범주형 변수 인코딩

- `LabelEncoder`
  - 범주형 변수의 각 카테고리(class)를 0부터 시작하는 정수로 변환
    - `['Red', 'Green', 'Blue']` → `[2, 1, 0]`
  - **문제점**
    - 모델이 숫자들 사이에 순서나 크기 관계가 있다고 오해하게 만들 수 있다. (e.g., `Red`(2) > `Green`(1))
    - 순서가 중요한 순서형(Ordinal) 변수(e.g., '학력': '고졸' < '대졸' < '석사')에 적합하며, 순서가 없는 명목형(Nominal) 변수에는 부적합
- `OneHotEncoder` / `pd.get_dummies()`
  - 순서가 없는 명목형 변수를 다루기 위한 가장 일반적인 방법
  - 각 카테고리를 새로운 열(column)로 만들고, 해당 카테고리에 속하면 1, 그렇지 않으면 0으로 채우는 방식
    - `['Red', 'Green']` → `is_Red` / `is_Green` 두 개의 열 생성
  - `OneHotEncoder`
    - `scikit-learn`에서 제공하며, 머신러닝 파이프라인에 통합하기 용이
    - 학습 데이터에 없는 카테고리가 테스트 데이터에 나타날 경우 에러를 제어하는 기능
  - `pd.get_dummies()`
    - `pandas`에서 제공하며, 사용법이 매우 간단
    - 탐색적 데이터 분석 단계에서 빠르게 사용하기 좋음

적용 가능한 상황

- `LabelEncoder`:
  - 변수의 카테고리 간에 명확한 순서 관계가 존재할 때 (e.g., `[ 'Bronze', 'Silver', 'Gold' ]` -> `[ 0, 1, 2 ]`)
  - 트리 기반 모델(Decision Tree, Random Forest 등)에서는 수치의 크기에 영향을 받지 않으므로 명목형 변수에 사용해도 큰 문제는 없음
    - 다른 모델과의 호환성을 위해 일반적으로는 권장되지 않음
- `OneHotEncoder` / `pd.get_dummies()`:
  - 변수의 카테고리 간에 순서 관계가 없는 명목형 변수에 사용 (e.g., '국가': `[ '한국', '미국', '일본' ]`)
  - 선형 모델, SVM, 신경망 등 대부분의 머신러닝 알고리즘에서 범주형 변수를 처리하는 표준적인 방법

예제 데이터프레임 생성

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

```
df = pd.DataFrame({'item': ['TV', 'Phone', 'Laptop', 'TV', 'Phone'],
                   'grade': ['A', 'C', 'B', 'B', 'A'], # 순서가 있는 변수
                   'status': ['new', 'used', 'new', 'refurbished', 'used'] # 순서
가 없는 변수
                  })
```

1. LabelEncoder

- **용도**: 순서형 변수를 정수로 매핑
- **주의사항**
  - `fit`, `transform`은 학습 데이터에, `transform`은 테스트 데이터에 적용
    - `fit_transform` = `fit`+`transform`
  - 2D 배열이 아닌 1D 배열(시리즈)을 입력으로 받음
- **코드 예시**

```
# 순서형 변수 'grade'에 적용
encoder_label = LabelEncoder()
df['grade_encoded'] = encoder_label.fit_transform(df['grade'])

# 매핑 관계 확인
print("--- LabelEncoder Classes ---")
print(encoder_label.classes_) # ['A', 'B', 'C']

print("\n--- LabelEncoder Result ---")
print(df[['grade', 'grade_encoded']])
'''
--- LabelEncoder Result ---
grade  grade_encoded
0     A              0
1     C              2
2     B              1
3     B              1
4     A              0
'''
```

- **결과 해석**
  - `grade` 열의 'A', 'B', 'C'가 각각 0, 1, 2로 변환
  - `classes_` 속성을 통해 어떤 문자가 어떤 숫자에 매핑되었는지 확인 가능 (알파벳 순서로 자동 매핑됨)

2. pd.get_dummies()

- **용도**: 명목형 변수를 원-핫 인코딩 (사용이 매우 간편 / 빠름)
- **주의사항**
  - 카테고리의 수가 너무 많으면(high cardinality), 변수의 차원이 급격하게 늘어나는 "차원의 저주" 문제가 발생
  - `drop_first=True` 옵션을 사용하여 다중공선성(multicollinearity) 문제를 방지
- **코드 예시**

```
# 'status' 열에 대해 원-핫 인코딩 수행
dummies = pd.get_dummies(df['status'], prefix='status')

# 원본 데이터프레임과 합치기
df_dummies = pd.concat([df, dummies], axis=1)

print("--- pd.get_dummies() Result ---")
print(df_dummies)

# 다중공선성 방지를 위해 첫 번째 카테고리 열 제거
dummies_dropped = pd.get_dummies(df['status'], prefix='status',
drop_first=True)
print("\n--- with drop_first=True ---")
print(dummies_dropped)
```

- **결과 해석**
  - `status` 열이 `status_new`, `status_refurbished`, `status_used` 세 개의 열로 확장됨
  - `drop_first=True`를 사용하면 `status_new` 열이 제거되고, 나머지 두 열의 값이 모두 0이면 `new` 상태임을 유추 가능

3. OneHotEncoder

- **용도**: `scikit-learn` 파이프라인 내에서 명목형 변수를 원-핫 인코딩
- **주의사항**
  - 기본적으로 결과를 희소 행렬(sparse matrix)로 반환
    - 일반적인 배열로 보려면 `sparse=False` (scikit-learn 1.2+ 버전에서는 `sparse_output`) 또는 `.toarray()`를 사용
  - 2D 배열을 입력으로 받으므로 `df[['status']]`와 같이 사용
- **코드 예시**

```
# 1. 인코더 객체 생성
# handle_unknown='ignore': 테스트 데이터에 새로운 카테고리가 나타나면 모두 0으로
처리
encoder_ohe = OneHotEncoder(sparse=False, handle_unknown='ignore')

# 2. 데이터에 fit_transform 적용
ohe_transformed = encoder_ohe.fit_transform(df[['status']])

# 3. 결과를 데이터프레임으로 변환
df_ohe = pd.DataFrame(ohe_transformed,
columns=encoder_ohe.get_feature_names())

print("--- OneHotEncoder Result ---")
print(df_ohe)
```

- **결과 해석**
  - `pd.get_dummies`와 유사하게 `status_new`, `status_refurbished`, `status_used` 열이 생성됨

- `OneHotEncoder`는 `fit`을 통해 학습 데이터의 카테고리들을 기억하므로, 테스트 데이터 변환 시 일관성을 유지하는 데 더 안정적 (`inverse_transform`으로 원상복구 쉬움)

장단점 및 대안

인코딩 방법	장점	단점	대안
`LabelEncoder`	구현이 간단하고, 변수의 차원이 늘어나지 않음.	카테고리 간에 존재하지 않는 순서 관계를 모델이 학습할 수 있음 (성능 저하 유발).	**순서형 변수에만 제한적으로 사용**. 명목형 변수에는 부적합.
`pd.get_dummies()`	사용법이 매우 간단하고 직관적임. 데이터프레임에 바로 적용 가능.	학습 데이터에 없던 카테고리가 테스트 데이터에 나타나면 에러 처리 없이 열의 개수가 달라져 문제가 발생할 수 있음.	`OneHotEncoder` (테스트 데이터 처리 기능 제공).
`OneHotEncoder`	`scikit-learn` 파이프라인과 완벽하게 호환됨. `handle_unknown` 옵션으로 테스트 데이터의 새로운 카테고리에 안정적으로 대처 가능.	`get_dummies`보다 사용법이 다소 복잡함 (2D 입력, 결과 변환 등).	`pd.get_dummies()` (탐색적 분석 단계에서 신속하게 사용).

카디널리티가 높은 변수 처리 대안

원-핫 인코딩은 카테고리 종류가 수백, 수천 개에 달하는 고차원(high cardinality) 데이터(e.g., '사용자 ID', '우편번호')에는 부적합합니다. 이 경우 다음과 같은 대안을 고려할 수 있습니다.

```
# 예제 데이터 생성
import pandas as pd

df = pd.DataFrame({
    'item': ['TV', 'Phone', 'Laptop', 'TV', 'Phone'],
    'grade': ['A', 'C', 'B', 'B', 'A'],
    'status': ['new', 'used', 'new', 'refurbished', 'used'],
    'target': [100, 200, 150, 120, 180]   # 예시 타겟 변수
})
```

- **Target Encoding (Mean Encoding)**: 각 카테고리를 해당 카테고리에 속하는 데이터들의 타겟 변수(종속변수)의 평균값으로 대체합니다. 강력한 방법이지만 오버피팅의 위험이 있어 교차 검증 등을 통한 정교한 구현이 필요합니다.

```
# status별 target 평균값으로 매핑
mean_map = df.groupby('status')['target'].mean()
df['status_target_enc'] = df['status'].map(mean_map)

print(df[['status', 'status_target_enc']])
```

- **Frequency Encoding**: 각 카테고리를 데이터에 나타나는 빈도수로 대체합니다.

```
# value_counts로 빈도수 추출
freq_map = df['status'].value_counts()
df['status_freq_enc'] = df['status'].map(freq_map)

print(df[['status', 'status_freq_enc']])
```

- **Binary Encoding**: 카테고리를 정수로 변환한 뒤, 이진수(binary)로 변환하고, 각 자릿수를 새로운 열로 만듭니다. 원-핫 인코딩보다 차원을 훨씬 적게 사용합니다.

```
# pip install category_encoders  (사전 설치 필요)

import category_encoders as ce

encoder = ce.BinaryEncoder(cols=['status'])
df_binary = encoder.fit_transform(df['status'])

print(df_binary)

# 직접 구현할 경우
# status를 정수로 변환
le = LabelEncoder()
df['status_int'] = le.fit_transform(df['status'])

# 정수를 2진수 문자열로 변환 후, 각 자리 분리
max_bits = df['status_int'].max().bit_length()
for i in range(max_bits):
    df[f'status_bin_{i}'] = df['status_int'].apply(lambda x: (x >> i) & 1)

print(df[['status', 'status_int'] + [f'status_bin_{i}' for i in
range(max_bits)]])
```