

1. train, test \rightarrow df_train, df_test
 - ~~df~~ target = 1 브이 \rightarrow pop.
 - 두 카테고리 브이 있으면 \rightarrow target.

2유형 - 컬러 구분된 Python 코드

!!!

2유형 : 무조건 다 맞춘다는 생각으로 외울 것은 외워버릴 것

6회 기출

!!!

```
import numpy as np
```

```
import pandas as pd
```

```
df_train = pd.read_csv(
```

'C:/Python/PyProject/24년프로젝트/12. 빅분기실기/빅분기/data/유형모음/기출문제6회_2유형_train.csv')

```
df_test = pd.read_csv(
```

'C:/Python/PyProject/24년프로젝트/12. 빅분기실기/빅분기/data/유형모음/기출문제6회_2유형_test.csv')

```
print(df_train)
```

```
print(df_test)
```

학습용 데이터 타겟 값 빼어내기 : 이거하면 df_train에는 더이상 'General_Health'가 없다.

```
train_target = df_train.pop('General_Health')
```

```
print(train_target)
```

ID같은 건 삭제

```
df_train.drop('ID', axis=1, inplace=True)
```

```
df_test.drop('ID', axis=1, inplace=True)
```

1. train, test \rightarrow df_train, df_test
 - ~~df~~ target = 1 브이 \rightarrow pop.
 - 두 카테고리 브이 있으면 \rightarrow target.
2. 원핫인코딩 (concat. get_dummies)
3. Min-Max Scalar.
4. 둘다 train, test. 둘다 \rightarrow train, test.
5. ~~train~~ model - selection.
 train-test-split.
 $\rightarrow X_{train}, X_{test}, y_{train}, y_{test}$
 $= \text{model}(X_{train}, y_{train})$

6. 블록 - ensemble - Random Forest.
 블록 - ensemble - Random Forest.
 (xgboost - XGBoost)

model =

~~model~~ \rightarrow model.fit(X_train, ~~y_train~~)

pred = model.predict(X_test)

7. 퍼포먼스 평가 et y-test, pred. 예상. 실제.

↓
 1. classification
 2. metrics.accuracy_score
 3. f1-score.
 4. rmse -
 or
 mae -

metrics. mat_mean

- squared.

(y-test, pred).

8. 히든 예측.

pred-test = model.predict(X-test)

9. result = pd.DataFrame({col: Na, pred: pred-test})

result = pd.to_csv(path).

구분 ##### 설명

원핫인코딩 범주형 데이터를 숫자 벡터로 변환하는 방법

왜 필요? 모델이 문자를 이해 못 하니까 숫자로 바꿔줘야 해

왜 concat? train/test에 있는 범주가 다를 수 있어서! 같이 인코딩해야 열 구조가 같아짐

pd.get_dummies() 판다스에서 자동으로 원핫인코딩해주는 함수야

원핫인코딩은 "값"을 기준으로 "컬럼"을 새로 만드는 거야

예를 들어 Checkup이라는 원래 컬럼엔 이런 값들이 있을 수 있어:

bash

복사

편집

'Within the past year'

'5 or more years ago'

'Within the past 2 years'

'Never'

'Within the past 5 years'

이걸 pd.get_dummies() 하면:

Checkup_Within the past year Checkup_5 or more years ago Checkup_Never

...

이렇게 각 값마다 하나의 새로운 열(column)이 생겨!

그래서 Checkup 1개가 여러 열로 쪼개지는 거야!

그러니까 왜 train/test에는 없던 컬럼이 생기냐면:

df_train에는 "Checkup_Never"가 없었는데

df_test엔 "Checkup_Never"가 있는 경우처럼,

```
# 합쳐서 처리하면 전체 범주가 다 드러나서 열이 더 많아지는 거야!
```

```
# 그래서 get_dummies() 하고 나면:
```

```
# plaintext
```

```
# 복사
```

```
# 편집
```

```
# 원래 컬럼 18개      더미 변수 포함해서 40~50개로 늘어남!
```

```
#     한눈에 보기
```

```
# 구분      컬럼 개수      예시
```

```
#     df_train.columns    18개 'Checkup', 'Sex', 'Age_Category' 등 원본 그대로
```

```
#     df_test.columns    18개 마찬가지
```

```
#     df_total_encoded.columns  50개 이상 Checkup_~, Sex_Male, Sex_Female,  
Age_Category_18-24 등등 분해된 버전
```

```
# ## 행은 train과 test를 합친 것수다.
```

```
# print(len(df_train))
```

```
# print(len(df_test))
```

```
# print(len(df_total))
```

$df_{total} = pd.concat([df_train, df_test])$

$df_belf = pd.get_dummies(df_total)$

```
df_total = pd.concat([df_train, df_test])
```

```
df_total = pd.get_dummies(df_total)
```

```
# print(df_train.columns)
```

```
# print(df_test.columns)
```

```
# print(df_total.columns)
```

$$\begin{aligned} \text{train} &= \text{df_total}.iloc[:\text{len(df_train)}] \\ \text{test} &= \text{df_total}.iloc[\text{len(df_train)}:] \end{aligned}$$

train와 test 영역 나눠주기 반드시 필요!!!!!!!!!!!!!!

```
train = df_total.iloc[:len(df_train)].copy()
```

```
test = df_total.iloc[len(df_train):].copy()
```

min_max 스케일링

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

#수치형만 할 수 있다.

```
num_col = train.select_dtypes(include=['number']).columns
```

Scalr = MinMaxScaler()
 num_col = train.select_dtypes(include=['number']).columns
 train = scaler.fit_transform(train)
 test = scaler.transform(test)

num_col = train.select_dtypes(include=['number']).columns
 select_dtypes(include=['number']).columns

 train[num_col] = scaler.fit_transform(train[num_col]) # train이 학습용이니까 train기준으로만 fit_transform 해서 스케일링한다.

test[num_col] = scaler.transform(test[num_col]) # 학습용으로 스케일링 된 모델 기준으로 test데이터도 스케일링 적용하는거다.

자 이제 머신러닝

```
from sklearn.model_selection import train_test_split
```

X_train, X_test, y_train, y_test = train_test_split(train, train_target, test_size=0.2) # train과 test 8:2로 한다는 소리

입력 데이터

train: 독립 변수(features)로, 모델에 입력되는 데이터.

train_target: 종속 변수(target)로, 모델이 예측하려는 값.

출력 데이터

```
# X_train: train 데이터의 80%를 학습용으로 분리.  
# X_test: train 데이터의 20%를 테스트용으로 분리.  
# y_train: train_target의 80%를 학습용으로 분리.  
# y_test: train_target의 20%를 테스트용으로 분리.
```

```
# 랜덤포레스트  
from sklearn.ensemble import RandomForestClassifier  
model = RandomForestClassifier(random_state=123)
```

random_state > 123

```
# 모델 피팅  
model.fit(X_train, y_train)  
pred = model.predict(X_test)
```

```
#평가  
from sklearn import metrics  
  
report = metrics.classification_report(y_test, pred) # ★순서 중요!! '정답을 예측과 비교'한다. 라고 이해하자.  
순서 기억!  
print(report)
```

```
f1_score = metrics.f1_score(y_test, pred, average='macro')  
print(f1_score)
```

```
# 이 모델로 실제 예측해보자.  
pred_test = model.predict(test)
```

```
result = pd.DataFrame({'pred' : pred_test})
```

```
print(result)
```

index를 제거하고 넣으려면 반드시 index=False를 해줘야한다.

```
result.to_csv('C:/Python/PyProject/24년프로젝트/12. 빅분기실기/빅분기/data/유형모음/예측결과.csv',index=False)
```

5회기출

result.to_csv(

, index=False)

```
import numpy as np
```

```
import pandas as pd
```

```
# train_set =  
pd.read_csv('C:/Python/PyProject/24년프로젝트/12. 빅분기실기/빅분기/data/유형모음/기출문제5회_2유형  
_train.csv')
```

```
# test_set =  
pd.read_csv('C:/Python/PyProject/24년프로젝트/12. 빅분기실기/빅분기/data/유형모음/기출문제5회_2유형  
_test.csv')
```

```
df_train = pd.read_csv(  
'C:/Python/PyProject/24년프로젝트/12. 빅분기실기/빅분기/data/유형모음/기출문제5회_2유형_train.csv')
```

```
df_test = pd.read_csv(  
'C:/Python/PyProject/24년프로젝트/12. 빅분기실기/빅분기/data/유형모음/기출문제5회_2유형_test.csv')
```

'''

다음에 주어진 데이터셋은 종고차와 관련된 데이터이다.

학습용 데이터를 이용하여 종고차의 판매 가격을 예측하고, 평가용 데이터에 대한 예측 결과를 csv파일로 제출하시오. (모델의 성능은 RMSE로 평가)

'''

```
#target 떼어내기
```

drop ('ID', axis=1, inplace=True)

```
target = df_train.pop('price')
```

```
# df_train = train_set.copy()
```

```
# df_test = test_set.copy()
```

```
# ID 제거
```

```
df_train.drop('ID', axis=1, inplace=True)
```

```
df_test.drop('ID', axis=1, inplace=True)
```

```
#원핫인코딩 및 갯더미
```

```
df_total = pd.concat([df_train, df_test])
```

```
df_total = pd.get_dummies(df_total)
```

```
print(df_total)
```

```
train = df_total.iloc[:len(df_train)].copy()
```

```
test = df_total.iloc[len(df_train):].copy()
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
#수치형 열
```

```
col_num = train.select_dtypes(include=['number']).columns
```

```
train[col_num] = scaler.fit_transform(train[col_num])
```

```
test[col_num] = scaler.transform(test[col_num])
```

#모델링

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test = train_test_split(train,target,test_size = 0.2)
```

Sklearn의

```
from xgboost import XGBRegressor
```

주의!!! classifier가 아니라 Regressor를 했기 때문에, 수치적인 연속적인 데이터다. 이런 경우는 분류로트가
나올 수가 없다.

회귀 검증은 mean_absolute_error나 mean_square_error를 사용해야한다.

```
model = XGBRegressor()
```

XGBRegressor (random_state = 123)

```
model.fit(X_train, y_train)
```

```
pred = model.predict(X_test)
```

```
from sklearn import metrics
```

```
report = metrics.root_mean_squared_error(y_test,pred)
```

```
print(report)
```

#최종 예측아웃풋

```
pred_result = model.predict(test)
```

```
result = pd.DataFrame({'pred' : pred_result})
```

```
print(result)
```

*Ensemble .
RandomForestRegressor.*

Xgbboost . XGBRegressor .

root_mean_squared_error .

```
result.to_csv('C:/Python/PyProject/24년프로젝트/12. 빅분기실기/빅분기/data/유형모음/5회_result.csv'  
,index=False)
```

```
# 번외로 랜덤포레스트 회귀를 한다면
```

```
from sklearn.ensemble import RandomForestRegressor
```

```
model = RandomForestRegressor(random_state=123)  
# model = RandomForestRegressor(n_estimators=100, max_depth=4)  
# model = RandomForestRegressor(random_state=123)  
model.fit(X_train,y_train)
```

```
pred = model.predict(X_test)
```

```
report = metrics.root_mean_squared_error(y_test,pred)
```

```
print(report)
```

```
pred_result = model.predict(test)
```

```
result = pd.DataFrame({'pred':pred_result})
```

```
result.to_csv('C:/Python/PyProject/24년프로젝트/12. 빅분기실기/빅분기/data/유형모음/5회_result(ra  
ndom_forest_regressor).csv',index=False)
```

7회 기출

```
import numpy as np
```

```
import pandas as pd

df_train = pd.read_csv('C:/Python/PyProject/빅분기/기출 및 필수모의/7회데이터셋/07.02.01-sales_train_dataset.csv')

df_test = pd.read_csv('C:/Python/PyProject/빅분기/기출 및 필수모의/7회데이터셋/07.02.02-sales_test_dataset_x.csv')

# 타겟 가져오기
target = df_train.pop('Sales')

# 원핫인코딩
total = pd.concat([df_train, df_test])
total = pd.get_dummies(total)

train = total.iloc[:len(df_train)].copy()
test = total.iloc[len(df_train):].copy()

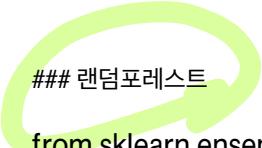
# 스케일링
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler()

# 수치형 칼럼 확인
num_col = train.select_dtypes(include = 'number').columns

train[num_col] = scale.fit_transform(train[num_col])
test[num_col] = scale.transform(test[num_col])
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(train, target, test_size = 0.2)

  
### 랜덤포레스트
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(random_state=123)

model.fit(X_train,y_train)

pred = model.predict(X_test)

from sklearn import metrics

rmse = metrics.root_mean_squared_error(y_test,pred)
print(rmse)

pred_result = model.predict(test)
print(len(pred_result))
result = pd.DataFrame({'BranchName' : df_test['BranchName'], 'Sales' : pred_result})
print(result)

df_answer = pd.read_csv('C:/Python/PyProject/빅분기/기출 및 필수모의/7회데이터셋/07.02.03-sales_test_dataset_y.csv')
```

```
rmse_result = metrics.root_mean_squared_error(df_answer['Sales'], pred_result)
print(rmse_result)
```

XGboost

```
from xgboost import XGBRegressor
model_XG = XGBRegressor()
model_XG.fit(X_train, y_train)
pred = model_XG.predict(X_test)
```

```
XG_rmse = metrics.root_mean_squared_error(y_test,pred)
print(XG_rmse)
```

```
XG_predict = model_XG.predict(test)
result = pd.DataFrame({'BranchName':df_test['BranchName'],'Sales' : XG_predict})
print(result)
```

df_answer = pd.read_csv('C:/Python/PyProject/빅분기/기출 및 필수모의/7회데이터셋/07.02.03-sales_test_dataset_y.csv')

XG_rmse_result = metrics.root_mean_squared_error(df_answer['Sales'],XG_predict)
print(XG_rmse_result)

print(f'랜덤포레스트 rmse :{rmse_result}, XG부스트 rmse :{XG_rmse_result}')

2. 범주형 인코딩 방식 비교

항목 자기 방식 책(예시 코드)

사용 도구 pd.get_dummies() (원핫인코딩) LabelEncoder() (라벨인코딩)

예시 수원 → [0,1,0] 수원 → 1

적용 대상 total = concat(train+test) 후 전체에 적용 train+test 전체 기준으로 fit, 후에 transform만 나눠서

모델 적합도 원핫은 비선형모델에 유리 (랜덤포레스트에 매우 적합) 라벨은 순서 관계를 암묵적으로 부여 (신경 써야 함)

결론:

자기처럼 원핫인코딩한 방식은 RandomForest, XGBoost 같은 모델에 훨씬 더 적합해!

라벨인코딩은 순서 의미가 있는 경우 (ex. 등급, 단계)에 적합하지만

의미 없는 범주에 쓰면 모델이 잘못 해석할 수 있어.

그래서 자기 방식이 훨씬 안전하고, 시험에서도 추천되는 방식이야!

종합 결론

비교 항목 자기 방식 책 방식 추천

수치형 스케일링 MinMaxScaler StandardScaler 둘 다 가능

범주형 인코딩 One-Hot (pd.get_dummies) LabelEncoder 자기 방식 우세

!!!