11.2_pandas

November 6, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- what is pandas?
- data processing
- data exploration
- read and save data

2 What is Pandas?

- Pandas is a Python library for conducting data analysis.
- First release was in 2010
- The Pandas name itself is derived from panel data, an econometrics term for multidimensional structured datasets, and a play on the phrase Python data analysis.
- Pandas provides high-level data structures and functions designed to make working with structured or tabular data intuitive and flexible.
- contains data structures and data manipulation tools designed to make data cleaning and analysis fast and convenient in Python.
- Works with **structured data**:
  - Tabular or spreadsheet-like data in which each column may be a different type (string, numeric, date, or otherwise). This includes most kinds of data commonly stored in relational databases or tab- or comma-delimited text files.
  - Multidimensional arrays (matrices).

  - Multiple tables of data interrelated by key columns (what would be primary or foreign keys for a SQL user).
  - Evenly or unevenly spaced time series.

2.1 Installation of Pandas

From a terminal:

```
pip install pandas
```

or

```
conda install pandas
```

`pandas` is included in conda installation, so our working environment should already have `pandas` installed.

[1]:
```
import pandas as pd
```

2.2 Core of Pandas: `DataFrame`

- The pandas `DataFrame` is a data structure that contains **two-dimensional** data and its corresponding row and column labels.
- Pandas blends the array-computing ideas of NumPy with the kinds of data manipulation capabilities found in spreadsheets and relational databases (such as SQL).
- `DataFrames` are widely used in data science, machine learning, scientific computing, and many other data-intensive fields.
- `DataFrames` are similar to SQL tables or the spreadsheets in Excel.
- In many cases, DataFrames are faster, easier to use, and more powerful than tables or spreadsheets because they're an integral part of the Python and NumPy ecosystems.

2.2.1 What is a Pandas `DataFrame`?

- Represents a rectangular table of data
- Contains an ordered, named collection of columns, each of which can be a different value type (numeric, string, Boolean, etc.)
- Has both a row and column index
- Can be thought of as a dictionary of `Series` all sharing the same index.

2.2.2 Creating a Pandas `DataFrame`

- Creating from a **dictionary** of **equal-length** lists or NumPy arrays
  - key is used as the column name (string)
  - value (**equal-length** lists or NumPy arrays) is used as the records
  - The resulting `DataFrame` will have its index assigned automatically
  - The columns are placed according to the order of the keys in data

  `pd.DataFrame(dict)`
- Creating from nested lists (sublists need to be **equal-length**) or a two-dimensional NumPy array
  - Column and row names can be specicified

`pd.DataFrame(array/nested lists, index= list, columns=list)`

[2]:
```
import numpy as np
```

[3]:
```
data = {"state": ["Ohio", "Ohio", "Ohio", "Nevada", "Nevada", "Nevada"],
        "year": [2000, 2001, 2002, 2001, 2002, 2003],
        "pop": [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
```

[4]:
```
data
```

[4]:
```
{'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
 'year': [2000, 2001, 2002, 2001, 2002, 2003],
 'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
```

[5]:
```
frame = pd.DataFrame(data)
frame
```

[5]:

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

[6]:
```
data = {"state": ["Ohio", "Ohio", "Ohio", "Nevada", "Nevada", "Nevada"],
        "pop": [1.5, 1.7, 3.6, 2.4, 2.9, 3.2],
        "year": [2000, 2001, 2002, 2001, 2002, 2003]
        }
frame = pd.DataFrame(data)
frame
```

[6]:

	state	pop	year
0	Ohio	1.5	2000
1	Ohio	1.7	2001
2	Ohio	3.6	2002
3	Nevada	2.4	2001
4	Nevada	2.9	2002
5	Nevada	3.2	2003

We can specify the order of the DataFrame's columns during the creation phase

[7]:
```
frame = pd.DataFrame(data, columns=["year", "state", "pop"])
frame
```

[7]:

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4

```
4  2002  Nevada  2.9
5  2003  Nevada  3.2
```

[8]:
```
frame = pd.DataFrame(data, columns=["year","state"])
frame
```

[8]:
```
   year   state
0  2000    Ohio
1  2001    Ohio
2  2002    Ohio
3  2001  Nevada
4  2002  Nevada
5  2003  Nevada
```

If you pass a column that isn't contained in the dictionary, it will appear with missing values in the result:

[9]:
```
frame = pd.DataFrame(data, columns=["year", "state", "pop", "poverty"])
frame
```

[9]:
```
   year   state  pop poverty
0  2000    Ohio  1.5     NaN
1  2001    Ohio  1.7     NaN
2  2002    Ohio  3.6     NaN
3  2001  Nevada  2.4     NaN
4  2002  Nevada  2.9     NaN
5  2003  Nevada  3.2     NaN
```

[10]:
```
frame.poverty
```

[10]:
```
0    NaN
1    NaN
2    NaN
3    NaN
4    NaN
5    NaN
Name: poverty, dtype: object
```

[11]:
```
frame.poverty = 0.5
```

[12]:
```
frame
```

[12]:
```
   year   state  pop  poverty
0  2000    Ohio  1.5      0.5
1  2001    Ohio  1.7      0.5
2  2002    Ohio  3.6      0.5
3  2001  Nevada  2.4      0.5
4  2002  Nevada  2.9      0.5
```

```
5  2003  Nevada  3.2      0.5
```

[13]:
```
type(frame)
```

[13]:
```
pandas.core.frame.DataFrame
```

Group exercise Create a pandas `DataFrame` using the four `array` variables. The `DataFrame` will have four columns with names `population`, `ward`, `year` and `poverty`:

```
ward = np.tile([1,2,3,4,5], 5)
year = np.array([2000] * 5 +[2001] * 5 + [2002] * 5 + [2003] * 5 + [2004] * 5)
population = np.random.randint(5000, size=(25,))
poverty = np.random.random(size=(25,))
```

Raise your hand when you are done!

[14]:
```
ward = np.tile([1,2,3,4,5], 5)
year = np.array([2000] * 5 +[2001] * 5 + [2002] * 5 + [2003] * 5 + [2004] * 5)
population = np.random.randint(5000, size=(25,))
poverty = np.random.random(size=(25,))
```

[15]:
```
dict_data = {"ward": ward,
             "year":year,
             "population": population,
             "poverty": poverty}
df = pd.DataFrame(dict_data)
df
```

[15]:

	ward	year	population	poverty
0	1	2000	3355	0.008886
1	2	2000	633	0.876133
2	3	2000	117	0.014302
3	4	2000	2396	0.411900
4	5	2000	97	0.747115
5	1	2001	4824	0.574371
6	2	2001	1237	0.473682
7	3	2001	2590	0.694412
8	4	2001	3879	0.676315
9	5	2001	4601	0.194569
10	1	2002	1055	0.067936
11	2	2002	801	0.137278
12	3	2002	2609	0.243622
13	4	2002	4204	0.865772
14	5	2002	3383	0.084191
15	1	2003	1159	0.642468
16	2	2003	1166	0.420943
17	3	2003	2335	0.354641
18	4	2003	4552	0.029051

```
19      5  2003         1663  0.469272
20      1  2004         3191  0.427377
21      2  2004         2188  0.610027
22      3  2004         4702  0.987751
23      4  2004          821  0.079957
24      5  2004         2365  0.792959
```

[16]:
```
ward = np.tile([1,2,3,4,5], 5) # 5 wards repeat 5 times
```

[17]:
```
ward
```

[17]:
```
array([1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2,
       3, 4, 5])
```

[18]:
```
[1,2,3,4,5]*5
```

[18]:
```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

[19]:
```
np.array([1,2,3,4,5]*5)
```

[19]:
```
array([1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2,
       3, 4, 5])
```

[20]:
```
year = np.array([2000] * 5 +[2001] * 5 + [2002] * 5 + [2003] * 5 + [2004] * 5)␣
  ↪# 2000, 2001, 2002, 2003, 2004 each for 5 times (5 wards)
year
```

[20]:
```
array([2000, 2000, 2000, 2000, 2000, 2001, 2001, 2001, 2001, 2001, 2002,
       2002, 2002, 2002, 2002, 2003, 2003, 2003, 2003, 2003, 2004, 2004,
       2004, 2004, 2004])
```

[21]:
```
np.random.randint?
```

[22]:
```
population = np.random.randint(5000, size=(25,))
```

[23]:
```
population
```

[23]:
```
array([3385, 1845,  154, 3574, 1292, 2780, 3082, 3514, 1551, 3962, 2910,
       2869,  480,   66, 3850, 2202, 3667, 3265, 2048, 4694, 4607,  764,
       2551, 4940,  388])
```

[24]:
```
poverty = np.random.random(size=(25,))
```

[25]:
```
poverty
```

[25]:
```
array([0.66122155, 0.63363207, 0.52138614, 0.1747214 , 0.75041163,
       0.53339757, 0.09957869, 0.92111871, 0.86711864, 0.46223953,
       0.50283289, 0.29849845, 0.02628136, 0.08119138, 0.36611933,
```

```
       0.83420546, 0.98986895, 0.07251114, 0.31857636, 0.60283935,
       0.51330556, 0.59138243, 0.34320289, 0.41459899, 0.19387829])
```

[26]:
```
df_ward = pd.DataFrame({'population': population,
                        'ward': ward,
                        'poverty': poverty,
                         'year': year})
df_ward
```

[26]:

	population	ward	poverty	year
0	3385	1	0.661222	2000
1	1845	2	0.633632	2000
2	154	3	0.521386	2000
3	3574	4	0.174721	2000
4	1292	5	0.750412	2000
5	2780	1	0.533398	2001
6	3082	2	0.099579	2001
7	3514	3	0.921119	2001
8	1551	4	0.867119	2001
9	3962	5	0.462240	2001
10	2910	1	0.502833	2002
11	2869	2	0.298498	2002
12	480	3	0.026281	2002
13	66	4	0.081191	2002
14	3850	5	0.366119	2002
15	2202	1	0.834205	2003
16	3667	2	0.989869	2003
17	3265	3	0.072511	2003
18	2048	4	0.318576	2003
19	4694	5	0.602839	2003
20	4607	1	0.513306	2004
21	764	2	0.591382	2004
22	2551	3	0.343203	2004
23	4940	4	0.414599	2004
24	388	5	0.193878	2004

Creating a pandas dataframe from a matrix/two-dimensional array

[27]:
```
data = np.arange(16).reshape((4, 4))
data
```

[27]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

[28]:
```
df_state = pd.DataFrame(data,
                        index=["Ohio", "Colorado", "Utah", "New York"],
```

```
                    columns=["one", "two", "three", "four"])
```

[29]:
```
df_state
```

[29]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

2.3 Exploring data with Pandas

[30]:
```
df_ward
```

[30]:

	population	ward	poverty	year
0	3385	1	0.661222	2000
1	1845	2	0.633632	2000
2	154	3	0.521386	2000
3	3574	4	0.174721	2000
4	1292	5	0.750412	2000
5	2780	1	0.533398	2001
6	3082	2	0.099579	2001
7	3514	3	0.921119	2001
8	1551	4	0.867119	2001
9	3962	5	0.462240	2001
10	2910	1	0.502833	2002
11	2869	2	0.298498	2002
12	480	3	0.026281	2002
13	66	4	0.081191	2002
14	3850	5	0.366119	2002
15	2202	1	0.834205	2003
16	3667	2	0.989869	2003
17	3265	3	0.072511	2003
18	2048	4	0.318576	2003
19	4694	5	0.602839	2003
20	4607	1	0.513306	2004
21	764	2	0.591382	2004
22	2551	3	0.343203	2004
23	4940	4	0.414599	2004
24	388	5	0.193878	2004

[31]:
```
df_ward.head() # first 5 rows
```

[31]:

	population	ward	poverty	year
0	3385	1	0.661222	2000
1	1845	2	0.633632	2000
2	154	3	0.521386	2000

```
3         3574     4  0.174721  2000
4         1292     5  0.750412  2000
```

[32]:
```
df_ward.head(2) # first 2 rows
```

[32]:
```
   population  ward   poverty  year
0         3385     1  0.661222  2000
1         1845     2  0.633632  2000
```

[33]:
```
df_ward.tail() # last 5 rows
```

[33]:
```
    population  ward   poverty  year
20        4607     1  0.513306  2004
21         764     2  0.591382  2004
22        2551     3  0.343203  2004
23        4940     4  0.414599  2004
24         388     5  0.193878  2004
```

[34]:
```
df_ward.tail(2) # last 2 rows
```

[34]:
```
    population  ward   poverty  year
23        4940     4  0.414599  2004
24         388     5  0.193878  2004
```

[35]:
```
df_ward.columns
```

[35]:
```
Index(['population', 'ward', 'poverty', 'year'], dtype='object')
```

[36]:
```
df_ward.shape
```

[36]:
```
(25, 4)
```

[37]:
```
len(df_ward)
```

[37]:
```
25
```

[38]:
```
df_ward.shape[0]
```

[38]:
```
25
```

[39]:
```
df_ward.shape[1]
```

[39]:
```
4
```

2.4 Indexing `DataFrame`

- indexing columns
- indexing rows
  - works analogously to NumPy array indexing (integer indexing)

    - `iloc`: integer-based indexing.
  - you can use the index values instead of only integers
    - `loc`: label-based indexing

[40]:
```
data
```

[40]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

[41]:
```
df_state = pd.DataFrame(data,
                        index=["Ohio", "Colorado", "Utah", "New York"],
                        columns=["one", "two", "three", "four"])
df_state
```

[41]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

[42]:
```
df_state[["three", "one"]]
```

[42]:

	three	one
Ohio	2	0
Colorado	6	4
Utah	10	8
New York	14	12

[43]:
```
df_state[["two"]]
```

[43]:

	two
Ohio	1
Colorado	5
Utah	9
New York	13

[44]:
```
df_state["two"]
```

[44]:
```
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int64
```

[45]:
```
type(df_state[["two"]])
```

[45]:
```
pandas.core.frame.DataFrame
```

[46]:
```
type(df_state["two"])
```

[46]:
```
pandas.core.series.Series
```

[47]:
```
df_state.two
```

[47]:
```
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int64
```

[48]:
```
df_state
```

[48]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

[49]:
```
df_state[1:3]
```

[49]:

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11

[50]:
```
df_state[:2]
```

[50]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

The row selection syntax `df_state[:2]` is provided as a convenience. Passing a single element or a list to the `[]` operator selects columns.

[51]:
```
df_state
```

[51]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

[52]:
```
df_state[2]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
```

```
File ~/anaconda3/lib/python3.11/site-packages/pandas/core/indexes/base.py:3653,
 ↪in Index.get_loc(self, key)
   3652 try:
-> 3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:

File ~/anaconda3/lib/python3.11/site-packages/pandas/_libs/index.pyx:147, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()

File ~/anaconda3/lib/python3.11/site-packages/pandas/_libs/index.pyx:176, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7080, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7088, in pandas._libs.hashtable.
 ↪PyObjectHashTable.get_item()

KeyError: 2

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[52], line 1
----> 1 df_state[2]

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/frame.py:3761, in␣
 ↪DataFrame.__getitem__(self, key)
   3759 if self.columns.nlevels > 1:
   3760     return self._getitem_multilevel(key)
-> 3761 indexer = self.columns.get_loc(key)
   3762 if is_integer(indexer):
   3763     indexer = [indexer]

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/indexes/base.py:3655,
 ↪in Index.get_loc(self, key)
   3653     return self._engine.get_loc(casted_key)
   3654 except KeyError as err:
-> 3655     raise KeyError(key) from err
   3656 except TypeError:
   3657     # If we have a listlike key, _check_indexing_error will raise
   3658     #  InvalidIndexError. Otherwise we fall through and re-raise
   3659     #  the TypeError.
   3660     self._check_indexing_error(key)

KeyError: 2
```

[53]:
```
df_state
```

[53]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

[54]:
```
df_state[:2]
```

[54]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

[55]:
```
df_state
```

[55]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

[56]:
```
df_state[1:3]
```

[56]:

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11

[57]:
```
df_state[-2:]
```

[57]:

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

2.4.1 "Row" selection on DataFrame with loc and iloc

- `loc`: label-based indexing
- `iloc`: integer-based indexing.

[58]:
```
df_state
```

[58]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

[59]:
```
df_state.loc["Colorado"]
```

[59]:
```
one      4
two      5
three    6
four     7
Name: Colorado, dtype: int64
```

[60]:
```
df_state.iloc[1]
```

[60]:
```
one      4
two      5
three    6
four     7
Name: Colorado, dtype: int64
```

[61]:
```
df_state.loc["Utah"]
```

[61]:
```
one       8
two       9
three    10
four     11
Name: Utah, dtype: int64
```

[62]:
```
df_state.iloc[1]
```

[62]:
```
one      4
two      5
three    6
four     7
Name: Colorado, dtype: int64
```

[63]:
```
df_state
```

[63]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

[64]:
```
df_state.loc[["Utah","Ohio"]]
```

[64]:

	one	two	three	four
Utah	8	9	10	11
Ohio	0	1	2	3

[65]:
```
df_state.iloc[[2,0]]
```

[65]:

	one	two	three	four
Utah	8	9	10	11

```
Ohio    0    1      2     3
```

Filter data with conditions

[66]:
```
df_state
```

[66]:
```
          one  two  three  four
Ohio        0    1      2     3
Colorado    4    5      6     7
Utah        8    9     10    11
New York   12   13     14    15
```

[67]:
```
df_state < 9
```

[67]:
```
             one    two  three   four
Ohio        True   True   True   True
Colorado    True   True   True   True
Utah        True  False  False  False
New York   False  False  False  False
```

[68]:
```
df_state[df_state < 9]
```

[68]:
```
          one  two  three  four
Ohio      0.0  1.0    2.0   3.0
Colorado  4.0  5.0    6.0   7.0
Utah      8.0  NaN    NaN   NaN
New York  NaN  NaN    NaN   NaN
```

[69]:
```
df_state[df_state < 9] = 9
```

[70]:
```
df_state
```

[70]:
```
          one  two  three  four
Ohio        9    9      9     9
Colorado    9    9      9     9
Utah        9    9     10    11
New York   12   13     14    15
```

[71]:
```
df_state
```

[71]:
```
          one  two  three  four
Ohio        9    9      9     9
Colorado    9    9      9     9
Utah        9    9     10    11
New York   12   13     14    15
```

[72]:
```
df_state["three"] == 10
```

[72]:
```
Ohio        False
Colorado    False
Utah         True
New York    False
Name: three, dtype: bool
```

[73]:
```
df_state.three == 10
```

[73]:
```
Ohio        False
Colorado    False
Utah         True
New York    False
Name: three, dtype: bool
```

[74]:
```
df_state[df_state.three==10]
```

[74]:

	one	two	three	four
Utah	9	9	10	11

try on the other DataFrame

[75]:
```
df_ward.head(3)
```

[75]:

	population	ward	poverty	year
0	3385	1	0.661222	2000
1	1845	2	0.633632	2000
2	154	3	0.521386	2000

[76]:
```
df_ward['population']
```

[76]:
```
0     3385
1     1845
2      154
3     3574
4     1292
5     2780
6     3082
7     3514
8     1551
9     3962
10    2910
11    2869
12     480
13      66
14    3850
15    2202
16    3667
17    3265
```

```
18    2048
19    4694
20    4607
21     764
22    2551
23    4940
24     388
Name: population, dtype: int64
```

[77]:
```
df_ward.population
```

[77]:
```
0     3385
1     1845
2      154
3     3574
4     1292
5     2780
6     3082
7     3514
8     1551
9     3962
10    2910
11    2869
12     480
13      66
14    3850
15    2202
16    3667
17    3265
18    2048
19    4694
20    4607
21     764
22    2551
23    4940
24     388
Name: population, dtype: int64
```

[78]:
```
df_ward
```

[78]:

	population	ward	poverty	year
0	3385	1	0.661222	2000
1	1845	2	0.633632	2000
2	154	3	0.521386	2000
3	3574	4	0.174721	2000
4	1292	5	0.750412	2000
5	2780	1	0.533398	2001

```
6          3082     2  0.099579  2001
7          3514     3  0.921119  2001
8          1551     4  0.867119  2001
9          3962     5  0.462240  2001
10         2910     1  0.502833  2002
11         2869     2  0.298498  2002
12          480     3  0.026281  2002
13           66     4  0.081191  2002
14         3850     5  0.366119  2002
15         2202     1  0.834205  2003
16         3667     2  0.989869  2003
17         3265     3  0.072511  2003
18         2048     4  0.318576  2003
19         4694     5  0.602839  2003
20         4607     1  0.513306  2004
21          764     2  0.591382  2004
22         2551     3  0.343203  2004
23         4940     4  0.414599  2004
24          388     5  0.193878  2004
```

[79]:
```
df_ward[0:4]
```

[79]:
```
   population  ward   poverty  year
0        3385     1  0.661222  2000
1        1845     2  0.633632  2000
2         154     3  0.521386  2000
3        3574     4  0.174721  2000
```

[80]:
```
df_ward[-4:]
```

[80]:
```
    population  ward   poverty  year
21         764     2  0.591382  2004
22        2551     3  0.343203  2004
23        4940     4  0.414599  2004
24         388     5  0.193878  2004
```

[81]:
```
df_ward[df_ward.ward==2]
```

[81]:
```
    population  ward   poverty  year
1         1845     2  0.633632  2000
6         3082     2  0.099579  2001
11        2869     2  0.298498  2002
16        3667     2  0.989869  2003
21         764     2  0.591382  2004
```

[82]:
```
df_ward[df_ward.population<1000]
```

[82]:
	population	ward	poverty	year
2	154	3	0.521386	2000
12	480	3	0.026281	2002
13	66	4	0.081191	2002
21	764	2	0.591382	2004
24	388	5	0.193878	2004

[83]:
```
df_ward[(df_ward.ward==2) & (df_ward.population < 1000)] # & binary operator to␣
↪perform and operation on lists of boolean values
```

[83]:
	population	ward	poverty	year
21	764	2	0.591382	2004

[84]:
```
(df_ward.ward==2) & (df_ward.population < 1000)
```

[84]:
```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21     True
22    False
23    False
24    False
dtype: bool
```

[85]:
```
df_ward[(df_ward.ward==2) | (df_ward.population < 1000)] # | binary operator to␣
↪perform or operation on lists of boolean values
```

[85]:
	population	ward	poverty	year
1	1845	2	0.633632	2000
2	154	3	0.521386	2000

```
6          3082     2  0.099579  2001
11         2869     2  0.298498  2002
12          480     3  0.026281  2002
13           66     4  0.081191  2002
16         3667     2  0.989869  2003
21          764     2  0.591382  2004
24          388     5  0.193878  2004
```

[86]:
```
df_ward[(~(df_ward.ward==2)) & (df_ward.population < 1000)] # not in ward 2 and less than 1000 population
```

[86]:
```
    population  ward   poverty  year
2          154     3  0.521386  2000
12         480     3  0.026281  2002
13          66     4  0.081191  2002
24         388     5  0.193878  2004
```

[87]:
```
df_ward[~((df_ward.ward==2) & (df_ward.population < 1000))] # not (in ward 2 and less than 1000 population)
```

[87]:
```
    population  ward   poverty  year
0          3385     1  0.661222  2000
1          1845     2  0.633632  2000
2           154     3  0.521386  2000
3          3574     4  0.174721  2000
4          1292     5  0.750412  2000
5          2780     1  0.533398  2001
6          3082     2  0.099579  2001
7          3514     3  0.921119  2001
8          1551     4  0.867119  2001
9          3962     5  0.462240  2001
10         2910     1  0.502833  2002
11         2869     2  0.298498  2002
12          480     3  0.026281  2002
13           66     4  0.081191  2002
14         3850     5  0.366119  2002
15         2202     1  0.834205  2003
16         3667     2  0.989869  2003
17         3265     3  0.072511  2003
18         2048     4  0.318576  2003
19         4694     5  0.602839  2003
20         4607     1  0.513306  2004
22         2551     3  0.343203  2004
23         4940     4  0.414599  2004
24          388     5  0.193878  2004
```

2.4.2 Group exercise

```
ward = np.tile([1,2,3,4,5], 5)
year = np.array([2000] * 5 +[2001] * 5 + [2002] * 5 + [2003] * 5 + [2004] * 5)
population = np.random.randint(5000, size=(25,))
poverty = np.random.random(size=(25,))
df_ward = pandas.DataFrame({'population': population,
                        'ward': ward,
                        'poverty': poverty})
```

Selecting records from `df_ward` that are in ward 3, larger than 500 population, and poverty rate less than 40%

When you are done, raise your hand

[88]:
```
df_ward[(df_ward.ward==3) & (df_ward.population > 500) & (df_ward.poverty<0.4)]
```

[88]:

	population	ward	poverty	year
17	3265	3	0.072511	2003
22	2551	3	0.343203	2004

2.5 Creating New Columns in an existing `DataFrame`

[89]:
```
df_ward.head()
```

[89]:

	population	ward	poverty	year
0	3385	1	0.661222	2000
1	1845	2	0.633632	2000
2	154	3	0.521386	2000
3	3574	4	0.174721	2000
4	1292	5	0.750412	2000

[90]:
```
df_ward.population * df_ward.poverty
```

[90]:
```
0     2238.234930
1     1169.051172
2       80.293465
3      624.454268
4      969.531829
5     1482.845242
6      306.901512
7     3236.811139
8     1344.901014
9     1831.393033
10    1463.243722
11     856.392042
12      12.615051
13       5.358631
14    1409.559418
```

```
15    1836.920423
16    3629.849458
17     236.748857
18     652.444387
19    2829.727892
20    2364.798695
21     451.816176
22     875.510574
23    2048.118989
24      75.224777
dtype: float64
```

[91]:
```
pop_pov = df_ward.population * df_ward.poverty # elementwise operation similar
    ↪to numpy array
pop_pov
```

[91]:
```
0     2238.234930
1     1169.051172
2       80.293465
3      624.454268
4      969.531829
5     1482.845242
6      306.901512
7     3236.811139
8     1344.901014
9     1831.393033
10    1463.243722
11     856.392042
12      12.615051
13       5.358631
14    1409.559418
15    1836.920423
16    3629.849458
17     236.748857
18     652.444387
19    2829.727892
20    2364.798695
21     451.816176
22     875.510574
23    2048.118989
24      75.224777
dtype: float64
```

[92]:
```
df_ward
```

[92]:
```
    population  ward   poverty  year
0         3385     1  0.661222  2000
```

```
1          1845     2  0.633632  2000
2           154     3  0.521386  2000
3          3574     4  0.174721  2000
4          1292     5  0.750412  2000
5          2780     1  0.533398  2001
6          3082     2  0.099579  2001
7          3514     3  0.921119  2001
8          1551     4  0.867119  2001
9          3962     5  0.462240  2001
10         2910     1  0.502833  2002
11         2869     2  0.298498  2002
12          480     3  0.026281  2002
13           66     4  0.081191  2002
14         3850     5  0.366119  2002
15         2202     1  0.834205  2003
16         3667     2  0.989869  2003
17         3265     3  0.072511  2003
18         2048     4  0.318576  2003
19         4694     5  0.602839  2003
20         4607     1  0.513306  2004
21          764     2  0.591382  2004
22         2551     3  0.343203  2004
23         4940     4  0.414599  2004
24          388     5  0.193878  2004
```

[93]:
```
df_ward['pop_pov'] = pop_pov.astype('int')
```

[94]:
```
df_ward.head()
```

[94]:
```
   population  ward   poverty  year  pop_pov
0        3385     1  0.661222  2000     2238
1        1845     2  0.633632  2000     1169
2         154     3  0.521386  2000       80
3        3574     4  0.174721  2000      624
4        1292     5  0.750412  2000      969
```

2.6 Aggregation/Groupby

[95]:
```
df_ward[df_ward.ward==1]
```

[95]:
```
    population  ward   poverty  year  pop_pov
0         3385     1  0.661222  2000     2238
5         2780     1  0.533398  2001     1482
10        2910     1  0.502833  2002     1463
15        2202     1  0.834205  2003     1836
20        4607     1  0.513306  2004     2364
```

[96]:
```
df_ward.groupby(by='ward').sum()
```

[96]:

ward	population	poverty	year	pop_pov
1	15884	3.044963	10010	9383
2	12227	2.612961	10010	6411
3	9964	1.884500	10010	4439
4	12179	1.856207	10010	4673
5	14186	2.375488	10010	7113

[97]:
```
df_ward.groupby(by='ward').sum()[['population', 'pop_pov']]
```

[97]:

ward	population	pop_pov
1	15884	9383
2	12227	6411
3	9964	4439
4	12179	4673
5	14186	7113

[98]:
```
ward_df = df_ward.groupby(by='ward').sum()[['population', 'pop_pov']]
```

[99]:
```
ward_df
```

[99]:

ward	population	pop_pov
1	15884	9383
2	12227	6411
3	9964	4439
4	12179	4673
5	14186	7113

[100]:
```
ward_df['poverty'] = ward_df.pop_pov / ward_df.population
```

[101]:
```
ward_df
```

[101]:

ward	population	pop_pov	poverty
1	15884	9383	0.590720
2	12227	6411	0.524331
3	9964	4439	0.445504
4	12179	4673	0.383693
5	14186	7113	0.501410

2.7 Joins/Merge

[102]:
```
ward_df
```

[102]:
```
      population  pop_pov   poverty
ward
1          15884     9383  0.590720
2          12227     6411  0.524331
3           9964     4439  0.445504
4          12179     4673  0.383693
5          14186     7113  0.501410
```

[103]:
```
df_ward
```

[103]:
```
    population  ward   poverty  year  pop_pov
0         3385     1  0.661222  2000     2238
1         1845     2  0.633632  2000     1169
2          154     3  0.521386  2000       80
3         3574     4  0.174721  2000      624
4         1292     5  0.750412  2000      969
5         2780     1  0.533398  2001     1482
6         3082     2  0.099579  2001      306
7         3514     3  0.921119  2001     3236
8         1551     4  0.867119  2001     1344
9         3962     5  0.462240  2001     1831
10        2910     1  0.502833  2002     1463
11        2869     2  0.298498  2002      856
12         480     3  0.026281  2002       12
13          66     4  0.081191  2002        5
14        3850     5  0.366119  2002     1409
15        2202     1  0.834205  2003     1836
16        3667     2  0.989869  2003     3629
17        3265     3  0.072511  2003      236
18        2048     4  0.318576  2003      652
19        4694     5  0.602839  2003     2829
20        4607     1  0.513306  2004     2364
21         764     2  0.591382  2004      451
22        2551     3  0.343203  2004      875
23        4940     4  0.414599  2004     2048
24         388     5  0.193878  2004       75
```

[104]:
```
df_all = df_ward.merge(ward_df,  on='ward')
```

[105]:
```
df_all
```

[105]:
```
    population_x  ward  poverty_x  year  pop_pov_x  population_y  pop_pov_y  \
0           3385     1   0.661222  2000       2238         15884       9383
1           2780     1   0.533398  2001       1482         15884       9383
```

2	2910	1	0.502833	2002	1463	15884	9383
3	2202	1	0.834205	2003	1836	15884	9383
4	4607	1	0.513306	2004	2364	15884	9383
5	1845	2	0.633632	2000	1169	12227	6411
6	3082	2	0.099579	2001	306	12227	6411
7	2869	2	0.298498	2002	856	12227	6411
8	3667	2	0.989869	2003	3629	12227	6411
9	764	2	0.591382	2004	451	12227	6411
10	154	3	0.521386	2000	80	9964	4439
11	3514	3	0.921119	2001	3236	9964	4439
12	480	3	0.026281	2002	12	9964	4439
13	3265	3	0.072511	2003	236	9964	4439
14	2551	3	0.343203	2004	875	9964	4439
15	3574	4	0.174721	2000	624	12179	4673
16	1551	4	0.867119	2001	1344	12179	4673
17	66	4	0.081191	2002	5	12179	4673
18	2048	4	0.318576	2003	652	12179	4673
19	4940	4	0.414599	2004	2048	12179	4673
20	1292	5	0.750412	2000	969	14186	7113
21	3962	5	0.462240	2001	1831	14186	7113
22	3850	5	0.366119	2002	1409	14186	7113
23	4694	5	0.602839	2003	2829	14186	7113
24	388	5	0.193878	2004	75	14186	7113

	poverty_y
0	0.590720
1	0.590720
2	0.590720
3	0.590720
4	0.590720
5	0.524331
6	0.524331
7	0.524331
8	0.524331
9	0.524331
10	0.445504
11	0.445504
12	0.445504
13	0.445504
14	0.445504
15	0.383693
16	0.383693
17	0.383693
18	0.383693
19	0.383693
20	0.501410
21	0.501410

```
22   0.501410
23   0.501410
24   0.501410
```

[106]:
```
df_all = df_ward.merge(ward_df, on='ward',
                       suffixes = ('_neighborhood', '_ward'))
```

[107]:
```
df_all
```

[107]:

	population_neighborhood	ward	poverty_neighborhood	year	\
0	3385	1	0.661222	2000	
1	2780	1	0.533398	2001	
2	2910	1	0.502833	2002	
3	2202	1	0.834205	2003	
4	4607	1	0.513306	2004	
5	1845	2	0.633632	2000	
6	3082	2	0.099579	2001	
7	2869	2	0.298498	2002	
8	3667	2	0.989869	2003	
9	764	2	0.591382	2004	
10	154	3	0.521386	2000	
11	3514	3	0.921119	2001	
12	480	3	0.026281	2002	
13	3265	3	0.072511	2003	
14	2551	3	0.343203	2004	
15	3574	4	0.174721	2000	
16	1551	4	0.867119	2001	
17	66	4	0.081191	2002	
18	2048	4	0.318576	2003	
19	4940	4	0.414599	2004	
20	1292	5	0.750412	2000	
21	3962	5	0.462240	2001	
22	3850	5	0.366119	2002	
23	4694	5	0.602839	2003	
24	388	5	0.193878	2004	

	pop_pov_neighborhood	population_ward	pop_pov_ward	poverty_ward
0	2238	15884	9383	0.590720
1	1482	15884	9383	0.590720
2	1463	15884	9383	0.590720
3	1836	15884	9383	0.590720
4	2364	15884	9383	0.590720
5	1169	12227	6411	0.524331
6	306	12227	6411	0.524331
7	856	12227	6411	0.524331
8	3629	12227	6411	0.524331
9	451	12227	6411	0.524331

10	80	9964	4439	0.445504
11	3236	9964	4439	0.445504
12	12	9964	4439	0.445504
13	236	9964	4439	0.445504
14	875	9964	4439	0.445504
15	624	12179	4673	0.383693
16	1344	12179	4673	0.383693
17	5	12179	4673	0.383693
18	652	12179	4673	0.383693
19	2048	12179	4673	0.383693
20	969	14186	7113	0.501410
21	1831	14186	7113	0.501410
22	1409	14186	7113	0.501410
23	2829	14186	7113	0.501410
24	75	14186	7113	0.501410

[108]:
```
df_all[df_all.poverty_neighborhood > df_all.poverty_ward]
```

[108]:

	population_neighborhood	ward	poverty_neighborhood	year	\
0	3385	1	0.661222	2000	
3	2202	1	0.834205	2003	
5	1845	2	0.633632	2000	
8	3667	2	0.989869	2003	
9	764	2	0.591382	2004	
10	154	3	0.521386	2000	
11	3514	3	0.921119	2001	
16	1551	4	0.867119	2001	
19	4940	4	0.414599	2004	
20	1292	5	0.750412	2000	
23	4694	5	0.602839	2003	

	pop_pov_neighborhood	population_ward	pop_pov_ward	poverty_ward
0	2238	15884	9383	0.590720
3	1836	15884	9383	0.590720
5	1169	12227	6411	0.524331
8	3629	12227	6411	0.524331
9	451	12227	6411	0.524331
10	80	9964	4439	0.445504
11	3236	9964	4439	0.445504
16	1344	12179	4673	0.383693
19	2048	12179	4673	0.383693
20	969	14186	7113	0.501410
23	2829	14186	7113	0.501410

Which ward has the highest average poverty rate?

[109]:
```
df_all.poverty_neighborhood.idxmax()
```

[109]: 8

[110]:
```
df_all.loc[df_all['poverty_neighborhood'].idxmax()]
```

[110]:
```
population_neighborhood     3667.000000
ward                           2.000000
poverty_neighborhood           0.989869
year                        2003.000000
pop_pov_neighborhood        3629.000000
population_ward            12227.000000
pop_pov_ward                6411.000000
poverty_ward                   0.524331
Name: 8, dtype: float64
```

Which ward in which year has the lowest poverty rate?

[111]:
```
df_all.poverty_neighborhood.idxmin()
```

[111]: 12

[112]:
```
df_all.loc[df_all['poverty_neighborhood'].idxmin()]
```

[112]:
```
population_neighborhood     480.000000
ward                          3.000000
poverty_neighborhood          0.026281
year                       2002.000000
pop_pov_neighborhood         12.000000
population_ward            9964.000000
pop_pov_ward               4439.000000
poverty_ward                  0.445504
Name: 12, dtype: float64
```

2.8 Reading and Writing Data with Pandas

- Pandas features a number of functions for reading tabular data as a `DataFrame` object.
- Works with many different data formats
- Works with different data source:
  - reading text files and other more efficient on-disk formats
  - loading data from databases
  - interacting with network sources like web APIs

2.8.1 An example with working with csv files

- `read_csv` function: Load delimited data from a file, URL, or file-like object; use comma as default delimiter
  - A long list of optional arguments to deal with messy data in the real world
- `to_csv` method (associated with a `DataFrame` instance): Writing to a csv file

[113]:
```
df1 = pd.read_csv("ex1.csv")
df1
```

[113]:

	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

If only the path is supplied, the first row of the file will be used as the header (column names) of the `DataFrame` object and column names are inferred from the first line of the file.

[114]:
```
df2 = pd.read_csv("ex1.csv", header=None)
df2
```

[114]:

	0	1	2	3	4
0	a	b	c	d	message
1	1	2	3	4	hello
2	5	6	7	8	world
3	9	10	11	12	foo

If `header=None`, integer index starting from 0 will be used as column names.

[115]:
```
df3 = pd.read_csv("ex1.csv", names=["col1", "col2", "col3", "col4", "col5"])
df3
```

[115]:

	col1	col2	col3	col4	col5
0	a	b	c	d	message
1	1	2	3	4	hello
2	5	6	7	8	world
3	9	10	11	12	foo

We can pass a list of column names to the argument `names`

[116]:
```
df4 = pd.read_csv("ex1.csv", index_col="message")
df4
```

[116]:

message	a	b	c	d
hello	1	2	3	4
world	5	6	7	8
foo	9	10	11	12

We can specify the column name/index in the argument `index_col` as the row labels of the DataFrame

[117]:
```
df4 = pd.read_csv("ex1.csv", index_col=4)
df4
```

[117]:

message	a	b	c	d

```
hello    1   2   3   4
world    5   6   7   8
foo      9  10  11  12
```

[118]:
```
df5 = pd.read_csv("ex1.csv", skiprows=[1,2])
df5
```

[118]:
```
   a   b   c   d message
0  9  10  11  12     foo
```

Argument `skiprows`: Line numbers to skip (0-indexed) or number of lines to skip (int) at the start of the file.

[119]:
```
df6 = pd.read_csv("ex1.csv", skiprows=2)
df6
```

[119]:
```
   5   6   7   8 world
0  9  10  11  12   foo
```

Dealing with missing values

- To control which values are parsed as missing values (which are signified by NaN), specify a string in `na_values`.
- If you specify a list of strings, then all values in it are considered to be missing values.
- If you specify a number (a float, like 5.0 or an integer like 5), the corresponding equivalent values will also imply a missing value (in this case effectively [5.0, 5] are recognized as NaN).

[120]:
```
df_ex5 = pd.read_csv("ex5.csv")
df_ex5
```

[120]:
```
  something  a   b     c   d message
0       one  1   2   3.0   4     NaN
1       two  5   6   NaN   8   world
2     three  9  10  11.0  12     foo
```

[121]:
```
df_ex5 = pd.read_csv("ex5.csv", na_values=["one", 1])
df_ex5
```

[121]:
```
  something    a   b     c   d message
0       NaN  NaN   2   3.0   4     NaN
1       two  5.0   6   NaN   8   world
2     three  9.0  10  11.0  12     foo
```

[122]:
```
df_ex5.dropna() #Drop the rows where at least one element is missing.
```

[122]:
```
  something    a   b     c   d message
2     three  9.0  10  11.0  12     foo
```

[123]:
```
df_ex5.dropna(axis='columns') # Drop the columns where at least one element is␣
↪missing.
```

[123]:

	b	d
0	2	4
1	6	8
2	10	12

[124]:
```
df_ex5.dropna(subset=["something"]) #Define in which columns to look for␣
↪missing values.
```

[124]:

	something	a	b	c	d	message
1	two	5.0	6	NaN	8	world
2	three	9.0	10	11.0	12	foo

Save a Dataframe to a csv file

[125]:
```
df4.to_csv("data/output1.csv")
```

Read panda's documentation to better understand the functionality of pandas's `read_csv` function.

3 Further readings

- Python for Data Analysis, 3E, by Wes McKinney