# Advanced pandas

In [1]:
```
import numpy as np
import pandas as pd
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
PREVIOUS_MAX_ROWS = pd.options.display.max_rows
pd.options.display.max_rows = 20
np.set_printoptions(precision=4, suppress=True)
```

## Categorical Data

### Background and Motivation

In [2]:
```
import numpy as np; import pandas as pd
values = pd.Series(['apple', 'orange', 'apple',
                    'apple'] * 2)
values
```

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

In [3]:
```
pd.unique(values)
```

Out[3]:
```
array(['apple', 'orange'], dtype=object)
```

In [4]:
```
pd.value_counts(values)
```

Out[4]:
```
apple     6
orange    2
dtype: int64
```

In [5]:
```
values = pd.Series([0, 1, 0, 0] * 2)
dim = pd.Series(['apple', 'orange'])
values
```

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

In [6]:
```
dim
```

Out[6]:
```
0     apple
1    orange
dtype: object
```

In [7]:
```
dim.take(values)
```

Out[7]:
```
0     apple
1    orange
0     apple
0     apple
0     apple
1    orange
0     apple
0     apple
dtype: object
```

## Categorical Type in pandas

In [8]:
```
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[8]:

| | basket_id | fruit | count | weight |
|---|---|---|---|---|
| **0** | 0 | apple | 5 | 3.858058 |
| **1** | 1 | orange | 8 | 2.612708 |
| **2** | 2 | apple | 4 | 2.995627 |
| **3** | 3 | apple | 7 | 2.614279 |
| **4** | 4 | apple | 12 | 2.990859 |
| **5** | 5 | orange | 8 | 3.845227 |
| **6** | 6 | apple | 5 | 0.033553 |
| **7** | 7 | apple | 4 | 0.425778 |

In [9]:
```
fruit_cat = df['fruit'].astype('category')
fruit_cat
```

Out[9]:
```
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 [10]:
```
c = fruit_cat.values
type(c)
```

Out[10]:
```
pandas.core.arrays.categorical.Categorical
```

In [11]:
```
c.categories
```

Out[11]:
```
Index(['apple', 'orange'], dtype='object')
```

In [12]:
```
c.codes
```

Out[12]:
```
array([0, 1, 0, 0, 0, 1, 0, 0], dtype=int8)
```

In [13]:
```
df['fruit'] = df['fruit'].astype('category')
df.fruit
```

Out[13]:
```
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 [14]:
```
my_categories = pd.Categorical(['foo', 'bar', 'baz', 'foo', 'bar'])
my_categories
```

Out[14]:
```
['foo', 'bar', 'baz', 'foo', 'bar']
Categories (3, object): ['bar', 'baz', 'foo']
```

In [15]:
```
categories = ['foo', 'bar', 'baz']
codes = [0, 1, 2, 0, 0, 1]
my_cats_2 = pd.Categorical.from_codes(codes, categories)
my_cats_2
```

Out[15]:
```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo', 'bar', 'baz']
```

In [16]:
```
ordered_cat = pd.Categorical.from_codes(codes, categories,
                                        ordered=True)
ordered_cat
```

Out[16]:
```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

In [17]:
```
my_cats_2.as_ordered()
```

Out[17]:
```
['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
Categories (3, object): ['foo' < 'bar' < 'baz']
```

## Computations with Categoricals

In [18]:
```
np.random.seed(12345)
draws = np.random.randn(1000)
draws[:5]
```

Out[18]:
```
array([-0.2047,  0.4789, -0.5194, -0.5557,  1.9658])
```

In [19]:
```
bins = pd.qcut(draws, 4)
bins
```

Out[19]:
```
[(-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, right]): [(-2.9499999999999997, -0.684] < (-0.68
4, -0.0101] < (-0.0101, 0.63] < (0.63, 3.928]]
```

In [20]:
```
bins = pd.qcut(draws, 4, labels=['Q1', 'Q2', 'Q3', 'Q4'])
bins
```

Out[20]:
```
['Q2', 'Q3', 'Q2', 'Q2', 'Q4', ..., 'Q3', 'Q2', 'Q1', 'Q3', 'Q4']
Length: 1000
Categories (4, object): ['Q1' < 'Q2' < 'Q3' < 'Q4']
```

In [21]:
```
bins.codes[:10]
```

Out[21]:
```
array([1, 2, 1, 1, 3, 3, 2, 2, 3, 3], dtype=int8)
```

In [22]:
```
bins = pd.Series(bins, name='quartile')
results = (pd.Series(draws)
           .groupby(bins)
           .agg(['count', 'min', 'max'])
           .reset_index())
results
```

Out[22]:

| | 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 [23]:
```
results['quartile']
```

Out[23]:
```
0    Q1
1    Q2
2    Q3
3    Q4
Name: quartile, dtype: category
Categories (4, object): ['Q1' < 'Q2' < 'Q3' < 'Q4']
```

### Better performance with categoricals

In [24]:
```
N = 10000000
draws = pd.Series(np.random.randn(N))
labels = pd.Series(['foo', 'bar', 'baz', 'qux'] * (N // 4))
```

In [25]:
```
categories = labels.astype('category')
```

In [26]:
```
labels.memory_usage()
```

Out[26]:
```
80000128
```

In [27]:
```
categories.memory_usage()
```

Out[27]:
```
10000332
```

In [28]:
```
%time _ = labels.astype('category')
```

```
Wall time: 851 ms
```

## Categorical Methods

In [29]:
```
s = pd.Series(['a', 'b', 'c', 'd'] * 2)
s
```

Out[29]:
```
0    a
1    b
2    c
3    d
4    a
5    b
6    c
7    d
dtype: object
```

In [30]:
```
cat_s = s.astype('category')
cat_s
```

Out[30]:
```
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 [31]:
```
cat_s.cat.codes
```

Out[31]:
```
0    0
1    1
2    2
3    3
4    0
5    1
6    2
7    3
dtype: int8
```

In [32]:
```
cat_s.cat.categories
```

Out[32]:
```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

In [33]:
```
actual_categories = ['a', 'b', 'c', 'd', 'e']
cat_s2 = cat_s.cat.set_categories(actual_categories)
cat_s2
```

Out[33]:
```
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 [34]:
```
cat_s.value_counts()
```

Out[34]:
```
a    2
b    2
c    2
d    2
dtype: int64
```

In [35]:
```
cat_s2.value_counts()
```

Out[35]:
```
a    2
b    2
c    2
d    2
e    0
dtype: int64
```

In [36]:
```
cat_s3 = cat_s[cat_s.isin(['a', 'b'])]
cat_s3
```

Out[36]:
```
0    a
1    b
4    a
5    b
dtype: category
Categories (4, object): ['a', 'b', 'c', 'd']
```

In [37]:
```
cat_s3.cat.remove_unused_categories()
```

Out[37]:
```
0    a
1    b
4    a
5    b
dtype: category
Categories (2, object): ['a', 'b']
```

## Creating dummy variables for modeling

In [41]:
```
cat_s = pd.Series(['a', 'b', 'c', 'd'] * 2, dtype='category')
cat_s
```

Out[41]:
```
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 [42]:
```
pd.get_dummies(cat_s)
```

Out[42]:

| | 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 |

# Advanced GroupBy Use

## Group Transforms and "Unwrapped" GroupBys

In [38]:
```
df = pd.DataFrame({'key': ['a', 'b', 'c'] * 4,
                   'value': np.arange(12.)})
df
```

Out[38]:

| | 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 [43]:
```
g = df.groupby('key').value
g.mean()
```

Out[43]:
```
key
a    4.5
b    5.5
c    6.5
Name: value, dtype: float64
```

In [44]:
```
g.transform(lambda x: x.mean())
```

Out[44]:
```
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 [45]:
```
g.transform('mean')
```

Out[45]:
```
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 [46]:
```
g.transform('count')
```

Out[46]:
```
0     4
1     4
2     4
3     4
4     4
5     4
6     4
7     4
8     4
9     4
10    4
11    4
Name: value, dtype: int64
```

In [47]:
```
g.transform('sum')
```

Out[47]:
```
0     18.0
1     22.0
2     26.0
3     18.0
4     22.0
5     26.0
6     18.0
7     22.0
8     26.0
9     18.0
10    22.0
11    26.0
Name: value, dtype: float64
```

In [48]:
```
g.transform(lambda x: x * 2)
```

Out[48]:
```
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 [49]:
```
g.transform(lambda x: x.rank(ascending=False))
```

Out[49]:
```
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 [50]:
```
def normalize(x):
    return (x - x.mean()) / x.std()
```

In [51]:
```
g.transform(normalize)
```

Out[51]:
```
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 [52]:
```
g.apply(normalize)
```

Out[52]:
```
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 [53]:
```
g.transform('mean')
```

Out[53]:
```
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 [54]:
```
normalized = (df['value'] - g.transform('mean')) / g.transform('std')
normalized
```

Out[54]:
```
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
```

## Grouped Time Resampling

In [55]:
```
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[55]:

| | 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 [56]:
```
df.set_index('time').resample('5min').count()
```

Out[56]:

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

In [57]:
```
df2 = pd.DataFrame({'time': times.repeat(3),
                    'key': np.tile(['a', 'b', 'c'], N),
                    'value': np.arange(N * 3.)})
df2[:7]
```

Out[57]:

| | 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 [61]:
```
time_key = pd.Grouper(freq='5min')
```

In [62]:
```
resampled = (df2.set_index('time')
             .groupby(['key', time_key])
             .sum())
resampled
```

Out[62]:

| | | value |
|---|---|---|
| **key** | **time** | |
| **a** | **2017-05-20 00:00:00** | 30.0 |
| | **2017-05-20 00:05:00** | 105.0 |
| | **2017-05-20 00:10:00** | 180.0 |
| **b** | **2017-05-20 00:00:00** | 35.0 |
| | **2017-05-20 00:05:00** | 110.0 |
| | **2017-05-20 00:10:00** | 185.0 |
| **c** | **2017-05-20 00:00:00** | 40.0 |
| | **2017-05-20 00:05:00** | 115.0 |
| | **2017-05-20 00:10:00** | 190.0 |

In [63]:
```
resampled.reset_index()
```

Out[63]:

| | key | time | value |
|---|---|---|---|
| **0** | a | 2017-05-20 00:00:00 | 30.0 |
| **1** | a | 2017-05-20 00:05:00 | 105.0 |
| **2** | a | 2017-05-20 00:10:00 | 180.0 |
| **3** | b | 2017-05-20 00:00:00 | 35.0 |
| **4** | b | 2017-05-20 00:05:00 | 110.0 |
| **5** | b | 2017-05-20 00:10:00 | 185.0 |
| **6** | c | 2017-05-20 00:00:00 | 40.0 |
| **7** | c | 2017-05-20 00:05:00 | 115.0 |
| **8** | c | 2017-05-20 00:10:00 | 190.0 |

# Techniques for Method Chaining

In [65]:
```
"""python
df = load_data()
df2 = df[df['col2'] < 0]
df2['col1_demeaned'] = df2['col1'] - df2['col1'].mean()
result = df2.groupby('key').col1_demeaned.std()
"""
```

```
  File "C:\Users\Usuário\AppData\Local\Temp\ipykernel_2072\1158896120.py", line 1
    ```python
    ^
SyntaxError: invalid syntax
```
```

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

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

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

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

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

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

### The pipe Method

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

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

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

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

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

In [ ]:
```
pd.options.display.max_rows = PREVIOUS_MAX_ROWS
```

## Conclusion