

# 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

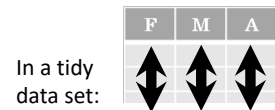
```
df = pd.DataFrame(  
    {"a": [4, 5, 6],  
     "b": [7, 8, 9],  
     "c": [10, 11, 12]},  
    index = pd.MultiIndex.from_tuples(  
        [('d', 1), ('d', 2),  
         ('e', 2)], names=['n', 'v']))  
Create DataFrame with a MultiIndex
```

## Method Chaining

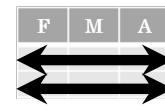
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

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

## Tidy Data – A foundation for wrangling in pandas

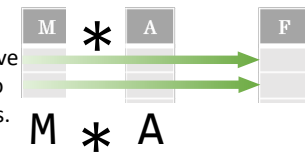


Each **variable** is saved  
in its own **column**



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 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>&amp;,  , ~, ^, 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'

## Summarize Data

**df['w'].value\_counts()**  
Count number of rows with each unique value of variable

**len(df)**  
# of rows in DataFrame.

**df.shape**  
Tuple of # of rows, # of columns in DataFrame.

**df['w'].nunique()**  
# of distinct values in a column.

**df.describe()**  
Basic descriptive and statistics for each column (or GroupBy).



pandas provides a large set of [summary functions](#) that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

<b>sum()</b> Sum values of each object.	<b>min()</b> Minimum value in each object.
<b>count()</b> Count non-NA/null values of each object.	<b>max()</b> Maximum value in each object.
<b>median()</b> Median value of each object.	<b>mean()</b> Mean value of each object.
<b>quantile([0.25, 0.75])</b> Quantiles of each object.	<b>var()</b> Variance of each object.
<b>apply(function)</b> Apply function to each object.	<b>std()</b> Standard deviation of each object.

## Group Data



**df.groupby(by="col")**  
Return a GroupBy object, grouped by values in column named "col".

**df.groupby(level="ind")**  
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

<b>size()</b> Size of each group.	<b>agg(function)</b> Aggregate group using function.
--------------------------------------	---

## Windows

**df.expanding()**  
Return an Expanding object allowing summary functions to be applied cumulatively.

**df.rolling(n)**  
Return a Rolling object allowing summary functions to be applied to windows of length n.

## Handling Missing Data

**df.dropna()**  
Drop rows with any column having NA/null data.

**df.fillna(value)**  
Replace all NA/null data with value.

## Make New Columns



**df.assign(Area=lambda df: df.Length\*df.Height)**  
Compute and append one or more new columns.

**df['Volume'] = df.Length\*df.Height\*df.Depth**  
Add single column.

**pd.qcut(df.col, n, labels=False)**  
Bin column into n buckets.



pandas provides a large set of [vector functions](#) that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

<b>max(axis=1)</b> Element-wise max.	<b>min(axis=1)</b> Element-wise min.
<b>clip(lower=-10, upper=10)</b> Trim values at input thresholds	<b>abs()</b> Absolute value.

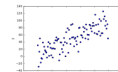
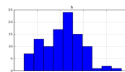
The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

<b>shift(1)</b> Copy with values shifted by 1.	<b>shift(-1)</b> Copy with values lagged by 1.
<b>rank(method='dense')</b> Ranks with no gaps.	<b>cumsum()</b> Cumulative sum.
<b>rank(method='min')</b> Ranks. Ties get min rank.	<b>cummax()</b> Cumulative max.
<b>rank(pct=True)</b> Ranks rescaled to interval [0, 1].	<b>cummin()</b> Cumulative min.
<b>rank(method='first')</b> Ranks. Ties go to first value.	<b>cumprod()</b> Cumulative product.

## Plotting

**df.plot.hist()**  
Histogram for each column

**df.plot.scatter(x='w', y='h')**  
Scatter chart using pairs of points



## Combine Data Sets

adf		bdf		
x1	x2	x1	x3	
A	1	A	T	+
B	2	B	F	
C	3	D	T	

### Standard Joins

**pd.merge(adf, bdf, how='left', on='x1')**  
Join matching rows from bdf to adf.

**pd.merge(adf, bdf, how='right', on='x1')**  
Join matching rows from adf to bdf.

**pd.merge(adf, bdf, how='inner', on='x1')**  
Join data. Retain only rows in both sets.

**pd.merge(adf, bdf, how='outer', on='x1')**  
Join data. Retain all values, all rows.

### Filtering Joins

**adf[adf.x1.isin(bdf.x1)]**  
All rows in adf that have a match in bdf.

**adf[~adf.x1.isin(bdf.x1)]**  
All rows in adf that do not have a match in bdf.

ydf		zdf		
x1	x2	x1	x2	
A	1	B	2	+
B	2	C	3	
C	3	D	4	

### Set-like Operations

**pd.merge(ydf, zdf)**  
Rows that appear in both ydf and zdf (Intersection).

**pd.merge(ydf, zdf, how='outer')**  
Rows that appear in either or both ydf and zdf (Union).

**pd.merge(ydf, zdf, how='outer', indicator=True)**  
**.query('\_merge == "left\_only"')**  
**.drop(columns=['\_merge'])**  
Rows that appear in ydf but not zdf (Setdiff).