COMP9321 Data Services Engineering

Term1, 2024

Week 3: Data Wrangling/Pre-processing

Removing Unnecessary Data

- Some times you don’t need all the data in the tables so it might help you achieve better performance if you remove the irrelevant data.
- Some columns or rows might be useless for you in the analysis due to having many missing values and replacing them with default values would produce wrong insights.
- Sometimes you are restricted from storage capacity perspective and hence you need to keep what is relevant to the job and drop the others.
- Python has a very good function Drop() to help you with this

Dropping Columns/Raws with NaN values

- Dropping Columns with all NaN values

Example:

```
 ohio  Colorado  Utah
0   NaN        12    11
1   NaN        33     7
2   NaN        44     4
3   NaN        32    22
```

data.dropna(axis=1, how=’all’)

```
 Colorado  Utah
0        12    11
1        33     7
2        44     4
3        32    22
```

*http://www.developintelligence.com/blog/2017/08/data-cleaning-pandas-python/

** https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html

Dropping Columns/Raws with NaN values

- Dropping Raws with all NaN values

Example:

```
  ohio  Colorado  Utah
0   NaN        NaN   NaN
1  12.0      33.0   7.0
2  23.0      44.0   4.0
3   34.0      32.0  22.0
```

```
data2.dropna(axis=0, how='all')
```

```
  ohio  Colorado  Utah
1  12.0      33.0   7.0
2  23.0      44.0   4.0
3   34.0      32.0  22.0
```

***http://www.developintelligence.com/blog/2017/08/data-cleaning-pandas-python/**

**** https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html**

Dropping Columns that are not needed

Example:

```
  ohio  Colorado  Utah
0   NaN       NaN   NaN
1  12.0      33.0   7.0
2  23.0      44.0   4.0
3   34.0      32.0  22.0
```

```
to_drop = [‘ohio’, ‘Utah’]
data2.drop(to_drop, inplace=True, axis=1)
```

```
Colorado
1   33.0
2   44.0
3    32.0
```

*http://www.developintelligence.com/blog/2017/08/data-cleaning-pandas-python/

** https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html

Dropping Rows on a Condition

- To drop a row based on a condition you use

```
df = df.drop(df[<some boolean condition>].index)
```

Example:

```
  ohio  Colorado  Utah
0   32         0      10.0
1  12.0      33.0   7.0
2  23.0      44.0   4.0
3   34.0     32.0   22.0
```

```
df.drop(df[df.Colorado == 0].index,
inplace=True)
```

```
      ohio  Colorado  Utah
1    12.0      33.0   7.0
2    23.0      44.0   4.0
3    34.0     32.0   22.0
```

*http://www.developintelligence.com/blog/2017/08/data-cleaning-pandas-python/

** https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html

Dropping Duplicate Rows

- To drop duplicate rows we use drop_duplicates function

Example:

```
  ohio  Colorado  Utah
0   32         0       10.0
1  12.0      33.0   7.0
2  23.0      44.0   4.0
3   34.0     32.0   22.0
4   12.0      33.0   7.0
```

```
df.drop_duplicates()
```

```
      ohio  Colorado  Utah
0   32          0       10.0
1    12.0      33.0   7.0
2   23.0      44.0   4.0
3   34.0     32.0   22.0
```

*http://www.developintelligence.com/blog/2017/08/data-cleaning-pandas-python/

** https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html

Filling Missing Values

- Sometimes you have in mind a suitable value (e.g., default, mean…etc) that would be useful to replace the NaN values. You can use df.fillna()

Example:

```
  ohio  Colorado  Utah
0   NaN       66.0   30.0
1  12.0      33.0   7.0
2  23.0      NaN   4.0
3   34.0     32.0  22.0
```

Default_values = {“ohio": 10, " Colorado ": 11, "Utah ": 22}

df.fillna(value=Default_values)

```
    ohio    Colorado  Utah
0   10     66.0       30.0
1  12.0      33.0     7.0
2  23.0      11        4.0
3   34.0      32.0    22.0
```

*http://www.developintelligence.com/blog/2017/08/data-cleaning-pandas-python/

** https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html

Formatting data

- Data read from source may not have the correct format (e.g., reading integer as a string)
- Some strings in the data have spacing which might not play well with your analysis at some point.
- The date/time format may not appropriate for your analysis
- Some times the data is generated by a computer program, so it probably has some computer-generated column names, too. Those can be hard to read and understand while working.

Formatting data Examples

- Example1 (change data type on read):

```
df = pd.read_csv(‘mydata.csv’, dtype={‘Integer_Column’: int})
```

- Example2 (change data type in dataframe)

```
df['column'] = df['column'].to_numeric()
df['column'] = df['column'].astype(str)
```

- Example3 (Spacing within the values):

```
data[‘Column_with_spacing’].str.strip()
```

Formatting data Examples

- Example4 (unnecessary time item in the date field):

df[‘MonthYear’] = pd.to_datetime(df[‘MonthYear’])

df[‘MonthYear’] = df[‘MonthYear’].apply(lambda x: x.date())

- Example5 (rename columns)

data = data.rename(columns = {‘Bad_Name1’:Better_Name1’, ‘Bad_Name2’:’Better_name2’})

Manipulating the data

- Merging Data
- Applying a function to data
- Pivot tables
- Change the index of a dataframe
- Groupby

Merging Data

- Sometimes in order to have complete dataset you need to Concatenate two datasets when reading from source.

Example:

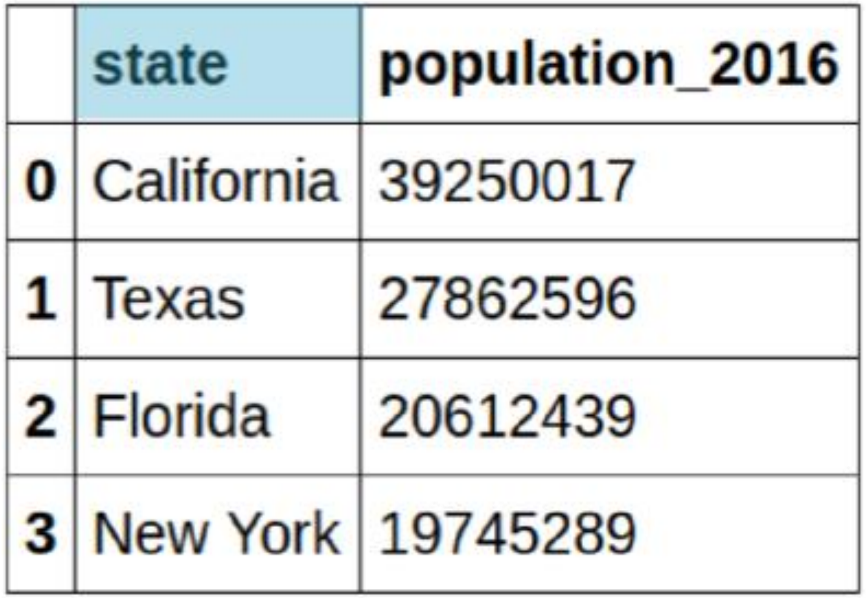
```
Dataset1=pd.read_csv(‘datasets/project1/dataset1.csv’)
Dataset2=pd.read_csv(‘datasets/project1/dataset2.csv’)


Full_data=pd.concat[Dataset1, Dataset2], ignore_index=True)
```

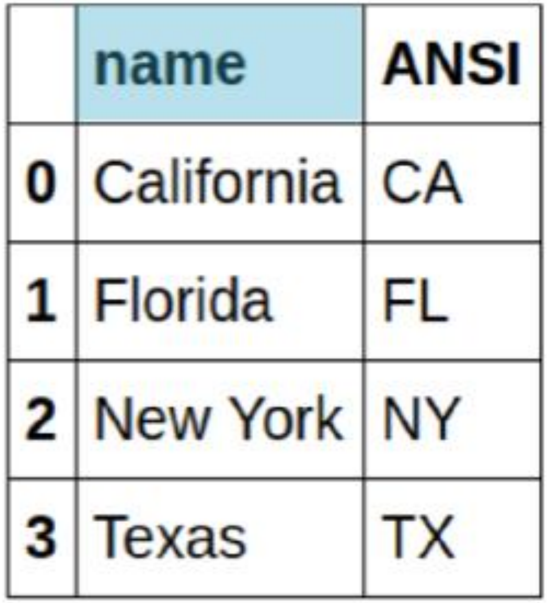
https://pandas.pydata.org/docs/user_guide/merging.html

Merging Data (Cont’d)

- Sometimes in order to have complete dataset you need to merge two Dataframes



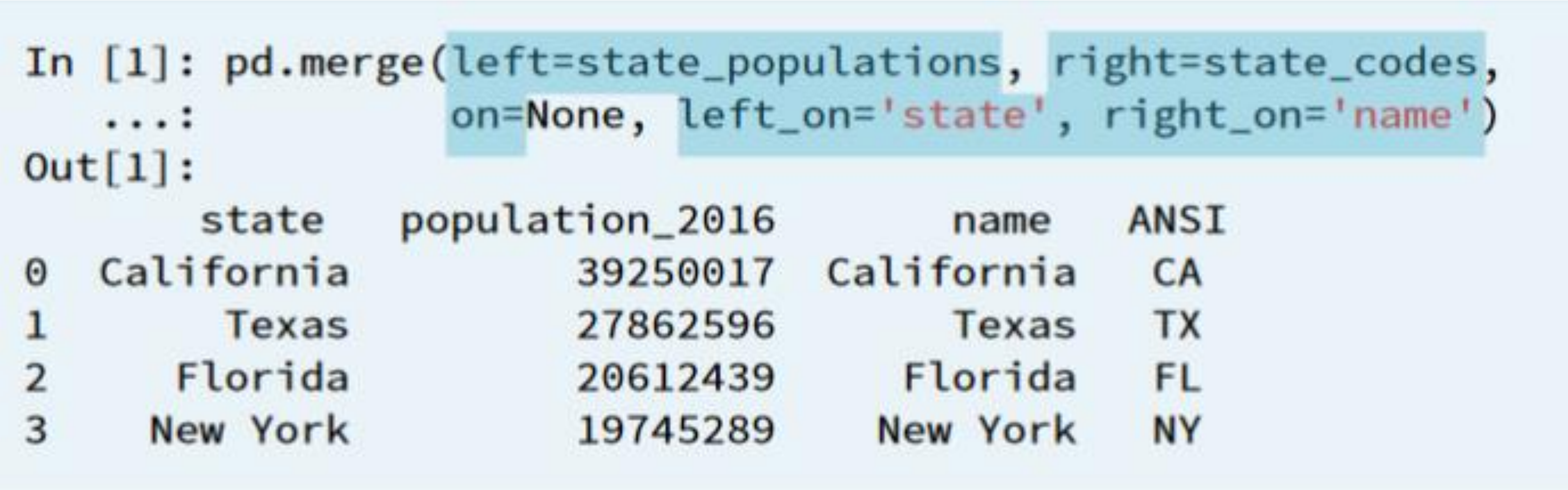
	state	population_2016
0	California	39250017
1	Texas	27862596
2	Florida	20612439
3	New York	19745289



	name	ANSI
0	California	CA
1	Florida	FL
2	New York	NY
3	Texas	TX

https://s3.amazonaws.com/assets.datacamp.com/production/course_3639/slides/ch3_slides.pdf

Merging Data (Cont’d)



```
In [1]: pd.merge(left=state_populations, right=state_codes,
   ...:          on=None, left_on='state', right_on='name')
Out[1]:
        state  population_2016        name ANSI
0  California         39250017  California   CA
1       Texas         27862596       Texas   TX
2     Florida         20612439     Florida   FL
3    New York         19745289    New York   NY
```

https://s3.amazonaws.com/assets.datacamp.com/production/course_3639/slides/ch3_slides.pdf

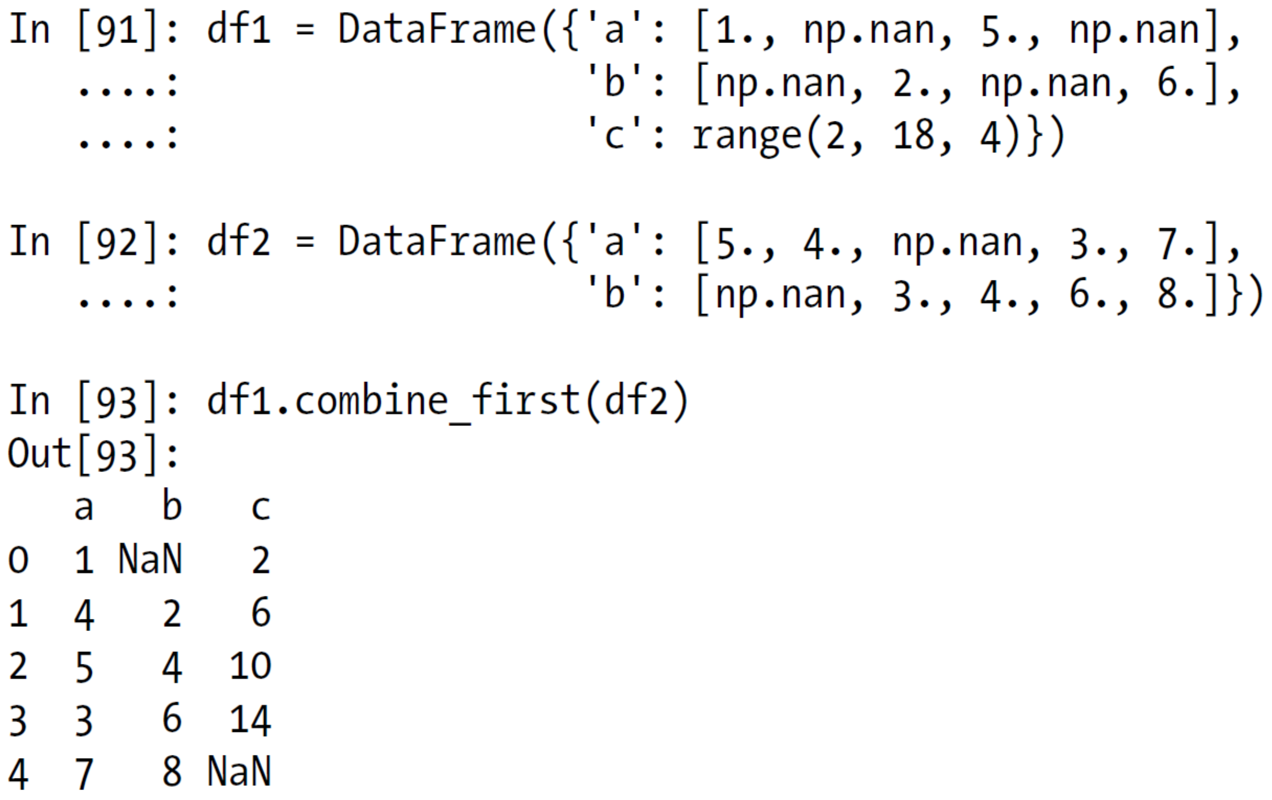
Merging Data (Pandas Vs SQL)

Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames
cross	CROSS JOIN	Create the cartesian product of rows of both frames

https://pandas.pydata.org/docs/user_guide/merging.html

Patching your Data

combine_first can do some sort of “patching” missing data in the calling object with data from the object you pass



```
In [91]: df1 = DataFrame({'a': [1., np.nan, 5., np.nan],
   ....:                  'b': [np.nan, 2., np.nan, 6.],
   ....:                  'c': range(2, 18, 4)})

In [92]: df2 = DataFrame({'a': [5., 4., np.nan, 3., 7.],
   ....:                  'b': [np.nan, 3., 4., 6., 8.]})

In [93]: df1.combine_first(df2)
Out[93]:
   a    b    c
0  1  NaN    2
1  4    2    6
2  5    4   10
3  3    6   14
4  7    8  NaN
```

https://s3.amazonaws.com/assets.datacamp.com/production/course_3639/slides/ch3_slides.pdf

Applying a function to the entire dataset

- Sometimes You need to apply a function on the level of the entire dataset (e.g., removing, adding, averaging)

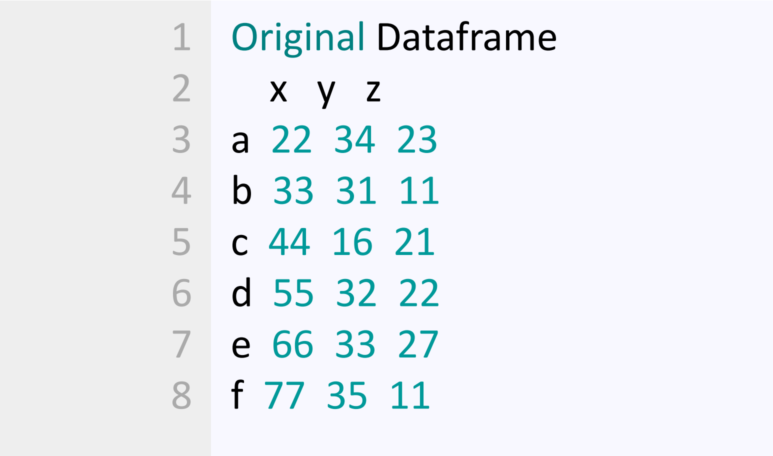
```
def cleaning_function(row_data):
        # Computation steps
        # Computation steps
df.apply(cleaning_function, axis=1)
```

http://nbviewer.jupyter.org/github/jvns/pandas-cookbook/blob/v0.2/cookbook/Chapter%207%20-%20Cleaning%20up%20messy%20data.ipynb

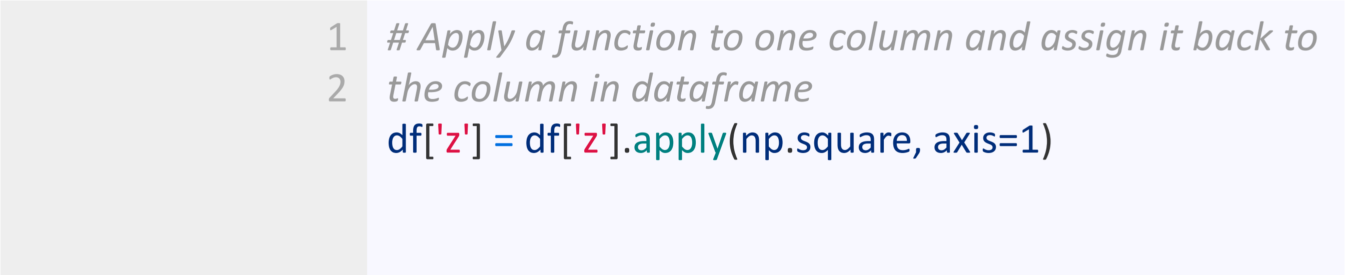
Applying a Function to Columns

- Sometimes You need to apply a function on the level of Columns

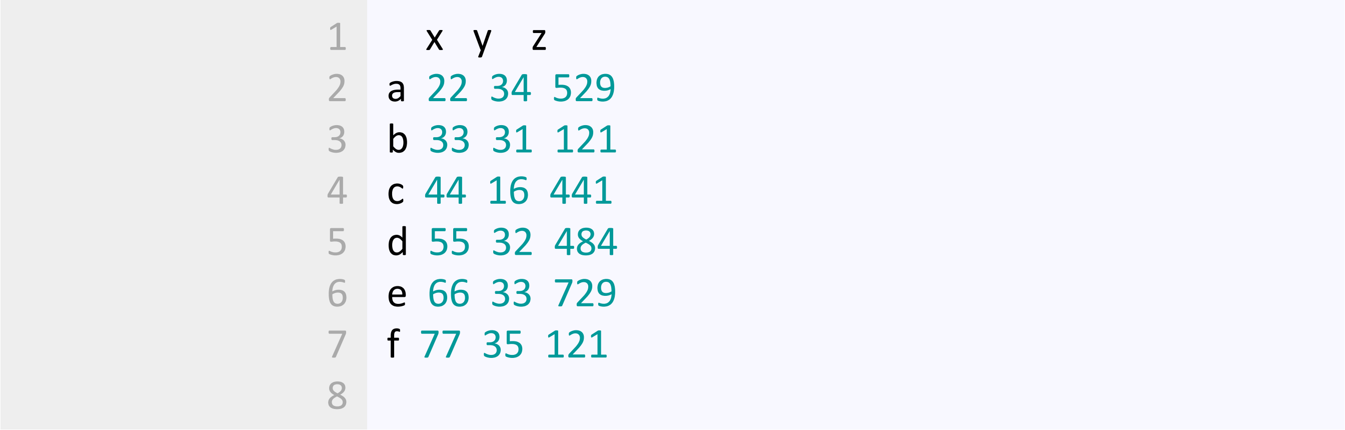
Example:



```
Original Dataframe
   x  y  z
a  22  34  23
b  33  31  11
c  44  16  21
d  55  32  22
e  66  33  27
f  77  35  11
```



```
# Apply a function to one column and assign it back to the column in dataframe
df['z'] = df['z'].apply(np.square, axis=1)
```



```
   x  y   z
a  22  34  529
b  33  31  121
c  44  16  441
d  55  32  484
e  66  33  729
f  77  35  121
```

https://thispointer.com/pandas-apply-a-function-to-single-or-selected-columns-or-rows-in-dataframe/

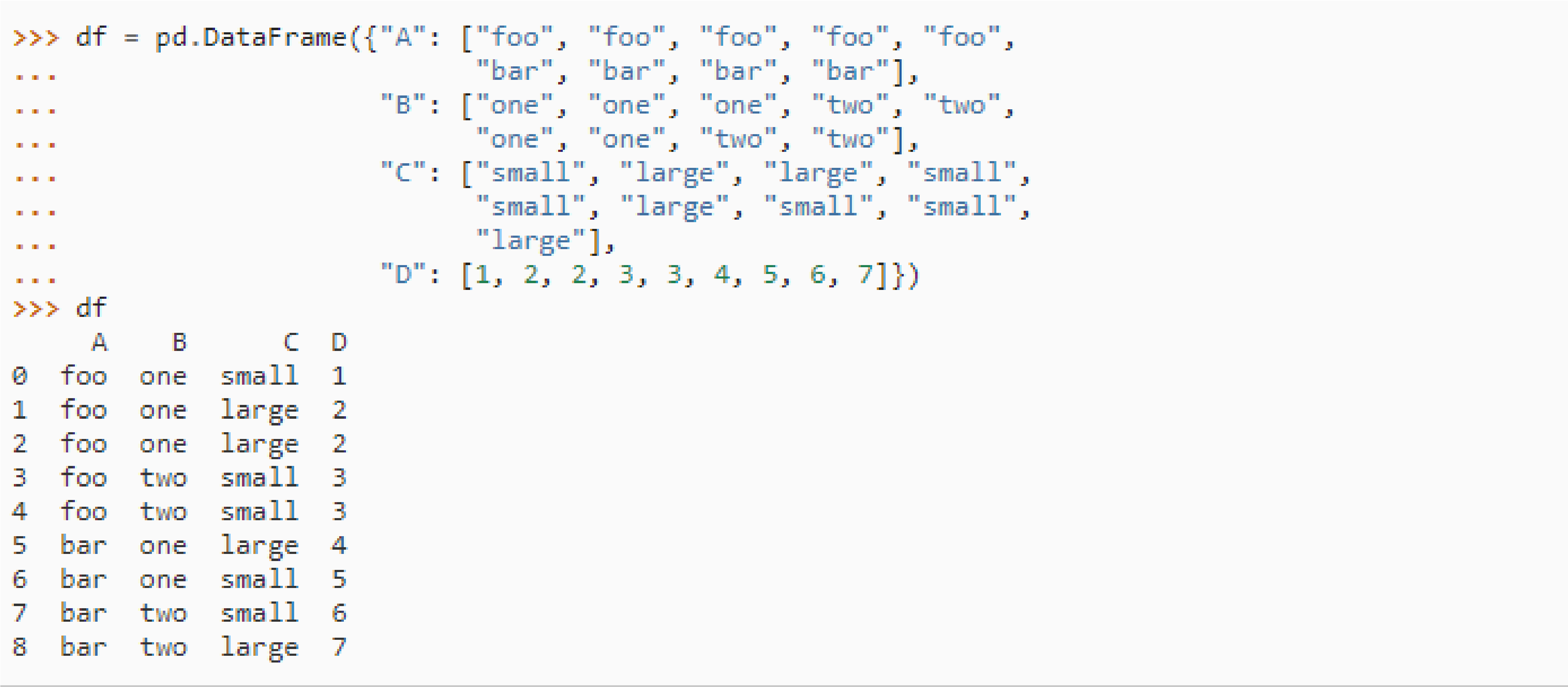
Pivot Tables

- Summary tables
- Introduce new columns from calculations
- Table can have multiple Indexes
- Excel is famous for it

*http://www.developintelligence.com/blog/2017/08/data-cleaning-pandas-python/

** https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html

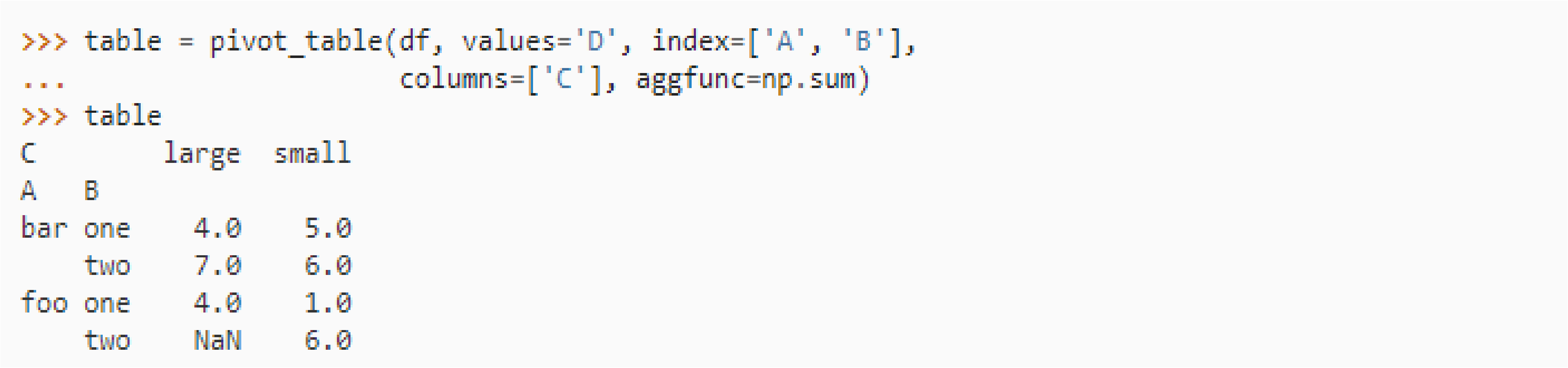
Pivot Table Example



```
>>> df = pd.DataFrame({"A": ["foo", "foo", "foo", "foo", "foo",
...                          "bar", "bar", "bar", "bar"],
...                    "B": ["one", "one", "one", "two", "two",
...                          "one", "one", "two", "two"],
...                    "C": ["small", "large", "large", "small",
...                          "small", "large", "small", "small",
...                          "large"],
...                    "D": [1, 2, 2, 3, 3, 4, 5, 6, 7]})
>>> df
     A    B      C  D
0  foo  one  small  1
1  foo  one  large  2
2  foo  one  large  2
3  foo  two  small  3
4  foo  two  small  3
5  bar  one  large  4
6  bar  one  small  5
7  bar  two  small  6
8  bar  two  large  7
```

http://pbpython.com/pandas-pivot-table-explained.html

Pivot Table Example



```
>>> table = pivot_table(df, values='D', index=['A', 'B'],
...                     columns=['C'], aggfunc=np.sum)
>>> table
C        large  small
A   B
bar one    4.0    5.0
    two    7.0    6.0
foo one    4.0    1.0
    two    NaN    6.0
```

Groupby

- Groupby splits the data into different groups depending on a variable of your choice.
- The output from a groupby and aggregation operation is it a Pandas Series or a Pandas Dataframes?
    - As a rule of thumb, if you calculate more than one column of results, your result will be a Dataframe. For a single column of results, the agg function, by default, will produce a Series.

Groupby Example

- If our dataset is tweets extracted from Twitter and we want to group all the tweets by the username and count the number of tweets each user has

```
Our_grouped_tweets= df.groupby(‘username’) [‘tweets’].count()
```

Indexing the Dataframe

- Sometimes it is helpful to use a uniquely valued identifying field of the data as its index
  - How to check uniqueness? (df[‘Unique_column'].is_unique)
  - How to set the index? (df = df.set_index(' Unique_column’))
  - Is it necessary to have unique vales in column? No, but it will affect the performance
- Pandas supports Multi-axes indexing:

.loc()	Label based
.iloc()	Integer based

Sorting Data

- Sometimes it is required to sort the data according to one or multiple columns.
- Pandas allow this using the function .sort_values()

Example:

```
df = pd.DataFrame({'col1' : ['A', 'A', 'B', np.nan, 'D', 'C'],'col2' : [2, 1, 9, 8, 7, 4], 'col3': [0, 1, 9, 4, 2, 3]})
```

```
df.sort_values(by=['col1'])
```

```
  col1 col2 col3
0  A    2    0
1  A    1    1
2  B    9    9
5  C    4    3
4  D    7    2
3  NaN  8    4
```

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sort_values.html

Questions?

Useful Read

- Python for Data Analysis, Wes McKinney
- https://online.hbs.edu/blog/post/data-wrangling
- https://www.altexsoft.com/blog/datascience/preparing-your-dataset-for-machine-learning-8-basic-techniques-that-make-your-data-better/
- https://pandas.pydata.org/pandas-docs/stable/tutorials.html
- https://www.dataquest.io/blog/machine-learning-preparing-data/
- https://thispointer.com/pandas-apply-a-function-to-single-or-selected-columns-or-rows-in-dataframe/
- https://datacarpentry.org/python-ecology-lesson/05-merging-data/index.html
- https://towardsdatascience.com/introduction-to-pandas-apply-applymap-and-map-5d3e044e93ff#:~:text=apply()%20is%20used%20to,a%20Series%20with%20another%20value.