CSS 330 Data wrangling and visualization

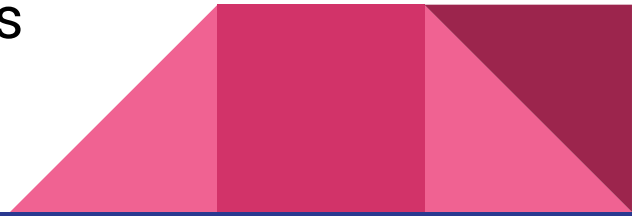
Lecture 6

Instructor: Nabigazinova Elnura

Data Wrangling: Clean, Transform and Merge

There are a number of scenarios when you are cleaning the data following are some of them:

- Inconsistent column names
- Missing data
- Outliers
- Duplicate rows
- Untidy
- Need to process columns
- The column type signals unexpected values



Basic steps when working with dataframes

Read the csv file and load into the dataframe **df : df=pd.read_csv(“”)**

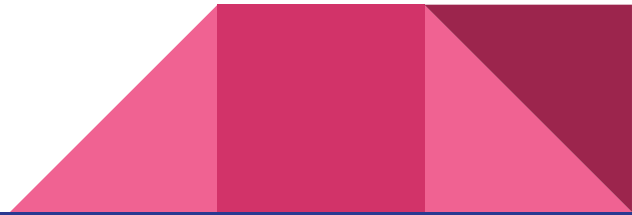
Visually inspecting first and last 5 data frames: **df.head(),df.tail()**

df.columns: to return indexes of columns.

df.shape: to return the number of rows and columns of data.

df.info(): to get additional information about the dataframe.

Frequency count for column values : **df.columnname.value_counts(dropna=False)**



Tidy data

Data problem we are fixing: Columns containing values instead of variable.

Solution: **pd.melt()**

In [1]:
```
import pandas as pd
df=pd.read_csv('Tidydata.csv')
df
```

Out[1]:

	name	treatment a	treatment b
0	Daniel	NaN	42
1	John	12.0	31
2	Jane	24.0	27

In [2]:
```
pd.melt(frame=df,id_vars='name',value_vars=['treatment a','treatment b'],var_name='treatment',value_name='result')
```

Out[2]:

	name	treatment	result
0	Daniel	treatment a	NaN
1	John	treatment a	12.0
2	Jane	treatment a	24.0
3	Daniel	treatment b	42.0
4	John	treatment b	31.0
5	Jane	treatment b	27.0

Pivoting data

Pivoting-turn unique values into separate columns.

In [3]:
```
import pandas as pd
df=pd.read_csv('Pivot.csv ')
df
```

Out[3]:

	date	element	value
0	1/10/2010	tmax	27.8
1	2/10/2010	tmax	14.5
2	1/10/2010	tmin	10.5
3	2/10/2010	tmin	6.5

In [4]:
```
df.pivot(index='date',columns='element',values='value')
```

Out[4]:

element	tmax	tmin
date		
1/10/2010	27.8	10.5
2/10/2010	14.5	6.5

When duplicate entries are there then while pivoting, we will get an error so we need to use pivot_table instead of it.

In [5]:
```
import pandas as pd
df=pd.read_csv('Pivot.csv ')
df
```

Out[5]:

	date	element	value
0	1/10/2010	tmax	27.8
1	2/10/2010	tmax	14.5
2	1/10/2010	tmin	10.5
3	2/10/2010	tmin	6.5
4	2/10/2010	tmin	4.5

In [6]:
```
df.pivot(index='date',columns='element',values='value')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-9b1981f187e1> in <module>
----> 1 df.pivot(index='date',columns='element',values='value')

~\Anaconda3\lib\site-packages\pandas\core\frame.py in pivot(self, index, columns, values)
   5626     def pivot(self, index=None, columns=None, values=None):
   5627         from pandas.core.reshape.pivot import pivot
-> 5628         return pivot(self, index=index, columns=columns, values=values)
   5629
```

In [8]:
```
import numpy as np
df.pivot_table(index='date',columns='element',values='value',aggfunc=np.mean)
```

Out[8]:

element	tmax	tmin
date		
1/10/2010	27.8	10.5
2/10/2010	14.5	5.5

Globbing

Problem: Can individually load if there are few datasets but what if there are thousands of datasets?

Solution: glob function to find files based on the pattern.

Globbing- pattern matching for file names

- Wildcards: *? Eg. Any csv file= *.csv, any single character=file_?.csv
- Returns list of filenames.
- We can use this list to load into a separate dataframe.

Import glob

```
csv_files=glob.glob('*.csv')

list_data=[]

for filename in csv_files:

data=pd.read_csv(filename)

list_data.append(data)
```

Regular expression to clean strings

String Manipulation:

Many built-in and external libraries.

‘re’ library for a regular expression.

Pattern matching:

Examples:

Number -> Regular Expression

17 -> \d*

$17 -> \$\d*

$17.00 -> \$\d*\.\d*

$17.89 -> \$\d*\.\d{2}

DATA WRANGLING WITH PANDAS

DEFINING A DATAFRAMES

	col1	col2	col3
1	a	b	c
2	d	e	f
3	g	h	i

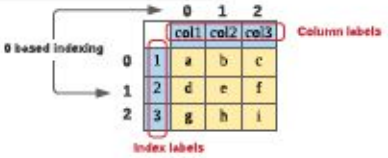
OPTION 1

```
DataFrame([[ 'a', 'b', 'c' ],
           [ 'd', 'e', 'f' ],
           [ 'g', 'h', 'i' ] ],
           index = [ 1, 2, 3],
           columns = [ 'col1', 'col2', 'col3'])
```

OPTION 2

```
DataFrame({ 'col1' :  [ 'a', 'd', 'g' ],
            'col2' :  [ 'b', 'e', 'h' ],
            'col3' :  [ 'c', 'f', 'i' ]},
            index = [ 1, 2, 3 ])
```

RETRIEVING DATA




```
Int64Index([1, 2, 3], dtype='int64')
```

df.columns df.index

```
Index(['col1', 'col2', 'col3'], dtype='object')
```

df.values

```
array([['a', 'b', 'c'],
       ['d', 'e', 'f'],
       ['g', 'h', 'i']], dtype = object)
```

df.loc[[0:2], [0:2]]

	col1	col2
1	a	b
2	d	e

df.loc[[1,2], ['col1', 'col2']]

COMBINING DATAFRAMES

pd.merge(df_1, df_2, how = '', on= '')

	col1	col2
1	A	2
2	K	9
3	Z	4

+

	col1	col3
1	A	3
2	K	8
3	X	1

= ?

	col1	col2	col3
1	A	2	3
2	K	9	8
3	Z	4	NaN

pd.merge(df_1, df_2, how = 'left', on= 'col1')

	col1	col2	col3
1	A	2	3
2	K	9	8
3	X	NaN	1

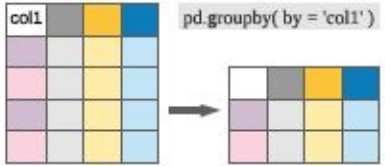
pd.merge(df_1, df_2, how = 'right', on= 'col1')

	col1	col2	col3
1	A	2	3
2	K	9	8

pd.merge(df_1, df_2, how = 'inner', on= 'col1')

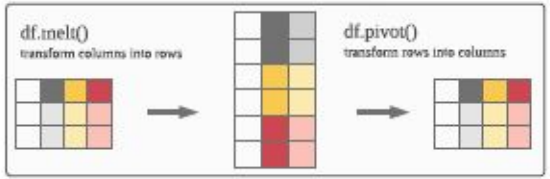
	col1	col2	col3
1	A	2	3
2	K	9	8
3	Z	4	NaN
4	X	NaN	1

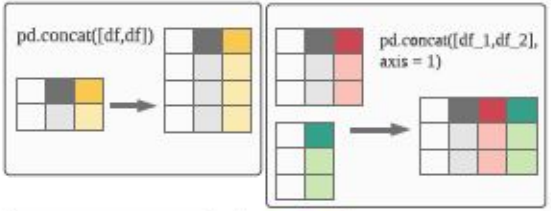
pd.merge(df_1, df_2, how = 'outer', on= 'col1')

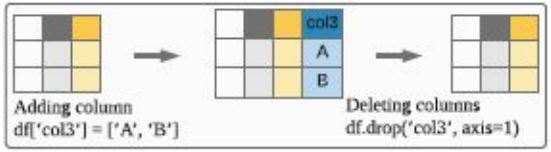


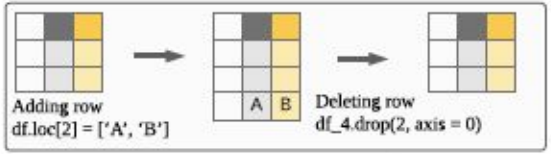
pd.groupby(by = 'col1')

RESHAPING DATAFRAMES

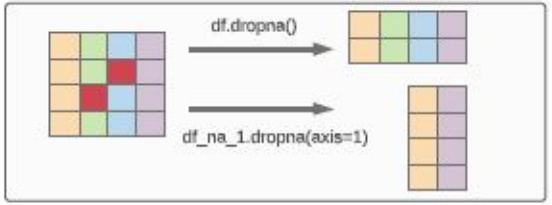








DEALING WITH NULL VALUES



Conclusion

Additional links:

1. https://medium.com/analytics-vidhya/cleaning-data-in-python-edfe6395ef77
2. https://realpython.com/python-data-cleaning-numpy-pandas/
3. https://www.nobledesktop.com/learn/python/data-wrangling-python-guide
4. https://www.tutorialspoint.com/python_data_science/python_data_wrangling.htm
5. https://medium.com/swlh/data-wrangling-in-python-basics-8bd7d81a8710