# ADVANCED PANDAS

In [91]:

```
import numpy as np
import pandas as pd
```

In [3]:

```
# Background and Motivation


# Frequently, a column in a table may contain repeated instances of a smaller set of disti
nct values.
    # We have already seen functions like unique and value_counts,
    # which enable us to extract the distinct values from an array and compute their frequ
encies, respectively:

import numpy as np; import pandas as pd


values = pd.Series(['apple', 'orange', 'apple','apple'] * 2)

values
```

Out[3]:

```
0     apple
1    orange
2     apple
3     apple
4     apple
5    orange
6     apple
7     apple
dtype: object
```

In [4]:

```
pd.unique(values)
```

Out[4]:

```
array(['apple', 'orange'], dtype=object)
```

In [5]:

```
pd.value_counts(values)
```

Out[5]:

```
apple     6
orange    2
dtype: int64
```

In [7]:

```
# Many data systems (for data warehousing, statistical computing, or other uses) have deve
loped specialized
    # approaches for representing data with repeated values for more efficient storage and
computation.
    # In data warehousing, a best practice is to use socalled dimension tables containing
 the distinct values
    # and storing the primary observations as integer keys referencing the dimension tabl
e:

values = pd.Series([0, 1, 0, 0] * 2)

dim = pd.Series(['apple', 'orange'])
```

In [8]:

```
values
```

Out[8]:

```
0    0
1    1
2    0
3    0
4    0
5    1
6    0
7    0
dtype: int64
```

In [9]:

```
dim
```

Out[9]:

```
0     apple
1    orange
dtype: object
```

In [11]:

```
# We can use the take method to restore the original Series of strings:

dim.take(values)
```

Out[11]:

```
0     apple
1    orange
0     apple
0     apple
0     apple
1    orange
0     apple
0     apple
dtype: object
```

In [ ]:

```
# This representation as integers is called the categorical or dictionary-encoded represen
tation.
    # The array of distinct values can be called the categories, dictionary, or levels of
 the data.
    # The integer values that reference the categories are called the category codes or si
mply codes.

# The categorical representation can yield significant performance improvements when you a
re doing analytics.
    # You can also perform transformations on the categories while leaving the codes unmod
ified.
    # Some example transformations that can be made at relatively low cost are:

#    • Renaming categories

#    • Appending a new category without changing the order or position of the existing cat
egories
```

In [12]:

```
# Categorical Type in pandas

# pandas has a special Categorical type for holding data that uses the integer-based
    # categorical representation or encoding.

# Let’s consider the example Series from before:


fruits = ['apple', 'orange', 'apple', 'apple'] * 2

N = len(fruits)

df = pd.DataFrame({'fruit': fruits,
                   'basket_id': np.arange(N),
                   'count': np.random.randint(3, 15, size=N),
                   'weight': np.random.uniform(0, 4, size=N)},
                  columns=['basket_id', 'fruit', 'count', 'weight'])


df
```

Out[12]:

| | basket_id | fruit | count | weight |
|---|---|---|---|---|
| **0** | 0 | apple | 4 | 2.188239 |
| **1** | 1 | orange | 6 | 3.023703 |
| **2** | 2 | apple | 5 | 0.357328 |
| **3** | 3 | apple | 8 | 2.489471 |
| **4** | 4 | apple | 3 | 3.331382 |
| **5** | 5 | orange | 9 | 3.562300 |
| **6** | 6 | apple | 12 | 2.087699 |
| **7** | 7 | apple | 11 | 0.365287 |

In [14]:

```
df.to_excel('fruitbasket.xlsx')
```

In [16]:

```
# Above, df['fruit'] is an array of Python string objects. We can convert it to categorica
l by calling:

fruit_cat = df['fruit'].astype('category')

fruit_cat
```

Out[16]:

```
0     apple
1    orange
2     apple
3     apple
4     apple
5    orange
6     apple
7     apple
Name: fruit, dtype: category
Categories (2, object): [apple, orange]
```

In [17]:

```
# The values for fruit_cat are not a NumPy array, but an instance of pandas.Categorical:

c = fruit_cat.values
type(c)
```

Out[17]:

```
pandas.core.arrays.categorical.Categorical
```

In [18]:

```
# The Categorical object has categories and codes attributes:

c.categories
```

Out[18]:

```
Index(['apple', 'orange'], dtype='object')
```

In [19]:

```
c.codes
```

Out[19]:

```
array([0, 1, 0, 0, 0, 1, 0, 0], dtype=int8)
```

In [20]:

```python
# You can convert a DataFrame column to categorical by assigning the converted result:

df['fruit'] = df['fruit'].astype('category')

df.fruit
```

Out[20]:

```
0     apple
1    orange
2     apple
3     apple
4     apple
5    orange
6     apple
7     apple
Name: fruit, dtype: category
Categories (2, object): [apple, orange]
```

In [21]:

```python
# You can also create pandas.Categorical directly from other types of Python sequences:


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

my_categories
```

Out[21]:

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

In [22]:

```python
# If you have obtained categorical encoded data from another source,
    # you can use the alternative from_codes constructor:

categories = ['foo', 'bar', 'baz']

codes = [0, 1, 2, 0, 0, 1]

my_cats_2 = pd.Categorical.from_codes(codes, categories)

my_cats_2
```

Out[22]:

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

In [23]:

```
# Unless explicitly specified, categorical conversions assume no specific ordering of the
 categories.
    # So the categories array may be in a different order depending on the ordering of the
input data.
    # When using from_codes or any of the other constructors,
    # you can indicate that the categories have a meaningful ordering:


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

ordered_cat
```

Out[23]:

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

In [24]:

```
# The above output [foo < bar < baz] indicates that 'foo' precedes 'bar' in the ordering,a
nd so on.
    # An unordered categorical instance can be made ordered with as_ordered:


my_cats_2.as_ordered()
```

Out[24]:

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

In [ ]:

```
# As a last note, categorical data need not be strings, even though I have only showed str
ing examples.
# A categorical array can consist of any immutable value types.
```

In [25]:

```
# Computations with Categoricals

# Using Categorical in pandas compared with the non-encoded version (like an array of stri
ngs)
    # generally behaves the same way.
    # Some parts of pandas, like the groupby function, perform better when working with ca
tegoricals.
    # There are also some functions that can utilize the ordered flag.

#Let’s consider some random numeric data, and use the pandas.qcut binning function.
    # This return pandas.Categorical:

np.random.seed(12345)

draws = np.random.randn(1000)

draws[:5]
```

Out[25]:

```
array([-0.20470766,  0.47894334, -0.51943872, -0.5557303 ,  1.96578057])
```

In [26]:

```
draws
```

Out[26]:

```
array([-2.04707659e-01,  4.78943338e-01, -5.19438715e-01, -5.55730304e-01,
        1.96578057e+00,  1.39340583e+00,  9.29078767e-02,  2.81746153e-01,
        7.69022568e-01,  1.24643474e+00,  1.00718936e+00, -1.29622111e+00,
        2.74991633e-01,  2.28912879e-01,  1.35291684e+00,  8.86429341e-01,
       -2.00163731e+00, -3.71842537e-01,  1.66902531e+00, -4.38569736e-01,
       -5.39741446e-01,  4.76985010e-01,  3.24894392e+00, -1.02122752e+00,
       -5.77087303e-01,  1.24121276e-01,  3.02613562e-01,  5.23772068e-01,
        9.40277775e-04,  1.34380979e+00, -7.13543985e-01, -8.31153539e-01,
       -2.37023165e+00, -1.86076079e+00, -8.60757398e-01,  5.60145293e-01,
       -1.26593449e+00,  1.19827125e-01, -1.06351245e+00,  3.32882716e-01,
       -2.35941881e+00, -1.99542955e-01, -1.54199553e+00, -9.70735912e-01,
       -1.30703025e+00,  2.86349747e-01,  3.77984111e-01, -7.53886535e-01,
        3.31285650e-01,  1.34974221e+00,  6.98766888e-02,  2.46674110e-01,
       -1.18616011e-02,  1.00481159e+00,  1.32719461e+00, -9.19261558e-01,
       -1.54910644e+00,  2.21845987e-02,  7.58363145e-01, -6.60524328e-01,
        8.62580083e-01, -1.00319021e-02,  5.00093559e-02,  6.70215594e-01,
        8.52965032e-01, -9.55868852e-01, -2.34933207e-02, -2.30423388e+00,
       -6.52468841e-01, -1.21830198e+00, -1.33260971e+00,  1.07462269e+00,
        7.23641505e-01,  6.90001853e-01,  1.00154344e+00, -5.03087391e-01,
       -6.22274225e-01, -9.21168608e-01, -7.26213493e-01,  2.22895546e-01,
        5.13161009e-02, -1.15771947e+00,  8.16706936e-01,  4.33609606e-01,
        1.01073695e+00,  1.82487521e+00, -9.97518248e-01,  8.50591099e-01,
       -1.31577601e-01,  9.12414152e-01,  1.88210680e-01,  2.16946144e+00,
       -1.14928205e-01,  2.00369736e+00,  2.96101523e-02,  7.95253156e-01,
        1.18109754e-01, -7.48531548e-01,  5.84969738e-01,  1.52676573e-01,
       -1.56565729e+00, -5.62540188e-01, -3.26641392e-02, -9.29006202e-01,
       -4.82572646e-01, -3.62638461e-02,  1.09539006e+00,  9.80928477e-01,
       -5.89487686e-01,  1.58170009e+00, -5.28734826e-01,  4.57001871e-01,
        9.29968759e-01, -1.56927061e+00, -1.02248698e+00, -4.02826924e-01,
        2.20486863e-01, -1.93401108e-01,  6.69158336e-01, -1.64898482e+00,
       -2.25279725e+00, -1.16683222e+00,  3.53607102e-01,  7.02110171e-01,
       -2.74569205e-01, -1.39142188e-01,  1.07657222e-01, -6.06545125e-01,
       -4.17064408e-01, -1.70070368e-02, -1.22414528e+00, -1.80083991e+00,
        1.63473620e+00,  9.89008302e-01,  4.57940143e-01,  5.55154410e-01,
        1.30671972e+00, -4.40553570e-01, -3.01350280e-01,  4.98791490e-01,
       -8.23991040e-01,  1.32056584e+00,  5.07964786e-01, -6.53437675e-01,
        1.86979514e-01, -3.91725249e-01, -2.72292975e-01, -1.71414356e-02,
        6.80320749e-01,  6.35512357e-01, -7.57176502e-01,  7.18085834e-01,
       -3.04273076e-01, -1.67779025e+00,  4.26986085e-01, -1.56373985e+00,
       -3.67487521e-01,  1.04591253e+00,  1.21995436e+00, -2.47699116e-01,
       -4.16232132e-01, -1.16747004e-01, -1.84478762e+00,  2.06870785e+00,
       -7.76967474e-01,  1.44016687e+00, -1.10557360e-01,  1.22738699e+00,
        1.92078426e+00,  7.46433038e-01,  2.22465959e+00, -6.79400410e-01,
        7.27368782e-01, -8.68730734e-01, -1.21385091e+00, -4.70630931e-01,
       -9.19241697e-01, -8.38826689e-01,  4.35155305e-01, -5.57804717e-01,
       -5.67454871e-01, -3.72641553e-01, -9.26556901e-01,  1.75510839e+00,
        1.20980999e+00,  1.27002473e+00, -9.74378127e-01, -6.34709255e-01,
       -3.95700752e-01, -2.89435900e-01, -7.34297072e-01, -7.28504679e-01,
        8.38775073e-01,  2.66893213e-01,  7.21194339e-01,  9.10982642e-01,
       -1.02090261e+00, -1.41341604e+00,  1.29660784e+00,  2.52275209e-01,
        1.12748110e+00, -5.68363447e-01,  3.09362168e-01, -5.77385473e-01,
       -1.16863407e+00, -8.25019972e-01, -2.64440949e+00, -1.52985803e-01,
       -7.51921003e-01, -1.32609252e-01,  1.45729970e+00,  6.09511845e-01,
       -4.93779257e-01,  1.23997988e+00, -1.35722140e-01,  1.43004181e+00,
       -8.46852451e-01,  6.03282130e-01,  1.26357226e+00, -2.55490556e-01,
```

```
-4.45688380e-01,  4.68366681e-01, -9.61603924e-01, -1.82450454e+00,
 6.25428156e-01,  1.02287238e+00,  1.10742460e+00,  9.09370895e-02,
-3.50108657e-01,  2.17957016e-01, -8.94813130e-01, -1.74149395e+00,
-1.05225574e+00,  1.43660279e+00, -5.76207386e-01, -2.42029443e+00,
-1.06232963e+00,  2.37372262e-01,  9.57369064e-04,  6.52531808e-02,
-1.36752411e+00, -3.02800519e-02,  9.40489321e-01, -6.42436751e-01,
 1.04017925e+00, -1.08292226e+00,  4.29213588e-01, -2.36223669e-01,
 6.41817816e-01, -3.31660557e-01,  1.39407223e+00, -1.07674194e+00,
-1.92465982e-01, -8.71187651e-01,  4.20851997e-01, -1.21141107e+00,
-2.58866912e-01, -5.81646850e-01, -1.26042063e+00,  4.64574793e-01,
-1.07024091e+00,  8.04222698e-01, -1.56735508e-01,  2.01039001e+00,
-8.87104430e-01, -9.77936232e-01, -2.67217350e-01,  4.83337822e-01,
-4.00332733e-01,  4.49880415e-01,  3.99593953e-01, -1.51574804e-01,
-2.55793406e+00,  1.60806841e-01,  7.65250677e-02, -2.97204166e-01,
-1.29427402e+00, -8.85180013e-01, -1.87496526e-01, -4.93560000e-01,
-1.15412964e-01, -3.50744607e-01,  4.46973764e-02, -8.97756316e-01,
 8.90873502e-01, -1.15118516e+00, -2.61230270e+00,  1.14125019e+00,
-8.67135525e-01,  3.83583258e-01, -4.37030164e-01,  3.47488810e-01,
-1.23017904e+00,  5.71078139e-01,  6.00612128e-02, -2.25523994e-01,
 1.34972614e+00,  1.35029973e+00, -3.86653322e-01,  8.65989542e-01,
 1.74723360e+00, -1.41024614e+00, -3.78241525e-01, -3.45820667e-01,
 3.80062465e-01,  1.88994673e-01,  1.32329820e+00, -2.26458859e+00,
-9.14978611e-01, -4.78964164e-01,  1.04718450e+00,  9.23948418e-01,
-1.14150141e-01,  4.05802443e-01,  2.88451808e-01, -4.34788492e-01,
 3.58755633e-01, -3.88244944e-01,  2.12874629e+00,  1.40960468e+00,
-1.05434278e-01,  7.00428367e-01,  2.09285188e+00, -1.36971796e-01,
-9.30489423e-01,  3.27497234e-01,  1.30301308e+00, -1.40940236e+00,
-1.44125945e-01, -7.16414024e-01,  1.03614186e-01, -1.49571856e+00,
-1.17489356e+00,  2.61399909e+00, -6.89307399e-01, -7.51652636e-01,
 6.36280961e-01, -1.15764416e+00,  6.14679924e-01,  1.02139111e+00,
 6.68272492e-01, -8.09535482e-01, -9.08124573e-01,  1.51228939e+00,
 9.51174320e-02,  1.18466855e+00,  6.37032934e-01, -5.39274587e-01,
-5.51009929e-02, -1.13592579e+00, -1.70506254e-01, -1.15808727e+00,
 1.10459914e+00,  6.34238100e-01,  1.25968335e+00,  9.64930776e-01,
-4.34445938e-01, -8.79602860e-01, -6.94838230e-01,  1.22637405e+00,
 4.57278649e-01,  1.15698625e-01,  1.01404235e+00, -1.13500772e+00,
-2.63370582e-01,  1.30642518e+00, -1.61084101e+00, -1.02662069e+00,
 1.24157279e+00, -1.56759532e-01, -2.44909553e+00, -1.03394802e+00,
 1.59953363e+00,  4.74070770e-01,  1.51325927e-01, -5.42173166e-01,
-4.75496217e-01,  1.06402612e-01, -1.30822819e+00,  2.17318475e+00,
 5.64561217e-01, -1.90480765e-01, -9.16934390e-01, -9.75813606e-01,
 2.21230269e+00,  7.39306826e-02,  1.81859456e+00, -1.58153100e+00,
-7.74363409e-01,  5.52936493e-01,  1.06061390e-01,  3.92752804e+00,
-2.55125599e-01,  8.54137315e-01, -3.64806532e-01,  1.31101737e-01,
-6.97613896e-01,  1.33564946e+00, -1.51038858e-01,  4.42937851e-01,
 9.41571221e-01,  5.33363953e-01,  3.56266193e-01, -1.01153190e-02,
 1.41575317e+00,  5.66105533e-01,  4.56487352e-01,  1.94788105e-01,
-6.55053757e-01, -5.65230094e-01,  3.17687312e+00,  9.59532542e-01,
-9.75339835e-01, -1.11674210e+00, -1.10437634e+00, -8.98755175e-01,
-1.36663236e+00,  4.51401732e-01, -1.58722181e+00, -7.32788830e-01,
-5.14562627e-01, -6.11876053e-01, -3.15986670e-02,  2.34127888e-01,
 2.71936281e-01,  1.23076935e+00,  1.28830616e+00,  8.51813508e-01,
-1.52918061e+00, -1.55171499e+00,  2.97292917e-01,  3.44791201e-01,
-3.98103360e-01,  4.29404057e-01, -2.85944649e-01, -2.22843700e+00,
 6.65587850e-02,  4.89964761e-01,  1.86792585e+00,  2.07043780e+00,
-2.45383227e-01,  7.62302108e-01,  1.29015475e-01,  6.27075914e-01,
-1.06223474e+00, -1.49950345e+00,  5.45154193e-01,  4.00823267e-01,
```

```
-1.94623039e+00,  5.05031784e-01, -9.10489138e-01, -2.19696285e-01,
 4.08055383e-01, -6.03145411e-01, -3.61132594e-01,  5.64024524e-01,
-1.05661656e+00,  1.39195005e+00, -1.76126767e+00, -9.11633128e-01,
 6.58339264e-01, -1.57946584e+00,  4.57705976e-01, -1.83268092e-01,
-6.85483896e-01,  1.06431315e-01, -3.18236084e-01,  4.33712973e-01,
 5.71825929e-01,  5.67106029e-01,  8.15769654e-02, -3.02334871e-01,
-7.26916388e-01,  1.80335113e-01, -5.20208734e-01,  3.98092081e-01,
-9.16935074e-01, -8.26501345e-02, -1.93969081e+00,  1.40799436e+00,
 1.51240649e+00,  5.26493120e-01, -2.66930833e-01,  8.62284048e-01,
 8.38030660e-02, -1.87233890e+00, -9.62791017e-01,  8.00668292e-02,
 1.28726442e-01, -4.79120340e-01, -6.40280504e-01,  7.45973801e-01,
-6.22547063e-01,  9.36289315e-01,  7.50018377e-01, -5.67150282e-02,
 2.30067451e+00,  5.69497467e-01,  1.48940968e+00,  1.26425028e+00,
-7.61837213e-01, -3.31616898e-01, -1.75131543e+00,  6.28894111e-01,
 2.82501864e-01, -1.33813943e+00, -5.00606850e-01,  1.21645030e-01,
 1.70832347e+00, -9.70999448e-01, -6.19332343e-01, -7.26708132e-01,
 1.22165542e+00,  5.03699288e-01, -1.38787408e+00,  2.04851420e-01,
 6.03705216e-01,  5.45680309e-01,  2.35477019e-01,  1.11834994e-01,
-1.25150375e+00, -2.94934350e+00,  6.34634161e-01,  1.24157016e-01,
 1.29762249e+00, -1.68693341e+00,  1.08953905e+00,  2.06088174e+00,
-2.41235326e-01, -9.47872180e-01,  6.76294029e-01, -6.53356162e-01,
-6.52295298e-01,  5.28827604e-01,  3.57793249e-01,  1.88649360e-01,
 8.69416879e-01, -5.06674481e-02, -7.16364575e-01, -1.03258721e-01,
-1.14103658e+00, -5.00776901e-01, -3.89301370e-01, -4.73850530e-01,
 1.28664304e-01,  1.53694305e-01,  4.44790058e-01,  1.28531667e-01,
 2.52529866e-01, -9.40638663e-01,  1.00214545e+00, -5.25414984e-01,
-8.87400936e-01,  1.83131360e+00, -9.23029332e-01,  7.00537687e-01,
-8.92151198e-01,  2.30074000e+00, -8.17765299e-01,  5.13759632e-01,
 6.23586943e-01,  1.48920593e+00,  1.94047867e+00,  5.43237129e-01,
 5.06190912e-01,  1.66201449e+00, -1.18920250e+00,  9.35974490e-02,
-5.39163905e-01, -1.43739560e+00,  1.87937386e-01, -4.50454457e-01,
-5.16878232e-01, -9.56356677e-02,  3.16423805e-01,  6.03334657e-01,
-1.49459146e+00, -1.10894079e-01,  2.41289404e-01, -5.82645109e-01,
-2.41112652e-01,  2.36360537e-01,  1.24720725e-01,  1.04632598e+00,
-2.73091856e-01, -5.34834020e-01, -3.06563305e-01, -1.62242665e-01,
-1.08323220e+00,  7.08401493e-01,  1.52074304e+00,  2.90343183e-01,
-6.83066330e-01, -9.50312866e-01,  4.00709936e-01, -1.26071684e-01,
 3.98204888e-01,  1.41638473e-01, -2.64141422e-01, -4.52212074e-01,
 7.58201973e-01, -5.15583498e-01, -5.91202322e-01,  8.96745784e-01,
-9.71437524e-01,  1.84080991e+00,  1.53881232e-01, -2.74083943e-01,
-1.78492569e+00,  9.81006686e-01, -8.73717140e-01, -1.01563442e+00,
-4.11243537e-01,  1.46562117e+00, -1.00621906e+00, -9.02147762e-01,
 7.52769143e-01, -4.90508527e-01, -5.24672210e-01, -6.99195861e-01,
 3.52360939e-01,  6.81025983e-02, -9.30341707e-01,  8.45399560e-01,
 1.64723816e-02,  8.44962955e-01,  1.85083395e+00,  2.20742409e-02,
-1.36917902e+00,  8.87203523e-01,  1.43311821e-02, -7.41547051e-02,
-4.85647878e-02,  1.23502145e+00, -4.33294924e-01,  1.39103546e+00,
 8.20210741e-01, -2.47423465e-01,  3.02270746e-01,  5.43980361e-01,
-9.42368504e-01, -1.26638281e+00,  9.37249545e-01, -7.20102245e-01,
-1.59395154e+00, -3.75497816e-01, -9.58703834e-01,  7.94336400e-01,
-1.60510784e+00,  5.43710253e-01,  9.25166364e-01, -1.46962860e+00,
-3.99592346e-01,  1.41734264e+00, -8.97608668e-01,  1.84480502e+00,
 1.25316821e+00, -1.49093242e+00, -2.77339246e-02,  1.37523596e+00,
-2.52081701e-02, -6.67880179e-01, -2.86801753e+00,  2.10688543e-01,
 1.28715531e+00, -5.74305988e-01,  4.95326647e-01,  3.96049590e-01,
 5.88798190e-01, -1.28175713e+00,  2.02992261e+00, -5.01944516e-01,
-1.59284566e-01, -1.49621630e+00,  1.14477139e-02,  4.19445985e-01,
```

```
 2.05121388e+00, -3.68765333e-01, -1.68925468e+00,  1.47681161e-01,
-1.80998392e-01,  1.58059054e-01, -3.96615422e-01, -4.00236630e-01,
-8.24895666e-01, -2.44440446e-01,  1.21945743e+00, -4.33630492e-01,
 8.61183873e-01, -3.34503693e-01,  1.59559959e-01, -9.84164476e-01,
 7.54084974e-01, -2.84391662e-01,  3.24797530e-01, -8.85424602e-01,
-1.28089348e+00,  1.96109935e-01,  9.54644156e-01, -8.00971332e-01,
 1.58514730e-02,  1.08755329e+00, -6.31242820e-01, -2.26893249e-02,
 6.85879242e-01,  5.19179208e-01,  1.82701892e-01,  2.04647381e-01,
-2.65986356e-01, -2.27288704e-04,  1.23945232e+00, -8.19715256e-01,
-2.60388907e-01,  5.19140257e-01,  1.43091645e-01, -1.16677747e-01,
 1.49674411e+00, -1.48427438e+00, -1.67118276e+00,  9.17173409e-01,
-7.58014151e-01,  2.06479240e+00, -8.50778396e-01,  4.99450713e-01,
-7.92663655e-02, -1.40329264e+00,  1.57894791e+00,  3.69028988e-04,
 9.00884914e-01, -4.54869220e-01, -8.64546645e-01,  1.12911990e+00,
 5.78744129e-02, -4.33738666e-01,  9.26976374e-02, -1.39782015e+00,
 1.45782265e+00, -1.76756916e-01, -2.54240300e-01, -1.26343750e+00,
 4.52262741e-01, -8.40117409e-01, -5.02678071e-01,  5.13392587e-01,
 1.64165300e+00,  5.80790036e-01, -1.70734027e+00, -1.78355431e-01,
-8.28459954e-01,  1.28631168e+00, -4.06452362e-01,  1.56632047e-01,
 5.21066804e-02,  9.55813177e-01,  7.43191501e-01, -4.86323084e-01,
 1.92046727e+00, -6.52749023e-01, -1.73303777e-01, -3.60410082e-01,
-3.80413977e-01, -1.29813981e+00,  5.27919008e-01, -9.31002763e-02,
 4.01184681e-01, -1.02583380e-01,  3.08690977e-02,  2.61610051e+00,
-7.85577945e-01, -5.06998121e-01, -2.01820572e+00, -6.76853138e-01,
 2.66674368e+00,  1.45145615e+00,  6.34628855e-01, -5.02826864e-01,
 5.12931659e-01,  1.75677937e+00, -9.74310801e-01,  6.80397048e-01,
 9.55798726e-01,  1.50153548e+00, -7.56265648e-01,  4.73504604e-01,
 1.71374345e+00, -1.14769922e+00,  2.90322050e-03, -1.10057036e+00,
-2.97531782e-01,  5.02409078e-01, -9.87418981e-04, -6.74560278e-01,
 2.97958279e-01,  1.46557314e+00, -3.03628594e-01, -9.94479885e-01,
 1.89889991e-01, -1.68402957e+00, -4.58380742e-01,  5.43405908e-01,
-1.18726426e+00, -4.12641693e-01,  1.17712535e+00, -3.13704165e-01,
 1.57903162e+00,  3.75388236e-01, -1.56813882e+00, -9.00886519e-01,
 6.52345519e-01,  8.71600314e-01,  2.68216170e-01,  9.47681220e-01,
 1.47267588e-01, -1.77245546e+00,  5.92419611e-01,  9.03254745e-02,
 6.51121454e-01, -8.11946962e-02,  8.01897603e-01,  1.39845227e-01,
-5.01002762e-01, -1.28302559e-01,  4.14605966e-01,  6.04577786e-01,
 2.13409475e+00,  9.41187837e-01, -9.31456796e-01, -1.24667539e-01,
 2.00696291e-01,  1.80256286e-01, -3.20370097e-01, -1.59612803e+00,
-1.28169898e+00,  1.50258575e+00,  6.53538002e-01, -3.19536626e-01,
 9.55094011e-01,  2.61995955e-01,  1.60792901e-01, -5.71680642e-01,
 3.51660059e-01,  1.11498006e+00,  1.18326826e+00,  1.06094106e+00,
 5.10712630e-01, -9.38783998e-01, -5.46496141e-01,  5.90029971e-01,
 1.48218524e+00,  1.02118104e-01,  2.65438049e-01,  3.19307433e-03,
-2.59501150e+00, -1.55556933e+00,  1.10299596e+00,  5.54736504e-01,
-1.28901164e+00,  3.85241648e-01, -1.71729173e+00, -1.01835313e+00,
 5.16353173e-02,  5.03298710e-01, -5.43186231e-01, -5.06678417e-01,
 7.29652833e-01,  4.34273363e-01, -1.13367361e+00,  1.42395334e+00,
 2.66351537e-01, -8.54264393e-01, -5.50596561e-01, -6.19109859e-01,
 1.03893339e+00, -9.10610825e-01,  5.29952567e-01, -8.47143615e-03,
-1.12903825e+00,  5.69854191e-01, -8.63391622e-01, -1.35614427e+00,
-5.71515569e-02, -1.08621122e-01,  1.65238409e+00, -1.35092808e+00,
-5.46096737e-01,  9.91400184e-01,  2.20099739e+00,  4.27899791e-01,
 2.90468283e-01,  6.11953096e-01, -5.12450984e-01, -7.24230691e-01,
 1.69288190e+00, -2.99339120e-01,  1.57172719e+00,  4.61444067e-01,
-6.73829701e-01, -1.14103626e+00, -1.22891798e+00, -1.15928246e+00,
-3.20829018e-01,  1.08834758e+00, -9.06203145e-01, -4.64152062e-01,
```

```
       -5.13378373e-01,  1.61783768e+00, -8.16650606e-01,  2.44719605e-01,
       -1.31109423e+00,  3.88406495e-01,  1.59237371e+00,  8.70399037e-01,
        3.35249325e-01,  6.48959907e-01, -1.83151791e-01,  5.00241270e-01,
        1.36882639e+00,  8.95091842e-01,  6.47293372e-01, -5.67878708e-01,
       -5.79517447e-01, -7.51448573e-01,  1.07551918e+00, -6.21142360e-01,
        1.87855572e+00,  1.26023993e+00,  3.10050973e-01,  1.06402292e-01,
        2.48012997e-01, -1.39383959e+00, -6.69436307e-01, -5.66791474e-01,
       -3.81778903e-01, -9.46546907e-01, -1.06510300e+00, -1.33182618e+00,
       -9.86453191e-01, -3.78391147e-01,  7.64711975e-01,  6.03594165e-02,
        6.18509999e-01, -4.84921020e-01, -2.80530240e-01,  4.06962904e-01,
        1.02518779e+00,  2.54751681e-01,  8.75239905e-02,  7.06983543e-02,
       -5.73152603e-01,  1.22892597e+00, -9.62201893e-01,  1.52555676e+00,
        8.27282589e-01,  9.12470470e-01, -1.27292343e-01,  6.34316641e-01,
       -1.53089843e+00, -1.29070149e+00, -5.26228341e-01, -1.13223396e+00,
       -4.99797127e-01, -7.28463087e-01, -5.83144170e-01,  3.29290657e-01,
       -8.26860798e-01, -5.36867983e-01, -5.62980134e-01,  9.18404800e-01,
       -7.93993782e-02, -2.78624683e-01, -1.30459539e-01, -1.39699761e+00,
       -2.44713889e-01,  8.30253911e-01,  2.40821202e-01, -9.15697123e-01,
       -2.22527996e+00, -6.63067012e-01, -3.21194764e-01,  4.98388165e-01,
        3.80338976e-01, -1.06703532e+00,  2.55452172e-01,  2.11128719e+00,
       -6.34189962e-01,  1.36875577e+00, -9.70649489e-01,  6.54245334e-01,
       -1.17189522e+00, -3.15987198e-03, -7.45604825e-01,  1.59829089e+00,
       -9.13399998e-01,  2.40291209e+00, -5.89360262e-01,  1.07657442e-01,
       -1.39297516e-01, -1.15992573e+00,  6.18964782e-01,  1.37389047e+00])
```

In [27]:

```python
# Let's compute a quartile binning of this data and extract some statistics:

bins = pd.qcut(draws, 4)

bins
```

Out[27]:

```
[(-0.684, -0.0101], (-0.0101, 0.63], (-0.684, -0.0101], (-0.684, -0.0101],
(0.63, 3.928], ..., (-0.0101, 0.63], (-0.684, -0.0101], (-2.9499999999999997,
-0.684], (-0.0101, 0.63], (0.63, 3.928]]
Length: 1000
Categories (4, interval[float64]): [(-2.9499999999999997, -0.684] < (-0.684,
-0.0101] < (-0.0101, 0.63] < (0.63, 3.928]]
```

In [29]:

```
# While useful, the exact sample quartiles may be less useful for producing a report than
 quartile names.
    # We can achieve this with the labels argument to qcut:


bins = pd.qcut(draws, 4, labels=['Q1', 'Q2', 'Q3', 'Q4'])

bins
```

Out[29]:

```
[Q2, Q3, Q2, Q2, Q4, ..., Q3, Q2, Q1, Q3, Q4]
Length: 1000
Categories (4, object): [Q1 < Q2 < Q3 < Q4]
```

In [30]:

```python
bins.codes
```

Out[30]:

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

In [31]:

```
bins.codes[:10]
```

Out[31]:

```
array([1, 2, 1, 1, 3, 3, 2, 2, 3, 3], dtype=int8)
```

In [35]:

```
# The labeled bins categorical does not contain information about the bin edges in the dat
a,
    # so we can use groupby to extract some summary statistics:


bins = pd.Series(bins, name='quartile')

results = (pd.Series(draws).groupby(bins).agg(['count','min', 'max']).reset_index())

results
```

Out[35]:

| | quartile | count | min | max |
|---|---|---|---|---|
| **0** | Q1 | 250 | -2.949343 | -0.685484 |
| **1** | Q2 | 250 | -0.683066 | -0.010115 |
| **2** | Q3 | 250 | -0.010032 | 0.628894 |
| **3** | Q4 | 250 | 0.634238 | 3.927528 |

In [38]:

```
results2 = (pd.Series(draws).groupby(bins).agg(['count', 'std', 'var','min', 'max', 'sum'
]).reset_index())

results2
```

Out[38]:

| | quartile | count | std | var | min | max | sum |
|---|---|---|---|---|---|---|---|
| **0** | Q1 | 250 | 0.450448 | 0.202903 | -2.949343 | -0.685484 | -303.995226 |
| **1** | Q2 | 250 | 0.191283 | 0.036589 | -0.683066 | -0.010115 | -90.605646 |
| **2** | Q3 | 250 | 0.188923 | 0.035692 | -0.010032 | 0.628894 | 76.960037 |
| **3** | Q4 | 250 | 0.518827 | 0.269181 | 0.634238 | 3.927528 | 315.290125 |

In [39]:

```
# The 'quartile' column in the result retains the original categorical information,
    # including ordering, from bins:


results['quartile']
```

Out[39]:

```
0    Q1
1    Q2
2    Q3
3    Q4
Name: quartile, dtype: category
Categories (4, object): [Q1 < Q2 < Q3 < Q4]
```

In [42]:

```
# Better performance with categoricals

# If you do a lot of analytics on a particular dataset, converting to categorical can yiel
d
    # substantial overall performance gains.
    # A categorical version of a DataFrame column will often use significantly less memor
y, too.

# Let's consider some Series with 10 million elements and a small number of distinct categ
ories:


N = 10000000

draws = pd.Series(np.random.randn(N))

labels = pd.Series(['foo', 'bar', 'baz', 'qux'] * (N // 4))
```

In [43]:

```
draws
```

Out[43]:

```
0         -0.991821
1          0.840305
2          1.242114
3          0.217027
4         -0.427622
             ...
9999995    0.606146
9999996   -1.507969
9999997   -0.319387
9999998    1.338074
9999999    0.739566
Length: 10000000, dtype: float64
```

In [44]:

```
labels
```

Out[44]:

```
0          foo
1          bar
2          baz
3          qux
4          foo
          ...
9999995    qux
9999996    foo
9999997    bar
9999998    baz
9999999    qux
Length: 10000000, dtype: object
```

In [45]:

```
# Now we convert labels to categorical:


categories = labels.astype('category')
```

In [57]:

```
# Now we note that labels uses significantly more memory than categories:


labels.memory_usage()
```

Out[57]:

```
80000128
```

In [58]:

```
categories.memory_usage()
```

Out[58]:

```
10000320
```

In [59]:

```
# The conversion to category is not free, of course, but it is a one-time cost:


%time _ = labels.astype('category')
```

```
Wall time: 1.19 s
```

In [ ]:

```
# GroupBy operations can be significantly faster with categoricals because the underlying
    # algorithms use the integer-based codes array instead of an array of strings.
```

In [60]:

```
# Categorical Methods

# Series containing categorical data have several special methods similar to the Series.st
r
    # specialized string methods.  This also provides convenient access to the categoriesa
nd codes.

# Consider the Series:


s = pd.Series(['a', 'b', 'c', 'd'] * 2)

cat_s = s.astype('category')

cat_s
```

Out[60]:

```
0    a
1    b
2    c
3    d
4    a
5    b
6    c
7    d
dtype: category
Categories (4, object): [a, b, c, d]
```

In [61]:

```
# The special attribute cat provides access to categorical methods:

cat_s.cat.codes
```

Out[61]:

```
0    0
1    1
2    2
3    3
4    0
5    1
6    2
7    3
dtype: int8
```

In [62]:

```
cat_s.cat.categories
```

Out[62]:

```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

In [63]:

```
# Suppose that we know the actual set of categories for this data extends beyond the
    # four values observed in the data. We can use the set_categories method to change the
m:

actual_categories = ['a', 'b', 'c', 'd', 'e']

cat_s2 = cat_s.cat.set_categories(actual_categories)

cat_s2
```

Out[63]:

```
0    a
1    b
2    c
3    d
4    a
5    b
6    c
7    d
dtype: category
Categories (5, object): [a, b, c, d, e]
```

In [64]:

```
# While it appears that the data is unchanged,
    # the new categories will be reflected in operations that use them.
    # For example, value_counts respects the categories, if present:

cat_s.value_counts()
```

Out[64]:

```
d    2
c    2
b    2
a    2
dtype: int64
```

In [67]:

```
cat_s2.value_counts()
```

Out[67]:

```
d    2
c    2
b    2
a    2
e    0
dtype: int64
```

In [68]:

```
# In large datasets, categoricals are often used as a convenient tool for memory savings
    # and better performance.

# After you filter a large DataFrame or Series, many of the categories may not appear in t
he data.
    # To help with this, we can use the remove_unused_categories method to trim unobserved
categories:


cat_s3 = cat_s[cat_s.isin(['a', 'b'])]

cat_s3
```

Out[68]:

```
0    a
1    b
4    a
5    b
dtype: category
Categories (4, object): [a, b, c, d]
```

In [69]:

```
cat_s3.cat.remove_unused_categories()
```

Out[69]:

```
0    a
1    b
4    a
5    b
dtype: category
Categories (2, object): [a, b]
```

In [ ]:

```
# Categorical methods for Series in pandas


  #             Method                        Description


  #             add_categories                Append new (unused) categories at end of ex
isting categories

  #             as_ordered                    Make categories ordered

  #             as_unordered                  Make categories unordered

  #             remove_categories             Remove categories, setting any removed valu
es to null

  #             remove_unused_categories      Remove any category values which do not app
ear in the data

  #             rename_categories             Replace categories with indicated set of ne
w category names;
  #                                             cannot change the number of categories

  #             reorder_categories            Behaves like rename_categories, but can als
o change
  #                                             the result to have ordered categories

  #             set_categories                Replace the categories with the indicated s
et of new categories;
  #                                             can add or remove categories
```

In [70]:

```
# When you’re using statistics or machine learning tools,
    # you’ll often transform categorical data into dummy variables, also known as one-hot
 encoding.
    # This involves creating a DataFrame with a column for each distinct category;
    # these columns contain 1s for occurrences of a given category and 0 otherwise.

#Consider the previous example:


cat_s = pd.Series(['a', 'b', 'c', 'd'] * 2, dtype='category')
```

In [72]:

```
#The pandas.get_dummies function converts this one-dimensional categorical data into a Dat
aFrame
    # containing the dummy variable:


pd.get_dummies(cat_s)
```

Out[72]:

| | a | b | c | d |
|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 0 |
| **1** | 0 | 1 | 0 | 0 |
| **2** | 0 | 0 | 1 | 0 |
| **3** | 0 | 0 | 0 | 1 |
| **4** | 1 | 0 | 0 | 0 |
| **5** | 0 | 1 | 0 | 0 |
| **6** | 0 | 0 | 1 | 0 |
| **7** | 0 | 0 | 0 | 1 |

In [73]:

```
# Advanced GroupBy Use

# Group Transforms and “Unwrapped” GroupBys

# Apart from the apply method in grouped operations for performing transformations,
    # there is another built-in method called transform, which is similar to apply but imp
oses more constraints
    # on the kind of function you can use:

#        • It can produce a scalar value to be broadcast to the shape of the group
#        • It can produce an object of the same shape as the input group
#        • It must not mutate its input


# Let’s consider a simple example for illustration:


df = pd.DataFrame({'key': ['a', 'b', 'c'] * 4, 'value': np.arange(12.)})

df
```

Out[73]:

| | key | value |
|---|---|---|
| **0** | a | 0.0 |
| **1** | b | 1.0 |
| **2** | c | 2.0 |
| **3** | a | 3.0 |
| **4** | b | 4.0 |
| **5** | c | 5.0 |
| **6** | a | 6.0 |
| **7** | b | 7.0 |
| **8** | c | 8.0 |
| **9** | a | 9.0 |
| **10** | b | 10.0 |
| **11** | c | 11.0 |

In [75]:

```
# Here are the group means by key:


g = df.groupby('key').value

g.mean()
```

Out[75]:

```
key
a    4.5
b    5.5
c    6.5
Name: value, dtype: float64
```

In [76]:

```
# Suppose instead we wanted to produce a Series of the same shape as df['value']
    # but with values replaced by the average grouped by 'key'.
    # We can pass the function lambda x: x.mean() to transform:

g.transform(lambda x: x.mean())
```

Out[76]:

```
0     4.5
1     5.5
2     6.5
3     4.5
4     5.5
5     6.5
6     4.5
7     5.5
8     6.5
9     4.5
10    5.5
11    6.5
Name: value, dtype: float64
```

In [77]:

```
# For built-in aggregation functions, we can pass a string alias as with the GroupBy agg m
ethod:

g.transform('mean')
```

Out[77]:

```
0     4.5
1     5.5
2     6.5
3     4.5
4     5.5
5     6.5
6     4.5
7     5.5
8     6.5
9     4.5
10    5.5
11    6.5
Name: value, dtype: float64
```

In [78]:

```
# Like apply, transform works with functions that return Series,
    # but the result must be the same size as the input.
    # For example, we can multiply each group by 2 using a lambda function:


g.transform(lambda x: x * 2)
```

Out[78]:

```
0      0.0
1      2.0
2      4.0
3      6.0
4      8.0
5     10.0
6     12.0
7     14.0
8     16.0
9     18.0
10    20.0
11    22.0
Name: value, dtype: float64
```

In [79]:

```
# As a more complicated example, we can compute the ranks in descending order for each gro
up:

g.transform(lambda x: x.rank(ascending=False))
```

Out[79]:

```
0     4.0
1     4.0
2     4.0
3     3.0
4     3.0
5     3.0
6     2.0
7     2.0
8     2.0
9     1.0
10    1.0
11    1.0
Name: value, dtype: float64
```

In [80]:

```
# Consider a group transformation function composed from simple aggregations:


def normalize(x):
    return (x - x.mean()) / x.std()
```

In [81]:

```
# We can obtain equivalent results in this case either using transform or apply:

g.transform(normalize)
```

Out[81]:

```
0    -1.161895
1    -1.161895
2    -1.161895
3    -0.387298
4    -0.387298
5    -0.387298
6     0.387298
7     0.387298
8     0.387298
9     1.161895
10    1.161895
11    1.161895
Name: value, dtype: float64
```

In [82]:

```
g.apply(normalize)
```

Out[82]:

```
0    -1.161895
1    -1.161895
2    -1.161895
3    -0.387298
4    -0.387298
5    -0.387298
6     0.387298
7     0.387298
8     0.387298
9     1.161895
10    1.161895
11    1.161895
Name: value, dtype: float64
```

In [83]:

```
# Built-in aggregate functions like 'mean' or 'sum' are often much faster than a general a
pply function.
    # These also have a “fast past” when used with transform.
    # This allows us to perform a so-called unwrapped group operation:

g.transform('mean')
```

Out[83]:

```
0     4.5
1     5.5
2     6.5
3     4.5
4     5.5
5     6.5
6     4.5
7     5.5
8     6.5
9     4.5
10    5.5
11    6.5
Name: value, dtype: float64
```

In [84]:

```
normalized = (df['value'] - g.transform('mean')) / g.transform('std')

normalized
```

Out[84]:

```
0    -1.161895
1    -1.161895
2    -1.161895
3    -0.387298
4    -0.387298
5    -0.387298
6     0.387298
7     0.387298
8     0.387298
9     1.161895
10    1.161895
11    1.161895
Name: value, dtype: float64
```

In [ ]:

```
# While an unwrapped group operation may involve multiple group aggregations,
    # the overall benefit of vectorized operations often outweighs this.
```

In [85]:

```
# Grouped Time Resampling

# For time series data, the resample method is semantically a group operation based on a t
ime intervalization.

# Here's a small example table:


N = 15

times = pd.date_range('2017-05-20 00:00', freq='1min', periods=N)

df = pd.DataFrame({'time': times, 'value': np.arange(N)})

df
```

Out[85]:

| | time | value |
|---|---|---|
| **0** | 2017-05-20 00:00:00 | 0 |
| **1** | 2017-05-20 00:01:00 | 1 |
| **2** | 2017-05-20 00:02:00 | 2 |
| **3** | 2017-05-20 00:03:00 | 3 |
| **4** | 2017-05-20 00:04:00 | 4 |
| **5** | 2017-05-20 00:05:00 | 5 |
| **6** | 2017-05-20 00:06:00 | 6 |
| **7** | 2017-05-20 00:07:00 | 7 |
| **8** | 2017-05-20 00:08:00 | 8 |
| **9** | 2017-05-20 00:09:00 | 9 |
| **10** | 2017-05-20 00:10:00 | 10 |
| **11** | 2017-05-20 00:11:00 | 11 |
| **12** | 2017-05-20 00:12:00 | 12 |
| **13** | 2017-05-20 00:13:00 | 13 |
| **14** | 2017-05-20 00:14:00 | 14 |

In [86]:

```
# Here, we can index by 'time' and then resample:

df.set_index('time').resample('5min').count()
```

Out[86]:

| | value |
|---|---|
| **time** | |
| **2017-05-20 00:00:00** | 5 |
| **2017-05-20 00:05:00** | 5 |
| **2017-05-20 00:10:00** | 5 |

In [87]:

```
df.set_index('time').resample('2min').count()
```

Out[87]:

| | value |
|---|---|
| **time** | |
| **2017-05-20 00:00:00** | 2 |
| **2017-05-20 00:02:00** | 2 |
| **2017-05-20 00:04:00** | 2 |
| **2017-05-20 00:06:00** | 2 |
| **2017-05-20 00:08:00** | 2 |
| **2017-05-20 00:10:00** | 2 |
| **2017-05-20 00:12:00** | 2 |
| **2017-05-20 00:14:00** | 1 |

In [88]:

```
#Suppose that a DataFrame contains multiple time series, marked by an additional group key column:

df2 = pd.DataFrame({'time': times.repeat(3), 'key': np.tile(['a', 'b', 'c'], N), 'value': np.arange(N * 3.)})


df2[:7]
```

Out[88]:

| | time | key | value |
|---|---|---|---|
| **0** | 2017-05-20 00:00:00 | a | 0.0 |
| **1** | 2017-05-20 00:00:00 | b | 1.0 |
| **2** | 2017-05-20 00:00:00 | c | 2.0 |
| **3** | 2017-05-20 00:01:00 | a | 3.0 |
| **4** | 2017-05-20 00:01:00 | b | 4.0 |
| **5** | 2017-05-20 00:01:00 | c | 5.0 |
| **6** | 2017-05-20 00:02:00 | a | 6.0 |

In [89]:

```
df2
```

Out[89]:

| | time | key | value |
|---|---|---|---|
| **0** | 2017-05-20 00:00:00 | a | 0.0 |
| **1** | 2017-05-20 00:00:00 | b | 1.0 |
| **2** | 2017-05-20 00:00:00 | c | 2.0 |
| **3** | 2017-05-20 00:01:00 | a | 3.0 |
| **4** | 2017-05-20 00:01:00 | b | 4.0 |
| **5** | 2017-05-20 00:01:00 | c | 5.0 |
| **6** | 2017-05-20 00:02:00 | a | 6.0 |
| **7** | 2017-05-20 00:02:00 | b | 7.0 |
| **8** | 2017-05-20 00:02:00 | c | 8.0 |
| **9** | 2017-05-20 00:03:00 | a | 9.0 |
| **10** | 2017-05-20 00:03:00 | b | 10.0 |
| **11** | 2017-05-20 00:03:00 | c | 11.0 |
| **12** | 2017-05-20 00:04:00 | a | 12.0 |
| **13** | 2017-05-20 00:04:00 | b | 13.0 |
| **14** | 2017-05-20 00:04:00 | c | 14.0 |
| **15** | 2017-05-20 00:05:00 | a | 15.0 |
| **16** | 2017-05-20 00:05:00 | b | 16.0 |
| **17** | 2017-05-20 00:05:00 | c | 17.0 |
| **18** | 2017-05-20 00:06:00 | a | 18.0 |
| **19** | 2017-05-20 00:06:00 | b | 19.0 |
| **20** | 2017-05-20 00:06:00 | c | 20.0 |
| **21** | 2017-05-20 00:07:00 | a | 21.0 |
| **22** | 2017-05-20 00:07:00 | b | 22.0 |
| **23** | 2017-05-20 00:07:00 | c | 23.0 |
| **24** | 2017-05-20 00:08:00 | a | 24.0 |
| **25** | 2017-05-20 00:08:00 | b | 25.0 |
| **26** | 2017-05-20 00:08:00 | c | 26.0 |
| **27** | 2017-05-20 00:09:00 | a | 27.0 |
| **28** | 2017-05-20 00:09:00 | b | 28.0 |
| **29** | 2017-05-20 00:09:00 | c | 29.0 |
| **30** | 2017-05-20 00:10:00 | a | 30.0 |
| **31** | 2017-05-20 00:10:00 | b | 31.0 |
| **32** | 2017-05-20 00:10:00 | c | 32.0 |
| **33** | 2017-05-20 00:11:00 | a | 33.0 |

| | time | key | value |
|---|---|---|---|
| **34** | 2017-05-20 00:11:00 | b | 34.0 |
| **35** | 2017-05-20 00:11:00 | c | 35.0 |
| **36** | 2017-05-20 00:12:00 | a | 36.0 |
| **37** | 2017-05-20 00:12:00 | b | 37.0 |
| **38** | 2017-05-20 00:12:00 | c | 38.0 |
| **39** | 2017-05-20 00:13:00 | a | 39.0 |
| **40** | 2017-05-20 00:13:00 | b | 40.0 |
| **41** | 2017-05-20 00:13:00 | c | 41.0 |
| **42** | 2017-05-20 00:14:00 | a | 42.0 |
| **43** | 2017-05-20 00:14:00 | b | 43.0 |
| **44** | 2017-05-20 00:14:00 | c | 44.0 |

In [105]:

```
# To do the same resampling for each value of 'key', we introduce the pandas.TimeGrouper o
bject:

time_key = pd.TimeGrouper('5min')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-105-5cac671d75a8> in <module>
      1 # To do the same resampling for each value of 'key', we introduce the
pandas.TimeGrouper object:
      2 
----> 3 time_key = pd.TimeGrouper('5min')

~\anaconda3\Anaconda3-2020\lib\site-packages\pandas\__init__.py in __getattr_
_(name)
    260             return _SparseArray
    261 
--> 262         raise AttributeError(f"module 'pandas' has no attribute '{nam
e}'")
    263 
    264 

AttributeError: module 'pandas' has no attribute 'TimeGrouper'
```

In [106]:

```
# We can then set the time index, group by 'key' and time_key, and aggregate:

resampled = (df2.set_index('time').groupby(['key', time_key]).sum())
resampled
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-106-3e00b757a79d> in <module>
      1 # We can then set the time index, group by 'key' and time_key, and ag
gregate:
      2
----> 3 resampled = (df2.set_index('time').groupby(['key', time_key]).sum())
      4 resampled

NameError: name 'time_key' is not defined
```

In [107]:

```
resampled.reset_index()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-107-8b23ed8a5956> in <module>
----> 1 resampled.reset_index()

NameError: name 'resampled' is not defined
```

In [ ]:

```
# Techniques for Method Chaining

# When applying a sequence of transformations to a dataset, you may find yourself creating
numerous
    # temporary variables that are never used in your analysis.

# Consider this example, for instance:

df = load_data()
df2 = df[df['col2'] < 0]
df2['col1_demeaned'] = df2['col1'] - df2['col1'].mean()
result = df2.groupby('key').col1_demeaned.std()
```

In [ ]:

```
# While we're not using any real data above, the example highlights some new methods.
    # First, the DataFrame.assign method is a functional alternative to column assignments
of
    # the form df[k] = v.
    # Rather than modifying the object in-place, it returns a new DataFrame with the indic
ated modifications.
    # So these statements are equivalent:

# Usual non-functional way
df2 = df.copy()
df2['k'] = v

# Functional assign way
df2 = df.assign(k=v)
```

In [ ]:

```
# Assigning in-place may execute faster than using assign, but assign enables easier metho
d chaining:

result = df2.assign(col1_demeaned=df2.col1 - df2.col2.mean()).groupby('key').col1_demeaned
.std()
```

In [ ]:

```
# One thing to keep in mind when doing method chaining is that you may need to refer to te
mporary objects.
    # In the preceding example, we cannot refer to the result of load_data until it has be
en assigned
    # # to the temporary variable df.
    # To help with this, assign and many other pandas functions accept function-like argum
ents,
    # also known as callables.

# To show callables in action, consider a fragment of the example from before:


df = load_data()
df2 = df[df['col2'] < 0]
```

In [ ]:

```
# This can be rewritten as:

df = load_data()[lambda x: x['col2'] < 0]
```

In [ ]:

```
# Above, the result of load_data is not assigned to a variable,
    # so the function passed into [] is then bound to the object at that stage of the meth
od chain.

# We can continue, then, and write the entire sequence as a single chained expression:

result = load_data()[lambda x: x.col2 < 0].
          assign(col1_demeaned=lambda x: x.col1 - x.col1.mean()).
          groupby('key').col1_demeaned.std()
```

In [ ]:

```
# Whether you prefer to write code in this style is a matter of taste, and splitting up th
e
    # expression into multiple steps may make your code more readable.
```

In [ ]:

```
# The pipe Method

# You can accomplish a lot with built-in pandas functions and the approaches to
    # method chaining with callables that we just looked at.
    # However, sometimes you need to use your own functions or functions from third-party
 libraries.
    # This is where the pipe method comes in.

# Consider a sequence of function calls:

a = f(df, arg1=v1)
b = g(a, v2, arg3=v3)
c = h(b, arg4=v4)
```

In [ ]:

```
# When using functions that accept and return Series or DataFrame objects,
    # you can rewrite this using calls to pipe:

result = df.pipe(f, arg1=v1).pipe(g, v2, arg3=v3).pipe(h, arg4=v4)
```

In [ ]:

```
# The statement f(df) and df.pipe(f) are equivalent, but pipe makes chained invocation eas
ier.

# A potentially useful pattern for pipe is to generalize sequences of operations into reus
able functions.
    # As an example, let’s consider substracting group means from a column:

g = df.groupby(['key1', 'key2'])
df['col1'] = df['col1'] - g.transform('mean')
```

In [ ]:

```
# Suppose that you wanted to be able to demean more than one column and easily change the
 group keys.
    # Additionally, you might want to perform this transformation in a method chain.

# Here is an example implementation:

def group_demean(df, by, cols):
    result = df.copy()
    g = df.groupby(by)
    for c in cols:
        result[c] = df[c] - g[c].transform('mean')
    return result
```

In [ ]:

```
# Then it is possible to write:

result = df[df.col1 < 0].pipe(group_demean, ['key1', 'key2'], ['col1'])
```

# The End