0. 기본 라이브러리

In [2]:
```
# 기본 라이브러리 호출
import warnings
warnings.filterwarnings('ignore')
warnings.warn("once")
import pandas as pd
import numpy as np
import scipy as sp
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from scipy import stats
from scipy.stats import norm, skew
# 선형모델을 추정하는 라이브러리
import statsmodels.formula.api as smf
import statsmodels.api as sm
import statsmodels.stats.api as sms
from patsy import dmatrices
color = sns.color_palette()

pd.set_option('display.float_format','{:,.2f}'.format) # 소수점 2번째 자리까지 표현
pd.set_option('display.max_columns', None) # 모든 컬럼 표시
pd.set_option('display.max_colwidth', None) # 컬럼내용 전체 표시

#Graph에 한글을 표시하기 위한 코드
import matplotlib
from matplotlib import font_manager, rc
import platform
# font_name = font_manager.FontProperties(fname="c:/Windows/Fonts/malgun.ttf").get_name()
font_name = font_manager.FontProperties(fname="malgun.ttf").get_name()
rc('font', family=font_name)\

matplotlib.rcParams['axes.unicode_minus'] = False
```

1. 연관규칙 (사용 데이터 : lotto)

Q1) 연관규칙분석을 수행하기 위해 lotto 데이터셋을 transaction 데이터로 변환하시오.

- 단, 본 분석에서 로또번호가 추첨된 순서는 고려하지 않고 분석을 수행하도록 한다.
- 변환된 데이터에서 가장 많이 등장한 상위 10개의 로또번호를 막대그래프로 출력하고 이에 대해 설명하시오.

In [24]:
```
df = pd.read_csv('./data/lotto.csv')
print(df.shape)
df.head()
```

```
(859, 7)
```

Out[24]:

	time_id	num1	num2	num3	num4	num5	num6
0	859	8	22	35	38	39	41
1	858	9	13	32	38	39	43
2	857	6	10	16	28	34	38
3	856	10	24	40	41	43	44
4	855	8	15	17	19	43	44

1) melt를 사용하는 방법

In [4]:
```
melt = pd.melt(df, id_vars = ['time_id']) # time_id 기준, 컬럼을 variable로하여 값을 분해함
```

In [25]:
```
# value(실제 나온 숫자)를 기준으로 그루핑하여 variable을 count 내림차순으로 정렬 후 상위 10개만 뽑아냄
pivot = melt.groupby('value')[['variable']].count().sort_values(by = 'variable', ascending = False)[:10].reset_index()
```

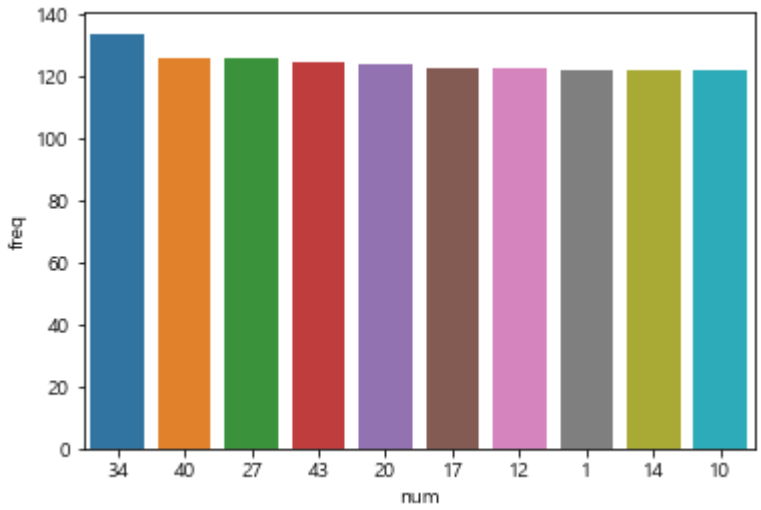
In [ ]:
```
sns.barplot(x = 'value', y = 'variable', data = pivot, order = pivot['value'])
my
```

2) mlxtend를 사용하는 방법

In [46]:
```
from mlxtend.preprocessing import TransactionEncoder
dataset = df[df.columns[1:]].values.tolist() # df의 num 컬럼들의 값을 list형태로 dataset에 저장
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset) # dataset을 np.array 형태로 변환 후 fitting
te_df = pd.DataFrame(te_ary, columns = te.columns_) # te_ary를 dataframe 형태로 변환

# top 10 시각화
best10 = pd.DataFrame(columns = ['num', 'freq']) # 상위 10개 숫자를 저장하기 위한 dataframe 생성
best10['num'] = te_df.sum(axis = 0).index # te_df의 세로 합을 구한 후 해당 번호를 num컬럼에 저장
best10['freq'] = te_df.sum(axis = 0).values # te_df의 세로 합을 구한 후 합계를 freq컬럼에 저장
best10 = best10.sort_values(by = 'freq', ascending = False).head(10).reset_index(drop = True) #freq를 기준으로 내림차순 정렬 후 상위 10개만 저장
sns.barplot(data = best10, x = 'num', y = 'freq', order = best10['num'])
```

Out[46]: <AxesSubplot:xlabel='num', ylabel='freq'>



Q2) 변환한 데이터에 대해 apriori함수를 사용하여 다음 괄호 안의 조건을 반영하여 연관규칙을 생성하고, 이를 'rules_1'이라는 변수에 저장하여 결과를 해석하시오.

- (최소 지지도 : 0.002, 최소 신뢰도 : 0.8, 최소조합 항목 수 : 2개, 최대조합 항목 수 : 6개)
- 그리고 도출된 연관규칙들을 향상도를 기준으로 내림차순 정렬하여 상위 30개의 규칙을 확인하고, 이를 데이터프레임으로 변환하여 csv파일로 출력하시오.

최소 지지도, 최대 조합 항목 설정

In [47]:
```
# 파이썬 Apriori에서는 최소 신뢰도 및 조합항목 수 조절 불가
from mlxtend.frequent_patterns import association_rules, apriori
frequent_itemsets = apriori(te_df,
                            min_support = 0.002, # 최소 지지도
                            max_len = 6,
                            use_colnames = True) # 컬럼명 사용
```

최소 조합 항목 설정

In [48]:
```
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: le
n(x)) # itemsets 컬럼내 값의 길이 계산
```

In [49]:
```
# 최소 조합항목수가 2 이상 & 최소 지지도가 0.002 이상
frequent_itemsets[(frequent_itemsets['length'] >= 2) & (frequent_itemsets['sup
port'] >= 0.002)]
# 지지도 : A,B가 동시에 일어난 횟수 / 전체 횟수
```

Out[49]:

	support	itemsets	length
45	0.02	(1, 2)	2
46	0.02	(1, 3)	2
47	0.01	(1, 4)	2
48	0.02	(1, 5)	2
49	0.01	(1, 6)	2
...	...	...	...
6358	0.00	(40, 43, 13, 14, 26)	5
6359	0.00	(14, 15, 18, 21, 26)	5
6360	0.00	(40, 14, 27, 30, 31)	5
6361	0.00	(34, 44, 15, 19, 21)	5
6362	0.00	(36, 43, 16, 26, 31)	5

6318 rows × 3 columns

규칙을 생성하며 최소 신뢰도 설정

In [50]:
```
rule_1 = association_rules(frequent_itemsets, metric="confidence", # 조건절 {co
nfidence : 신뢰도, lift : 향상도}
                           min_threshold=0.8 # 최소값
                          )
# 신뢰도(confidence) : A,B가 동시에 일어난 횟수 / A가 일어난 횟수
# 향상도(lift) : A가 주어지지 않았을 때의 B의 확률 대비 A가 주어졌을 때의 B의 확률
# 향상도가 1보다 크거나(+관계) 작다면(-관계) 우연적 기회보다 우수함
# 서로 독립이면 향상도는 1
```

결과 해석

In [51]:
```
rule_1.shape

### 704개의 규칙이 생성되었음
```

Out[51]:
```
(704, 9)
```

In [52]:
```
# 규칙의 조합항목 개수 컬럼 생성
rule_1['antecedents_length'] = rule_1['antecedents'].apply(lambda x: len(x))
rule_1['consequents_length'] = rule_1['consequents'].apply(lambda x: len(x))
rule_1['total_length'] = rule_1['antecedents_length'] + rule_1['consequents_le
ngth']
rule_1 = rule_1.sort_values(by = 'lift', ascending = False)
```

In [53]:
```
rule_1['antecedents_length'].value_counts()

### 좌측항이 3개 항목으로 조합된 규칙은 657건, 4개 항목으로 조합된 규칙은 47건임.
```

Out[53]:
```
3    657
4     47
Name: antecedents_length, dtype: int64
```

In [54]:
```
rule_1['consequents_length'].value_counts()
### 우측항이 1개 항목으로 조합된 규칙은 679건, 2개 항목으로 조합된 규칙은 25건임.
```

Out[54]:
```
1    679
2     25
Name: consequents_length, dtype: int64
```

In [55]:
```
rule_1['total_length'].value_counts()
### 전체 규칙 중 632개의 규칙은 4개 항목의 조합으로 이루어져 있으며, 72개의 규칙은 5개의
 항목 조합으로 이루어져 있음
```

Out[55]:
```
4    632
5     72
Name: total_length, dtype: int64
```

In [56]:
```
a = rule_1.describe(include = 'all')
a
```

Out[56]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverag
count	704	704	704.00	704.00	704.00	704.00	704.00	704.0
unique	654	70	nan	nan	nan	nan	nan	na
top	(26, 21, 14)	(43)	nan	nan	nan	nan	nan	na
freq	3	28	nan	nan	nan	nan	nan	na
mean	NaN	NaN	0.00	0.13	0.00	1.00	9.30	0.0
std	NaN	NaN	0.00	0.02	0.00	0.00	9.99	0.0
min	NaN	NaN	0.00	0.01	0.00	1.00	6.41	0.0
25%	NaN	NaN	0.00	0.13	0.00	1.00	7.04	0.0
50%	NaN	NaN	0.00	0.14	0.00	1.00	7.34	0.0
75%	NaN	NaN	0.00	0.14	0.00	1.00	7.74	0.0
max	NaN	NaN	0.00	0.16	0.00	1.00	78.09	0.0

In [58]:
```
print(a.loc['min','lift'])  ### 규칙들에 대한 향상도의 최소값은 6.41로 꽤 높게 나타났음
print(a.loc['mean','support']) ### 항목들의 교집합 확률을 의미하는 지지도의 평균은 0.002363으로 나타났다.
```

```
6.41044776119403
0.002363014604720083
```

특정 항목에 대한 연관성규칙 결과 확인

In [59]:
```
rule_1[rule_1['consequents'] == frozenset([34])]
```

Out[59]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	c
542	(17, 42, 45)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
693	(19, 44, 21, 15)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
50	(2, 4, 31)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
471	(25, 44, 14)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
486	(41, 19, 15)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
579	(24, 22, 31)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
8	(1, 5, 13)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
68	(2, 21, 15)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
72	(2, 28, 15)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
539	(32, 17, 33)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
561	(19, 44, 21)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
419	(12, 37, 36)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
641	(24, 31, 22, 7)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
203	(17, 5, 29)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
651	(10, 36, 44, 22)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
282	(24, 31, 7)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
192	(5, 29, 13)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
603	(42, 45, 23)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	
281	(7, 22, 31)	(34)	0.00	0.16	0.00	1.00	6.41	0.00	

In [60]:
```
rule_1[rule_1['consequents'] == frozenset([34])]['support'] * 100
```

Out[60]:
```
542   0.23
693   0.23
50    0.35
471   0.23
486   0.23
579   0.23
8     0.23
68    0.23
72    0.23
539   0.35
561   0.23
419   0.23
641   0.23
203   0.23
651   0.23
282   0.23
192   0.23
603   0.23
281   0.23
Name: support, dtype: float64
```

- 총 19개의 규칙이 도출되었으며 (1,5,13)번과 (34)번이 함께 추첨될 확률은 지지도(support) 확인 결과 0.23% 이다. 이 규칙의 향상도(lift)는 6.41로 (34)만 추첨되었을 때보다 (1,5,13)번이 뽑히고 (34)번이 뽑힐 확률이 약 6.41배 높다는 걸 의미한다.

연관분석 추가 코드

In [13]:
```
# Association 을 위한 전처리
import pandas as pd
from mlxtend.frequent_patterns import apriori

# 데이터 프레임 물건 리스트 변환
df = pd.DataFrame([[1, 'banana'],[2, 'banana'],[2, 'apple'],[3, 'banana']], co
lumns=['a','b'])
def toList(x):
  return list(set(x))
df1 = df.groupby('a').b.apply(lambda x: toList(x)).reset_index()
# 2개 이상만 추출
df1['leng'] = df1.b.apply(lambda x: len(x) >= 2)

# 원하는 연관분석 형태로 변환
dataset = list(df1.b)

# 조금더 확실하게 볼수 있는 더 많은 데이터셋으로 전환
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

# 원하는 변수들을 인덱스/컬럼 으로 재정렬
# 각 제품의 포함 여부를 one-hot encoding하여 array 로 변환
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)

# 변환된 array를 dataframe으로 변환 후 확인
df = pd.DataFrame(te_ary, columns=te.columns_)

# 컬럼 이름
print(te.columns_)

# 혹시 1 또는 0으로 변경하고 싶다면
pd.DataFrame(te_ary.astype('int'), columns=te.columns_)

# 원래 이중 리스트로 변환
te.inverse_transform(te_ary)

# 연관규칙 분석을 위한 apriori 알고리즘 사용
from mlxtend.frequent_patterns import apriori

# 지지도 도출 -> 수가 많을 수 있으므로 min_support 로 일정 이상의 지지도만 도출 (default
=0.5)
frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)

# 특정 개수 이상의 itemset만 추출
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: le
n(x))
frequent_itemsets[frequent_itemsets['length'] >=2]

# 특정 아이템(Eggs) 이 포함된 것만 추출
frequent_itemsets[frequent_itemsets['itemsets'].apply(lambda x: 'Eggs' in list
(x))]
```

```
# 연관 규칙 도출
from mlxtend.frequent_patterns import association_rules

# 최소 신뢰도 0.7이상인것만 추출
association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# 최소 향상도 0.7이상인것만 추출
association_rules(frequent_itemsets, metric="lift", min_threshold=0.7)

# antecedents 1개인거만 추출
rules[rules.apply(lambda x: True if len(x.antecedents) else false, axis=1)]

# lift 제일 큰것찾기 = 상호 정보량이 가장 큰것 찾기(lift max)
df[df.antecedents == frozenset({'Eggs'})].sort_values(by='lift', ascending=Fal
se)

# antecedents가 Eggs 이고, consequents 가 하나일때 lift 젤 작은것 찾기 = 가장 멀리있
어도 되는 물품
rules[(rules.antecedents == frozenset({'Eggs'})) & (rules.consequents.apply(la
mbda x: len(x) ==1))].sort_values(by='lift')

# antecedents에 특정 단어 'Eggs' 있는거 찾기
rules[rules.antecedents.apply(lambda x: 'Eggs' in x)]

# 특정 antecedents 만 찾기
rules[rules.antecedents == frozenset({'Eggs'})]

# fronzenset 에서 값 추출하기
[i for i in frozenset({'Apple', 'Banana'})]
```