

TO START

```
import numpy as np
import pandas as pd
```

SERIES (similar to numpy array)

<code>pd.Series(data = list)</code>	create series from list
<code>pd.Series(data=list, index=labels)</code>	create series with index
<code>pd.Series(np_arr)</code>	create series from numpy array
<code>pd.Series(np_arr, labels)</code>	create series with index
<code>pd.Series(dict)</code>	create series from dictionary
<code>pd.Series[num]</code>	indexing
<code>ser1 + ser2</code>	sum two Series

Pandas series differs from numpy arrays because series **can have axis labels**, instead of just a number location. It also **doesn't need to hold numeric data**, it can hold any arbitrary Python Object, also functions (although unlikely used).

Note: the terms "data=" and "index=" can be omitted.

DATAFRAMES and INDEXING

<code>df = pd.DataFrame()</code>	* create dataframe
<code>df['col']</code>	* select col
<code>df.loc['row']</code>	select row
<code>df.iloc['row']</code>	select a row by its index
<code>df.col</code>	select a column (opt.2 - avoid)

DATAFRAMES and INDEXING (cont)

<code>df[['col1','col2']]</code>	*	take two columns
<code>type(df['col'])</code>		column type
<code>df['new_col'] = [1,2,3]</code>		insert column
<code>df.drop('row',axis=0)*</code>		drop row
<code>df.drop('col',axis=1)</code>		drop column
<code>df.drop('col',axis=1, inplace=True)*</code>		permanent drop
<code>df.loc['row1','col1']</code>		select a row and a column
<code>df.loc[['r1', 'r2'],['c1','c2']]*</code>		select 2 rows and 2 columns
<code>df>condition</code>		return boolean
<code>df[df>cond]</code>		return values
<code>df[df['col']>0]</code>		return rows of col that satisfy condition
<code>df[df['col1']>0]['col2']</code>		return col2 that satisfy cond. on col1
<code>df[df['c1']>0][['c2','c3']]</code>		return c2 & c3 that satisfy cond. on col1
<code>df[(cond1) & (cond2)]</code>		return values that satisfy cond1 & cond2
<code>df[(cond1) (cond2)]</code>		return values that satisfy cond1 cond2
<code>df.reset_index()</code>		add num index

DATAFRAMES and INDEXING (cont)

<code>df['new_col'] = 'NY LA'</code>	add col quickly
<code>.split()</code>	
<code>df.set_index('col')</code>	set a column as index
<code>df.set_index('col', inplace=True)</code>	make it permanent

!!THERE IS ALSO MULTI-INDEXING"

DataFrame function take a **data** (the values), **index** (the name of the index column), **columns** (the name of the column) parameters.

Columns are series.

take two columns: note the double brackets `[[ ]]`

axis=0 can be omitted, is the default value.

inplace=True will apply the result to the original dataframe. Without it, you are not changing the dataframe.

r = row.

c = column.

MISSING DATA

<code>df.isnull()</code>	check for na
<code>df.dropna()</code>	drop all rows with at least 1 na
<code>df.dropna(axis=1)</code>	drop all cols with at least 1 na
<code>df.dropna(thresh=n)</code>	keep with at least n value/s
<code>df.fillna(value='value')</code>	replace na
<code>df['col'].fillna(value=)</code>	replace using funct
<code>df['col'].mean()</code>	



By DarioPittera (aggialavura)

Not published yet.

Last updated 24th June, 2019.

Page 1 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

GROUPBY

df.groupby('col')	group rows by a col
grouped_df.count()	use cnt function
grouped_df.mean()	use mean function
grouped_df.std()	use std function
grouped_df.min()	use min function
grouped_df.max()	use max function
grouped_df.describe()	df descriptives
grouped_df('col').count().loc['row']	apply function and take a row
... .transpose()	rotate results
... .transpose()['row']	rotate and take a row

MERGING, JOINING, CONCATENATING

pd.concat([df1, df2, df3])	concatenate dfs
pd.concat([...], axis=1)	concatenate by col
pd.merge()*	merge two dfs
df1.join(df2)	join two dfs

pd.merge() takes "df1", "df2", "how=", "on=" parameters. "how=" can be "inner"/"outer"/"left"/"right", "on=" has to be a column/s key.

join() is similar to merge but works on indexes that can be different. It also cn take the "how=" argument.

METHODS and FUNCTIONS

df['c'].unique()	return unique values
df['c'].nunique()	count unique val
df['c'].value_counts()	count how many of same values

METHODS and FUNCTIONS (cont)

df['c1'].apply(func)	apply func to df
df['c1'].apply(len)	apply len
df['c1'].apply(sum)	apply sum
... .apply(lambda x:x+2)	apply lambda
df.index	return idx names
df.info()	return df info
df.describe()	return df stats
df.columns	return col names
del df['col']*	delete col from df
df.sort_values(by='col')	sort df values
df.pivot_table()*	create a pivot tbl

del differs from ".drop()" because it will permanently remove a column from the df.

pivot_table() takes "values=", "index=", "columns=" parameters. It reads: "Create a table from df, with values of colx, index of colx2, and divided by values in colx3"

INPUT and OUPUT code to start

```
# to import HTML tables
conda install lxml
conda install html5lib
conda install Beautiful Soup4
# to use SQL
from sqlalchemy import create_engine
engine = create_engine('sqlite:///memory:')
df.to_sql('data', engine)
sql_df = pd.read_sql('data', engine)
```

INPUT and OUPUT operations

pwd	ask nb route
df = pd.read_csv('example')	read csv
df = pd.read_excel('name', sheet_name='name')	read excel
df = pd.read_html('address')	read html
df.to_csv('str', index=False)	save as csv
df.to_excel('name', sheet_name='name', index=False)	save as xlsx



By DarioPittera (aggialavura)

cheatography.com/aggialavura/
www.dariopittera.com

Not published yet.

Last updated 24th June, 2019.

Page 2 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>