Python - Pandas Cheat Sheet

by DarioPittera (aggialavura) via cheatography.com/83764/cs/19829/

TO START

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

SERIES (similar to numpy array)

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

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

DATAFRAMES and INDEXING (cont)

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

DATAFRAMES and INDEXING (cont)

df['new_col'] = 'NY LA' .split()	add col quickly
df.set_index('col')	set a column as index
df.set_index('col', inplace=True)	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

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

By **DarioPittera** (aggialavura)

cheatography.com/aggialavura/
www.dariopittera.com

Not published yet.
Last updated 24th June, 2019.

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 Beauti ful Soup4
# to use SQL
from sqlalchemy import create -
_engine
engine = create _en gin e(' -
sql ite :// /:m emo ry:')
df.to_ sql ('d ata', engine)
sql_df = pd.rea d_s ql( 'da -
ta' ,co n=e ngine)
```

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