

Data Wrangling

with pandas Cheat Sheet

<http://pandas.pydata.org>

[Pandas API Reference](#) [Pandas User Guide](#)

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:



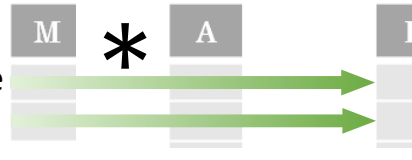
Each **variable** is saved in its own **column**

&



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.



M * A = F

Reshaping Data – Change layout, sorting, reindexing, renaming



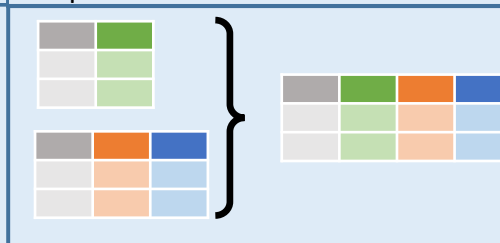
`pd.melt(df)`
Gather columns into rows.



`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(columns=['Length', 'Height'])`
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.sample(frac=0.5)  
Randomly select fraction of rows.  
df.sample(n=10)  
Randomly select n rows.  
df.nlargest(n, 'value')  
Select and order top n entries.  
df.nsmallest(n, 'value')  
Select and order bottom n entries.  
df.head(n)  
Select first n rows.  
df.tail(n)  
Select last n rows.
```

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.
```

Using query

`query()` allows Boolean expressions for filtering rows.

```
df.query('Length > 7')  
df.query('Length > 7 and Width < 8')  
df.query('Name.str.startswith("abc")', engine="python")
```

Subsets - rows and columns

Use `df.loc[]` and `df.iloc[]` to select only rows, only columns or both.
Use `df.at[]` and `df.iat[]` to access a single value by row and column.
First index selects rows, second index columns.

```
df.iloc[10:20]  
Select rows 10-20.  
df.iloc[:, [1, 2, 5]]  
Select columns in positions 1, 2 and 5 (first column is 0).  
df.loc[:, 'x2':'x4']  
Select all columns between x2 and x4 (inclusive).  
df.loc[df['a'] > 10, ['a', 'c']]  
Select rows meeting logical condition, and only the specific columns.  
df.iat[1, 2]  
Access single value by index  
df.at[4, 'A']  
Access single value by label
```

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

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