_pandas

August 27, 2025

[265]:
```
import pandas as pd
import numpy as np
```

1 Data structures : Series

[266]:
```
s = pd.Series([1, 3, 5, np.nan, 6, 8])
print(s)
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

[267]:
```
s = pd.Series(data = np.arange(4),
              index = [f"row_{i}" for i in range(4)], # By default, `index =␣
 ↪0,1,2,3...`
              name = "col", #    `name = None`
              dtype = "float32")
print(s)
```

```
row_0    0.0
row_1    1.0
row_2    2.0
row_3    3.0
Name: col, dtype: float32
```

2 Data structures : DataFrame

[268]:
```
df = pd.DataFrame([[1,2,3],
                   [4,5,6],
                   [7,8,9],
                   [10,11,12]])
print(df)
```

```
    0   1   2
0   1   2   3
1   4   5   6
2   7   8   9
3  10  11  12
```

[269]:
```
df = pd.DataFrame(np.arange(1, 13).reshape(4, 3))
print(df)
print(df.to_numpy())
```

```
    0   1   2
0   1   2   3
1   4   5   6
2   7   8   9
3  10  11  12
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

[270]:
```
df = pd.DataFrame(np.arange(1, 13).reshape(4, 3),
                  index = [f'row_{i}' for i in range(4)],
                  columns = [f'col_{i}' for i in range(3)])
print(df.columns)
print(df.index)
print(df)
```

```
Index(['col_0', 'col_1', 'col_2'], dtype='object')
Index(['row_0', 'row_1', 'row_2', 'row_3'], dtype='object')
       col_0  col_1  col_2
row_0      1      2      3
row_1      4      5      6
row_2      7      8      9
row_3     10     11     12
```

[271]:
```
df = df.rename(columns={"col_0": "_col0",
                        "col_1": "_col1",})
print(df)
```

```
       _col0  _col1  col_2
row_0      1      2      3
row_1      4      5      6
row_2      7      8      9
row_3     10     11     12
```

[272]:
```
df = pd.DataFrame(
    {
        "col_0": [1, 4, 7, 10],
        "col_1": [2, 5, 8, 11],
```

```
        "col_2": [3, 6, 9, 12],
    },
    index = [f'row_{i}' for i in range(4)]
)
print(df)
```

```
       col_0  col_1  col_2
row_0      1      2      3
row_1      4      5      6
row_2      7      8      9
row_3     10     11     12
```

3 dtype

[273]:
```
print(pd.date_range("2000-2-28", freq="D", periods=3))
```

```
DatetimeIndex(['2000-02-28', '2000-02-29', '2000-03-01'],
dtype='datetime64[ns]', freq='D')
```

[274]:
```
data = {
    "date": ["2023-01-15", "2023-02-20", "2023-02-25", "2023-03-05"],
    "value": [10, 20, 15, 30]
}
df = pd.DataFrame(data)
df["date"] = pd.to_datetime(df["date"])
df["month"] = df["date"].dt.month
df["weekday"] = df["date"].dt.weekday
print(df)
```

```
        date  value  month  weekday
0 2023-01-15     10      1        6
1 2023-02-20     20      2        0
2 2023-02-25     15      2        5
3 2023-03-05     30      3        6
```

[275]:
```
df = pd.DataFrame({
    "name": ["Alice", "Bob", "Cathy", "David", "Eva"],
    "grade": ["A", "B", "A", "C", "B"]
})

df["grade"] = df["grade"].astype("category")
```

4 Getitem

[276]:
```
df = pd.DataFrame(np.arange(1, 13).reshape(4, 3),
                  index = [f'row_{i}' for i in range(4)],
                  columns = [f'col_{i}' for i in range(3)])
print(df["col_2"])   # return `pd.Series`
print(df[["col_2"]]) # return `pd.DataFrame`
```

```
row_0     3
row_1     6
row_2     9
row_3    12
Name: col_2, dtype: int64
       col_2
row_0      3
row_1      6
row_2      9
row_3     12
```

[277]:
```
print(df.loc["row_1"])
```

```
col_0    4
col_1    5
col_2    6
Name: row_1, dtype: int64
```

[278]:
```
# Selecting rows
print(df.head(2))
print(df.tail(3))
```

```
       col_0  col_1  col_2
row_0      1      2      3
row_1      4      5      6
       col_0  col_1  col_2
row_1      4      5      6
row_2      7      8      9
row_3     10     11     12
```

[279]:
```
# Selecting rows
print(df[1:3])
print(df["row_1":"row_3"]) # close segment
```

```
       col_0  col_1  col_2
row_1      4      5      6
row_2      7      8      9
       col_0  col_1  col_2
row_1      4      5      6
row_2      7      8      9
row_3     10     11     12
```

[280]:
```
print(df[["col_0", "col_2"]])
```

```
       col_0  col_2
row_0      1      3
row_1      4      6
row_2      7      9
row_3     10     12
```

5 Selection by label and index

[281]:
```
print(df.loc[:, ["col_0", "col_2"]])
print(df.iloc[:, [0,2]])
```

```
       col_0  col_2
row_0      1      3
row_1      4      6
row_2      7      9
row_3     10     12
       col_0  col_2
row_0      1      3
row_1      4      6
row_2      7      9
row_3     10     12
```

[282]:
```
print(df.loc["row_0":"row_1", "col_0":"col_1"])
print(df.iloc[0:2, 0:2])
```

```
       col_0  col_1
row_0      1      2
row_1      4      5
       col_0  col_1
row_0      1      2
row_1      4      5
```

[283]:
```
print(df.at["row_1", "col_1"])
print(df.iat[1, 1])
```

```
5
5
```

6 `df.drop()` to delete columns

[284]:
```
print(df)
df.drop(columns=["col_1"], inplace=True)
print(df)
```

```
       col_0  col_1  col_2
row_0      1      2      3
```

```
row_1      4      5      6
row_2      7      8      9
row_3     10     11     12
       col_0  col_2
row_0      1      3
row_1      4      6
row_2      7      9
row_3     10     12
```

7 pd.concat , pd.merge and pd.join

All three return a new pd.DataFrame.

[285]:
```
df1 = pd.DataFrame(
    {
        "A": ["A0", "A1"],
        "B": ["B0", "B1"],
    },
    index=[0, 1],
)

df2 = pd.DataFrame(
    {
        "C": ["C0", "C1"],
        "D": ["D0", "D1"],
    },
    index=[0, 1],
)

print(pd.concat([df1, df2])) # By default, `axis=0`.
print(pd.concat([df1, df2], axis = 1))
```

```
     A    B    C    D
0   A0   B0  NaN  NaN
1   A1   B1  NaN  NaN
0  NaN  NaN   C0   D0
1  NaN  NaN   C1   D1
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
```

[286]:
```
left = pd.DataFrame({"key": ["foo", "baz", "bar"],
                     "lval": [1, 2, 3]})
right = pd.DataFrame({"key": ["foo", "bar", "car", "quz"],
                      "rval": [4, 5, 6, 7]})
print(pd.merge(left, right, on="key")) # By default, how="inner", which keeps␣
 ↪only the intersection of the two keys.
```

```
print(pd.merge(left, right, on="key", how="outer")) # how="outer" means taking␣
↪the union of the two keys.
```

```
   key  lval  rval
0  foo     1     4
1  bar     3     5
   key  lval  rval
0  bar   3.0   5.0
1  baz   2.0   NaN
2  car   NaN   6.0
3  foo   1.0   4.0
4  quz   NaN   7.0
```

[287]:
```
print(pd.merge(left, right, on="key", how="left"))
print(pd.merge(left, right, on="key", how="right"))
```

```
   key  lval  rval
0  foo     1   4.0
1  baz     2   NaN
2  bar     3   5.0
   key  lval  rval
0  foo   1.0     4
1  bar   3.0     5
2  car   NaN     6
3  quz   NaN     7
```

[288]:
```
left = pd.DataFrame({"A": ["A0", "A1", "A2"],
                     "B": ["B0", "B1", "B2"]},
                    index=["K0", "K1", "K2"])

right = pd.DataFrame({"C": ["C0", "C2", "C3"],
                      "D": ["D0", "D2", "D3"]},
                     index=["K0", "K2", "K3"])

print(pd.concat([left, right], axis = 1)) # Keep both indexes
print(left.join(right)) # Merge on index, keeping only the **left** DataFrame's␣
↪index.
```

```
      A    B    C    D
K0   A0   B0   C0   D0
K1   A1   B1  NaN  NaN
K2   A2   B2   C2   D2
K3  NaN  NaN   C3   D3
     A   B    C    D
K0  A0  B0   C0   D0
K1  A1  B1  NaN  NaN
K2  A2  B2   C2   D2
```

8 MultiIndex

[289]:
```
arrays = [
    ["bar", "bar", "bar", "baz", "foo", "foo", "qux", "qux"],
    ["one", "two", "three", "one", "one", "two", "one", "two"],
]
df = pd.DataFrame(np.arange(8),
                  index = pd.MultiIndex.from_arrays(arrays, names=["first",␣
 ↪"second"]))
print(df)
```

```
              0
first second
bar   one     0
      two     1
      three   2
baz   one     3
foo   one     4
      two     5
qux   one     6
      two     7
```

[290]:
```
iterables = [["bar", "baz", "foo", "qux"], ["one", "two"]]
df = pd.DataFrame({0 : np.arange(8),
                   1 : np.arange(8),
                   2 : np.arange(8)},
                  index = pd.MultiIndex.from_product(iterables, names=["first",␣
 ↪"second"]))
print(df)
```

```
              0  1  2
first second
bar   one     0  0  0
      two     1  1  1
baz   one     2  2  2
      two     3  3  3
foo   one     4  4  4
      two     5  5  5
qux   one     6  6  6
      two     7  7  7
```

[291]:
```
dft = df.T
print(dft)
print(dft["bar"])
```

```
first  bar     baz     foo     qux
second one two one two one two one two
0        0   1   2   3   4   5   6   7
1        0   1   2   3   4   5   6   7
```

```
2        0   1   2   3   4   5   6   7
second  one  two
0         0    1
1         0    1
2         0    1
```

[292]:
```
print(df.loc["bar"])
print(df.loc["bar" : "baz", 1:3])
```

```
        0  1  2
second
one     0  0  0
two     1  1  1
              1  2
first second
bar   one     0  0
      two     1  1
baz   one     2  2
      two     3  3
```

[293]:
```
def mklbl(prefix, n):
    return [f"{prefix}{i}" for i in range(n)]

miindex = pd.MultiIndex.from_product(
    [mklbl("A", 2), mklbl("B", 2), mklbl("C", 3)]
)

micolumns = pd.MultiIndex.from_tuples(
    [("a", "foo"), ("a", "bar"), ("b", "foo"), ("b", "bah")], names=["lvl0",␣
 ↪"lvl1"]
)

dfmi = (
    pd.DataFrame(
        np.arange(len(miindex) * len(micolumns)).reshape(
            (len(miindex), len(micolumns))
        ),
        index=miindex,
        columns=micolumns,
    )
)
print(dfmi)
```

```
lvl0       a       b
lvl1     foo bar foo bah
A0 B0 C0   0   1   2   3
      C1   4   5   6   7
      C2   8   9  10  11
   B1 C0  12  13  14  15
```

```
      C1  16  17  18  19
      C2  20  21  22  23
A1 B0 C0  24  25  26  27
      C1  28  29  30  31
      C2  32  33  34  35
   B1 C0  36  37  38  39
      C1  40  41  42  43
      C2  44  45  46  47
```

[294]:
```
print(dfmi.loc[(slice("A1"), slice(None), ["C0", "C2"]), :])
```

```
lvl0       a       b
lvl1     foo bar foo bah
A0 B0 C0   0   1   2   3
      C2   8   9  10  11
   B1 C0  12  13  14  15
      C2  20  21  22  23
A1 B0 C0  24  25  26  27
      C2  32  33  34  35
   B1 C0  36  37  38  39
      C2  44  45  46  47
```

9 pd.stack and pd.unstack

[295]:
```
print(dfmi.unstack()) # Removing the last level is equivalent to `dfmi.
  ↪unstack(2)`.
```

```
lvl0    a                       b
lvl1  foo         bar         foo         bah
       C0  C1  C2  C0  C1  C2  C0  C1  C2  C0  C1  C2
A0 B0   0   4   8   1   5   9   2   6  10   3   7  11
   B1  12  16  20  13  17  21  14  18  22  15  19  23
A1 B0  24  28  32  25  29  33  26  30  34  27  31  35
   B1  36  40  44  37  41  45  38  42  46  39  43  47
```

[296]:
```
print(dfmi.unstack(1))
```

```
lvl0    a               b
lvl1  foo     bar     foo     bah
       B0  B1  B0  B1  B0  B1  B0  B1
A0 C0   0  12   1  13   2  14   3  15
   C1   4  16   5  17   6  18   7  19
   C2   8  20   9  21  10  22  11  23
A1 C0  24  36  25  37  26  38  27  39
   C1  28  40  29  41  30  42  31  43
   C2  32  44  33  45  34  46  35  47
```

[297]:
```
print(dfmi.unstack([0,2]))
```

```
lvl0   a                                           …   b                        \
lvl1 foo                        bar                …  foo             bah        
      A0           A1           A0           A1    …   A0  A1           A0        
      C0  C1  C2  C0  C1  C2  C0  C1  C2  C0       …   C2  C0  C1  C2  C0  C1  C2
B0     0   4   8  24  28  32   1   5   9  25       …   10  26  30  34   3   7  11
B1    12  16  20  36  40  44  13  17  21  37       …   22  38  42  46  15  19  23

lvl0
lvl1
      A1
      C0  C1  C2
B0    27  31  35
B1    39  43  47

[2 rows x 24 columns]
```

10 pd.set_index and inplace=True

[298]:
```python
df = pd.DataFrame({"A": np.arange(8),
                   "B": ["bar", "bar", "bar", "baz", "foo", "foo", "qux",␣
 ↪"qux"],
                   "C": ["one", "two", "three", "one", "one", "two", "one",␣
 ↪"two"]})
df1 = df.set_index("B") # By default, `inplace=False`, so the operation returns␣
 ↪a new `pd.DataFrame`.
print(df1)
```

```
     A      C
B
bar  0    one
bar  1    two
bar  2  three
baz  3    one
foo  4    one
foo  5    two
qux  6    one
qux  7    two
```

[299]:
```python
df = pd.DataFrame({"A": np.arange(8),
                   "B": ["bar", "bar", "bar", "baz", "foo", "foo", "qux",␣
 ↪"qux"],
                   "C": ["one", "two", "three", "one", "one", "two", "one",␣
 ↪"two"]})
df.set_index(["B", "C"], inplace=True) # `inplace=True` means directly␣
 ↪modifying the `df` itself.
print(df)
```

```
            A
B   C
bar one     0
    two     1
    three   2
baz one     3
foo one     4
    two     5
qux one     6
    two     7
```

[300]:
```
df.reset_index(["B"], inplace=True)
print(df)
```

```
          B  A
C
one     bar  0
two     bar  1
three   bar  2
one     baz  3
one     foo  4
two     foo  5
one     qux  6
two     qux  7
```

11 pd.groupby and pd.apply

[301]:
```
df = pd.DataFrame(
    {
        "A": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "B": [1, 2, 1, 2, 3, 2, 1, 1],
        "C": np.random.randint(0, 10, 8)
    }
)
print(df)
```

```
       A  B  C
0    one  1  5
1    one  2  7
2    two  1  4
3  three  2  3
4    two  3  4
5    two  2  1
6    one  1  1
7  three  1  0
```

[302]:
```
print(df.groupby("A").mean())
```

```
              B         C
A
one    1.333333  4.333333
three  1.500000  1.500000
two    2.000000  3.000000
```

[303]:
```
def f(x):
    return pd.Series([x, x ** 2], index=["x", "x^2"])
s = pd.Series(np.random.randint(0, 30, 5))
print(s)
print(s.apply(f))
```

```
0     0
1     2
2     7
3     8
4    10
dtype: int32
    x  x^2
0   0    0
1   2    4
2   7   49
3   8   64
4  10  100
```

[304]:
```
def f(group : pd.DataFrame)-> pd.DataFrame:
    return pd.DataFrame({'original': group,
                         'demeaned': group - group.mean()})

print(df.groupby('A')['B'].apply(f))
```

```
         original  demeaned
A
one   0         1 -0.333333
      1         2  0.666667
      6         1 -0.333333
three 3         2  0.500000
      7         1 -0.500000
two   2         1 -1.000000
      4         3  1.000000
      5         2  0.000000
```

[305]:
```
print(df.groupby("A").agg(["sum", "mean", "std"]))
```

```
        B                         C
      sum      mean       std sum      mean       std
A
one     4  1.333333  0.577350  13  4.333333  3.055050
three   3  1.500000  0.707107   3  1.500000  2.121320
```

```
two      6  2.000000  1.000000    9  3.000000  1.732051
```

[306]:
```
print(df.groupby("A")["B"].agg(min_height="min",
                               max_height="max",))
```

```
       min_height  max_height
A
one             1           2
three           1           2
two             1           3
```

iterator :

[307]:
```
df = pd.DataFrame(
    {
        "A": ["one", "one", "two", "three", "two", "two", "one", "three"],
        "B": [1, 2, 1, 2, 3, 2, 1, 1],
        "C": np.random.randint(0, 10, 8)
    }
)
for name, g in df.groupby("A"):
    print(f"{name};\n{g}")
```

```
one;
     A  B  C
0  one  1  1
1  one  2  2
6  one  1  7
three;
       A  B  C
3  three  2  9
7  three  1  4
two;
     A  B  C
2  two  1  7
4  two  3  0
5  two  2  8
```

12 Working with missing data

[308]:
```
s = pd.Series([1, 2], dtype="Int64").reindex([0, 1, 2])
print(s)
print(s.isna())
print(s.notna())
```

```
0       1
1       2
2    <NA>
dtype: Int64
```

```
0    False
1    False
2     True
dtype: bool
0     True
1     True
2    False
dtype: bool
```

[309]:
```
df = pd.DataFrame([np.nan, 10, 6, np.nan, np.nan, 3, np.nan])
print(df.dropna())
```

```
      0
1  10.0
2   6.0
5   3.0
```

[310]:
```
print(df.fillna(1)) # Fill `np.nan` with `1`.
```

```
      0
0   1.0
1  10.0
2   6.0
3   1.0
4   1.0
5   3.0
6   1.0
```

[311]:
```
print(df.ffill()) # Fill `np.nan` with the previous value.
```

```
      0
0   NaN
1  10.0
2   6.0
3   6.0
4   6.0
5   3.0
6   3.0
```

13 value counts and sort

[312]:
```
s = pd.Series([3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5])
print(s.value_counts())
```

```
5    3
1    2
3    2
4    1
9    1
```

```
2    1
6    1
Name: count, dtype: int64
```

[313]:
```
df = pd.DataFrame({"name" : ["A", "B", "C", "D"]},
                  index = [1, 0, 2, 4])
print(df.sort_index())
```

```
  name
0    B
1    A
2    C
4    D
```

[314]:
```
df = pd.DataFrame({"id" : [1, 0, 2, 4],
                   "name" : ["A", "B", "C", "D"]})
print(df.sort_values(by="id", ascending=False))
```

```
   id name
3   4    D
2   2    C
0   1    A
1   0    B
```