파이썬 라이브러리를 활용한 데이터 분석

14장 데이터 분석 예제

2020.07.10금 2h

14장 데이터 분석 예제

미국 농무부
영양소 정보 분석

1h

음식의 영양소 정보 제공

- **미 농무부(USDA)**
  - 개발자 윌리엄스: JSON 형식으로 제공
- **각 음식**
  - 숫자로 된 고유 id
  - 제공량: portions
  - 영양소: nutrients
    - **그 음식이 제공하는 다양한 영양소를 정보**
      - **원소는 한 가지 영양소에 대한 정보**

분석 준비

- **내장 모듈 json**
  - 메소드 load로 읽기
- **db**
  - 여러 개(6636)의 사전이 모인 리스트
- **첫 원소**
  - 키 확인
  - 키 nutrients의 첫 원소 표시
  - 키 nutrients의 모든 원소를 DataFrame으로 생성

```
In [189]: import json
          db = json.load(open('datasets/usda_food/database.json'))
          len(db)
Out[189]: 6636
```

```
In [191]: db[0].keys()
Out[191]: dict_keys(['id', 'description', 'tags', 'manufacturer', 'group', 'portions', 'nutrients'])
```

```
In [192]: db[0]['nutrients'][0]
Out[192]: {'value': 25.18,
           'units': 'g',
           'description': 'Protein',
           'group': 'Composition'}
```

리스트 'nutrients'의 첫 원소

```
In [196]: nutrients = pd.DataFrame(db[0]['nutrients'])
          nutrients.head()
```

Out[196]:

	value	units	description	group
0	25.18	g	Protein	Composition
1	29.20	g	Total lipid (fat)	Composition
2	3.06	g	Carbohydrate, by difference	Composition
3	3.28	g	Ash	Other
4	376.00	kcal	Energy	Energy

```
In [195]: nutrients.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 162 entries, 0 to 161
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   value        162 non-null    float64
 1   units        162 non-null    object
 2   description  162 non-null    object
 3   group        162 non-null    object
dtypes: float64(1), object(3)
memory usage: 5.2+ KB
```

음식 그룹 분포 파악

- **DF info**
  - 모든 음식 정보에서 추출할 필드 목록을 4개 지정
    - **음식의 이름**
    - **그룹**
    - **Id**
    - **제조사**

- **메소드 value_counts()**
  - 음식 그룹의 분포 파악
    - **자동으로 내림차순으로 정렬**

```
In [198]: info_keys = ['description', 'group', 'id', 'manufacturer']
          info = pd.DataFrame(db, columns=info_keys)
          info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6636 entries, 0 to 6635
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   description   6636 non-null   object
 1   group         6636 non-null   object
 2   id            6636 non-null   int64
 3   manufacturer  5195 non-null   object
dtypes: int64(1), object(3)
memory usage: 207.5+ KB
```

```
In [199]: info.head()
```

Out[199]:

	description	group	id	manufacturer
0	Cheese, caraway	Dairy and Egg Products	1008	
1	Cheese, cheddar	Dairy and Egg Products	1009	
2	Cheese, edam	Dairy and Egg Products	1018	
3	Cheese, feta	Dairy and Egg Products	1019	
4	Cheese, mozzarella, part skim milk	Dairy and Egg Products	1028	

```
In [200]: pd.value_counts(info.group)[:10]
```

```
Out[200]: Vegetables and Vegetable Products    812
          Beef Products                        618
          Baked Products                       496
          Breakfast Cereals                    403
          Fast Foods                           365
          Legumes and Legume Products          365
          Lamb, Veal, and Game Products        345
          Sweets                               341
          Fruits and Fruit Juices              328
          Pork Products                        328
          Name: group, dtype: int64
```

모든 영양소 정보를 위한 데이터프레임 생성

- **모든 db의 원소(한 식품)에 대해서**
  - 음식의 영양소 리스트를 하나의 데이터프레임에 변환
  - 음식의 id 칼럼 추가
  - 리스트에 계속 데이터프레임 추가
- **이 리스트를 메소드 concat로 합침**
  - 총 389355 개
    - **식품은 6636개 이지만 이 음식의 영양소는 모두 389355 개**

In [212]:
```
nutrients = []

for rec in db:
    fnuts = pd.DataFrame(rec['nutrients'])
    fnuts['id'] = rec['id']
    nutrients.append(fnuts)

nutrients = pd.concat(nutrients, ignore_index=True)
nutrients.tail()
```

Out[212]:

	value	units	description	group	id
389350	0.000	mcg	Vitamin B-12, added	Vitamins	43546
389351	0.000	mg	Cholesterol	Other	43546
389352	0.072	g	Fatty acids, total saturated	Other	43546
389353	0.028	g	Fatty acids, total monounsaturated	Other	43546
389354	0.041	g	Fatty acids, total polyunsaturated	Other	43546

In [213]:
```
nutrients.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 389355 entries, 0 to 389354
Data columns (total 5 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   value        389355 non-null  float64
 1   units        389355 non-null  object
 2   description  389355 non-null  object
 3   group        389355 non-null  object
 4   id           389355 non-null  int64
dtypes: float64(1), int64(1), object(3)
memory usage: 14.9+ MB
```

중복된 데이터 제거

- **중복 데이터**
  - 14179 개

```
In [214]: nutrients.duplicated().sum()  # number of duplicates
Out[214]: 14179
```

```
In [215]: nutrients = nutrients.drop_duplicates()
          nutrients.tail()
```

Out[215]:

	value	units	description	group	id
389350	0.000	mcg	Vitamin B-12, added	Vitamins	43546
389351	0.000	mg	Cholesterol	Other	43546
389352	0.072	g	Fatty acids, total saturated	Other	43546
389353	0.028	g	Fatty acids, total monounsaturated	Other	43546
389354	0.041	g	Fatty acids, total polyunsaturated	Other	43546

```
In [211]: nutrients.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 375176 entries, 0 to 389354
Data columns (total 5 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   value        375176 non-null  float64
 1   units        375176 non-null  object
 2   description  375176 non-null  object
 3   group        375176 non-null  object
 4   id           375176 non-null  int64
dtypes: float64(1), int64(1), object(3)
memory usage: 17.2+ MB
```

두 데이터프레임 머지를 위해 중복되는 열 이름 수정

- **데이터프레임 info(음식 정보), nutrients(모든 음식의 영양소 정보)**

col_mapping = {'description' : 'food',
'group' : 'fgroup'}

col_mapping = {'description' : 'nutrient',
'group' : 'nutgroup'}

```
In [219]: info.head()
```

Out[219]:

	description	group	id	manufacturer
0	Cheese, caraway	Dairy and Egg Products	1008	
1	Cheese, cheddar	Dairy and Egg Products	1009	
2	Cheese, edam	Dairy and Egg Products	1018	
3	Cheese, feta	Dairy and Egg Products	1019	
4	Cheese, mozzarella, part skim milk	Dairy and Egg Products	1028	

```
In [220]: col_mapping = {'description' : 'food',
                        'group'       : 'fgroup'}
          info = info.rename(columns=col_mapping, copy=False)
          info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6636 entries, 0 to 6635
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   food          6636 non-null   object
 1   fgroup        6636 non-null   object
 2   id            6636 non-null   int64
 3   manufacturer  5195 non-null   object
dtypes: int64(1), object(3)
memory usage: 207.5+ KB
```

```
In [221]: nutrients.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 375176 entries, 0 to 389354
Data columns (total 5 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   value        375176 non-null  float64
 1   units        375176 non-null  object
 2   description  375176 non-null  object
 3   group        375176 non-null  object
 4   id           375176 non-null  int64
dtypes: float64(1), int64(1), object(3)
memory usage: 17.2+ MB
```

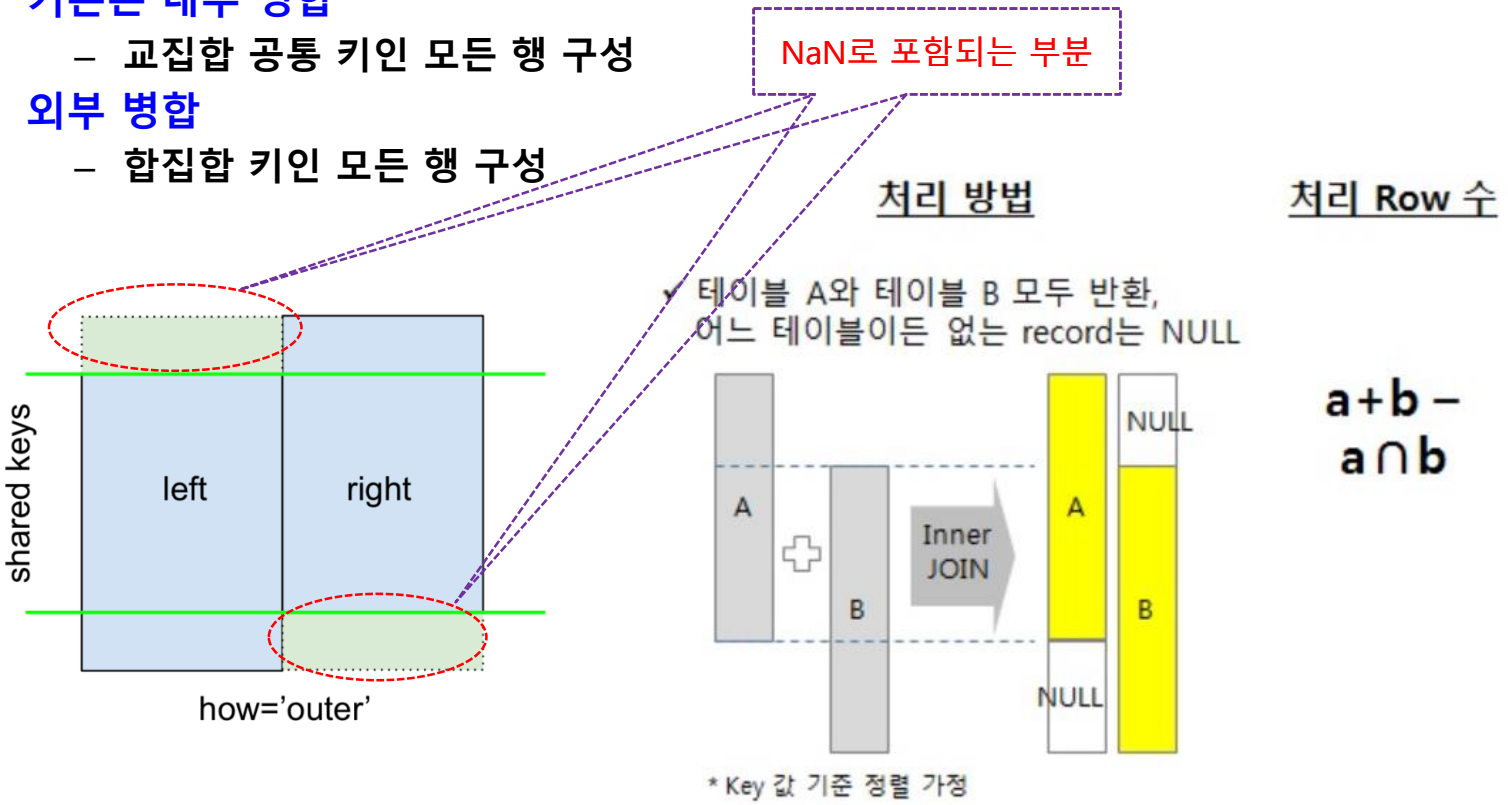
```
In [222]: col_mapping = {'description' : 'nutrient',
                         'group' : 'nutgroup'}
          nutrients = nutrients.rename(columns=col_mapping, copy=False)
          nutrients.head()
```

Out[222]:

	value	units	nutrient	nutgroup	id
0	25.18	g	Protein	Composition	1008
1	29.20	g	Total lipid (fat)	Composition	1008
2	3.06	g	Carbohydrate, by difference	Composition	1008
3	3.28	g	Ash	Other	1008
4	376.00	kcal	Energy	Energy	1008

Merge 병합 복습: 외부 병합

- **합치다 merge**
  - 두 데이터 프레임의 공통 열 혹은 인덱스를 기준으로 두 개의 테이블을 합침
  - 공통 키에 대한 합침
    - **기본은 내부 병합**
      - **교집합 공통 키인 모든 행 구성**
    - **외부 병합**
      - **합집합 키인 모든 행 구성**



두 테이블의 병합 결과 ndata

- **테이블 nutrients와 info**
  - on='id', how='outer' : 열 id가 공통인 행 모두를 합침

```
In [223]: ndata = pd.merge(nutrients, info, on='id', how='outer')
          ndata.head()
```

Out[223]:

	value	units	nutrient	nutgroup	id	food	fgroup	manufacturer
0	25.18	g	Protein	Composition	1008	Cheese, caraway	Dairy and Egg Products	
1	29.20	g	Total lipid (fat)	Composition	1008	Cheese, caraway	Dairy and Egg Products	
2	3.06	g	Carbohydrate, by difference	Composition	1008	Cheese, caraway	Dairy and Egg Products	
3	3.28	g	Ash	Other	1008	Cheese, caraway	Dairy and Egg Products	
4	376.00	kcal	Energy	Energy	1008	Cheese, caraway	Dairy and Egg Products	

```
In [224]: ndata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 375176 entries, 0 to 375175
Data columns (total 8 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   value         375176 non-null  float64
 1   units         375176 non-null  object
 2   nutrient      375176 non-null  object
 3   nutgroup      375176 non-null  object
 4   id            375176 non-null  int64
 5   food          375176 non-null  object
 6   fgroup        375176 non-null  object
 7   manufacturer  293054 non-null  object
dtypes: float64(1), int64(1), object(6)
memory usage: 25.8+ MB
```

영양소 수가 375176이며 id가 한쪽에만 있는 행이 없으므로 병합만 행도 375176임, 즉 영양소에 식품의 정보인 열이 추가된 것임

음식 그룹과 영양소 종류별 중간 값 그래프

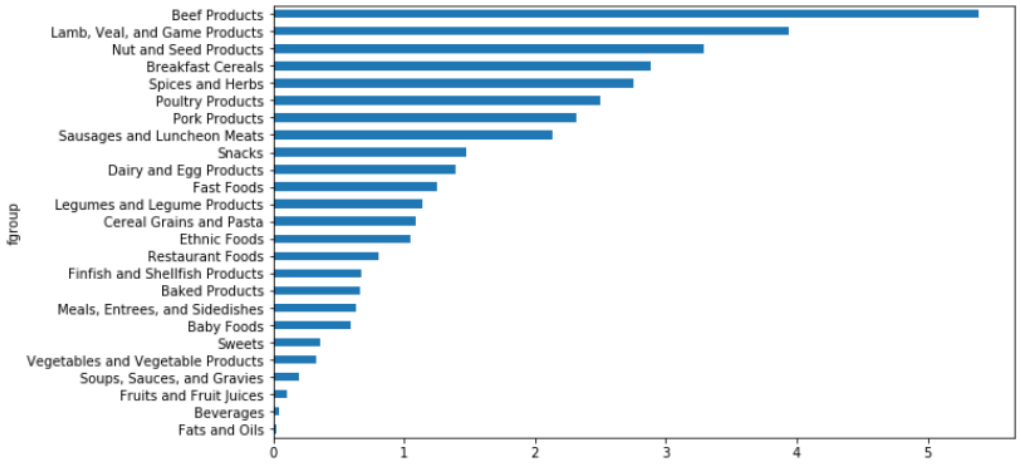
- **음식 그룹별 아연 함량의 중간 값**

```
In [227]: result = ndata.groupby(['nutrient', 'fgroup'])['value'].quantile(0.5)
          result
```

```
Out[227]: nutrient          fgroup
          Adjusted Protein  Sweets                                12.900
                            Vegetables and Vegetable Products      2.180
          Alanine           Baby Foods                            0.085
                            Baked Products                        0.248
                            Beef Products                         1.550
                                                                  ...
          Zinc, Zn          Snacks                                1.470
                            Soups, Sauces, and Gravies            0.200
                            Spices and Herbs                      2.750
                            Sweets                                0.360
                            Vegetables and Vegetable Products     0.330
          Name: value, Length: 2246, dtype: float64
```

```
In [228]: # fig = plt.figure()
          result['Zinc, Zn'].sort_values().plot(kind='barh')
```

```
Out[228]: <matplotlib.axes._subplots.AxesSubplot at 0x1e0f12f8ec8>
```



각 영양소가 어떤 음식에 가장 많이 들어 있는 지?

- **아미노산이 많이 들어가 있는 음식**

```
In [233]: by_nutrient = ndata.groupby(['nutgroup', 'nutrient'])
          by_nutrient

Out[233]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001E0D6769388>

In [234]: get_maximum = lambda x: x.loc[x.value.idxmax()]
          get_minimum = lambda x: x.loc[x.value.idxmin()]

          max_foods = by_nutrient.apply(get_maximum)[['value', 'food']]

          # make the food a little smaller
          max_foods.food = max_foods.food.str[:50]

In [235]: max_foods.loc['Amino Acids']['food']

Out[235]: nutrient
          Alanine                            Gelatins, dry powder, unsweetened
          Arginine                                Seeds, sesame flour, low-fat
          Aspartic acid                                  Soy protein isolate
          Cystine              Seeds, cottonseed flour, low fat (glandless)
          Glutamic acid                                  Soy protein isolate
                                                ...
          Serine           Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
          Threonine        Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
          Tryptophan        Sea lion, Steller, meat with fat (Alaska Native)
          Tyrosine         Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
          Valine           Soy protein isolate, PROTEIN TECHNOLOGIES INTE...
          Name: food, Length: 19, dtype: object
```

14장 데이터 분석 예제

2012년 연방선거관리 위원회 데이터베이스

1h

2012년 대선 정치활동 후원금

- **2012년 6월 기부자 정보**
  - 150메가 csv 파일
    - **P00000001-ALL.csv**
      - **백 만개 이상의 행: 1,001,731**
    - **정보: 열은 18개**
      - **후보 이름, 기부자 정보, 기부금액**
      - **직업, 고용주(형태), 주소**
- **read_csv()**
  - 옵션 low_memory
    - **low_memory bool, 기본 True**
    - **False**
      - **파일 전체를 읽음**

```
In [5]: fec = pd.read_csv('datasets/fec/P00000001-ALL.csv')
        fec.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001731 entries, 0 to 1001730
Data columns (total 16 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   cmte_id            1001731 non-null  object
 1   cand_id            1001731 non-null  object
 2   cand_nm            1001731 non-null  object
 3   contbr_nm          1001731 non-null  object
 4   contbr_city        1001712 non-null  object
 5   contbr_st          1001727 non-null  object
 6   contbr_zip         1001620 non-null  object
 7   contbr_employer    988002 non-null   object
 8   contbr_occupation  993301 non-null   object
 9   contb_receipt_amt  1001731 non-null  float64
 10  contb_receipt_dt   1001731 non-null  object
 11  receipt_desc       14166 non-null    object
 12  memo_cd            92482 non-null    object
 13  memo_text          97770 non-null    object
 14  form_tp            1001731 non-null  object
 15  file_num           1001731 non-null  int64
dtypes: float64(1), int64(1), object(14)
memory usage: 122.3+ MB
```

```
In [4]: pd.read_csv?
```

```
In [6]: fec.iloc[123456]
```

```
Out[6]: cmte_id                           C00431445
        cand_id                           P80003338
        cand_nm                       Obama, Barack
        contbr_nm                       ELLMAN, IRA
        contbr_city                           TEMPE
        contbr_st                                AZ
        contbr_zip                        852816719
        contbr_employer      ARIZONA STATE UNIVERSITY
        contbr_occupation                  PROFESSOR
        contb_receipt_amt                         50
        contb_receipt_dt                  01-DEC-11
        receipt_desc                            NaN
        memo_cd                                 NaN
        memo_text                               NaN
        form_tp                                SA17A
        file_num                              772372
        Name: 123456, dtype: object
```

모든 정당의 후보 목록

- **메소드 unique()**
- **소속 정당**
  - 사전 parties로 함수 map() 적용
- **열 'party' 추가**

```
In [14]: # Add it as a column
         fec['party'] = fec.cand_nm.map(parties)
         fec['party'].value_counts()

Out[14]: Democrat      593746
         Republican    407985
         Name: party, dtype: int64
```

```
In [8]: unique_cands = fec.cand_nm.unique()
        unique_cands

Out[8]: array(['Bachmann, Michelle', 'Romney, Mitt', 'Obama, Barack',
               "Roemer, Charles E. 'Buddy' III", 'Pawlenty, Timothy',
               'Johnson, Gary Earl', 'Paul, Ron', 'Santorum, Rick',
               'Cain, Herman', 'Gingrich, Newt', 'McCotter, Thaddeus G',
               'Huntsman, Jon', 'Perry, Rick'], dtype=object)

In [9]: unique_cands[2]

Out[9]: 'Obama, Barack'

In [12]: parties = {'Bachmann, Michelle': 'Republican',
                    'Cain, Herman': 'Republican',
                    'Gingrich, Newt': 'Republican',
                    'Huntsman, Jon': 'Republican',
                    'Johnson, Gary Earl': 'Republican',
                    'McCotter, Thaddeus G': 'Republican',
                    'Obama, Barack': 'Democrat',
                    'Paul, Ron': 'Republican',
                    'Pawlenty, Timothy': 'Republican',
                    'Perry, Rick': 'Republican',
                    "Roemer, Charles E. 'Buddy' III": 'Republican',
                    'Romney, Mitt': 'Republican',
                    'Santorum, Rick': 'Republican'}

In [10]: fec.cand_nm[123456:123461]

Out[10]: 123456    Obama, Barack
         123457    Obama, Barack
         123458    Obama, Barack
         123459    Obama, Barack
         123460    Obama, Barack
         Name: cand_nm, dtype: object

In [13]: fec.cand_nm[123456:123461].map(parties)

Out[13]: 123456    Democrat
         123457    Democrat
         123458    Democrat
         123459    Democrat
         123460    Democrat
         Name: cand_nm, dtype: object
```

전처리

- 기부 금액이 양수인 것만
  - 음수는 다시 돌려준 환급 금액
- 양대 후보의 것만 따로
  - 버락 오마마, 미트 롬니

```
In [15]: (fec.contb_receipt_amt > 0).value_counts()
```

```
Out[15]: True     991475
         False     10256
         Name: contb_receipt_amt, dtype: int64
```

```
In [18]: fec = fec[fec.contb_receipt_amt > 0]
         fec.head()
```

Out[18]:

	cmte_id	cand_id	cand_nm	contbr_nm	contbr_city	contbr_st	contbr_zip	contbr_employer	contbr_occupat
0	C00410118	P20002978	Bachmann, Michelle	HARVEY, WILLIAM	MOBILE	AL	3.6601e+08	RETIRED	RETIF
1	C00410118	P20002978	Bachmann, Michelle	HARVEY, WILLIAM	MOBILE	AL	3.6601e+08	RETIRED	RETIF
2	C00410118	P20002978	Bachmann, Michelle	SMITH, LANIER	LANETT	AL	3.68633e+08	INFORMATION REQUESTED	INFORMATI REQUEST
3	C00410118	P20002978	Bachmann, Michelle	BLEVINS, DARONDA	PIGGOTT	AR	7.24548e+08	NONE	RETIF
4	C00410118	P20002978	Bachmann, Michelle	WARDENBURG, HAROLD	HOT SPRINGS NATION	AR	7.19016e+08	NONE	RETIF

```
In [20]: fec_mrbo = fec[fec.cand_nm.isin(['Obama, Barack', 'Romney, Mitt'])]
         fec_mrbo.head()
```

Out[20]:

	cmte_id	cand_id	cand_nm	contbr_nm	contbr_city	contbr_st	contbr_zip	contbr_employer	contbr_occupatio
411	C00431171	P80003353	Romney, Mitt	ELDERBAUM, WILLIAM	DPO	AA	3.4023e+08	US GOVERNMENT	FOREIG SERVIC OFFICE
412	C00431171	P80003353	Romney, Mitt	ELDERBAUM, WILLIAM	DPO	AA	3.4023e+08	US GOVERNMENT	FOREIG SERVIC OFFICE
413	C00431171	P80003353	Romney, Mitt	CARLSEN, RICHARD	APO	AE	9.128e+07	DEFENSE INTELLIGENCE AGENCY	INTELLIGENC ANALYS
414	C00431171	P80003353	Romney, Mitt	DELUCA, PIERRE	APO	AE	9.128e+07	CISCO	ENGINEE
415	C00431171	P80003353	Romney, Mitt	SARGENT, MICHAEL	APO	AE	9.01201e+07	RAYTHEON TECHNICAL SERVICES CORP	COMPUTE SYSTEM ENGINEE

직업과 고용주에 따른 기부 통계 전처리

- **직업별 전체 기부 숫자**

In [21]:
```
fec.contbr_occupation.value_counts()[:10]
```

Out[21]:
```
RETIRED                                   233990
INFORMATION REQUESTED                      35107
ATTORNEY                                   34286
HOMEMAKER                                  29931
PHYSICIAN                                  23432
INFORMATION REQUESTED PER BEST EFFORTS     21138
ENGINEER                                   14334
TEACHER                                    13990
CONSULTANT                                 13273
PROFESSOR                                  12555
Name: contbr_occupation, dtype: int64
```

- **직업 매핑으로 직업을 단순화**
  - 사전에 정보가 없는 것은 그대로 사용
    - **메소드 get(x, x)**

In [22]:
```
occ_mapping = {
   'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
   'INFORMATION REQUESTED' : 'NOT PROVIDED',
   'INFORMATION REQUESTED (BEST EFFORTS)' : 'NOT PROVIDED',
   'C.E.O.': 'CEO'
}

# If no mapping provided, return x
f = lambda x: occ_mapping.get(x, x)
fec.contbr_occupation = fec.contbr_occupation.map(f)
```

- **고용주도 매핑으로 단순화**

In [23]:
```
emp_mapping = {
   'INFORMATION REQUESTED PER BEST EFFORTS' : 'NOT PROVIDED',
   'INFORMATION REQUESTED' : 'NOT PROVIDED',
   'SELF' : 'SELF-EMPLOYED',
   'SELF EMPLOYED' : 'SELF-EMPLOYED',
}

# If no mapping provided, return x
f = lambda x: emp_mapping.get(x, x)
fec.contbr_employer = fec.contbr_employer.map(f)
```

최소 200 백만 불 이상 기부한 직업

- 피벗 테이블 by_occupation

In [24]:
```
by_occupation = fec.pivot_table('contb_receipt_amt',
                                index='contbr_occupation',
                                columns='party', aggfunc='sum')
over_2mm = by_occupation[by_occupation.sum(1) > 2000000]
over_2mm
```

Out[24]:

party	Democrat	Republican
contbr_occupation		
ATTORNEY	11141982.97	7.477194e+06
CEO	2074974.79	4.211041e+06
CONSULTANT	2459912.71	2.544725e+06
ENGINEER	951525.55	1.818374e+06
EXECUTIVE	1355161.05	4.138850e+06
HOMEMAKER	4248875.80	1.363428e+07
INVESTOR	884133.00	2.431769e+06
LAWYER	3160478.87	3.912243e+05
MANAGER	762883.22	1.444532e+06
NOT PROVIDED	4866973.96	2.056547e+07
OWNER	1001567.36	2.408287e+06
PHYSICIAN	3735124.94	3.594320e+06
PRESIDENT	1878509.95	4.720924e+06
PROFESSOR	2165071.08	2.967027e+05
REAL ESTATE	528902.09	1.625902e+06
RETIRED	25305116.38	2.356124e+07
SELF-EMPLOYED	672393.40	1.640253e+06

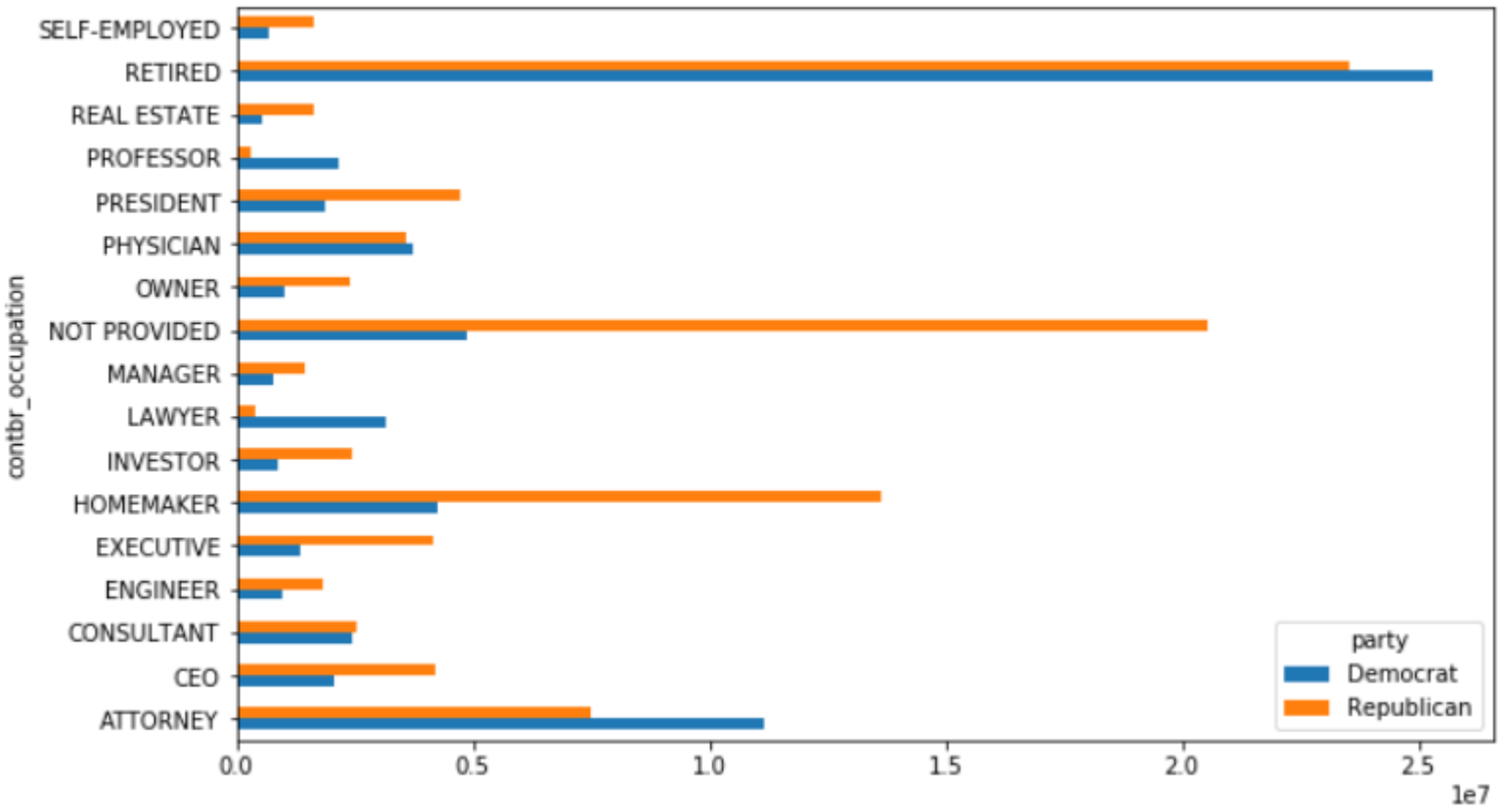
In [25]:
```
by_occupation = fec.pivot_table('contb_receipt_amt',
                                index='contbr_occupation',
                                columns='party', aggfunc='sum')
by_occupation.head()
```

Out[25]:

party	Democrat	Republican
contbr_occupation		
MIXED-MEDIA ARTIST / STORYTELLER	100.0	NaN
AREA VICE PRESIDENT	250.0	NaN
RESEARCH ASSOCIATE	100.0	NaN
TEACHER	500.0	NaN
THERAPIST	3900.0	NaN

정당 별 최다 기부자 직업

```
In [26]: over_2mm.plot(kind='barh')
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1d34241e408>
```



후보 별 고액 기부자 직군 7등

- 후보 이름으로 그룹 후 top 메소드 사용

```
In [32]: def get_top_amounts(group, key, n=5):
             totals = group.groupby(key)['contb_receipt_amt'].sum()
             return totals.nlargest(n)

In [33]: grouped = fec_mrbo.groupby('cand_nm')
         grouped.apply(get_top_amounts, 'contbr_occupation', n=7)

Out[33]: cand_nm        contbr_occupation
         Obama, Barack  RETIRED                                   25305116.38
                        ATTORNEY                                  11141982.97
                        INFORMATION REQUESTED                      4866973.96
                        HOMEMAKER                                  4248875.80
                        PHYSICIAN                                  3735124.94
                        LAWYER                                     3160478.87
                        CONSULTANT                                 2459912.71
         Romney, Mitt   RETIRED                                   11508473.59
                        INFORMATION REQUESTED PER BEST EFFORTS    11396894.84
                        HOMEMAKER                                  8147446.22
                        ATTORNEY                                   5364718.82
                        PRESIDENT                                  2491244.89
                        EXECUTIVE                                  2300947.03
                        C.E.O.                                     1968386.11
         Name: contb_receipt_amt, dtype: float64
```

후보 별 고액 고용주 10등

```
In [34]: grouped.apply(get_top_amounts, 'contbr_employer', n=10)
```

Out[34]:

cand_nm	contbr_employer	
Obama, Barack	RETIRED	22694358.85
	SELF-EMPLOYED	17080985.96
	NOT EMPLOYED	8586308.70
	INFORMATION REQUESTED	5053480.37
	HOMEMAKER	2605408.54
	SELF	1076531.20
	SELF EMPLOYED	469290.00
	STUDENT	318831.45
	VOLUNTEER	257104.00
	MICROSOFT	215585.36
Romney, Mitt	INFORMATION REQUESTED PER BEST EFFORTS	12059527.24
	RETIRED	11506225.71
	HOMEMAKER	8147196.22
	SELF-EMPLOYED	7409860.98
	STUDENT	496490.94
	CREDIT SUISSE	281150.00
	MORGAN STANLEY	267266.00
	GOLDMAN SACH & CO.	238250.00
	BARCLAYS CAPITAL	162750.00
	H.I.G. CAPITAL	139500.00

Name: contb_receipt_amt, dtype: float64

기부 규모별 분석

- **구간을 위한 배열**
  - bins = np.array([0, 1, 10, 100, 1000, 10000, 100000, 1000000, 10000000])

```
In [35]: bins = np.array([0, 1, 10, 100, 1000, 10000,
                          100000, 1000000, 10000000])
         labels = pd.cut(fec_mrbo.contb_receipt_amt, bins)
         labels
```

```
Out[35]: 411         (10, 100]
         412       (100, 1000]
         413       (100, 1000]
         414         (10, 100]
         415         (10, 100]
                      ...
         701381      (10, 100]
         701382    (100, 1000]
         701383       (1, 10]
         701384      (10, 100]
         701385    (100, 1000]
         Name: contb_receipt_amt, Length: 694282, dtype: cate
         Categories (8, interval[int64]): [(0, 1] < (1, 10] <
         < (100000, 1000000] < (1000000, 10000000]]
```

```
In [36]: grouped = fec_mrbo.groupby(['cand_nm', labels])
         grouped.size().unstack(0)
```

Out[36]:

cand_nm	Obama, Barack	Romney, Mitt
contb_receipt_amt		
(0, 1]	493	77
(1, 10]	40070	3681
(10, 100]	372280	31853
(100, 1000]	153991	43357
(1000, 10000]	22284	26186
(10000, 100000]	2	1
(100000, 1000000]	3	0
(1000000, 10000000]	4	0

```
In [127]: grouped = fec_mrbo.groupby(['cand_nm', labels])
          grouped.size()
```

```
Out[127]: cand_nm        contb_receipt_amt
          Obama, Barack  (0, 1]                493
                         (1, 10]              40070
                         (10, 100]           372280
                         (100, 1000]         153991
                         (1000, 10000]        22284
                         (10000, 100000]          2
                         (100000, 1000000]        3
                         (1000000, 10000000]      4
          Romney, Mitt   (0, 1]                 77
                         (1, 10]              3681
                         (10, 100]           31853
                         (100, 1000]         43357
                         (1000, 10000]       26186
                         (10000, 100000]          1
                         (100000, 1000000]        0
                         (1000000, 10000000]      0
          dtype: int64
```

후보와 버킷 별 전체 금액 대비 기부금액 비율

- **기부 금액을 모두 더한 후**
  - 버킷 별로 정규화

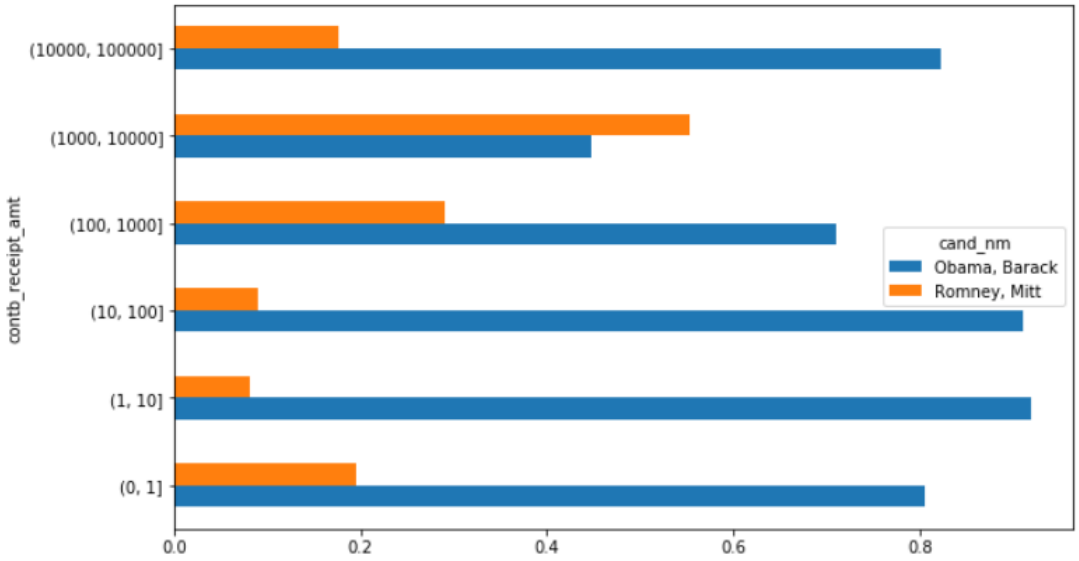
```
In [37]: bucket_sums = grouped.contb_receipt_amt.sum().unstack(0)
         normed_sums = bucket_sums.div(bucket_sums.sum(axis=1), axis=0)
         normed_sums.head()
```

Out[37]:

cand_nm	Obama, Barack	Romney, Mitt
contb_receipt_amt		
(0, 1]	0.805182	0.194818
(1, 10]	0.918767	0.081233
(10, 100]	0.910769	0.089231
(100, 1000]	0.710176	0.289824
(1000, 10000]	0.447326	0.552674

```
In [38]: normed_sums[:-2].plot(kind='barh')
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x1d3461e0f88>



다음 자료는 무엇일까요?

- **후보 별로 어느 구간의 기부금액이 많을까?**

In [140]:
```
pd.options.display.float_format = '{:.6f}'.format
# 후보별 버킷 구간의 기부 금액 비율
bucket_sums1 = grouped.contb_receipt_amt.sum().unstack(0)
normed_sums1 = bucket_sums.div(bucket_sums.sum(axis=0), axis=1)
normed_sums1.head()
```

Out[140]:

cand_nm	Obama, Barack	Romney, Mitt
contb_receipt_amt		
(0, 1]	0.000002	0.000001
(1, 10]	0.002482	0.000338
(10, 100]	0.149318	0.022503
(100, 1000]	0.403294	0.253163
(1000, 10000]	0.380885	0.723852

In [137]:
```
normed_sums1.sum()
```

Out[137]:
```
cand_nm
Obama, Barack    1.0
Romney, Mitt     1.0
dtype: float64
```

주별 기부 금액

- **다음 결과에서**
  - 후보자를 열로 올리려면
    - **unstack(0)**
      - **인덱스의 수준 0인 후보자를 열로 이동**

```
In [41]: grouped = fec_mrbo.groupby(['cand_nm', 'contbr_st'])
         sumgrp = grouped.contb_receipt_amt.sum()
         sumgrp.head()
```

```
Out[41]: cand_nm         contbr_st
         Obama, Barack   AA            56405.00
                         AB             2048.00
                         AE            42973.75
                         AK           281840.15
                         AL           543123.48
         Name: contb_receipt_amt, dtype: float64
```

```
In [39]: grouped = fec_mrbo.groupby(['cand_nm', 'contbr_st'])
         totals = grouped.contb_receipt_amt.sum().unstack(0).fillna(0)
         totals.head()
```

Out[39]:

cand_nm	Obama, Barack	Romney, Mitt
contbr_st		
AA	56405.00	135.00
AB	2048.00	0.00
AE	42973.75	5680.00
AK	281840.15	86204.24
AL	543123.48	527303.51

```
In [40]: totals = totals[totals.sum(1) > 100000]
         totals[:10]
```

Out[40]:

cand_nm	Obama, Barack	Romney, Mitt
contbr_st		
AK	281840.15	86204.24
AL	543123.48	527303.51
AR	359247.28	105556.00
AZ	1506476.98	1888436.23
CA	23824984.24	11237636.60
CO	2132429.49	1506714.12
CT	2068291.26	3499475.45
DC	4373538.80	1025137.50
DE	336669.14	82712.00
FL	7318178.58	8338458.81

- **주별 기부 총액이 100,000 불 이상인 지역**

각 후보에 대한 주별 전체 기부 금액의 상대적인 비율

- 각 행을 전체 기부금액으로 나눔

```
In [49]: percent = totals.div(totals.sum(1), axis=0)
         percent[:10]
```

Out[49]:

cand_nm contbr_st	Obama, Barack	Romney, Mitt
AK	0.765778	0.234222
AL	0.507390	0.492610
AR	0.772902	0.227098
AZ	0.443745	0.556255
CA	0.679498	0.320502
CO	0.585970	0.414030
CT	0.371476	0.628524
DC	0.810113	0.189887
DE	0.802776	0.197224
FL	0.467417	0.532583

```
In [148]: percent = totals.div(totals.sum(0), axis=1)
          percent.sort_values(by='Obama, Barack', ascending=False)
```

Out[148]:

cand_nm contbr_st	Obama, Barack	Romney, Mitt
CA	0.175344	0.127215
IL	0.121022	0.041077
NY	0.107833	0.115290
FL	0.053859	0.094395
MA	0.048935	0.053325
...	...	...
FM	0.000004	0.000000
QU	0.000004	0.000000
UK	0.000000	0.000028
FF	0.000000	0.001121
XX	0.000000	0.004531

67 rows × 2 columns