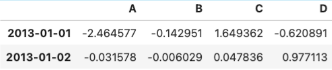
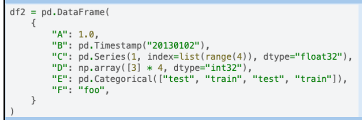
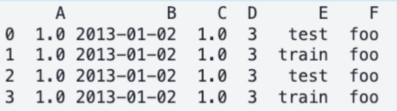
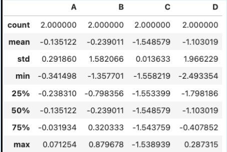
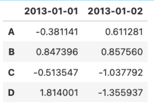
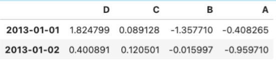
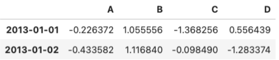
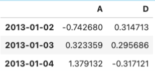
Pandas and Numpy Quick Reference Sheet
Version 1.0.3 by Joff Thyer
Rivergum Security LLC

Concept	Description	Additional Information
import pandas as pd	conventional way to import into Python	Pandas and numpy tend to go together in pairs
import numpy as np	conventional way to import into Python	
Pandas "Series" object	a one dimensional labeled array	
Pandas "DataFrame" object	a two dimensional data structure that holds a table	
s = pd.Series([1,2,np.nan,4])	creating a Pandas series	
dates = pd.date_range("20130101", periods=6)	create a Pandas date range with 6 periods (defaults to days)	
df = pd.DataFrame(np.random.randn(2, 4), index=dates, columns=list("ABCD"))	generate a dataframe with index of date range, and 4 columns.	 A B C D 2013-01-01 -2.464577 -0.142951 1.649362 -0.620891 2013-01-02 -0.031578 -0.006029 0.047836 0.977113
np.random.randn(6, 4)	creates a 6 x 4 random floating point array from standard normal distribution	


df2 = pd.DataFrame(
    {
        "A": 1.0,
        "B": pd.Timestamp("20130102"),
        "C": pd.Series(1, index=list(range(4)), dtype="float32"),
        "D": np.array([3] * 4, dtype="int32"),
        "E": pd.Categorical(["test", "train", "test", "train"]),
        "F": "foo",
    }
)
	generates a dataframe from a dictionary whereby the column names are the keys in the dictionary	
		A B C D E F
		0 1.0 2013-01-02 1.0 3 test foo
		1 1.0 2013-01-02 1.0 3 train foo
		2 1.0 2013-01-02 1.0 3 test foo
		3 1.0 2013-01-02 1.0 3 train foo

DataFrame.head()	view the first few rows of a dataframe	
DataFrame.tail()	view the last few rows of a dataframe	
DataFrame.to_numpy()	return the numpy array of a dataframe without index of column names	

df.describe()
DataFrame.describe()
	yield basic statistical information about a dataframe.	
		A B C D
		count 2.000000 2.000000 2.000000 2.000000
		mean -0.135122 -0.239011 -1.548579 -1.103019
		std 0.291860 1.582066 0.013633 1.966229
		min -0.341498 -1.357701 -1.558219 -2.493354
		25% -0.238310 -0.798356 -1.553399 -1.798186
		50% -0.135122 -0.239011 -1.548579 -1.103019
		75% -0.031934 0.320333 -1.543759 -0.407852
		max 0.071254 0.879678 -1.538939 0.287315

df.T	transpose the data (matrix)	 2013-01-01 2013-01-02 A -0.381141 0.611281 B 0.847396 0.857560 C -0.513547 -1.037792 D 1.814001 -1.355937
df.sort_index(axis=1, ascending=False)	sort by axis 1 (rows). Axis=0 would be sort by columns.	 D C B A 2013-01-01 1.824799 0.089128 -1.357710 -0.408265 2013-01-02 0.400891 0.120501 -0.015997 -0.959710
df.sort_values(by="B")	sort by values in column "B"	
df[0:2]	select two rows by slicing	 A B C D 2013-01-01 -0.226372 1.055556 -1.368256 0.556439 2013-01-02 -0.433582 1.116840 -0.098490 -1.283374
df["2013-01-01":"2013-01-02"]	select rows by slicing on date index	
df["A"]	select data by column name	
df.loc[dates[0]]	select data by label	
df.loc['2013-01-02':'2013-01-04', ['A', 'D']]	slice by dates and select specific columns	 A D 2013-01-02 -0.742680 0.314713 2013-01-03 0.323359 0.295686 2013-01-04 1.379132 -0.317121
df.at[dates[0], 'C']	fast access to a single scalar in column C	
DataFrame.iloc()	select by position	
DataFrame.iat()	select by position	
df.iloc[1:3, :]	slicing rows explicity	
df.iloc[:, 1:3]	slicing columns explicity	
df.mean()	calculate the mean of column data	
df.mean(axis=1)	calculate the mean of rows	
DataFrame.agg()	aggregate user a user defined function over a specified axis	
DataFrame.transform()	call a user defined function across the whole dataframe	
df.value_counts()	perform a frequency count on the dataframe	