들어가며 - 데이터처리가 빠른 툴(Tool)

비행기의 년도별 정시운항 데이터를 분석한다고 해봅시다. 아래 그림은

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Year	Month	DayofMor	DayOfWe	DepTime	CRSDepTi	ArrTime	CRSArrTir	UniqueCa	FlightNum	TailNum	ActualElap	CRSElapse	AirTime
2	1987	10	14	3	741	730	912	849	PS	1451	NA	91	79	NA
3	1987	10	15	4	729	730	903	849	PS	1451	NA	94	79	NA
4	1987	10	17	6	741	730	918	849	PS	1451	NA	97	79	NA
5	1987	10	18	7	729	730	847	849	PS	1451	NA	78	79	NA
6	1987	10	19	1	749	730	922	849	PS	1451	NA	93	79	NA
7	1987	10	21	3	728	730	848	849	PS	1451	NA	80	79	NA
8	1987	10	22	4	728	730	852	849	PS	1451	NA	84	79	NA
9	1987	10	23	5	731	730	902	849	PS	1451	NA	91	79	NA
10	1987	10	24	6	744	730	908	849	PS	1451	NA	84	79	NA
11	1987	10	25	7	729	730	851	849	PS	1451	NA	82	79	NA
12	1987	10	26	1	735	730	904	849	PS	1451	NA	89	79	NA
13	1987	10	28	3	741	725	919	855	PS	1451	NA	98	90	NA
14	1987	10	29	4	742	725	906	855	PS	1451	NA	84	90	NA
15	1987	10	31	6	726	725	848	855	PS	1451	NA	82	90	NA
16	1987	10	1	4	936	915	1035	1001	PS	1451	NA	59	46	NA

http://stat-computing.org/dataexpo/2009/the-data.html 에 있는 항공 운항 관련 자료를 다운받은 것입니다. 비행기의 출발지, 도착지, 시점 등과 같은 29항목으로 구성되어 있습니다. 1987년도 하나만 다운받아도 압축을 풀고 엑셀로 열어보려고 하니 데이터가 너무 많아 저장을 할 수 없다는 메시지가 뜰 정도로 부피가 큰 자료입니다.

데이터 건수	데이터 용량
1,000건	약 100KB
1만 건	약 1MB
10만 건	약 10MB
100만 건	약 100MB
1,000만 건	약 1GB
5,000만 건	약 5GB

최근 년도까지 이 자료들을 취합하여 위의 표에서 나오는 건수별로 데이터를 누적해보니 용량이 최대 5GB까지 나왔다고 합니다. 이 자료에서 (1) 열(26항목) 중 연, 월, 출발지연시간, 도착지연시간, 출발공항, 총지연시간(출발+도착) 등 5개 열 선택 (2) 행 선택:

출발공항이 SAN인 행 선택 (3) 열계산: 출발지연시간과 도착지연을 합한 시간을 계산

	A	B	C	D	E
1	Year	Month	ArrDelay	DepDelay	Origin
2	1987	10	23	11	SAN
3	1987	10	14	-1	SAN
4	1987	10	29	11	SAN
5	1987	10	-2	-1	SAN
6	1987	10	33	19	SAN
7	1987	10	-1	-2	SAN
8	1987	10	3	-2	SAN
9	1987	10	13	1	SAN
10	1987	10	19	14	SAN
11	1987	10	2	-1	SAN
12	1987	10	15	5	SAN
13	1987	10	24	16	SAN
14	1987	10	11	17	SAN
15	1987	10	-7	1	SAN
16	1987	10	34	21	SFO

하여 총지연시간이라는 새로운 컬럼을 생성하고 저장 (4) 정렬: 총지연시간을 오름차순으로 정렬하는 4가지 전처리에 대해 툴별 처리속도를 비교해 본 자료를 아래 도표에 나타내었습니다.

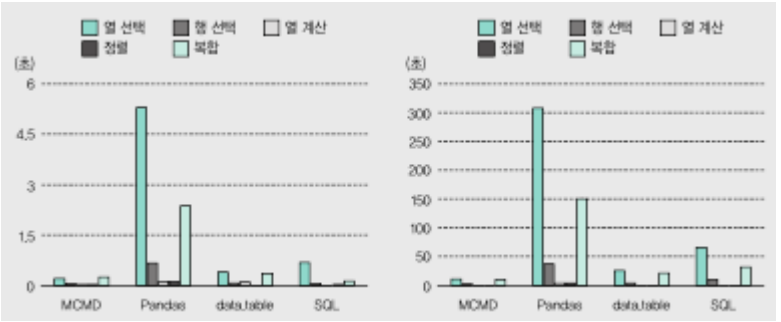
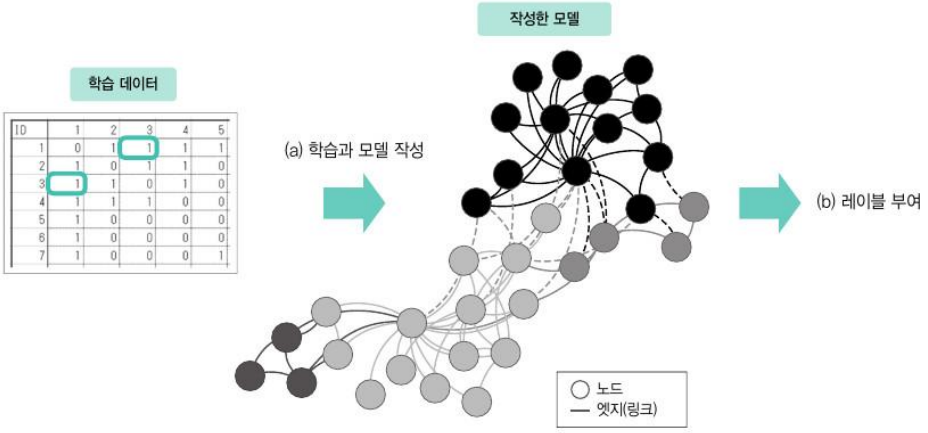


도표 왼쪽은 100만건, 오른쪽은 5,000만건에 대한 처리속도를 보여줍니다. 판다스가 다른 툴에 비해 처리속도가 떨어짐을 알 수 있습니다. 다만, 5,000만건인데, 300초를 상회하는 정도라면 그렇게 심각한 처리지연일지는 잘 모르겠습니다. 툴의 사용 편의성 등에 따라 복합적으로 툴의 교체를 고려하는 것이 더 좋을 것 같습니다.

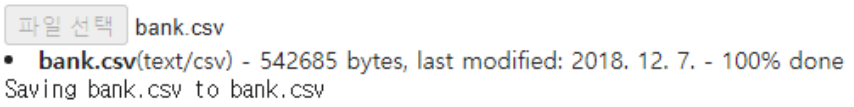


잠깐, 위 그림을 좀 봐주세요. 네트워크분석(클러스터링)에 따른 사람의 그룹화를 진행하는 모습입니다. 비지도 학습의 예를 나타낸 것으로 어떤 태권도 클럽에 소속된 부원 34명에 대한 서로간의 소통 관계를 표시한 그림입니다. 소통이 있으면 1, 없으면 0입니다. 노드는 부원이고 에지는 상대방과의 소통을 나타냅니다. 노드 그룹이 총 4종류이므로 4개의 그룹이 있음을 알 수 있습니다.

정형 데이터의 전처리(1)

https://roadbook.co.kr/242에서 자료를 다운받고 압축을 풀어야 합니다.

```
from google.colab import files
uploaded = files.upload()
```



구글코랩에서 PC의 자료를 받기 위해서는 위 그림과 같이 files를 import하고 upload 메소드를 실행하여 파일 열기 대화상자를 불러내야 합니다.

```
import pandas as pd
```

```
bank_df = pd.read_csv('bank.csv', sep=',')
bank_df.head()
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	NaN	5	may	261	1	-1	0	NaN	no
1	36	technician	single	secondary	no	265	yes	yes	NaN	5	may	348	1	-1	0	NaN	no
2	25	blue-collar	married	secondary	no	-7	yes	no	NaN	5	may	365	1	-1	0	NaN	no
3	53	technician	married	secondary	no	-3	no	no	NaN	5	may	1666	1	-1	0	NaN	no
4	24	technician	single	secondary	no	-103	yes	yes	NaN	5	may	145	1	-1	0	NaN	no

‘bank.csv’를 구글코랩에 업로드했으면 판다스를 사용하여 읽습니다. 그리고 처음에 나오는 자료 5줄을 출력해봅니다.

```
bank_df.shape
```

```
(7234, 17)
```

bank 자료는 7234행과 17열로 구성된 자료임을 알 수 있습니다. 또한 각 컬럼의 자료타입을 아래 그림에 나타내었습니다. int64는 정수, object는 문자열임을 알 수 있습니다.

```
bank_df.dtypes
```

```
age           int64
job          object
marital      object
education    object
default      object
balance       int64
housing      object
loan         object
contact      object
day           int64
month        object
duration      int64
campaign      int64
pdays         int64
previous      int64
poutcome     object
y            object
dtype: object
```

또한 자료에서 누락된 항목이 있는지 알아보기 위해 행별 조사를 해보았습니다. True이면 누락 항목이 있다는 뜻이겠죠?

```
bank_df.isnull().any(axis=1)
```

```
0        True
1        True
2        True
3        True
4        True
        ...
7229    False
7230    False
7231    False
7232     True
7233     True
Length: 7234, dtype: bool
```

```
bank_df.isnull().any(axis=0)
```

```
age          False
job           True
marital      False
education     True
default      False
balance      False
housing      False
loan         False
contact       True
day          False
month        False
duration     False
campaign     False
pdays        False
previous     False
poutcome      True
y            False
dtype: bool
```

아울러, 컬럼에서도 누락 항목여부를 조사하였습니다. 일단 결손값이 있는 것을 확인했으니 행과 항목별로 몇개의 결손값이 있는지를 조사해야 하겠죠?

```
bank_df.isnull().sum(axis=1)
```

```
0       2
1       2
2       2
3       2
4       2
       ..
7229    0
7230    0
7231    0
7232    1
7233    1
Length: 7234, dtype: int64
```

행에서는 일단 처음 5줄과 마지막 5줄에 2개 및 1개의 결손값이 눈에 띕니다.

```
bank_df.isnull().sum(axis=0)
```

```
age            0
job           44
marital        0
education    273
default        0
balance        0
housing        0
loan           0
contact     2038
day            0
month          0
duration       0
campaign       0
pdays          0
previous       0
poutcome    5900
y              0
dtype: int64
```

열쪽에서는 poutcome 항목이 상당히 많은 결손값을 보여줍니다.

```
np.sort(bank_df.isnull().sum(axis=1))
```

```
array([0, 0, 0, ..., 4, 4, 4])
```

결손값 오름차순으로 정렬해보니 행에서 제일 많은 결손 항목이 있는 곳은 4임을 알 수 있습니다. 이제 통계량을 알아봅시다.

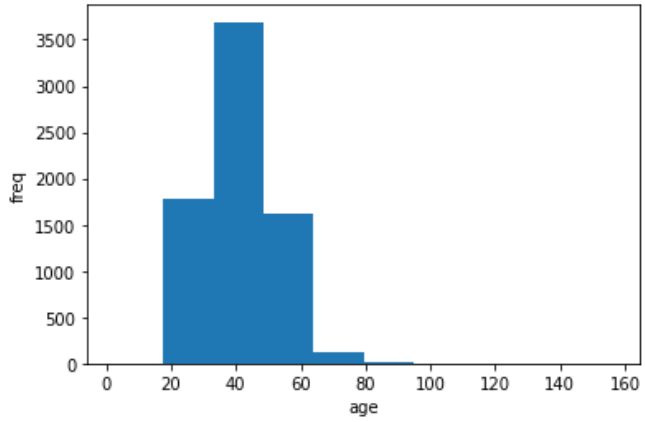
```
bank_df.describe()
```

	age	balance	day	duration	campaign	pdays	previous
count	7234.000000	7234.000000	7234.000000	7234.000000	7234.000000	7234.000000	7234.000000
mean	40.834808	1374.912911	15.623860	262.875311	2.713989	40.277716	0.565939
std	10.706442	3033.882933	8.307826	268.921065	2.983740	99.188008	1.825100
min	2.000000	-3313.000000	1.000000	0.000000	1.000000	-1.000000	0.000000
25%	33.000000	74.000000	8.000000	103.000000	1.000000	-1.000000	0.000000
50%	39.000000	453.500000	16.000000	183.000000	2.000000	-1.000000	0.000000
75%	48.000000	1470.750000	21.000000	321.750000	3.000000	-1.000000	0.000000
max	157.000000	81204.000000	31.000000	3366.000000	44.000000	850.000000	40.000000

이제 이 데이터를 도표로 나타내어 보겠습니다. Matplotlib를 사용하면 편리합니다.

```
import matplotlib.pyplot as plt
```

```
plt.hist(bank_df['age'])
plt.xlabel('age')
plt.ylabel('freq')
plt.show()
```



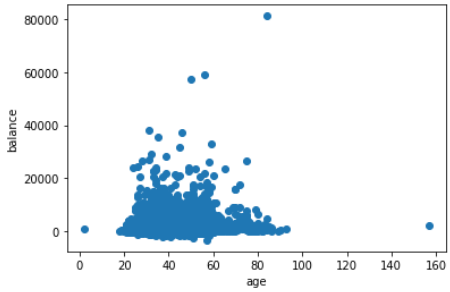
나이외에 다른 항목들도 히스토그램을 나타낼 수 있겠죠?

```
bank_df.describe(include='object')
```

	job	marital	education	default	housing	loan	contact	month	poutcome	y
count	7190	7234	6961	7234	7234	7234	5196	7234	1334	7234
unique	11	3	3	2	2	2	2	12	3	2
top	management	married	secondary	no	yes	no	cellular	may	failure	no
freq	1560	4343	3745	7101	4058	6066	4697	2202	772	6381

위 그림은 문자열 항목에 대한 통계를 나타낸 것입니다. count는 각 항목의 갯수입니다. unique는 각 항목의 서로다른 값의 개수죠. top은 제일 많이 나타난 값을, freq는 빈도를 나타냅니다.

```
plt.scatter(bank_df['age'],bank_df['balance'])
plt.xlabel('age')
plt.ylabel('balance')
plt.show()
bank_df[['age','balance']].corr()
```



	age	balance
age	1.000000	0.112364
balance	0.112364	1.000000

age와 balance의 산포도를 나타내고 상관관계를 계산해본 것입니다. 두 항목 사이의 관계는 0.11정도로 거의 관계가 없음이 판명되었습니다. 상관계수는 -1부터 1사이의 범위에서 움직이며 절대값이 클수록

상관관계가 강합니다. 다른 항목도 산포도를 구해보세요. 상관관계가 있는 항목이 있습니까?

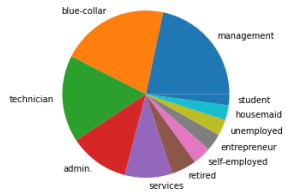
이제 데이터형이 문자열인 항목 중 job의 원 그래프를 작성해보겠습니다.

```
bank_df['job'].value_counts(ascending=False, normalize=True)
```

```
management     0.216968
blue-collar    0.208484
technician     0.167733
admin.         0.115994
services       0.091933
retired        0.048818
self-employed  0.035605
entrepreneur   0.033241
unemployed     0.031015
housemaid      0.028929
student        0.021280
Name: job, dtype: float64
```

직업별 발생회수를 내림차순으로 표시하였습니다. normalize 매개변수를 True로 설정하여 발생회수 전체가 1이 되도록 정규화하였습니다.

```
job_label = bank_df['job'].value_counts(ascending=False,
                                        normalize=True).index
job_vals = bank_df['job'].value_counts(ascending=False,
                                       normalize=True).values
plt.pie(job_vals, labels=job_label)
plt.axis('equal')
plt.show()
```



위 그림은 직업별 분포를 파이차트로 나타낸 것입니다.