데이터 전처리2

데이터 변환

- **단위 환산**
  - 같은 데이터세트 안에서 서로 다른 측정 단위를 사용한다면, 전체 데이터의 일관성 측면에서 문제가 발생

- **자료형 변환**
  - 예를 들어, 숫자가 문자열로 저장된 경우에 숫자형 (int 또는 float)으로 변환 필요
  - 문자열.astype('float')로 문자열을 실수형으로 변환

실습1.1 단위환산

- 자동차 연비 데이터세트로 실습
  - 다운로드 사이트: http://archive.ics.uci.edu/ml/datasets/auto+mpg
  - mpg변수는 영미권 단위인 '갤런당 마일(mile per gallon)'을 사용
  - 한국에서 익숙한 '리터당 킬로미터(km/l)로 변환하기

```
df = pd.read_csv('./auto-mpg.csv', header=None)
```

```
df.columns = ['mpg','cylinders','displacement','horsepower','weight','acceleration','model year','origin','name']
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

실습1.2 단위환산

- 연비 단위환산
  - 1마일: 1.609km, 1갤런: 3.785l -> 1mpg = 0.425km/l
  - round(2) 소수점 아래 둘째 자리 반올림

```
mpg2kpl = 1.60934/3.78541
mpg2kpl
```

```
0.42514285110463595
```

```
df['kpl'] = df['mpg']*mpg2kpl
df['kpl'] = df['kpl'].round(2)
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	name	kpl
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu	7.65
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320	6.38
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite	7.65
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst	6.80
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino	7.23

실습2.1 자료형 변환

- 데이터타입 확인
  - 자동차 연비 데이터세트 http://archive.ics.uci.edu/ml/datasets/auto+mpg
  - 데이터 읽는 부분은
    1.1예제 참조
  - horsepower는 마력으로
    연산을 위해서 숫자형 이어야
    하지만 문자열로 되어 있음
  - cf. df.dtypes속성도 데이터
    프레임의 각 열의 자료형 확인

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    int64
 8   name          398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

실습2.2 자료형 변환

- **horsepower가 문자열이었던 이유?**
  - unique( )메소드 : 고유값만 반환
  - '?'가 오염된 데이터로 있어 csv파일을 데이터프레임으로 변환하는 과정에서 문자열로 인식된 것으로 보임

```
print(df['horsepower'].unique())
```

```
['130.0' '165.0' '150.0' '140.0' '198.0' '220.0' '215.0' '225.0' '190.0'
 '170.0' '160.0' '95.00' '97.00' '85.00' '88.00' '46.00' '87.00' '90.00'
 '113.0' '200.0' '210.0' '193.0' '?' '100.0' '105.0' '175.0' '153.0'
 '180.0' '110.0' '72.00' '86.00' '70.00' '76.00' '65.00' '69.00' '60.00'
 '80.00' '54.00' '208.0' '155.0' '112.0' '92.00' '145.0' '137.0' '158.0'
 '167.0' '94.00' '107.0' '230.0' '49.00' '75.00' '91.00' '122.0' '67.00'
 '83.00' '78.00' '52.00' '61.00' '93.00' '148.0' '129.0' '96.00' '71.00'
 '98.00' '115.0' '53.00' '81.00' '79.00' '120.0' '152.0' '102.0' '108.0'
 '68.00' '58.00' '149.0' '89.00' '63.00' '48.00' '66.00' '139.0' '103.0'
 '125.0' '133.0' '138.0' '135.0' '142.0' '77.00' '62.00' '132.0' '84.00'
 '64.00' '74.00' '116.0' '82.00']
```

실습2.3 자료형 변환

- **값 치환**
  - '?'를 모두 제거하기 위해 NaN 으로 변환
  - replace( ) 메소드 사용
  - horsepower가 누락데이터가 6개 있음! ('?' 였던 곳)

```
import numpy as np
df['horsepower'].replace('?',np.nan,inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    object
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    int64
 8   name          398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

실습2.4 자료형 변환

- 누락데이터 제거
  - dropna( ) 메소드 사용
  - 행의 개수 392개 확인

```
#누락행 삭제
df.dropna(subset=['horsepower'],inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    object
 4   weight        392 non-null    float64
 5   acceleration  392 non-null    float64
 6   model year    392 non-null    int64
 7   origin        392 non-null    int64
 8   name          392 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 30.6+ KB
```

실습2.5 자료형 변환

- 데이터타입 변환
  - .astype( ) 메소드로 자료형 변환
  - 문자열.astype('float')로 문자열을 실수형으로 변환
  - horsepower의 자료형 float64로 변경 확인

```
df['horsepower'] = df['horsepower'].astype('float')
print(df['horsepower'].dtypes)
```

```
float64
```

범주형 데이터

- **범주형 데이터** Categorical Data
  - 몇 개의 범주 또는 항목의 형태로 나누어진 데이터
  - 예) 성별(남/여), 혈액형(A, B, O, AB), 평점(5점, 3점, 1점)

- **대표적 데이터 분류법 (예시)**
  - 범주형 데이터 / 수치형 데이터
  - 수치형 데이터는 숫자 형태로 측정되는 데이터
  - 수치형 데이터는 값의 연속성에 따라 이산형과 연속형으로 구분
  - 이산형 데이터 vs. 범주형 데이터의 경계는?

범주형 데이터 관련

- **수치형 데이터를 범주형 데이터로 변환**
  - 머신러닝알고리즘 등에서 연속된 데이터로 훈련이 어려운 경우에 필요
  - 구간 분할(Binning) 전처리 기법

- **범주형 데이터를 수치형 데이터로 변환 (이산형)**
  - 카테고리를 나타내는 범주형 데이터를 머신러닝알고리즘 등에 바로 적용할 수 없는 경우에 필요
  - Label Encoding
  - One-Hot Encoding

구간분할

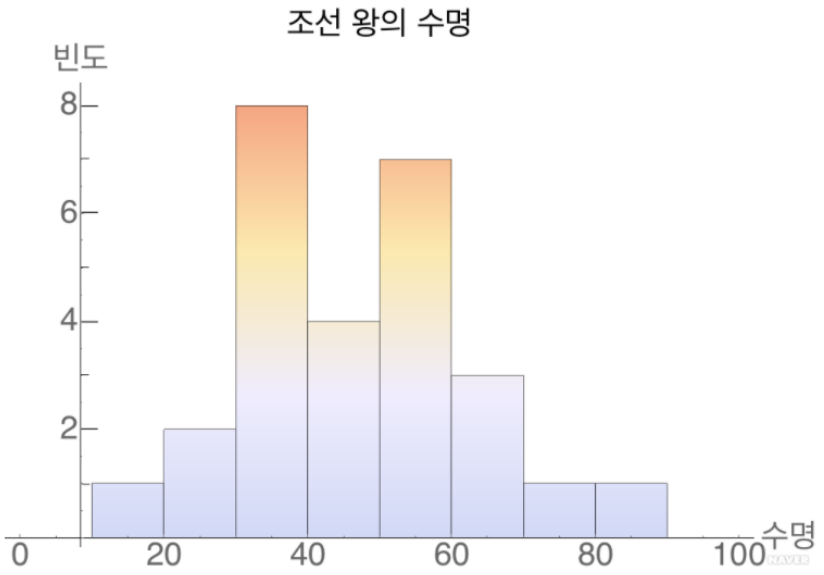
- **구간 분할**binning
  - 변수 구간화라고도 하며, 연속 변수를 일정한 구간으로 나누고 각 구간을 범주형 이산 변수로 변환하는 과정
  - 예를 들면, 소득을 소득분위로, 나이를 연령층으로 나누는 것 등
  - cf. histogram

- **필요성**
  - 데이터 분석 알고리즘에 따라서는 연속 데이터를 그대로 사용하기 보다는 일정한 구간(bin)으로 나누어서 분석하는 것이 효율적인 경우가 있음
  - 이상치 문제 완화 등에도 필요

히스토그램

- 히스토그램
  - 측정값이 존재하는 범위를 몇 개의 구간(계급)으로 나누는 경우, 각 구간을 밑변으로 하고 그 구간에 속하는 측정값의 출현 도수에 비례하는 면적을 갖는 기둥(직사각형)으로 배열한 그림
  - 도수/빈도frequency

    예) 구간별 왕의 수
  - 30~39세 사이의 수명을 가진 조선 왕은 몇 명?



[Naver]

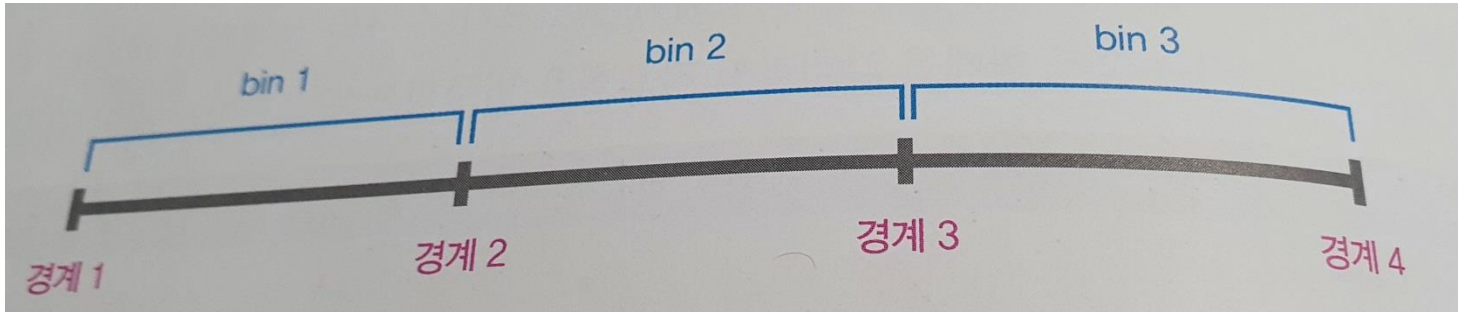
구간 분할 예제

- 마력horsepower 변수를 구간 분할해 보자
  - ‘horsepower’열은 엔진 출력을 나타냄
  - 엔진 출력을 숫자로 표시하는 대신 ‘저출력’, ‘보통출력’, ‘고출력’ 등 구간으로 나누어 표시하는 것이 효율적일 수 있음

- 실습
  - ‘horsepower’열을 숫자값으로 변환한 실습2.5에 이어서 코딩

구간 분할 경계값 구하기

- Numpy라이브러리의 histogram( ) 메소드 활용 방법
  - count, bin_dividers = np.histogram(df[‘horsepower’], bins=3)
  - 나누려는 구간(bin) 개수를 bins옵션에 입력하면 각 구간에 속하는 값의 개수(count)와 경계값 리스트(bin_divederes)를 반환
  - 모두 4개의 경계값을 생성하고 3개의 구간이 만들어짐



실습3.1 경계값 구하기

- Numpy의 메소드 np.histogram( )
  - 실습2.5에 이어서

```
count, bin_dividers = np.histogram(df['horsepower'],bins=3)
print(count, bin_dividers)
```

```
[257 103  32] [ 46.         107.33333333 168.66666667 230.        ]
```

  - 최소값: 46, 최대값: 230
  - 최대값과 최소값의 차를 구간 개수(bins)로 나누고, 최소값에 그 간격을 더하여 다음 경계값을 생성
  - 예) bins=3인 경우, (230-46)/3 = 61.333 (등 간격)

    최소값 다음 경계: 46+61.333 = 107.333
  - 각 구간의 도수/빈도는 257, 103, 32

구간 분할하기

- Pandas의 cut() 메소드 활용 방법
  - histogram()등을 통해 경계값이 구해졌으면, 경계값의 리스트(bin_dividers)를 bins옵션에 할당
  - 각 구간의 이름 리스트(bin_names)를 labels옵션에 할당
  - inclule_lowest=True옵션을 설정하면 각 구간의 낮은 경계값을 포함

실습3.2 구간 분할하기

```
bin_names = ['저출력','보통출력','고출력']
df['hp_bin'] = pd.cut(df['horsepower'], bins=bin_dividers, labels=bin_names, include_lowest=True)
df[['horsepower','hp_bin']].head(15)
```

	horsepower	hp_bin
0	130.0	보통출력
1	165.0	보통출력
2	150.0	보통출력
3	150.0	보통출력
4	140.0	보통출력
5	198.0	고출력
6	220.0	고출력
7	215.0	고출력
8	225.0	고출력
9	190.0	고출력
10	170.0	고출력
11	160.0	보통출력
12	150.0	보통출력
13	225.0	고출력
14	95.0	저출력

범주형 데이터 처리

- **필요성**
  - 카테고리를 나타내는 범주형 데이터를 회귀분석 등 머신러닝 알고리즘에 바로 적용할 수 없는 경우에 필요
  - 즉, 컴퓨터가 인식 가능한 입력값으로 변환해야 하는 경우

- **범주형 데이터 삭제법?**
  - 해당 열을 삭제하는 가장 손쉬운 방법
  - 단, 해당 열에 유용한 정보가 없는 경우에 사용

- **범주형 데이터를 수치형 데이터로 변환 (이산형)**
  - Label Encoding
  - One-Hot Encoding

Label Encoding

- **Label Encoding**
  - 각각의 범주에(unigue category) 대해 다른 정수로 변환

Breakfast
Every day
Never
Rarely
Most days
Never

Breakfast
3
0
1
2
0

[Kaggle]

- 주의
  - 범주에 따라 수치값의 크기가 달라 훈련시 가중치 부작용 가능
  - 범주형 데이터중에서도 순서형 데이터에 적용

순서형 데이터

- **순서형 데이터**ordinal data
  - 범주형 데이터중에서 난이도나 평점과 같이 각각의 범주에 순서가 주어질 때
  - 주어진 예제와 같이 "Never" (0) < "Rarely" (1) < "Most days" (2) < "Every day" (3) 순으로 범주에 순서가 있는 경우
  - 가중치 효과로 인한 부작용 적음

- **적용사례**
  - Decision Trees나 Random Forests와 같은 Tree-Based Models의 경우, 순서형 데이터에 대해 Label Encoding 적용

실습4.1 Label Encoding

- Algorithm 구현
  - 실습3.2에 이어서
  - 저출력부터 0,1,2 할당하는 함수 구현
  - hp_bin열의 Label을 0,1,2로 변환

```
#algorithm
def label_encoding(data):
    if data=='저출력':
        return 0
    elif data=="보통출력":
        return 1
    else:
        return 2
```

```
df['hp_le']=df['hp_bin'].apply(label_encoding)
df[['horsepower','hp_bin', 'hp_le']].head(15)
```

	horsepower	hp_bin	hp_le
0	130.0	보통출력	1
1	165.0	보통출력	1
2	150.0	보통출력	1
3	150.0	보통출력	1
4	140.0	보통출력	1
5	198.0	고출력	2
6	220.0	고출력	2
7	215.0	고출력	2
8	225.0	고출력	2
9	190.0	고출력	2
10	170.0	고출력	2
11	160.0	보통출력	1
12	150.0	보통출력	1
13	225.0	고출력	2
14	95.0	저출력	0

실습4.2 LabelEncoder()

- Scikit-learn preprocessing
  - 실습3.2에 이어서
  - LabelEncoder()를 import해서
  - fit_transform()으로 label encoding
  - 단, 주어진 String을 알파벳순으로 0,1,2,… 할당

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

```
le=LabelEncoder()
df['hp_le']=le.fit_transform(df['hp_bin'])
df[['horsepower','hp_bin','hp_le']].head(15)
```

	horsepower	hp_bin	hp_le
0	130.0	보통출력	1
1	165.0	보통출력	1
2	150.0	보통출력	1
3	150.0	보통출력	1
4	140.0	보통출력	1
5	198.0	고출력	0
6	220.0	고출력	0
7	215.0	고출력	0
8	225.0	고출력	0
9	190.0	고출력	0
10	170.0	고출력	0
11	160.0	보통출력	1
12	150.0	보통출력	1
13	225.0	고출력	0
14	95.0	저출력	2

명목형 데이터

- **명목형 데이터**nominal data
  - 범주형 데이터중에서 성별이나 혈액형과 같이 각각의 범주에 아무런 순서가 없고 단순히 분류를 목적으로 하는 데이터

- **명목형 데이터를 Label Encoding하는 경우의 문제점**
  - 형액형에 대해 “A형" (0) < “B형" (1) < “O형" (2) < “AB형" (3) 순으로 Encoding하는 경우, AB형에 가중치 효과를 주어 잘못된 훈련이 될 수 있음

더미변수

- 더미 변수dummy variable
  - 더미 변수란 변수를 숫자 0 또는 1로만 표현하는 것
  - 여기서 0과 1은 수의 크고 작음을 나타내지 않고, 어떤 특성feature이 있는지 없는지 여부만을 표시
  - 해당 특성이 존재하면 1로 표현하고, 존재하지 않으면 0으로 구분

- 명목형데이터의 수치형데이터 변환에 적용
  - 명목형데이터를 더미변수로 변환하면 **Label Encoding**으로 인한 부작용 해소 가능

One-Hot Encoding

- **One-Hot Encoding**
  - 각각의 범주에(unigue category) 대해 별도의 열을 만들고 더미변수(Dummy Variable)로 변환

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

[Kaggle]

- **목적**
  - Label Encoding에 의한 가중치 부작용 문제 해소
  - 범주형 데이터중에서도 명목형 데이터에 적용

One-Hot Encoding시 주의

- 주의
  - One-Hot Encoding은 하나의 범주형 변수에 대해 각각의 범주가 너무 많을 경우, 성능이 저하됨
  - 즉, 이전 예와 같이 범주형 변수 Color에 대해 각각의 범주가 Red, Yellow, Green의 세가지 정도일 때는 문제가 없지만, 색의 개수가 50개가 되면 큰 성능 저하로 One-Hot Encoding 적용이 어려움
  - 통상의 기준은 15개 이상인 경우 적용이 어려운 것으로 판단

실습5.1 One-Hot Encoding

- 자동차 연비 데이터세트로 실습
  - 다운로드 사이트: http://archive.ics.uci.edu/ml/datasets/auto+mpg
  - 실습대상 : 명목형 데이터인 origin 변수
  - origin은 차량의 원산지를 의미

```
df = pd.read_csv('./auto-mpg.csv', header=None)
```

```
df.columns = ['mpg','cylinders','displacement','horsepower','weight','acceleration','model year','origin','name']
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

실습5.2 명목형 데이터

- 명목형 데이터 origin 변수 변환
  - origin 변수의 unique값은 1, 2, 3이고, 데이터타입은 int64

```
print(df['origin'].unique())
print(df['origin'].dtypes)
```

```
[1 3 2]
int64
```

  - origin 변수를 국가명으로 변환

```
df['origin'].replace({1:'USA',2:'EU',3:'JPN'},inplace=True)
print(df['origin'].unique())
print(df['origin'].dtypes)
```

```
['USA' 'JPN' 'EU']
object
```

```
df[['origin','name']].tail(15)
```

	origin	name
383	JPN	honda civic
384	JPN	honda civic (auto)
385	JPN	datsun 310 gx
386	USA	buick century limited
387	USA	oldsmobile cutlass ciera (diesel)
388	USA	chrysler lebaron medallion
389	USA	ford granada l
390	JPN	toyota celica gt
391	USA	dodge charger 2.2
392	USA	chevrolet camaro
393	USA	ford mustang gl
394	EU	vw pickup
395	USA	dodge rampage
396	USA	ford ranger
397	USA	chevy s-10

실습5.3 더미 변수 구하기

- Pandas의 get_dummies() 메소드
  - 범주형 변수의 모든 고유값을 각각 새로운 더미 변수로 변환
  - 예제의 범주형 변수 'origin'열의 고유값 3개가 각각 새로운 더미 변수 열의 이름이 됨
  - 각 더미 변수가 본래 속해 있던 행에는 1이 설정되고, 속하지 않았던 다른 행에는 0이 설정

```
origin_dummies=pd.get_dummies(df['origin'])
origin_dummies.tail(15)
```

	EU	JPN	USA
383	0	1	0
384	0	1	0
385	0	1	0
386	0	0	1
387	0	0	1
388	0	0	1
389	0	0	1
390	0	1	0
391	0	0	1
392	0	0	1
393	0	0	1
394	1	0	0
395	0	0	1
396	0	0	1
397	0	0	1

참고도서

➢ reference

[1] 파이썬을 활용한 데이터길들이기

- 프로그래밍인사이트

[2] 모두의 데이터분석

- 길벗

[3] 파이썬머신러닝 판다스데이터분석

- 정보문화사

End