7.5.2 판다스의 Categorical 확장형

파이썬 시퀀스에서 pandas.Categorical 직접 생성

```
my_categories = pd.Categorical(['foo', 'bar', 'baz', 'foo', 'bar'])
my_categories
```

```
['foo', 'bar', 'baz', 'foo', 'bar']
Categories (3, object): ['bar', 'baz', 'foo']
```

```
categories = ['foo', 'bar', 'baz']
codes = [0, 1, 2, 0, 0, 1]
my_cats_2 = pd.Categorical.from_codes(codes, categories)
my_cats_2
```

//범주와 범주 코드 미리 정의

```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo', 'bar', 'baz']
```

순서 지정

```
ordered_cat = pd.Categorical.from_codes(codes, categories, ordered=True)
ordered_cat
```

```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

```
my_cats_2.as_ordered()
```

```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

foo, bar, baz 순서를 가짐

기존에 정의된 범주와 범주 코드가 있다면 from codes로 범주형 데이터를 생성할 수 있습니다.

From codes의 ordered 속성을 True값으로 설정하거나 범주형 인스턴스이기 때문에 as ordered 메서드를 사용해 정렬할 수도 있습니다.

In [103]:
```
categories = ['foo', 'bar', 'baz']
codes = [0, 1, 2, 0, 0, 1]
my_cats_2 = pd.Categorical.from_codes(codes, categories)
my_cats_2
```

Out[103]:
```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo', 'bar', 'baz']
```

In [104]:
```
ordered_cat = pd.Categorical.from_codes(codes, categories,
                                        ordered=True)
ordered_cat
```

Out[104]:
```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

In [105]:
```
my_cats_2.as_ordered()
```

Out[105]:
```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

In [106]:
```
rng = np.random.default_rng(seed=12345)
draws = rng.standard_normal(1000)
draws[:5]
```

Out[106]:
```
array([-1.4238,  1.2637, -0.8707, -0.2592, -0.0753])
```

In [107]:
```
bins = pd.qcut(draws, 4)
bins
```

Out[107]:
```
[(-3.121, -0.675], (0.687, 3.211], (-3.121, -0.675], (-0.675, 0.0134], (-0.675, 0.0134], ..., (0.0134, 0.687], (0.0134, 0.687], (-0.675, 0.0134], (0.0134, 0.687], (-0.675, 0.0134]]
Length: 1000
Categories (4, interval[float64, right]): [(-3.121, -0.675] < (-0.675, 0.0134] < (0.0134, 0.687] < (0.687, 3.211]]
```

In [108]:
```
bins = pd.qcut(draws, 4, labels=['Q1', 'Q2', 'Q3', 'Q4'])
bins
bins.codes[:10]
```

Out[108]:
```
array([0, 3, 0, 1, 1, 0, 0, 2, 2, 0], dtype=int8)
```

In [109]:
```
bins = pd.Series(bins, name='quartile')
results = (pd.Series(draws)
           .groupby(bins)
           .agg(['count', 'min', 'max'])
           .reset_index())
```

```
columns={"three": "peekaboo"}) #축 이름 중 일부만 변경
```

Out [186]:

	one	two	peekaboo	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

7.2.5 이산화

In [187]:
```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

In [188]:
```
bins = [18, 25, 35, 60, 100]
age_categories = pd.cut(ages, bins)
```

In [190]:
```
age_categories #범주형 객체
```

Out [190]:
```
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35, 60], (25, 35]]
Length: 12
Categories (4, interval[int64, right]): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
```

In [285]:
```
age_categories.codes
```

In [286]:
```
age_categories.categories?
```

In [193]:
```
age_categories.categories[0]
```

Out [193]:
```
Interval(18, 25, closed='right')
```

In [194]:
```
pd.value_counts(age_categories) #각 그룹의 개수
```

Out [194]:
```
(18, 25]     5
(25, 35]     3
(35, 60]     3
(60, 100]    1
dtype: int64
```

In [196]:
```
pd.cut(ages, bins, right=False) #상한, 하한값 포함 여부
```

Out [196]:
```
[[18, 25), [18, 25), [25, 35), [25, 35), [18, 25), ..., [25, 35), [60, 100), [35, 60), [35, 60), [25, 35)]
Length: 12
Categories (4, interval[int64, left]): [[18, 25) < [25, 35) < [35, 60) < [60, 100)]
```

In [270]:
```
import re
text = "foo    bar₩t baz  ₩tqux"
re.split(r"₩s+", text) #정규 표현식이 컴파일되고 split메서드 실행
```

Out[270]:
```
['foo', 'bar', 'baz', 'qux']
```

In [271]:
```
regex = re.compile(r"₩s+")
regex.split(text)
```

Out[271]:
```
['foo', 'bar', 'baz', 'qux']
```

In [272]:
```
regex.findall(text) #정규 표현식에 매칭되는 패턴 목록
```

Out[272]:
```
['    ', '₩t ', '  ₩t']
```

In [77]:
```
text = """Dave dave@google.com
Steve steve@gmail.com
Rob rob@gmail.com
Ryan ryan@yahoo.com"""
pattern = r"[A-ZO-9._%+-]+@[A-ZO-9.-]+₩.[A-Z]{2,4}"

# re.IGNORECASE makes the regex case insensitive
regex = re.compile(pattern, flags=re.IGNORECASE)
```

In [78]:
```
regex.findall(text)
```

Out[78]:
```
['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

In [79]:
```
m = regex.search(text)
m
text[m.start():m.end()]
```

Out[79]:
```
'dave@google.com'
```

In [80]:
```
print(regex.match(text))
```

```
None
```

In [81]:
```
print(regex.sub("REDACTED", text))
```

```
Dave REDACTED
Steve REDACTED
Rob REDACTED
Ryan REDACTED
```