Lecture 7: Pandas

The Python library, not the cute Chinese bear

Import statements

```python
import pandas
import pandas as pd
from pandas import DataFrame

pandas.DataFrame
pd.DataFrame
DataFrame
```

Import statements

```
1  import pandas
2  import pandas as pd
3  from pandas import DataFrame
4
5  pandas.DataFrame
6  pd.DataFrame
7  DataFrame
```

Import statements

Correct by convention

```
import pandas
import pandas as pd
from pandas import DataFrame

pandas.DataFrame
pd.DataFrame
DataFrame
```

Our first DataFrame

```
data = {'month'   :[1, 2, 3],
        'unemp_il':[5.6, 5.7, 5.8],
        'unemp_wi':[6.1, 6.0, 6.1],
        'gdp_il'  :[6, 6, 7],
        'gdp_wi'  :[4, 5, 4]}
```

```
df = pd.DataFrame(data)
```

Our first DataFrame

```
data = {'month'   :[1, 2, 3],
        'unemp_il':[5.6, 5.7, 5.8],
        'unemp_wi':[6.1, 6.0, 6.1],
        'gdp_il'  :[6, 6, 7],
        'gdp_wi'  :[4, 5, 4]}
```

```
df = pd.DataFrame(data)
```

Library

Common name for a DataFrame instance when there is <u>only one</u> dataframe

Class object being called

Argument passed into the DataFrame classes __init__ method

Our first DataFrame

```
data = {'month'   :[1, 2, 3],
        'unemp_il':[5.6, 5.7, 5.8],
        'unemp_wi':[6.1, 6.0, 6.1],
        'gdp_il'  :[6, 6, 7],
        'gdp_wi'  :[4, 5, 4]}
```

```
df = pd.DataFrame(data)
```

```
In [12]: df
Out[12]:
   month  unemp_il  unemp_wi  gdp_il  gdp_wi
0      1       5.6       6.1       6       4
1      2       5.7       6.0       6       5
2      3       5.8       6.1       7       4
```

Anatomy of a DataFrame

Column names

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
0	1	5.6	6.1	6	4
1	2	5.7	6.0	6	5
2	3	5.8	6.1	7	4

Index names

Anatomy of a DataFrame

Instance of the pd.DataFrame class

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
0	1	5.6	6.1	6	4
1	2	5.7	6.0	6	5
2	3	5.8	6.1	7	4

Instance of the pd.Series class

Anatomy of a DataFrame

Instance of the str class

Instance of the pd.DataFrame class

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
0	1	5.6	6.1	6	4
1	2	5.7	6.0	6	5
2	3	5.8	6.1	7	4

Instance of the pd.Series class

Instance of the float class

Subsetting a dataframe

- Select by column
- Select by index (row)
- Select by column and index

- Select by conditionals

Selecting by column

```
   month  unemp_il  unemp_wi  gdp_il  gdp_wi
0      1       5.6       6.1       6       4
1      2       5.7       6.0       6       5
2      3       5.8       6.1       7       4
```

```
In [13]: df['month']
Out[13]:
0    1
1    2
2    3
Name: month, dtype: int64
```

```
In [14]: df[['unemp_il', 'gdp_il']]
Out[14]:
   unemp_il  gdp_il
0       5.6       6
1       5.7       6
2       5.8       7
```

Selecting by column

```
   month  unemp_il  unemp_wi  gdp_il  gdp_wi
0      1       5.6       6.1       6       4
1      2       5.7       6.0       6       5
2      3       5.8       6.1       7       4
```

```
In [13]: df['month']
Out[13]:
0    1
1    2
2    3
Name: month, dtype: int64
```

Why the double-square-brackets?

```
In [14]: df[['unemp_il', 'gdp_il']]
Out[14]:
   unemp_il  gdp_il
0       5.6       6
1       5.7       6
2       5.8       7
```

Selecting by column

```
   month  unemp_il  unemp_wi  gdp_il  gdp_wi
0      1       5.6       6.1       6       4
1      2       5.7       6.0       6       5
2      3       5.8       6.1       7       4
```

Selecting one column takes a string, but selecting multiple is done with a list

```
In [15]: cols = ['unemp_wi', 'gdp_wi']
    ...: df[cols]
Out[15]:
   unemp_wi  gdp_wi
0       6.1       4
1       6.0       5
2       6.1       4
```

Selecting by column

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
0	1	5.6	6.1	6	4
1	2	5.7	6.0	6	5
2	3	5.8	6.1	7	4

Can be a good spot for a list comprehension

```
In [16]: df[[c for c in df.columns if c.endswith('wi')]]
Out[16]:
   unemp_wi  gdp_wi
0       6.1       4
1       6.0       5
2       6.1       4
```

Selecting by column

```
   month  unemp_il  unemp_wi  gdp_il  gdp_wi
0      1       5.6       6.1       6       4
1      2       5.7       6.0       6       5
2      3       5.8       6.1       7       4
```

DataFrame instances have a “columns” attribute that we can iterate over

Can be a good spot for a list comprehension

```
In [16]: df[[c for c in df.columns if c.endswith('wi')]]
Out[16]:
   unemp_wi  gdp_wi
0       6.1       4
1       6.0       5
2       6.1       4
```

Applying operations over columns

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
0	1	5.6	6.1	6	4
1	2	5.7	6.0	6	5
2	3	5.8	6.1	7	4

We can apply nearly any **function** that takes a list-like object to a column

```
In [17]: import numpy as np
    ...: np.mean(df['unemp_il'])
Out[17]: 5.7
```

Applying operations over columns

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
0	1	5.6	6.1	6	4
1	2	5.7	6.0	6	5
2	3	5.8	6.1	7	4

We can apply nearly any **function** that takes a list-like object to a column

```
In [17]: import numpy as np
    ...: np.mean(df['unemp_il'])
Out[17]: 5.7
```

But in most cases DataFrames also have their own **methods**

```
In [18]: df['unemp_il'].mean()
Out[18]: 5.7
```

Selecting by rows

```
   month  unemp_il  unemp_wi  gdp_il  gdp_wi
0      1       5.6       6.1       6       4
1      2       5.7       6.0       6       5
2      3       5.8       6.1       7       4
```

Same slicing notation as with lists

```
In [19]: df[1:]
Out[19]:
   month  unemp_il  unemp_wi  gdp_il  gdp_wi
1      2       5.7       6.0       6       5
2      3       5.8       6.1       7       4
```

Selecting by rows

Renamed index

```
In [21]: df.index = ['a', 'b', 'c']
    ...: df
Out[21]:
   month  unemp_il  unemp_wi  gdp_il  gdp_wi
a      1       5.6       6.1       6       4
b      2       5.7       6.0       6       5
c      3       5.8       6.1       7       4
```

Selecting by rows

Using the ".loc" method for row ***names***

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
a	1	5.6	6.1	6	4
b	2	5.7	6.0	6	5
c	3	5.8	6.1	7	4

```
In [22]: df.loc['a']
Out[22]:
month        1.0
unemp_il     5.6
unemp_wi     6.1
gdp_il       6.0
gdp_wi       4.0
Name: a, dtype: float64
```

```
In [23]: df.loc[['a', 'c']]
Out[23]:
```

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
a	1	5.6	6.1	6	4
c	3	5.8	6.1	7	4

Selecting by rows

Using the ".iloc" method for row ***positions***

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
a	1	5.6	6.1	6	4
b	2	5.7	6.0	6	5
c	3	5.8	6.1	7	4

```
In [24]: df.iloc[0]
Out[24]:
month        1.0
unemp_il     5.6
unemp_wi     6.1
gdp_il       6.0
gdp_wi       4.0
Name: a, dtype: float64
```

```
In [25]: df.iloc[[0, 2]]
Out[25]:
```

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
a	1	5.6	6.1	6	4
c	3	5.8	6.1	7	4

Selecting by rows and columns

	month	unemp_il	unemp_wi	gdp_il	gdp_wi
a	1	5.6	6.1	6	4
b	2	5.7	6.0	6	5
c	3	5.8	6.1	7	4

Using ".loc" to select by row names and column names

```
In [26]: df.loc[['a', 'c'], ['unemp_il', 'gdp_il']]
Out[26]:
   unemp_il  gdp_il
a       5.6       6
c       5.8       7
```

Rows first, columns second

Using ".iloc" to select by row positions and column positions

```
In [27]: df.iloc[[0, 2], [1, 3]]
Out[27]:
   unemp_il  gdp_il
a       5.6       6
c       5.8       7
```

Selecting by columns

```
In [35]: df1 = df[['unemp_il', 'unemp_wi']]
    ...: df1
Out[35]:
   unemp_il  unemp_wi
a       5.6       6.1
b       5.7       6.0
c       5.8       6.1
```

Notation for “all”, in this case rows

```
In [36]: df2 = df.loc[:, ['unemp_il', 'unemp_wi']]
    ...: df2
Out[36]:
   unemp_il  unemp_wi
a       5.6       6.1
b       5.7       6.0
c       5.8       6.1
```

View vs Full Copy

```
In [38]: df1['il_rescaled'] = df1['unemp_il'] * .01
__main__:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

?

```
In [39]: df2['il_rescaled'] = df2['unemp_il'] * .01
```

View vs Full Copy

```
In [38]: df1['il_rescaled'] = df1['unemp_il'] * .01
__main__:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

?

```
In [39]: df2['il_rescaled'] = df2['unemp_il'] * .01
```

df1 created with [] notation passing a list
df2 created with the .loc method

df1 is a view
df2 is a full copy

View vs Full Copy

```
In [38]: df1['il_rescaled'] = df1['unemp_il'] * .01
__main__:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
In [39]: df2['il_rescaled'] = df2['unemp_il'] * .01
```

```
In [40]: df1
Out[40]:
   unemp_il  unemp_wi  il_rescaled
a       5.6       6.1        0.056
b       5.7       6.0        0.057
c       5.8       6.1        0.058
```

```
In [41]: df2
Out[41]:
   unemp_il  unemp_wi  il_rescaled
a       5.6       6.1        0.056
b       5.7       6.0        0.057
c       5.8       6.1        0.058
```

```
df['col 1']
df.loc[:, 'col 1']
df.iloc[:, 0]
```

```
df[:1]
df.iloc[0]
df.loc['a']
```

	"col 1"	"col 2"	"col 3"
"a"	1	4	7
"b"	2	5	8
"c"	3	6	9

```
df.columns
```

```
df.index
```

```
df.loc['a', 'col 1']
df.iloc[0, 0]
```

	“col 1”	“col 2”	“col 3”
“a”	1	4	7
“b”	2	5	8
“c”	3	6	9

```
df.loc[[‘a’, ‘b’], [‘col 1’, ‘col 2’]]
df.iloc[[0, 1], [0, 1]]
```