ㅇㅇ마이스터 고등학교

파이썬 활용 AI 프로그래밍

『 1-2』

2024. 05. 20. - 2024. 05. 24

Prepared by DaeKyeong Kim

Ph.D.

KOREATECH 한국기술교육대학교

목차

『3과목』 AI와 Data pre-processing

AI 학습 모델 파이프라인

Syllabus

학습목표

- 이 워크샵에서는 Data Analysis Pipeline에 대해 알 수 있습니다.

눈높이 체크

- Data Analysis Pipeline을 알고 계신가요?

1. AI 학습 모델 파이프라인

캘리포니아의 주택 가격 모델 만들어 보기

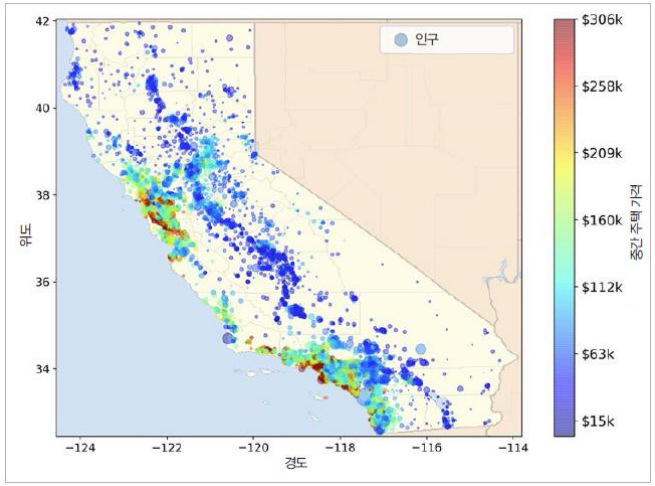
- 부동산 회사에 최근에 고용된 데이터 과학자인 것처럼 가장하여 예제 프로젝트를 처음부터 끝까지 진행합니다. 거쳐야 할 주요 단계는 다음과 같습니다.

- 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. AI 학습 모델 파이프라인

Problem와 question

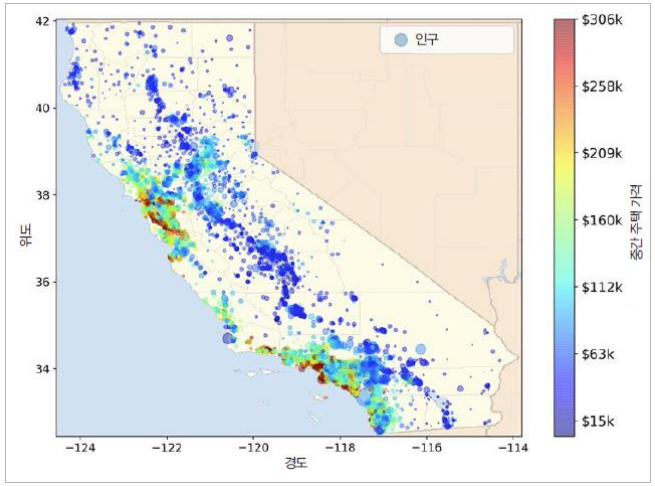
- 이 장에서 우리는 StatLib 저장소(The original dataset appeared in R. Kelley Pace and Ronald Barry, “Sparse Spatial Autoregressions,” Statistics & Probability Letters 33, no. 3 (1997): 291–297.)에서 캘리포니아 주택 가격 데이터 세트를 선택했습니다. 이 데이터 세트는 1990년 캘리포니아 인구 조사 데이터를 기반으로 했습니다. 그것은 정확히 최근의 것은 아니지만(당시 Bay Area에서 여전히 좋은 집을 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. AI 학습 모델 파이프라인

Problem와 question

- 이 장에서 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. AI 학습 모델 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 깨닫는다. 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. AI 학습 모델 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? 보자: 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.

1. AI 학습 모델 파이프라인

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. 수학식 2-1은 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. AI 학습 모델 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. AI 학습 모델 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

```
(base) C:\Users\DaeKyeong>cd C:\DEV\python-works\adsp

(base) C:\DEV\python-works\adsp>jupyter notebook
```

1. AI 학습 모델 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. AI 학습 모델 파이프라인

데이터 수집

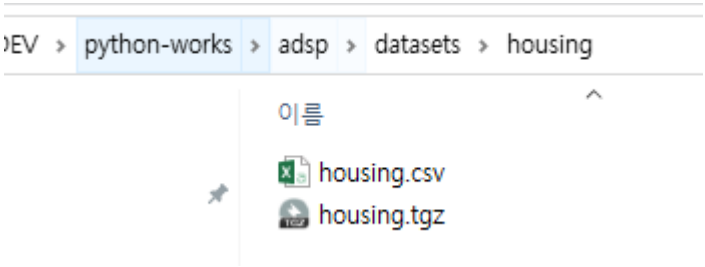
```
import os
import tarfile
from six.moves import urllib

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):
        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. AI 학습 모델 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\python-works\ad네>tree /f
폴더 PATH의 목록입니다.
볼륨 일련 번호는 2019-9371입니다.
C:.
│  library.py
│  프로토타입1.ipynb
│
├─.ipynb_checkpoints
│        프로토타입1-checkpoint.ipynb
└─datasets
    └─housing
            housing.csv
            housing.tgz


C:\DEV\python-works\adsp>
```

1. AI 학습 모델 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. AI 학습 모델 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. AI 학습 모델 파이프라인

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
```

1. AI 학습 모델 파이프라인

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	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

- 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.12 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. AI 학습 모델 파이프라인

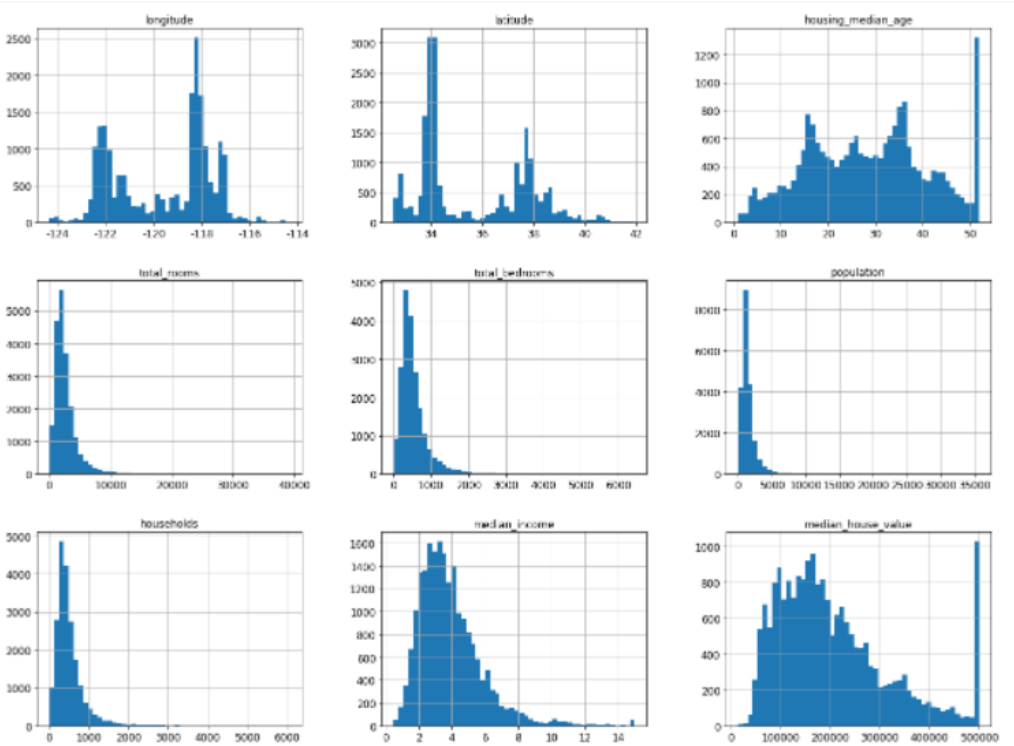
Insights 얻기

- 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.

1. AI 학습 모델 파이프라인

Insights 얻기

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



1. AI 학습 모델 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. AI 학습 모델 파이프라인

Insights 얻기

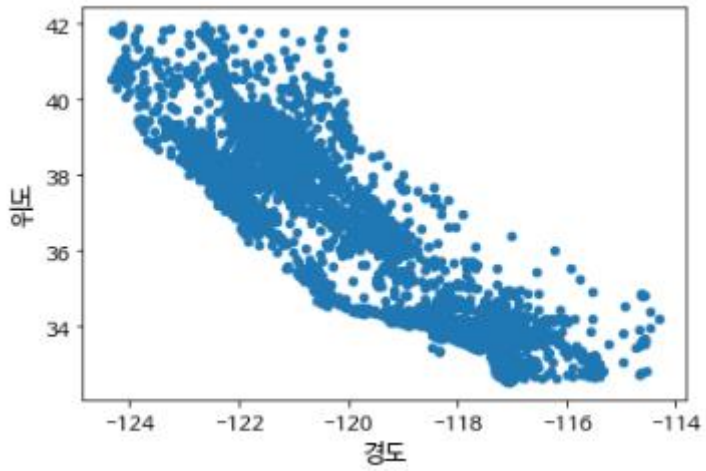
- 주택의 중위연령과 주택가격 중위도 상한선에 두었다. 후자는 대상 속성(레이블)이기 때문에 심각한 문제일 수 있습니다. 기계 학습 알고리즘은 가격이 한도를 초과하지 않는다는 것을 학습할 수 있습니다. 이것이 문제인지 아닌지 확인하려면 클라이언트 팀(시스템 출력을 사용할 팀)에 확인해야 합니다. $500,000를 넘어서도 정확한 예측이 필요하다고 말하면 주로 두 가지 옵션이 있습니다.라벨에 상한선이 있는 지역에 대해 적절한 라벨을 수집합니다.비. 교육 세트(및 테스트 세트에서도 해당 구역을 제거합니다. 시스템이 $500,000를 초과하는 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. AI 학습 모델 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')
```

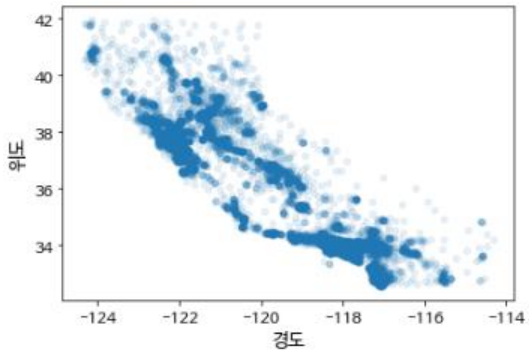


1. AI 학습 모델 파이프라인

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')
```



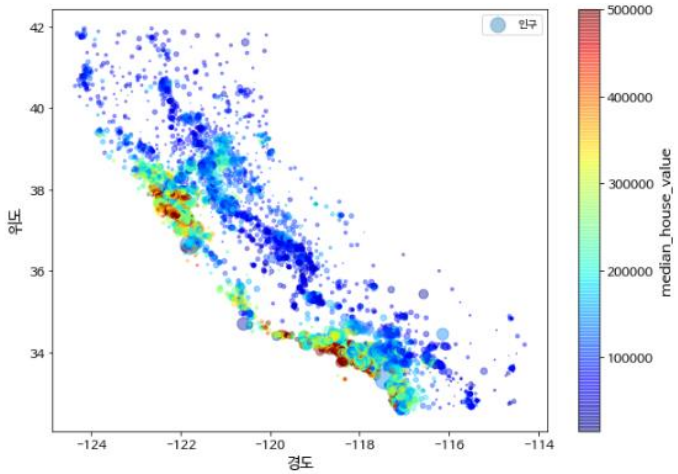
- 고밀도 지역, 즉 베이 지역과 로스앤젤레스와 샌디에이고 주변과 센트럴 밸리, 특히 새크라멘토와 프레즈노 주변에서 상당히 고밀도로 길게 늘어선 줄을 명확하게 볼 수 있습니다. 매개변수를 다양하게 조절해봐야 합니다.

1. AI 학습 모델 파이프라인

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.

```
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. AI 학습 모델 파이프라인

Insights 얻기

- 이 이미지는 주택 가격이 위치(예: 바다와 가까움)와 인구 밀도에 매우 밀접한 관련이 있음을 알려줍니다.
- 클러스터링 알고리즘을 사용하여 주요 클러스터를 감지하고 클러스터 중심에 대한 근접성을 측정하는 새로운 기능을 추가하는 것이 유용할 것입니다.

1. AI 학습 모델 파이프라인

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. AI 학습 모델 파이프라인

데이터 정제(Data Cleaning)

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

1. AI 학습 모델 파이프라인

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
```

1. AI 학습 모델 파이프라인

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
```

1. AI 학습 모델 파이프라인

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)
```

1. AI 학습 모델 파이프라인

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

1. AI 학습 모델 파이프라인

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

1. AI 학습 모델 파이프라인

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
```

1. AI 학습 모델 파이프라인

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')
```

1. AI 학습 모델 파이프라인

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 OneHotEncoder클래스를 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>
```

1. AI 학습 모델 파이프라인

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.]])
```

1. AI 학습 모델 파이프라인

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.]])
```

1. AI 학습 모델 파이프라인

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. AI 학습 모델 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. AI 학습 모델 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.

1. AI 학습 모델 파이프라인

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]
```

1. AI 학습 모델 파이프라인

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")
```

1. AI 학습 모델 파이프라인

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					

1. AI 학습 모델 파이프라인

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)
```

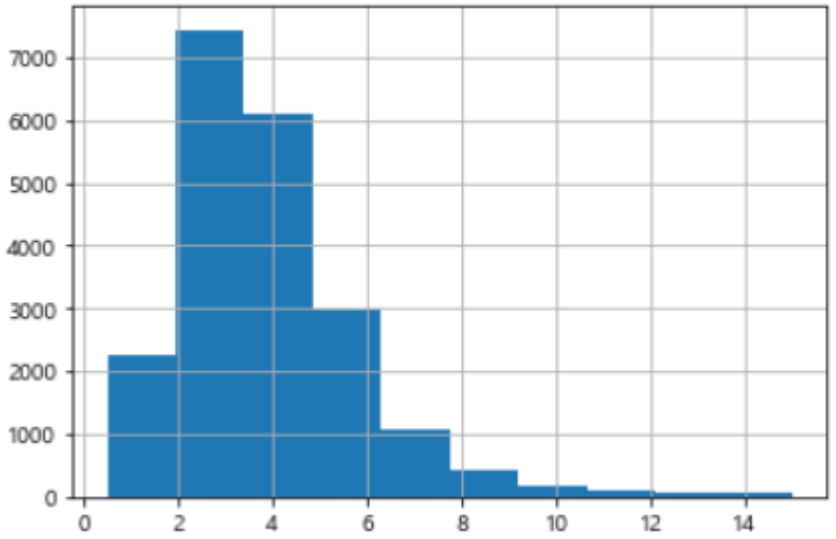
1. AI 학습 모델 파이프라인

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. AI 학습 모델 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. AI 학습 모델 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)
```

1. AI 학습 모델 파이프라인

Test Set 생성

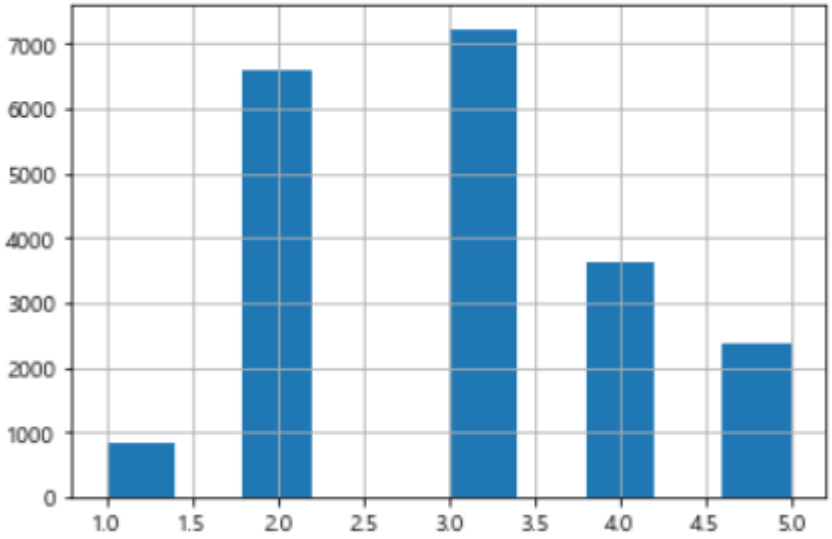
- 5 이상은 5로 레이블합니다.

```
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
```

1. AI 학습 모델 파이프라인

Test Set 생성

```
housing["income_cat"].hist()
```



1. AI 학습 모델 파이프라인

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
```

1. AI 학습 모델 파이프라인

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
```

1. AI 학습 모델 파이프라인

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["Ove
rall"] - 100
```

1. AI 학습 모델 파이프라인

Test Set 생성

- 계층화된 샘플링과 순수 무작위 샘플링의 샘플링 편향 비교

```
compare_props
>>
```

Overall	Stratified	Random	Rand. %error	Strat. %error	
income_cat					
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

1. AI 학습 모델 파이프라인

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()
```

1. AI 학습 모델 파이프라인

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()
```

1. AI 학습 모델 파이프라인

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. AI 학습 모델 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

1. AI 학습 모델 파이프라인

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)
```

1. AI 학습 모델 파이프라인

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. AI 학습 모델 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.

1. AI 학습 모델 파이프라인

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
```

1. AI 학습 모델 파이프라인

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'")
```

1. AI 학습 모델 파이프라인

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
```

1. AI 학습 모델 파이프라인

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)
```

1. AI 학습 모델 파이프라인

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
```

1. AI 학습 모델 파이프라인

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. AI 학습 모델 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),
    ])
```

1. AI 학습 모델 파이프라인

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.        ]])
```

1. AI 학습 모델 파이프라인

Transformation Pipelines

- 전체 파이프라인 실행

```
housing_prepared.shape
>>

(16512, 17)
```

1. AI 학습 모델 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.

2. Data pre-processing

Data pre-processing

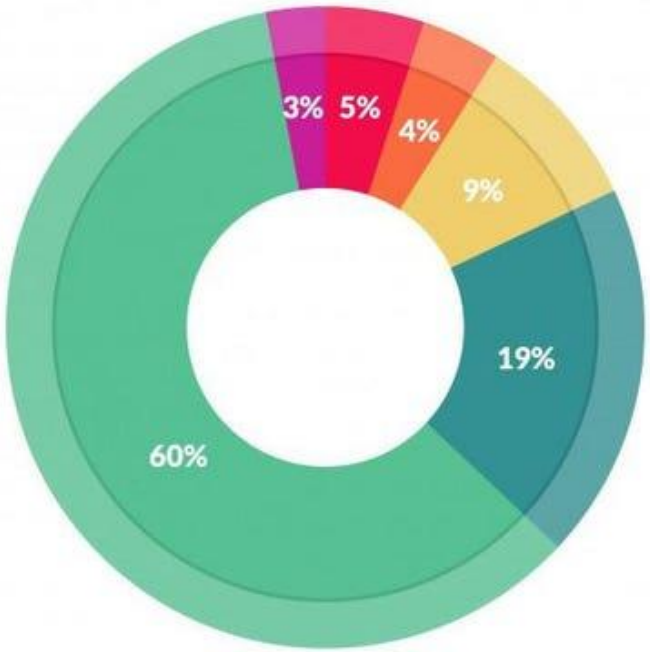
- **Data pre-processing?**
- Data pre-processing에 대한 논의는 플랫폼 제공업체인 CrowdFlower의 약 80명의 데이터 과학자를 대상으로 한 설문 조사에 대한 Gil Press의 “Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says: Mar 23, 2016 ”를 통해서이다.
- “데이터 준비는 데이터 과학자 작업의 약 80%를 차지합니다. 데이터 과학자는 데이터를 정리하고 구성하는 데 시간의 60%를 사용합니다. 데이터 세트 수집은 시간의 19%로 두 번째입니다. 즉, 데이터 과학자는 분석을 위한 데이터 준비 및 관리에 시간의 약 80%를 소비합니다.”

출처 : https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#16ece2467f75

2. Data pre-processing

Data pre-processing

- **Data pre-processing?**



What data scientists spend the most time doing

- *Building training sets: 3%*
- *Cleaning and organizing data: 60%*
- *Collecting data sets; 19%*
- *Mining data for patterns: 9%*
- *Refining algorithms: 4%*
- *Other: 5%*

출처 : https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#16ece2467f75

2. Data pre-processing

Data pre-processing

- Gil Press는 또한 같은 글에서 “In 2009, data scientist Mike Driscoll popularized the term “data munging,” describing the “painful process of cleaning, parsing, and proofing one’s data” as one of the three Gendery skills of data geeks.” 라고 말하며, Data pre-processing에 대한 통찰을 보여주고 있는 데, 그것을 정리하면 “data munging”이라 하며 그 내용은 다음과 같다.

- Data cleaning and organizing
- Data cleaning, parsing, and proofing

출처 : https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#16ece2467f75

2. Data pre-processing

Data pre-processing

● 그렇다면, “data munging”은 무엇인지 계속 살펴보면,

“

Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics…
The goal of data wrangling is to assure quality and useful data…Data wrangling typically follows a set of general steps which begin with extracting the data in a raw form from the data source, "munging" the raw data (e.g. sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and future use.

“

출처 : https://en.wikipedia.org/wiki/Data_wrangling

2. Data pre-processing

Data pre-processing

- **데이터 랭글링?**
- 데이터 랭글링 data wrangling은 원본 데이터를 정제하고 사용 가능한 형태로 구성하기 위한 변환 과정을 광범위하게 의미하는 비공식적인 용어. 데이터 랭글링은 데이터 전처리의 한 단계에 불과하지만 중요한 단계
- **Gather**
  데이터를 얻는 방법으로는 다운로드(Download), 웹 스크레이핑(Web scrapping), API(Application Programming Interface)가 있다.
- **Assess**
  얻은 데이터를 읽고 데이터가 깨끗한지 아닌지 판단하는 단계.
- **Clean**
  데이터를 정제하는 방법으로는 Define, Code, Test가 있습니다. 2단계(Assess)에서 발견된 데이터의 문제점을 보고 어떤 부분을 정제할지 정의하고(Define), 정제하기 위한 코드를 짜고(Code), 잘 정제가 되었는지 테스트를 해보는(Test) 것.
- **Reassess and Iterate**
  다시 2단계로 돌아가 데이터가 잘 정제되었는지 판단을 합니다. 추가로 정제해야 할 부분이 있다면 다시 2-3-4 단계를 반복.
- **Store (optional)**
  나중에 다시 사용하기 위해 저장하는 단계.

2. Data pre-processing

Data pre-processing

- 그렇다면, Data pre-processing 에 대한 정의를 좀 더 살펴보자.

"… manipulation or dropping of data… Examples of data preprocessing include cleaning, instance selection, normalization, one hot encoding, transformation, feature extraction and selection, etc. The product of data preprocessing is the final training set.

…

Tasks of data preprocessing

- Data cleansing
- Data editing
- Data reduction
- Data wrangling"

출처 : https://en.wikipedia.org/wiki/Data_pre-processing

2. Data pre-processing

Data pre-processing

- 다음 예를 보자,

예	설명

		Gender	Pregnant
Adult	1	Male	No
	2	Female	Yes
	3	~~Male~~	~~Yes~~
	4	Female	No
	5	~~Male~~	~~Yes~~

성별이 남성 또는 여성이고 임신 여부를 가진 5명의 불가능한 데이터 조합의 성인 데이터 셋

출처 : https://en.wikipedia.org/wiki/Data_pre-processing

2. Data pre-processing

Data pre-processing

- 다음 예를 보자,

Data cleansing				설명
		Gender	Pregnant	• 데이터세트에 존재하는 이러한 데이터가 사용자 입력 오류 또는 데이터 손상으로 인해 발생한 것으로 판단할 수 있기 때문에 이러한 데이터를 제거. • 이러한 데이터를 삭제해야 하는 이유는 불가능한 데이터가 데이터 마이닝 프로세스의 후반 단계에서 계산 또는 데이터 조작 프로세스에 영향을 미치기 때문.
Adult	1	Male	No	
	2	Female	Yes	
	4	Female	No	

~~Male~~	~~Yes~~

출처 : https://en.wikipedia.org/wiki/Data_pre-processing

2. Data pre-processing

Data pre-processing

- 다음 예를 보자,

Data editing	설명

		Gender	Pregnant
Adult	1	Male	No
	2	Female	Yes
	3	**Female**	**Yes**
	4	Female	No
	5	**Female**	**Yes**

~~**Male**~~	~~**Yes**~~

- 성인이 여성이라고 가정하고 그에 따라 변경할 수 있다.
- 데이터 마이닝 프로세스의 후반 단계에서 데이터 조작을 수행할 때 데이터를 보다 명확하게 분석할 수 있도록 데이터 세트를 편집

출처 : https://en.wikipedia.org/wiki/Data_pre-processing

2. Data pre-processing

Data pre-processing

- 다음 예를 보자,

Data reduction			설명
		Gender	Pregnant
Adult	2	Female	Yes
	4	Female	No
	1	Male	No
	3	Male	Yes
	5	Male	Yes

- 데이터를 성별로 정렬.
- 이를 통해 데이터 세트를 단순화하고 더 집중하고 싶은 성별을 선택할 수 있다.

출처 : https://en.wikipedia.org/wiki/Data_pre-processing

2. Data pre-processing

Data pre-processing

● Data pre-processing 에 대한 또 다른 예를 보자,

“

- 데이터 결합 (예: 행 결합, 열 결합, JOIN)
- 데이터 분할, 필터링, 샘플링
- 파생변수 생성 (예: 날짜 → 주말/평일 구분, 점수 → 등급)
- 더미변수 생성 (원-핫 인코딩, 예: 성별 → 0/1)
- 결측치 처리 (제거·보간)
- 이상치 처리 (제거·보간)
- 스케일 조정 (예: MixMax → 0~1, 표준점수, 로그스케일)
- 자료형 변경 (예: String → Datetime, String → Integer)
- 기타 데이터 수정·보정

”

출처 : https://zetawiki.com/wiki/데이터_전처리

2. Data pre-processing

Data pre-processing

- 마지막으로 NAVER 지식백과를 통해, Data pre-processing 에 대한 정의를 살펴보면

“

원자료(raw data)를 데이터 분석 목적과 방법에 맞는 형태로 처리하기 위하여 불필요한 정보를 분리 제거하고 가공하기 위한 예비적인 조작…데이터의 측정 오류를 줄이고 잡음(noise), 왜곡, 편차를 최소화한다. 정밀도, 정확도, 이상 값(outlier), 결측 값(missing value), 모순, 불일치, 중복 등의 문제를 해결하기 위한 방법으로 다음 전처리 방법들을 사용한다.

- 데이터 정제(cleansing): 결측 값(missing value; 빠진 데이터)들을 채워 넣고, 이상치를 식별 또는 제거하고, 잡음 섞인 데이터를 평활화(smoothing)하여 데이터의 불일치성을 교정하는 기술
- 데이터 변환(transformation): 데이터 유형 변환 등 데이터 분석이 쉬운 형태로 변환하는 기술. 정규화(normalization), 집합화(aggregation), 요약(summarization), 계층 생성 등의 방법을 활용한다.
- 데이터 필터링(filtering): 오류 발견, 보정, 삭제 및 중복성 확인 등의 과정을 통해 데이터의 품질을 향상하는 기술
- 데이터 통합(integration): 데이터 분석이 용이하도록 유사 데이터 및 연계가 필요한 데이터들을 통합하는 기술
- 데이터 축소(reduction): 분석 시간을 단축할 수 있도록 데이터 분석에 활용되지 않는 항목 등을 제거하는 기술

”

출처 : https://terms.naver.com/entry.naver?docId=850102&cid=50333&categoryId=50333

2. Data pre-processing

Data pre-processing

- **이상의 내용을 통해, Data pre-processing 작업을 요약해 보면 다음과 같다.**

① Data cleaning and organizing
② Data cleaning, parsing, and proofing
③ the raw data (e.g. sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and future use
④ Data cleansing
⑤ Data editing
⑥ Data reduction
⑦ Data wrangling
⑧ 데이터 정제(cleansing)
⑨ 데이터 변환(transformation)
⑩ 데이터 필터링(filtering)
⑪ 데이터 통합(integration)
⑫ 데이터 축소(reduction)

① 데이터 결합 (예: 행 결합, 열 결합, JOIN)
② 데이터 분할, 필터링, 샘플링
③ 파생변수 생성 (예: 날짜 → 주말/평일 구분, 점수 → 등급)
④ 더미변수 생성 (원-핫 인코딩, 예: 성별 → 0/1)
⑤ 결측치 처리 (제거.보간)
⑥ 이상치 처리 (제거.보간)
⑦ 스케일 조정 (예: MixMax → 0~1, 표준점수, 로그스케일)
⑧ 자료형 변경 (예: String → Datetime, String → Integer)
⑨ 기타 데이터 수정.보정

2. Data pre-processing

ML Pipeline

- **Hidden Technical Debt in Machine Learning Systems**

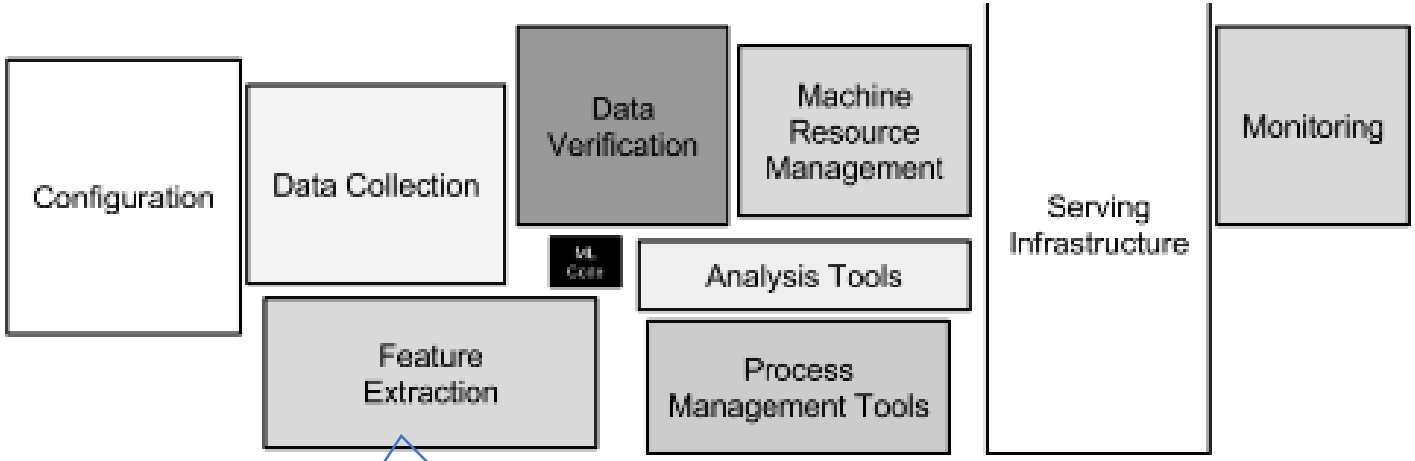


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

① Brainstorming or testing features
② Deciding what features to create
③ Creating features
④ Testing the impact of the identified features on the task
⑤ Improving your features if needed
⑥ Repeat

출처 : https://proceedings.neurips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf

2. Data pre-processing

방법론과 ML Pipeline, 그리고 Data pre-processing

- 각 방법론과 ML Pipeline은 Data pre-processing과 Data acquisition, Data Understanding, Transformation을 구분 속에서 이해하고 있다.

Feature engineering과 Data pre-processing

Feature engineering or feature extraction or feature discovery is the process of using domain knowledge to extract features (characteristics, properties, attributes) from raw data.

Typical engineered features

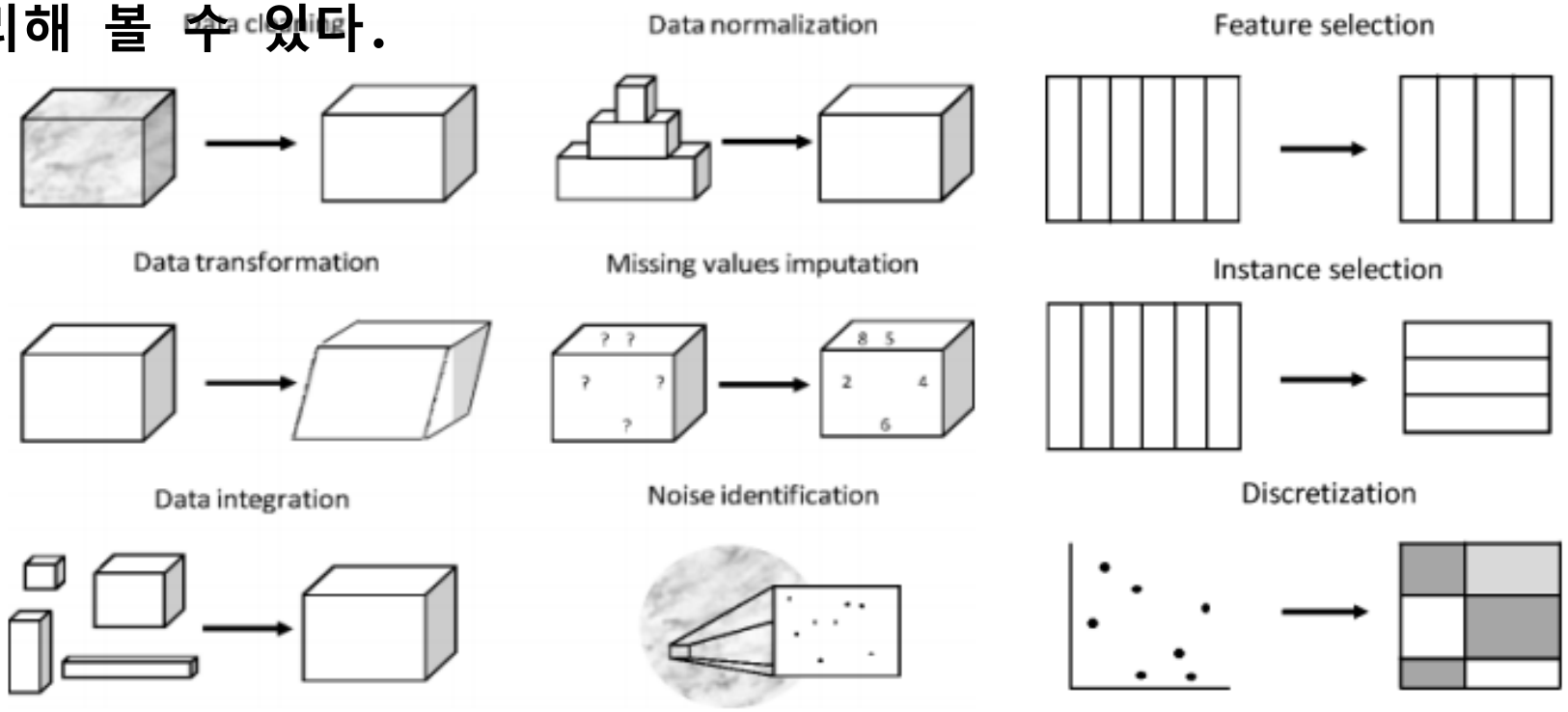
- Numerical transformations (like taking fractions or scaling)
- Category encoder like one-hot or target encoder (for categorical data)
- Clustering
- Group aggregated values
- Principal component analysis (for numerical data)
- Feature construction

출처 : https://en.wikipedia.org/wiki/Feature_engineering

2. Data pre-processing

Feature engineering과 Data pre-processing

- 도메인 지식을 활용하여 특징(Feature)를 만들어내는 과정으로서의 Feature engineering은 데이터 관찰과 Data pre-processing 을 전제로 한다.
- 따라서 Data pre-processing 은 다음 그림처럼 몇 가지 기법으로 정리해 볼 수 있다.



출처 : Big data preprocessing: methods and prospects | Big Data Analytics | Full Text (biomedcentral.com)

『3과목』 AI와 Data pre-processing

Data Structures

Syllabus

학습목표

- 이 워크샵에서는 Pandas의 Series와 DataFrame을 사용할 수 있다.

눈높이 체크

- Pandas를 알고 계신가요?
- Series와 DataFrame을 알고 계신가요?

1. Pandas

라이브러리 준비

- 데이터 처리와 분석을 위한 파이썬 라이브러리.
- R의 data.frame을 본떠서 설계한 DataFrame이라는 데이터 구조를 기반으로 만들어졌다.
- NumPy가 파이썬에서 수치연산을 잘 할 수 있도록 지원하는 라이브러리인데 이를 기반으로 tabular 데이터(표 형태의 정형데이터)를 보다 편하게 다룰 수 있도록 하는 라이브러리가 Pandas이다. 특히 Pandas를 사용하는 시점부터 파일 입출력(read, write)이 매우 편해지고 데이터 필터링, 그래프, 결측치 처리, 시계열 분석, 스타일링 등 슬슬 엑셀을 대체할 수 있는 기능을 맛보게 된다.

- 다음 명령으로 설치한다.

```
pip install pandas
```

- 임포트할 때는 보통 pd라는 별명으로 임포트한다.

```
import pandas as pd

pd.__version__
```

1. Pandas

Data Structures

- 수집, 저장, 데이터 셋 적재 Gathering Data 후 DataFrame에 담을 수 있어야 한다. 대부분의 데이터는 시계열(series)이나 표(table)의 형태로 나타낼 수 있다.
- 데이터 랭글링에 사용되는 가장 일반적인 데이터 구조는 데이터프레임이다. 사용하기 쉽고 기능이 많음. 판다스(Pandas) 패키지는 이러한 데이터를 다루기 위한 시리즈(Series) 클래스와 데이터프레임(DataFrame) 클래스를 제공한다.
- 데이터프레임은 표 형식 데이터로서 PANDAS를 통해 사용할 수 있다.

1. Pandas

Data Structures

- PANDAS를 이해하기위한 핵심 중 하나는 데이터 모델을 이해하는 것입니다. Pandas의 핵심에는 세 가지 데이터 구조가 있다.

DATA STRUCTURE	DIMENSIONALITY	SPREADSHEET ANALOG
Series	1D	Column
DataFrame	2D	Single Sheet
Panel	3D	Multiple Sheets

- 가장 널리 사용되는 데이터 구조는 각각 배열 데이터와 테이블 형식 데이터를 처리하는 Series 및 DataFrame. 스프레드 시트 세계와의 비유는 이러한 유형 간의 기본적인 차이점을 보여준다. DataFrame은 행과 열이있는 시트와 비슷하지만 Series는 데이터의 단일 열과 비슷합니다. 패널은 시트 그룹. 마찬가지로, 팬더에서 패널은 여러 데이터 프레임을 가질 수 있으며, 각 프레임은 차례로 여러 시리즈를 가질 수 있다.

2. Series

개요

- Pandas 라이브러리의 기본 객체이자 1차원 객체인 시리즈(Series) 객체는 1차원 배열과 유사하지만, 각각의 원소에 라벨을 붙일 수 있다. 이 라벨을 통해 인덱싱(indexing)이 가능하며, 인덱싱을 통해 원소에 접근할 수 있다. 그리고 NumPy의 어레이 객체보다 강력하고 다양한 메서드를 지원하며 심지어 간단한 그래프도 쉽게 그릴 수 있다.
- 시리즈는 인덱스와 값으로 이루어져 있으며 각 요소는 .index와 .values 어트리뷰트로 접근할 수 있다.특징
- 레이블 또는 데이터의 위치를 지정한 추출 가능.
- 산술 연산 가능.

Index

Artist	Data
0	145
1	142
2	38
3	13

value

2. Series

생성

- 시리즈 객체의 생성은 Series() 함수에 주로 리스트나 NumPy 어레이 객체를 입력으로 넣어 할 수 있으며 필요시 인덱스를 각 원소의 개수만큼 할당할 수 있다.
- 비어있는 시리즈 객체를 생성할 경우 함수 내부에 아무것도 쓰지 않는다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.Series()
결과값1	Series([], dtype: object)
비고	

2. Series

생성

- 리스트와 NumPy 어레이를 사용하여 시리즈 객체를 생성하는 예제이다. 리스트도 중첩이 아니고 어레이도 1차원을 입력으로 하는 것을 알 수 있다. 기본 객체나 NumPy 객체와는 다르게 인덱스가 직접적으로 표기된 것을 알 수 있다. 이것이 Pandas 객체가 다른 객체와 차별화 되는 요소라고 할 수 있겠다. Pandas 객체의 경우 인덱스를 이렇게 확인할 수 있고 원하는 인덱스로 바꿀 수도 있으며 인덱스기반의 다양한 연산 또한 가능하다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.Series([100, 200, 300]) pd.Series(np.array([100, 200, 300]))
결과값1	0 100 1 200 2 300 dtype: int64
비고	

2. Series

생성

- 인덱스를 지정하는 시리즈 객체 생성은 다음과 같다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.Series([100, 200, 300], index = ["A", "B", "C"])
결과값1	A 100 B 200 C 300 dtype: int64
비고	

2. Series

생성

- 인덱스를 지정하여 시리즈를 생성하고자 한다면 딕셔너리 객체를 사용할수도 있다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.Series({"A": 100, "B": 200, "C": 300})
결과값1	A 100 B 200 C 300 dtype: int64
비고	

2. Series

Series CRUD

- 시리즈 객체의 인덱싱은 크게 인덱서(indexer)를 사용하는 것과 사용하지 않는 것이 있다. 색인은 목록을 사용하여 두 번째 매개 변수로 지정.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	george_dupe = pd.Series([10, 7, 1, 22],index=['1968', '1969', '1970', '1971'],name='George Songs') george_dupe
결과값1	1968 10 1969 7 1970 1 1971 22 Name: George Songs, dtype: int64
비고	

Series CRUD

● Series CRUD

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	george_dupe.index george_dupe.values george_dupe.value_counts
결과값1	Index(['1968', '1969', '1970', '1971'], dtype='object') array([10, 7, 1, 22], dtype=int64) <bound method IndexOpsMixin.value_counts of 1968 10 1969 7 1970 1 1971 22 Name: George Songs, dtype: int64>
비고	

2. Series

Series CRUD

- Reading

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	george_dupe['1968'] george_dupe != 1 george_dupe.loc[george_dupe != 1]
결과값1	10 1968 True 1969 True 1970 False 1971 True Name: George Songs, dtype: bool 1968 10 1969 7 1971 22 Name: George Songs, dtype: int64
비고	

2. Series

Series CRUD

- Updating

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	george_dupe['1969'] = 6 george_dupe['1969']
결과값1	6
비고	

- Deletion

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	del george_dupe['1971'] george_dupe
결과값1	1968 10 1969 6 1970 1 Name: George Songs, dtype: int64
비고	

2. Series

인덱싱

- 인덱서는 순수하게 정수만 사용가능한 .iloc[]와 다양한 입력을 받는 .loc[] 두 가지가 있다.

METHOD	WHEN TO USE
속성 액세스	이름이 유효한 경우 단일 색인 이름에 대한 값 가져 오기속성 이름.
인덱스액세스	이름이 아닌 경우 단일 인덱스 이름에 대한 값 가져 오기 / 설정유효한 속성 이름.
.iloc	인덱스 위치 또는 위치별로 값 가져 오기 / 설정. (반 개방슬라이스 간격)
.loc	색인 레이블로 값 가져 오기 / 설정. (슬라이스 폐쇄 간격)
.iat	인덱스 위치별로 numpy 배열 결과 가져 오기 / 설정.
.at	인덱스 레이블로 numpy 배열 결과 가져 오기 / 설정

- loc 메소드는 특정 범위의 데이터를 인덱싱하는데 사용할 수 있다. 반면 at 메소드는 딱 하나의 데이터를 인덱싱할 때 사용한다. i는 integer의 첫글자를 따서 붙여진 것이다. 인덱스와 컬럼명을 사용하지 않고 대신 행과 컬럼의 위치에 대해서 정수를 사용하고 싶으면 iloc과 iat을 사용하면 된다.

2. Series

인덱싱

- **Indexing Summary**

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	import pandas as pd df = pd.DataFrame([[95, 92, 88], [84, 67, 88], [91, 99, 68], [87, 79, 81], [77, 92, 85]], index=['A', 'B', 'C', 'D', 'E'], columns=['math', 'english', 'history'],) df
결과값1	math english history A 95 92 88 B 84 67 88 C 91 99 68 D 87 79 81 E 77 92 85
비고	

2. Series

인덱싱

- **Indexing Summary**

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	print(df.loc[['A', 'B', 'C'], 'english']) print(df.loc[['A', 'B', 'C'], ['english', 'history']]) print(df.at['C', 'english']) print(df.iloc[1:3, 0:2]) print(df.iat[4, 0])
결과값1	A 92 B 67 C 99 Name: english, dtype: int64
결과값2	english history A 92 88 B 67 88 C 99 68
결과값3	99
결과값4	math english B 84 67 C 91 99
결과값5	77
비고	

2. Series

Question 1

다음 요구대로 시리즈 요소에 접근해 보시오.

```
george_dupe = pd.Series([10, 7, 1, 22],index=['1968', '1969', '1970',
'1971'],name='George Songs')
george_dupe
>>
1968    10
1969     7
1970     1
1971    22
Name: George Songs, dtype: int64
```

1. 첫 번째 요소에 접근합니다. 결과는 10.
2. loc를 사용하여 색인 이름을 기반으로 "1968" 연도의 요소에 접근. 결과는 10.
3. iloc를 사용하여 위치 기반으로 0부터 2까지(3미만)의 범위에 해당하는 요소들을 가져옵니다.
4. loc를 사용하여 색인 이름으로 "1968"부터 "1970"까지의 범위에 해당하는 요소들을 가져옵니다.

2. Series

Answer 1

```
print(george_dupe.iloc[0])

print(george_dupe.loc["1968"])

print(george_dupe.iloc[0:3])

print(george_dupe.loc["1968":"1970"])
>>
10

10

1968    10
1969     6
1970     1
Name: George Songs, dtype: int64

1968    10
1969     7
1970     1
Name: George Songs, dtype: int64
```

2. Series

인덱싱

- george_dupe.at['1970']와 george_dupe.loc['1970'] 모두 Pandas Series에서 특정 레이블(인덱스)에 해당하는 값을 가져오는데 사용됩니다. 하지만 두 방법에는 약간의 차이가 있습니다.

george_dupe.at['1970']

- 위 코드는 at 메서드를 사용하여 레이블(인덱스)이 "1970"인 요소에 접근합니다. 따라서 결과는 해당 요소의 값인 1이 됩니다.

george_dupe.loc['1970']

- 이 코드는 loc 메서드를 사용하여 레이블(인덱스)이 "1970"인 요소에 접근합니다. loc 메서드는 슬라이싱과 유사한 방식으로 레이블을 기반으로 데이터를 가져옵니다. 여기서는 "1970"이라는 레이블을 가진 요소를 반환하며, 결과는 1이 됩니다.

- 따라서 george_dupe.at['1970']와 george_dupe.loc['1970'] 모두 같은 값을 반환하며, 이 경우에는 시리즈에서 '1970'이라는 인덱스에 해당하는 값인 1을 가져옵니다.

2. Series

Series Slicing

- Series Slicing(시리즈 슬라이싱)은 Pandas 라이브러리에서 Series 객체의 일부를 선택하는 것을 의미합니다. 시리즈 슬라이싱은 인덱스를 기반으로 하여 특정 범위의 데이터를 추출하는 방법을 제공합니다.

SLICE	RESULT
0:1	First item
:1	First item (start default is 0)
:-2	처음부터 두 번째 항목부터 마지막 항목까지
::2	다른 모든 항목을 건너 뛰고 처음부터 끝까지 가져옵니다.

2. Series

Series Slicing

- 예를 들어, 다음과 같이 Series를 생성하고 슬라이싱하는 방법을 살펴보겠습니다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	george_dupe.iloc[0:2]
결과값1	1968 10 1969 7 Name: George Songs, dtype: int64
비고	

2. Series

Question 1

```
names = ['민준', '서연', '현우', '민서', '동현', '수빈']
sdata = pd.Series(names)
S1 = sdata[3:6]
print(S1.values)
print(S1)
```

의 결과는

```
['민서' '동현' '수빈']
3    민서
4    동현
5    수빈
dtype: object
```

이다. S1에서 '동현'을 출력하는 코드로 맞는 것은?

① print(S1[1])
② print(S1[4])

2. Series

Answer 1

② print(S1[4])

```
#일반적인 인덱스 사용 시
b = S1[1:2]
print(b)
>>
4    동현
```

2. Series

Boolean Arrays

- Boolean Arrays(불리언 배열)는 조건을 충족하는지 여부를 나타내기 위해 True 또는 False 값으로 이루어진 배열을 말합니다. Pandas에서는 불리언 배열을 사용하여 데이터를 필터링하거나 특정 조건에 따라 원하는 데이터를 선택하는 데 유용하게 활용됩니다.
- 예를 들어, 다음은 불리언 배열을 사용하여 데이터를 필터링하는 간단한 예제입니다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	mask = george_dupe > 7 mask
결과값1	1968 True 1969 False 1970 False 1970 True Name: George Songs, dtype: bool
비고	

2. Series

Iteration

- 두 개의 Pandas Series를 생성하고, 첫 번째 Series인 songs_66의 각 값을 반복하며 출력하는 반복문입니다. 반복문에서 for value in songs_66:은 Series의 각 값을 차례로 반복하며, 해당 값들을 출력합니다. 그러나 이 코드는 None(결측치) 값을 가진 항목이 있기 때문에 에러가 발생할 수 있습니다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	songs_66 = pd.Series([3, None , 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts') for value in songs_66: print(value)
결과값1	3.0 nan 11.0 9.0
비고	

2. Series

Question 1

Pandas에서는 None(결측치) 대신에 NaN(Not a Number)을 사용하므로, None 값이 있는 Series를 반복할 때 에러를 피하려면 None을 NaN으로 변경하거나, None을 건너뛰도록 처리해야 합니다. songs_66 Series에서 None 값을 NaN으로 변경하여 반복하시오.

Answer 1

```
import pandas as pd
import numpy as np

# 두 개의 Series 생성
songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts')
songs_66 = songs_66.replace({None: np.nan})  # None 값을 NaN으로 변경

# 반복문을 통해 songs_66의 각 값을 출력
for value in songs_66:
    print(value)

>>
3.0
nan
11.0
9.0
```

2. Series

Question 2

None 값이 있는 경우 해당 값이 None이면 출력하지 않도록 처리하시오.dropna() 메서드는 None이나 NaN 값을 제거하여 출력합니다.

Answer 2

```
import pandas as pd

# 두 개의 Series 생성
songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts')

# 반복문을 통해 songs_66의 각 값을 출력
for value in songs_66.dropna():
    print(value)

>>
3.0
11.0
9.0
```

2. Series

Overloaded operations

- 두 개의 Pandas Series를 생성하고, 두 시리즈를 더한 후에 그 결과를 확인. songs_66와 songs_69 두 시리즈의 각 요소를 동일한 인덱스끼리 더하여 새로운 Series를 만듭니다. 그러나 이 덧셈 연산에서 인덱스가 서로 다르게 배치되어 있기 때문에, 같은 인덱스를 가진 요소들끼리는 더해지고, 그렇지 않은 요소는 NaN(결측치)로 처리됩니다.

파일	소스코드
실습환경	준비 Tf38_cpu

소스코드	import pandas as pd
	
	# 두 개의 Series 생성
	songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'],
	name='Counts')
	songs_69 = pd.Series([18, 22, 7, 5], index=['John', 'Paul', 'George', 'Ringo'], name='Counts')
	
	# 두 Series를 더한 결과
	result = songs_66 + songs_69
	print(result)

결과값1	George 10.0 John 29.0 Paul 31.0 Ringo NaN Name: Counts, dtype: float64
비고	

2. Series

Reset Index

- reset_index() 메서드는 Pandas Series나 DataFrame의 인덱스를 기본 숫자형 인덱스로 재설정하는 데 사용됩니다. 그러나 이 메서드는 호출된 Series나 DataFrame 자체를 변경하는 것이 아니라, 새로운 변경된 결과를 반환합니다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts') songs_66.reset_index()
결과값1	index Counts 0 George 3.0 1 Ringo NaN 2 John 11.0 3 Paul 9.0
비고	

2. Series

Reset Index

- songs_66.reset_index()를 호출했을 때, songs_66 Series의 인덱스는 변경되지 않고 그대로 유지됩니다. reset_index() 메서드를 사용하여 새로운 DataFrame 또는 Series를 생성하려면 반환된 결과를 새로운 변수에 할당해야 합니다. 예를 들어, reset_index()를 사용하여 새로운 DataFrame을 만들고 결과를 확인하는 방법은 다음과 같습니다.

```
import pandas as pd

songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts')
reset_songs_66 = songs_66.reset_index()

print(reset_songs_66)
>>
index  Counts
0  George     3.0
1   Ringo     NaN
2    John    11.0
3    Paul     9.0
```

- 이렇게 하면 reset_songs_66은 기본 숫자형 인덱스로 재설정된 새로운 DataFrame 또는 Series가 됩니다. 결과를 출력하면 기존의 인덱스가 새로운 열로 추가된 것을 확인할 수 있습니다.

2. Series

Series Methods

- Pandas Series에는 다양한 메서드가 내장되어 있어 데이터를 조작하고 분석하는 데 유용합니다. 이러한 메서드들은 데이터를 처리하고 조작하는 데 있어서 편리한 기능을 제공합니다.
- 아래는 Pandas Series에서 자주 사용되는 메서드 몇 가지입니다.

- head() 및 tail(): Series의 처음 또는 끝 부분 일부를 보여줍니다.

```
series.head()  # 처음 부분을 출력
series.tail()  # 끝 부분을 출력
```

- describe(): 요약 통계 정보를 제공합니다.

```
series.describe()
```

- value_counts(): 각 값의 빈도수를 세어줍니다.

```
series.value_counts()
```

- unique() 및 nunique(): 고유한 값의 배열을 반환하거나 고유한 값의 개수를 반환합니다.

```
series.unique()   # 고유한 값들의 배열 반환
series.nunique()  # 고유한 값의 개수 반환
```

- sort_values(): 값에 따라 Series를 정렬합니다.

```
series.sort_values()  # 값에 따라 정렬
```

2. Series

Series Methods

- fillna(): 결측값을 다른 값으로 채웁니다.

```
series.fillna(value)  # 결측값을 지정한 값으로 채움
```

- isnull() 및 notnull(): 각 요소가 결측값인지 아닌지를 확인합니다.

```
series.isnull()   # 결측값이면 True, 아니면 False 반환
series.notnull()  # 결측값이 아니면 True, 아니면 False 반환
```

- apply(): 함수를 Series의 각 요소에 적용합니다.

```
series.apply(func)  # 지정한 함수를 각 요소에 적용
```

- map(): 딕셔너리와 같은 매핑을 사용하여 Series 값을 변환합니다.

```
series.map(mapping_dict)  # 딕셔너리를 사용하여 Series 값을 변환
```

- astype(): Series의 데이터 타입을 변경합니다.

```
series.astype(new_dtype)  # 데이터 타입을 지정한 타입으로 변경
```

- 이 외에도 Pandas Series는 다양한 유용한 메서드와 기능을 제공합니다. 이러한 메서드들은 데이터 조작과 분석에 유용하며, 데이터를 더 쉽게 처리할 수 있도록 도와줍니다.

2. Series

Series Methods

- Statistics

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	songs_66 = pd.Series([3, None , 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts') songs_66 songs_66.sum(0) songs_66.mean() songs_66.median()
결과값1	George 3.0 Ringo NaN John 11.0 Paul 9.0 Name: Counts, dtype: float64 23.0 In [ ]: 7.666666666666667 9.0
비고	

2. Series

Series Methods

- **Statistics**

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	songs_66.quantile() songs_66.quantile(.1) songs_66.quantile(.9) songs_66.describe()
결과값1	9.0 4.2 10.6 count 3.000000 mean 7.666667 std 4.163332 min 3.000000 25% 6.000000 50% 9.000000 75% 10.000000 max 11.000000 Name: Counts, dtype: float64
비고	

2. Series

Series Methods

- Statistics

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	songs_66.min() songs_66.idxmin() songs_66.max() songs_66.idxmax() songs_66.var() songs_66.std()
결과값1	3.0 'George‘ 11.0 'John‘ 17.333333333333336 4.163331998932266
비고	

3. DataFrame

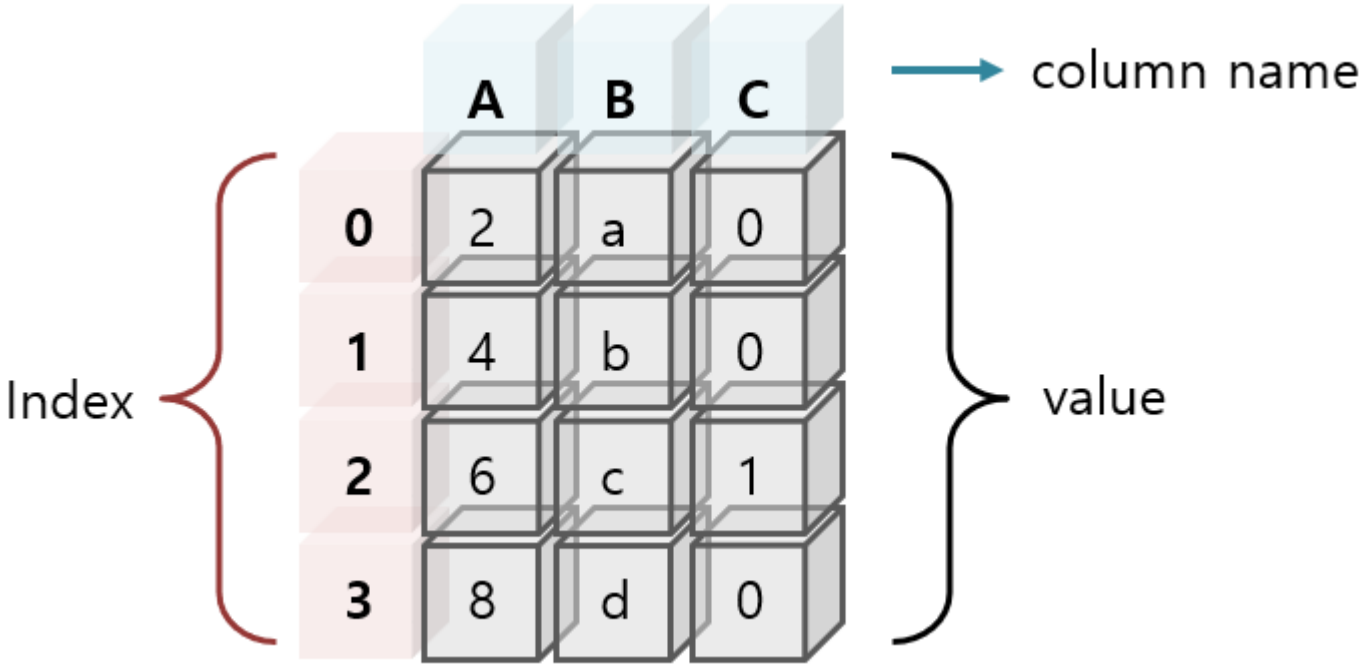
개요

- Pandas 라이브러리의 기본 객체이자 2차원 객체인 데이터프레임(DataFrame) 객체는 2차원 배열과 유사하지만, 각각 열과 행에 라벨을 붙일 수 있다. 이 라벨을 통해 인덱싱(indexing)이 가능하며, 인덱싱을 통해 원소에 접근할 수 있다. 그리고 NumPy 의 어레이 객체보다 강력하고 다양한 메서드를 지원하며 심지어 간단한 그래프도 쉽게 그릴 수 있다. 그리고 시리즈(Series) 객체를 여러개 이어붙인 것이 데이터프레임이라고 할 수 있다.
- 데이터프레임은 엄밀히 말하면 인덱스가 행과 열에 각각 있는데 일반적으로 데이터프레임에서 인덱스라고 하면 행(row)의 인덱스를 뜻하며 열의 인덱스는 변수명(column name)으로 지칭한다. 그리고 인덱스/변수명/값 각 요소는 .index/.columns/.values 어트리뷰트로 접근할 수 있다.

3. DataFrame

개요

- Pandas 데이터프레임 구조



3. DataFrame

생성

- 데이터프레임 객체의 생성은 DataFrame() 함수에 리스트나 NumPy 어레이 객체를 입력으로 넣어 생성할 수 있으며 딕셔너리를 사용하여 생성하는 경우도 많다. 비어있는 데이터프레임 객체를 생성할 경우 함수 내부에 아무것도 쓰지 않는다.데이터프레임 생성 후 개별적으로 각 열 정의

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	import pandas as pd pd.DataFrame()
결과값1	
비고	

3. DataFrame

생성

- 데이터 프레임 생성 후 개별적으로 각 열 정의. 빈 데이터프레임을 만들고 'Name', 'Age', 'Driver' 열을 추가하여 데이터프레임을 확인합니다.

파일	소스코드
실습환경	준비 Tf38_cpu

소스코드	# 라이브러리를 임포트합니다.
	import pandas as pd
	
	# 데이터프레임을 만듭니다.
	dataframe = pd.DataFrame()
	
	# 열을 추가합니다.
	dataframe['Name'] = ['Jacky Jackson', 'Steven Stevenson']
	dataframe['Age'] = [38, 25]
	dataframe['Driver'] = [True, False]
	
	# 데이터프레임을 확인합니다.
	dataframe

결과값1	Name Age Driver 0 Jacky Jackson 38 True 1 Steven Stevenson 25 False
비고	

3. DataFrame

생성

● 리스트 기반 생성

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.DataFrame([[1, 2, 3]])
결과값1	N0 1 2 0 1 2 3
비고	

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.DataFrame([[1, 2], [3, 4]])
결과값1	0 1 0 1 2 1 3 4
비고	

3. DataFrame

생성

- 리스트 기반 생성
  - 원본 리스트 전달로 생성

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	import numpy as np import pandas as pd data = [['Jacky Jackson', 38, True], ['Steven Stevenson', 25, False]] pd.DataFrame(data, columns=['Name', 'Age', 'Driver'])
결과값1	Name Age Driver 0 Jacky Jackson 38 True 1 Steven Stevenson 25 False
비고	

3. DataFrame

생성

- NumPy 어레이 기반 생성

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	import numpy as np import pandas as pd data = [['Jacky Jackson', 38, True], ['Steven Stevenson', 25, False]] matrix = np.array(data) pd.DataFrame(matrix, columns=['Name', 'Age', 'Driver'])
결과값1	Name Age Driver 0 Jacky Jackson 38 True 1 Steven Stevenson 25 False
비고	

3. DataFrame

생성

- 딕셔너리 기반 생성
- 딕셔너리의 키는 데이터프레임 객체의 변수명으로 된다.열 이름과 매핑한 딕셔너리 사용 생성

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	data = {'Name': ['Jacky Jackson', 'Steven Stevenson'], 'Age': [38, 25], 'Driver': [True, False]} pd.DataFrame(data)
결과값1	Name Age Driver 0 Jacky Jackson 38 True 1 Steven Stevenson 25 False
비고	

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	data = [{'Name': 'Jacky Jackson', 'Age': 38, 'Driver': True}, {'Name': 'Steven Stevenson', 'Age': 25, 'Driver': False}] pd.DataFrame(data, index=['row1', 'row2'])
결과값1	Name Age Driver row1 Jacky Jackson 38 True row2 Steven Stevenson 25 False
비고	

2. Series

Question 1

새로운 Series를 DataFrame에 추가하는 다음 코드는 pandas 2.0.0 버전 이후부터 'append()' Method가 완전히 제거되었기 때문에 더 이상 작동하지 않습니다. 트러블 슈팅하시오.

```
import pandas as pd

# 데이터프레임을 만듭니다.
dataframe = pd.DataFrame()

# 열을 만듭니다.
new_person = pd.Series(['Molly Mooney', 40, True], index=['Name','Age','Driver'])

# 방법 1: append() 메서드 사용
dataframe = dataframe.append(new_person, ignore_index=True)

# 결과 확인
print(dataframe)
>>
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C…

AttributeError: 'DataFrame' object has no attribute 'append'
```

2. Series

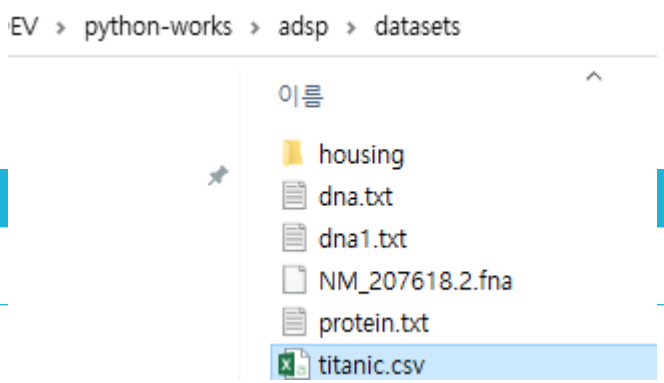
Answer 1

```
import pandas as pd

# 데이터프레임을 만듭니다.
dataframe = pd.DataFrame()

# 열을 추가합니다.
dataframe['Name'] = ['Jacky Jackson', 'Steven Stevenson']
dataframe['Age'] = [38, 25]
dataframe['Driver'] = [True, False]

# 새로운 행을 만듭니다.
new_person = pd.Series(['Molly Mooney', 40, True], index=['Name', 'Age', 'Driver'])

# 방법 2: loc[]을 사용하여 새로운 행 추가
dataframe.loc[len(dataframe)] = new_person

# 결과 확인
print(dataframe)
>>
          Name  Age  Driver
0     Jacky Jackson   38    True
1  Steven Stevenson   25   False
2      Molly Mooney   40    True
```

3. DataFrame

생성

- 데이터 가져오기 / 차원 확인



파일	소스코드
실습환경	준비 Tf38_cpu

소스코드	# 라이브러리를 임포트합니다.
	import pandas as pd
	
	# 데이터를 적재합니다.
	dataframe = pd.read_csv("datasets//titanic.csv")
	
	# 두 개의 행을 확인합니다.
	dataframe.head(2)
	
	# 차원을 확인합니다.
	dataframe.shape

결과값1	PassengerId Survived Pclass Name Gender Age SibSp Parch Ticket Fare Cabin Embarked 0 1 0 3 Braund, Mr. Owen Harris male 22.0 1 0 A/5 21171 7.2500 NaN S 1 2 1 1 Cumings, Mrs. John Bradley (Florence Briggs Th... female 38.0 1 0 PC 17599 71.2833 C85 C
결과값2	(891, 12)
비고	

3. DataFrame

DataFrame 생성과 몇 가지 검토

- 인덱스 지정

• 데이터프레임 객체 생성시 리스트나 어레이를 사용할 경우 변수명에 0부터 1씩 증가하는 값이 자동 부여되는데 직접 지정하고자 한다면 columns 인자에 값을 입력하면 된다. 덱스가 직접적으로 표기된 것을 알 수 있다. 시리즈 객체와 마찬가지로 데이터프레임 또한 인덱스 기반 연산도 가능하다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.DataFrame([[1, 2], [3, 4]], columns = ["col1", "col2"])
결과값1	col1 col2 0 1 2 1 3 4
비고	

3. DataFrame

DataFrame 생성과 몇 가지 검토

- 인덱스 지정
- 인덱스를 지정하는 데이터프레임 객체 생성은 다음과 같다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.DataFrame([[1, 2], [3, 4]], index = [100, 200])
결과값1	0 1 100 1 2 200 3 4
비고	

3. DataFrame

DataFrame 생성과 몇 가지 검토

- **주의점**
- 데이터프레임을 생성할 때는 각 변수에 할당되는 원소의 개수가 같아야 된다. 만약 다음과 같은 코드로 변수를 생성하고자 한다면 에러가 발생하는 것을 볼 수 있다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.DataFrame(dict(col1 = [1, 2], col2 = [3, 4, 5]))
결과값1	~\AppData\Roaming\Python\Python38\site-packages\pandas\core\internals\construction.py in _extract_index(data) 653 lengths = list(set(raw_lengths)) 654 if len(lengths) > 1: --> 655 raise ValueError("All arrays must be of the same length") 656 657 if have_dicts: ValueError: All arrays must be of the same length
비고	

3. DataFrame

Question 1

주어진 코드는 NumPy의 randn() 함수를 사용하여 3x4 형태의 랜덤한 숫자로 채워진 2차원 배열을 생성한 후, 이를 Pandas DataFrame으로 변환해 보자.

Answer 1

```
import numpy as np
import pandas as pd

# 3x4 형태의 랜덤한 숫자로 채워진 2차원 배열 생성
r = np.random.randn(3, 4)
print(r)  # 생성된 랜덤 배열 출력

# 생성된 배열을 DataFrame으로 변환
d = pd.DataFrame(r, index=['one', 'two', 'three'], columns=['a', 'b', 'c', 'd'])
print(d)  # DataFrame 출력
>>
[[-3.03230092  0.1323021   1.06509304 -1.11343787]
 [-0.13112835 -0.40795351  0.95316737  0.18322308]
 [ 0.06377442  0.54432592  0.94636602  0.29810242]]
              a         b         c         d
one   -3.032301  0.132302  1.065093 -1.113438
two   -0.131128 -0.407954  0.953167  0.183223
three  0.063774  0.544326  0.946366  0.298102
```

3. DataFrame

DataFrame 특정 행 선택과 조작

- DataFrame은 'state', 'year', 'pop' 열을 포함하고 있으며, 각 열에는 주어진 딕셔너리에 기반한 값들이 들어 있습니다. 여기서 'state'는 주의명, 'year'는 해당 연도, 'p

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	dic = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'], 'year': [2000, 2001, 2002, 2001, 2002], 'pop': [1.5, 1.7, 3.6, 2.4, 2.9]} d2 = pd.DataFrame(dic) d2
결과값1	state year pop 0 Ohio 2000 1.5 1 Ohio 2001 1.7 2 Ohio 2002 3.6 3 Nevada 2001 2.4 4 Nevada 2002 2.9
비고	

3. DataFrame

DataFrame 특정 행 선택과 조작

- DataFrame의 특정 속성을 사용하여 인덱스 및 열을 확인

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	pd.DataFrame(d2, columns = ['year', 'state', 'pop']) d2.index d2.columns 3 in d2.index 'state' in d2.columns
결과값1	year state pop 0 2000 Ohio 1.5 1 2001 Ohio 1.7 2 2002 Ohio 3.6 3 2001 Nevada 2.4 4 2002 Nevada 2.9
결과값1	RangeIndex(start=0, stop=5, step=1) Index(['state', 'year', 'pop'], dtype='object') True True
비고	

3. DataFrame

DataFrame 특정 행 선택과 조작

- DataFrame의 특정 속성을 사용하여 인덱스 및 열을 확인

파일	소스코드
	1. pd.DataFrame(d2, columns=['year', 'state', 'pop'])는 d2 DataFrame에서 'year', 'state', 'pop' 열을 선택하여 새로운 DataFrame을 만듭니다. 2. d2.index는 DataFrame의 인덱스를 반환합니다. 3. d2.columns는 DataFrame의 열을 반환합니다. 4. 3 in d2.index는 인덱스에 3이 있는지 여부를 확인합니다. (True 또는 False 반환) 5. 'state' in d2.columns는 열에 'state'가 있는지 여부를 확인합니다. (True 또는 False 반환)
비고	

3. DataFrame

DataFrame 특정 행 선택과 조작

- 열 선택
  - 열의 이름으로 선택

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	d['a'] d2['state']
결과값1	one -1.075684 two -0.159286 three -0.047937 Name: a, dtype: float64
결과값2	0 Ohio 1 Ohio 2 Ohio 3 Nevada 4 Nevada Name: state, dtype: object
비고	

3. DataFrame

DataFrame 특정 행 선택과 조작

- 열 삽입, 행 선택
- d2['debt'] = np.arange(5) 코드는 DataFrame에 'debt'라는 이름의 열을 추가하고, 0부터 4까지의 값을 할당하는 것입니다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	d2['debt'] = np.arange(5) d2 d2.loc[3] d2.iloc[3]
결과값1	state year pop debt 0 Ohio 2000 1.5 0 1 Ohio 2001 1.7 1 2 Ohio 2002 3.6 2 3 Nevada 2001 2.4 3 4 Nevada 2002 2.9 4
결과값2	state Nevada year 2001 pop 2.4 debt 3 Name: 3, dtype: object
	state Nevada year 2001 pop 2.4 debt 3 Name: 3, dtype: object
비고	

3. DataFrame

DataFrame 특정 행 선택과 조작

- 행과 열 교체
- d2.T는 DataFrame의 전치(transpose)를 나타냅니다. DataFrame의 전치는 행과 열을 바꾸는 작업을 의미합니다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	d2 d2.T
결과값1	

	state	year	pop	debt
0	Ohio	2000	1.5	0
1	Ohio	2001	1.7	1
2	Ohio	2002	3.6	2
3	Nevada	2001	2.4	3
4	Nevada	2002	2.9	4

결과값2

	0	1	2	3	4
state	Ohio	Ohio	Ohio	Nevada	Nevada
year	2000	2001	2002	2001	2002
pop	1.5	1.7	3.6	2.4	2.9
debt	0	1	2	3	4

비고

3. DataFrame

DataFrame 특정 행 선택과 조작

- reindex() 메서드를 사용하여 새로운 인덱스로 DataFrame을 재색인하는 작업

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	data = np.arange(9).reshape((3, 3)) data d3 = pd.DataFrame(data, index=['a', 'b', 'c'],columns=['Ohio', 'Texas', 'California']) d3 d4 = d3.reindex(['a', 'b', 'c', 'd'],fill_value=0) d4
결과값1	array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
결과값2	Ohio Texas California a 0 1 2 b 3 4 5 c 6 7 8
결과값3	Ohio Texas California a 0 1 2 b 3 4 5 c 6 7 8 d 0 0 0
비고	

3. DataFrame

DataFrame 특정 행 선택과 조작

- np.arange(9).reshape((3, 3))는 0부터 8까지의 값을 포함하는 3x3 형태의 NumPy 배열을 생성합니다.
- pd.DataFrame(data, index=['a', 'b', 'c'], columns=['Ohio', 'Texas', 'California'])는 주어진 NumPy 배열을 사용하여 DataFrame을 생성합니다. 여기서 'a', 'b', 'c'는 행 인덱스이고, 'Ohio', 'Texas', 'California'는 열 이름입니다.
- d3.reindex(['a', 'b', 'c', 'd'], fill_value=0)는 d3 DataFrame을 'a', 'b', 'c', 'd'로 재색인(reindex)합니다. 'd'는 기존에 없던 인덱스이므로, fill_value=0을 사용하여 새로운 행을 추가하고 해당 행을 0으로 채웁니다.

3. DataFrame

DataFrame 특정 행 선택과 조작

- **행 또는 열 삭제**
- 주어진 코드는 Pandas DataFrame에서 특정 행을 제거하는 방법을 보여줍니다. drop() 메서드는 DataFrame에서 특정 행이나 열을 제거하는 데 사용됩니다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	d5 = d4.drop('c') d5 d5 = d4.drop(['c', 'd']) d5
결과값1	Ohio Texas California a 0 1 2 b 3 4 5 d 0 0 0
결과값2	Ohio Texas California a 0 1 2 b 3 4 5
비고	

3. DataFrame

DataFrame 특정 행 선택과 조작

- 행 또는 열 삭제
- 여기서 axis=1은 열을 나타내며, drop() 메서드에서 axis 매개변수를 사용하여 열을 삭제합니다. 기본적으로 axis=0으로 설정되어 있어 행을 삭제합니다. 만약 열을 삭제하려면 axis=1을 명시해야 합니다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	d5 = d4.drop('Ohio', axis=1) d5 d5 = d4.drop(['Ohio', 'Texas'], axis=1) d5
결과값1	Texas California a 1 2 b 4 5 c 7 8 d 0 0
결과값2	California a 2 b 5 c 8 d 0
비고	

3. DataFrame

DataFrame 특정 행 선택과 조작

- Pandas DataFrame에서 sort_index() 메서드를 사용하여 인덱스 또는 열을 기준으로 정렬하는 방법을 보여줍니다.
- sort_index() 메서드는 DataFrame의 행 또는 열을 인덱스 또는 열 이름을 기준으로 정렬하는 데 사용됩니다. ascending=False를 지정하면 내림차순으로 정렬됩니다. axis=0은 인덱스(행)을 기준으로 정렬하며, axis=1은 열을 기준으로 정렬합니다.

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	d4.sort_index(ascending=False) # DataFrame d4의 인덱스를 내림차순으로 정렬 d4.sort_index(axis=1) # DataFrame d4의 열 이름을 기준으로 정렬 d4.sort_index(ascending=False, axis=1) # DataFrame d4의 열 이름을 내림차순으로 정렬
결과값1	Ohio Texas California d 0 0 0 c 6 7 8 b 3 4 5 a 0 1 2
결과값2	California Ohio Texas a 0 1 2 b 3 4 5 c 6 7 8 d 0 0 0
결과값3	Texas Ohio California a 1 0 2 b 4 3 5 c 7 6 8 d 0 0 0
비고	

3. DataFrame

DataFrame Methods

- 합, 평균, 최소, 최대 구하기

파일	소스코드
실습환경	준비 Tf38_cpu
소스코드	d4 d4.sum() d4.sum(axis=1) d4.mean() d4.mean(axis=1)
결과값1	Ohio Texas California a 0 1 2 b 3 4 5 c 6 7 8 d 0 0 0
결과값2	Ohio 9 Texas 12 California 15 dtype: int64 a 3 b 12 c 21 d 0 dtype: int64
결과값3	Ohio 2.25 Texas 3.00 California 3.75 dtype: float64 a 1.0 b 4.0 c 7.0 d 0.0
비고	