Pandas Series

In [1]:

```
import pandas as pd
series = pd.Series([1,2,3,4,5])
print(series)
```

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

Creating a Series Using a Specified Index

In [2]:

```
series = pd.Series([1,2,3,4,5], index=['a','b','c','d','c'])  # note the du
print(series)
```

```
a    1
b    2
c    3
d    4
c    5
dtype: int64
```

Accessing Elements in a Series

In [3]:

```
print(series[2])          # 3
# same as
print(series.iloc[2])     # 3  - based on the position of the index
```

```
3
3
```

In [4]:

```
print(series['d'])        # 4
# same as
print(series.loc['d'])    # 4 - based on the label in the index
```

```
4
4
```

In [5]:

```
print(series['c'])         # more than 1 row has the index 'c'
```

```
c    3
c    5
dtype: int64
```

In [6]:

```
print(series[2:])          # returns a Series
print(series.iloc[2:])     # returns a Series
```

```
c    3
d    4
c    5
dtype: int64
c    3
d    4
c    5
dtype: int64
```

Specifying a Datetime Range as the Index of a Series

In [7]:

```
dates1 = pd.date_range('20190525', periods=12)
print(dates1)
```

```
DatetimeIndex(['2019-05-25', '2019-05-26', '2019-05-27', '2019-05-28',
               '2019-05-29', '2019-05-30', '2019-05-31', '2019-06-01',
               '2019-06-02', '2019-06-03', '2019-06-04', '2019-06-05'],
              dtype='datetime64[ns]', freq='D')
```

In [8]:

```
series = pd.Series([1,2,3,4,5,6,7,8,9,10,11,12])
series.index = dates1
print(series)
```

```
2019-05-25     1
2019-05-26     2
2019-05-27     3
2019-05-28     4
2019-05-29     5
2019-05-30     6
2019-05-31     7
2019-06-01     8
2019-06-02     9
2019-06-03    10
2019-06-04    11
2019-06-05    12
Freq: D, dtype: int64
```

Date Ranges

In [9]:

```
dates2 = pd.date_range('2019-05-01', periods=12, freq='M')
print(dates2)
```

```
DatetimeIndex(['2019-05-31', '2019-06-30', '2019-07-31', '2019-08-31',
               '2019-09-30', '2019-10-31', '2019-11-30', '2019-12-31',
               '2020-01-31', '2020-02-29', '2020-03-31', '2020-04-30'],
              dtype='datetime64[ns]', freq='M')
```

In [10]:

```
dates2 = pd.date_range('2019-05-01', periods=12, freq='MS')
print(dates2)
```

```
DatetimeIndex(['2019-05-01', '2019-06-01', '2019-07-01', '2019-08-01',
               '2019-09-01', '2019-10-01', '2019-11-01', '2019-12-01',
               '2020-01-01', '2020-02-01', '2020-03-01', '2020-04-01'],
              dtype='datetime64[ns]', freq='MS')
```

In [11]:

```
dates2 = pd.date_range('05-01-2019', periods=12, freq='MS')
print(dates2)
```

```
DatetimeIndex(['2019-05-01', '2019-06-01', '2019-07-01', '2019-08-01',
               '2019-09-01', '2019-10-01', '2019-11-01', '2019-12-01',
               '2020-01-01', '2020-02-01', '2020-03-01', '2020-04-01'],
              dtype='datetime64[ns]', freq='MS')
```

In [12]:

```
dates3 = pd.date_range('2019/05/17 09:00:00', periods=8, freq='H')
print(dates3)
```

```
DatetimeIndex(['2019-05-17 09:00:00', '2019-05-17 10:00:00',
               '2019-05-17 11:00:00', '2019-05-17 12:00:00',
               '2019-05-17 13:00:00', '2019-05-17 14:00:00',
               '2019-05-17 15:00:00', '2019-05-17 16:00:00'],
              dtype='datetime64[ns]', freq='H')
```

Pandas DataFrame

In [5]:

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5,4),
                  columns=list('ABCD'))
print(df)
```

```
          A         B         C         D
0  1.176839 -0.568883  0.175213 -0.702281
1 -1.719396  0.408782  0.856516 -0.480629
2  0.274366  0.868352 -0.404123 -1.682070
3 -1.965048  0.321428 -1.259396 -1.029488
4  0.802132  1.194193 -0.502426  0.393425
```

In [4]:

```
np.random.randn?
```

Specifying the Index in a DataFrame

In [14]:

```
df = pd.read_csv('data.csv')                    # load dataframe from CSV fi
days = pd.date_range('20190525', periods=10)
df.index = days
print(df)
```

```
                   A         B         C         D
2019-05-25  0.187497  1.122150 -0.988277 -1.985934
2019-05-26  0.360803 -0.562243 -0.340693 -0.986988
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-28 -0.279572 -0.702492  0.252265  0.958977
2019-05-29  0.537438 -1.737568  0.714727 -0.939288
2019-05-30  0.070011 -0.516443 -1.655689  0.246721
2019-05-31  0.001268  0.951517  2.107360 -0.108726
2019-06-01 -0.185258  0.856520 -0.686285  1.104195
2019-06-02  0.387023  1.706336 -2.452653  0.260466
2019-06-03 -1.054974  0.556775 -0.945219 -0.030295
```

In [15]:

```
print(df.index)
```

```
DatetimeIndex(['2019-05-25', '2019-05-26', '2019-05-27', '2019-05-28',
               '2019-05-29', '2019-05-30', '2019-05-31', '2019-06-01',
               '2019-06-02', '2019-06-03'],
              dtype='datetime64[ns]', freq='D')
```

In [16]:

```
print(df.values)
```

```
[[ 1.874970e-01  1.122150e+00 -9.882770e-01 -1.985934e+00]
 [ 3.608030e-01 -5.622430e-01 -3.406930e-01 -9.869880e-01]
 [-4.062700e-02  6.733300e-02 -4.529780e-01  6.862230e-01]
 [-2.795720e-01 -7.024920e-01  2.522650e-01  9.589770e-01]
 [ 5.374380e-01 -1.737568e+00  7.147270e-01 -9.392880e-01]
 [ 7.001100e-02 -5.164430e-01 -1.655689e+00  2.467210e-01]
 [ 1.268000e-03  9.515170e-01  2.107360e+00 -1.087260e-01]
 [-1.852580e-01  8.565200e-01 -6.862850e-01  1.104195e+00]
 [ 3.870230e-01  1.706336e+00 -2.452653e+00  2.604660e-01]
 [-1.054974e+00  5.567750e-01 -9.452190e-01 -3.029500e-02]]
```

```
In [17]: print(df.describe())
```

```
                A          B          C          D
count  10.000000  10.000000  10.000000  10.000000
mean   -0.001639   0.174188  -0.444744  -0.079465
std     0.451656   1.049677   1.267397   0.971164
min    -1.054974  -1.737568  -2.452653  -1.985934
25%    -0.149100  -0.550793  -0.977513  -0.731647
50%     0.035640   0.312054  -0.569632   0.108213
75%     0.317477   0.927768   0.104026   0.579784
max     0.537438   1.706336   2.107360   1.104195
```

```
In [18]: print(df.mean(0))    # 0 means compute the mean for each columns
```

```
A   -0.001639
B    0.174188
C   -0.444744
D   -0.079465
dtype: float64
```

```
In [19]: print(df.mean(1))   # 1 means compute the mean for each row
```

```
2019-05-25   -0.416141
2019-05-26   -0.382280
2019-05-27    0.064988
2019-05-28    0.057294
2019-05-29   -0.356173
2019-05-30   -0.463850
2019-05-31    0.737855
2019-06-01    0.272293
2019-06-02   -0.024707
2019-06-03   -0.368428
Freq: D, dtype: float64
```

Extracting from DataFrames

Selecting the First and Last Five Rows

```
In [20]: print(df.head())
```

```
                   A         B         C         D
2019-05-25  0.187497  1.122150 -0.988277 -1.985934
2019-05-26  0.360803 -0.562243 -0.340693 -0.986988
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-28 -0.279572 -0.702492  0.252265  0.958977
2019-05-29  0.537438 -1.737568  0.714727 -0.939288
```

```
In [21]: print(df.head(8))     # prints out the first 8 rows
```

```
                   A         B         C         D
2019-05-25  0.187497  1.122150 -0.988277 -1.985934
2019-05-26  0.360803 -0.562243 -0.340693 -0.986988
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-28 -0.279572 -0.702492  0.252265  0.958977
2019-05-29  0.537438 -1.737568  0.714727 -0.939288
2019-05-30  0.070011 -0.516443 -1.655689  0.246721
2019-05-31  0.001268  0.951517  2.107360 -0.108726
2019-06-01 -0.185258  0.856520 -0.686285  1.104195
```

```
In [22]: print(df.tail())
```

```
                   A         B         C         D
2019-05-30  0.070011 -0.516443 -1.655689  0.246721
2019-05-31  0.001268  0.951517  2.107360 -0.108726
2019-06-01 -0.185258  0.856520 -0.686285  1.104195
2019-06-02  0.387023  1.706336 -2.452653  0.260466
2019-06-03 -1.054974  0.556775 -0.945219 -0.030295
```

```
In [23]: print(df.tail(8))     # prints out the last 8 rows
```

```
                   A         B         C         D
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-28 -0.279572 -0.702492  0.252265  0.958977
2019-05-29  0.537438 -1.737568  0.714727 -0.939288
2019-05-30  0.070011 -0.516443 -1.655689  0.246721
2019-05-31  0.001268  0.951517  2.107360 -0.108726
2019-06-01 -0.185258  0.856520 -0.686285  1.104195
2019-06-02  0.387023  1.706336 -2.452653  0.260466
2019-06-03 -1.054974  0.556775 -0.945219 -0.030295
```

Selecting a Specific Column in a DataFrame

In [24]:
```
print(df['A'])
# same as
print(df.A)
```

```
2019-05-25    0.187497
2019-05-26    0.360803
2019-05-27   -0.040627
2019-05-28   -0.279572
2019-05-29    0.537438
2019-05-30    0.070011
2019-05-31    0.001268
2019-06-01   -0.185258
2019-06-02    0.387023
2019-06-03   -1.054974
Freq: D, Name: A, dtype: float64
2019-05-25    0.187497
2019-05-26    0.360803
2019-05-27   -0.040627
2019-05-28   -0.279572
2019-05-29    0.537438
2019-05-30    0.070011
2019-05-31    0.001268
2019-06-01   -0.185258
2019-06-02    0.387023
2019-06-03   -1.054974
Freq: D, Name: A, dtype: float64
```

In [25]:
```
print(df[['A', 'B']])
```

```
                   A         B
2019-05-25  0.187497  1.122150
2019-05-26  0.360803 -0.562243
2019-05-27 -0.040627  0.067333
2019-05-28 -0.279572 -0.702492
2019-05-29  0.537438 -1.737568
2019-05-30  0.070011 -0.516443
2019-05-31  0.001268  0.951517
2019-06-01 -0.185258  0.856520
2019-06-02  0.387023  1.706336
2019-06-03 -1.054974  0.556775
```

Slicing Based on Row Number

In [26]:
```
print(df[2:4])
```

```
                   A         B         C         D
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-28 -0.279572 -0.702492  0.252265  0.958977
```

```
In [27]: print(df.iloc[2:4])       # 2 rows
```

```
                   A         B         C         D
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-28 -0.279572 -0.702492  0.252265  0.958977
```

```
In [28]: print(df.iloc[2:5])       # 3 rows
```

```
                   A         B         C         D
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-28 -0.279572 -0.702492  0.252265  0.958977
2019-05-29  0.537438 -1.737568  0.714727 -0.939288
```

```
In [29]: print(df.iloc[[2,4]])     # 2 rows
```

```
                   A         B         C         D
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-29  0.537438 -1.737568  0.714727 -0.939288
```

```
In [30]: # print(df[[2,4]])    # error; need to use the iloc indexer
         print(df.iloc[2])     # prints out row number 2
```

```
A   -0.040627
B    0.067333
C   -0.452978
D    0.686223
Name: 2019-05-27 00:00:00, dtype: float64
```

Slicing Based on Row and Column Numbers

```
In [31]: print(df.iloc[2:4, 1:4])        # 2 rows, 3 columns
```

```
                   B         C         D
2019-05-27  0.067333 -0.452978  0.686223
2019-05-28 -0.702492  0.252265  0.958977
```

```
In [32]: print(df.iloc[[2,4], [1,3]])    # 2 rows, 2 columns
```

```
                   B         D
2019-05-27  0.067333  0.686223
2019-05-29 -1.737568 -0.939288
```

Slicing Based on Labels

In [33]:
```
print(df['20190601':'20190603'])
```

```
                   A         B         C         D
2019-06-01 -0.185258  0.856520 -0.686285  1.104195
2019-06-02  0.387023  1.706336 -2.452653  0.260466
2019-06-03 -1.054974  0.556775 -0.945219 -0.030295
```

In [34]:
```
print(df.loc['20190601':'20190603'])
```

```
                   A         B         C         D
2019-06-01 -0.185258  0.856520 -0.686285  1.104195
2019-06-02  0.387023  1.706336 -2.452653  0.260466
2019-06-03 -1.054974  0.556775 -0.945219 -0.030295
```

In [35]:
```
print(df.loc['20190601':'20190603', 'A':'C'])
```

```
                   A         B         C
2019-06-01 -0.185258  0.856520 -0.686285
2019-06-02  0.387023  1.706336 -2.452653
2019-06-03 -1.054974  0.556775 -0.945219
```

In [36]:
```
print(df.loc['20190601':'20190603', ['A','C']])
```

```
                   A         C
2019-06-01 -0.185258 -0.686285
2019-06-02  0.387023 -2.452653
2019-06-03 -1.054974 -0.945219
```

In [37]:
```
print(df.loc['20190601'])
```

```
A   -0.185258
B    0.856520
C   -0.686285
D    1.104195
Name: 2019-06-01 00:00:00, dtype: float64
```

In [38]:
```
# print(df.loc[['20190601','20190603']])   # KeyError
```

In [12]:

```python
from datetime import datetime
date1 = datetime(2019, 6, 1, 0, 0, 0)
date2 = datetime(2019, 6, 3, 0, 0, 0)
print(df.loc[[date1,date2]])
```

```
---------------------------------------------------------------------------
--
KeyError                                  Traceback (most recent call las
t)
<ipython-input-12-b7636d7fc292> in <module>
      2 date1 = datetime(2019, 6, 1, 0, 0, 0)
      3 date2 = datetime(2019, 6, 3, 0, 0, 0)
----> 4 print(df.loc[[date1,date2]])

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py in __geti
tem__(self, key)
   1498 
   1499             maybe_callable = com.apply_if_callable(key, self.obj)
-> 1500             return self._getitem_axis(maybe_callable, axis=axis)
   1501 
   1502     def _is_scalar_access(self, key):

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py in _getit
em_axis(self, key, axis)
   1900                     raise ValueError('Cannot index with multidime
nsional key')
   1901 
-> 1902                 return self._getitem_iterable(key, axis=axis)
   1903 
   1904             # nested tuple slicing

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py in _getit
em_iterable(self, key, axis)
   1203             # A collection of keys
   1204             keyarr, indexer = self._get_listlike_indexer(key, axi
s,
-> 1205                                                          raise_mi
ssing=False)
   1206             return self.obj._reindex_with_indexers({axis: [keyar
r, indexer]},
   1207                                                    copy=True, all
ow_dups=True)

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py in _get_l
istlike_indexer(self, key, axis, raise_missing)
   1159         self._validate_read_indexer(keyarr, indexer,
   1160                                     o._get_axis_number(axis),
-> 1161                                     raise_missing=raise_missing)
   1162         return keyarr, indexer
   1163 

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py in _valid
ate_read_indexer(self, key, indexer, axis, raise_missing)
   1244                 raise KeyError(
   1245                     u"None of [{key}] are in the [{axis}]".format
(
-> 1246                                key=key, axis=self.obj._get_axis_name(axi
s)))
   1247
   1248                # We (temporarily) allow for some missing keys with .
loc, except in

KeyError: "None of [DatetimeIndex(['2019-06-01', '2019-06-03'], dtype='da
tetime64[ns]', freq=None)] are in the [index]"
```

In [40]:
```
print(df.loc[date1, ['A','C']])
```

```
A   -0.185258
C   -0.686285
Name: 2019-06-01 00:00:00, dtype: float64
```

Selecting a Single Cell in a DataFrame

In [14]:
```
from datetime import datetime
d = datetime(2019, 6, 3, 0, 0, 0)
print(df.at[d,'B'])
```

```
---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
<ipython-input-14-2dfeb65fb828> in <module>
      1 from datetime import datetime
      2 d = datetime(2019, 6, 3, 0, 0, 0)
----> 3 print(df.at[d,'B'])

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py in __geti
tem__(self, key)
   2267                 raise ValueError('Invalid call for scalar access
 (getting)!')
   2268 
-> 2269         key = self._convert_key(key)
   2270         return self.obj._get_value(*key, takeable=self._takeable)
   2271 

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py in _conve
rt_key(self, key, is_setter)
   2349             if ax.is_integer():
   2350                 if not is_integer(i):
-> 2351                     raise ValueError("At based indexing on an int
eger index "
   2352                                      "can only have integer index
ers")
   2353             else:

ValueError: At based indexing on an integer index can only have integer i
ndexers
```

Selecting Based on Cell Value

In [42]:
```
print(df[(df.A > 0) & (df.B>0)])
```

```
                   A         B         C         D
2019-05-25  0.187497  1.122150 -0.988277 -1.985934
2019-05-31  0.001268  0.951517  2.107360 -0.108726
2019-06-02  0.387023  1.706336 -2.452653  0.260466
```

Transforming DataFrames

In [43]:
```
print(df.transpose())
```

```
   2019-05-25  2019-05-26  2019-05-27  2019-05-28  2019-05-29  2019-05-30
\
A    0.187497    0.360803   -0.040627   -0.279572    0.537438    0.070011
B    1.122150   -0.562243    0.067333   -0.702492   -1.737568   -0.516443
C   -0.988277   -0.340693   -0.452978    0.252265    0.714727   -1.655689
D   -1.985934   -0.986988    0.686223    0.958977   -0.939288    0.246721

   2019-05-31  2019-06-01  2019-06-02  2019-06-03
A    0.001268   -0.185258    0.387023   -1.054974
B    0.951517    0.856520    1.706336    0.556775
C    2.107360   -0.686285   -2.452653   -0.945219
D   -0.108726    1.104195    0.260466   -0.030295
```

In [44]:
```
print(df.T)
```

```
   2019-05-25  2019-05-26  2019-05-27  2019-05-28  2019-05-29  2019-05-30
\
A    0.187497    0.360803   -0.040627   -0.279572    0.537438    0.070011
B    1.122150   -0.562243    0.067333   -0.702492   -1.737568   -0.516443
C   -0.988277   -0.340693   -0.452978    0.252265    0.714727   -1.655689
D   -1.985934   -0.986988    0.686223    0.958977   -0.939288    0.246721

   2019-05-31  2019-06-01  2019-06-02  2019-06-03
A    0.001268   -0.185258    0.387023   -1.054974
B    0.951517    0.856520    1.706336    0.556775
C    2.107360   -0.686285   -2.452653   -0.945219
D   -0.108726    1.104195    0.260466   -0.030295
```

In [45]:
```
def checkSeriesOrDataframe(var):
    if isinstance(var, pd.DataFrame):
        return 'Dataframe'
    if isinstance(var, pd.Series):
        return 'Series'
```

Sorting Data in a DataFrame

Sorting by Index

```
In [46]: print(df.sort_index(axis=0, ascending=False))  # axis = 0 means sort by
                                                          # index
```

```
                   A         B         C         D
2019-06-03 -1.054974  0.556775 -0.945219 -0.030295
2019-06-02  0.387023  1.706336 -2.452653  0.260466
2019-06-01 -0.185258  0.856520 -0.686285  1.104195
2019-05-31  0.001268  0.951517  2.107360 -0.108726
2019-05-30  0.070011 -0.516443 -1.655689  0.246721
2019-05-29  0.537438 -1.737568  0.714727 -0.939288
2019-05-28 -0.279572 -0.702492  0.252265  0.958977
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-26  0.360803 -0.562243 -0.340693 -0.986988
2019-05-25  0.187497  1.122150 -0.988277 -1.985934
```

```
In [47]: print(df.sort_index(axis=1, ascending=False))  # axis = 1 means sort by
                                                          # column
```

```
                   D         C         B         A
2019-05-25 -1.985934 -0.988277  1.122150  0.187497
2019-05-26 -0.986988 -0.340693 -0.562243  0.360803
2019-05-27  0.686223 -0.452978  0.067333 -0.040627
2019-05-28  0.958977  0.252265 -0.702492 -0.279572
2019-05-29 -0.939288  0.714727 -1.737568  0.537438
2019-05-30  0.246721 -1.655689 -0.516443  0.070011
2019-05-31 -0.108726  2.107360  0.951517  0.001268
2019-06-01  1.104195 -0.686285  0.856520 -0.185258
2019-06-02  0.260466 -2.452653  1.706336  0.387023
2019-06-03 -0.030295 -0.945219  0.556775 -1.054974
```

Sorting by Value

```
In [48]: print(df.sort_values('A', axis=0))
```

```
                   A         B         C         D
2019-06-03 -1.054974  0.556775 -0.945219 -0.030295
2019-05-28 -0.279572 -0.702492  0.252265  0.958977
2019-06-01 -0.185258  0.856520 -0.686285  1.104195
2019-05-27 -0.040627  0.067333 -0.452978  0.686223
2019-05-31  0.001268  0.951517  2.107360 -0.108726
2019-05-30  0.070011 -0.516443 -1.655689  0.246721
2019-05-25  0.187497  1.122150 -0.988277 -1.985934
2019-05-26  0.360803 -0.562243 -0.340693 -0.986988
2019-06-02  0.387023  1.706336 -2.452653  0.260466
2019-05-29  0.537438 -1.737568  0.714727 -0.939288
```

```
In [49]: print(df.sort_values('20190601', axis=1))
```

```
                   C         A         B         D
2019-05-25 -0.988277  0.187497  1.122150 -1.985934
2019-05-26 -0.340693  0.360803 -0.562243 -0.986988
2019-05-27 -0.452978 -0.040627  0.067333  0.686223
2019-05-28  0.252265 -0.279572 -0.702492  0.958977
2019-05-29  0.714727  0.537438 -1.737568 -0.939288
2019-05-30 -1.655689  0.070011 -0.516443  0.246721
2019-05-31  2.107360  0.001268  0.951517 -0.108726
2019-06-01 -0.686285 -0.185258  0.856520  1.104195
2019-06-02 -2.452653  0.387023  1.706336  0.260466
2019-06-03 -0.945219 -1.054974  0.556775 -0.030295
```

Applying Functions to a DataFrame

```
In [50]: import math
         sq_root = lambda x: math.sqrt(x) if x > 0 else x
         sq      = lambda x: x**2
```

```
In [51]: print(df.B.apply(sq_root))
```

```
2019-05-25    1.059316
2019-05-26   -0.562243
2019-05-27    0.259486
2019-05-28   -0.702492
2019-05-29   -1.737568
2019-05-30   -0.516443
2019-05-31    0.975457
2019-06-01    0.925484
2019-06-02    1.306268
2019-06-03    0.746174
Freq: D, Name: B, dtype: float64
```

```
In [52]: print(df.B.apply(sq))
```

```
2019-05-25    1.259221
2019-05-26    0.316117
2019-05-27    0.004534
2019-05-28    0.493495
2019-05-29    3.019143
2019-05-30    0.266713
2019-05-31    0.905385
2019-06-01    0.733627
2019-06-02    2.911583
2019-06-03    0.309998
Freq: D, Name: B, dtype: float64
```

In [53]:
```
# df.apply(sq_root)    # ValueError
```

In [54]:
```
df.apply(sq)
```

Out[54]:

	A	B	C	D
2019-05-25	0.035155	1.259221	0.976691	3.943934
2019-05-26	0.130179	0.316117	0.116072	0.974145
2019-05-27	0.001651	0.004534	0.205189	0.470902
2019-05-28	0.078161	0.493495	0.063638	0.919637
2019-05-29	0.288840	3.019143	0.510835	0.882262
2019-05-30	0.004902	0.266713	2.741306	0.060871
2019-05-31	0.000002	0.905385	4.440966	0.011821
2019-06-01	0.034321	0.733627	0.470987	1.219247
2019-06-02	0.149787	2.911583	6.015507	0.067843
2019-06-03	1.112970	0.309998	0.893439	0.000918

In [55]:
```
for column in df:
    df[column] = df[column].apply(sq_root)
print(df)
```

```
                   A         B         C         D
2019-05-25  0.433009  1.059316 -0.988277 -1.985934
2019-05-26  0.600669 -0.562243 -0.340693 -0.986988
2019-05-27 -0.040627  0.259486 -0.452978  0.828386
2019-05-28 -0.279572 -0.702492  0.502260  0.979274
2019-05-29  0.733102 -1.737568  0.845415 -0.939288
2019-05-30  0.264596 -0.516443 -1.655689  0.496710
2019-05-31  0.035609  0.975457  1.451675 -0.108726
2019-06-01 -0.185258  0.925484 -0.686285  1.050807
2019-06-02  0.622112  1.306268 -2.452653  0.510359
2019-06-03 -1.054974  0.746174 -0.945219 -0.030295
```

In [56]:
```
print(df.apply(np.sum, axis=0))
```

```
A    1.128665
B    1.753438
C   -4.722444
D   -0.185696
dtype: float64
```

In [57]:

```
print(df.apply(np.sum, axis=1))
```

```
2019-05-25   -1.481886
2019-05-26   -1.289255
2019-05-27    0.594267
2019-05-28    0.499470
2019-05-29   -1.098339
2019-05-30   -1.410826
2019-05-31    2.354015
2019-06-01    1.104747
2019-06-02   -0.013915
2019-06-03   -1.284314
Freq: D, dtype: float64
```

Adding and Removing Rows and Columns in a DataFrame

In [58]:

```
import pandas as pd

data = {'name': ['Janet', 'Nad', 'Timothy', 'June', 'Amy'],
        'year': [2012, 2012, 2013, 2014, 2014],
        'reports': [6, 13, 14, 1, 7]}

df = pd.DataFrame(data, index =
       ['Singapore', 'China', 'Japan', 'Sweden', 'Norway'])
print(df)
```

```
              name  year  reports
Singapore    Janet  2012        6
China          Nad  2012       13
Japan      Timothy  2013       14
Sweden        June  2014        1
Norway         Amy  2014        7
```

Adding a Column

In [59]:

```
import numpy as np

schools = np.array(["Cambridge","Oxford","Oxford","Cambridge","Oxford"])
df["school"] = schools
print(df)
```

```
              name  year  reports     school
Singapore    Janet  2012        6  Cambridge
China          Nad  2012       13     Oxford
Japan      Timothy  2013       14     Oxford
Sweden        June  2014        1  Cambridge
Norway         Amy  2014        7     Oxford
```

Removing Rows

In [60]:

```
print(df.drop(['China', 'Japan']))  # drop rows based on value of index
```

```
             name  year  reports     school
Singapore   Janet  2012        6  Cambridge
Sweden       June  2014        1  Cambridge
Norway        Amy  2014        7     Oxford
```

In [61]:

```
print(df[df.name != 'Nad'])          # drop row based on column value
```

```
               name  year  reports     school
Singapore     Janet  2012        6  Cambridge
Japan       Timothy  2013       14     Oxford
Sweden         June  2014        1  Cambridge
Norway          Amy  2014        7     Oxford
```

In [62]:

```
print(df.drop(df.index[1]))
# same as df.drop['China']
```

```
               name  year  reports     school
Singapore     Janet  2012        6  Cambridge
Japan       Timothy  2013       14     Oxford
Sweden         June  2014        1  Cambridge
Norway          Amy  2014        7     Oxford
```

In [63]:

```
print(df.drop(df.index[[1,2]]))      # remove the second and third row
```

```
             name  year  reports     school
Singapore   Janet  2012        6  Cambridge
Sweden       June  2014        1  Cambridge
Norway        Amy  2014        7     Oxford
```

In [64]:

```
print(df.drop(df.index[-2]))         # remove second last row
```

```
               name  year  reports     school
Singapore     Janet  2012        6  Cambridge
China           Nad  2012       13     Oxford
Japan       Timothy  2013       14     Oxford
Norway          Amy  2014        7     Oxford
```

Removing Columns

In [65]:

```
print(df.drop('reports', axis=1))    # drop column
```

```
                name  year     school
Singapore      Janet  2012  Cambridge
China            Nad  2012     Oxford
Japan        Timothy  2013     Oxford
Sweden          June  2014  Cambridge
Norway           Amy  2014     Oxford
```

In [66]:

```
print(df.drop(df.columns[1], axis=1))       # drop using columns number
```

```
                name  reports     school
Singapore      Janet        6  Cambridge
China            Nad       13     Oxford
Japan        Timothy       14     Oxford
Sweden          June        1  Cambridge
Norway           Amy        7     Oxford
```

In [67]:

```
print(df.drop(df.columns[[1,3]], axis=1))   # drop multiple columns
```

```
                name  reports
Singapore      Janet        6
China            Nad       13
Japan        Timothy       14
Sweden          June        1
Norway           Amy        7
```

Generating a Crosstab

In [68]:

```
df = pd.DataFrame(
    {
        "Gender": ['Male','Male','Female','Female','Female'],
        "Team"  : [1,2,3,3,1]
    })
print(df)
```

```
   Gender  Team
0    Male     1
1    Male     2
2  Female     3
3  Female     3
4  Female     1
```

In [69]:
```
print("Displaying the distribution of genders in each team")
print(pd.crosstab(df.Gender, df.Team))
```

```
Displaying the distribution of genders in each team
Team    1  2  3
Gender
Female  1  0  2
Male    1  1  0
```

In [70]:
```
print(pd.crosstab(df.Team, df.Gender))
```

```
Gender  Female  Male
Team
1            1     1
2            0     1
3            2     0
```

In [ ]: