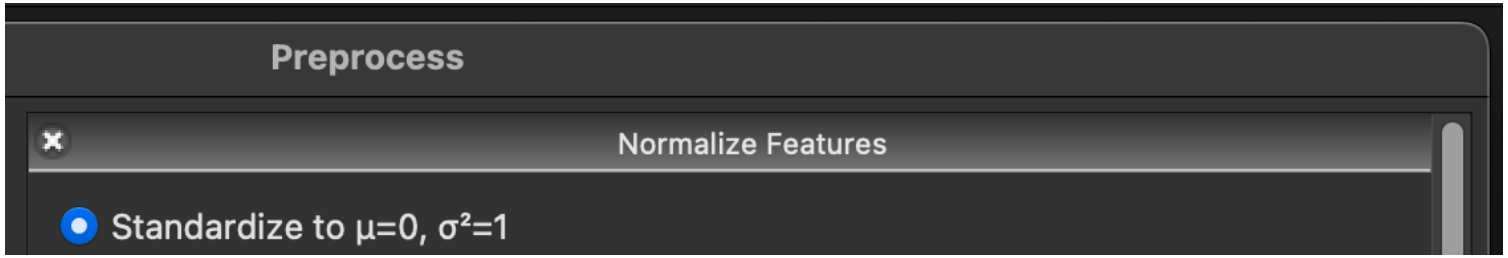
Action Output

	Time	Action	Response	Duration / Fetch Time
1	23:40:27	SELECT * FROM DS2023.loan LIMIT 0, 1000	614 row(s) returned	0.017 sec / 0.0022 sec

data_split_train_test

train set과 test set의 구분: 모델을 fitting 하는 train set과, 적용된 모델을 test set을 이용하여 Predict 하기.

```python
def data_split_train_test(self):
    self.X_train_input, self.X_test_input, self.y_train, self.y_test = train_test_split(self.X, self.y, test_size=0.3, random_state=42)

def preprocess(self):
    ss=StandardScaler()
    ss.fit(self.X_train_input)
    ss.fit(self.X_test_input)
    self.X_train=ss.transform(self.X_train_input)
    self.X_test=ss.transform(self.X_test_input)
```



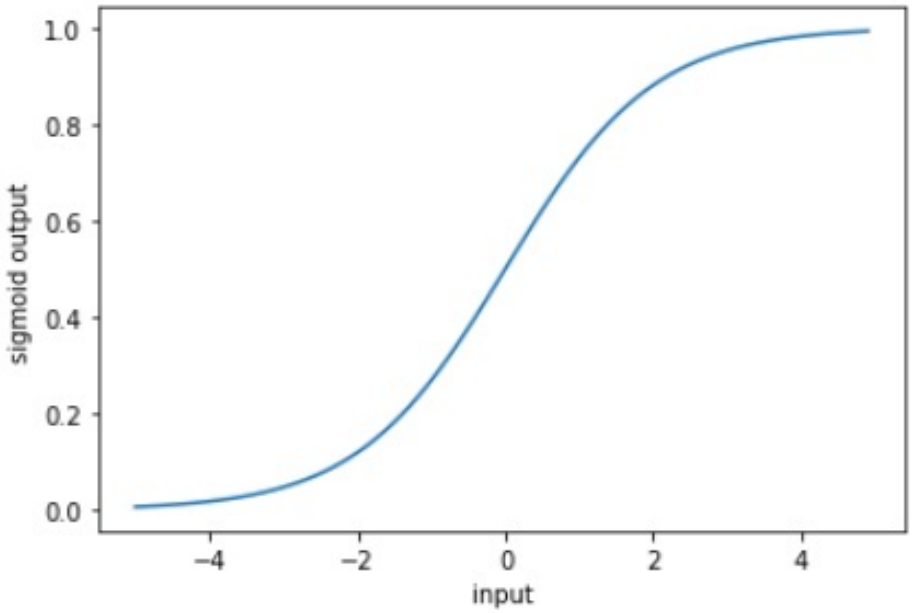
Logistic Regression

회귀 식으로 y 값 구함 -> 시그모이드 함수 통과 -> 0.5 넘으면 해당 클래스 정답으로 예측

```
def train_and_test_LogisticRegression(self):
    lr=LogisticRegression()
    lr_model=lr.fit(self.X_train, self.y_train)
    print(f"상위 5개의 predict={lr.predict(self.X_train[:5])}\n")
    print(f"상위 5개의 predict prob.={lr.predict_proba(self.X_train[:5])}\n")
    decisions_train=lr.decision_function(self.X_train[:5])
    print(f"상위 5개의 decision={decisions_train}\n")
    sigmoid_train=expit(decisions_train)
    print(f"상위 5개의 시그모이드 함수 통과={sigmoid_train}\n")

    lr_model.predict(self.X_test)
    decisions=lr.decision_function(self.X_test)
    self.y_predict=expit(decisions)
    pprint(f"test result={self.y_predict}\n")
```

```
상위 5개의 predict=[1 1 1 1 1]

상위 5개의 predict prob.=[[0.27240944 0.72759056]
 [0.28139255 0.71860745]
 [0.284008   0.715992  ]
 [0.29391544 0.70608456]
 [0.30440575 0.69559425]]
```

$$y = w_1x_1 + w_2x_2 + \ldots w_nx_n + b$$



Output/Equation

```
상위 5개의 decision=[0.98243222 0.93756457 0.92466659 0.87644289 0.826405  ]

상위 5개의 시그모이드 함수 통과=[0.72759056 0.71860745 0.715992   0.70608456 0.69559425]
```

```
('test result=[0.71460139 0.70577191 0.7143821  0.69431848 0.69478057 '
 '0.70940496\n'
 ' 0.70529443 0.69241983 0.72462014 0.71341397 0.68580268 0.71756252\n'
 ' 0.70447678 0.6698898  0.69760392 0.70090583 0.6925198  0.70966175\n'
 ' 0.71331498 0.687791   0.67588526 0.73284675 0.71200973 0.71547985\n'
 ' 0.63964773 0.70587228 0.7155487  0.70853124 0.70374183 0.67994499\n'
 ' 0.69277098 0.70515889 0.7043833  0.70438761 0.71005713 0.72362632\n'
 ' 0.72044587 0.69947782 0.70274166 0.72790879 0.70340303 0.71357122\n'
 ' 0.6895704  0.70619872 0.69887684 0.70563713 0.72645416 0.70290183\n'
 ' 0.71269696 0.6950359  0.72884937 0.70689107 0.7052932  0.69931127\n'
 ' 0.72213829 0.70910982 0.72112137 0.7231207  0.71649109 0.71822854\n'
 ' 0.69625545 0.71419355 0.65660127 0.6964624  0.70578336 0.70640841\n'
 ' 0.7051598  0.63902104 0.691325   0.70742914 0.69771946 0.69613188\n'
 ' 0.69303485 0.71853734 0.72453036 0.67510234 0.74041178 0.66403127\n'
 ' 0.693151   0.69477692 0.69985638 0.70409322 0.70301047 0.6779201\n'
 ' 0.71635217 0.71516344 0.72521945 0.70528156 0.70369587 0.59212864\n'
 ' 0.75100725 0.68935799 0.73608696 0.71055966 0.71663529 0.70660939\n'
 ' 0.70242982 0.70075789 0.69722609 0.70829175 0.70211063 0.70778442\n'
 ' 0.70191348 0.73091065 0.75377648 0.71243462 0.70985041 0.69369029\n'
 ' 0.70226238 0.69065716 0.70258243 0.70912659 0.70534745 0.71245271\n'
 ' 0.69757434 0.70133827 0.7102342  0.7136054  0.68221282 0.69151166\n'
 ' 0.7095513  0.70706928 0.55434359 0.73689708 0.71153748 0.72078333\n'
 ' 0.71419654 0.69702448 0.76593436 0.70973763 0.70441222 0.70218432\n'
 ' 0.70106223 0.71074466 0.71478764 0.69894764 0.72296883 0.70652254\n'
 ' 0.72955207 0.73355025 0.70397563 0.71328377 0.69015478 0.77867137\n'
```

Logistic Regression

$$P(y_i = j|x_i) = \frac{e^{b_{0j}+b_{1j}x_{i1}+\cdots+b_{pj}x_{ip}}}{1+\sum_{k=1}^{K-1} e^{b_{0k}+b_{1k}x_{i1}+\cdots+b_{pk}x_{ip}}}, i = 1,\ldots,n, j = 1,\ldots,K-1,$$

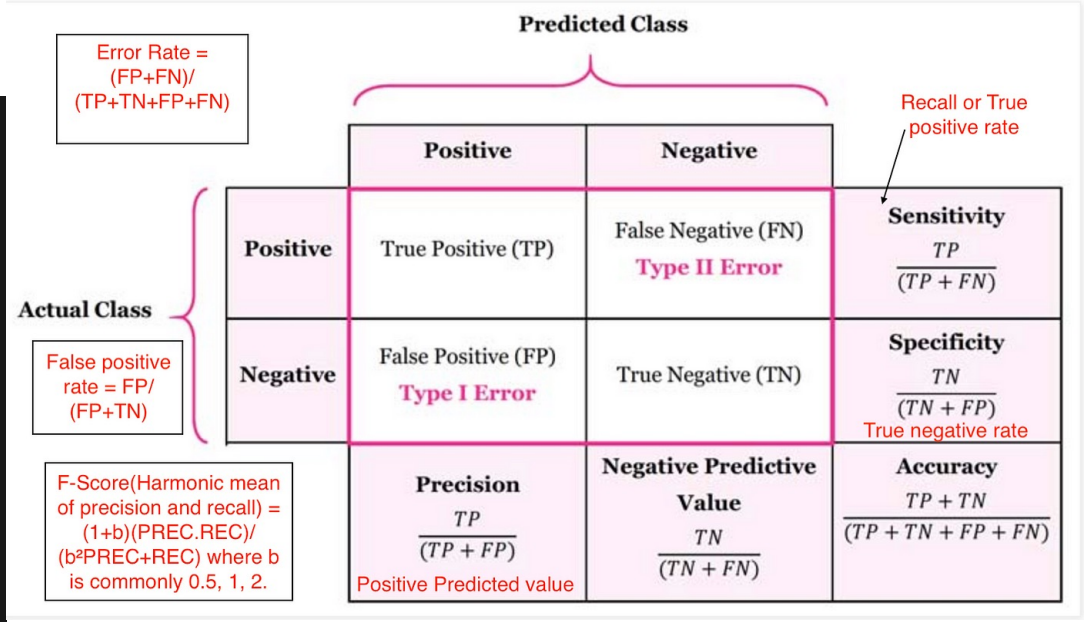
$$P(y_i = K|x_i) = \frac{1}{1+\sum_{k=1}^{K-1} e^{b_{0k}+b_{1k}x_{i1}+\cdots+b_{pk}x_{ip}}}$$

$$\begin{aligned} b_{1j} &= b_{0j} + b_{1j}(x+1) - [b_{0j} + b_{1j}x] \\ &= \log \frac{P(y=j|x+1)}{P(y=K|x+1)} - \log \frac{P(y=j|x)}{P(y=K|x)} \\ &= \log \frac{\frac{P(y=j|x+1)}{P(y=K|x+1)}}{\frac{P(y=j|x)}{P(y=K|x)}} \end{aligned}$$

성능 평가

Confusion Matrix를 활용하여 예측의 성능을 확인하기

```python
def classification_LR_performance_eval_binary(self, y_test, y_predict):
    tp, tn, fp, fn = 0,0,0,0

    for y, yp in zip(y_test, y_predict):
        if y == 1 and yp >= 0.5:
            tp += 1
        elif y == 1 and yp <= 0.5:
            fn += 1
        elif y == 0 and yp >= 0.5:
            fp += 1
        else:
            tn += 1

    accuracy = (tp+tn)/(tp+tn+fp+fn)
    precision = (tp)/(tp+fp)
    recall = (tp)/(tp+fn)
    f1_score = 2*precision*recall / (precision+recall)

    print("\n-----LR__eval_Iteration-----")
    print(f"accuracy={accuracy}")
    print(f"preceision={precision}")
    print(f"recall={recall}")
    print(f"f1_score={f1_score}\n")

    return accuracy, precision, recall, f1_score
```

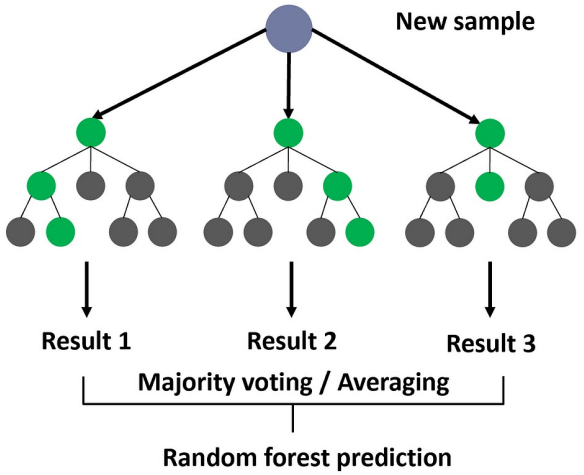


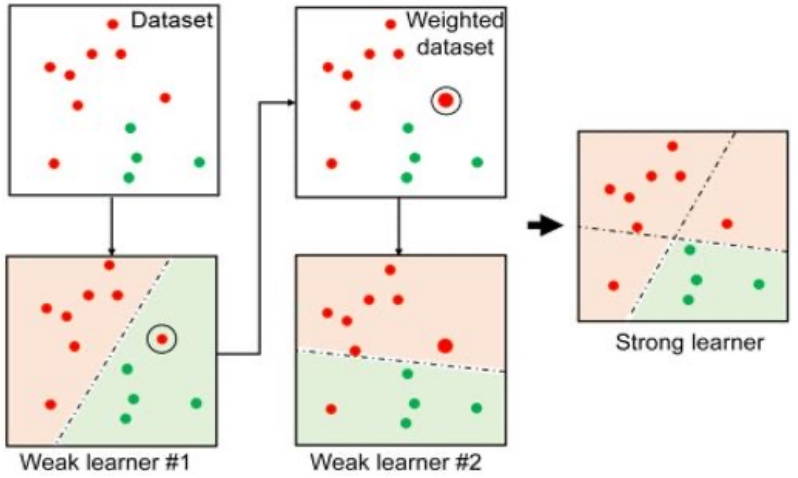

```
-----LR__eval_Iteration-----
accuracy=0.6486486486486487
preceision=0.6486486486486487
recall=1.0
f1_score=0.7868852459016393
```

The Others

나머지 두개의 앙상블 모델, fit하고 predict 하면 결과값이 바로 나옴 (간단한 과정)

```
def train_and_test_RandomForest(self):
    rf = RandomForestClassifier(random_state= 42)
    rf_model = rf.fit(self.X_train, self.y_train)
    self.y_predict = rf_model.predict(self.X_test)

def train_and_test_AdaBoost(self):
    ab = AdaBoostClassifier(random_state=42)
    ab_model = ab.fit(self.X_train, self.y_train)
    self.y_predict = ab_model.predict(self.X_test)
```





1.11. Ensembles: Gradient boosting, random forests, bagging, voting, stacking

- 1.11.1. Gradient-boosted trees
- 1.11.2. Random forests and other randomized tree ensembles
- 1.11.3. Bagging meta-estimator
- 1.11.4. Voting Classifier
- 1.11.5. Voting Regressor
- 1.11.6. Stacked generalization
- 1.11.7. AdaBoost

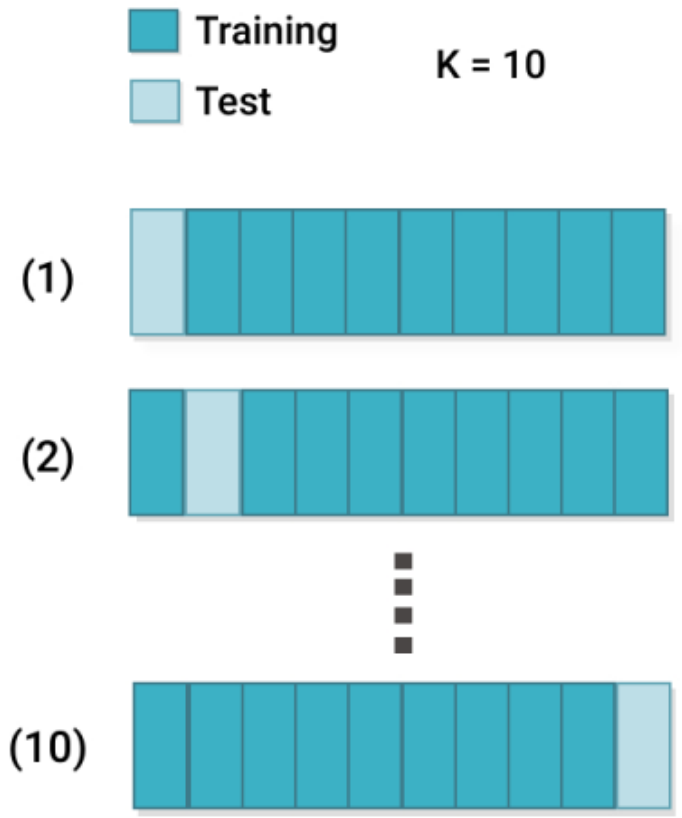
Accuracy, Precision, Recall & F1 Score

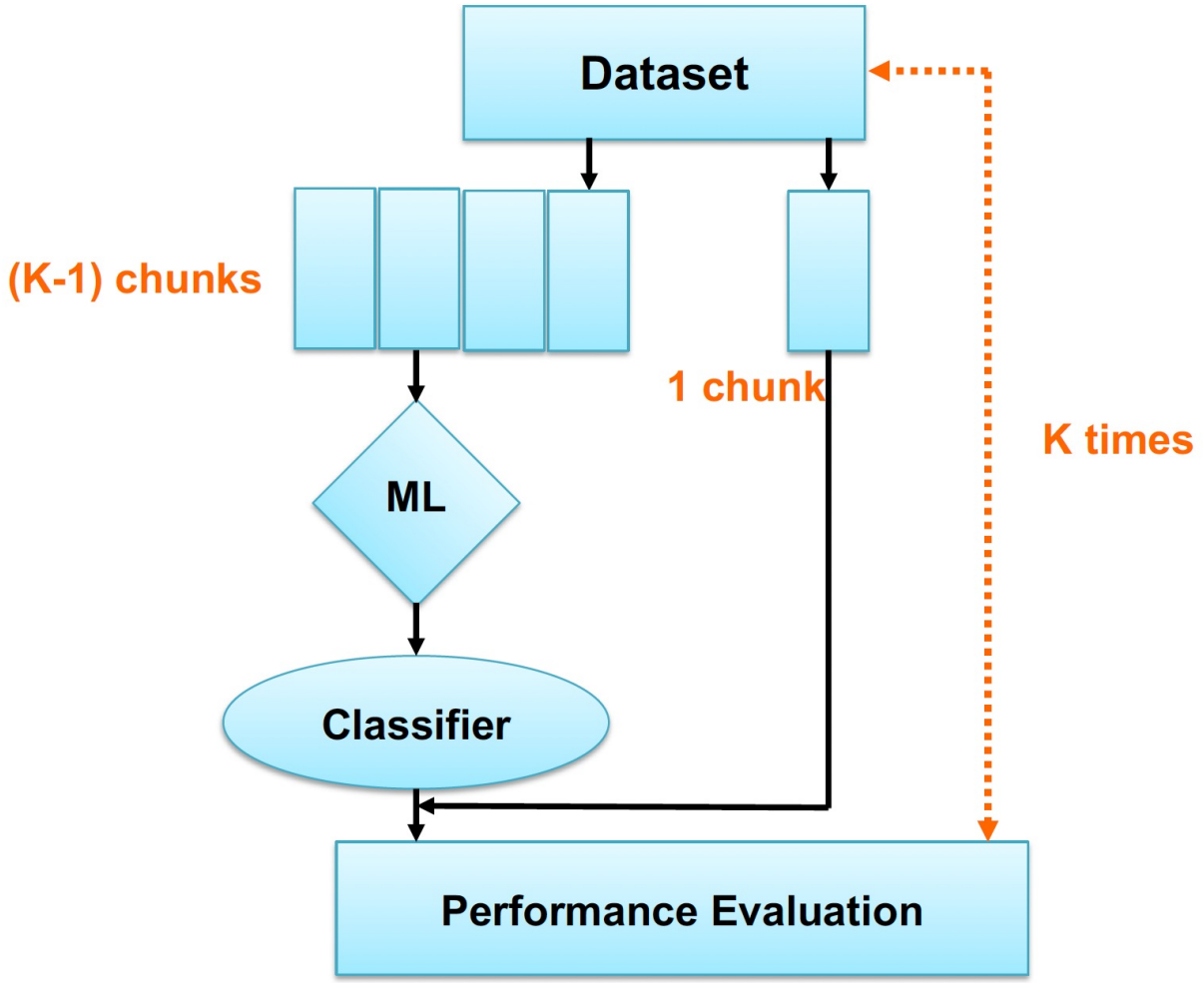
Logistic Regression 보다는 조금 더 직관적인 분류 후 평가 과정

```python
def classification_performance_eval_binary(self, y_test, y_predict):
    tp, tn, fp, fn = 0,0,0,0

    for y, yp in zip(y_test, y_predict):
        if y == 1 and yp == 1:
            tp += 1
        elif y == 1 and yp == 0:
            fn += 1
        elif y == 0 and yp == 1:
            fp += 1
        else:
            tn += 1

    accuracy = (tp+tn)/(tp+tn+fp+fn)
    precision = (tp)/(tp+fp)
    recall = (tp)/(tp+fn)
    f1_score = 2*precision*recall / (precision+recall)

    return accuracy, precision, recall, f1_score
```

K-Fold 교차검증

Cross Validation: K겹의 층으로 train/test data를 나누고, 각 factor들 평가, 보통 iteration들의 평균을 확인함





K-Fold

5겹의 fold 수, 모델 fit/predict 과정은 필요 상 다시 재배치

```python
def binary_LogisticRegression_KFold_performance(self):

    kf = KFold(n_splits=5, random_state=42, shuffle=True)

    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores = []

    for train_index, test_index in kf.split(self.X):
        X_train, X_test = self.X[train_index], self.X[test_index]
        y_train, y_test = self.y[train_index], self.y[test_index]

        lr=LogisticRegression()
        lr_model=lr.fit(X_train, y_train)
        lr_model.predict(X_test)
        decisions=lr.decision_function(X_test)
        y_predict=expit(decisions)

        accuracy, precision, recall, f1_score=self.classification_LR_performance_eval_binary(y_test, y_predict)
        accuracy_scores.append(accuracy)
        precision_scores.append(precision)
        recall_scores.append(recall)
        f1_scores.append(f1_score)
```

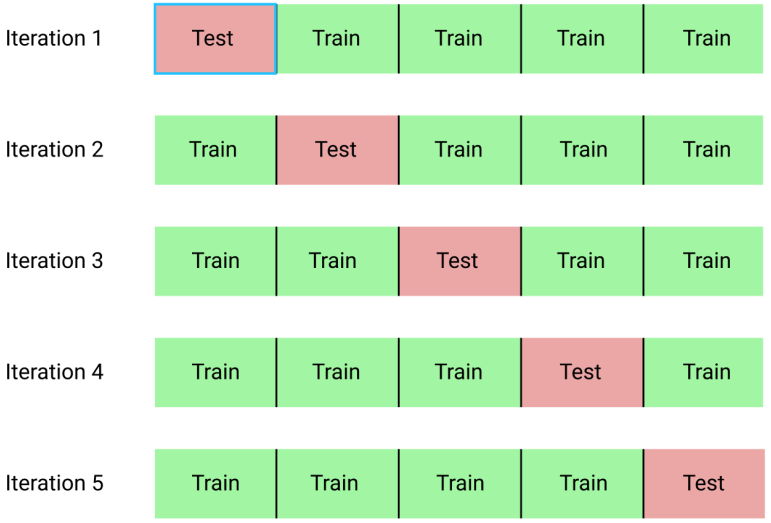
```python
average_accuracy = np.mean(accuracy_scores)
average_precision = np.mean(precision_scores)
average_recall = np.mean(recall_scores)
average_f1_score = np.mean(f1_scores)

print("\nLR\nAverage Accuracy:", average_accuracy)
print("Average Precision:", average_precision)
print("Average Recall:", average_recall)
print("Average F1 Score:", average_f1_score)
```

```
LR
Average Accuracy: 0.6873250699720113
Average Precision: 0.6873250699720113
Average Recall: 1.0
Average F1 Score: 0.8143614425139691
```

K-Fold

Iteration 별 각각 다른 factor들의 값을 확인할 수 있음

```
    print("\n-----LR__eval_Iteration-----")
    print(f"accuracy={accuracy}")
    print(f"preceision={precision}")
    print(f"recall={recall}")
    print(f"f1_score={f1_score}\n")

    return accuracy, precision, recall, f1_score
```




```
KFold

-----LR__eval_Iteration-----
accuracy=0.6504065040650406
preceision=0.6504065040650406
recall=1.0
f1_score=0.7881773399014779


-----LR__eval_Iteration-----
accuracy=0.6585365853658537
preceision=0.6585365853658537
recall=1.0
f1_score=0.7941176470588235


-----LR__eval_Iteration-----
accuracy=0.7235772357723578
preceision=0.7235772357723578
recall=1.0
f1_score=0.839622641509434


-----LR__eval_Iteration-----
accuracy=0.6991869918699187
preceision=0.6991869918699187
recall=1.0
```

K-Fold : RandomForest

```python
def binary_RandomForest_KFold_performance(self):

    kf = KFold(n_splits=5, random_state=42, shuffle=True)

    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores = []

    for train_index, test_index in kf.split(self.X):
        X_train, X_test = self.X[train_index], self.X[test_index]
        y_train, y_test = self.y[train_index], self.y[test_index]

        rf = RandomForestClassifier(random_state= 42)
        rf_model = rf.fit(X_train, y_train)
        y_predict = rf_model.predict(X_test)

        accuracy, precision, recall, f1_score=self.classification_performance_eval_binary(y_test, y_predict)
        accuracy_scores.append(accuracy)
        precision_scores.append(precision)
        recall_scores.append(recall)
        f1_scores.append(f1_score)

    average_accuracy = np.mean(accuracy_scores)
    average_precision = np.mean(precision_scores)
    average_recall = np.mean(recall_scores)
    average_f1_score = np.mean(f1_scores)

    print("\nRF\nAverage Accuracy:", average_accuracy)
    print("Average Precision:", average_precision)
    print("Average Recall:", average_recall)
    print("Average F1 Score:", average_f1_score)
```

K-Fold : AdaBoost

```python
def binary_AdaBoost_KFold_performance(self):

    kf = KFold(n_splits=5, random_state=42, shuffle=True)

    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f1_scores = []

    for train_index, test_index in kf.split(self.X):
        X_train, X_test = self.X[train_index], self.X[test_index]
        y_train, y_test = self.y[train_index], self.y[test_index]

        ab = AdaBoostClassifier(random_state=42)
        ab_model = ab.fit(X_train, y_train)
        y_predict = ab_model.predict(X_test)

        accuracy, precision, recall, f1_score=self.classification_performance_eval_binary(y_test, y_predict)
        accuracy_scores.append(accuracy)
        precision_scores.append(precision)
        recall_scores.append(recall)
        f1_scores.append(f1_score)

    average_accuracy = np.mean(accuracy_scores)
    average_precision = np.mean(precision_scores)
    average_recall = np.mean(recall_scores)
    average_f1_score = np.mean(f1_scores)

    print("\nAB\nAverage Accuracy:", average_accuracy)
    print("Average Precision:", average_precision)
    print("Average Recall:", average_recall)
    print("Average F1 Score:", average_f1_score)
```

Binary Class(loan.data)

최종 평가 (train_test_split)

```
-----LR__eval_Iteration-----
accuracy=0.6486486486486487
preceision=0.6486486486486487
recall=1.0
f1_score=0.7868852459016393


-----train_test_split-----
accuracy=0.6108108108108108
preceision=0.6463414634146342
recall=0.8833333333333333
f1_score=0.7464788732394365


-----train_test_split-----
accuracy=0.6162162162162163
preceision=0.6416184971098265
recall=0.925
f1_score=0.7576791808873719
```

factor	Top score method
Accuracy	Logistic Regression
Precision	Logistic Regression
Recall	Logistic Regression
F1 Score	Logistic Regression

Binary Class(loan.data)

최종 평가 (K-Fold)

```
LR
Average Accuracy: 0.6873250699720113
Average Precision: 0.6873250699720113
Average Recall: 1.0
Average F1 Score: 0.8143614425139691

RF
Average Accuracy: 0.6140210582433694
Average Precision: 0.6898460686835384
Average Recall: 0.7994691954823913
Average F1 Score: 0.7396914823611154

AB
Average Accuracy: 0.6514860722377716
Average Precision: 0.6814429559086891
Average Recall: 0.9296010155264576
Average F1 Score: 0.7855043079525326
```

factor	Top score method
Accuracy	Logistic Regression
Precision	RandomForest
Recall	Logistic Regression
F1 Score	Logistic Regression

Binary Class(loan.data)

결과 출력용 코드

```python
def binary_LogisticRegression_train_test_performance():
    clf = class_loan_classification(import_data_flag=True)
    clf.load_data_for_binary_classification(Loan_Status='Y')
    clf.data_split_train_test()
    clf.preprocess()
    clf.train_and_test_LogisticRegression()
    clf.classification_LR_performance_eval_binary(clf.y_test, clf.y_predict)

def binary_RandomForest_train_test_performance():
    clf = class_loan_classification(import_data_flag=True)
    clf.load_data_for_binary_classification(Loan_Status='Y')
    clf.data_split_train_test()
    clf.preprocess()
    clf.train_and_test_RandomForest()
    clf.classification_performance_eval_binary(clf.y_test, clf.y_predict)

def binary_AdaBoost_train_test_performance():
    clf = class_loan_classification(import_data_flag=True)
    clf.load_data_for_binary_classification(Loan_Status='Y')
    clf.data_split_train_test()
    clf.preprocess()
    clf.train_and_test_AdaBoost()
    clf.classification_performance_eval_binary(clf.y_test, clf.y_predict)
```

```python
def binary_LogisticRegression_KFold_performance():
    clf = class_loan_classification(import_data_flag=False)
    clf.load_data_for_binary_classification(Loan_Status='Y')
    print("\nKFold")
    clf.binary_LogisticRegression_KFold_performance()

def binary_RandomForest_KFold_performance():
    clf = class_loan_classification(import_data_flag=False)
    clf.load_data_for_binary_classification(Loan_Status='Y')
    clf.binary_RandomForest_KFold_performance()

def binary_AdaBoost_KFold_performance():
    clf = class_loan_classification(import_data_flag=False)
    clf.load_data_for_binary_classification(Loan_Status='Y')
    clf.binary_AdaBoost_KFold_performance()
```

```python
if __name__ == "__main__":
    binary_LogisticRegression_train_test_performance()
    binary_RandomForest_train_test_performance()
    binary_AdaBoost_train_test_performance()

    binary_LogisticRegression_KFold_performance()
    binary_RandomForest_KFold_performance()
    binary_AdaBoost_KFold_performance()
```

Conclusion

Logistic Regression은 독립변수가 어떤 값을 가지든 상관없이 종속변수는 확률값을 가진다.
Logistic Regression은 로짓(오즈)변환을 통해 Binary 분류를 하였다.

Train set/Test set 구분과 K-Fold 교차 분석 두 경우 모두, 성능은 Logistic Regression이 압도적인 값을 산출하였다.

'ApplicantIncome'과 'LoanAmount' 두 가지의 column 만을 이용하여 분류를 진행하였기에,
전체 데이터를 올바르게 분류하고, 극단적으로 치우치지 않기엔 한계에 있던 것으로 추정된다.

결측값의 수가 꽤 많았는데, 이 또한 모두 0으로 대치하였던 것 또한 설명력이 떨어지는 요인으로 추정된다.
추가적으로, 두 column의 correlation 또한 상당히 높은 수치임을 사후적으로 확인하였다.

-correlation table

O	P	Q	R	S
	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
ApplicantIncome	1			
CoapplicantIncome	-0.116604581	1		
LoanAmount	**0.570909039**	0.188619404	1	
Loan_Amount_Term	-0.045306087	-0.059878009	0.039447251	1

Multi Class

Multi Class(wine.data)

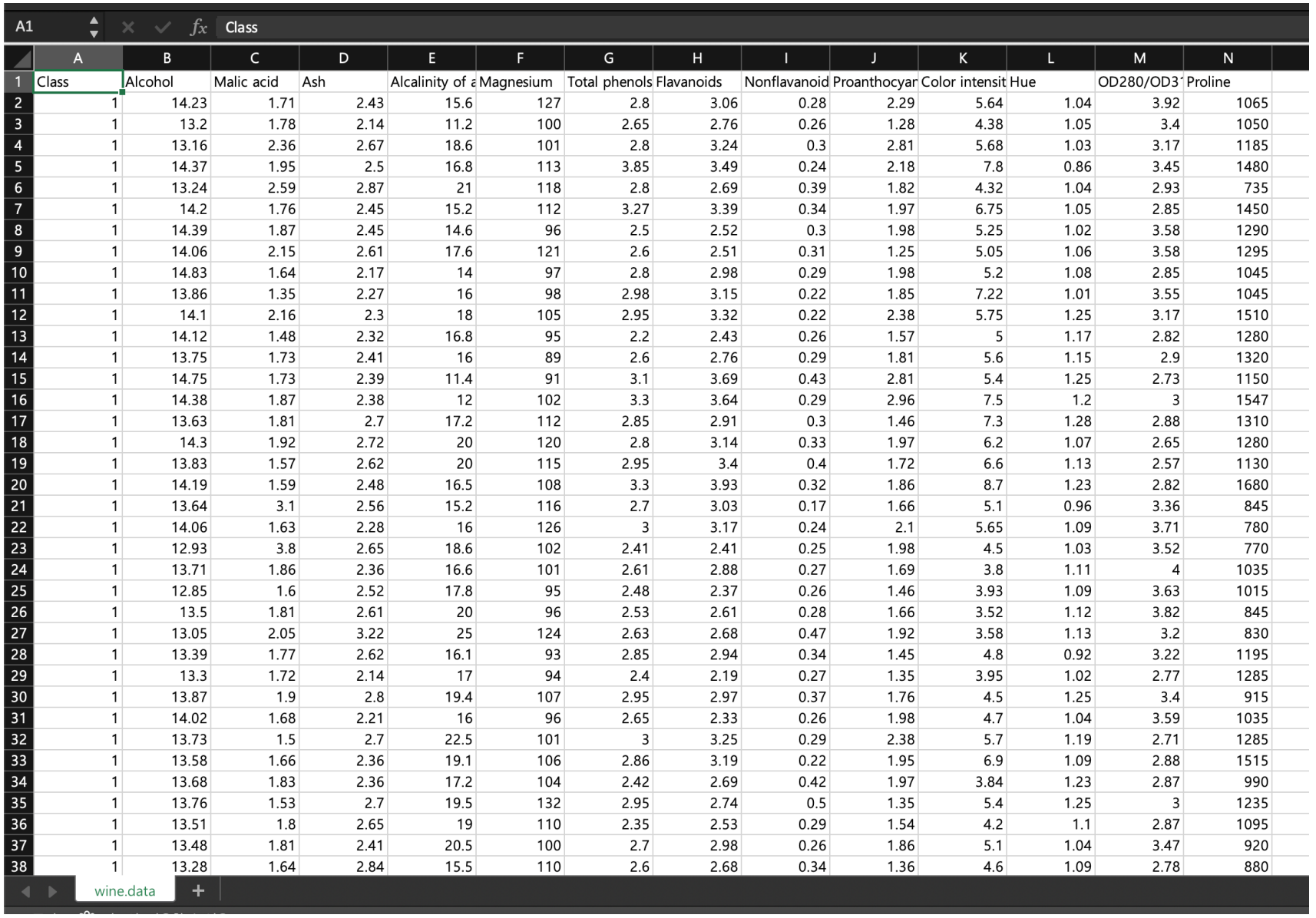
csv 데이터 설명: Wine의 1/2/3 Class 분류, 본 과제에서는 전체 항목을 모델 피팅에 사용하였음.

Dataset2: wine classification

- Class: 와인의 종류를 나타내는 목표 변수. 분류 작업에서 이 값을 예측함. 총 3개의 고유한 클래스

- Alcohol: 와인의 알코올 함량.
- Malic acid: 와인에 포함된 말산의 양.
- Ash: 와인의 재분 함량.
- Alcalinity of ash: 와인의 재분의 알칼리성.
- Magnesium: 와인에 포함된 마그네슘의 양.
- Total phenols: 와인에 포함된 총 페놀의 양.
- Flavanoids: 와인에 포함된 플라보노이드의 양. 플라보노이드는 페놀의 하위 분류임.
- Nonflavanoid phenols: 와인에 포함된 비플라보노이드 페놀의 양.
- Proanthocyanins: 와인에 포함된 프로안토시아닌의 양.
- Color intensity: 와인의 색상 강도.
- Hue: 와인의 색상.
- OD280/OD315 of diluted wines: 와인이 희석된 상태에서의 OD280/OD315 비율. 이 비율은 와인의 품질을 나타내는 지표로 사용됨.
- Proline: 와인에 포함된 프롤린의 양.

Excel

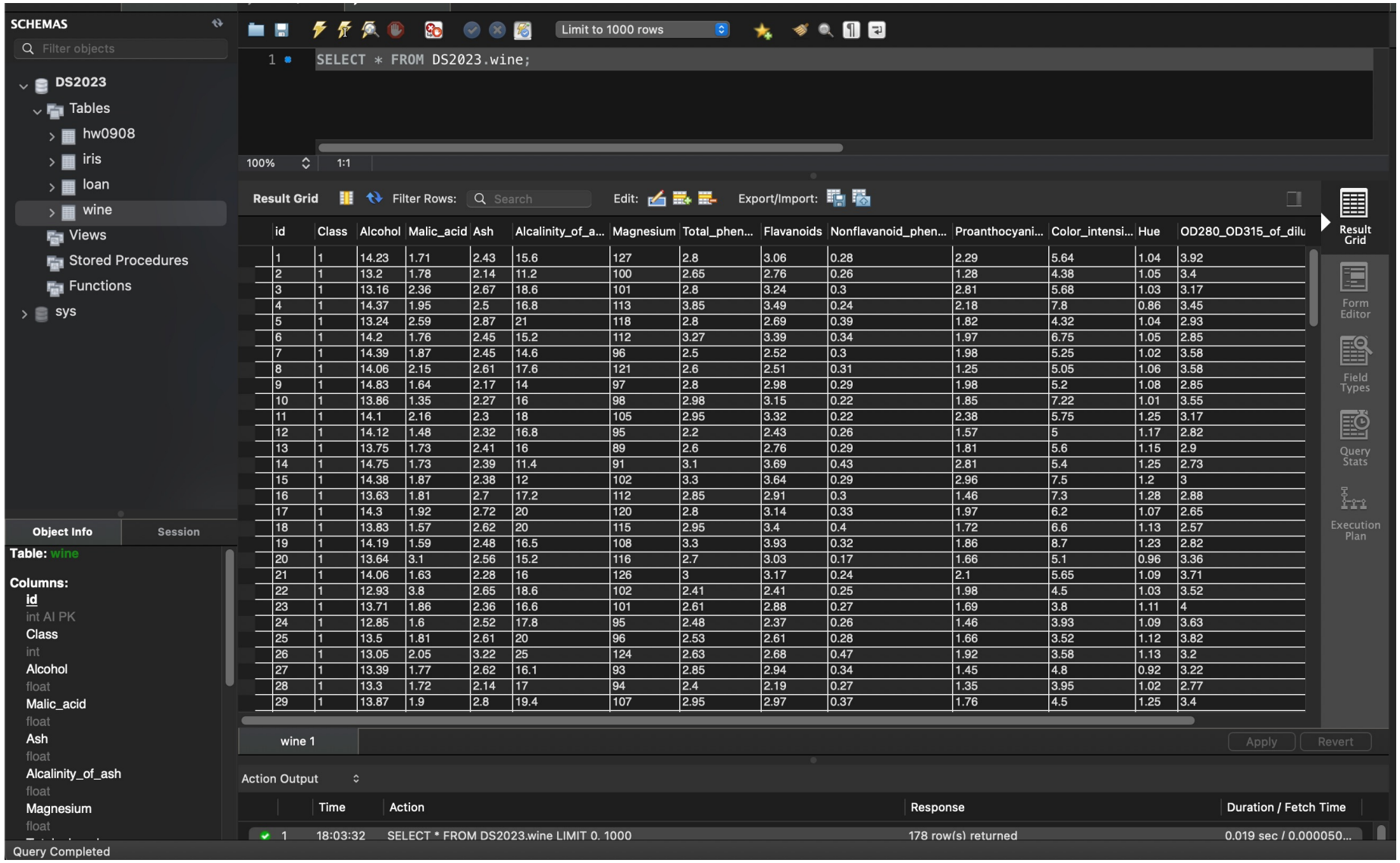


A1 | Class

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Class	Alcohol	Malic acid	Ash	Alcalinity of a	Magnesium	Total phenols	Flavanoids	Nonflavanoid	Proanthocyar	Color intensit	Hue	OD280/OD3	Proline
2	1	14.23	1.71	2.43	15.6	127	2.8	3.06	0.28	2.29	5.64	1.04	3.92	1065
3	1	13.2	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.4	1050
4	1	13.16	2.36	2.67	18.6	101	2.8	3.24	0.3	2.81	5.68	1.03	3.17	1185
5	1	14.37	1.95	2.5	16.8	113	3.85	3.49	0.24	2.18	7.8	0.86	3.45	1480
6	1	13.24	2.59	2.87	21	118	2.8	2.69	0.39	1.82	4.32	1.04	2.93	735
7	1	14.2	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
8	1	14.39	1.87	2.45	14.6	96	2.5	2.52	0.3	1.98	5.25	1.02	3.58	1290
9	1	14.06	2.15	2.61	17.6	121	2.6	2.51	0.31	1.25	5.05	1.06	3.58	1295
10	1	14.83	1.64	2.17	14	97	2.8	2.98	0.29	1.98	5.2	1.08	2.85	1045
11	1	13.86	1.35	2.27	16	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045
12	1	14.1	2.16	2.3	18	105	2.95	3.32	0.22	2.38	5.75	1.25	3.17	1510
13	1	14.12	1.48	2.32	16.8	95	2.2	2.43	0.26	1.57	5	1.17	2.82	1280
14	1	13.75	1.73	2.41	16	89	2.6	2.76	0.29	1.81	5.6	1.15	2.9	1320
15	1	14.75	1.73	2.39	11.4	91	3.1	3.69	0.43	2.81	5.4	1.25	2.73	1150
16	1	14.38	1.87	2.38	12	102	3.3	3.64	0.29	2.96	7.5	1.2	3	1547
17	1	13.63	1.81	2.7	17.2	112	2.85	2.91	0.3	1.46	7.3	1.28	2.88	1310
18	1	14.3	1.92	2.72	20	120	2.8	3.14	0.33	1.97	6.2	1.07	2.65	1280
19	1	13.83	1.57	2.62	20	115	2.95	3.4	0.4	1.72	6.6	1.13	2.57	1130
20	1	14.19	1.59	2.48	16.5	108	3.3	3.93	0.32	1.86	8.7	1.23	2.82	1680
21	1	13.64	3.1	2.56	15.2	116	2.7	3.03	0.17	1.66	5.1	0.96	3.36	845
22	1	14.06	1.63	2.28	16	126	3	3.17	0.24	2.1	5.65	1.09	3.71	780
23	1	12.93	3.8	2.65	18.6	102	2.41	2.41	0.25	1.98	4.5	1.03	3.52	770
24	1	13.71	1.86	2.36	16.6	101	2.61	2.88	0.27	1.69	3.8	1.11	4	1035
25	1	12.85	1.6	2.52	17.8	95	2.48	2.37	0.26	1.46	3.93	1.09	3.63	1015
26	1	13.5	1.81	2.61	20	96	2.53	2.61	0.28	1.66	3.52	1.12	3.82	845
27	1	13.05	2.05	3.22	25	124	2.63	2.68	0.47	1.92	3.58	1.13	3.2	830
28	1	13.39	1.77	2.62	16.1	93	2.85	2.94	0.34	1.45	4.8	0.92	3.22	1195
29	1	13.3	1.72	2.14	17	94	2.4	2.19	0.27	1.35	3.95	1.02	2.77	1285
30	1	13.87	1.9	2.8	19.4	107	2.95	2.97	0.37	1.76	4.5	1.25	3.4	915
31	1	14.02	1.68	2.21	16	96	2.65	2.33	0.26	1.98	4.7	1.04	3.59	1035
32	1	13.73	1.5	2.7	22.5	101	3	3.25	0.29	2.38	5.7	1.19	2.71	1285
33	1	13.58	1.66	2.36	19.1	106	2.86	3.19	0.22	1.95	6.9	1.09	2.88	1515
34	1	13.68	1.83	2.36	17.2	104	2.42	2.69	0.42	1.97	3.84	1.23	2.87	990
35	1	13.76	1.53	2.7	19.5	132	2.95	2.74	0.5	1.35	5.4	1.25	3	1235
36	1	13.51	1.8	2.65	19	110	2.35	2.53	0.29	1.54	4.2	1.1	2.87	1095
37	1	13.48	1.81	2.41	20.5	100	2.7	2.98	0.26	1.86	5.1	1.04	3.47	920
38	1	13.28	1.64	2.84	15.5	110	2.6	2.68	0.34	1.36	4.6	1.09	2.78	880

wine.data

MySQL Workbench



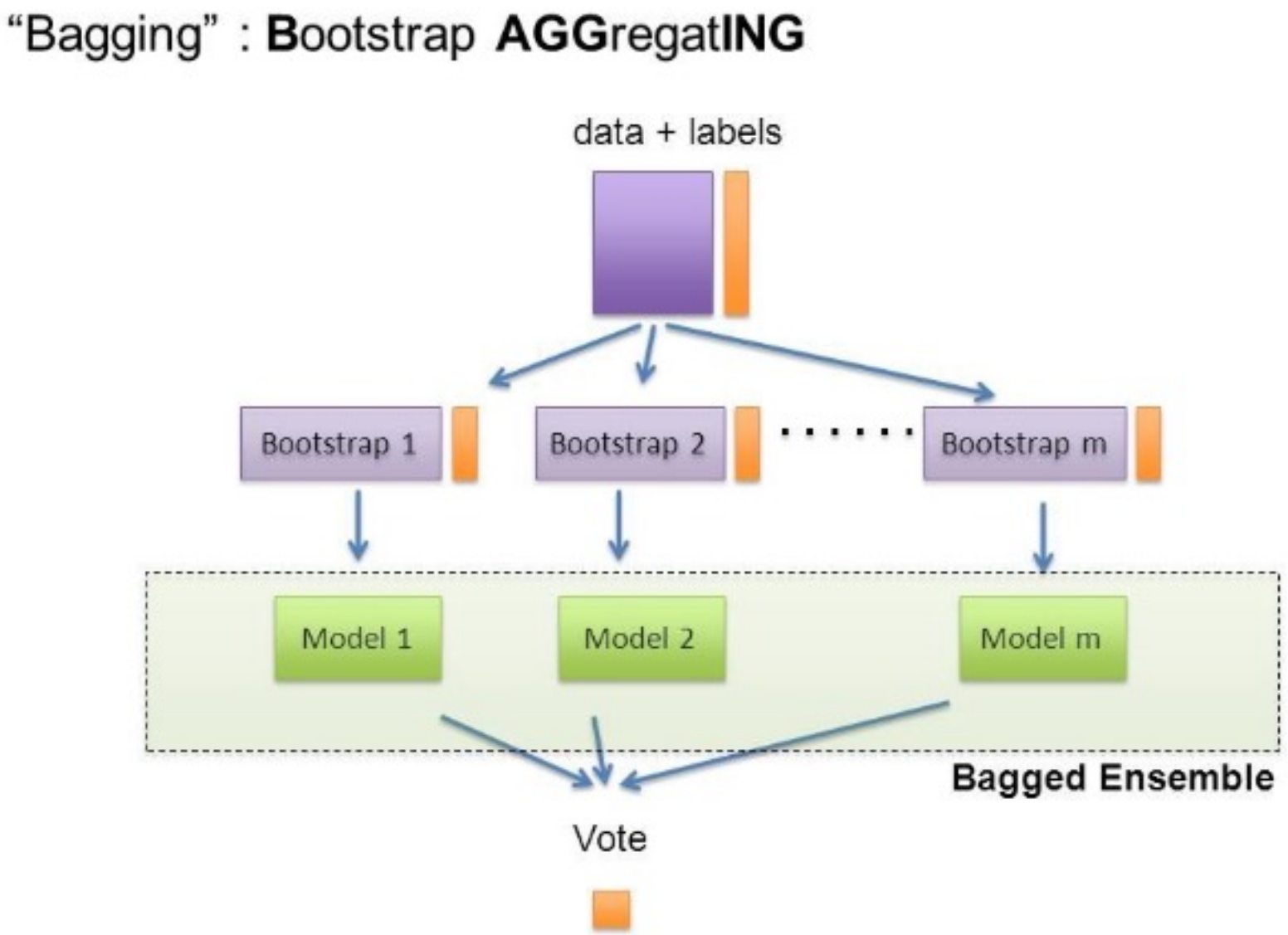
load_data_for_multiclass_classification

Primar key를 제외한 모든 column 값을 분류 예측에 사용하였음

```
def load_data_for_multiclass_classification(self):
    sql = "select * from wine;"
    self.cur.execute(sql)

    data = self.cur.fetchall()

    self.X = [ (t['Alcohol'], t['Malic_acid'],t['Ash'],t['Alcalinity_of_ash'],
                t['Magnesium'],t['Total_phenols'],t['Flavanoids'],t['Nonflavanoid_phenols'],
                t['Proanthocyanins'],t['Color_intensity'],t['Hue'],
                t['OD280_OD315_of_diluted_wines'],t['Proline']) for t in data ]
    self.X = np.array(self.X)

    self.y =  [t['Class'] for t in data]
    self.y = np.array(self.y)
```

```
def data_split_train_test(self):
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(self.X, self.y, test_size=0.3, random_state=42)
```

Bagging


"Bagging" : Bootstrap AGGregatING
data + labels
Bootstrap 1
Bootstrap 2
Bootstrap m
Model 1
Model 2
Model m
Bagged Ensemble
Vote

data_split_train_and_test

method: GradientBoosting, RandomForest, AdaBoost (앙상블 기법들)

```
def train_and_test_GB(self):
    gb = GradientBoostingClassifier(random_state= 42)
    gb_model = gb.fit(self.X_train, self.y_train)
    self.y_predict = gb_model.predict(self.X_test)
    print(f"GB : self.y_predict[:10]={self.y_predict[:10]}")
    print(f"GB : self.y_test[:10]={self.y_test[:10]}")

def train_and_test_RandomForest(self):
    rf=RandomForestClassifier(random_state= 42)
    rf_model=rf.fit(self.X_train, self.y_train)
    self.y_predict = rf_model.predict(self.X_test)
    print(f"\nRF : self.y_predict[:10]={self.y_predict[:10]}")
    print(f"RF : self.y_test[:10]={self.y_test[:10]}")

def train_and_test_AdaBoost(self):
    ab = AdaBoostClassifier(random_state=42)
    ab_model = ab.fit(self.X_train, self.y_train)
    self.y_predict = ab_model.predict(self.X_test)
    print(f"\nAB : self.y_predict[:10]={self.y_predict[:10]}")
    print(f"AB : self.y_test[:10]={self.y_test[:10]}")
```

classfication_report, confusion_matrix

```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report, confusion_matrix
```

3.3.2.14. Multi-label confusion matrix

The **multilabel_confusion_matrix** function computes class-wise (default) or sample-wise (samplewise=True) multilabel confusion matrix to evaluate the accuracy of a classification. multilabel_confusion_matrix also treats multiclass data as if it were multilabel, as this is a transformation commonly applied to evaluate multiclass problems with binary classification metrics (such as precision, recall, etc.).

When calculating class-wise multilabel confusion matrix C, the count of true negatives for class i is $C_{i,0,0}$, false negatives is $C_{i,1,0}$, true positives is $C_{i,1,1}$ and false positives is $C_{i,0,1}$.

```
def classification_performance_eval_multiclass(self, y_test, y_predict, output_dict=False):
    target_names=['C1', 'C2', 'C3']
    labels = [1,2,3]
    self.classification_report = classification_report(y_test, y_predict, target_names=target_names, labels=labels, output_dict=output_dict)
    self.confusion_matrix = confusion_matrix(y_test, y_predict, labels=labels)
    #pprint(f"[classification_report]\n{self.classification_report}")
    pprint(f"[confusion_matrix]\n{self.confusion_matrix}")
```

Output

Confusion Matrix (for multi class)

```
GB : self.y_predict[:10]=[1 1 2 1 2 1 2 3 2 1]
GB : self.y_test[:10]=[1 1 3 1 2 1 2 3 2 3]
'[confusion_matrix]\n[[19  0  0]\n [ 2 19  0]\n [ 1  2 11]]'

RF : self.y_predict[:10]=[1 1 3 1 2 1 2 3 2 3]
RF : self.y_test[:10]=[1 1 3 1 2 1 2 3 2 3]
'[confusion_matrix]\n[[19  0  0]\n [ 0 21  0]\n [ 0  0 14]]'

AB : self.y_predict[:10]=[1 1 3 1 2 1 2 3 2 3]
AB : self.y_test[:10]=[1 1 3 1 2 1 2 3 2 3]
'[confusion_matrix]\n[[19  0  0]\n [ 1 20  0]\n [ 0  3 11]]'
```

GB

Actual \ Predict			
	19	0	0
	2	19	0
	1	2	11

RF

Actual \ Predict			
	19	0	0
	0	21	0
	0	0	14

AB

Actual \ Predict			
	19	0	0
	1	20	0
	0	3	11

classification_report

RandomForest 분석이 모두 1.0으로 극단적으로 치운친 값을 도출, 나머지 두 모델만을 비교하였음

```
('[classification_report]\n'
 '              precision    recall  f1-score   support\n'
 '\n'
 '          C1       0.86      1.00      0.93        19\n'
 '          C2       0.90      0.90      0.90        21\n'
 '          C3       1.00      0.79      0.88        14\n'
 '\n'
 '    accuracy                           0.91        54\n'
 '   macro avg       0.92      0.90      0.90        54\n'
 'weighted avg       0.91      0.91      0.91        54\n')
```

Class 1

factor	Method
Precision	AB (0.95)
Recall	-
F1 Score	AB (0.97)

```
'[classification_report]\n'
'              precision    recall  f1-score   support\n'
'\n'
'          C1       1.00      1.00      1.00        19\n'
'          C2       1.00      1.00      1.00        21\n'
'          C3       1.00      1.00      1.00        14\n'
'\n'
'    accuracy                           1.00        54\n'
'   macro avg       1.00      1.00      1.00        54\n'
'weighted avg       1.00      1.00      1.00        54\n')
```

Class 2

factor	Method
Precision	GB (0.90)
Recall	AB (0.95)
F1 Score	AB (0.91)

```
('[classification_report]\n'
 '              precision    recall  f1-score   support\n'
 '\n'
 '          C1       0.95      1.00      0.97        19\n'
 '          C2       0.87      0.95      0.91        21\n'
 '          C3       1.00      0.79      0.88        14\n'
 '\n'
 '    accuracy                           0.93        54\n'
 '   macro avg       0.94      0.91      0.92        54\n'
 'weighted avg       0.93      0.93      0.92        54\n')
```

Class 3

factor	Method
Precision	-
Recall	-
F1 Score	-

K-Fold 교차 분석

sklearn.KFold 기능을 활용, kfold_reports 라는 list를 만들어서 평가의 결과를 기입하였음.

```
def multiclass_GB_KFold_performance(self):
    print("\n\n\nGB_KFold\n")
    kfold_reports = []
    kf = KFold(n_splits=5, random_state=42, shuffle=True)
    for train_index, test_index in kf.split(self.X):
        X_train, X_test = self.X[train_index], self.X[test_index]
        y_train, y_test = self.y[train_index], self.y[test_index]

        gb = GradientBoostingClassifier(random_state= 42)
        gb_model = gb.fit(X_train, y_train)
        y_predict = gb_model.predict(X_test)

        self.classification_performance_eval_multiclass(y_test, y_predict, output_dict=True)
        kfold_reports.append(pd.DataFrame(self.classification_report).transpose())

    for s in kfold_reports:
        print('\n_', s)

    mean_report = pd.concat(kfold_reports).groupby(level=0).mean()
    print('\n\nGB result:\nmean\n', mean_report)
```

```
def multiclass_RF_KFold_performance(self):
    print("\nRF_KFold\n")
    kfold_reports = []
    kf = KFold(n_splits=5, random_state=42, shuffle=True)
    for train_index, test_index in kf.split(self.X):
        X_train, X_test = self.X[train_index], self.X[test_index]
        y_train, y_test = self.y[train_index], self.y[test_index]

        rf = RandomForestClassifier(random_state= 42)
        rf_model = rf.fit(X_train, y_train)
        y_predict = rf_model.predict(X_test)
        self.classification_performance_eval_multiclass(y_test, y_predict, output_dict=True)
        kfold_reports.append(pd.DataFrame(self.classification_report).transpose())

    for s in kfold_reports:
        print('\n_', s)

    mean_report = pd.concat(kfold_reports).groupby(level=0).mean()
    print('\nRF result:\nmean\n', mean_report)

def multiclass_AB_KFold_performance(self):
    print("\nAB_KFold\n")
    kfold_reports = []
    kf = KFold(n_splits=5, random_state=42, shuffle=True)
    for train_index, test_index in kf.split(self.X):
        X_train, X_test = self.X[train_index], self.X[test_index]
        y_train, y_test = self.y[train_index], self.y[test_index]

        ab = AdaBoostClassifier(random_state=42)
        ab_model = ab.fit(X_train, y_train)
        y_predict = ab_model.predict(X_test)
        self.classification_performance_eval_multiclass(y_test, y_predict, output_dict=True)
        kfold_reports.append(pd.DataFrame(self.classification_report).transpose())

    for s in kfold_reports:
        print('\n_', s)

    mean_report = pd.concat(kfold_reports).groupby(level=0).mean()
    print('\nAB result:\nmean\n', mean_report)
```

K-Fold 교차 분석

Results

각 모델마다 5번의 iteration이 있으며, 최종으로는 평균값을 활용

GB_KFold

```
'[confusion_matrix]\n[[14  0  0]\n [ 1 13  0]\n [ 0  1  7]]'
'[confusion_matrix]\n[[10  2  0]\n [ 0 13  0]\n [ 0  0 11]]'
'[confusion_matrix]\n[[10  0  0]\n [ 2 14  1]\n [ 0  0  9]]'
'[confusion_matrix]\n[[11  1  0]\n [ 0 13  0]\n [ 0  0 10]]'
'[confusion_matrix]\n[[11  0  0]\n [ 1 13  0]\n [ 0  1  9]]'
```

_	precision	recall	f1-score	support
C1	0.933333	1.000000	0.965517	14.000000
C2	0.928571	0.928571	0.928571	14.000000
C3	1.000000	0.875000	0.933333	8.000000
accuracy	0.944444	0.944444	0.944444	0.944444
macro avg	0.953968	0.934524	0.942474	36.000000
weighted avg	0.946296	0.944444	0.943997	36.000000

_	precision	recall	f1-score	support
C1	1.000000	0.833333	0.909091	12.000000
C2	0.866667	1.000000	0.928571	13.000000
C3	1.000000	1.000000	1.000000	11.000000
accuracy	0.944444	0.944444	0.944444	0.944444
macro avg	0.955556	0.944444	0.945887	36.000000
weighted avg	0.951852	0.944444	0.943903	36.000000

_	precision	recall	f1-score	support
C1	0.833333	1.000000	0.909091	10.000000
C2	1.000000	0.823529	0.903226	17.000000
C3	0.900000	1.000000	0.947368	9.000000
accuracy	0.916667	0.916667	0.916667	0.916667

→ 다섯번의 iteration

GB result:
mean

	precision	recall	f1-score	support
C1	0.936667	0.950000	0.939349	11.800000
C2	0.930476	0.936134	0.930381	14.200000
C3	0.980000	0.955000	0.965614	9.600000
accuracy	0.943968	0.943968	0.943968	0.943968
macro avg	0.949048	0.947045	0.945114	35.600000
weighted avg	0.949112	0.943968	0.943571	35.600000

RF result:
mean

	precision	recall	f1-score	support
C1	0.981818	1.000000	0.990476	11.800000
C2	1.000000	0.961086	0.979500	14.200000
C3	0.961818	1.000000	0.979950	9.600000
accuracy	0.983175	0.983175	0.983175	0.983175
macro avg	0.981212	0.987029	0.983309	35.600000
weighted avg	0.984755	0.983175	0.983128	35.600000

AB result:
mean

	precision	recall	f1-score	support
C1	0.966667	0.804762	0.872348	11.800000
C2	0.821766	0.972851	0.885383	14.200000
C3	0.981818	0.849747	0.900000	9.600000
accuracy	0.888571	0.888571	0.888571	0.888571
macro avg	0.923417	0.875787	0.885911	35.600000
weighted avg	0.912044	0.888571	0.887221	35.600000

Multi Class(wine.data)

최종평가

Top Accuracy : RF

Class 1

factor	Method
Precision	RF (0.98)
Recall	RF (1.0)
F1 Score	RF (0.99)

Class 2

factor	Method
Precision	RF (1.0)
Recall	AB (0.97)
F1 Score	RF (0.97)

Class 3

factor	Method
Precision	AB (0.98)
Recall	RF (1.0)
F1 Score	RF (0.98)

Multi Class(wine.data)

결과 출력용 코드

```
def multiclass_GB_train_test_performance():
    clf = class_wine_classification(import_data_flag=True)
    clf.load_data_for_multiclass_classification()
    clf.data_split_train_test()
    clf.train_and_test_GB()
    clf.classification_performance_eval_multiclass(clf.y_test, clf.y_predict)

def multiclass_RandomForest_train_test_performance():
    clf = class_wine_classification(import_data_flag=True)
    clf.load_data_for_multiclass_classification()
    clf.data_split_train_test()
    clf.train_and_test_RandomForest()
    clf.classification_performance_eval_multiclass(clf.y_test, clf.y_predict)

def multiclass_AdaBoost_train_test_performance():
    clf = class_wine_classification(import_data_flag=True)
    clf.load_data_for_multiclass_classification()
    clf.data_split_train_test()
    clf.train_and_test_AdaBoost()
    clf.classification_performance_eval_multiclass(clf.y_test, clf.y_predict)
```

```
def multiclass_GB_KFold_performance_class():
    clf = class_wine_classification(import_data_flag=True)
    clf.load_data_for_multiclass_classification()
    clf.multiclass_GB_KFold_performance()

def multiclass_RF_KFold_performance_class():
    clf = class_wine_classification(import_data_flag=True)
    clf.load_data_for_multiclass_classification()
    clf.multiclass_RF_KFold_performance()

def multiclass_AB_KFold_performance_class():
    clf = class_wine_classification(import_data_flag=True)
    clf.load_data_for_multiclass_classification()
    clf.multiclass_AB_KFold_performance()
```

```
if __name__ == "__main__":
    multiclass_GB_train_test_performance()
    multiclass_RandomForest_train_test_performance()
    multiclass_AdaBoost_train_test_performance()

    multiclass_GB_KFold_performance_class()
    multiclass_RF_KFold_performance_class()
    multiclass_AB_KFold_performance_class()
```

Conclusion

wine.data.csv 전체 column을 활용하여 분류 분석을 진행하였다.
추가적으로, 과대적합/과소적합 문제를 해결을 위해 모델 선정은,
여러개의 분류기를 활용하는 '앙상블 기법'의 모델들을 선정하였다.

Gradient Boosting, RandomForest, AdaBoost

기본적인 구조는, 여러 개의 Bootstrp 자료를 생성하고 각 자료의 예측모형을 만든 후 결합하여,
최종 예측모형을 만드는 방법이다. 각 자료에서는 동일한 크기의 표본을 랜덤 복원추출 하게 된다.

대체적으로 좋은 성능을 보여준 'RandomForest' 기법은 여러 분류 중 다수결로 최종 결과를 구하게 된다.
Decision Tree와 유사한 형태를 띄며, 각 마디에서 최적의 분할이 아닌 표본추출 과정이 한 번 더 반복되어,
추출된 표본의 대상을 최적의 분할을 실시한다.

변수 제거 없이 실행되므로 정확도 측면에서 좋은 결과를 얻었고,
이상값에 민감하지 않다는 요인이 장점으로 작용했다고 추정한다.

End