데이터 전처리

데이터 전처리

- 데이터 전처리(Preprocessing)
  - 데이터 전처리는 ML 알고리즘 만큼 중요, Garbage In Garbage Out

- 데이터 전처리 종류
  - 데이터 클린징
  - 결손 데이타(NaN/Null) 처리
  - 데이터 인코딩(레이블 인코딩, 원-핫 인코딩)
  - 피처 스케일링(Feature Scaling)과 정규화
  - 이상치 제거
  - Feature 선택, 추출 및 가공 → Feature Engineering

데이터 전처리

■ 데이터 전처리(Preprocessing)의 종류

▪ 결손 데이타(NaN/Null) 처리

- Null 값이 얼마 안되는 경우는 피처의 평균값 등으로 간단히 대체, Null 값이 대부분이라면 해당 피처는 삭제하는 것이 바람직
- Null 값이 일정 수준 이상인 경우 → Null을 단순히 평균값으로 대체할 경우 왜곡이 심해짐, 정밀한 대체 값 선정할 필요

▪ 데이터 인코딩 - 레이블 인코딩(Label encoding), 원-핫인코딩(One-Hot encoding)

- **사이킷런 머신러닝 알고리즘은 문자열을 입력 값으로 허용하지 않음**

  → 모든 문자열 값은 인코딩 되어 숫자형으로 변환해야 함

- **문자열 피쳐의 종류**

  ≫카테고리형 피처 - 숫자형 코드 값으로 표현

  ≫텍스트형 피처 - 피처 벡터화 등의 기법으로 벡터화 하거나 불필요하다고 생각되면 피처 삭제

  (예) 주민번호, 단순 문자열 아이디

▪ 피처 스케일링(Feature Scaling)과 정규화

- 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업
- 피처 스케일링 방법 : 표준화(Standardization), 정규화(Normalization)

▪ 이상치(outlier) 제거

▪ Feature 선택, 추출 및 가공 → Feature Engineering

데이터 전처리

결손 데이터(Missing Data) 처리 - NaN/Null 처리

- 통계처리에 있어서 missing data 처리 문제는 연구자에게 가장 골치 아픈 문제 중 하나
- 실제 우리가 사용하는 대부분의 데이터는 누락된 데이터가 존재
- 분석하기 전에 누락된 자료를 제거하거나 합당한 대체 값으로 교체
- 통계 패키지가 누락된 자료에 대한 디폴트 방법을 제공하기도 함, 자료의 특성에 따라 누락 데이터를 처리
- **사이킷런의 SimpleImputer 클래스 사용, 판다스의 isnan() → fillna() 또는 dropna() 사용**

1) 데이터 로드

```
1  import pandas as pd
2  import numpy as np
3  from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
4
5  # 데이터 로드
6  path = "../data/DataPreprocess.csv"
7  df1 = pd.read_csv(path)
8  df1.head()
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes


```
1  df1.shape
```

```
(10, 4)
```

데이터 전처리

- 결손 데이터(Missing Data) 처리 - NaN/Null 처리
- 2) 데이터와 레이블 나누기

```
1  # 데이터와 레이블 나누기 - 종속변수(반응변수)와 독립변수(설명변수) 나누기
2  x = df1.values[:, :-1]   # 데이터
3  y = df1.values[:, -1]    # 레이블(정답)
4  x, y
```

```
(array([['France', 44.0, 72000.0],
        ['Spain', 27.0, 48000.0],
        ['Germany', 30.0, 54000.0],
        ['Spain', 38.0, 61000.0],
        ['Germany', 40.0, nan],
        ['France', 35.0, 58000.0],
        ['Spain', nan, 52000.0],
        ['France', 48.0, 79000.0],
        ['Germany', 50.0, 83000.0],
        ['France', 37.0, 67000.0]], dtype=object),
 array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
       dtype=object))
```

- 3) 결손 데이터 처리

사이킷런의 SimpleImputer 클래스

strategy : string, optional (default='mean')

- 결측값 대치 값
- str 클래스
- strategy='mean' 평균값으로 대치 (디폴트)
- strategy='median' 중앙값으로 대치
- strategy='most_frequent' 최빈값 (mode)으로 대치
- strategy='constant', fill_value=1 특정값으로 대치, 예) transformer = SimpleImputer(strategy='constant', fill_value=1)

데이터 전처리

- 결손 데이터(Missing Data) 처리 - NaN/Null 처리
- 3) 결손 데이터 처리

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.impute import SimpleImputer
4
5 imputer = SimpleImputer(strategy='mean')
6 imputer = transformer.fit(x[:, 1:3])
7 x[:, 1:3] = transformer.transform(x[:, 1:3])
8 #트랜스포머의 transform() 함수는 결과를 넘파이 배열로 리턴
9 x
```

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

```
1 df2 = pd.DataFrame(x[:, 1:3])
2 df2.describe
```

```
<bound method NDFrame.describe of          0        1
0       44    72000
1       27    48000
2       30    54000
3       38    61000
4       40  63777.8
5       35    58000
6  38.7778    52000
7       48    79000
8       50    83000
9       37    67000>
```

데이터 전처리

- 피처 스케일링(Feature Scaling)과 정규화
  - 서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업 → 피처 스케일링
  - 피처 스케일링 방법 : 표준화(Standardization), 정규화(Normalization)

- 1) 표준화(Standardization)
  - 값의 분포를 평균이 0이고 분산이 1인 가우시안 정규분포를 가진 값으로 변환 -> StandardScaler

$$\frac{xi - mean(x)}{stdev(x)}$$

  - 서포트 벡터머신이나 선형회귀, 로지스틱 회귀는 데이터가 정규분포를 가지고 있다고 가정하고 구현됐기 때문에 사전에 데이터를 표준화를 적용하는 것은 예측 성능 향상에 중요한 요소가 될 수 있다.

- 2) 정규화(Normalization)
  - 서로 다른 피처의 크기를 통일하기 위해 크기를 변환해 주는 개념 -> MinMaxScaler

$$\frac{xi - min(x)}{max(x)-min(x)}$$

  - (예) 피처 A 거리 0~100KM, 피처 B 금액 0~100,000,000원 → 표준화 → 최소 0 ~ 최대 1로 변환
  - 데이터의 분포가 정규 분포가 아닐 경우 적용

데이터 전처리

- 피처 스케일링(Feature Scaling)과 정규화
- 1) 표준화 - 사이킷런의 StandardScaler 클래스 사용

```
1 from sklearn.preprocessing import StandardScaler # 표준화 지원 클래스
2
3 sc_x = StandardScaler()
4 sc_x.fit_transform(x[:, 1:3])
5 x[:, 1:3] = sc_x.transform(x[:, 1:3])
6 x
```

```
array([['France', 0.7199314321591973, 0.711012758872281],
       ['Spain', -1.6236751448696793, -1.3643758345927564],
       ['Germany', -1.2100975136292893, -0.845528686226497],
       ['Spain', -0.1072238303215827, -0.2402070131325274],
       ['Germany', 0.16849459050534396, nan],
       ['France', -0.5208014615619727, -0.499630587315674],
       ['Spain', nan, -1.0184777356819168],
       ['France', 1.2713682738130507, 1.3163344319662502],
       ['Germany', 1.5470866946399773, 1.6622325308770898],
       ['France', -0.24508304073504605, 0.2786401352337316]], dtype=object)
```

데이터 전처리

- 피처 스케일링(Feature Scaling)과 정규화
- 2) 정규화 - 사이킷런의 MinMaxScaler 사용

```
1  from sklearn.preprocessing import MinMaxScaler # 정규화 지원 모듈
2
3  mmsc_x = MinMaxScaler()
4  mmsc_x.fit_transform(x[:, 1:3])
5  x[:, 1:3] = mmsc_x.transform(x[:, 1:3])
6  x
```

```
array([['France', 0.7391304347826089, 0.6857142857142855],
       ['Spain', 0.0, 0.0],
       ['Germany', 0.1304347826086958, 0.1714285714285149],
       ['Spain', 0.4782608695652175, 0.3714285714285144],
       ['Germany', 0.5652173913043479, 0.4507936507936075],
       ['France', 0.3478260869565173, 0.2857142857142856],
       ['Spain', 0.5120772946859904, 0.1142857142857432],
       ['France', 0.9130434782608696, 0.8857142857142857],
       ['Germany', 1.0, 1.0],
       ['France', 0.4347826086956523, 0.5428571428571427]], dtype=object)
```

데이터 전처리

■ 실습 문제) 피처 스케일링(Feature Scaling)과 정규화

```
1  from sklearn.datasets import load_iris
2  import pandas as pd
3
4  # iris 데이터셋 불러오기
5  iris = load_iris()
6  iris_data = iris.data
7  iris_df = pd.DataFrame(data=iris_data, columns=iris.feature_names)
8
9  print(iris_df)
10 print('feature 들의 평균 값')
11 print(iris_df.mean())
12 print('₩nfeature 들의 분산 값')
13 print(iris_df.var())
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                  5.1               3.5                1.4               0.2
1                  4.9               3.0                1.4               0.2
2                  4.7               3.2                1.3               0.2
3                  4.6               3.1                1.5               0.2
4                  5.0               3.6                1.4               0.2
..                 ...               ...                ...               ...
145                6.7               3.0                5.2               2.3
146                6.3               2.5                5.0               1.9
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8
```

```
feature 들의 평균 값
sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
dtype: float64

feature 들의 분산 값
sepal length (cm)    0.685694
sepal width (cm)     0.189979
petal length (cm)    3.116278
petal width (cm)     0.581006
dtype: float64
```

데이터 전처리

■ 실습 문제) 피처 스케일링(Feature Scaling)과 정규화

표준화 - StandardScaler

```
1  from sklearn.preprocessing import StandardScaler
2
3  # StandardScaler객체 생성
4  scaler = StandardScaler()
5
6  # StandardScaler 로 데이터 셋 변환. fit( ) 과 transform( ) 호출.
7  scaler.fit(iris_df)
8  iris_scaled = scaler.transform(iris_df)
9
10 # transform( )시 scale 변환된 데이터 셋이 numpy ndarry로 반환되어 이를 DataFrame으로 변환
11 iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
12
13 print(iris_df_scaled)
14 print('₩nfeature 들의 평균 값')
15 print(iris_df_scaled.mean())
16 print('₩nfeature 들의 분산 값')
17 print(iris_df_scaled.var())
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0            -0.900681          1.019004          -1.340227         -1.315444
1            -1.143017         -0.131979          -1.340227         -1.315444
2            -1.385353          0.328414          -1.397064         -1.315444
3            -1.506521          0.098217          -1.283389         -1.315444
4            -1.021849          1.249201          -1.340227         -1.315444
..                 ...               ...                ...               ...
145           1.038005         -0.131979           0.819596          1.448832
146           0.553333         -1.282963           0.705921          0.922303
147           0.795669         -0.131979           0.819596          1.053935
148           0.432165          0.788808           0.933271          1.448832
149           0.068662         -0.131979           0.762758          0.790671

[150 rows x 4 columns]
```

```
feature 들의 평균 값
sepal length (cm)   -1.690315e-15
sepal width (cm)    -1.842970e-15
petal length (cm)   -1.698641e-15
petal width (cm)    -1.409243e-15
dtype: float64

feature 들의 분산 값
sepal length (cm)    1.006711
sepal width (cm)     1.006711
petal length (cm)    1.006711
petal width (cm)     1.006711
dtype: float64
```

데이터 전처리

■ 실습 문제) 피처 스케일링(Feature Scaling)과 정규화

정규화 - MinMaxScaler

```
1  from sklearn.preprocessing import MinMaxScaler
2
3  # MinMaxScaler객체 생성
4  scaler = MinMaxScaler()
5
6  # MinMaxScaler 로 데이터 셋 변환. fit() 과 transform() 호출.
7  scaler.fit(iris_df)
8  iris_scaled = scaler.transform(iris_df)
9
10 # transform()시 scale 변환된 데이터 셋이 numpy ndarry로 반환되어 이를 DataFrame으로 변환
11 iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
12
13 print(iris_df_scaled)
14 print('₩nfeature들의 최소 값')
15 print(iris_df_scaled.min())
16 print('₩nfeature들의 최대 값')
17 print(iris_df_scaled.max())
```

```
     sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0             0.222222          0.625000           0.067797          0.041667
1             0.166667          0.416667           0.067797          0.041667
2             0.111111          0.500000           0.050847          0.041667
3             0.083333          0.458333           0.084746          0.041667
4             0.194444          0.666667           0.067797          0.041667
..                 ...               ...                ...               ...
145           0.666667          0.416667           0.711864          0.916667
146           0.555556          0.208333           0.677966          0.750000
147           0.611111          0.416667           0.711864          0.791667
148           0.527778          0.583333           0.745763          0.916667
149           0.444444          0.416667           0.694915          0.708333

[150 rows x 4 columns]
feature들의 최소 값
sepal length (cm)    0.0
sepal width (cm)     0.0
petal length (cm)    0.0
petal width (cm)     0.0
dtype: float64

feature들의 최대 값
sepal length (cm)    1.0
sepal width (cm)     1.0
petal length (cm)    1.0
petal width (cm)     1.0
dtype: float64
```

데이터 전처리

- 데이터 인코딩(레이블 인코딩, 원-핫 인코딩)
  - 1) 레이블 인코딩(Label Encoding)
    - 데이타 인코딩의 한 방법, 카테고리 피처를 코드형의 숫자 값을 변환하는 것
      (예) 'France': 0, 'Germany': 1, 'Spain':2
    - 사이킷런의 LabelEncoder 클래스로 구현
      - LabelEncoder 객체로 생성한 후 fit()과 transform()을 호출해 레이블 인코딩 수행

```
1 x
```

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtyp
```

```
1 from sklearn.preprocessing import LabelEncoder
2
3 le = LabelEncoder()
4 le.fit(x[:, 0])
5 new_x = le.transform(x[:, 0])
6 new_x
```

```
array([0, 2, 1, 2, 1, 0, 2, 0, 1, 0])
```

```
1 x[:, 0] = new_x
2 pd.DataFrame(x, columns=['Country', 'Age', 'Salary'])
```

	Country	Age	Salary
0	0	44	72000
1	2	27	48000
2	1	30	54000
3	2	38	61000
4	1	40	NaN
5	0	35	58000
6	2	NaN	52000
7	0	48	79000
8	1	50	83000
9	0	37	67000

'France': 0, 'Germany': 1, 'Spain':2

- 레이블 인코딩의 문제점
  - 문자열을 숫자형 카테고리 값으로 변경은 하지만, 숫자 값인 경우 크고 작음에 대한 특성이 작용하기 때문에
  - 특정 ML 알고리즘에서 가중치가 더 부여되거나 중요하게 인식할 가능성이 있다.
  - 선형회귀와 같은 알고리즘에는 적용하면 안됨, 트리계열 ML 알고리즘은 숫자의 특성을 반영하지 않으므로 사용해도 무방

color	color_red	color_blue	color_green
red	1	0	0
green	0	0	1
blue	0	1	0

데이터 전처리

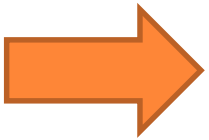
- 데이터 인코딩(레이블 인코딩, 원-핫 인코딩)
- 2) 원-핫 인코딩(One-Hot encoding)
  - 카테고리형 데이터에 적용, 피처 값의 유형에 따라 새로운 피처를 추가해 고유 값에 해당하는 컬럼에만 1을 표시하고, 나머지 컬럼에는 0을 표시하는 방식
  - 행 형태로 되어 있는 고유값을 열 형태로 차원 변환한 뒤, 고유값에 해당하는 컬럼에만 1을 표기

- 원-핫 인코딩 구현 방법
- (1) 사이킷런의 OneHotEncoder 클래스로 구현
  - 카테고리 피처 → 레이블 인코딩 → 원-핫 인코딩
- (2) 판다스의 get_demmmies() 메서드를 이용한 원-핫 인코딩
  - 숫자형 값으로 변환 없이도 바로 원-핫 인코딩 가능

데이터 전처리

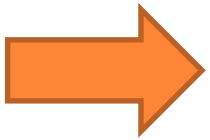
- 데이터 인코딩(레이블 인코딩, 원-핫 인코딩)
- 2) 원-핫 인코딩(One-Hot encoding)

	Country
0	France
1	Spain
2	Germany
3	Spain
4	Germany
5	France
6	Spain
7	France
8	Germany
9	France



레이블 인코딩

	Country
0	0
1	2
2	1
3	2
4	1
5	0
6	2
7	0
8	1
9	0



원-핫 인코딩

	French	Germany	Spain
0	1.0	0.0	0.0
1	0.0	0.0	1.0
2	0.0	1.0	0.0
3	0.0	0.0	1.0
4	0.0	1.0	0.0
5	1.0	0.0	0.0
6	0.0	0.0	1.0
7	1.0	0.0	0.0
8	0.0	1.0	0.0
9	1.0	0.0	0.0

데이터 전처리

- 데이터 인코딩(레이블 인코딩, 원-핫 인코딩)
- 2) 원-핫 인코딩(One-Hot encoding)

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

# 1) 레이블 인코딩 진행
le = LabelEncoder()
le.fit(x[:, 0])
new_x = le.transform(x[:, 0])
# 2차원 데이터로 변환
new_x = new_x.reshape(-1,1)
new_x

# # 2) 원-핫 인코딩 진행
ohe = OneHotEncoder()
ohe.fit(new_x)
new_ohe = ohe.transform(new_x)
new_ohe.toarray()
```

```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.]])
```

```
pd.DataFrame(new_ohe.toarray(), columns = ['French', 'Germany', 'Spain'])
```

	French	Germany	Spain
0	1.0	0.0	0.0
1	0.0	0.0	1.0
2	0.0	1.0	0.0
3	0.0	0.0	1.0
4	0.0	1.0	0.0
5	1.0	0.0	0.0
6	0.0	0.0	1.0
7	1.0	0.0	0.0
8	0.0	1.0	0.0
9	1.0	0.0	0.0

```
new_ohe.toarray().shape
```

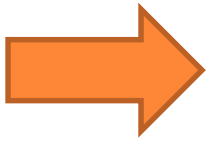
```
(10, 3)
```

데이터 전처리

- 데이터 인코딩(레이블 인코딩, 원-핫 인코딩)
- 3) 판다스 get_dummies()로 원-핫 인코딩 구현
  - 숫자형 값으로 변환 없이도 바로 원-핫 인코딩 가능

```
1 df1
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes



판다스 get_dummies()

```
1 import pandas as pd
2
3 pd.get_dummies(df1.iloc[:, 0])
```

	France	Germany	Spain
0	1	0	0
1	0	0	1
2	0	1	0
3	0	0	1
4	0	1	0
5	1	0	0
6	0	0	1
7	1	0	0
8	0	1	0
9	1	0	0

실습예제
데이터 전처리

실습 예제 - 데이터 전처리

- 공공데이타-건강검진 데이터로 전처리(Preprocessing)
- 건강검진 공공 데이터 읽어오기
  - http://data.go.kr → '건강검진' 키워드 입력

실습 예제 - 데이터 전처리

- 공공데이타-건강검진 데이터로 전처리(Preprocessing)
- 건강검진 공공 데이터 읽어오기
  - 건강검진정보(2017) csv 파일 → 다운로드 클릭

미리보기

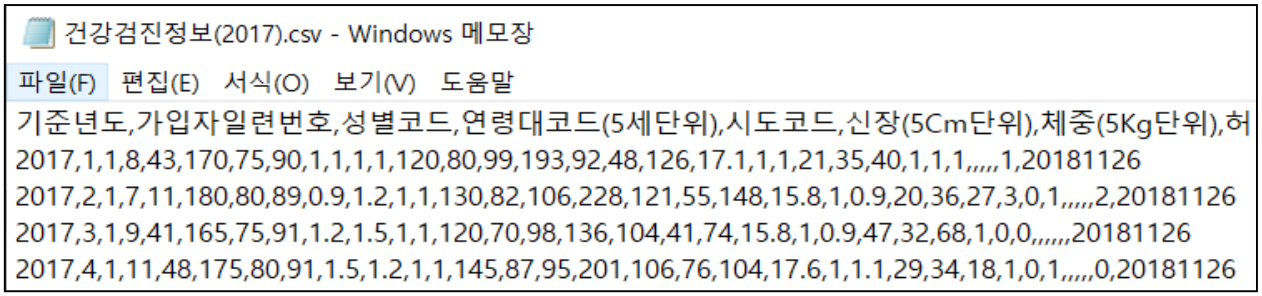
※ 파일 데이터의 일부 내용을 제공하고 있으며, 전체 내용이 필요한 경우 해당 파일을 다운로드 받으시기 바랍니다.

기준년도	가입자일련번호	성별코드	연령대코드(5세단위)	시도코드	신장(5Cm단위)	체중(5Kg단위)	허리둘레	시
2017	1	1	8	43	170	75	90	
2017	2	1	7	11	180	80	89	
2017	3	1	9	41	165	75	91	
2017	4	1	11	48	175	80	91	
2017	5	1	11	30	165	60	80	
2017	6	1	11	41	165	55	75	
2017	7	2	10	27	150	55	69	
2017	8	1	8	48	175	65	84.2	
2017	9	1	12	41	170	75	84	
2017	10	1	9	41	175	75	82	
2017	11	1	10	41	155	55	79.2	
2017	12	1	14	27	155	75	98	

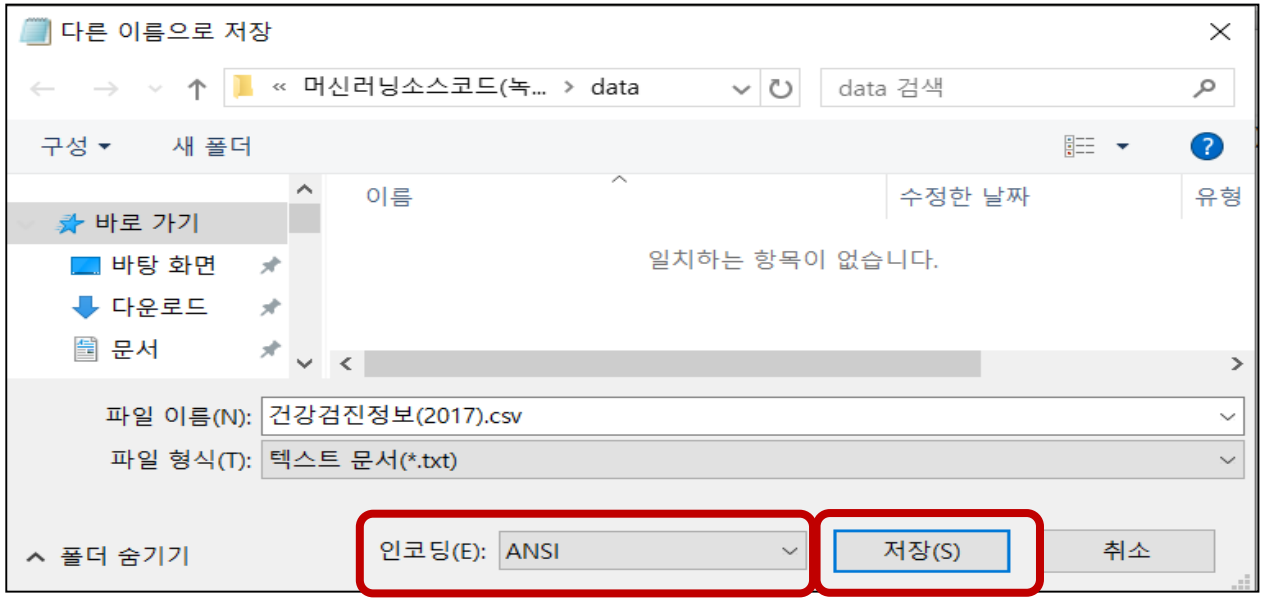
Sheet1

실습 예제 - 데이터 전처리

- 공공데이타-건강검진 데이터로 전처리(Preprocessing)
- 건강검진 공공 데이터 읽어오기
  - 건강검진정보(2017).csv 파일 → 인코딩을 'UTF-8'로 변경
    - notepad로 파일 열기



  - 파일 - 다른이름으로 저장하기 - 인코딩 'UTF-8' 로 바꾸어 저장



- **공공데이타-건강검진 데이터로 전처리(Preprocessing)**

공공 데이타 읽어오기

- 공공 데이타 : https://www.data.go.kr 건강검진 데이타 다운로드

```
df = pd.read_csv(r"../data/건강검진정보(2017).csv", encoding='utf-8')  ## raw 문자열로 지정한다
```

데이타 살펴보기

```
df.shape                                                    ## 데이터의 형상을 확인한다
```

```
(1000000, 34)
```

```
df.info()                                                   ## 데이터프레임의 열의 자료형을 확인한다
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 34 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   기준년도            1000000 non-null  int64
 1   가입자일련번호         1000000 non-null  int64
 2   성별코드            1000000 non-null  int64
 3   연령대코드(5세단위)     1000000 non-null  int64
 4   시도코드            1000000 non-null  int64
 5   신장(5Cm단위)       1000000 non-null  int64
 6   체중(5Kg단위)       1000000 non-null  int64
```

- 공공데이타-건강검진 데이터로 전처리(Preprocessing)

1. 결측치 제거

1) 결측치 확인 - isna(), isnull()

```
df_1 = df.isna()                    ## 데이터 프레임의 결측값을 확인한다.
```

```
df_1.sum()                          ## 컬럼별 결측값을 sum 메소드로 확인한다
```

```
. . .
```

```
df_1.sum().sum()                    ## 총 결측값을 sum 메소드로 확인한다
```

```
4609044
```

2) 결측치 제거 및 결측치 대체 - fillna(), dropna()

```
df = df.fillna(0)                   ## 모든 결측값을 0으로 처리한다
```

3) 결측치 대체 후 확인

```
df_2 = df.isna()
```

```
df_2.sum().sum()
```

```
0
```

- 공공데이타-건강검진 데이터로 전처리(Preprocessing)

2. 피처 엔지니어링 - 피처 선택 및 새로운 피처 생성

1) 기존 컬럼을 이용해 새로운 데이타프레임 생성

```
1 df_new = df [['성별코드','신장(5Cm단위)']].copy()
```

```
1 df_new.shape
```

```
(1000000, 2)
```

2) 새로운 피처(컬럼) 추가 - count

```
1 df_new.loc[:,"count"] = 1        ##  count  열을 추가한다
```

```
1 df_new.head()
```

	성별코드	신장(5Cm단위)	count
0	1	170	1
1	1	180	1
2	1	165	1
3	1	175	1
4	1	165	1

- **공공데이타-건강검진 데이터로 전처리(Preprocessing)**

3) 피봇 테이블 만들기

```
1  import numpy as np
```

```
1  df_pivot = pd.pivot_table(df_new,  index=['신장(5Cm단위)'],        ## 피봇 테이블을 만든다. 인덱스는 신장
2                            columns=['성별코드'],                   ## 열은 성별코드
3                            values="count",                         ## 들어갈 데이터는 count
4                            aggfunc= np.sum)                        ## 계산는 합으로 처리
```

```
1  df_pivot.head()
```

성별코드	1	2
신장(5Cm단위)		
130	NaN	93.0
135	NaN	1301.0
140	6.0	9273.0
145	99.0	39507.0
150	2986.0	105849.0

- 공공데이타-건강검진 데이터로 전처리(Preprocessing)

4) 결측치 대체

```
df_pivot = df_pivot.fillna(0)                    ## 결측치를 0으로 처리한다
```

```
df_pivot.head()
```

성별코드	1	2
신장(5Cm단위)		
130	0.0	93.0
135	0.0	1301.0
140	6.0	9273.0
145	99.0	39507.0
150	2986.0	105849.0

3. 기술 통계량 확인

```
df_pivot.describe()                              ## 기술 통계량을 확인한다
```

성별코드	1	2
count	13.000000	13.000000
mean	40800.769231	36122.307692
std	56802.893131	52056.939757
min	0.000000	0.000000
25%	99.000000	93.000000
50%	6630.000000	8432.000000
75%	70336.000000	43597.000000
max	159110.000000	147717.000000

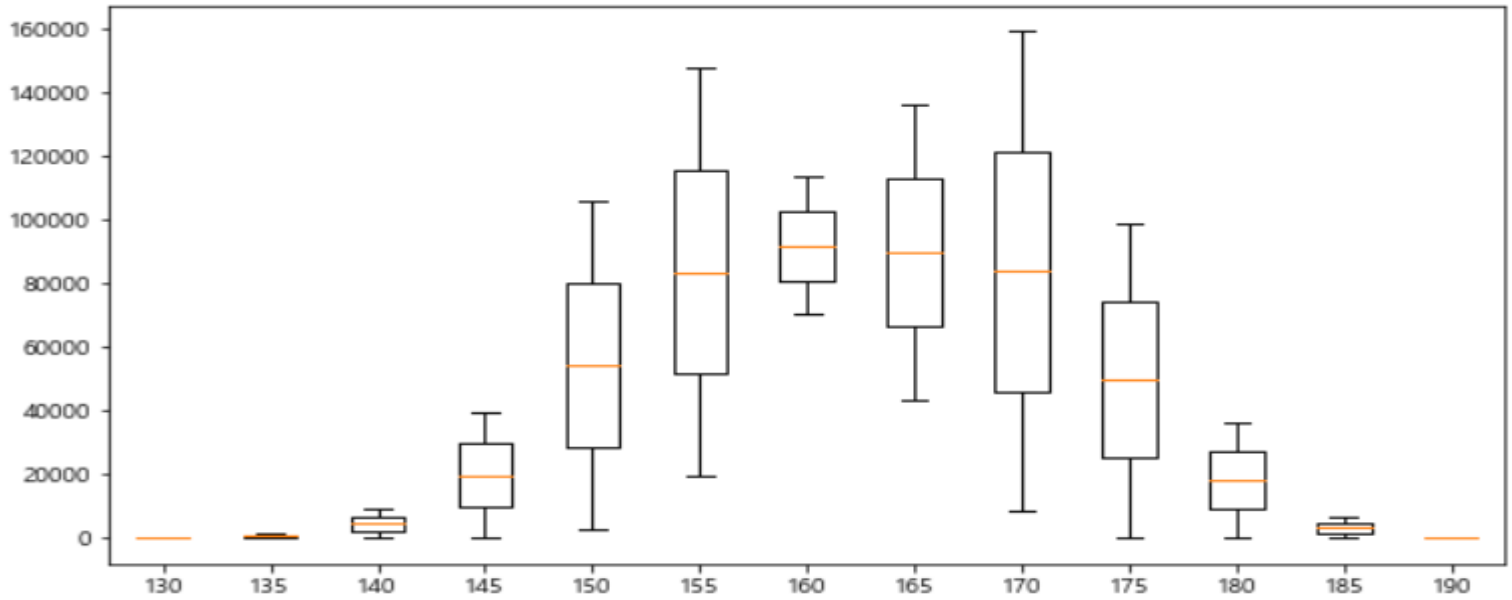
- 공공데이타-건강검진 데이터로 전처리(Preprocessing)

4. 그래프 시각화

```
1  import matplotlib.pyplot as plt
2  %matplotlib inline
```

2) Box 그래프 내의 xtick 변경하기

```
1  plt.figure(figsize=(10,5))
2  plt.boxplot(df_pivot)
3  plt.xticks([x+1 for x in range(0, df_pivot.index.shape[0])], [x for x in df_pivot.index])
4  ## 신장에 대한 정보로 x축이 레이블을 변경한다
5  plt.show()
```



- 공공데이타-건강검진 데이터로 전처리(Preprocessing)

3) 성별 키 분포 확인

```
1  plt.figure(figsize=(10,5))
2  plt.bar(df_pivot.index, df_pivot[1], color='r', label='male')
3  ## 남성과 여성으로 구분해서 막대 그래프를 그린다
4  plt.bar(df_pivot.index + 0.9, df_pivot[2], color='g', label='female')
5  plt.legend()
```

```
<matplotlib.legend.Legend at 0x1e1c1b23248>
```