Pandas Basics – Data Science For Hackers

Name	Description	Example
IMPORTING LIBRARIES		
import	Imports various python libraries	`Import Pandas`
From <*library*> **import ***	Imports all functions from library	`From sklearn import *`
Import <*library*> ***as*** <*reference_name*>	Imports library with a new name for easier coding NOTE: Pandas is usually imported as pd, Numpy as np and Scipy as sp.	`Import pandas as pd`
from <*library*> import <*function1, function2*>	Imports specific functions. Good for preserving memory with large libraries	`From numpy import array, random`
READING DATA		
pandas.read_csv	Reads csv files into pandas DataFrame	`pd.read_csv('file.csv')`
pandas.read_excel	Reads excel files into pandas DataFrame	`pd.read_excel('file.xls', sheetname='sheet1')`
pandas.read_json	Reads JSON files or URL's into pandas DataFrame	`pd.read_json('file://localhost/path/to/table.json')`
pandas.read_html	Reads HTML files or URL's into pandas DataFrame	`pd.read_html('http://pandas.pydata.org/getpandas.html')`
pandas.read_sql	Reads from SQL database into pandas DataFrame	`pd.read_sql('SELECT * FROM Data')`
CREATE SERIES AND DATA FRAMES		

pandas.Series(<data>, index=<index>)	Creates a pandas Series object.	`series1 = pd.Series( ['a','b','c','d', 'e'])`
		`series2 = pd.DataFrame([series2,series1]))`

pandas.DataFrame(<data>, <index>, <column_names>)	Creates a pandas data frame object. B	`df = pd.DataFrame([series1, series2], columns=['one','two','three','four'])`
REFERENCING AND SLICING DATAFRAMES AND SERIES		
<*Series*>[**n**]	Slices the Series and returns object at	`Series1[2] → 'c'`

	index n. Also works for string indexing (Note: Python is 0 indexed)	

<Series>[n:z]	Slices the Series and returns objects at index n through z	`Series[0:2] → 'a'`
		`'b',`
		`'c'`

<Series>[-n:]	Slices the Series and returns the last n object in the Series	`Series[-1:] → 'e'`

<Series>[n:z:a]	Slices the Series and returns very a object between n and z	`Series1[1:6:2] → 'b'`
		`'d'`

<Series>.head(n)	Returns the top n items in the Series. Default 5.	`Series1.head(2) → 'a'`
		`'b'`

<Series>.tail(n)	Returns the bottom n items in the Series. Default 5.	`Series1.tail(1) → 'd'`
		`'e'`

<DataFrame>[:n]	Returns the first n rows in a DataFrame	`df[1] →` **Zero** | **One** | **Two** | **Three** | **Four** Apple | Blueberry | Cherry | Dog | Cat
<DataFrame>['*<column>*']	Returns the column of the DataFrame	`df['three'] → dog`

<DataFrame>.loc[*n*]	Returns row at index n. Also works with strings	`df.loc[1] →`
		`0 apple`
		`1 blueberry`
		`2 cherry`
		`3 dog`
		`4 cat`

<DataFrame>.sample[n]	Returns a pseudo random sample of n rows	`Data.sample(10)`
SORTING		
<Series>.sort()	Sorts Series by specified value.	
<DataFrame>.sort()	Sorts DataFrame by specified value.	
<Series>.sort_index()	Sorts Series by index value.	

<DataFrame>.**sort_index()**	Sorts DataFrame by index value.	
DEALING WITH NULLS		
***<Series>*.dropna()**	Drops nulls from Series	
***<DataFrame>*.dropna()**	Drops rows with ANY nulls from DataFrame	
***<DataFrame>*.dropna(how='all')**	Drops rows with ALL nulls from DataFrame	
***<Series>*.fillna(value=<x>)**	Fills nulls with specified values	
***<DataFrame>*.fillna(value=<x>)**	Fills nulls with specified values	
OTHER MANIPULATIONS		
***<DataFrame>*.drop('<column>', axis = 1)**	Drops specified column from DataFrame	
<DataFrame>.**T**	Transposes DataFrame	
<DataFrame>. **sum(axis = 0)**	Returns the sum of each columns	
***<DataFrame>*.sum(axis = 1)**	Returns the sum of each row	
***<DataFrame>*['*<column>*'] = x**	Returns a new column of value x in DataFrame	
*<DataFrame>**.apply(<function>)***	Applies a function to the DataFrame	
FILTERING DATA		

<Series>[<Series><conditions>]	Returns a boolean if elements of the series match the condition	`Series3 = pd.Series([1,2,3,4,5])`
		`Series3 = Series3[Series3>3] →  4`
		`5`

<DataFrame>[['Column1','Column2']]	Returns new dataframe only with specified columns	`df2[['one','two']]  →` **One** / **Two** b / c blueberry / cherry
<DataFrame>[<Dataframe>['<Column Name>] <condition>]	Returns new dataframe containing only cases where condition is met.	`df[df['two']=='cherry'] →` **Zero** / **One** / **Two** / **Three** / **Four** Apple / Blueber / Cherry / Dog / Cat

		ry
MERGING DATASETS		
pandas.concat()	Creates the union of the two pandas objects	`pd.concat([data, data2])` `pd.concat([series1,series2])`
pandas.merge()	Merges the two datasets together	`pd.merge(data,data2)` `data.merge(data2)`
<*Series*>**.append()**	Appends object to Series	`series1.append(series2)`
<*DataFrame*>**.append()**	Appends object to DataFrame	`data.append(data2)`
DATA EXPLORATION		
<*Series*>.**abs()**	Returns absolute value of series elements	
<*Series*>.count()	Counts total elements in series	
<*Series*>.max()	Returns series maximum	
<*Series*>.min()	Returns series minimum	
<*Series*>.mean()	Returns series mean	
<*Series*>.median()	Returns series median	
<*Series*>.mode()	Returns series mode	
<*Series*>.quantile([q]**)**	Returns the 'q' quantile of a series	
<*Series*>.sum()	Returns the sum of the series	
<*Series*>.std()	Returns the standard deviation of the series	
<*Series*>.var()	Returns the total variance of the series	
<*Series*>.describe()	Returns series sum count, mean, standard deviation, min, quartiles, max for quantitative data. Returns series count, unique elements, most common element, and the frequency of the most common element for qualitative data.	
<*Series*>.unique()	Returns unique values in the series	
<*Series*>.value_counts()	Returns the count of the unique values in the series	

***<Series1>*.corr(*<Series2>*)**	Returns correlation coefficient of the two series	
IP ADDRESS LIBRARY		
import ipaddress	Import ip address library	
ipaddress.ip_address*(<IP Address as string>)*	converts string object to the ipaddress object (ipv4 or ipv6)	`X =ipaddress.ip_address('192.168.0.1')`
ipaddress.ip_network('*<IP Network as string>*')	converts string object to ip network object.	`Y = ipaddress.ip_network('192.168.0.1/28')`
*<ip_address object>***.is_private**	returns boolean for private or public addresses.	`x.is_private  → TRUE`
USER AGENT LIBRARY		
from user_agents import parse	Imports the parse function from the user agent library	
parse('*<UserAgentString>*')	parses out useragent string	`ua = parse('Mozilla/5.0 (BlackBerry; U; BlackBerry 9700; pt) AppleWebKit/534.8+ (KHTML, like Gecko) Version/6.0.0.546 Mobile Safari/534.8+')`
<user_agent object>.**browser**	returns user agent browser	`ua.browser  → Browser(family='BlackBerry WebKit', version=(6,), version_string='6')`
<user_agent object>.**browser.family**	returns browser family	`ua.browser.family →  'BlackBerry WebKit'`
<user_agent object>.**browser.version**	returns version number as integer	`ua.browser.version →  (6,)`
<user_agent object>.**version_string**	returns version number as string	`ua.browser.version_string → '6'`
<user_agent object>.**os**	returns operating system	`ua.os → OperatingSystem(family='BlackBerry OS', version=(6,), version_string='6')`
<user_agent object>.**os.family**	returns OS family	`ua.os.family → 'BlackBerry OS'`

***<user_agent object>*.os.version**	returns version number as list of integers	`ua.os.version →  (6,)`
***<user_agent object>*.os.version_string**	returns version number as string.	`ua.os.version_string → '6'`
***<user_agent object>*.is_pc**	True if user agent is PC	`ua.is_pc → False`
***<user_agent object>*.is_mobile**	True if user agent is mobile	`ua.is_mobile → True`
***<user_agent object>*.is_tablet**	True if user agent is tablet	`ua.is_tablet → False`
***<user_agent object>*.is_touch_capable**	True if user agent is touch capable.	`ua.is_touch_capable → False`
***<user_agent object>*.is_bot**	True if user agent is bot.	`ua.is_bot → False`