파이썬 머신러닝 판다스 데이터 분석 (요약)

PART 1. 판다스 입문

> 판다스는 시리즈와 데이터 프레임이라는 구조화된 데이터 형식을 갖는다.

1. 시리즈

- 데이터가 순차적으로 나열된 1차원의 배열 형태
- 인덱스는 데이터값과 일대일 대응

- 시리즈 만들기
  - 일반적으로는 딕셔너리를 시리즈로 변환하는 방법을 많이 사용
  - 판다스 내장 함수인 Series() 안에 딕셔너리를 함수의 인자로 전달한다.

```
pandas.Series(딕셔너리 객체 이름)
ex. sr = pandas.Series(dict_data)
```

- 인덱스 구조
  - 인덱스 배열

    ```
    Series객체.index
    ```

  - 데이터값 배열

    ```
    Series객체.values
    ```

- 원소 선택
  - 정수형 위치 인덱스 : 대괄호([]) 안에 위치를 나타내는 숫자 입력
  - 문자역 위치 인덱스 : 대괄호([]) 안에 이름과 함께 따옴표("", '')를 입력

2. 데이터프레임

- 데이터프레임은 2차원 배열 형태
- 여러 개의 열벡터들이 같은 행 인덱스 기준으로 줄지어 결합된 2차원 벡터 또는 행렬

- 데이터 프레임 만들기

```
pandas.DataFrame(딕셔너리 객체 이름)
ex. df = pandas.DataFrame(dict_data)
```

- 행의 인덱스와 열 이름 설정

```
pandas.DataFrame(데이터프레임, index=[행 이름 배열, ~], columns=[열 이름 배
열,~])
```

- 행의 인덱스와 열 이름 변경

```
데이터프레임.index = [새로운 행 인데스 배열, ~]
데이터프레임.colunms = [새로운 열 이름 배열, ~]
```

- 행의 인덱스와 열 이름 일부 변경

```
데이터프레임.rename(index = {기존 인덱스 : 새 인덱스, ~})
데이터프레임.rename(colunms = {기존 이름 : 새 이름, ~})
```

- 행 / 열 삭제
  - 보통 삭제하는 명령으로 drop()메서드를 사용하며 axis를 축 옵션으로 행 또는 열을 지정한다.

```
행 삭제 : 데이터프레임.drop(행 인덱스 또는 배열, axis=0)
열 삭제 : 데이터프레임.drop(열 이름 또는 배열, axis=1)

여러 개를 삭제할 때 배열(리스트)를 준다.
삭제한 개체는 새로운 객체로 반환된 것이기 때문에, 기존 객체를 변경하고 싶다면
inplace = True 라는 옵션을 추가한다.
```

- 행 선택
  - 데이터프레임의 행 데이터를 선택하기 위해서는 loc와 iloc 인덱서를 사용한다.
  - 인덱스 이름을 사용할 때 loc을 이용한다.
  - 정수형 위치 인덱스를 사용할 때는 iloc을 이용한다.

```
데이터프레임.loc['행 이름']
데이터프레임.iloc[정수형 숫자]
```

구분	loc	iloc
탐색 대상	인덱스 이름	정수형 위치 인덱스
범위 지정	가능(범위의 끝 포함)	가능(범위의 끝 제외)
	ex. ['a':'c'] = 'a', 'b', 'c'	ex. [3:6] = 3, 4, 5

- 열 선택
  - 열 데이터 한 개만 선택할 때

    ```
    데이터프레임['열 이름'] or 데이터프레임.열 이름
    ```

  - 열 데이터를 여러 개 선택할 때

    ```
    데이터프레임[[열 이름, 열 이름, ...]]
    주의해서 봐야할 것은 배열(리스트)안에 배열(리스트)이 하나 더 있다는 것이다.
    ```

- 원소 선택
  - 행 인덱스와 열 이름을 [행, 열] 형식의 2차원 좌표로 입력한다.

    ```
    인덱스 이름 방식 : 데이터프레임.loc[행 인덱스, 열 이름]
    정수 위치 인덱스 방식 : 데이터프레임.iloc[행 번호, 열 번호]
    ```

- 행 추가

  ```
  데이터프레임.loc['새로운 행 이름'] = 데이터값 또는 배열
  ```

- 열 추가

  ```
  데이터프레임['추가하려는 열 이름'] = 데이터값 또는 배열
  ```

- 원소 값 변경

  ```
  데이터프레임.loc['행 이름']['열 이름'] = 새로운 값
  데이터프레임.iloc[행 숫자][열 숫자] = 새로운 값
  ```

- 행 / 열 위치 바꾸기

```
데이터프레임.transpose()
데이터프레임.T

이때 바꾼 값을 유지하고 싶다면 inplace=True 를 준다.
```

- 특정 열을 행 인덱스로 설정
  - set_index 메소드를 사용하여 행 인덱스를 새로 지정한다.
  - 이때 기존 행 인덱스는 삭제된다.

```
데이터프레임.set_index(['열 이름'])
데이터프레임.set_index('열 이름')
```

- 행 인덱스 재배열
  - 기존 객체를 변경하지 않고 새로운 데이터프레임 객체를 반환한다.

```
데이터프레임.reindex(새로운 인덱스 배열)
ex. 새로운 인덱스 배열 = ['r1', 'r2', 'r3']
```

- 행 인덱스 초기화
  - reset_index() 메소드를 활용하여 set_index()로 인해 변화한 행 인덱스를 정수형 위치 인덱스로 초기화한다.

```
데이터프레임.reset_index()

이때 기존행 인덱스는 열로 이동한다.
```

- 행 인덱스를 기준으로 데이터프레임 정렬
  - 행 인덱스를 기준으로 데이터프레임 값을 정렬한다.

```
데이터프레임.sort_index(ascending =True / False)

이때 asceding = True는 오름차순 ascending = False는 내림차순 이다.
```

- 특정 열의 데이터 값을 기준으로 데이터프레임 정렬

```
데이터프레임.sotr_values(by='열 이름', asceding= True / False)
```

3. 산술연산

> 판다스 객체의 산술연산은 3단계를 거친다.

> 1. 행/열 인덱스를 기준으로 모든 원소를 정렬한다.
> 2. 동일한 위치에 있는 원소끼리 일대일로 대응시킨다.
> 3. 일대일 대응이 되는 원소끼리 연산을 처리한다.
>
> 다만 대응되는 원소가 없으면 NaN으로 처리한다.

- 시리즈 vs 숫자
  - 시리즈 객체에 어떤 숫자를 더하면 시리즈의 개별 원소에 각각 숫자를 더한 계산결과가 시리즈 객체로 리턴된다.

```
시리즈객체 + 연산자(+, -, *,/) + 숫자
```

- 시리즈 vs 시리즈
  - 시리즈의 모든 인덱스에 대하여 같은 인덱스를 가진 원소끼리 계산한다.
  - 인덱스로 주어진 순서가 다르더라도 판다스는 같은 인덱스를 찾아 정렬한 후 계산한다.
  - 연산을 하는 두 시리즈의 원소 개수가 다르거나, 시리즈의 크기가 같더라도 인덱스 값이 다른 경우가 있다. 동일한 인덱스를 찾지 못하는 경우, 판다스는 유효한 값이 존재하지 않는다는 의미를 갖는 NaN으로 처리한다.

```
시리즈객체1 + 연산자(+, -, *,/) + 시리즈객체2
```

- 연산 메소드
  - 개체 사이에 공통된 인덱스가 없거나 NaN이 포함된 경우 연산 결과가 NaN이 반환되는데, 이를 피하기 위해서는 연산 메소드에 fill_value 옵션을 설정하여 적용한다.

```
누락된 NaN 대신 숫자 0을 입력하여 적용

덧셈 : 시리즈객체1.add(시리즈객체2, fill_value=0)
뺄셈 : 시리즈객체1.sub(시리즈객체2, fill_value=0)
곱셈 : 시리즈객체1.mul(시리즈객체2, fill_value=0)
나눗셈 : 시리즈객체1.div(시리즈객체2, fill_value=0)
```

- 데이터프레임 연산
  - 데이터프레임에 어떤 숫자를 더하면 모든 원소에 숫자를 더한다. (다른 사칙연산도 동일)
  - 이때 데이터프레임은 계산된 새로운 값으로 바뀐다.

```
데이터프레임 + 연산자(+, -, *,/) + 숫자
```

- 데이터프레임 vs 데이터프레임
  - 각 데이터프레임의 같은 행, 같은 열 위치에 있는 원소끼리 계산한다.
  - 해단 결과값을 원래 위치에 다시 입력하여 데이터프레임을 만든다.
  - 이때 한쪽에 원소가 존재하지 않거나 NaN이면 연산 결과는 NaN이다

```
데이터프레임1 + 연산자(+, -, *,/) + 데이터프레임2
```

PART 2. 데이터 입출력

1. CSV

- CSV 파일 불러오기

```
pandas.read_csv('파일경로/파일이름.csv')
```

- CSV 파일 저장하기

```
데이터프레임.to_csv('파일경로/파일이름.csv')
```

2. Excel

- Excel 파일 불러오기

```
pandas.read_excel('파일경로/파일이름.xlsx')
```

- Excel 파일 저장하기

```
데이터프레임.to_excel('파일경로/파일이름.xlsx')
```

- 여러 개의 데이터프레임을 하나의 Excel 파일로 저장

```
pandas.ExcelWriter('파일경로/파일이름.xlsx')

ex.
writer = pandas.ExcelWriter('./data/df_excelwriter.xlsx')
df1.to_excel(writer, sheet_name='sheet1')
df2.to_excel(writer, sheet_name='sheet2')
writer.save()
```

3. JSON

- JSON 파일 불러오기

```
pandas.read_json('파일경로/파일이름.json')
```

- JSON 파일 저장하기

```
데이터프레임.to_json('파일경로/파일이름.json')
```

4. HTML

- HTML 웹페이지에서 표 속성 가져오기
    - read_html()함수는 HTML 웹 페이지에 있는 `<table>`태그에서 표 형식의 데이터를 모두 찾아 데이터프레임으로 변환한다. 이때 각각 변환되기 때문에 여러 개의 데이터프레임을 원소로 갖는 리스트가 반환된다.

```
pandas.read_html('웹 주소(url)')
pandas.read_html('파일경로/파일이름.html')
```

PART 3. 데이터 살펴보기

1. 데이터프레임 구조

- 데이터 내용 미리보기

```
데이터프레임.head(n)
데이터프레임.tail(n)
```

- 데이터프레임의 크기(행, 열)
    - 데이터프레임 클래스의 shape 속성은 행과 열의 개수를 튜플 형태로 보여준다.

```
데이터프레임.shape
```

- 데이터프레임의 기본 정보
    - info()메서드를 사용하면 클래스 유형, 행 인덱스의 구성, 열 이름의 종류와 개수, 각 열의 자료형과 개수, 메모리 할당량에 관한 정보가 출력된다.

```
데이터프레임.info()
```

- 데이터프레임의 기술 통계 정보 요약
    - describe()메서드를 적용하면 산술(숫자) 데이터를 갖는 열에 대한 주요 기술 통계 정보(평균, 표준편차, 최대값, 최솟값, 중간값 등)을 요약하여 출력한다.
    - 산술 데이터가 아닌 열에 대한 정보를 포함하고 싶을 때는, include='all' 옵션을 추가한다. 이때 문자열 결과는 최빈값과 빈도수에 대한 정보가 출력된다.

```
데이터프레임.describe()
데이터프레임.describe(include='all' )
```

- 각 열의 데이터 개수
  - count() 메서드는 데이터프레임의 각 열이 가지고 있는 데이터 개수를 시리즈 객체로 반환한다/

```
데이터프레임.count()
```

- 각 열의 고유값 개수
  - value_counts() 메서드는 시리즈 객체의 고유값 개수를 세는 데 사용한다.
  - dropna=True 옵션을 설정하면 데이터 값 중에서 NaN을 제외하고 개수를 계산한다.
  - dropna=False 가 기본옵션이다.

```
데이터프레임['열 이름'].value_counts()
```

2. 통계 함수 적용

- 평균값
  - 데이터프레임에 mean() 메서드를 적용하면, 산술 데이터를 갖는 모든 열의 평균값을 각각 계산하여 시리즈 객체로 반환한다.
  - 데이터프레임의 특정 열을 선택하여 평균값을 계산할 수도 있다.

```
데이터프레임.mean()
데이터프레임['열 이름'].mean()
```

- 중간값
  - 데이터프레임에 median() 메서드를 적용하면, 산술 데이터를 갖는 모든 열의 중간값을 각각 계산하여 시리즈 객체로 반환한다.
  - 데이터프레임의 특정 열을 선택하여 중간값을 계산할 수도 있다.

```
데이터프레임.median()
데이터프레임["열 이름"].median()
```

- 최대값
  - 데이터프레임에 max() 메서드를 적용하면 데이터프레임의 각 열이 갖는 데이터값 중에서 최대값을 계산하여 시리즈로 반환한다.
  - 특정열을 선택하여 계산할 수도 있다.
  - 문자열 데이터는 아스키코드로 크고 작음을 비교한다.

```
모든 열의 최대값 : 데이터프레임객체.max()
특정 열의 최대값 : 데이터프레임객체['열 이름'].max()
```

- 최소값

- 데이터프레임에 min() 메서드를 적용하면 데이터프레임의 각 열이 갖는 데이터값 중에서 최소값을 계산하여 시리즈로 반환한다.
- 특정열을 계산하여 계산할 수도 있다.
- 문자열 데이터는 아스키코드로 크고 작음을 비교한다.

```
모든 열의 최소값 : 데이터프레임객체.min()
특정 열의 최소값 : 데이터프레임객체['열 이름'].min()
```

- 표준편차
  - 데이터프레임에 std() 메서드를 적용하면 산술 데이터를 갖는 열의 표준편차를 계산하여 시리즈로 반환한다.
  - 특정열을 계산하여 계산할 수도 있다.
  - 문자열 데이터는 비교하지 않는다.

```
모든 열의 표준편차 : 데이터프레임객체.std()
특정 열의 표준편차 : 데이터프레임객체['열 이름'].std()
```

- 상관계수
  - 데이터프레임에 corr() 메서드를 적용하면 두 열 간의 상관계수를 계산한다.
  - 산술 데이터를 갖는 모든 열에 대해 2개씩 서로 짝을 짓고, 각각의 경우에 대하여 상관계수를 계산한다.
  - 문자열 데이터는 계산이 불가능하기 때문에 포함하지 않는다.

```
모든 열의 상관계수 : 데이터프레임객체.corr()
특정 열의 상관계수 : 데이터프레임객체[['열 이름', '열 이름', ~]].corr()
```

3. 판다스 내장 그래프 도구 활용

> 판다스 그래프 도구를 사용하는 방법은 간단하다.
>
> 시리즈 또는 데이터프레임 객체에 plot() 메서드를 적용하여 그래프를 그린 뒤, kind 옵션으로 종류를 선택한다.

kind 옵션	설명	kind 옵션	설명
'line'	선 그래프	'kde'	커널 밀도 그래프
'bar'	수직 막대 그래프	'area'	면적 그래프
'barh'	수평 막대 그래프	'pie'	파이 그래프
'his'	히스토그램	'scatter'	산점도 그래프
'box'	박스 그래프	'hexbin'	고밀도 산점도 그래프

- 선 그래프
  - 데이터프레임 객체에 plot()메서드를 적용할 때 다른 옵션을 추가하지 않으면 가장 기본적인 선 그래프를 그린다.

```
선 그래프 : 데이터프레임객체.plot()
```

- 막대 그래프
  - plot()메서드 안에 kind='bar' 옵션을 추가한다.

```
막대 그래프 : 데이터프레임객체.plot(kind='bar')
```

- 히스토그램
  - plot()메서드 안에 kind='hist' 옵션을 추가한다.

```
히스토그램 : 데이터프레임객체.plot(kind='hist')
```

- 산점도
  - plot()메서드 안에 kind='scatter' 옵션을 넣고, 데이터프레임의 열 중에서 서로 비교할 두 변수를 설정한다.

```
산점도 : 데이터프레임객체.plot(x='특정 열', y='특정 열', kind='scatter')
```

- 박스 플롯
  - plot()메서드 안에 kind='box' 옵션을 입력한다.
  - 박스플롯을 통해 'o' 표시를 통해 이상값을 확인할 수 있다.

```
박스 플롯 : 데이터프레임['열 이름', '열 이름', ~].plot(kind='box')

이때 하나만 보고 싶다면 데이터프레임에 컬럼을 하나만 지정하면 된다.
```

PART 4. 시각화 도구

1. Matplotlib - 기본 그래프 도구

> 보통 import matplotlib.pyplot as plt 로 약어를 설정해준다.

1.1 선 그래프

- 선 그래프는 연속하는 데이터 값들을 직선 또는 곡선으로 연결하여 데이터 값 사이의 관계를 나타낸다.
- 특히 **시계열 데이터**와 같이 연속적인 값의 변화와 패턴을 파악하는데 적합하다.
- 선 그래프는 기본 옵션이기 때문에 옵션을 설정하지 않고 plot 함수를 쓰면 선 그래프가 나온다.

```
# 스타일 서식 지정
plt.style.use('ggplot')

# 그림 사이즈 지정
plt.figure(figsize=(가로숫자,세로숫자))

# x축, y축 데이터를 plot 함수에 입력 => 시리즈의 인덱스를 x축 데이터로, 데이터값을 y축 데이터로 전달
plt.plot(특정 열이나 행, 특정 열이나 행, marker='지정 알파벳', markersize=숫자입력)

# 판다스 객체 자체를 plot함수에 입력하는 것도 가능하다.

# 그래프 객체에 차트 제목을 추가할 때 title()함수를 이용한다.
plt.title("제목 입력", size=숫자)

# 축 이름 추가
plt.xlabel("x축 이름", size=숫자)
plt.ylabel("y축 이름", size=숫자)

# matplotlib 한글 문제 해결 ==> 해당 코드를 그대로 쓰면 된다.
from matplotlib import font_manager, rc
font_path = "./폰트파일위치/폰트.ttf"
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc('font', family=font_name)

# x축, y축 범위 지정 (최솟값, 최댓값)
plt.xlim(최솟값, 최댓값)
plt.ylim(최솟값, 최댓값)

# x축 눈금 라벨 회전 / vertical 대신 숫자를 입력해도 된다.
plt.xsticks(rotation='vertical')

# 범례표시 / best는 최적의 위치를 선정해주는 자동기능이다.
plt.legend(labels=['범례 이름'], loc='best', fontsize=숫자)

# 그래프 출력
plt.show()
```

PART 5. 데이터 사전 처리

1. 누락 데이터 처리

> 유효한 데이터 값이 존재하지 않는 누락 데이터를 NaN으로 표시한다.

- 누락 데이터 확인
  - info() 메서드로 데이터프레임 요약정보를 출력하면 NaN 값의 개수를 보여준다.
  - value_count() 메서드로 특정 열의 누락 데이터를 확인할 수 있다. 이때 누락 데이터의 개수를 확인하려면 반드시 `dropna=False` 라는 옵션을 적용해야 한다. 그렇지 않으면 NaN값을 제외하고 유효한 데이터 개수만을 구한다.
  - isnull() 메서드와 notnull() 메서드를 통해 직접적으로 누락 데이터를 찾을 수 있다.

```
isnull() : 누락데이터면 True 반환, 유효한 데이터면 False를 반환
notnull() : 유효한 데이터면 True 반환, 누락 데이터면 False를 반환

이때 True는 1로 계산되고 False는 0으로 판별되기 때문에, sum(axis=0) 메소드를 활용하여 True(1)의 합을 구할 수 있다.
```

- 누락 데이터 제거
  - dropna() 메서드를 적용해 NaN 값을 갖는 행 또는 열을 삭제할 수 있다.
  - subset옵션으로 열을 한정할 수 있다.
  - how 옵션으로 NaN 값에 따라 삭제 조건을 줄 수 있다.

```
누락 데이터 삭제(기준점) : 데이터프레임객체.dropna(axis= 0 또는 1, thresh=n)
누락 데이터 삭제(조건) : 데이터프레임객체.dropna(subset=['열 이름'], how='any' or 'all',axis= 0 또는 1)

이때 n은 결측치가 n개 이상인 행 또는 열을 삭제하라는 기준점을 주는 것이다.
how옵션의 any는 NaN가 하나라도 존재하는 행 또는 열을 삭제라하는 의미이고, all은 모든 데이터가 NaN인 경우에만 삭제하라는 의미이다.
```

- 누락 데이터 치환
  - 일반적으로 누락 데이터는 평균값, 최빈값, 이전값 등으로 활용하여 대체한다.
  - 판다스의 fillna() 메서드로 변경한다.
  - 이때 fillna()를 적용한 값을 유지하려면 inplace = True 옵션을 추가해야 한다.
  - 직전값, 직후값으로 치환하려면 method 옵션을 활용해 전달한다.

```
누락 데이터 치환 : 데이터프레임객체.fillna(변경할 인자, inplace=True)
누락 데이터를 직전값으로 치환 : fillna(method='ffill')
누락 데이터를 직후값으로 치환 : fillna(method='bfill')
```

- 누락 데이터가 NaN으로 표시되지 않는 경우
  - 데이터셋 중 누락 데이터를 0, -, ? 와 같은 값으로 입력해놓은 데이터가 존재한다.
  - 이러한 데이터를 다루기 위해서는 replace() 메서드를 활용해 numpy에서 지원하는 np.nan으로 치환해 NaN 값으로 먼저 변경해 주는 것이 좋다.

```
누락 데이터 통일 : 데이터프레임객체.replace('기존 데이터값', np.nan, inplace=True)
```

2. 중복 데이터 처리

- 중복데이터 확인
  - duplicated() 메서드를 통해 행의 레코드가 중복되는지 여부를 확인한다.
  - 전에 나온 행들과 비교해 중복되는 행이면 True, 처음 나온 행이면 False를 반환한다.
  - 이때 0행 데이터는 첫번째로 오기 때문에 무조건 False처리 된다.
  - unique() 메서드를 통해 중복되지 않는 값들을 확인할 수 있다.

```
모든 중복데이터 확인 = 데이터프레임객체.duplicated()
특정 열 중복데이터 확인 = 데이터프레임객체['열 이름'].duplicated()
```

- 중복데이터 제거
  - drop_duplicated() 메서드를 통해 중복데이터를 제거할 수 있다.
  - 원본 객체를 변경하려면 inplace=True 옵션을 준다.
  - subset 옵션에 열 이름 리스트를 전달하며 특정 열에서만 중복데이터를 제거할 수 있다.

```
모든 중복데이터 제거 = 데이터프레임객체.drop_duplicated()
특정 열 중복데이터 제거 = 데이터프레임객체.drop_duplicated(subset=['열 이름','열 이름', ~], inplace=True)
```

3. 데이터 표준화

> 서로 다른 단위가 섞여 있거나 다른 형식으로 표현된 데이터를 일관성있게 표준화 하는 작업이다.

- 단위 환산
  - round(n) 메서드를 사용해 소수점 단위를 맞출 수 있다.

- 자료형 변환
  - 숫자형을 문자열로, 문자열을 숫자형으로 변환하기 전, dtypes() 메서드를 확인해 데이터 유형을 파악한다.
  - astype() 메서드를 활용해 자료형을 변환한다.

```
자료형 변환 : 데이터프레임['열 이름'].astype()
이때 astype() 안에는 'float', 'category', 'str' 등을 인자로 주어 변환 가능하다.
```

4. 범주형(카테고리) 데이터 처리

- 구간 분할

> 연속 변수를 일정한 구간으로 나누고, 각 구간을 범주형 이산 변수로 변환하는 과정을 구간 분할이라고 한다.

  - cut()함수를 이용하면 연속 데이터를 여러 구간으로 나누고 범주형 데이터로 변환 가능하다.
  - 이때 나누려는 구간(bin)의 개수를 bins라는 옵션에 입력하면 각 구간에 속하는 값의 개수와 경계값 리스트(bin_drivers)를 반환한다.
  - include_lowest = True 옵션을 사용하면 각 구간의 낮은 경계값을 포함한다.

```
데이터 구간 분할 : pd.cut(x=데이터객체['열 이름']
                        bins = '경계값 리스트'
                        labels = '새로운 경계값 리스트 이름'
                        include_lowest = True  # 첫 경계값 포함)
```

- 더미 변수

> 카테고리를 나타내는 범주형 데이터를 회귀분석 등 머신러닝 알고리즘에 바로 사용할 수 없는 경우에는, 컴퓨터가 인식 가능한 입력값으로 변환해야 한다. 이때 숫자 0 또는 1로 표현되는 더미변수를 사용한다. 이는 어떤 특성이 있는지 없는지 여부만을 표시하는 것이다.

  - 범주형 데이터를 컴퓨터로 인식할 수 있도록 숫자 0과 1로만 구성되는 원핫벡터로 변환한다고 해서 원핫 인코딩이라 부르기도 한다.
  - get_dummies() 함수를 사용하면 범주형 변수의 모든 고유값을 각각 새로운 더미 변수로 변환가능하다.
  - 더미변수로 변환한 이후에는 sklearn 등을 통해 여러 분석을 수행한다.

```
더미변수로 변환 : pd.get_dummies(데이터프레임['열 이름'])
이때 '열 이름'에 해당하는 컬럼은 카테고리 자료형이어야 한다.
```

5. 정규화

> 각 변수에 들어있는 숫자 데이터의 상대적 크기 차이로 인해 머신러닝 분석결과가 달라질 수 있다. 따라서 숫자 데이터의 상대적인 크기 차이를 제거할 필요가 있다. 각 열에 속하는 데이터값을 동일한 크기 기준으로 나눈 비율로 나타내는 것을 정규화라 한다.
>
> 정규화 과정을 거친 데이터 범위는 0~1 또는 -1 ~ 1이다.

- 각 열의 데이터를 `해당 열의 최대값의 절대값`으로 나누는 방법
  - 가장 큰 값을 최대값으로 나누면 자기자신이기 때문에 1이 된다.

```
데이터프레임.열 이름 = 데이터프레임.열 이름 / abs(데이터프레임.열 이름.max())
```

- 각 열의 데이터 중 `최대값과 최솟값을 뺀 값`으로 나누는 방법
  - 해당 열을 최대값으로 나누면 0~1 사이의 값들로 정규화가 이루어 진다.

```
min_x = 데이터프레임.열 이름 - 데이터프레임.열 이름.min()
min_max = 데이터프레임.열 이름.max() - 데이터프레임.열 이름.min()
데이터프레임 = min_x / min_max
```

6. 시계열 데이터

> 시계열 데이터는 주식, 환율과 같은 금융데이터를 다루기 위해서 반드시 필요하다. 판다스 시간 표시 방식 중 시계열 데이터 표현에 자주 이용되는 두 가지 유형이 있다.
>
> 특정한 시점을 기록하는 Timestamp와 두 시점 사이의 일정 기간을 나타내는 Period가 그것이다.

- 다른 자료형을 시계열 객체로 변환
  - 시간 데이터들은 시간자료형으로 기록되지 않고, 문자열 또는 숫자로 저장되는 경우가 많다.
  - to_datetime() 함수를 사용하면 문자열 등 다른 자료형을 Timestamp를 나타내는 datetime64 자료형으로 변환가능하다.
  - 보통 datetime64형 컬럼을 새로 만든 뒤, 이전 컬럼을 지우는 형태로 자료정리한다.

```
시계열 객체로 변환 : pd.to_datetime(데이터프레임['열 이름'])
```

- Timestamp를 Period로 변환
  - 판다스 to_period() 함수를 이용하면 Timestamp 객체를 일정 기간을 나타내는 period 객체로 변환할 수 있다.
  - 이때 freq옵션에 기준이 되는 기간을 설정한다.

옵션	설명	옵션	설명
D	day(1일)	B	business day(휴일 제외 / 주 5일)
W	week(1주)	H	hour(시간)
M	month end(월말)	T	minute(분)
MS	month begin(월초)	S	second(초)
Q	quater end(분기 말)	L	millisecond
QS	quater begin(분기 초)	U	mircosecond
A	year end(연말)	N	nanosecond
AS	year begin(연초)		

- 시계열 데이터 만들기
  - 판다스 date_range() 함수를 사용하면 여러 개의 날짜(Timestamp)가 들어있는 배열 형태의 시계열 데이터를 만들 수 있다.

```
Timestamp 배열 : pd.date_range(start='20xx-xx-xx',    # 날짜 범위의 시작
                               end=None,      # 날짜 범위의 끝 / None = 지정 안함
                               periods=n     # n에 생성하고 싶은 개수를 설정
                               freq='MS'     # freq로 기간 설정
                               tz='Asia/Seoul') # 타임존 설정

이때 만약 freq에 3M을 주면 3개월 간격으로 기간이 설정된다.
```

  - 판다스 period_range() 함수는 여러 개의 기간이 들어있는 시계열 데이터를 만든다.

```
Period 배열 : pd.period_range(start='20xx-xx-xx',     # 날짜 범위의 시작
                              end=None,      # 날짜 범위의 끝 / None = 지정 안
함
                              periods=n     # n에 생성하고 싶은 period 개수를
설정
                              freq='MS')        # freq로 기간 설정
```

- 시계열 데이터 활용 - 날짜데이터 분리 및 날짜 인덱스 활용
  - 연-월-일 날짜 데이터(Timestamp 형)에서 일부를 분리하여 추출할 수 있다.

```
데이터프레임['열 이름'].dt.year # 연도 추출
데이터프레임['열 이름'].dt.month # 월 추출
데이터프레임['열 이름'].dt.day # 일 추출
이때 '열 이름'에 포함된 데이터는 Timestamp이다.
```

  - 날짜데이터(Period 형)에서도 연-월, 연도를 추출할 수 있다.

```
데이터프레임['열 이름'].dt.to_period(frqp='A') # 연도 추출
데이터프레임['열 이름'].dt.to_period(frqp='M') # 연-월 추출
```

  - set_index() 메서드로 날짜 데이터를 행 인덱스로 만들 수 있다.

```
데이터프레임.set_index('열 이름', inplace=True)
```

PART 6. 데이터프레임의 다양한 응용

1. 함수 매핑

> 함수 매핑은 시리즈 또는 데이터프레임의 개별 원소를 트겅 함수에 일대일 대응시키는 과정이다.

- 시리즈 원소에 함수 매핑
  - 시리즈 객체에 apply() 메서드를 적용하면 인자로 전달하는 매핑 함수에 시리즈의 모든 원소를 하나씩 입력하고 함수의 리턴값을 돌려받는다.

```
시리즈의 원소에 함수 매핑 : 시리즈객체.apply(매핑함수)
```

- 데이터프레임 원소에 함수 매핑
  - 데이터프레임의 개별 원소에 특정 함수를 매핑하려면 applyamp() 메서드를 활용한다.
  - 매핑함수에 데이터프레임의 각 원소를 하나씩 넣어서 리턴값으로 돌려받는다.

```
데이터프레임의 원소에 함수 매핑 : 데이터프레임객체.applymap(매핑함수)
```

- 데이터프레임의 각 열에 함수 매핑

- 데이터프레임에 apply(axis=0) 메서드를 적용하면 모든 열을 하나씩 분리하여 매핑함수의 인자로 각 열(시리즈)이 전달된다.
- 매핑함수에 따라 반환되는 객체의 종류가 다르다.
- 시리즈를 입력받고 시리즈를 반환하는 함수를 매핑하면, 데이터프레임을 반환한다.
- 한편 시리즈를 입력받아서 하나의 값을 반환하는 함수를 매핑하면 시리즈를 반환한다.

```
데이터프레임의 열에 함수 매핑 : 데이터프레임객체.apply(매핑함수, axis=0)
axis=0 옵션의 경우 따로 설정하지 않아도 apply 함수 디폴트값으로 적용된다.
```

- 데이터프레임의 각 행에 함수 매핑
  - 데이터프레임 객체에 apply(axis=1) 메서드를 적용하면 데이터프레임의 각 행을 매핑함수의 인자로 전달한다.

```
데이터프레임의 행에 함수 매핑 : 데이터프레임객체.apply(매핑함수, axis=1)
```

- 데이터프레임 객체에 함수 매핑
  - 데이터프레임 객체를 함수에 매핑하려면 pipe() 메서드를 활용한다.
  - 이때 사용하는 함수가 반환하는 리턴값에 따라 pipe() 메서드가 반환하는 객체의 종류가 결정된다.

```
데이터프레임객체에 함수 매핑 : 데이터프레임객체.pipe(매핑함수)
```

2. 열 재구성

- 열 순서 변경
  - 열 이름을 원하는 순서대로 정리해서 리스트를 만들고, 데이터프레임에서 열을 다시 선택하는 방식으로 열 순서를 바꿀 수 있다.

```
데이터프레임의 열 순서 변경 : 데이터프레임 객체[재구성한 열 이름 리스트]
```

- 열 분리
  - 하나의 열이 여러 가지 정보를 담고 있을 때 각 정보를 서로 분리해서 사용하는 경우가 있다.

```
시리즈의 문자열 리스트 인덱싱 : 시리즈 객체.str.get(인덱스)
```

3. 필터링

- 불린 인덱싱
  - 시리즈 객체에 어떤 조건식을 적용하면 각 원소에 대해 참/거짓을 판별하여 불린(참, 거짓)으로 구성된 시리즈를 반환한다.

```
데이터프레임의 불린 인덱싱 : 데이터프레임객체[불린 시리즈]
```

- isin()메서드 활용
  - 데이터프레임의 열에 isin() 메서드를 적용하면 특정 값을 가진 행들을 따로 추출할 수 있다.

```
isin() 메서드를 활용한 필터링 : 데이터프레임의 열 객체.isin(추출 값의 리스트)
```

4. 데이터프레임 합치기

> 판다스에서 데이터프레임을 합치거나 연결할 때 사용하는 함수와 메서드는 여러가지이다.
>
> 대표적으로는 concat(), merge(), join() 등이 있다.

- 데이터프레임 연결
  - 기본옵션은 axis=0 이기 때문에, 위 아래 행 방향으로 연결된다.
  - axis=1로 주면, 데이터프레임을 좌우 열 방향으로 연결한다.
  - join='outer'도 기본옵션이기 때문에, 각 데이터프레임의 합집합으로 연결된다.
  - join='inner'로 주면 데이터프레임의 교집합이 기준이 된다.
  - ignore_index=True 옵션은 기존 행 인덱스를 무시하고, 새로운 행 인덱스를 설정한다.

```
데이터프레임의 연결 : pandas.concat(데이터프레임의 리스트)
```

- 데이터프레임 병합
  - concat()이 여러 데이터 프레임을 이어 붙이듯 연결하는 것이라면, merge()는 어떤 기준에 의해 두 데이터프레임을 병합하는 개념이다.
  - 이때 어떤 기준이 되는 열이나 인덱스를 key 라고 한다. 키가 되는 열이나 인덱스는 반드시 양쪽 데이터프레임에 모두 존재해야 한다.
  - 키가 되는 열을 지정해 줄 때는 on='키가 되는 열 이름'을 지정한다.
  - 각자 키가 되는 열이 이름이 다르다면, left_on과 right_on을 통해 열을 설정하면 된다.
  - how는 데이터프레임의 결합방식을 지칭한다. 'outer'이면 합집합, 'inner'이면 교집합 방식이다.

```
데이터프레임 병합 : pandas.merge(기준이 되는 데이터프레임, 합치고자 하는 데이터프레임, how='inner', on=None)
```

- 데이터프레임 결합
  - join()메서드는 기본적으로 merge()함수와 비슷하지만, 두 데이터프레임의 행 인덱스를 기준으로 결합하는 점에서 차이를 갖는다.
  - 다만 on=keys 옵션을 적용하면 행 인덱스 대신 다른 열을 기준으로 결합하는 것이 가능하다.
  - how 옵션을 통해 왼쪽에 붙일 지, 오른쪽에 붙일 지 선택가능하다.

```
행 인덱스를 기준으로 결합 : 기준이 되는 데이터프레임.join(합치고자 하는 데이터프레임, how='left')
```

5. 그룹 연산

> 특정 기준을 적용하여 몇 개의 그룹으로 분할하여 처리하는 것으 그룹연산이라고 한다.
>
> 그룹연산은 데이터를 집계, 변환, 필터링하는데 효율적이다.
>
> 1) 분할 : 데이터를 특정 조건에 의해 분할
>
> 2) 적용 : 데이터를 집계, 변환, 필터링하는데 필요한 메소드 적용
>
> 3) 결합 : 2단계의 처리 결과를 하나로 통합

분할단계

- 1개 열을 기준으로 그룹화
  - groupby()메서드는 데이터프레임의 특정 열을 기준으로 데이터프레임을 분할하여 그룹 객체를 반환한다.
  - 기준이 되는 열은 1개도 가능하고, 여러 열을 리스트로 입력할 수도 있다.
  - 이때 그룹 객체에서 get_group()메서드를 적용하면 특정 그룹만을 선택할 수 있다.

```
그룹 연산(분할) : 데이터프레임.groupby(기준이 되는 열)
```

- 여러 열을 기준으로 그룹화
  - groupby()메서드에 여러 개의 열을 리스트로 전달하면 반환되는 그룹 객체의 인덱스는 다중 구조를 갖는다.
  - 예컨대 groupby()메서드에 두 열을 인자로 전달하면 두 열이 갖는 원소 값들로 만들 수 있는 모든 조합으로 키를 생성한다. 이때 조합의 형식은 튜플로 지정된다.

```
그룹 연산(분할) : 데이터프레임.groupby(기준이 되는 열 리스트)
```

적용-결합 단계

- 데이터 집계
  - 그룹 객체에 다양한 연산을 적용하는 것을 데이터 집계라 한다.
  - 이때 count()함수는 데이터가 컬럼마다 몇 개인지, size()는 행마다 몇 개인지를 알려준다.

```
평균 데이터 집계(내장함수) : group객체.mean()

이외에도 max(), min(), sum(), count(), size(), var(), std(), describe(),
info(), first(), last() 등이 있다.
```

  - 집계 연산을 처리하는 사용자 정의 함수를 그룹 객체에 적용하려면 agg()메서드를 사용한다.

```
agg()메서드 데이터 집계 : group객체.agg(매핑함수)
```

  - 동시에 여러 개의 함수를 사용하여 각 그룹별 데이터에 대한 집계연산을 처리할 수도 있다. 이때 각각의 열에 여러 개의 함수를 일괄 적용할 때는 리스트 형태로 인수를 전달하고, 열마다 다른 종류의 함수를 적용하려면 {열:함수}형태의 딕셔너리를 전달한다.

```
모든 열에 여러 함수를 매핑 : group객체.agg([함수1, 함수2, 함수3, ...])
각 열마다 다른 함수를 매핑 : group객체.agg({'열1':함수1, '열2':함수2, ...})
```

- 그룹 연산 데이터 변환
  - transform()메서드는 그룹별로 구분하여 각 원소에 함수를 적용하지만 그룹별 집계 대신 각 원소의 본래 행 인덱스와 열 이름을 기준으로 연산 결과를 반환한다.
  - 즉 그룹연산의 결과를 원본 데이터프레임과 같은 형태로 변형하여 정리하는 것이다.

```
데이터 변환 연산 : group객체.transform(매핑함수)
```

- 그룹 객체 필터링
  - 그룹객체에 filter()메서드를 적용할 때, 조건식을 가진 함수를 전달하면 조건이 참인 그룹만을 남긴다.

```
그룹객체 필터링 : group객체.filter(조건식 함수)
```

- 그룹 객체 함수 매핑
  - apply()메서드는 판다스 객체의 개별 원소를 특정 함수에 일대일로 매핑한다.
  - 대부분의 연산을 그룹 객체에도 적용할 수 있다.

```
범용 메서드 : group객체.apply(매핑함수)
```

6. 멀티 인덱스

> 판다스는 행 인덱스를 여러 레벨로 구현할 수 있도록 멀티 인덱스 클래스 지원한다.

- 멀티인덱스에서 하나의 인덱스만 사용하는 방법

```
데이터프레임.loc['열 이름']
```

- 멀티인덱스에서 두 개의 인덱스를 사용하는 방법

```
데이터프레임.loc[('열 이름1', '열 이름2')]
```

7. 피벗

> pivot_table()함수는 엑셀에서 사용하는 피벗테이블과 비슷한 기능을 처리한다. 피벗테이블을 구성하는 4가지 요소(행 인덱스, 열 인덱스, 데이터 값, 데이터 집계 함수)에 적용할 데이터프레임의 열을 각각 지정하여 함수의 인자로 전달한다.

- 피벗테이블 함수
  - 피벗테이블 함수의 인자에는 각 2개 이상의 열을 입력할 수 있다.

```
피벗테이블 = pandas.pivot_table(데이터프레임,
                                index='행 위치에 들어갈 열',
                                columns='열 위치에 들어갈 열',
                                value='데이터로 사용할 열',
                                aggfunc='데이터 집계 함수')
```
이때 데이터 집계 함수에 ()표시는 넣지 않는다. 예컨대 aggfunc='mean'과 같다.