11.2_pandas

November 1, 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
```

```
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R)
SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel
Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX)
instructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R)
SSE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel
Library 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX)
instructions.
```

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	563	0.015778
1	2	2000	1317	0.224249
2	3	2000	4879	0.531786
3	4	2000	1616	0.410666
4	5	2000	1541	0.323633
5	1	2001	243	0.694773
6	2	2001	2602	0.532491
7	3	2001	1224	0.959288
8	4	2001	2831	0.976427
9	5	2001	1680	0.965904
10	1	2002	3228	0.130646

```
11     2  2002        4582  0.690822
12     3  2002         264  0.137662
13     4  2002        4841  0.540827
14     5  2002        4295  0.331185
15     1  2003        2192  0.947102
16     2  2003        2450  0.266429
17     3  2003         109  0.905399
18     4  2003        2815  0.883970
19     5  2003        1866  0.080623
20     1  2004        3674  0.386779
21     2  2004        1224  0.343806
22     3  2004        4467  0.990861
23     4  2004        2463  0.516205
24     5  2004        4938  0.021159
```

[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([1975, 3581, 1879, 4918, 3641, 2300, 3555, 2807, 3325, 4907, 4717,
        766, 3857,  367, 1574, 3894, 4967, 2649, 3301, 2446, 3926, 1512,
       3764, 4221, 4320])
```

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

[25]:
```
poverty
```

[25]:
```
array([0.08753019, 0.49780678, 0.14976281, 0.11843266, 0.16970574,
       0.66374778, 0.09276944, 0.85159874, 0.59909053, 0.4531564 ,
       0.44682565, 0.22725061, 0.64221256, 0.08981608, 0.96149229,
       0.98113784, 0.67392056, 0.56841866, 0.94060747, 0.6080263 ,
       0.18987905, 0.41738673, 0.25595313, 0.5169994 , 0.71953999])
```

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

[26]:

	population	ward	poverty	year
0	1975	1	0.087530	2000
1	3581	2	0.497807	2000
2	1879	3	0.149763	2000
3	4918	4	0.118433	2000
4	3641	5	0.169706	2000
5	2300	1	0.663748	2001
6	3555	2	0.092769	2001
7	2807	3	0.851599	2001
8	3325	4	0.599091	2001
9	4907	5	0.453156	2001
10	4717	1	0.446826	2002
11	766	2	0.227251	2002
12	3857	3	0.642213	2002
13	367	4	0.089816	2002
14	1574	5	0.961492	2002
15	3894	1	0.981138	2003
16	4967	2	0.673921	2003
17	2649	3	0.568419	2003
18	3301	4	0.940607	2003
19	2446	5	0.608026	2003
20	3926	1	0.189879	2004
21	1512	2	0.417387	2004
22	3764	3	0.255953	2004
23	4221	4	0.516999	2004
24	4320	5	0.719540	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	1975	1	0.087530	2000
1	3581	2	0.497807	2000
2	1879	3	0.149763	2000
3	4918	4	0.118433	2000
4	3641	5	0.169706	2000
5	2300	1	0.663748	2001
6	3555	2	0.092769	2001
7	2807	3	0.851599	2001
8	3325	4	0.599091	2001
9	4907	5	0.453156	2001
10	4717	1	0.446826	2002
11	766	2	0.227251	2002
12	3857	3	0.642213	2002
13	367	4	0.089816	2002
14	1574	5	0.961492	2002
15	3894	1	0.981138	2003
16	4967	2	0.673921	2003
17	2649	3	0.568419	2003
18	3301	4	0.940607	2003
19	2446	5	0.608026	2003
20	3926	1	0.189879	2004
21	1512	2	0.417387	2004
22	3764	3	0.255953	2004
23	4221	4	0.516999	2004
24	4320	5	0.719540	2004

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

[31]:

	population	ward	poverty	year
0	1975	1	0.087530	2000
1	3581	2	0.497807	2000
2	1879	3	0.149763	2000
3	4918	4	0.118433	2000
4	3641	5	0.169706	2000

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

[32]:

	population	ward	poverty	year
0	1975	1	0.087530	2000
1	3581	2	0.497807	2000

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

[33]:

	population	ward	poverty	year
20	3926	1	0.189879	2004
21	1512	2	0.417387	2004
22	3764	3	0.255953	2004
23	4221	4	0.516999	2004
24	4320	5	0.719540	2004

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

[34]:

	population	ward	poverty	year
23	4221	4	0.516999	2004
24	4320	5	0.719540	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]:
```
df_state = pd.DataFrame(data,
                        index=["Ohio", "Colorado", "Utah", "New York"],
                        columns=["one", "two", "three", "four"])
df_state
```

[40]:

	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

[41]:
```
df_state[["three", "one"]]
```

[41]:

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

[42]:
```
df_state[["two"]]
```

[42]:

	two
Ohio	1
Colorado	5
Utah	9
New York	13

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

[43]:
```
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int64
```

[44]:
```
type(df_state[["two"]])
```

[44]:
```
pandas.core.frame.DataFrame
```

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

[45]:
```
pandas.core.series.Series
```

[46]:
```
df_state.two
```

[46]:
```
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int64
```

[47]:
```
df_state
```

[47]:

	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

[48]:
```
df_state[1:3]
```

[48]:

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

[49]:
```
df_state[:2]
```

[49]:

	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.

[50]:
```
df_state
```

[50]:

	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

[51]:
```
df_state[2]
```

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

```
File ~/opt/anaconda3/lib/python3.9/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 ~/opt/anaconda3/lib/python3.9/site-packages/pandas/_libs/index.pyx:147, in
 ↪pandas._libs.index.IndexEngine.get_loc()

File ~/opt/anaconda3/lib/python3.9/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[51], line 1
----> 1 df_state[2]

File ~/opt/anaconda3/lib/python3.9/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 ~/opt/anaconda3/lib/python3.9/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
```

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

[52]:
```
          one  two  three  four
Ohio        0    1      2     3
Colorado    4    5      6     7
```

[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[1:3]
```

[54]:
```
          one  two  three  four
Colorado    4    5      6     7
Utah        8    9     10    11
```

[55]:
```
df_state[-2:]
```

[55]:
```
          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.

[56]:
```
df_state
```

[56]:
```
          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
```

[57]:
```
df_state.loc["Colorado"]
```

[57]:
```
one      4
two      5
three    6
four     7
Name: Colorado, dtype: int64
```

[58]:
```
df_state.loc["Utah"]
```

[58]:
```
one       8
two       9
three    10
four     11
Name: Utah, dtype: int64
```

[59]:
```
df_state.iloc[1]
```

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

[60]:
```
df_state
```

[60]:

	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

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

[61]:

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

[62]:
```
df_state.iloc[[2,0]]
```

[62]:

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

Filter data with conditions

[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 < 9
```

[64]:

	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
```

[65]:
```
df_state[df_state < 9]
```

[65]:
```
          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
```

[66]:
```
df_state[df_state < 9] = 9
```

[67]:
```
df_state
```

[67]:
```
          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
```

[68]:
```
df_state
```

[68]:
```
          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
```

[69]:
```
df_state.three== 10
```

[69]:
```
Ohio        False
Colorado    False
Utah         True
New York    False
Name: three, dtype: bool
```

[70]:
```
df_state[df_state.three==10]
```

[70]:
```
      one  two  three  four
Utah    9    9     10    11
```

try on the other `DataFrame`

[71]:
```
df_ward.head(2)
```

[71]:
```
   population  ward   poverty  year
0        1975     1  0.087530  2000
1        3581     2  0.497807  2000
```

[72]:
```
df_ward['population']
```

[72]:
```
0     1975
1     3581
2     1879
3     4918
4     3641
5     2300
6     3555
7     2807
8     3325
9     4907
10    4717
11     766
12    3857
13     367
14    1574
15    3894
16    4967
17    2649
18    3301
19    2446
20    3926
21    1512
22    3764
23    4221
24    4320
Name: population, dtype: int64
```

[73]:
```
df_ward.population
```

[73]:
```
0     1975
1     3581
2     1879
3     4918
4     3641
5     2300
6     3555
7     2807
8     3325
9     4907
10    4717
11     766
12    3857
13     367
14    1574
15    3894
```

```
16    4967
17    2649
18    3301
19    2446
20    3926
21    1512
22    3764
23    4221
24    4320
Name: population, dtype: int64
```

[74]:
```
df_ward.head(2)
```

[74]:
```
   population  ward   poverty  year
0        1975     1  0.087530  2000
1        3581     2  0.497807  2000
```

[75]:
```
df_ward[0:4]
```

[75]:
```
   population  ward   poverty  year
0        1975     1  0.087530  2000
1        3581     2  0.497807  2000
2        1879     3  0.149763  2000
3        4918     4  0.118433  2000
```

[76]:
```
df_ward[-4:]
```

[76]:
```
    population  ward   poverty  year
21        1512     2  0.417387  2004
22        3764     3  0.255953  2004
23        4221     4  0.516999  2004
24        4320     5  0.719540  2004
```

[77]:
```
df_ward[df_ward.ward==2]
```

[77]:
```
    population  ward   poverty  year
1         3581     2  0.497807  2000
6         3555     2  0.092769  2001
11         766     2  0.227251  2002
16        4967     2  0.673921  2003
21        1512     2  0.417387  2004
```

[78]:
```
df_ward[df_ward.population<1000]
```

[78]:
```
    population  ward   poverty  year
11         766     2  0.227251  2002
13         367     4  0.089816  2002
```

[79]:
```
df_ward[(df_ward.ward==2) & (df_ward.population < 1000)] # & binary operator to␣
↪perform and operation on lists of boolean values
```

[79]:

	population	ward	poverty	year
11	766	2	0.227251	2002

[80]:
```
(df_ward.ward==2) & (df_ward.population < 1000)
```

[80]:
```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11     True
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
dtype: bool
```

[81]:
```
df_ward[(df_ward.ward==2) | (df_ward.population < 1000)] # | binary operator to␣
↪perform or operation on lists of boolean values
```

[81]:

	population	ward	poverty	year
1	3581	2	0.497807	2000
6	3555	2	0.092769	2001
11	766	2	0.227251	2002
13	367	4	0.089816	2002
16	4967	2	0.673921	2003
21	1512	2	0.417387	2004

[82]:
```
df_ward[(~(df_ward.ward==2)) & (df_ward.population < 1000)] # not in ward 2 and␣
↪less than 1000 population
```

[82]:

	population	ward	poverty	year
13	367	4	0.089816	2002

[83]:
```
df_ward[~((df_ward.ward==2) & (df_ward.population < 1000))] # not (in ward 2
↪and less than 1000 population)
```

[83]:

	population	ward	poverty	year
0	1975	1	0.087530	2000
1	3581	2	0.497807	2000
2	1879	3	0.149763	2000
3	4918	4	0.118433	2000
4	3641	5	0.169706	2000
5	2300	1	0.663748	2001
6	3555	2	0.092769	2001
7	2807	3	0.851599	2001
8	3325	4	0.599091	2001
9	4907	5	0.453156	2001
10	4717	1	0.446826	2002
12	3857	3	0.642213	2002
13	367	4	0.089816	2002
14	1574	5	0.961492	2002
15	3894	1	0.981138	2003
16	4967	2	0.673921	2003
17	2649	3	0.568419	2003
18	3301	4	0.940607	2003
19	2446	5	0.608026	2003
20	3926	1	0.189879	2004
21	1512	2	0.417387	2004
22	3764	3	0.255953	2004
23	4221	4	0.516999	2004
24	4320	5	0.719540	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

[84]:
```
df_ward[(df_ward.ward==3) & (df_ward.population > 500) & (df_ward.poverty<0.4)]
```

[84]:
```
     population  ward   poverty  year
2          1879     3  0.149763  2000
22         3764     3  0.255953  2004
```

2.5 Creating New Columns in an existing `DataFrame`

[85]:
```
df_ward.head()
```

[85]:
```
   population  ward   poverty  year
0        1975     1  0.087530  2000
1        3581     2  0.497807  2000
2        1879     3  0.149763  2000
3        4918     4  0.118433  2000
4        3641     5  0.169706  2000
```

[86]:
```
pop_pov = df_ward.population * df_ward.poverty # elementwise operation similar to numpy array
pop_pov
```

[86]:
```
0      172.872128
1     1782.646067
2      281.404327
3      582.451801
4      617.898586
5     1526.619886
6      329.795376
7     2390.437651
8     1991.976026
9     2223.638472
10    2107.676601
11     174.073966
12    2477.013854
13      32.962500
14    1513.388857
15    3820.550765
16    3347.363400
17    1505.741032
18    3104.945246
19    1487.232331
20     745.465164
21     631.088736
22     963.407587
23    2182.254446
24    3108.412762
dtype: float64
```

[87]:
```
df_ward
```

[87]:

	population	ward	poverty	year
0	1975	1	0.087530	2000
1	3581	2	0.497807	2000
2	1879	3	0.149763	2000
3	4918	4	0.118433	2000
4	3641	5	0.169706	2000
5	2300	1	0.663748	2001
6	3555	2	0.092769	2001
7	2807	3	0.851599	2001
8	3325	4	0.599091	2001
9	4907	5	0.453156	2001
10	4717	1	0.446826	2002
11	766	2	0.227251	2002
12	3857	3	0.642213	2002
13	367	4	0.089816	2002
14	1574	5	0.961492	2002
15	3894	1	0.981138	2003
16	4967	2	0.673921	2003
17	2649	3	0.568419	2003
18	3301	4	0.940607	2003
19	2446	5	0.608026	2003
20	3926	1	0.189879	2004
21	1512	2	0.417387	2004
22	3764	3	0.255953	2004
23	4221	4	0.516999	2004
24	4320	5	0.719540	2004

[88]:
```
df_ward['pop_pov'] = pop_pov.astype('int')
```

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

[89]:

	population	ward	poverty	year	pop_pov
0	1975	1	0.087530	2000	172
1	3581	2	0.497807	2000	1782
2	1879	3	0.149763	2000	281
3	4918	4	0.118433	2000	582
4	3641	5	0.169706	2000	617

2.6 Aggregation/Groupby

[90]:
```
df_ward[df_ward.ward==1]
```

[90]:

	population	ward	poverty	year	pop_pov
0	1975	1	0.087530	2000	172
5	2300	1	0.663748	2001	1526
10	4717	1	0.446826	2002	2107
15	3894	1	0.981138	2003	3820

```
20        3926     1  0.189879  2004      745
```

[91]:
```
df_ward.groupby(by='ward').sum()
```

[91]:
```
      population   poverty   year  pop_pov
ward
1          16812  2.369121  10010     8370
2          14381  1.909134  10010     6263
3          14956  2.467946  10010     7616
4          16132  2.264946  10010     7891
5          16888  2.911921  10010     8948
```

[92]:
```
df_ward.groupby(by='ward').sum()[['population', 'pop_pov']]
```

[92]:
```
      population  pop_pov
ward
1          16812     8370
2          14381     6263
3          14956     7616
4          16132     7891
5          16888     8948
```

[93]:
```
ward_df = df_ward.groupby(by='ward').sum()[['population', 'pop_pov']]
```

[94]:
```
ward_df
```

[94]:
```
      population  pop_pov
ward
1          16812     8370
2          14381     6263
3          14956     7616
4          16132     7891
5          16888     8948
```

[95]:
```
ward_df['poverty'] = ward_df.pop_pov / ward_df.population
```

[96]:
```
ward_df
```

[96]:
```
      population  pop_pov   poverty
ward
1          16812     8370  0.497859
2          14381     6263  0.435505
3          14956     7616  0.509227
4          16132     7891  0.489152
5          16888     8948  0.529844
```

2.7 Joins/Merge

[97]:
```
ward_df
```

[97]:

ward	population	pop_pov	poverty
1	16812	8370	0.497859
2	14381	6263	0.435505
3	14956	7616	0.509227
4	16132	7891	0.489152
5	16888	8948	0.529844

[98]:
```
df_ward
```

[98]:

	population	ward	poverty	year	pop_pov
0	1975	1	0.087530	2000	172
1	3581	2	0.497807	2000	1782
2	1879	3	0.149763	2000	281
3	4918	4	0.118433	2000	582
4	3641	5	0.169706	2000	617
5	2300	1	0.663748	2001	1526
6	3555	2	0.092769	2001	329
7	2807	3	0.851599	2001	2390
8	3325	4	0.599091	2001	1991
9	4907	5	0.453156	2001	2223
10	4717	1	0.446826	2002	2107
11	766	2	0.227251	2002	174
12	3857	3	0.642213	2002	2477
13	367	4	0.089816	2002	32
14	1574	5	0.961492	2002	1513
15	3894	1	0.981138	2003	3820
16	4967	2	0.673921	2003	3347
17	2649	3	0.568419	2003	1505
18	3301	4	0.940607	2003	3104
19	2446	5	0.608026	2003	1487
20	3926	1	0.189879	2004	745
21	1512	2	0.417387	2004	631
22	3764	3	0.255953	2004	963
23	4221	4	0.516999	2004	2182
24	4320	5	0.719540	2004	3108

[99]:
```
df_all = df_ward.merge(ward_df,  on='ward')
```

[100]:
```
df_all
```

[100]:

	population_x	ward	poverty_x	year	pop_pov_x	population_y	pop_pov_y	\
0	1975	1	0.087530	2000	172	16812	8370	
1	2300	1	0.663748	2001	1526	16812	8370	

2	4717	1	0.446826	2002	2107	16812	8370
3	3894	1	0.981138	2003	3820	16812	8370
4	3926	1	0.189879	2004	745	16812	8370
5	3581	2	0.497807	2000	1782	14381	6263
6	3555	2	0.092769	2001	329	14381	6263
7	766	2	0.227251	2002	174	14381	6263
8	4967	2	0.673921	2003	3347	14381	6263
9	1512	2	0.417387	2004	631	14381	6263
10	1879	3	0.149763	2000	281	14956	7616
11	2807	3	0.851599	2001	2390	14956	7616
12	3857	3	0.642213	2002	2477	14956	7616
13	2649	3	0.568419	2003	1505	14956	7616
14	3764	3	0.255953	2004	963	14956	7616
15	4918	4	0.118433	2000	582	16132	7891
16	3325	4	0.599091	2001	1991	16132	7891
17	367	4	0.089816	2002	32	16132	7891
18	3301	4	0.940607	2003	3104	16132	7891
19	4221	4	0.516999	2004	2182	16132	7891
20	3641	5	0.169706	2000	617	16888	8948
21	4907	5	0.453156	2001	2223	16888	8948
22	1574	5	0.961492	2002	1513	16888	8948
23	2446	5	0.608026	2003	1487	16888	8948
24	4320	5	0.719540	2004	3108	16888	8948

	poverty_y
0	0.497859
1	0.497859
2	0.497859
3	0.497859
4	0.497859
5	0.435505
6	0.435505
7	0.435505
8	0.435505
9	0.435505
10	0.509227
11	0.509227
12	0.509227
13	0.509227
14	0.509227
15	0.489152
16	0.489152
17	0.489152
18	0.489152
19	0.489152
20	0.529844
21	0.529844

```
22   0.529844
23   0.529844
24   0.529844
```

[101]:
```
df_all = df_ward.merge(ward_df, on='ward',suffixes = ('_year', '_allyears'))
```

[102]:
```
df_all
```

[102]:

	population_year	ward	poverty_year	year	pop_pov_year
0	1975	1	0.087530	2000	172
1	2300	1	0.663748	2001	1526
2	4717	1	0.446826	2002	2107
3	3894	1	0.981138	2003	3820
4	3926	1	0.189879	2004	745
5	3581	2	0.497807	2000	1782
6	3555	2	0.092769	2001	329
7	766	2	0.227251	2002	174
8	4967	2	0.673921	2003	3347
9	1512	2	0.417387	2004	631
10	1879	3	0.149763	2000	281
11	2807	3	0.851599	2001	2390
12	3857	3	0.642213	2002	2477
13	2649	3	0.568419	2003	1505
14	3764	3	0.255953	2004	963
15	4918	4	0.118433	2000	582
16	3325	4	0.599091	2001	1991
17	367	4	0.089816	2002	32
18	3301	4	0.940607	2003	3104
19	4221	4	0.516999	2004	2182
20	3641	5	0.169706	2000	617
21	4907	5	0.453156	2001	2223
22	1574	5	0.961492	2002	1513
23	2446	5	0.608026	2003	1487
24	4320	5	0.719540	2004	3108

	population_allyears	pop_pov_allyears	poverty_allyears
0	16812	8370	0.497859
1	16812	8370	0.497859
2	16812	8370	0.497859
3	16812	8370	0.497859
4	16812	8370	0.497859
5	14381	6263	0.435505
6	14381	6263	0.435505
7	14381	6263	0.435505
8	14381	6263	0.435505
9	14381	6263	0.435505
10	14956	7616	0.509227

```
11                14956              7616          0.509227
12                14956              7616          0.509227
13                14956              7616          0.509227
14                14956              7616          0.509227
15                16132              7891          0.489152
16                16132              7891          0.489152
17                16132              7891          0.489152
18                16132              7891          0.489152
19                16132              7891          0.489152
20                16888              8948          0.529844
21                16888              8948          0.529844
22                16888              8948          0.529844
23                16888              8948          0.529844
24                16888              8948          0.529844
```

[103]:
```
df_all[df_all.poverty_year > df_all.poverty_allyears]
```

[103]:
```
    population_year  ward  poverty_year  year  pop_pov_year  \
1              2300     1      0.663748  2001          1526
3              3894     1      0.981138  2003          3820
5              3581     2      0.497807  2000          1782
8              4967     2      0.673921  2003          3347
11             2807     3      0.851599  2001          2390
12             3857     3      0.642213  2002          2477
13             2649     3      0.568419  2003          1505
16             3325     4      0.599091  2001          1991
18             3301     4      0.940607  2003          3104
19             4221     4      0.516999  2004          2182
22             1574     5      0.961492  2002          1513
23             2446     5      0.608026  2003          1487
24             4320     5      0.719540  2004          3108

    population_allyears  pop_pov_allyears  poverty_allyears
1                 16812              8370          0.497859
3                 16812              8370          0.497859
5                 14381              6263          0.435505
8                 14381              6263          0.435505
11                14956              7616          0.509227
12                14956              7616          0.509227
13                14956              7616          0.509227
16                16132              7891          0.489152
18                16132              7891          0.489152
19                16132              7891          0.489152
22                16888              8948          0.529844
23                16888              8948          0.529844
24                16888              8948          0.529844
```

Which ward has the highest average poverty rate?

[104]:
```
df_all.poverty_allyears.idxmax()
```

[104]:
```
20
```

[105]:
```
df_all.loc[df_all['poverty_allyears'].idxmax()]
```

[105]:
```
population_year         3641.000000
ward                       5.000000
poverty_year               0.169706
year                    2000.000000
pop_pov_year             617.000000
population_allyears    16888.000000
pop_pov_allyears        8948.000000
poverty_allyears           0.529844
Name: 20, dtype: float64
```

Which ward in which year has the lowest poverty rate?

[106]:
```
df_all.poverty_year.idxmin()
```

[106]:
```
0
```

[107]:
```
df_all.loc[df_all['poverty_year'].idxmin()]
```

[107]:
```
population_year         1975.000000
ward                       1.000000
poverty_year               0.087530
year                    2000.000000
pop_pov_year             172.000000
population_allyears    16812.000000
pop_pov_allyears        8370.000000
poverty_allyears           0.497859
Name: 0, 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

[108]:
```
df1 = pd.read_csv("ex1.csv")
df1
```

[108]:

	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.

[109]:
```
df2 = pd.read_csv("ex1.csv", header=None)
df2
```

[109]:

	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.

[110]:
```
df3 = pd.read_csv("ex1.csv", names=["col1", "col2", "col3", "col4", "col5"])
df3
```

[110]:

	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`

[111]:
```
df4 = pd.read_csv("ex1.csv", index_col="message")
df4
```

[111]:

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

[112]:
```
df4 = pd.read_csv("ex1.csv", index_col=4)
df4
```

[112]:

message	a	b	c	d

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

[113]:
```
df5 = pd.read_csv("ex1.csv", skiprows=[1,2])
df5
```

[113]:
```
    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.

[114]:
```
df6 = pd.read_csv("ex1.csv", skiprows=2)
df6
```

[114]:
```
    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).

[115]:
```
df_ex5 = pd.read_csv("ex5.csv")
df_ex5
```

[115]:
```
  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
```

[116]:
```
df_ex5 = pd.read_csv("ex5.csv", na_values=["one", 1])
df_ex5
```

[116]:
```
  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
```

[117]:
```
df_ex5.dropna() #Drop the rows where at least one element is missing.
```

[117]:
```
  something    a   b     c   d message
2     three  9.0  10  11.0  12     foo
```

[118]:
```
df_ex5.dropna(axis='columns') # Drop the columns where at least one element is␣
↪missing.
```

[118]:

	b	d
0	2	4
1	6	8
2	10	12

[119]:
```
df_ex5.dropna(subset=["something"]) #Define in which columns to look for␣
↪missing values.
```

[119]:

	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

[120]:
```
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