『 3과목 :』
데이터 마트와 데이터 전처리

- Data Mart & Data Preprocessing
- Data Structures
- Data Gathering(Collect, Acquisition), Data Ingestion
- Data Invest & Exploratory Data Analysis, Data Visualization
- Data Cleansing (정제)
- Data Integration (통합)
- Data Reduction (축소)
- Data Transformation (변환)
- Feature Engineering & Data Encoding
- **Cross Validation & Data Splitting**
- Data Quality Assessment and Model Performance Evaluation
- 『3과목』 Self 점검

Copyright 2022. DaeKyeong all rights reserved

Syllabus

학습목표

- 이 워크샵에서는 Cross Validation & Data Splitting 에 대해 알 수 있습니다.

눈높이 체크

- Cross Validation & Data Splitting 을 알고 계신가요?

1. 교차 검증 (Cross Validation, CV)

교차검증?

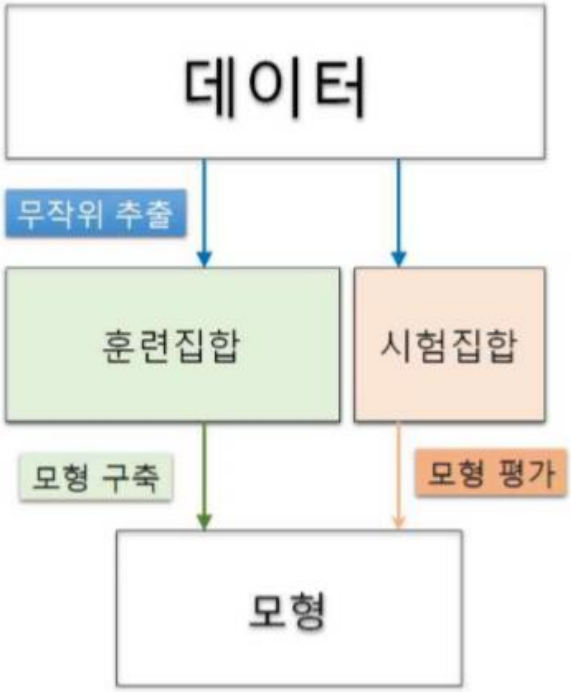
- 데이터 전처리 파이프라인을 만들고 모델을 훈련한 다음 교차검증으로 평가.
  - 처음 모델을 훈련하고 어떤 성능 지표(정확도, 제곱 오차 등)를 사용하여 동작하는 계산, 그러나 이러한 방법은 모델을 훈련한 데이터로 모델을 평가함으로 원하는 것과 다름.
  - 따라서 훈련 데이터가 아닌 이전에 본 적 없는 새로운 데이터에서 잘 동작하는 지를 평가해야 함
  - 데이터를 여러 개의 **fold(조각)**으로 나눈 후, 여러 번 훈련·평가를 반복하여 평균 성능을 평가하는 방식.

  - K-Fold Cross Validation
    - 데이터를 K등분하고, 그 중 1개는 테스트용, 나머지는 학습용으로 사용
    - K번 반복하면서 테스트 데이터를 바꿔가며 평균 성능 측정
    - 이외 에도 StratifiedKFold, Leave-One-Out, ShuffleSplit 등 다양한 교차 검증이 있음

1. 교차 검증 (Cross Validation, CV)

홀드아웃 교차 방법 개념

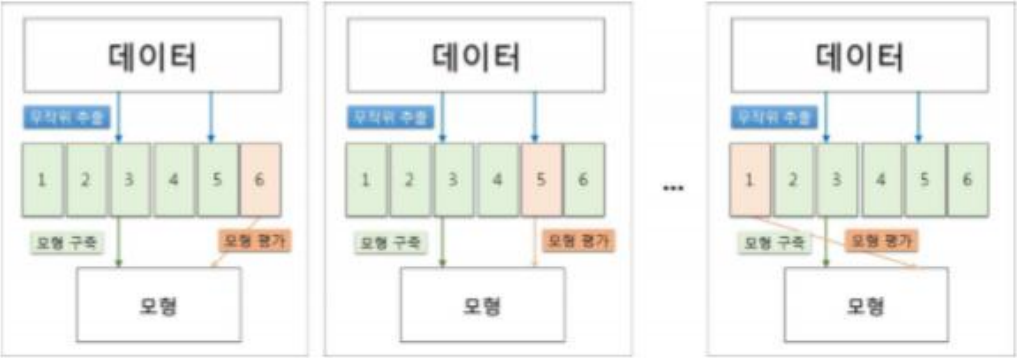
- 개념
  - 데이터 집합을 서로 겹치지 않는 훈련 집합(training set)과 시험 집합(test set)으로 무작위로 구분한 후, 훈련 집합을 이용하여 분석 모형을 구축하고 시험 집합을 이용하여 분석 모형의 성능을 평가하는 기법이다(P. Tan, M. Steinbach, and V. Kumar, 2007). 훈련 집합과 시험 집합의 비율은 50-50, 70-30 등 사용자에 의해서 결정된다.



1. 교차 검증 (Cross Validation, CV)

다중 교차 방법

- **개념**
  - 데이터 집합을 무작위로 동일 크기를 갖는 k개의 부분 집합으로 나누고, 그중 1개를 시험 집합으로, 나머지 k-1개를 훈련 집합으로 선정하여 분석 모형을 평가한다(P. Tan, M. Steinbach, and V. Kumar, 2007). 이러한 방식으로 모든 부분 집합들을 시험 집합으로 정확히 1회씩 선정하여 총 k번 반복한다.
- **특징**
  - 이 방법은 홀드아웃 교차 검증과는 달리 모든 데이터 집합을 훈련 집합 및 시험 집합으로 사용하기 때문에 분석 모형의 평가 결과가 편향되지 않는다는 장점이 있다. [그림]은 k=6인 다중 교차 검증의 개념을 나타낸다.



1. 교차 검증 (Cross Validation, CV)

기타

- Random subsampling:Holdout 방식 반복
  - K개의 부분 데이터 셋 사용: 각 데이터 셋은 랜덤
  - 최종 성능은 각 실험 성능의 평균으로 도출
- stratified sampling
  - 각 클래스로부터 일정 비율 샘플 추출
  - 전체 데이터에서 무작위로 추출할 경우 표본이 특정 클래스에 편중될 수 있기 때문에 사용
- Botstrap
  - 중복 추출 허용

1. 교차 검증 (Cross Validation, CV)

교차검증?

- 예시 (5-Fold Cross Validation)

Fold	Train 데이터	Test 데이터
1	Fold 2~5	Fold 1
2	Fold 1,3~5	Fold 2
…	…	…
5	Fold 1~4	Fold 5

1. 교차 검증 (Cross Validation, CV)

교차검증 모델 만들기

- 10개의 홀더를 사용, 평가 점수는 cv_results 에 저장

```
# 라이브러리를 임포트합니다.
from sklearn import datasets
from sklearn import metrics
from sklearn.model_selection import KFold, cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# 숫자 데이터셋을 로드합니다.
digits = datasets.load_digits()

# 특성 행렬을 만듭니다.
features = digits.data

# 타깃 벡터를 만듭니다.
target = digits.target
```

1. 교차 검증 (Cross Validation, CV)

교차검증 모델 만들기

```
# 로지스틱 회귀 객체를 만듭니다.
logit = LogisticRegression()

# 표준화 객체를 생성합니다.
standardizer = StandardScaler()

# 표준화한 다음 로지스틱 회귀를 실행하는 파이프라인을 만듭니다.
pipeline = make_pipeline(standardizer, logit)

# k-폴드 교차검증을 만듭니다.
kf = KFold(n_splits=10, shuffle=True, random_state=1)

# k-폴드 교차검증을 수행합니다.
cv_results = cross_val_score(pipeline, # 파이프라인
                             features, # 특성 행렬
                             target, # 타깃 벡터
                             cv=kf, # 교차 검증 기법
                             scoring="accuracy", # 평가 지표
                             n_jobs=-1) # 모든 CPU 코어 사용
```

1. 교차 검증 (Cross Validation, CV)

교차검증 모델 만들기

```
# 평균을 계산합니다.
print("평균 : {}".format(cv_results.mean()))

# 10개 폴드의 점수를 모두 확인하기
print("10개 폴드의 점수 : {}".format(cv_results))
```

```
>>

평균 : 0.9693916821849783
10개 폴드의 점수 : [0.97777778 0.98888889 0.96111111 0.94444444 0.97777778 0.98333333
 0.95555556 0.98882682 0.97765363 0.93854749]
```

1. 교차 검증 (Cross Validation, CV)

사이킷런의 pipeline 패키지

- 사이킷런의 pipeline 패키지는 교차검증 기법을 사용할 때 손쉽게 구현하도록 함. 먼저 standardizer로 전처리를 하고 logit 회귀를 훈련하는 파이프라인을 생성함

```
# 라이브러리를 임포트합니다.
from sklearn.model_selection import train_test_split

# 훈련 세트와 테스트 세트를 만듭니다.
features_train, features_test, target_train, target_test = train_test_split(
    features, target, test_size=0.1, random_state=1)

# 표준화 객체를 만듭니다.
standardizer = StandardScaler()
```

1. 교차 검증 (Cross Validation, CV)

사이킷런의 pipeline 패키지

```
# 훈련 세트로 standardizer의 fit 메서드를 호출합니다.
standardizer.fit(features_train)

# 훈련 세트와 테스트 세트에 모두 적용합니다.
features_train_std = standardizer.transform(features_train)
features_test_std = standardizer.transform(features_test)

# 파이프라인을 만듭니다.
pipeline = make_pipeline(standardizer, logit)

# k-폴드 교차 검증 수행
cv_results = cross_val_score(pipeline, # 파이프라인
                             features, # 특성 행렬
                             target, # 타깃 벡터
                             cv=kf, # 교차검증
                             scoring="accuracy", # 평가 지표
                             n_jobs=-1) # 모든 CPU 코어 사용
```

1. 교차 검증 (Cross Validation, CV)

사이킷런의 pipeline 패키지

```
# 평균을 계산합니다.
print("평균 : {}".format(cv_results.mean()))

# 10개 폴드의 점수를 모두 확인하기
print("10개 폴드의 점수 : {}".format(cv_results))
```

```
>>
평균 : 0.9693916821849783
10개 폴드의 점수 : [0.97777778 0.98888889 0.96111111 0.94444444 0.97777778 0.98333333
 0.95555556 0.98882682 0.97765363 0.93854749]
```

1. 교차 검증 (Cross Validation, CV)

ShuffleSplit

- ShuffleSplit는 반복 횟수에 상관없이 훈련 폴드와 테스트 폴드 크기를 임의로 지정할 수 있음.

```
from sklearn.model_selection import ShuffleSplit

# ShuffleSplit 분할기를 만듭니다.
ss = ShuffleSplit(n_splits=10, train_size=0.5, test_size=0.2, random_state=42)

# 교차검증을 수행합니다.
cv_results = cross_val_score(pipeline, # 파이프라인
                             features, # 특성 행렬
                             target, # 타깃 벡터
                             cv=ss, # 교차 검증 기법
                             scoring="accuracy", # 평가 지표
                             n_jobs=-1) # 모든 CPU 코어 사용
```

1. 교차 검증 (Cross Validation, CV)

ShuffleSplit

```
# 평균을 계산합니다.
print("평균 : {}".format(cv_results.mean()))

# 10개 폴드의 점수를 모두 확인하기
print("10개 검증 점수 개수 : {}".format(len(cv_results)))
```

```
>>
평균 : 0.9630555555555554
10개 검증 점수 개수 : 10
```

1. 교차 검증 (Cross Validation, CV)

RepeatedKFold

- RepeatedKFold를 사용해 교차검증을 반복 실행. n_splits 기본값 5, n_repeats 기본값은 10

```
from sklearn.model_selection import RepeatedKFold

# RepeatedKFold 분할기를 만듭니다.
rfk = RepeatedKFold(n_splits=10, n_repeats=5, random_state=42)

# 교차검증을 수행합니다.
cv_results = cross_val_score(pipeline, # 파이프라인
                             features, # 특성 행렬
                             target, # 타깃 벡터
                             cv=rfk, # 교차 검증 기법
                             scoring="accuracy", # 평가 지표
                             n_jobs=-1) # 모든 CPU 코어 사용
```

1. 교차 검증 (Cross Validation, CV)

RepeatedKFold

```
# 평균을 계산합니다.
print("평균 : {}".format(cv_results.mean()))

# 10개 폴드의 점수를 모두 확인하기
print("10개 검증 점수 개수 : {}".format(len(cv_results)))
>>
평균 : 0.9695065176908755
10개 검증 점수 개수 : 50
```

2. Data Splitting

데이터 분할(Data Splitting)?

- 데이터 분할(Data Splitting)은 기계 학습에서 중요한 단계 중 하나. 데이터를 학습용과 테스트용으로 나누는 것으로, 모델을 훈련하고 테스트하기 위해 데이터셋을 두 부분으로 분할하는 작업.

- 학습용 데이터와 테스트용 데이터 분리:
  - 일반적으로 데이터의 대부분(70-80%)을 학습에 사용하고, 나머지 부분(20-30%)을 모델의 성능 평가를 위한 테스트에 사용.
  - 최근에는 전체 데이터를 **훈련용(Train)**과 **검증/테스트용(Test/Validation)**으로 나눔.

2. Data Splitting

데이터 세트 준비 및 분할

- 일반적으로, 머신 러닝 기반 데이터 분석 진행 시, 특히 지도학습 기반 모델 적용을 할 때는 전체 데이터 세트를 사용하여 한꺼번에 분석하지 않고, 학습용 데이터 세트와 평가용(테스트) 데이터 세트로 분할하여 분석을 진행.
- 분석하고자 하는 목적 및 데이터 세트 특성에 따라 머신 러닝 기법 적용을 위한 훈련 데이터 세트와 테스트 데이터 세트 분할 기준을 판단할 수 있다. 평가 데이터 세트를 사용하는 목적은 '미지의 데이터를 예측하는 능력, 즉 머신 러닝 기반 분석모델의 일반화 능력을 측정하고 성능을 향상하기 위함이다.'라고 할 수 있다.
- 그러나 일반적으로 훈련 데이터와 유사하면서 목적에 적합한 평가 데이터를 별도로 구하기는 쉽지 않으므로, 모델링을 위해 주어진 데이터를 학습용 데이터와 평가용 데이터로 분할해서 머신 러닝 기법을 적용하게 되는 것.
- 해결하고자 하는 이슈와 적용할 기법에 따라 교차검증 필요성을 판단하여, 훈련 데이터 세트와 검증 데이터 세트를 분할하고, 적합한 교차검증 K값을 결정할 수 있다

2. Data Splitting

데이터 세트 준비 및 분할

- 머신 러닝 기반 데이터 분석을 진행함에 있어서, 머신 러닝 기법이 주로 하는 역할은 주어 진 데이터 세트를 학습하여 최적의 모수(파라미터)를 도출하는 것과 이를 바탕으로 특정 설명변수(혹은 특성(Feature))가 주어졌을 때 목적변수 (혹은 반응변수)의 값을 예측하는 과정이라고 할 수 있습니다.
- 그런데 주어진 훈련 데이터 세트에 포함된 데이터는 엄밀히 말하면 '우연에 의해 얻어진 값'이라고 볼 수 있으므로 새로운 목적변수(혹은 반응변수) 등의 값을 예측하기 위해 얻어지는 신규 데이터 세트는 원래의 훈련 데이터 세트와 동일한 데이터가 아닙니다. 그러므로, 훈련데이터에서 나타난 패턴들과 신규 데이터의 패턴이 정확하게 일치할 가능성은 상당히 낮습니다.

2. Data Splitting

데이터 세트 준비 및 분할

- 따라서 머신러닝 기반 모델을 학습하는 데 있어서, 훈련 데이터 세트가 가지고 있는 특성을 너무 많이 반영하게 되면 훈련 데이터 세트의 패턴만 잘 표현하게 되는 '과적합(Overfitting)'이 발생하게 되고, 새로운 데이터가 주어졌을 때 정확하게 예측할 수 있는 '일반화(Generalization)' 능력은 오히려 떨어지게 됩니다.
- 그래서 이러한 현상을 방지하고자 일반적으로 데이터 세트를 훈련용 데이터 세트와 평가용 데이터 세트로 분할하고 훈련용 데이터 세트로 학습한 머신 러닝 모델이 평가용 데이터 세트의 목적변수(혹은 반응변수)를 얼마나 정확하게 예측하는지를 측정하여 이러한 기준치를 모델 성능의 평가 기준으로 삼게 되는 것입니다.

2. Data Splitting

데이터 세트 분할 방법 및 절차

1. 일정 비율로 학습용과 평가용 세트로 데이터 분할

- 데이터의 일부를 훈련 데이터, 나머지를 평가데이터로 분리합니다. 특별한 경우가 아니라면 일반적으로 학습용과 평가용 데이터 각각의 분할은 전체 데이터에서 랜덤하게 특정 비율로 학습용 데이터를 추출하고, 학습용 데이터에 사용되지 않은 나머지 데이터를 평가용 데이터로 취하는 방법을 따릅니다. 이때 훈련 데이터와 평가데이터를 분할하는 비율은 정해진 원칙이 있는 것은 아니나, 모델을 훈련시키는 과정 자체에 더 많은 비중을 할당합니다.
- 일반적으로 훈련 데이터를 60%~80%, 평가데이터를 40%~20% 정도로 할당합니다. 그러나 절대적인 기준은 아니며, 실무 상황에서는 분석의 목적이나 연구수행자의 판단과 경험을 통해 분할 비율을 정하게 됩니다. 다만 데이터 세트 분할 시 중요한 점은 실제 훈련된 모델의 성능은 학습용 데이터 세트 크기가 작아질수록 나빠지게 되므로, 너무 많은 데이터를 평가용 데이터로 분할하는 것은 최종 성능에 오히려 나쁜 영향을 끼칠 수 있다는 점입니다.

2. Data Splitting

데이터 세트 분할 방법 및 절차

2. 학습(훈련) 데이터로부터 머신러닝 모델링 수행

- 머신 러닝 모델링을 수행할 때 여러 가지 기법을 적용하여 기법 간의 성능을 비교할 수도 있고, 동일 기법 내에서도 추정방법을 변경하거나 파라미터를 다양하게 변경하는 등의 과정을 거치게 됩니다. 여기서 모델링 성능에 대한 보다 정교한 검증을 위해 교차검증 (Cross-Validation) 방법을 수행하는 경우도 있습니다.
- 교차검증은 훈련 데이터를 통한 모델링 훈련 시 훈련 데이터 내에서 별도의 검증 데이터(Validation Data)를 할당하여 모델링 및 평가를 반복하는 것으로서, 앞서 1단계에서 분할한 평가 데이터가 학습모델링에 사용되지 않는 반면, 교차검증 데이터는 학습모델링에 사용된다는 점이 다릅니다.

2. Data Splitting

데이터 세트 분할 방법 및 절차

3. 평가 데이터를 이용한 모델 성능 평가

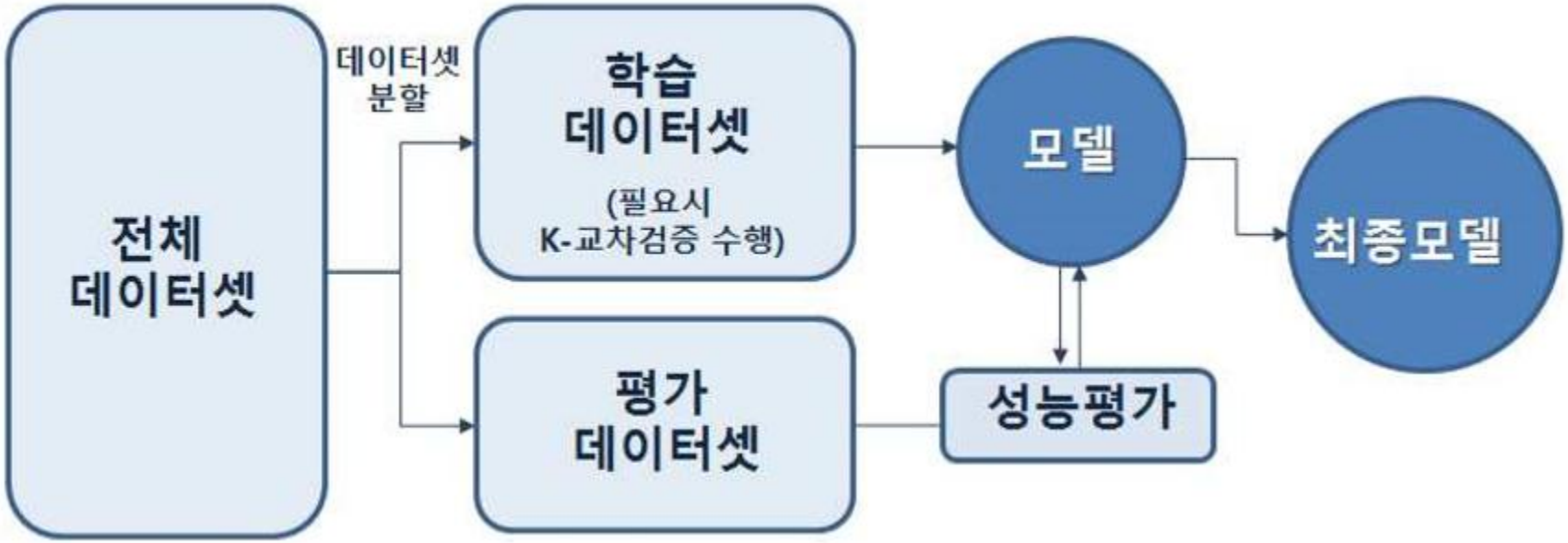
- 2단계에서 만들어진 모델에 평가 데이터를 적용해 성능을 평가합니다. 만일 성능이 만족스럽지 않다면, 앞의 2단계로 다시 돌아가게 됩니다.
- 여기서 평가 데이터는 원래의 전체 데이 터 세트로부터 최초 분리해낸 뒤, 모델링 과정에서 이용되지 않다가 2단계 등을 통해 모델이 만들어지고 난 뒤, 해당 모델의 성능을 평가하기 위해 3단계에서 사용됩니다.
- 이런 점에서 모델링 과정 자체에서 검증의 목적으로 반복적으로 사용되는 검증 데이터와는 그 활용목적이 다르다고 할 수 있습니다.

2. Data Splitting

데이터 세트 분할 방법 및 절차

4. 최종 모델 결과 제출

- 3단계에서 평가 데이터를 이용하여 수행한 성능 평가 및 예측결과가 기준치에 부합하거나, 목적에 적합하다고 판단될 경우 분석 모델링 과정을 종료하고, 최종 분석결과를 제출하게 됩니다.
- 훈련 데이터와 평가 데이터 분할 통한 머신러닝 모델링 절차



2. Data Splitting

iris - K-Fold

```
# 모델 생성 및 학습
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

# 예측 및 평가
pred_train = tree.predict(X_train)
pred_val = tree.predict(X_val)
train_score = accuracy_score(y_train, pred_train)
val_score = accuracy_score(y_val, pred_val)
print("train set의 예측결과: {:.4f}, validation set의 예측결과: {:.4f}".format(train_score, val_score))

# test set 평가
pred_test = tree.predict(X_test)
test_score = accuracy_score(y_test, pred_test)
print("최종평가(test set): {:.4f}".format(test_score))
```

```
>>
train set의 예측결과: 1.0000, validation set의 예측결과: 0.9333
최종평가(test set): 1.0000
```

2. Data Splitting

iris - K-Fold

```
# 모델 생성
tree = DecisionTreeClassifier()
# 모델 학습 - train set
tree.fit(X_train, y_train)
# 예측 및 평가 - train/validation set
pred_train = tree.predict(X_train)
pred_val = tree.predict(X_val)

train_score = accuracy_score(y_train, pred_train)
val_score = accuracy_score(y_val, pred_val)
print("train set의 예측결과: {}, validation set의 예측결과: {}".format(train_score, val_score))
# ==> train set의 예측결과: 1.0, validation set의 예측결과: 0.9166666666666666

# 최종 평가 - test
pred_test = tree.predict(X_test)
test_score = accuracy_score(y_test, pred_test)
print("최종평가(test set): {}".format(test_score))
#==> 최종평가(test set): 0.9666666666666667

>>
```

```
train set의 예측결과: 1.0, validation set의 예측결과: 0.9333333333333333
최종평가(test set): 1.0
```

2. Data Splitting

iris - K-Fold

```
#KFold
#•지정한 개수(K)만큼 분할한다.
from sklearn.model_selection import KFold

# K-Fold K-개수, fold 나눠진 데이터셋 - 5개
kfold = KFold(n_splits=5)
acc_train_list = [] # 평가지표들을 저장할 리스트
acc_test_list = []

ex = kfold.split(X)
print(type(ex))
next(ex) # 반복자의 다음 요소 구하기
>>
<class 'generator'>
(array([ 30,  31,  32,  33,  34,  35,  36,  37,  38,  39,  40,  41,  42,
        43,  44,  45,  46,  47,  48,  49,  50,  51,  52,  53,  54,  55,
        56,  57,  58,  59,  60,  61,  62,  63,  64,  65,  66,  67,  68,
        69,  70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  80,  81,
        82,  83,  84,  85,  86,  87,  88,  89,  90,  91,  92,  93,  94,
        95,  96,  97,  98,  99, 100, 101, 102, 103, 104, 105, 106, 107,
       108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120,
       121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133,
       134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146,
       147, 148, 149]),
 array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]))
```

2. Data Splitting

iris - K-Fold

```
for train_index, test_index in kfold.split(X):
    # index를 이용해 훈련/검증 데이터셋 추출
    X_train, y_train = X[train_index], y[train_index]
    X_test, y_test = X[test_index], y[test_index]

    # 모델생성
    tree = DecisionTreeClassifier()
    # 학습
    tree.fit(X_train, y_train)
    # 검증
    pred_train = tree.predict(X_train)
    pred_test = tree.predict(X_test)

    acc_train = accuracy_score(y_train, pred_train)
    acc_test = accuracy_score(y_test, pred_test)
    acc_train_list.append(acc_train)
    acc_test_list.append(acc_test)

acc_test_list, np.mean(acc_test_list)
>>

([1.0, 0.9666666666666667, 0.9, 0.9333333333333333, 0.7666666666666667],
 np.float64(0.9133333333333333))
```

2. Data Splitting

iris - Stratified K 폴드

- **K-Fold의 문제점**
  - 원 데이터셋의 row 순서대로 분할하기 때문에 불균형 문제가 발생할 수 있다.
- **Stratified K 폴드**
  - 나뉜 fold 들에 label들이 같은(또는 거의 같은) 비율로 구성 되도록 나눈다.

```
from sklearn.model_selection import StratifiedKFold

s_fold = StratifiedKFold(n_splits=3)
acc_train_list = []
acc_test_list = []

ex = s_fold.split(X, y) #label의 class 별 동일한 분포로 나눈다.
type(ex)
>>

generator
```

2. Data Splitting

iris - Stratified K 폴드

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

    # 모델생성
    tree = DecisionTreeClassifier()
    # 학습
    tree.fit(X_train, y_train)
    # 검증
    pred_train = tree.predict(X_train)
    pred_test = tree.predict(X_test)
    acc_train = accuracy_score(y_train, pred_train)
    acc_test = accuracy_score(y_test, pred_test)

    acc_train_list.append(acc_train)
    acc_test_list.append(acc_test)

acc_train_list, np.mean(acc_train_list)
#==> [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]

acc_test_list, np.mean(acc_test_list)
#==> ([0.98, 0.92, 0.96, 0.98, 0.92, 0.96], 0.9533333333333333)
>>
([0.98, 0.92, 1.0], 0.9666666666666667)
```

2. Data Splitting

iris - cross_val_score( )

- cross_val_score( )
  - 데이터셋을 K개로 나누고 K번 반복하면서 평가하는 작업을 처리해 주는 함수
  - 주요매개변수
    - estimator: 학습할 평가모델객체
    - X: feature
    - y: label
    - scoring: 평가지표
    - cv: 나눌 개수 (K)
  - 반환값: array - 각 반복마다의 평가점수

2. Data Splitting

iris - cross_val_score( )

```
from sklearn.model_selection import cross_val_score
tree = DecisionTreeClassifier()

score_list = cross_val_score(tree, X, y, scoring='accuracy', cv=3)

score_list
#==> array([0.98, 0.94, 0.98])
np.mean(score_list)
# ==> 0.9666666666666667
>>

0.9666666666666667
```

3.Quiz

Question 1

다음과 같이 데이터세트를 준비합니다.

```
df = pd.read_csv("datasets/diamonds.csv")
df.head(2)
```

- "price" 변수를 종속변수로 취급해, 종속변수와 나머지 변수를 분리해 봅니다.

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31

3.Quiz

Answer 1

```
import pandas as pd

df = pd.read_csv("datasets/diamonds.csv")
df.head(2)
>>
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31

```
df_X = df.copy()
ser_y = df_X.pop("price")
df_X.head(2)
>>
```

	carat	cut	color	clarity	depth	table	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	3.89	3.84	2.31

```
ser_y.head(2)
>>
0    326
1    326
Name: price, dtype: int64
```

3.Quiz

Question 2

앞선 Q1을 참조하여, diamonds.csv 파일을 훈련 데이터 셋과 테스트 데이터 셋을 8:2로 나눠봅니다.

Answer 2

```
df_tr_X, df_te_X, ser_tr_y, ser_te_y = train_test_split(df_X,
                                                        ser_y,
                                                        train_size = 0.8,
                                                        random_state = 123)
len(df_tr_X), len(df_te_X), len(ser_tr_y), len(ser_te_y)
>>
```

```
(43152, 10788, 43152, 10788)
```

3.Quiz

Question 3

데이터세트를 분할하기 위해 “Xgrp” 변수명으로 별도의 변수가 준비되었을 경우, 필터링을 통해 데이터 세트를 분리해 보자.

	col1	col2	Xgrp
0	0	100	train
1	1	101	train
2	2	102	train
3	3	103	test
4	4	104	test

3.Quiz

Answer 3

```
import pandas as pd

df1 = pd.DataFrame(dict(col1 = range(5),
                        col2 = range(100, 105),
                        Xgrp = ["train"] * 3 + ["test"] * 2))
df1
```

```
>>
        col1    col2    Xgrp
0       0       100     train
1       1       101     train
2       2       102     train
3       3       103     test
4       4       104     test
```

```
df1_train = df1.loc[df1["Xgrp"] == "train", ]
df1_test  = df1.loc[df1["Xgrp"] == "test", ]
len(df1_train), len(df1_test)
```

```
>>
(3, 2)
```

3.Quiz

Question 4

데이터세트가 다음과 같을 때, “ID”변수의 값이 5의 배수가 아닌 행은 “df2_train” 객체에 할당하고 “ID”변수의 값이 5의 배수인 행은 “df2_test” 객체에 할당하시오.

	ID	value
0	1	100
1	2	101
2	3	102
3	4	103
4	5	104

3.Quiz

Answer 4

```
import pandas as pd

df2 = pd.DataFrame(dict(ID = range(1, 6),
                        value = range(100, 105)))
df2
>>
```

	ID	value
0	1	100
1	2	101
2	3	102
3	4	103
4	5	104

```
df2_train = df2.loc[df2["ID"] % 5 != 0, ]
df2_test  = df2.loc[df2["ID"] % 5 == 0, ]
len(df2_train), len(df2_test)

>>
(4, 1)
```

3.Quiz

Question 5

데이터세트가 다음과 같이 변수에 문자와 숫자가 섞여있을 경우, “ID_num”변수를 기준으로 3의 배수 여부에 따라 데이터를 분리하시오.

	ID	value
0	lot_001	100
1	lot_002	101
2	lot_003	102
3	lot_004	103
4	lot_005	104
5	lot_006	105
6	lot_007	106
7	lot_008	107
8	lot_009	108
9	lot_010	109
10	lot_011	110

3.Quiz

Answer 5

```
import pandas as pd

df3 = pd.DataFrame(dict(ID = range(1, 12),
                        value = range(100, 111)))
df3["ID"] = "lot_" + df3["ID"].astype("str").str.zfill(3)
df3

>>
```

	ID	value
0	lot_001	100
1	lot_002	101
2	lot_003	102
3	lot_004	103
4	lot_005	104
5	lot_006	105
6	lot_007	106
7	lot_008	107
8	lot_009	108
9	lot_010	109
10	lot_011	110

3.Quiz

Answer 5

```
df3["ID_1"] = df3["ID"].str.replace("lot_", "")
df3["ID_2"] = df3["ID"].str.replace("[^0-9]", "", regex = True)
df3["ID_3"] = df3["ID"].str.split("_", expand = True)[1]
df3["ID_num"] = df3["ID_1"].astype("int")
df3
>>
```

	ID	value	ID_1	ID_2	ID_3	ID_num
0	lot_001	100	001	001	001	1
1	lot_002	101	002	002	002	2
2	lot_003	102	003	003	003	3
3	lot_004	103	004	004	004	4
4	lot_005	104	005	005	005	5
5	lot_006	105	006	006	006	6
6	lot_007	106	007	007	007	7
7	lot_008	107	008	008	008	8
8	lot_009	108	009	009	009	9
9	lot_010	109	010	010	010	10
10	lot_011	110	011	011	011	11

3.Quiz

Answer 5

```
df3_train = df3.loc[df3["ID_num"] % 3 != 0, ]
df3_test  = df3.loc[df3["ID_num"] % 3 == 0, ]
len(df3_train), len(df3_test)

>>
(8, 3)
```

3.Quiz

Question 6

데이터세트가 다음과 같이 변수에 문자와 숫자가 섞여있을 경우, “ID” 변수의 값이 5의 배수가 아닌 행은 “df4_train” 객체에 할당하고 “ID” 변수의 값이 5의 배수인 행은 “df4_test” 객체에 할당 분리하시오.

	ID	value
0	lotA81023	100
1	lotB0021	101
2	order_329145	102
3	order_12932_Z3	103
4	re-498	104

3.Quiz

Answer 6

```
import pandas as pd

df4 = pd.DataFrame(dict(ID = ["lotA81023", "lotB0021", "order_329145", "order_12932_Z3",
"re-498"],
                        value = range(100, 105)))
df4

>>

            ID          value
0           lotA81023100
1           lotB0021  101
2           order_329145        102
3           order_12932_Z3      103
4           re-498      104
```

3.Quiz

Answer 6

```
df4["ID_1"] = df4["ID"].str.extract("([0-9]{3,})")
df4["ID_num"] = df4["ID_1"].astype("int")
df4
```

```
>>
          ID          value     ID_1      ID_num
0         lotA81023100        81023     81023
1         lotB0021  101       0021      21
2         order_329145        102       329145    329145
3         order_12932_Z3      103       12932     12932
4         re-498    104       498       498
```

```
df4_train = df4.loc[df4["ID_num"] % 5 != 0, ]
df4_test  = df4.loc[df4["ID_num"] % 5 == 0, ]
(len(df4_train), len(df4_test))
```

```
>>
(4, 1)
```

『 3과목 :』
데이터 마트와 데이터 전처리

- Data Mart & Data Preprocessing
- Data Structures
- Data Gathering(Collect, Acquisition), Data Ingestion
- Data Invest & Exploratory Data Analysis, Data Visualization
- Data Cleansing (정제)
- Data Integration (통합)
- Data Reduction (축소)
- Data Transformation (변환)
- Feature Engineering & Data Encoding
- Cross Validation & Data Splitting
- Data Quality Assessment and Model Performance Evaluation
- 『3과목』 Self 점검

Copyright 2022. DaeKyeong all rights reserved

Syllabus

학습목표

- 이 워크샵에서는 Data Quality Assessment and Model Performance Evaluation 에 대해 알 수 있습니다.

눈높이 체크

- Data Quality Assessment and Model Performance Evaluation 을 알고 계신가요?

1.Data Quality Validation

데이터 품질 검증(Data Quality Validation)이란?

- 데이터 품질 검증이란, 수집된 데이터가 정확하고, 일관되며, 완전하고, 유효한지 점검하는 과정.즉, “이 데이터를 믿고 써도 되는가?” 를 확인하는 작업

왜 데이터 품질 검증이 중요한가?

이유	설명
품질이 낮은 데이터	→ 잘못된 분석 결과 유발
머신러닝 성능 저하	→ 모델이 잘못된 패턴 학습
보고서 오류	→ 잘못된 비즈니스 의사결정

1.Data Quality Validation

주요 데이터 품질 검증 항목

- 데이터 분석의 목적을 달성하고, 최종 사용자의 기대를 만족하게 하기 위해 데이터가 확보하고 있어야 할 성질을 말한다. 데이터 품질을 보장하기 위해 만족하여야 할 요소로는 정확성(accuracy), 완전성(completeness), 적시성(timeliness), 일관성(consistency)이 존재한다(최상균, 전순천, 2013).

항목	설명	예시
정확성 (Accuracy)	실제 값과의 일치 여부	"성별"에 123 입력
완전성 (Completeness)	누락 없이 값이 채워졌는지	주소가 빈칸이면
일관성 (Consistency)	동일한 데이터가 동일하게 표현되는지	"남" / "남자" 혼재
유효성 (Validity)	정의된 포맷, 범위 안에 있는지	날짜: 2023-13-40
중복성 (Uniqueness)	중복된 레코드가 있는지	동일 고객이 두 번 입력
적시성 (Timeliness)	최신 데이터인지	2010년 고객 정보

1.Data Quality Validation

데이터 품질 검증을 위한 만족 요건

- 데이터 품질을 검증하기 위해서는 데이터 품질 요소들이 다음과 같은 요건을 만족해야 한다

1. 정확성

– 저장된 데이터는 대상을 올바로 나타내는 값을 가져야 함

2. 완전성

– 저장된 데이터는 대상을 올바로 나타내는 값을 가져야 함

3. 적시성

– 저장된 데이터는 대상을 올바로 나타내는 값을 가져야 함

4. 일관성

– 저장된 데이터는 대상을 올바로 나타내는 값을 가져야 함

1.Data Quality Validation

데이터 무결성

- 데이터 무결성(data integrity)이란 다수의 사용자가 데이터베이스에 접근하여 적재, 삽입, 삭제, 수정 등의 작업을 수행할 때 데이터가 불일치하지 않는 특성을 말한다.(___(2015.12.8.). 데이터 무결성 (Data Integrity). http://bongbonge.tistory.com/100에서 2016.7. 8. 검색). 데이터 무결성은 데이터 품질을 만족하는 데이터가 분석 목적을 달성하기 위해 공유되어 사용될 때 지켜져야 할 성질이다.

1.Data Quality Validation

데이터 무결성

- 데이터 무결성을 확보하기 위해서는 다음과 같은 필요 요건을 만족해야 한다.

1. 개체 무결성
– 기본 키(primary key)는 반드시 값을 가지며 그 값은 유일해야 함
2. 참조 무결성
– 외래 키(foreign key)값은 참조하는 테이블의 기본 키값 혹은 빈값 중 하나를 가져야 함
3. 속성 무결성
– 속성값은 지정된 데이터 형식을 만족하는 값을 가져야 함
4. 키 무결성
– 하나의 테이블에 적어도 하나의 키가 존재해야 함
5. 도메인 무결성
– 속성값은 미리 정의된 도메인 범위 안의 값을 가져야 함
6. 사용자 정의 무결성
– 모든 데이터는 업무 규칙(business rule)을 준수해야 함

1.Data Quality Validation

실제 수행 방법 예시 (Pandas 기반)

```
import pandas as pd

# 예시 데이터프레임
df = pd.DataFrame({
    '고객명': ['홍길동', '김철수', None],
    '생년': [1985, 2020, 1890],
    '이메일': ['hong@gmail.com', 'not-an-email', 'kim@naver.com']
})

# 평가 예시
print("결측치 확인:\n", df.isnull().sum())
print("생년 유효성 확인:\n", df[(df['생년'] < 1900) | (df['생년'] > 2024)])
print("이메일 유효성 확인:\n", df[~df['이메일'].str.contains('@')])
```

```
결측치 확인:
 고객명    1
생년     0
이메일    0
dtype: int64
생년 유효성 확인:
     고객명    생년            이메일
2  None  1890  kim@naver.com
이메일 유효성 확인:
    고객명    생년           이메일
1  김철수  2020  not-an-email
```

개인정보보호

- **데이터 비식별화(de-identification)의 목적과 개념**

◦ **목적**

– 데이터 비식별화의 목적은 데이터에 포함된 개인정보를 삭제하거나 다른 정보로 대체하여 데이터 내에서 특정 개인을 식별하지 못하게 하기 위함이다(양현철, 신신애, 김진철, 2014).

1.Data Quality Validation

개인정보보호

◦ 개념

– 개인정보란 이름, 주민등록번호에서 DNA에 이르기까지 그것을 이용해 특정 개인을 식별할 가능성을 내포한 데이터를 말한다. 데이터 비식별화는 개인을 식별할 수 있는 잠재성을 가진 데이터를 식별할 수 없거나 식별하기 어려운 데이터로 가공하는 일련의 과정을 일컫는다. 비식별화는 SNS와 같은 개인정보를 포함하고 있는 데이터에 대한 분석이 증가하면서 그 중요성이 대두하고 있다. 미국 연방거래위원회(2012)는 보고서에서 다음과 같은 세 가지 비식별화 조치사항을 명시하였다.

- 소비자, 컴퓨터 또는 다른 장치와 결합할 수 있는 개인정보는 반드시 비식별화되어야 함
- 공개된 정보에 대해서는 재식별화를 시도하지 않아야 함
- 타 기업 등에 비식별화된 데이터 제공 시 데이터를 재식별화하지 않도록 계약상 명문화하도록 함

1.Data Quality Validation

개인정보보호

- 대표적인 비식별화 기법으로는 다음과 같은 것들이 있다

◦ 가명처리(pseudonymisation)

– 식별 가능한 변수값을 다른 값으로 대체

– 예) 김치국, 38세, 수원 거주 -> 홍길동, 38세, 수원 거주

◦ 총계처리(aggregation)

– 개인정보 보호를 위해 데이터를 총합하거나 평균을 사용

– 예) A 직원 연봉 4,500만, B 직원 연봉 5,200만, C 직원 연봉 4,600만

– -> 평균 연봉 4,766만

◦ 데이터 값 제거(data reduction)

– 개인 식별에 유의한 변수값 제거

– 예) 김치국, 38세, 수원 거주 -> 38세 남, 수원 거주

1.Data Quality Validation

개인정보보호

- 대표적인 비식별화 기법으로는 다음과 같은 것들이 있다

◦ 범주화(data suppression)

– 데이터값을 범주화하여 명확한 값을 대체

– 예) 김치국, 38세, 수원 거주 -> 김치국, 30대, 경기도 거주

◦ 데이터 마스킹(data masking)

– 개인 식별에 유의한 변수값을 보이지 않도록 처리

– 예) 김치국, 38세, 수원 거주 -> 김**, 38세, 수원 거주

2. 안전성 확보 조치 기준

개요

- AI 개발자 및 서비스 제공자는 정당한 이익과 정보주체 권리 사이의 명백한 우선관계를 확인하기 어려운 경우, 정보주체 권리에 대한 제약 또는 침해를 예방·방지하기 위한 안전성 확보 조치를 충분히 시행하는 것이 바람직함

학습데이터 수집 출처 검증·관리

- 사례 : 데이터 출처 검증을 위한 고려사항

1. 불법 복제물, 아동 성착취물 등 위법한 데이터가 거래되거나 거래될 가능성이 높은 도메인 (예: 딥웹, 다크웹)으로부터 학습데이터 수집 금지
2. 개인정보가 집적되어 있을 개연성이 높은 웹사이트(예: 개인정보 색인·거래 사이트) 배제
3. 로봇배제표준(robots.txt) 준수
4. 저작권, 디자인권 등 지식재산권 존중

2. 안전성 확보 조치 기준

학습데이터 수집 출처 검증·관리

● 사례 : 개인 식별자 삭제 또는 비식별화 사례

발화데이터 비식별화 예시

[혈액형]
나의 혈액형은 [BLOOD_TYPE_1]. 어떤 혈액형과의 성향이 가장 잘 맞을까?
[병명]
나 [CONDITION_1] 어제 쉬었어.
[CONDITION_2] 좋은 음식이 뭐가 있니?
[복용약/량]
나 어제 [DRUG_1] 먹었는데, 효과 좋더라. 너도 이걸로 먹어. 하루에 [DOSE_
[항정신성 의약품]
나 아는 형이 [DRUG_2] 줘서 먹었는데, 쩔더라
[외상/상처]
나 [CONDITION_3] 있는데, 어떻게 극복할 수 있을까?
[의료 행위]
어제 [MEDICAL_PROCESS_1], [CONDITION_4]이 심하다는 데, 살 빼려면 어
어제 [MEDICAL_PROCESS_2], 무릎 연골이 부숴져서 나 이제 못 걸을 수 있다
어제 [OCCUPATION_1] 쌤이 [CONDITION_5] [MEDICAL_PROCESS_3]? 같은
[전문적/통계적 지식]
어제 [OCCUPATION_1] 쌤한테 들었는데, [CONDITION_6] 치료율이 [STATIS
[은행_계좌]
우리 은행 [BANK_ACCOUNT_1] [MONEY_1] 보내줘
하나 [CREDIT_CARD_1] 주인 이름이 어떻게 되니?
[신용카드]
[ORGANIZATION_1] [CREDIT_CARD_2] [CREDIT_CARD_EXPIRATION_1] [
[금융_거래정보]

발화데이터 비식별화 예시(Wrtn)

■고유식별정보
- 주민등록번호
- 운전면허번호
- 여권번호

■민감정보
- 종교
- 정치이념
- 노동조합명
- 질병명
- 범죄 경력 자료

■기타개인정보
- 계좌번호
- 신용카드정보 등

비식별화 정보 유형 예시

2. 안전성 확보 조치 기준

미세조정을 통한 안전장치 추가

- 학습데이터에는 편향적이거나 부정확한 정보, 민감한 사적정보가 포함될 수 있어 사전 정제 과정이 수반되는 경우가 많으나, 이로써 모든 위험이 예방되는 것은 아니기 때문에 미세조정(Supervised Fine-Tuning, SFT), 사람 피드백 기반 강화 학습 (Reinforcement Learning with Human Feedback, RLHF) 등의 미세조정 기법 적용을 고려할 수 있음

❖ 최근 RLHF에 소요되는 막대한 비용(사람 레이블러 동원에 필요한 비용)과 사람의 주관적 편향성, 기술적 복잡성 등에 대한 한계를 보완하기 위하여 RLHF를 대체하는 방법론(예: Direct Preference Optimization, DPO)등이 꾸준히 연구 중으로, 향후 이러한 기술적 발전을 고려하여 안전장치를 확보하는 것이 바람직함

2. 안전성 확보 조치 기준

미세조정 종류

- 파라미터 효율 미세조정(Parameter Efficient Fine-Tuning(PEFT))
  - 사전학습된 모델 파라미터(매개변수)를 동결하고 소수의 파라미터를 의도된 용도에 맞게 미세조정하는 것으로 학습 비용과 시간을 최소화하는 방법
- 지도학습 기반 미세조정(Supervised Fine-Tuning(SFT))
  - 비지도학습으로 만들어진 생성AI를 지도학습적으로 미세조정하는 과정으로, 바람직한 답변을 생성하도록 미리 정제되거나 레이블링된 데이터를 추가 학습
  - ※ (예) 개인의 사생활을 묻는 프롬프트에 대하여 답변을 거부하는 내용의 답안을 학습시킴

2. 안전성 확보 조치 기준

미세조정 종류

- 사람 피드백 기반 강화학습(Reinforcement Learning with Human Feedback(RLHF))
  - 보상모델 생성(Reward Model Creation) : AI 모델이 생성한 출력물에 사람(라벨러)이 점수 또는 순위를 부여하고, 이를 토대로 보상모델을 훈련
  - ※ (예) 개인의 사생활을 묻는 프롬프트에 대하여 사생활이 포함된 답변에는 (-1)의 보상을, 회피하는 답변에는 (+1)의 보상을 제공
  - 정책 최적화(Policy Optimization): 보상모델을 사용하여 AI 모델의 정책을 최적화하는 단계로, 주로 정책 그라디언트 강화학습 알고리즘인 PPO(Proximal Policy Optimization)을 활용하여 미세조정

2. 안전성 확보 조치 기준

미세조정을 통한 안전장치 추가

- 다양한 미세조정 방식 비교

다양한 미세조정 기법	파라미터 효율 미세조정 (PEFT)	지도학습 기반 미세조정 (SFT)	사람 피드백 기반 강화학습 (RLHF)
	Base LLM Tunable 미세조정 답변생성	Base LLM Tunable 미세조정 답변생성	SFT 등 LLM Tunable 미세조정 답변생성
방법	소수의 파라미터 조정(~0.01%)	이상적 답변 생성을 위한 추가학습	보상모델 생성 및 정책 최적화
학습데이터	X백개 이상	X만개 ~ XX만개	X만개 ~ XX만개
학습 비용	비교적 저렴	비교적 비쌈	비교적 비쌈
학습 시간	Minutes	Days	Days

3. Metric Evaluation

모델 성능 평가(Metric Evaluation)?

- 머신러닝과 데이터 분석에서 **모델 성능 평가(Metric Evaluation)**는 결과의 신뢰성과 품질을 판단하는 핵심 과정.
- 그중 Accuracy는 가장 기본적이고 직관적인 지표이며, 다양한 성능 지표와 함께 사용.

성능 측정 지표 (Performance Metrics)란?

- 성능 지표는 모델이 예측을 얼마나 잘 수행했는지를 수치적으로 표현한 값

3. Metric Evaluation

성능 측정 개요

- 객관적인 성능 측정의 중요성
  - 모델 선택할 때 중요
  - 현장 설치 여부 결정할 때 중요
- 일반화generalization 능력
  - 학습에 사용하지 않았던 새로운 데이터에 대한 성능
  - 가장 확실한 방법은 실제 현장에 설치하고 성능 측정 → 비용 때문에 실제 적용 어려움
  - 주어진 데이터를 분할하여 사용하는 지혜 필요
  - O/X의 문제가 아닌, 얼마나 일치하는가?

3. Metric Evaluation

성능 측정 개요

- 여러가지 평가 방법들
  - Confusion Matrix 기반
  - Accuracy
  - Precision
  - Recall
  - F1 score
  - AUC (Area Under the Curve) & ROC (Receiver Operating Characteristic)

3. Metric Evaluation

혼동 행렬과 성능 측정 기준

- 혼동 행렬(Confusion Matrix)

◦ 부류 별로 옳은 분류와 틀린 분류의 개수를 기록한 행렬

◦ 이진 분류에서 긍정positive과 부정negative

◦ 검출하고자 하는 것이 긍정(환자가 긍정이고 정상인이 부정, 불량품이 긍정이고 정상이 부정)

◦ 참 긍정(TP), 거짓 부정(FN), 거짓 긍정(FP), 참 부정(TN)의 네 경우

	실제 Positive	실제 Negative
예측 Positive	**TP** (True Positive)	**FP** (False Positive)
예측 Negative	**FN** (False Negative)	**TN** (True Negative)

3. Metric Evaluation

혼동 행렬과 성능 측정 기준

- 위의 혼동 행렬로부터 계산될 수 있는 주요 평가 지표(Metric)는 대표적으로 정확도 (accuracy), 오차 비율(error rate) = 1-정확도, 민감도(sensitivity 혹은 재현율 : Recall, Hit Ratio, TP Rate 등으로도 부름), 특이도(specificity), 거짓 긍정률(FP Rate), 정밀도 (precision) 등이 있으며, 이 중에서 정확도, 민감도, 정밀도 등은 특히 많이 사용되는 지표이다.
- 또한, 민감도와 정밀도를 조합한 F-Measure(혹은 F1-Score) 및 분석 모델의 예측값 과 실제값이 정확히 일치하는 정도를 계량화한 카파 통계(Kappa Statistic) 등이 정의되어 있다.

3. Metric Evaluation

주요 성능 지표 정리

지표	수식	설명
정확도 (Accuracy)	(TP + TN) / (TP + TN + FP + FN)	전체 샘플 중 정답 비율
정밀도 (Precision, PPV)	TP / (TP + FP)	Positive 예측 중 실제 Positive 비율
재현율 / 민감도 (Recall, Sensitivity, TPR)	TP / (TP + FN)	실제 Positive 중 맞춘 비율
특이도 (Specificity, TNR)	TN / (TN + FP)	실제 Negative 중 맞춘 비율
NPV (Negative Predictive Value)	TN / (TN + FN)	Negative 예측 중 실제 Negative 비율
F1-Score (F-Measure)	2 × (Precision × Recall) / (Precision + Recall)	정밀도와 재현율의 조화 평균
Fall-out (FPR, 위양성률)	FP / (FP + TN)	실제 Negative 중 잘못 Positive로 예측한 비율
Balanced Accuracy	(TPR + TNR) / 2	민감도와 특이도의 평균

3. Metric Evaluation

학습 데이터셋, 테스트 데이터셋

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df_train = pd.read_csv("datasets/titanic_train.csv")
df_test = pd.read_csv("datasets/titanic_test.csv")
df_train.head(5)

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# 특성과 타깃 분리
X_train = df_train.drop(columns=['survived'])
y_train = df_train['survived']
X_test = df_test.drop(columns=['survived'])
y_test = df_test['survived’]

# 문자열 데이터를 원-핫 인코딩하기 위한 열 선택
categorical_features = X_train.select_dtypes(include=['object']).columns
numeric_features = X_train.select_dtypes(exclude=['object']).columns
```

3. Metric Evaluation

모델링

```
# 전처리 파이프라인 생성
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline([
            ('imputer', SimpleImputer(strategy='mean')),
            ('scaler', StandardScaler())
        ]), numeric_features),
        ('cat', Pipeline([
            ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
            ('onehot', OneHotEncoder(handle_unknown='ignore'))
        ]), categorical_features)
    ])
# Logistic Regression 분류 모델링

from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve, roc_auc_score
import matplotlib.pyplot as plt


# 로지스틱 회귀 모델을 포함한 파이프라인 생성
lr = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(random_state=0))
])
```

3. Metric Evaluation

모델 학습

```
# 모델 학습
lr.fit(X_train, y_train)

# 테스트 데이터셋에 대한 예측
y_pred = lr.predict(X_test)
y_pred_probability = lr.predict_proba(X_test)[:, 1]
```

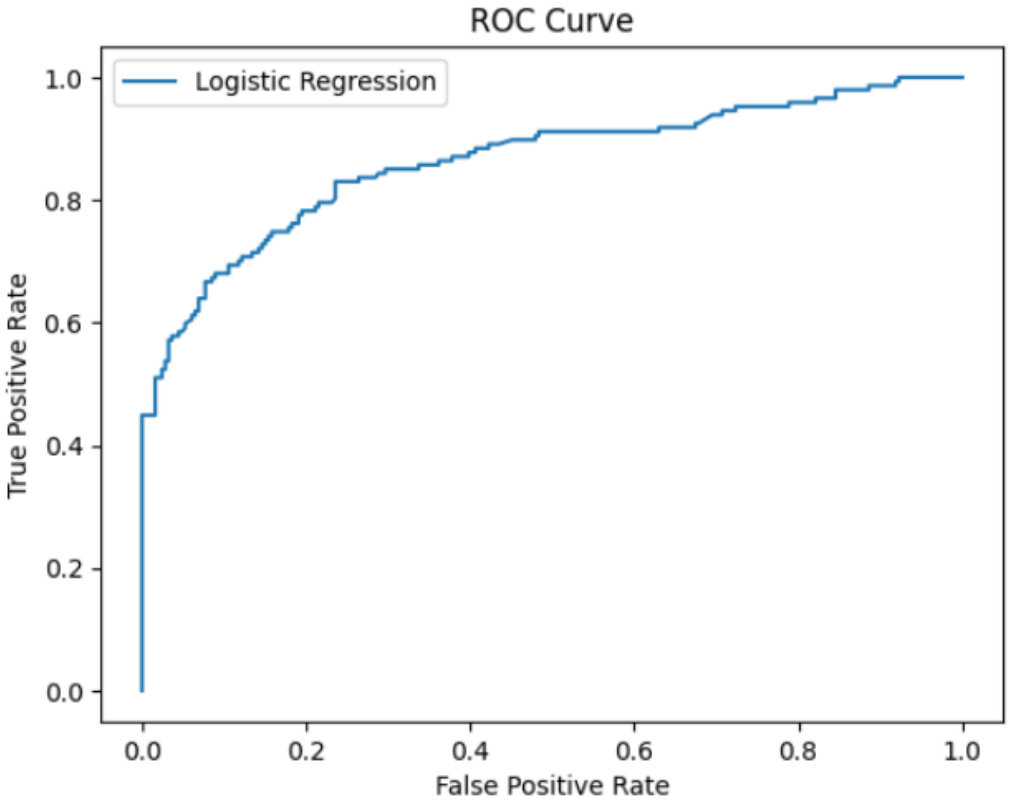
분류 모델 평가

```
# 평가 지표 출력
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
print(f'Precision: {precision_score(y_test, y_pred)}')
print(f'Recall: {recall_score(y_test, y_pred)}')
print(f'F1 Score: {f1_score(y_test, y_pred)}')
print(f'ROC AUC Score: {roc_auc_score(y_test, y_pred_probability)}')
```

3. Metric Evaluation

ROC Curve 그리기

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probability)
plt.plot(fpr, tpr, label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



3. Metric Evaluation

분류 모델 평가

```
from sklearn.metrics import confusion_matrix

# Confusion Matrix를 출력합니다.
confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(confmat)
```

```
[[216  30]
 [ 44 103]]
```

accuracy: 0.811704834605598
Precision : 0.756
Recall : 0.673
F1 : 0.712

(216+103) / (216+103+44+30)

실제값(Reference)

예측값(Prediction)	Y	N
Y	True Positive(TP)	False Positive(FP)
N	False Negative(FN)	True Negative(TN)

Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$	전체 예측에서(예측이 Y이든 N이든 무관하게) 옳은 예측의 비율

4. 모델 성능 평가 예

실습 모듈 불러오기

Scikit Learn 제공 Toy Data를 사용한 실습 모듈 불러오기

데이터 셋과 의사결정 트리 기반 분류기 관련 클래스 불러오기

```
from sklearn import datasets
from sklearn.tree import DecisionTreeClassifier
```

테스트를 위해 사용하는 데이터 셋의 분리 방법 관련 모듈 불러오기

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
```

성능 평가 관련 모듈 불러오기

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import mean_squared_error
```

4. 모델 성능 평가 예

데이터 셋 구성

데이터 셋 준비

```
data = datasets.load_breast_cancer()
X = data.data
y = data.target
```

홀드 아웃

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

4. 모델 성능 평가 예

결정 트리 분류 모델 생성

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

clf

```
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
clf
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

테스트 데이터에 대한 예측 값

y_pred = clf.predict(X_test)

4. 모델 성능 평가 예

모델 성능 평가

```
print("Confusion Matrix")
print(confusion_matrix(y_test ,y_pred))
```

```
Confusion Matrix
[[43  1]
 [ 3 67]]
```

```
print("Accuracy")
print(accuracy_score(y_test, y_pred, normalize = True))
```

```
Accuracy
0.9649122807017544
```

모델 성능 평가

```
print("Classification Report")
print(classification_report(y_test, y_pred))
```

```
Classification Report
             precision    recall  f1-score   support

          0       0.93      0.98      0.96        44
          1       0.99      0.96      0.97        70

avg / total       0.97      0.96      0.97       114
```

```
print("AUC")
print(roc_auc_score(y_test, y_pred))
```

```
AUC 0.9672077922077924
```

```
print("Mean Squared Error")
print(mean_squared_error(y_test, y_pred))
```

```
Mean Squared Error
0.03508771929824561
```

4. 모델 성능 평가 예

K 홀드 아웃 검증을 통한 실습

```
skf = StratifiedKFold(n_splits=10)
skf.get_n_splits(X, y)
print(skf)
-----

for train_index, test_index in skf.split(X, y):
    print("Train set :", train_index)
    print("Test set:", test_index)
-----
Train set : [ 25  26  27  28  29  30  31  32  33  34  35  36  38  39  40  41  42  43
  44  45  47  53  54  56  57  62  64  65  70  72  73  75  77  78  82  83
  85  86  87  91  94  95  99 100 102 103 104 105 106 107 108 109 110 111
 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129
 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147
 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165.
....
```

4. 모델 성능 평가 예

K 홀드 아웃 검증을 통한 실습

성능 평가

```
clf = DecisionTreeClassifier()
scores = cross_val_score(clf, X, y, cv=skf)
print("K Fold Cross Validation Score")
print(scores)
print("Average Accuracy")
print(scores.mean())
-----
K Fold Cross Validation Score
[0.89655172 0.87931034 0.92982456 0.87719298 0.92982456 0.89473684
 0.87719298 0.94642857 0.92857143 0.96428571]
Average Accuracy
0.9123919713075791
```

4. 모델 성능 평가 예

K 홀드 아웃 검증을 통한 실습

데이터 셋 구성 shuffle

```
skf_sh = StratifiedKFold(n_splits=10, shuffle=True)
skf_sh.get_n_splits(X, y)
print(skf_sh)
-----


for train_index, test_index in skf_sh.split(X, y):
    print("Train set:", train_index)
    print("Test set:", test_index)
-----
Train set: [  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
  18  19  20  21  22  25  26  27  29  30  31  32  33  34  35  36  37  38
  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57
  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  75  76  78
  80  81  82  83  85  86  87  88  89  90  92  93  95  96  99 100 101 102
 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
…..
```

4. 모델 성능 평가 예

K 홀드 아웃 검증을 통한 실습

```
clf = DecisionTreeClassifier()
scores = cross_val_score(clf, X, y, cv=skf_sh)
print("K Fold Cross Validation Score")
print(scores)
print("Average Accuracy")
print(scores.mean())
-----
K Fold Cross Validation Score
[0.93103448 0.96551724 0.87719298 0.9122807  0.96491228 0.94736842
 0.9122807  0.92857143 0.92857143 0.91071429]
Average Accuracy
0.9278443954714373
```

평균 Accurac가 0.93으로 약간 상승한 것을 확인할 수 있습니다.

『 3과목 :』
데이터 마트와 데이터 전처리

- Data Mart & Data Preprocessing
- Data Structures
- Data Gathering(Collect, Acquisition), Data Ingestion
- Data Invest & Exploratory Data Analysis, Data Visualization
- Data Cleansing (정제)
- Data Integration (통합)
- Data Reduction (축소)
- Data Transformation (변환)
- Feature Engineering & Data Encoding
- Cross Validation & Data Splitting
- Data Quality Assessment and Model Performance Evaluation

- **『3과목』 Self 점검**

Copyright 2022. DaeKyeong all rights reserved

Syllabus

학습목표

- 이 워크샵에서는 Data Analysis Pipeline에 대해 알 수 있습니다.

눈높이 체크

- Data Analysis Pipeline을 알고 계신가요?

1. 데이터 분석 파이프라인

캘리포니아의 주택 가격 모델 만들어 보기

- 부동산 회사에 최근에 고용된 데이터 과학자인 것처럼 가장하여 예제 프로젝트를 처음부터 끝까지 진행합니다. 거쳐야 할 주요 단계는 다음과 같습니다.

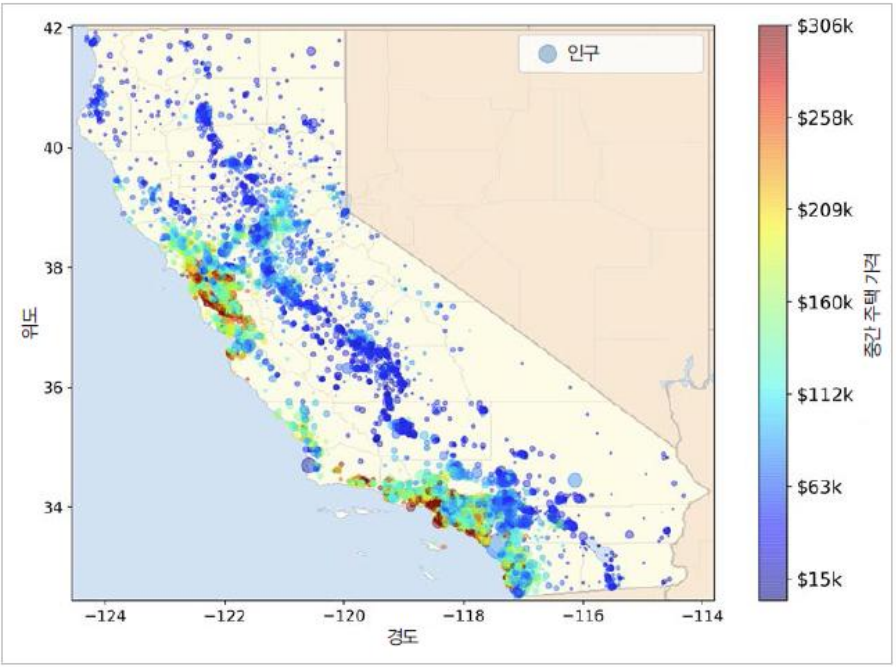
- 비즈니스 목표를 정의합니다.
- 데이터를 가져옵니다.
- 데이터를 발견하고 시각화하여 통찰력을 얻습니다.
- 기계 학습 알고리즘을 위한 데이터를 준비합니다.
- 모델을 선택하여 학습시킵니다.
- ~~모델을 미세 조정합니다.~~
- ~~솔루션을 제시합니다.~~
- ~~시스템을 시작, 모니터링 및 유지 관리합니다.~~

1. 데이터 분석 파이프라인

Problem와 question

- 이 워크샵에서 우리는 StatLib 저장소(The original dataset appeared in R. Kelley Pace and Ronald Barry, "Sparse Spatial Autoregressions," Statistics & Probability Letters 33, no. 3 (1997): 291–297.)에서 캘리포니아 주택 가격 데이터 세트를 선택했습니다.

- 이 데이터 세트는 1990년 캘리포니아 인구 조사 데이터를 기반으로 했습니다. 그것은 정확히 최근의 것은 아니지만(당시 Bay Area에서 여전히 좋은 집을 살 수 있었습니다), 학습을 위한 많은 자질을 가지고 있으므로 우리는 그것이 최근의 데이터인 척 할 것입니다.
- 또한 범주 속성을 추가하고 교육 목적으로 몇 가지 기능을 제거했습니다.



1. 데이터 분석 파이프라인

Problem와 question

- 다음 질문은 현재 솔루션이 어떻게 생겼는지(있는 경우)입니다. 문제 해결 방법에 대한 통찰력뿐만 아니라 참조 성능을 제공하는 경우가 많습니다. 상사는 현재 지역 주택 가격이 전문가에 의해 수동으로 추정된다고 대답합니다. 팀은 지역에 대한 최신 정보를 수집하고 중간 주택 가격을 얻을 수 없는 경우 복잡한 규칙을 사용하여 추정합니다.이것은 비용이 많이 들고 시간이 많이 걸리며 추정치가 크지 않습니다.
- 실제 중간 주택 가격을 알아내는 경우에 그들은 종종 그들의 추정치가 20% 이상 빗나갔다는 것을 깨닫습니다. 이것이 회사가 해당 지역에 대한 다른 데이터가 주어지면 해당 지역의 중간 주택 가격을 예측하는 모델을 훈련하는 것이 유용할 것이라고 생각하는 이유입니다. 인구 조사 데이터는 수천 개 지역의 중간 주택 가격과 기타 데이터를 포함하기 때문에 이러한 목적을 위해 활용하기에 훌륭한 데이터 세트처럼 보입니다.자, 이 모든 정보를 가지고 이제 시스템 설계를 시작할 준비가 되었습니다.

1. 데이터 분석 파이프라인

문제 정의

- 먼저 문제의 틀을 잡아야 합니다.
- 지도, 비지도 또는 강화 학습입니까? 분류 작업입니까, 회귀 작업입니까, 아니면 다른 것입니까? 일괄 학습 또는 온라인 학습 기술을 사용해야 합니까?
- 계속 읽기 전에 잠시 멈추고 이 질문에 스스로 답해 보십시오.
- 답을 찾으셨나요?
- 레이블이 지정된 교육 예제가 제공되기 때문에 분명히 일반적인 지도 학습 작업입니다(각 인스턴스에는 예상 출력, 즉 해당 지역의 중간 주택 가격이 제공됨). 또한 값을 예측해야 하므로 일반적인 회귀 작업이기도 합니다. 보다 구체적으로 말하면 시스템이 예측을 위해 여러 기능을 사용하기 때문에 다중 회귀 문제입니다(지역 인구, 중위 소득 등을 사용함). 또한 각 지역에 대해 단일 값만 예측하려고 하기 때문에 일변량 회귀 문제입니다. 지구당 여러 값을 예측하려고 하면 다변수 회귀 문제가 됩니다. 마지막으로, 시스템에 들어오는 데이터의 지속적인 흐름이 없고, 빠르게 변화하는 데이터에 적응할 필요가 없으며, 데이터가 메모리에 들어갈 만큼 작기 때문에 일반 배치 학습이 잘 될 것입니다.

성능 측정 지표 선택

- 다음 단계는 성과 측정을 선택하는 것입니다. 회귀 문제에 대한 일반적인 성능 측정은 RMSE(평균제곱근 오차)입니다. 시스템이 일반적으로 예측에서 얼마나 많은 오류를 범하는지에 대한 아이디어를 제공하며 큰 오류에 대해 더 높은 가중치를 부여합니다. 아래, 수학식은 RMSE를 계산하는 수학 공식을 보여줍니다.

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

- MSE는 일반적으로 회귀 작업에 선호되는 성능 측정이지만 일부 상황에서는 다른 기능을 사용하는 것을 선호할 수 있습니다. 예를 들어, 많은 이상치 구역이 있다고 가정합니다. 이 경우 평균 절대 오차(평균 절대 편차라고도 함) 사용을 고려할 수 있습니다.

$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right|$$

1. 데이터 분석 파이프라인

가정 확인

- 마지막으로 (귀하 또는 다른 사람이) 지금까지 가정한 사항을 나열하고 확인하는 것이 좋습니다. 이것은 초기에 심각한 문제를 포착할 수 있습니다.
- 예를 들어, 귀하의 시스템이 출력하는 지역 가격은 다운스트림 머신 러닝 시스템에 공급될 것이며 우리는 이러한 가격이 그대로 사용될 것이라고 가정합니다. 그러나 다운스트림 시스템이 실제로 가격을 범주(예: "저렴함", "중간" 또는 "고가")로 변환한 다음 가격 자체 대신 해당 범주를 사용한다면 어떻게 될까요? 이 경우 가격을 완벽하게 맞추는 것은 전혀 중요하지 않습니다.
- 시스템이 카테고리를 올바르게 지정하기만 하면 됩니다. 그렇다면 문제는 회귀 작업이 아니라 분류 작업으로 프레임이 지정되어야 합니다. 몇 달 동안 회귀 시스템에서 작업한 후에 이것을 찾고 싶지 않을 것입니다.

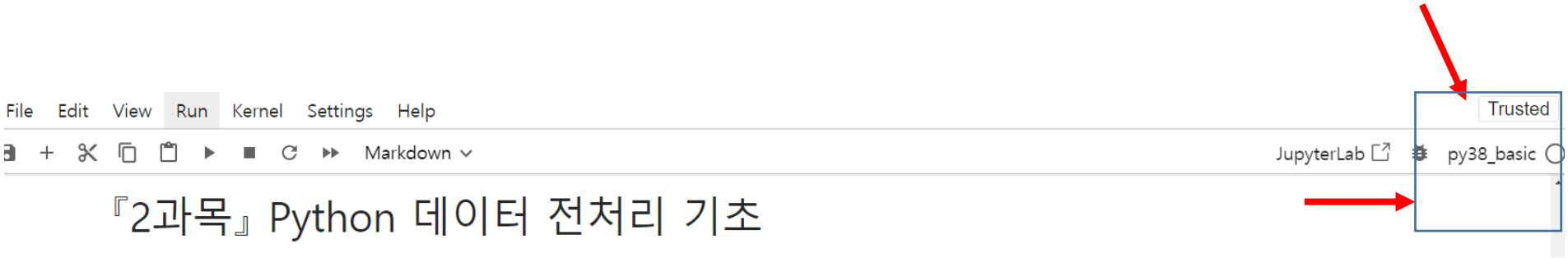
1. 데이터 분석 파이프라인

작업환경 만들기

- 첫째, 작업 공간에 Untitled.ipynb라는 새 노트북 파일을 만듭니다. 둘째, 이 노트북을 실행하기 위해 Jupyter Python 커널을 시작합니다. 셋째, 이 노트북을 새 탭에서 엽니다. Untitled를 클릭하고 새 이름을 입력합니다. 프로토타입1.ipynb

>> 작업환경 만들기

```
(py3_10_basic) PS C:\DEV> jupyter notebook
[I 2024-05-29 10:44:08.583 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-05-29 10:44:08.600 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-05-29 10:44:08.600 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-05-29 10:44:08.616 ServerApp] notebook | extension was successfully linked.
[I 2024-05-29 10:44:09.052 ServerApp] notebook_shim | extension was successfully linked.
```



기타 라이브러리 설치 확인

```
import sys
import numpy as np
print("numpy 버전 : {}".format(np.__version__))
print("python 버전 : {}".format(sys.version))
import pandas as pd
print("pandas 버전 : {}".format(pd.__version__))
import matplotlib
print("matplotlib 버전 : {}".format(matplotlib.__version__))
import scipy as sp
print("scipy 버전 : {}".format(sp.__version__))
import IPython
print("IPython 버전 : {}".format(IPython.__version__))
import sklearn
print("sklearn : {}".format(sklearn.__version__))
```

```
numpy 버전 : 1.24.3
python 버전 : 3.8.19 (default, Mar 20 2024, 19:55:45) [MSC v.1916 64 bit (AMD64)]
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[3], line 5
      3 print("numpy 버전 : {}".format(np.__version__))
      4 print("python 버전 : {}".format(sys.version))
----> 5 import pandas as pd
      6 print("pandas 버전 : {}".format(pd.__version__))
      7 import matplotlib

ModuleNotFoundError: No module named 'pandas'
```

기타 라이브러리 설치

(py3_10_basic) PS C:\DEV>

(py3_10_basic) PS C:\DEV> pip install --user numpy pandas matplotlib scipy

```
Collecting pandas
  Downloading pandas-1.5.3-cp38-cp38-win_amd64.whl (11.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.0/11.0 MB 8.3 MB/s eta 0:00:00
Collecting matplotlib
  Downloading matplotlib-3.7.0-cp38-cp38-win_amd64.whl (7.7 MB)

...
```

(py38_basic) C:\DEV> pip install scikit-learn

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

done
```

(py3_10_basic) PS C:\DEV> jupyter notebook

```
[I 2024-05-29 10:44:08.583 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-05-29 10:44:08.600 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-05-29 10:44:08.600 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-05-29 10:44:08.616 ServerApp] notebook | extension was successfully linked.
[I 2024-05-29 10:44:09.052 ServerApp] notebook_shim | extension was successfully linked.
```

기타 라이브러리 설치 확인

```
import sys
import numpy as np
print("numpy 버전 : {}".format(np.__version__))
print("python 버전 : {}".format(sys.version))
import pandas as pd
print("pandas 버전 : {}".format(pd.__version__))
import matplotlib
print("matplotlib 버전 : {}".format(matplotlib.__version__))
import scipy as sp
print("scipy 버전 : {}".format(sp.__version__))
import IPython
print("IPython 버전 : {}".format(IPython.__version__))
import sklearn
print("sklearn : {}".format(sklearn.__version__))
```

```
import sklearn
print("sklearn : {}".format(sklearn.__version__))
```

```
numpy 버전 : 1.24.3
python 버전 : 3.8.19 (default, Mar 20 2024, 19:55:45) [MSC v.1916 64 bit (AMD64)]
pandas 버전 : 2.0.3
matplotlib 버전 : 3.7.5
scipy 버전 : 1.10.1
IPython 버전 : 8.12.3
```

1. 데이터 분석 파이프라인

데이터 수집

- housing.csv라는 쉼표로 구분된 값(CSV) 파일이 포함된 단일 압축 파일housing.tgz를 다운로드하기만 하면 됩니다.

  - 웹 브라우저를 사용하여 다운로드하고 tar xzf housing.tgz를 실행하여 파일 압축을 풀고 CSV 파일을 추출할 수 있지만 그렇게 하려면 작은 기능을 만드는 것이 좋습니다.
  - 최신 데이터를 가져와야 할 때마다 실행할 수 있는 작은 스크립트를 작성할 수 있으므로 데이터가 정기적으로 변경되는 경우 특히 유용합니다(또는 정기적으로 자동으로 수행하도록 예약된 작업을 설정할 수 있음). 데이터 가져오기 프로세스를 자동화하는 것은 여러 시스템에 데이터 세트를 설치해야 하는 경우에도 유용합니다.

1. 데이터 분석 파이프라인

환경 설정

```
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family'] = 'Malgun Gothic' # Windows
# matplotlib.rcParams['font.family'] = 'AppleGothic' # Mac
matplotlib.rcParams['font.size'] = 15 # 글자 크기
matplotlib.rcParams['axes.unicode_minus'] = False # 한글 폰트 사용 시, 마이너스 글자가 깨지는 현상을 해결
```

1. 데이터 분석 파이프라인

데이터 수집

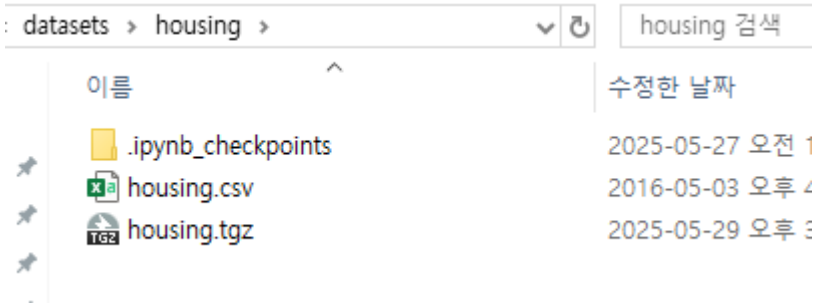
```
import os
import tarfile
from six.moves import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path): # housing_path 디렉토리가 이미 시스템에 존재하는지 확인
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

1. 데이터 분석 파이프라인

데이터 수집

- 이제 fetch_housing_data()를 호출하면 작업 공간에 datasets/housing 디렉토리가 생성되고, housing.tgz 파일을 다운로드하고, 이 디렉토리에서 housing.csv를 추출합니다.

fetch_housing_data()




```
C:\DEV\PycharmProjects\data_pre_processing>tree /f
폴더 PATH의 목록입니다.
볼륨 일련 번호는 2019-9371입니다....

...

└──datasets
    └──housing
            housing.csv
            housing.tgz
```

1. 데이터 분석 파이프라인

데이터 가져오기

- 이제 Pandas를 사용하여 데이터를 로드해 보겠습니다. 다시 한 번 데이터를 로드하는 작은 함수를 작성해야 합니다.

```
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)

housing = load_housing_data()
housing.head()
```

>>

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

1. 데이터 분석 파이프라인

Insights 얻기

- info() 메서드는 데이터, 특히 총 행 수, 각 속성의 유형 및 null이 아닌 값의 수에 대한 빠른 설명을 얻는 데 유용합니다. 데이터 세트에는 20,640개의 인스턴스가 있습니다. 각 행은 하나의 지구를 나타냅니다. 10개의 속성이 있습니다: 경도, 위도, house_median_age, total_rooms, total_bedrooms, 인구, 가구, median_ income, median_house_value 및 ocean_proximity.

housing.info()

```
>>
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

1. 데이터 분석 파이프라인

Insights 얻기

- 이 데이터 세트는 1970년대 보스턴 대도시 지역의 주택 단위에 대한 정보를 포함합니다. 데이터 세트에는 13개의 특징과 1개의 목표 변수가 있습니다.
- 특징:
  - 경도 (longitude): 주택이 위치한 지역의 경도입니다. 지리적으로 동경을 나타냅니다.
  - 위도 (latitude): 주택이 위치한 지역의 위도입니다. 지리적으로 북위를 나타냅니다.
  - 주택 중위 연령 (housing_median_age): 해당 지역의 주택의 중위 연령입니다. 즉, 해당 지역의 주택이 중간으로 분포하는 연식을 의미합니다.
  - 총 방 수 (total_rooms): 주택 내의 전체 방의 수입니다.
  - 총 침실 수 (total_bedrooms): 주택 내의 총 침실 수입니다.
  - 인구 (population): 해당 지역의 총 인구 수입니다.
  - 가구 (households): 주택 내의 총 가구 수입니다.
  - 중위 소득 (median_income): 해당 지역의 중위 소득입니다. 이는 가구의 중간 소득을 나타냅니다.
  - 해양 근접도 (ocean_proximity): 해당 주택이 바다에 얼마나 가깝게 위치하는지를 나타내는 속성으로, 일반적으로 범주형 데이터로 처리됩니다. 가능한 값은 '내륙', '해변', '바다로 접근 가능', '아주 가까움' 등이 있을 수 있습니다.
- 목표 변수:
  - 중위 주택 가격 (median_house_value): 해당 지역의 주택 가격의 중간값입니다. 이 값은 예측하려는 목표값일 수 있습니다.

Insights 얻기

- Ocean_proximity 필드를 제외한 모든 속성은 숫자입니다. 해당 유형은 객체이므로 모든 종류의 Python 객체를 보유할 수 있지만 CSV 파일에서 이 데이터를 로드했기 때문에 텍스트 속성이어야 한다는 것을 알 수 있습니다. 상위 5개 행을 보면 Ocean_proximity 열의 값이 반복적이라는 것을 알 수 있습니다. 이는 아마도 범주 속성일 수 있음을 의미합니다. value_counts() 메서드를 사용하여 어떤 범주가 존재하고 각 범주에 몇 개의 지구가 속하는지 확인할 수 있습니다.

```
housing["ocean_proximity"].value_counts()
```

```
>>
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

Insights 얻기

housing.describe()

>>

	Longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count		20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000
mean		-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671
std		2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822
min		-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900
25%		-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400
50%		-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800
75%		-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250
max		-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100

- 개수, 평균, 최소 및 최대. null 값은 무시됩니다(예: total_bedrooms 수는 20,640이 아니라 20,433임). std 행은 값이 얼마나 분산되어 있는지 측정하는 표준 편차를 보여줍니다.
- 25%, 50% 및 75% 행은 해당 백분위수를 표시합니다. 백분위수는 그룹에서 주어진 관찰 비율보다 낮은 값을 나타냅니다. 예를 들어, 구역의 25%는 18보다 낮은 주택_중간값을 갖고 있고 50%는 29세보다 낮고 75%는 37세보다 낮습니다. 이들은 종종 25번째 백분위수(또는 1사분위수), 중앙값 및 75번째 백분위수라고 합니다. (또는 3사분위수).

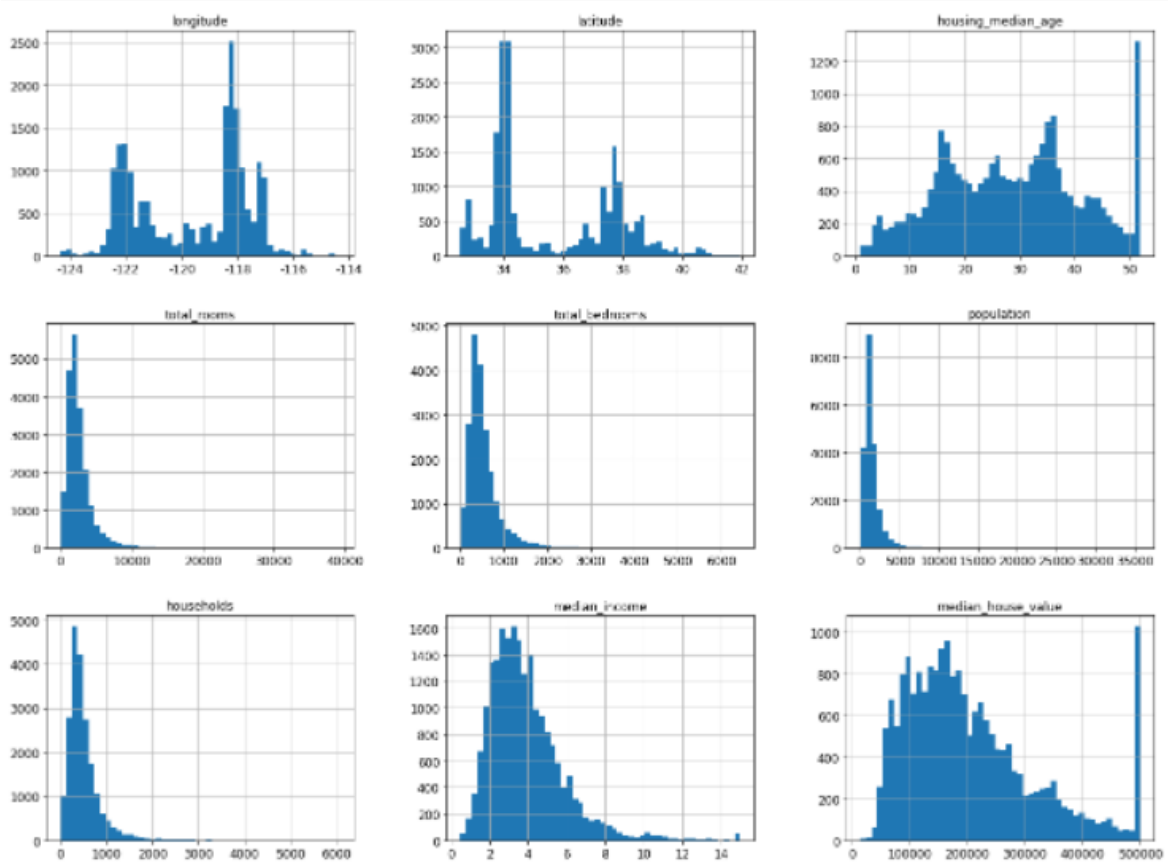
1. 데이터 분석 파이프라인

Insights 얻기

- 처리 중인 데이터 유형을 빠르게 파악하는 또 다른 방법은 각 숫자 속성에 대한 히스토그램을 그리는 것입니다. hist() 메서드는 Matplotlib에 의존하며, 이는 차례로 사용자 지정 그래픽 백엔드에 의존하여 화면에 그립니다.
- 따라서 플롯을 작성하기 전에 Matplotlib에서 사용할 백엔드를 지정해야 합니다. 가장 간단한 옵션은 Jupyter의 마법 명령 %matplotlib를 인라인으로 사용하는 것입니다. 이것은 Jupyter에 Matplotlib를 설정하도록 지시하여 Jupyter의 자체 백엔드를 사용합니다. 그런 다음 플롯은 노트북 자체 내에서 렌더링됩니다. Jupyter는 셀이 실행될 때 자동으로 플롯을 표시하므로 show() 호출은 Jupyter 노트북에서 선택 사항입니다.

Insights 얻기

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



1. 데이터 분석 파이프라인

Insights 얻기

- 히스토그램은 지정된 값 범위(가로 축)를 갖는 인스턴스(세로 축)의 수를 보여줍니다. 이 속성을 한 번에 하나씩 플로팅하거나 전체 데이터세트에 대해 hist() 메서드를 호출할 수 있습니다. 그러면 각 숫자 속성에 대한 히스토그램이 플로팅됩니다. 예를 들어, 800개가 약간 넘는 구역의 median_house_value가 약 $100,000인 것을 볼 수 있습니다.

- 이 히스토그램에서 몇 가지 사항을 확인할 수 있습니다.

  - 중위소득 속성이 미국 달러(USD)로 표시되지 않는 것처럼 보입니다.
  - 데이터를 수집한 팀과 확인한 후 데이터가 더 높은 중위 소득에 대해 15(실제로는 15.0001)로, 더 낮은 중위 소득에 대해 0.5(실제로는 0.4999)로 조정되고 상한선이 설정되었음을 알려줍니다.
  - 숫자는 대략 수만 달러를 나타냅니다(예: 3은 실제로 약 $30,000를 의미함).
  - 전처리된 속성으로 작업하는 것은 기계 학습에서 일반적이며 반드시 문제가 되는 것은 아니지만 데이터가 계산된 방식을 이해하려고 노력해야 합니다.

1. 데이터 분석 파이프라인

Insights 얻기

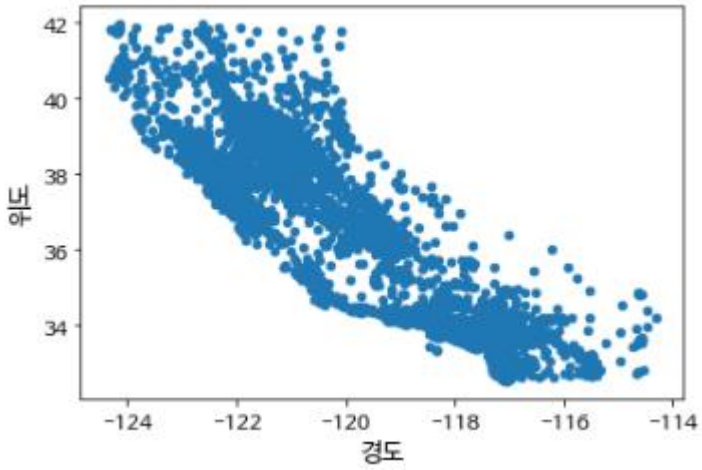
- 주택의 중위연령과 주택가격 중위도 상한선에 두었습니다. 후자는 대상 속성(레이블)이기 때문에 심각한 문제일 수 있습니다. 기계 학습 알고리즘은 가격이 한도를 초과하지 않는다는 것을 학습할 수 있습니다. 이것이 문제인지 아닌지 확인하려면 클라이언트 팀(시스템 출력을 사용할 팀)에 확인해야 합니다. $500,000를 넘어서도 정확한 예측이 필요하다고 말하면 주로 두 가지 옵션이 있습니다.라벨에 상한선이 있는 지역에 대해 적절한 라벨을 수집합니다. (시스템이 $500,000를 초과하는 값을 예측하는 경우 시스템이 제대로 평가되지 않아야 하기 때문)에서 제거하십시오.
- 이러한 속성은 매우 다른 척도를 가지고 있습니다.
- 마지막으로, 많은 히스토그램은 꼬리가 무겁습니다. 왼쪽보다 중앙값의 오른쪽으로 훨씬 더 확장됩니다. 이것은 일부 기계 학습 알고리즘이 패턴을 감지하는 것을 조금 더 어렵게 만들 수 있습니다. 우리는 나중에 더 많은 종 모양의 분포를 갖도록 이러한 속성을 변환하려고 시도할 것입니다.

1. 데이터 분석 파이프라인

Insights 얻기

- 데이터 이해를 위한 탐색과 시각화-지리 데이터 시각화
  - 지리 정보(위도 및 경도)가 있으므로 모든 지역의 산점도를 만들어 데이터를 시각화하는 것이 좋습니다.

```
ax = housing.plot(kind="scatter", x="longitude", y="latitude")
ax.set(xlabel='경도', ylabel='위도')
```

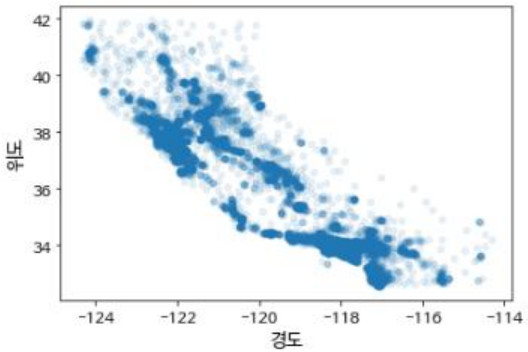


1. 데이터 분석 파이프라인

Insights 얻기

- 앞선 시각화는 캘리포니아처럼 보이지만 그 외에는 특별한 패턴을 보기 어렵습니다. 알파 옵션을 0.1로 설정하면 데이터 포인트 밀도가 높은 곳을 훨씬 쉽게 시각화할 수 있습니다.

```
ax = housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
ax.set(xlabel='경도', ylabel='위도')
```



- 고밀도 지역, 즉 베이 지역과 로스앤젤레스와 샌디에이고 주변과 센트럴 밸리, 특히 새크라멘토와 프레즈노 주변에서 상당히 고밀도로 길게 늘어선 줄을 명확하게 볼 수 있습니다. 매개변수를 다양하게 조절해봐야 합니다.

1. 데이터 분석 파이프라인

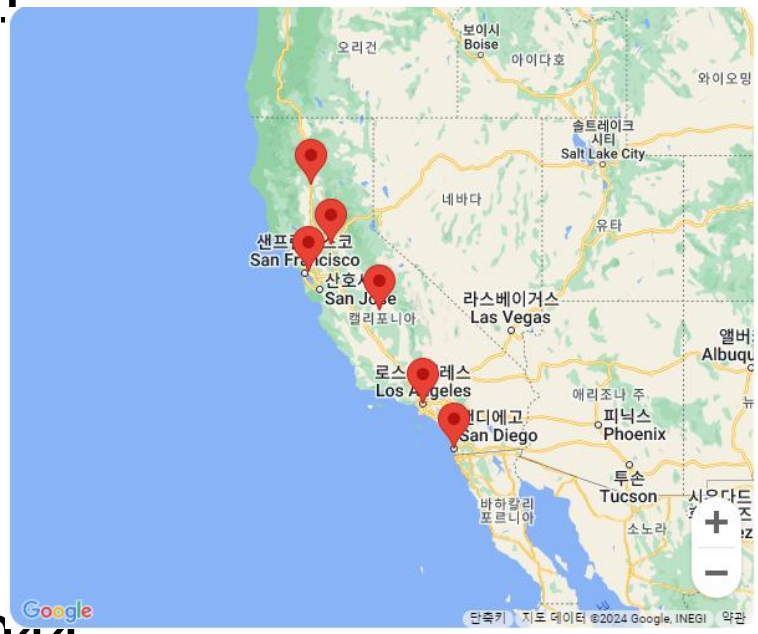
Insights 얻기

- 베이 지역 (Bay Area):
  - 샌프란시스코: 위도 37.7749, 경도 -122.4194
  - 오크랜드: 위도 37.8044, 경도 -122.2711
  - 샌호세: 위도 37.3382, 경도 -121.8863

- 로스앤젤레스 (Los Angeles):
  - 위도 34.0522, 경도 -118.2437

- 샌디에이고 (San Diego):
  - 위도 32.7157, 경도 -117.1611

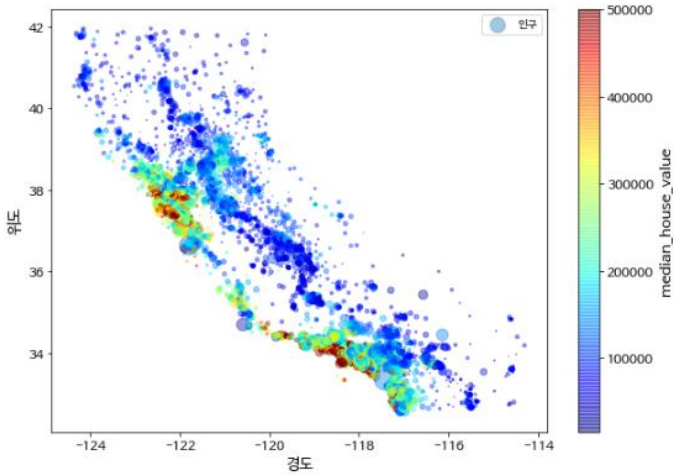
- 센트럴 밸리 (Central Valley):
  - 새크라멘토: 위도 38.5816, 경도 -121.4944
  - 프레즈노: 위도 36.7372, 경도 -119.7871



Insights 얻기

- 이제 주택가격을 살펴봅니다. 각 원의 반지름은 해당 구역의 인구(옵션 s)를 나타내고 색상은 가격(옵션 c)을 나타냅니다. 파란색(낮은 값)에서 빨간색(높은 가격)에 이르는 jet이라는 미리 정의된 색상 맵(옵션 cmap)을 사용합니다. 인구 밀도가 높을수록 해당 지역의 산점도 점의 색상은 붉은색에 가깝게 나타날 것으로 예상

```
ax = housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
    s=housing["population"]/100, label="인구", figsize=(10,7),
    c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
    sharex=False)
ax.set(xlabel='경도', ylabel='위도')
plt.legend()
```



1. 데이터 분석 파이프라인

Insights 얻기

- 이 이미지는 주택 가격이 위치(예: 바다와 가까움)와 인구 밀도가 높을 수록 높다라는 것을 알 수 있고, 위치와 인구 밀도와 밀접한 관련이 있음을 알려줍니다.
- 클러스터링 알고리즘을 사용하여 주요 클러스터를 감지하고 클러스터 중심에 대한 근접성을 측정하는 새로운 기능을 추가하는 것이 유용할 것입니다.

데이터 정제(Data Cleaning)

- 이제 데이터를 준비할 때입니다.
  - 먼저 데이터 세트를 통해, 예측 변수와 레이블을 분리하겠습니다.

```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

>>

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
290	-122.16	37.77	47.0	1256.0	NaN	570.0	218.0	4.3750	161900.0	NEAR BAY
341	-122.17	37.75	38.0	992.0	NaN	732.0	259.0	1.6196	85100.0	NEAR BAY
538	-122.28	37.78	29.0	5154.0	NaN	3741.0	1273.0	2.5762	173400.0	NEAR BAY
563	-122.24	37.75	45.0	891.0	NaN	384.0	146.0	4.9489	247100.0	NEAR BAY
696	-122.10	37.69	41.0	746.0	NaN	387.0	161.0	3.9063	178400.0	NEAR BAY

1. 데이터 분석 파이프라인

데이터 정제(Data Cleaning)

- 앞서, 예: total_bedrooms 수는 20,640이 아니라 20,433임에서 보듯이,

```
housing.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000

- 누락된 기능으로 작동할 수 없으므로 이를 처리하는 몇 가지 기능을 만들어 보겠습니다. total_bedrooms 속성에 일부 누락된 값이 있다는 것을 이전에 알았으므로 이를 수정하겠습니다. 세 가지 옵션이 있습니다.

① 해당 구역을 제거합니다.
② 전체 속성을 제거합니다.
③ 값을 일부 값(0, 평균, 중앙값 등)으로 설정합니다.

데이터 정제(Data Cleaning)

- DataFrame의 dropna(), drop() 및 fillna() 메서드를 사용하여 이러한 작업을 쉽게 수행할 수 있습니다.

```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # 옵션 1
```

```
>>
longitude latitude housing_median_age total_rooms total_bedrooms population households
median_income median_house_value ocean_proximity
```

```
sample_incomplete_rows.drop("total_bedrooms", axis=1)      # 옵션 2
```

```
>>
longitude latitude housing_median_age total_rooms population households median_income
median_house_value ocean_proximity
290 -122.16 37.77 47.0 1256.0 570.0 218.0 4.3750 161900.0 NEAR
BAY
341 -122.17 37.75 38.0 992.0 732.0 259.0 1.6196 85100.0 NEAR
BAY
538 -122.28 37.78 29.0 5154.0 3741.0 1273.0 2.5762 173400.0 NEAR
BAY
563 -122.24 37.75 45.0 891.0 384.0 146.0 4.9489 247100.0 NEAR
BAY
696 -122.10 37.69 41.0 746.0 387.0 161.0 3.9063 178400.0 NEAR
BAY
```

데이터 정제(Data Cleaning)

- 옵션 3을 선택하면 훈련 세트의 중앙값을 계산하고 훈련 세트의 누락된 값을 채우는 데 사용해야 하지만 계산한 중앙값을 저장하는 것도 잊지 마십시오. 나중에 시스템을 평가할 때 테스트 세트의 누락된 값을 교체하고 시스템이 가동되어 새 데이터의 누락된 값을 교체할 때 필요합니다.

```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # 옵션 3
sample_incomplete_rows
```

```
>>
                longitude latitude housing_median_age total_rooms total_bedrooms population households
median_income median_house_value ocean_proximity
290             -122.16   37.77    47.0               1256.0      435.0         570.0        218.0      4.3750
161900.0 NEAR BAY
341             -122.17   37.75    38.0               992.0       435.0         732.0        259.0      1.6196
85100.0 NEAR BAY
538             -122.28   37.78    29.0               5154.0      435.0         3741.0       1273.0     2.5762
173400.0 NEAR BAY
563             -122.24   37.75    45.0               891.0       435.0         384.0        146.0      4.9489
247100.0 NEAR BAY
696             -122.10   37.69    41.0               746.0       435.0         387.0        161.0      3.9063
178400.0 NEAR BAY
```

데이터 정제(Data Cleaning)

- Scikit-Learn은 누락된 값을 처리하는 간편한 클래스인 SimpleImputer를 제공합니다. 사용 방법은 다음과 같습니다. 먼저 각 속성의 누락된 값을 해당 속성의 중앙값으로 대체하도록 지정하여 SimpleImputer 인스턴스를 생성해야 합니다.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

- 중앙값은 숫자 속성에서만 계산할 수 있으므로 텍스트 속성 ocean_proximity을 제외한 데이터 복사본을 만들어야 합니다.

```
housing_num = housing.drop('ocean_proximity', axis=1)
```

- 이제 fit() 메서드를 사용하여 imputer 인스턴스를 훈련 데이터에 맞출 수 있습니다.

```
imputer.fit(housing_num)
```

훈련 세트 변환

- 이제 이 "훈련된" imputer 를 사용하여 누락된 값을 학습된 중앙값으로 대체하여 훈련 세트를 변환할 수 있습니다.

```
X = imputer.transform(housing_num)
```

- 결과는 변환된 기능을 포함하는 일반 NumPy 배열입니다. Pandas DataFrame에 다시 넣고 싶다면 간단합니다.

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index = list(housing.index.values))
housing_tr.loc[sample_incomplete_rows.index.values]
>>
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
290	-122.16	37.77	47.0	1256.0	435.0	570.0	218.0	4.3750	161900.0
341	-122.17	37.75	38.0	992.0	435.0	732.0	259.0	1.6196	85100.0
538	-122.28	37.78	29.0	5154.0	435.0	3741.0	1273.0	2.5762	173400.0
563	-122.24	37.75	45.0	891.0	435.0	384.0	146.0	4.9489	247100.0
696	-122.10	37.69	41.0	746.0	435.0	387.0	161.0	3.9063	178400.0

훈련 세트 변환

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns)
housing_tr.head()
>>
```

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0

Text와 Categorical 속성 처리하기

- 앞서 텍스트 및 범주 속성 처리 시 범주형 속성 ocean_proximity를 생략했는데, 이는 텍스트 속성이기 때문에 중앙값을 계산할 수 없기 때문입니다.

```
housing_cat = housing['ocean_proximity']
housing_cat.head(10)
>>
0    NEAR BAY
1    NEAR BAY
2    NEAR BAY
3    NEAR BAY
4    NEAR BAY
5    NEAR BAY
6    NEAR BAY
7    NEAR BAY
8    NEAR BAY
9    NEAR BAY
Name: ocean_proximity, dtype: object
```

Text와 Categorical 속성 처리하기

- 대부분의 기계 학습 알고리즘은 어쨌든 숫자로 작업하는 것을 선호하므로 이러한 범주를 텍스트에서 숫자로 변환해 보겠습니다. 이를 위해 판다스의 factorize() 메소드를 사용할 수 있습니다.

```
housing_cat_encoded, housing_categories = housing_cat.factorize()
housing_cat_encoded[:10]
>>
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

- housing_cat_encoded는 완전히 숫자입니다. factorize() 메소드는 카테고리 리스트도 반환해줍니다.

```
housing_categories
>>
Index(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'], dtype='object')
```

Text와 Categorical 속성 처리하기

- OneHotEncoder
  - 범주가 "<1H OCEAN"일 때 하나의 속성은 1이고(그렇지 않으면 0), 범주가 "INLAND"일 때 다른 속성은 1입니다(그렇지 않으면 0) 등입니다. 하나의 속성만 1(핫)이고 다른 속성은 0(콜드)이기 때문에 이것을 원-핫 인코딩이라고 합니다. 새 속성을 더미 속성이라고도 합니다.
  - Scikit-Learn은 범주형 값을 원-핫 벡터20로 변환하는 OneHotEncoder 클래스를 제공합니다.

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(categories='auto')
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
housing_cat_1hot
>>
<20640x5 sparse matrix of type '<class 'numpy.float64'>'
                with 20640 stored elements in Compressed Sparse Row format>
```

Text와 Categorical 속성 처리하기

- 출력은 NumPy 배열 대신 SciPy 희소 행렬입니다. 이는 수천 개의 범주가 있는 범주 속성이 있을 때 매우 유용합니다. 원-핫 인코딩 후에 우리는 수천 개의 열이 있는 행렬을 얻고 행렬은 행당 하나의 1을 제외하고는 0으로 가득 차 있습니다. 0을 저장하기 위해 많은 양의 메모리를 사용하는 것은 매우 낭비이므로 대신 희소 행렬은 0이 아닌 요소의 위치만 저장합니다. 대부분의 일반 2D 배열처럼 사용할 수 있지만, 실제로 이를 (밀도가 높은) NumPy 배열로 변환하려면 toarray() 메서드를 호출하면 됩니다.

```
housing_cat_1hot.toarray()
>>
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       ...,
       [0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 1., 0., 0.]])
```

Text와 Categorical 속성 처리하기

```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder(categories='auto')
housing_cat_reshaped = housing_cat.values.reshape(-1, 1)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat_reshaped)
housing_cat_1hot
```

- 기본 인코딩은 원-핫 벡터이고 희소 행렬로 반환됩니다. toarray() 메소드를 사용하여 밀집 배열로 바꿀 수 있습니다.

```
housing_cat_1hot.toarray()
>>
array([[0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       ...,
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

Text와 Categorical 속성 처리하기

- 또는 encoding 매개변수를 "onehot-dense"로 지정하여 희소 행렬대신 밀집 행렬을 얻을 수 있습니다. 0.20 버전의 OneHotEncoder는 sparse=Fasle 옵션을 주어 밀집 행렬을 얻을 수 있습니다.

```
# cat_encoder = CategoricalEncoder(encoding="onehot-dense")
cat_encoder = OneHotEncoder(categories='auto', sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat_reshaped)
housing_cat_1hot
>>
array([[0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0.],
       ...,
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

```
cat_encoder.categories_
>>
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

1. 데이터 분석 파이프라인

Feature Scaling

- 데이터에 적용해야 하는 가장 중요한 변환 중 하나는 Feature Scaling입니다. 몇 가지 예외를 제외하고 기계 학습 알고리즘은 입력 수치 속성이 매우 다른 척도를 가질 때 잘 수행되지 않습니다. 주택 데이터의 경우에서도, 총 방 수는 약 6에서 39,320 사이이고 중위 소득은 0에서 15 사이입니다.
- 모든 속성이 동일한 스케일을 갖도록 하는 두 가지 일반적인 방법이 있습니다. 최소-최대 스케일링(min-max scaling)와 표준화(standardization)가 그것입니다.
- min-max scaling 은 간단합니다.(정규화라고도 함) 0~1 범위에 들도록 값을 이동하고 스케일을 조정하면 됩니다. 데이터에서 최솟값을 뺀 후 최댓값과 최솟값의 차이로 나누면 이렇게 할 수 있습니다. Scikit-Learn은 이를 위해 MinMaxScaler라는 변환기를 제공합니다.
- 표준화는 먼저 평균을 뺀 후, 표준편차로 나누어 결과 분포의 분산이 1이 되도록 합니다.

1. 데이터 분석 파이프라인

Test Set 생성

- 테스트 세트를 만드는 것은 이론적으로 매우 간단합니다. 일반적으로 데이터 세트의 20%(또는 데이터 세트가 매우 큰 경우 그 이하)인 일부 인스턴스를 무작위로 선택하고 따로 저장하면 됩니다.

- 일관된 출력을 위해 유사난수 초기화

```
import numpy as np

np.random.seed(42)
```

- 프로그램을 다시 실행하면 다른 테스트 세트가 생성됩니다! 시간이 지남에 따라 사용자(또는 기계 학습 알고리즘)는 전체 데이터 세트를 보게 되며, 이는 피하고 싶은 것입니다.한 가지 솔루션은 첫 번째 실행에서 테스트 세트를 저장한 다음 후속 실행에서 로드하는 것입니다. 또 다른 옵션은 np.random.permutation()을 호출하기 전에 난수 생성기의 시드(예: np.ran dom.seed(42))14를 설정하여 항상 동일한 셔플 인덱스를 생성하도록 하는 것입니다.

Test Set 생성

- 사이킷런에 train_test_split() 함수가 있습니다.

```
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

- 앞 서 작성한 함수를 호출해 사용합니다.

```
train_set, test_set = split_train_test(housing, 0.2)
print(len(train_set), "train +", len(test_set), "test")
>>
16512 train + 4128 test
```

Test Set 생성

- 그러나 이 두 솔루션은 다음에 업데이트된 데이터 세트를 가져올 때 중단됩니다. 일반적인 솔루션은 각 인스턴스의 식별자를 사용하여 테스트 세트에 들어갈지 여부를 결정하는 것입니다(인스턴스에 고유하고 변경할 수 없는 식별자가 있다고 가정). 예를 들어, 각 인스턴스 식별자의 해시를 계산하고 해시가 최대 해시 값의 20% 이하이면 해당 인스턴스를 테스트 세트에 넣을 수 있습니다. 이렇게 하면 데이터 세트를 새로 고치더라도 테스트 세트가 여러 실행에서 일관되게 유지됩니다. 새 테스트 세트에는 새 인스턴스의 20%가 포함되지만 이전에 훈련 세트에 있던 인스턴스는 포함되지 않습니다. 다음은 이를 구현한 코드입니다.

```
from zlib import crc32

def test_set_check(identifier, test_ratio):
    return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2>>32

def split_train_test_by_id(data, test_ratio, id_column):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

Test Set 생성

- 불행히도 주택 데이터 세트에는 식별자 열이 없습니다. 가장 간단한 솔루션은 행 인덱스를 ID로 사용하는 것입니다.

```
housing_with_id = housing.reset_index()   # `index` 열이 추가된 데이터프레임이 반환됩니다.
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

- 행 인덱스를 고유 식별자로 사용하는 경우 새 데이터가 데이터 세트 끝에 추가되고 행이 삭제되지 않도록 해야 합니다. 이것이 가능하지 않은 경우 가장 안정적인 기능을 사용하여 고유 식별자를 구축할 수 있습니다. 예를 들어, 지구의 위도와 경도는 수백만 년 동안 안정적으로 보장되므로 다음과 같이 ID로 결합할 수 있습니다.

```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

Test Set 생성

test_set.head()

>>

index	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	
	median_income	median_house_value	ocean_proximity	id				
59	59	-122.29	37.82	2.0	158.0	43.0	94.0	57.0	2.5625
	60000.0	NEAR BAY	-122252.18					
60	60	-122.29	37.83	52.0	1121.0	211.0	554.0	187.0	3.3929
	75700.0	NEAR BAY	-122252.17					
61	61	-122.29	37.82	49.0	135.0	29.0	86.0	23.0	6.1183
	75000.0	NEAR BAY	-122252.18					
62	62	-122.29	37.81	50.0	760.0	190.0	377.0	122.0	0.9011
	86100.0	NEAR BAY	-122252.19					
67	67	-122.29	37.80	52.0	1027.0	244.0	492.0	147.0	2.6094
	81300.0	NEAR BAY	-122252.20					

Test Set 생성

- Scikit-Learn은 데이터 세트를 다양한 방식으로 여러 하위 집합으로 분할하는 몇 가지 기능을 제공합니다. 가장 간단한 함수는 train_test_split으로, 앞서 정의한 split_train_test 함수와 거의 동일한 작업을 수행하며 몇 가지 추가 기능이 있습니다. 먼저 random_state 매개변수를 사용하여 이전에 설명한 대로 임의 생성기 시드를 설정할 수 있고, 두 번째로 동일한 수의 행이 있는 여러 데이터 세트를 전달할 수 있으며 동일한 인덱스로 분할합니다(이는 매우 유용합니다. 예를 들어 레이블에 대해 별도의 DataFrame이 있는 경우)

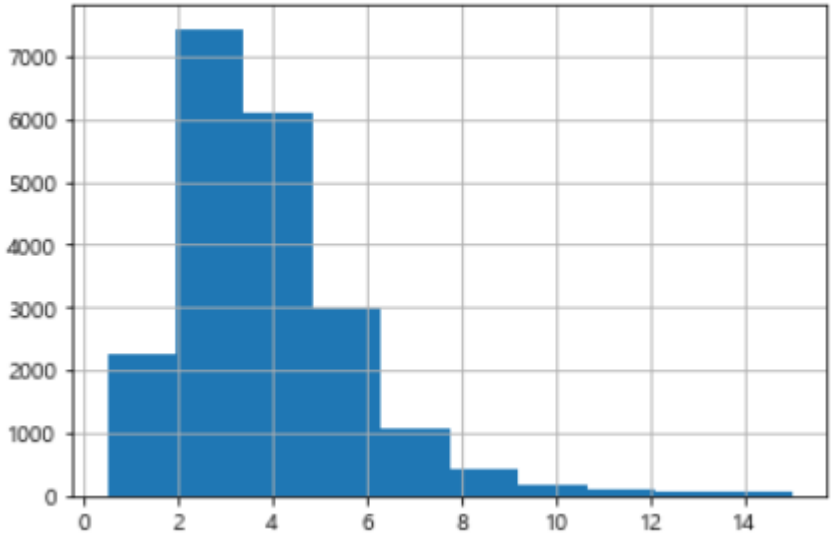
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

Test Set 생성

test_set.head()
>>

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	
	median_income	median_house_value	ocean_proximity					
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	47700.0
	INLAND							
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	45800.0
	INLAND							
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	
	500001.0	NEAR BAY						
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	
	218600.0	<1H OCEAN						
9814-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	278000.0	NEAR
OCEAN								

housing["median_income"].hist()



1. 데이터 분석 파이프라인

Test Set 생성

- 지금까지 우리는 순전히 무작위 샘플링 방법을 고려했습니다.
  - 일반적으로 데이터세트가 충분히 큰 경우(특히 속성 수에 비해) 문제가 없지만 그렇지 않은 경우 상당한 샘플링 편향이 발생할 위험이 있습니다.
  - 설문조사 회사가 1,000명에게 전화를 걸어 몇 가지 질문을 하기로 결정할 때 전화번호부에서 무작위로 1,000명을 뽑는 것이 아닙니다. 그들은 이 1,000명이 전체 인구를 대표할 수 있도록 노력합니다. 예를 들어, 미국 인구는 51.3%의 여성과 48.7%의 남성으로 구성되어 있으므로 미국에서 잘 수행된 설문 조사는 표본에서 이 비율을 유지하려고 할 것입니다.
- 513명의 여성과 487명의 남성. 이것을 층화 표본 추출이라고 합니다.
  - 모집단은 계층이라고 하는 동질적인 하위 그룹으로 나뉘고 테스트 세트가 전체 모집단을 대표하도록 보장하기 위해 각 층에서 올바른 수의 인스턴스가 표본 추출됩니다.
  - 순전히 무작위 샘플링을 사용하는 경우 여성이 49% 미만이거나 여성이 54% 이상인 편향된 테스트 세트를 샘플링할 확률이 약 12%입니다.
  - 어느 쪽이든, 설문 조사 결과는 상당히 편향될 것입니다.

1. 데이터 분석 파이프라인

Test Set 생성

- 중위 소득이 중위 주택 가격을 예측하는 데 매우 중요한 속성이라고 말한 전문가와 이야기를 나눴다고 가정해 보겠습니다. 테스트 세트가 전체 데이터 세트의 다양한 소득 범주를 대표하는지 확인하고 싶을 수 있습니다. 중위 소득 히스토그램을 더 자세히 살펴보겠습니다.
- 대부분의 중위 소득 값은 1.5에서 6 사이(즉, $15,000–$60,000)에 모여 있지만 일부 중위 소득은 6을 훨씬 초과합니다. 각 계층에 대한 데이터 세트의 인스턴스 수가 충분하지 않으면 계층의 중요도 추정치가 편향될 수 있습니다. 즉, 계층이 너무 많아서는 안 되며 각 계층은 충분히 커야 합니다.

- 소득 카테고리 개수를 제한하기 위해 1.5로 나눕니다.

```
housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
```

- "median_income" 열은 해당 지역의 중간 소득을 나타내는데, 이 값을 1.5로 나눈 후 올림하여 얻은 값은 소득 카테고리를 나타냅니다.

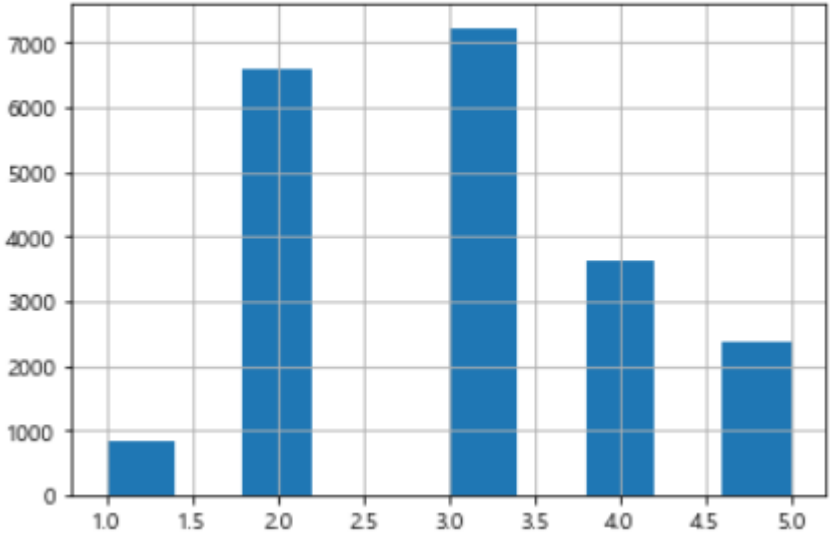
Test Set 생성

- 5 이상은 5로 레이블합니다.

```
housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
housing["income_cat"].value_counts()
>>
income_cat
3.0    7236
2.0    6581
4.0    3639
5.0    2362
1.0     822
Name: count, dtype: int64
```

Test Set 생성

housing["income_cat"].hist()



Test Set 생성

- 이제 소득 범주를 기반으로 계층화된 샘플링을 수행할 준비가 되었습니다. 이를 위해 Scikit-Learn의 StratifiedShuffleSplit 클래스를 사용할 수 있습니다.

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]

strat_test_set["income_cat"].value_counts() / len(strat_test_set)
>>
income_cat
3.0    0.350533
2.0    0.318798
4.0    0.176357
5.0    0.114341
1.0    0.039971
Name: count, dtype: float64
```

Test Set 생성

```
housing["income_cat"].value_counts() / len(housing)
>>
```

```
3.0    0.350581
2.0    0.318847
4.0    0.176308
5.0    0.114438
1.0    0.039826
Name: income_cat, dtype: float64
```

Test Set 생성

- 계층화된 샘플링과 순수 무작위 샘플링의 샘플링 편향 비교

```
def income_cat_proportions(data):
    return data["income_cat"].value_counts() / len(data)

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)

compare_props = pd.DataFrame({
    "Overall": income_cat_proportions(housing),
    "Stratified": income_cat_proportions(strat_test_set),
    "Random": income_cat_proportions(test_set),
}).sort_index()
compare_props["Rand. %error"] = 100 * compare_props["Random"] / compare_props["Overall
"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] / compare_props["Overal
l"] - 100
```

Test Set 생성

- 계층화된 샘플링과 순수 무작위 샘플링의 샘플링 편향 비교

compare_props
>>

income_cat	Overall	Stratified	Random	Rand. %error	Strat. %error
1.0	0.039826	0.039971	0.040213	0.973236	0.364964
2.0	0.318847	0.318798	0.324370	1.732260	-0.015195
3.0	0.350581	0.350533	0.358527	2.266446	-0.013820
4.0	0.176308	0.176357	0.167393	-5.056334	0.027480
5.0	0.114438	0.114341	0.109496	-4.318374	-0.084674

– 이 표는 각 소득 카테고리(income_cat)에 대한 샘플의 비율을 보여줍니다.
- Overall: 전체 데이터셋에서 각 소득 카테고리의 비율입니다.
- Stratified: 계층화된 샘플링에 의해 추출된 각 소득 카테고리의 비율입니다.
- Random: 순수 무작위 샘플링에 의해 추출된 각 소득 카테고리의 비율입니다.
- Rand. %error: 순수 무작위 샘플링에 대한 표본 오차(퍼센트)입니다.
- Strat. %error: 계층화된 샘플링에 대한 표본 오차(퍼센트)입니다.

– 이 표를 통해 다음을 알 수 있습니다.
- 계층화된 샘플링은 소득 카테고리별로 전반적으로 더 정확한 비율을 유지하고 있습니다.
- 순수 무작위 샘플링은 표본 오차가 더 높은 경향이 있습니다. 특히, 상대적으로 소수인 카테고리에서 오차가 크게 나타날 수 있습니다.
- 계층화된 샘플링은 보다 일관된 결과를 제공하며, 순수 무작위 샘플링보다 더 신뢰할 수 있습니다.

Test Set 생성

- 이제 데이터가 원래대로 돌아가도록 income_cat 속성을 제거해야 합니다.

```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)


housing = strat_train_set.copy()
```

Custom Transformers

- Scikit-Learn은 많은 유용한 변환기를 제공하지만 사용자 정의 정리 작업 또는 특정 속성 결합과 같은 작업을 위해 자체 변환기를 작성해야 합니다. 변환기가 파이프라인과 같은 Scikit-Learn 기능과 원활하게 작동하기를 원할 것이며 Scikit-Learn은 덕 타이핑(상속이 아님)에 의존하기 때문에 클래스를 만들고 세 가지 방법을 구현하기만 하면 됩니다.
- fit()(자기 반환), transform() 및 fit_transform(). TransformerMixin을 기본 클래스로 추가하기만 하면 마지막 항목을 무료로 얻을 수 있습니다. 또한 BaseEstima tor를 기본 클래스로 추가하고 생성자에서 *args 및 **kargs를 피하면 자동 하이퍼파라미터 조정에 유용한 두 가지 추가 메서드(get_params() 및 set_params())가 생깁니다. 예를 들어, 다음은 앞에서 논의한 결합된 속성을 추가하는 작은 변환기 클래스입니다.

1. 데이터 분석 파이프라인

Custom Transformers

- 이 예에서 변환기에는 기본적으로 True로 설정되는 add_bedrooms_per_room이라는 하이퍼파라미터가 있습니다(적당한 기본값을 제공하는 것이 도움이 되는 경우가 많습니다). 이 하이퍼파라미터를 사용하면 이 속성을 추가하는 것이 기계 학습 알고리즘에 도움이 되는지 여부를 쉽게 확인할 수 있습니다. 더 일반적으로, 100% 확신할 수 없는 데이터 준비 단계를 제어하기 위해 하이퍼파라미터를 추가할 수 있습니다. 이러한 데이터 준비 단계를 자동화할수록 더 많은 조합을 자동으로 시도할 수 있으므로 훌륭한 조합을 찾을 가능성이 훨씬 높아집니다

Custom Transformers

```
from sklearn.base import BaseEstimator, TransformerMixin

# 컬럼 인덱스
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

Transformation Pipelines

– 수치 특성을 전처리하기 위한 파이프라인을 만듭니다(0.20 버전에 새로 추가된 SimpleImputer 클래스로 변경합니다) Scikit-Learn은 이러한 변환 시퀀스를 돕기 위해 Pipeline 클래스를 제공합니다. 다음은 숫자 속성에 대한 작은 파이프라인입니다.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

```
>>
array([[-1.32783522,  1.05254828,  0.98214266, ...,  0.62855945,
       -0.04959654, -1.02998783],
      [-1.32284391,  1.04318455, -0.60701891, ...,  0.32704136,
       -0.09251223, -0.8888972 ],
      [-1.33282653,  1.03850269,  1.85618152, ...,  1.15562047,
       -0.02584253, -1.29168566],
      ...,
      [-0.8237132 ,  1.77823747, -0.92485123, ..., -0.09031802,
       -0.0717345 ,  0.02113407],
      [-0.87362627,  1.77823747, -0.84539315, ..., -0.04021111,
       -0.09122515,  0.09346655],
      [-0.83369581,  1.75014627, -1.00430931, ..., -0.07044252,
       -0.04368215,  0.11327519]])
`
```

1. 데이터 분석 파이프라인

Transformation Pipelines

- 파이프라인 생성자는 일련의 단계를 정의하는 이름/추정자 쌍 목록을 사용합니다. 마지막 추정기를 제외한 모든 추정기는 변환기여야 합니다(즉, fit_transform() 메서드가 있어야 함). 이름은 원하는 대로 지정할 수 있습니다(고유하고 이중 밑줄 "__"이 포함되지 않는 한). 나중에 초매개변수 조정에 유용할 것입니다.
- 파이프라인의 fit() 메서드를 호출하면 모든 변환기에서 순차적으로 fit_transform()을 호출하여 각 호출의 출력을 다음 호출의 매개변수로 전달하고 최종 추정기에 도달할 때까지 fit()을 호출합니다.
- 이 예에서 마지막 추정기는 변환기인 StandardScaler이므로 파이프라인에는 모든 변환을 순서대로 데이터에 적용하는 trans form() 메서드(물론 fit_transform() 메서드도 있습니다. 우리가 사용한 것).
- 지금까지 범주 열과 숫자 열을 별도로 처리했습니다. 각 열에 적절한 변환을 적용하여 모든 열을 처리할 수 있는 단일 변환기를 갖는 것이 더 편리할 것입니다.
- 버전 0.20에서 Scikit-Learn은 이러한 목적으로 ColumnTransformer를 도입했으며 좋은 소식은 Pandas DataFrames와 잘 작동한다는 것입니다.

Transformation Pipelines

- 판단스 DataFrame 컬럼의 일부를 선택하는 변환기를 만듭니다

```
from sklearn.base import BaseEstimator, TransformerMixin

# 사이킷런이 DataFrame을 바로 사용하지 못하므로
# 수치형이나 범주형 컬럼을 선택하는 클래스를 만듭니다.
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

Transformation Pipelines

- CategoricalEncoder class를 직접 소스에 추가

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils import check_array
from sklearn.preprocessing import LabelEncoder
from scipy import sparse

class CategoricalEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, encoding='onehot', categories='auto', dtype=np.float64,
                 handle_unknown='error'):
        self.encoding = encoding
        self.categories = categories
        self.dtype = dtype
        self.handle_unknown = handle_unknown

    def fit(self, X, y=None):
        if self.encoding not in ['onehot', 'onehot-dense', 'ordinal']:
            template = ("encoding should be either 'onehot', 'onehot-dense' "
                        "or 'ordinal', got %s")
            raise ValueError(template % self.handle_unknown)

        if self.handle_unknown not in ['error', 'ignore']:
            template = ("handle_unknown should be either 'error' or "
                        "'ignore', got %s")
            raise ValueError(template % self.handle_unknown)

        if self.encoding == 'ordinal' and self.handle_unknown == 'ignore':
            raise ValueError("handle_unknown='ignore' is not supported for"
                             " encoding='ordinal'")
```

Transformation Pipelines

- CategoricalEncoder class를 직접 소스에 추가

```
...
    X = check_array(X, dtype=object, accept_sparse='csc', copy=True)
    n_samples, n_features = X.shape

    self._label_encoders_ = [LabelEncoder() for _ in range(n_features)]

    for i in range(n_features):
        le = self._label_encoders_[i]
        Xi = X[:, i]
        if self.categories == 'auto':
            le.fit(Xi)
        else:
            valid_mask = np.in1d(Xi, self.categories[i])
            if not np.all(valid_mask):
                if self.handle_unknown == 'error':
                    diff = np.unique(Xi[~valid_mask])
                    msg = ("Found unknown categories {0} in column {1}"
                           " during fit".format(diff, i))
                    raise ValueError(msg)
            le.classes_ = np.array(np.sort(self.categories[i]))

    self.categories_ = [le.classes_ for le in self._label_encoders_]

    return self
```

Transformation Pipelines

- CategoricalEncoder class를 직접 소스에 추가

```
...

    def transform(self, X):
        X = check_array(X, accept_sparse='csc', dtype=object, copy=True)
        n_samples, n_features = X.shape
        X_int = np.zeros_like(X, dtype=int)
        X_mask = np.ones_like(X, dtype=bool)

        for i in range(n_features):
            valid_mask = np.in1d(X[:, i], self.categories_[i])

            if not np.all(valid_mask):
                if self.handle_unknown == 'error':
                    diff = np.unique(X[~valid_mask, i])
                    msg = ("Found unknown categories {0} in column {1}"
                           " during transform".format(diff, i))
                    raise ValueError(msg)
            else:
                X_mask[:, i] = valid_mask
                X[:, i][~valid_mask] = self.categories_[i][0]

            X_int[:, i] = self._label_encoders_[i].transform(X[:, i])

        if self.encoding == 'ordinal':
            return X_int.astype(self.dtype, copy=False)
```

Transformation Pipelines

- CategoricalEncoder class를 직접 소스에 추가

...

```
mask = X_mask.ravel()
n_values = [cats.shape[0] for cats in self.categories_]
n_values = np.array([0] + n_values)
indices = np.cumsum(n_values)

column_indices = (X_int + indices[:-1]).ravel()[mask]
row_indices = np.repeat(np.arange(n_samples, dtype=np.int32),
                        n_features)[mask]
data = np.ones(n_samples * n_features)[mask]

out = sparse.csc_matrix((data, (row_indices, column_indices)),
                        shape=(n_samples, indices[-1]),
                        dtype=self.dtype).tocsr()
if self.encoding == 'onehot-dense':
    return out.toarray()
else:
    return out
```

Transformation Pipelines

- 하나의 큰 파이프라인에 이들을 모두 결합하여 수치형과 범주형 특성을 전처리합니다. 0.20 버전에 추가된 SimpleImputer를 사용합니다.

```
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('cat_encoder', CategoricalEncoder(encoding="onehot-dense")),
    ])
```

1. 데이터 분석 파이프라인

Transformation Pipelines

- 사이킷런 0.20 버전에 추가된 ColumnTransformer로 만든 full_pipline을 사용합니다.

```
from sklearn.pipeline import FeatureUnion

full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),
    ])
```

Transformation Pipelines

- 전체 파이프라인 실행

```
housing_prepared = full_pipeline.fit_transform(housing)
housing_prepared
>>

array([[-0.94135046,  1.34743822,  0.02756357, ...,  0.        ,
        0.        ,  0.        ],
       [ 1.17178212, -1.19243966, -1.72201763, ...,  0.        ,
        0.        ,  1.        ],
       [ 0.26758118, -0.1259716 ,  1.22045984, ...,  0.        ,
        0.        ,  0.        ],
       ...,
       [-1.5707942 ,  1.31001828,  1.53856552, ...,  0.        ,
        0.        ,  0.        ],
       [-1.56080303,  1.2492109 , -1.1653327 , ...,  0.        ,
        0.        ,  0.        ],
       [-1.28105026,  2.02567448, -0.13148926, ...,  0.        ,
        0.        ,  0.        ]])
```

Transformation Pipelines

- 전체 파이프라인 실행

```
housing_prepared.shape
>>

(16512, 17)
```

1. 데이터 분석 파이프라인

훈련 세트에 대한 훈련 및 평가

- 문제의 틀을 잡고 데이터를 가져와 탐색하고 훈련 세트와 테스트 세트를 샘플링하고 변환 파이프라인을 작성하여 기계 학습 알고리즘을 위해 데이터를 자동으로 정리하고 준비했습니다. 이제 기계 학습 모델을 선택하고 학습할 준비가 되었습니다.

```
housing_labels = strat_train_set["median_house_value"].copy()
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

- 완료! 이제 작동하는 선형 회귀 모델을 만들었습니다.

Hurray !

THANK YOU.