

Data Wrangling

with pandas

Cheat Sheet

<http://pandas.pydata.org>

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(  
    {"a" : [4 ,5, 6],  
     "b" : [7, 8, 9],  
     "c" : [10, 11, 12]},  
    index = [1, 2, 3])  
Specify values for each column.
```

```
df = pd.DataFrame(  
    [[4, 7, 10],  
     [5, 8, 11],  
     [6, 9, 12]],  
    index=[1, 2, 3],  
    columns=['a', 'b', 'c'])  
Specify values for each row.
```

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

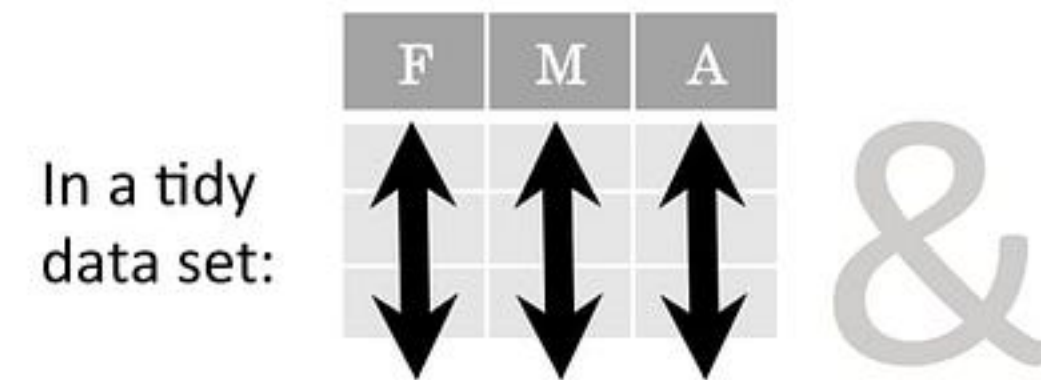
```
df = pd.DataFrame(  
    {"a" : [4 ,5, 6],  
     "b" : [7, 8, 9],  
     "c" : [10, 11, 12]},  
    index = pd.MultiIndex.from_tuples(  
        [('d',1),('d',2),('e',2)],  
        names=['n','v']))  
Create DataFrame with a MultiIndex
```

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

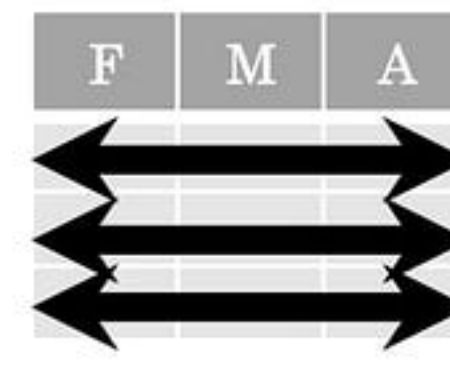
```
df = (pd.melt(df)  
     .rename(columns={  
         'variable' : 'var',  
         'value' : 'val'})  
     .query('val >= 200'))
```

Tidy Data – A foundation for wrangling in pandas



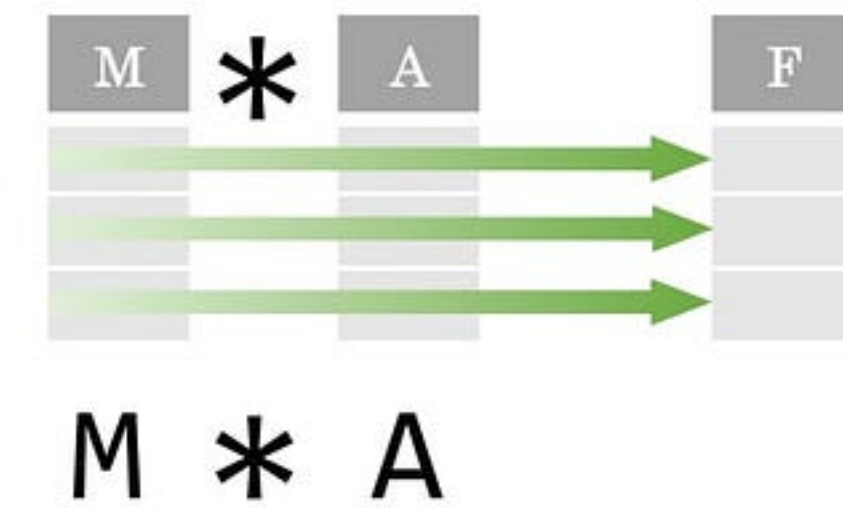
In a tidy data set:

Each **variable** is saved in its own **column**



Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Reshaping Data – Change the layout of a data set



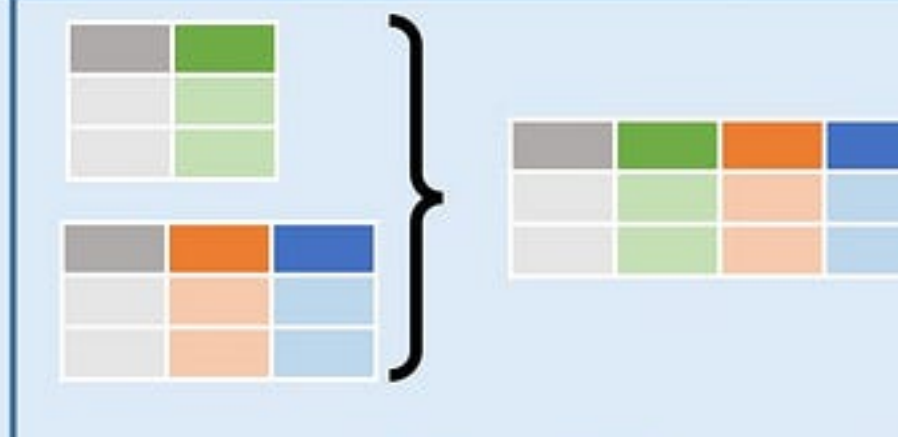
`pd.melt(df)`
Gather columns into rows.



`df.pivot(columns='var', values='val')`
Spread rows into columns.



`pd.concat([df1,df2])`
Append rows of DataFrames



`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`

Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`

Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`

Rename the columns of a DataFrame

`df.sort_index()`

Sort the index of a DataFrame

`df.reset_index()`

Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(['Length','Height'], axis=1)`

Drop columns from DataFrame

Subset Observations (Rows)



`df[df.Length > 7]`

Extract rows that meet logical criteria.

`df.drop_duplicates()`

Remove duplicate rows (only considers columns).

`df.head(n)`

Select first n rows.

`df.tail(n)`

Select last n rows.

`df.sample(frac=0.5)`

Randomly select fraction of rows.

`df.sample(n=10)`

Randomly select n rows.

`df.iloc[10:20]`

Select rows by position.

`df.nlargest(n, 'value')`

Select and order top n entries.

`df.nsmallest(n, 'value')`

Select and order bottom n entries.

Subset Variables (Columns)



`df[['width','length','species']]`

Select multiple columns with specific names.

`df['width']` or `df.width`

Select single column with specific name.

`df.filter(regex='regex')`

Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length\$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]\$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species\$).*\$'	Matches strings except the string 'Species'

`df.loc[:, 'x2':'x4']`

Select all columns between x2 and x4 (inclusive).

`df.iloc[:, [1,2,5]]`

Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`

Select rows meeting logical condition, and only the specific columns.

Logic in Python (and pandas)			
<	Less than	<code>!=</code>	Not equal to
>	Greater than	<code>df.column.isin(values)</code>	Group membership
==	Equals	<code>pd.isnull(obj)</code>	Is NaN
<=	Less than or equals	<code>pd.notnull(obj)</code>	Is not NaN
>=	Greater than or equals	<code>&, , ~, ^, df.any(), df.all()</code>	Logical and, or, not, xor, any, all