# Lecture 8: Pandas 2

The continuing saga of the cute Chinese bear

# Contents

- Defining path statements
- Reading from CSV
- Basic exploration
- Dates
- Renaming
- Subsetting by conditionals
- Applying functions using map, apply, and applymap

# Path statements

Absolute path:

```
4   path = r'c:\users\jeff levy\downloads\GDPC1.csv'
```

Relative path:

```
4   path = 'GDPC1.csv'
```

Relative to what?

# Path statements

Absolute path:

```
4    path = r'c:\users\jeff levy\downloads\GDPC1.csv'
```

Relative path:

```
4    path = 'GDPC1.csv'
```

Relative to what?

```
In [5]: os.getcwd()
Out[5]: 'C:\\Users\\Jeff Levy\\Documents\\WPy64-3940\\scripts'
```

# Raw strings

Absolute path: ?

```
4   path = r'c:\users\jeff levy\downloads\GDPC1.csv'
```

Relative path:

```
4   path = 'GDPC1.csv'
```

```
In [49]: print('Hello world!\nHow are you?\n')
Hello world!
How are you?
```

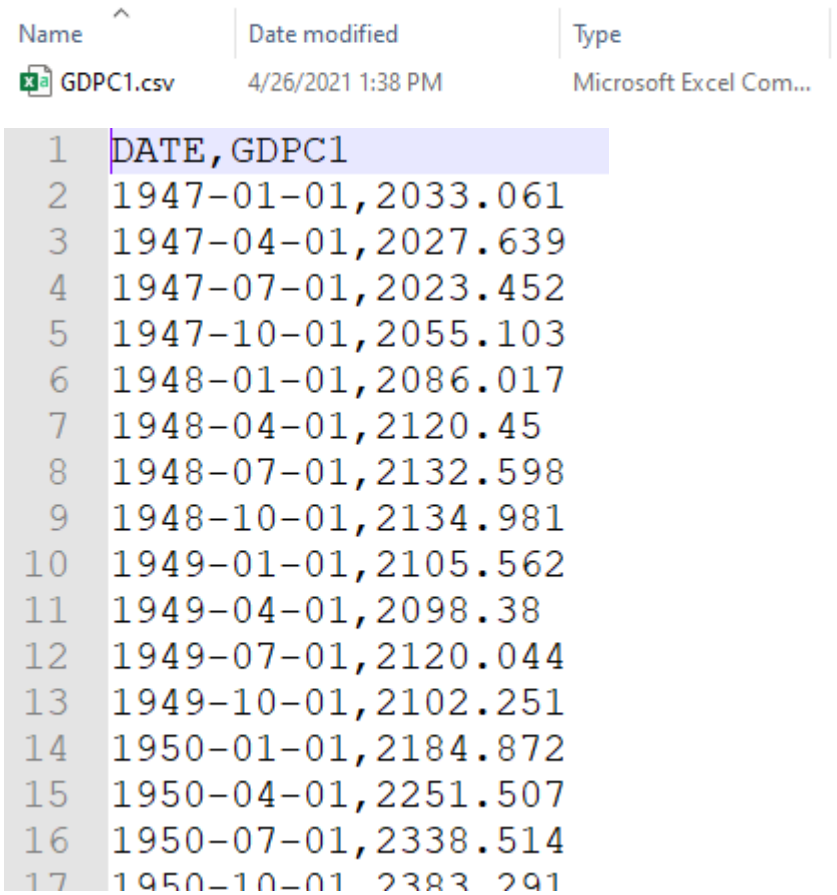
```
In [50]: print(r'Hello world!\nHow are you?\n')
Hello world!\nHow are you?\n
```

# Path statements

```
 9   import os
10   base_path = r'c:\users\jeff levy\downloads'
11   path = os.path.join(base_path, 'GDPC1.csv')
```

```
In [55]: path
Out[55]: 'c:\\users\\jeff levy\\downloads\\GDPC1.csv'
```

# Reading from CSV



| Name | Date modified | Type |
| --- | --- | --- |
| GDPC1.csv | 4/26/2021 1:38 PM | Microsoft Excel Com... |

```
1  DATE,GDPC1
2  1947-01-01,2033.061
3  1947-04-01,2027.639
4  1947-07-01,2023.452
5  1947-10-01,2055.103
6  1948-01-01,2086.017
7  1948-04-01,2120.45
8  1948-07-01,2132.598
9  1948-10-01,2134.981
10 1949-01-01,2105.562
11 1949-04-01,2098.38
12 1949-07-01,2120.044
13 1949-10-01,2102.251
14 1950-01-01,2184.872
15 1950-04-01,2251.507
16 1950-07-01,2338.514
17 1950-10-01,2383.291
```

```
In [56]: df = pd.read_csv(path)
    ...: df
Out[56]:
            DATE      GDPC1
0     1947-01-01   2033.061
1     1947-04-01   2027.639
2     1947-07-01   2023.452
3     1947-10-01   2055.103
4     1948-01-01   2086.017
..           ...        ...
291   2019-10-01  19253.959
292   2020-01-01  19010.848
293   2020-04-01  17302.511
294   2020-07-01  18596.521
295   2020-10-01  18794.426

[296 rows x 2 columns]
```

# Reading from CSV

## pandas.read_csv¶

pandas.read_csv(*filepath_or_buffer, sep=<object object>, delimiter=None, header='infer', names=None, index_col=None, usecols=None, squeeze=False, prefix=None, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=False, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, dialect=None, error_bad_lines=True, warn_bad_lines=True, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None, storage_options=None*)

Read a comma-separated values (csv) file into DataFrame. **[source]**

Also supports optionally iterating or breaking of the file into chunks.

Additional help can be found in the online docs for IO Tools.

# Basic DataFrame exploration

```
In [57]: df.head()
Out[57]:
         DATE     GDPC1
0  1947-01-01  2033.061
1  1947-04-01  2027.639
2  1947-07-01  2023.452
3  1947-10-01  2055.103
4  1948-01-01  2086.017
```

```
In [58]: df.tail()
Out[58]:
         DATE     GDPC1
291  2019-10-01  19253.959
292  2020-01-01  19010.848
293  2020-04-01  17302.511
294  2020-07-01  18596.521
295  2020-10-01  18794.426
```

```
In [69]: df.shape
Out[69]: (296, 2)
```

# Data Types and Dates

```
In [59]: df.dtypes
Out[59]:
DATE       object
GDPC1     float64
dtype: object
```

Object means string or other

Why isn’t our date a date?

# Data Types and Dates

```
In [63]: df['DATE_2'] = pd.to_datetime(df['DATE'])
    ...: df.head()
Out[63]:
         DATE     GDPC1     DATE_2
0  1947-01-01  2033.061 1947-01-01
1  1947-04-01  2027.639 1947-04-01
2  1947-07-01  2023.452 1947-07-01
3  1947-10-01  2055.103 1947-10-01
4  1948-01-01  2086.017 1948-01-01
```

```
In [64]: df.dtypes
Out[64]:
DATE                  object
GDPC1                float64
DATE_2    datetime64[ns]
dtype: object
```

# Data Types and Dates

```
In [67]: df = pd.read_csv(path, parse_dates=['DATE'])
    ...: df.head()
Out[67]:
        DATE     GDPC1
0 1947-01-01  2033.061
1 1947-04-01  2027.639
2 1947-07-01  2023.452
3 1947-10-01  2055.103
4 1948-01-01  2086.017
```

```
In [68]: df.dtypes
Out[68]:
DATE      datetime64[ns]
GDPC1            float64
dtype: object
```

# Renaming

```
In [64]: df.head()
Out[64]:
        DATE      GDPC1
0 1947-01-01   2033.061
1 1947-04-01   2027.639
2 1947-07-01   2023.452
3 1947-10-01   2055.103
4 1948-01-01   2086.017
```

```
34    names = {'DATE':'date', 'GDPC1':'gdp'}
35    df = df.rename(names, axis=1)
```

```
In [68]: df.head()
Out[68]:
        date       gdp
0 1947-01-01   2033.061
1 1947-04-01   2027.639
2 1947-07-01   2023.452
3 1947-10-01   2055.103
4 1948-01-01   2086.017
```

# Renaming

```
In [64]: df.head()
Out[64]:
         DATE     GDPC1
0 1947-01-01  2033.061
1 1947-04-01  2027.639
2 1947-07-01  2023.452
3 1947-10-01  2055.103
4 1948-01-01  2086.017
```

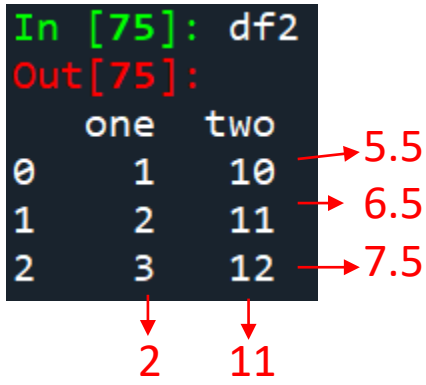
```
34  names = {'DATE':'date', 'GDPC1':'gdp'}
35  df = df.rename(names, axis=1)
```

?

```
In [68]: df.head()
Out[68]:
         date       gdp
0 1947-01-01  2033.061
1 1947-04-01  2027.639
2 1947-07-01  2023.452
3 1947-10-01  2055.103
4 1948-01-01  2086.017
```

# Interlude: DataFrame axis

Estimate the mean…
but in which direction?



# Interlude: DataFrame axis

```
In [75]: df2
Out[75]:
   one  two
0    1   10
1    2   11
2    3   12
```

Axis is a kwarg with a default value of 0

```
In [76]: df2.mean()
Out[76]:
one      2.0
two     11.0
dtype: float64
```

```
In [77]: df2.mean(axis=1)
Out[77]:
0    5.5
1    6.5
2    7.5
dtype: float64
```

# Renaming

```
In [64]: df.head()
Out[64]:
         DATE     GDPC1
0 1947-01-01  2033.061
1 1947-04-01  2027.639
2 1947-07-01  2023.452
3 1947-10-01  2055.103
4 1948-01-01  2086.017
```

```
34  names = {'DATE':'date', 'GDPC1':'gdp'}
35  df = df.rename(names, axis=1)
```

```
In [68]: df.head()
Out[68]:
         date       gdp
0 1947-01-01  2033.061
1 1947-04-01  2027.639
2 1947-07-01  2023.452
3 1947-10-01  2055.103
4 1948-01-01  2086.017
```

Alternatively:

```
43  df = df.rename(columns=names)
```

# Subsetting by conditionals

```
In [78]: df.head()
Out[78]:
         date        gdp
0 1947-01-01   2033.061
1 1947-04-01   2027.639
2 1947-07-01   2023.452
3 1947-10-01   2055.103
4 1948-01-01   2086.017
```

```
In [79]: df.tail()
Out[79]:
           date        gdp
291 2019-10-01  19253.959
292 2020-01-01  19010.848
293 2020-04-01  17302.511
294 2020-07-01  18596.521
295 2020-10-01  18794.426
```

# Subsetting by conditionals

```
In [78]: df.head()
Out[78]:
         date        gdp
0 1947-01-01   2033.061
1 1947-04-01   2027.639
2 1947-07-01   2023.452
3 1947-10-01   2055.103
4 1948-01-01   2086.017
```

```
In [79]: df.tail()
Out[79]:
           date        gdp
291 2019-10-01  19253.959
292 2020-01-01  19010.848
293 2020-04-01  17302.511
294 2020-07-01  18596.521
295 2020-10-01  18794.426
```

```
In [80]: df[df['gdp'] > 10000]
Out[80]:
           date        gdp
187 1993-10-01  10091.049
188 1994-01-01  10188.954
189 1994-04-01  10327.019
190 1994-07-01  10387.382
191 1994-10-01  10506.372
..          ...        ...
291 2019-10-01  19253.959
292 2020-01-01  19010.848
293 2020-04-01  17302.511
294 2020-07-01  18596.521
295 2020-10-01  18794.426

[109 rows x 2 columns]
```

# Subsetting by conditionals

```
In [80]: df[df['gdp'] > 10000]
Out[80]:
           date          gdp
187 1993-10-01   10091.049
188 1994-01-01   10188.954
189 1994-04-01   10327.019
190 1994-07-01   10387.382
191 1994-10-01   10506.372
..          ...         ...
291 2019-10-01   19253.959
292 2020-01-01   19010.848
293 2020-04-01   17302.511
294 2020-07-01   18596.521
295 2020-10-01   18794.426

[109 rows x 2 columns]
```

```
In [85]: df['gdp'] > 10000
Out[85]:
0      False
1      False
2      False
3      False
4      False
       ...
291     True
292     True
293     True
294     True
295     True
Name: gdp, Length: 296, dtype: bool
```

# Subsetting by conditionals

```
In [86]: df2
Out[86]:
   one  two
0    1   10
1    2   11
2    3   12
```

```
In [87]: df2[[True, False, True]]
Out[87]:
   one  two
0    1   10
2    3   12
```

# Subsetting by conditionals

```
54  import datetime
55  start_date = datetime.datetime(1999, 1, 1)
```

```
In [90]: df[df['date'] > start_date]
Out[90]:
           date         gdp
209 1999-04-01   12498.694
210 1999-07-01   12662.385
211 1999-10-01   12877.593
212 2000-01-01   12924.179
213 2000-04-01   13160.842
..          ...         ...
```

# Subsetting by conditionals

```
In [91]: df[(df['date'] > start_date) & (df['gdp'] < 17000)]
Out[91]:
          date         gdp
209 1999-04-01   12498.694
210 1999-07-01   12662.385
211 1999-10-01   12877.593
212 2000-01-01   12924.179
213 2000-04-01   13160.842
..          ...         ...
265 2013-04-01   16403.180
266 2013-07-01   16531.685
267 2013-10-01   16663.649
268 2014-01-01   16616.540
269 2014-04-01   16841.475

[61 rows x 2 columns]
```

# Function applied to each cell in one column

```
In [97]: df
Out[97]:
   col1  col2  col3
a     1     5     9
b     2     6    10
c     x     7    11
d     4     8    12
```

```
68    def some_math(x):
69        return x * 2
```

```
In [101]: df['col1'].map(some_math)
Out[101]:
a     2
b     4
c    xx
d     8
Name: col1, dtype: object
```

**Map**: applies a function to values in one column, one at a time

# Function applied to every column

```
74   df.loc['c', 'col1'] = 3
75
76   def zscore(x):
77       return (x - x.mean()) / x.std()
```

```
In [104]: df.apply(zscore)
Out[104]:
       col1      col2      col3
a -1.161895 -1.161895 -1.161895
b -0.387298 -0.387298 -0.387298
c  0.387298  0.387298  0.387298
d  1.161895  1.161895  1.161895
```

**Apply**: applies a function to values in every column, one entire column at a time

# Function applied to all cells in all columns

```
In [105]: df.applymap(some_math)
Out[105]:
   col1  col2  col3
a     2    10    18
b     4    12    20
c     6    14    22
d     8    16    24
```

**Applymap**: applies a function to values in all columns, one cell at a time

# Anonymous functions

```
def my_func(a, b):
    return (a + b) / 2


lambda a, b: (a + b) / 2
```

# Anonymous functions

```
def my_func(a, b):
    return (a + b) / 2


lambda a, b: (a + b) / 2
```

# Anonymous functions

```
def my_func(a, b):
    return (a + b) / 2


lambda a, b: (a + b) / 2
```

# Anonymous functions

??

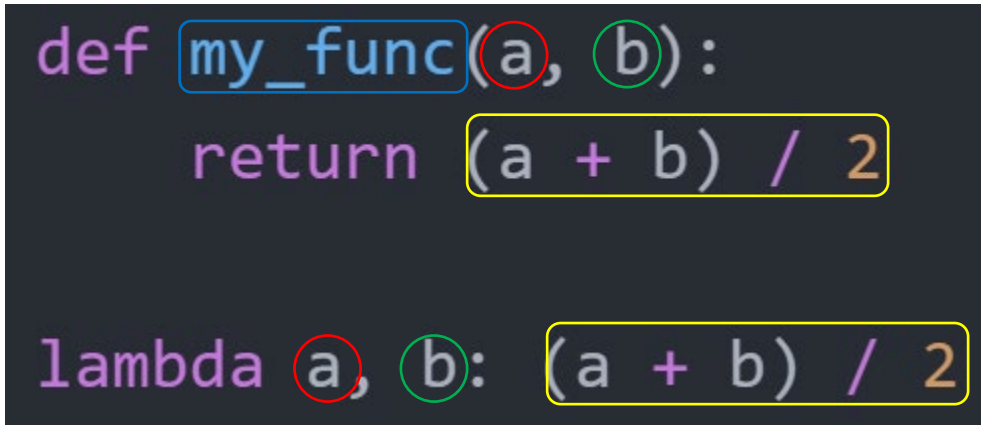
```
def my_func(a, b):
    return (a + b) / 2


lambda a, b: (a + b) / 2
```

Where did the function name go?

# Anonymous functions

??




Where did the function name go?

```
my_func = lambda a, b: (a + b) / 2
```

Would make the two identical, but this is incorrect usage of a lambda according to PEP8.

# Anonymous functions

```
76  def zscore(x):
77      return (x - x.mean()) / x.std()
```

```
In [104]: df.apply(zscore)
Out[104]:
       col1      col2      col3
a -1.161895 -1.161895 -1.161895
b -0.387298 -0.387298 -0.387298
c  0.387298  0.387298  0.387298
d  1.161895  1.161895  1.161895
```

```
In [20]: df.apply(lambda x: (x - x.mean()) / x.std())
Out[20]:
       col1      col2      col3
a -1.161895 -1.161895 -1.161895
b -0.387298 -0.387298 -0.387298
c  0.387298  0.387298  0.387298
d  1.161895  1.161895  1.161895
```

# Map

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

```
df[‘col1’].map(fn)
```

# Apply axis=0

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

```
df.apply(fn, axis=0)
```

# Apply axis=1

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

```
df.apply(fn, axis=1)
```

# Applymap

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

```
df.applymap(fn)
```

# Map, Apply, Applymap

```
In [106]: df
Out[106]:
   col1  col2  col3
a     1     5     9
b     2     6    10
c     3     7    11
d     4     8    12
```

```
68    def some_math(x):
69        return x * 2
```

```
76    def zscore(x):
77        return (x - x.mean()) / x.std()
```

```
68    def some_math(x):
69        return x * 2
```

**Map: argument x takes on each value of a cell**

**Apply: argument x takes on the value of one entire column or row**

**Applymap: argument x takes on each value of a cell**