01장. 웹에서 주문 수를 분석하는 테크닉

> 💡
> **REMEMBER**

- 구체적인 목적에 상관없이 데이터 전체를 파악하는 것이 중요. 이를 위해 **되도록 상세하게 나와 있는 데이터를 기준으로 가공**하는 것이 좋음
- 데이터 조인시 , **추가하고 싶은 칼럼**이 무엇인지와 **공통되는 칼럼**(key)이 무엇인지 생각
- 데이터를 결합해서 계산해야 한다면 되도록 검산이 가능한 데이터 칼럼을 찾고 , 계산 후 검산하기
- 분석을 위해 데이터를 파악할 때는 **결손치의 개수**와 **전체를 파악할 수 있는 숫자**감을 위주로 , 추가적으로 **데이터의 기간**도 파악하면 좋음
- 분석은 **시계열 부터** 진행하는 것이 효율적.
  이 때, 전체 데이터를 한 번에 분석하면 시계열 변화를 잘못 파악할 수 있으므로 범위로 좁혀서 분석하는 것이 하나의 방법

> 📚 **Plus Topic**

🔶 datetime
🔶 pivot table
🔶 **groupby(그룹 분석)**

🔶 datetime

- Reference

  https://steadiness-193.tistory.com/227

  https://moondol-ai.tistory.com/180

  https://docs.python.org/ko/3/library/datetime.html#strftime-strptime-behavior

  https://dojang.io/mod/page/view.php?id=2463

  https://datascienceschool.net/01 python/02.15 파이썬에서 날짜와 시간 다루기.html

1. **datetime 변환**

- 데이터가 날짜 정보를 담고 있더라도 대부분 object형으로 저장되어 있음
- datetime형으로 변환해줘야 편리

▼ **pd.to_datetime()** : datetime 형으로 변환

- infer_datetime_format = True : 포맷을 자동으로 확인하고 변환
- format = '%d/%m/%y' : 포맷을 직접 지정
  - format입력시 구분되어 있는 문자열을 동일하게 입력

시간 형식 지정자	의미	결과
%a	요일 (짧은 이름)	Sun, Mon, ..., Sat
%A	요일 (긴 이름)	Sunday, Monday...
%w	요일 (숫자, 0부터 일요일)	0, 1, 2, 3, 4, 5, 6
%d	날짜 (2자리)	01, 02, ..., 31
%b	월 (짧은 이름)	Jan, Feb, ..., Dec
%B	월 (긴 이름)	January, Feburary,...
%m	월 (숫자)	01, 02, ..., 12
%y	연 (2자리)	00, 01, ..., 99
%Y	연 (4자리)	1960, 2002, 2020, ...
%H	시간 (24시간)	00, 01, ..., 23
%I	시간 (12시간)	01, 02, ..., 12
%M	분 (2자리)	00, 01, ..., 59
%S	초 (2자리)	00, 01, ..., 59

2. **필요한 데이터 뽑아내서 사용하기**
   - dataframe의 칼럼에서 뽑아 내야 하는 경우 **dt**연산자를 붙여서 사용하고,
     하나의 행에서만 뽑아 내는 경우 바로 사용 가능
   - .year / .month / .month_name / .day / .quarter / .weekday_name
   - .hour / .minute / .second
   - .is_month_start / .is_month_end / .is_quarter_start / .is_quarter_end / .is_year_start / .is_year_end
   - .date : YYYY-MM-DD
   - .weekday : 요일을 숫자로(월요일 0부터 시작)
   - .weekofyear: 연 기준으로 몇 주째인지
   - .dayofweek : 연기준 몇 일째인지
   - .days_in_month : 월 일수
   - .is_leap_year : 윤년 여부

3. **파이썬 내장 모듈 datetime**

   자료형

   ▼ **date**
   - 이상적인 달력에서의 날짜
   - **date(year,month,day)**
   - 모든 인자가 필수이며, 일반적인 날짜의 범위를 벗어날 경우 ValueError 발생

   - Class Method
     - **today()** : 현재 지역 날짜 반환 (=date.fromtimestamp(time.time())
     - **fromtimestamp(timestamp)** : 해당하는 지역 날짜를 반환
       localtime() 함수에서 지원하는 값 범위를 벗어나면 OverflowError 발생
     - **fromordinal(ordinal)** : 역산 그레고리력 서수에 해당하는 date 반환
     - **fromisoformat(date_string)** : YYYY-MM-DD 형식으로 제공된 date_string에 해당하는 date 반환

   - Class attribute

    - .min : 표현 가능한 가장 이른 date
    - .max : 표현 가능한 가장 낮은 date
    - .resolution : 같지 않은 date 객체 간의 가능한 가장 작은 차이

- Instance Method
    - **replace(year,month,day)**
    - **timetuple()** : time.struct_time을 반환
    - **toordinal()** : 역산 그레고리력 서수
    - **weekday()** : 정수로 요일 반환 (월요일 0)
    - **isoweekday()** : 정수로 요일 반환 (월요일 1)
    - **isocalendar()** : year,week 및 weekday가 있는 named tuple 객체 반환
    - **isoformat()** : 날짜를 나타내는 문자열을 반환
    - **str()** : isoformat()과 동일
    - **ctime()** : 날짜를 나타내는 문자열을 반환
    - **strftime(format)** : 날짜를 나타내는 문자열을 반환
    - **__format(format)** : strftime과 동일

    - Instance attribute
        - .year
        - .month
        - .day

▼ time

- 특정 날짜와 관계없는 시간을 나타냄
- tzinfo 객체를 통해 조정 가능
- time(hour=0,minute=0,second=0,microsecond=0,tzinfo=None,*,fold=0)
- tzinfo의 기본값만 None, 일반적인 범위를 넘어가면 ValueError 발생

    - Class attribute
        - .min
        - .max
        - .resolution

- Instance Method
    - **replace(hour,minute,secod,microsecond,tzinfo,*,fold=0)**
    - **isoformat(timespec='auto')** : 시간을 나타내는 문자열 반환
    - **__ str__()**
    - **strftime(format)**
    - **__ format__(format)**
    - **utcoffset()**
    - **dst()**
    - **tzname()**

- Instance attribute
  - .hour
  - .minute
  - .second
  - .microsecond
  - .tzinfo
  - .fold

▼ **datetime**

- date 객체와 time 객체의 모든 정보를 포함하는 단일 객체
- **datetime(year,month,day,hour=0,minute=0,second=0,microsecond=0,tzinfo=None,*,fold=0)**
- year,month,day 인자는 필수 , 일반적 범위를 넘어가면 ValueError 발생

- Class Method
  - **today()**
  - **now(**)
  - **utcnow()** : tzinfo가 None인 현재 UTC 날짜와 시간 반환
  - **fromtimestamp(timestamp,tz=None)**
  - **utcfromtimestamp(timestamp)**
  - **fromordinal(ordinal)**
  - **combine(date,time,tzinfo)**
  - **fromisoformat(date_string)**
  - **fromisocalendar(year,week,day)**
  - **strptime(date_string,format)**

- Class attribute
  - .min
  - .max
  - .resolution

- Instance Method
  - **replace(year,month,day,hour,minute,second,microsecond,tzinfo,*,fold=0)**
  - **astimezone(tz=None)** : 새로운 tzinfo를 갖는 datetime 객체 반환
  - **utcoffset()** : tzinfo가 None이면 None 반환 , 그렇지 않으면 self.tzinfo.utcoffset(self) 반환
  - **dst()** : tzinfo가 None이면 None 반환, 그렇지 않으면 self.tzinfo.dst(self) 반환
  - **tzname()** : tzinfo가 None이면 None 반환, 그렇지 않으면 self.tzinfo.tzname(self) 반환
  - **timetuple()** : time.struct_time 반환
  - **utctimetuple()**
  - **toordinal()**
  - **timestamp()**
  - **weekday()**

    - **isoweekday()**
    - **isocalendar()**
    - **isoformat(sep,timespec='auto')**
    - **__ str__()**
    - **ctime()**
    - **strftime(format)**
    - **__ format__(format)**

- Instance attribute
    - .year
    - .month
    - .day
    - .hour
    - .minute
    - .second
    - .microsecond
    - .tzinfo
    - .fold

▼ **timedelta**

- 두 날짜나 시간의 차이인 기간을 나타냄
- **timedelta(days=0,seconds=0,microseconds=0,milliseconds=0,minutes=0,hours=0,weeks=0)**
- days,seconds,microseconds는 정규화되어 나타내짐

- Class attribute
    - .min : 가장 음수인 timedelta 객체
    - .max : 가장 양수인 timedelta 객체
    - .resolution : 같지 않은 timedelta 객체 간의 가능한 가장 작은 차이
- Instance attribute
    - .days
    - .seconds
    - .microseconds

▼ **tzinfo**

- 특정 시간대에 대한 정보를 캡쳐하고 싶을 때 사용하는 추상 베이스 클레스
- datetime과 time 객체의 생성자에 전달 가능

아래의 method들을 구현해야 함.

- **utcoffset(dt)** : 지역 시간의 UTC로부터 오프셋을 UTC의 동쪽에 있을 때 양의 값을 갖는 timedelta 객체로 반환
- **dst(dt)** : 일광 절약 시간(dst) 조정
- **tzname(dt)** : 객체 dt에 해당하는 시간대 이름을 문자열로 반환
- **fromutc(dt)** : 날짜와 시간 데이터를 조정하여 self의 지역 시간으로 동등한 datetime 반환

▼ **timezone**

- tzinfo의 sub class
- 각 인스턴스는 UTC로부터의 고정 오프셋으로 정의된 시간대를 나타냄
- **timezone(offset,name=None)**

- **utcoffset(dt)** : timezone 인스턴스가 구축될 때 지정된 고정값 반환
- **tzname(dt)**
- **dst(dt)** : 항상 None 반환
- **fromutc(dt)** : dt+offset을 반환
- **utc**

strftime()과 **strptime()**

- strftime() : 주어진 포맷에 따라 datetime객체를 문자열로 변환
- strptime() : 주어진 포맷으로 문자열을 datetime객체로 구문 분석

- strftime()은 date, datetime , time 객체 모두 지원
- strptime()이 반환하는 객체는 datetime

```
time.strftime('%Y-%m-%d',time.localtime(time.time()))
datetime.datetime.strptime('2018-05-19','%Y-%m-%d')
```

4. **dateutil 패키지**

- strptime()을 사용할 때 형식 문자열을 사용자가 제공해야함
- dateutil 패키지의 parse 함수를 쓰면 자동으로 형식 문자열을 찾아 datetime 클래스 객체를 만들어줌

```
from dateutil.parser import parse

parse('2021-07-01')
parse("Apr 16, 2016 04:05:32 PM")
```

🔶 pivot table

- Reference

  https://ko.wikipedia.org/wiki/피벗_테이블

  https://wikidocs.net/46755

  https://datascienceschool.net/01 python/04.07 피봇테이블과 그룹분석.html

1. **정의**
   - 커다란 표의 데이터를 요약하는 통계표
   - 데이터의 합계,평균 등이 포함
   - 데이터 처리의 한 기법
   - 유용한 정보에 집중할 수 있도록 통계를 정렬 또는 재정렬

2. 적용 : **pd.pivot_table()**
    - pivot table을 만들기 위해서는 grouping이 필수적
    - 그룹핑하고 싶은 칼럼명(=기준)을 **index** 파라미터에 입력
      하나일 경우에는 ' ' 로 입력 , 두 개 이상의 경우 리스트 형태로 입력
    - 데이터의 평균값이 아닌 다른 통계를 보기 위해서는 **aggfunc** 파라미터 입력
        - 합계, 개수 , 표준편차 , 분산 등
        - aggfunc ='sum' / aggfunc =sum / aggfunc =[sum] / aggfunc=['sum']
        - 모두 가능하나, 리스트로 설정하면 칼럼명이 상단에 나오고 설정하지 않으면 칼럼명이 상단에 나오지 않음
    - 특정 칼럼의 값만 보고 싶은 경우 **values**에 칼럼 이름 지정
    - 관심 있는 values를 추가로 구분하고 싶으면 선택 옵션 **columns** 지정
    - **fill_value** 옵션을 통해 NaN 값 처리
    - **margins=True**를 지정하면 총계가 보여짐 , **margins_name**으로 이름 지정 가능
    - pivot_table의 결과를 인덱스에 대해 필터링 가능
        - **query()** 함수 이용
        - query() 함수 안의 전체를 쌍따옴표로 묶고, 필터링 하고 싶은 인덱스 값 리스트 처리

```
df1 = pd.pivot_table(df,index=['계정코드','계정과목'],values= ~)
df1.query("계정과목 ==['제품매출','상품매출']")
```

    - pivot_table 결과의 순서 수정
        - pivot_table에 대해 칼럼 인덱스 구조 파악
        - 원하는 순서로 수정

```
df1.columns
df1.reindex_axis(['차변금액','대변금액'],level=1,axis=1)
```

3. pivot 메소드
    - pandas의 **pivot()**

연도	2005	2010	2015
도시			
부산	3512547.0	3393191.0	3448737.0
서울	9762546.0	9631482.0	9904312.0
인천	NaN	263203.0	2890451.0

        - 행 인덱스로 사용할 열 이름, 열 인덱스로 사용할 열 이름, 데이터로 사용할 열 이름 순서로 입력
        - 행 인덱스나 열 인덱스를 리스트로 지정하면 다중 인덱스 피봇 테이블을 생성
        - 행 인덱스와 열 인덱스 조건을 만족하는 데이터가 2개 이상이라면 에러 발생 → group analysis 진행

- set_index와 unstack을 조합해도 동일한 결과
    - set_index 안에는 행 인덱스, 열 인덱스 순서로 입력
    - set_index 결과에서 데이터로 사용할 열 추출
    - 해당 결과에 unstack 명령

```
df.pivot("도시","연도","인구")
df.set_index(["도시","연도"])[["인구"]].unstack()
```

🔶 groupby(그룹 분석)

- Reference

  https://datascienceschool.net/01 python/04.07 피봇테이블과 그룹분석.html

1. **관련 개념 정리**
   - 피봇테이블이란 두 개의 열을 각각 행 인덱스, 열 인덱스로 사용하여 데이터를 조회한 것 (pivot)
   - 행,열 조건을 만족하는 데이터가 하나 이상이라면 그룹 분석(groupby)
   - pivot과 groupby의 중간 성격을 가지는 것이 pivot_table
     : groupby처럼 그룹 분석을 하지만 최종적으로는 pivot처럼 피봇테이블을 만듬 (groupby 결과에 unstack을 자동 적용하여 2차원적인 형태로 변환)

2. **그룹 분석의 메서드**
   - 분석하고자 하는 시리즈나 데이터 프레임을 그룹화
   - 그룹 객체에 대해 그룹 연산을 수행

   - 그룹연산 메서드 결과에 **unstack** 명령을 추가하면 피봇 테이블 형태
     - 펼칠 칼럼(피봇 테이블의 열)을 문자열로 지정
   - 그룹연산의 메서드는 그룹 데이터를 나타내는 GroupBy 객체에 사용 가능
     - **size**,**count**
     - **mean**, **median**, **min**, **max**
     - **sum**, **prod**,**std**,**var**,**quantile**
     - **first**,**last**
     - **agg**,**aggregate** : 원하는 그룹연산이 없는 경우 직접 함수를 만들고 전달 , 여러 그룹 연산을 동시에 하고 싶은 경우 리스트 전달

   ▼ **describe** : 하나의 그룹 대표값이 아니라 여러 개의 값을 데이터프레임으로 구함

```
iris.groupby(iris.species).describe().T
```

species		setosa	versicolor	virginica
sepal_length	count	50.000000	50.000000	50.000000
	mean	5.006000	5.936000	6.588000
	std	0.352490	0.516171	0.635880
	min	4.300000	4.900000	4.900000
	25%	4.800000	5.600000	6.225000
	50%	5.000000	5.900000	6.500000
	75%	5.200000	6.300000	6.900000
	max	5.800000	7.000000	7.900000
sepal_width	count	50.000000	50.000000	50.000000
	mean	3.428000	2.770000	2.974000
	std	0.379064	0.313798	0.322497
	min	2.300000	2.000000	2.200000
	25%	3.200000	2.525000	2.800000
	50%	3.400000	2.800000	3.000000
	75%	3.675000	3.000000	3.175000

▼ **apply** : describe와 결과는 동일하나 원하는 그룹연산이 없는 경우 사용

```python
def top3_petal_length(df):
    return df.sort_values(by="petal_length", ascending=False)[:3]

iris.groupby(iris.species).apply(top3_petal_length)
```

		sepal_length	sepal_width	petal_length	petal_width	species
species						
setosa	24	4.8	3.4	1.9	0.2	setosa
	44	5.1	3.8	1.9	0.4	setosa
	23	5.1	3.3	1.7	0.5	setosa
versicolor	83	6.0	2.7	5.1	1.6	versicolor
	77	6.7	3.0	5.0	1.7	versicolor
	72	6.3	2.5	4.9	1.5	versicolor
virginica	118	7.7	2.6	6.9	2.3	virginica
	117	7.7	3.8	6.7	2.2	virginica
	122	7.7	2.8	6.7	2.0	virginica

▼ **transform** : 그룹별 계산을 통해 데이터 자체를 변형

```python
def q3cut(s):
    return pd.qcut(s, 3, labels=["소", "중", "대"]).astype(str)


iris["petal_length_class"] = iris.groupby(iris.species).petal_length.transform(q3cut)
iris[["petal_length", "petal_length_class"]].tail(10)
```

	petal_length	petal_length_class
140	5.6	중
141	5.1	소
142	5.1	소
143	5.9	대
144	5.7	중
145	5.2	소
146	5.0	소
147	5.2	소
148	5.4	중
149	5.1	소