데이터 분석 기초

- 데이터 정제
  - 매출, 고객 데이터 분석
  - 데이터 오류 및 결측치
  - 오류/결측 데이터 수정 방법
  - 분석용 데이터 생성
- 데이터 가로 결합(조인)
  - pd.merge()

2021년 4월 22일

대리점 데이터 가공

- **사용 데이터**
  - 대리점 매출 이력
  - 고객 정보 데이터
- **목적**
  - 데이터 분석과 예측을 위한 데이터 가공하는 방법 습득
- **가공 이유**
  - 시스템에 의해 관리되는 데이터와는 다르게 대리점 데이터는 수작업이 들어감
    - 날짜 등의 입력 실수 발생 가능성이 높아짐
    - 데이터 누락 발생
    - 오류가 많이 포함됨
  - 기존 데이터 예와 동일하게 처리시 잘못된 결과가 도출되거나 데이터 가공 처리가 안되는 문제 발생 가능성이 높아짐
    - 공백 차이도 데이터 분석 시 오작동 발생
    - 비즈니스 실무 현장에서 엑셀로 작업하는 경우 기계적인 입력 확인이 선택적임
    - 각각 다른 부서에서 수집된 데이터를 다루는 경우 통합에 어려움 발생
      - » 데이터 형식, 크기 등이 서로 다름

대리점 데이터 사전 분석

- 상품 26가지 : A~Z
- 매출 이력과 고객정보 데이터는 담당 사원이 시스템에 직접 입력
- 집계 기간에 상품 단가의 변동이 없었음
- 매출 이력 : 시스템에서 csv 파일로 출력함
- 고객 정보 : 대리점 관리자가 주 별로 집계하여 엑셀로 관리

파일이름	개요
out.csv	매출 이력 기간 : 2019년 1월 ~2019년 7월
cust.xlsx	대리점에서 관리하는 고객 정보

데이터 읽기

- **각 데이터를 pandas로 읽어 처음 5행 확인**

```
1 cust_data = pd.read_excel("cust.xlsx")
2 cust_data.head()
```

	고객이름	지역	등록일
0	김 현성	H시	2018-01-04 00:00:00
1	김 도윤	E시	42782
2	김 지한	A시	2018-01-07 00:00:00
3	김 하윤	F시	42872
4	김 시온	E시	43127

```
1 out_data = pd.read_csv("out.csv")
2 out_data.head()
```

	purchase_date	item_name	item_price	customer_name
0	2019-06-13 18:02	상품A	100.0	김가온
1	2019-07-13 13:05	상 품 S	NaN	김우찬
2	2019-05-11 19:42	상 품 a	NaN	김유찬
3	2019-02-12 23:40	상품Z	2600.0	김재현
4	2019-04-22 3:09	상품a	NaN	김강현

  - 데이터의 정합성에 문제가 있다.
    - item_name : 데이터 오류
    - item_price : 결측치 존재
    - 데이터에 나타나는 입력 오류나 표기 방법의 차이가 부정합을 일으킴

- **마지막 5행 확인**

데이터 정합성

- 데이터 오류 예

분류	컬럼	설명
날짜	2019-10-10 2019/10/10 10/10/2019 2019년 10월 10일	같은 날짜라도 포맷이 다르면 서로 다른 문자열 데이터 이런 오류를 자동으로 수정해 주는 언어도 있지만 여러 가지 포맷이 섞여 있는 경우 주의해야 함
이름	홍길동 홍길동 홍 길동 홍길 동 홍 길동	공백 유무로 시스템에서는 다른 데이터로 인식

- 데이터 오류를 제거하고 정합성을 보장하는 것이 데이터 분석의 기초
- 애매모호한 상태로 분석하면 결과의 신빙성이나 신뢰성을 보장 못함
- 정합성을 갖추기 위해서는 먼저 데이터의 속성이나 의미를 이해해야 함
- 매출 이력

	purchase_date	item_name	item_price	customer_name
0	2019-06-13 18:02	상품A	100.0	김가은
1	2019-07-13 13:05	상 품 S	NaN	김우찬
2	2019-05-11 19:42	상 품 a	NaN	김유찬
3	2019-02-12 23:40	상품Z	2600.0	김재현
4	2019-04-22 3:09	상품a	NaN	김강현

컬럼 확인

오류 파악

데이터 오류 찾기

▪ 매출 이력

▪ item_name 추출 후 오류 확인

```
1  out_data["item_name"].head()
```

```
0     상품A
1    상 품 S
2    상 품 a
3     상품Z
4     상품a
Name: item_name, dtype: object
```

알파벳 대소문자 섞여 있음
공백이 포함되어 있음

– 상품A, 상품a, 상품 a는 각각 다른 상품으로 집계됨

▪ item_price 추출 후 오류 확인

```
1  out_data["item_price"].head()
```

```
0     100.0
1       NaN
2       NaN
3    2600.0
4       NaN
Name: item_price, dtype: float64
```

결측치 NaN 있음

▪ 고객

	고객이름	지역	등록일
0	김 현성	H시	2018-01-04 00:00:00
1	김 도윤	E시	42782
2	김 지한	A시	2018-01-07 00:00:00
3	김 하윤	F시	42872
4	김 시온	E시	43127

고객 데이터의
오류 또는 결측치는?

원본 데이터 집계

- **매출 이력(out_data)에서 상품 별로 월 매출 건수를 구하는 집계**

```
1 out_data["purchase_date"] = pd.to_datetime(out_data["purchase_date"])
2 out_data["purchase_month"] = out_data["purchase_date"].dt.strftime("%Y%m")
3 res = out_data.pivot_table(index="purchase_month", columns="item_name", aggfunc="size", fill_value=0)
4 res
```

item_name	상품n	상품E	상품M	상품P	상품S	상품W	상품X	상품W	상품O	상품Q	...	상품k	상품l	상품o	상품p	상품r	상품s	상품t	상품v	상품x	상품y
purchase_month																					
201901	1	0	0	0	0	0	0	0	0	0	...	1	1	1	0	0	0	0	0	0	0
201902	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	1	1	1	0	0
201903	0	1	1	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
201904	0	0	0	0	0	0	0	1	0	1	...	0	0	0	0	0	1	0	0	0	0
201905	0	0	0	0	1	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	1
201906	0	0	0	0	0	1	0	0	0	0	...	0	0	0	1	0	0	0	0	1	0
201907	0	0	0	0	0	0	0	0	1	0	...	0	0	1	0	2	0	0	0	0	0

7 rows × 99 columns

- pd.to_datetime() : datetime 형으로 변환
- 새로운 컬럼 purchase_month에 년월 단위로 값 추가
  - 판다스 datetime의 dt를 사용해 년, 월 추출
  - strftime으로 년월 형식으로 추가
    - » %Y, %m, %d : 년, 월, 일

	purchase_date	item_name	item_price	customer_name
0	2019-06-13 18:02	상품A	100.0	김가은
1	2019-07-13 13:05	상 품 S	NaN	김우찬
2	2019-05-11 19:42	상 품 a	NaN	김유찬
3	2019-02-12 23:40	상품Z	2600.0	김재현
4	2019-04-22 3:09	상품a	NaN	김강현

```
1 out_data
```

	purchase_date	item_name	item_price	customer_name	purchase_month
0	2019-06-13 18:02:00	상품A	100.0	김가온	201906
1	2019-07-13 13:05:00	상 품 S	NaN	김우찬	201907
2	2019-05-11 19:42:00	상 품 a	NaN	김유찬	201905
3	2019-02-12 23:40:00	상품Z	2600.0	김재현	201902
4	2019-04-22 03:09:00	상품a	NaN	김강현	201904
...	...	...	...	...	...
2994	2019-02-15 02:56:00	상품Y	2500.0	김정민	201902
2995	2019-06-22 04:03:00	상품M	1300.0	김재원	201906
2996	2019-03-29 11:14:00	상품Q	NaN	김지율	201903
2997	2019-07-14 12:56:00	상품H	NaN	김승주	201907
2998	2019-07-21 00:31:00	상품D	400.0	정준기	201907

2999 rows × 5 columns

- 세로축 : 구입년월
- 가로축 : 상품명
- 집계 : 상품 건수
- 상품명의 대소문자에 따라 다른 상품으로 집계
- 상품 개수는 26개 => 결과는 99개로 나타남

item_name	상 품 n	상품 E	상품 M	상품 P	상품 S	상품 W	상품 X	상품 W	상 품 O	상 품 Q	...	상품 k	상품 l	상품 o	상품 p	상품 r	상품 s	상품 t	상품 v	상품 x	상품 y
purchase_month																					
201901	1	0	0	0	0	0	0	0	0	0	...	1	1	1	0	0	0	0	0	0	0
201902	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	1	1	1	0	0
201903	0	1	1	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
201904	0	0	0	0	0	0	0	1	0	1	...	0	0	0	0	0	1	0	0	0	0
201905	0	0	0	0	1	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	1
201906	0	0	0	0	0	1	0	0	0	0	...	0	0	0	1	0	0	0	0	1	0
201907	0	0	0	0	0	0	0	0	1	0	...	0	0	1	0	2	0	0	0	0	0

7 rows × 99 columns

원본 데이터 집계

- 매출 이력(out_data)에서 상품 별로 월 매출 합계를 구하는 집계

```
res = out_data.pivot_table(index="purchase_month",
                           columns="item_name",
                           values="item_price",
                           aggfunc="sum",
                           fill_value=0)
res
```

item_name	상품n	상품E	상품M	상품P	상품S	상품W	상품X	상품W	상품O	상품Q	...	상품k	상품l	상품o	상품p	상품r	상품s	상품t	상품v	상품x	상품y
purchase_month																					
201901	1400	0	0	0	0	0	0	0	0	0	...	1100	1200	1500	0	0	0	0	0	0	0
201902	0	0	0	0	0	0	2400	0	0	0	...	0	0	0	0	0	1900	2000	2200	0	0
201903	0	500	1300	1600	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
201904	0	0	0	0	0	0	0	2300	0	1700	...	0	0	0	0	0	1900	0	0	0	0
201905	0	0	0	0	1900	0	0	0	0	0	...	0	1200	0	0	0	0	0	0	0	2500
201906	0	0	0	0	0	2300	0	0	0	0	...	0	0	0	1600	0	0	0	0	2400	0
201907	0	0	0	0	0	0	0	0	0	0	...	0	0	1500	0	1800	0	0	0	0	0

7 rows × 99 columns

```
out_data
```

	purchase_date	item_name	item_price	customer_name	purchase_month
0	2019-06-13 18:02:00	상품A	100.0	김가온	201906
1	2019-07-13 13:05:00	상 품 S	NaN	김우찬	201907
2	2019-05-11 19:42:00	상 품 a	NaN	김유찬	201905
3	2019-02-12 23:40:00	상품Z	2600.0	김재현	201902
4	2019-04-22 03:09:00	상품a	NaN	김강현	201904
...	...	...	...	...	...
2994	2019-02-15 02:56:00	상품Y	2500.0	김정민	201902
2995	2019-06-22 04:03:00	상품M	1300.0	김재원	201906
2996	2019-03-29 11:14:00	상품Q	NaN	김지율	201903
2997	2019-07-14 12:56:00	상품H	NaN	김승주	201907
2998	2019-07-21 00:31:00	상품D	400.0	정준기	201907

2999 rows × 5 columns

- 세로축 : 구입년월
- 가로축 : 상품명
- 집계 : 월 매출 합계
- 데이터 오류 상태로 집계 및 분석 수행시 전혀 의미없는 결과 도출
- 데이터 가공 : 분석의 전처리 과정이므로 중요

상품명 오류 수정

- **공백 수정, 대문자로 수정**
- **현황 파악 => 수정 후의 결과가 올바른지 아닌지 판정하기 위해 중요**
- **상품명 개수 확인**

```
1  print(len(pd.unique(out_data["item_name"])))
```

```
99
```

  - item_name의 중복을 제외한 데이터 건수 : pd.unique() 사용 => 99개

- **데이터 오류 수정**

```
1  out_data["item_name"] = out_data["item_n
2  out_data["item_name"] = out_data["item_n
3  out_data["item_name"] = out_data["item_n
4  out_data.sort_values(by=["item_name"], a
```

  - str.upper() : 대문자로 변환
  - str.replace() : 공백 제거
  - sort_values()
    - item_name으로 정렬
    - 오름차순

	purchase_date	item_name	item_price	customer_name	purchase_month
0	2019-06-13 18:02:00	상품A	100.0	김가온	201906
1748	2019-05-19 20:22:00	상품A	100.0	김시훈	201905
223	2019-06-25 08:13:00	상품A	100.0	김유진	201906
1742	2019-06-13 16:03:00	상품A	100.0	김건희	201906
1738	2019-02-10 00:28:00	상품A	100.0	김하랑	201902
...	...	...	...	...	...
2880	2019-04-22 00:36:00	상품Y	NaN	김동욱	201904
2881	2019-04-30 14:21:00	상품Y	NaN	김하준	201904
1525	2019-01-24 10:27:00	상품Y	2500.0	김범준	201901
1361	2019-05-28 13:45:00	상품Y	2500.0	김수현	201905
3	2019-02-12 23:40:00	상품Z	2600.0	김재현	201902

2999 rows × 5 columns

상품명 오류 수정

- **결과 검증 필수**

```
1  print(len(pd.unique(out_data["item_name"])))
2  print(pd.unique(out_data["item_name"]))
```

```
26
['상품A' '상품S' '상품Z' '상품V' '상품O' '상품U' '상품L' '상품C' '상품I' '상품R' '상품X' '상품G'
 '상품P' '상품Q' '상품Y' '상품N' '상품W' '상품E' '상품K' '상품B' '상품F' '상품D' '상품M' '상품H'
 '상품T' '상품J']
```

금액의 결측치 수정

- **현장에서 조사를 거쳐 수정하거나 부서원이 결측치를 채워 넣는 등의 여러 가지 대응 가능**
- **이번 예에서는 프로그램으로 수정 가능**
  - 상품 단가가 집계 중에 변하지 않았기 때문에 가능
  - 데이터 내의 결측치 확인
    - is_null() : 각 데이터 셀의 결측치 유무 확인
    - any() : 결측치가 1개라도 있다면 참 반환
  - item_price가 True이므로 결측치 존재함

```
1  out_data.isnull().any()
```

```
purchase_date     False
item_name         False
item_price         True
customer_name     False
purchase_month    False
dtype: bool
```

금액의 결측치 수정

- 결측치는 같은 상품의 단가 이용 : 상품 단가가 집계 중에 변하지 않았기 때문에 가능
  - item_price 컬럼의 결측치 유무 확인
    - » flg_is_null : 어느 행에 결측치가 있는지 저장한 변수

```
1 flg_is_null = out_data["item_price"].isnull()
2 flg_is_null
```

```
0       False
1        True
2        True
3       False
4        True
        ...
2994    False
2995    False
2996     True
2997     True
2998    False
Name: item_price, Length: 2999, dtype: bool
```

금액의 결측치 수정

- 결측치가 있는 상품명 리스트 작성 : flg_is_null을 이용
  - loc을 사용하여 조건에 일치하는 데이터 추출 => 중복 제거 => 리스트로 생성
    - » out_data.loc[flg_is_null, 'item_name'] : 컬럼 item_name에서 flg_is_null 조건에 일치하는 행 추출

```
1 flg_is_null = out_data["item_price"].isnull()
2 flg_is_null
```

```
0       False
1        True
2        True
3       False
4        True
        ...
2994    False
2995    False
2996     True
2997     True
2998    False
Name: item_price, Length: 2999, dtype: bool
```

```
1 out_data.loc[flg_is_null, "item_name"]
```

```
1       상품S
2       상품A
4       상품A
6       상품A
14      상품A
        ...
2987    상품K
2990    상품O
2992    상품C
2996    상품Q
2997    상품H
Name: item_name, Length: 387, dtype: object
```

```
1 out_data.loc[flg_is_null, "item_name"].unique()
```

```
array(['상품S', '상품A', '상품P', '상품N', '상품W', '상품R', '상품I', '상품L', '상품F',
       '상품O', '상품B', '상품C', '상품V', '상품Q', '상품U', '상품K', '상품T', '상품X',
       '상품E', '상품M', '상품G', '상품J', '상품D', '상품H', '상품Y'], dtype=object)
```

금액의 결측치 수정

- 결측치가 있는 상품명 리스트에 가격 추가
  - 결측치가 있는 상품과 같은 상품이며, 금액이 입력되어 있는 행을 loc 로 찾기 => 금액 가져오기
  - 부정연산자 ~
    - » ~flg_is_null : flg_is_null==False로 사용 가능

prod_list 로 설정

```
1  list(out_data.loc[flg_is_null, "item_name"].unique())
```

```
['상품S',
 '상품A',
 '상품P',
 '상품N',
 '상품W',
 '상품R',
 '상품I',
 '상품L',
 '상품F',
 '상품O',
 '상품B',
 '상품C',
 '상품V',
 '상품Q',
 '상품U',
 '상품K',
 '상품T',
 '상품X',
 '상품E',
 '상품M',
 '상품G',
 '상품J',
 '상품D',
 '상품H',
 '상품Y']
```

```
1  for trg in prod_list:
2      price = out_data.loc[(~flg_is_null) & (out_data["item_name"] == trg), "item_price"].max()
3      print(trg, price)
```

```
상품S 1900.0
상품A 100.0
상품P 1600.0
상품N 1400.0
상품W 2300.0
상품R 1800.0
상품I 900.0
상품L 1200.0
상품F 600.0
상품O 1500.0
상품B 200.0
상품C 300.0
상품V 2200.0
상품Q 1700.0
상품U 2100.0
상품K 1100.0
상품T 2000.0
상품X 2400.0
상품E 500.0
상품M 1300.0
상품G 700.0
상품J 1000.0
상품D 400.0
상품H 800.0
상품Y 2500.0
```

```
1  flg_is_null = out_data["item_price"].isnull()
2  flg_is_null
```

```
0       False
1        True
2        True
3       False
4        True
        ...
2994    False
2995    False
2996     True
2997     True
2998    False
Name: item_price, Length: 2999, dtype: bool
```

```
1  out_data
```

	purchase_date	item_name	item_price	customer_name	purchase_month
0	2019-06-13 18:02:00	상품A	100.0	김가온	201906
1	2019-07-13 13:05:00	상 품 S	NaN	김우찬	201907
2	2019-05-11 19:42:00	상 품 a	NaN	김유찬	201905
3	2019-02-12 23:40:00	상품Z	2600.0	김재현	201902
4	2019-04-22 03:09:00	상품a	NaN	김강현	201904
...	...	...	...	...	...
2994	2019-02-15 02:56:00	상품Y	2500.0	김정민	201902
2995	2019-06-22 04:03:00	상품M	1300.0	김재원	201906
2996	2019-03-29 11:14:00	상품Q	NaN	김지율	201903
2997	2019-07-14 12:56:00	상품H	NaN	김승주	201907
2998	2019-07-21 00:31:00	상품D	400.0	정준기	201907

2999 rows × 5 columns

금액의 결측치 수정

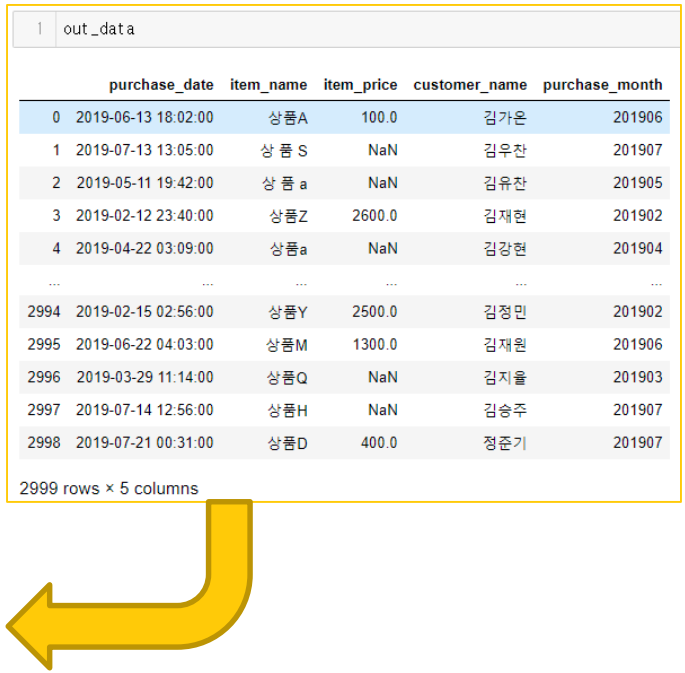
- 결측치가 있는 상품명 리스트에 가격 추가
  - 금액 가져온 후 해당 금액으로 데이터 수정

```
for trg in list(out_data.loc[flg_is_null, "item_name"].unique()):
    price = out_data.loc[(~flg_is_null) & (out_data["item_name"] == trg), "item_price"].max()
    out_data["item_price"].loc[(flg_is_null) & (out_data["item_name"]==trg)] = price
```

```
out_data
```

	purchase_date	item_name	item_price	customer_name	purchase_month
0	2019-06-13 18:02:00	상품A	100.0	김가온	201906
1	2019-07-13 13:05:00	상품S	1900.0	김우찬	201907
2	2019-05-11 19:42:00	상품A	100.0	김유찬	201905
3	2019-02-12 23:40:00	상품Z	2600.0	김재현	201902
4	2019-04-22 03:09:00	상품A	100.0	김강현	201904
...	...	...	...	...	...
2994	2019-02-15 02:56:00	상품Y	2500.0	김정민	201902
2995	2019-06-22 04:03:00	상품M	1300.0	김재원	201906
2996	2019-03-29 11:14:00	상품Q	1700.0	김지율	201903
2997	2019-07-14 12:56:00	상품H	800.0	김승주	201907
2998	2019-07-21 00:31:00	상품D	400.0	정준기	201907

2999 rows × 5 columns



금액의 결측치 수정

- 결측치가 있는 상품명 리스트에 가격 추가 후 검증

```
1  out_data.isnull().any()
```

```
purchase_date     False
item_name         False
item_price        False
customer_name     False
purchase_month    False
dtype: bool
```

- 각 상품의 금액이 정상적으로 수정되었는지 확인

```
1  for trg in list(out_data["item_name"].sort_values().unique()):
2      print(trg + "의최고가 : " + str(out_data.loc[out_data["item_name"]==trg]["item_price"].max())
3            + "의최저가 : " + str(out_data.loc[out_data["item_name"]==trg]["item_price"].min(skipna=False)))
```

```
상품A의최고가 : 100.0의최저가 : 100.0
상품B의최고가 : 200.0의최저가 : 200.0
상품C의최고가 : 300.0의최저가 : 300.0
상품D의최고가 : 400.0의최저가 : 400.0
상품E의최고가 : 500.0의최저가 : 500.0
상품F의최고가 : 600.0의최저가 : 600.0
상품G의최고가 : 700.0의최저가 : 700.0
상품H의최고가 : 800.0의최저가 : 800.0
상품I의최고가 : 900.0의최저가 : 900.0
상품J의최고가 : 1000.0의최저가 : 1000.0
상품K의최고가 : 1100.0의최저가 : 1100.0
상품L의최고가 : 1200.0의최저가 : 1200.0
상품M의최고가 : 1300.0의최저가 : 1300.0
상품N의최고가 : 1400.0의최저가 : 1400.0
상품O의최고가 : 1500.0의최저가 : 1500.0
상품P의최고가 : 1600.0의최저가 : 1600.0
상품Q의최고가 : 1700.0의최저가 : 1700.0
상품R의최고가 : 1800.0의최저가 : 1800.0
상품S의최고가 : 1900.0의최저가 : 1900.0
상품T의최고가 : 2000.0의최저가 : 2000.0
상품U의최고가 : 2100.0의최저가 : 2100.0
상품V의최고가 : 2200.0의최저가 : 2200.0
상품W의최고가 : 2300.0의최저가 : 2300.0
상품X의최고가 : 2400.0의최저가 : 2400.0
상품Y의최고가 : 2500.0의최저가 : 2500.0
상품Z의최고가 : 2600.0의최저가 : 2600.0
```

고객 이름 오류 수정

- **고객/매출 이력 데이터에서 고객 이름 확인**

```
1  cust_data["고객이름"].head()
```

```
0    김 현성
1    김 도윤
2    김 지한
3    김 하윤
4    김 시온
Name: 고객이름, dtype: object
```

```
1  out_data["customer_name"].head()
```

```
0    김가온
1    김우찬
2    김유찬
3    김재현
4    김강현
Name: customer_name, dtype: object
```

- 고객 데이터의 고객 이름 : 성과 이름 사이 공백 존재
  - 공백 : 1칸, 2칸, 없는 등의 서식이 혼재된 상태
  - 매출 이력과 고객 정보를 결합하면 정상적인 결합이 어려움
  - 고객 이름만이 키가 될 수 있음(out_data와 cust_data 사이의 공통 컬럼은 고객 이름 1개)
  - 이름의 공백 제거는 매출 이력 때와 동일

```
1  cust_data["고객이름"] = cust_data["고객이름"].str.replace("  ", "")
2  cust_data["고객이름"] = cust_data["고객이름"].str.replace(" ", "")
3  cust_data["고객이름"].head()
```

```
0    김현성
1    김도윤
2    김지한
3    김하윤
4    김시온
Name: 고객이름, dtype: object
```

날짜 오류 수정

- **고객 데이터의 등록일 오류 수정**
  - 날짜 서식이 여러 종류임
  - 날짜의 경우 엑셀에서는 다른 서식이 혼재할 가능성 있음
  - 고객 데이터의 날짜를 동일한 포맷으로 통일
    - 숫자로 된 등록일 위치 확인

```
1 flg_is_serial = cust_data["등록일"].astype("str").str.isdigit()
2 flg_is_serial.sum()
```

```
22
```

» astype() : 데이터프레임의 데이터 타입 변경
» str.isdigit() : 등록일이 숫자이면 참 반환

```
1 cust_data
```

	고객이름	지역	등록일
0	김 현성	H시	2018-01-04 00:00:00
1	김 도윤	E시	42782
2	김 지한	A시	2018-01-07 00:00:00
3	김 하윤	F시	42872
4	김 시온	E시	43127
...	...	...	...
195	김 재희	G시	2017-06-20 00:00:00
196	김 도영	E시	2018-06-20 00:00:00
197	김 이안	F시	2017-04-29 00:00:00
198	김 시현	H시	2019-04-19 00:00:00
199	김 서우	D시	2019-04-23 00:00:00

200 rows × 3 columns

```
1 cust_data["등록일"]
```

```
0      2018-01-04 00:00:00
1                    42782
2      2018-01-07 00:00:00
3                    42872
4                    43127
              ...
195    2017-06-20 00:00:00
196    2018-06-20 00:00:00
197    2017-04-29 00:00:00
198    2019-04-19 00:00:00
199    2019-04-23 00:00:00
Name: 등록일, Length: 200, dtype: object
```

```
1 flg_is_serial
```

```
0      False
1       True
2      False
3       True
4       True
       ...
195    False
196    False
197    False
198    False
199    False
Name: 등록일, Length: 200, dtype: bool
```

날짜 오류 수정

- **고객 데이터의 등록일 오류 수정**
  - 고객 데이터의 날짜를 동일한 포맷으로 통일
    - 숫자로 된 등록일을 날짜로 수정
      - » pd.to_timedelta() : 숫자를 날짜로 변경하는 함수
      - » 엑셀과 2일 차이 => 아래 코드에서 -2해야 함
        : 엑셀은 1부터 시작, 1900/02/29는 평년인데 넣어 둠

```
1  fromSerial = pd.to_timedelta(cust_data.loc[flg_is_serial, "등록일"].astype("float"), unit="D") + pd.to_datetime("1900/01/01")
2  fromSerial
```

-2

```
1      2017-02-18
3      2017-05-19
4      2018-01-29
21     2017-07-06
27     2017-06-17
47     2017-01-08
49     2017-07-15
53     2017-04-10
76     2018-03-31
80     2018-01-12
99     2017-06-01
114    2018-06-05
118    2018-01-31
122    2018-04-18
139    2017-05-27
143    2017-03-26
155    2017-01-21
172    2018-03-24
179    2017-01-10
183    2017-07-26
186    2018-07-15
192    2018-06-10
Name: 등록일, dtype: datetime64[ns]
```

```
1  cust_data["등록일"]
```

```
0      2018-01-04 00:00:00
1                    42782
2      2018-01-07 00:00:00
3                    42872
4                    43127
              ...
195    2017-06-20 00:00:00
196    2018-06-20 00:00:00
197    2017-04-29 00:00:00
198    2019-04-19 00:00:00
199    2019-04-23 00:00:00
Name: 등록일, Length: 200, dtype: object
```

```
1  flg_is_serial
```

```
0      False
1       True
2      False
3       True
4       True
       ...
195    False
196    False
197    False
198    False
199    False
Name: 등록일, Length: 200, dtype: bool
```

날짜 오류 수정

- **고객 데이터의 등록일 오류 수정**
  - 고객 데이터의 날짜를 동일한 포맷으로 통일
    - 날짜로 변환된 데이터의 서식 통일
      - » object(문자열) 형식의 등록일을 날짜 서식으로 변경

```
1  fromString = pd.to_datetime(cust_data.loc[~flg_is_serial, "등록일"])
2  fromString
```

```
0      2018-01-04
2      2018-01-07
5      2017-06-20
6      2018-06-11
7      2017-05-19
          ...
195    2017-06-20
196    2018-06-20
197    2017-04-29
198    2019-04-19
199    2019-04-23
Name: 등록일, Length: 178, dtype: datetime64[ns]
```

      - » 등록일을 세로로 합하기

```
1  cust_data["등록일"] = pd.concat([fromSerial, fromString])
2  cust_data
```

	고객이름	지역	등록일
0	김현성	H시	2018-01-04
1	김도윤	E시	2017-02-16
2	김지한	A시	2018-01-07
3	김하윤	F시	2017-05-17
4	김시은	E시	2018-01-27
...	...	...	...
195	김재희	G시	2017-06-20
196	김도영	E시	2018-06-20
197	김이안	F시	2017-04-29
198	김시현	H시	2019-04-19
199	김서우	D시	2019-04-23

200 rows × 3 columns

날짜 오류 수정

- **고객 데이터의 등록일 오류 수정**
  - 고객 데이터의 날짜를 동일한 포맷으로 통일
    - 등록일로부터 등록연월을 추출 후 집계

```
1  cust_data["등록연월"] = cust_data["등록일"].dt.strftime("%Y%m")
2  rslt = cust_data.groupby("등록연월").count()["고객이름"]
3  print(rslt)
4  print(len(cust_data))
```

피벗테이블로 변경 가능

```
등록연월
201701    15
201702    11
201703    14
201704    15
201705    14
201706    13
201707    17
201801    13
201802    15
201803    17
201804     5
201805    19
201806    13
201807    17
201904     2
Name: 고객이름, dtype: int64
200
```

합하면 200

  - 등록일 컬럼에 숫자가 남아있는지 다시 확인

```
1  flg_is_serial = cust_data["등록일"].astype("str").str.isdigit()
2  flg_is_serial.sum()
```

```
0
```

고객 이름을 키로 두 개의 데이터 결합(조인)

- 매출 이력 데이터와 고객 정보 데이터를 가로로 결합

```
1  join_data = pd.merge(out_data,
2                       cust_data,
3                       left_on="customer_name",
4                       right_on="고객이름",
5                       how="left")
6  join_data
```

```
1  join_data = join_data.drop("customer_name", axis=1)
2  join_data
```

	purchase_date	item_name	item_price	customer_name	purchase_month	고객이름	지역	등록일	등록연월
0	2019-06-13 18:02:00	상품A	100.0	김가온	201906	김가온	C시	2017-01-26	201701
1	2019-07-13 13:05:00	상품S	1900.0	김우찬	201907	김우찬	C시	2018-04-07	201804
2	2019-05-11 19:42:00	상품A	100.0	김유찬	201905	김유찬	A시	2018-06-19	201806
3	2019-02-12 23:40:00	상품Z	2600.0	김재현	201902	김재현	D시	2018-07-22	201807
4	2019-04-22 03:09:00	상품A	100.0	김강현	201904	김강현	D시	2017-06-07	201706
...	...	...	...	...	...	...	...	...	...
2994	2019-02-15 02:56:00	상품Y	2500.0	김정민	201902	김정민	B시	2017-07-01	201707
2995	2019-06-22 04:03:00	상품M	1300.0	김재원	201906	김재원	E시	2018-03-31	201803
2996	2019-03-29 11:14:00	상품Q	1700.0	김지율	201903	김지율	B시	2017-03-15	201703
2997	2019-07-14 12:56:00	상품H	800.0	김승주	201907	김승주	E시	2018-07-13	201807
2998	2019-07-21 00:31:00	상품D	400.0	정준기	201907	정준기	B시	2017-02-05	201702

2999 rows × 9 columns

	purchase_date	item_name	item_price	purchase_month	고객이름	지역	등록일	등록연월
0	2019-06-13 18:02:00	상품A	100.0	201906	김가온	C시	2017-01-26	201701
1	2019-07-13 13:05:00	상품S	1900.0	201907	김우찬	C시	2018-04-07	201804
2	2019-05-11 19:42:00	상품A	100.0	201905	김유찬	A시	2018-06-19	201806
3	2019-02-12 23:40:00	상품Z	2600.0	201902	김재현	D시	2018-07-22	201807
4	2019-04-22 03:09:00	상품A	100.0	201904	김강현	D시	2017-06-07	201706
...	...	...	...	...	...	...	...	...
2994	2019-02-15 02:56:00	상품Y	2500.0	201902	김정민	B시	2017-07-01	201707
2995	2019-06-22 04:03:00	상품M	1300.0	201906	김재원	E시	2018-03-31	201803
2996	2019-03-29 11:14:00	상품Q	1700.0	201903	김지율	B시	2017-03-15	201703
2997	2019-07-14 12:56:00	상품H	800.0	201907	김승주	E시	2018-07-13	201807
2998	2019-07-21 00:31:00	상품D	400.0	201907	정준기	B시	2017-02-05	201702

2999 rows × 8 columns

데이터 정제

- 데이터 가공으로 분석에 적합한 데이터 형태가 된 것을 데이터 정제라 함

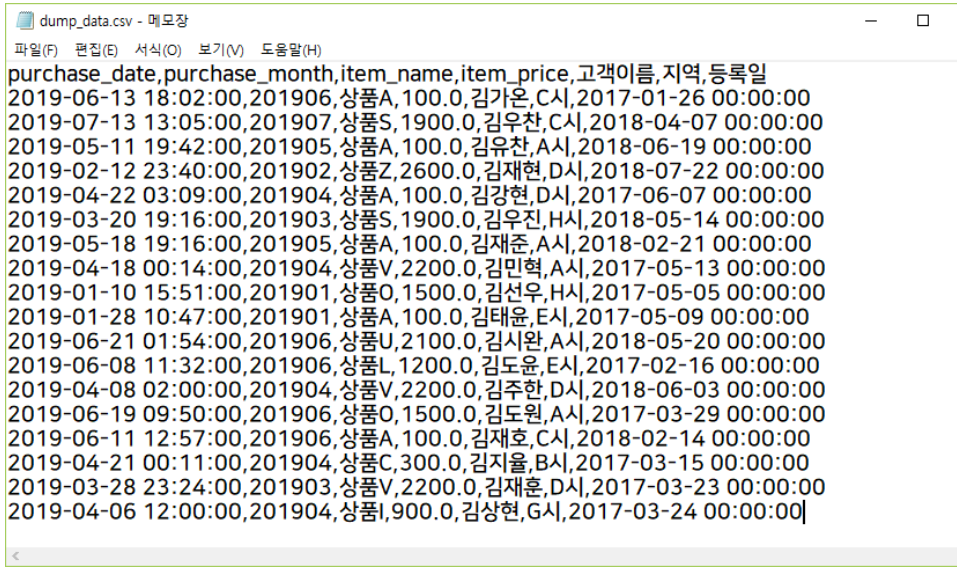
정제한 데이터 덤프

- 데이터 분석을 위해 정제한 데이터를 파일로 출력(덤프)
- 분석 시 출력한 파일을 읽어서 사용
- 데이터 정제 과정을 반복 할 필요 없음
- 덤프 전 컬럼의 순서를 변경
  - purchase_date와 purchase_month는 가까이 있는 편이 다루기 편리함
  - 컬럼 배치 조정 후 파일로 저장하는 것이 직관적으로 이해하기 쉬움

```
1 dump_data = join_data[["purchase_date", "purchase_month", "item_name", "item_price", "고객이름", "지역", "등록일"]]
2 dump_data
```

	purchase_date	purchase_month	item_name	item_price	고객이름	지역	등록일
0	2019-06-13 18:02:00	201906	상품A	100.0	김가온	C시	2017-01-26
1	2019-07-13 13:05:00	201907	상품S	1900.0	김우찬	C시	2018-04-07
2	2019-05-11 19:42:00	201905	상품A	100.0	김유찬	A시	2018-06-19
3	2019-02-12 23:40:00	201902	상품Z	2600.0	김재현	D시	2018-07-22
4	2019-04-22 03:09:00	201904	상품A	100.0	김강현	D시	2017-06-07
...	...	...	...	...	...	...	...
2994	2019-02-15 02:56:00	201902	상품Y	2500.0	김정민	B시	2017-07-01
2995	2019-06-22 04:03:00	201906	상품M	1300.0	김재원	E시	2018-03-31
2996	2019-03-29 11:14:00	201903	상품Q	1700.0	김지율	B시	2017-03-15
2997	2019-07-14 12:56:00	201907	상품H	800.0	김승주	E시	2018-07-13
2998	2019-07-21 00:31:00	201907	상품D	400.0	정준기	B시	2017-02-05

2999 rows × 7 columns

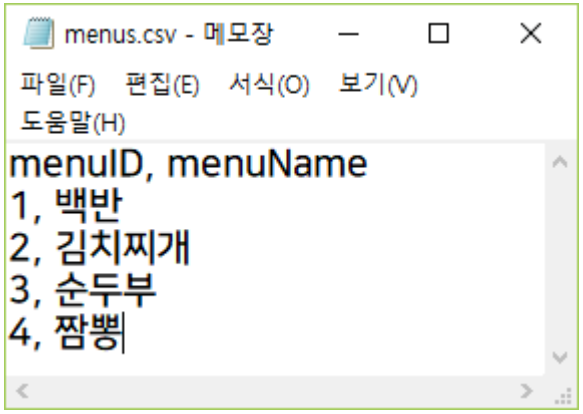


- 덤프

```
1 dump_data.to_csv("dump_data.csv", index=False)
```

가로 통합(조인, merge())

- 실습 파일 : menus.csv, meals.xlsx



	A	B	C
1	menuNo	selDate	amount
2	1	2011-05-01	4000
3	4	2011-05-01	3800
4	2	2011-05-02	4000
5	1	2011-05-03	4000
6	9	2011-05-05	2500
7	4	2011-05-05	3800

```
1  menu_list = pd.read_csv('menus.csv', encoding='cp949')
```

```
1  meal_list = pd.read_excel('meals.xlsx', header=0)
```

menu_list

menuID	menuName
1	백반
2	김치찌개
3	순두부
4	짬뽕

meal_list

menuNo	selDate	amount
1	2011-05-01	4000
4	2011-05-01	3800
2	2011-05-02	4000
1	2011-05-03	4000
9	2011-05-05	2500
4	2011-05-05	3800

memory

가로 통합(조인, merge())

- **meal_list에 menuName 추가**
  - cross join(크로스 조인) : menus✕meals

테이블 : menus

menuID	menuName
1	백반
2	김치찌개
3	순두부
4	짬뽕

테이블 : meals

menuID	selDate	amount
1	2011-05-01	4000
4	2011-05-01	3800
2	2011-05-02	4000
1	2011-05-03	4000
9	2011-05-05	2500
4	2011-05-05	3800

```
select *
from menus, meals
```

크로스 조인(cross join) : menus✕meals 결과

테이블 : menus✕meals

일련번호	menus.menuID	menuName	meals.menuID	selDate	amount
1	1	백반	1	2011-05-01	4000
2	1	백반	4	2011-05-01	3800
3	1	백반	2	2011-05-02	4000
4	1	백반	1	2011-05-03	4000
5	1	백반	9	2011-05-05	2500
6	1	백반	4	2011-05-05	3800
7	2	김치찌개	1	2011-05-01	4000
8	2	김치찌개	4	2011-05-01	3800
9	2	김치찌개	2	2011-05-02	4000
10	2	김치찌개	1	2011-05-03	4000
11	2	김치찌개	9	2011-05-05	2500
12	2	김치찌개	4	2011-05-05	3800
13	3				4000
14	3				3800
15	3	순두부	2	2011-05-02	4000
16	3	순두부	1	2011-05-03	4000
17	3	순두부	9	2011-05-05	2500
18	3	순두부	4	2011-05-05	3800
19	4	짬뽕	1	2011-05-01	4000
20	4	짬뽕	4	2011-05-01	3800
21	4	짬뽕	2	2011-05-02	4000
22	4	짬뽕	1	2011-05-03	4000
23	4	짬뽕	9	2011-05-05	2500
24	4	짬뽕	4	2011-05-05	3800

menus.menuID = meals.menuID

조인(join) : menus⋈meals

```
select *
from menus, meals
where menus.menuID = meals.menuID
```

테이블 : menus⋈meals

menus.menuID	menuName	meals.menuID	selDate	amount
1	백반	1	2011-05-01	4000
1	백반	1	2011-05-03	4000
2	김치찌개	2	2011-05-02	4000
4	짬뽕	4	2011-05-01	3800
4	짬뽕	4	2011-05-05	3800

가로 통합(조인, merge())

- **meal_list에 menuName 추가**
  - inner join : 조인 조건에 해당하는 행만 결과로 나타냄(조인에 참여하는 행만 출력)

```
1 meal_list
```

	menuNo	selDate	amount
0	1	2011-05-01	4000
1	4	2011-05-01	3800
2	2	2011-05-02	4000
3	1	2011-05-03	4000
4	9	2011-05-05	2500
5	4	2011-05-05	3800

⋈

```
1 menu_list
```

	menuID	menuName
0	1	백반
1	2	김치찌개
2	3	순두부
3	4	짬뽕

```
1 inner_join
```

	menuNo	selDate	amount	menuID	menuName
0	1	2011-05-01	4000	1	백반
1	1	2011-05-03	4000	1	백반
2	4	2011-05-01	3800	4	짬뽕
3	4	2011-05-05	3800	4	짬뽕
4	2	2011-05-02	4000	2	김치찌개

이렇게 출력하도록 변경하세요.

```
1 inner_join
```

	menuNo	menuID	menuName	selDate	amount
0	1	1	백반	2011-05-01	4000
1	1	1	백반	2011-05-03	4000
2	4	4	짬뽕	2011-05-01	3800
3	4	4	짬뽕	2011-05-05	3800
4	2	2	김치찌개	2011-05-02	4000

가로 통합(조인, merge())

- **meal_list에 menuName 추가**
  - left outer join : 조인에 참여하지 않는 meal_ list의 행도 출력

```
1 meal_list
```

	menuNo	selDate	amount
0	1	2011-05-01	4000
1	4	2011-05-01	3800
2	2	2011-05-02	4000
3	1	2011-05-03	4000
4	9	2011-05-05	2500
5	4	2011-05-05	3800

기준 데이터(테이블)

⋈

```
1 menu_list
```

	menuID	menuName
0	1	백반
1	2	김치찌개
2	3	순두부
3	4	짬뽕

```
1 left_join
```

	menuNo	selDate	amount	menuID	menuName
0	1	2011-05-01	4000	1.0	백반
1	4	2011-05-01	3800	4.0	짬뽕
2	2	2011-05-02	4000	2.0	김치찌개
3	1	2011-05-03	4000	1.0	백반
4	9	2011-05-05	2500	NaN	NaN
5	4	2011-05-05	3800	4.0	짬뽕

```
1 inner_join
```

	menuNo	selDate	amount	menuID	menuName
0	1	2011-05-01	4000	1	백반
1	1	2011-05-03	4000	1	백반
2	4	2011-05-01	3800	4	짬뽕
3	4	2011-05-05	3800	4	짬뽕
4	2	2011-05-02	4000	2	김치찌개

가로 통합(조인, merge())

- **meal_list에 menuName 추가**
  - right outer join : 조인에 참여하지 않는 menu_list의 행도 출력

```
1 meal_list
```

	menuNo	selDate	amount
0	1	2011-05-01	4000
1	4	2011-05-01	3800
2	2	2011-05-02	4000
3	1	2011-05-03	4000
4	9	2011-05-05	2500
5	4	2011-05-05	3800

⋈

```
1 menu_list
```

	menuID	menuName
0	1	백반
1	2	김치찌개
2	3	순두부
3	4	짬뽕

기준 데이터(테이블)

```
1 right_join
```

	menuNo	selDate	amount	menuID	menuName
0	1.0	2011-05-01	4000.0	1	백반
1	1.0	2011-05-03	4000.0	1	백반
2	4.0	2011-05-01	3800.0	4	짬뽕
3	4.0	2011-05-05	3800.0	4	짬뽕
4	2.0	2011-05-02	4000.0	2	김치찌개
5	NaN	NaT	NaN	3	순두부

```
1 inner_join
```

	menuNo	selDate	amount	menuID	menuName
0	1	2011-05-01	4000	1	백반
1	1	2011-05-03	4000	1	백반
2	4	2011-05-01	3800	4	짬뽕
3	4	2011-05-05	3800	4	짬뽕
4	2	2011-05-02	4000	2	김치찌개

```
1 left_join
```

	menuNo	selDate	amount	menuID	menuName
0	1	2011-05-01	4000	1.0	백반
1	4	2011-05-01	3800	4.0	짬뽕
2	2	2011-05-02	4000	2.0	김치찌개
3	1	2011-05-03	4000	1.0	백반
4	9	2011-05-05	2500	NaN	NaN
5	4	2011-05-05	3800	4.0	짬뽕