# 04. CBR & AR

## 1. Case Based Reasoning (K-Nearest Neighbor Classfication)

### 1.1 KNN

KNN은 머신러닝에 자주 사용되는 알고리즘 중 하나로, 새로운 데이터가 주어졌을 때, 기존 데이터 중 가장 가까운 k개의 데이터를 정보로 새로운 데이터를 예측하는 방법론이다. 예측 방법은 입력 변수간의 거리를 구하여 가장 가까운 k개의 데이터를 최근접 이웃으로 정하고, 이를 기반으로 목표 변수의 범주를 정하는 알고리즘이다.

이는 군집분석과 매우 다른 개념인데, 군집분석은 비지도 학습으로, 입력변수 등이 유사한 것끼리 묶고 사전에 정의된 카테고리가 없다 보니 정확한 점수를 평가하는 것이 불가능하다. 하지만, 분류의 경우 지도학습이고, 사전에 정의된 카테고리를 주어진 입력변수로 모형을 생성하기 때문에 평가가 가능하다.

K 개수만큼 주변에 있는 sample들의 정보를 이용하여 새로운 관측치의 종속변수를 예측하는 방법이기에, 종속변수가 범주형 변수 / 연속형 변수에 따라 데이터의 거리를 구하는 방법과 종속변수를 추정하기 위한 기준 값이 달라진다. 범주형 변수의 경우에는 k-nearest neighbors 중 가장 많이 나타나는 범주로 추정하고, 연속형 변수의 경우에는 k-nearest neighbors의 대표값(평균)으로 종속변수를 추정한다. 본 과제에서는 범주형과 연속형을 구분하여 진행하였고, 범주형 데이터는 Pima-Indians Diabetes data를 사용하고 연속형 데이터는 보스턴 집값 데이터를 사용하고자 한다.

### 1.2 연속형 변수

#### 1) 데이터 설명

Python sklearn dataset에 있는 보스턴의 집값 데이터를 활용하였다. 해당 데이터는 1978년 보스턴 교외의 506개의 주택에 대해 14개 범주의 수치를 정리한 것이다. 14개의 범주는 다음과 같다.

| | |
|---|---|
| **CRIM** | 도시의 1인당 범죄율 |
| **ZN** | 25,000 평방 피트가 넘는 주택 비율 |

| | |
|---|---|
| **INDUS** | 타운당 비소매 사업 면적 비율 |
| **CHAS** | 길이 찰스강과 경계하면 1, 아니면 0 |
| **NOX** | 일산화질소 농도 (0.1ppm 단위) |
| **RM** | 주택 당 평균 방 수 |
| **AGE** | 1940년 이전에 건축된 주인이 거주하는 주택 비율 |
| **DIS** | 보스턴의 5개 고용센터까지 가중치 거리 |
| **RAD** | 방사형 고속도로로의 접근성 지수 |
| **TAX** | $10,000 당 최대 재산세율 |
| **PTRATIO** | 타운별 학생-교사 비율 |
| **B** | 아프리카계 미국인의 비율 |
| **LSTAT** | 저소득층 계층의 비율 |
| **MEDV (target)** | **주인이 거주하는 주택 가격의 중간값 ($1,000 단위)** |

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRATIO | B | LSTAT | MEDV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4.98 | 24.0 |
| 1 | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9.14 | 21.6 |
| 2 | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4.03 | 34.7 |
| 3 | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 | 2.94 | 33.4 |
| 4 | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 | 5.33 | 36.2 |

해당 데이터는 총 506개의 데이터로 구성되어있다. 총 14개의 변수 중 설명 변수는 13개이고, 설명 변수 중 범주형 변수가 1개 존재하고, 나머지는 연속형 변수이다. 본격적인 KNN 알고리즘 적용을 하기 전에 70%와 30%로 Train set과 Test set을 구분하였다. 또한 설명변수의 단위가 비율, 거리 등으로 제각각이므로 정규화를 통해 범위를 맞춘 이후 알고리즘 적용을 진행하였다.

```
# Train: Test = 7:3 분리
from sklearn.model_selection import train_test_split
x = df[['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT']]
y = df[["MEDV"]]
train_x, valid_x, train_y, valid_y = train_test_split(x, y, test_size = 0.3, random_state = 2021)
```

```
# Z-score standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

train_x_std = scaler.fit_transform(train_x)
valid_x_std = scaler.transform(valid_x)
```

이웃을 결정할 때, 데이터 간 거리를 기준으로 가중치를 두어 구하게 된다. 종속변수가 연속형인 데이터에 KNN을 적용할 때 사용되는 거리 측정 방식은 대표적으로 Euclidean 측정방식과 Manhattan 측정방식이 있다. Euclidean 거리 측정방식은 피타고라스 정리에 의해 두 지점 사이의 최단 거리를 구하는 공식으로 계산되고, 맨하탄 방식은 맨하탄의 블록 지형과 같이 블록으로 거리를 계산하여 사용된다.

$$\text{Manhattan distance} = \sum_{j=1}^{J} |x_j - y_j|^2$$

$$\text{Euclidian distance} = \sqrt{\sum_{j=1}^{J} (x_j - y_j)^2}$$

거리 별로 k를 바꿔가며 적용을 해보았고, Mean squared error 값과 Regressor score 두가지 수치를 이용하여 비교를 진행하였다.

**2) KNN 알고리즘 적용 – Euclidean Distance**

먼저 k=3 일 때를 계산해보았다.

```
from sklearn.neighbors import KNeighborsRegressor
regressor = KNeighborsRegressor(n_neighbors = 3, weights = "distance",metric = 'euclidean')

regressor.fit(train_x_std, train_y)
```

```
KNeighborsRegressor(metric='euclidean', n_neighbors=3, weights='distance')
```

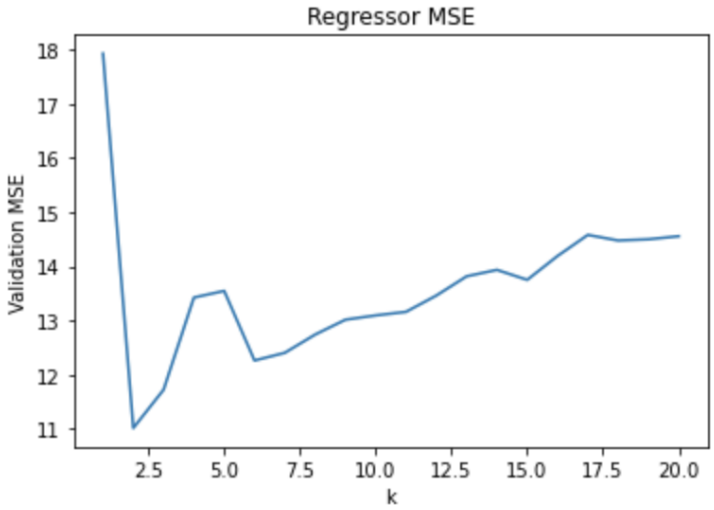
```
import pandas as pd
from sklearn import svm, metrics
from math import sqrt
pred_y = regressor.predict(valid_x_std)
mse = metrics.mean_squared_error(valid_y, pred_y)
rmse = sqrt(metrics.mean_squared_error(valid_y, pred_y))

print(mse)
print(rmse)
```

```
11.723678123941927
3.423985707321502
```

```
print('{:.3f}'.format(regressor.score(valid_x_std, valid_y)))
```

```
0.820
```

K=3일때 MSE는 약 11.72가 나왔고, RMSE는 3.42가 나온 것으로 확인할 수 있었다. 또한 scroe 값이 82%가 나왔다. 하지만 k=3인 것인 최적인지 확인하기 위하여 MSE에 따른 그래프를 그려 가장 작은 MSE를 갖는 k값을 찾아내었다.




K를 1부터 20으로 설정을 한 이후 얻은 결과 그래프이다. 그래프에서 확인할 수 있듯이, 앞서 진행한 k=3도 작은 MSE를 갖지만, k=2일 때가 가장 작은 MSE임을 확인할 수 있었다.

```
from sklearn.neighbors import KNeighborsRegressor
regressor = KNeighborsRegressor(n_neighbors = 2, weights = "distance",metric = 'euclidean')

regressor.fit(train_x_std, train_y)
```

```
KNeighborsRegressor(metric='euclidean', n_neighbors=2, weights='distance')
```

```
pred_y = regressor.predict(valid_x_std)
mse = metrics.mean_squared_error(valid_y, pred_y)
rmse = sqrt(metrics.mean_squared_error(valid_y, pred_y))

print(mse)
print(rmse)
print('{:.3f}'.format(regressor.score(valid_x_std, valid_y)))
```

```
11.015216177421626
3.318917922670223
0.831
```

이에 따른 k=2일 때의 적용 결과이다. K=3일 때보다 낮은 MSE인 11.02를 확인할 수 있었고, 그에 따라 RMSE도 3.32가 나왔다. 이때 Regressor score값음 83.1%로 확인할 수 있었다.

| Euclidean | K = 3 | K = 2 |
|---|---|---|
| **MSE** | 11.72 | 11.02 |
| **RMSE** | 3.42 | 3.32 |
| **Score** | 82% | 83.1% |

따라서, 거리값 가중치를 Euclidean 거리 방식을 이용했을 때에는, k=2일 때가 가장 적합하다는 결론을 도출 할 수 있었다.

### 3) KNN 알고리즘 적용 – Manhattan Distance

먼저 k=3 일 때를 계산해보았다.

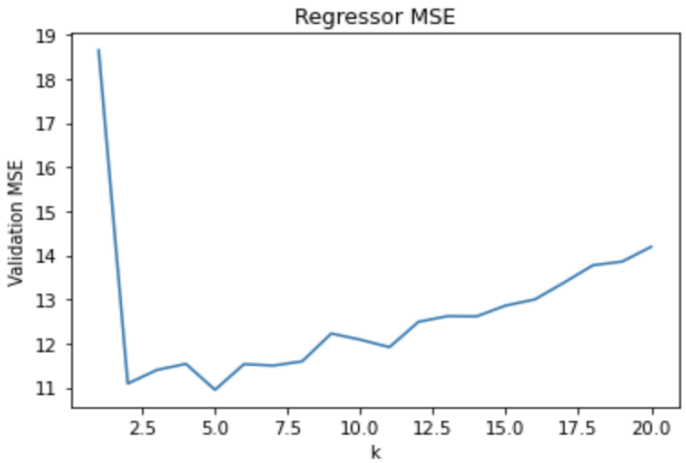
```
from sklearn.neighbors import KNeighborsRegressor
regressor = KNeighborsRegressor(n_neighbors = 3, weights = "distance",metric = 'manhattan')

regressor.fit(train_x_std, train_y)
```

```
KNeighborsRegressor(metric='manhattan', n_neighbors=3, weights='distance')
```

```
pred_y = regressor.predict(valid_x_std)
mse = metrics.mean_squared_error(valid_y, pred_y)
rmse = sqrt(metrics.mean_squared_error(valid_y, pred_y))

print(mse)
print(rmse)
print('{:.3f}'.format(regressor.score(valid_x_std, valid_y)))
```

```
11.396710060286235
3.3759013700471514
0.825
```

K=3일때 MSE는 약 11.40이 나왔고, RMSE는 3.38이 나온 것으로 확인할 수 있었다. 이에 따른 스코어 값은 82.5%가 나왔다. 앞선 유클리디언 거리와 비교하였을 때 같은 K=3일 때 스코어인 82%보다 높은 82.5%가 나온 것도 확인 가능했다. 하지만 3이 최적의 k이인지는 앞선 방식과 동일하게 MSE 그래프를 통해 확인하고자 하였다.



그래프를 시각적으로 확인했을 때는 k=2와 k=5일 때 작은 MSE가 나온다는 것을 확인할 수 있어 k=2일때, k=5일 때 둘다 MSE값과 score를 계산하였다.

```
from sklearn.neighbors import KNeighborsRegressor
regressor = KNeighborsRegressor(n_neighbors = 2, weights = "distance",metric = 'manhattan')

regressor.fit(train_x_std, train_y)
```

```
KNeighborsRegressor(metric='manhattan', n_neighbors=2, weights='distance')
```

```
pred_y = regressor.predict(valid_x_std)
mse = metrics.mean_squared_error(valid_y, pred_y)
rmse = sqrt(metrics.mean_squared_error(valid_y, pred_y))

print(mse)
print(rmse)
```

```
11.089004504316614
3.3300156913018615
```

```
print('{:.3f}'.format(regressor.score(valid_x_std, valid_y)))
```

```
0.829
```

먼저, k=2일 때의 결과이다. MSE는 11.09 RMSE는 3.33, score 값은 82.9%가 나온 것을 확인할 수 있었다. K=3일 때보다 더 좋은 결과가 나온 것을 확인하였다.

```
from sklearn.neighbors import KNeighborsRegressor
regressor = KNeighborsRegressor(n_neighbors = 5, weights = "distance",metric = 'manhattan')

regressor.fit(train_x_std, train_y)
```

```
KNeighborsRegressor(metric='manhattan', weights='distance')
```

```
pred_y = regressor.predict(valid_x_std)
mse = metrics.mean_squared_error(valid_y, pred_y)
rmse = sqrt(metrics.mean_squared_error(valid_y, pred_y))

print(mse)
print(rmse)
print('{:.3f}'.format(regressor.score(valid_x_std, valid_y)))
```

```
10.946764824964594
3.308589552205682
0.832
```

위는 k=5일때의 결과이다. MSE는 10.95, RMSE는 3.31 그리고 score 값은 83.2%가 나왔다. K=5일 때는 2와 3일 때보다 더 적합한 결과가 나온 것을 확인할 수 있었다.

| Manhattan | K = 3 | K = 2 | K = 5 |
|---|---|---|---|
| **MSE** | 11.40 | 11.09 | 10.95 |
| **RMSE** | 3.38 | 3.33 | 3.31 |
| **Score** | 82.5% | 82.9% | 83.2% |

비교 분석 결과 Manhattan 거리 방식을 적용할 때에는 k=5일 때가 적합하다는 결론을 내릴 수 있었다.

#### 4) 연속형 변수 KNN 결론

| | K = 2 (Euclidean) | K = 5 (Manhattan) |
|---|---|---|
| **MSE** | 11.02 | 10.95 |
| **RMSE** | 3.32 | 3.31 |
| **Score** | 83.1% | 83.2% |

보스턴 집값 데이터를 통해 연속형 종속변수의 KNN 알고리즘을 적용한 결과이다. 거리에 따른 가중치를 주었고, Euclidean과 Manhattan 거리 측정 방식을 이용하여 적용하였다. 유클리디언에서는 k=2임의 결론을 내렸고, 맨하탄에서는 K=5일 때 결론을 내렸다. 최종 비교 분석을 진행하였을 때는, 본 데이터에서는 k=5 일 때 Manhattan 거리 측정방식을 이용하여, 거리값을 가중치로 두었을 때 가장 적합한 결과가 나온 다는 결론을 내릴 수 있었다.

### 1.3 범주형 변수

#### 1) Categorical Data : Pima Indians Diabetes Database

Categorical Data로 Kaggle의 'Pima Indians Diabetes Data'를 사용했다. 이 데이터는 미국립 당뇨병, 소화기병 및 신장병 연구소(the National Institute of Diabetes and Digestive and Kidney Diseases)에서 제공이 되었다. 해당 데이터의 9개의 Feature는 다음과 같다.

| Feature | Explanation |
|---|---|
| 1. Pregnancies | 임신 횟수 |
| 2. Glucose | 2시간 동안 평균 혈장 포도당 농도 |
| 3. BloodPressure | 혈압(mm/Hg) |
| 4. SkinThickness | 피하 두께 |
| 5. Insulin | 2시간 동안 평균 인슐린 수치 |
| 6. BMI | 체질량 지수 |
| 7. DiabetesPedigreeFunction | 당뇨 혈통 검사(함수) |
| 8. Age | 나이 |
| 9. Outcome | 0 : 정상 / 1 : Diabetes |

이 데이터는 또한, 21이상 여성에 대해 조사가 되었다. 종속변수인 'Outcome'을 제외한 독립변수 8개는 모두 연속형 데이터였다. 총 768개의 조사 결과로, 다음과 같이 간단하게 확인할 수 있었다.

```
df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies                 768 non-null int64
Glucose                     768 non-null int64
BloodPressure               768 non-null int64
SkinThickness               768 non-null int64
Insulin                     768 non-null int64
BMI                         768 non-null float64
DiabetesPedigreeFunction    768 non-null float64
Age                         768 non-null int64
Outcome                     768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

**2) 데이터 전처리**

데이터 값들의 분포와 이상값, 그리고 Null 값을 확인하는 과정을 거쳤다.

```
df.describe()
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| count | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| mean | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | 0.471876 | 33.240885 | 0.348958 |
| std | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 | 11.760232 | 0.476951 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 | 21.000000 | 0.000000 |
| 25% | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 | 24.000000 | 0.000000 |
| 50% | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| 75% | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| max | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

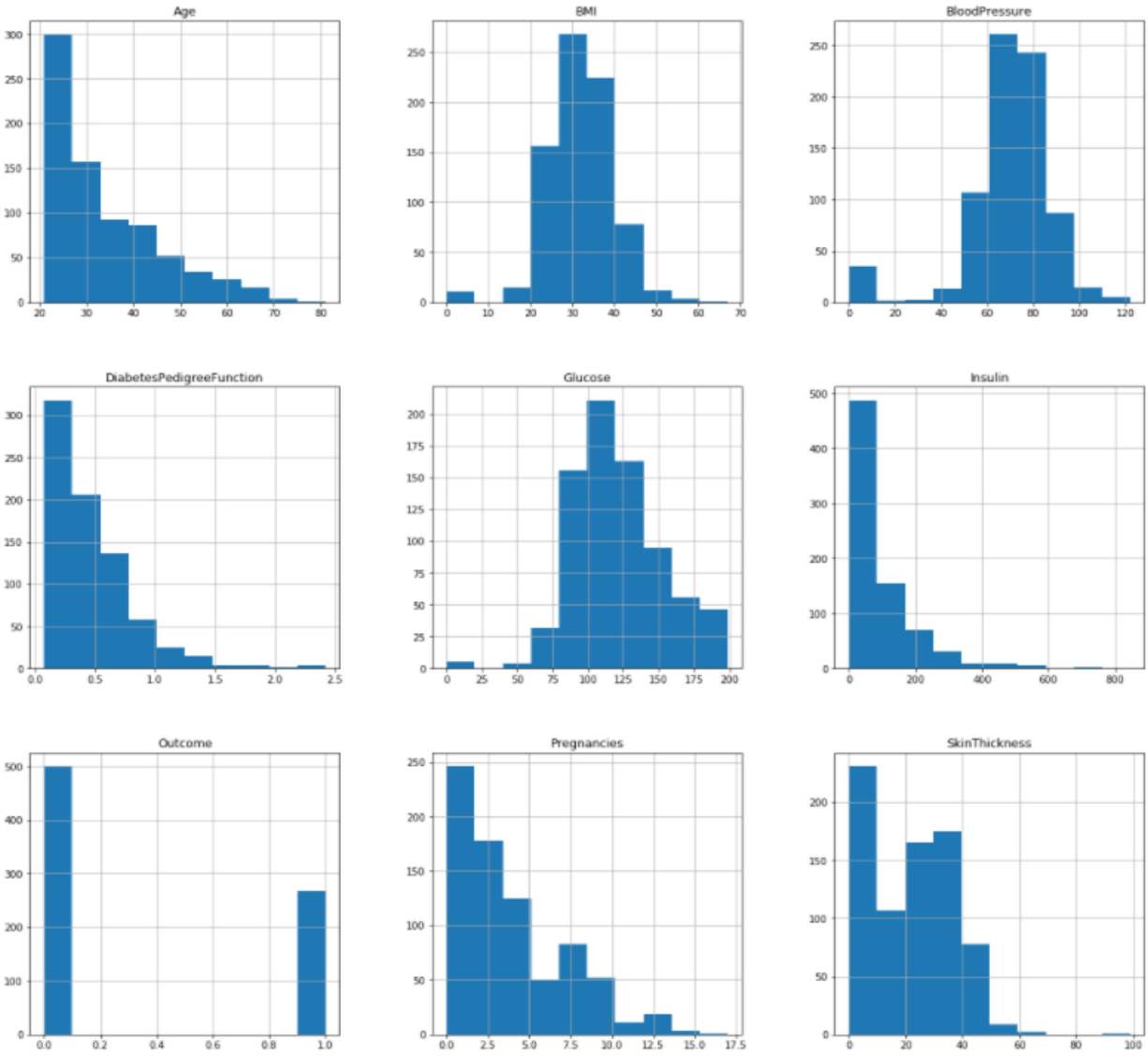
일반적으로 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI' 데이터에 대해서 Min 값이 0이 나오는 것은 이상하다고 볼 수 있다. 그 이유는 사람이 위 항목에 대해 0인 수치를 기록하면, 그것 자체가 오류이거나 missing vlaue라고 볼 수 있다. 따라서 위 5가지 항목에 대해 적당한 값으로 바꿔주는 과정이 필요하다.

```
diabetes_data_copy = df.copy(deep = True)
diabetes_data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diabetes_data_copy[['Glucose','BloodPressure','SkinThickr

## showing the count of Nans
print(diabetes_data_copy.isnull().sum())
```

```
Pregnancies                 0
Glucose                     5
BloodPressure              35
SkinThickness             227
Insulin                   374
BMI                        11
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

<0 값을 Null 값으로 대체해준 모습>

아래 Histogram은 데이터의 전반적인 형태를 보기 위해 나타났다. 5개의 항목 중 'BMI', 'BloodPressure' 항목은 평균에 가깝게 분포하고 있어 평균으로 Null(0) 값을 대체하고, 'SkinThickness', 'Insulin', 'BMI' 세 항목은 치우쳐져 있는 것을 아래 히스토그램 에서 확인할 수 있다. 그렇기 때문에 평균보다는 중앙값으로 Null(0) 값을 대체하여 진행을 하였다. (다음 페이지 히스토그램)




```
diabetes_data_copy['Glucose'].fillna(diabetes_data_copy['Glucose'].mean(), inplace = True)
diabetes_data_copy['BloodPressure'].fillna(diabetes_data_copy['BloodPressure'].mean(), inplace = True)
diabetes_data_copy['SkinThickness'].fillna(diabetes_data_copy['SkinThickness'].median(), inplace = True)
diabetes_data_copy['Insulin'].fillna(diabetes_data_copy['Insulin'].median(), inplace = True)
diabetes_data_copy['BMI'].fillna(diabetes_data_copy['BMI'].median(), inplace = True)
```

위 코드를 통해 Null(0) 값을 대체 해준 뒤 진행을 하였다.

y값을 확인한 결과, 다음과 같이 0에 편향되어 있었기 때문에 Scaling후 Shuffle 한 후, imbalanced 한 데이터를 맞춰주기 위한 다음과 같은 과정을 거쳤다.

```
y.value_counts()
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X =  pd.DataFrame(sc_X.fit_transform(diabetes_data_copy.drop(["Outcome"],axis = 1),),
        columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'])
```

```
X.head()
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.639947 | 0.865108 | -0.033518 | 0.670643 | -0.181541 | 0.166619 | 0.468492 | 1.425995 |
| 1 | -0.844885 | -1.206162 | -0.529859 | -0.012301 | -0.181541 | -0.852200 | -0.365061 | -0.190672 |
| 2 | 1.233880 | 2.015813 | -0.695306 | -0.012301 | -0.181541 | -1.332500 | 0.604397 | -0.105584 |
| 3 | -0.844885 | -1.074652 | -0.529859 | -0.695245 | -0.540642 | -0.633881 | -0.920763 | -1.041549 |
| 4 | -1.141852 | 0.503458 | -2.680669 | 0.670643 | 0.316566 | 1.549303 | 5.484909 | -0.020496 |

```
import sklearn
x_shuffled = sklearn.utils.shuffle(X, random_state=2021)
y_shuffled =sklearn.utils.shuffle(y, random_state=2021)

import imblearn
from imblearn.under_sampling import RandomUnderSampler
X_resampled, y_resampled = RandomUnderSampler(random_state=2021).fit_resample(x_shuffled, y_shuffled)
```

```
X_resampled
```

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age |
|---|---|---|---|---|---|---|---|---|
| 0 | -0.547919 | -7.787567e-01 | -1.026200 | -1.378189 | -0.239461 | 0.326720 | -0.827139 | -0.956462 |
| 1 | 0.342981 | -5.486156e-01 | -0.033518 | -0.012301 | 2.135236 | 0.646920 | -0.944924 | -0.445935 |
| 2 | -0.844885 | -9.431432e-01 | -1.357094 | -2.061133 | -0.181541 | -1.448937 | -0.165733 | -0.956462 |
| 3 | -0.250952 | 9.308627e-01 | 0.297376 | -0.012301 | -0.181541 | -1.667255 | -0.799958 | 0.319855 |
| 4 | -0.547919 | -4.499837e-01 | -1.687988 | -0.353773 | -0.899742 | 0.006519 | -0.464725 | -0.956462 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 531 | 0.639947 | -4.672145e-16 | -0.364412 | 1.353586 | -0.181541 | 0.952566 | 0.770504 | 0.660206 |
| 532 | 1.530847 | 1.128126e+00 | 1.124610 | -0.126125 | 0.165975 | 0.268501 | 2.165799 | 0.745293 |
| 533 | 0.342981 | -4.672145e-16 | 0.628269 | 0.329171 | -0.181541 | 1.243657 | -0.380161 | 0.319855 |
| 534 | -0.844885 | -1.540881e-01 | 1.290057 | -0.581421 | 0.050137 | 0.297610 | -0.208015 | 0.575118 |
| 535 | 0.342981 | 7.605298e-02 | 0.131929 | -0.012301 | -0.181541 | 0.224838 | -0.760696 | 0.404942 |

536 rows × 8 columns

```
y_resampled.value_counts()
```

```
1    268
0    268
Name: Outcome, dtype: int64
```

### 3) KNN 분석

#### A. KNN : metric = Euclidean

```
from sklearn.model_selection import train_test_split
training_data, validation_data , training_labels, validation_labels = train_test_split(X_resampled, y_resampled, test_size = 0.3, random_st
```

```
print(len(training_data))
print(len(training_labels))

print(len(validation_data))
print(len(validation_labels))
```

```
375
375
161
161
```

위의 전처리 과정을 통해 Scaling과 Shuffle을 해준 데이터를 위와 같이 7:3으로 Train과 Test Set으로 구분했다. Train data에는 375개의 데이터, Test(Validation) data에는 161개의 데이터로 나눠졌다. 그 이후 sklearn에 내장되어 있는 KneighborsClassifier을 통해 모델링을 해주었다.

```
# knn 모델 (k=3)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 3, weights="distance", metric="euclidean")
```

```
classifier.fit(training_data, training_labels)
```

```
KNeighborsClassifier(metric='euclidean', n_neighbors=3, weights='distance')
```

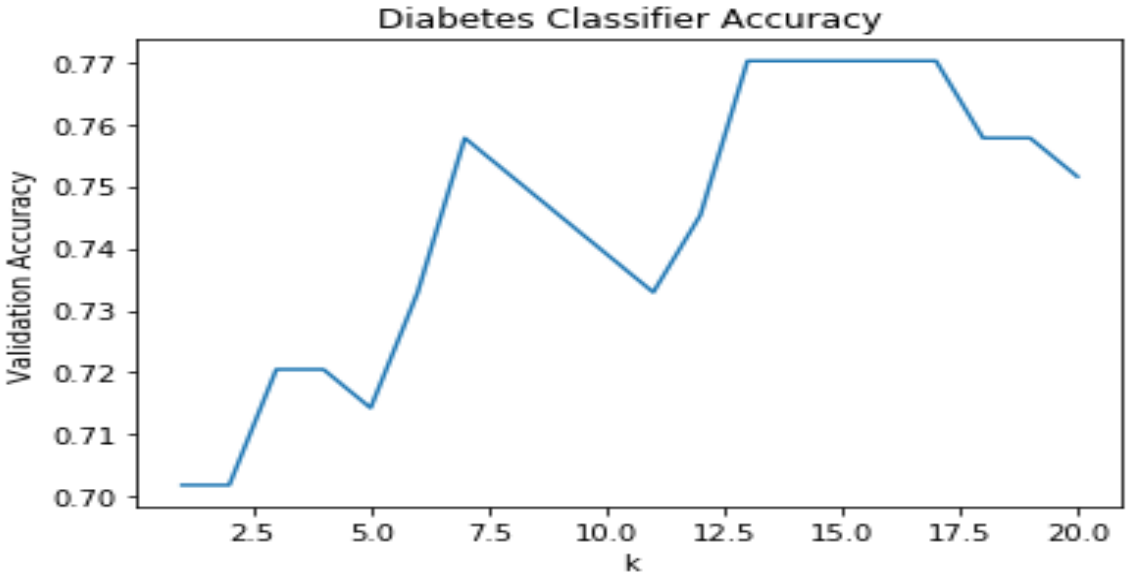
```
classifier.predict(validation_data)
```

```
array([0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 0], dtype=int64)
```

```
print(classifier.score(validation_data, validation_labels)) # accuracy
```

```
0.7204968944099379
```

K=3 일 때는 위와 같이, Score(Accuracy)가 약 0.72(72%)정도 나왔다. 더 최적의 k를 찾기 위해, k 값마다 Score(Accuracy)를 확인해주었다.




```
max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda x: x+1, test_scores_ind))))
```

```
Max test score 77.01863354037268 % and k = [13, 14, 15, 16, 17]
```

K값에 따라 다음과 같이 Accuracy가 나오게 되었다. 밑의 코드를 통해 정확한 Max Score를 다시 한 번 확인했다. k=13부터 17까지 Score가 같았고, 그에 따른 최적의 K = 13이라고 판단했다. (제일 크기가 작기 때문) 그때의 Accuracy는 약 77.01%가 나왔다.

Confusion Matrix 를 확인한 결과 다음과 같았다.

| Predicted | 0 | 1 |
|---|---|---|
| Actual | | |
| 0 | 57 | 24 |
| 1 | 13 | 67 |

- 올바르게 예측한 경우는 (0,0)과 (1,1)인 총 124가지, 잘못 예측한 경우는 총 37가지였다.

**B. KNN : metric = manhattan**

다음은 거리 계산을 'manhattan' 방식으로 한 과정이다.

```
# knn 모델 (k=3)
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 3, weights="distance", metric="manhattan")
```

```
classifier.fit(training_data, training_labels)
```

```
KNeighborsClassifier(metric='manhattan', n_neighbors=3, weights='distance')
```

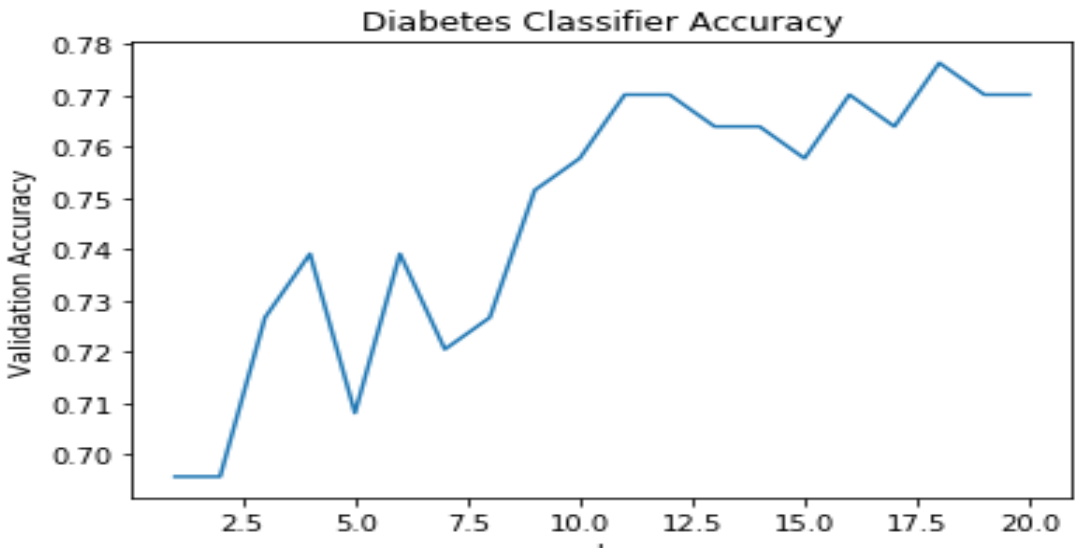
```
classifier.predict(validation_data)
```

```
array([0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0], dtype=int64)
```

```
print(classifier.score(validation_data, validation_labels)) # accuracy
```

```
0.7267080745341615
```

K=3 일 때는 위와 같이, Score(Accuracy)가 약 0.727(72.7%)정도 나왔다. 더 최적의 k를 찾기 위해, k 값마다 Score(Accuracy)를 확인해주었다.




```
max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda x: x+1, test_scores_ind))))
```

```
Max test score 77.63975155279503 % and k = [18]
```

다음과 같이 K = 18일 때, Accuracy가 가장 높게 나왔다. 그때의 Accuracy는 77.64%가 나왔다.

Confusion Matrix 를 확인한 결과 다음과 같았다.

- 올바르게 예측한 경우는 (0,0)과 (1,1)인 총 125가지, 잘못 나눈 예측한 경우는 총 36가지였다.

| Predicted | 0 | 1 |
|---|---|---|
| Actual | | |
| 0 | 60 | 21 |
| 1 | 15 | 65 |

**3) Category 데이터 KNN 분석 결론**

| | K = 13 (Euclidean) | K = 18 (Manhattan) |
|---|---|---|
| **옳바르게 예측** | 124 | 125 |
| **잘못 예측** | 37 | 36 |
| **Score** | 77.02% | 77.64% |

Pima Indians 당뇨병 데이터를 통해 Category 종속변수에 대한 KNN분석을 한 결과, Euclidean 거리 방식으로는 K = 13일 때, Accuracy가 가장 높게 나왔고, Manhattan 방식으로는 K = 18일 때 Accuracy가 가장 높게 나왔다. 두 방법으로 비교 분석을 했을 때, Score는 Manhattan 방식이 약 0.6% 나마 높게 나왔다. 이 Dataset에 대해서는 Manhattan 방식이 조금 더 적합해 보인다.

## 2. Association Rule

### 2.1 Association Rule

연관 규칙이란, 데이터 내부에 존재하는 품목, 혹은 요소들 간의 상호 관련성을 의미한다. 이러한 상호 관련성을 찾는 분석법을 연관규칙 분석이라고 한다. 흔히 '장바구니 분석'이라고 알려져 있다.

이를 통해 주어진 데이터로부터 규칙을 만들어낼 수 있으며, 규칙은 '만일 ~라면'에 해당하는 조건절(Antecedent)과 뒷부분에 해당하는 결과절(Consequent)로 구성되어 있고, 조건절 또는 결과절을 구성하는 아이템들의 집합을 아이템 집합(Item set)이라 한다.

단 여기에서 조건절 아이템 집합과 결과절 아이템 집합은 말 그대로 집합, 여러 개 아이템이 들어가도 되지만 상호배반(mutually exclusive)이 되어야 한다는 조건이 있다.

이러한 많은 규칙가운데 규칙의 효용성을 드러내 주는 Support(지지도), Confidence(신뢰도), Lift(향상도)라는 3가지 지표가 있다

- **Support(지지도)**: 전체 관측치 중 A와 B를 모두 포함하는 관측치의 비율이며, 빈발 아이템 집합을 판별하는데 쓰인다.

$$Support = \frac{\text{A와 B를 동시에 포함하는 항목 수}}{\text{전체 항목 수}} = \Pr(A \cap B)$$

- **Confidence(신뢰도)**: 전체 관측치에서 A를 포함하는 관측치 중 B를 포함하는 관측치의 비율이며, 아이템 집합 간의 연관성 강도를 측정하는데 쓰인다.

$$Confidence = \frac{\text{A와 B를 동시에 포함하는 항목 수}}{\text{A를 포함하는 항목 수}} = \frac{\Pr(A \cap B)}{\Pr(A)}$$

- **Lift(향상도)**: A와 B를 모두 포함하는 관측치와 A와 B를 각각 포함하는 관측치의 비율이며, 생성된 규칙이 실제 효용가치가 있는지를 판별하는 데 사용되며, 조건절과 결과절이 서로 독립일 때와 비교해, 두 사건이 동시에 얼마나 발생하는지 비율로 나타난다.

$$Lift = \frac{\text{A와 B를 동시에 포함하는 항목 수}}{\text{A를 포함하는 항목 수} \times \text{B를 포함하는 항목 수}} = \frac{\Pr(A \cap B)}{\Pr(A) \times \Pr(B)}$$

만약 A와 B가 독립적이라면 Lift값이 1이 나올 것이며, 관련성이 있다면 Lift값이 1보다 크거나 작게 나올 것이다. 1보다 클 경우 양의 상관관계, 1보다 작을 경우 음의 상관관계가 있다고 판단할 수 있다. 또한 향상도가 2라면 두 사건이 독립이라는 걸 가정했을 때 대비 2배로

긍정적인 연관관계를 나타난다.

규칙의 효용성은 지지도, 신뢰도, 향상도 세 가지를 모두 반영해 평가한다. 임의의 규칙1이 규칙2보다 효과적인 규칙이라는 이야기를 하려면 세 지표 모두 클 경우에만 그렇다고 결론을 내릴 수 있다.

### 2.2 데이터 설명

AR을 진행하기 위한 데이터로 여행 리뷰 평점 데이터를 사용하였다. 해당 데이터는 구글 리뷰에서 가져온 데이터로, 5455명의 사용자가 유럽의 공원, 교회, 리조트, 해변 등등 총 24가지 장소유형에 어떤 평점을 남겼는지를 모아서 만들어져 있다.

```
df=pd.read_csv("google_review_ratings.csv")
df
```

| | User | Category 1 | Category 2 | Category 3 | Category 4 | Category 5 | Category 6 | Category 7 | Category 8 | Category 9 | ... | Category 16 | Category 17 | Category 18 | Category 19 | Category 20 | Category 21 | Category 22 | Category 23 | Category 24 | Unnamed: 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | User 1 | 0.00 | 0.00 | 3.63 | 3.65 | 5.00 | 2.92 | 5.00 | 2.35 | 2.33 | ... | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 | NaN |
| 1 | User 2 | 0.00 | 0.00 | 3.63 | 3.65 | 5.00 | 2.92 | 5.00 | 2.64 | 2.33 | ... | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 | NaN |
| 2 | User 3 | 0.00 | 0.00 | 3.63 | 3.63 | 5.00 | 2.92 | 5.00 | 2.64 | 2.33 | ... | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 | NaN |
| 3 | User 4 | 0.00 | 0.50 | 3.63 | 3.63 | 5.00 | 2.92 | 5.00 | 2.35 | 2.33 | ... | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 | NaN |
| 4 | User 5 | 0.00 | 0.00 | 3.63 | 3.63 | 5.00 | 2.92 | 5.00 | 2.64 | 2.33 | ... | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 | NaN |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 5451 | User 5452 | 0.91 | 5.00 | 4.00 | 2.79 | 2.77 | 2.57 | 2.43 | 1.09 | 1.77 | ... | 0.66 | 0.65 | 0.66 | 0.69 | 5.00 | 1.05 | 5.0 | 5.0 | 1.56 | NaN |
| 5452 | User 5453 | 0.93 | 5.00 | 4.02 | 2.79 | 2.78 | 2.57 | 1.77 | 1.07 | 1.76 | ... | 0.65 | 0.64 | 0.65 | 1.59 | 1.62 | 1.06 | 5.0 | 5.0 | 1.09 | NaN |
| 5453 | User 5454 | 0.94 | 5.00 | 4.03 | 2.80 | 2.78 | 2.57 | 1.75 | 1.05 | 1.75 | ... | 0.65 | 0.63 | 0.64 | 0.74 | 5.00 | 1.07 | 5.0 | 5.0 | 1.11 | NaN |
| 5454 | User 5455 | 0.95 | 4.05 | 4.05 | 2.81 | 2.79 | 2.44 | 1.76 | 1.03 | 1.74 | ... | 0.64 | 0.63 | 0.64 | 0.75 | 5.00 | 1.08 | 5.0 | 5.0 | 1.12 | NaN |
| 5455 | User 5456 | 0.95 | 4.07 | 5.00 | 2.82 | 2.80 | 2.57 | 2.42 | 1.02 | 1.74 | ... | 0.64 | 0.62 | 0.63 | 0.78 | 5.00 | 1.08 | 5.0 | 5.0 | 1.17 | NaN |

5456 rows × 26 columns

그리고 column이름에 규칙해석의 용이성을 위해 실제 의미를 반영하고 NaN값을 없애주었다.

```
df = df.drop("Unnamed: 25", axis=1)
```

```
df.columns = ['User',
              '교회', '리조트', '해변', '공원', '영화관', '박물관',
              '쇼핑몰', '동물원', '식당', '펍/바', '지역서비스', '버거/피자',
              '호텔', '주스바', '미술관', '클럽', '수영장', '체육관',
              '빵집', '스파', '카페', '경치', '기념관', '정원']
```

```
df
```

| | User | 교회 | 리조트 | 해변 | 공원 | 영화관 | 박물관 | 쇼핑몰 | 동물원 | 식당 | ... | 미술관 | 클럽 | 수영장 | 체육관 | 빵집 | 스파 | 카페 | 경치 | 기념관 | 정원 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | User 1 | 0.00 | 0.00 | 3.63 | 3.65 | 5.00 | 2.92 | 5.00 | 2.35 | 2.33 | ... | 1.74 | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 |
| 1 | User 2 | 0.00 | 0.00 | 3.63 | 3.65 | 5.00 | 2.92 | 5.00 | 2.64 | 2.33 | ... | 1.74 | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 |
| 2 | User 3 | 0.00 | 0.00 | 3.63 | 3.63 | 5.00 | 2.92 | 5.00 | 2.64 | 2.33 | ... | 1.74 | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 |
| 3 | User 4 | 0.00 | 0.50 | 3.63 | 3.63 | 5.00 | 2.92 | 5.00 | 2.35 | 2.33 | ... | 1.74 | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 |
| 4 | User 5 | 0.00 | 0.00 | 3.63 | 3.63 | 5.00 | 2.92 | 5.00 | 2.64 | 2.33 | ... | 1.74 | 0.59 | 0.50 | 0.00 | 0.50 | 0.00 | 0.00 | 0.0 | 0.0 | 0.00 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 5451 | User 5452 | 0.91 | 5.00 | 4.00 | 2.79 | 2.77 | 2.57 | 2.43 | 1.09 | 1.77 | ... | 5.00 | 0.66 | 0.65 | 0.66 | 0.69 | 5.00 | 1.05 | 5.0 | 5.0 | 1.56 |
| 5452 | User 5453 | 0.93 | 5.00 | 4.02 | 2.79 | 2.78 | 2.57 | 1.77 | 1.07 | 1.76 | ... | 0.89 | 0.65 | 0.64 | 0.65 | 1.59 | 1.62 | 1.06 | 5.0 | 5.0 | 1.09 |
| 5453 | User 5454 | 0.94 | 5.00 | 4.03 | 2.80 | 2.78 | 2.57 | 1.75 | 1.05 | 1.75 | ... | 0.87 | 0.65 | 0.63 | 0.64 | 0.74 | 5.00 | 1.07 | 5.0 | 5.0 | 1.11 |
| 5454 | User 5455 | 0.95 | 4.05 | 4.05 | 2.81 | 2.79 | 2.44 | 1.76 | 1.03 | 1.74 | ... | 5.00 | 0.64 | 0.63 | 0.64 | 0.75 | 5.00 | 1.08 | 5.0 | 5.0 | 1.12 |
| 5455 | User 5456 | 0.95 | 4.07 | 5.00 | 2.82 | 2.80 | 2.57 | 2.42 | 1.02 | 1.74 | ... | 0.85 | 0.64 | 0.62 | 0.63 | 0.78 | 5.00 | 1.08 | 5.0 | 5.0 | 1.17 |

5456 rows × 25 columns

또한 데이터가 범주형 변수가 아닌 0~5까지의 변수형 범주이기 때문에 사용자가 해당 장소를 긍정적으로 평가했는지, 부정적으로 평가했는지 나타내는 범주형 변수로 바꾸어 주었다.
'긍정적인 평가'의 기준으로 전체 평점 데이터의 평균인 2.0점은 아예 평점을 내리지 않은 0점도 포함하기 때문에 적절하지 않다고 판단했다. 1~5점의 평균인 3점의 경우 데이터의 신뢰도가 너무 높게 측정되는 현상이 나타났다. 따라서 전체 평점 데이터를 오름차순으로 나열했을 때 상위 20%에 해당하는 3.21로 '긍정적인 평가'의 기준을 결정하였다.

In [119]:
```
for i in range(1, len(df.columns)):
    df.fillna(i)
    df.iloc[:,i] = df.iloc[:,i].apply(lambda x: 1 if float(x) >= 3.21 else 0)
```

In [120]:
```
df
```

Out [120]:

| | User | 교회 | 리조트 | 해변 | 공원 | 영화관 | 박물관 | 쇼핑몰 | 동물원 | 식당 | ... | 미술관 | 클럽 | 수영장 | 체육관 | 빵집 | 스파 | 카페 | 경치 | 기념관 | 정원 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | User 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | User 2 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | User 3 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | User 4 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | User 5 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 5451 | User 5452 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 5452 | User 5453 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 5453 | User 5454 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 5454 | User 5455 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 5455 | User 5456 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |

5456 rows × 25 columns

또한 각 transaction(이 경우엔 user)마다의 관측치들을 한 list안에 넣고 one-hot encoding을 하여 association rule을 적용시키기 위한 준비를 했다.

In [128]:
```
te = TransactionEncoder()
te_ary = te.fit(target_list).transform(target_list) # 위 장바구니data를 one-hot encoding
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)

df_encoded
```

Out[128]:

| | 경치 | 공원 | 교회 | 기념관 | 동물원 | 리조트 | 미술관 | 박물관 | 버거/피자 | 뱅집 | ... | 영화관 | 정원 | 주스바 | 지역서비스 | 체육관 | 카페 | 클럽 | 펍/바 | 해변 | 호텔 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | False | True | False | False | False | False | False | False | False | False | ... | True | False | False | False | False | False | False | False | True | False |
| 1 | False | True | False | False | False | False | False | False | False | False | ... | True | False | False | False | False | False | False | False | True | False |
| 2 | False | True | False | False | False | False | False | False | False | False | ... | True | False | False | False | False | False | False | False | True | False |
| 3 | False | True | False | False | False | False | False | False | False | False | ... | True | False | False | False | False | False | False | False | True | False |
| 4 | False | True | False | False | False | False | False | False | False | False | ... | True | False | False | False | False | False | False | False | True | False |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 5451 | True | False | False | True | False | True | True | False | False | False | ... | False | False | False | False | False | False | False | False | True | False |
| 5452 | True | False | False | True | False | True | False | False | False | False | ... | False | False | False | False | False | False | False | False | True | False |
| 5453 | True | False | False | True | False | True | False | False | False | False | ... | False | False | False | False | False | False | False | False | True | False |
| 5454 | True | False | False | True | False | True | True | False | False | False | ... | False | False | False | False | False | False | False | False | True | False |
| 5455 | True | False | False | True | False | True | False | False | False | False | ... | False | False | False | False | False | False | False | False | True | False |

5456 rows × 24 columns

## 2.3 **Association Rule적용**

### 2.3.1 Apriori 알고리즘 적용을 통한 데이터 항목화

Apriori알고리즘은 항목에 최소 지지도를 설정한 후 이보다 높은 항목들을 묶은 후, 또 다시 최소 지지도를 기준으로 높은 항목의 묶음들을 찾는 것을 반복하는 알고리즘이다.
가능한 모든 경우의 수를 탐색하여 지지도, 신뢰도, 향상도가 높은 규칙들을 찾아내고싶지만, 아이템 수가 증가할수록 계산에 소요되는 시간이 기하급수적으로 증가하게된다. 해당 데이터의 경우 두개의 묶음만 고려한다 해도 552개의 묶음이 생성되고, 3개 묶음은 무려 12,144개 생성되기에 Apriori 알고리즘을 적용하여 빈발 집합(frequent item sets)만을 고려해 연관 규칙을 생성하기로 한다.

먼저 최소 지지도 요건을 설정한다. 최소 지지도(support)값을 0.1로 지정하게 되면 24개의 여행지중 약 70%인 17개의 여행지가 남게되고, 30%인 7개의 여행지가 제외되게된다. 이 경우 apriori 알고리즘에 따르면, 여전히 많은 경우의 수를 고려해야하기에 최소 지지도 요건을 올려주기로하였다.
그리하여 지지도를 0.2로 올리게되면 약 52%인 13개의 여행지가 선택되고 약 48%의 여행지를 아이템집합에서 제외할수있게되기에 계산을 위한 시간을 줄이면서도 많은 여행지를 제외하지않게되어 적절한 균형점을 이루었다고 판단하여 진행하였다.

```
# support >= 0.1
frequent_itemsets = apriori(df_encoded, min_support=0.1, use_colnames=True)
frequent_itemsets.iloc[0:20]
```

| | support | itemsets |
|---|---|---|
| 0 | 0.209861 | (경치) |
| 1 | 0.305352 | (공원) |
| 2 | 0.130865 | (기념관) |
| 3 | 0.301686 | (동물원) |
| 4 | 0.213710 | (리조트) |
| 5 | 0.290872 | (미술관) |
| 6 | 0.397544 | (박물관) |
| 7 | 0.154142 | (버거/피자) |
| 8 | 0.558651 | (쇼핑몰) |
| 9 | 0.428336 | (식당) |
| 10 | 0.381232 | (영화관) |
| 11 | 0.100440 | (정원) |
| 12 | 0.242669 | (주스바) |
| 13 | 0.266679 | (지역서비스) |
| 14 | 0.344391 | (펍/바) |
| 15 | 0.217559 | (해변) |
| 16 | 0.191532 | (호텔) |
| 17 | 0.110887 | (공원, 경치) |
| 18 | 0.103922 | (경치, 영화관) |
| 19 | 0.164589 | (공원, 박물관) |

```
# support >= 0.2
frequent_itemsets = apriori(df_encoded, min_support=0.2, use_colnames=True)
frequent_itemsets.iloc[0:20]
```

| | support | itemsets |
|---|---|---|
| 0 | 0.209861 | (경치) |
| 1 | 0.305352 | (공원) |
| 2 | 0.301686 | (동물원) |
| 3 | 0.213710 | (리조트) |
| 4 | 0.290872 | (미술관) |
| 5 | 0.397544 | (박물관) |
| 6 | 0.558651 | (쇼핑몰) |
| 7 | 0.428336 | (식당) |
| 8 | 0.381232 | (영화관) |
| 9 | 0.242669 | (주스바) |
| 10 | 0.266679 | (지역서비스) |
| 11 | 0.344391 | (펍/바) |
| 12 | 0.217559 | (해변) |
| 13 | 0.239553 | (공원, 영화관) |
| 14 | 0.269611 | (쇼핑몰, 동물원) |
| 15 | 0.258614 | (식당, 동물원) |
| 16 | 0.212610 | (동물원, 펍/바) |
| 17 | 0.318732 | (쇼핑몰, 박물관) |
| 18 | 0.203629 | (식당, 박물관) |
| 19 | 0.264663 | (박물관, 영화관) |

### 2.3.2 Confidence(신뢰도) 관점 규칙 추출

앞에서 support값 0.2를 기준으로 나온 frequent_itemsets를 기준으로 최소 confidence값을 설정해 유의미한 규칙 추출 진행하였다. 최소 confidence값은 0.5를 지정해 antecedents를 선택한 User의 과반수 이상은 consequents를 선택하는 규칙을 추출하고자 하였다.

```
# confidence >= 0.5
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
rules
# min_threshold의 default = 0.8
# antecedents : 조건절
# consequents : 결과절
# 만일 (조건절)이라면 (결과절).
```

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| 0 | (공원) | (영화관) | 0.301136 | 0.333944 | 0.229289 | 0.761412 | 2.280057 | 0.128726 | 2.791657 |
| 1 | (영화관) | (공원) | 0.333944 | 0.301136 | 0.229289 | 0.686608 | 2.280057 | 0.128726 | 2.229999 |
| 2 | (동물원) | (쇼핑몰) | 0.244135 | 0.535007 | 0.207295 | 0.849099 | 1.587079 | 0.076681 | 3.081444 |
| 3 | (식당) | (동물원) | 0.396628 | 0.244135 | 0.205462 | 0.518022 | 2.121869 | 0.108631 | 1.568257 |
| 4 | (동물원) | (식당) | 0.244135 | 0.396628 | 0.205462 | 0.841592 | 2.121869 | 0.108631 | 3.808967 |
| 5 | (박물관) | (쇼핑몰) | 0.343292 | 0.535007 | 0.258614 | 0.753337 | 1.408087 | 0.074951 | 1.885133 |
| 6 | (박물관) | (영화관) | 0.343292 | 0.333944 | 0.217559 | 0.633743 | 1.897750 | 0.102918 | 1.818546 |
| 7 | (영화관) | (박물관) | 0.333944 | 0.343292 | 0.217559 | 0.651482 | 1.897750 | 0.102918 | 1.884287 |
| 8 | (식당) | (쇼핑몰) | 0.396628 | 0.535007 | 0.289406 | 0.729667 | 1.363845 | 0.077208 | 1.720075 |
| 9 | (쇼핑몰) | (식당) | 0.535007 | 0.396628 | 0.289406 | 0.540939 | 1.363845 | 0.077208 | 1.314361 |
| 10 | (펍/바) | (쇼핑몰) | 0.309567 | 0.535007 | 0.203812 | 0.658378 | 1.230596 | 0.038191 | 1.361131 |
| 11 | (식당) | (펍/바) | 0.396628 | 0.309567 | 0.256598 | 0.646950 | 2.089852 | 0.133815 | 1.955623 |
| 12 | (펍/바) | (식당) | 0.309567 | 0.396628 | 0.256598 | 0.828893 | 2.089852 | 0.133815 | 3.526284 |

이를 통해 규칙을 추출하면 다음과 같습니다.

이는 Lift는 고려하지 않고 단순히 신뢰도 값만 고려한 규칙으로, 신뢰도가 1에 가까울 경우 독립적임을 의미하기 때문에 Lift를 함께 고려해 규칙 추출을 진행하였다.

### 2.3.3 Lift(향상도)를 고려한 최종 규칙 추출 및 해석

Lift의 값이 1에 가까울수록 두 항목이 독립적임을 의미하고 1에서 양의 방향으로 멀어질수록 양의 연관성이 강하게 나타난다. 3.3.1과 3.3.2를 통해 최소지지도를 설정한 후 최소 신뢰도를

In [143]:
```
# confidence >= 0.5
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
rules
# min_threshold의 default = 0.8
# antecedents : 조건절
# consequents : 결과절
# 만일 (조건절)이라면 (결과절).
```

Out [143]:

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction |
|---|---|---|---|---|---|---|---|---|---|
| 0 | (영화관) | (공원) | 0.333944 | 0.301136 | 0.229289 | 0.686608 | 2.280057 | 0.128726 | 2.229999 |
| 1 | (공원) | (영화관) | 0.301136 | 0.333944 | 0.229289 | 0.761412 | 2.280057 | 0.128726 | 2.791657 |
| 2 | (동물원) | (쇼핑몰) | 0.244135 | 0.535007 | 0.207295 | 0.849099 | 1.587079 | 0.076681 | 3.081444 |
| 3 | (동물원) | (식당) | 0.244135 | 0.396628 | 0.205462 | 0.841592 | 2.121869 | 0.108631 | 3.808967 |
| 4 | (식당) | (동물원) | 0.396628 | 0.244135 | 0.205462 | 0.518022 | 2.121869 | 0.108631 | 1.568257 |
| 5 | (박물관) | (쇼핑몰) | 0.343292 | 0.535007 | 0.258614 | 0.753337 | 1.408087 | 0.074951 | 1.885133 |
| 6 | (영화관) | (박물관) | 0.333944 | 0.343292 | 0.217559 | 0.651482 | 1.897750 | 0.102918 | 1.884287 |
| 7 | (박물관) | (영화관) | 0.343292 | 0.333944 | 0.217559 | 0.633743 | 1.897750 | 0.102918 | 1.818546 |
| 8 | (쇼핑몰) | (식당) | 0.535007 | 0.396628 | 0.289406 | 0.540939 | 1.363845 | 0.077208 | 1.314361 |
| 9 | (식당) | (쇼핑몰) | 0.396628 | 0.535007 | 0.289406 | 0.729667 | 1.363845 | 0.077208 | 1.720075 |
| 10 | (펍/바) | (쇼핑몰) | 0.309567 | 0.535007 | 0.203812 | 0.658378 | 1.230596 | 0.038191 | 1.361131 |
| 11 | (펍/바) | (식당) | 0.309567 | 0.396628 | 0.256598 | 0.828893 | 2.089852 | 0.133815 | 3.526284 |
| 12 | (식당) | (펍/바) | 0.396628 | 0.309567 | 0.256598 | 0.646950 | 2.089852 | 0.133815 | 1.955623 |

설정해 규칙을 추출했다.

그 결과로 추출된 다음 규칙들은 모두 Lift의 값이 1.2를 넘었기에 충분히 양의 연관성을 가지고 있다고 판단할 수 있었다. 두번의 과정을 통해 나온 결과값은 최소 신뢰도와 최소 지지도를 만족시켰기 때문에 최종 규칙은 Lift값을 기준으로 상위 5개를 선정하기로 결정하였다. 규칙들은 다음과 같다.

① 공원과 영화관 모두 긍정적으로 평가한 사람은 전체의 약 22%이며, 공원을 긍정적으로 평가한 사람의 약 76%는 영화관을 긍정적으로 평가하였다고 말할수있다. 또한 lift값이 2.28임을 통해 공원과 영화관이 독립일 경우에 비해 약 2.28배의 유의미한 관계를 가진다고 해석할수있다.
(Support=0.229289, Confidence: 0.761412, Lift: 2.280057)

② 동물원과 식당 모두 긍정적으로 평가한 사람은 전체의 약 20%이며, 동물원을 긍정적으로 평가한 사람의 약 84%는 식당을 긍정적으로 평가하였다고 말할수있다. 또한 lift값이 약 2.12임을 통해 동물원과 식당이 독립일 경우에 비해 약 2.12배의 유미한 관계를 가진다고 해석할수있다.
(Support=0.205462, Confidence: 0.841592, Lift: 2.121869)

③ 펍/바와 식당 모두 긍정적으로 평가한 사람은 전체의 약 25%이며, 펍/바를 긍정적으로 평가한 사람의 약 82%는 식당을 긍정적으로 평가하였다고 말할수있다. 또한 lift값이 약 2.08임을 통해 펍/바와 식당이 독립일 경우에 비해 약 2.08배의 유미한 관계를 가진다고 해석할수있다.
(Support=0.256598, Confidence: 0.828893, Lift: 2.089852)

④ 영화관과 박물관 모두 긍정적으로 평가한 사람은 전체의 약 21%이며, 영화관을 긍정적으로 평가한 사람의 약 82%는 박물관을 긍정적으로 평가하였다고 말할수있다. 또한 lift값이 약 1.89임을 통해 영화관과 박물관이 독립일 경우에 비해 약 1.89배의 유미한 관계를 가진다고 해석할수있다.
(Support=0.217599, Confidence: 0.651482, Lift: 1.897750)

⑤ 동물원과 쇼핑몰 모두 긍정적으로 평가한 사람은 전체의 약 20%이며, 동물원을 긍정적으로 평가한 사람의 약 84%는 쇼핑몰을 긍정적으로 평가하였다고 말할수있다. 또한 lift값이 약 1.58임을 통해 동물원과 쇼핑몰이 독립일 경우에 비해 약 1.58배의 유미한 관계를 가진다고 해석할수있다.
(Support=0.207295, Confidence: 0.849099, Lift: 1.587079)