General

Pandas API Reference — Seaborn API Reference
Pandas User Guide — Matplotlib API Reference

Creating DataFrames

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

use IO-Tools to import from files

```
df = pd.read_csv("filepath")
```

```
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.

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

Display & Visualize data

Display options for DataFrames:

```
pd.set_option('display.max_rows', 4)
pd.reset_option('display.max_rows')
with pd.option_context('display.max_rows', 4):
```

only sets options within the „with“ codeblock

Style options for DataFrames (Cell highlighting, heatmapping ..)

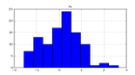
Visualize Data in plots

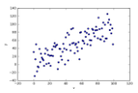
```
df.plot.hist()
```

Histogram for each column

```
df.plot.scatter(x='w',y='h')
```

Scatter chart using pairs of points



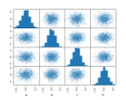


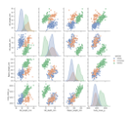
```
df.plotting.scatter_matrix()
```

Matrix of scatter plots and histograms (for regression)

```
seaborn.pairplot(df,
  hue='column_name')
```

Matrix of pairwise relationship (for classification)





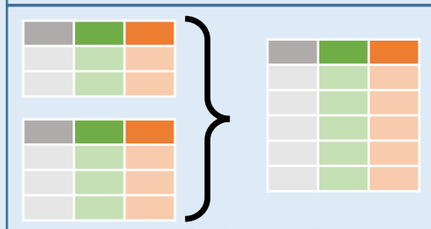
Reshaping Data – Change layout, sorting, reindexing, renaming

```
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 – Selecting Data: rows or columns

```
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[row_ind, column_ind]
df.iloc[10:20]
```

Select rows by position.

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

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

```
df.loc[row_ind, column_ind]
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[['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*.

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

Select and order top n entries.

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

Select and order bottom n entries.

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

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'

Cheatsheet for pandas (http://pandas.pydata.org/) inspired by Rstudio Data Wrangling Cheatsheet written by Irv Lustig, Princeton Consultants

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

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

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

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

`df.shape`
Length and width of dataset

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:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data

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

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

Possibly use df.reset_index() after!

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

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

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.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

Windows

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

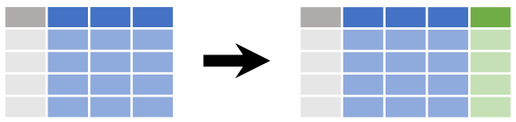
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

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

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

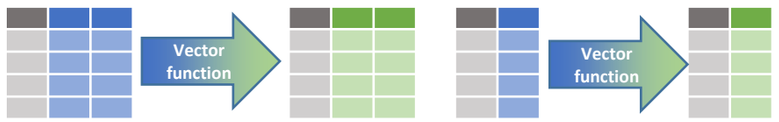
Make New Columns



`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:

`max(axis=1)`
Element-wise max.

`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

`abs()`
Absolute value.

Apply Functions

`df.pipe()` when chaining functions; tablewise fct. application

`df.apply()` row (axis=1) or column(axis=0) wise fct. application

`df.agg()`and `df.transform()` use multiple aggregating operations

`df.assign()` assigns new columns to a df

`df.applymap()` elementwise function application

Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

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

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

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

x1	x2	x3
A	1	T
B	2	F

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

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

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

Filtering Joins

x1	x2
A	1
B	2

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

x1	x2
C	3

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

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

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

x1	x2
A	1
B	2
C	3
D	4

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

x1	x2
A	1

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

Cheatsheet for pandas (http://pandas.pydata.org/) inspired by Rstudio Data Wrangling Cheatsheet written by Irv Lustig, Princeton Consultants