데이터 가공 및 시각화

- 1. 전처리
  - 결측값 처리: 단순대치, 평균 대치, 단순확률 대치 (Hot-deck, nearest neighbor), 다중 대치, knnImputation, centralimputation
  - 클래스불균형: 업샘플링 (SMOTE, Boaderline SMOTE, Adasyn), 다운샘플링
  - 이상값 처리: 극단값 절단, 조정
  - **변수 변환, 스케일링: 수치형 변수 변환(로그변환, 제곱근변환, 지수변환, 제곱변환, Box-cox 변환, 표준화, 정규화), 범주형 변수 변환(범주형 변수 인코딩, 대규모 범주형 변수처리), 날짜 및 변수 변환, 피쳐스케일링**
  - 원핫인코딩(더미변수), 컬럼 트랜스퍼, 구간분할, 이산화, 피쳐선택
- 1. 표본 추출: 단순랜덤 추출법, 계통추출법, 집락추출법, 층화추출법
- 1. 데이터 분할: 구축/검정/시험용, 홀드아웃방법, 교차확인방법 (10 fold 교차분석), 부트스트랩
- 1. 그래프 그리기:
  - 산점도, 막대그래프, 선그래프, 히트맵, 서브플롯, 트리맵, 도넛차트, 버블차트, 히스토그램, 체르노프 페이스, 스타차트, 다차원척도법, 평행좌표계
  - 도식화와 시각화

범주형 변수 변환 (Categorical feature)

특정 애플리케이션에 가장 적합한 데이터 표현을 찾는 것을 특성 공학 (feature engineering)이라고 한다. 올바른 데이터 표현은 지도 학습 모델에서 적절한 매개변수를 선택하는 것보다 성능에 더 큰 영향을 미친다. 여기서는 범주형 특성 (categorical feature) 혹은 이산형 특성 (discrete feature)를 변환하는 방법들을 살펴보려고 한다.

GBDT와 같이 결정트리에 기반을 두는 모델에서는 레이블 인코딩으로 범주형 변수를 변환하는 게 가장 편리하지만, 타겟 인코딩이 더 효과적일 때도 많다. 다만 타겟 인코딩은 데이터 정보 누출의 위험이 있다. 원핫인코딩이 가장 전통적인 방식이고, 신경망의 경우에는 임베딩 계층을 변수별로 구성하는게 조금 번거롭지만 유효하다.

- 범주형 변수 변환
  - 원핫인코딩(One-hot-encoding), 더미코딩(dummy coding), 이펙트코딩(Effect coding), 숫자로 표현된 범주형 특성, 레이블인코딩(Label encoding), 특징 해싱(Feature Hashing), 빈도인코딩(Frequency encoding)

범주형 변수 변환 - 1) 원핫인코딩 (One-hot-encoding) with get_dummies, OneHotEncoder, ColumnTransformer

One-out-of-N encoding, 가변수(dummy variable)라고도 한다. 범주형 변수를 0 또는 1 값을 가진 하나 이상의 새로운 특성으로 바꾼 것이다. 0과 1로 표현된 변수는 선형 이진 분류 공식에 적용할 수 있어서 개수에 상관없이 범주마다 하나의 특성으로 표현한다.

원핫인코딩은 통계학의 dummy coding과 비슷하지만 완전히 같지는 않다. 간편하게 하려고 각 범주를 각기 다른 이진 특성으로 바꾸었기 때문이다. 이는 분석의 편리성 (데이터 행렬의 랭크 부족 현상을 피하기 위함) 때문이다.

훈련데이터와 테스트데이터 모두를 포함하는 df를 사용해서 get_dummies 함수를 호출하든지 또는 각각 get_dummies를 호출한 후에 훈련 세트와 테스트 세트의 열이름을 비교해서 같은 속성인지를 확인해야 한다.

특징의 개수가 범주형 변수의 레벨 개수에 따라 증가하기 때문에 정보가 적은 특징이 대량 생성돼서 학습에 필요한 계산 시간이나 메모리가 급증한다. 따라서 범주형 변수의 레벨이 너무 많을 때는 다른 인코딩 방법을 검토하거나 범주형 변수의 레벨 개수를 줄이거나 빈도가 낮은 범주를 기타 범주로 모아 정리하는 방법을 써야 한다.

구현이 쉽고 가장 정확하며 온라인 학습이 가능한 반면, 계산 측면에서 비효율적이고 범주 수가 증가하는 경우에 적합하지 않고, 선형 모델 외에는 적합하지 않으며, 대규모 데이터셋일 경우 대규모 분산 최적화가 필요하다.

pandas의 get_dummies(데이터) 함수를 사용하거나 scikit learn의 OneHotEncoder 혹은 ColumnTransformer를 사용할 수 있다.

- OneHotEncoder는 모든 특성을 범주형이라고 가정하여 수치형 열을 포함한 모든 열에 인코딩을 수행한다. 문자열 특성과 정수 특성이 모두 변환되는 것이다.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('/Users/benny/Desktop/datascience/heart.csv', na_values=['',
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         916 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

In [4]:
```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder(sparse=False) # sparse=True면 DataFrame 반환
print(ohe.fit_transform(df))
print(ohe.get_feature_names())
```

```
[[0. 0. 0. ... 1. 1. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 1. 1. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 1. 1. 0.]]
['x0_28' 'x0_29' 'x0_30' 'x0_31' 'x0_32' 'x0_33' 'x0_34' 'x0_35' 'x0_36'
 'x0_37' 'x0_38' 'x0_39' 'x0_40' 'x0_41' 'x0_42' 'x0_43' 'x0_44' 'x0_45'
 'x0_46' 'x0_47' 'x0_48' 'x0_49' 'x0_50' 'x0_51' 'x0_52' 'x0_53' 'x0_54'
 'x0_55' 'x0_56' 'x0_57' 'x0_58' 'x0_59' 'x0_60' 'x0_61' 'x0_62' 'x0_63'
 'x0_64' 'x0_65' 'x0_66' 'x0_67' 'x0_68' 'x0_69' 'x0_70' 'x0_71' 'x0_72'
 'x0_73' 'x0_74' 'x0_75' 'x0_76' 'x0_77' 'x1_F' 'x1_M' 'x2_ASY' 'x2_ATA'
 'x2_NAP' 'x2_TA' 'x3_0' 'x3_80' 'x3_92' 'x3_94' 'x3_95' 'x3_96' 'x3_98'
 'x3_100' 'x3_101' 'x3_102' 'x3_104' 'x3_105' 'x3_106' 'x3_108' 'x3_110'
```

```
'x3_112' 'x3_113' 'x3_114' 'x3_115' 'x3_116' 'x3_117' 'x3_118' 'x3_120'
'x3_122' 'x3_123' 'x3_124' 'x3_125' 'x3_126' 'x3_127' 'x3_128' 'x3_129'
'x3_130' 'x3_131' 'x3_132' 'x3_133' 'x3_134' 'x3_135' 'x3_136' 'x3_137'
'x3_138' 'x3_139' 'x3_140' 'x3_141' 'x3_142' 'x3_143' 'x3_144' 'x3_145'
'x3_146' 'x3_148' 'x3_150' 'x3_152' 'x3_154' 'x3_155' 'x3_156' 'x3_158'
'x3_160' 'x3_164' 'x3_165' 'x3_170' 'x3_172' 'x3_174' 'x3_178' 'x3_180'
'x3_185' 'x3_190' 'x3_192' 'x3_200' 'x4_0' 'x4_85' 'x4_100' 'x4_110'
'x4_113' 'x4_117' 'x4_123' 'x4_126' 'x4_129' 'x4_131' 'x4_132' 'x4_139'
'x4_141' 'x4_142' 'x4_147' 'x4_149' 'x4_152' 'x4_153' 'x4_156' 'x4_157'
'x4_159' 'x4_160' 'x4_161' 'x4_163' 'x4_164' 'x4_165' 'x4_166' 'x4_167'
'x4_168' 'x4_169' 'x4_170' 'x4_171' 'x4_172' 'x4_173' 'x4_174' 'x4_175'
'x4_176' 'x4_177' 'x4_178' 'x4_179' 'x4_180' 'x4_181' 'x4_182' 'x4_183'
'x4_184' 'x4_185' 'x4_186' 'x4_187' 'x4_188' 'x4_190' 'x4_192' 'x4_193'
'x4_194' 'x4_195' 'x4_196' 'x4_197' 'x4_198' 'x4_199' 'x4_200' 'x4_201'
'x4_202' 'x4_203' 'x4_204' 'x4_205' 'x4_206' 'x4_207' 'x4_208' 'x4_209'
'x4_210' 'x4_211' 'x4_212' 'x4_213' 'x4_214' 'x4_215' 'x4_216' 'x4_217'
'x4_218' 'x4_219' 'x4_220' 'x4_221' 'x4_222' 'x4_223' 'x4_224' 'x4_225'
'x4_226' 'x4_227' 'x4_228' 'x4_229' 'x4_230' 'x4_231' 'x4_232' 'x4_233'
'x4_234' 'x4_235' 'x4_236' 'x4_237' 'x4_238' 'x4_239' 'x4_240' 'x4_241'
'x4_242' 'x4_243' 'x4_244' 'x4_245' 'x4_246' 'x4_247' 'x4_248' 'x4_249'
'x4_250' 'x4_251' 'x4_252' 'x4_253' 'x4_254' 'x4_255' 'x4_256' 'x4_257'
'x4_258' 'x4_259' 'x4_260' 'x4_261' 'x4_262' 'x4_263' 'x4_264' 'x4_265'
'x4_266' 'x4_267' 'x4_268' 'x4_269' 'x4_270' 'x4_271' 'x4_272' 'x4_273'
'x4_274' 'x4_275' 'x4_276' 'x4_277' 'x4_278' 'x4_279' 'x4_280' 'x4_281'
'x4_282' 'x4_283' 'x4_284' 'x4_285' 'x4_286' 'x4_287' 'x4_288' 'x4_289'
'x4_290' 'x4_291' 'x4_292' 'x4_293' 'x4_294' 'x4_295' 'x4_297' 'x4_298'
'x4_299' 'x4_300' 'x4_302' 'x4_303' 'x4_304' 'x4_305' 'x4_306' 'x4_307'
'x4_308' 'x4_309' 'x4_310' 'x4_311' 'x4_312' 'x4_313' 'x4_315' 'x4_316'
'x4_318' 'x4_319' 'x4_320' 'x4_321' 'x4_322' 'x4_325' 'x4_326' 'x4_327'
'x4_328' 'x4_329' 'x4_330' 'x4_331' 'x4_333' 'x4_335' 'x4_336' 'x4_337'
'x4_338' 'x4_339' 'x4_340' 'x4_341' 'x4_342' 'x4_344' 'x4_347' 'x4_349'
'x4_353' 'x4_354' 'x4_355' 'x4_358' 'x4_360' 'x4_365' 'x4_369' 'x4_384'
'x4_385' 'x4_388' 'x4_392' 'x4_393' 'x4_394' 'x4_404' 'x4_407' 'x4_409'
'x4_412' 'x4_417' 'x4_458' 'x4_466' 'x4_468' 'x4_491' 'x4_518' 'x4_529'
'x4_564' 'x4_603' 'x5_0' 'x5_1' 'x6_LVH' 'x6_Normal' 'x6_ST' 'x7_60'
'x7_63' 'x7_67' 'x7_69' 'x7_70' 'x7_71' 'x7_72' 'x7_73' 'x7_77' 'x7_78'
'x7_80' 'x7_82' 'x7_83' 'x7_84' 'x7_86' 'x7_87' 'x7_88' 'x7_90' 'x7_91'
'x7_92' 'x7_93' 'x7_94' 'x7_95' 'x7_96' 'x7_97' 'x7_98' 'x7_99' 'x7_100'
'x7_102' 'x7_103' 'x7_104' 'x7_105' 'x7_106' 'x7_107' 'x7_108' 'x7_109'
'x7_110' 'x7_111' 'x7_112' 'x7_113' 'x7_114' 'x7_115' 'x7_116' 'x7_117'
'x7_118' 'x7_119' 'x7_120' 'x7_121' 'x7_122' 'x7_123' 'x7_124' 'x7_125'
'x7_126' 'x7_127' 'x7_128' 'x7_129' 'x7_130' 'x7_131' 'x7_132' 'x7_133'
'x7_134' 'x7_135' 'x7_136' 'x7_137' 'x7_138' 'x7_139' 'x7_140' 'x7_141'
'x7_142' 'x7_143' 'x7_144' 'x7_145' 'x7_146' 'x7_147' 'x7_148' 'x7_149'
'x7_150' 'x7_151' 'x7_152' 'x7_153' 'x7_154' 'x7_155' 'x7_156' 'x7_157'
'x7_158' 'x7_159' 'x7_160' 'x7_161' 'x7_162' 'x7_163' 'x7_164' 'x7_165'
'x7_166' 'x7_167' 'x7_168' 'x7_169' 'x7_170' 'x7_171' 'x7_172' 'x7_173'
'x7_174' 'x7_175' 'x7_176' 'x7_177' 'x7_178' 'x7_179' 'x7_180' 'x7_181'
'x7_182' 'x7_184' 'x7_185' 'x7_186' 'x7_187' 'x7_188' 'x7_190' 'x7_192'
'x7_194' 'x7_195' 'x7_202' 'x8_N' 'x8_Y' 'x9_-2.6' 'x9_-2.0' 'x9_-1.5'
'x9_-1.1' 'x9_-0.9' 'x9_-0.8' 'x9_-0.7' 'x9_-0.5' 'x9_-0.1' 'x9_0.0'
'x9_0.1' 'x9_0.2' 'x9_0.3' 'x9_0.4' 'x9_0.5' 'x9_0.6' 'x9_0.7' 'x9_0.8'
'x9_0.9' 'x9_1.0' 'x9_1.1' 'x9_1.2' 'x9_1.3' 'x9_1.4' 'x9_1.5' 'x9_1.6'
'x9_1.7' 'x9_1.8' 'x9_1.9' 'x9_2.0' 'x9_2.1' 'x9_2.2' 'x9_2.3' 'x9_2.4'
'x9_2.5' 'x9_2.6' 'x9_2.8' 'x9_2.9' 'x9_3.0' 'x9_3.1' 'x9_3.2' 'x9_3.4'
'x9_3.5' 'x9_3.6' 'x9_3.7' 'x9_3.8' 'x9_4.0' 'x9_4.2' 'x9_4.4' 'x9_5.0'
'x9_5.6' 'x9_6.2' 'x9_nan' 'x10_Down' 'x10_Flat' 'x10_Up' 'x11_0' 'x11_1']
```

In [6]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
ct = ColumnTransformer([
    ('scaling', StandardScaler(), ['Age', 'RestingBP', 'Cholesterol', 'Fastin
                                   'MaxHR','Oldpeak']),
    ('onehot', OneHotEncoder(sparse=False), ['Sex', 'ChestPainType', 'Resting
```

```
                                                'ExerciseAngina', 'ST_Slope', 'He
    ])
OC = ct.fit_transform(df)
OC
```

Out[6]:
```
array([[-1.4331398 ,  0.41090889,  0.82507026, ...,  1.        ,
         1.        ,  0.        ],
       [-0.47848359,  1.49175234, -0.17196105, ...,  0.        ,
         0.        ,  1.        ],
       [-1.75135854, -0.12951283,  0.7701878 , ...,  1.        ,
         1.        ,  0.        ],
       ...,
       [ 0.37009972, -0.12951283, -0.62016778, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.37009972, -0.12951283,  0.34027522, ...,  0.        ,
         0.        ,  1.        ],
       [-1.64528563,  0.30282455, -0.21769643, ...,  1.        ,
         1.        ,  0.        ]])
```

범주형 변수 변환 - 2) 더미코딩 (Dummy coding) with get_dummies(drop_first=True)

더미코딩은 pandas의 get_dummies 함수에서 파라미터 drop_first=True를 설정함으로써 구현할 수 있다.

범주형 변수의 레벨이 n개일때 해당 레벨 개수만큼 가변수를 만들면 다중공선성이 생기기 때문에 이를 방지하기 위해 n-1개의 가변수를 만드는 방법을 쓰는 것이 더미코딩이다.

In [7]:
```
dummies = pd.get_dummies(df)
print(list(df.columns), df.shape)
print(list(dummies.columns), dummies.shape) # 각 범주형 특성의 값마다 새로운 특성이 됨
```

```
['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS', 'Rest
ingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope', 'HeartDisease'] (91
8, 12)
['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak', 'HeartDis
ease', 'Sex_F', 'Sex_M', 'ChestPainType_ASY', 'ChestPainType_ATA', 'ChestPainT
ype_NAP', 'ChestPainType_TA', 'RestingECG_LVH', 'RestingECG_Normal', 'RestingE
CG_ST', 'ExerciseAngina_N', 'ExerciseAngina_Y', 'ST_Slope_Down', 'ST_Slope_Fla
t', 'ST_Slope_Up'] (918, 21)
```

In [8]:
```
pd.get_dummies(df, drop_first=True).shape
```

Out[8]:
```
(918, 16)
```

범주형 변수 변환 - 3) 이펙트코딩 (Effect coding)

통계학에서 나온 범주형 변수에 대한 또 다른 변형이다. 더미코딩과 유사하지만 기준 범주가 모두 -1의 벡터로 표현된다는 것이 차이점이다. 선형 회귀 모델의 결과를 해석하기가 더 쉽다. 이펙트 코딩에서는 기준 범주를 나타내는 단일 feature가 없기 때문에 기준 범주의 효과는 다른 모든 범주의 계수의 음수 합계로서 별도로 계산해야 한다.

여러 개의 범주형 변수를 모델에서 다룬다면 이펙트 코딩이든 더미 코딩이든 큰 차이가 없지만, 두 개의 범주형 변수가 상호작용이 있는 경우에는 이펙트 코딩이 더 이점을 가진다. 이펙트 코딩으로 합리적인 주효과와 상호작용의 추정치를 얻을 수 있다. 더미코딩의 경우, 상호작용 추정치는 괜찮지만 주효과는 진짜 주효과가 아니라 simple effect에 더 가깝다.

범주형 변수 변환 - 4) 숫자로 표현된 범주형 특성 with get_dummies

데이터 취합 방식에 따라 범주형 변수인데 숫자로 인코딩된 경우가 많다. 예를 들어 문자열이 아닌 답안 순서대로 0~8까지의 숫자로 채워지는 설문응답 데이터가 있다. 이 값은 이산적이기 때문에 연속형 변수로 다루면 안 된다.

숫자 특성도 가변수로 만들고 싶다면 get_dummies를 사용하여 아래와 같이 적용하면 된다.

- get_dummies(columns=[숫자 특성도 포함하여 인코딩하려는 열을 나열])
- 데이터프레임 단에서 숫자 특성을 str 속성으로 변경해준 뒤 get_dummies 진행

In [9]:
```
df_ = df.copy()
df_['HeartDisease'] = df_['HeartDisease'].astype(str)
dummies2 = pd.get_dummies(df_)
print(list(dummies2.columns), dummies.shape)
```

```
['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak', 'Sex_F',
'Sex_M', 'ChestPainType_ASY', 'ChestPainType_ATA', 'ChestPainType_NAP', 'Chest
PainType_TA', 'RestingECG_LVH', 'RestingECG_Normal', 'RestingECG_ST', 'Exercis
eAngina_N', 'ExerciseAngina_Y', 'ST_Slope_Down', 'ST_Slope_Flat', 'ST_Slope_U
p', 'HeartDisease_0', 'HeartDisease_1'] (918, 21)
```

범주형 변수 변환 - 5) 레이블 인코딩 (Label encoding) with LabelEncoder

각 레벨을 단순히 정수로 변환하는 방법이다. Ordinal encoding이라고도 한다. 5개의 레벨이 있는 범주형 변수는 각 레벨이 0~4까지의 수치로 바뀐다.

사전 순으로 나열했을 때의 인덱스 수치는 대부분 본질적인 의미가 없다. 따라서 결정 트리 모델에 기반을 둔 방법이 아닐 경우 레이블 인코딩으로 변환한 특징을 학습에 직접 이용하는 건 그다지 적절하지 않다. 결정트리에서는 범주형 변수의 특정 레벨만 목적 변수에 영향을 줄 때도 분기를 반복함으로써 예측값에 반영할 수 있으므로 학습에 활용할 수 있다.

GBDT모델에서 레이블 인코딩은 범주형 변수를 변환하는 기본적인 방법이다.

In [11]:
```
from sklearn.preprocessing import LabelEncoder
LEdf = pd.DataFrame()
for col in df.columns:
    le = LabelEncoder()
    le.fit(df[col])
    LEdf[col]=le.transform(df[col])
LEdf
```

Out[11]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exerc
0	12	1	1	41	147	0	1	98	
1	21	0	2	55	40	0	1	82	
2	9	1	1	31	141	0	2	25	
3	20	0	0	39	72	0	1	34	
4	26	1	2	49	53	0	1	48	
...	...	...	...	...	...	...	...	...	
913	17	1	3	14	122	0	1	58	

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exerc
914	40	1	0	45	51	1	1	67	
915	29	1	0	31	9	0	1	41	
916	29	0	1	31	94	0	0	100	
917	10	1	2	39	35	0	1	99	

918 rows × 12 columns

In [12]:
```
df
```

Out[12]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exerc
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	
...	...	...	...	...	...	...	...	...	
913	45	M	TA	110	264	0	Normal	132	
914	68	M	ASY	144	193	1	Normal	141	
915	57	M	ASY	130	131	0	Normal	115	
916	57	F	ATA	130	236	0	LVH	174	
917	38	M	NAP	138	175	0	Normal	173	

918 rows × 12 columns

범주형 변수 변환 - 6) 특징 해싱 (Feature Hashing) with FeatureHasher

원핫인코딩으로 변환한 뒤 특징의 수는 범주의 레벨 수와 같아지는데 특징 해싱은 그 수를 줄이는 변환방법이다. 변환 후의 특징 수를 먼저 정해두고(파라미터 n_features) 해시 함수를 이용하여 레벨별로 플래그를 표시할 위치를 경정한다.

원핫인코딩에서는 레벨마다 서로 다른 위치에 플래그를 표시하지만 특징 해싱에서는 변환 후에 정해진 특징 수가 범주의 레벨 수보다 적으므로 해시 함수에 따른 계산에 의해 다른 레벨에서도 같은 위치에 플래그를 표시할 수 있다.

구현하기 쉽고, 모델 학습에 비용이 적게 들며, 새로운 범주 추가가 쉽고, 희귀 범주 처리가 쉽고, 온라인 학습이 가능한 장점을 가지고 있다. 반면, 선형 또는 커널 모델에만 적합하고 해시된 feature는 해석이 불가하며 정확도에 대해 엇갈린 보고가 있다.

Scikit Learn의 FeatureHasher 함수로 각 열을 대상으로

In [17]:
```
from sklearn.feature_extraction import FeatureHasher
```

```
FHdf = pd.DataFrame(None)
for col in df.columns:
    fh = FeatureHasher(n_features=3, input_type='string')
    hash_df = fh.fit_transform(df[[col]].astype(str).values)
    hash_df = pd.DataFrame(hash_df.todense(), columns=[f'{col}_{i}' for i in
    FHdf = pd.concat([FHdf, hash_df], axis=1)
FHdf
```

Out[17]:

	Age_0	Age_1	Age_2	Sex_0	Sex_1	Sex_2	ChestPainType_0	ChestPainType_1	ChestP
0	-1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
1	0.0	-1.0	0.0	0.0	0.0	-1.0	-1.0	0.0	
2	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
3	-1.0	0.0	0.0	0.0	0.0	-1.0	0.0	0.0	
4	-1.0	0.0	0.0	0.0	1.0	0.0	-1.0	0.0	
...	...	...	...	...	...	...	...	...	
913	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
914	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	
915	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
916	1.0	0.0	0.0	0.0	0.0	-1.0	0.0	0.0	
917	-1.0	0.0	0.0	0.0	1.0	0.0	-1.0	0.0	

918 rows × 36 columns

범주형 변수 변환 - 7) 빈도 인코딩 (Frequency Encoding) with value_counts, map

각 레벨의 출현 횟수 혹은 출현 빈도로 범주형 변수를 대체하는 방법이다. 각 레벨의 출현 빈도와 목적변수 간에 관련성이 있을 때 유효하다.

레이블 인코딩의 변형으로서 사전순으로 나열한 순서 인덱스가 아닌 출현 빈도순으로 나열하는 인덱스를 만들기 위해 사용할 수도 있다. 동률의 값이 발생할 수 있으니 주의해야 한다. 또한, 수치형 변수 스케일링과 마찬가지로 학습데이터와 테스트 데이터를 따로따로 정의하여 변환해버리면 다른 의미의 변수가 되므로 조심해야 한다.

In [18]:

```
FEdf = df.copy()
for col in FEdf.columns:
    freg = FEdf[col].value_counts()
    FEdf[col] = FEdf[col].map(freg)
FEdf
```

Out[18]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exerc
0	13	725	173	107	6	704	552	10	
1	21	193	203	50	3	704	552	10	
2	11	725	173	118	5	704	178	9	
3	31	193	496	17	7	704	552	8	
4	51	725	203	55	7	704	552	20	

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exerc
...	...	...	...	...	...	...	...	...	
913	18	725	46	58	6	704	552	11	
914	10	725	496	8	6	214	552	6	
915	38	725	496	118	1	704	552	16	
916	38	193	173	118	6	704	188	7	
917	16	725	203	17	4	704	552	7	

918 rows × 12 columns

In [ ]: